# MODUL AJAR
# DATA VISUALISASI


**INSTITUT TEKNOLOGI DAN BISNIS**
**(ITB) STIKOM BALI**
**2023**

**LEMBAR PENGESAHAN**

MODUL AJAR

| | |
|---|---|
| NAMA MODUL | : DATA VISUALISASI |
| PROGRAM STUDI | : SISTEM INFORMASI |
| TAHUN AJARAN | : 2022/2023 GENAP |

DISAHKAN PADA :

Tanggal/Tahun : 2 Januari 2023

DISETUJUI

K.A Program Studi Sistem Informasi

Ricky Aurelius Nurtanto Diaz, S.Kom.,M.T

NIDN. 0820128601

Denpasar, 2 Januari 2023

Dosen Penyusun

Dr. Gede Angga Pradipta S.T.,M.Eng

NIDN. 0819078803

**Satuan Acara Perkuliahan (SAP)**

Program Studi : Sistem Informasi

Kode Mata kuliah : SI9379

Nama Mata kuliah : Data Visualiasasi

Semester/SKS : Genap/4

**Deskripsi**

Visualisasi data adalah proses menggunakan elemen visual seperti diagram, grafik, atau peta untuk merepresentasikan data. Visualisasi data menerjemahkan yang kompleks, bervolume tinggi, atau numerik menjadi representasi visual yang lebih mudah diproses. Alat visualisasi data meningkatkan dan mengotomatiskan proses komunikasi visual untuk mendapatkan akurasi dan detail.

**Tujuan Instruksional Umum**

1. Mahasiswa memiliki pengetahuan dan memahami tentang tipe data dan karakteristiknya.
2. Mahasiswa memiliki pengatuhan berbagai macam tipe plot data.
3. Mahasiswa mampu menyelesaikan studi kasus terkait dengan visualisasi data.

**KATA PENGANTAR**

Puji syukur kehadirat Tuhan Yang Maha Esa atas segala rahmat-Nya sehingga modul ajar matakuliah Data Visualisasi ini bisa tersusun hingga selesai. Penulis berharap semoga modul ini bisa memenuhi kebutuhan peserta didik mata kuliah Data Visualisasi. Pembahasan modul ini dimulai dengan menjelaskan tujuan yang akan dicapai pada mata kuliah Data Visualisasi. Penulis sadar masih banyak kekurangan didalam penyusunan modul ini, karena keterbatasan pengetahuan serta pengalaman. Untuk itu penulis mengharapkan kritik dan saran yang membangun dari pembaca demi kesempurnaan modul ini.

Denpasar, Januari 2023

Penulis

# DAFTAR ISI

# MODUL 1: Pendahuluan

Selamat datang di dunia visualisasi data! Mungkin kamu pernah mendengar pepatah lama: "Sebuah gambar bernilai seribu kata." Di era informasi saat ini, ungkapan itu telah berubah menjadi "Sebuah visualisasi bernilai seribu spreadsheet."

Ketika data mengalir ke dalam hidup kita dengan kecepatan yang belum pernah ada sebelumnya, kemampuan untuk memahami dan menceritakan kisah melalui data menjadi semakin penting. Matplotlib adalah salah satu alat yang memungkinkan kita untuk melakukan hal tersebut. Sebelum kita menyelam lebih dalam ke dalam ocean Matplotlib, mari kita jelajahi mengapa visualisasi data itu penting dan apa sebenarnya Matplotlib itu.

## 1.1. Mengapa Visualisasi Data Penting?

Bayangkan kamu diberikan sebuah spreadsheet dengan ribuan baris dan kolom data. Kamu diberi tugas untuk menganalisisnya dan memberikan laporan. Bagaimana cara kamu mulai? Membaca setiap baris? Mungkin, tapi itu akan memakan waktu lama dan kamu mungkin kehilangan gambaran besar dari data tersebut. Inilah saatnya visualisasi data berperan.

Visualisasi data bukan hanya tentang membuat grafik atau diagram. Itu tentang mengkomunikasikan informasi. Dengan visualisasi yang efektif, kamu dapat:

- Pemahaman Cepat: Dengan satu pandangan, kamu dapat segera menangkap tren, korelasi, dan anomali dalam data.
- Menceritakan Kisah: Grafik dan diagram dapat menceritakan kisah yang menarik dan mendalam tentang data yang kamu miliki. Apakah penjualan menurun? Apakah ada korelasi antara dua variabel tertentu?
- Membuat Keputusan yang Tepat: Keputusan berbasis data adalah keputusan yang cerdas. Dengan visualisasi data, kamu dapat memahami informasi yang diperlukan untuk membuat keputusan dengan lebih baik dan lebih cepat.
- Engagement: Orang lebih cenderung terlibat dengan grafik atau animasi daripada baris data mentah.

## 1.2. Apa itu Matplotlib?

Jadi, apa sebenarnya Matplotlib itu?

Matplotlib adalah salah satu perpustakaan visualisasi data yang paling populer dan serbaguna di Python. Dibuat oleh John D. Hunter pada tahun 2003, perpustakaan ini telah menjadi standar de facto untuk visualisasi data dengan Python, terutama ketika digunakan bersama dengan perpustakaan lain seperti Pandas.

Beberapa kelebihan Matplotlib antara lain:

- Serbaguna: Dari plot garis sederhana hingga visualisasi 3D kompleks, Matplotlib dapat menanganinya.
- Kustomisasi Tinggi: Hampir setiap aspek dari plot dapat dikustomisasi untuk memenuhi kebutuhanmu.
- Integrasi dengan Pandas: Sebagai perpustakaan analisis data yang paling populer di Python, Pandas bekerja dengan sempurna dengan Matplotlib, memungkinkan kamu untuk dengan mudah memvisualisasikan data dari DataFrame.

Tapi, tentu saja, seperti semua alat, Matplotlib memiliki kurva belajar. Jangan khawatir, itu sebabnya kamu di sini, bukan? Bersama-sama, kita akan menjelajahi semua potensi dan keajaiban yang ditawarkan oleh Matplotlib.

## 1.3. Instalasi dan Persiapan Awal

Sebelum kita memulai petualangan visualisasi, ada beberapa langkah yang perlu kita lakukan. Pertama-tama, kita perlu memastikan bahwa Matplotlib sudah terinstal di sistem kita.

Jika kamu belum menginstalnya, kamu bisa melakukannya dengan mudah menggunakan pip, manajer paket Python:

Di terminal:

```
pip install matplotlib
```

Di jupyter notebook:

```
!pip install matplotlib
```

Setelah instalasi selesai, kita siap untuk memulai. Sepanjang buku ini, kita akan menggunakan berbagai dataset untuk memahami fitur-fitur Matplotlib dan bagaimana menggunakannya untuk menceritakan kisah melalui data.

Kamu mungkin bertanya-tanya apa yang akan kita pelajari selanjutnya. Apakah kita akan langsung memulai dengan membuat plot? Atau mungkin kita akan membahas berbagai jenis visualisasi? Jawabannya adalah... kamu akan menemukannya sendiri! Setiap bab akan membawa kamu ke aspek berbeda dari Matplotlib, memandu kamu melalui proses pembelajaran yang menarik dan interaktif.

Ketika kamu menyelesaikan bab ini, aku berharap kamu telah memahami pentingnya visualisasi data dan bagaimana Matplotlib dapat membantumu dalam misi ini. Bersiaplah untuk petualangan yang menarik, penuh dengan warna, bentuk, dan cerita.

Selanjutnya, kita akan memulai dengan berbagai tipe plot dasar yang dapat kamu buat dengan Matplotlib. Jadi, apakah kamu siap untuk memulai petualangan ini?

# MODUL 2: Berbagai Tipe Plot Dasar

## 2.1. Line Plot

Salah satu jenis visualisasi data yang paling dasar, namun sangat efektif, adalah line plot (atau sering disebut juga sebagai plot garis). Dalam dunia yang penuh dengan data, terkadang yang sederhana adalah yang terbaik. Line plot memungkinkan kamu untuk dengan cepat melihat tren dan pola dalam data berurutan.

**Kenapa Line Plot?**
Dalam dunia bisnis, ilmu, dan banyak bidang lainnya, ada kebutuhan untuk melacak perubahan dari waktu ke waktu. Bagaimana harga saham berubah setiap hari? Bagaimana perubahan suhu selama seminggu? Semua pertanyaan ini dapat dijawab dengan line plot.

### 2.1.1. Membuat Line Plot dengan Matplotlib

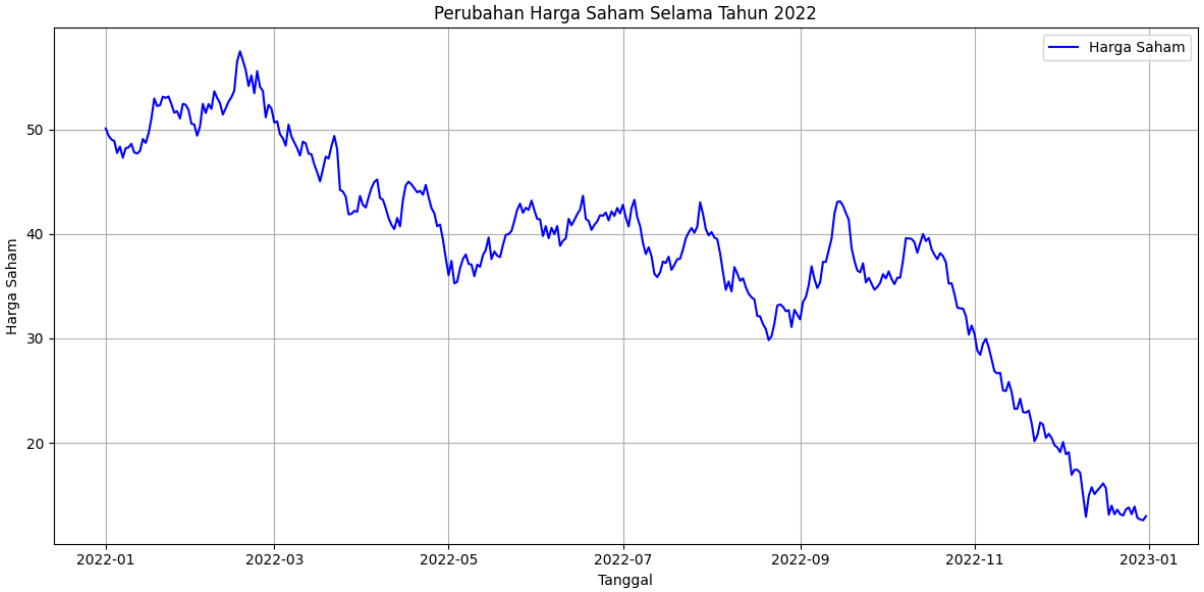
Sebelum kita melangkah lebih jauh, mari kita buat sebuah line plot sederhana. Kita akan menggunakan data sintetis untuk menggambarkan perubahan harga saham selama setahun.

Mari kita siapkan data tersebut:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Membuat data sintetis
tanggal = pd.date_range(start="2022-01-01", end="2022-12-31", freq='D')
harga_saham = np.random.randn(len(tanggal)).cumsum() + 50  # Random walk

df =  pd.DataFrame({
    'Tanggal':  tanggal,
    'Harga Saham': harga_saham
})

# Membuat line plot
plt.figure(figsize=(12, 6))
plt.plot(df['Tanggal'], df['Harga Saham'], color='blue', label='Harga
Saham')
plt.title('Perubahan Harga Saham Selama Tahun 2022')
plt.xlabel('Tanggal')
plt.ylabel('Harga Saham')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

Output:




Tadaa! Inilah line plot pertama kita. Dengan hanya beberapa baris kode, kamu telah menciptakan visualisasi yang informatif dan menarik. Tapi tunggu sebentar, Matplotlib memiliki lebih banyak hal menarik untuk ditawarkan.

### 2.1.2. Kustomisasi Line Plot

Line plot di atas cukup sederhana, bukan? Mari kita perindah sedikit. Dengan Matplotlib, kamu memiliki kebebasan penuh untuk mengkustomisasi tampilan plot. Mulai dari warna garis, gaya garis, hingga penanda di setiap titik data, semuanya bisa kamu atur!

Misalnya, bagaimana jika kita ingin garisnya berwarna hijau dengan marker berbentuk bintang? Mari kita coba!

```
# Kustomisasi line plot
plt.figure(figsize=(12, 6))
plt.plot(df['Tanggal'], df['Harga Saham'], color='green', marker='*',
linestyle='-', label='Harga Saham')
plt.title('Perubahan Harga Saham Selama Tahun 2022 dengan Kustomisasi')
plt.xlabel('Tanggal')
plt.ylabel('Harga Saham')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:


Keren, bukan? Dengan sedikit sentuhan kustomisasi, kamu telah memberikan nuansa yang berbeda pada visualisasi data. Kustomisasi memungkinkan kamu untuk menyesuaikan visualisasi sesuai dengan narasi atau tema yang ingin kamu sampaikan.

### 2.1.3. Menambahkan Multiple Lines

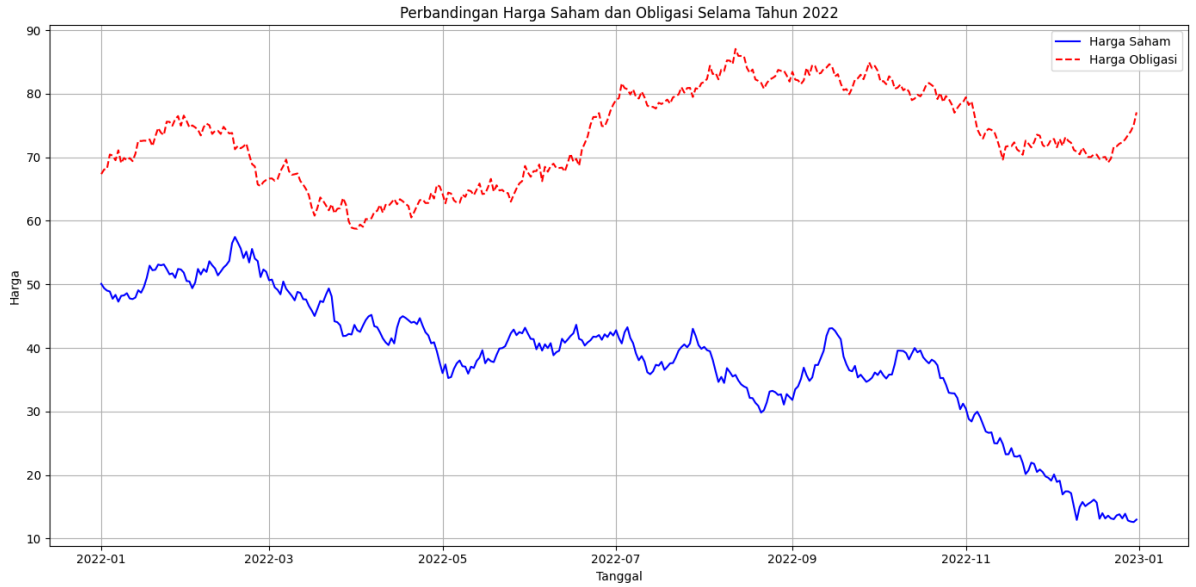
Terkadang, kita mungkin ingin membandingkan lebih dari satu set data dalam satu grafik. Dengan line plot, kamu bisa dengan mudah menambahkan lebih dari satu garis.

Misalnya, selain harga saham, mari kita tambahkan data sintetis tentang harga obligasi dan lihat bagaimana kedua harga tersebut berubah dari waktu ke waktu.

```
# Menambahkan data sintetis harga obligasi
harga_obligasi = np.random.randn(len(tanggal)).cumsum() + 70  # Random
walk
df['Harga Obligasi'] = harga_obligasi

# Membuat plot dengan multiple lines
plt.figure(figsize=(14, 7))
plt.plot(df['Tanggal'], df['Harga Saham'], color='blue', label='Harga
Saham')
plt.plot(df['Tanggal'], df['Harga Obligasi'], color='red',
linestyle='--', label='Harga Obligasi')
plt.title('Perbandingan Harga Saham dan Obligasi Selama Tahun 2022')
plt.xlabel('Tanggal')
plt.ylabel('Harga')
plt.legend()
plt.grid(True)
plt.tight_layout()
```

```
plt.show()
```

Output:



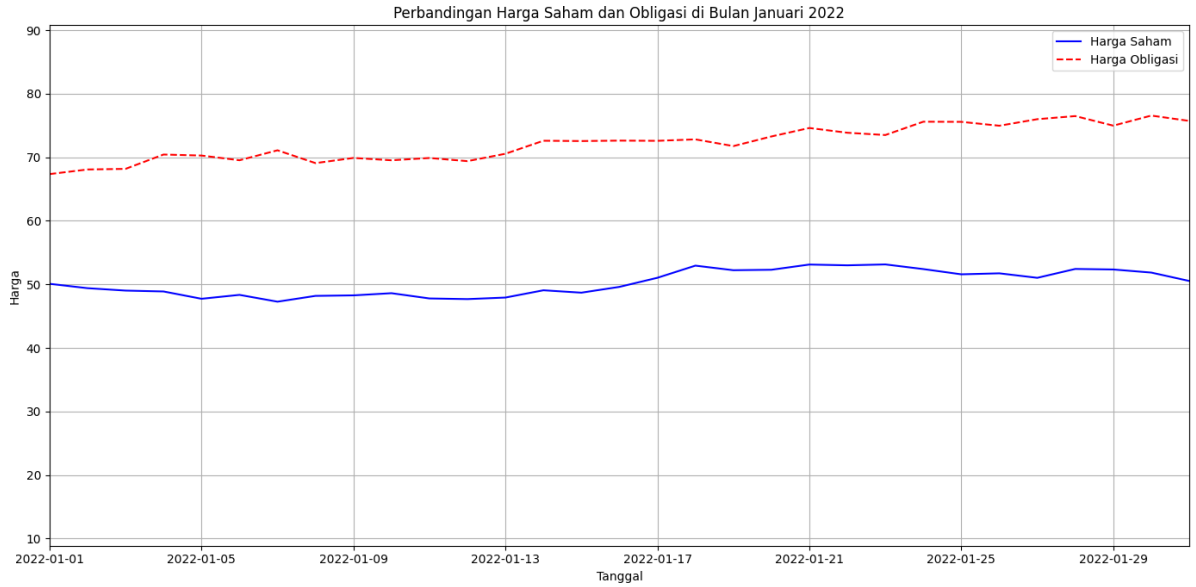

Perhatikan bagaimana kita dengan mudah menambahkan garis kedua ke plot kita dengan memanggil fungsi plot() lagi. Dengan menggunakan legenda, kita dapat dengan mudah memberi tahu pembaca apa arti dari setiap garis.

### 2.1.4. Zooming dan Memfokuskan Area Tertentu

Dalam beberapa kasus, mungkin ada bagian tertentu dari grafik yang ingin kamu perbesar untuk menunjukkan detail lebih lanjut. Misalnya, mari kita fokus pada perubahan harga saham dan obligasi selama bulan Januari 2022.

```
# Memfokuskan pada bulan Januari 2022
plt.figure(figsize=(14, 7))
plt.plot(df['Tanggal'], df['Harga Saham'], color='blue', label='Harga
Saham')
plt.plot(df['Tanggal'], df['Harga Obligasi'], color='red',
linestyle='--', label='Harga Obligasi')
plt.title('Perbandingan Harga Saham dan Obligasi di Bulan Januari 2022')
plt.xlabel('Tanggal')
plt.ylabel('Harga')
plt.xlim(pd.Timestamp('2022-01-01'), pd.Timestamp('2022-01-31'))
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:



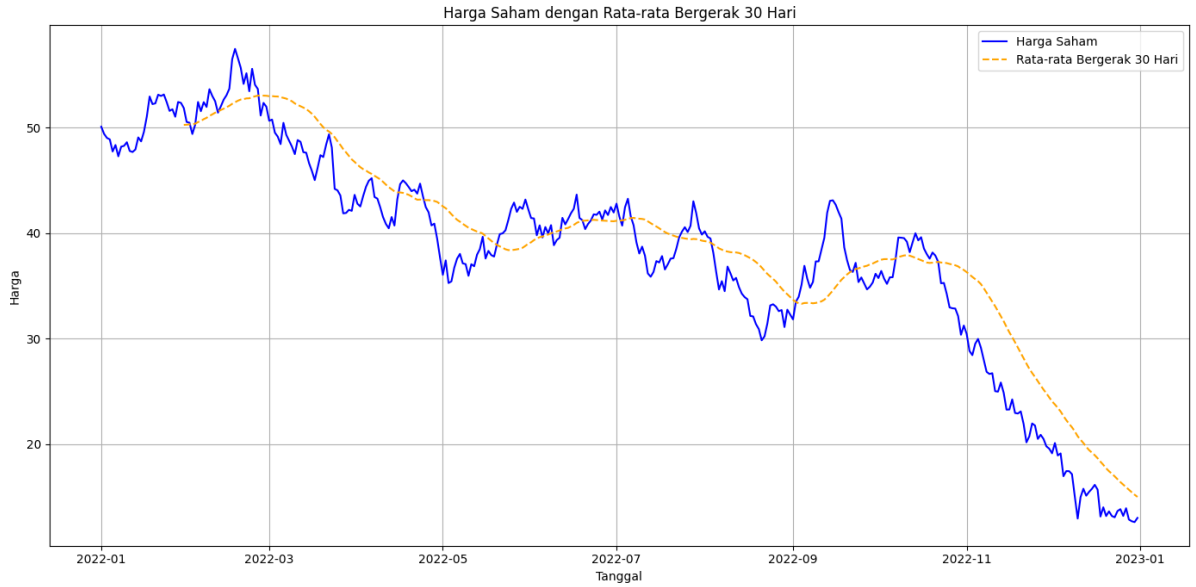

Dengan menentukan batas sumbu x, kita dapat dengan mudah memfokuskan pada periode waktu tertentu. Fitur ini sangat berguna saat kamu ingin menyoroti peristiwa atau anomali tertentu dalam data.

### 2.1.5. Menggunakan Garis Berbeda untuk Menunjukkan Trend

Misalkan kamu ingin menunjukkan trend keseluruhan dalam data harga saham kita. Salah satu cara untuk melakukannya adalah dengan menambahkan garis tren. Mari kita gunakan metode rolling mean (rata-rata bergerak) untuk menunjukkan tren harga saham selama 30 hari terakhir.

```
# Menghitung rata-rata bergerak selama 30 hari
df['Rata-rata Bergerak 30 Hari'] = df['Harga
Saham'].rolling(window=30).mean()

# Plot data harga saham dengan rata-rata bergeraknya
plt.figure(figsize=(14, 7))
plt.plot(df['Tanggal'], df['Harga Saham'], color='blue', label='Harga
Saham')
plt.plot(df['Tanggal'], df['Rata-rata Bergerak 30 Hari'],
color='orange', linestyle='--', label='Rata-rata Bergerak 30 Hari')
plt.title('Harga Saham dengan Rata-rata Bergerak 30 Hari')
plt.xlabel('Tanggal')
plt.ylabel('Harga')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:



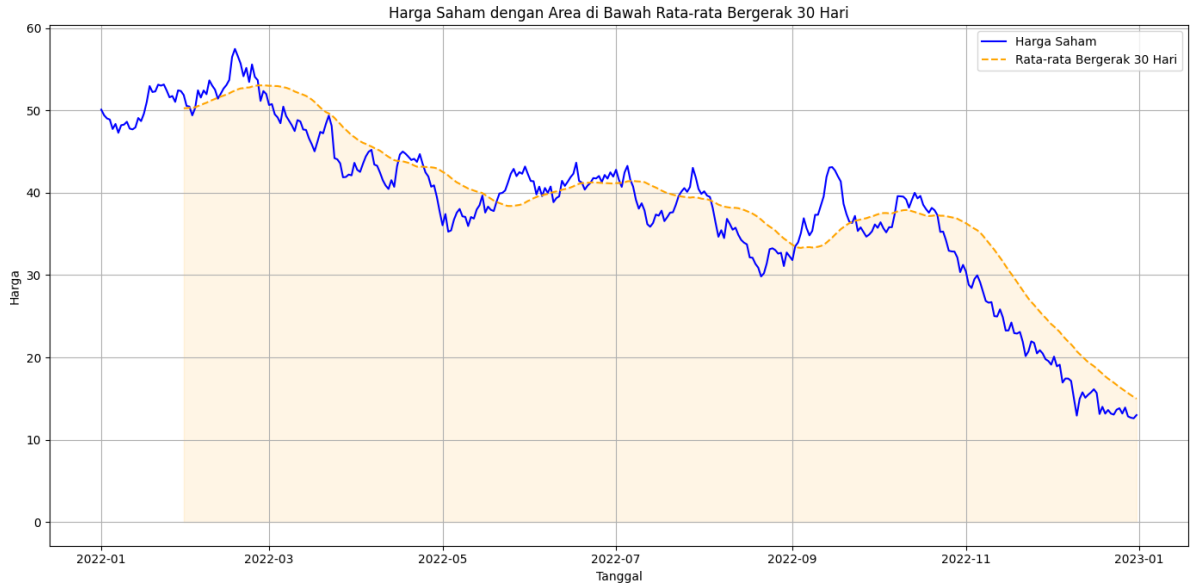

Dengan menambahkan garis tren berdasarkan rata-rata bergerak, kamu dapat dengan cepat melihat bagaimana harga saham berfluktuasi sekitar tren jangka menengah. Garis oranye memberikan gambaran umum tentang ke mana arah harga saham bergerak.

### 2.1.6. Mengisi Area di Bawah Kurva

Kadang-kadang, kamu mungkin ingin menyoroti area tertentu dalam plot. Dengan Matplotlib, kamu dapat dengan mudah mengisi area di bawah kurva, yang bisa sangat berguna untuk menunjukkan volume, kisaran, atau pentingnya area tertentu. Misalnya, mari kita soroti area di bawah rata-rata bergerak untuk menunjukkan pentingnya tren ini.

```
# Plot dengan area di bawah kurva
plt.figure(figsize=(14, 7))
plt.plot(df['Tanggal'], df['Harga Saham'], color='blue', label='Harga
Saham')
plt.plot(df['Tanggal'], df['Rata-rata Bergerak 30 Hari'],
color='orange', linestyle='--', label='Rata-rata Bergerak 30 Hari')
plt.fill_between(df['Tanggal'], df['Rata-rata Bergerak 30 Hari'],
color='orange', alpha=0.1)  # Mengisi area dengan transparansi
plt.title('Harga Saham dengan Area di Bawah Rata-rata Bergerak 30 Hari')
plt.xlabel('Tanggal')
plt.ylabel('Harga')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:




Dengan mengisi area di bawah garis rata-rata bergerak, kita menambahkan dimensi visual lain ke plot kita. Area berwarna oranye muda menyoroti pentingnya tren yang ditunjukkan oleh rata-rata bergerak.

### 2.1.7. Menggunakan Subplot untuk Menampilkan Beberapa Grafik

Terkadang, mungkin berguna untuk menampilkan beberapa grafik bersama-sama untuk membandingkan data. Dengan Matplotlib, kamu dapat dengan mudah membuat subplot untuk menampilkan lebih dari satu grafik dalam satu frame. Misalnya, mari kita bandingkan harga saham dengan harga obligasi dalam dua plot yang berbeda namun dalam satu frame.

```
# Membuat subplot
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(14, 10))

# Plot harga saham
axes[0].plot(df['Tanggal'], df['Harga Saham'], color='blue',
label='Harga Saham')
axes[0].set_title('Harga Saham Selama Tahun 2022')
axes[0].set_ylabel('Harga Saham')
axes[0].grid(True)
axes[0].legend()

# Plot harga obligasi
axes[1].plot(df['Tanggal'], df['Harga Obligasi'], color='red',
label='Harga Obligasi')
axes[1].set_title('Harga Obligasi Selama Tahun 2022')
axes[1].set_xlabel('Tanggal')
axes[1].set_ylabel('Harga Obligasi')
axes[1].grid(True)
```

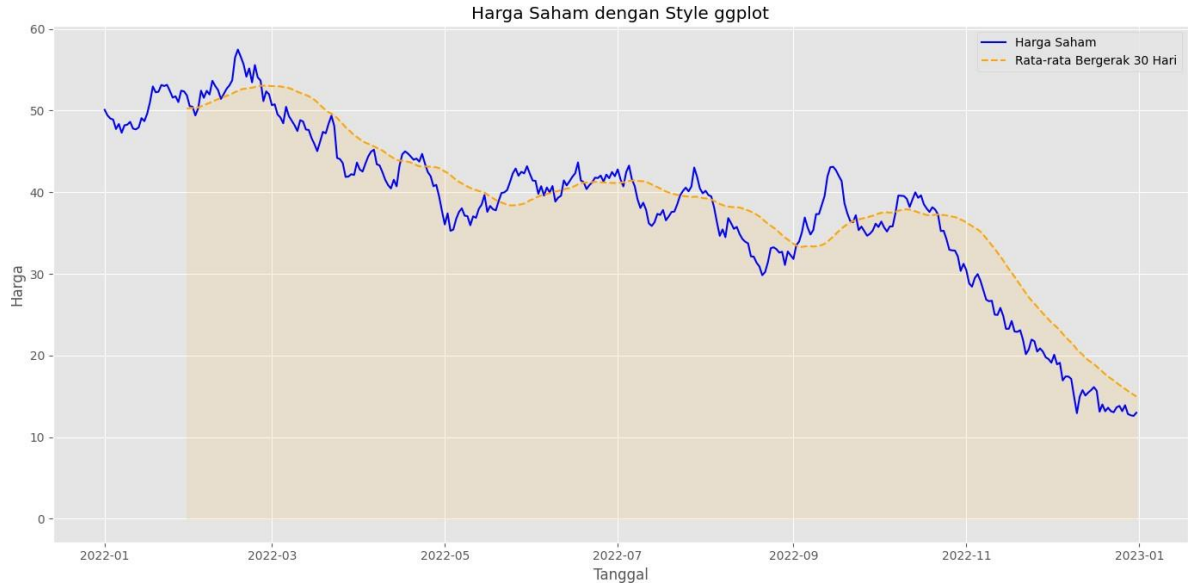
```
axes[1].legend()

plt.tight_layout()
plt.show()
```

Output:


Dengan menggunakan subplot, kamu sekarang dapat melihat perbandingan langsung antara harga saham dan harga obligasi dalam satu tampilan. Ini memudahkan pembaca untuk memahami kedua set data sekaligus.

### 2.1.8. Menggunakan Style Berbeda

Matplotlib datang dengan beberapa style bawaan yang memungkinkan kamu untuk dengan cepat mengubah tampilan plot kamu. Misalnya, mari kita gunakan style 'ggplot' yang terinspirasi dari paket ggplot2 di R untuk memberikan tampilan berbeda pada grafik kita.




Dengan mengganti style, kita telah memberikan nuansa yang berbeda pada plot kita. Style 'ggplot' memberikan tampilan yang lebih modern dan bersih, dengan palet warna yang berbeda dan estetika yang ditingkatkan.

### 2.1.9. Kasus: Perbandingan Penjualan Produk Selama 12 Bulan

Bayangkan kamu bekerja di sebuah perusahaan ritel, dan kamu ingin mengevaluasi penjualan bulanan dari dua produk berbeda selama setahun terakhir.

Mari kita mulai dengan membuat data sintetis untuk skenario ini.

```
# Membuat data sintetis untuk penjualan produk
bulan = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
"Oct", "Nov", "Dec"]
penjualan_produk_A = np.random.randint(50, 200, 12) * 100
penjualan_produk_B = np.random.randint(50, 200, 12) * 100

df_penjualan = pd.DataFrame({
    'Bulan': bulan,
    'Produk A': penjualan_produk_A,
    'Produk B': penjualan_produk_B
})

df_penjualan
```

Output:

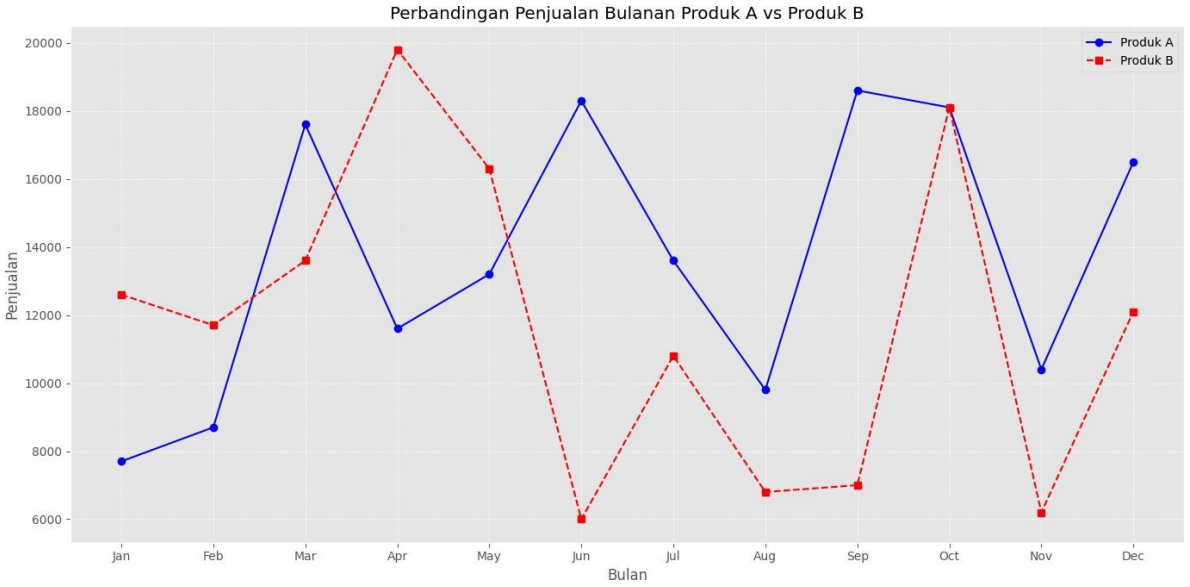
| | Bulan | Produk A | Produk B |
|---|---|---|---|
| 0 | Jan | 7700 | 12600 |
| 1 | Feb | 8700 | 11700 |
| 2 | Mar | 17600 | 13600 |
| 3 | Apr | 11600 | 19800 |
| 4 | May | 13200 | 16300 |
| 5 | Jun | 18300 | 6000 |
| 6 | Jul | 13600 | 10800 |
| 7 | Aug | 9800 | 6800 |
| 8 | Sep | 18600 | 7000 |
| 9 | Oct | 18100 | 18100 |
| 10 | Nov | 10400 | 6200 |
| 11 | Dec | 16500 | 12100 |

Kami telah menyiapkan data sintetis yang menunjukkan penjualan bulanan dari Produk A dan Produk B. Sekarang, mari kita visualisasikan data ini dengan line plot untuk membandingkan penjualan kedua produk tersebut.

Visualisasi Penjualan Bulanan Produk A dan Produk B

```
# Membuat line plot untuk penjualan bulanan kedua produk
plt.figure(figsize=(14, 7))
plt.plot(df_penjualan['Bulan'], df_penjualan['Produk A'], color='blue',
marker='o', label='Produk A')
plt.plot(df_penjualan['Bulan'], df_penjualan['Produk B'], color='red',
marker='s', linestyle='--', label='Produk B')
plt.title('Perbandingan Penjualan Bulanan Produk A vs Produk B')
plt.xlabel('Bulan')
plt.ylabel('Penjualan')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Grafik di atas menunjukkan perbandingan penjualan bulanan antara Produk A dan Produk B. Kamu dapat dengan mudah melihat bagaimana penjualan dari masing-masing produk berfluktuasi dari bulan ke bulan. Dengan marker dan gaya garis yang berbeda, kita dapat dengan jelas membedakan antara dua produk tersebut.

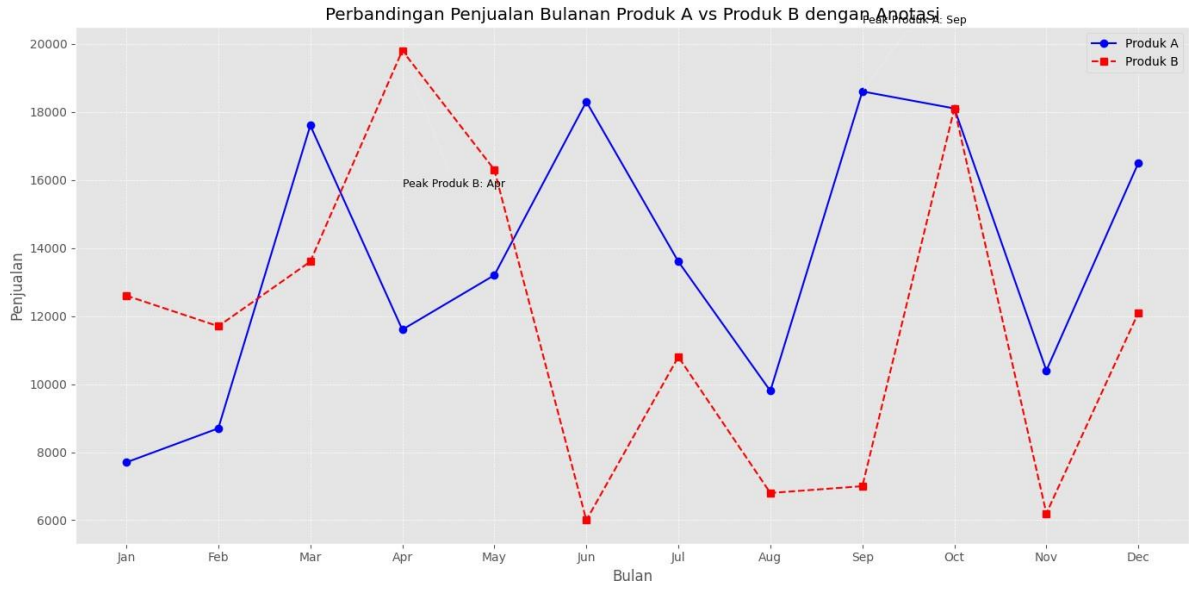
Menyoroti Pencapaian Tertinggi

Sebagai tambahan, kita mungkin ingin menyoroti bulan di mana penjualan mencapai puncaknya. Dengan menambahkan anotasi, kita dapat menunjukkan puncak penjualan untuk masing-masing produk. Mari kita lakukan itu.

```
# Menemukan bulan dengan penjualan tertinggi untuk masing-masing produk
bulan_peak_A = df_penjualan['Bulan'][df_penjualan['Produk A'].idxmax()]
nilai_peak_A = df_penjualan['Produk A'].max()

bulan_peak_B = df_penjualan['Bulan'][df_penjualan['Produk B'].idxmax()]
nilai_peak_B = df_penjualan['Produk B'].max()

# Membuat plot dengan anotasi
plt.figure(figsize=(14, 7))
plt.plot(df_penjualan['Bulan'], df_penjualan['Produk A'], color='blue',
marker='o', label='Produk A')
plt.plot(df_penjualan['Bulan'], df_penjualan['Produk B'], color='red',
marker='s', linestyle='--', label='Produk B')

# Menambahkan anotasi
plt.annotate(f'Peak Produk A: {bulan_peak_A}',
             xy=(bulan_peak_A, nilai_peak_A),
             xytext=(bulan_peak_A, nilai_peak_A + 2000),
             arrowprops=dict(facecolor='black', arrowstyle='->'),
             fontsize=9)
```

```
plt.annotate(f'Peak Produk B: {bulan_peak_B}',
             xy=(bulan_peak_B, nilai_peak_B),
             xytext=(bulan_peak_B, nilai_peak_B - 4000),
             arrowprops=dict(facecolor='black', arrowstyle='->'),
             fontsize=9)

plt.title('Perbandingan Penjualan Bulanan Produk A vs Produk B dengan Anotasi')
plt.xlabel('Bulan')
plt.ylabel('Penjualan')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dengan menambahkan anotasi, kita telah memberikan informasi tambahan tentang bulan di mana masing-masing produk mencapai penjualan tertingginya. Anotasi ini menambahkan konteks dan membuat grafik menjadi lebih informatif.

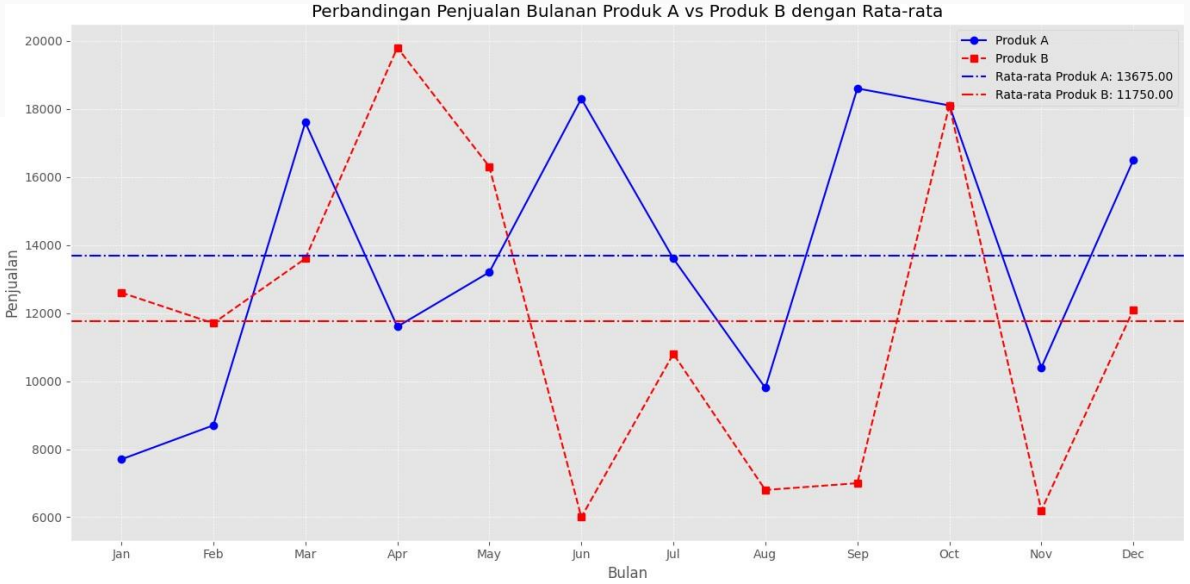
Menampilkan Rata-rata Penjualan

Selain melihat penjualan bulanan, mungkin juga berguna untuk mengetahui rata-rata penjualan selama setahun. Mari kita tambahkan garis horizontal yang menunjukkan rata-rata penjualan untuk masing-masing produk.

```
# Menghitung rata-rata penjualan
rata2_A = df_penjualan['Produk A'].mean()
rata2_B = df_penjualan['Produk B'].mean()
```

```
# Membuat plot dengan garis horizontal rata-rata
plt.figure(figsize=(14, 7))
plt.plot(df_penjualan['Bulan'], df_penjualan['Produk A'],color='blue', marker='o', label='Produk A')
plt.plot(df_penjualan['Bulan'], df_penjualan['Produk B'], color='red', marker='s', linestyle='--',
label='Produk B')

# Menambahkan garis horizontal rata-rata
plt.axhline(rata2_A, color='blue', linestyle='-.',label=f'Rata-rata Produk A: {rata2_A:.2f}')
plt.axhline(rata2_B, color='red', linestyle='-.',label=f'Rata-rata Produk B: {rata2_B:.2f}')

plt.title('Perbandingan Penjualan Bulanan Produk A vs Produk Bdengan Rata-rata')
plt.xlabel('Bulan')
plt.ylabel('Penjualan')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5) plt.tight_layout()
plt.show()
```

Output:




Dengan menambahkan garis horizontal, kita sekarang dapat dengan mudah melihat bagaimana penjualan bulanan dari masing-masing produk dibandingkan dengan rata-rata penjualannya selama setahun. Garis putus-putus memberikan referensi visual yang memungkinkan kita untuk melihat di mana bulan tertentu berada di atas atau di bawah rata-rata.

## 2.2. Scatter Plot

**Mengapa Scatter Plot?**
Bayangkan kamu memiliki dua set data dan ingin mengetahui apakah ada hubungan antara keduanya. Bagaimana cara terbaik untuk melihatnya? Ya, dengan scatter plot!

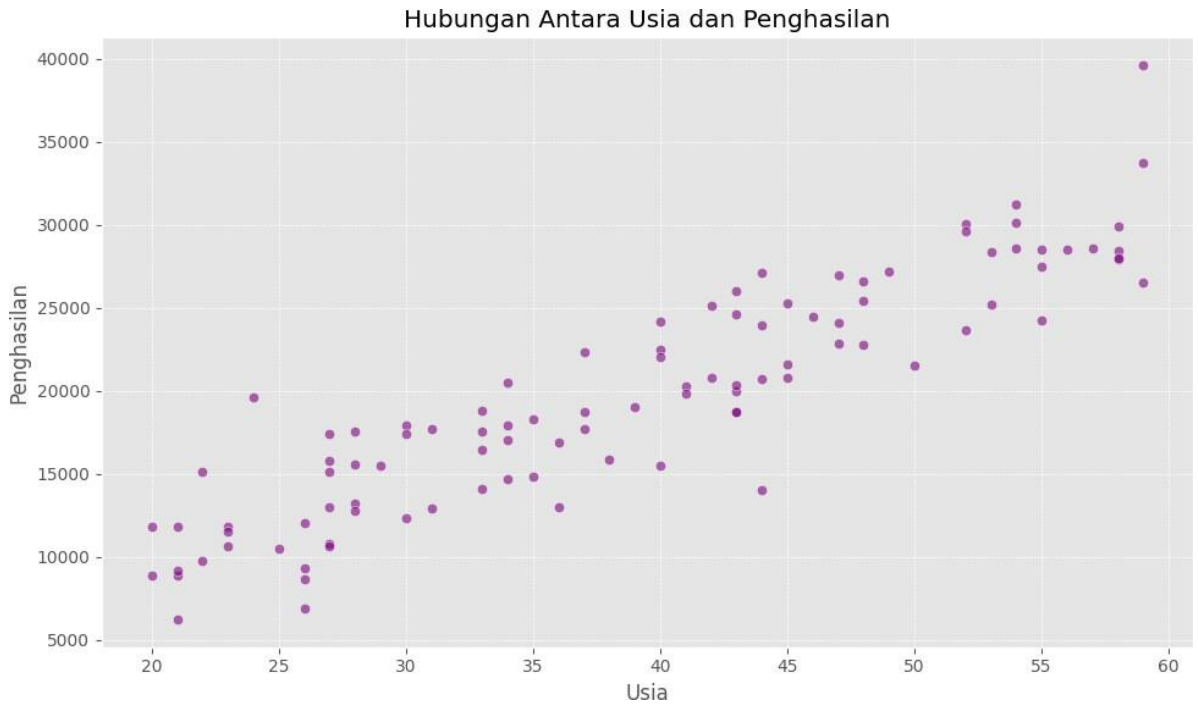
Scatter plot memungkinkan kita untuk memvisualisasikan dua variabel sebagai titik-titik dalam koordinat kartesius. Dengan demikian, kita dapat dengan cepat melihat distribusi dan hubungan antara variabel-variabel tersebut.

### 2.2.1. Membuat Scatter Plot dengan Matplotlib

Untuk memulai, mari kita ciptakan scatter plot sederhana yang menunjukkan hubungan antara usia dan penghasilan dari sekelompok orang. Kita akan gunakan data sintetis untuk tujuan ini.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Membuat data sintetis untuk usia dan penghasilan
np.random.seed(42)  # Menentukan seed untuk konsistensi
usia = np.random.randint(20, 60, 100)
penghasilan = (usia * 500) + np.random.randn(100) * 3000  # Hubungan
linear dengan noise

df_usia_penghasilan = pd.DataFrame({
    'Usia': usia,
    'Penghasilan': penghasilan
})

# Membuat scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df_usia_penghasilan['Usia'],
df_usia_penghasilan['Penghasilan'], color='purple', alpha=0.6,
edgecolors='white')
plt.title('Hubungan Antara Usia dan Penghasilan')
plt.xlabel('Usia')
plt.ylabel('Penghasilan')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dari scatter plot di atas, kita dapat melihat hubungan antara usia dan penghasilan. Sepertinya ada kecenderungan untuk penghasilan meningkat seiring dengan bertambahnya usia.

### 2.2.2. Kustomisasi Scatter Plot

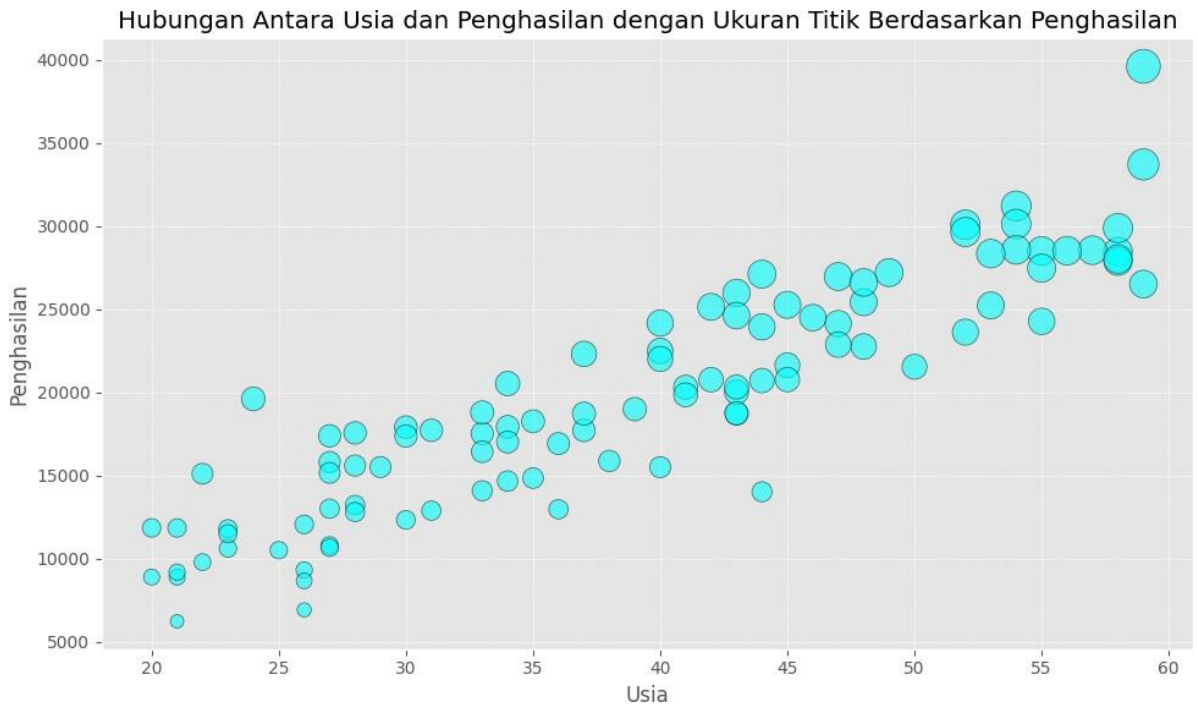
Seperti yang kita lihat sebelumnya dengan line plot, Matplotlib juga memberikan fleksibilitas penuh dalam mengkustomisasi tampilan scatter plot.

#### 2.2.2.1. Mengubah Ukuran Titik

Misalnya, mungkin kita ingin menggambarkan penghasilan dengan ukuran titik yang berbeda berdasarkan besarnya penghasilan tersebut. Mari kita coba.

```
# Menggunakan penghasilan untuk menentukan ukuran titik
sizes = df_usia_penghasilan['Penghasilan'] / 100

plt.figure(figsize=(10, 6))
plt.scatter(df_usia_penghasilan['Usia'],
df_usia_penghasilan['Penghasilan'], color='cyan', alpha=0.6, s=sizes,
edgecolors='black')
plt.title('Hubungan Antara Usia dan Penghasilan dengan Ukuran Titik
Berdasarkan Penghasilan')
plt.xlabel('Usia')
plt.ylabel('Penghasilan')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dengan mengubah ukuran titik berdasarkan penghasilan, kita sekarang memiliki dimensi tambahan dalam visualisasi kita. Titik yang lebih besar menunjukkan penghasilan yang lebih tinggi, memberikan gambaran lebih jelas tentang distribusi penghasilan dalam kelompok usia tertentu.

#### 2.2.2.2. Menggunakan Warna untuk Menunjukkan Kategori Lain

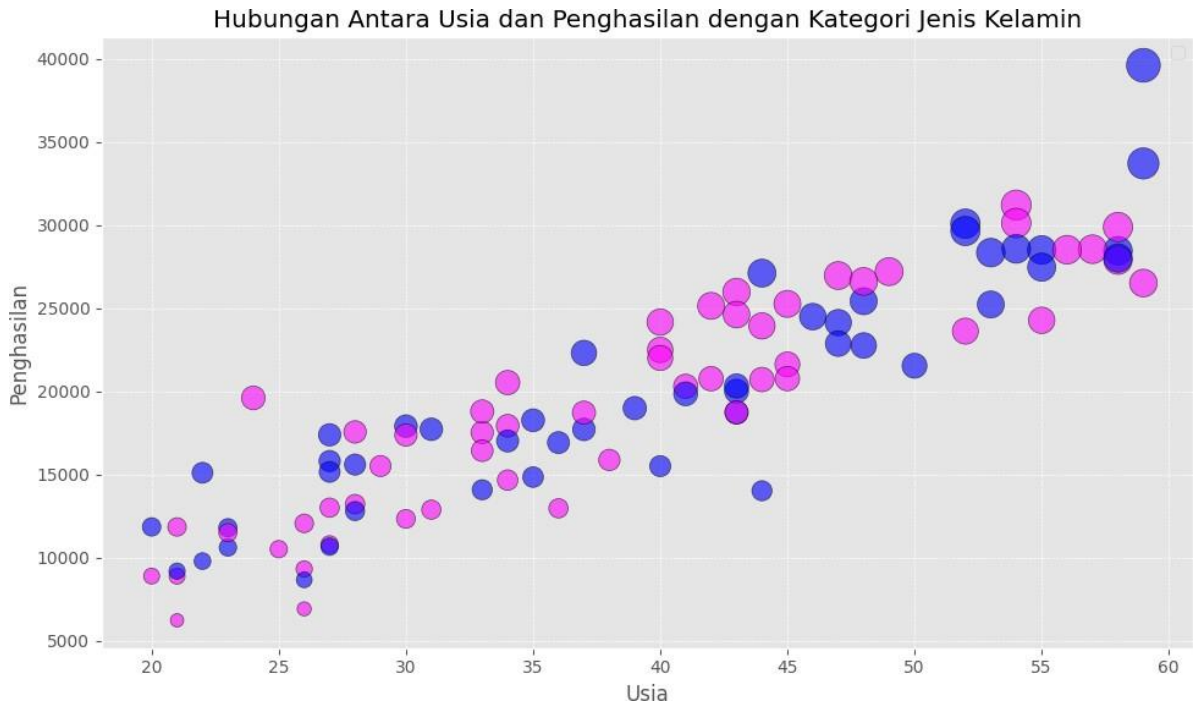
Bayangkan jika kita juga ingin mengetahui jenis kelamin dari setiap individu dalam data kita. Kita bisa menggunakan warna yang berbeda untuk mewakili laki-laki dan perempuan, sehingga kita bisa melihat distribusi gender serta hubungan antara usia dan penghasilan.

Mari kita tambahkan kolom 'Jenis Kelamin' ke data kita dan visualisasikan dengan scatter plot.

```
# Menambahkan data sintetis untuk jenis kelamin
jenis_kelamin = np.random.choice(['Laki-laki', 'Perempuan'], 100)
df_usia_penghasilan['Jenis Kelamin'] = jenis_kelamin

# Menggunakan warna berbeda untuk jenis kelamin
colors = df_usia_penghasilan['Jenis Kelamin'].map({'Laki-laki': 'blue',
'Perempuan': 'magenta'})

plt.figure(figsize=(10, 6))
scatter = plt.scatter(df_usia_penghasilan['Usia'],
df_usia_penghasilan['Penghasilan'], c=colors, alpha=0.6, s=sizes,
edgecolors='black', label=colors)
```

```
plt.title('Hubungan Antara Usia dan Penghasilan dengan Kategori Jenis
Kelamin')
plt.xlabel('Usia')
plt.ylabel('Penghasilan')

# Legenda
handles, labels = scatter.legend_elements(prop="colors")
plt.legend(handles, ['Laki-laki', 'Perempuan'])

plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dengan menambahkan warna berdasarkan jenis kelamin, kita sekarang dapat melihat distribusi gender dalam data serta hubungan antara usia dan penghasilan. Warna biru mewakili laki-laki, sedangkan magenta mewakili perempuan.

### 2.2.3. Kasus: Performa Iklan Online

Bayangkan kamu adalah seorang analis pemasaran di sebuah perusahaan e-commerce. Kamu telah menjalankan beberapa kampanye iklan online dan ingin menilai efektivitasnya. Untuk tujuan ini, kamu memutuskan untuk membandingkan "klik" dengan "konversi" (jumlah orang yang membeli produk setelah mengklik iklan).

Membuat Data Sintetis

```
# Membuat data sintetis untuk klik dan konversi
np.random.seed(45)  # Menentukan seed untuk konsistensi
klik = np.random.randint(100, 1000, 50)
konversi = klik * 0.1 + np.random.randn(50) * 30  # Konversi sekitar 10%
dari klik dengan noise

df_iklan = pd.DataFrame({
    'Klik': klik,
    'Konversi': konversi
})

df_iklan.head()
```

Output:

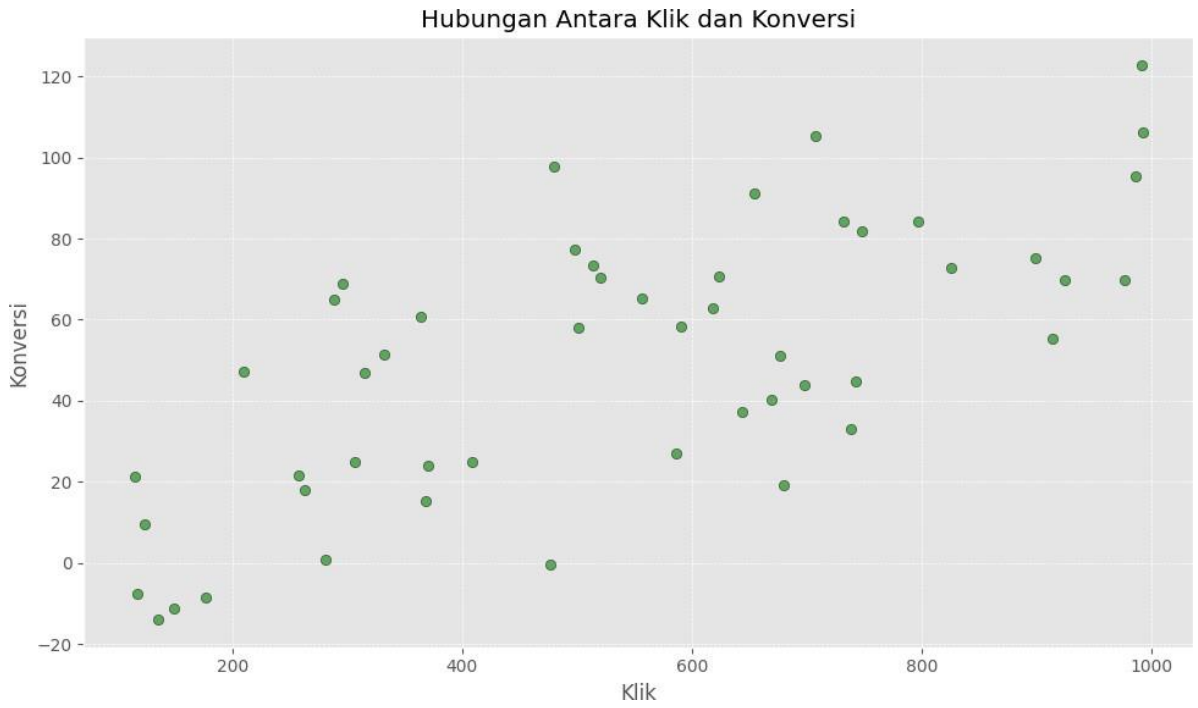
| | Klik | Konversi |
|---|---|---|
| 0 | 514 | 73.426021 |
| 1 | 743 | 44.938169 |
| 2 | 992 | 106.222797 |
| 3 | 644 | 37.322692 |
| 4 | 707 | 105.306320 |

#### 2.2.3.1. Membuat Scatter Plot Dasar

Pertama, mari kita visualisasikan data ini dalam scatter plot untuk melihat hubungan antara jumlah klik dan konversi.

```
# Membuat scatter plot dasar untuk klik vs konversi
plt.figure(figsize=(10, 6))
plt.scatter(df_iklan['Klik'], df_iklan['Konversi'], color='green',
alpha=0.6, edgecolors='black')
plt.title('Hubungan Antara Klik dan Konversi')
plt.xlabel('Klik')
plt.ylabel('Konversi')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dari scatter plot di atas, kita dapat melihat hubungan yang positif antara jumlah klik dan konversi. Namun, ada beberapa variasi, yang mungkin disebabkan oleh faktor-faktor lain seperti kualitas iklan, waktu tayang, atau audiens target.

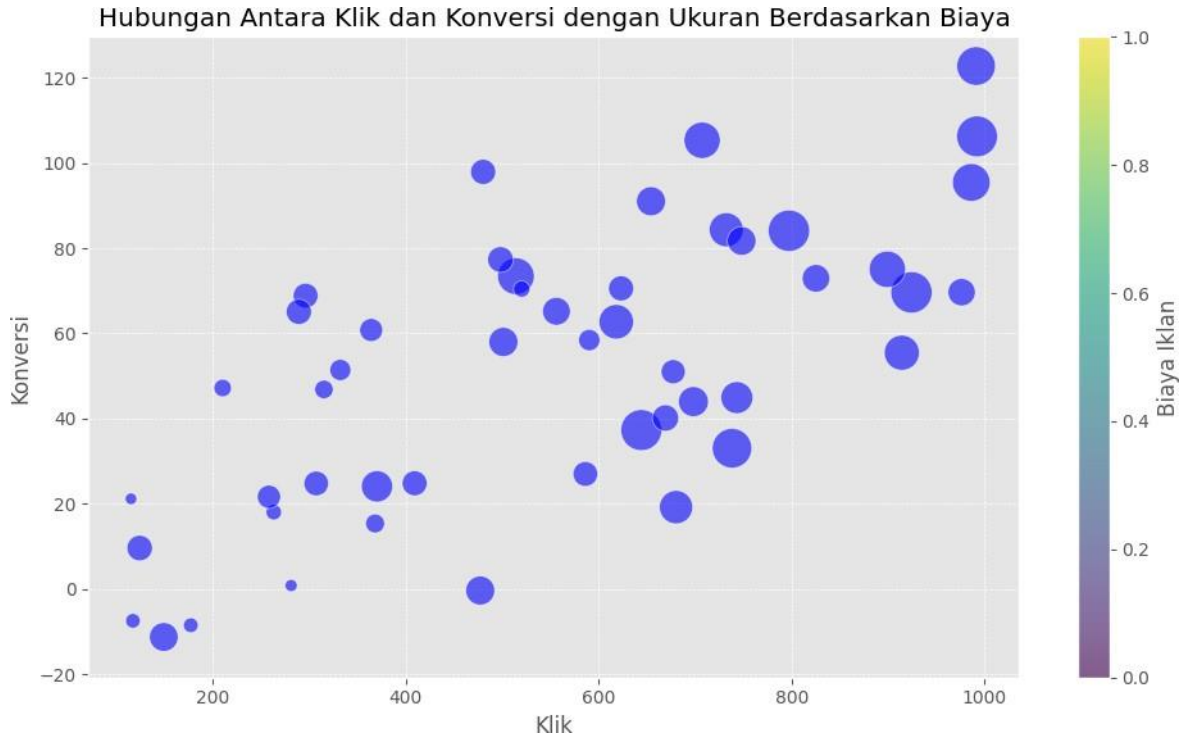
#### 2.2.3.2. Menambahkan Ukuran berdasarkan Biaya Iklan

Misalkan kita juga ingin mengetahui berapa banyak yang kita belanjakan untuk setiap kampanye iklan. Kita bisa menampilkan informasi ini dengan mengatur ukuran titik-titik berdasarkan biaya iklan. Semakin besar biaya iklan, semakin besar ukuran titiknya. Mari kita tambahkan kolom 'Biaya' ke data kita dan visualisasikan dengan scatter plot.

```
# Menambahkan data sintetis untuk biaya iklan
biaya = (klik * 0.05) + np.random.randn(50) * 10  # Biaya sekitar 5% dari klik dengan noise
df_iklan['Biaya'] = biaya

# Menggunakan biaya untuk menentukan ukuran titik
sizes = df_iklan['Biaya'] * 10

plt.figure(figsize=(10, 6))
scatter = plt.scatter(df_iklan['Klik'], df_iklan['Konversi'], c='blue', alpha=0.6, s=sizes, edgecolors='white', linewidth=0.5)
plt.title('Hubungan Antara Klik dan Konversi dengan Ukuran Berdasarkan Biaya')
plt.xlabel('Klik')
plt.ylabel('Konversi')
```

```
# Menambahkan colorbar untuk ukuran
cbar = plt.colorbar(scatter)
cbar.set_label('Biaya Iklan')

plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dengan menyesuaikan ukuran titik berdasarkan biaya iklan, kita sekarang dapat melihat seberapa banyak biaya yang dikeluarkan untuk setiap kampanye iklan. Titik-titik yang lebih besar menunjukkan biaya yang lebih tinggi.

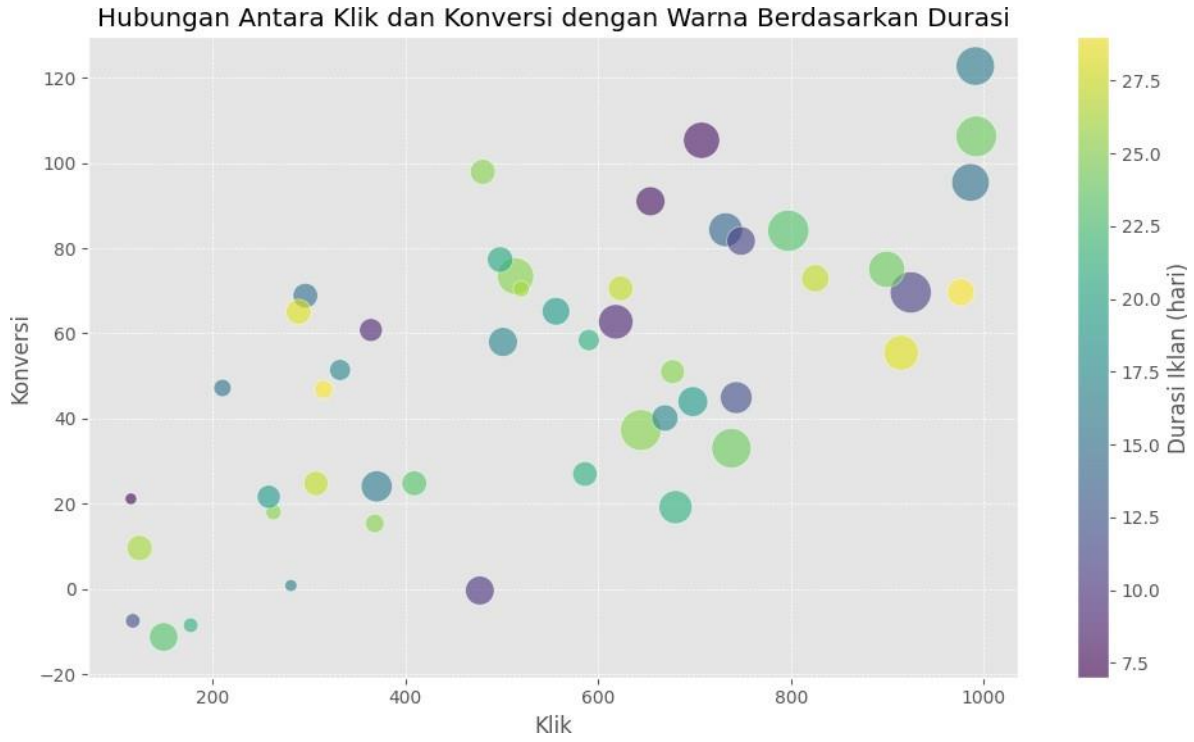
#### 2.2.3.3. Menggunakan Warna berdasarkan Durasi Iklan

Misalkan kita juga ingin tahu durasi setiap kampanye iklan. Mungkin kampanye yang berlangsung lebih lama memiliki konversi yang lebih tinggi? Kita bisa menampilkan informasi ini dengan mengatur warna titik-titik berdasarkan durasi iklan. Mari kita tambahkan kolom 'Durasi' (dalam hari) ke data kita dan visualisasikan dengan scatter plot.

```
# Menambahkan data sintetis untuk durasi iklan
durasi = np.random.randint(7, 30, 50)  # Durasi antara 7 hingga 30 hari
df_iklan['Durasi'] = durasi

# Menggunakan durasi untuk menentukan warna titik
```

```
colors = df_iklan['Durasi']

plt.figure(figsize=(10, 6))
scatter = plt.scatter(df_iklan['Klik'], df_iklan['Konversi'], c=colors,
alpha=0.6, s=sizes, edgecolors='white', linewidth=0.5, cmap='viridis')
plt.title('Hubungan Antara Klik dan Konversi dengan Warna Berdasarkan
Durasi')
plt.xlabel('Klik')
plt.ylabel('Konversi')

# Menambahkan colorbar untuk durasi
cbar = plt.colorbar(scatter)
cbar.set_label('Durasi Iklan (hari)')

plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dengan mengatur warna titik berdasarkan durasi iklan, kita sekarang dapat melihat seberapa lama setiap kampanye iklan berlangsung. Palet warna "viridis" memberikan representasi visual yang jelas dari durasi iklan, dengan warna yang lebih terang menunjukkan durasi yang lebih pendek, dan warna yang lebih gelap menunjukkan durasi yang lebih panjang.

## 2.3. Histogram

Apakah kamu pernah bertanya-tanya tentang distribusi data dalam suatu dataset? Atau bagaimana frekuensi dari suatu kisaran nilai dalam data? Jawabannya ada di histogram!

Histogram adalah alat visualisasi dasar yang memberikan gambaran tentang distribusi data. Ia menunjukkan seberapa sering suatu nilai (atau kisaran nilai) muncul dalam dataset. Meskipun histogram mungkin terlihat seperti bar chart, ada perbedaan mendasar antara keduanya. Dalam histogram, data dikelompokkan ke dalam bin atau interval dan tinggi batang menunjukkan berapa banyak data poin yang ada dalam bin tersebut.

**Mengapa Menggunakan Histogram?**
Pemahaman Distribusi Data: Histogram memberikan gambaran cepat tentang distribusi suatu variabel. Dari histogram, kamu bisa menilai bentuk, pusat, dan penyebaran distribusi.
Identifikasi Outliers: Puncak atau lekukan yang tidak biasa dapat mengindikasikan adanya outliers atau keanehan dalam data.

Perbandingan Distribusi: Dengan membandingkan histogram dari dua dataset atau lebih, kamu bisa mendapatkan wawasan tentang bagaimana variabel berperilaku di antara kelompok yang berbeda.

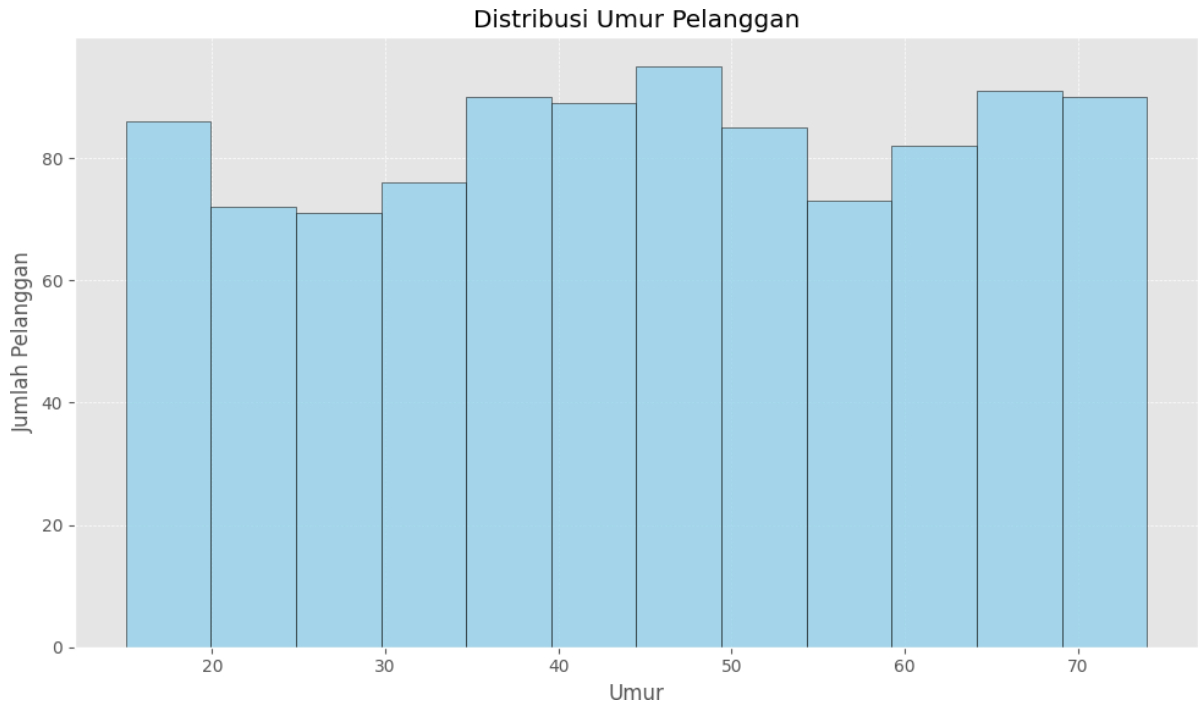
### 2.3.1. Basic Histogram

Misalkan kamu bekerja di sebuah perusahaan ritel dan ingin memahami bagaimana distribusi umur pelanggan yang berbelanja di toko kamu. Ini akan membantu dalam merencanakan strategi pemasaran yang ditargetkan untuk kelompok usia tertentu.

Kita akan memulai dengan membuat dataset sintetis yang merepresentasikan umur pelanggan.

```
# Membuat data sintetis untuk umur pelanggan
np.random.seed(42)
umur_pelanggan = np.random.randint(15, 75, 1000)  # Umur antara 15
hingga 75 tahun

df_umur = pd.DataFrame({'Umur': umur_pelanggan})

# Membuat histogram untuk umur pelanggan
plt.figure(figsize=(10, 6))
plt.hist(df_umur['Umur'], bins=12, color='skyblue', edgecolor='black',
alpha=0.7)
plt.title('Distribusi Umur Pelanggan')
plt.xlabel('Umur')
plt.ylabel('Jumlah Pelanggan')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dari histogram di atas, kita dapat melihat distribusi umur pelanggan yang berbelanja di toko kita. Batang-batang vertikal menunjukkan rentang usia (misalnya, 15-20 tahun, 20-25 tahun, dan seterusnya), sementara tinggi batang menunjukkan jumlah pelanggan dalam rentang usia tersebut.

Sejauh ini, terlihat bahwa sebagian besar pelanggan berusia antara 25 hingga 40 tahun. Namun, ada juga sejumlah pelanggan yang berusia lebih tua, hingga 75 tahun.

### 2.3.2. Pemahaman Parameter Histogram

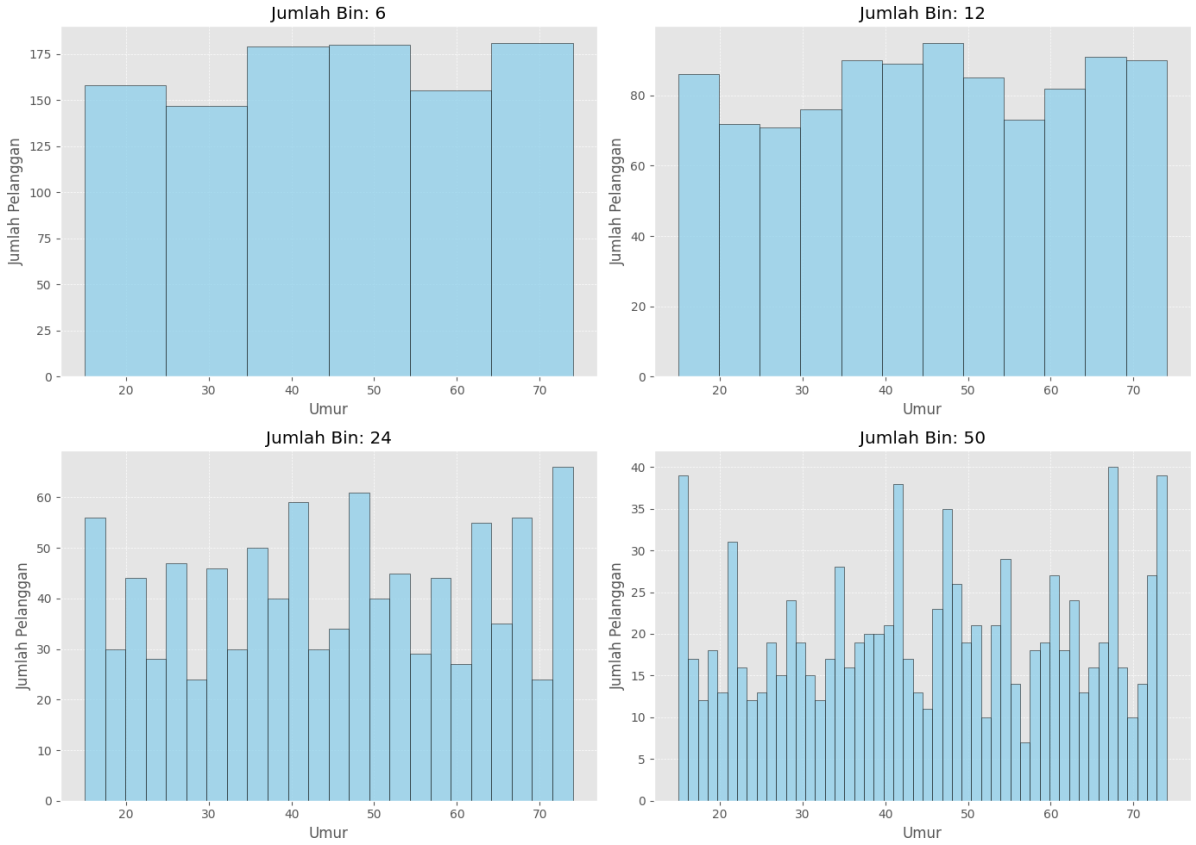
Salah satu aspek kunci dari histogram adalah pemilihan "bin" yang tepat. "Bin" adalah rentang nilai dalam histogram. Jumlah dan ukuran bin dapat mempengaruhi interpretasi distribusi data.

Mari kita coba beberapa pengaturan bin yang berbeda untuk melihat bagaimana mereka mempengaruhi tampilan histogram.

```
# Membuat beberapa histogram dengan jumlah bin yang berbeda
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 10))
bins_values = [6, 12, 24, 50]

for ax, bins_val in zip(axes.ravel(), bins_values):
    ax.hist(df_umur['Umur'], bins=bins_val, color='skyblue',
edgecolor='black', alpha=0.7)
    ax.set_title(f'Jumlah Bin: {bins_val}')
    ax.set_xlabel('Umur')
    ax.set_ylabel('Jumlah Pelanggan')
```

```
    ax.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.tight_layout()
plt.show()
```

Output:




Dari keempat histogram di atas, kita dapat melihat bagaimana pemilihan bin mempengaruhi tampilan distribusi:

- Bin = 6: Dengan hanya 6 bin, kita mendapatkan gambaran kasar tentang distribusi. Namun, detail yang lebih halus dari data mungkin hilang.
- Bin = 12: Ini memberikan keseimbangan antara detail dan interpretasi. Di sini, kita dapat melihat beberapa puncak dan lekukan yang lebih jelas dalam distribusi.
- Bin = 24: Dengan 24 bin, kita mendapatkan lebih banyak detail. Namun, dengan banyaknya bin, histogram mungkin mulai terlihat sedikit berantakan.
- Bin = 50: Di sini, setiap bin mungkin hanya mewakili beberapa tahun. Meskipun kita mendapatkan gambaran yang sangat detail, ini mungkin terlalu detail untuk interpretasi yang mudah.

Pemilihan jumlah bin yang tepat tergantung pada ukuran dataset dan tujuan analisis. Dalam banyak kasus, kamu mungkin perlu bereksperimen dengan beberapa nilai untuk menemukan yang paling sesuai.

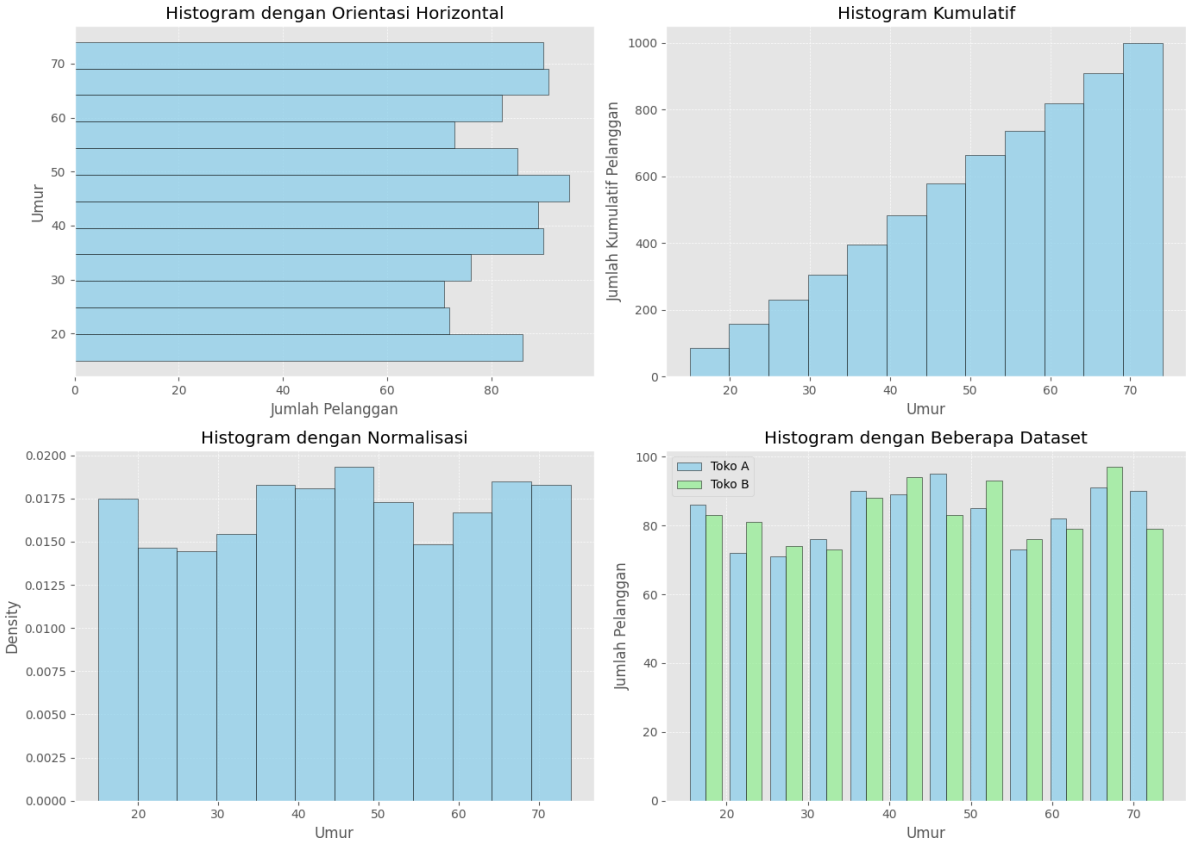
### 2.3.3. Kustomisasi Histogram

Selain menyesuaikan jumlah bin, ada banyak cara lain untuk menyesuaikan histogram agar sesuai dengan kebutuhan kamu. Mari kita coba beberapa kustomisasi lanjutan!

```
# Kustomisasi histogram dengan orientasi horizontal, kumulatif,dan normalisasi
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 10))

# Histogram dengan orientasi horizontal
axes[0, 0].hist(df_umur['Umur'], bins=12, color='skyblue', edgecolor='black',
alpha=0.7, orientation='horizontal') axes[0, 0].set_title('Histogram dengan Orientasi
Horizontal') axes[0, 0].set_ylabel('Umur')
axes[0, 0].set_xlabel('Jumlah Pelanggan')

# Histogram kumulatif
axes[0, 1].hist(df_umur['Umur'], bins=12, color='skyblue', edgecolor='black', alpha=0.7,
cumulative=True)
axes[0, 1].set_title('Histogram Kumulatif') axes[0,
1].set_xlabel('Umur')
axes[0, 1].set_ylabel('Jumlah Kumulatif Pelanggan')

# Histogram dengan density (normalisasi)
axes[1, 0].hist(df_umur['Umur'], bins=12, color='skyblue', edgecolor='black', alpha=0.7,
density=True)
axes[1, 0].set_title('Histogram dengan Normalisasi') axes[1,
0].set_xlabel('Umur')
axes[1, 0].set_ylabel('Density')

# Histogram dengan beberapa dataset (misalnya menambahkan dataumur pelanggan dari toko lain)
umur_pelanggan_toko_lain = np.random.randint(15, 75, 1000)
axes[1, 1].hist([df_umur['Umur'], umur_pelanggan_toko_lain], bins=12, color=['skyblue',
'lightgreen'], edgecolor='black', alpha=0.7, label=['Toko A', 'Toko B'])
axes[1, 1].set_title('Histogram dengan Beberapa Dataset') axes[1,
1].set_xlabel('Umur')
axes[1, 1].set_ylabel('Jumlah Pelanggan') axes[1, 1].legend()

for ax in axes.ravel():
    ax.grid(True, which='both', linestyle='--', linewidth=0.5) plt.tight_layout()
```

```
plt.show()
```

Output:




Dari visualisasi di atas, kita dapat melihat beberapa variasi histogram:

- Histogram dengan Orientasi Horizontal: Terkadang, mengubah orientasi plot dapat memberikan perspektif yang berbeda atau lebih sesuai dengan data tertentu.
- Histogram Kumulatif: Menunjukkan distribusi kumulatif dari data. Ini bisa berguna jika kamu ingin melihat berapa persentase total pelanggan yang berada di bawah usia tertentu.
- Histogram dengan Normalisasi: Dikenal juga sebagai histogram densitas, di mana area di bawah kurva sama dengan 1. Ini berguna ketika kamu ingin melihat proporsi daripada jumlah absolut.
- Histogram dengan Beberapa Dataset: Dalam kasus ini, kita membandingkan distribusi umur pelanggan dari dua toko berbeda. Menggunakan warna yang berbeda untuk setiap dataset memudahkan perbandingan.

Dengan berbagai kustomisasi ini, histogram dapat menjadi alat yang sangat fleksibel dan informatif untuk memahami distribusi data.

## 2.4. Bar Chart

Seiring dengan berjalannya waktu, kamu mungkin pernah melihat berbagai grafik yang menampilkan informasi dalam bentuk batang vertikal atau horizontal. Ya, itu adalah Bar Chart atau Grafik Batang! Bar chart merupakan salah satu alat visualisasi paling populer dan sering digunakan dalam laporan, presentasi, dan analisis data.

**Mengapa Menggunakan Bar Chart?**

Grafik batang adalah cara yang efektif untuk membandingkan item antara berbagai kategori. Keunggulan utamanya adalah:

- Kemudahan Interpretasi: Grafik batang mudah dipahami oleh semua kalangan.
- Fleksibilitas: Bisa digunakan untuk data kategorikal maupun numerik.
- Perbandingan: Memudahkan perbandingan antar kategori.

### 2.4.1. Jenis-jenis Bar Chart

Ada beberapa jenis bar chart yang sering digunakan, di antaranya:

- Vertical Bar Chart: Batang vertikal digunakan untuk merepresentasikan data.
- Horizontal Bar Chart: Batang horizontal digunakan, biasanya untuk data kategorikal dengan nama kategori yang panjang atau banyak kategori.
- Stacked Bar Chart: Data ditampilkan dalam bentuk tumpukan, memungkinkan untuk membandingkan total antar kategori sekaligus melihat komposisi dari setiap kategori.
- Grouped Bar Chart: Mirip dengan stacked, tetapi batang dikelompokkan berdampingan alih-alih ditumpuk.

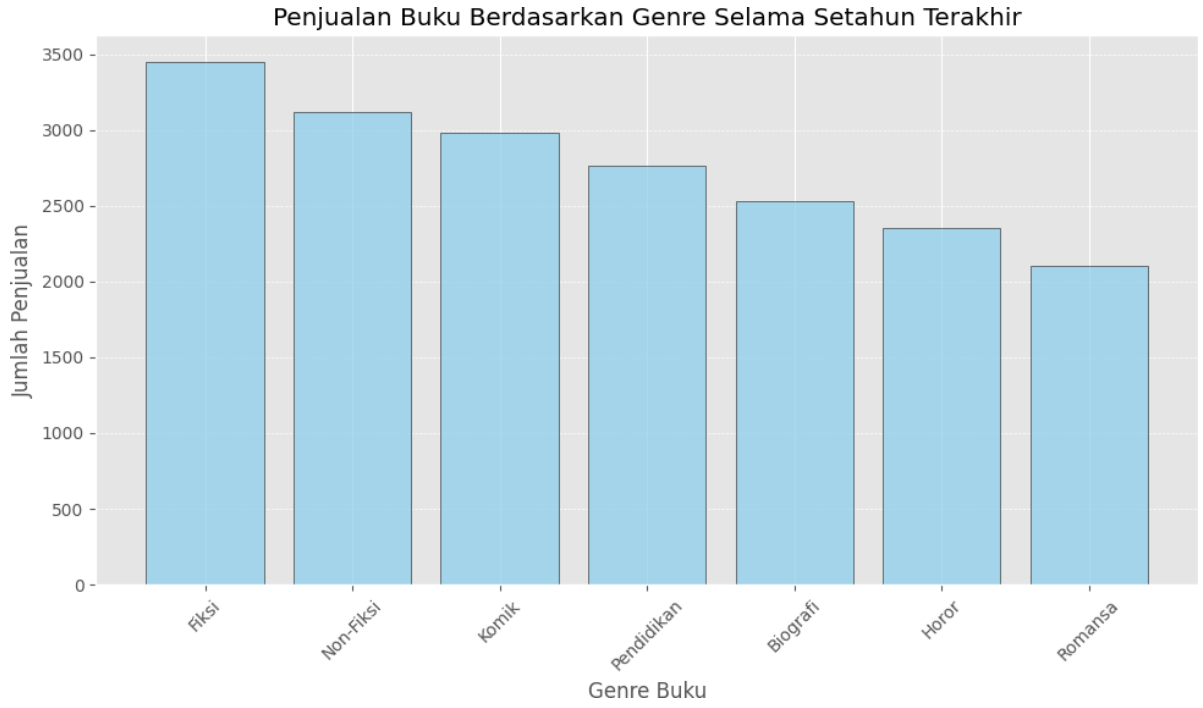
### 2.4.2. Basic Bar Chart

Bayangkan kamu adalah seorang analis di sebuah perusahaan penerbit buku. Kamu ingin memvisualisasikan penjualan buku berdasarkan genre selama setahun terakhir. Mari kita mulai dengan membuat dataset sintetis yang merepresentasikan situasi ini.

```
# Membuat data sintetis untuk penjualan buku berdasarkan genre
genres = ['Fiksi', 'Non-Fiksi', 'Komik', 'Pendidikan', 'Biografi',
'Horor', 'Romansa']
penjualan = [3450, 3120, 2980, 2760, 2530, 2350, 2100]

df_penjualan = pd.DataFrame({
    'Genre': genres,
    'Penjualan': penjualan
})

# Membuat vertical bar chart untuk penjualan buku berdasarkan genre
plt.figure(figsize=(10, 6))
```

```
plt.bar(df_penjualan['Genre'], df_penjualan['Penjualan'],
color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Penjualan Buku Berdasarkan Genre Selama Setahun Terakhir')
plt.xlabel('Genre Buku')
plt.ylabel('Jumlah Penjualan')
plt.xticks(rotation=45)
plt.grid(True, which='both', linestyle='--', linewidth=0.5, axis='y')
plt.tight_layout()
plt.show()
```

Output:




Dari grafik batang di atas, kita mendapatkan gambaran jelas tentang penjualan buku berdasarkan genre. Fiksi tampaknya menjadi genre yang paling populer, diikuti oleh Non-Fiksi dan Komik. Horor, Romansa, dan Biografi memiliki penjualan yang lebih rendah dibandingkan genre lainnya.

### 2.4.3. Variasi Bar Chart

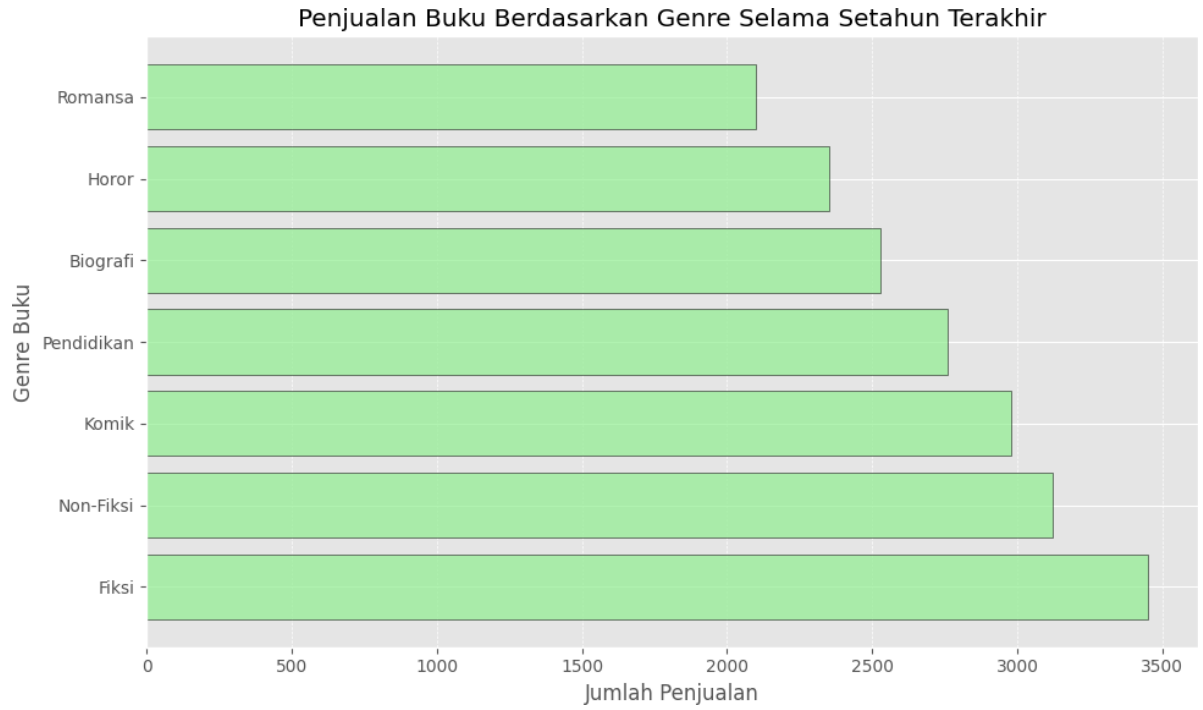
Mari kita coba beberapa variasi dari grafik batang untuk mendapatkan wawasan yang lebih dalam dari data kita.

**1. Horizontal Bar Chart**

Grafik batang horizontal dapat digunakan ketika kita memiliki label kategori yang panjang atau hanya ingin mengubah orientasi grafik untuk keperluan estetika. Mari kita coba memvisualisasikan data kita dalam bentuk horizontal.

```
# Membuat horizontal bar chart untuk penjualan buku berdasarkan genre
plt.figure(figsize=(10, 6))
plt.barh(df_penjualan['Genre'], df_penjualan['Penjualan'],
color='lightgreen', edgecolor='black', alpha=0.7)
plt.title('Penjualan Buku Berdasarkan Genre Selama Setahun Terakhir')
plt.ylabel('Genre Buku')
plt.xlabel('Jumlah Penjualan')
plt.grid(True, which='both', linestyle='--', linewidth=0.5, axis='x')
plt.tight_layout()
plt.show()
```

Output:




Grafik batang horizontal memberikan perspektif yang berbeda dari data yang sama. Dengan format ini, label kategori mungkin lebih mudah dibaca, terutama jika ada banyak kategori atau nama kategori yang panjang.

**2. Stacked Bar Chart**

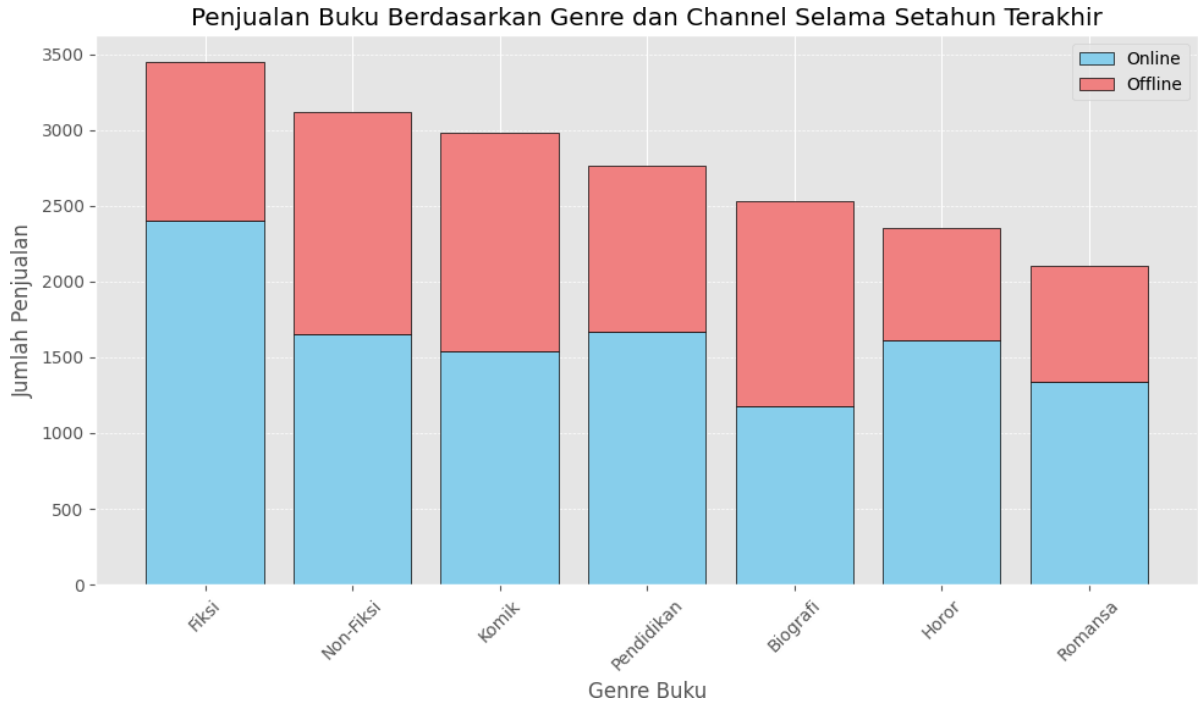
Bagaimana jika kita ingin menambahkan dimensi lain ke data kita? Misalkan, selain mengetahui penjualan total, kita juga ingin tahu berapa banyak dari penjualan tersebut yang berasal dari penjualan online dan offline.

Mari kita tambahkan data sintetis untuk skenario tersebut dan visualisasikan dengan Stacked Bar Chart.

```
# Menambahkan data sintetis untuk penjualan online dan offline
df_penjualan['Online'] = df_penjualan['Penjualan'] *
```

```
np.random.uniform(0.4, 0.7, len(df_penjualan))
df_penjualan['Offline'] = df_penjualan['Penjualan'] -
df_penjualan['Online']

# Membuat stacked bar chart untuk penjualan buku berdasarkan genre
plt.figure(figsize=(10, 6))
plt.bar(df_penjualan['Genre'], df_penjualan['Online'], label='Online',
color='skyblue', edgecolor='black')
plt.bar(df_penjualan['Genre'], df_penjualan['Offline'], label='Offline',
bottom=df_penjualan['Online'], color='lightcoral', edgecolor='black')
plt.title('Penjualan Buku Berdasarkan Genre dan Channel Selama Setahun
Terakhir')
plt.xlabel('Genre Buku')
plt.ylabel('Jumlah Penjualan')
plt.xticks(rotation=45)
plt.grid(True, which='both', linestyle='--', linewidth=0.5, axis='y')
plt.legend()
plt.tight_layout()
plt.show()
```

Output:




Dari Stacked Bar Chart di atas, kita dapat memahami dua dimensi sekaligus:

Total Penjualan untuk Setiap Genre: Ini diwakili oleh tinggi total dari setiap batang. Pembagian Penjualan Online dan Offline: Bagian biru mewakili penjualan online, sementara bagian merah mewakili penjualan offline.

Dengan grafik ini, kita dapat dengan mudah melihat bahwa sebagian besar penjualan untuk genre buku tertentu berasal dari saluran online, sementara beberapa lainnya memiliki distribusi yang hampir sama antara online dan offline.

## 2.5. Pie Chart

Apakah kamu pernah ingin mengetahui bagian mana dari suatu keseluruhan yang paling mendominasi? Atau bagaimana komposisi dari suatu keseluruhan dibagi? Jika ya, maka Pie Chart, atau yang dikenal juga dengan diagram lingkaran, adalah jawabannya.

Pie Chart adalah representasi grafis yang menampilkan data dalam bentuk potongan-potongan lingkaran. Setiap potongan lingkaran (atau "slice") mewakili kategori tertentu dan ukurannya proporsional dengan jumlah yang diwakilinya dari keseluruhan.

### 2.5.1. Keuntungan dan Kekurangan Pie Chart

Sebelum kita masuk ke contoh dan kustomisasi, mari kita bahas cepat tentang kapan harus dan kapan tidak harus menggunakan Pie Chart.

Keuntungan:

- Intuitif: Mudah dipahami dan diinterpretasikan. Pembaca dapat dengan cepat memahami proporsi relatif antara kategori.
- Visual: Dengan penggunaan warna dan potongan, grafik ini dapat menarik dan informatif.

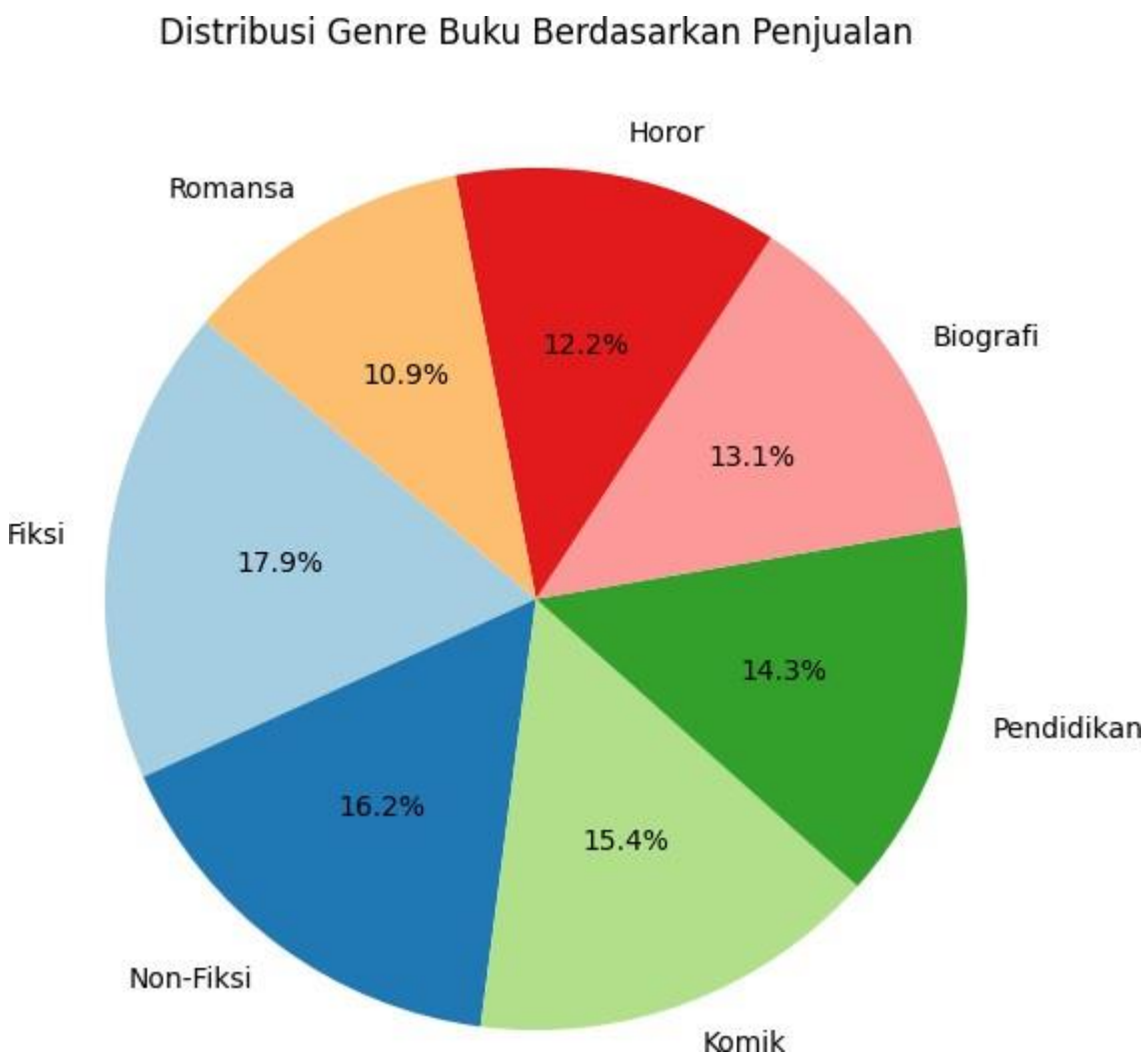
Kekurangan:

- Keterbatasan Kategori: Jika ada terlalu banyak kategori, Pie Chart dapat terlihat berantakan dan sulit dibaca.
- Perbandingan Kesulitan: Sulit membandingkan potongan yang berdekatan dalam ukuran, terutama jika perbedaannya kecil.
- Kasus Dasar: Distribusi Genre Buku

Bayangkan kamu memiliki toko buku dan ingin mengetahui proporsi genre buku yang kamu jual. Pie Chart dapat memberikan gambaran visual yang cepat tentang distribusi genre buku. Mari kita gunakan data penjualan buku yang sebelumnya telah kita buat.

```
# Membuat Pie Chart untuk distribusi genre buku berdasarkan penjualan
plt.figure(figsize=(10, 7))
plt.pie(df_penjualan['Penjualan'], labels=df_penjualan['Genre'],
autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
plt.title('Distribusi Genre Buku Berdasarkan Penjualan')
plt.show()
```

Output:




Dari Pie Chart di atas, kamu dapat melihat distribusi penjualan buku berdasarkan genre. Setiap potongan lingkaran mewakili suatu genre, dan ukurannya menunjukkan proporsi penjualannya dalam total penjualan buku. Label dan persentase di setiap potongan membantu kamu memahami proporsi relatif genre tersebut.

Namun, Pie Chart ini adalah contoh dasar. Mari kita jelajahi lebih lanjut bagaimana kita bisa memperkaya dan meningkatkan kejelasan Pie Chart kita.

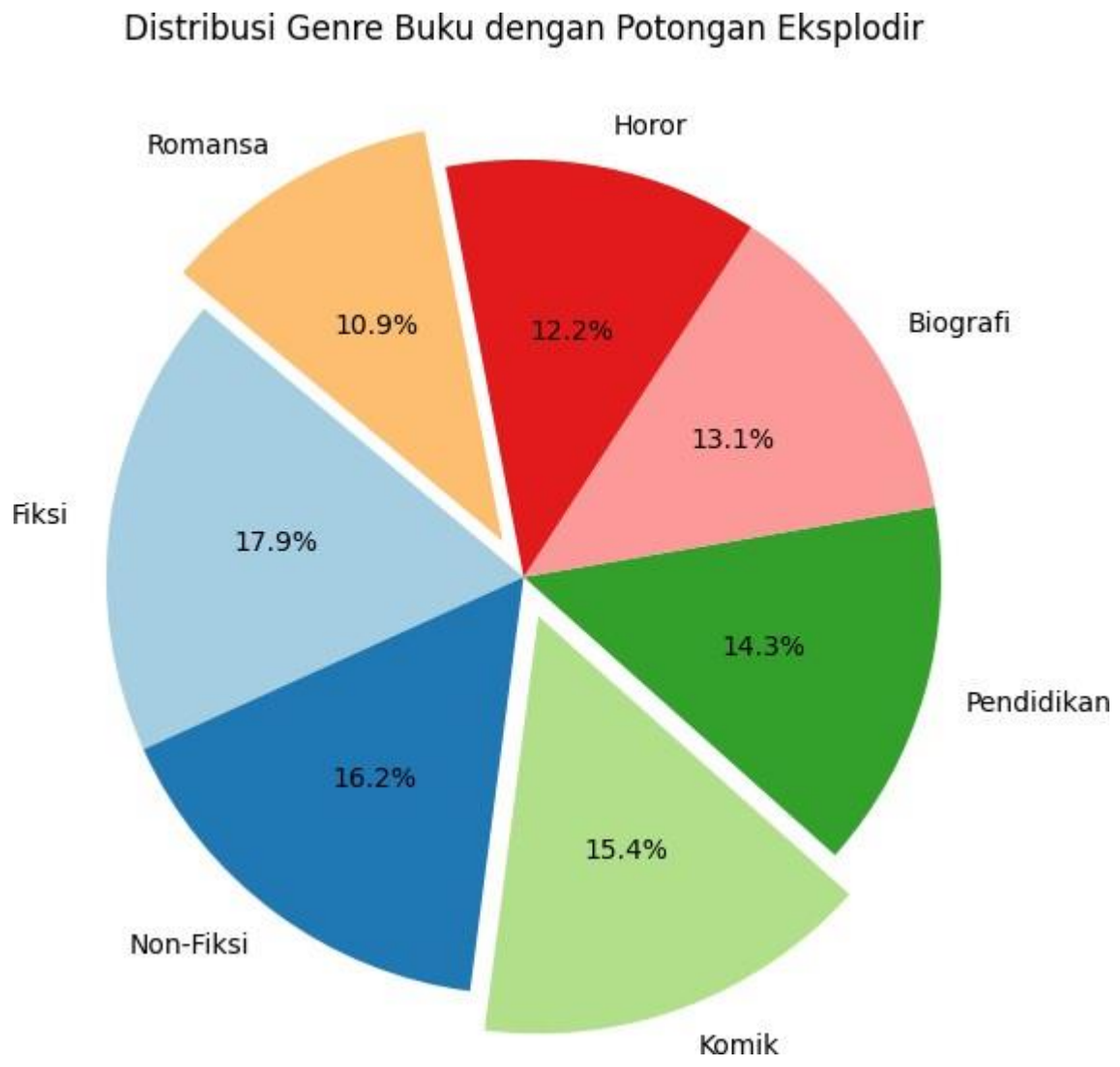
### 2.5.2. Eksplodir Potongan Pie Chart

Salah satu cara untuk menyoroti atau memberi penekanan pada satu atau lebih potongan dari Pie Chart adalah dengan "mengeksplodir" potongan tersebut. Ini berarti potongan tersebut ditarik keluar dari pusat lingkaran untuk menonjol.

Misalnya, kita ingin menyoroti genre "Komik" dan "Romansa" karena alasan tertentu, mungkin karena mereka adalah genre best-seller atau mungkin ada promosi khusus yang berlangsung. Mari kita eksplodir kedua potongan tersebut.

```
# Menentukan potongan mana yang akan dieksplodir
explode = [0.1 if genre in ["Komik", "Romansa"] else 0 for genre in
df_penjualan['Genre']]

plt.figure(figsize=(10, 7))
plt.pie(df_penjualan['Penjualan'], labels=df_penjualan['Genre'],
autopct='%1.1f%%', startangle=140, explode=explode,
colors=plt.cm.Paired.colors)
plt.title('Distribusi Genre Buku dengan Potongan Eksplodir')
plt.show()
```

Output:



Kedua potongan "Komik" dan "Romansa" kini menonjol dari Pie Chart, memberi penekanan visual pada genre-genre tersebut. Teknik ini bisa sangat berguna ketika kamu ingin menyoroti informasi tertentu atau memberikan konteks tambahan kepada pembaca.

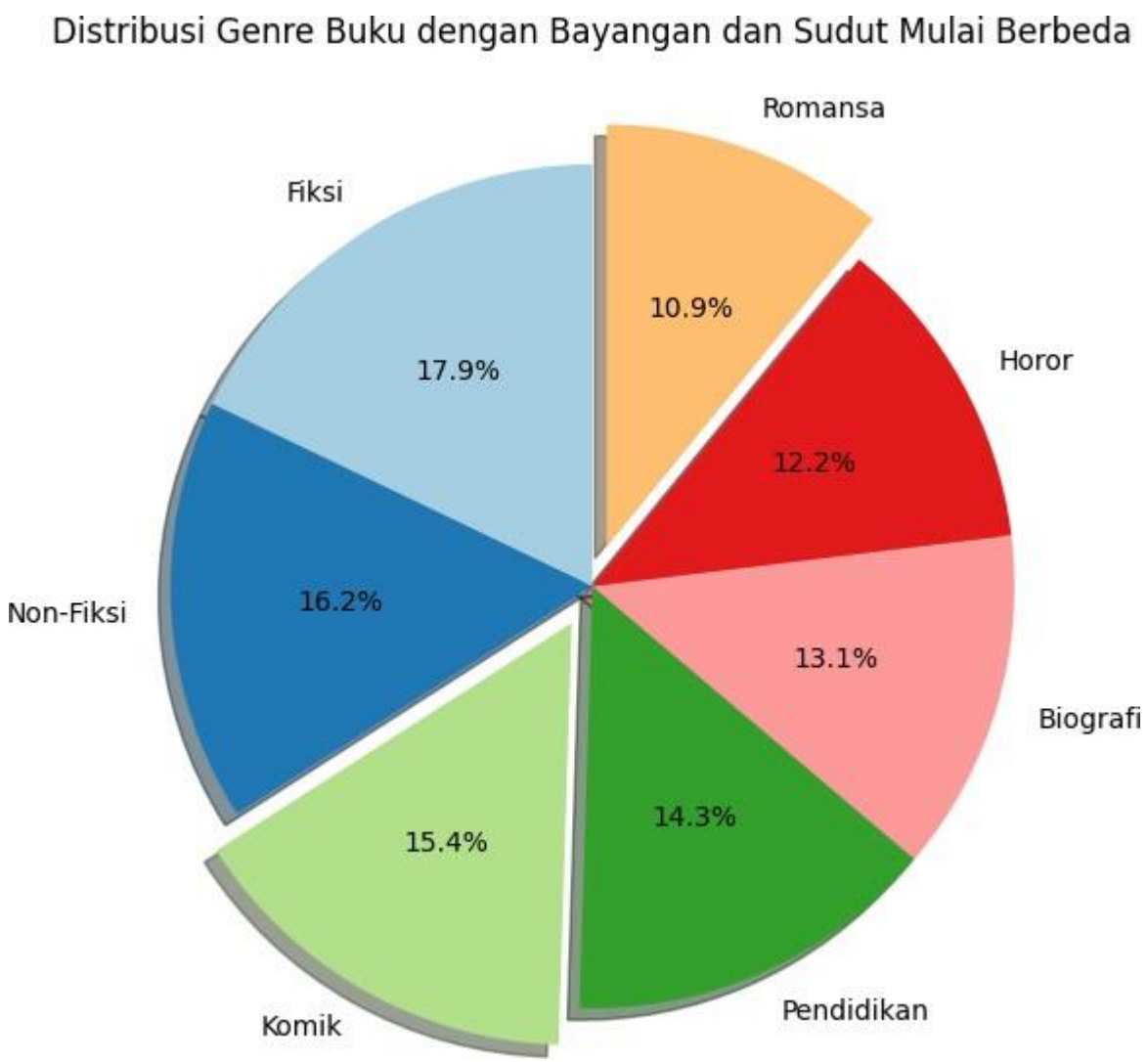
### 2.5.3. Menambahkan Shadow dan Mengubah Start Angle

Menambahkan bayangan bisa memberikan efek kedalaman pada Pie Chart, membuatnya lebih menarik secara visual. Selain itu, mengubah sudut mulai dari Pie Chart memungkinkan kamu untuk memposisikan potongan-potongan dengan cara tertentu, sehingga lebih mudah dibaca atau menonjol.

Mari kita coba kedua kustomisasi ini pada Pie Chart kita.

```
plt.figure(figsize=(10, 7))
plt.pie(df_penjualan['Penjualan'], labels=df_penjualan['Genre'],
autopct='%1.1f%%', startangle=90, explode=explode,
colors=plt.cm.Paired.colors,shadow=True)
plt.title('Distribusi Genre Buku dengan Bayangan dan Sudut Mulai
Berbeda')
plt.show()
```

Output:




Dengan menambahkan bayangan dan mengubah sudut mulai, Pie Chart kita kini memiliki tampilan yang sedikit berbeda. Bayangan memberikan kesan kedalaman, sementara mengubah start angle memastikan Pie Chart dimulai dari atas dan berputar searah jarum jam, memberikan tata letak yang mungkin lebih familiar bagi beberapa pembaca.

### 2.5.4. Donut Chart

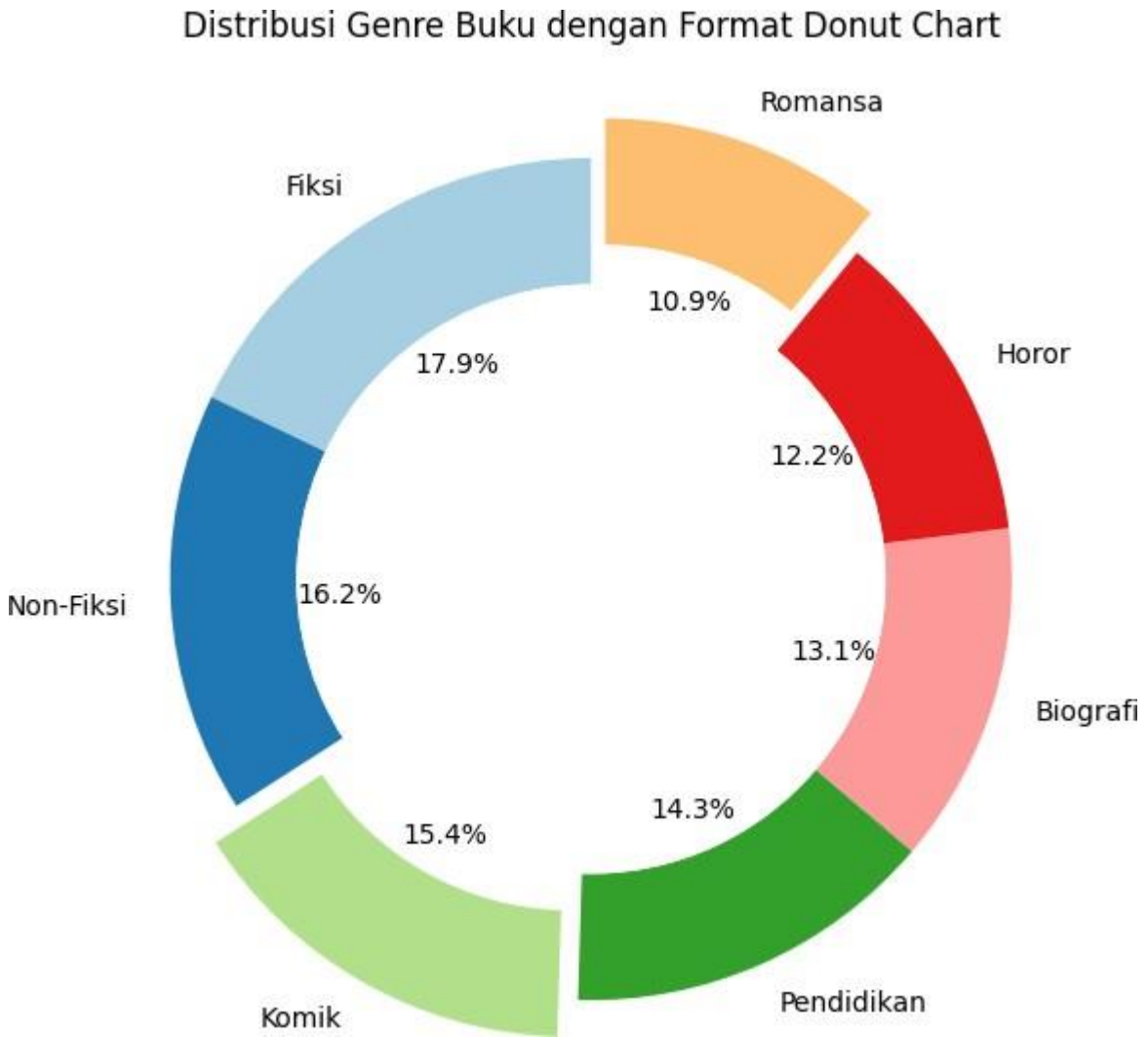
Varian dari Pie Chart yang populer adalah Donut Chart. Ini mirip dengan Pie Chart, tetapi memiliki lubang di tengah, memberikan bentuk donat. Donut Chart bisa menjadi pilihan estetika atau digunakan untuk menampilkan informasi tambahan di tengah grafik.

Mari kita coba mengubah Pie Chart kita menjadi Donut Chart.

```
plt.figure(figsize=(10, 7))
plt.pie(df_penjualan['Penjualan'], labels=df_penjualan['Genre'],
autopct='%1.1f%%', startangle=90, explode=explode,
colors=plt.cm.Paired.colors, wedgeprops=dict(width=0.3))
```

```
plt.gca().add_artist(plt.Circle((0,0),0.70,fc='white'))
plt.title('Distribusi Genre Buku dengan Format Donut Chart')
plt.show()
```

Output:




Kini Pie Chart kita telah bertransformasi menjadi Donut Chart. Dengan lubang di tengah, kita memiliki ruang tambahan yang dapat digunakan untuk teks, gambar, atau informasi tambahan lainnya. Dalam contoh ini, kita sengaja meninggalkan ruang tersebut kosong, namun dalam aplikasi nyata, itu bisa menjadi tempat yang berguna untuk judul, total data, atau informasi penting lainnya.

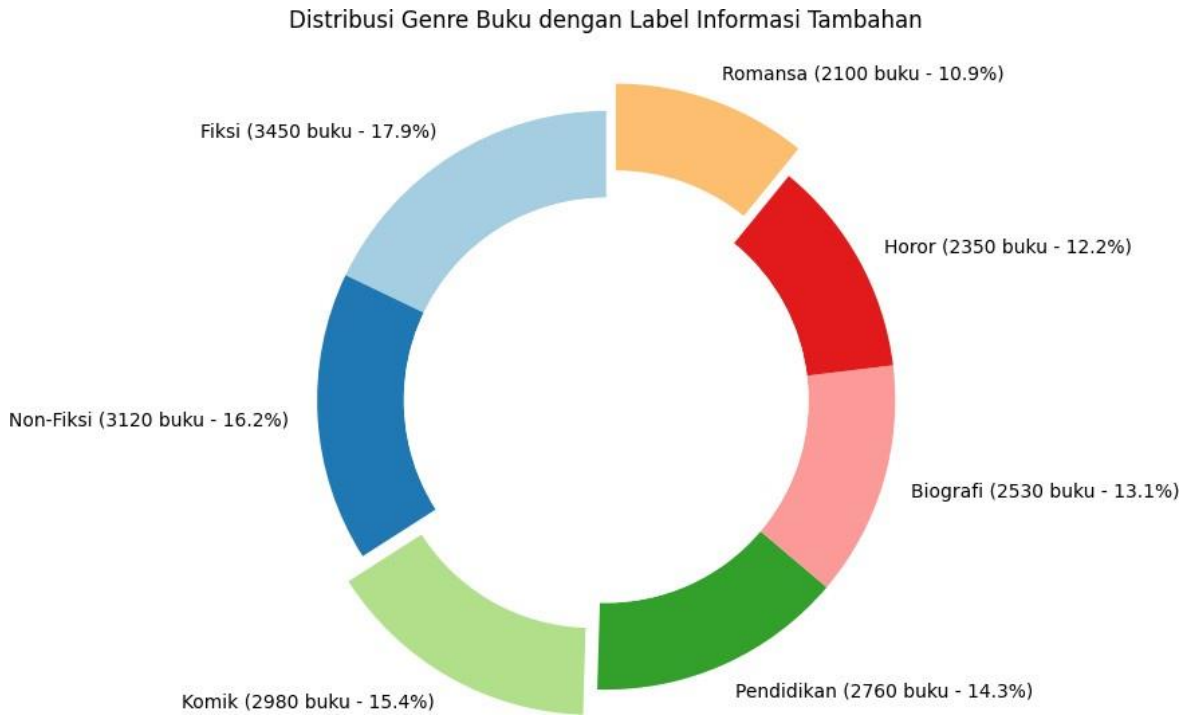
### 2.5.5. Label dengan Informasi Tambahan

Label pada Pie Chart biasanya berisi nama kategori. Namun, kita bisa menambahkan informasi lebih lanjut ke label untuk memberikan konteks tambahan. Misalnya, selain menampilkan persentase, kita juga bisa menampilkan jumlah sebenarnya.

Mari kita coba menambahkan jumlah penjualan sebenarnya ke label Pie Chart kita.

```
# Membuat label dengan informasi tambahan
labels_dengan_info = [f"{genre} ({jumlah} buku -
{jumlah/df_penjualan['Penjualan'].sum()*100:.1f}%)"
                      for genre, jumlah in zip(df_penjualan['Genre'],
df_penjualan['Penjualan'])]

plt.figure(figsize=(10, 7))
plt.pie(df_penjualan['Penjualan'], labels=labels_dengan_info,
startangle=90, explode=explode, colors=plt.cm.Paired.colors,
wedgeprops=dict(width=0.3))
plt.gca().add_artist(plt.Circle((0,0),0.70,fc='white'))
plt.title('Distribusi Genre Buku dengan Label Informasi Tambahan')
plt.show()
```

Output:




Dengan menambahkan informasi jumlah buku dan persentasenya pada setiap label, pembaca sekarang memiliki gambaran yang lebih lengkap tentang distribusi penjualan buku. Hal ini memungkinkan pembaca untuk mendapatkan detail yang lebih mendalam hanya dengan melihat Pie Chart, tanpa perlu merujuk ke tabel atau sumber data lainnya.

## 2.6. Area Plot

Bayangkan kamu sedang berdiri di tepi bukit yang luas, menatap ke arah horizon. Lalu, kamu melihat guratan-guratan yang menandakan ketinggian dan kedalaman dari setiap bagian bukit tersebut. Area Plot, atau yang sering dikenal sebagai Area Chart, memberikan visualisasi yang serupa dengan guratan-guratan tersebut, di mana area di bawah garis diisi dengan warna atau pola tertentu.

**Mengenal Area Plot**

Area Plot adalah variasi dari Line Plot. Sama seperti Line Plot, Area Plot digunakan untuk memvisualisasikan data deret waktu. Namun, perbedaannya terletak pada pengisian area di bawah garis dengan warna atau pola, memberikan kesan "kedalaman" dan menekankan besarnya suatu nilai relatif terhadap sumbu horizontal.

**Kapan Menggunakan Area Plot?**

Area Plot paling efektif ketika kamu ingin menampilkan dan membandingkan bagian-bagian dari keseluruhan sepanjang waktu. Beberapa penggunaan umum meliputi:

- Menunjukkan komposisi dari data sepanjang waktu.
- Memvisualisasikan volume total seiring berjalannya waktu.
- Menyoroti perbedaan antara beberapa seri data.

### 2.6.1. Basic Area Plot

Mari kita mulai dengan contoh sederhana. Bayangkan kamu memiliki data penjualan buku per bulan selama satu tahun. Kamu ingin mengetahui bagaimana tren penjualan tersebut berjalan. Mari kita buat data sintetisnya terlebih dahulu.

```
import pandas as pd
import numpy as np

# Membuat data sintetis
np.random.seed(42)
bulan = ['Jan', 'Feb', 'Mar', 'Apr', 'Mei', 'Jun', 'Jul', 'Agu', 'Sep',
'Okt', 'Nov', 'Des']
penjualan = np.random.randint(500, 1500, 12).tolist()

df_penjualan_buku = pd.DataFrame({
    'Bulan': bulan,
    'Penjualan': penjualan
})

df_penjualan_buku
```

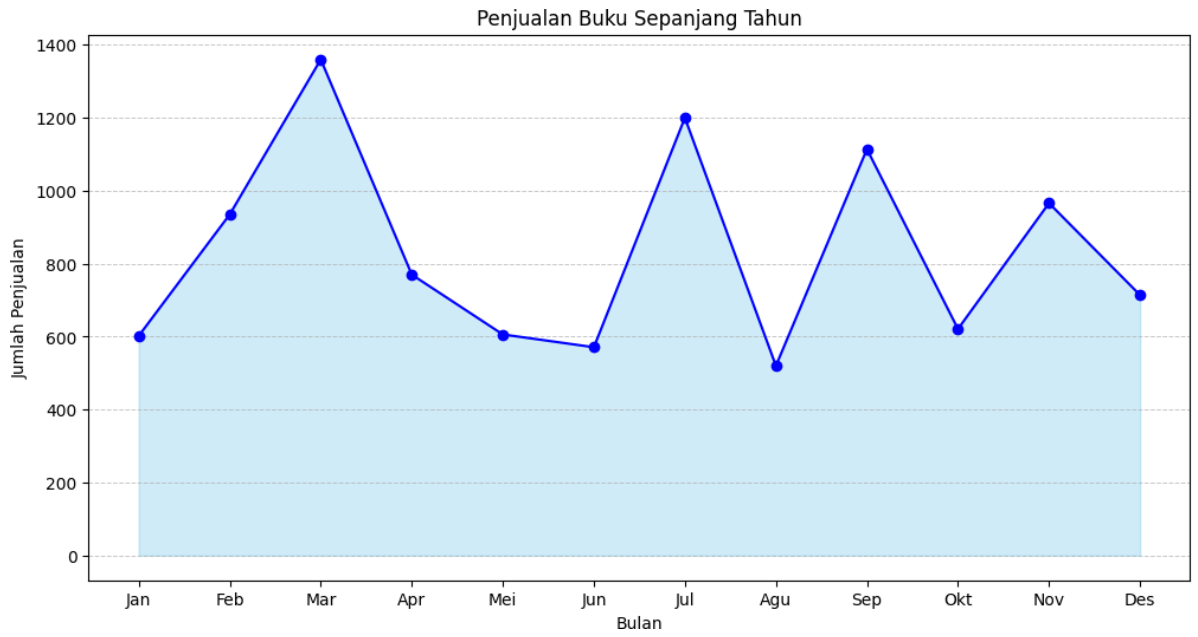
Output:

| | Bulan | Penjualan |
|---|---|---|
| 0 | Jan | 602 |
| 1 | Feb | 935 |
| 2 | Mar | 1360 |
| 3 | Apr | 770 |
| 4 | Mei | 606 |
| 5 | Jun | 571 |
| 6 | Jul | 1200 |
| 7 | Agu | 520 |
| 8 | Sep | 1114 |
| 9 | Okt | 621 |
| 10 | Nov | 966 |
| 11 | Des | 714 |

Dari data di atas, kita memiliki informasi mengenai penjualan buku untuk setiap bulan selama satu tahun. Mari kita visualisasikan data ini dengan Area Plot sederhana untuk memahami tren penjualan buku sepanjang tahun.

```
plt.figure(figsize=(12, 6))
plt.fill_between(df_penjualan_buku['Bulan'],
df_penjualan_buku['Penjualan'], color='skyblue', alpha=0.4)
plt.plot(df_penjualan_buku['Bulan'], df_penjualan_buku['Penjualan'],
color='blue', marker='o')
plt.title('Penjualan Buku Sepanjang Tahun')
plt.xlabel('Bulan')
plt.ylabel('Jumlah Penjualan')
plt.grid(True, axis='y', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Voila! Area Plot di atas menunjukkan visualisasi penjualan buku dari bulan Januari hingga Desember. Area yang diisi warna biru muda menandakan volume penjualan, sementara garis biru menunjukkan tren penjualan. Melalui visualisasi ini, kamu dapat dengan cepat melihat bahwa ada peningkatan penjualan signifikan di bulan Maret dan Juli, serta penurunan di bulan Agustus.

### 2.6.2. Stacked Area Plot

Ketika memiliki beberapa kategori data yang ingin ditampilkan dalam satu Area Plot, Stacked Area Plot adalah pilihan yang tepat. Misalnya, kamu memiliki data penjualan buku berdasarkan genre: Fiksi, Non-Fiksi, dan Komik. Dengan Stacked Area Plot, kamu dapat melihat bagaimana masing-masing genre berkontribusi terhadap total penjualan sepanjang tahun.

Mari kita ciptakan data sintetis untuk penjualan buku berdasarkan genre dan visualisasikan dengan Stacked Area Plot.

```
# Membuat data sintetis untuk genre buku
penjualan_fiksi = np.random.randint(100, 500, 12).tolist()
penjualan_non_fiksi = np.random.randint(50, 300, 12).tolist()
penjualan_komik = np.random.randint(10, 150, 12).tolist()

df_penjualan_genre = pd.DataFrame({
    'Bulan': bulan,
    'Fiksi': penjualan_fiksi,
    'Non-Fiksi': penjualan_non_fiksi,
    'Komik': penjualan_komik
})
```

```
df_penjualan_genre
```

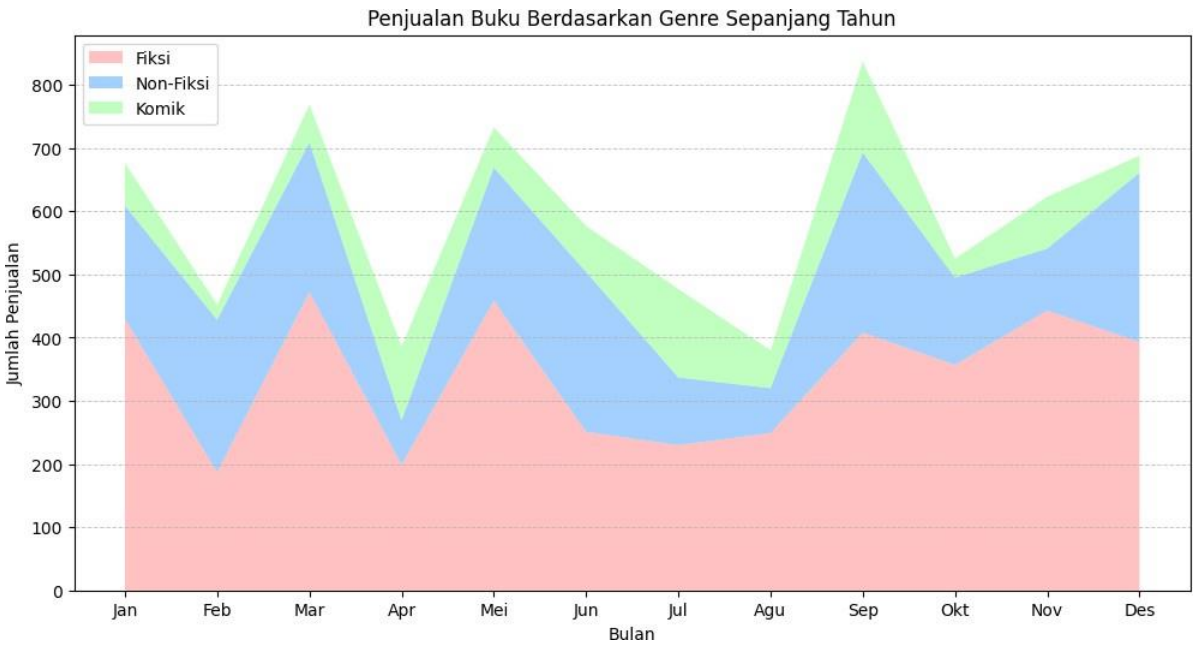
Output:

| | Bulan | Fiksi | Non-Fiksi | Komik |
|---|---|---|---|---|
| 0 | Jan | 430 | 179 | 68 |
| 1 | Feb | 187 | 241 | 24 |
| 2 | Mar | 472 | 237 | 60 |
| 3 | Apr | 199 | 70 | 117 |
| 4 | Mei | 459 | 210 | 64 |
| 5 | Jun | 251 | 253 | 73 |
| 6 | Jul | 230 | 107 | 140 |
| 7 | Agu | 249 | 71 | 60 |
| 8 | Sep | 408 | 285 | 144 |
| 9 | Okt | 357 | 138 | 30 |
| 10 | Nov | 443 | 98 | 82 |
| 11 | Des | 393 | 268 | 27 |

Kini kita memiliki data penjualan buku untuk tiga genre berbeda sepanjang tahun. Mari kita visualisasikan data tersebut dengan Stacked Area Plot untuk melihat kontribusi masing-masing genre terhadap total penjualan setiap bulannya.

```
plt.figure(figsize=(12, 6))

# Membuat Stacked Area Plot
plt.stackplot(df_penjualan_genre['Bulan'], df_penjualan_genre['Fiksi'],
df_penjualan_genre['Non-Fiksi'], df_penjualan_genre['Komik'],
              labels=['Fiksi', 'Non-Fiksi', 'Komik'], colors=['#FF9999',
'#66B2FF', '#99FF99'], alpha=0.6)

plt.title('Penjualan Buku Berdasarkan Genre Sepanjang Tahun')
plt.xlabel('Bulan')
plt.ylabel('Jumlah Penjualan')
plt.legend(loc='upper left')
plt.grid(True, axis='y', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dari Stacked Area Plot di atas, kamu dapat melihat bagaimana setiap genre buku berkontribusi terhadap penjualan total setiap bulannya. Sebagai contoh, buku bergenre Fiksi memiliki penjualan yang paling tinggi di sebagian besar bulan, diikuti oleh Non-Fiksi dan Komik.

Warna yang berbeda memungkinkan kamu untuk dengan cepat membedakan antara genre, dan area yang bertumpuk menunjukkan bagaimana komposisi penjualan berubah dari bulan ke bulan.

### 2.6.3. Area Plot dengan Transparansi

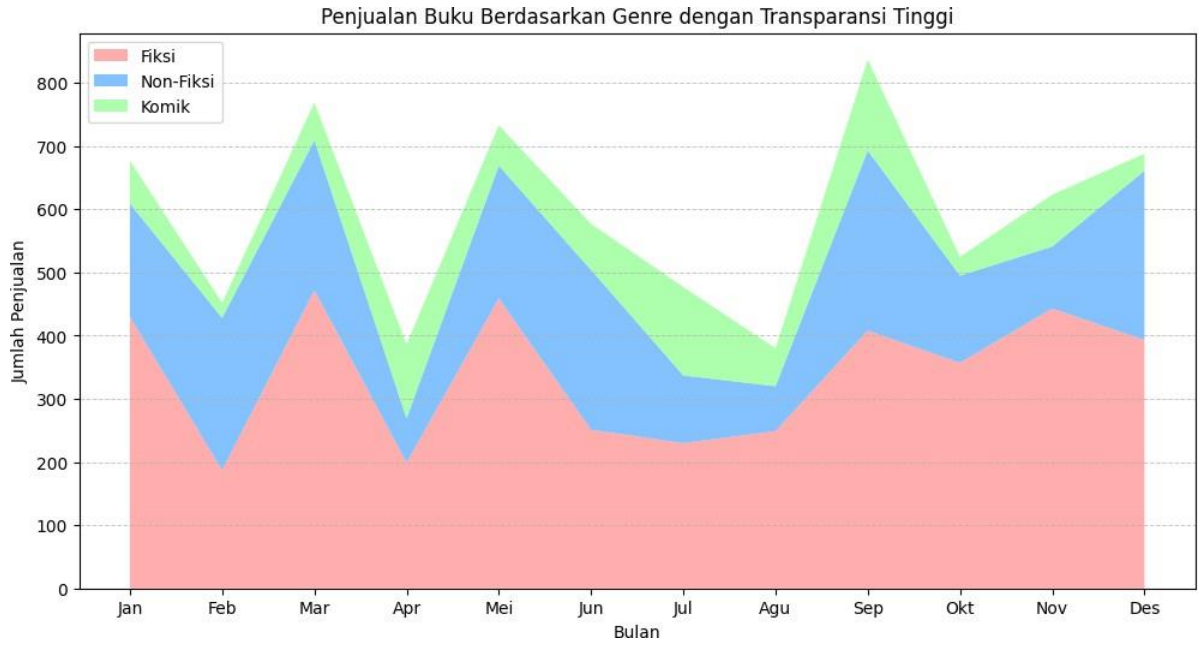
Mengatur transparansi area dapat membantu dalam membaca grafik, terutama ketika memiliki banyak kategori yang ditumpuk bersama. Dengan meningkatkan transparansi, potongan-potongan dari setiap kategori yang berada di bawah kategori lainnya menjadi lebih terlihat.

Mari kita lihat bagaimana menambahkan transparansi pada Area Plot kita.

```
plt.figure(figsize=(12, 6))

# Membuat Stacked Area Plot dengan transparansi yang lebih tinggi
plt.stackplot(df_penjualan_genre['Bulan'], df_penjualan_genre['Fiksi'],
df_penjualan_genre['Non-Fiksi'], df_penjualan_genre['Komik'],
              labels=['Fiksi', 'Non-Fiksi', 'Komik'], colors=['#FF9999',
'#66B2FF', '#99FF99'], alpha=0.8)

plt.title('Penjualan Buku Berdasarkan Genre dengan Transparansi Tinggi')
plt.xlabel('Bulan')
```

```
plt.ylabel('Jumlah Penjualan')
plt.legend(loc='upper left')
plt.grid(True, axis='y', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dengan meningkatkan transparansi, perbedaan antar-genre menjadi lebih jelas, terutama di area di mana beberapa genre tumpang tindih. Ini membantu dalam memahami proporsi relatif dari masing-masing genre dalam konteks keseluruhan.

Area Plot dengan Basis Negatif

Terkadang, kamu mungkin ingin memvisualisasikan data yang memiliki nilai positif dan negatif. Dalam hal ini, kamu bisa menggunakan basis negatif untuk Area Plot, di mana area di bawah sumbu-x akan diisi untuk nilai negatif.

Sebagai contoh, bayangkan kamu memiliki data perubahan stok buku di gudang setiap bulan. Nilai positif menunjukkan penambahan stok, sementara nilai negatif menunjukkan pengurangan stok. Mari kita ciptakan data sintetis untuk skenario ini.

```
# Membuat data sintetis untuk perubahan stok buku
perubahan_stok = np.random.randint(-200, 200, 12).tolist()

df_perubahan_stok = pd.DataFrame({
    'Bulan': bulan,
    'Perubahan': perubahan_stok
})
```

```
df_perubahan_stok
```

Output:

| | Bulan | Perubahan |
|---|---|---|
| 0 | Jan | 187 |
| 1 | Feb | -112 |
| 2 | Mar | 115 |
| 3 | Apr | -187 |
| 4 | Mei | 41 |
| 5 | Jun | 64 |
| 6 | Jul | 145 |
| 7 | Agu | -148 |
| 8 | Sep | 185 |
| 9 | Okt | 139 |
| 10 | Nov | -109 |
| 11 | Des | 166 |

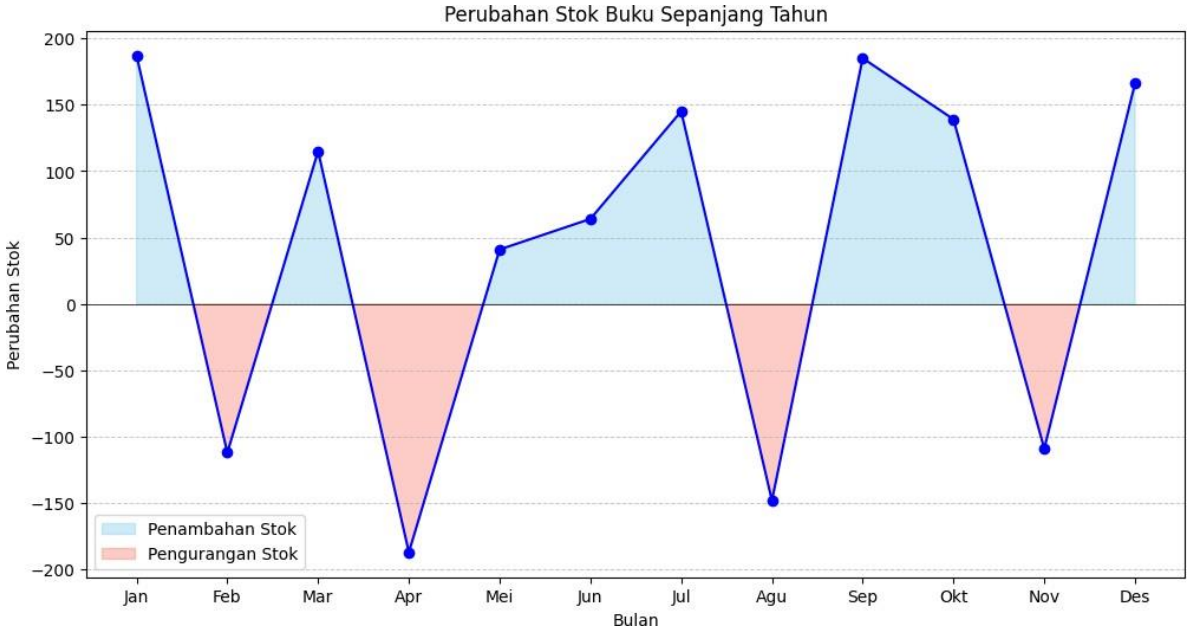
Kita memiliki data perubahan stok buku untuk setiap bulan. Seperti yang bisa kamu lihat, ada bulan-bulan di mana stok buku bertambah (nilai positif) dan ada juga bulan di mana stok berkurang (nilai negatif).

Mari kita visualisasikan data ini dengan Area Plot yang memiliki basis negatif. Ini akan memberikan gambaran visual tentang kapan toko menambah atau mengurangi stok buku mereka.

```
plt.figure(figsize=(12, 6))
plt.fill_between(df_perubahan_stok['Bulan'],
df_perubahan_stok['Perubahan'], color='skyblue',
where=(df_perubahan_stok['Perubahan'] > 0), interpolate=True, alpha=0.4,
label="Penambahan Stok")
plt.fill_between(df_perubahan_stok['Bulan'],
df_perubahan_stok['Perubahan'], color='salmon',
where=(df_perubahan_stok['Perubahan'] < 0), interpolate=True, alpha=0.4,
label="Pengurangan Stok")
plt.plot(df_perubahan_stok['Bulan'], df_perubahan_stok['Perubahan'],
color='blue', marker='o')

plt.title('Perubahan Stok Buku Sepanjang Tahun')
plt.xlabel('Bulan')
plt.ylabel('Perubahan Stok')
plt.axhline(0, color='black',linewidth=0.5)
plt.grid(True, axis='y', linestyle='--', linewidth=0.7, alpha=0.7)
plt.legend()
```

```
plt.show()
```

Output:




Dalam Area Plot di atas, area berwarna biru muda menunjukkan bulan-bulan dengan penambahan stok, sedangkan area berwarna merah muda menunjukkan bulan-bulan dengan pengurangan stok. Dengan memvisualisasikan data dengan cara ini, kamu dengan cepat dapat mengidentifikasi tren dan pola dalam manajemen stok.

## 2.7. Box Plot

Seiring dengan berkembangnya era data, kebutuhan untuk memahami distribusi dan variasi data menjadi semakin penting. Salah satu alat yang paling berguna untuk memahami distribusi data adalah Box Plot (juga dikenal sebagai whisker plot). Box Plot memberikan ringkasan grafis dari satu atau beberapa kelompok data. Plot ini bisa memberikan banyak informasi tentang distribusi, kecenderungan sentral, variabilitas, dan bentuk data.

Apa sih yang membuat Box Plot begitu spesial? Mari kita selami lebih dalam.

**Apa Itu Box Plot?**
Box Plot adalah metode grafis untuk menggambarkan kelompok data numerik melalui kuartil mereka. Box Plot bukan hanya sumur data statistik, tetapi juga visual yang jelas dan detail. Dengan satu pandangan, kamu bisa dengan cepat memahami distribusi data, melihat kemungkinan adanya pencilan (outliers), serta membandingkan distribusi antara beberapa set data.

### 2.7.1. Komponen Box Plot

Sebelum kita mulai membuat Box Plot, penting untuk memahami komponennya:

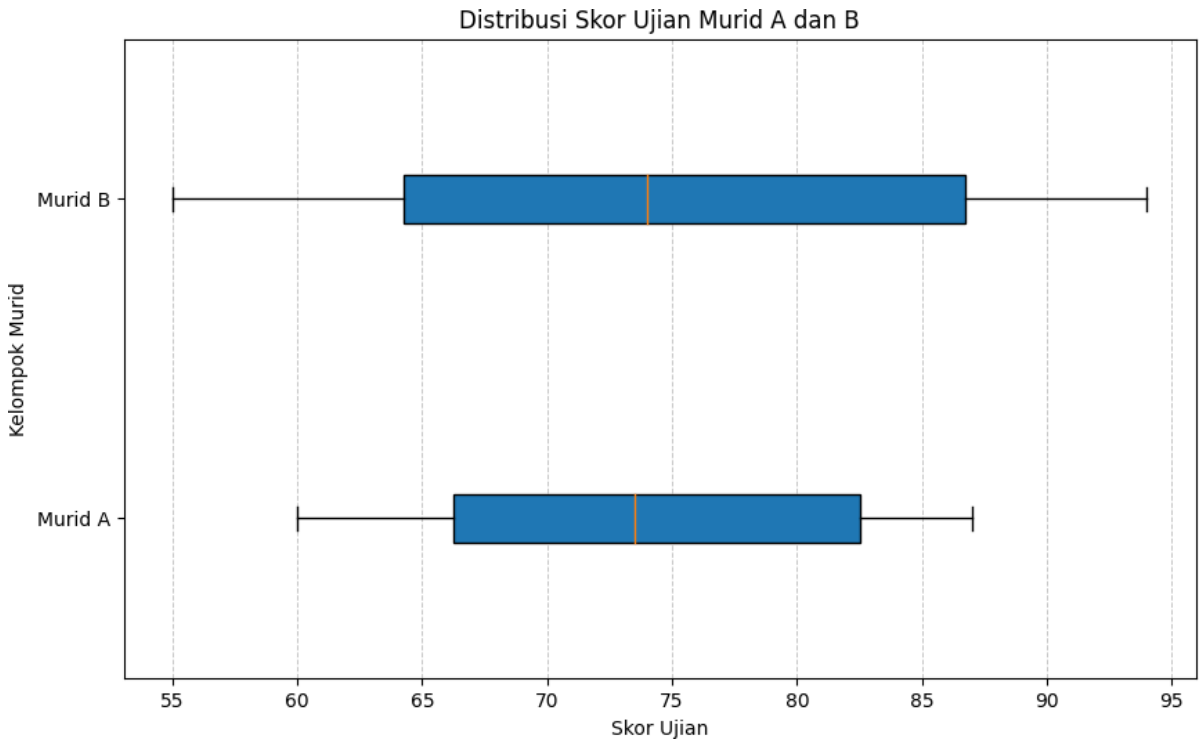
- Median (Q2/50th Persentil): Nilai tengah dataset.
- Kuartil Pertama (Q1/25th Persentil): Median dari setengah data pertama.
- Kuartil Ketiga (Q3/75th Persentil): Median dari setengah data kedua.
- IQR (Interquartile Range): Jarak antara Kuartil Ketiga dan Kuartil Pertama.
- Whiskers: Garis yang keluar dari kotak untuk menunjukkan variasi di luar kuartil atas dan bawah.
- Pencilan: Poin data yang jauh dari whiskers.

Mari kita mulai dengan membuat data sintetis dan menggambarkan Box Plot sederhana untuk memahami bagaimana komponen-komponen ini ditampilkan.

```
# Membuat data sintetis untuk skor ujian
np.random.seed(0)
skor_murid_A = np.random.randint(60, 90, 30)
skor_murid_B = np.random.randint(55, 95, 30)

df_skor = pd.DataFrame({
    'Murid_A': skor_murid_A,
    'Murid_B': skor_murid_B
})

# Menggambar Box Plot sederhana
plt.figure(figsize=(10, 6))
plt.boxplot([df_skor['Murid_A'], df_skor['Murid_B']], vert=False,
patch_artist=True, labels=['Murid A', 'Murid B'])
plt.title('Distribusi Skor Ujian Murid A dan B')
```

```
plt.xlabel('Skor Ujian')
plt.ylabel('Kelompok Murid')
plt.grid(True, axis='x', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dalam Box Plot di atas, kita dapat melihat distribusi skor ujian untuk dua kelompok murid: Murid A dan Murid B. Dari plot ini, kita dapat menarik beberapa kesimpulan:

- Kotak: Bagian tengah dari box plot ini menunjukkan kuartil pertama (Q1) dan kuartil ketiga (Q3) dari data. Lebarnya kotak ini (dalam hal ini, dari kiri ke kanan) menunjukkan jangkauan antarkuartil (IQR) dari data.
- Garis Tengah di Kotak: Garis ini menunjukkan median (Q2) dari data, yang mewakili titik tengah dataset.
- Whiskers: Garis-garis yang keluar dari kotak. Dalam plot ini, whiskers dihitung sebagai 1.5 * IQR. Semua data di luar whiskers ini dianggap sebagai pencilan dan ditandai dengan titik.
- Pencilan: Dalam plot ini, tidak ada pencilan yang terlihat, tetapi jika ada, mereka akan ditandai dengan titik di luar whiskers.

Dengan sekilas pandang, kamu sudah bisa memahami distribusi data, mengetahui mana yang memiliki variasi lebih besar, dan menilai apakah ada pencilan atau tidak. Namun, keajaiban Box Plot tidak berhenti di sini. Plot ini juga memungkinkan kamu untuk membandingkan distribusi dari beberapa kelompok data secara bersamaan, seperti yang kita lihat dengan Murid A dan Murid B.

**Kustomisasi dan Interpretasi**

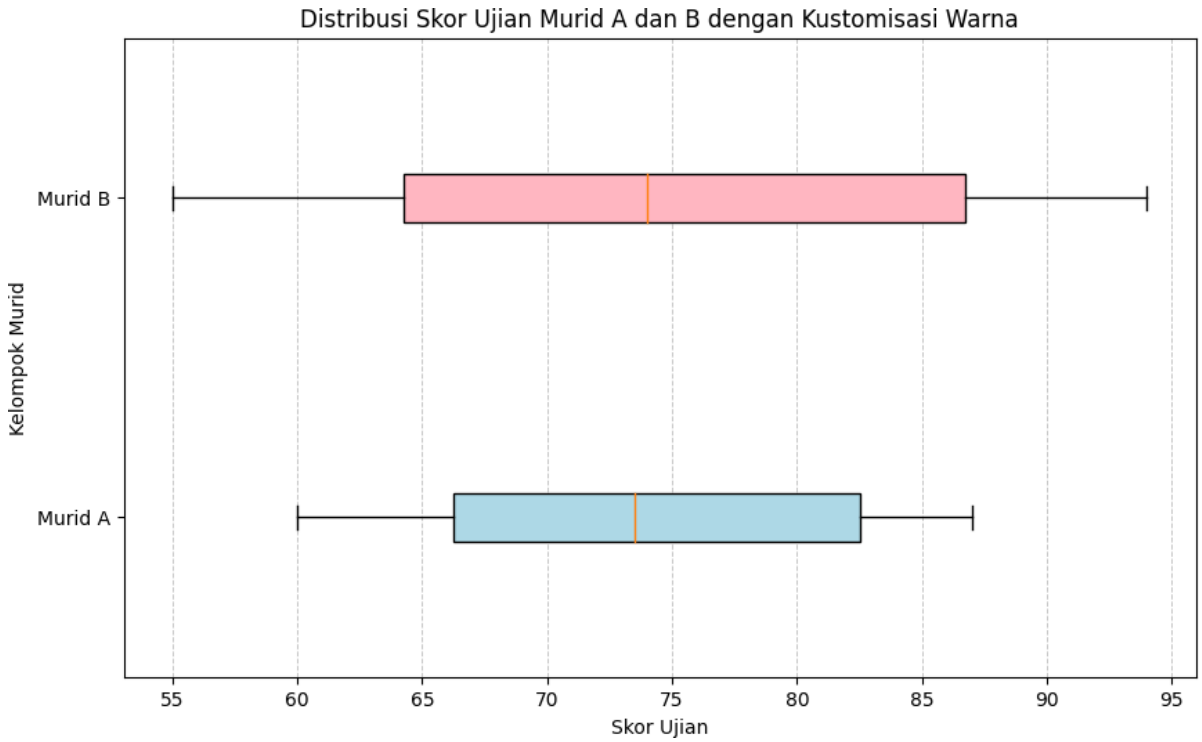
Box Plot, meskipun sederhana, memberikan banyak informasi. Namun, untuk memaksimalkan kegunaannya, penting untuk mengetahui cara mengkustomisasi dan menginterpretasikannya dengan benar.

### 2.7.2. Kustomisasi Warna dan Bentuk

Kustomisasi visual dapat membantu menekankan aspek-aspek tertentu dari data atau menjadikan grafik lebih mudah dibaca. Mari kita coba beberapa kustomisasi pada Box Plot kita.

```
# Menggambar Box Plot dengan kustomisasi
plt.figure(figsize=(10, 6))

# Kustomisasi warna boxplot
colors = ['lightblue', 'lightpink']
bp = plt.boxplot([df_skor['Murid_A'], df_skor['Murid_B']], vert=False,
patch_artist=True, labels=['Murid A', 'Murid B'])

# Mengkustomisasi warna
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)

plt.title('Distribusi Skor Ujian Murid A dan B dengan Kustomisasi
Warna')
plt.xlabel('Skor Ujian')
plt.ylabel('Kelompok Murid')
plt.grid(True, axis='x', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:



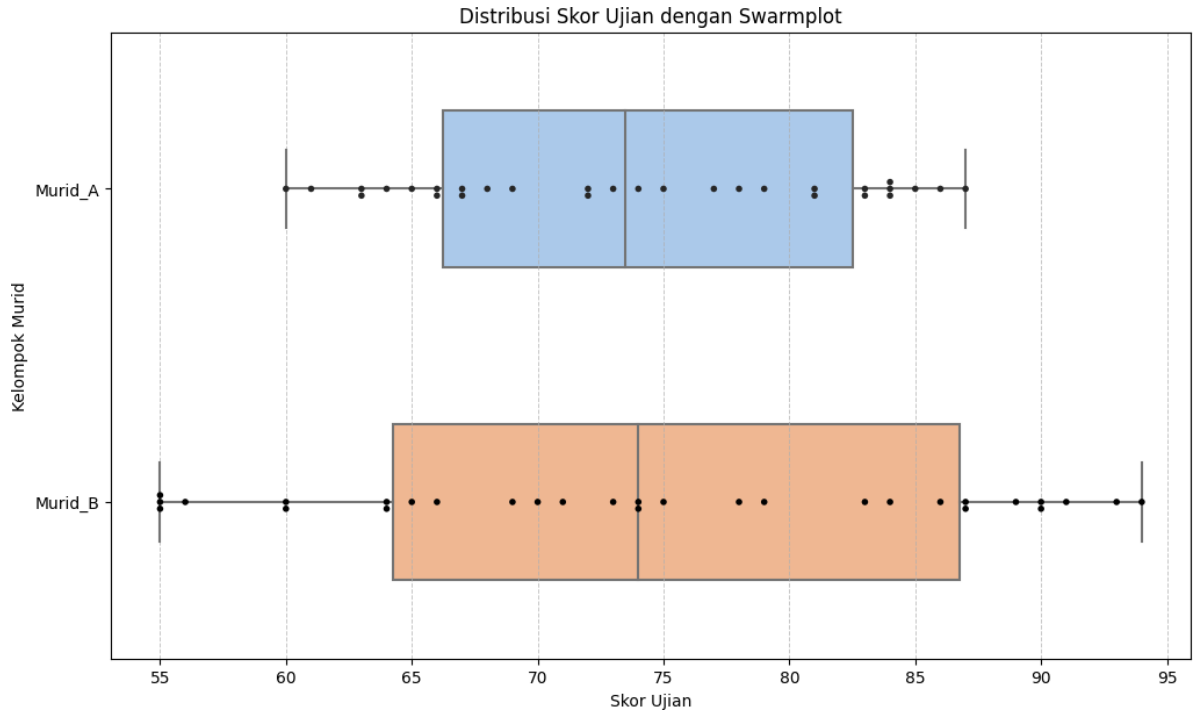

Dengan sedikit kustomisasi warna, Box Plot kita kini tampak lebih menarik dan informatif. Murid A diwakili dengan warna biru muda, sementara Murid B dengan warna merah muda.

### 2.7.3. Menambahkan Swarmplot ke Box Plot

Salah satu kritik terhadap Box Plot adalah bahwa mereka tidak menunjukkan densitas distribusi data. Untungnya, dengan bantuan pustaka seperti seaborn, kita dapat menambahkan "swarmplot" ke Box Plot kita, yang menampilkan setiap titik data individu tanpa menumpuknya. Ini memberikan wawasan tambahan tentang densitas data. Mari kita coba tambahkan swarmplot ke plot kita.

```
import seaborn as sns

plt.figure(figsize=(12, 7))

# Membuat boxplot dengan seaborn untuk kustomisasi lebih lanjut
sns.boxplot(data=df_skor, orient='h', palette='pastel', width=0.5)

# Menambahkan swarmplot ke boxplot
sns.swarmplot(data=df_skor, orient='h', color='k', size=4)

plt.title('Distribusi Skor Ujian dengan Swarmplot')
plt.xlabel('Skor Ujian')
plt.ylabel('Kelompok Murid')
plt.grid(True, axis='x', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dalam visualisasi di atas, swarmplot memungkinkan kita untuk melihat setiap titik data individu. Titik-titik hitam menunjukkan lokasi setiap skor ujian individu. Dengan kombinasi box plot dan swarmplot, kita mendapatkan gambaran yang lebih lengkap tentang distribusi data.

### 2.7.4. Mengidentifikasi Pencilan dengan Box Plot

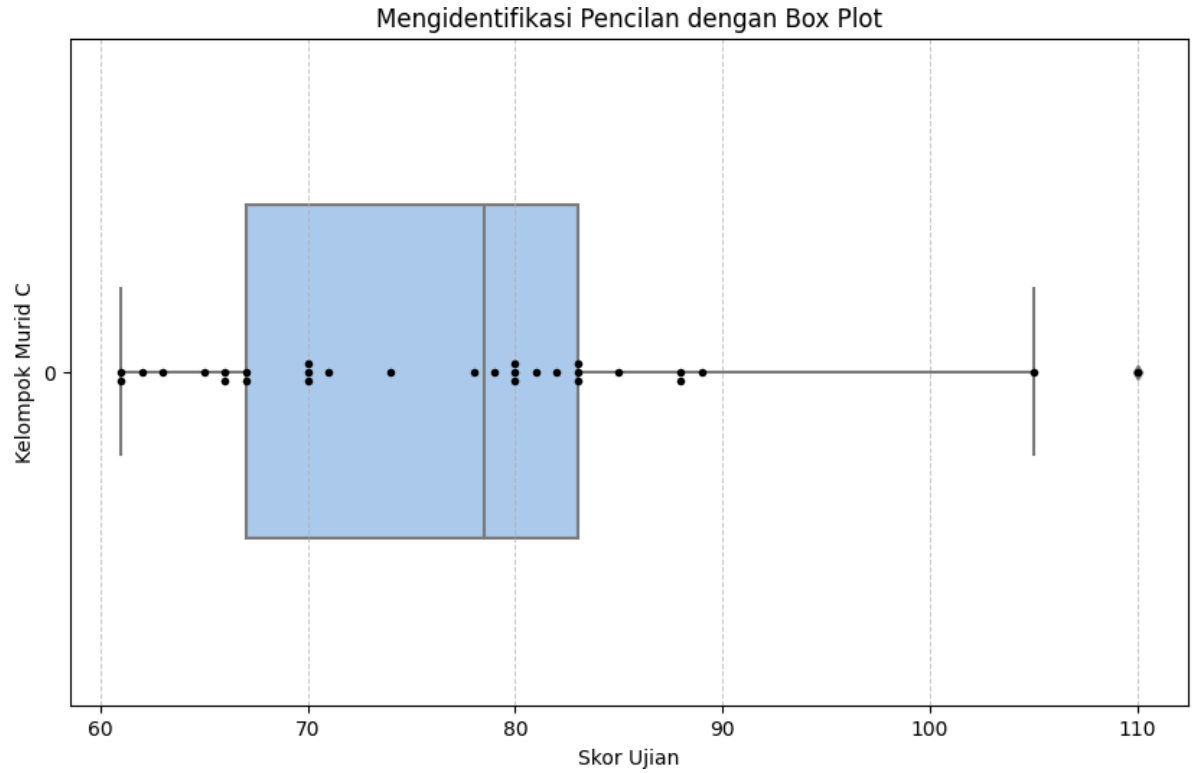
Salah satu kegunaan utama Box Plot adalah kemampuannya untuk dengan mudah mengidentifikasi pencilan. Pencilan adalah titik data yang jauh dari titik data lainnya. Dalam konteks Box Plot, pencilan biasanya ditandai sebagai titik-titik yang berada di luar "whiskers".

Mari kita coba membuat data dengan beberapa pencilan dan visualisasikan dengan Box Plot.

```
# Membuat data dengan pencilan
np.random.seed(42)
skor_murid_C = np.random.randint(60, 90, 28).tolist() + [105, 110]  # 
Menambahkan dua pencilan

df_skor['Murid_C'] = skor_murid_C

# Menggambar Box Plot untuk mengidentifikasi pencilan
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_skor['Murid_C'], orient='h', palette='pastel', 
width=0.5)
```

```
sns.swarmplot(data=df_skor['Murid_C'], orient='h', color='k', size=4)

plt.title('Mengidentifikasi Pencilan dengan Box Plot')
plt.xlabel('Skor Ujian')
plt.ylabel('Kelompok Murid C')
plt.grid(True, axis='x', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Pada visualisasi di atas, kamu dapat dengan jelas melihat dua pencilan (titik-titik yang berada di sebelah kanan whisker). Skor 105 dan 110 jauh dari skor murid lainnya, yang berkisar antara 60 hingga 90. Dengan Box Plot, identifikasi pencilan menjadi sangat sederhana.

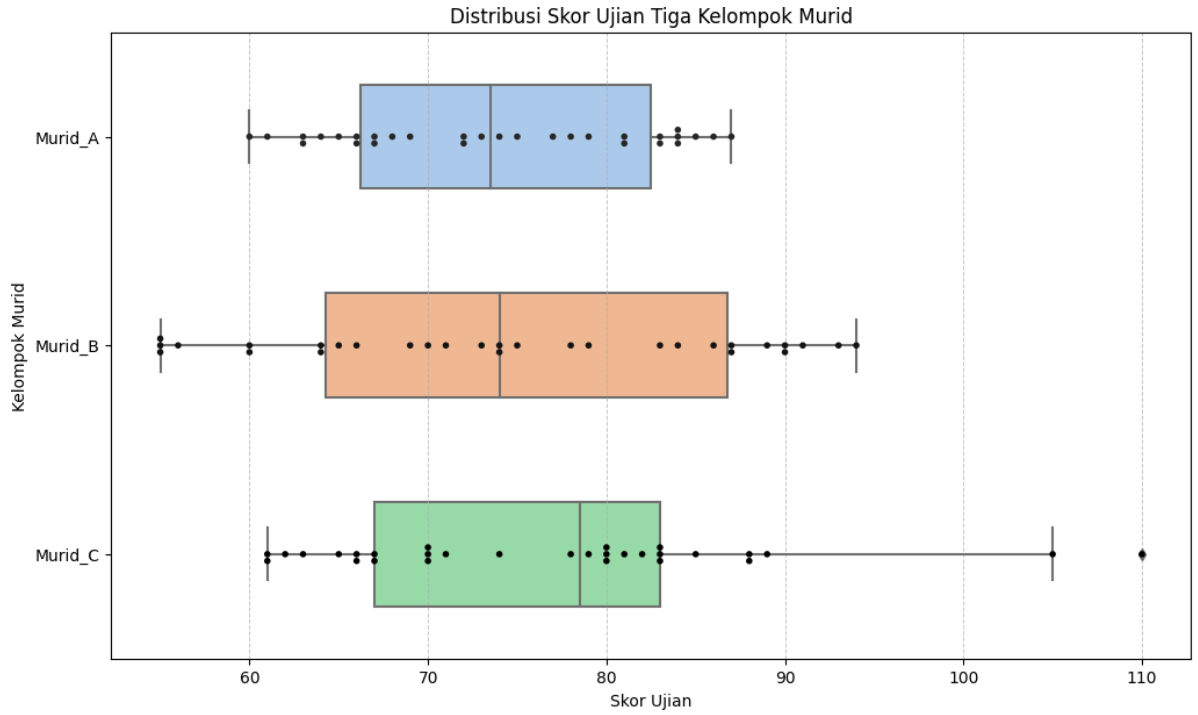
### 2.7.5. Box Plot untuk Membandingkan Beberapa Kelompok

Salah satu keunggulan Box Plot adalah kemampuannya untuk membandingkan distribusi data antara beberapa kelompok dalam satu visualisasi. Bayangkan kamu ingin membandingkan skor ujian dari beberapa kelas di sekolah. Dengan Box Plot, kamu dapat dengan cepat menilai kelas mana yang memiliki median tertinggi, variasi skor yang lebih rendah, atau bahkan kelas mana yang memiliki banyak pencilan.

Mari kita coba memvisualisasikan skor ujian dari tiga kelas berbeda menggunakan Box Plot.

```
# Menggambar Box Plot untuk membandingkan tiga kelompok murid
plt.figure(figsize=(12, 7))

# Membuat boxplot dengan seaborn untuk kustomisasi lebih lanjut
sns.boxplot(data=df_skor, orient='h', palette='pastel', width=0.5)

# Menambahkan swarmplot ke boxplot
sns.swarmplot(data=df_skor, orient='h', color='k', size=4)

plt.title('Distribusi Skor Ujian Tiga Kelompok Murid')
plt.xlabel('Skor Ujian')
plt.ylabel('Kelompok Murid')
plt.grid(True, axis='x', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dari Box Plot di atas, kita bisa mendapatkan wawasan berharga mengenai distribusi skor ujian dari tiga kelompok murid:

- Kelas A (Murid A) cenderung memiliki median skor yang lebih tinggi dibandingkan dengan kelas lainnya, tetapi juga memiliki variasi yang lebih luas.
- Kelas B (Murid B) memiliki median skor yang lebih rendah daripada Kelas A namun lebih tinggi daripada Kelas C, dan variasinya juga lebih sempit.
- Kelas C (Murid C) memiliki median skor yang paling rendah, dengan dua pencilan yang jauh lebih tinggi dari skor murid lainnya di kelas tersebut.

Dengan informasi ini, seorang pendidik mungkin dapat menilai kelas mana yang memerlukan perhatian lebih atau bahkan mengidentifikasi apakah ada murid tertentu yang memerlukan bimbingan lebih lanjut.

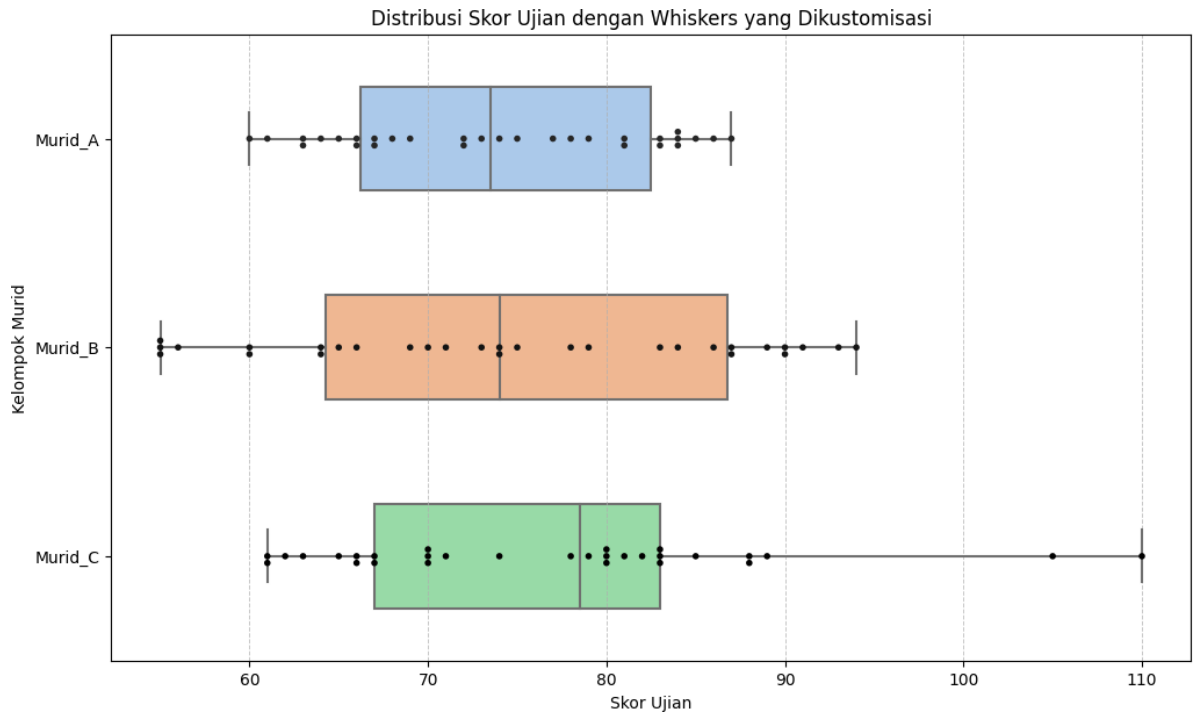
### 2.7.6. Mengkustomisasi Whiskers

Secara default, "whiskers" pada Box Plot di matplotlib dihitung sebagai 1,5 * IQR (Interquartile Range). Namun, terkadang kamu mungkin ingin mengatur panjang whiskers ini sesuai kebutuhan analisis. Misalnya, jika kamu ingin whiskers mencakup semua data tanpa ada pencilan, kamu dapat mengaturnya.

Mari kita coba mengkustomisasi panjang whiskers pada Box Plot kita.

```
# Menggambar Box Plot dengan whiskers yang dikustomisasi untuk mencakup semua data
plt.figure(figsize=(12, 7))

# Membuat boxplot dengan seaborn dengan whiskers yang dikustomisasi
sns.boxplot(data=df_skor, orient='h', palette='pastel', width=0.5, whis=[0, 100])  # Whis diatur dari 0% ke 100%

# Menambahkan swarmplot ke boxplot
sns.swarmplot(data=df_skor, orient='h', color='k', size=4)

plt.title('Distribusi Skor Ujian dengan Whiskers yang Dikustomisasi')
plt.xlabel('Skor Ujian')
plt.ylabel('Kelompok Murid')
plt.grid(True, axis='x', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dengan mengkustomisasi panjang whiskers, Box Plot kita sekarang mencakup semua data, termasuk yang sebelumnya dianggap sebagai pencilan. Dalam kasus ini, whiskers diperpanjang untuk mencakup seluruh rentang data dari 0% hingga 100%.

Ini sangat berguna jika kamu ingin menekankan keseluruhan distribusi data tanpa memberi label khusus pada data tertentu sebagai pencilan. Namun, penting untuk diingat bahwa memodifikasi whiskers dapat mengubah interpretasi plot, jadi selalu pastikan untuk menjelaskan kustomisasi apa pun yang kamu lakukan saat mempresentasikan atau menganalisis data.

### 2.7.7. Box Plot dengan Variabel Kategorikal

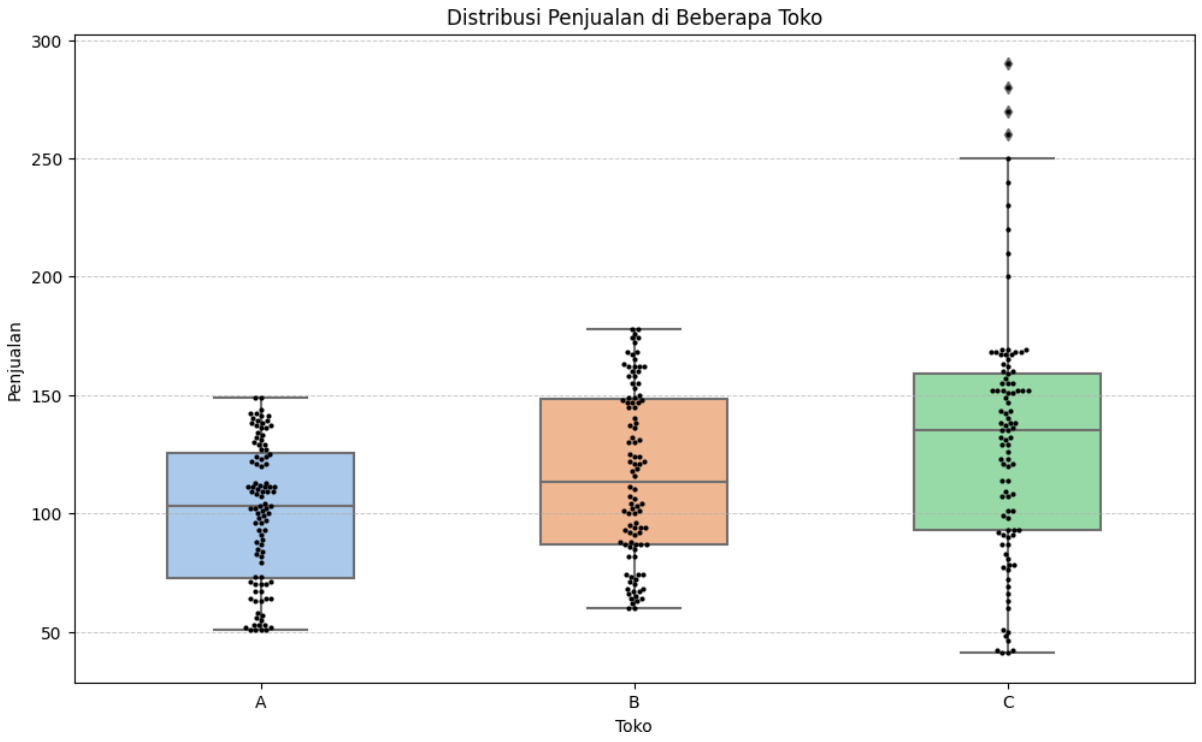
Box Plot tidak hanya berguna untuk membandingkan distribusi dari beberapa kelompok numerik, tetapi juga sangat efektif untuk memvisualisasikan distribusi numerik dalam kaitannya dengan variabel kategorikal. Misalkan kamu memiliki data tentang penjualan produk di beberapa toko. Dengan Box Plot, kamu dapat dengan cepat melihat toko mana yang memiliki distribusi penjualan tertinggi, variasi penjualan, serta potensi pencilan.

Mari kita ciptakan contoh dengan data sintetis untuk memvisualisasikan hal ini.

```
# Membuat data sintetis untuk penjualan di beberapa toko
np.random.seed(42)
toko_A = np.random.randint(50, 150, 100)
toko_B = np.random.randint(60, 180, 100)
toko_C = np.random.randint(40, 170, 90).tolist() + [200, 210, 220, 230,
240, 250, 260, 270, 280, 290]  # Menambahkan beberapa pencilan
```

```
df_penjualan = pd.DataFrame({
    'Toko': ['A']*100 + ['B']*100 + ['C']*100,
    'Penjualan': toko_A.tolist() + toko_B.tolist() + toko_C
})

# Menggambar Box Plot untuk membandingkan penjualan di beberapa toko
plt.figure(figsize=(12, 7))
sns.boxplot(x='Toko', y='Penjualan', data=df_penjualan,
palette='pastel', width=0.5)
sns.swarmplot(x='Toko', y='Penjualan', data=df_penjualan, color='k',
size=3)

plt.title('Distribusi Penjualan di Beberapa Toko')
plt.xlabel('Toko')
plt.ylabel('Penjualan')
plt.grid(True, axis='y', linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dari Box Plot di atas, kita dapat memahami berbagai aspek distribusi penjualan di tiga toko:

- Toko A memiliki median penjualan yang lebih rendah dibandingkan dengan toko lainnya dan variasi penjualannya juga relatif sempit.
- Toko B memiliki median penjualan yang lebih tinggi dan variasi yang lebih luas dibandingkan dengan Toko A.

- Toko C memiliki median penjualan yang hampir sama dengan Toko B, tetapi ada beberapa pencilan yang menunjukkan penjualan yang sangat tinggi pada beberapa hari tertentu.

Kombinasi Box Plot dengan swarmplot memberikan gambaran yang jelas tentang distribusi penjualan di setiap toko serta densitas penjualan pada setiap level penjualan. Ini sangat berguna untuk memahami kinerja toko dan menentukan strategi penjualan di masa mendatang.

### 2.7.8. Box Plot untuk Analisis Multivariat

Box Plot tidak hanya terbatas pada analisis univariat atau bivariat. Kamu juga bisa memanfaatkan Box Plot dalam analisis multivariat untuk membandingkan distribusi data numerik terhadap lebih dari satu variabel kategorikal.

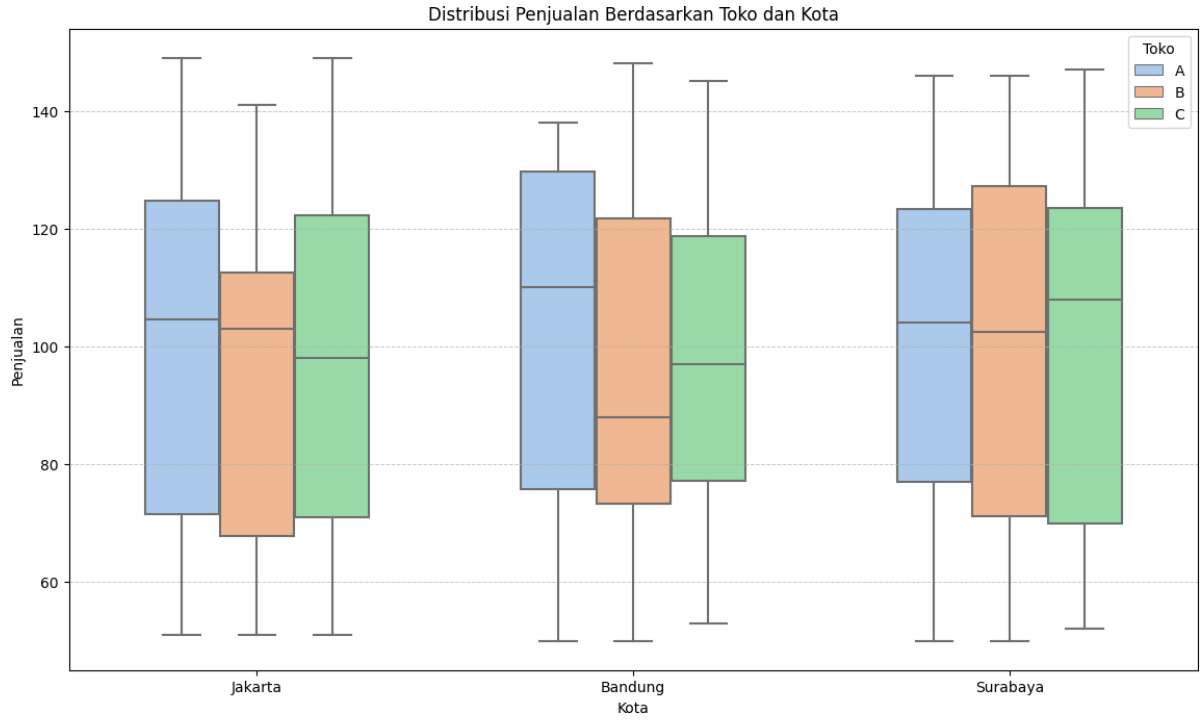
Misalnya, anggap kamu memiliki data penjualan dari beberapa toko di berbagai kota. Dengan Box Plot, kamu dapat membandingkan distribusi penjualan berdasarkan toko dan kota dalam satu visualisasi.

Mari kita ciptakan contoh dengan data sintetis untuk memvisualisasikan hal ini.

```
# Membuat data sintetis untuk penjualan di beberapa toko di beberapa
kota
np.random.seed(42)
kota = ['Jakarta', 'Bandung', 'Surabaya']
toko = ['A', 'B', 'C']

data = []
for k in  kota:
    for t in toko:
        jumlah_penjualan = np.random.randint(50, 150, 30)
        for j in jumlah_penjualan:
            data.append([k, t, j])

df_multivariat = pd.DataFrame(data, columns=['Kota', 'Toko',
'Penjualan'])

# Menggambar Box Plot untuk analisis multivariat
plt.figure(figsize=(14, 8))
sns.boxplot(x='Kota', y='Penjualan', hue='Toko', data=df_multivariat,
palette='pastel', width=0.6)

plt.title('Distribusi Penjualan Berdasarkan Toko dan Kota')
plt.xlabel('Kota')
plt.ylabel('Penjualan')
plt.grid(True, axis='y', linestyle='--', linewidth=0.7, alpha=0.7)
```

```
plt.legend(title='Toko', loc='upper right')
plt.show()
```

Output:




Dalam visualisasi di atas, Box Plot memberi kita kemampuan untuk memahami distribusi penjualan berdasarkan dua dimensi: toko dan kota. Sehingga, kita dapat menarik beberapa kesimpulan:

- Jakarta: Di kota ini, Toko C memiliki median penjualan yang paling tinggi, diikuti oleh Toko B dan Toko A.
- Bandung: Semua toko memiliki median penjualan yang hampir serupa, tetapi Toko C memiliki variasi penjualan yang lebih luas dibandingkan dengan toko lainnya.
- Surabaya: Toko A tampaknya mendominasi dengan median penjualan tertinggi, sementara Toko B dan Toko C memiliki median yang lebih rendah.

Analisis multivariat dengan Box Plot ini memberikan gambaran mendalam tentang bagaimana distribusi penjualan di setiap toko bervariasi di berbagai kota. Informasi ini bisa menjadi sangat berharga bagi pemilik bisnis untuk menilai kinerja masing-masing toko di berbagai lokasi dan merencanakan strategi pemasaran yang lebih efektif.

## 2.9. Contour Plot

Apakah kamu pernah melihat peta yang menampilkan garis-garis berkelok-kelok yang tampak seperti jalur di peta topografi? Itulah contoh dari contour plot. Sebuah contour plot atau plot kontur adalah representasi grafis dari tiga variabel kontinu di dua dimensi. Dua dari variabel tersebut diwakili oleh sumbu X dan Y, sedangkan yang ketiga diwakili oleh kontur garis atau warna diplot.

**Apa Itu Contour Plot?**

Contour plot adalah cara visualisasi 3D data dalam 2 dimensi. Ini bekerja dengan menempatkan kontur di sekitar data kamu di mana setiap kontur mewakili area di mana variabel ke-3 memiliki nilai tertentu. Contoh paling umum dari ini adalah peta topografi, di mana setiap kontur mewakili ketinggian tertentu.

Namun, dalam analisis data, contour plot sering digunakan untuk mengeksplorasi hubungan antara tiga variabel kontinu. Sebagai contoh, jika kamu ingin memahami hubungan antara suhu, kelembaban, dan jumlah produk yang terjual, contour plot adalah alat yang sempurna.

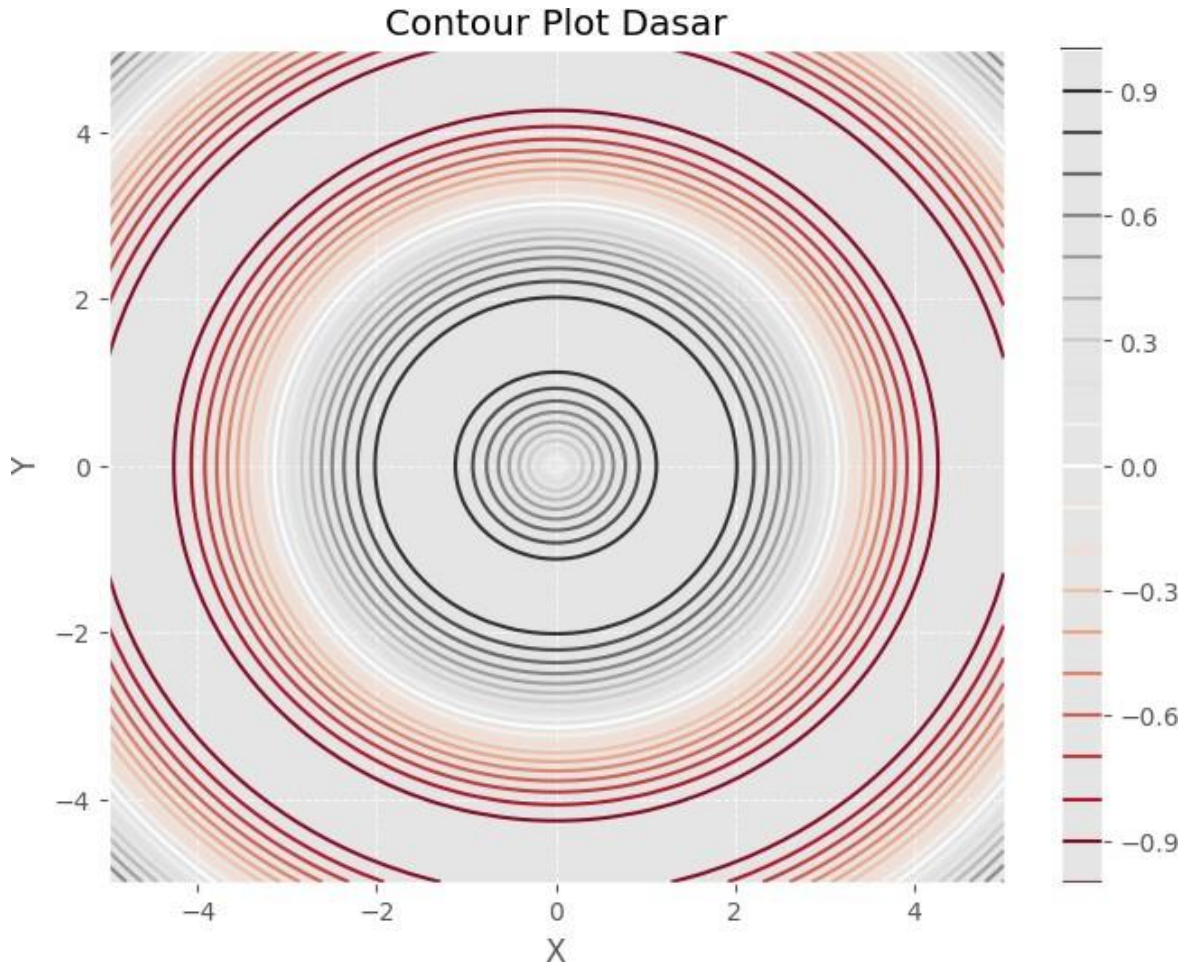
### 2.9.1. Basic Contour Plot

Mari kita mulai dengan contoh sederhana untuk memahami cara kerja contour plot. Kita akan menggunakan data sintetis yang mewakili dua variabel independen, X dan Y, dan satu variabel dependen, Z.

```
# Membuat data sintetis
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Menggambar Contour Plot
plt.figure(figsize=(8, 6))
contour = plt.contour(X, Y, Z, 20, cmap='RdGy')
plt.colorbar(contour)
plt.title('Contour Plot Dasar')
plt.xlabel('X')
plt.ylabel('Y')
```

```
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Pada plot kontur di atas, garis-garis kontur menunjukkan area di mana fungsi Z memiliki nilai tertentu. Warna yang lebih gelap menunjukkan area dengan nilai Z yang lebih rendah, sedangkan area yang lebih terang menunjukkan nilai Z yang lebih tinggi. Dengan kata lain, kamu bisa membayangkan plot kontur ini seperti peta topografi, di mana garis kontur mewakili ketinggian.

### 2.9.2. Memahami Interval Kontur

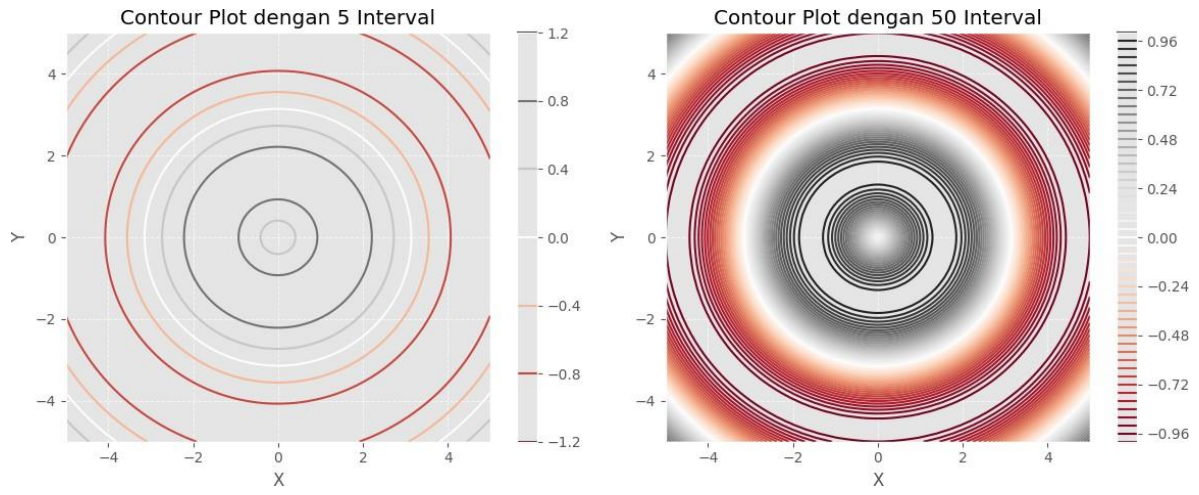
Salah satu aspek penting dari plot kontur adalah interval antara garis kontur. Dalam plot di atas, kita menggunakan 20 interval, tetapi tergantung pada data dan apa yang ingin kamu capai, kamu mungkin ingin mengatur interval ini.

Sebagai contoh, jika kamu memiliki data dengan variasi yang sangat halus, mungkin kamu ingin lebih banyak garis kontur untuk menyoroti perbedaan tersebut. Sebaliknya, jika kamu hanya tertarik pada tren umum, beberapa garis kontur mungkin sudah cukup.

Mari kita lihat efek dari mengubah jumlah interval kontur pada visualisasi kita.

```
# Menggambar Contour Plot dengan interval yang berbeda
plt.figure(figsize=(12, 5))

# Dengan 5 interval
plt.subplot(1, 2, 1)
contour5 = plt.contour(X, Y, Z, 5, cmap='RdGy')
plt.colorbar(contour5)
plt.title('Contour Plot dengan 5 Interval')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)

# Dengan 50 interval
plt.subplot(1, 2, 2)
contour50 = plt.contour(X, Y, Z, 50, cmap='RdGy')
plt.colorbar(contour50)
plt.title('Contour Plot dengan 50 Interval')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)

plt.tight_layout()
plt.show()
```

Output:




Dari kedua visualisasi di atas, kita dapat melihat perbedaan yang signifikan ketika mengubah jumlah interval kontur:

Dengan hanya 5 interval, plot kontur memberikan gambaran umum tentang distribusi data. Ini mungkin berguna jika kamu hanya ingin melihat pola umum dalam data.
Dengan 50 interval, plot kontur memberikan detail yang jauh lebih banyak, mengungkapkan nuansa halus dalam distribusi data. Meskipun ini memberikan informasi lebih banyak, mungkin juga terasa sedikit berlebihan tergantung pada konteks analisis kamu.

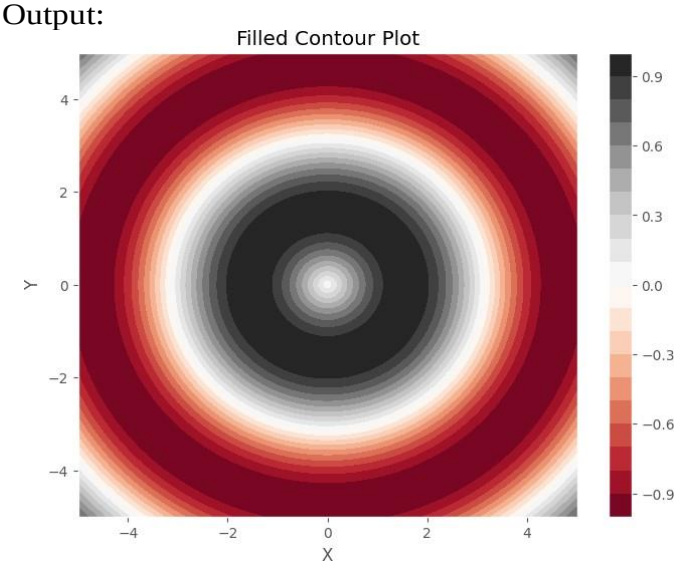
### 2.9.3. Contourf: Filled Contour Plot

Selain garis kontur biasa, Matplotlib juga menyediakan contourf yang memungkinkan kita untuk membuat "filled contour plot". Ini memungkinkan kita untuk menampilkan data dengan cara yang lebih visual dengan mengisi area antara garis kontur dengan warna.

Mari kita lihat contoh dari filled contour plot.

```
# Menggambar Filled Contour Plot
plt.figure(figsize=(8, 6))
filled_contour = plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar(filled_contour)
plt.title('Filled Contour Plot')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dalam "Filled Contour Plot" ini, selain garis kontur, kita juga melihat area antara garis-garis tersebut yang diisi dengan warna. Warna yang berbeda mewakili rentang nilai yang berbeda dari variabel Z. Ini memberikan gambaran yang lebih intuitif tentang bagaimana variabel Z berubah seiring dengan X dan Y.

Dengan menggunakan plot semacam ini:

- Kamu dapat dengan cepat mengidentifikasi area di mana fungsi Z memiliki nilai tertinggi atau terendah.

- Kamu juga mendapatkan gambaran visual yang jelas tentang bagaimana Z berubah di seluruh domain.

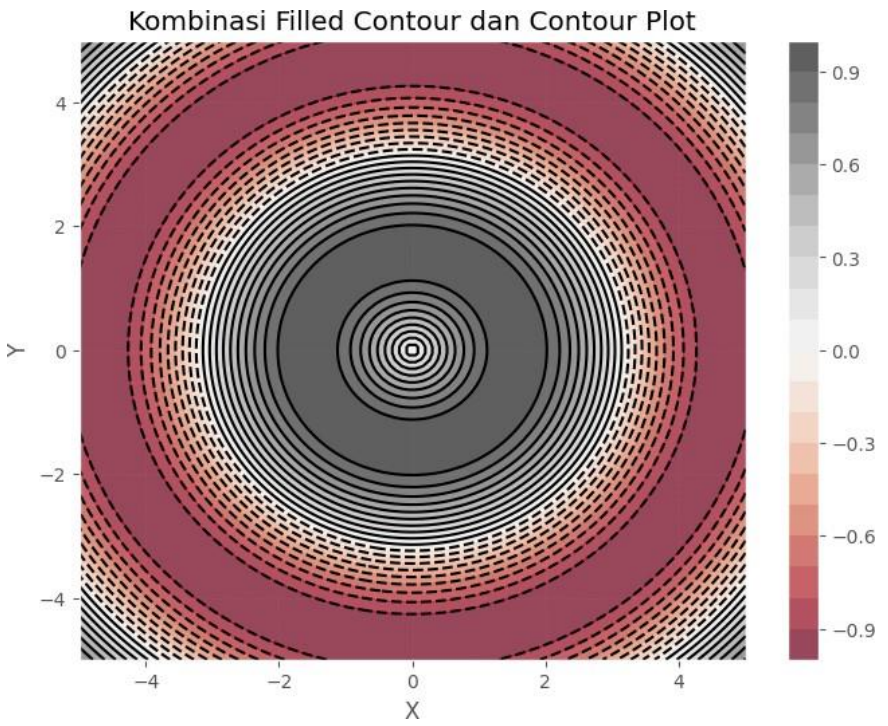
### 2.9.4. Kombinasi Contour dan Filled Contour Plot

Ada kalanya menggabungkan kedua jenis plot kontur ini bisa sangat bermanfaat. Dengan menggabungkan filled contour plot dan garis kontur, kita mendapatkan kejelasan dari garis kontur dan visualisasi intuitif dari filled contour plot.

Mari kita coba menggabungkan keduanya.

```
# Menggambar kombinasi dari Filled Contour Plot dan Contour Plot

plt.figure(figsize=(8, 6))
filled_contour = plt.contourf(X, Y, Z, 20, cmap='RdGy', alpha=0.7)
contour = plt.contour(X, Y, Z, 20, colors='black')
plt.colorbar(filled_contour)
plt.title('Kombinasi Filled Contour dan Contour Plot')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dalam visualisasi di atas, kita memiliki filled contour plot sebagai latar belakang dengan garis kontur yang ditambahkan di atasnya. Garis kontur hitam memberikan kontras yang jelas dengan warna-warna dari filled contour plot, memungkinkan kita untuk dengan mudah melihat dan menginterpretasi tingkat kontur tertentu. Menggabungkan kedua jenis plot kontur ini memberikan kita visualisasi yang penuh informasi namun tetap mudah dibaca.

## 2.10. Hexbin Plot

Seiring dengan bertambahnya volume data yang kita miliki, visualisasi data menjadi lebih menantang. Pada situasi di mana kita memiliki banyak titik data yang tumpang tindih, scatter plot mungkin tidak lagi informatif. Di sinilah Hexbin Plot datang untuk menyelamatkan.

**Apa Itu Hexbin Plot?**

Hexbin plot, atau histogram binning heksagonal, adalah metode untuk memvisualisasikan distribusi 2D dari banyak titik data. Alih-alih menunjukkan setiap titik sebagai titik individu, seperti yang dilakukan scatter plot, hexbin plot mengelompokkan titik-titik ke dalam sel heksagonal dan kemudian memberi warna setiap sel berdasarkan jumlah titik dalam sel tersebut. Dengan kata lain, sel heksagonal berfungsi seperti 'keranjang' yang mengumpulkan titik-titik data.

**Mengapa Hexagonal?**

Mungkin kamu bertanya-tanya, mengapa kita menggunakan bentuk heksagonal? Alasannya adalah heksagon adalah bentuk yang paling efisien untuk menutupi area 2D tanpa celah dan tanpa tumpang tindih.

**Kapan Menggunakan Hexbin Plot?**

Hexbin plot paling berguna ketika:

- Kamu memiliki banyak titik data.
- Distribusi data sulit dikenali dengan scatter plot karena banyaknya tumpang tindih.
- Kamu ingin memahami distribusi keseluruhan data, bukan titik data individu.

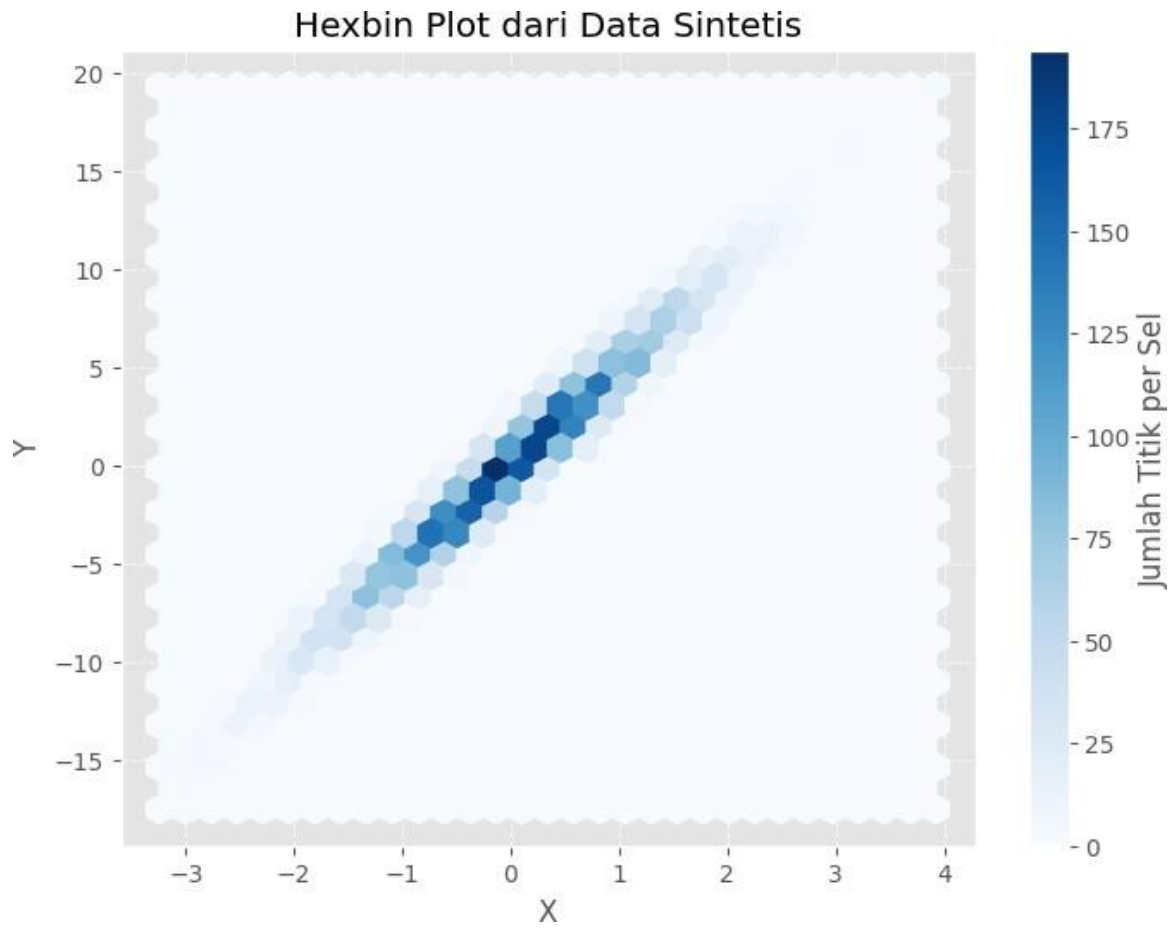
### 2.10.1 Basic Hexbin Plot

Untuk memulai, kita akan membuat data sintetis menggunakan pandas. Data ini akan mewakili koordinat X dan Y dari beberapa titik. Kita akan menggunakan matplotlib untuk membuat hexbin plot dari data ini.

```
# Membuat data sintetis untuk Hexbin Plot
np.random.seed(42)
x = np.random.randn(5000)
```

```
y = np.random.randn(5000) + 5 * x

df_hexbin = pd.DataFrame({
    'X': x,
    'Y': y
})

# Menggambar Hexbin Plot
plt.figure(figsize=(8, 6))
plt.hexbin(df_hexbin['X'], df_hexbin['Y'], gridsize=30, cmap='Blues')
plt.colorbar(label='Jumlah Titik per Sel')
plt.title('Hexbin Plot dari Data Sintetis')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:



Dalam visualisasi di atas, setiap sel heksagonal mewakili area di mana titik-titik data dikelompokkan. Warna yang lebih gelap menunjukkan bahwa ada lebih banyak titik di sel heksagonal tersebut, sedangkan warna yang lebih terang menunjukkan sebaliknya.

Dari Hexbin Plot ini, kita dapat dengan jelas melihat hubungan positif antara variabel X dan Y yang kita ciptakan. Selain itu, kita juga dapat melihat daerah mana yang memiliki konsentrasi titik data yang lebih tinggi.

### 2.10.2. Kustomisasi Hexbin Plot

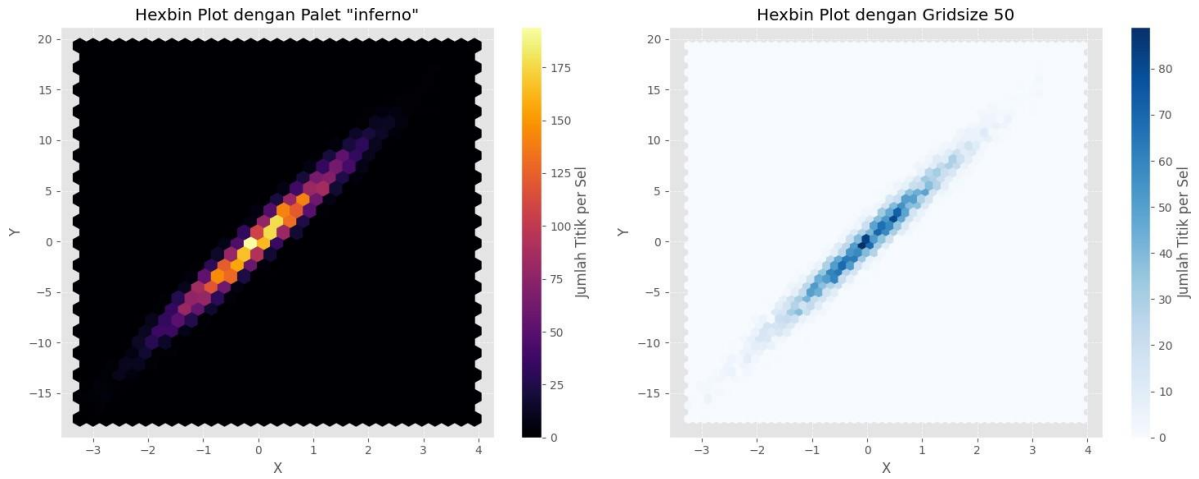
Seperti plot lainnya, ada berbagai cara untuk menyesuaikan Hexbin Plot agar sesuai dengan kebutuhan kita. Beberapa opsi kustomisasi yang mungkin kamu minati meliputi:

- Mengganti Palet Warna: Seperti yang kita bahas di subbab sebelumnya, palet warna adalah komponen penting dari visualisasi. Dengan Hexbin Plot, kamu dapat dengan mudah mengganti palet dengan parameter cmap.
- Mengatur Ukuran Sel: Kamu dapat mengatur ukuran sel dengan parameter gridsize. Nilai yang lebih tinggi akan menghasilkan sel yang lebih kecil, memungkinkan visualisasi yang lebih rinci tetapi mungkin lebih sulit dibaca.

Mari kita coba kustomisasi ini!

```
# Menggambar Hexbin Plot dengan kustomisasi berbeda
plt.figure(figsize=(15, 6))

# Plot dengan palet warna 'inferno'
plt.subplot(1, 2, 1)
plt.hexbin(df_hexbin['X'], df_hexbin['Y'], gridsize=30, cmap='inferno')
plt.colorbar(label='Jumlah Titik per Sel')
plt.title('Hexbin Plot dengan Palet "inferno"')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)

# Plot dengan gridsize yang lebih besar (sel yang lebih kecil)
plt.subplot(1, 2, 2)
plt.hexbin(df_hexbin['X'], df_hexbin['Y'], gridsize=50, cmap='Blues')
plt.colorbar(label='Jumlah Titik per Sel')
plt.title('Hexbin Plot dengan Gridsize 50')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)

plt.tight_layout()
plt.show()
```

Output:




Pada dua visualisasi di atas, kita dapat melihat bagaimana perubahan sederhana dapat memberikan nuansa yang berbeda pada Hexbin Plot:

- Palet "inferno" memberikan nuansa yang lebih "panas" pada plot, yang mungkin cocok untuk menampilkan intensitas atau urgensi.
- Gridsize 50 menghasilkan sel yang lebih kecil, yang memungkinkan kita untuk melihat detail yang lebih rinci dari distribusi titik data.

### 2.10.3. Menggunakan Hexbin Plot dengan Data Real-World

Kemampuan Hexbin Plot untuk menangani tumpang tindih titik data membuatnya sangat berguna dalam situasi dunia nyata. Misalnya, dalam analisis geospasial, di mana kita mungkin memiliki banyak titik data yang merepresentasikan lokasi fisik (seperti lokasi toko atau rumah), Hexbin Plot dapat membantu kita memahami daerah mana yang memiliki konsentrasi titik data yang lebih tinggi.

Namun, sebelum melanjutkan, penting untuk diingat bahwa seperti semua visualisasi data, Hexbin Plot memberikan gambaran kasar dari data. Meskipun sangat berguna untuk mendapatkan gambaran umum, selalu penting untuk mengeksplorasi data lebih lanjut untuk memahami nuansa dan detail yang mungkin hilang dalam visualisasi semacam ini.

Mari kita jelajahi lebih lanjut mengenai Hexbin Plot dengan skenario yang lebih kompleks. Sebagai contoh, bayangkan kamu memiliki dataset yang merepresentasikan penjualan produk di berbagai lokasi di suatu kota. Dataset ini memiliki koordinat geografis (lintang dan bujur) serta jumlah penjualan di setiap lokasi. Dengan Hexbin Plot, kita dapat memvisualisasikan di mana konsentrasi penjualan tertinggi berada.

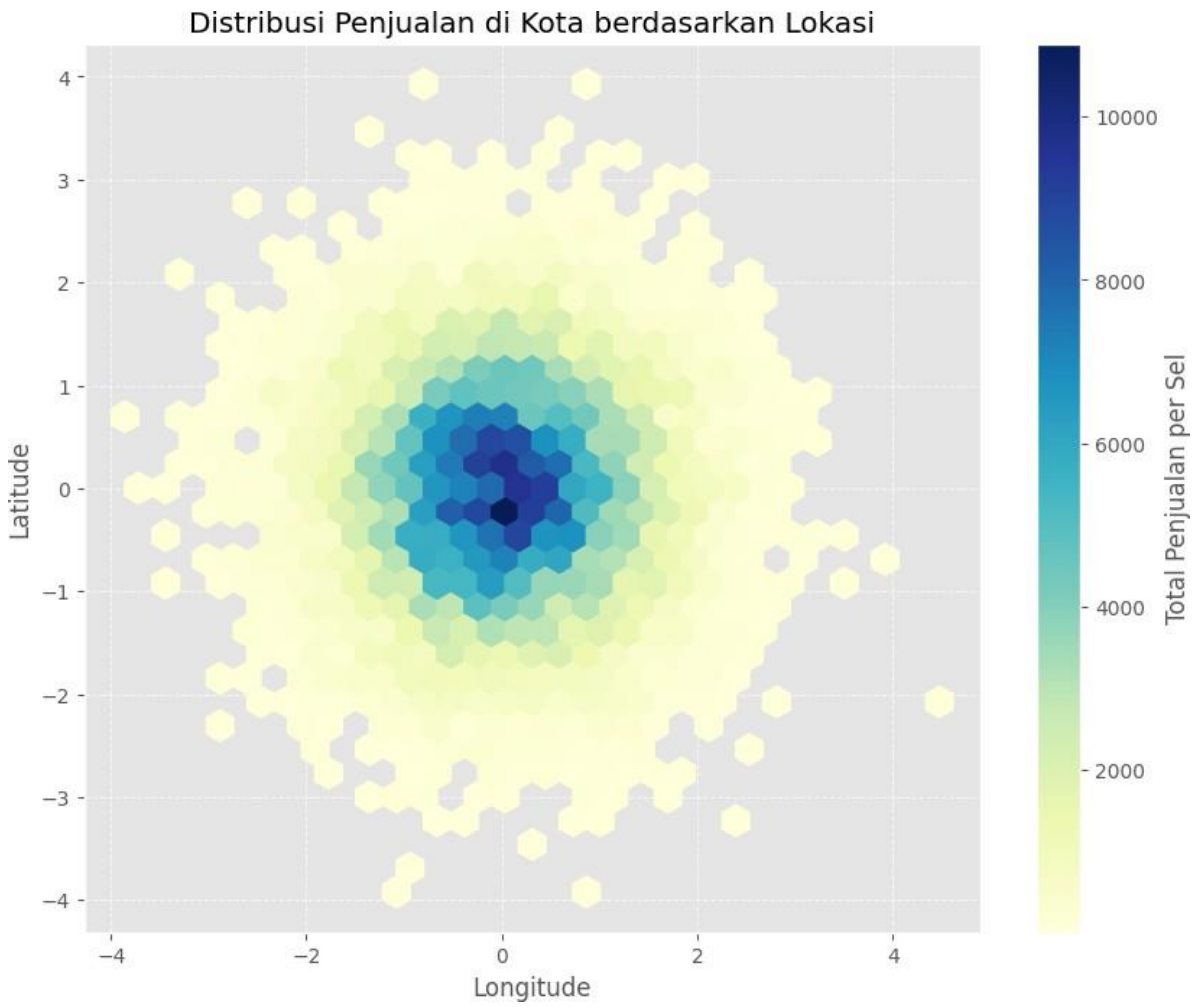
### 2.10.4 Membuat Hexbin Plot Berdasarkan Data Penjualan

Mari kita ciptakan data sintetis yang merepresentasikan penjualan produk di berbagai lokasi di suatu kota. Kami akan membuat data yang memiliki koordinat geografis (lintang dan bujur) serta jumlah penjualan di setiap lokasi.

Kota ini memiliki pusat perbelanjaan besar di tengah-tengah, sehingga kita akan memodelkan data sedemikian rupa sehingga ada konsentrasi penjualan yang lebih tinggi di dekat pusat kota dan penjualan yang lebih rendah di pinggiran kota.

```
# Membuat data sintetis berdasarkan skenario di atas
np.random.seed(42)
n_points = 10000

# Koordinat pusat kota
center_lat, center_lon = 0, 0

# Membuat koordinat lintang dan bujur dengan distribusi normal
lats = np.random.normal(center_lat, 1, n_points)
lons = np.random.normal(center_lon, 1, n_points)

# Membuat penjualan berdasarkan jarak dari pusat kota
distances_from_center = np.sqrt(lats**2 + lons**2)
sales = 100 - (distances_from_center * 20) + np.random.normal(0, 10,
n_points)
sales[sales < 0] = 0

df_sales = pd.DataFrame({
    'Latitude': lats,
    'Longitude': lons,
    'Sales': sales
})

# Menggambar Hexbin Plot berdasarkan jumlah penjualan
plt.figure(figsize=(10, 8))
hb = plt.hexbin(df_sales['Longitude'], df_sales['Latitude'],
C=df_sales['Sales'], gridsize=30, cmap='YlGnBu',
reduce_C_function=np.sum)
plt.colorbar(hb, label='Total Penjualan per Sel')
plt.title('Distribusi Penjualan di Kota berdasarkan Lokasi')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)
plt.show()
```

Output:




Dalam visualisasi di atas, kita dapat melihat bagaimana penjualan produk terdistribusi di seluruh kota. Seperti yang kita harapkan berdasarkan data sintetis yang kita ciptakan:

- Ada konsentrasi penjualan yang lebih tinggi (warna yang lebih gelap) di dekat pusat kota, yang ditandai dengan koordinat (0,0).
- Penjualan menurun saat kita bergerak menjauh dari pusat kota, dengan area pinggiran kota memiliki penjualan yang lebih rendah (warna yang lebih terang).

Dengan Hexbin Plot, kita dapat dengan cepat mendapatkan gambaran tentang daerah mana yang memiliki penjualan tertinggi dan mana yang mungkin memerlukan perhatian lebih untuk meningkatkan penjualan.

**Menginterpretasi Hexbin Plot**

Dalam bisnis, visualisasi seperti Hexbin Plot dapat memberikan wawasan berharga. Dalam contoh kita:

- Strategi Pemasaran: Daerah dengan penjualan tinggi mungkin sudah dikenal dengan produk kita. Namun, daerah dengan penjualan rendah bisa menjadi target untuk kampanye pemasaran yang lebih intensif.

- Distribusi Produk: Jika kita memiliki rantai pasokan atau jaringan distribusi, memahami daerah dengan permintaan tinggi dapat membantu mengoptimalkan distribusi produk.
- Ekspansi: Jika kita berencana untuk membuka toko atau outlet baru, memahami daerah dengan penjualan yang rendah namun potensial tinggi dapat menjadi kunci.

## 2.11. Quiver Plot

Di dunia visualisasi data, terkadang kita dihadapkan pada kebutuhan untuk menggambarkan vektor dalam ruang dua dimensi. Salah satu metode yang paling efektif untuk melakukan ini adalah dengan menggunakan Quiver Plot.

**Apa itu Quiver Plot?**
Quiver Plot, yang sering disebut juga sebagai field plot, adalah jenis visualisasi yang menggambarkan vektor sebagai panah dengan arah dan panjang tertentu. Plot semacam ini biasanya digunakan untuk memvisualisasikan aliran fluida, medan magnet, medan gravitasi, atau medan lain yang dapat didefinisikan oleh vektor.

**Mengapa Menggunakan Quiver Plot?**
Quiver Plot memungkinkan kita untuk:
- Memvisualisasikan Arah: Setiap panah menunjukkan arah vektor.
- Memahami Magnitudo: Panjang panah menunjukkan magnitudo atau kekuatan vektor.
- Dalam bidang ilmu seperti fisika dan teknik, memahami arah dan kekuatan dari vektor adalah kunci untuk memahami fenomena yang sedang dipelajari.

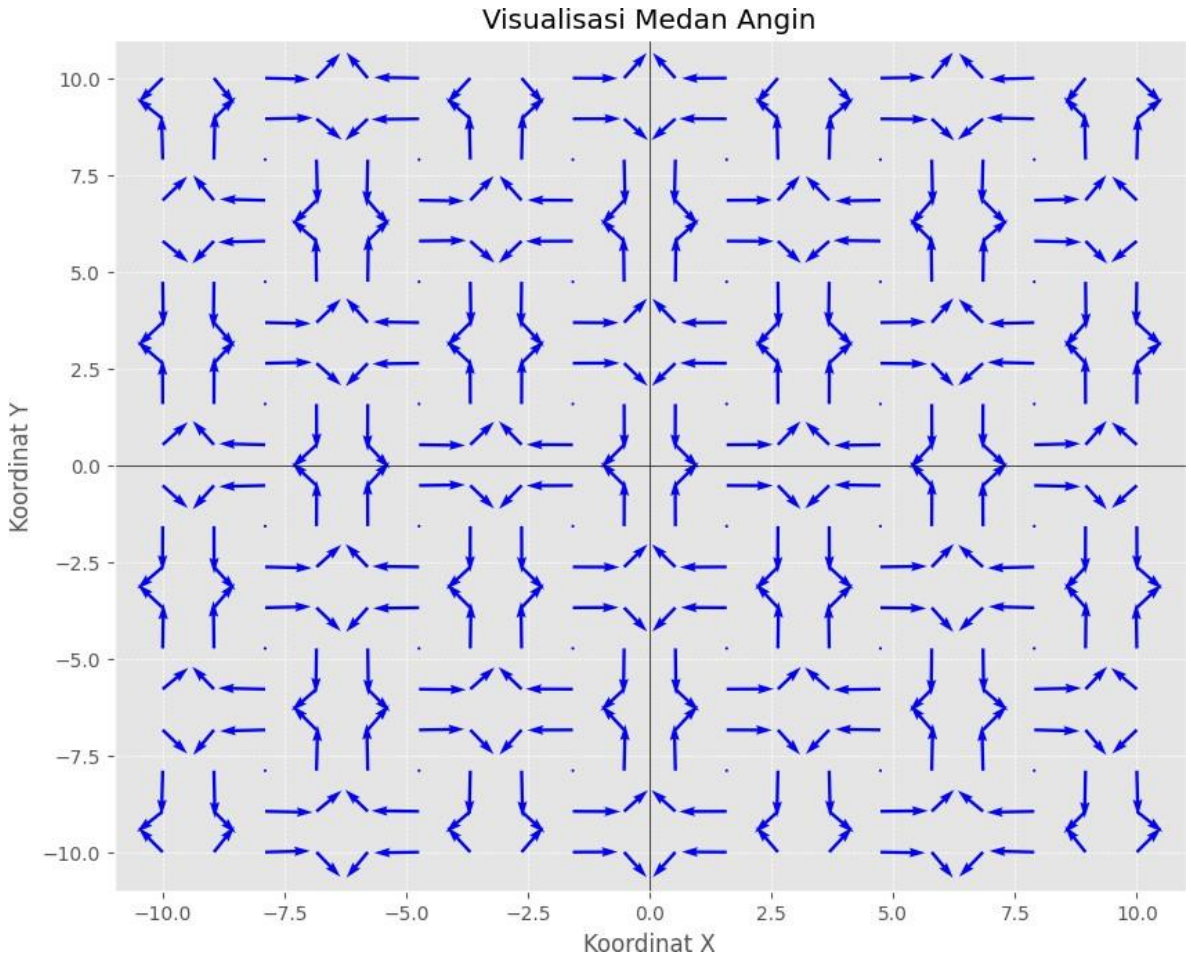
### 2.11.1. Basic Quiver Plot

Mari kita mulai dengan membuat Quiver Plot sederhana. Misalnya, kita ingin memvisualisasikan medan angin di suatu daerah. Untuk tujuan ini, kita akan membuat data sintetis yang merepresentasikan komponen X dan Y dari vektor angin di berbagai titik di daerah tersebut.

```
# Membuat data sintetis untuk Quiver Plot
x = np.linspace(-10, 10, 20)
y = np.linspace(-10, 10, 20)
X, Y = np.meshgrid(x, y)

# Membuat komponen U dan V dari vektor angin
U = np.sin(X)*np.cos(Y)
V = -np.cos(X)*np.sin(Y)

# Menggambar Quiver Plot
plt.figure(figsize=(10, 8))
```

```
plt.quiver(X, Y, U, V, scale=20, color='blue')
plt.title('Visualisasi Medan Angin')
plt.xlabel('Koordinat X')
plt.ylabel('Koordinat Y')
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```

Output:




Dalam visualisasi di atas, setiap panah mewakili vektor angin di titik tertentu. Arah panah menunjukkan arah angin, sedangkan panjang panah memberikan informasi tentang kecepatan angin. Dari plot ini, kamu dapat melihat bagaimana arah dan kecepatan angin berbeda di berbagai lokasi di daerah tersebut.

## 2.12. Stem Plot

Stem plot atau kadang disebut juga sebagai lollipop plot adalah jenis plot yang digunakan untuk memvisualisasikan data yang biasanya ditemui dalam bentuk diskrit atau berbentuk seri waktu. Stem plot ini dapat memberikan informasi yang jelas tentang distribusi data.

**Mengapa Stem Plot?**

Stem plot sering digunakan dalam analisis statistik dan penelitian ilmiah untuk:

- Menampilkan Distribusi Data: Stem plot dapat memberikan gambaran yang jelas tentang bagaimana data Anda didistribusikan, baik dalam hal frekuensi maupun kecenderungan sentral.
- Membandingkan Beberapa Set Data: Anda dapat membandingkan beberapa set data dalam satu stem plot, membuatnya lebih mudah untuk melihat perbedaan dan kesamaan antara set data tersebut.
- Menunjukkan Variasi dalam Data: Dengan stem plot, Anda dapat melihat seberapa besar variasi dalam data Anda, dan apakah ada outlier.

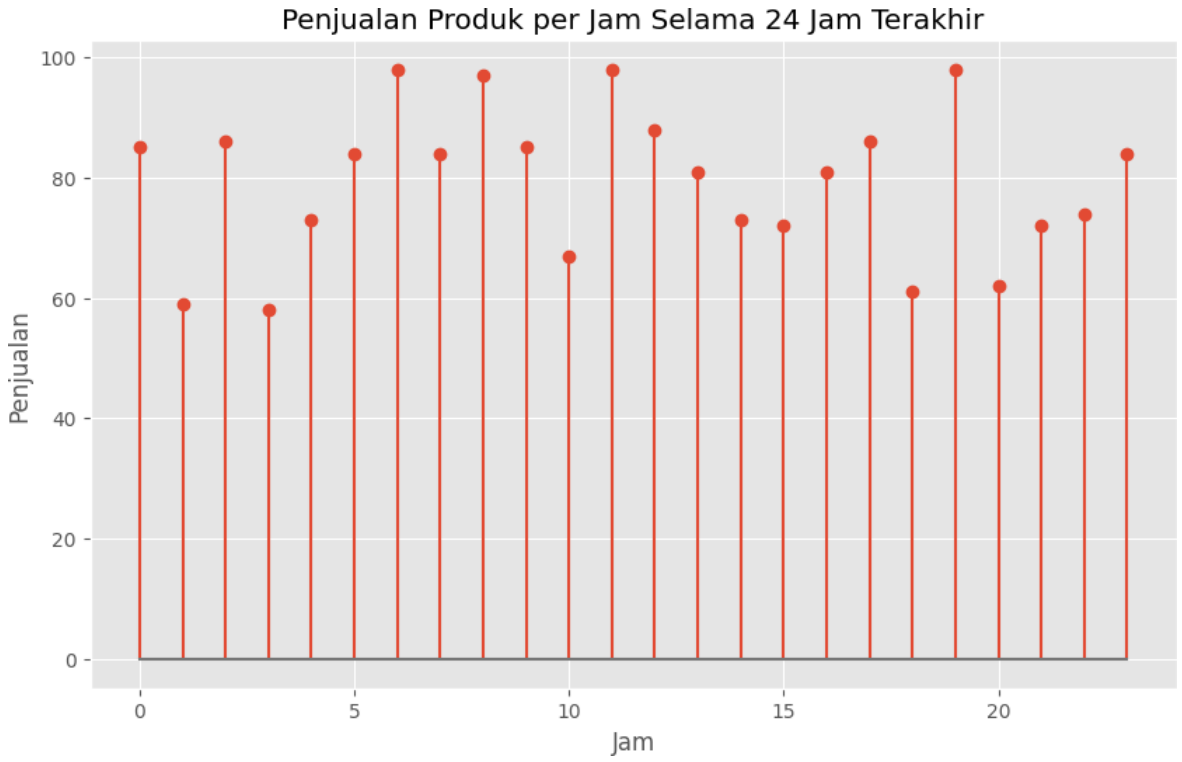
Meski memiliki banyak kegunaan, stem plot juga memiliki beberapa kekurangan. Salah satunya adalah bahwa mereka mungkin tidak sesuai untuk data dengan rentang nilai yang sangat luas, karena plot tersebut bisa menjadi sangat panjang dan sulit dibaca.

### 2.12.1. Basic Stem Plot

Untuk memulai, mari kita buat stem plot sederhana menggunakan matplotlib. Misalnya, kita memiliki data tentang penjualan produk per jam selama 24 jam terakhir di sebuah toko online. Kita ingin melihat bagaimana penjualan berubah sepanjang hari. Mari kita buat data sintetis ini dan visualisasikan menggunakan stem plot.

```
# Membuat data sintetis
hours = np.arange(24)
sales = np.random.randint(50, 100, size=24)

# Membuat stem plot
plt.figure(figsize=(10, 6))
markerline, stemlines, baseline = plt.stem(hours, sales,
use_line_collection=True)

# Menambahkan judul dan label
plt.title('Penjualan Produk per Jam Selama 24 Jam Terakhir')
plt.xlabel('Jam')
plt.ylabel('Penjualan')
plt.show()
```

Output:




Dalam stem plot di atas, sumbu X merepresentasikan jam dalam sehari, dan sumbu Y merepresentasikan jumlah penjualan. Setiap titik pada plot menunjukkan jumlah penjualan pada jam tertentu, dan garis yang menghubungkan titik ke sumbu X (batang "stem") membantu menunjukkan perbedaan antara titik data.

Stem plot ini memberikan gambaran jelas tentang bagaimana penjualan berubah selama sehari. Dengan melihat plot, kita dapat melihat bahwa ada fluktuasi dalam penjualan sepanjang hari.

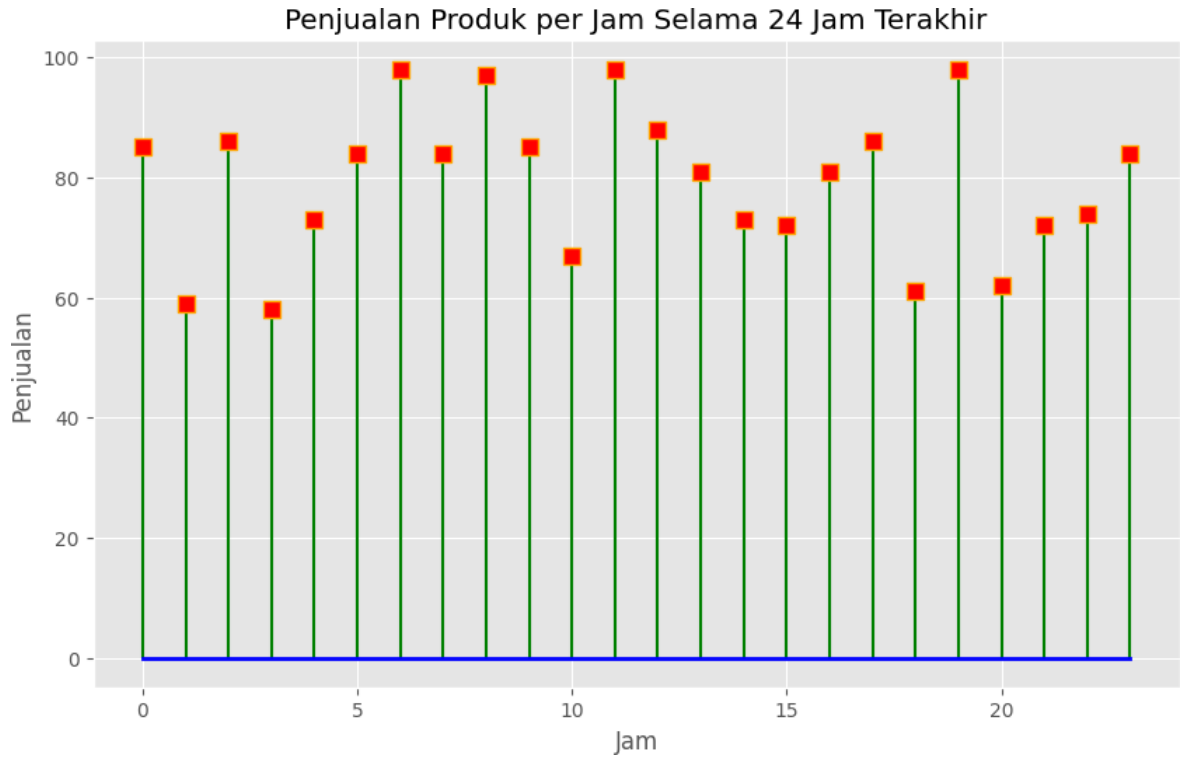
### 2.12.2. Mempersonalisasi Stem Plot

Salah satu keuntungan menggunakan matplotlib adalah fleksibilitasnya. Kita dapat memodifikasi hampir setiap aspek dari plot, termasuk stem plot. Sebagai contoh, kita dapat mengubah warna dan bentuk dari titik data, serta warna dari garis batang. Mari kita coba beberapa modifikasi ini pada stem plot kita.

```
# Membuat stem plot dengan kostumisasi
plt.figure(figsize=(10, 6))
markerline, stemlines, baseline = plt.stem(hours, sales,
use_line_collection=True)

# Mengubah warna dan bentuk marker
plt.setp(markerline, 'marker', 's', 'markersize', 8, 'markeredgecolor',
"orange", 'markerfacecolor', 'red')
```

```
# Mengubah warna dan ketebalan batang
plt.setp(stemlines, 'color', 'green', 'linewidth', 1.5)

# Mengubah warna dan ketebalan baseline
plt.setp(baseline, 'color', 'blue', 'linewidth', 2)

# Menambahkan judul dan label
plt.title('Penjualan Produk per Jam Selama 24 Jam Terakhir')
plt.xlabel('Jam')
plt.ylabel('Penjualan')

plt.show()
```

Output:




Dalam stem plot yang telah dimodifikasi ini, kita mengubah warna dan bentuk dari titik data, serta warna dari garis batang. Bentuk titik data telah diubah menjadi persegi ('s' dalam kode), dan warnanya telah diubah menjadi merah. Batangnya sekarang berwarna hijau, dan baseline yang mewakili sumbu X sekarang berwarna biru.

Perubahan ini tidak mengubah informasi yang disampaikan oleh plot, tetapi dapat membuat plot lebih menarik secara visual atau memudahkan pembacaan. Sebagai contoh, warna yang berbeda untuk titik dan batang bisa membuatnya lebih mudah untuk membedakan antara titik data dan batang pada plot yang sangat padat.

## 2.13. Error Bars

Mengukur dan menganalisis data adalah bagian penting dari banyak disiplin ilmu. Namun, ketika kita mengambil pengukuran, selalu ada beberapa ketidakpastian yang terkait dengan setiap pengukuran. Oleh karena itu, sangat penting untuk memahami dan menyajikan ketidakpastian ini dengan benar saat kita berbicara tentang data kita. Salah satu cara untuk melakukannya adalah dengan menggunakan error bars.

**Apa Itu Error Bars?**

Error bars adalah garis grafis yang mewakili variasi data dan digunakan pada plot data untuk menunjukkan ketidakpastian dalam setiap titik. Secara khusus, error bars memberi kita ide tentang seberapa jauh dari titik data yang diukur ini kita harapkan nilai sebenarnya berada.

**Mengapa Menggunakan Error Bars?**

Menggunakan error bars dalam visualisasi data kita memiliki beberapa keuntungan:

- Menyajikan Ketidakpastian: Error bars memberikan gambaran yang jelas tentang sejauh mana data kita mungkin bervariasi.
- Meningkatkan Kepercayaan: Menyajikan ketidakpastian data kita kepada pembaca atau pemirsa dapat meningkatkan kepercayaan mereka pada analisis kita.
- Membantu dalam Interpretasi: Error bars dapat membantu dalam interpretasi hasil, seperti ketika membandingkan dua set data.

### 2.13.1. Membuat Error Bars dengan Matplotlib

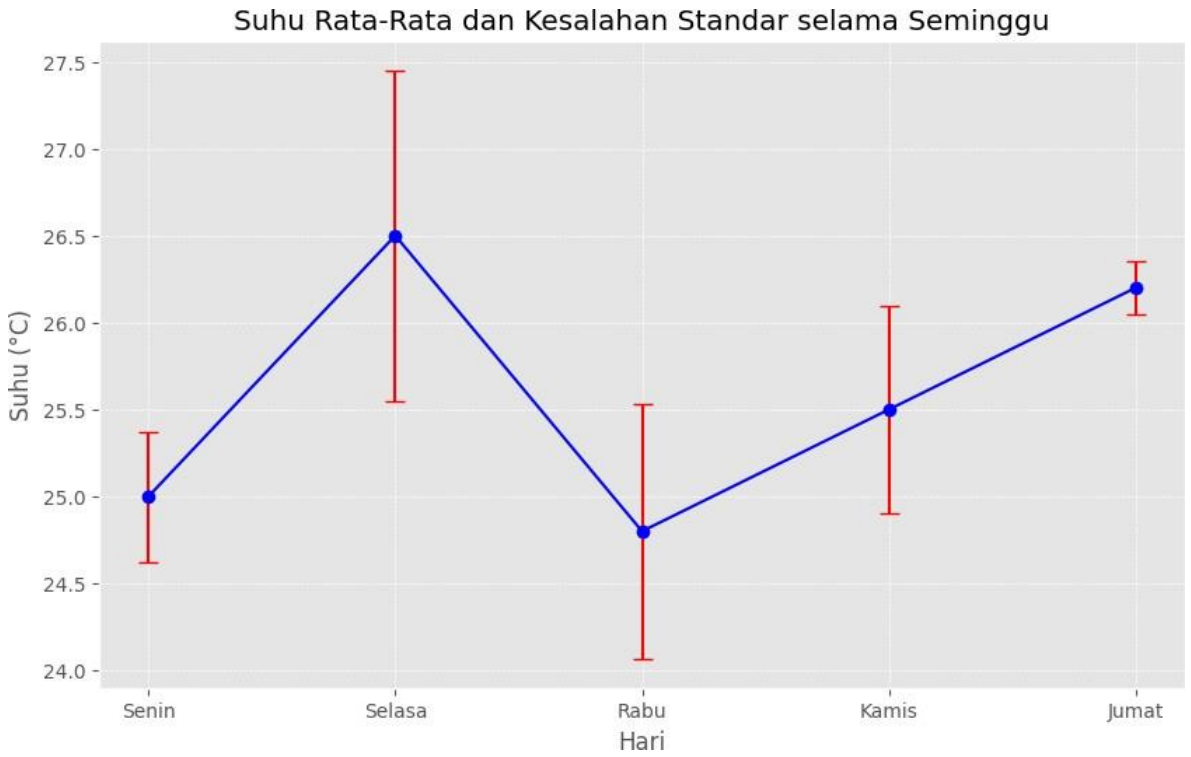
Mari kita lihat bagaimana membuat error bars menggunakan matplotlib. Kita akan mulai dengan kasus sederhana dimana kita memiliki beberapa pengukuran dan kita ingin menunjukkan ketidakpastian pada setiap pengukuran tersebut.

Misalnya, bayangkan kamu bekerja di laboratorium dan telah mengambil serangkaian pengukuran suhu. Setiap pengukuran memiliki kesalahan standar yang terkait dengannya. Mari kita visualisasikan data ini dengan error bars.

```
import numpy as np
import matplotlib.pyplot as plt

# Membuat data sintetis
np.random.seed(42)
days = np.array(['Senin', 'Selasa', 'Rabu', 'Kamis', 'Jumat'])
temperatures = np.array([25, 26.5, 24.8, 25.5, 26.2])
errors = np.random.rand(5)  # Kesalahan standar untuk setiap pengukuran

# Membuat plot dengan error bars
plt.figure(figsize=(10, 6))
plt.errorbar(days, temperatures, yerr=errors, fmt='o-', capsize=5,
```

```
color='blue', ecolor='red')

# Menambahkan judul dan label
plt.title('Suhu Rata-Rata dan Kesalahan Standar selama Seminggu')
plt.xlabel('Hari')
plt.ylabel('Suhu (°C)')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

Output:




Kini, kita berhasil memvisualisasikan data suhu dengan error bars yang merepresentasikan ketidakpastian untuk setiap pengukuran. Dalam plot di atas:

- Titik Biru: Menunjukkan rata-rata suhu yang diukur setiap hari.
- Garis Biru: Menghubungkan titik-titik untuk memberikan gambaran umum tentang tren suhu selama seminggu.
- Error Bars Merah: Menunjukkan kesalahan standar dari setiap pengukuran, memberikan gambaran tentang seberapa yakin kita dengan pengukuran tersebut.

Capsize pada ujung error bars (uji kecil di ujung bar merah) menambahkan sentuhan visual yang menjelaskan batas kesalahan. Ini memberikan gambaran yang lebih baik tentang seberapa besar variasi yang mungkin ada dari pengukuran rata-rata.

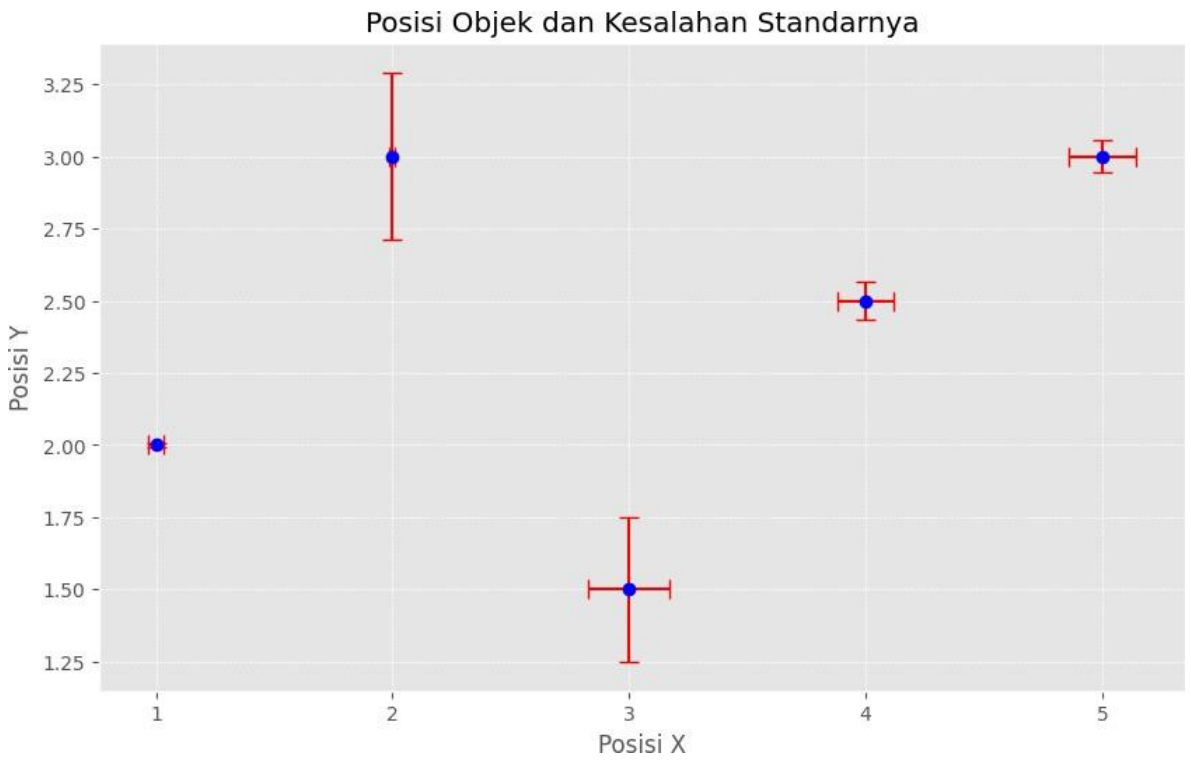
### 2.13.2. Error Bars untuk Data Dua Dimensi

Dalam contoh sebelumnya, kita melihat bagaimana menambahkan error bars ke sumbu Y. Namun, dalam beberapa kasus, kita mungkin ingin menambahkan error bars ke kedua sumbu (X dan Y). Misalnya, jika kita melakukan percobaan di mana kedua variabel memiliki ketidakpastian, maka akan berguna untuk menunjukkan error bars di kedua sumbu.

Mari kita lihat contoh di mana kita memiliki data titik (X, Y) dan kita memiliki ketidakpastian pada kedua pengukuran tersebut. Misalkan kita memiliki data tentang posisi suatu objek dalam dua dimensi, dan kita ingin menampilkan ketidakpastian posisi tersebut.

```
# Membuat data sintetis
positions = np.array([[1, 2], [2, 3], [3, 1.5], [4, 2.5], [5, 3]])
x_errors = np.random.rand(5) * 0.2  # Kesalahan standar untuk pengukuran
X
y_errors = np.random.rand(5) * 0.3  # Kesalahan standar untuk pengukuran
Y

# Membuat plot dengan error bars di kedua sumbu
plt.figure(figsize=(10, 6))
plt.errorbar(positions[:, 0], positions[:, 1], xerr=x_errors,
yerr=y_errors, fmt='o', color='blue', ecolor='red', capsize=5)

# Menambahkan judul dan label
plt.title('Posisi Objek dan Kesalahan Standarnya')
plt.xlabel('Posisi X')
plt.ylabel('Posisi Y')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

Output:




Dalam visualisasi di atas, kita dapat melihat posisi objek di ruang dua dimensi (X, Y) bersama dengan ketidakpastian dari masing-masing pengukuran:\

- Titik Biru: Menunjukkan posisi objek yang diukur.
- Error Bars Merah (Horisontal): Menunjukkan kesalahan standar dari pengukuran posisi X.
- Error Bars Merah (Vertikal): Menunjukkan kesalahan standar dari pengukuran posisi Y.

Dengan menambahkan error bars ke kedua sumbu, kita dapat dengan jelas menyampaikan ketidakpastian dari kedua pengukuran kepada pembaca atau pemirsa.

## 2.14. Filled Error Bands

Dalam analisis data dan ilmuwan yang berkaitan dengan visualisasi data, seringkali kita memerlukan cara yang lebih visual untuk menunjukkan ketidakpastian atau variasi dalam data kita. Salah satu cara yang efektif untuk melakukan ini adalah dengan menggunakan 'filled error bands'. Ini adalah area yang diisi di sekitar kurva dalam plot untuk menunjukkan variasi atau ketidakpastian dari data yang direpresentasikan oleh kurva tersebut.

**Apa Itu Filled Error Bands?**
Filled error bands adalah area berwarna yang mengelilingi garis dalam plot, yang menunjukkan rentang ketidakpastian atau variasi dari data yang direpresentasikan oleh

garis tersebut. Ini memberikan gambaran yang jelas tentang seberapa yakin kita dengan estimasi atau pengukuran kita, dan seberapa besar variasi yang mungkin kita harapkan.

**Mengapa Menggunakan Filled Error Bands?**
Menggunakan filled error bands dalam visualisasi data kita memiliki beberapa keuntungan:

- Visualisasi Ketidakpastian: Filled error bands memberikan cara yang sangat visual untuk menunjukkan ketidakpastian dalam data kita.
- Meningkatkan Kepercayaan: Menyajikan ketidakpastian data kita kepada pembaca atau pemirsa dapat meningkatkan kepercayaan mereka pada analisis kita.
- Membantu dalam Interpretasi: Filled error bands dapat membantu dalam interpretasi hasil, seperti ketika membandingkan dua set data.

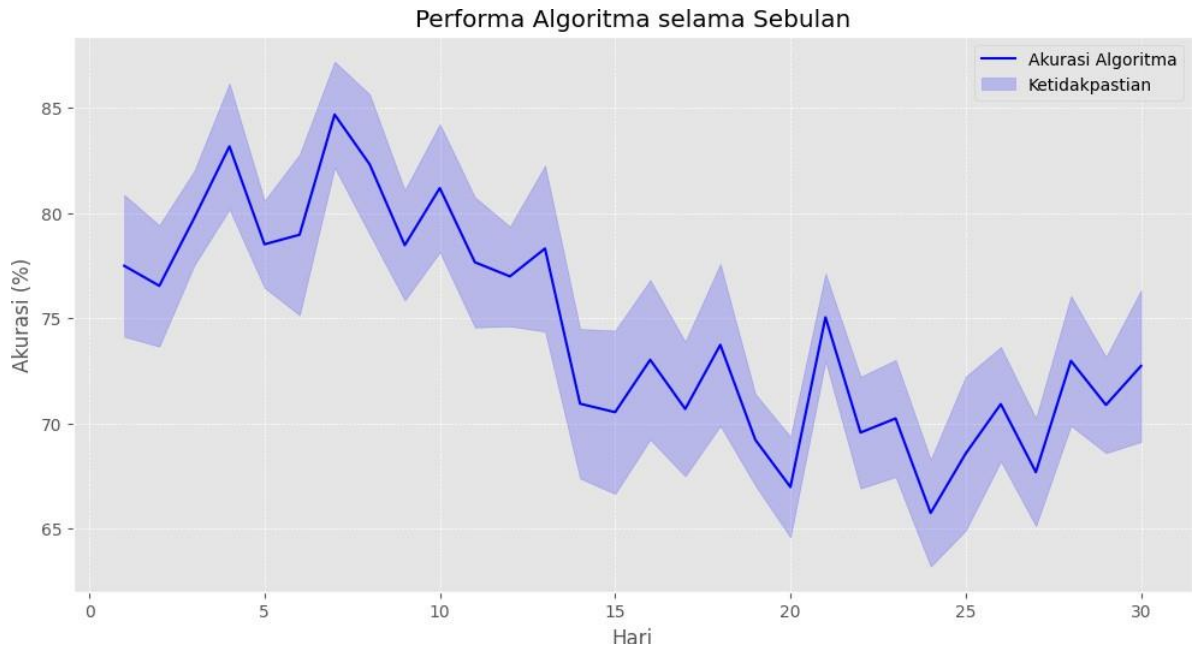
### 2.14.1. Membuat Filled Error Bands dengan Matplotlib

Mari kita lihat bagaimana membuat filled error bands menggunakan matplotlib. Sebagai contoh pertama, bayangkan kamu sedang menganalisis performa suatu algoritma selama waktu dan ingin menampilkan rata-rata performa serta ketidakpastiannya.

Misalkan kita memiliki data berikut: waktu (dalam hari) vs. akurasi algoritma (dalam persen), dengan rentang ketidakpastian untuk setiap pengukuran akurasi.

```
# Membuat data sintetis
np.random.seed(42)
days = np.linspace(1, 30, 30)
accuracy = 75 + np.sin(days/5)*5 + np.random.randn(30) * 3
error = (np.random.rand(30) + 1) * 2

# Membuat plot dengan filled error bands
plt.figure(figsize=(12, 6))
plt.plot(days, accuracy, label='Akurasi Algoritma', color='blue')
plt.fill_between(days, accuracy - error, accuracy + error, color='blue',
alpha=0.2, label='Ketidakpastian')

# Menambahkan judul, label, dan legenda
plt.title('Performa Algoritma selama Sebulan')
plt.xlabel('Hari')
plt.ylabel('Akurasi (%)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

Output:




Di plot di atas:

- Garis Biru: Mewakili rata-rata akurasi algoritma dari waktu ke waktu.
- Area Biru Muda: Ini adalah 'filled error band' yang mewakili ketidakpastian dalam pengukuran akurasi. Lebar band ini menunjukkan seberapa besar variasi yang kita harapkan dari rata-rata akurasi.

Dalam contoh ini, kita dapat melihat bahwa akurasi algoritma berfluktuasi dari waktu ke waktu, tetapi ada juga beberapa ketidakpastian yang terkait dengan setiap pengukuran akurasi. Filled error bands memberikan cara yang mudah dimengerti untuk memvisualisasikan ketidakpastian ini.

Dengan menampilkan ketidakpastian ini, kita memberikan pembaca atau pemirsa gambaran yang lebih jelas tentang seberapa yakin kita dengan estimasi atau pengukuran kita. Ini bisa sangat berguna, misalnya, saat membandingkan performa dari beberapa algoritma berbeda.

### 2.14.2. Kostumisasi Filled Error Bands

Salah satu kelebihan menggunakan matplotlib adalah fleksibilitas yang ditawarkannya dalam hal kostumisasi. Kita dapat mengubah warna, gaya, dan transparansi dari filled error bands untuk menyesuaikannya dengan kebutuhan visualisasi kita.

Mari kita coba beberapa variasi dari filled error bands untuk memahami bagaimana kita dapat menyesuaikan tampilannya. Sebagai contoh, kita akan mempertimbangkan data tentang pertumbuhan tanaman selama waktu dengan ketidakpastian yang berkaitan dengan pengukuran pertumbuhan.

```
# Membuat data sintetis untuk pertumbuhan tanaman
```

```
growth = np.linspace(5, 30, 30) + np.sin(days/5)*3 + np.random.randn(30)
* 2
growth_error = (np.random.rand(30) + 0.5) * 2

# Membuat plot dengan filled error bands yang dikostumisasi
plt.figure(figsize=(12, 6))
plt.plot(days, growth, label='Pertumbuhan Tanaman', color='green')
plt.fill_between(days, growth - growth_error, growth + growth_error,
color='green', alpha=0.1, label='Ketidakpastian', hatch='//')

# Menambahkan judul, label, dan legenda
plt.title('Pertumbuhan Tanaman selama Sebulan')
plt.xlabel('Hari')
plt.ylabel('Pertumbuhan (cm)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

Output:


Dalam visualisasi di atas:

- Garis Hijau: Menunjukkan pertumbuhan rata-rata tanaman dari waktu ke waktu.
- Area Hijau Berpetak: Ini adalah 'filled error band' yang mewakili ketidakpastian dalam pengukuran pertumbuhan. Pola garis miring ('hatch') menambahkan sentuhan visual lainnya untuk menunjukkan ketidakpastian.

Kita telah menggunakan pola garis miring di area error band untuk memberikan sentuhan visual tambahan. Ini dapat berguna jika kita ingin membedakan antara beberapa error bands atau jika kita ingin memberikan tampilan yang berbeda untuk visualisasi kita.

## 2.15. Polar Plot

Pernahkah kamu melihat grafik dengan sumbu yang terletak dalam bentuk lingkaran? Mungkin saat memeriksa arah angin, fase bulan, atau saat memahami prinsip dasar radar. Grafik tersebut dikenal sebagai polar plot atau plot polar. Dalam plot polar, data direpresentasikan dalam koordinat polar, bukan koordinat kartesius yang biasanya kita lihat.

**Mengapa Menggunakan Polar Plot?**

Polar plot digunakan untuk menyajikan data yang memiliki sifat periodik atau berputar. Beberapa contoh penggunaannya antara lain:

- Arah Angin: Mewakili kecepatan dan arah angin.
- Radar: Menunjukkan jarak dan arah objek dari pusat radar.
- Fase Bulan: Menunjukkan posisi bulan selama siklusnya.

### 2.15.1. Basic Polar Plot

Sebelum kita melangkah lebih jauh, mari kita pahami elemen dasar dari plot polar:

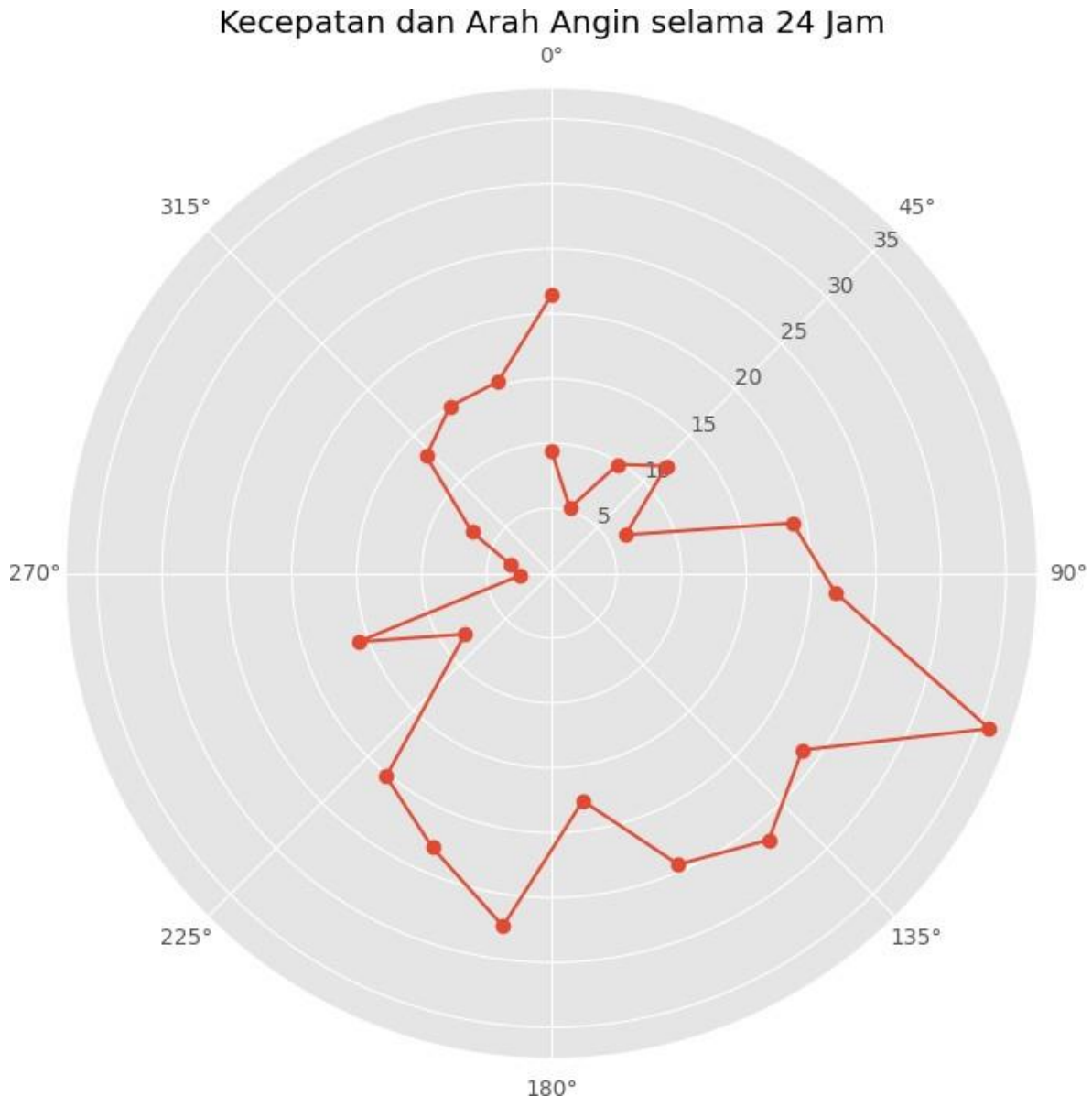
- Theta (θ): Ini adalah sudut dari sumbu horizontal positif (biasanya sumbu x kartesius) ke titik data.
- Radius (r): Ini adalah jarak dari titik data ke titik asal (pusat lingkaran).

Mari kita buat polar plot sederhana untuk memahami konsep ini. Sebagai contoh pertama, kita akan memvisualisasikan arah dan kecepatan angin selama 24 jam. Misalkan kita memiliki data kecepatan angin dan arah angin (dalam derajat) untuk setiap jam dari suatu hari.

```
# Membuat data sintetis untuk arah dan kecepatan angin
hours = np.linspace(0, 24, 24)
wind_speeds = np.abs(np.sin(hours/6)*25 + np.random.randn(24) * 5)
wind_directions = np.deg2rad(hours * 15) # Mengonversi derajat ke
radian

# Membuat polar plot
plt.figure(figsize=(8, 8))
ax = plt.subplot(1, 1, 1, projection='polar')
ax.plot(wind_directions, wind_speeds, marker='o', linestyle='-')

# Menambahkan judul dan label
ax.set_title('Kecepatan dan Arah Angin selama 24 Jam')
ax.set_theta_zero_location('N')  # Menetapkan 0 derajat ke arah Utara
```

```
ax.set_theta_direction(-1)  # Mengatur arah putaran menjadi searah jarum jam
ax.set_rlabel_position(45)  # Menetapkan posisi label radius

plt.show()
```

Output:




Penjelasan plot polar di atas:

- Sumbu Lingkaran: Menunjukkan arah angin dalam derajat. 0° menunjuk ke arah utara dan berputar searah jarum jam.
- Jarak dari Pusat: Menggambarkan kecepatan angin. Semakin jauh dari pusat, semakin besar kecepatan angin.
- Titik dan Garis: Menunjukkan kecepatan dan arah angin setiap jam selama 24 jam.

Dalam contoh ini, kita dapat melihat bagaimana arah dan kecepatan angin berubah selama sehari. Polar plot memberikan cara visual yang intuitif untuk memahami data seperti ini.

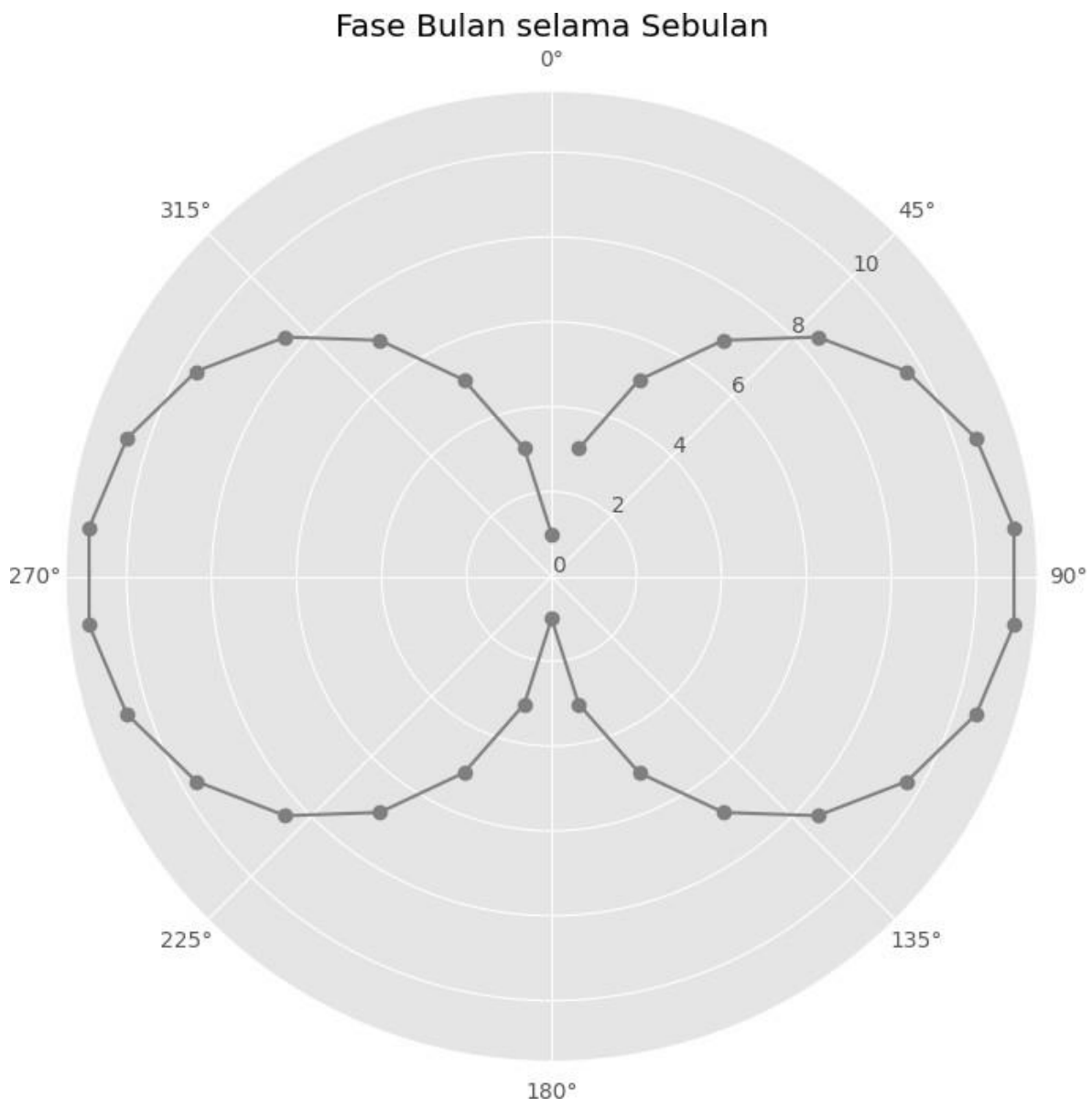
### 2.15.2. Kostumisasi Polar Plot

Salah satu kelebihan matplotlib adalah fleksibilitas yang ditawarkannya. Kamu bisa mengubah warna, gaya, dan aspek lain dari plot polar untuk menyesuaikannya dengan kebutuhan visualisasi.

Misalnya, bayangkan kamu ingin menampilkan data fase bulan dalam bentuk polar plot. Untuk tujuan ini, kita akan memvisualisasikan fase bulan selama 30 hari. Fase bulan akan diwakili oleh jarak dari pusat lingkaran, dengan setiap hari mewakili sudut tertentu. Mari kita buat visualisasi ini!

```
# Membuat data sintetis untuk fase bulan
moon_phase = np.abs(np.sin(days/30 * 2 * np.pi)) * 10 + 1
moon_directions = np.deg2rad(days * 12)  # Mengonversi derajat ke radian

# Membuat polar plot untuk fase bulan
plt.figure(figsize=(8, 8))
ax = plt.subplot(1, 1, 1, projection='polar')
ax.plot(moon_directions, moon_phase, marker='o', linestyle='-',
color='gray')

# Menambahkan judul dan label
ax.set_title('Fase Bulan selama Sebulan')
ax.set_theta_zero_location('N')  # Menetapkan 0 derajat ke arah Utara
ax.set_theta_direction(-1)  # Mengatur arah putaran menjadi searah jarum
jam
ax.set_rlabel_position(45)  # Menetapkan posisi label radius
ax.set_yticks(np.arange(0, 12, 2))

plt.show()
```

Output:




Dalam visualisasi di atas:

- Sumbu Lingkaran: Mewakili hari dalam sebulan. Setiap hari mewakili sudut tertentu dalam lingkaran.
- Jarak dari Pusat: Menggambarkan fase bulan. Saat jarak dari pusat mencapai puncak, itu menunjukkan bulan purnama, dan saat berada di dekat pusat, itu menunjukkan bulan baru.
- Titik dan Garis Abu-abu: Menunjukkan fase bulan setiap hari selama sebulan.

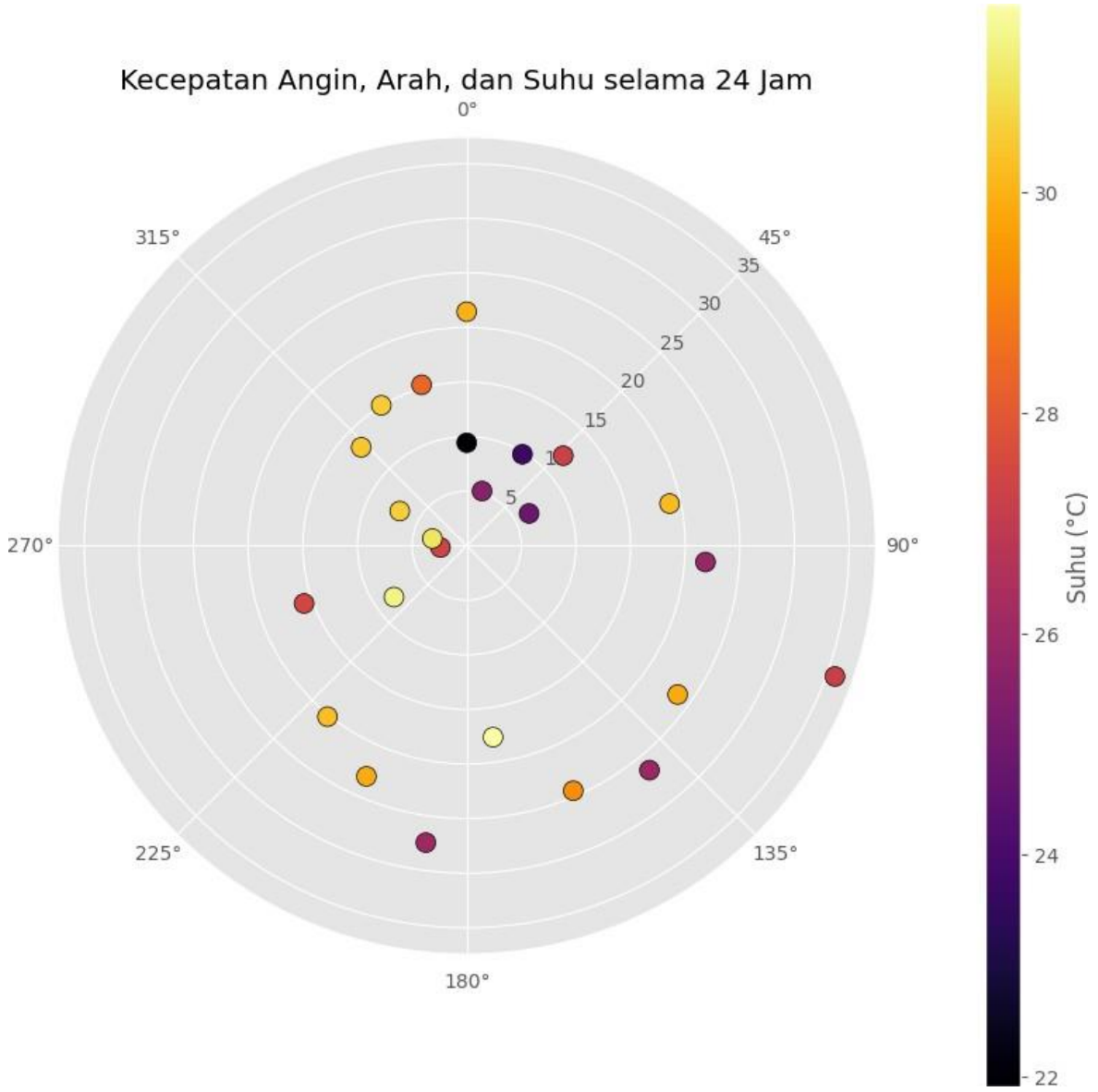
Visualisasi ini memberikan gambaran yang menarik tentang bagaimana fase bulan berubah selama sebulan. Dengan polar plot, kita dapat dengan mudah melihat siklus bulan dari bulan baru ke bulan purnama dan kembali lagi ke bulan baru.

### 2.15.3. Polar Plot dengan Variasi Warna

Kita juga bisa menambahkan variasi warna ke plot polar untuk menonjolkan beberapa aspek data. Sebagai contoh, kita akan memvisualisasikan suhu udara selama sehari dengan arah angin. Di sini, arah angin akan diwakili oleh sudut, jarak dari pusat akan menunjukkan kecepatan angin, dan warna akan menunjukkan suhu. Mari kita buat visualisasi ini!

```
# Membuat data sintetis untuk suhu, kecepatan, dan arah angin
temperatures = 25 + np.sin(hours/12)*5 + np.random.randn(24) * 2

# Membuat polar plot dengan variasi warna
plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, projection='polar')
scatter = ax.scatter(wind_directions, wind_speeds, c=temperatures,
cmap='inferno', s=100, edgecolors='black')

# Menambahkan judul, label, dan colorbar
ax.set_title('Kecepatan Angin, Arah, dan Suhu selama 24 Jam')
ax.set_theta_zero_location('N')  # Menetapkan 0 derajat ke arah Utara
ax.set_theta_direction(-1)  # Mengatur arah putaran menjadi searah jarum
jam
ax.set_rlabel_position(45)  # Menetapkan posisi label radius
cbar = plt.colorbar(scatter, orientation='vertical', pad=0.1, aspect=30)
cbar.set_label('Suhu (°C)')

plt.show()
```

Output:




Dalam visualisasi di atas:

- Sumbu Lingkaran: Mewakili arah angin selama 24 jam.
- Jarak dari Pusat: Menggambarkan kecepatan angin. Semakin jauh dari pusat, semakin besar kecepatan angin.
- Warna Titik: Menunjukkan suhu udara pada jam tertentu. Skala warna panas (seperti kuning dan oranye) menunjukkan suhu yang lebih tinggi, sedangkan warna yang lebih dingin (seperti ungu) menunjukkan suhu yang lebih rendah.

Dengan memasukkan informasi tambahan ke dalam visualisasi kita, kita dapat memberikan gambaran yang lebih lengkap tentang situasi yang sedang kita analisis. Di sini, kita dapat melihat bagaimana kecepatan angin, arah angin, dan suhu saling berhubungan selama 24 jam.

## 2.16. Step Plot

Pernahkah kamu melihat grafik di mana garis data tampak seperti tangga atau serangkaian langkah daripada garis halus yang biasa kita lihat? Itulah yang disebut sebagai 'Step Plot' atau 'Staircase Plot'. Plot jenis ini sangat berguna dalam situasi di mana kita ingin menunjukkan perubahan nilai data pada titik-titik tertentu dan bagaimana data tersebut tetap konstan di antara titik-titik tersebut.

**Mengapa Menggunakan Step Plot?**

Step plot biasanya digunakan dalam situasi di mana perubahan data terjadi pada interval tertentu dan tidak ada perubahan di antara interval tersebut. Beberapa contoh penggunaannya meliputi:

- Keuangan: Saat menampilkan data harga saham di mana harga berubah hanya pada saat transaksi terjadi dan tetap konstan di antaranya.
- Kontrol Proses: Untuk memonitor perubahan status mesin atau peralatan.
- Event Logging: Menyajikan log waktu kejadian tertentu.

### 2.16.1. Basic Step Plot

Mari kita mulai dengan membuat step plot sederhana untuk memahami konsepnya. Sebagai contoh pertama, kita akan memvisualisasikan harga saham perusahaan fiktif selama 10 hari.

```
import numpy as np
import matplotlib.pyplot as plt

# Membuat data sintetis untuk harga saham
days = np.arange(1, 11)
stock_prices = [50, 51, 52, 51, 50, 49, 48, 49, 50, 51]

# Membuat step plot
plt.figure(figsize=(10, 6))
plt.step(days, stock_prices, where='mid', color='blue', linestyle='-',
linewidth=2, label='Harga Saham')
plt.fill_between(days, stock_prices, step='mid', alpha=0.2,
color='blue')

# Menambahkan judul, label, dan legenda
plt.title('Harga Saham Perusahaan X Selama 10 Hari')
plt.xlabel('Hari')
plt.ylabel('Harga Saham')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

Output:


Dalam visualisasi di atas, kita dapat melihat bagaimana harga saham berubah setiap hari. Garis biru yang terlihat seperti tangga adalah inti dari step plot. Area biru muda di bawah garis memberikan efek visual tambahan yang menunjukkan area di bawah kurva.

Kunci dari step plot adalah parameter where yang menentukan di mana langkah harus dimulai. Dalam contoh ini, kita menggunakan where='mid', yang berarti langkah dimulai di tengah antara titik data sebelumnya dan titik data saat ini.

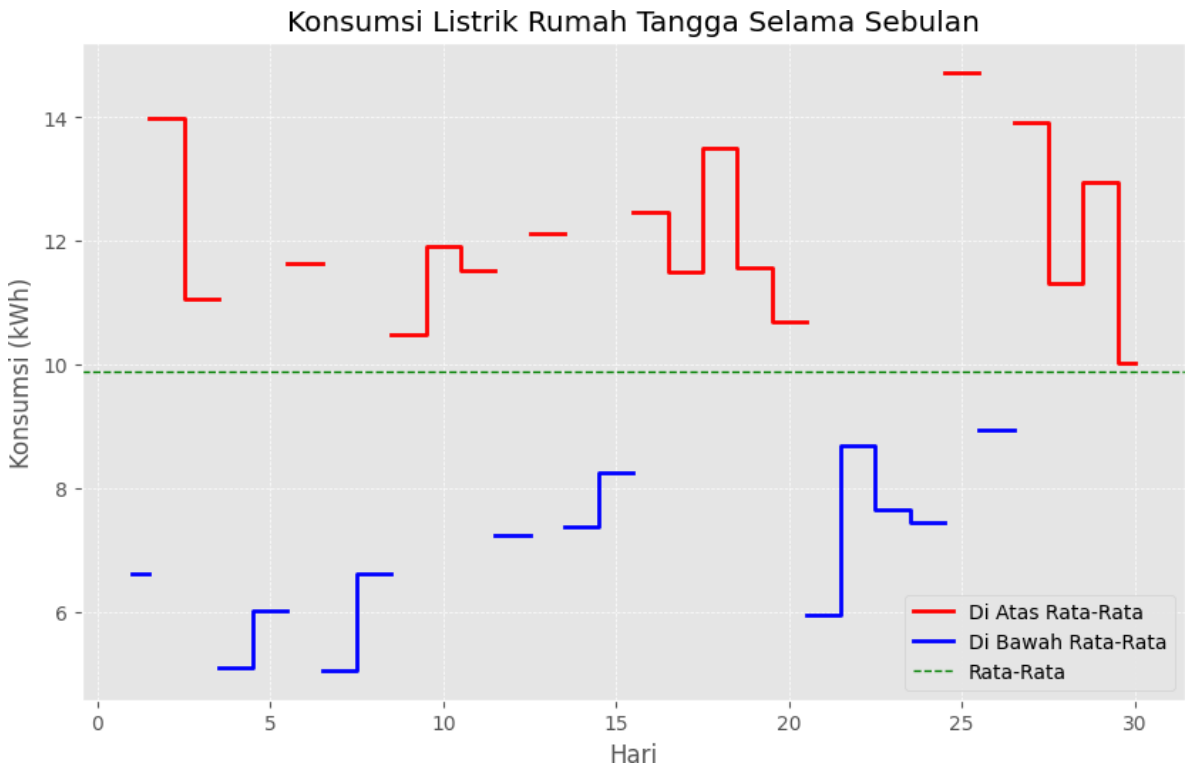
### 2.16.2. Step Plot dengan Variasi Warna

Mari kita coba variasi lain dari step plot. Misalkan kita memiliki data konsumsi listrik rumah tangga selama sebulan dan kita ingin menampilkan konsumsi ini dalam bentuk step plot dengan warna berbeda untuk konsumsi di atas dan di bawah rata-rata. Mari kita lihat bagaimana kita bisa melakukannya.

```
# Membuat data sintetis untuk konsumsi listrik
days_in_month = np.arange(1, 31)
power_consumption = np.random.uniform(5, 15, 30)

# Menghitung rata-rata konsumsi
avg_consumption = np.mean(power_consumption)

# Membuat step plot dengan variasi warna berdasarkan rata-rata
plt.figure(figsize=(10, 6))
```

```
above_avg = np.ma.masked_where(power_consumption <= avg_consumption,
power_consumption)
below_avg = np.ma.masked_where(power_consumption > avg_consumption,
power_consumption)

plt.step(days_in_month, above_avg, where='mid', color='red',
linestyle='-', linewidth=2, label='Di Atas Rata-Rata')
plt.step(days_in_month, below_avg, where='mid', color='blue',
linestyle='-', linewidth=2, label='Di Bawah Rata-Rata')
plt.axhline(avg_consumption, color='green', linestyle='--', linewidth=1,
label='Rata-Rata')

# Menambahkan judul, label, dan legenda
plt.title('Konsumsi Listrik Rumah Tangga Selama Sebulan')
plt.xlabel('Hari')
plt.ylabel('Konsumsi (kWh)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

Output:



## 2.17. Spectrogram

Selamat datang di dunia visualisasi yang menakjubkan dari Spectrogram! Pernahkah kamu berpikir bagaimana kita bisa "melihat" suara? Atau bagaimana kita dapat mengetahui komponen frekuensi dari sinyal waktu yang kompleks? Jawabannya terletak pada alat visualisasi yang disebut spectrogram.

**Apa itu Spectrogram?**
Spectrogram adalah representasi visual dari spektrum frekuensi suatu sinyal seiring berjalannya waktu. Dalam kata lain, spectrogram memungkinkan kita untuk "melihat" bagaimana energi dalam sinyal didistribusikan di berbagai frekuensi seiring waktu.

**Mengapa Menggunakan Spectrogram?**
Spectrogram sangat berguna dalam banyak aplikasi, termasuk:

- Analisis Suara dan Musik: Menentukan nada, harmonik, dan komponen lain dari suara.
- Pengenalan Suara: Membedakan kata-kata atau suara berdasarkan ciri spektrum frekuensinya.
- Kedokteran: Diagnostik medis menggunakan suara (seperti detak jantung).
- Telekomunikasi: Mengidentifikasi gangguan sinyal.

**Bagaimana Spectrogram Bekerja?**
Untuk menghasilkan spectrogram, sinyal waktu kontinu pertama-tama dibagi menjadi potongan-potongan pendek atau "frame". Kemudian, kita menghitung spektrum frekuensi untuk setiap frame tersebut. Akhirnya, spektrum dari semua frame disusun bersama untuk menghasilkan gambar 2D, di mana sumbu x mewakili waktu, sumbu y mewakili frekuensi, dan warna mewakili kekuatan sinyal pada frekuensi tertentu pada waktu tertentu.

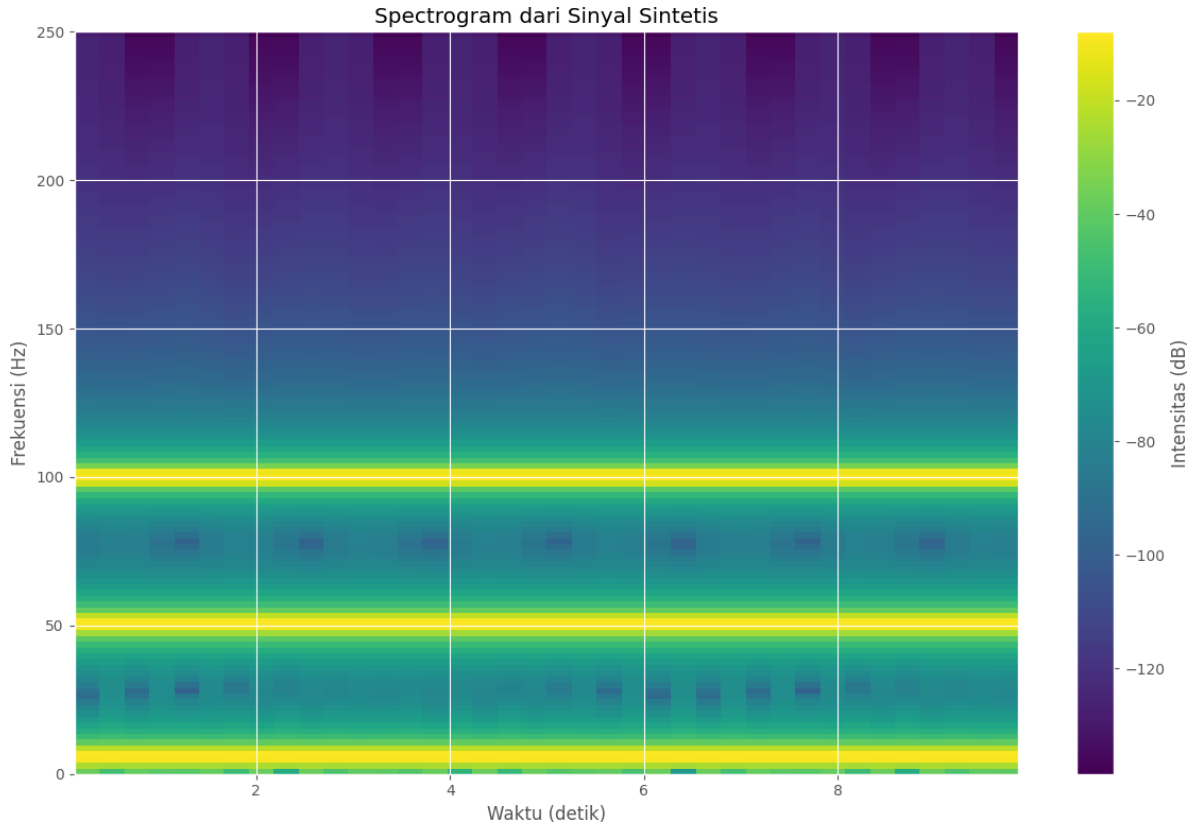
### 2.17.1. Basic Spectrogram

Untuk demonstrasi awal, mari kita buat sebuah sinyal sintetis yang menggabungkan beberapa gelombang sinus dengan frekuensi berbeda. Kemudian kita akan visualisasikan sinyal tersebut menggunakan spectrogram.

```
import numpy as np
import matplotlib.pyplot as plt

# Membuat sinyal sintetis
Fs = 500  # Frekuensi sampling
T = 1/Fs  # Interval sampling
t = np.arange(0,10,T)  # Vektor waktu

# Membuat sinyal yang merupakan kombinasi dari beberapa gelombang sinus
dengan frekuensi berbeda
f1, f2, f3 = 5, 50, 100  # Frekuensi
```

```
y1 = np.sin(2*np.pi*f1*t)
y2 = np.sin(2*np.pi*f2*t)
y3 = np.sin(2*np.pi*f3*t)
y = y1 + y2 + y3

# Menampilkan spectrogram
plt.figure(figsize=(12, 8))
Pxx, freqs, bins, im = plt.specgram(y, NFFT=256, Fs=Fs, noverlap=128,
cmap='viridis')
plt.colorbar(label='Intensitas (dB)')
plt.title('Spectrogram dari Sinyal Sintetis')
plt.xlabel('Waktu (detik)')
plt.ylabel('Frekuensi (Hz)')
plt.tight_layout()
plt.show()
```

Output:




Dalam visualisasi di atas, kamu bisa melihat spectrogram dari sinyal sintetis yang kita buat. Warna yang lebih terang pada gambar menunjukkan frekuensi yang memiliki amplitudo yang lebih besar pada waktu tertentu.

- Garis Horizontal Pertama (Paling Bawah): Mewakili gelombang sinus dengan frekuensi 5 Hz.
- Garis Horizontal Kedua: Mewakili gelombang sinus dengan frekuensi 50 Hz.

- Garis Horizontal Ketiga (Paling Atas): Mewakili gelombang sinus dengan frekuensi 100 Hz.

Dari spectrogram ini, kita dapat dengan mudah mengidentifikasi komponen frekuensi dari sinyal dan bagaimana amplitudo mereka berubah seiring waktu. Itulah kekuatan dari spectrogram!

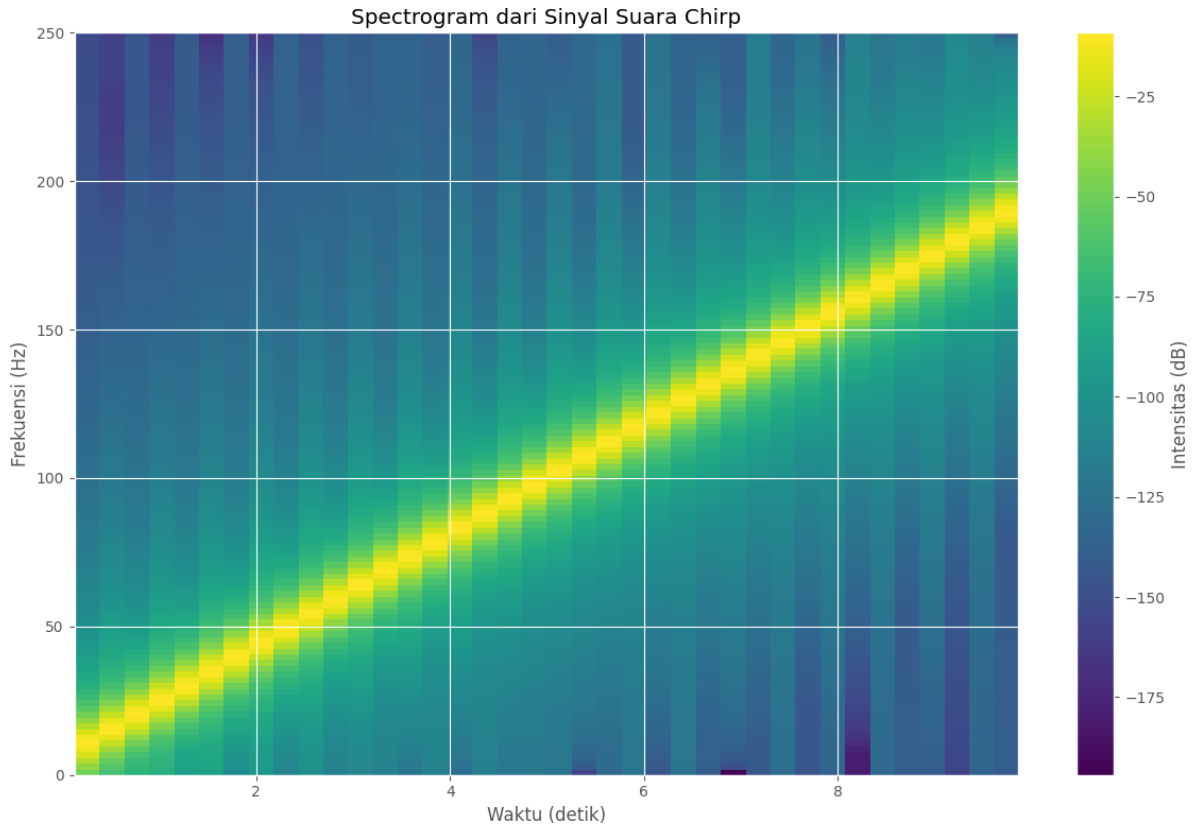
### 2.17.2. Menggunakan Spectrogram dalam Analisis Suara

Salah satu aplikasi paling umum dari spectrogram adalah dalam analisis suara. Dengan mengamati spectrogram dari klip suara, kita bisa mendapatkan wawasan tentang karakteristik suara tersebut: nada dasar, harmonik, serta fitur lainnya.

Mari kita coba dengan sinyal suara yang sedikit lebih kompleks. Misalkan kita memiliki suara yang dimulai dengan frekuensi rendah dan meningkat seiring waktu. Ini bisa mewakili suara instrumen musik yang dimainkan dengan cara tertentu atau suara binatang. Mari kita ciptakan sinyal seperti itu dan lihat spectrogramnya.

```
# Membuat sinyal suara yang frekuensinya meningkat seiring waktu
y_chirp = np.sin(2 * np.pi * (f1 + (f3 - f1) * t / max(t)) * t)

# Menampilkan spectrogram
plt.figure(figsize=(12, 8))
Pxx, freqs, bins, im = plt.specgram(y_chirp, NFFT=256, Fs=Fs,
noverlap=128, cmap='viridis')
plt.colorbar(label='Intensitas (dB)')
plt.title('Spectrogram dari Sinyal Suara Chirp')
plt.xlabel('Waktu (detik)')
plt.ylabel('Frekuensi (Hz)')
plt.tight_layout()
plt.show()
```

Output:




Pada visualisasi di atas, kita melihat representasi dari suara "chirp". Dalam kasus ini, "chirp" adalah sinyal yang frekuensinya meningkat (atau kadang-kadang menurun) seiring berjalannya waktu.

Kamu bisa melihat bagaimana intensitas bergerak dari bawah ke atas, menunjukkan kenaikan frekuensi sinyal dari waktu ke waktu. Ini adalah karakteristik khas dari sinyal "chirp". Spectrogram ini memberi kita pemahaman mendalam tentang bagaimana suara tersebut berubah seiring waktu dalam hal frekuensi.

# MODUL 3: Pengaturan dan Personalisasi Plot

## 3.1. Mengatur Judul dan Label

Dalam dunia visualisasi data, detail-detail kecil sering kali membuat perbedaan besar. Bayangkan menemukan grafik yang menarik di internet, tetapi tanpa judul atau label sumbu. Kamu mungkin bertanya-tanya: "Apa yang sedang aku lihat? Apa yang dimaksud dengan angka-angka ini? Apa tujuan dari grafik ini?" Inilah mengapa judul dan label sangat penting.

**Mengapa Judul dan Label Penting?**

- Klarifikasi: Judul dan label memberikan konteks kepada pembaca tentang apa yang mereka lihat.
- Detail: Label bisa memberikan informasi tentang satuan, skala, atau sumber data.
- Profesionalitas: Grafik yang rapi dengan judul dan label yang sesuai meningkatkan kredibilitasmu sebagai peneliti atau analis.

### 3.1.1. Memulai dengan Judul

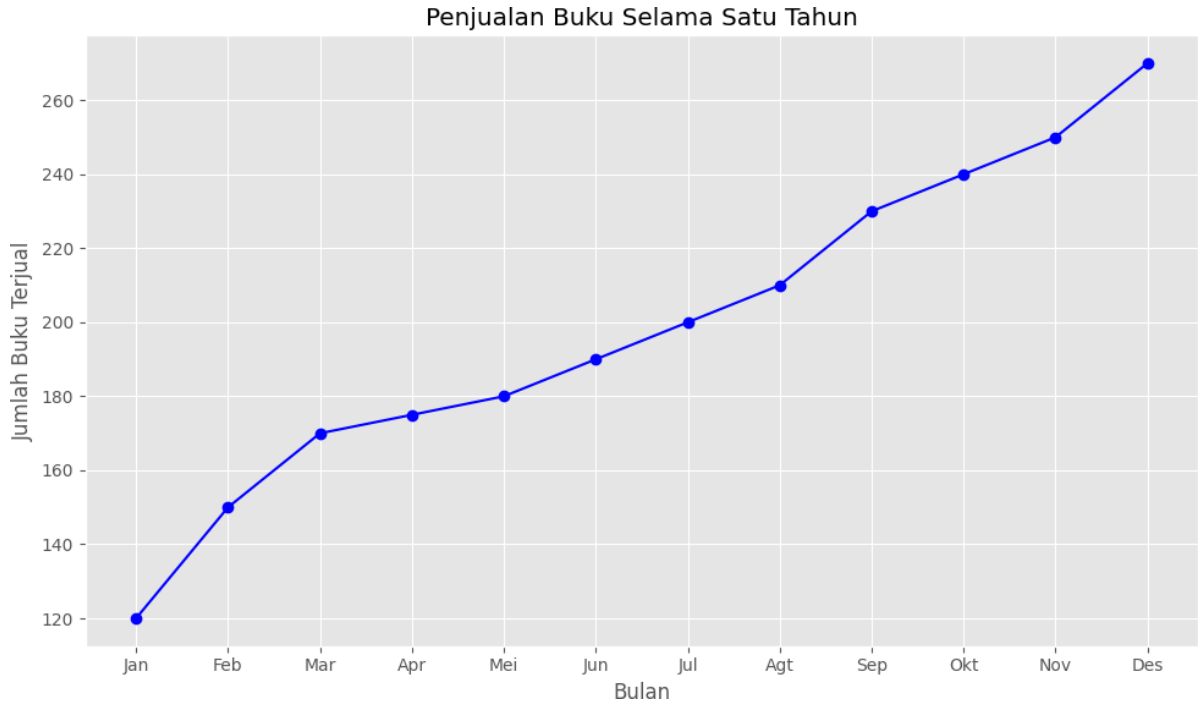
Judul harus memberikan gambaran umum tentang apa yang disajikan dalam grafik. Biasanya, judul yang baik adalah singkat, padat, dan informatif.

Mari kita mulai dengan contoh sederhana. Misalkan kamu memiliki data tentang penjualan buku di sebuah toko selama satu tahun. Dengan Matplotlib, menambahkan judul sangat mudah. Mari kita buat data sintetisnya terlebih dahulu.

```
import pandas as pd

# Membuat data sintetis penjualan buku
bulan = ['Jan', 'Feb', 'Mar', 'Apr', 'Mei', 'Jun', 'Jul', 'Agt', 'Sep',
'Okt', 'Nov', 'Des']
penjualan = [120, 150, 170, 175, 180, 190, 200, 210, 230, 240, 250, 270]

df_penjualan = pd.DataFrame({
    'Bulan': bulan,
    'Penjualan': penjualan
})

# Plotting data
plt.figure(figsize=(10, 6))
plt.plot(df_penjualan['Bulan'], df_penjualan['Penjualan'], marker='o',
color='b', linestyle='-')
```

```
plt.title('Penjualan Buku Selama Satu Tahun')
plt.xlabel('Bulan')
plt.ylabel('Jumlah Buku Terjual')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:




Seperti yang kamu lihat, grafik di atas menunjukkan penjualan buku selama satu tahun. Judul "Penjualan Buku Selama Satu Tahun" memberikan gambaran umum tentang apa yang ditampilkan oleh grafik. Sementara label pada sumbu x dan y ("Bulan" dan "Jumlah Buku Terjual") memberikan informasi lebih lanjut tentang data yang disajikan.

### 3.1.2. Memanfaatkan Label Sumbu

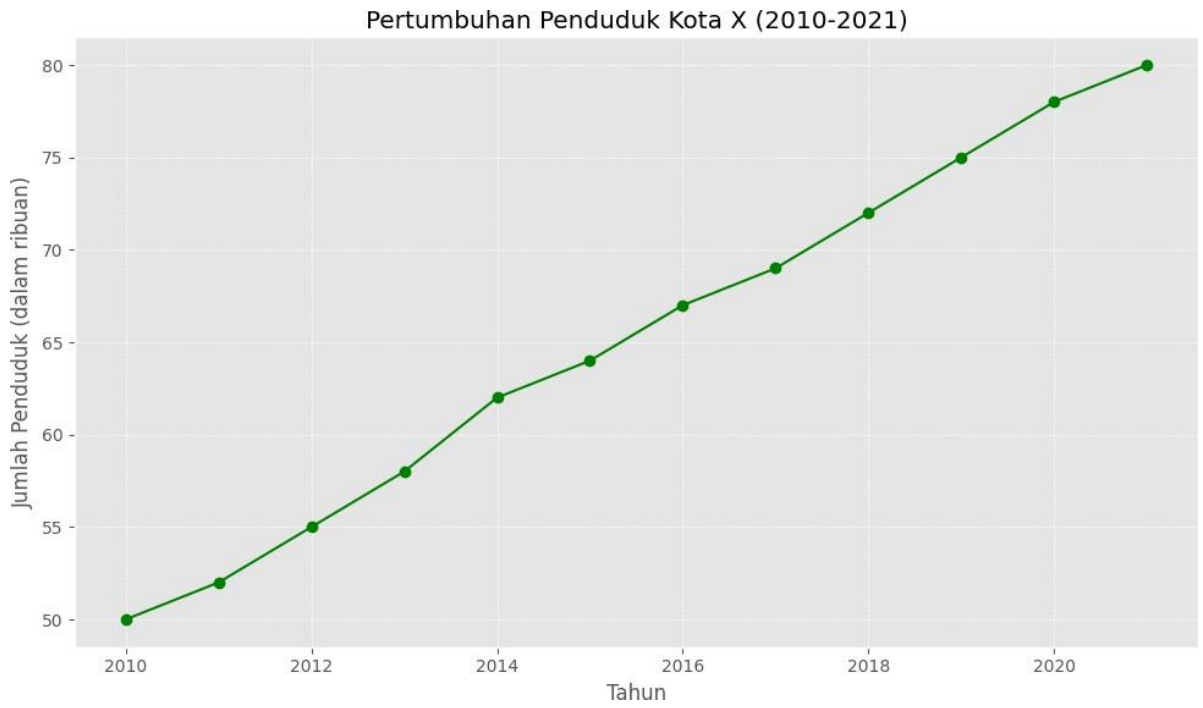
Menggunakan label sumbu yang tepat sangat penting untuk memastikan bahwa audiens memahami skala dan jenis data yang disajikan. Label sumbu harus jelas dan, jika perlu, menyertakan satuan pengukuran.

Mari kita pertimbangkan contoh lain. Misalkan kamu ingin menampilkan grafik tentang pertumbuhan penduduk di sebuah kota selama beberapa tahun terakhir. Di sini, penting untuk menunjukkan bahwa angka pada sumbu y mewakili "ribuan" penduduk. Mari kita lihat bagaimana ini bisa dilakukan dengan Matplotlib.

```
# Membuat data sintetis pertumbuhan penduduk
tahun = list(range(2010, 2022))
```

```
penduduk = [50, 52, 55, 58, 62, 64, 67, 69, 72, 75, 78, 80]  # dalam
ribuan

df_penduduk = pd.DataFrame({
    'Tahun': tahun,
    'Penduduk': penduduk
})

# Plotting data
plt.figure(figsize=(10, 6))
plt.plot(df_penduduk['Tahun'], df_penduduk['Penduduk'], marker='o',
color='g', linestyle='-')
plt.title('Pertumbuhan Penduduk Kota X (2010-2021)')
plt.xlabel('Tahun')
plt.ylabel('Jumlah Penduduk (dalam ribuan)')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dari grafik di atas, kita bisa melihat bagaimana penduduk Kota X telah berkembang selama beberapa tahun terakhir. Dengan menambahkan "dalam ribuan" ke label sumbu y, kita memberikan informasi yang jelas kepada pembaca bahwa, misalnya, angka 60 pada sumbu y sebenarnya mewakili 60.000 penduduk.

### 3.1.3. Mengatur Ukuran dan Gaya Font

Kadang-kadang, kamu mungkin ingin menekankan judul atau membuat label lebih mudah dibaca dengan mengubah ukuran font atau gayanya. Matplotlib memungkinkan kamu untuk melakukan hal ini dengan mudah. Sebagai contoh, mari kita buat judul lebih menonjol dan label sumbu dengan font miring.

```
# Plotting dengan pengaturan font khusus
plt.figure(figsize=(10, 6))
plt.plot(df_penjualan['Bulan'], df_penjualan['Penjualan'], marker='o',
color='b', linestyle='-')
plt.title('Penjualan Buku Selama Satu Tahun', fontsize=16,
fontweight='bold')
plt.xlabel('Bulan', fontsize=12, style='italic')
plt.ylabel('Jumlah Buku Terjual', fontsize=12, style='italic')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:


Dengan sedikit penyesuaian pada font, grafik kita sekarang memiliki penekanan yang lebih kuat pada judul, sementara label sumbu dengan font miring memberikan sentuhan estetika tambahan. Ingatlah bahwa tujuan utamanya adalah untuk membuat grafik lebih mudah dibaca dan dipahami, jadi pastikan untuk tidak berlebihan dengan kustomisasi.

### 3.1.4. Menggunakan Font Eksternal

Matplotlib memungkinkan kamu untuk menggunakan font eksternal, yang bisa sangat berguna jika kamu ingin menyelaraskan visualisasi dengan branding perusahaan atau tema khusus. Misalnya, mungkin kamu memiliki font khusus yang digunakan dalam semua materi pemasaran dan kamu ingin grafik kamu sesuai dengan estetika tersebut.

Mari kita lihat bagaimana kamu bisa mengatur font dengan menggunakan rcParams dari Matplotlib. (Catatan: Di sini kita akan menggunakan font default, karena kita tidak memiliki akses ke font eksternal di lingkungan ini.)

### 3.1.5. Posisi Judul

Selain mengubah ukuran dan gaya font judul, kamu juga bisa mengubah posisinya. Secara default, judul ditempatkan di atas sumbu plot, tetapi mungkin dalam beberapa kasus kamu ingin menggesernya sedikit atau menempatkannya di tengah.

```
import matplotlib.pyplot as plt
import pandas as pd

# Membuat data sintetis penjualan buku
bulan = ['Jan', 'Feb', 'Mar', 'Apr', 'Mei', 'Jun', 'Jul', 'Agt', 'Sep',
'Okt', 'Nov', 'Des']
penjualan = [120, 150, 170, 175, 180, 190, 200, 210, 230, 240, 250, 270]

df_penjualan = pd.DataFrame({
    'Bulan': bulan,
    'Penjualan': penjualan
})

# Plotting dengan pengaturan posisi judul
plt.figure(figsize=(10, 6))
plt.plot(df_penjualan['Bulan'], df_penjualan['Penjualan'], marker='o',
color='b', linestyle='-')
plt.title('Penjualan Buku Selama Satu Tahun', fontsize=16,
fontweight='bold',x=0.27 , y=0.91)
plt.xlabel('Bulan', fontsize=12)
plt.ylabel('Jumlah Buku Terjual', fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:


Seperti yang kamu lihat, dengan sedikit penyesuaian pada parameter y dari fungsi title, kita berhasil menggeser judul sedikit lebih tinggi dari posisi defaultnya. Hal ini mungkin berguna jika kamu memiliki elemen lain di bagian atas grafik yang dapat mengganggu pembacaan judul.

### 3.1.6. Mengatur Label Sumbu dengan Rotasi

Dalam beberapa situasi, label sumbu x bisa bertumpuk satu sama lain, terutama jika label tersebut panjang atau ada banyak titik data. Salah satu solusi untuk mengatasi masalah ini adalah dengan memutar label.

Mari kita coba memutar label sumbu x agar lebih mudah dibaca.

```
# Plotting dengan label sumbu x yang diputar
plt.figure(figsize=(12, 6))
plt.plot(df_penjualan['Bulan'], df_penjualan['Penjualan'], marker='o',
color='r', linestyle='-')
plt.title('Penjualan Buku Selama Satu Tahun', fontsize=16)
plt.xlabel('Bulan', fontsize=12)
plt.ylabel('Jumlah Buku Terjual', fontsize=12)
plt.xticks(rotation=45)  # Memutar label sumbu x sebesar 45 derajat
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:


## 3.2. Pengaturan Warna dan Marker

Warna dan marker merupakan dua komponen penting dalam visualisasi data, terutama dalam grafik garis dan scatter plot. Warna dapat digunakan untuk membedakan antar kategori, menyoroti tren tertentu, atau menampilkan kepadatan data. Sementara marker memberikan representasi visual dari setiap titik data, memungkinkan pembaca untuk melacak nilai tertentu atau menyoroti outlier.

Dalam subbab ini, kita akan mengeksplorasi berbagai cara untuk menyesuaikan warna dan marker dalam plot Matplotlib. Dari memilih palet warna yang sesuai hingga menggunakan marker yang berbeda untuk membedakan kategori, kita akan melihat bagaimana perubahan kecil ini dapat membuat perbedaan besar dalam interpretasi dan estetika grafik.

**Memilih Warna Plot**

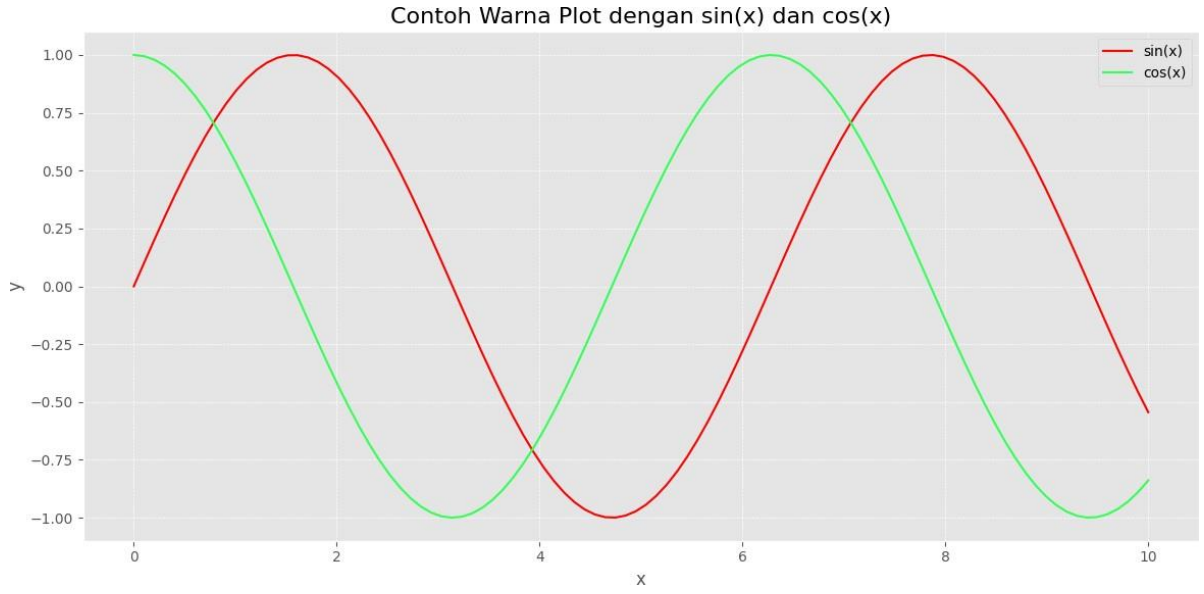
Dalam Matplotlib, kamu bisa dengan mudah mengatur warna plot dengan menggunakan parameter color. Terdapat berbagai cara untuk menentukan warna, diantaranya:

Nama warna dasar seperti 'red', 'blue', 'green'.
Kode warna HEX, seperti '#FF5733'.
Warna RGB dalam format tuple, seperti (0.5, 0.2, 0.8).
Mari kita coba beberapa metode ini.

```
# Membuat data sintetis
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Plotting dengan warna berbeda
```

```
plt.figure(figsize=(12, 6))

# Menggunakan nama warna
plt.plot(x, y1, label='sin(x)', color='red')

# Menggunakan kode warna HEX
plt.plot(x, y2, label='cos(x)', color='#33FF57')

plt.title('Contoh Warna Plot dengan sin(x) dan cos(x)', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:




Dalam contoh di atas, garis untuk fungsi sin(x) diwarnai merah, sedangkan garis untuk fungsi cos(x) memiliki warna hijau yang ditentukan dengan kode HEX. Pemilihan warna yang kontras memudahkan pembaca untuk membedakan antara dua fungsi tersebut.

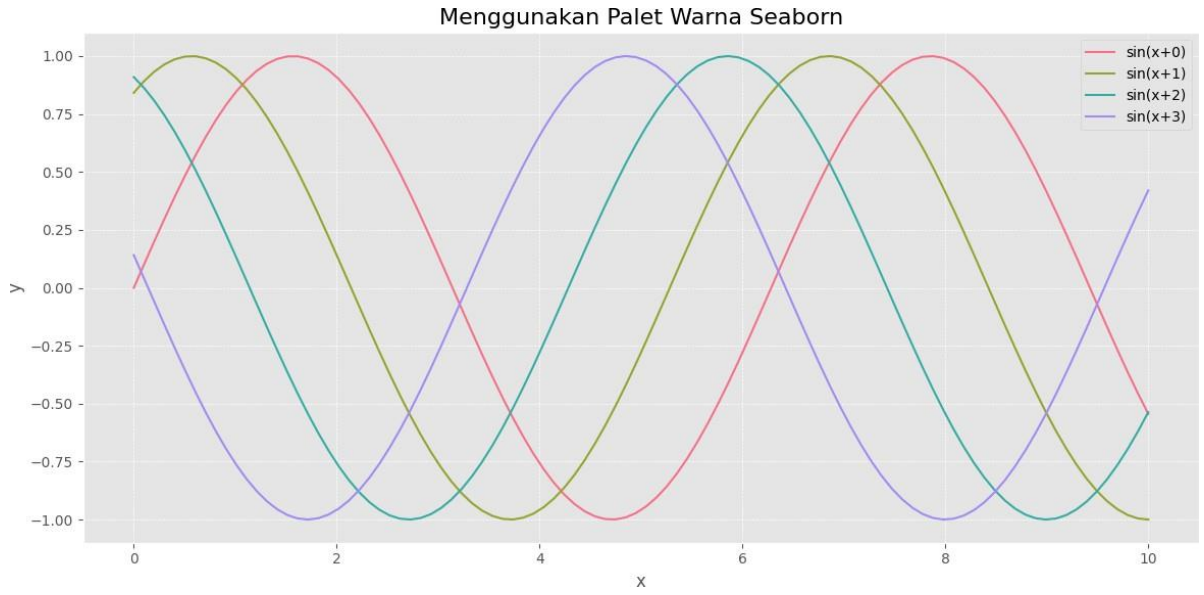
### 3.2.1. Menggunakan Palet Warna

Terutama dalam kasus di mana kamu memiliki banyak kategori atau seri data, mungkin sulit untuk memilih warna yang berbeda untuk setiap kategori. Di sinilah palet warna datang untuk menyelamatkan! Matplotlib dan library terkait seperti seaborn menawarkan palet warna yang telah ditentukan yang bisa kamu gunakan.

Mari kita coba plot beberapa seri data dengan palet warna seaborn.

```
import seaborn as sns

# Membuat data sintetis
x = np.linspace(0, 10, 100)
y_values = [np.sin(x), np.sin(x+1), np.sin(x+2), np.sin(x+3)]

# Menggunakan palet warna seaborn
colors = sns.color_palette("husl", len(y_values))

plt.figure(figsize=(12, 6))
for i, y in enumerate(y_values):
    plt.plot(x, y, label=f'sin(x+{i})', color=colors[i])

plt.title('Menggunakan Palet Warna Seaborn', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:




Dengan palet warna, kita dapat dengan mudah memilih warna yang harmonis dan berbeda untuk setiap seri data. Seaborn menyediakan berbagai palet warna yang dapat kamu gunakan, seperti "husl", "pastel", "deep", dan banyak lagi. Kamu dapat menyesuaikan palet ini sesuai kebutuhan visualisasi kamu.

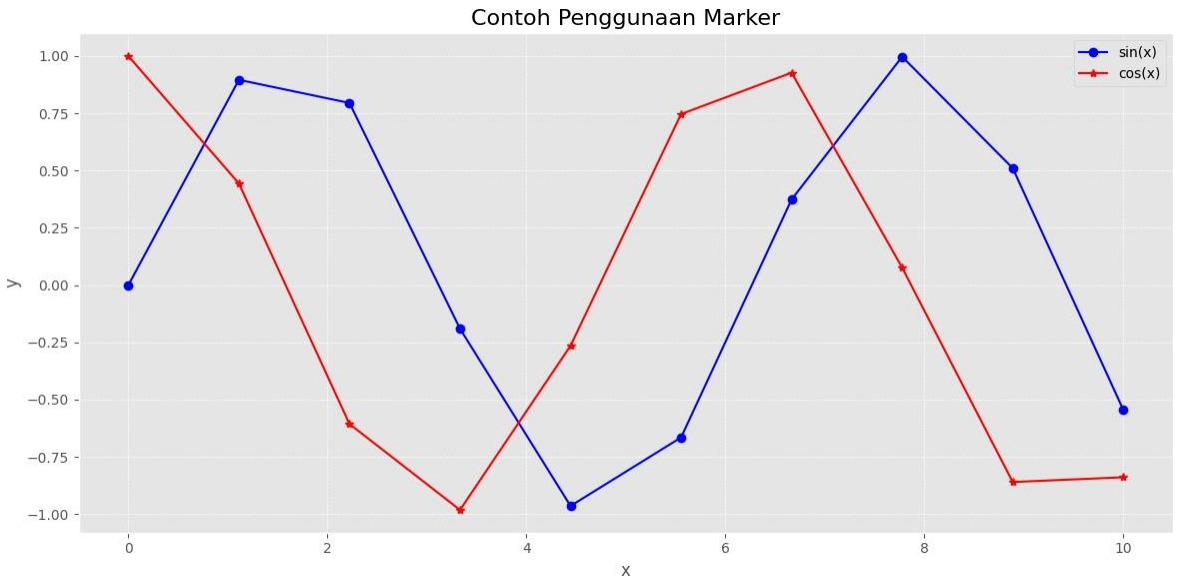
### 3.2.2. Mengatur Marker

Dalam grafik garis atau scatter plot, marker digunakan untuk menunjukkan posisi titik data. Matplotlib menawarkan berbagai pilihan marker, seperti lingkaran ('o'), bintang ('*'), persegi ('s'), dan banyak lainnya.

Mari kita coba beberapa jenis marker ini pada grafik.

```
# Membuat data sintetis
x = np.linspace(0, 10, 10)
y1 = np.sin(x)
y2 = np.cos(x)

# Plotting dengan marker berbeda
plt.figure(figsize=(12, 6))

# Menggunakan marker lingkaran
plt.plot(x, y1, 'o-', label='sin(x)', color='blue')

# Menggunakan marker bintang
plt.plot(x, y2, '*-', label='cos(x)', color='red')

plt.title('Contoh Penggunaan Marker', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:



Dalam contoh di atas, kita menggunakan marker lingkaran untuk seri data sin(x) dan marker bintang untuk cos(x). Selain membedakan antara dua seri data, marker juga membantu pembaca dalam mengidentifikasi posisi titik data yang sebenarnya.

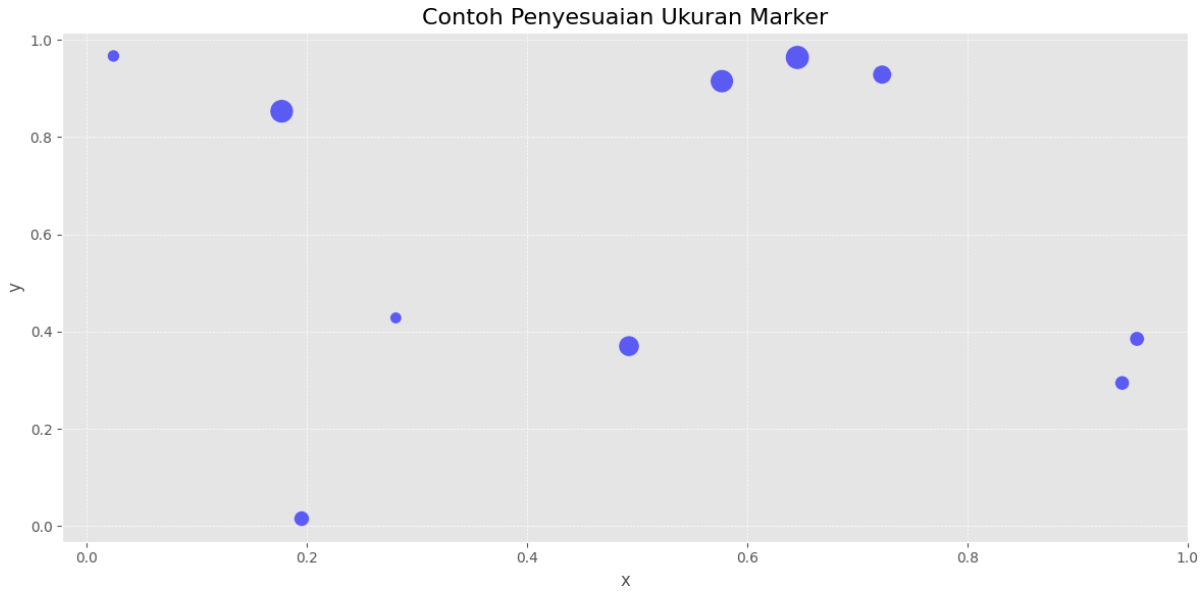
### 3.2.3. Mengatur Ukuran Marker

Kadang-kadang, kamu mungkin ingin menyesuaikan ukuran marker untuk menyoroti titik data tertentu atau untuk mencerminkan dimensi lain dari data. Misalnya, dalam scatter plot, ukuran marker dapat digunakan untuk menunjukkan kepadatan atau jumlah dari sesuatu.

Mari kita lihat bagaimana kita dapat menyesuaikan ukuran marker dalam plot.

```
# Membuat data sintetis
x = np.random.rand(10)
y = np.random.rand(10)
sizes = np.random.randint(50, 300, 10)  # Menghasilkan ukuran marker
yang berbeda

plt.figure(figsize=(12, 6))
plt.scatter(x, y, s=sizes, color='blue', alpha=0.6, edgecolors='w',
linewidth=0.5)

plt.title('Contoh Penyesuaian Ukuran Marker', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Pada contoh di atas, ukuran marker menunjukkan seberapa besar suatu titik dalam dataset sintetis. Dengan menyesuaikan ukuran marker, kamu bisa menambahkan dimensi tambahan ke visualisasi data kamu, membuatnya lebih informatif dan menarik.

Sebagai catatan, saat memodifikasi ukuran marker, pastikan untuk menjaga keseimbangan antara estetika dan keterbacaan. Marker yang terlalu besar dapat menutupi data lainnya, sementara marker yang terlalu kecil mungkin sulit dilihat.

### 3.2.4. Mengatur Opasitas dengan Parameter Alpha

Selain warna dan marker, opasitas (atau transparansi) adalah salah satu elemen yang bisa kamu modifikasi untuk meningkatkan estetika atau keterbacaan plot. Mengatur opasitas terutama berguna saat plot memiliki banyak titik data yang tumpang tindih, sehingga kamu bisa melihat distribusi dan kepadatan data dengan lebih jelas.

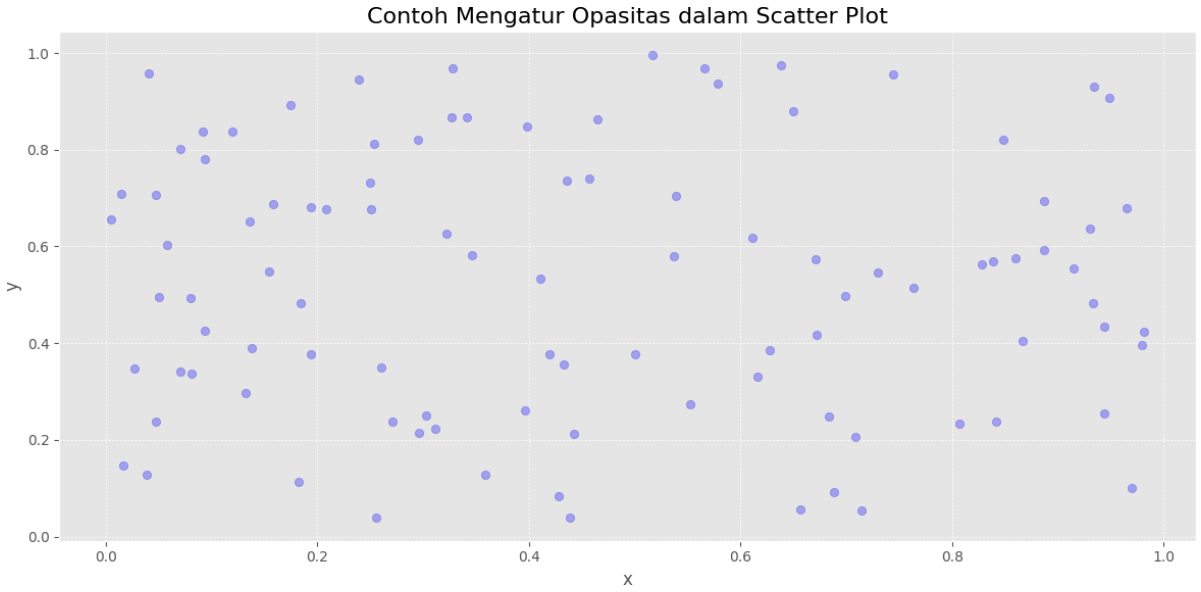
Dalam Matplotlib, kamu bisa mengatur opasitas dengan parameter alpha. Nilai alpha berkisar antara 0 (sepenuhnya transparan) hingga 1 (sepenuhnya tidak transparan).

Mari kita lihat bagaimana opasitas bekerja dengan plot titik.

```
# Membuat data sintetis
x = np.random.rand(100)
y = np.random.rand(100)

plt.figure(figsize=(12, 6))
plt.scatter(x, y, color='blue', alpha=0.3)  # Mengatur opasitas titik

plt.title('Contoh Mengatur Opasitas dalam Scatter Plot', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
```

```
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dengan mengatur opasitas titik dalam scatter plot, kamu dapat dengan mudah melihat area mana yang memiliki kepadatan titik yang lebih tinggi. Area dengan tumpang tindih titik yang lebih banyak akan tampak lebih gelap.

### 3.2.5. Menggunakan Gradients untuk Menyoroti Intensitas

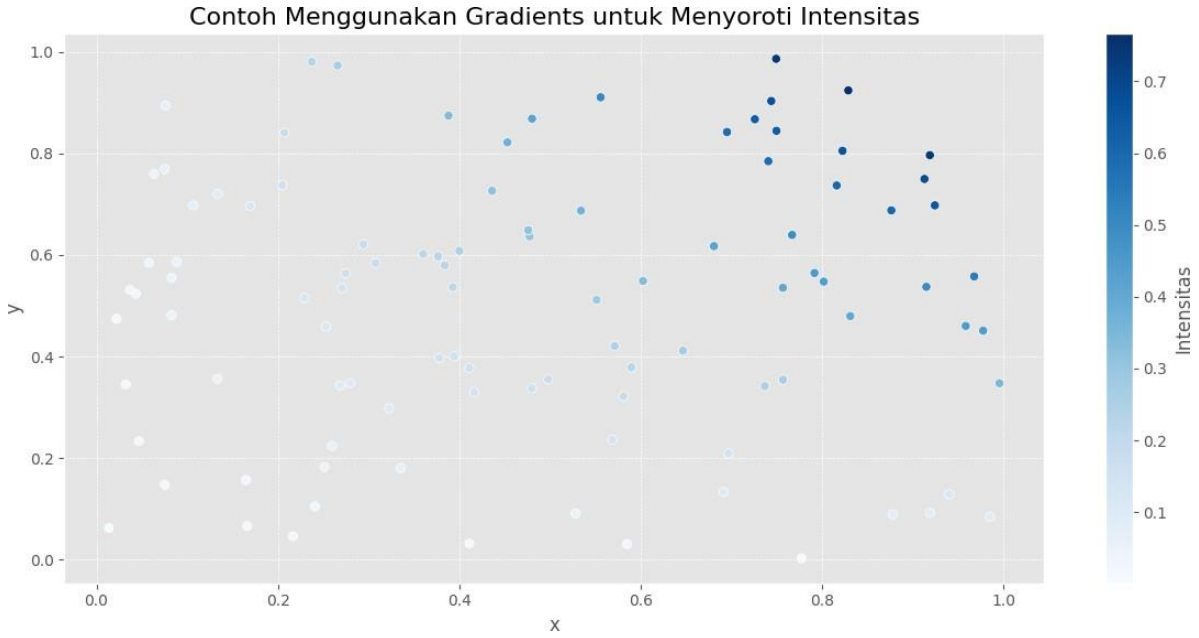
Dalam beberapa kasus, kamu mungkin ingin menggunakan gradasi warna untuk menyoroti intensitas atau kepadatan data. Ini bisa dilakukan dengan mengkombinasikan warna dan opasitas.

Misalnya, jika kamu memiliki data yang menunjukkan frekuensi tertentu pada koordinat tertentu, kamu bisa menggunakan gradasi warna untuk menunjukkan frekuensi tersebut. Semakin gelap warnanya, semakin tinggi frekuensinya.

```
# Membuat data sintetis
x = np.random.rand(100)
y = np.random.rand(100)
colors = x * y # Semakin besar produk x dan y, semakin gelap warnanya

plt.figure(figsize=(12, 6))
scatter = plt.scatter(x, y, c=colors, cmap='Blues', edgecolors='w',
linewidth=0.5)

# Menambahkan colorbar untuk indikasi intensitas
```

```
cbar = plt.colorbar(scatter)
cbar.set_label('Intensitas', fontsize=12)

plt.title('Contoh Menggunakan Gradients untuk Menyoroti Intensitas',
fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dalam contoh di atas, kita menggunakan gradasi warna biru untuk menyoroti intensitas data. Semakin tinggi produk dari koordinat x dan y, semakin gelap warna titiknya. Ini memberi kamu wawasan tambahan tentang distribusi dan intensitas data tanpa perlu menambahkan informasi tambahan.

### 3.2.6. Mengkombinasikan Warna, Opasitas, dan Marker

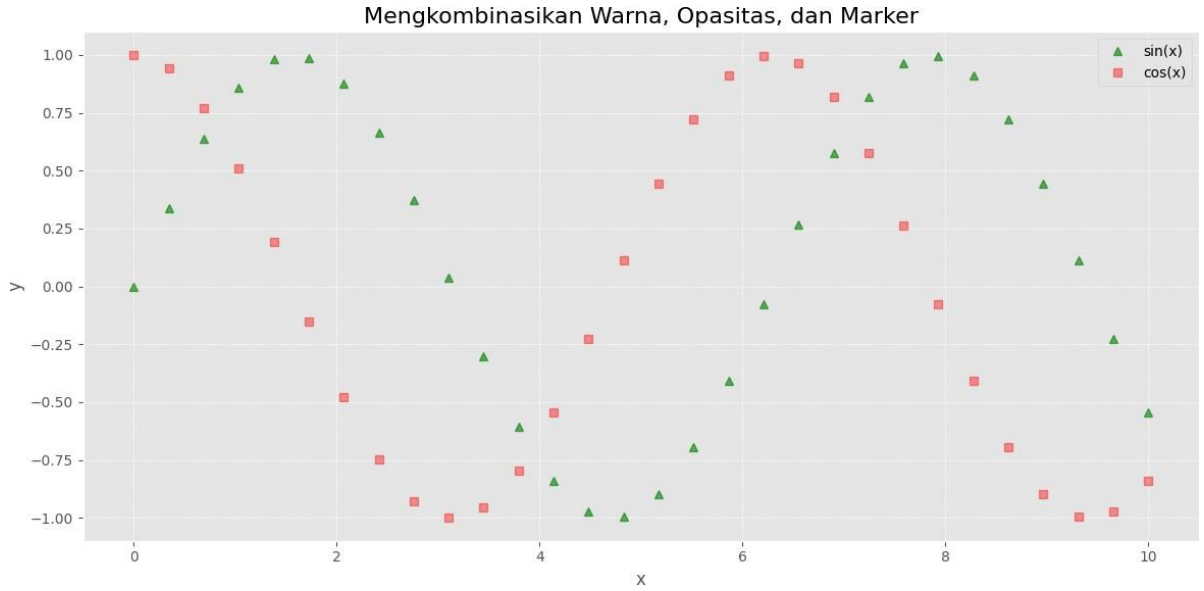
Ketika kamu memahami dasar-dasar warna, opasitas, dan marker, kamu bisa mulai mengkombinasikan ketiganya untuk menciptakan visualisasi yang sangat informatif dan menarik. Cobalah bereksperimen dengan kombinasi berbeda untuk menemukan tampilan yang paling sesuai dengan kebutuhanmu.

Mari kita lihat contoh di mana kita mengkombinasikan warna, opasitas, dan jenis marker yang berbeda.

```
# Membuat data sintetis
```

```
x = np.linspace(0, 10, 30)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(12, 6))

# Menggunakan marker segitiga dengan warna hijau dan opasitas 0.6
plt.plot(x, y1, '^', color='green', alpha=0.6, label='sin(x)')

# Menggunakan marker persegi dengan warna merah dan opasitas 0.4
plt.plot(x, y2, 's', color='red', alpha=0.4, label='cos(x)')

plt.title('Mengkombinasikan Warna, Opasitas, dan Marker', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```

Output:




Tampilan visualisasi di atas menunjukkan bagaimana kamu dapat mengkombinasikan warna, opasitas, dan marker untuk menciptakan plot yang informatif dan menarik secara visual. Elemen-elemen ini membantu dalam membedakan antara dua seri data dan memberikan konteks visual tambahan kepada pembaca.

Dengan memahami bagaimana cara mengatur warna, opasitas, dan marker, kamu dapat mengkustomisasi tampilan grafik sesuai dengan kebutuhanmu. Kombinasi dari semua elemen-elemen ini memungkinkan kamu untuk menyampaikan informasi dengan jelas dan

efektif, memastikan bahwa visualisasi data kamu tidak hanya menarik, tetapi juga informatif.

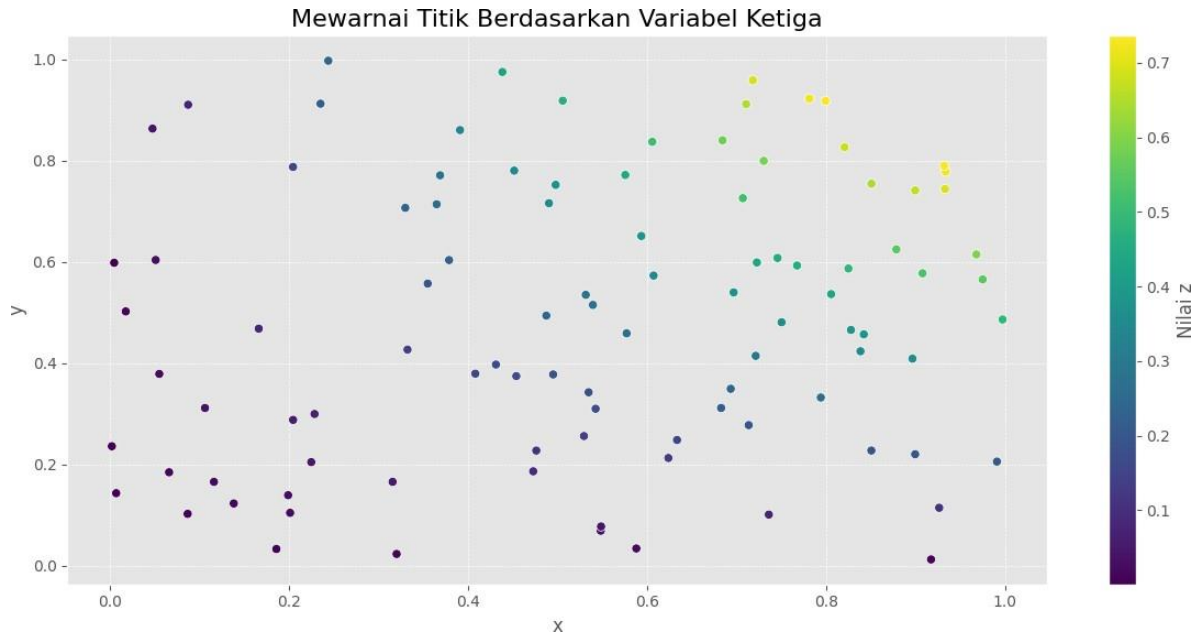
### 3.2.7. Penggunaan Warna Berdasarkan Variabel

Dalam beberapa kasus, kamu mungkin ingin memberi warna pada setiap titik berdasarkan variabel tertentu. Hal ini memungkinkan kamu untuk memvisualisasikan lebih dari dua dimensi dalam satu scatter plot. Misalnya, kamu memiliki data x dan y, namun kamu juga memiliki kolom z yang ingin kamu visualisasikan. Salah satu cara untuk melakukannya adalah dengan memberikan warna pada setiap titik berdasarkan nilai z.

Mari kita lihat contoh di mana kita memberikan warna pada titik berdasarkan variabel ketiga.

```
# Membuat data sintetis
x = np.random.rand(100)
y = np.random.rand(100)
z = x * y # Variabel ketiga

plt.figure(figsize=(12, 6))
scatter = plt.scatter(x, y, c=z, cmap='viridis', edgecolors='w',
linewidth=0.5)

# Menambahkan colorbar untuk indikasi nilai z
cbar = plt.colorbar(scatter)
cbar.set_label('Nilai z', fontsize=12)

plt.title('Mewarnai Titik Berdasarkan Variabel Ketiga', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dalam plot di atas, kita telah mewarnai setiap titik berdasarkan nilai dari variabel z (yang dalam kasus ini adalah hasil perkalian dari x dan y). Menggunakan warna dengan cara ini memberikan dimensi tambahan ke dalam visualisasi kita dan memungkinkan kita untuk melihat hubungan atau pola tambahan dalam data. Peta warna 'viridis' memberikan rentang warna dari kuning ke ungu, memberi tahu kita tentang rentang nilai z.

### 3.2.8. Kombinasi Warna dengan Ukuran

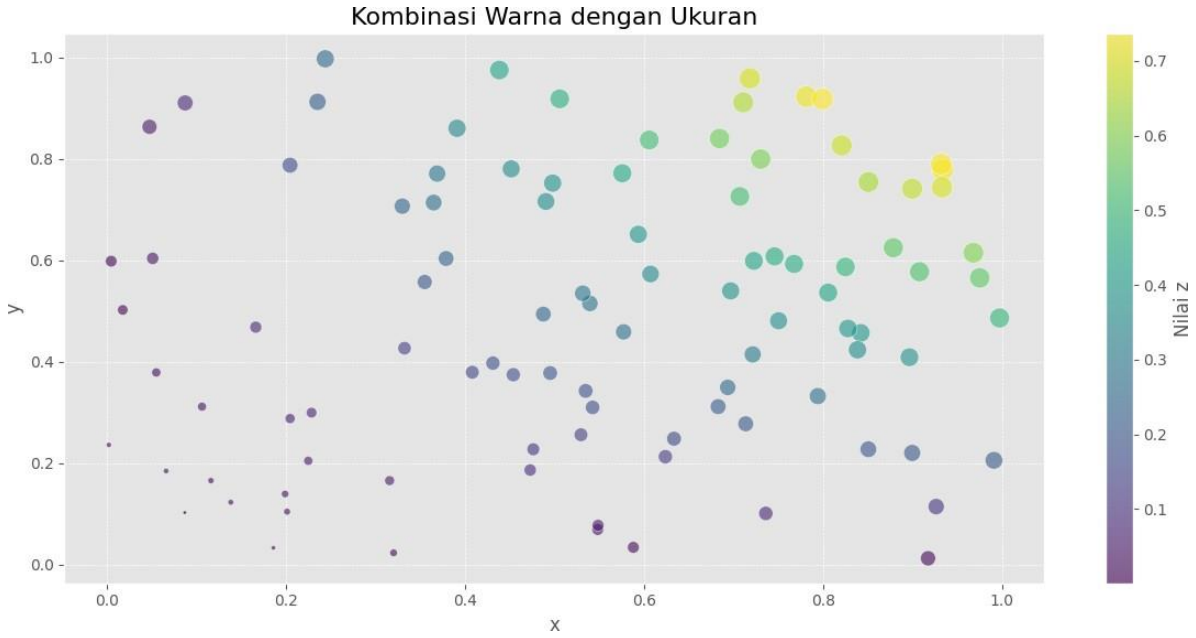
Selain menggunakan warna berdasarkan variabel, kamu juga bisa mengkombinasikannya dengan ukuran untuk memberikan lebih banyak informasi. Misalnya, warna bisa menunjukkan satu variabel, sementara ukuran menunjukkan variabel lainnya. Ini memungkinkan kamu untuk memvisualisasikan hingga empat dimensi dalam satu scatter plot.

Mari kita coba kombinasi tersebut:

```
# Membuat data sintetis
w = x + y  # Variabel keempat

# Normalisasi ukuran
sizes = (w - w.min()) / (w.max() - w.min()) * 200  # Mengubah rentang
ukuran menjadi 0-200

plt.figure(figsize=(12, 6))
scatter = plt.scatter(x, y, c=z, s=sizes, cmap='viridis',
edgecolors='w', linewidth=0.5, alpha=0.6)

# Menambahkan colorbar untuk indikasi nilai z
cbar = plt.colorbar(scatter)
```

```
cbar.set_label('Nilai z', fontsize=12)

plt.title('Kombinasi Warna dengan Ukuran', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

Output:




Dalam visualisasi di atas, setiap titik diberi warna berdasarkan nilai z dan ukurannya ditentukan oleh variabel w. Ini memberi kita kemampuan untuk memahami empat variabel sekaligus: posisi x, posisi y, warna (nilai z), dan ukuran (nilai w). Dengan teknik ini, kamu dapat memaksimalkan jumlah informasi yang disampaikan dalam satu plot.

### 3.2.9. Mewarnai Bar Plot

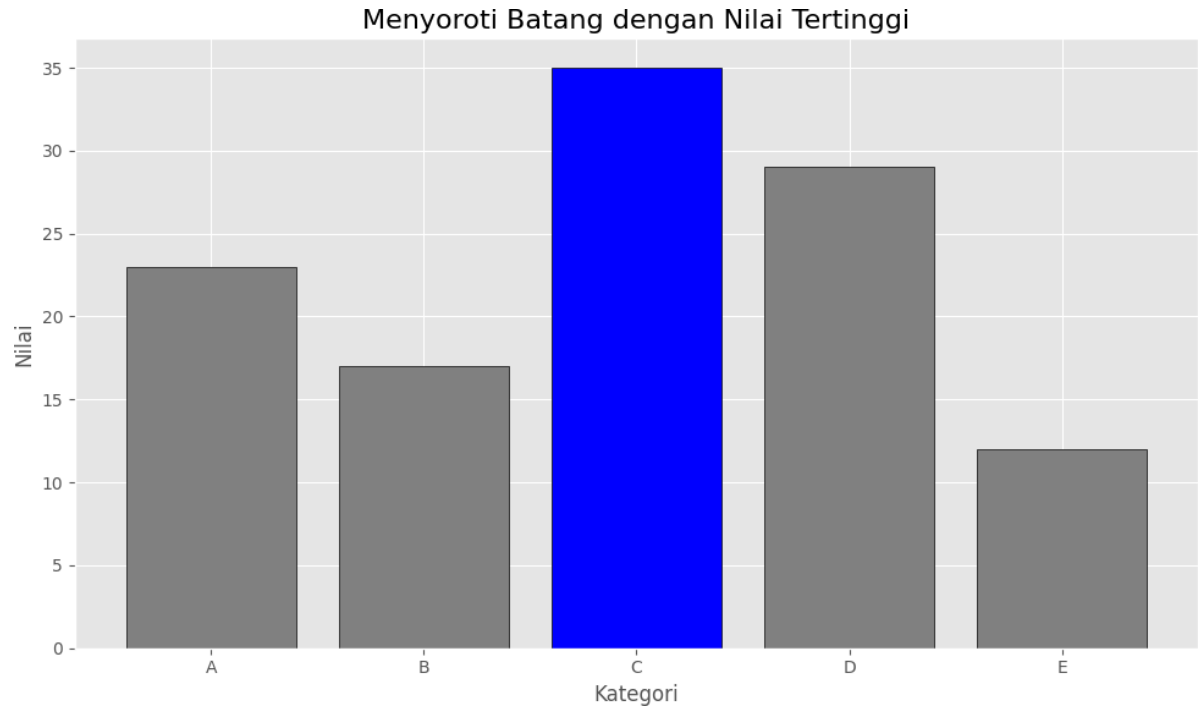
Tidak hanya pada scatter plot, kamu juga bisa memanipulasi warna pada bar plot atau jenis plot lainnya untuk memberikan informasi tambahan atau menyoroti bagian tertentu dari data. Misalnya, kamu mungkin ingin menyoroti batang dengan nilai tertinggi dalam bar plot dengan warna berbeda.

Mari kita lihat contoh sederhananya:

```
# Membuat data sintetis
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 17, 35, 29, 12]
```

```
# Menentukan warna berdasarkan nilai
colors = ['blue' if value == max(values) else 'gray' for value in
values]

plt.figure(figsize=(10, 6))
bars = plt.bar(categories, values, color=colors, edgecolor='black')

plt.title('Menyoroti Batang dengan Nilai Tertinggi', fontsize=16)
plt.xlabel('Kategori', fontsize=12)
plt.ylabel('Nilai', fontsize=12)
plt.tight_layout()
plt.show()
```

Output:




Pada bar plot di atas, kita menyoroti batang dengan nilai tertinggi dengan warna biru, sementara batang lainnya diwarnai abu-abu. Teknik semacam ini sangat berguna untuk segera menarik perhatian ke bagian data yang paling penting atau menarik.

Dengan memahami bagaimana mengkustomisasi warna dan marker, kamu dapat membuat visualisasi yang lebih informatif dan menarik. Tidak hanya itu, kamu juga bisa menyesuaikan tampilan grafik untuk cocok dengan tema atau branding tertentu. Dengan begitu, kamu bisa memastikan bahwa grafik kamu tidak hanya informatif, tetapi juga estetis dan menarik secara visual.

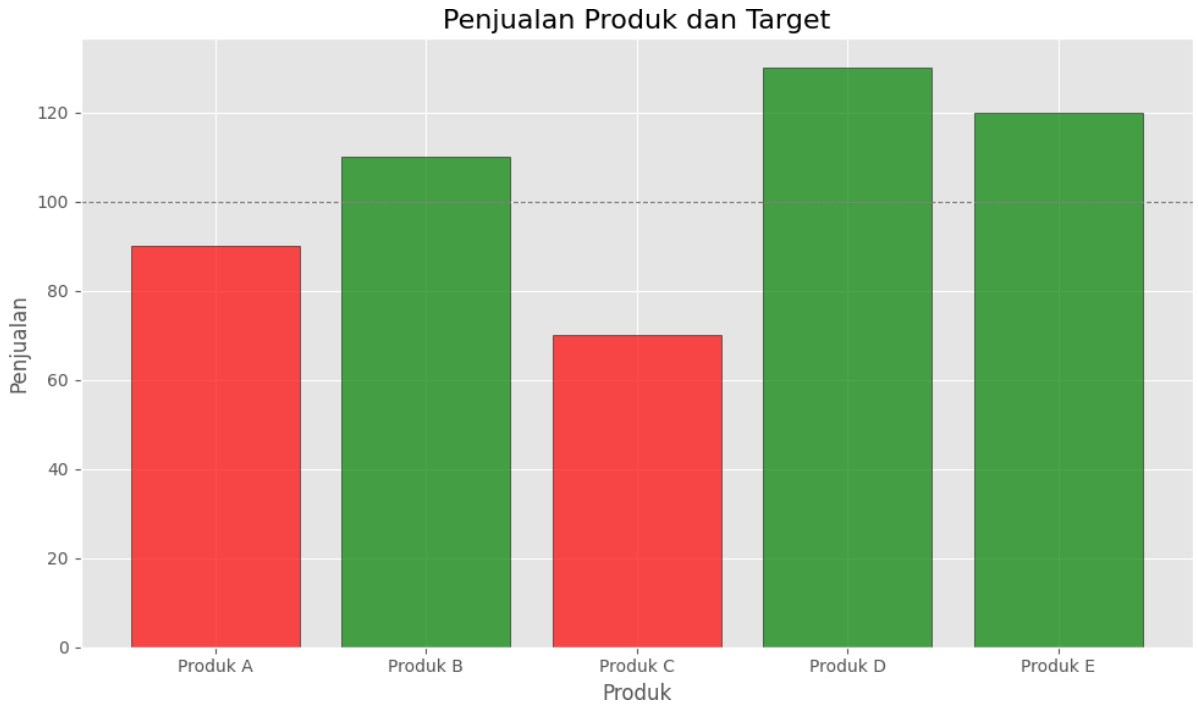
### 3.2.10. Mengatur Warna Berdasarkan Kondisi Tertentu

Dalam beberapa situasi, kamu mungkin ingin mewarnai data berdasarkan kondisi tertentu, bukan hanya berdasarkan nilai. Misalnya, anggap kamu memiliki dataset tentang penjualan produk dan kamu ingin menyoroti produk yang penjualannya di bawah target.

Mari kita coba dengan contoh sederhana:

```
# Membuat data sintetis
products = ['Produk A', 'Produk B', 'Produk C', 'Produk D', 'Produk E']
sales = [90, 110, 70, 130, 120]
target = 100

# Menentukan warna berdasarkan target penjualan
colors = ['red' if sale < target else 'green' for sale in sales]

plt.figure(figsize=(10, 6))
bars = plt.bar(products, sales, color=colors, edgecolor='black',
alpha=0.7)

# Menambahkan garis target
plt.axhline(y=target, color='gray', linestyle='--', linewidth=0.8)

plt.title('Penjualan Produk dan Target', fontsize=16)
plt.xlabel('Produk', fontsize=12)
plt.ylabel('Penjualan', fontsize=12)
plt.tight_layout()
plt.show()
```

Output:




Grafik di atas menampilkan penjualan lima produk. Batang yang berwarna merah menunjukkan produk yang penjualannya di bawah target, sementara yang berwarna hijau menunjukkan produk yang memenuhi atau melampaui target. Garis putus-putus abu-abu menandakan target penjualan. Dengan menggunakan warna berdasarkan kondisi seperti ini, kamu dapat dengan cepat mengidentifikasi area yang memerlukan perhatian atau tindakan.

### 3.2.11. Menggunakan Gradient Warna

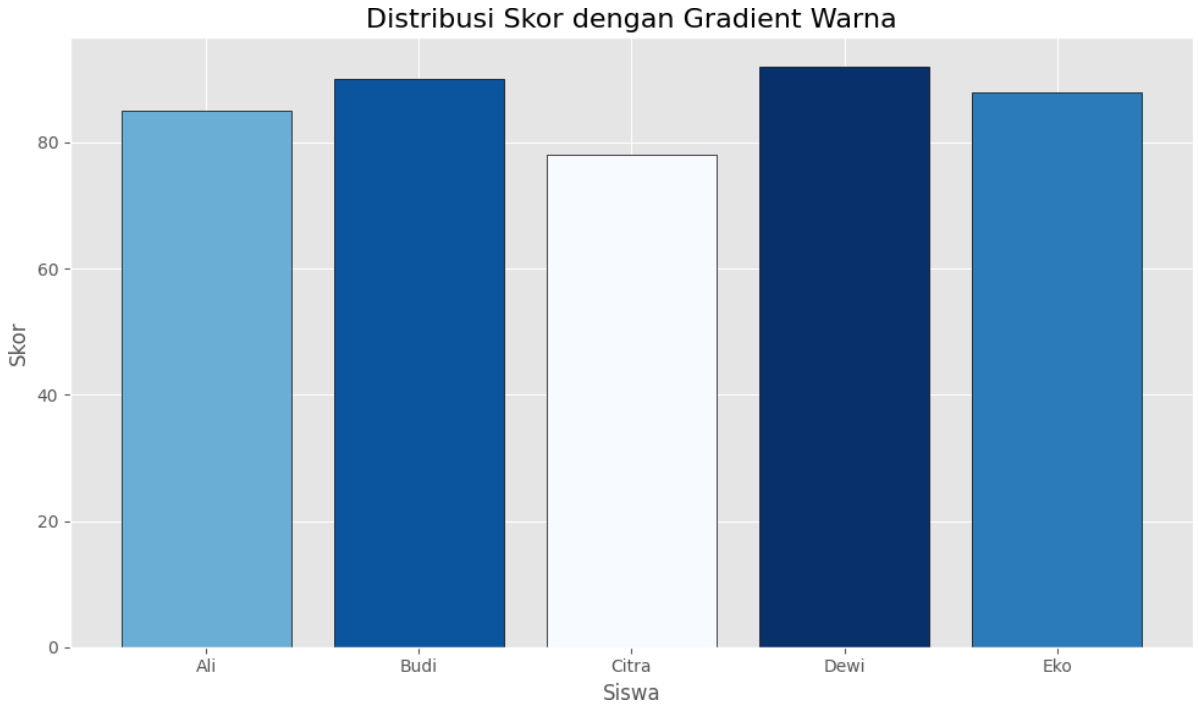
Gradient warna adalah teknik lain yang bisa kamu gunakan untuk menonjolkan perbedaan dalam data. Misalnya, kamu ingin menampilkan distribusi skor dalam kisaran tertentu, dan kamu ingin menyoroti skor yang tinggi dengan warna yang lebih gelap dan skor yang rendah dengan warna yang lebih terang.

Mari kita coba dengan contoh berikut:

```
# Membuat data sintetis
students = ['Ali', 'Budi', 'Citra', 'Dewi', 'Eko']
scores = [85, 90, 78, 92, 88]
max_score = max(scores)
min_score = min(scores)

# Menghitung gradient warna berdasarkan skor
colors = [(score - min_score) / (max_score - min_score) for score in
scores]
colors = plt.cm.Blues(colors)
```

```
plt.figure(figsize=(10, 6))
bars = plt.bar(students, scores, color=colors, edgecolor='black')

plt.title('Distribusi Skor dengan Gradient Warna', fontsize=16)
plt.xlabel('Siswa', fontsize=12)
plt.ylabel('Skor', fontsize=12)
plt.tight_layout()
plt.show()
```

Output:




Grafik di atas menampilkan distribusi skor lima siswa. Warna batang menggambarkan skor relatif dari setiap siswa; semakin gelap warnanya, semakin tinggi skornya. Teknik gradient warna ini memungkinkan kamu untuk menyoroti perbedaan dalam data dengan cara yang lebih halus.

### 3.2.12. Menggunakan Palet Warna Khusus

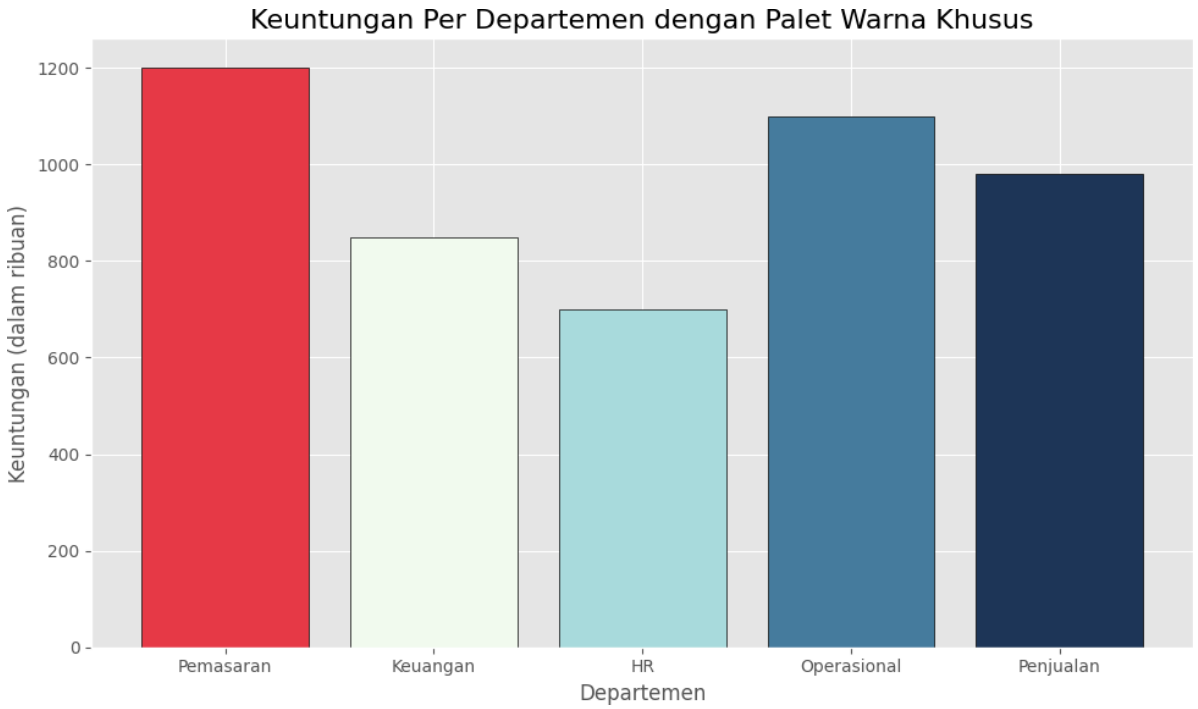
Kadang-kadang, kamu mungkin ingin menggunakan palet warna khusus yang sesuai dengan tema atau branding tertentu. Matplotlib menyediakan banyak palet warna yang bisa kamu gunakan, dan kamu juga bisa mendefinisikan palet warnamu sendiri.

Mari kita coba dengan contoh berikut:

```
# Membuat data sintetis
departments = ['Pemasaran', 'Keuangan', 'HR', 'Operasional',
'Penjualan']
```

```
profits = [1200, 850, 700, 1100, 980]

# Mendefinisikan palet warna khusus
custom_palette = ['#E63946', '#F1FAEE', '#A8DADC', '#457B9D', '#1D3557']

plt.figure(figsize=(10, 6))
bars = plt.bar(departments, profits, color=custom_palette,
edgecolor='black')

plt.title('Keuntungan Per Departemen dengan Palet Warna Khusus',
fontsize=16)
plt.xlabel('Departemen', fontsize=12)
plt.ylabel('Keuntungan (dalam ribuan)', fontsize=12)
plt.tight_layout()
plt.show()
```

Output:




Dalam grafik di atas, kita telah menerapkan palet warna khusus untuk menampilkan keuntungan per departemen. Palet warna yang telah didefinisikan menggabungkan berbagai warna yang serasi dan memberikan tampilan yang khas pada grafik. Dengan menggunakan palet warna yang kohesif dan menarik, kamu dapat membuat grafik yang tidak hanya informatif tetapi juga estetis.

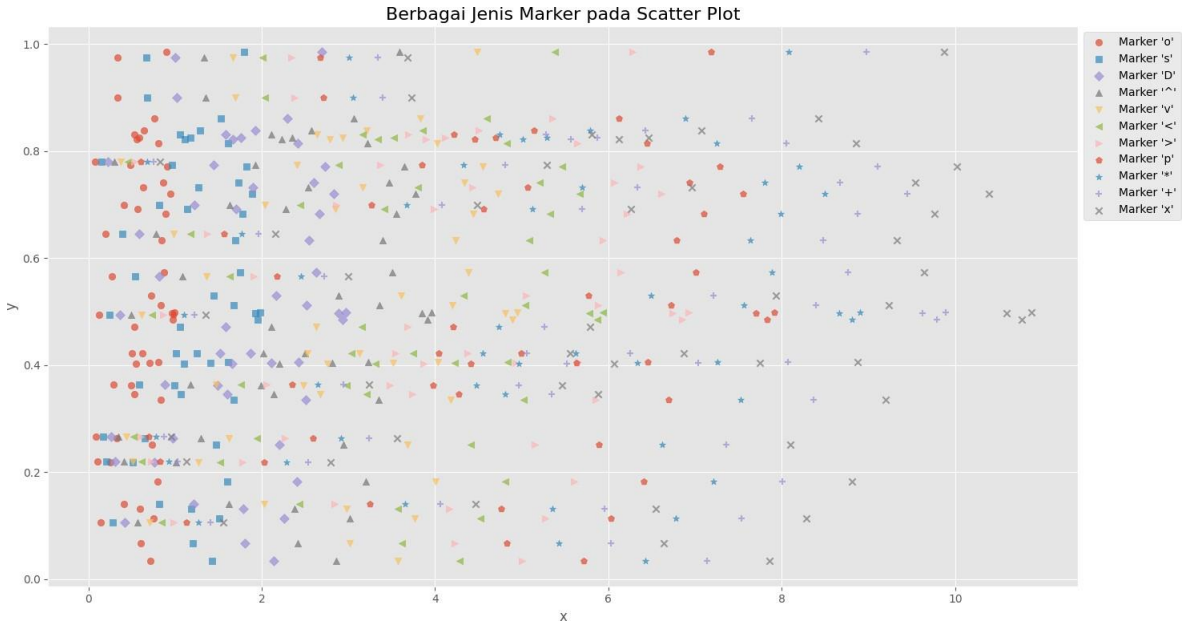
### 3.2.13. Menggunakan Marker yang Berbeda pada Scatter Plot

Marker dalam scatter plot memainkan peran penting dalam membedakan poin data. Matplotlib menawarkan berbagai pilihan marker yang dapat digunakan untuk menyesuaikan tampilan poin dalam scatter plot.

Mari kita lihat beberapa contoh:

```
# Membuat data sintetis
x = np.random.rand(50)
y = np.random.rand(50)

# Daftar marker yang bisa digunakan
markers = ['o', 's', 'D', '^', 'v', '<', '>', 'p', '*', '+', 'x']

plt.figure(figsize=(15, 8))
for idx, marker in enumerate(markers, 1):
    plt.scatter(x * idx, y, marker=marker, label=f"Marker '{marker}'",
alpha=0.7)

plt.title('Berbagai Jenis Marker pada Scatter Plot', fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```

Output:



Dalam scatter plot di atas, kita menunjukkan berbagai jenis marker yang dapat digunakan dalam Matplotlib. Dengan memilih marker yang tepat, kamu dapat menambahkan dimensi tambahan ke visualisasi atau membuatnya lebih mudah dibaca dan dipahami.

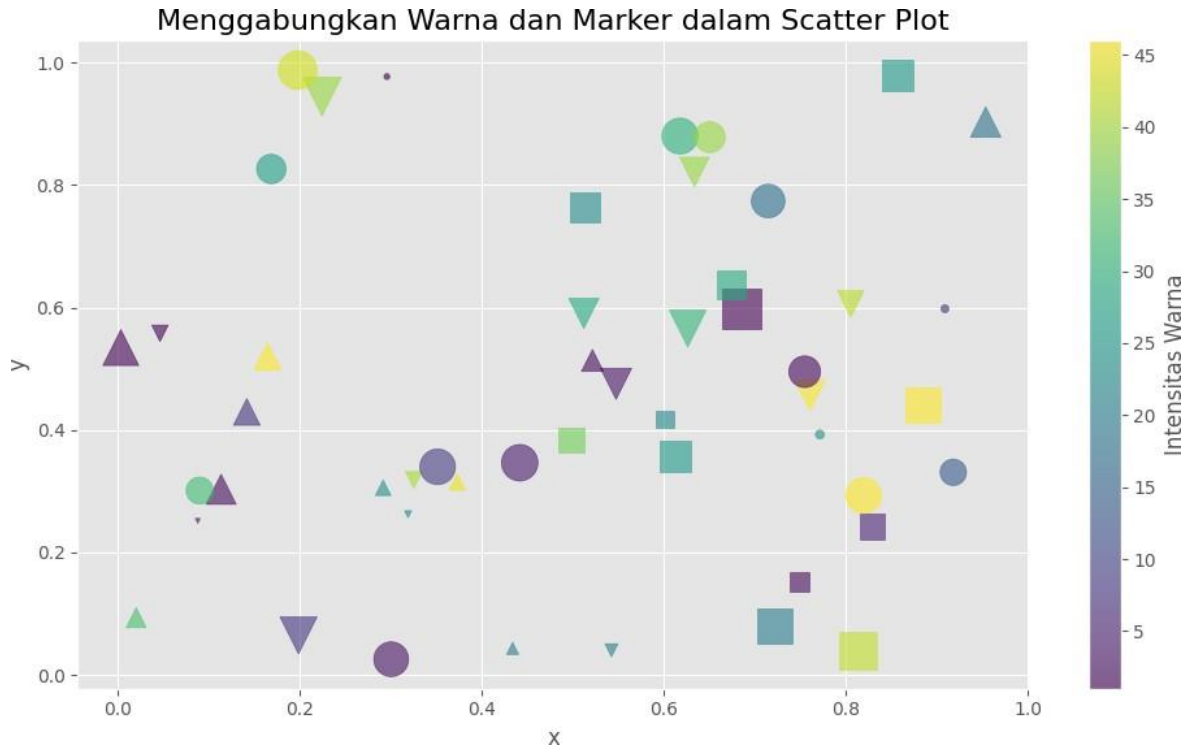
### 3.2.14. Menggabungkan Warna dan Marker untuk Menampilkan Data Multidimensi

Ketika kamu memiliki data dengan beberapa dimensi atau variabel, kombinasi warna dan marker dapat membantu untuk menampilkan informasi tersebut dalam satu visualisasi. Dengan demikian, kamu bisa mendapatkan wawasan lebih cepat tentang pola atau hubungan dalam data.

Mari kita lihat contoh sederhananya:

```
# Membuat data sintetis
np.random.seed(10)
x = np.random.rand(50)
y = np.random.rand(50)
colors = np.random.randint(0, 50, 50)
sizes = 500 * np.random.rand(50)
markers = np.random.choice(['o', 's', '^', 'v'], 50)

plt.figure(figsize=(10, 6))
for marker in set(markers):
    mask = markers == marker
    plt.scatter(x[mask], y[mask], c=colors[mask], s=sizes[mask],
marker=marker, cmap='viridis', alpha=0.6)

plt.colorbar(label='Intensitas Warna')
plt.title('Menggabungkan Warna dan Marker dalam Scatter Plot',
fontsize=16)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.tight_layout()
plt.show()
```

Output:




Dalam scatter plot di atas, kita telah menggabungkan beberapa aspek visualisasi:

- Posisi (x, y): Menggambarkan dua dimensi data.
- Warna: Intensitas warna poin menggambarkan dimensi ketiga dari data. Dalam kasus ini, kami menggunakan palet warna viridis, dan colorbar di sisi kanan mengindikasikan intensitas warna.
- Ukuran: Ukuran dari setiap titik menggambarkan dimensi keempat dari data.
- Bentuk Marker: Menggunakan bentuk marker yang berbeda sebagai dimensi kelima dari data.
-

Dengan menggabungkan warna, ukuran, dan bentuk marker, kita dapat menampilkan hingga lima dimensi data dalam satu visualisasi. Namun, penting untuk memastikan bahwa visualisasi tetap mudah dibaca dan tidak terlalu rumit, yang dapat membingungkan pembaca.

## 3.3. Menyimpan Grafik

Dalam proses analisis data dan visualisasi, seringkali kita perlu menyimpan grafik yang telah dibuat. Alasan menyimpan grafik bisa beragam, mulai dari kebutuhan dokumentasi, presentasi, publikasi, hingga pembuatan laporan. Oleh karena itu, menguasai cara menyimpan grafik dalam berbagai format dan kualitas adalah salah satu keterampilan yang penting.

Kamu pasti penasaran, bagaimana cara menyimpan grafik dengan mudah dan efisien? Bagaimana cara menyesuaikan resolusi, ukuran, dan formatnya? Mari kita jelajahi bersama dalam subbab ini.

### 3.3.1. Menyimpan Grafik dalam Format Gambar

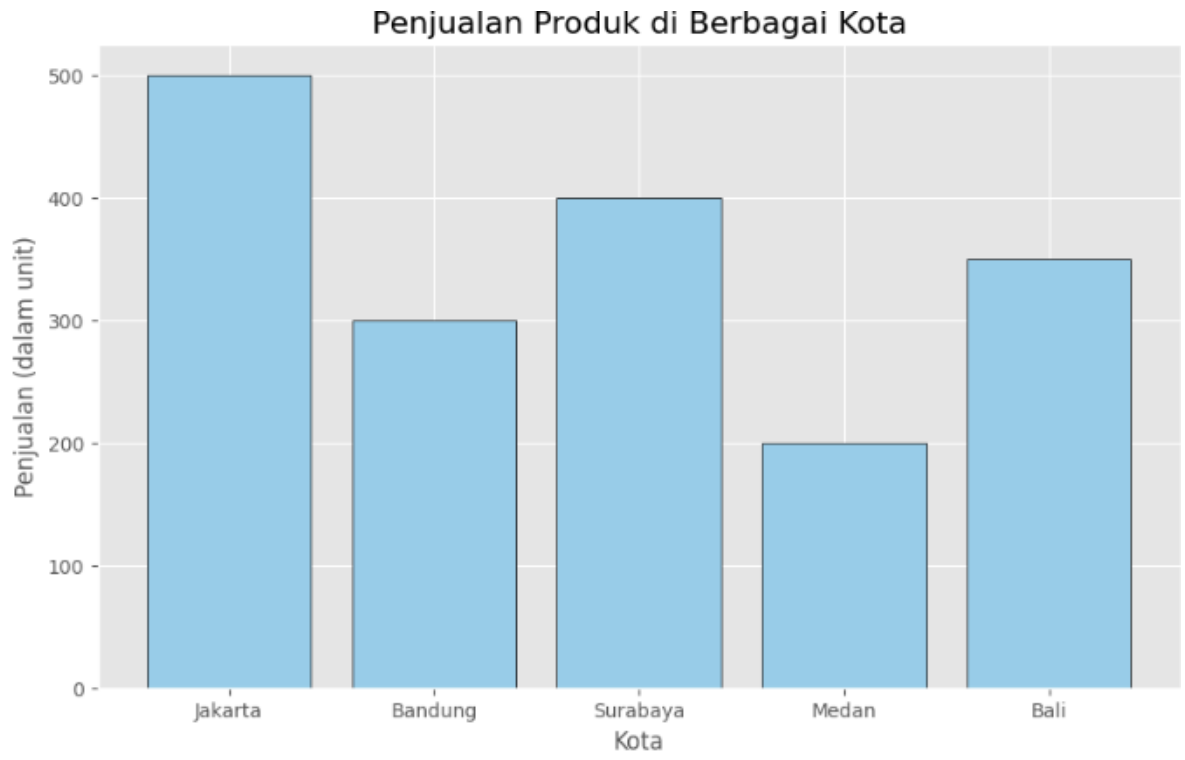
Matplotlib menyediakan fungsi savefig yang memungkinkan kamu untuk menyimpan grafik dalam berbagai format gambar, seperti PNG, JPEG, SVG, PDF, dan lainnya.

**Contoh 1: Menyimpan Grafik dalam Format PNG**
Mari kita mulai dengan contoh sederhana: menyimpan grafik batang dalam format PNG.

Kita akan membuat grafik batang yang menampilkan penjualan produk di berbagai kota, lalu menyimpannya dalam format PNG.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Data sintetis berbentuk pandas dataframe
sales_df = pd.DataFrame({
    'Kota': ['Jakarta', 'Bandung', 'Surabaya', 'Medan', 'Bali'],
    'Penjualan': [500, 300, 400, 200, 350]
})

# Membuat grafik batang
plt.figure(figsize=(10, 6))
plt.bar(sales_df['Kota'], sales_df['Penjualan'], color='skyblue',
edgecolor='black')
plt.title('Penjualan Produk di Berbagai Kota', fontsize=16)
plt.xlabel('Kota', fontsize=12)
plt.ylabel('Penjualan (dalam unit)', fontsize=12)

# Menyimpan grafik dalam format PNG
png_path = 'sales_plot.png'
plt.savefig(png_path, format='png', dpi=300)
plt.show()

# Path ke file yang telah disimpan
png_path
```

Output:




```
'sales_plot.png'
```

Grafik di atas menampilkan penjualan produk di berbagai kota, dan kita telah berhasil menyimpannya dalam format PNG.

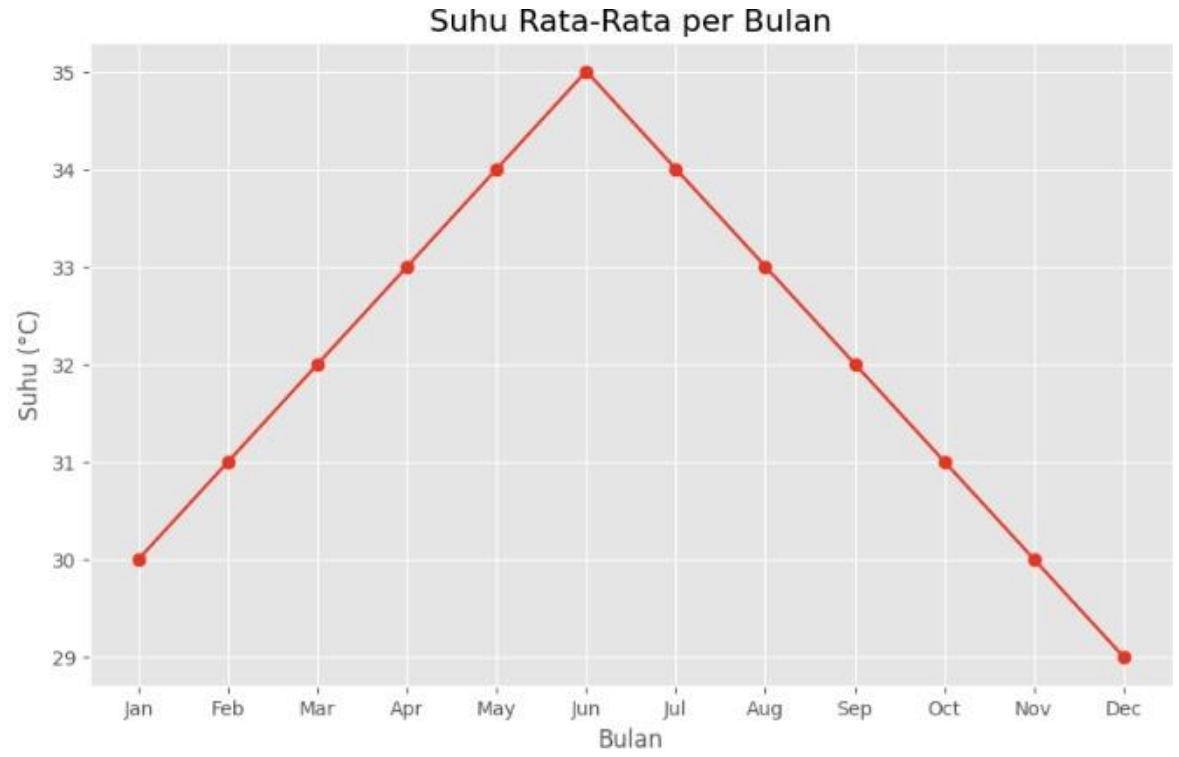
**Contoh 2: Menyimpan Grafik dalam Format PDF**

Misalnya, kamu ingin menyimpan grafik dalam format PDF untuk keperluan publikasi atau presentasi. Berikut adalah contohnya:

```
# Data sintetis lainnya
temperature_df = pd.DataFrame({
    'Bulan': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
'Sep', 'Oct', 'Nov', 'Dec'],
    'Suhu': [30, 31, 32, 33, 34, 35, 34, 33, 32, 31, 30, 29]
})

# Membuat grafik garis
plt.figure(figsize=(10, 6))
plt.plot(temperature_df['Bulan'], temperature_df['Suhu'], marker='o',
linestyle='-', color='red')
plt.title('Suhu Rata-Rata per Bulan', fontsize=16)
plt.xlabel('Bulan', fontsize=12)
plt.ylabel('Suhu (°C)', fontsize=12)

# Menyimpan grafik dalam format PDF
pdf_path = 'temperature_plot.pdf'
plt.savefig(pdf_path, format='pdf')
```

```
plt.show()

# Path ke file yang telah disimpan
pdf_path
```

Output:




```
'temperature_plot.pdf'
```

Grafik garis di atas menampilkan suhu rata-rata per bulan, dan kita telah berhasil menyimpannya dalam format PDF.

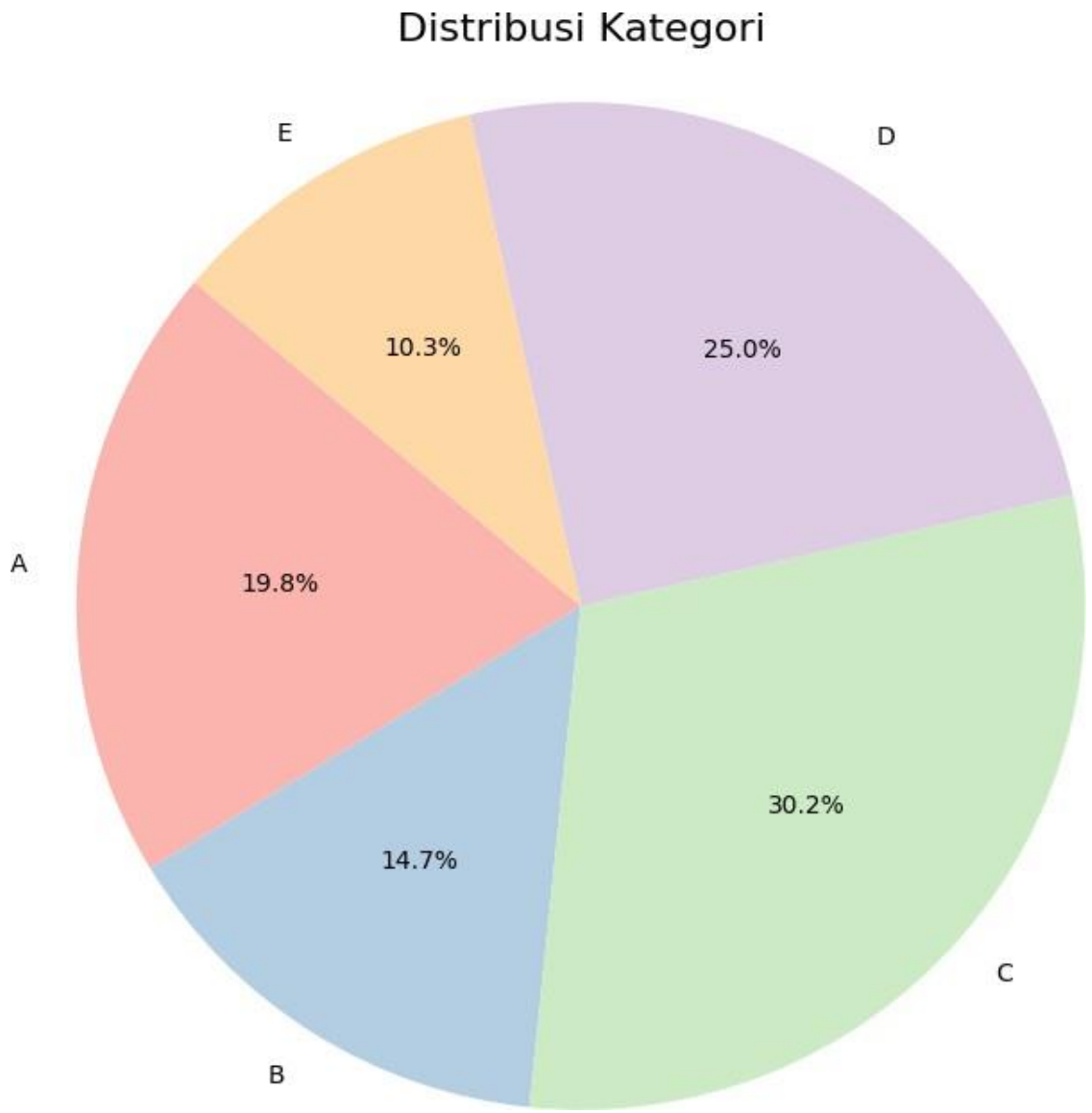
**Contoh 3: Mengatur Resolusi dan Ukuran Gambar**

Terkadang, kamu mungkin ingin mengatur resolusi dan ukuran gambar saat menyimpan. Misalnya, kamu ingin menyimpan gambar dengan resolusi tinggi untuk keperluan cetak. Berikut adalah contohnya:

```
import seaborn as sns

# Membuat grafik pie
plt.figure(figsize=(8, 8))
plt.pie(values, labels=categories, autopct='%1.1f%%', startangle=140,
colors=sns.color_palette('Pastel1'))
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.title('Distribusi Kategori', fontsize=16)
```

```
# Menyimpan grafik dalam format JPEG dengan resolusi tinggi
jpeg_path = 'pie_chart.jpeg'
plt.savefig(jpeg_path, format='jpeg', dpi=600)  # Set resolusi tinggi
plt.show()

# Path ke file yang telah disimpan
jpeg_path
```

Output:




Grafik pie di atas menampilkan distribusi kategori, dan kita telah berhasil menyimpannya dalam format JPEG dengan resolusi tinggi.

### 3.3.2. Menyimpan Grafik dalam Format Interaktif

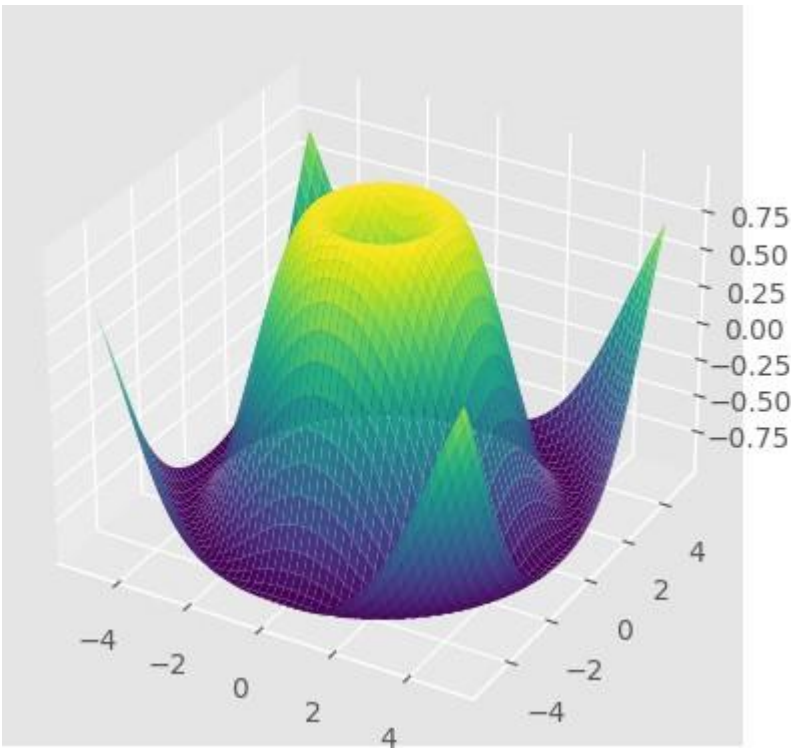
Selain menyimpan dalam format gambar, kamu juga dapat menyimpan grafik dalam format interaktif yang memungkinkan pengguna untuk berinteraksi dengan grafik, seperti memperbesar, memperkecil, atau memutar grafik 3D.

**Contoh 4: Menyimpan Grafik 3D dalam Format HTML**

Kamu dapat menggunakan pustaka mpld3 untuk menyimpan grafik 3D dalam format HTML. Berikut adalah contohnya:

```
!pip install mpld3
import mpld3
from mpl_toolkits.mplot3d import Axes3D

# Data sintetis
X = np.linspace(-5, 5, 100)
Y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Membuat grafik 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')

# Menyimpan dalam format HTML
mpld3.save_html(fig, "3d_plot.html")
```

Output:




Dengan menyimpan grafik dalam format interaktif, kamu dapat membagikan grafik dengan orang lain tanpa kehilangan kemampuan interaksi yang ada pada grafik tersebut.

### 3.3.3. Tips dan Trik Menyimpan Grafik

Ukuran Gambar: Gunakan argumen figsize saat membuat figure untuk mengatur ukuran gambar.
Resolusi: Gunakan argumen dpi saat menyimpan gambar untuk mengatur resolusi gambar.
Format: Pastikan kamu menentukan format yang sesuai saat menyimpan, misalnya 'png', 'jpeg', 'pdf', dll.

## 3.4. Anotasi dalam Grafik

Anotasi dalam grafik adalah elemen yang sangat penting dalam visualisasi data. Melalui anotasi, kita dapat menambahkan teks, panah, atau objek lain untuk memberikan penjelasan tambahan, menekankan poin tertentu, atau memberikan konteks tambahan tentang apa yang sedang ditampilkan dalam grafik. Dalam subbab ini, kita akan membahas berbagai cara untuk menambahkan anotasi dalam grafik menggunakan Matplotlib, termasuk beberapa contoh kasus yang akan membuat kamu paham betul tentang cara kerjanya.

### 3.4.1. Menambahkan Teks dalam Grafik

Menambahkan teks dalam grafik adalah salah satu cara paling sederhana untuk memberikan anotasi. Berikut adalah beberapa cara yang bisa kamu coba.

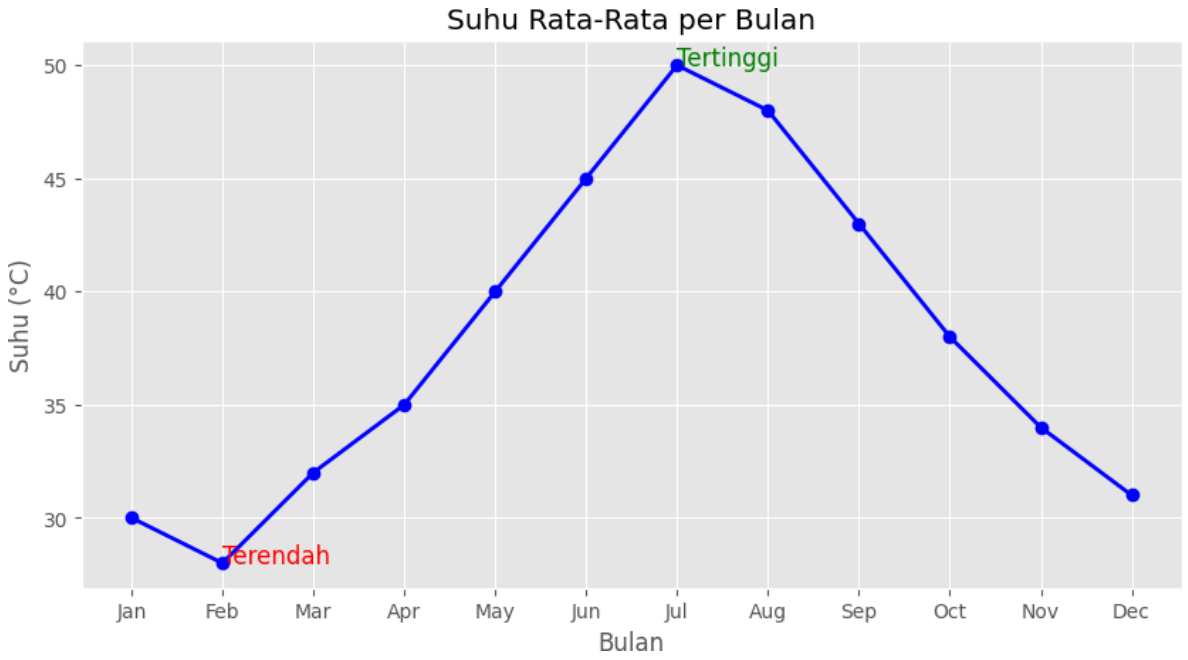
**Contoh 1: Menambahkan Teks Sederhana**
Kita akan mulai dengan contoh sederhana: menambahkan teks dalam grafik garis. Mari kita lihat bagaimana caranya.

Kita akan menggunakan data suhu rata-rata per bulan, sama seperti yang telah kita gunakan sebelumnya.

**Membuat Grafik**

```
import matplotlib.pyplot as plt

# Data sintetis
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
'Oct', 'Nov', 'Dec']
avg_temp = [30, 28, 32, 35, 40, 45, 50, 48, 43, 38, 34, 31]

# Membuat grafik garis
plt.figure(figsize=(10, 5))
plt.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
plt.title('Suhu Rata-Rata per Bulan')
plt.xlabel('Bulan')
plt.ylabel('Suhu (°C)')

# Menambahkan teks anotasi
plt.text(1, 28, 'Terendah', fontsize=12, color='red')
plt.text(6, 50, 'Tertinggi', fontsize=12, color='green')

plt.show()
```

Output:




Dalam grafik di atas, kita telah menambahkan teks "Terendah" pada bulan Februari, yang menandakan suhu terendah, dan "Tertinggi" pada bulan Juli, yang menandakan suhu tertinggi. Ini adalah contoh sederhana dari bagaimana kita bisa menambahkan teks ke dalam grafik untuk memberikan informasi tambahan.

Kita menggunakan metode plt.text(x, y, 'text') untuk menambahkan teks ke dalam grafik, di mana x dan y adalah koordinat posisi teks.

**Contoh 2: Menambahkan Anotasi dengan Panah**

Selain teks, kamu juga bisa menambahkan panah dalam anotasi untuk menunjuk ke titik tertentu dalam grafik. Ini bisa sangat berguna jika kamu ingin menekankan titik atau area tertentu dalam data.

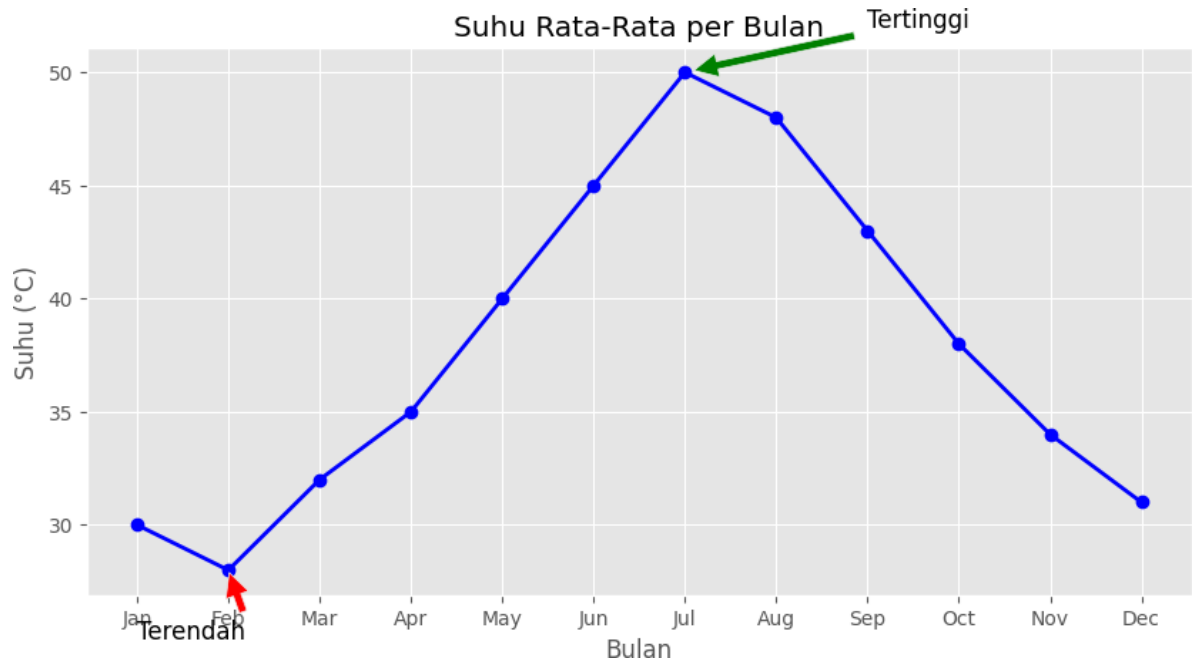
Mari kita lihat contoh berikut untuk memahami cara kerjanya.
Kita akan menggunakan data yang sama dengan contoh sebelumnya.

**Membuat Grafik dengan Anotasi Panah**

```
# Membuat grafik garis
plt.figure(figsize=(10, 5))
plt.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
plt.title('Suhu Rata-Rata per Bulan')
plt.xlabel('Bulan')
plt.ylabel('Suhu (°C)')

# Menambahkan anotasi dengan panah
```

```
plt.annotate('Terendah', xy=(1, 28), xytext=(0, 25),
             arrowprops=dict(facecolor='red', shrink=0.05), fontsize=12)
plt.annotate('Tertinggi', xy=(6, 50), xytext=(8, 52),
             arrowprops=dict(facecolor='green', shrink=0.05),
fontsize=12)

plt.show()
```

Output:




Dalam contoh di atas, kita telah menggunakan metode plt.annotate() untuk menambahkan anotasi dengan panah. Metode ini memungkinkan kita untuk menunjukkan titik atau area tertentu dalam grafik dengan menggunakan panah.

Sintaks plt.annotate('text', xy=(x, y), xytext=(x_text, y_text), arrowprops=dict(facecolor='color', shrink=0.05)) digunakan untuk menambahkan anotasi. Parameter xy menunjukkan koordinat yang ditunjuk oleh panah, sedangkan xytext menunjukkan posisi teks anotasi. Kamu juga bisa mengatur properti panah seperti warna dan ukuran dengan parameter arrowprops.

Dengan menambahkan anotasi, grafik menjadi lebih informatif dan mudah dimengerti. Ini adalah alat yang sangat berguna, terutama jika kamu ingin menekankan aspek tertentu dari data.

### 3.4.2. Anotasi Kustom

Terkadang, kamu mungkin ingin menambahkan anotasi yang lebih kompleks atau kustom. Misalnya, kamu mungkin ingin menambahkan bentuk geometris, seperti lingkaran atau persegi, atau kamu mungkin ingin menggabungkan beberapa elemen anotasi bersama-sama.

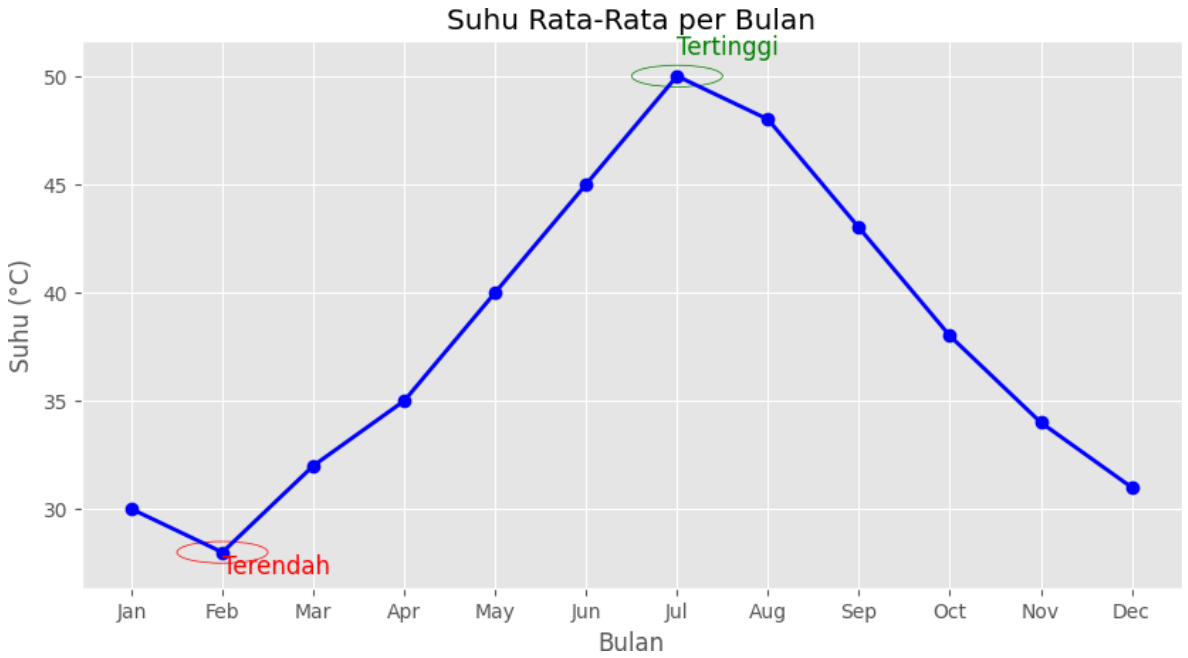
**Contoh 3: Menambahkan Lingkaran pada Titik Tertentu**
Dalam contoh ini, kita akan menambahkan lingkaran di sekitar titik tertinggi dan terendah dalam grafik.

Kita akan menggunakan data yang sama dengan contoh sebelumnya.

**Membuat Grafik dengan Lingkaran**

```
from matplotlib.patches import Circle
# Membuat grafik garis
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
ax.set_title('Suhu Rata-Rata per Bulan')
ax.set_xlabel('Bulan')
ax.set_ylabel('Suhu (°C)')

# Menambahkan lingkaran pada titik tertentu
circle_low = Circle((1, 28), radius=0.5, color='red', fill=False)
circle_high = Circle((6, 50), radius=0.5, color='green', fill=False)
ax.add_patch(circle_low)
ax.add_patch(circle_high)

# Menambahkan teks
ax.text(1, 27, 'Terendah', fontsize=12, color='red')
ax.text(6, 51, 'Tertinggi', fontsize=12, color='green')

plt.show()
```

Output:




Dalam contoh di atas, kita telah menambahkan dua lingkaran di sekitar titik suhu terendah dan tertinggi dalam grafik. Kita menggunakan kelas Circle dari Matplotlib untuk membuat lingkaran dan menambahkannya ke dalam grafik menggunakan metode ax.add_patch().

Metode ini memungkinkan kita untuk menambahkan bentuk geometris kustom ke dalam grafik, yang dapat menjadi alat yang sangat berguna untuk menekankan atau menyoroti area tertentu dalam data.

Anotasi seperti ini bisa sangat efektif dalam menyampaikan informasi tambahan tentang grafik dan memandu pemirsa untuk memahami apa yang ingin kamu sampaikan.

**Contoh 4: Anotasi Kombinasi**

Terkadang, kamu mungkin ingin mengkombinasikan beberapa elemen anotasi bersama-sama untuk menciptakan efek yang lebih kompleks. Mari kita lihat contoh dimana kita mengkombinasikan teks, panah, dan bentuk geometris.

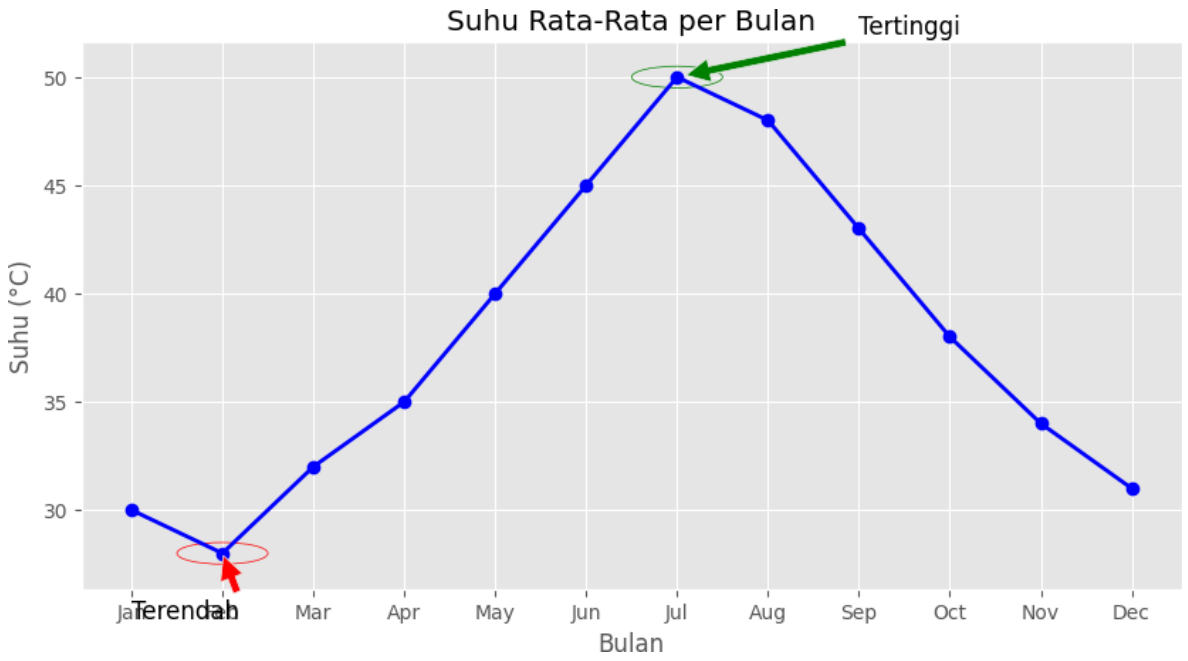
Kita akan menggunakan data yang sama dengan contoh sebelumnya.

**Membuat Grafik dengan Anotasi Kombinasi**

```
# Membuat grafik garis
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
ax.set_title('Suhu Rata-Rata per Bulan')
ax.set_xlabel('Bulan')
ax.set_ylabel('Suhu (°C)')
```

```
# Menambahkan lingkaran
circle_low = Circle((1, 28), radius=0.5, color='red', fill=False)
circle_high = Circle((6, 50), radius=0.5, color='green', fill=False)
ax.add_patch(circle_low)
ax.add_patch(circle_high)

# Menambahkan anotasi dengan panah
ax.annotate('Terendah', xy=(1, 28), xytext=(0, 25),
            arrowprops=dict(facecolor='red', shrink=0.05), fontsize=12)
ax.annotate('Tertinggi', xy=(6, 50), xytext=(8, 52),
            arrowprops=dict(facecolor='green', shrink=0.05),
fontsize=12)

plt.show()
```

Output:




Dalam contoh ini, kita telah mengkombinasikan anotasi teks, panah, dan bentuk geometris (lingkaran) untuk menekankan titik suhu terendah dan tertinggi dalam grafik. Ini adalah contoh bagaimana kamu dapat menggabungkan berbagai elemen anotasi untuk menciptakan visualisasi yang informatif dan menarik.

### 3.4.5. Menyoroti Area dengan Anotasi

Selain menyoroti titik atau menambahkan teks, kamu juga bisa menyoroti area tertentu dalam grafik dengan anotasi. Ini bisa sangat berguna jika kamu ingin menunjukkan rentang nilai atau periode waktu tertentu yang penting.

Contoh 5: Menyoroti Rentang Bulan dengan Anotasi

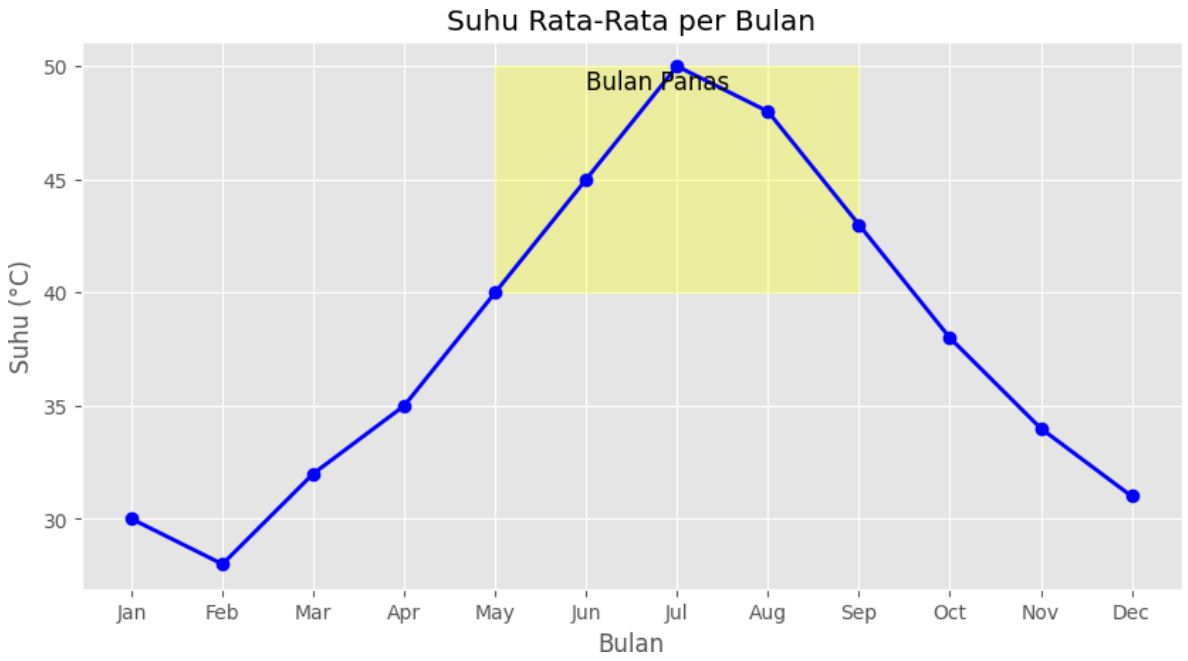
Kita akan menyoroti bulan-bulan dimana suhu rata-rata di atas 40°C.

Kita akan menggunakan data yang sama dengan contoh sebelumnya.

**Membuat Grafik dengan Area yang Disorot**

```
from matplotlib.patches import Rectangle

# Membuat grafik garis
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
ax.set_title('Suhu Rata-Rata per Bulan')
ax.set_xlabel('Bulan')
ax.set_ylabel('Suhu (°C)')

# Menyoroti area tertentu
highlight = Rectangle((4, 40), 4, 10, color='yellow', alpha=0.3)
ax.add_patch(highlight)

# Menambahkan teks
ax.text(5, 49, 'Bulan Panas', fontsize=12, color='black')

plt.show()
```

Output:




Pada contoh di atas, kita telah menyoroti area (atau rentang bulan) dimana suhu rata-rata berada di atas 40°C. Kita menggunakan kelas Rectangle dari Matplotlib untuk membuat area

yang disorot dan menambahkannya ke dalam grafik menggunakan metode ax.add_patch(). Parameter alpha digunakan untuk membuat area tersebut sedikit transparan sehingga kita masih dapat melihat grafik garis di bawahnya.

Menyoroti area tertentu dalam grafik adalah cara yang efektif untuk menunjukkan periode atau rentang nilai yang penting, dan memberikan konteks tambahan untuk membantu pemirsa memahami grafik dengan lebih baik.

### 3.4.6. Menggunakan Anotasi untuk Menunjukkan Tren

Selain menekankan titik atau area tertentu, anotasi juga dapat digunakan untuk menunjukkan tren dalam data. Dengan menambahkan garis atau panah, kamu dapat mengarahkan perhatian pemirsa ke arah tertentu atau menunjukkan perubahan seiring waktu.

**Contoh 6: Menunjukkan Tren Kenaikan Suhu**

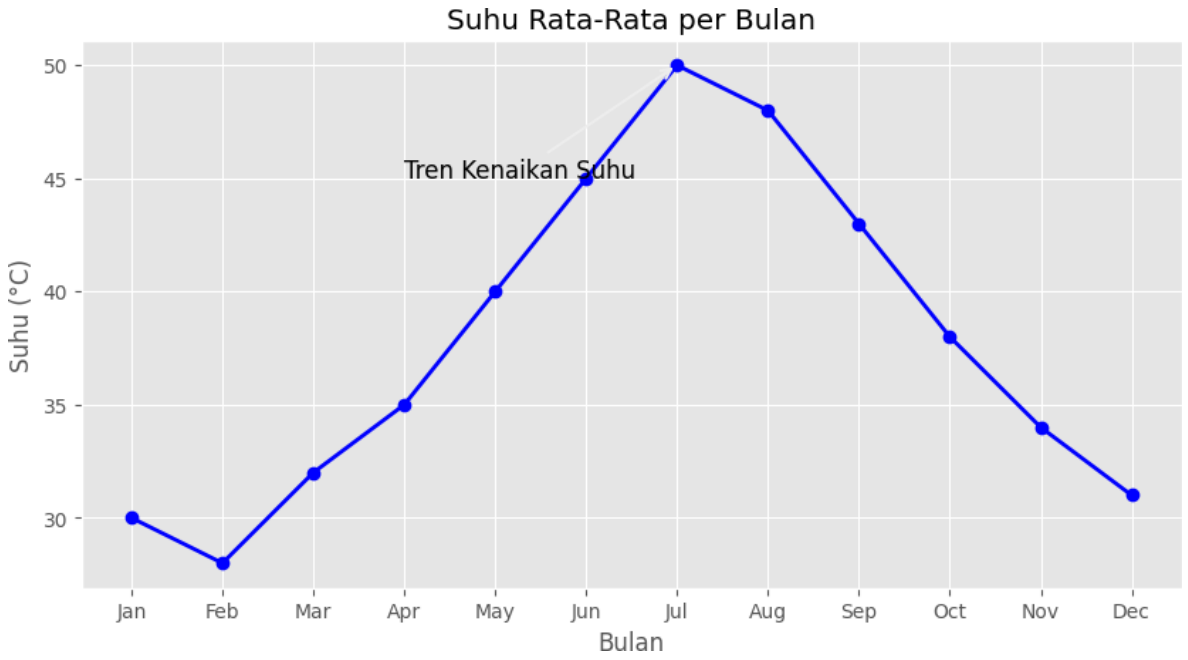
Dalam contoh ini, kita akan menunjukkan tren kenaikan suhu dari Januari hingga Juli.

Kita akan menggunakan data yang sama dengan contoh sebelumnya.

**Membuat Grafik dengan Tren yang Ditunjukkan**

```
# Membuat grafik garis
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
ax.set_title('Suhu Rata-Rata per Bulan')
ax.set_xlabel('Bulan')
ax.set_ylabel('Suhu (°C)')

# Menambahkan anotasi dengan panah untuk menunjukkan tren
ax.annotate('Tren Kenaikan Suhu', xy=(6, 50), xytext=(3, 45),
            arrowprops=dict(facecolor='orange', arrowstyle='->',
lw=1.5),
            fontsize=12, color='black')

plt.show()
```

Output:




Dalam visualisasi di atas, kita menggunakan panah untuk menunjukkan tren kenaikan suhu dari Januari hingga Juli. Menggunakan panah seperti ini adalah cara yang efektif untuk mengarahkan perhatian pemirsa ke arah tertentu dalam grafik dan menekankan tren atau perubahan seiring waktu.

### 3.4.7. Mengkombinasikan Beberapa Anotasi

Kamu tidak terbatas pada satu jenis anotasi dalam satu grafik. Sebenarnya, mengkombinasikan beberapa anotasi bisa sangat berguna untuk menyampaikan informasi yang lebih kompleks atau memberikan konteks tambahan tentang data.

**Contoh 7: Mengkombinasikan Anotasi Teks, Area, dan Tren**

Mari kita gabungkan teknik-teknik anotasi yang telah kita pelajari untuk membuat grafik yang lebih informatif.

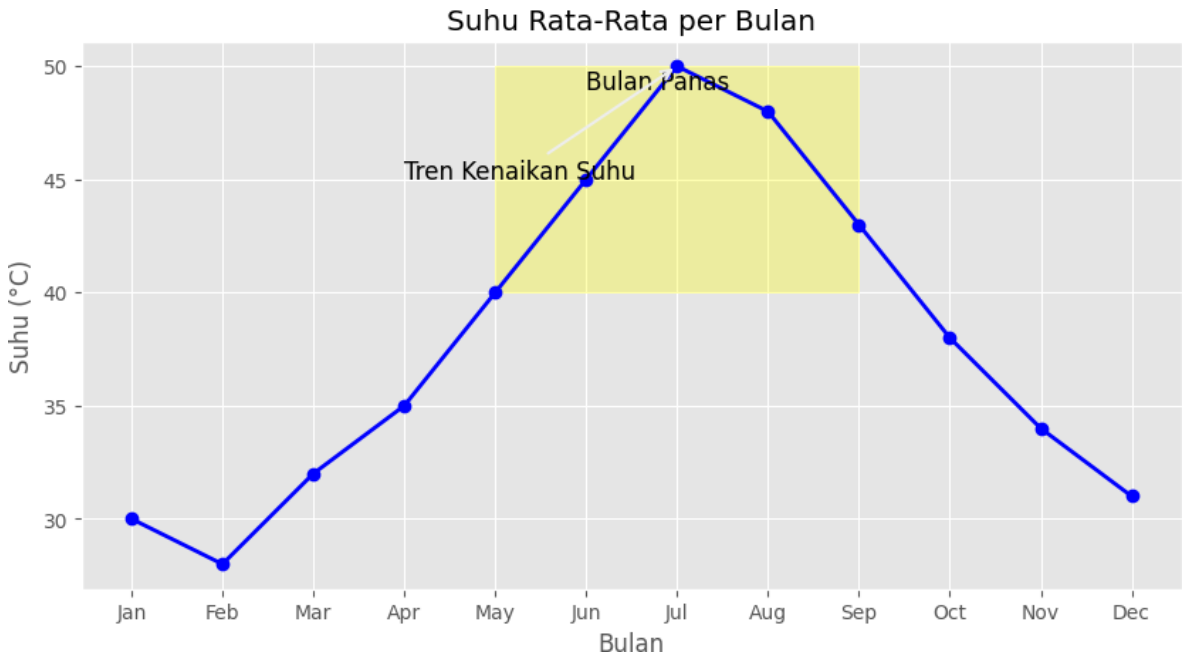
Kita akan menggunakan data yang sama dengan contoh sebelumnya.

**Membuat Grafik dengan Anotasi Gabungan**

```
# Membuat grafik garis
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
ax.set_title('Suhu Rata-Rata per Bulan')
ax.set_xlabel('Bulan')
ax.set_ylabel('Suhu (°C)')

# Menyoroti area tertentu
```

```
highlight = Rectangle((4, 40), 4, 10, color='yellow', alpha=0.3)
ax.add_patch(highlight)

# Menambahkan teks
ax.text(5, 49, 'Bulan Panas', fontsize=12, color='black')

# Menambahkan anotasi dengan panah untuk menunjukkan tren
ax.annotate('Tren Kenaikan Suhu', xy=(6, 50), xytext=(3, 45),
            arrowprops=dict(facecolor='orange', arrowstyle='->',
lw=1.5),
            fontsize=12, color='black')

plt.show()
```

Output:




Dalam grafik di atas, kita mengkombinasikan anotasi teks, panah, dan area yang disorot untuk memberikan gambaran lengkap tentang suhu rata-rata per bulan. Kita menggunakan warna kuning untuk menyoroti bulan-bulan panas dan panah oranye untuk menunjukkan tren kenaikan suhu.

Mengkombinasikan anotasi dalam satu grafik memungkinkan kamu untuk mengkomunikasikan lebih banyak informasi tanpa memerlukan ruang tambahan atau grafik tambahan. Namun, penting untuk memastikan bahwa grafik tetap mudah dibaca dan tidak terlalu penuh dengan informasi.

### 3.4.9. Anotasi Otomatis dengan Posisi Dinamis

Mengatur posisi anotasi secara manual dapat menjadi proses yang membosankan, terutama jika kamu memiliki banyak titik data. Untungnya, Matplotlib menyediakan beberapa teknik untuk membuat anotasi dengan posisi dinamis yang menyesuaikan diri dengan data.

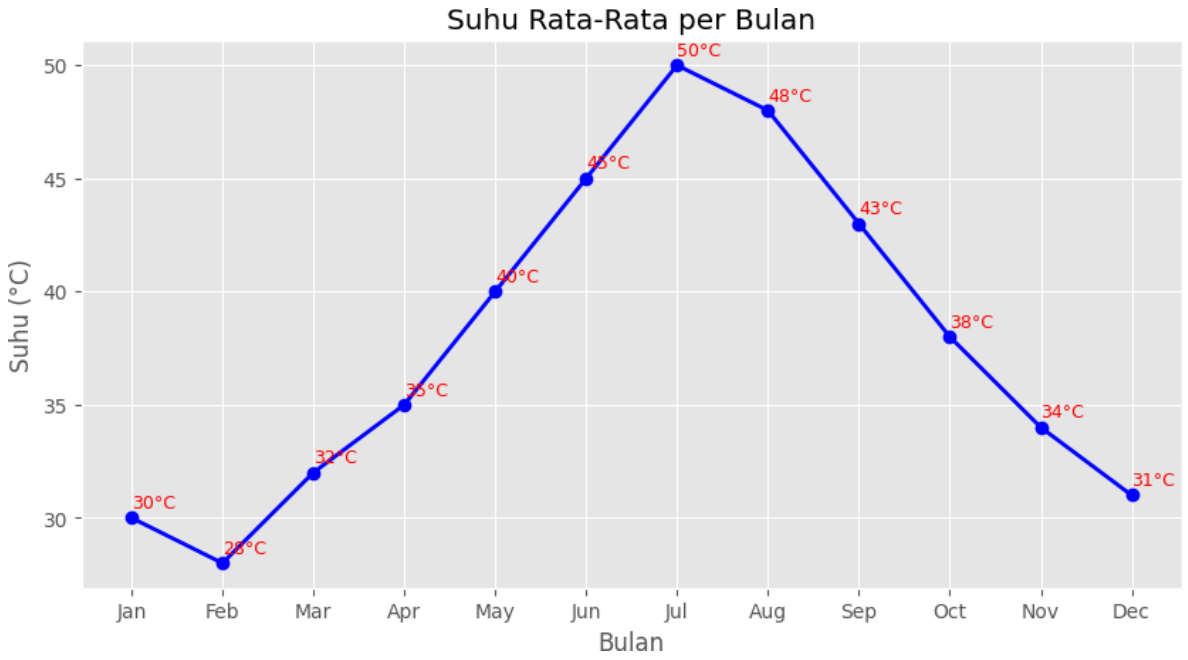
**Contoh 8: Anotasi Otomatis untuk Setiap Titik Data**
Kita akan menambahkan anotasi untuk setiap titik data dalam grafik garis, dengan posisi yang disesuaikan secara otomatis.

Kita akan menggunakan data yang sama dengan contoh sebelumnya.

**Membuat Grafik dengan Anotasi Otomatis**

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

# Membuat grafik garis
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
ax.set_title('Suhu Rata-Rata per Bulan')
ax.set_xlabel('Bulan')
ax.set_ylabel('Suhu (°C)')

# Menambahkan anotasi otomatis untuk setiap titik
for i, temp in enumerate(avg_temp):
    ax.annotate(f'{temp}°C', (months[i], temp), xytext=(0,5),
textcoords='offset points', fontsize=9, color='red')

plt.show()
```

Output:




Dalam contoh di atas, kita telah menambahkan anotasi ke setiap titik data dalam grafik garis, dengan posisi yang disesuaikan secara otomatis. Kita menggunakan parameter xytext untuk menentukan jarak anotasi dari titik data, dan textcoords='offset points' untuk menetapkan koordinat relatif.

Hasilnya adalah anotasi yang rapi dan sejajar dengan setiap titik data, tanpa perlu mengatur posisi masing-masing anotasi secara manual. Ini sangat berguna ketika kamu memiliki banyak titik data dan ingin menampilkan informasi tambahan untuk setiap titik.

**Contoh: Anotasi Otomatis untuk Nilai Maksimum dan Minimum**

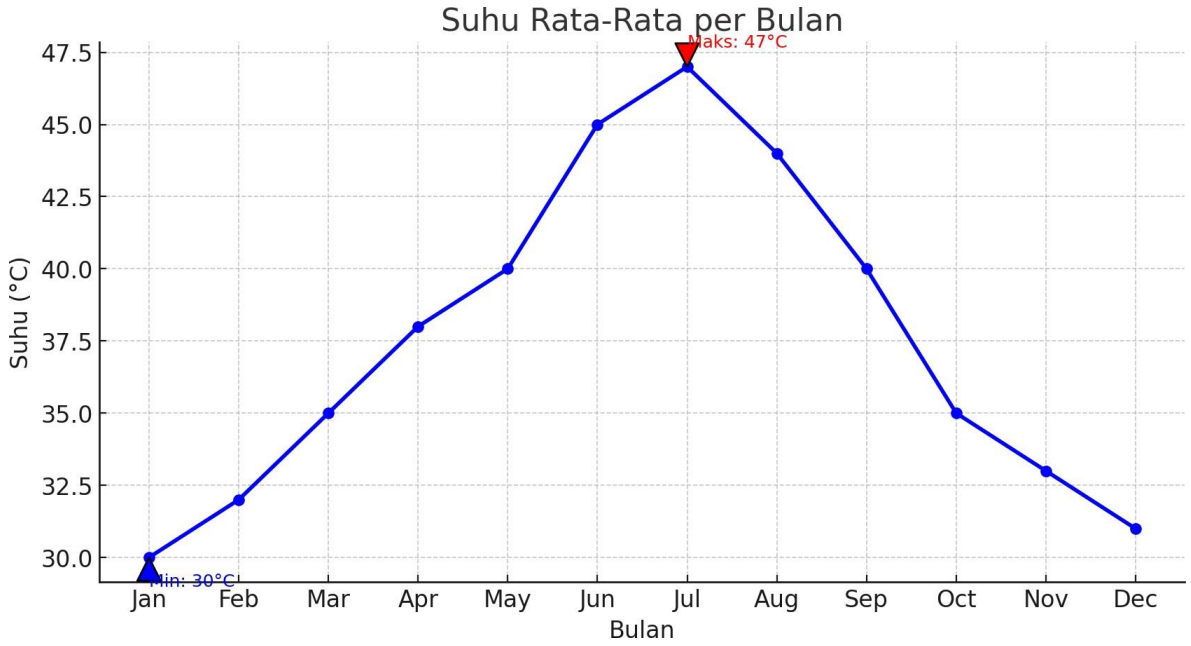
Kita akan menggunakan grafik garis yang sama dengan data suhu rata-rata per bulan. Kemudian, kita akan menambahkan anotasi otomatis untuk menyoroti nilai maksimum dan minimum dalam data.

**Membuat Grafik dengan Anotasi Otomatis untuk Nilai Maksimum dan Minimum**

```
# Menemukan indeks dan nilai maksimum dan minimum
max_index = avg_temp.index(max(avg_temp))
min_index = avg_temp.index(min(avg_temp))
max_value = max(avg_temp)
min_value = min(avg_temp)

# Membuat grafik garis
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(months, avg_temp, marker='o', color='b', linestyle='-',
linewidth=2)
ax.set_title('Suhu Rata-Rata per Bulan')
```

```
ax.set_xlabel('Bulan')
ax.set_ylabel('Suhu (°C)')

# Menambahkan anotasi otomatis untuk nilai maksimum
ax.annotate(f'Maks: {max_value}°C', (months[max_index], max_value),
            xytext=(0,10), textcoords='offset points', fontsize=9,
color='red', arrowprops=dict(facecolor='red'))

# Menambahkan anotasi otomatis untuk nilai minimum
ax.annotate(f'Min: {min_value}°C', (months[min_index], min_value),
            xytext=(0,-15), textcoords='offset points', fontsize=9,
color='blue', arrowprops=dict(facecolor='blue'))

plt.show()
```

Output:




Dalam grafik di atas, kita telah menambahkan anotasi otomatis untuk menyoroti nilai maksimum dan minimum dalam data. Anotasi ini mencakup teks yang menunjukkan nilai maksimum dan minimum, serta panah yang mengarah ke titik data yang relevan.

Cara ini berguna untuk menyoroti puncak dan lembah dalam data, yang dapat membantu pembaca untuk dengan cepat mengidentifikasi tren dan pola dalam grafik. Dengan menggunakan anotasi otomatis seperti ini, kamu dapat menambahkan informasi yang berharga tanpa perlu mengatur posisi anotasi secara manual.

Berikut adalah penjelasan dari kode di atas:

- Kita menggunakan fungsi max dan min untuk menemukan nilai maksimum dan minimum dalam data, serta index untuk menemukan indeks dari nilai-nilai ini.
- Kita menggunakan metode annotate untuk menambahkan anotasi ke grafik, dengan parameter xytext untuk menentukan jarak teks dari titik data, dan arrowprops untuk menambahkan panah.
- Kita menggunakan warna berbeda untuk anotasi maksimum dan minimum, yang membuatnya lebih mudah dibedakan.

## 3.5. Pengaturan Grid dan Sumbu

Pengaturan grid dan sumbu dalam plot adalah aspek penting dari personalisasi yang dapat meningkatkan kualitas visualisasi data. Dengan mengatur grid dan sumbu dengan benar, kamu dapat membantu pembaca lebih mudah membaca dan memahami plot. Mari kita telusuri berbagai cara untuk mengatur grid dan sumbu dalam Matplotlib.

### 3.5.1. Mengatur Grid

Grid adalah kumpulan garis horizontal dan vertikal yang membentuk pola kotak pada plot. Grid membantu dalam mengukur dan membandingkan data dalam plot.

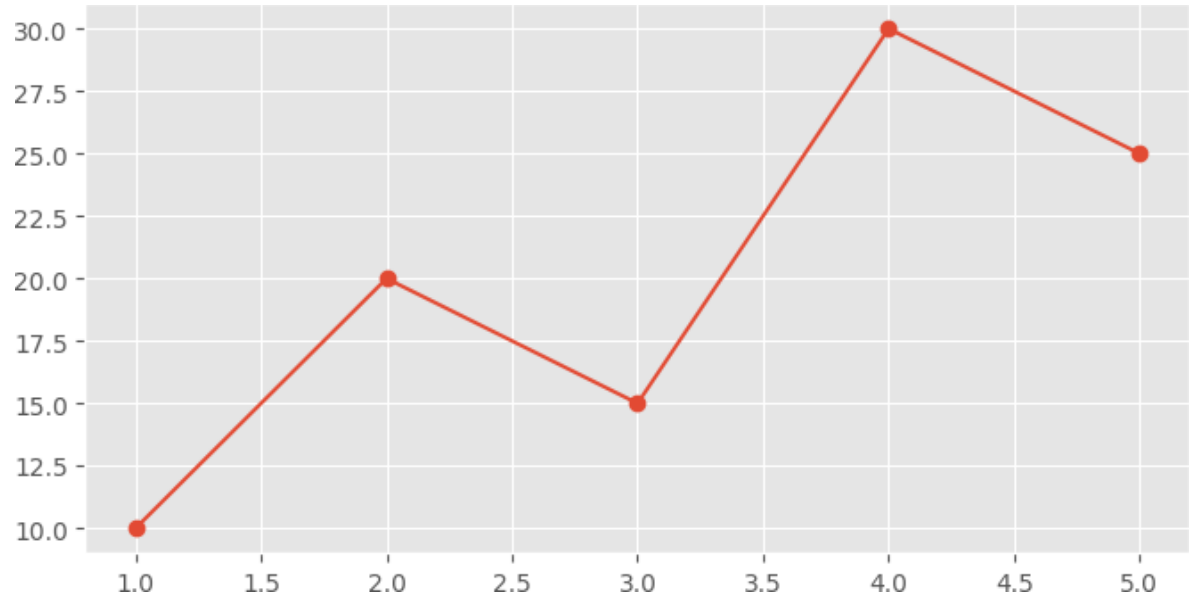
#### 3.5.1.1. Menambahkan Grid Standar

Untuk menambahkan grid standar ke plot, kamu dapat menggunakan metode grid dari objek axes. Mari kita lihat contoh bagaimana menambahkan grid ke plot garis sederhana.

**Contoh: Menambahkan Grid Standar**

```
# Data sintetis
x_values = [1, 2, 3, 4, 5]
y_values = [10, 20, 15, 30, 25]

# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_values, y_values, marker='o', linestyle='-')

# Menambahkan grid
ax.grid(True)

# Menampilkan plot
plt.show()
```

Output:




Dalam contoh di atas, kita menggunakan metode grid(True) untuk menambahkan grid standar ke plot. Hasilnya adalah grid dengan garis putus-putus yang membantu pembaca dalam mengukur dan membandingkan data.

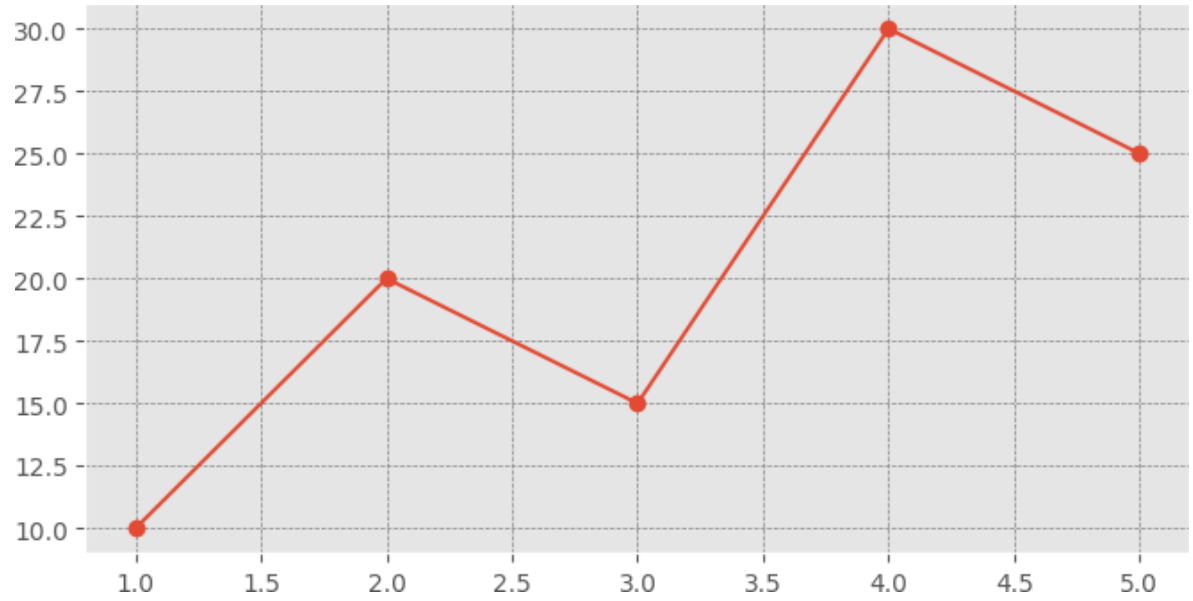
#### 3.5.1.2. Mengatur Gaya Grid

Kamu juga dapat mengatur gaya grid, seperti warna, lebar garis, dan jenis garis. Ini dapat membantu kamu menyelaraskan grid dengan tema visual plot.

**Contoh: Mengatur Gaya Grid**

```
# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_values, y_values, marker='o', linestyle='-')

# Menambahkan grid dengan gaya khusus
ax.grid(color='grey', linestyle='--', linewidth=0.5)

# Menampilkan plot
plt.show()
```

Output:




Dalam contoh di atas, kita mengatur warna grid menjadi abu-abu, jenis garis menjadi putus-putus, dan lebar garis menjadi 0.5. Dengan menggabungkan gaya ini, kita dapat membuat grid yang lebih halus yang sesuai dengan estetika plot.

### 3.5.2. Pengaturan Sumbu

Mengatur sumbu adalah bagian penting dari personalisasi plot. Ini termasuk mengubah rentang sumbu, mengatur label sumbu, dan mengontrol tampilan tick sumbu.

#### 3.5.2.1. Mengatur Rentang Sumbu

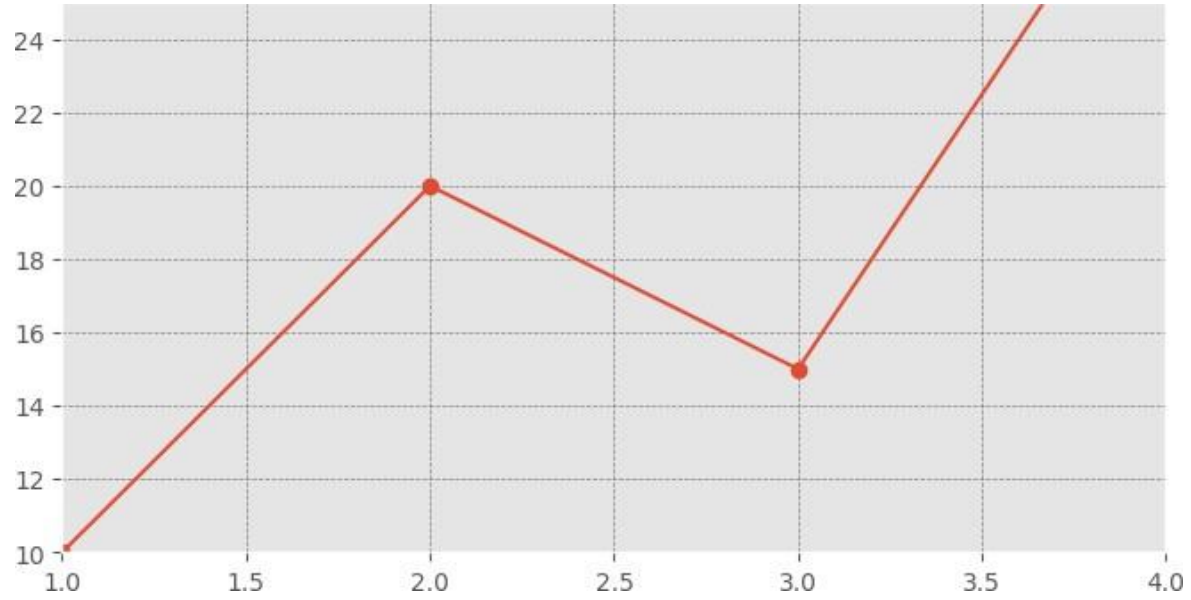
Kamu dapat mengontrol rentang sumbu x dan y dengan menggunakan metode xlim dan ylim. Ini memungkinkan kamu untuk fokus pada bagian spesifik dari data atau untuk menyelaraskan berbagai plot.

**Contoh: Mengatur Rentang Sumbu**

```
# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_values, y_values, marker='o', linestyle='-')

# Mengatur rentang sumbu x dan y
ax.set_xlim(1, 4)
ax.set_ylim(10, 25)

# Menambahkan grid
ax.grid(color='grey', linestyle='--', linewidth=0.5)

# Menampilkan plot
```

```
plt.show()
```

Output:




Dalam contoh di atas, kita menggunakan metode set_xlim dan set_ylim untuk mengatur rentang sumbu x dan y, masing-masing dari 1 hingga 4 dan dari 10 hingga 25. Hasilnya adalah plot yang difokuskan pada bagian spesifik dari data.

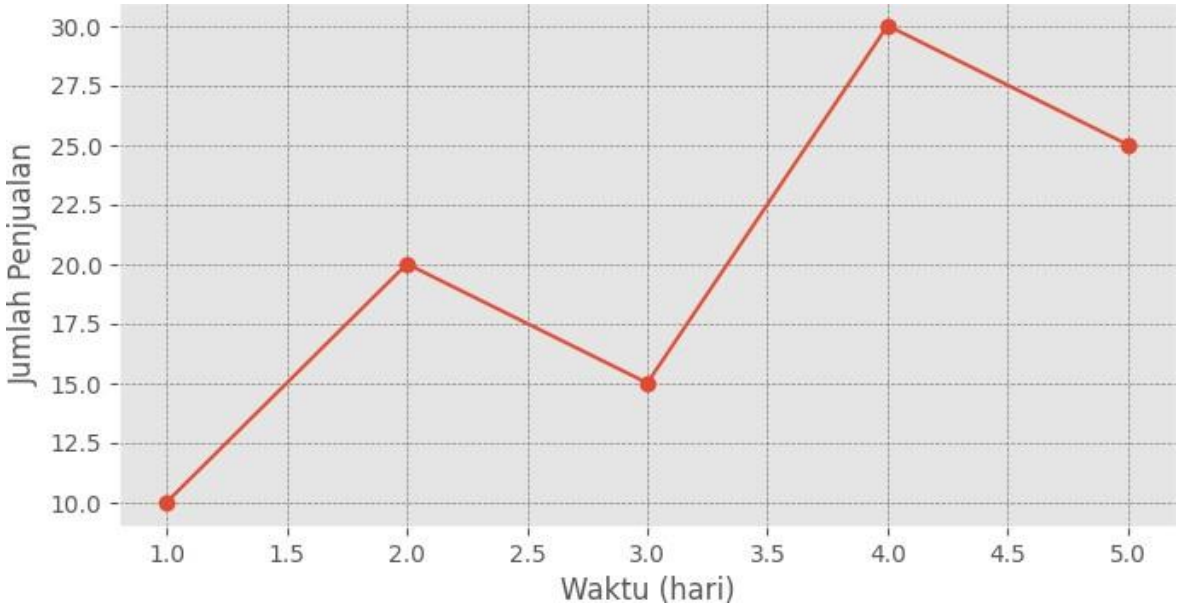
#### 3.5.2.2. Mengatur Label Sumbu

Label sumbu memberikan konteks kepada pembaca tentang apa yang diwakili oleh sumbu x dan y. Kamu dapat menambahkan label sumbu dengan menggunakan metode set_xlabel dan set_ylabel.

**Contoh: Mengatur Label Sumbu**

```
# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_values, y_values, marker='o', linestyle='-')

# Menambahkan label sumbu
ax.set_xlabel('Waktu (hari)')
ax.set_ylabel('Jumlah Penjualan')

# Menambahkan grid
ax.grid(color='grey', linestyle='--', linewidth=0.5)

# Menampilkan plot
plt.show()
```

Output:




Dalam contoh ini, kita menambahkan label sumbu dengan menggunakan metode set_xlabel dan set_ylabel. Label sumbu x adalah "Waktu (hari)" dan label sumbu y adalah "Jumlah Penjualan". Ini memberikan konteks tambahan kepada pembaca tentang apa yang diwakili oleh data dalam plot.

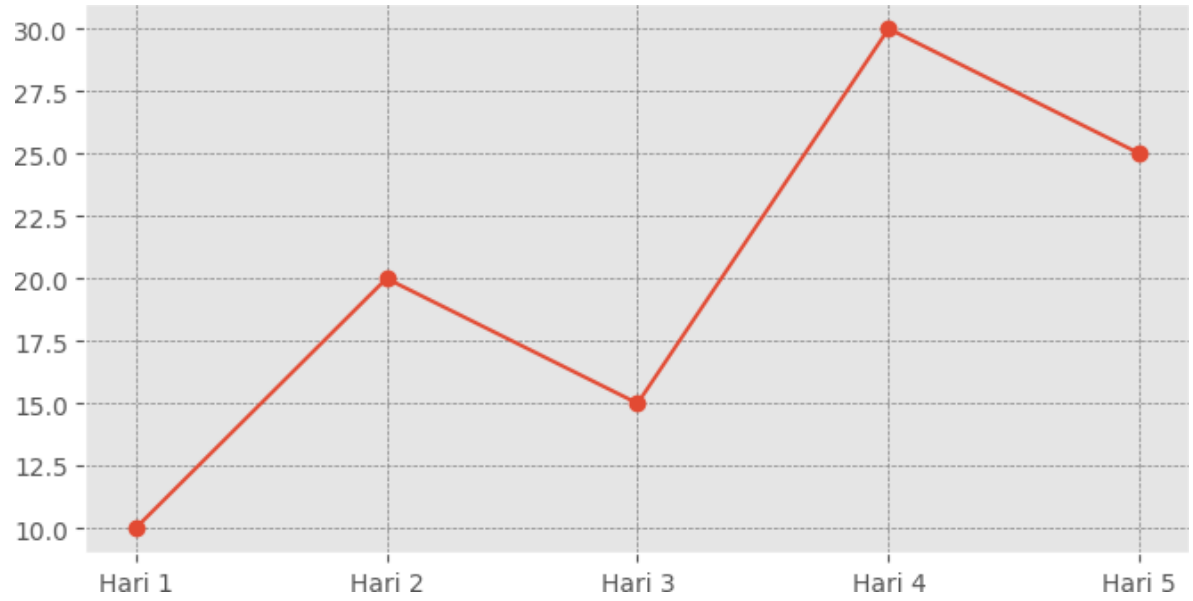
#### 3.5.2.3. Mengatur Tick Sumbu

Tick sumbu adalah penanda pada sumbu yang menunjukkan nilai tertentu. Kamu dapat mengontrol penempatan dan label tick sumbu dengan menggunakan metode xticks dan yticks.

**Contoh: Mengatur Tick Sumbu**

```
# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_values, y_values, marker='o', linestyle='-')

# Menambahkan tick sumbu x
ax.set_xticks([1, 2, 3, 4, 5])
ax.set_xticklabels(['Hari 1', 'Hari 2', 'Hari 3', 'Hari 4', 'Hari 5'])

# Menambahkan grid
ax.grid(color='grey', linestyle='--', linewidth=0.5)

# Menampilkan plot
plt.show()
```

Output:




Di sini, kita menggunakan metode set_xticks untuk menentukan posisi tick sumbu x, dan metode set_xticklabels untuk memberikan label khusus kepada tick tersebut. Dalam contoh ini, kita mengganti tick sumbu x dengan label "Hari 1", "Hari 2", dll., yang memberikan interpretasi tambahan kepada data.

### 3.5.3. Kombinasi Pengaturan Grid dan Sumbu

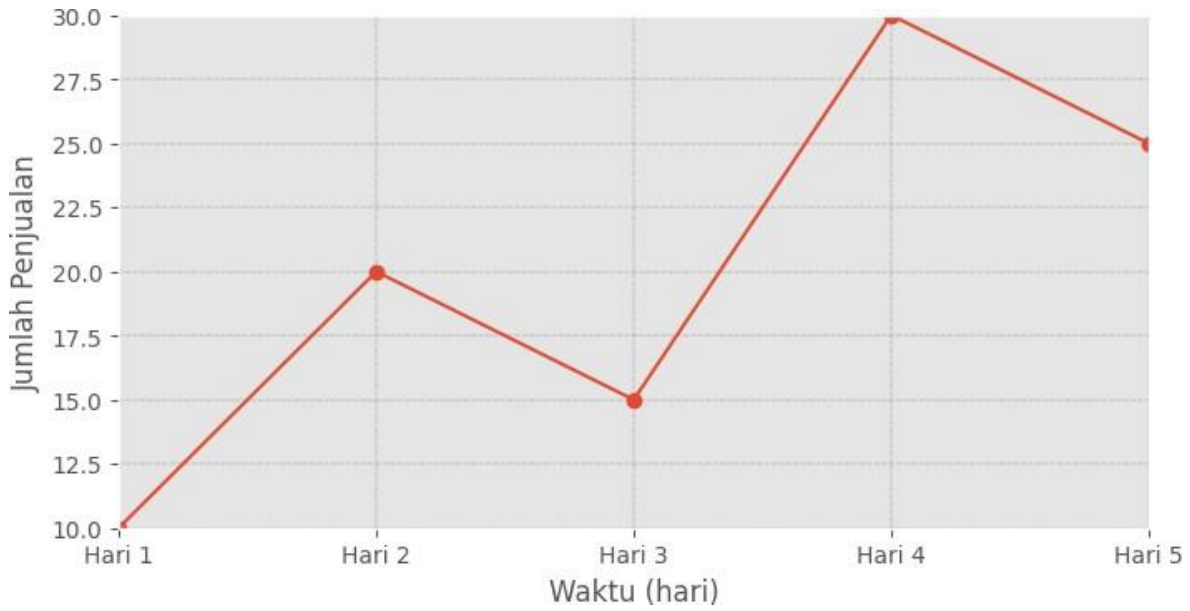
Kombinasi pengaturan grid dan sumbu dapat membantu kamu menciptakan plot yang informatif dan estetis. Kuncinya adalah menemukan keseimbangan antara informasi dan tampilan visual.

**Contoh: Kombinasi Pengaturan Grid dan Sumbu**

```
# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_values, y_values, marker='o', linestyle='-')

# Mengatur rentang sumbu
ax.set_xlim(1, 5)
ax.set_ylim(10, 30)

# Menambahkan label sumbu
ax.set_xlabel('Waktu (hari)')
ax.set_ylabel('Jumlah Penjualan')

# Menambahkan tick sumbu x
ax.set_xticks([1, 2, 3, 4, 5])
ax.set_xticklabels(['Hari 1', 'Hari 2', 'Hari 3', 'Hari 4', 'Hari 5'])
```

```
# Menambahkan grid dengan gaya khusus
ax.grid(color='grey', linestyle='-.', linewidth=0.3)

# Menampilkan plot
plt.show()
```

Output:




Contoh di atas adalah gabungan dari semua teknik yang telah kita pelajari sejauh ini. Kita telah mengatur rentang sumbu, menambahkan label sumbu, mengatur tick sumbu, dan menambahkan grid dengan gaya khusus.

### 3.5.4. Mengatur Sumbu Logaritmik

Dalam beberapa kasus, menggunakan skala logaritmik pada sumbu bisa lebih informatif, terutama jika data yang ingin ditampilkan memiliki rentang yang sangat luas. Skala logaritmik dapat dengan mudah diatur di Matplotlib.

#### 3.5.4.1. Menggunakan Skala Logaritmik pada Sumbu Y

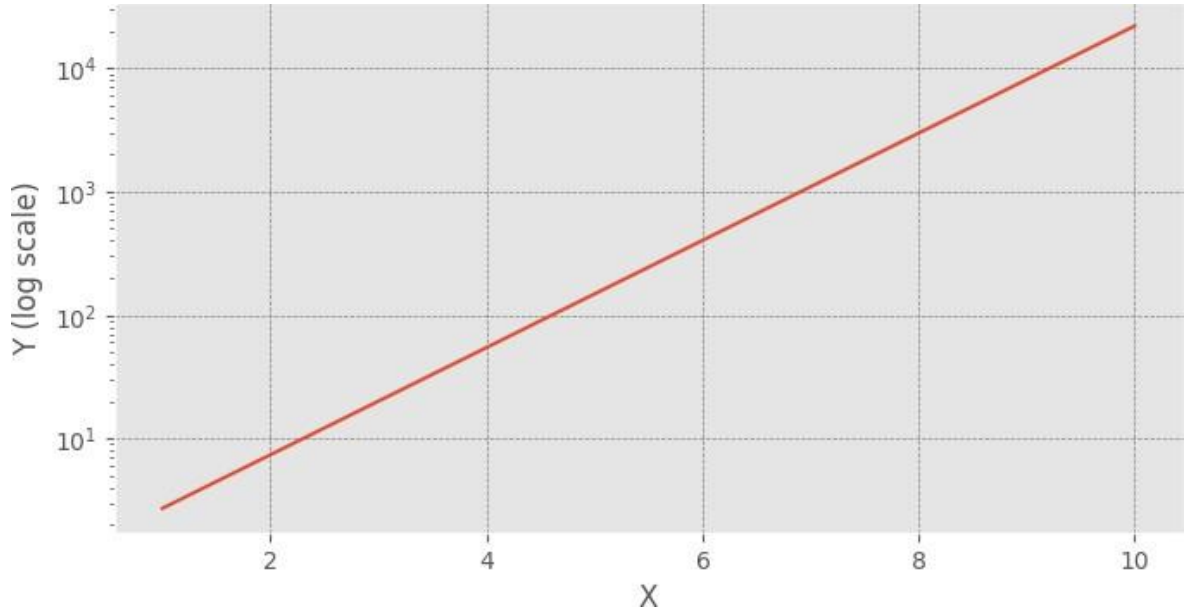
Mari kita lihat bagaimana kita bisa menggunakan skala logaritmik pada sumbu y.

Contoh: Skala Logaritmik pada Sumbu Y

```
import numpy as np

# Data sintetis dengan rentang luas
x_log = np.linspace(1, 10, 100)
y_log = np.exp(x_log)

# Membuat plot
```

```
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_log, y_log, marker='', linestyle='-')

# Mengatur skala logaritmik pada sumbu y
ax.set_yscale('log')

# Menambahkan label sumbu
ax.set_xlabel('X')
ax.set_ylabel('Y (log scale)')

# Menambahkan grid
ax.grid(color='grey', linestyle='--', linewidth=0.5)

# Menampilkan plot
plt.show()
```

Output:




Di sini, kita menggunakan skala logaritmik pada sumbu y dengan mengatur ax.set_yscale('log'). Skala logaritmik ini membantu menampilkan data yang memiliki rentang yang sangat luas dalam satu plot, sehingga mudah dipahami. Ini sangat bermanfaat dalam kasus-kasus seperti distribusi pendapatan, distribusi ukuran partikel, dan banyak lagi.

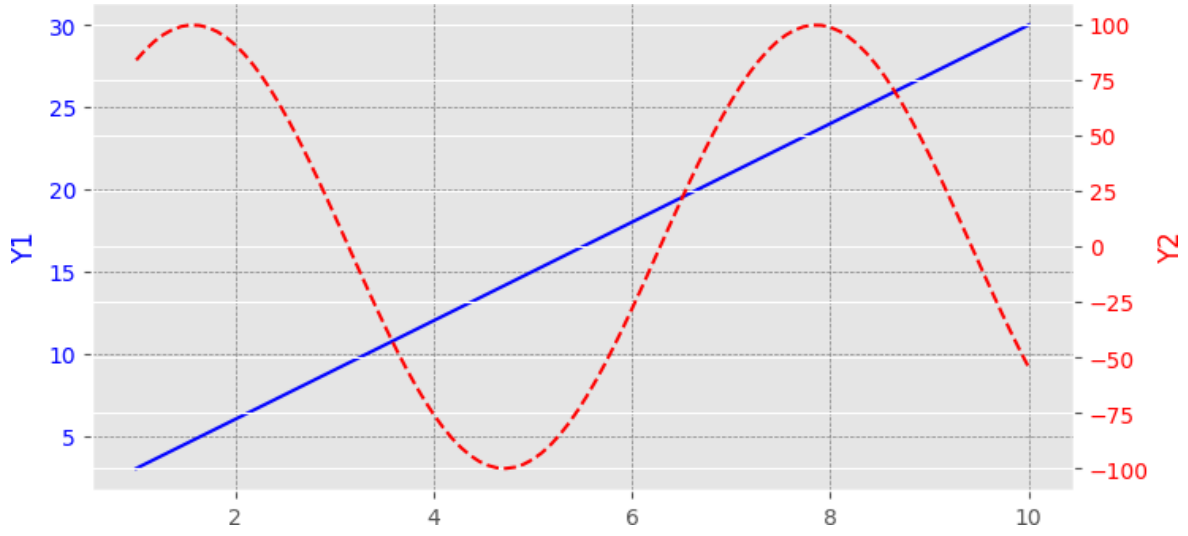
#### 3.5.4.2. Mengatur Sumbu Ganda

Dalam beberapa situasi, kamu mungkin ingin menampilkan dua set data yang berbeda dalam satu plot, tetapi dengan sumbu yang berbeda. Misalnya, jika kamu memiliki dua

variabel yang berhubungan tetapi memiliki unit yang berbeda, kamu dapat menggunakan sumbu ganda.

```
# Data sintetis
x_dual = np.linspace(1, 10, 100)
y1_dual = x_dual * 3
y2_dual = np.sin(x_dual) * 100

# Membuat plot dengan sumbu y ganda
fig, ax1 = plt.subplots(figsize=(8, 4))
ax2 = ax1.twinx()

# Plot pertama (sumbu y kiri)
ax1.plot(x_dual, y1_dual, color='blue', linestyle='-')
ax1.set_ylabel('Y1', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')
ax1.grid(color='grey', linestyle='--', linewidth=0.5)

# Plot kedua (sumbu y kanan)
ax2.plot(x_dual, y2_dual, color='red', linestyle='--')
ax2.set_ylabel('Y2', color='red')
ax2.tick_params(axis='y', labelcolor='red')

plt.show()
```

Output:




Pada contoh di atas, kita menggunakan metode twinx() untuk membuat sumbu y kedua yang berbagi sumbu x yang sama dengan sumbu y pertama. Setiap sumbu y ini dapat memiliki label, tick, dan warna yang berbeda, sehingga memungkinkan untuk menampilkan dua set data yang berbeda dalam satu plot.

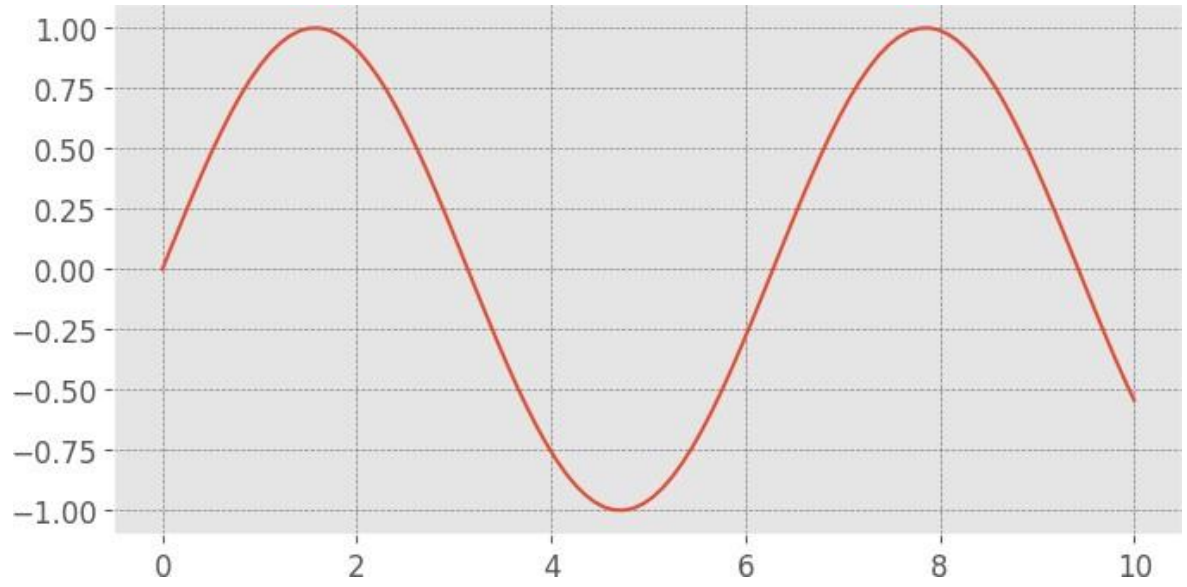
### 3.5.5. Menyesuaikan Ukuran Tick Sumbu

Kadang-kadang, kamu mungkin ingin menyesuaikan ukuran tick sumbu untuk membuat plot lebih mudah dibaca atau lebih sesuai dengan tema visual yang kamu pilih.

**Contoh: Menyesuaikan Ukuran Tick Sumbu**

```
# Data sintetis
x_tick = np.linspace(0, 10, 100)
y_tick = np.sin(x_tick)

# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_tick, y_tick, marker='', linestyle='-')

# Menyesuaikan ukuran tick sumbu
ax.tick_params(axis='both', labelsize=12)

# Menambahkan grid
ax.grid(color='grey', linestyle='--', linewidth=0.5)

# Menampilkan plot
plt.show()
```

Output:




Di contoh ini, kita menggunakan metode tick_params dengan parameter labelsize untuk mengatur ukuran tick sumbu. Ini memungkinkan kita untuk menyesuaikan tampilan plot sesuai dengan kebutuhan.

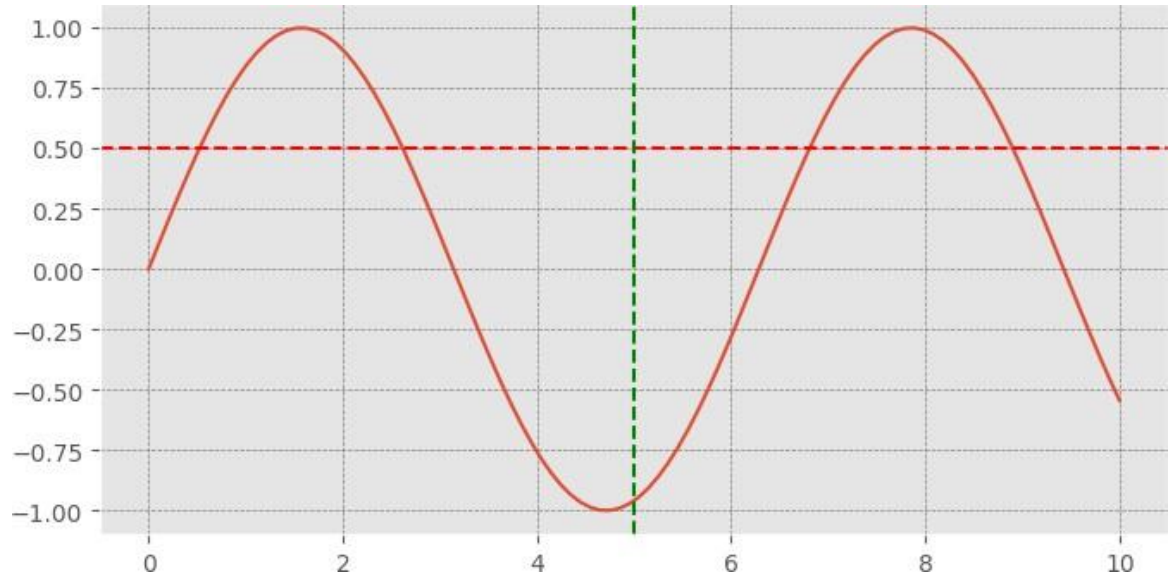
### 3.5.6. Menambahkan Garis Horizontal dan Vertikal

Ada saatnya kamu mungkin ingin menambahkan garis horizontal atau vertikal ke plot untuk menunjukkan ambang batas atau nilai referensi tertentu. Matplotlib menyediakan metode sederhana untuk melakukannya.

**Contoh: Menambahkan Garis Horizontal dan Vertikal**

```
# Data sintetis
x_line = np.linspace(0, 10, 100)
y_line = np.sin(x_line)

# Membuat plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_line, y_line, marker='', linestyle='-')

# Menambahkan garis horizontal pada y=0.5
ax.axhline(y=0.5, color='red', linestyle='--')

# Menambahkan garis vertikal pada x=5
ax.axvline(x=5, color='green', linestyle='--')

# Menambahkan grid
ax.grid(color='grey', linestyle='--', linewidth=0.5)

# Menampilkan plot
plt.show()
```

Output:



Di sini, kita menambahkan garis horizontal dan vertikal menggunakan metode axhline dan axvline. Parameter y dan x digunakan untuk menentukan posisi garis, sementara warna dan gaya garis dapat disesuaikan sesuai kebutuhan.

Garis horizontal dan vertikal ini sering digunakan untuk menunjukkan ambang batas atau nilai referensi dalam analisis ilmiah atau teknis.

# 3.6. Legenda dan Kustomisasi

### 3.6.1. Pengantar

Legenda adalah salah satu komponen terpenting dalam visualisasi data. Ini memungkinkan pembaca untuk memahami apa yang direpresentasikan oleh masing-masing komponen visual dalam plot. Seiring dengan legenda, kustomisasi tambahan pada plot dapat meningkatkan estetika dan keterbacaan data visual. Mari kita jelajahi lebih dalam tentang bagaimana kita bisa mengoptimalkan legenda dan melakukan kustomisasi tambahan pada plot kita.

### 3.6.2. Menambahkan dan Mengatur Legenda

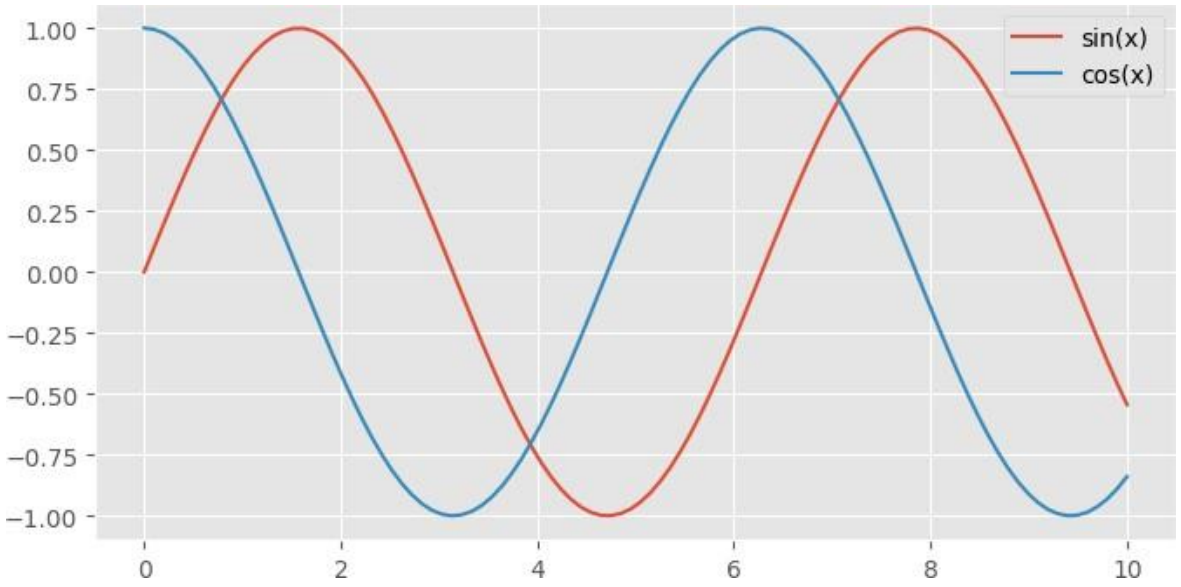
#### 3.6.2.1. Legenda Dasar

Membuat legenda dengan matplotlib sangatlah mudah. Kamu hanya perlu menambahkan label ke setiap elemen yang ingin ditampilkan di legenda dan kemudian memanggil metode legend().

**Contoh: Menambahkan Legenda Sederhana**

```
# Data sintetis
x_legend = np.linspace(0, 10, 100)
y1_legend = np.sin(x_legend)
y2_legend = np.cos(x_legend)

# Membuat plot dengan legenda
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x_legend, y1_legend, label='sin(x)')
ax.plot(x_legend, y2_legend, label='cos(x)')

# Menambahkan legenda
ax.legend()

# Menampilkan plot
plt.show()
```

Output:




Seperti yang kamu lihat di atas, kita menambahkan label ke setiap garis saat memanggil metode plot(). Kemudian, dengan memanggil metode legend(), kita bisa menampilkan legenda yang menunjukkan label untuk setiap garis.

#### 3.6.2.2. Mengatur Posisi Legenda

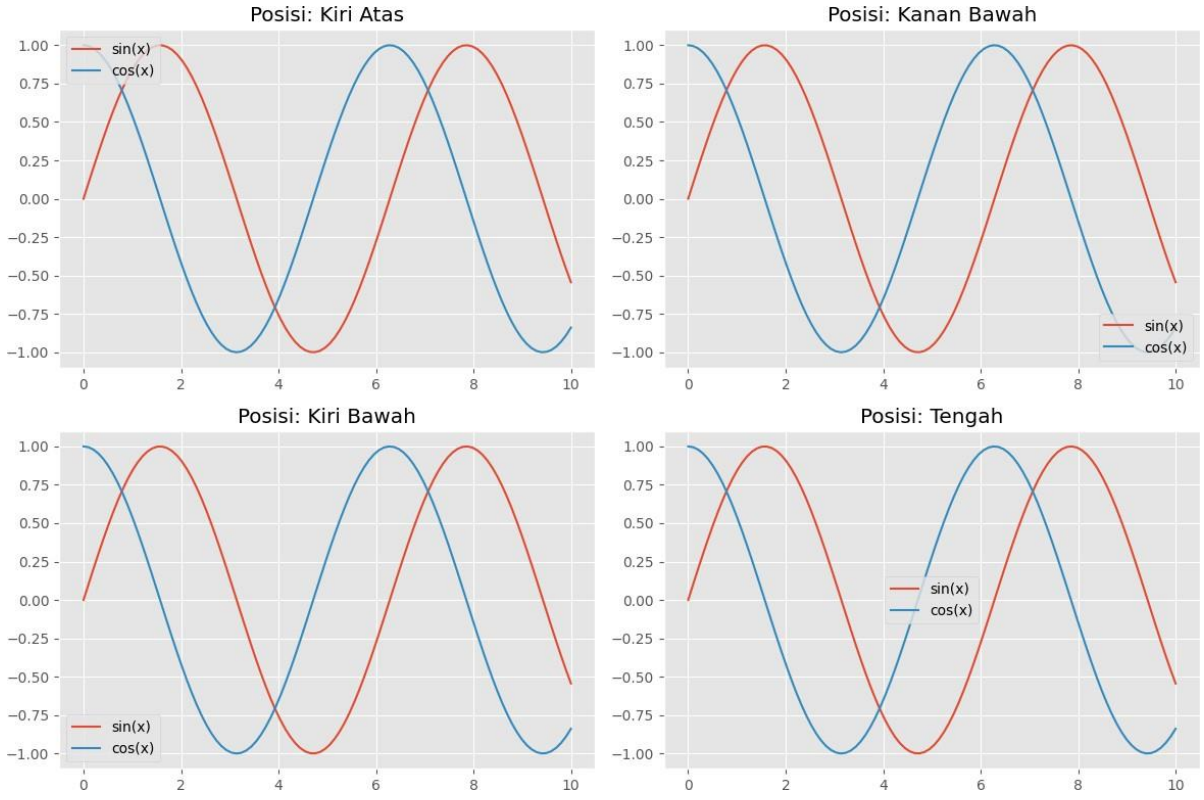
Standar penempatan legenda adalah di pojok kanan atas plot. Namun, terkadang kamu mungkin ingin memindahkannya ke lokasi lain agar tidak mengganggu tampilan data. Untungnya, Matplotlib menyediakan opsi untuk memindahkan legenda ke berbagai lokasi dalam plot.

**Contoh: Mengatur Posisi Legenda**

```
# Definisikan kembali data sintetis
x_legend = np.linspace(0, 10, 100)
y1_legend = np.sin(x_legend)
y2_legend = np.cos(x_legend)

# Membuat plot dengan legenda di berbagai posisi
fig, axs = plt.subplots(2, 2, figsize=(12, 8))

# Posisi kiri atas
axs[0, 0].plot(x_legend, y1_legend, label='sin(x)')
axs[0, 0].plot(x_legend, y2_legend, label='cos(x)')
axs[0, 0].legend(loc='upper left')
axs[0, 0].set_title('Posisi: Kiri Atas')

# Posisi kanan bawah
axs[0, 1].plot(x_legend, y1_legend, label='sin(x)')
axs[0, 1].plot(x_legend, y2_legend, label='cos(x)')
```

```
axs[0, 1].legend(loc='lower right')
axs[0, 1].set_title('Posisi: Kanan Bawah')

# Posisi kiri bawah
axs[1, 0].plot(x_legend, y1_legend, label='sin(x)')
axs[1, 0].plot(x_legend, y2_legend, label='cos(x)')
axs[1, 0].legend(loc='lower left')
axs[1, 0].set_title('Posisi: Kiri Bawah')

# Posisi tengah
axs[1, 1].plot(x_legend, y1_legend, label='sin(x)')
axs[1, 1].plot(x_legend, y2_legend, label='cos(x)')
axs[1, 1].legend(loc='center')
axs[1, 1].set_title('Posisi: Tengah')

plt.tight_layout()
plt.show()
```

Output:




Kamu bisa melihat bagaimana legenda diletakkan di berbagai posisi dalam plot. Dengan parameter loc pada metode legend(), kita dapat dengan mudah mengatur posisi legenda. Ada banyak pilihan posisi yang tersedia, seperti 'upper right', 'upper left', 'lower right', 'lower left', 'center', dan banyak lagi.

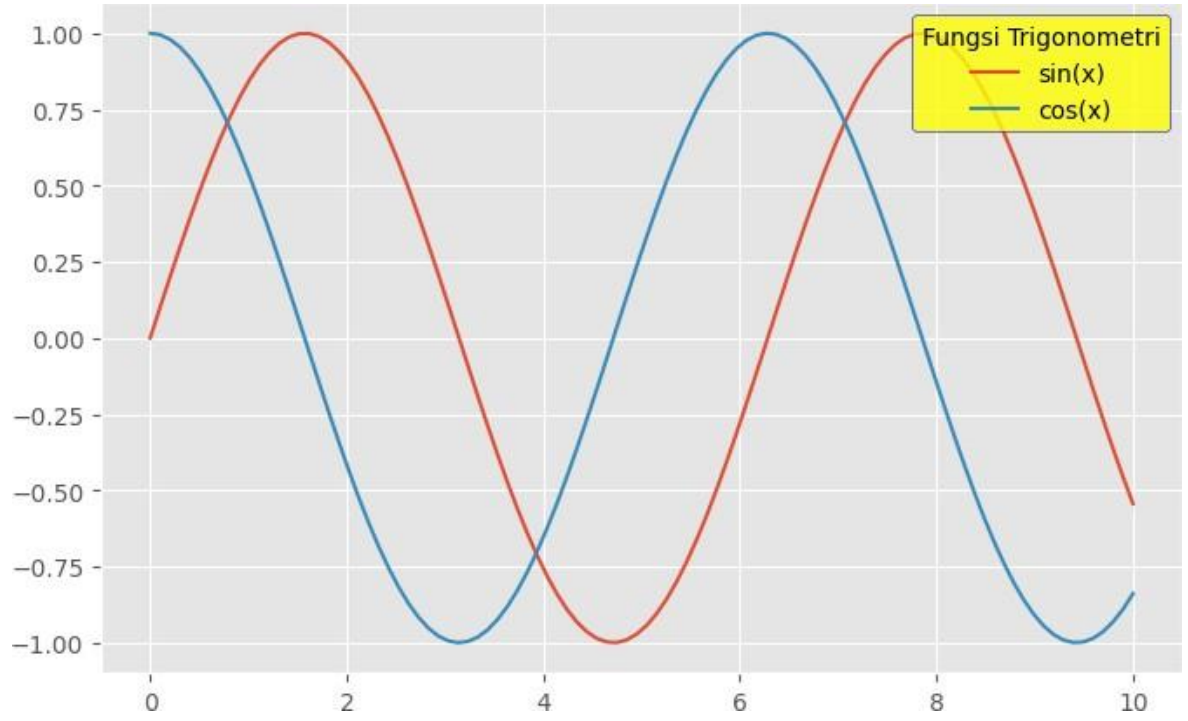
### 3.6.2.3. Mengubah Ukuran dan Gaya Legenda

Mungkin kamu ingin mengubah ukuran teks atau gaya kotak legenda. Mari kita jelajahi beberapa kustomisasi yang bisa kamu lakukan.

**Contoh: Mengatur Ukuran Teks dan Gaya Kotak Legenda**

```
# Membuat plot dengan kustomisasi legenda
fig, ax = plt.subplots(figsize=(8, 5))

# Plot data
ax.plot(x_legend, y1_legend, label='sin(x)')
ax.plot(x_legend, y2_legend, label='cos(x)')

# Menambahkan legenda dengan kustomisasi
ax.legend(loc='upper right', fontsize=10, edgecolor='blue',
facecolor='yellow', title='Fungsi Trigonometri')

plt.show()
```

Output:




Dari contoh di atas, kamu bisa melihat bahwa kita telah mengubah ukuran teks legenda dengan parameter fontsize, memberikan warna tepi dengan edgecolor, dan mengatur warna latar belakang dengan facecolor. Selain itu, kita juga menambahkan judul ke legenda dengan parameter title.

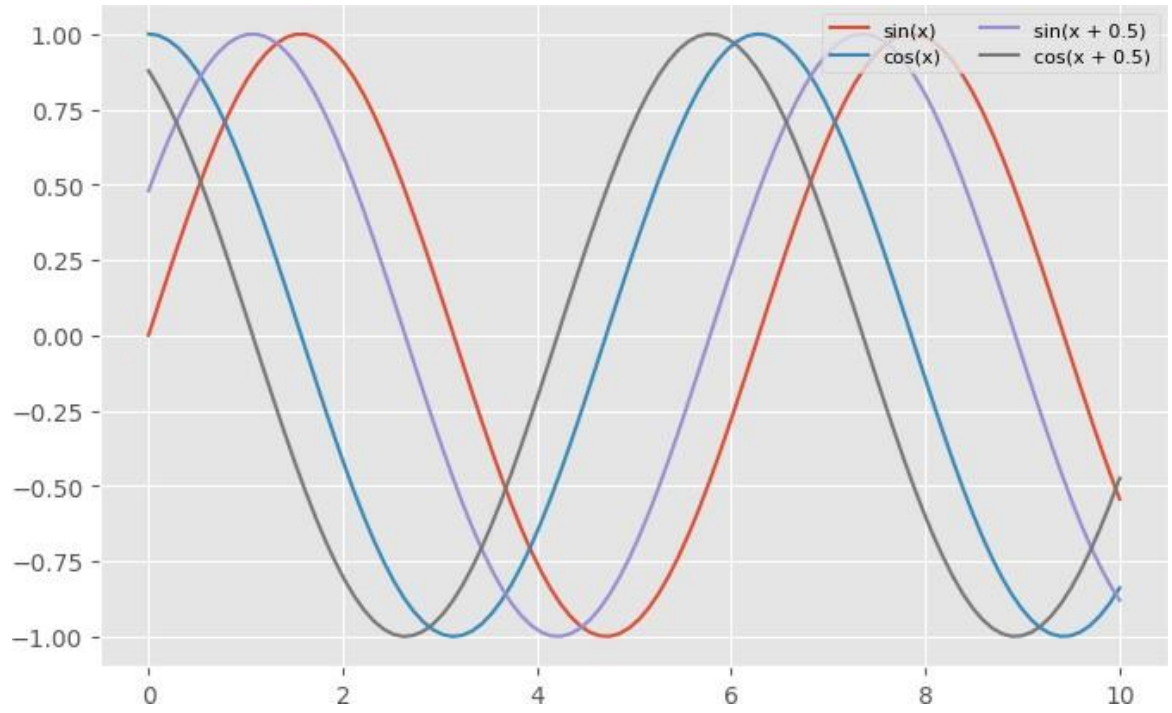
### 3.6.2.4. Legenda dengan Banyak Kolom

Jika kamu memiliki banyak entri legenda dan ingin menampilkannya dalam beberapa kolom agar lebih rapi, kamu bisa menggunakan parameter ncol pada metode legend().

**Contoh: Legenda dengan Dua Kolom**

```
# Data sintetis tambahan
y3_legend = np.sin(x_legend + 0.5)
y4_legend = np.cos(x_legend + 0.5)

# Membuat plot dengan legenda 2 kolom
fig, ax = plt.subplots(figsize=(8, 5))

# Plot data
ax.plot(x_legend, y1_legend, label='sin(x)')
ax.plot(x_legend, y2_legend, label='cos(x)')
ax.plot(x_legend, y3_legend, label='sin(x + 0.5)')
ax.plot(x_legend, y4_legend, label='cos(x + 0.5)')

# Menambahkan legenda dengan 2 kolom
ax.legend(loc='upper right', ncol=2, fontsize=8)

plt.show()
```

Output:



Dalam plot di atas, legenda telah dibagi menjadi dua kolom, membuat tampilan lebih rapi dan kompak. Parameter ncol memungkinkan kamu mengontrol jumlah kolom dalam legenda.

### 3.6.3. Kustomisasi Tambahan

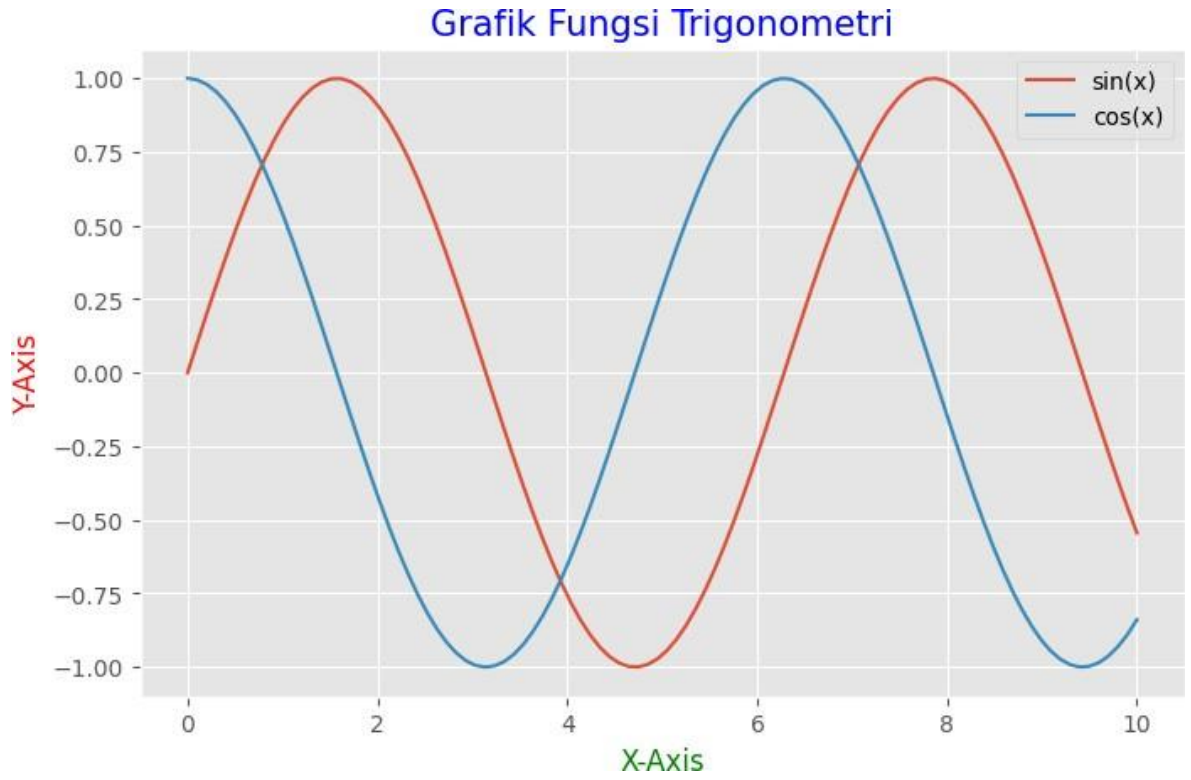
#### 3.6.3.1. Mengatur Kustomisasi Teks

Kustomisasi teks dalam plot bukan hanya terbatas pada legenda. Kamu bisa mengubah font, ukuran, warna, dan gaya teks untuk judul, label sumbu, dan lain-lain.

Contoh: Mengubah Ukuran dan Warna Teks Judul dan Label Sumbu

```
# Membuat plot dengan kustomisasi teks
fig, ax = plt.subplots(figsize=(8, 5))

# Plot data
ax.plot(x_legend, y1_legend, label='sin(x)')
ax.plot(x_legend, y2_legend, label='cos(x)')

# Menambahkan judul dengan ukuran dan warna kustom
ax.set_title('Grafik Fungsi Trigonometri', fontsize=15, color='blue')

# Menambahkan label sumbu dengan ukuran dan warna kustom
ax.set_xlabel('X-Axis', fontsize=12, color='green')
ax.set_ylabel('Y-Axis', fontsize=12, color='red')

# Menambahkan legenda
ax.legend()

plt.show()
```

Output:




Pada contoh di atas, kita menggunakan metode set_title untuk mengatur judul plot dengan ukuran font dan warna kustom. Untuk label sumbu, kita menggunakan metode set_xlabel dan set_ylabel. Hasilnya, kita mendapatkan plot dengan teks yang lebih menarik dan informatif.

#### 3.6.3.2. Mengubah Style Plot

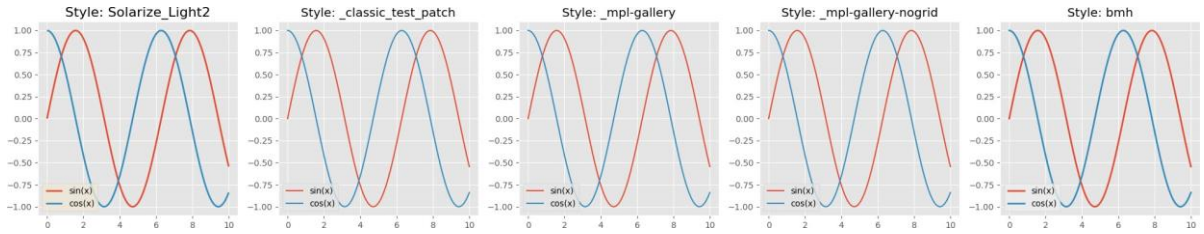
Kamu juga dapat mengubah gaya plot secara keseluruhan menggunakan berbagai style yang telah disediakan oleh Matplotlib.

**Contoh: Menggunakan Gaya Plot Berbeda**

```
# Daftar beberapa gaya yang tersedia
available_styles = plt.style.available[:5]

# Membuat plot dengan gaya berbeda
fig, axs = plt.subplots(1, 5, figsize=(20, 4))

for i, style in enumerate(available_styles):
    with plt.style.context(style):
        axs[i].plot(x_legend, y1_legend, label='sin(x)')
        axs[i].plot(x_legend, y2_legend, label='cos(x)')
        axs[i].legend()
        axs[i].set_title(f'Style: {style}')
```

```
plt.tight_layout()
plt.show()
```

Output:




Dalam contoh di atas, kita menggunakan lima gaya plot berbeda dari yang tersedia di Matplotlib. Dengan menggunakan context manager plt.style.context(style), kita bisa menerapkan gaya tersebut ke setiap subplot. Ini memberi kita cara yang mudah untuk mengeksplorasi dan menerapkan gaya yang berbeda untuk plot kita.

# MODUL 4: Gaya dan Estetika

## 4.1. Menggunakan Style

Membuat plot yang informatif adalah hal penting, tetapi menambahkan sentuhan estetika ke dalam plot membuatnya lebih menarik dan menyenangkan untuk dilihat. Dalam subbab ini, kita akan menjelajahi cara menggunakan berbagai style yang tersedia dalam Matplotlib, mengkustomisasi style sendiri, dan mengaplikasikan gaya yang berbeda untuk plot yang berbeda. Kamu akan belajar bagaimana mengubah tampilan plot kamu dengan beberapa baris kode sederhana.

### 4.1.1. Apa Itu Style?

Dalam konteks visualisasi data, style adalah kumpulan pengaturan estetika yang mengatur penampilan plot. Hal ini mencakup, tetapi tidak terbatas pada, warna, font, garis, marker, dan lain-lain. Dengan menggunakan style, kamu dapat dengan cepat mengubah tampilan plot kamu tanpa harus secara manual mengatur setiap parameter.

Matplotlib menyediakan berbagai style yang dapat kamu gunakan, dan kamu juga dapat membuat style kamu sendiri. Mari kita mulai dengan cara menggunakan style yang telah disediakan.

### 4.1.2. Menggunakan Style yang Tersedia

Matplotlib menyediakan sejumlah style yang telah ditentukan sebelumnya. Ini memungkinkan kamu untuk dengan cepat mengubah tampilan plot kamu dengan satu baris kode. Berikut adalah cara menggunakan style yang telah disediakan:

#### 4.1.2.1. Melihat Style yang Tersedia

Untuk melihat daftar style yang tersedia, kamu dapat menggunakan plt.style.available. Mari kita lihat beberapa style yang bisa kamu gunakan.

```
import matplotlib.pyplot as plt

# Menampilkan daftar beberapa style yang tersedia dalam Matplotlib
available_styles = plt.style.available[:10]
available_styles
```

Output:

['Solarize_Light2', '_classic_test_patch', '_mpl-gallery',
'_mpl-gallery-nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot']

Berikut adalah beberapa style yang tersedia dalam Matplotlib:

- Solarize_Light2: Style dengan pencahayaan yang lembut.

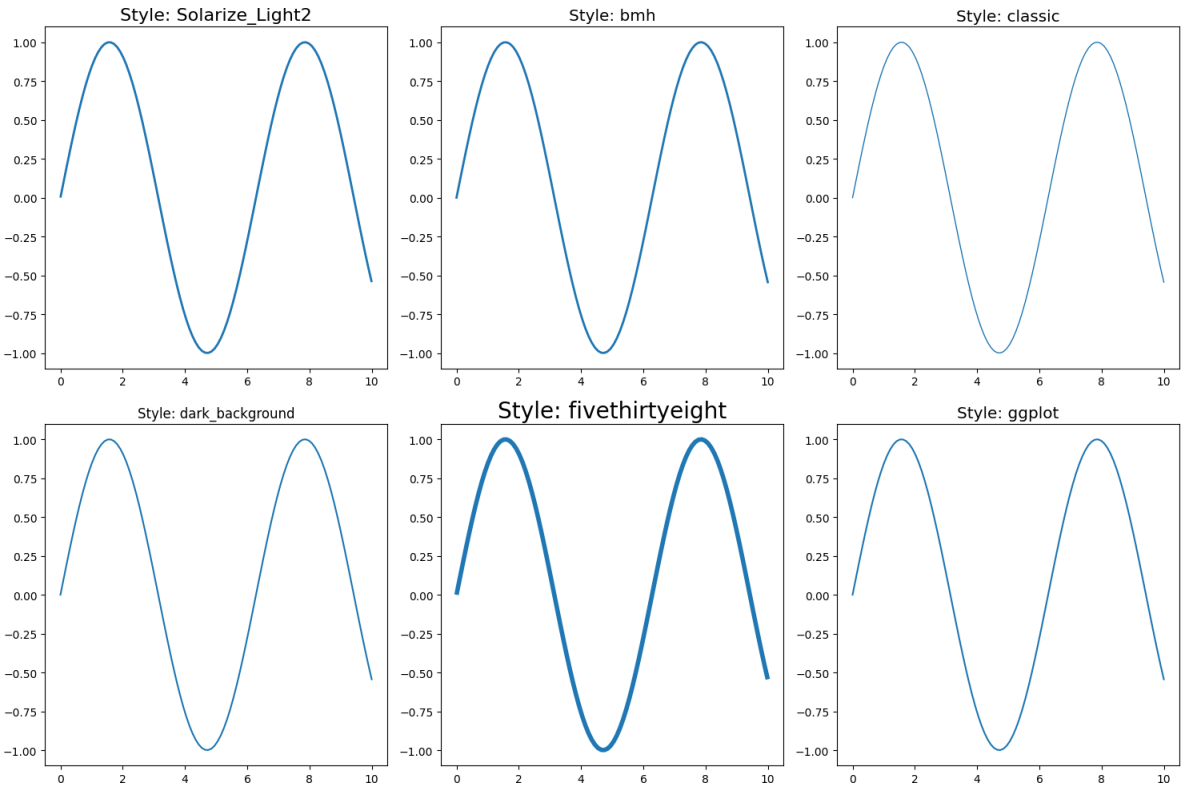
- bmh: Style yang terinspirasi oleh situs blog "Bayesian Methods for Hackers".
- classic: Gaya klasik Matplotlib.
- dark_background: Gaya dengan latar belakang gelap.
- fast: Gaya yang dioptimalkan untuk kecepatan rendering.
- fivethirtyeight: Gaya yang terinspirasi oleh situs FiveThirtyEight.
- ggplot: Gaya yang terinspirasi oleh ggplot (sistem plotting di R).
- grayscale: Gaya dengan skala abu-abu.
- seaborn: Gaya yang terinspirasi oleh library Seaborn.

#### 4.1.2.2. Menggunakan Style dalam Plot

Kamu dapat menggunakan salah satu style ini dengan mudah dalam plot kamu. Berikut adalah beberapa contoh.

```
import numpy as np

# Data sintetis
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Contoh penggunaan beberapa style yang berbeda
fig, axs = plt.subplots(2, 3, figsize=(15, 10))
styles = ['Solarize_Light2', 'bmh', 'classic', 'dark_background',
'fivethirtyeight', 'ggplot']

for ax, style in zip(axs.flat, styles):
    with plt.style.context(style):
        ax.plot(x, y)
        ax.set_title(f'Style: {style}')

plt.tight_layout()
plt.show()
```

Output:




Dalam contoh di atas, kita menggunakan beberapa style yang berbeda untuk menggambar plot yang sama. Dengan hanya satu baris kode tambahan untuk setiap plot, kita dapat sepenuhnya mengubah tampilannya.

### 4.1.3. Membuat Style Sendiri

Tidak hanya menggunakan style yang telah disediakan, kamu juga dapat membuat style kamu sendiri. Ini memungkinkan kamu untuk menentukan pengaturan estetika yang akan kamu gunakan berulang kali dalam proyek atau seluruh portofolio pekerjaan kamu.

#### 4.1.3.1. Mendefinisikan Style Sendiri

Untuk membuat style sendiri, kamu perlu mendefinisikan pengaturan dalam dictionary dan menggunakan plt.rc untuk mengaplikasikannya. Berikut adalah contoh bagaimana kamu dapat mendefinisikan style sendiri:

```
my_style = {
    'lines.linewidth': 2,
    'lines.marker': 'o',
    'lines.markersize': 8,
    'xtick.labelsize': 12,
    'ytick.labelsize': 12,
    'font.size': 14,
    'axes.labelsize': 14,
```

```
    'axes.titlesize': 16,
    'axes.grid': True,
    'grid.color': '#dddddd',
    'axes.facecolor': '#f0f0f0',
    'figure.facecolor': '#ffffff'
}

plt.rc(my_style)
```

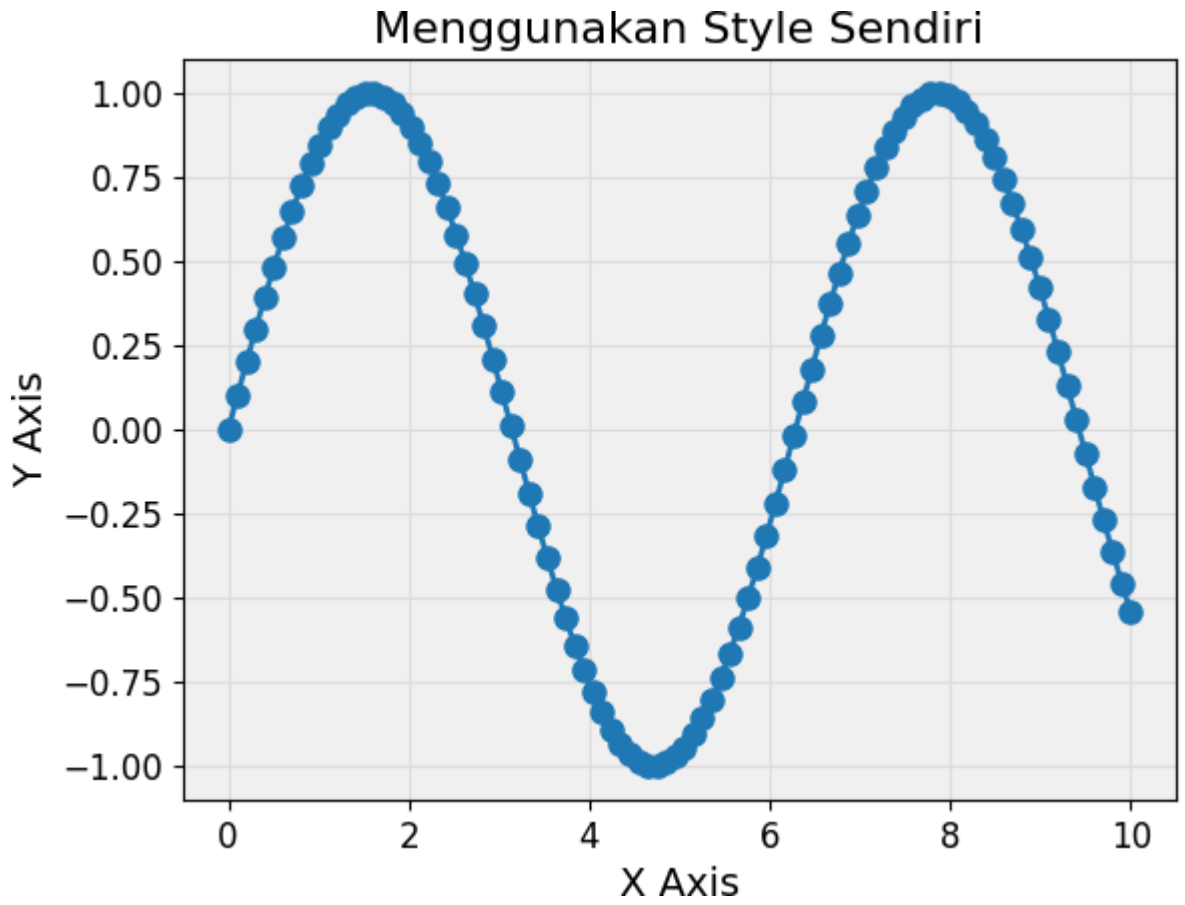
Dalam contoh di atas, kita telah mendefinisikan berbagai pengaturan, termasuk lebar garis, ukuran marker, ukuran label sumbu, dan warna.

#### 4.1.3.2. Menggunakan Style Sendiri

Setelah mendefinisikan style, kamu dapat menggunakannya dalam plot kamu seperti biasa. Berikut adalah contoh penggunaan style yang telah kita definisikan:

```
# Mendefinisikan style sendiri
my_style = {
    'lines.linewidth': 2,
    'lines.marker': 'o',
    'lines.markersize': 8,
    'xtick.labelsize': 12,
    'ytick.labelsize': 12,
    'font.size': 14,
    'axes.labelsize': 14,
    'axes.titlesize': 16,
    'axes.grid': True,
    'grid.color': '#dddddd',
    'axes.facecolor': '#f0f0f0',
    'figure.facecolor': '#ffffff'
}

# Mengaplikasikan style
with plt.rc_context(my_style):
    plt.plot(x, y)
    plt.title('Menggunakan Style Sendiri')
    plt.xlabel('X Axis')
    plt.ylabel('Y Axis')
    plt.grid(True)
    plt.show()
```

Output:




Dalam contoh di atas, kita telah menggunakan style yang telah kita definisikan sendiri untuk menggambar plot. Seperti yang kamu lihat, style ini memberikan tampilan yang bersih dan rapi, dengan garis yang tebal, marker berbentuk lingkaran, dan grid yang halus. Semua ini dapat dikustomisasi sesuai keinginan kamu.

## 4.2. Palet Warna

Warna adalah bagian integral dari visualisasi data. Mereka tidak hanya menambah estetika tetapi juga membantu dalam mengkomunikasikan informasi yang tepat. Subbab ini akan membahas cara menggunakan palet warna dalam Matplotlib dan bagaimana cara memilih palet yang tepat untuk visualisasi data kamu.

### 4.2.1. Apa Itu Palet Warna?

Palet warna adalah kumpulan warna yang digunakan dalam plot. Palet ini bisa berupa sekumpulan warna yang disiapkan atau bisa juga kamu definisikan sendiri. Palet yang dipilih dapat memberi kesan yang berbeda dan mengkomunikasikan informasi yang berbeda.

### 4.2.2. Menggunakan Palet Warna Bawaan Matplotlib

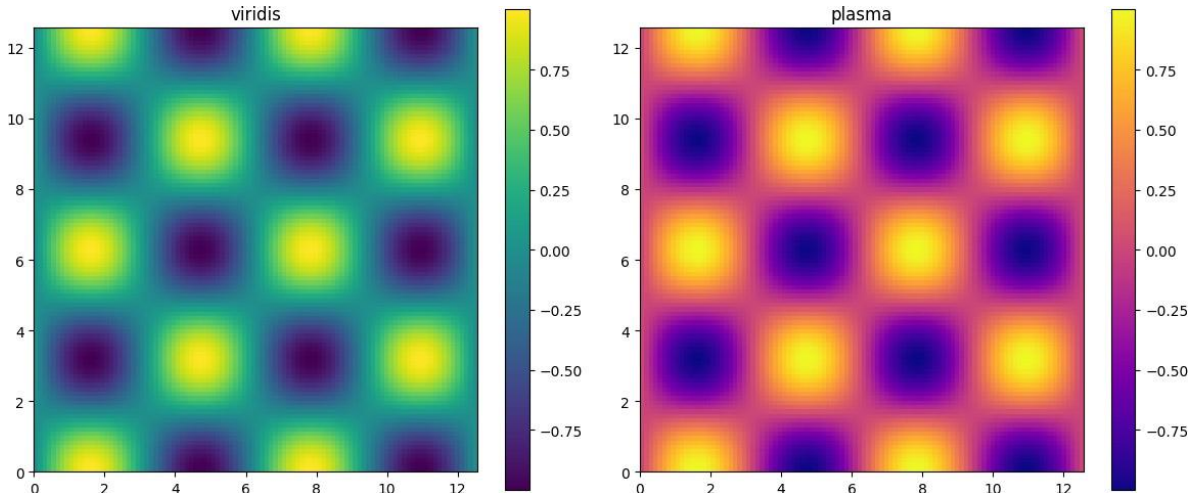
Matplotlib menyediakan berbagai palet warna bawaan yang bisa kamu gunakan langsung. Beberapa palet populer termasuk:

- 'viridis'
- 'plasma'
- 'inferno'
- 'magma'
- 'cividis'

Kamu bisa menerapkannya dengan menggunakan fungsi plt.set_cmap. Mari kita coba plot heatmap dengan palet ‘viridis’ dan ‘plasma’:

```
# Generating a 2D grid of values
x = np.linspace(0, 4 * np.pi, 100)
y = np.linspace(0, 4 * np.pi, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(X) * np.cos(Y)

# Creating the heatmaps
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# Using viridis colormap
c1 = ax[0].imshow(Z, cmap='viridis', extent=[0, 4*np.pi, 0, 4*np.pi])
ax[0].set_title('viridis')
fig.colorbar(c1, ax=ax[0])

# Using plasma colormap
c2 = ax[1].imshow(Z, cmap='plasma', extent=[0, 4*np.pi, 0, 4*np.pi])
ax[1].set_title('plasma')
fig.colorbar(c2, ax=ax[1])

plt.tight_layout()
plt.show()
```

Output:




### 4.2.3. Membuat Palet Warna Sendiri

Terkadang, palet bawaan mungkin tidak sesuai dengan kebutuhan kamu. Dalam hal ini, kamu dapat membuat palet warna sendiri. Berikut adalah contoh cara membuat palet warna dengan tiga warna:

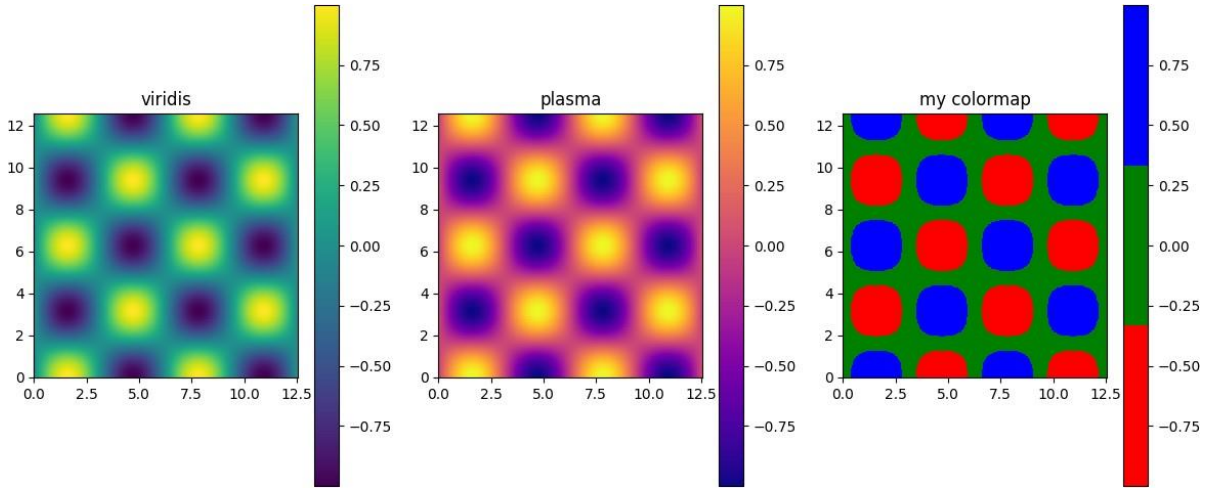
```
from matplotlib.colors import ListedColormap

my_palette = ListedColormap(['red', 'green', 'blue'])
```

Kamu kemudian bisa menerapkan palet ini ke plot kamu. Misalnya:

```
# Generating a 2D grid of values
x = np.linspace(0, 4 * np.pi, 100)
y = np.linspace(0, 4 * np.pi, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(X) * np.cos(Y)

# Creating the heatmaps
fig, ax = plt.subplots(1, 3, figsize=(12, 5))

# Using viridis colormap
c1 = ax[0].imshow(Z, cmap='viridis', extent=[0, 4*np.pi, 0, 4*np.pi])
ax[0].set_title('viridis')
fig.colorbar(c1, ax=ax[0])

# Using plasma colormap
c2 = ax[1].imshow(Z, cmap='plasma', extent=[0, 4*np.pi, 0, 4*np.pi])
ax[1].set_title('plasma')
fig.colorbar(c2, ax=ax[1])

# Using your colormap
```

```
c3 = ax[2].imshow(Z, cmap=my_palette, extent=[0, 4*np.pi, 0, 4*np.pi])
ax[2].set_title('my colormap')
fig.colorbar(c3, ax=ax[2])

plt.tight_layout()
plt.show()
```

Output:

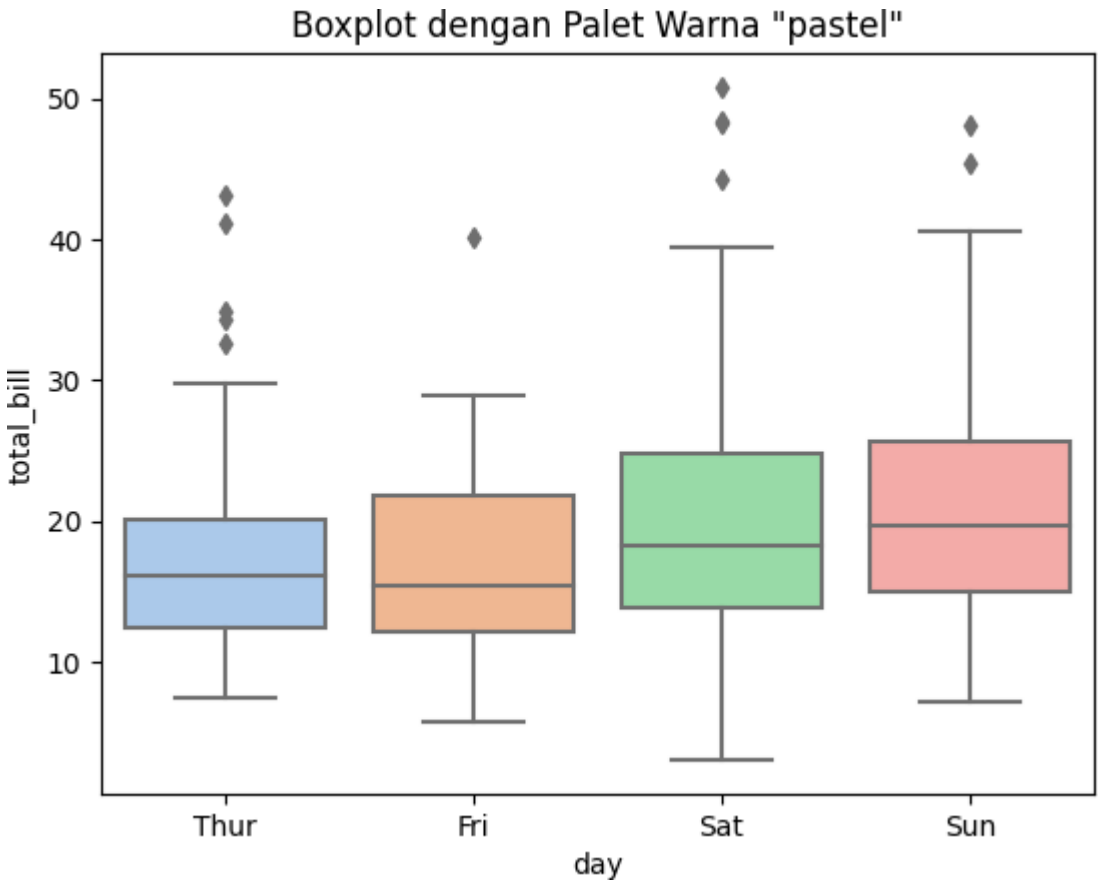



### 4.2.4. Plot Seaborn dengan Palet Warna Kategorikal

Seaborn menyediakan banyak palet warna kategorikal yang cocok untuk jenis data yang berbeda. Mari kita coba beberapa di antaranya.

```
import seaborn as sns

tips = sns.load_dataset('tips')
sns.boxplot(x='day', y='total_bill', data=tips, palette='pastel')
plt.title('Boxplot dengan Palet Warna "pastel"')
plt.show()
```

Output:



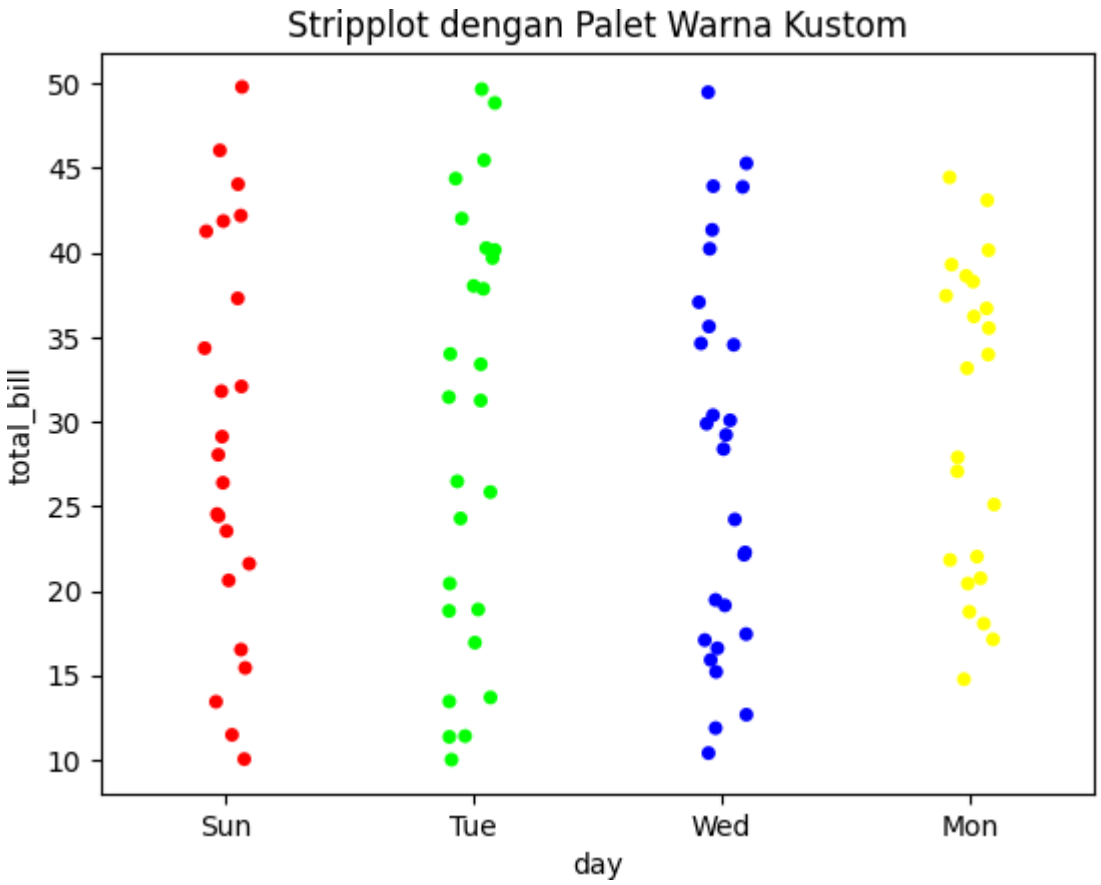

Boxplot di atas menggunakan palet warna "pastel," yang menyediakan kontras lembut antara kategori yang berbeda. Dalam kasus ini, setiap hari diberi warna yang berbeda, memungkinkan kamu untuk dengan mudah membedakan antara mereka.

### 4.2.6. Personalisasi Palet Warna

Kamu juga dapat membuat palet warna kustom dengan menentukan warna secara manual. Ini berguna jika kamu ingin menyelaraskan plot dengan tema khusus atau memiliki persyaratan warna yang spesifik.

```
# Membuat palet warna kustom
custom_palette = sns.color_palette(['#FF0000', '#00FF00', '#0000FF',
'#FFFF00'])

# Membuat plot menggunakan palet kustom
sns.stripplot(x='day', y='total_bill', data=data_tips_sintetis,
palette=custom_palette)
plt.title('Stripplot dengan Palet Warna Kustom')
plt.show()
```

Output:




Stripplot di atas menggunakan palet warna kustom yang didefinisikan dengan warna spesifik. Warna-warna ini memberikan tampilan yang sangat berbeda dari palet bawaan, dan bisa menjadi cara yang efektif untuk menekankan aspek tertentu dari data kamu.

Warna dalam palet ini didefinisikan dalam format hexadecimal, memberikan kontrol penuh atas warna yang digunakan. Dengan cara ini, kamu dapat menyesuaikan plot dengan merek atau tema visual lainnya yang relevan dengan proyek kamu.

## 4.3. Tema dan Background

Sebelum kita benar-benar menyelam ke lautan visualisasi data, ada satu aspek yang sering diabaikan tetapi memiliki dampak besar pada kesan pertama audiens: tampilan latar belakang dan tema dari plot kita.

**Apa Itu Tema?**

Dalam konteks visualisasi data, tema adalah kombinasi dari elemen-elemen desain yang membentuk tampilan keseluruhan dari grafik atau plot kita. Ini termasuk, tetapi tidak terbatas pada, warna latar belakang, gaya garis, warna garis, jenis font, dan sebagainya. Dengan kata lain, tema adalah "pakaian" dari plot kita.

### 4.3.1. Menggunakan Tema Bawaan Seaborn

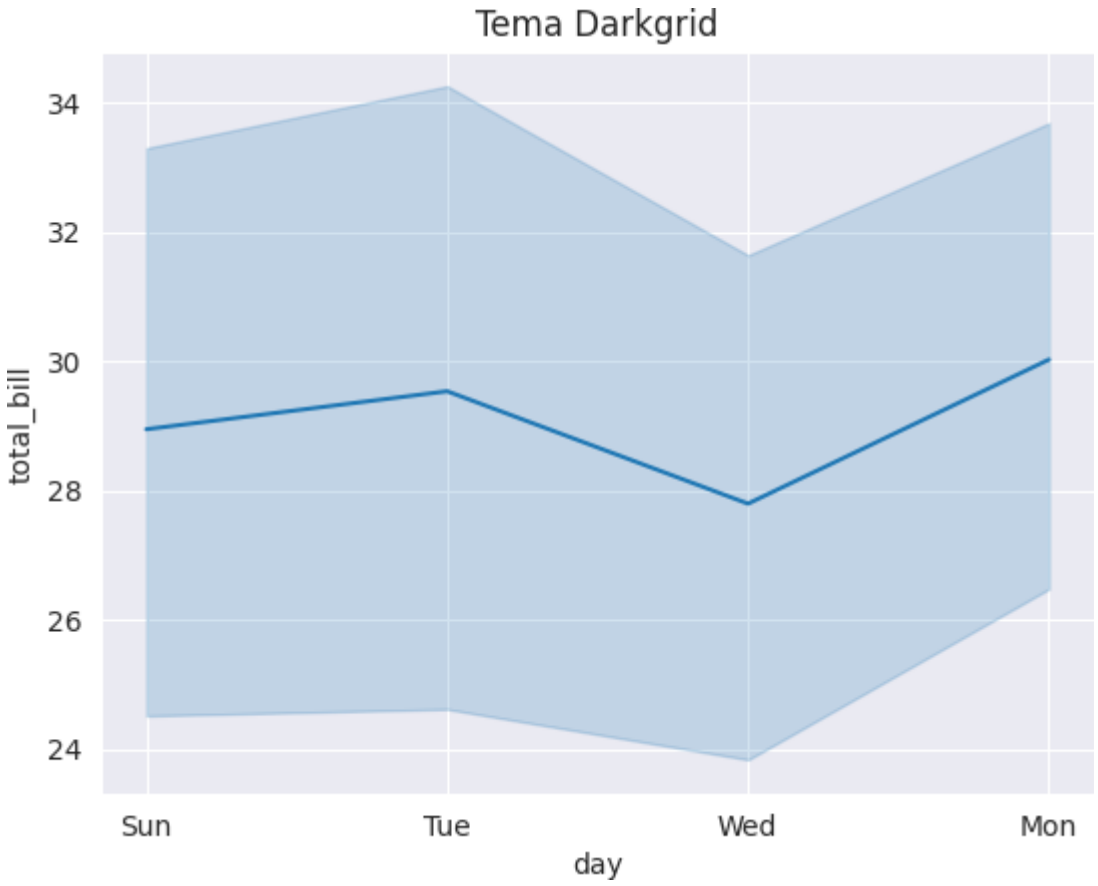
Seaborn, library yang berbasis pada Matplotlib, menawarkan serangkaian tema bawaan yang dapat dengan cepat mengubah tampilan plot kamu. Mari kita lihat beberapa dari tema-tema ini.

**1. Tema Darkgrid**

Ini adalah tema default dari Seaborn. Tema ini menampilkan latar belakang gelap dengan garis-garis grid putih, membuat data kamu menonjol dengan baik.

```
sns.set_style("darkgrid")
sns.lineplot(data=data_tips_sintetis, x='day', y='total_bill')
plt.title('Tema Darkgrid')
plt.show()
```

Output:



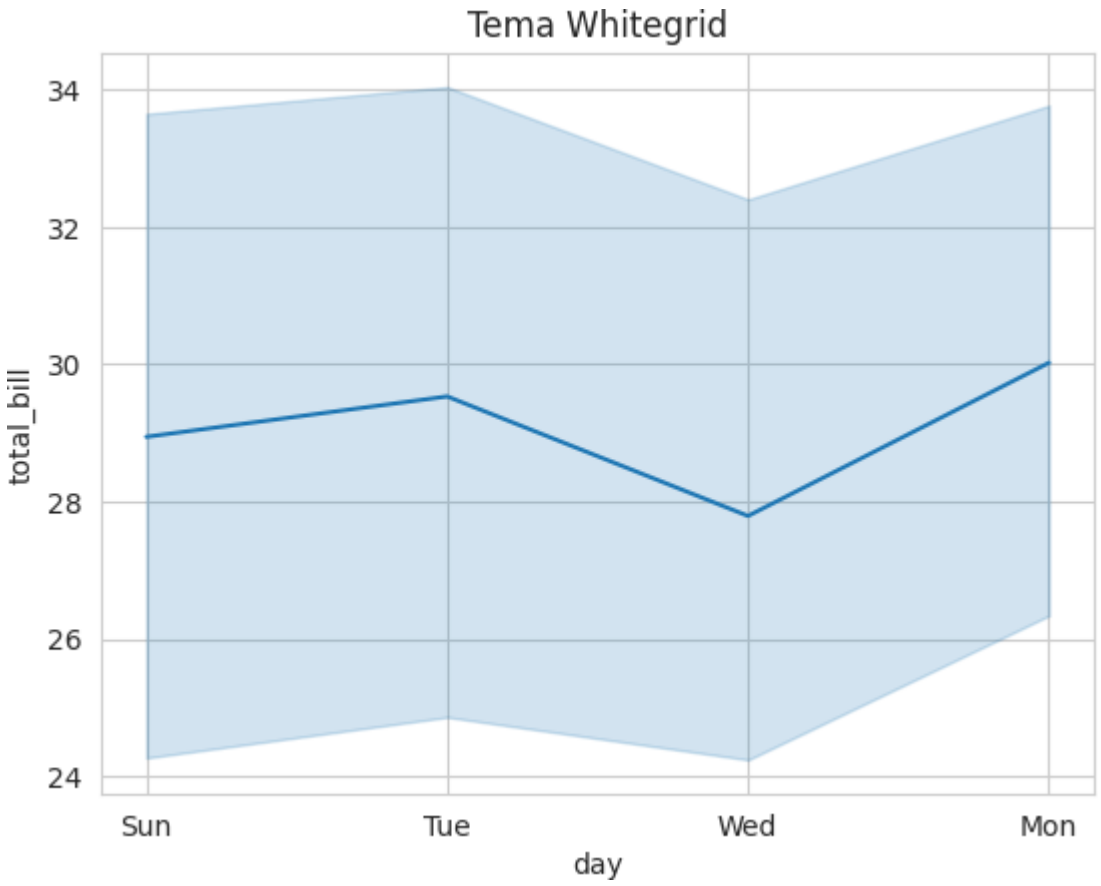

Seperti yang dapat kamu lihat, tema "darkgrid" menawarkan latar belakang gelap dengan garis-garis grid putih, yang menonjolkan data dengan baik. Ini sangat berguna ketika kamu ingin menekankan data daripada elemen-elemen lain dari plot.

**2. Tema Whitegrid**

Tema ini hampir mirip dengan "darkgrid", tetapi dengan latar belakang yang lebih cerah. Garis-garis grid tetap ada, tetapi dengan warna yang lebih gelap untuk menonjolkan data.

```
sns.set_style("whitegrid")
sns.lineplot(data=data_tips_sintetis, x='day', y='total_bill')
plt.title('Tema Whitegrid')
plt.show()
```

Output:



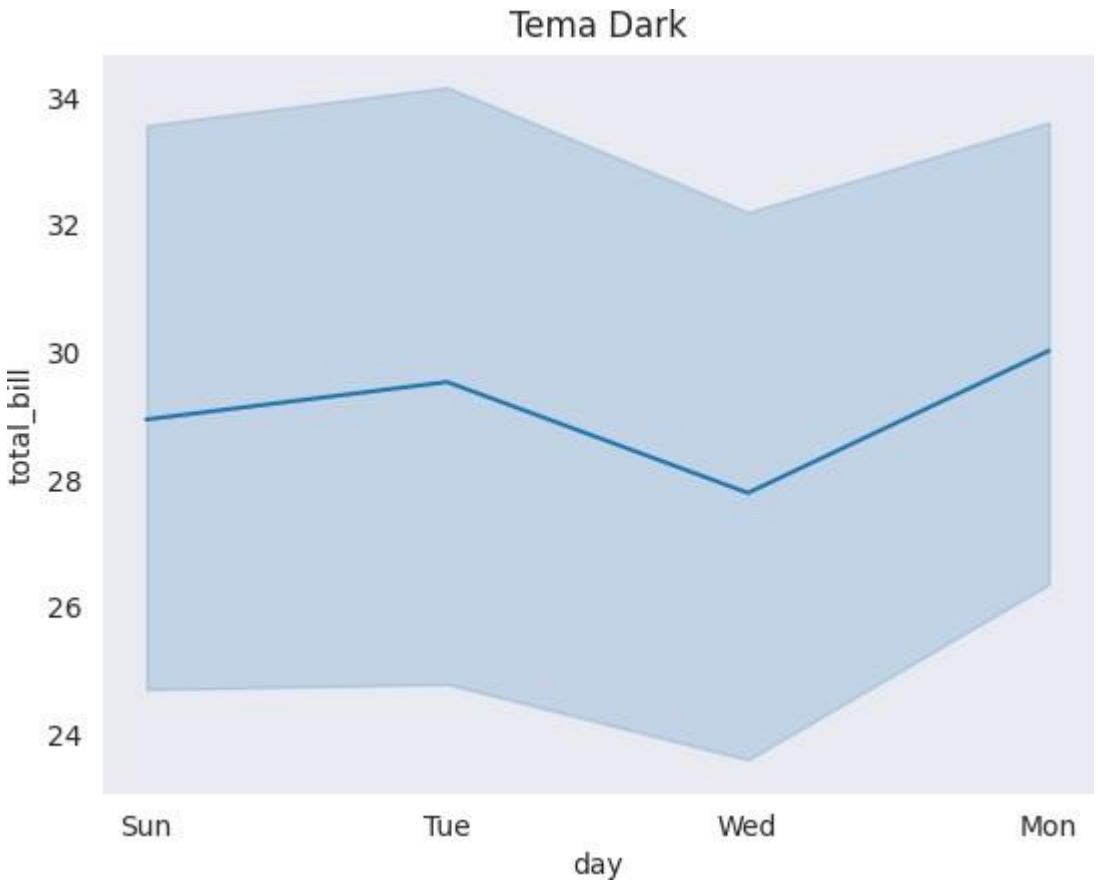

Tema "whitegrid" menghadirkan tampilan yang lebih cerah dan bersih. Garis-garis grid yang gelap membantu dalam menginterpretasikan data dengan lebih mudah pada latar belakang yang putih. Tema ini sangat cocok untuk presentasi atau laporan resmi di mana kamu ingin tampilan yang lebih profesional.

**3. Tema Dark**

Tema ini menampilkan latar belakang yang gelap tanpa garis-garis grid. Ini memberikan tampilan yang lebih minimalis, memfokuskan perhatian audiens sepenuhnya pada data.

```
sns.set_style("dark")
sns.lineplot(data=data_tips_sintetis, x='day', y='total_bill')
plt.title('Tema Dark')
plt.show()
```

Output:

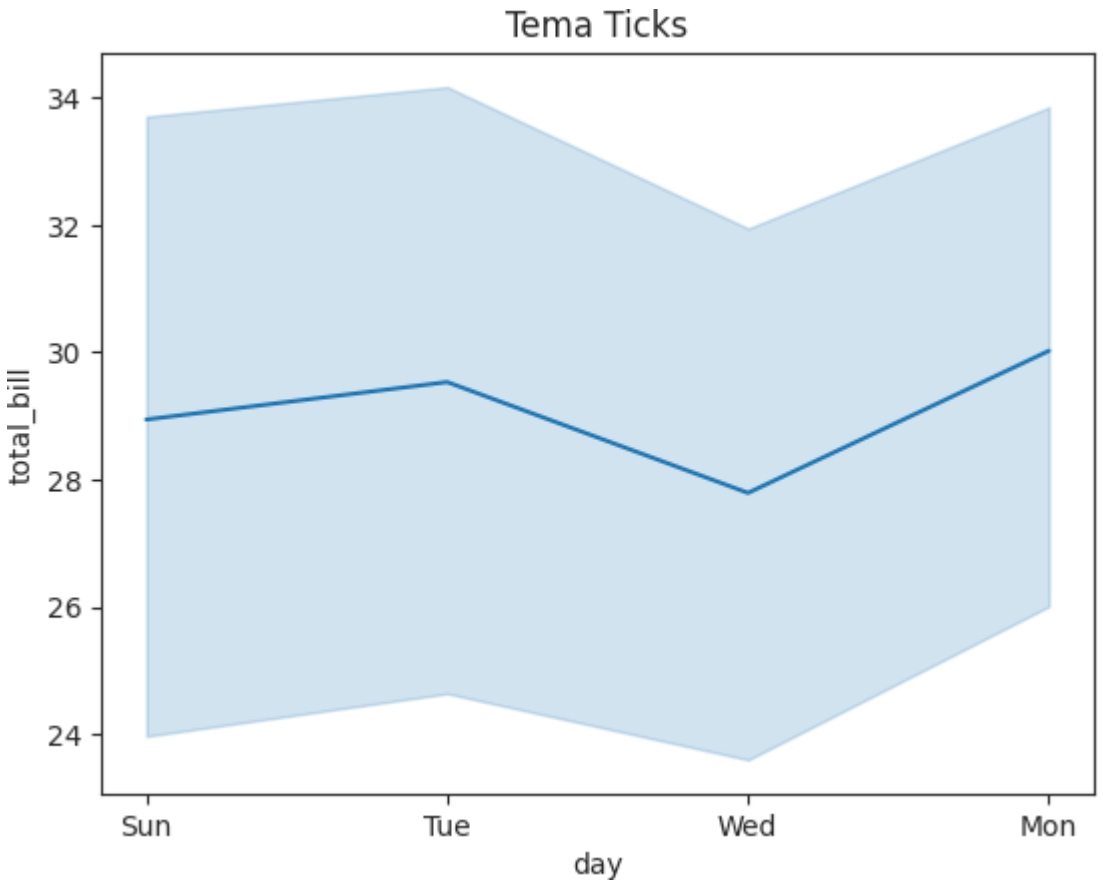



**4. Tema Ticks**

Tema ini adalah variasi dari tema "white", tetapi dengan penambahan 'ticks' atau tanda di sepanjang sumbu. Ini menambahkan sedikit detail visual tanpa mengganggu kesederhanaan dari latar belakang putih.

```
sns.set_style("ticks")
sns.lineplot(data=data_tips_sintetis, x='day', y='total_bill')
plt.title('Tema Ticks')
plt.show()
```

Output:




Seperti yang kamu lihat, tema "ticks" menambahkan tanda kecil di sepanjang sumbu, memberikan sedikit informasi tambahan tanpa mengorbankan kesederhanaan tampilan. Ini cocok untuk situasi di mana kamu ingin memberikan sedikit konteks tambahan tanpa mengganggu keseluruhan estetika.

## 4.4. Efek Bayangan

Bayangan dapat membuat elemen-elemen dalam grafik terlihat lebih nyata dan menonjol. Ini memberikan ilusi kedalaman dan dimensi yang dapat meningkatkan pengalaman visual.

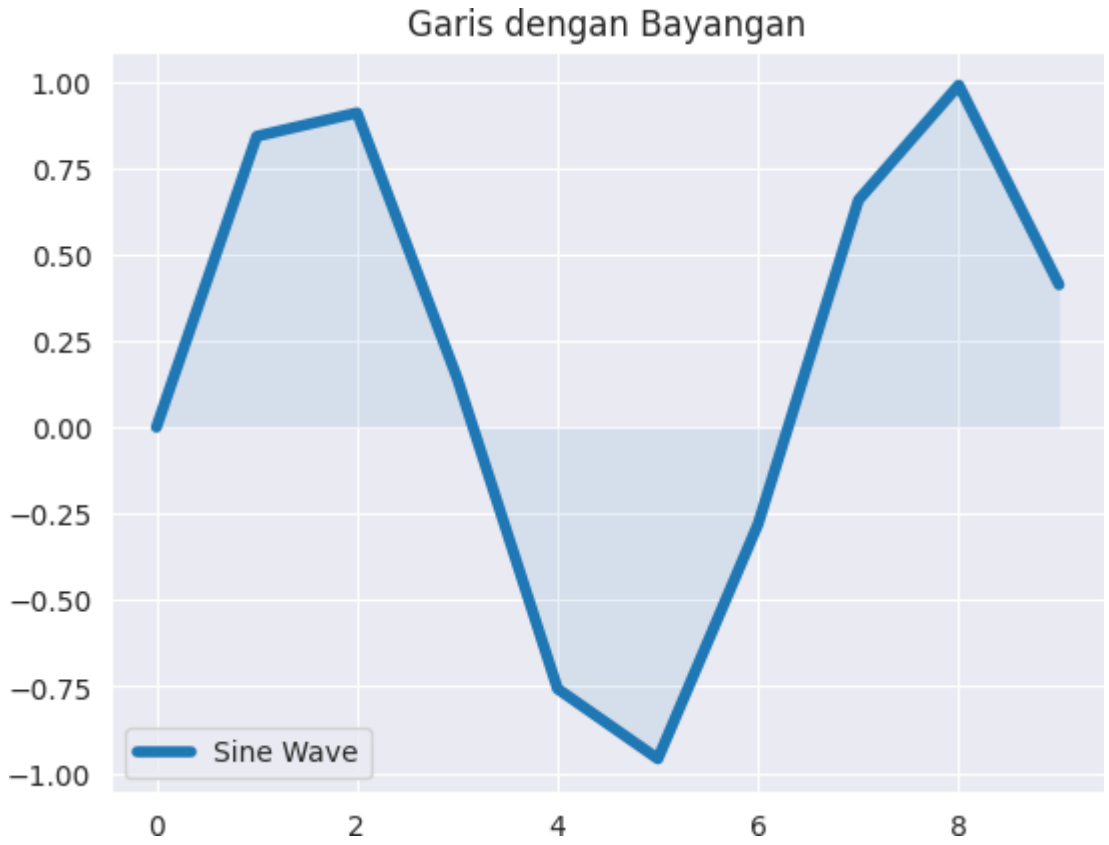
Bayangan pada Garis Plot
Misalkan kita memiliki data berikut dalam bentuk pandas DataFrame:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.DataFrame({
    'x': np.arange(10),
    'y': np.sin(np.arange(10))
})
```

Untuk menambahkan bayangan pada garis, kita bisa menggunakan metode fill_between. Mari kita coba:

```
plt.plot(data['x'], data['y'], lw=4, label='Sine Wave')
plt.fill_between(data['x'], data['y'], alpha=0.1)
plt.legend()
plt.title('Garis dengan Bayangan')
plt.show()
```

Output:




### 4.4.1. Bayangan pada Batang (Bar)

Bayangan juga dapat digunakan pada diagram batang untuk memberikan efek 3D.

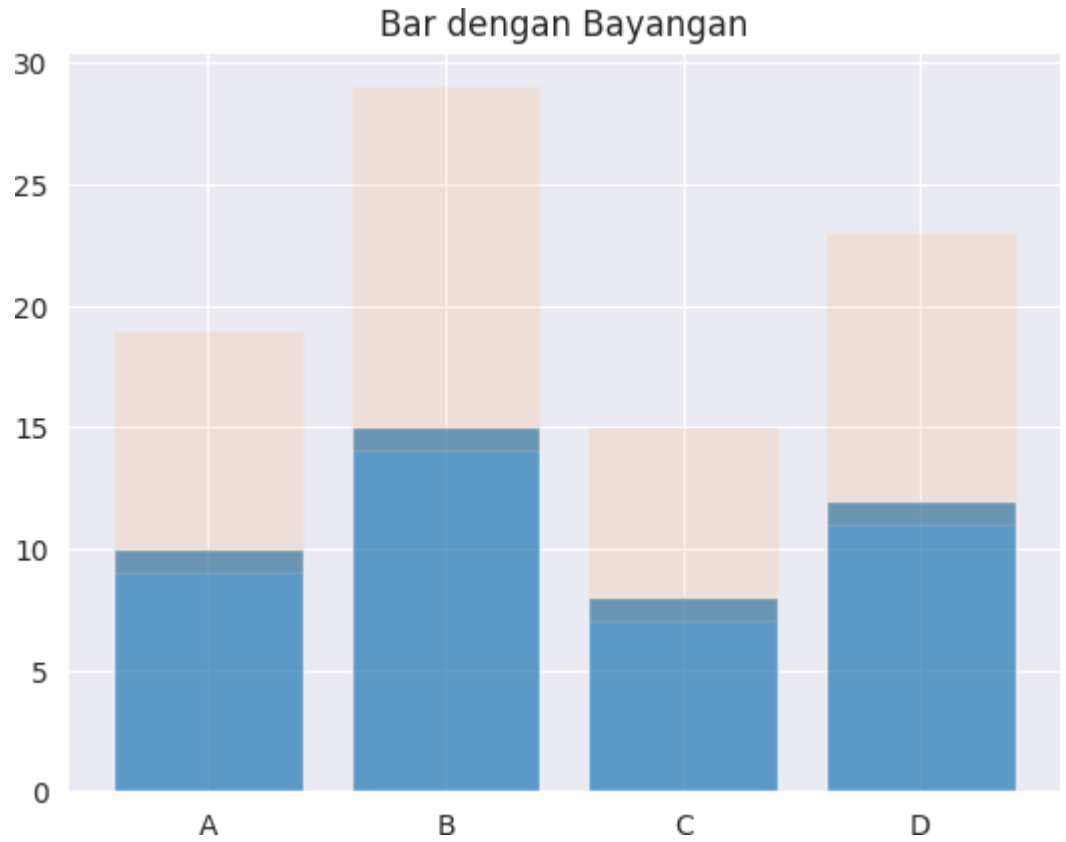
Misalkan kita memiliki data berikut:

```
data_bar = pd.DataFrame({
    'Category': ['A', 'B', 'C', 'D'],
    'Value': [10, 15, 8, 12]
})
```

Untuk membuat bayangan pada batang, kita bisa menggunakan kombinasi dari bar dan barh. Kode berikut akan menunjukkan cara melakukannya:

```
plt.bar(data_bar['Category'], data_bar['Value'], alpha=0.7)
plt.bar(data_bar['Category'], data_bar['Value'], alpha=0.1,
bottom=data_bar['Value'] - 1)
plt.title('Bar dengan Bayangan')
plt.show()
```

Output:




Kita berhasil menambahkan bayangan ke bawah setiap batang dalam diagram batang. Efek ini memberikan sensasi kedalaman dan membuat plot terlihat lebih menarik.

### 4.4.2. Transparansi

Transparansi adalah cara lain untuk menambahkan nuansa visual ke dalam grafik. Ini bisa sangat berguna ketika kamu memiliki banyak elemen yang saling tumpang tindih.

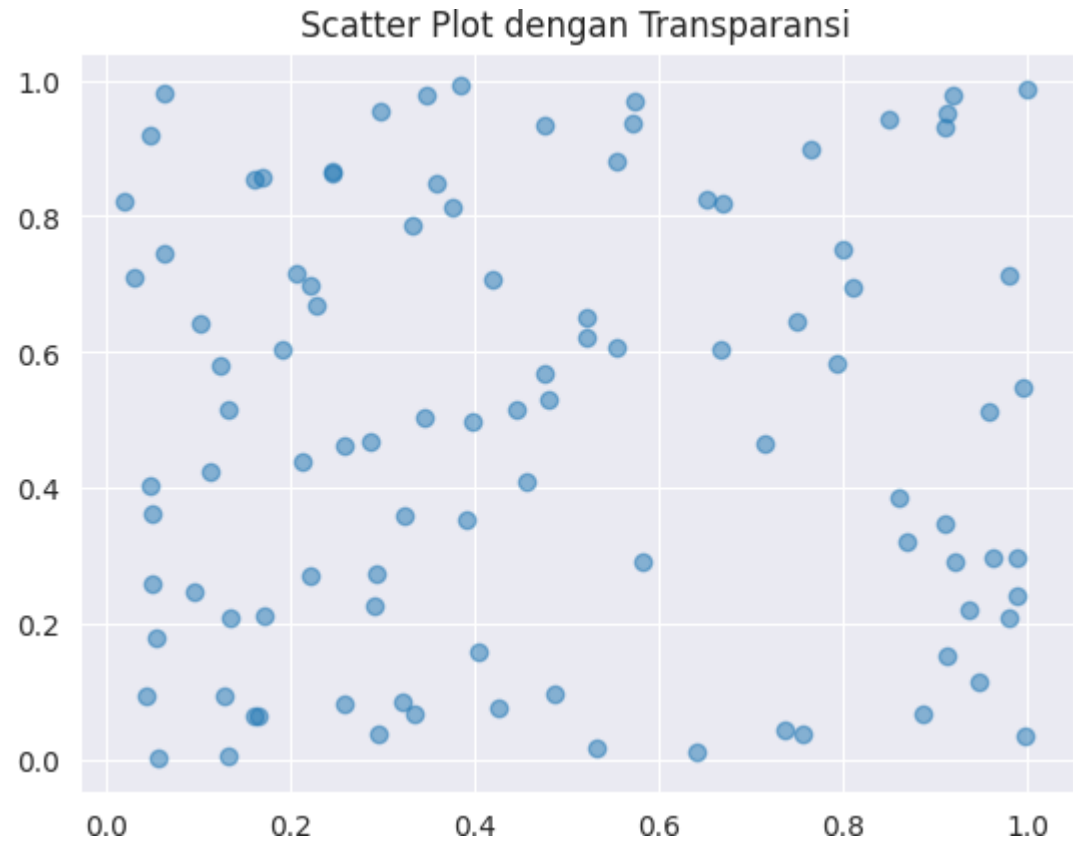
Transparansi pada Scatter Plot
Misalkan kita memiliki data berikut:

```
data_scatter = pd.DataFrame({
    'x': np.random.rand(100),
    'y': np.random.rand(100)
})
```

Kita bisa menggunakan parameter alpha untuk mengatur tingkat transparansi pada scatter plot:

```
plt.scatter(data_scatter['x'], data_scatter['y'], alpha=0.5)
plt.title('Scatter Plot dengan Transparansi')
plt.show()
```

Output:




Hasilnya adalah scatter plot dengan titik-titik yang semi-transparan. Dengan transparansi ini, kita dapat melihat area mana yang memiliki titik yang saling tumpang tindih, sehingga membantu dalam analisis data.

### 4.4.3. Transparansi pada Histogram

Transparansi juga bisa sangat berguna dalam histogram, terutama saat kamu ingin membandingkan distribusi dari dua set data yang berbeda.

Misalkan kita memiliki dua set data berikut:

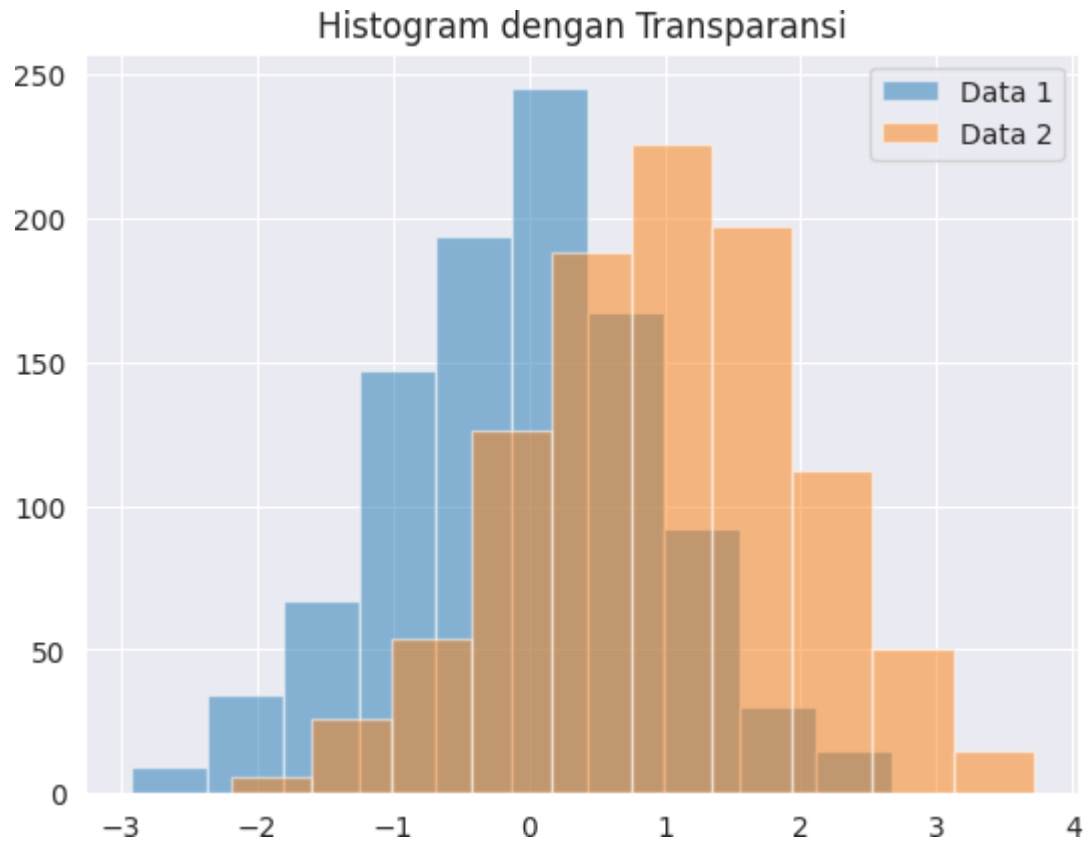
```
data_hist1 = np.random.normal(0, 1, 1000)
data_hist2 = np.random.normal(1, 1, 1000)
```

Kita bisa membuat histogram dengan transparansi sebagai berikut:

```
plt.hist(data_hist1, alpha=0.5, label='Data 1')
plt.hist(data_hist2, alpha=0.5, label='Data 2')
plt.legend()
plt.title('Histogram dengan Transparansi')
```

```
plt.show()
```

Output:




Dalam histogram ini, transparansi memungkinkan kita untuk melihat distribusi kedua set data dalam ruang yang sama. Hal ini membantu kita dalam memahami bagaimana kedua distribusi tersebut berinteraksi dan membantu dalam analisis lebih lanjut.

# MODUL 5: Tips dan Trik Lanjutan

## 5.1. Mengoptimalkan Performa

Visualisasi data adalah seni dan sains yang memungkinkan kita untuk memahami dan mengomunikasikan pola dalam data. Namun, saat kita berurusan dengan dataset yang sangat besar, kita mungkin menemukan bahwa visualisasi yang kita ciptakan tidak hanya memerlukan waktu lama untuk dihasilkan, tetapi juga bisa terasa lamban dan tidak responsif. Dalam situasi ini, mengoptimalkan performa visualisasi kita menjadi sangat penting.

**Kenapa Harus Mengoptilamkan Performa?**
Apakah kamu pernah menunggu berjam-jam hanya untuk melihat plot yang kamu buat? Atau, apakah kamu pernah frustrasi dengan plot interaktif yang bergerak lambat atau bahkan membeku? Ini adalah tanda bahwa kamu perlu mengoptimalkan performa visualisasi.

**Bagaimana Caranya?**
Mari kita bahas beberapa teknik yang dapat digunakan untuk mengoptimalkan performa visualisasi dengan Matplotlib.

### 5.1.1. Reduksi Data

Salah satu cara termudah untuk mempercepat visualisasi adalah dengan mengurangi jumlah data yang akan diplot.

**Contoh: Plot Scatter dengan Subsampling**
Misalkan kita memiliki dataset besar yang berisi satu juta titik, dan kita ingin membuat plot scatter. Langkah pertama adalah membuat data sintetis:

```
import numpy as np
import pandas as pd

np.random.seed(42)
N = 1000000
x = np.random.randn(N)
y = np.random.randn(N)
df_large = pd.DataFrame({'x': x, 'y': y})
```

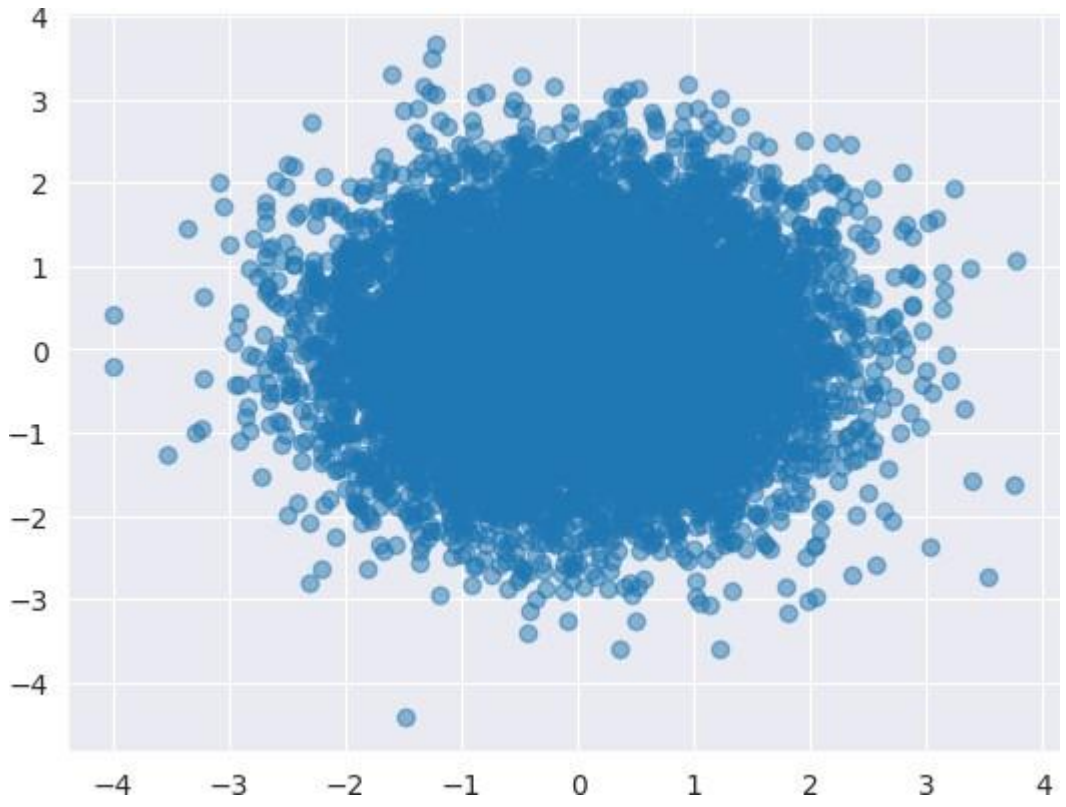
Output:
Jika kita mencoba membuat plot scatter langsung dari data ini, kita akan menemukan bahwa prosesnya sangat lambat. Solusinya adalah melakukan subsampling dari data. Mari kita ambil sampel acak dari 1% dari data:

```
df_sampled = df_large.sample(frac=0.01)
```

Kemudian, kita bisa membuat plot scatter dari data yang sudah disampling:

```
import matplotlib.pyplot as plt

plt.scatter(df_sampled['x'], df_sampled['y'], alpha=0.5)
plt.show()
```

Output:




Dengan hanya mengambil sebagian kecil dari data, kita dapat secara signifikan meningkatkan kecepatan plot tanpa kehilangan terlalu banyak informasi.

### 5.1.2. Memanfaatkan Plotting Paralel

Jika kamu memiliki mesin multi-core, kamu bisa memanfaatkan semua core untuk melakukan plotting secara paralel.

**Contoh: Plotting Histogram dengan Multiprocessing**

Misalkan kita memiliki empat dataset yang berbeda, dan kita ingin membuat histogram dari masing-masing.

Mari kita ciptakan empat dataset acak:

```
datasets = [np.random.randn(1000000) for _ in range(4)]
```

Kemudian kita bisa menggunakan library multiprocessing untuk melakukan plot secara paralel:

```
from multiprocessing import Pool

def plot_histogram(data):
    plt.hist(data, bins=50)
    plt.show()

with Pool() as pool:
    pool.map(plot_histogram, datasets)
```

Dengan menggunakan paralelisme, kita dapat memanfaatkan semua core CPU untuk mempercepat proses plotting.

## 5.2. Menggunakan Plugins dan Ekstensi

Matplotlib adalah perpustakaan yang sangat fleksibel dan dapat dikustomisasi, yang memungkinkan kamu untuk tidak hanya mengubah tampilan plot, tetapi juga menambahkan fungsi tambahan melalui plugins dan ekstensi. Dalam subbab ini, kita akan menjelajahi beberapa cara menambahkan plugins dan ekstensi untuk meningkatkan visualisasi data.

**1. MPLD3: Mengintegrasikan D3.js dengan Matplotlib**

MPLD3 menyediakan cara untuk memvisualisasikan plot Matplotlib dalam bentuk interaktif menggunakan D3.js. Ini memungkinkan kamu untuk mengeksplorasi data dalam cara yang lebih dinamis.

Contoh penggunaan MPLD3:

```
import matplotlib.pyplot as plt
import mpld3

x = [1, 2, 3, 4]
y = [10, 20, 30, 40]

plt.plot(x, y)
mpld3.show()
```

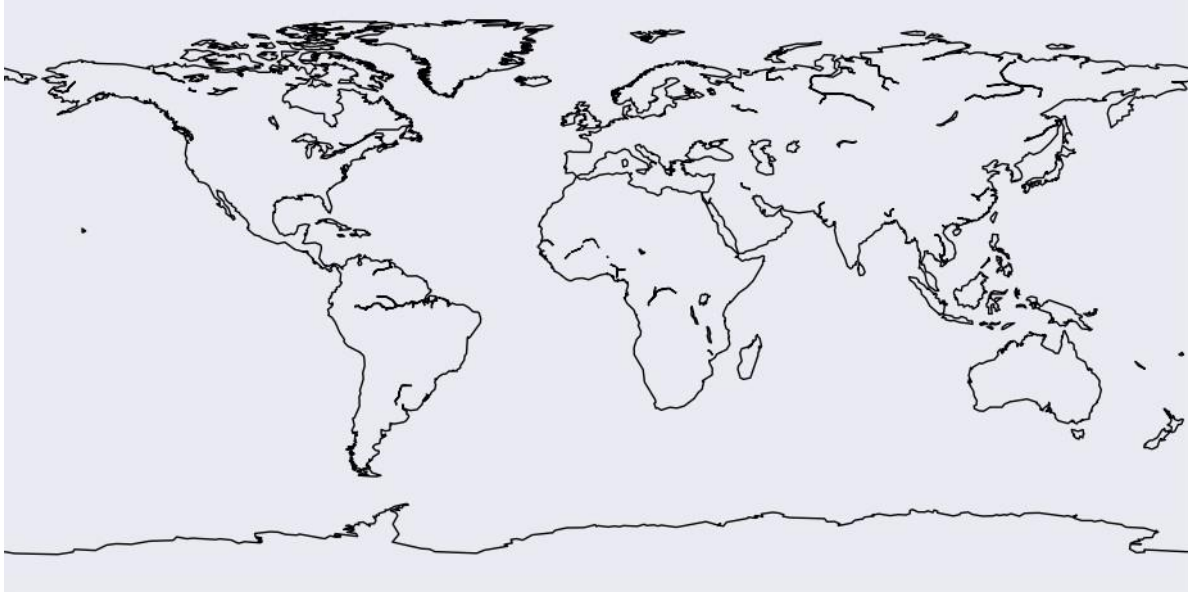
**2. Basemap: Peta Geografis dalam Plot**

Basemap adalah ekstensi yang memungkinkan kamu untuk membuat plot geografis. Kamu bisa menggambar peta dunia, menambahkan garis pantai, sungai, negara, dan lainnya.

Contoh sederhana:

```
from mpl_toolkits.basemap import Basemap

plt.figure(figsize=(12, 6))
map = Basemap()
map.drawcoastlines()
plt.show()
```

Output:



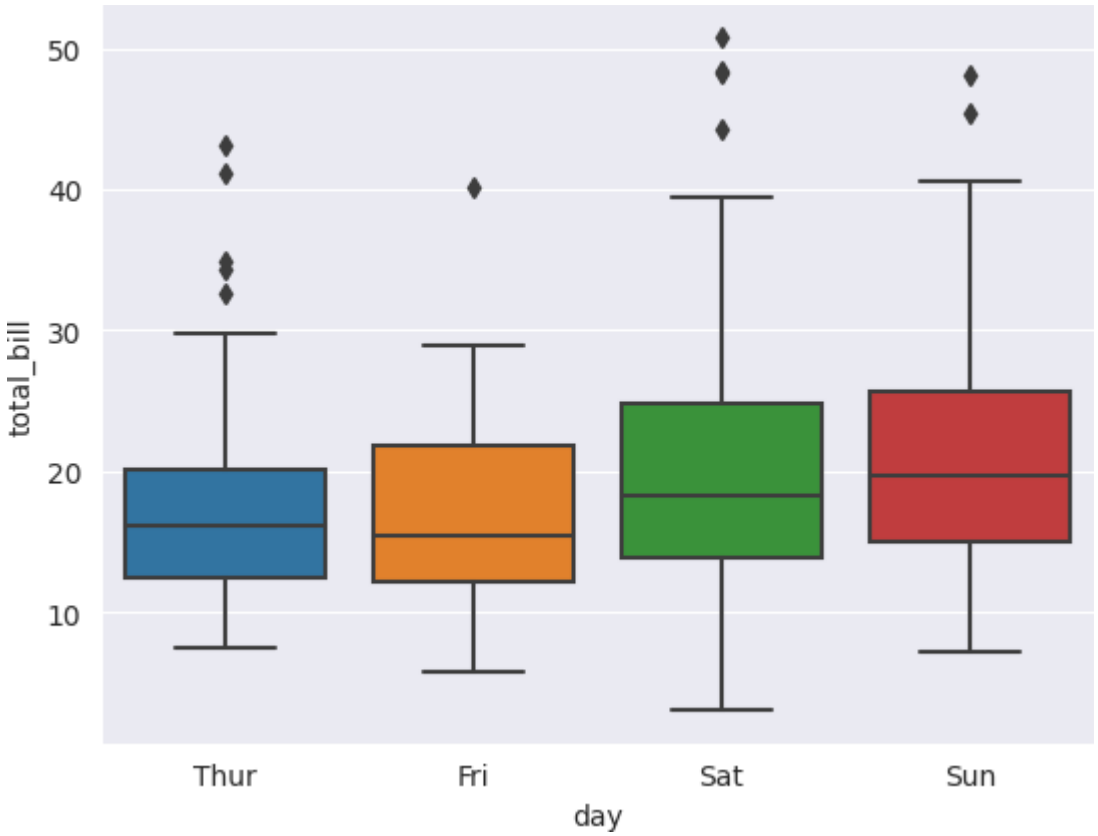
**3. Seaborn: Estetika yang Ditingkatkan**

Seaborn membangun di atas Matplotlib dan menyediakan antarmuka yang lebih tinggi untuk menggambar grafik statistik yang menarik. Ini juga memungkinkan kamu untuk membuat tema yang lebih kompleks dengan lebih mudah.

Contoh plot dengan Seaborn:

```
import seaborn as sns

tips = sns.load_dataset("tips")
sns.boxplot(x="day", y="total_bill", data=tips)
plt.show()
```

Output:

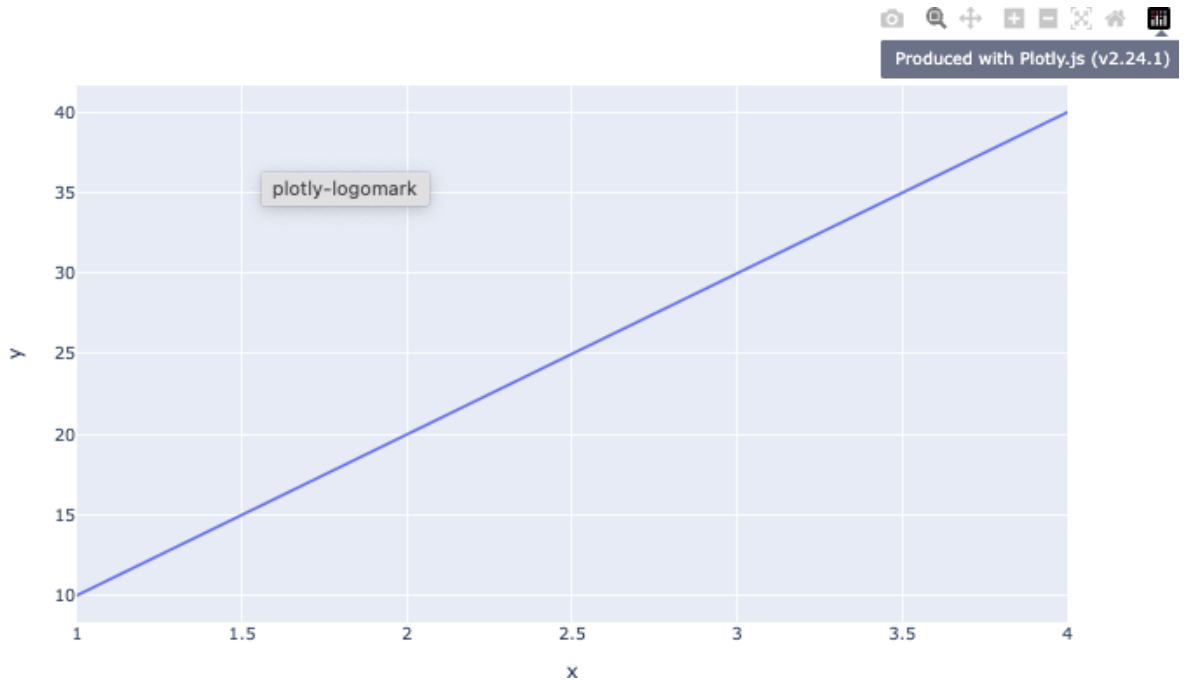



**4. Plotly: Plot Interaktif**

Plotly adalah perpustakaan yang menyediakan plot interaktif. Kamu bisa menggunakannya bersama Matplotlib untuk membuat plot yang dapat di-zoom, di-drag, dan lainnya.

```
import plotly.express as px

fig = px.line(x=x, y=y)
fig.show()
```

Output:

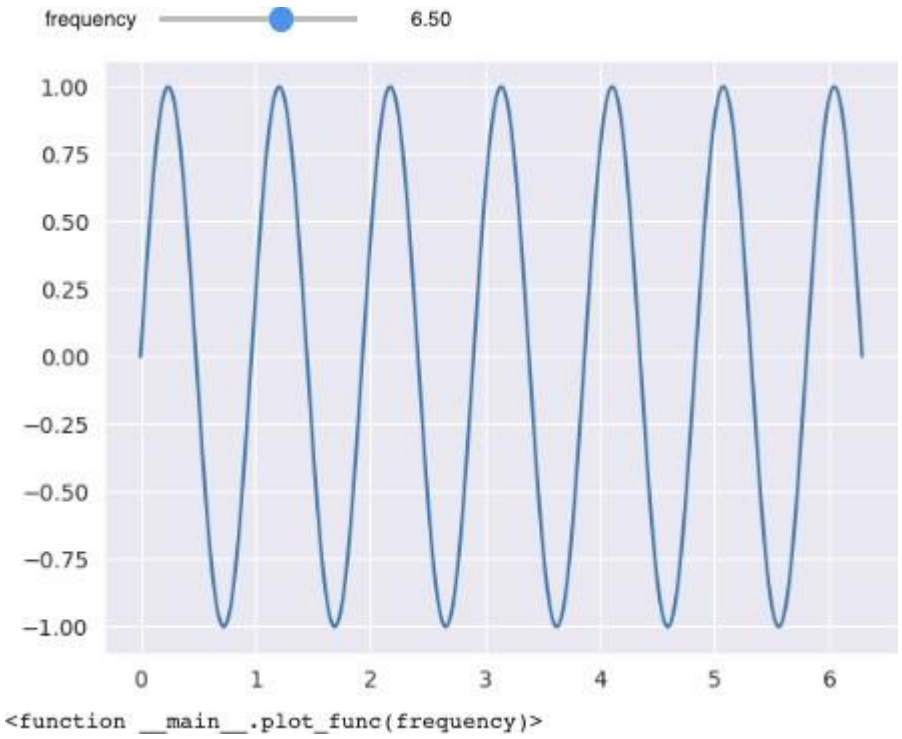



**5. Menggunakan Widgets untuk Interaksi**

Kamu juga dapat menggunakan widgets untuk menambahkan interaksi ke plot kamu. Misalnya, slider yang mengubah parameter dalam plot.

```
from ipywidgets import interact

def plot_func(frequency):
    x = np.linspace(0, 2*np.pi, 1000)
    y = np.sin(frequency * x)
    plt.plot(x, y)
    plt.show()

interact(plot_func, frequency=(1, 10, 0.1))
```

Output:




## 5.3. Sumber Daya Tambahan

### 5.3.1. Dokumentasi Resmi Matplotlib

Dokumentasi adalah sumber daya pertama yang harus kamu kunjungi saat membutuhkan bantuan. Dokumentasi Matplotlib adalah tempat yang bagus untuk memulai:
https://matplotlib.org/stable/contents.html
https://matplotlib.org/stable/tutorials/introductory/sample_plots.html
https://matplotlib.org/stable/gallery/index.html

### 5.3.2. Tutorial dan Kursus Online

Ada banyak tutorial dan kursus online yang dapat membantu kamu memahami berbagai aspek Matplotlib dan visualisasi data:
https://www.datacamp.com/courses/introduction-to-data-visualization-with-matplotlib
https://www.coursera.org/specializations/data-science-python

### 5.3.3. Buku

Buku adalah sumber daya yang bagus untuk belajar dalam-dalam, dan beberapa buku tentang Matplotlib dan visualisasi data adalah:

- "Storytelling with Data: A Guide to Visualizing Information for Impact" oleh Cole Nussbaumer Knaflic.

Deskripsi singkat: Buku ini menekankan pada bagaimana menceritakan kisah melalui visualisasi data dan bagaimana memilih visualisasi yang tepat untuk data Anda.

- "The Visual Display of Quantitative Information" oleh Edward R. Tufte.
  Deskripsi singkat: Salah satu klasik dalam bidang visualisasi data, buku ini mengeksplorasi prinsip-prinsip desain grafis yang baik dan bagaimana menerapkannya pada grafik statistik.

- "D3.js in Action" oleh Elijah Meeks.
  Deskripsi singkat: Untuk mereka yang tertarik dengan visualisasi data berbasis web, D3.js adalah salah satu pustaka yang paling populer. Buku ini memberikan panduan mendalam tentang cara menggunakan D3.js.

- "Interactive Data Visualization for the Web" oleh Scott Murray.
  Deskripsi singkat: Buku lain tentang D3.js, tetapi dengan pendekatan yang lebih praktis dan berorientasi pada pemula.

- "Data Visualization: A Practical Introduction" oleh Kieran Healy.
  Deskripsi singkat: Buku ini menawarkan pendekatan praktis untuk visualisasi data dengan menggunakan R dan ggplot2.

### 5.3.4. Forum dan Komunitas

Komunitas pengguna dan pengembang Matplotlib yang aktif dapat menjadi sumber bantuan yang sangat berharga. Beberapa tempat untuk mencari bantuan adalah:

- Stack Overflow
- ChatGPT
- Datasans 🙂

Ingat, visualisasi data adalah bidang yang kompleks dan dinamis. Terus eksplorasi, belajar, dan praktik adalah kunci untuk menjadi ahli dalam bidang ini. Jangan ragu untuk bereksperimen, bertanya, dan berkolaborasi dengan orang lain. Selamat berpetualang dalam dunia visualisasi data!

# Terimakasih