shadowfox-beginner

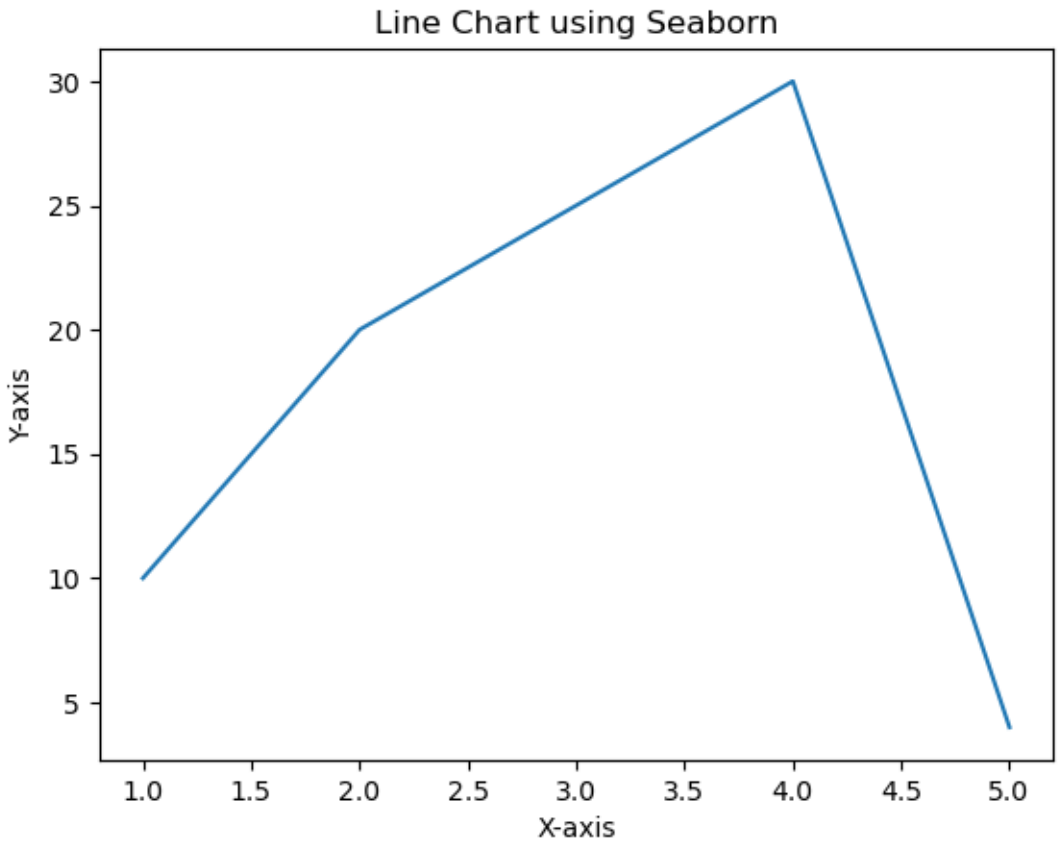
August 30, 2025

[19]:
```
#This line imports the pandas library and renames it as pd for convenience.
#Pandas is a data analysis and manipulation library used to work with␣
 ↪structured data like Excel, CSV, SQL, etc.
#It provides powerful tools like DataFrame (2D table) and Series (1D array) to␣
 ↪store and process data.
import pandas as pd
df = pd.read_excel("IPL sample data.xlsx")
print(df.head())
```

```
    Pick                                                  Y->  Clean Pick  \
0  Throw                                                  Y->  Good Throw
1   Runs  "+" stands for runs saved "-" stands for runs …         NaN
2    NaN                                                  NaN         NaN
3    NaN                                          Match No.     Innings
4    NaN                                            IPL2367           1

              N->         Fumble        C->            Catch  DC->  \
0             N->      Bad throw       DH->        Dirct Hit  RO->
1            NaN            NaN        NaN              NaN   NaN
2            NaN            NaN        NaN              NaN   NaN
3          Teams    Player Name  BallCount         Position  Pick
4  Delhi Capitals  Rilee russouw        0.1  Short mid wicket     n

  Dropped Catch   S->       Stumping Unnamed: 11          Unnamed: 12
0       Run Out  MR->  Missed Runout         NaN                  NaN
1           NaN   NaN            NaN         NaN                  NaN
2           NaN   NaN            NaN         NaN                  NaN
3         Throw  Runs      Overcount       Venue              Stadium
4           NaN     1              1       Delhi  Arun Jaitly Stadium
```

[23]:
```
# Imports the Seaborn library for creating beautiful and easy statistical plots.
#Imports the NumPy library to create numerical arrays for the data.
# Imports the pandas library to create and manage structured tabular data using␣
 ↪DataFrame.
# Creates a NumPy array x which stores the values for the X-axis.
import seaborn as sns
import numpy as np
```

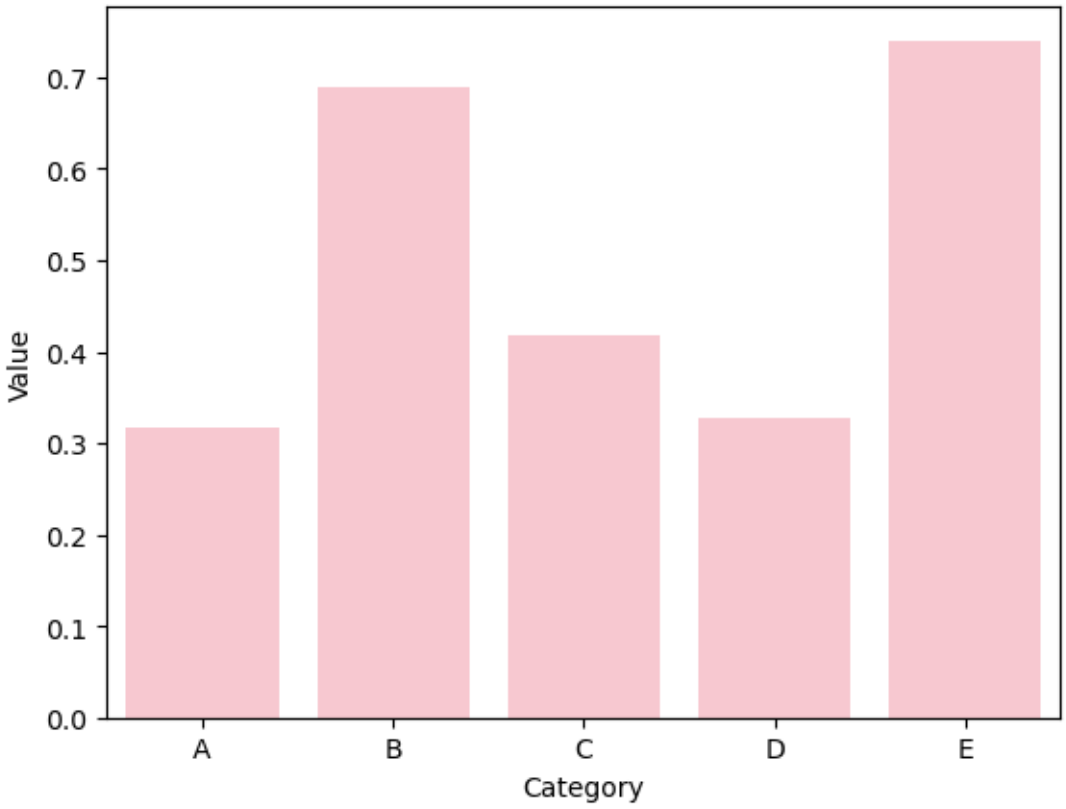
```
import pandas as pd

x = np.array([1, 2, 3, 4, 5])
y = np.array([10, 20, 25, 30, 4])
df = pd.DataFrame({
    'X': x,
    'Y': y
})
sns.lineplot(x='X', y='Y', data=df)
plt.title('Line Chart using Seaborn')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



[27]:
```
# Creates a NumPy array x containing categorical labels ('A', 'B', 'C', 'D',␣
↪'E') - these will be used on the X-axis.
# Creates a NumPy array y of 5 random float values between 0 and 1, which will␣
↪be used on the Y-axis.
#Converts the x and y arrays into a pandas DataFrame with two columns:
#'Category' (for labels)
```

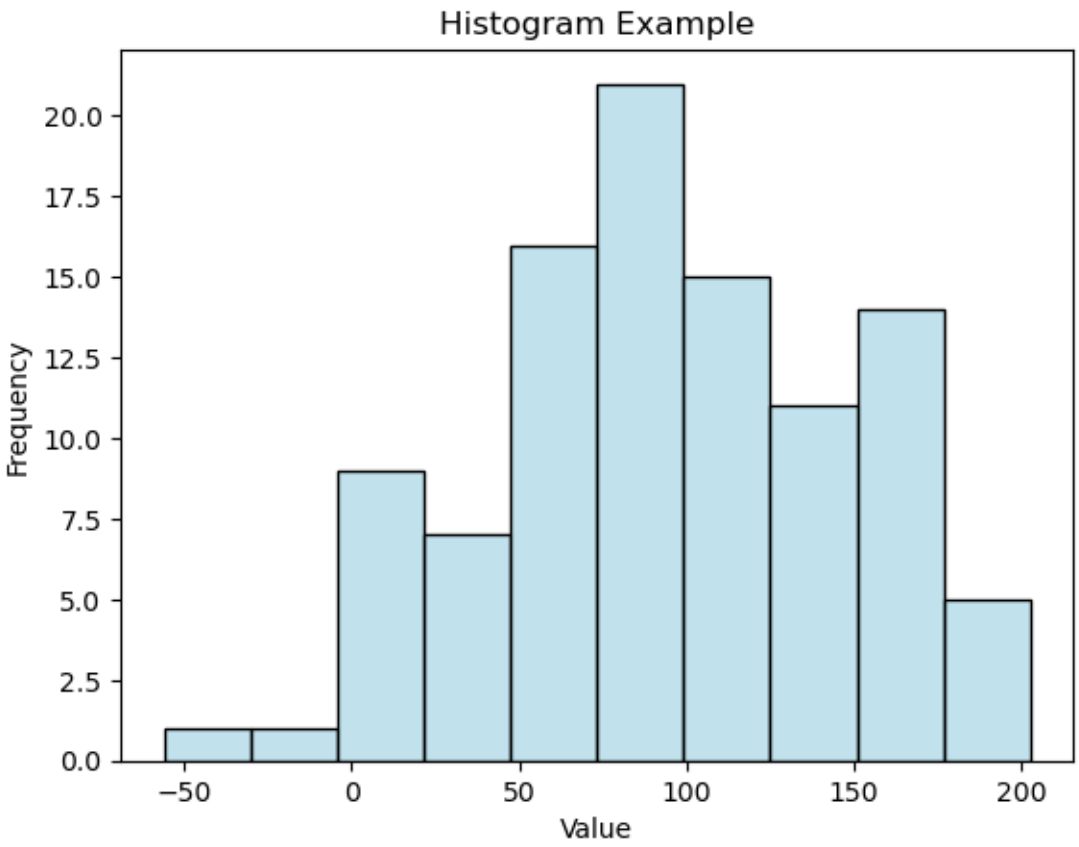
```
#'Value' (for numerical values)
x = np.array(['A', 'B', 'C', 'D', 'E'])
y = np.random.rand(5)
df = pd.DataFrame({'Category': x, 'Value': y})
sns.barplot(data=df, x='Category', y='Value', color='pink')
```

[27]: <Axes: xlabel='Category', ylabel='Value'>

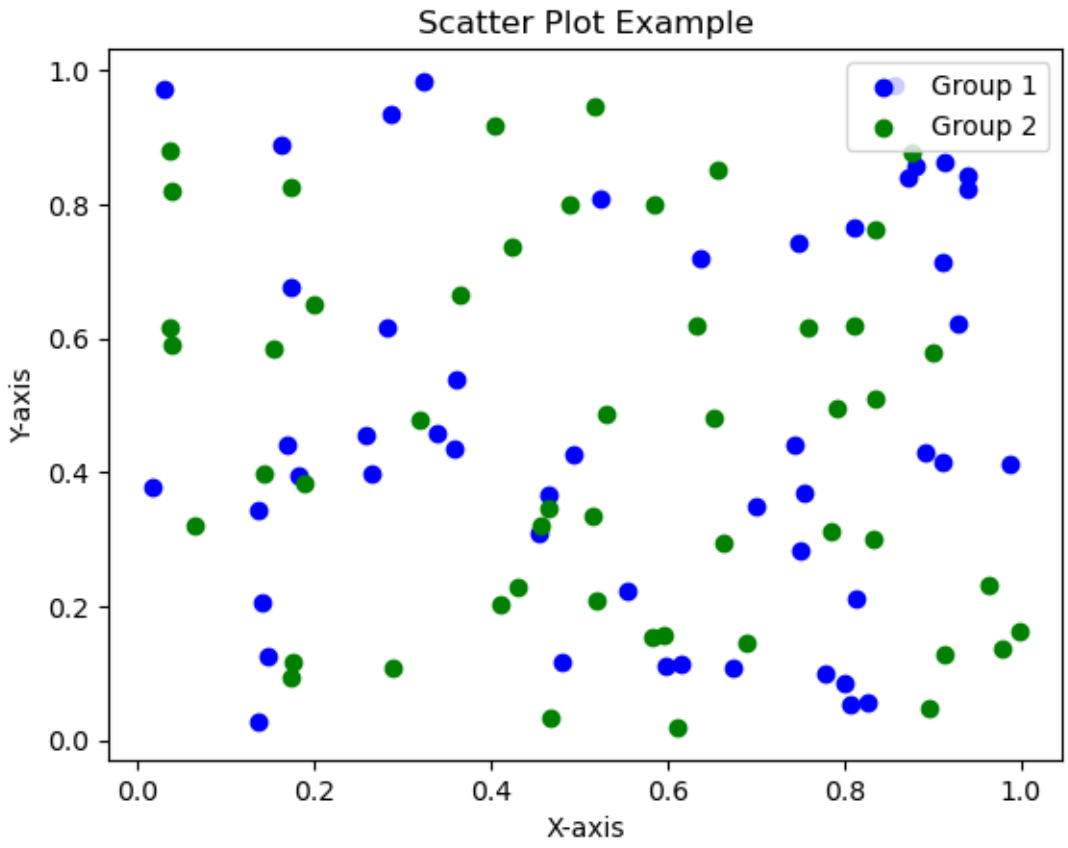


[29]:
```
#A histogram is used to display the distribution of continuous numerical data.
#Seaborn provides the **histplot()** function to create histograms easily.
#It divides data into bins (intervals) and shows the frequency of data points␣
 ↪in each bin.
#Helps to understand the shape, spread, and central tendency of the data.
data = np.random.normal(100, 50, 100)
df = pd.DataFrame({'Value': data})
ax = sns.histplot(data=df, x='Value', bins=10, color='lightblue',␣
 ↪edgecolor='black')
ax.set_title('Histogram Example')
ax.set_xlabel('Value')
ax.set_ylabel('Frequency')
```

[29]: Text(0, 0.5, 'Frequency')

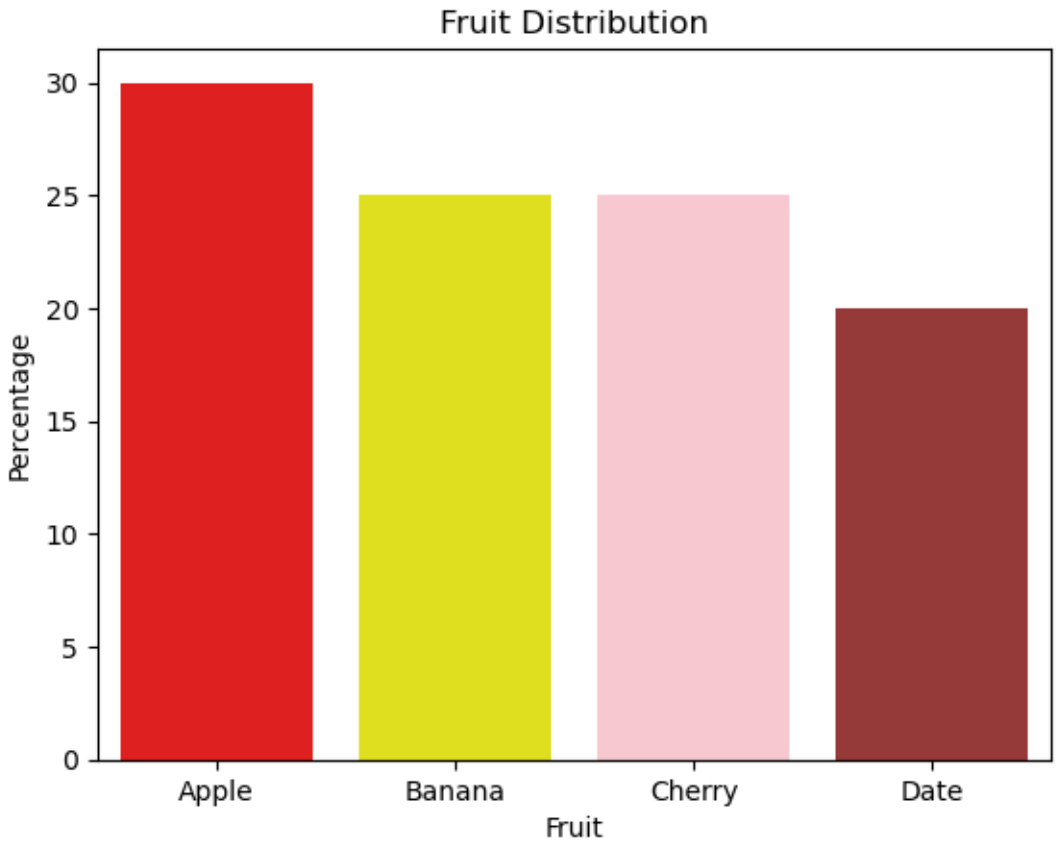


[20]:
```
x1 = np.random.rand(50)
y1 = np.random.rand(50)
x2 = np.random.rand(50)
y2 = np.random.rand(50)

plt.scatter(x1, y1, c='blue', label='Group 1')
plt.scatter(x2, y2, c='green', label='Group 2')
plt.title('Scatter Plot Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
```



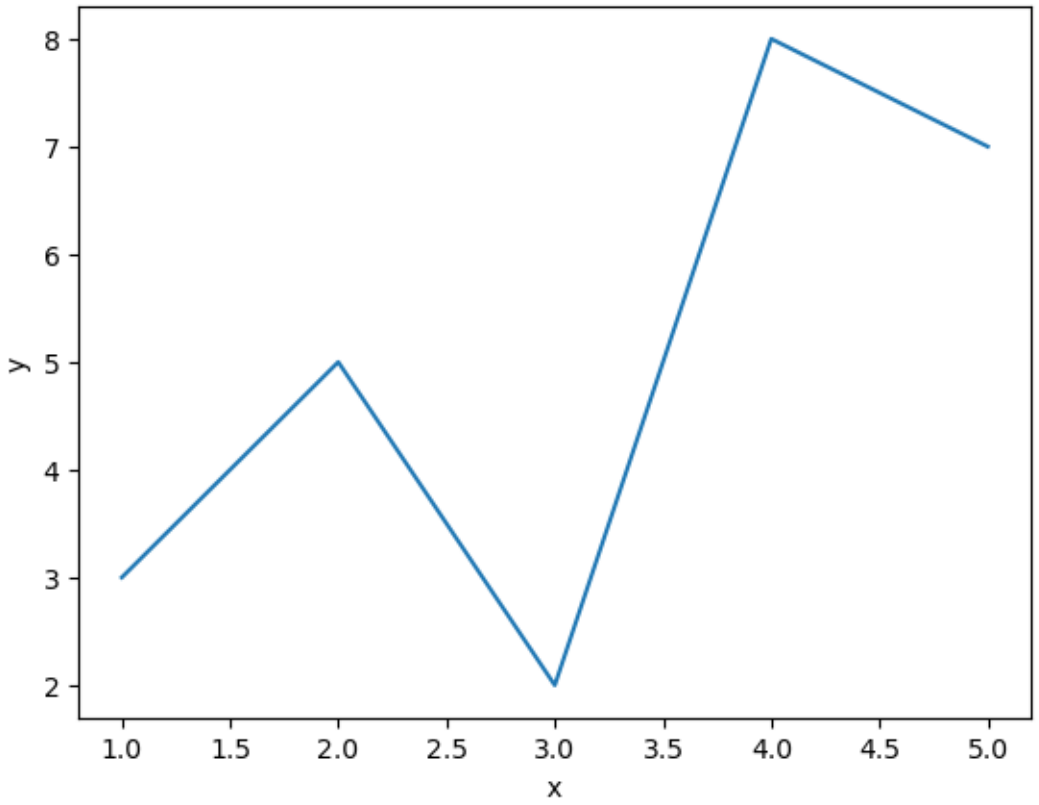
[31]:
```
#Seaborn's barplot() is used to create categorical bar charts that show the␣
 ↪relationship between a category and a numerical value.
#A DataFrame is created with fruit names and their corresponding percentage␣
 ↪values.
#The hue parameter is set to the same categorical column ('Fruit') to assign␣
 ↪individual colors to each bar using the palette.
labels = ['Apple', 'Banana', 'Cherry', 'Date']
sizes = [30, 25, 25, 20]
colors = ['red', 'yellow', 'pink', 'brown']
df = pd.DataFrame({'Fruit': labels, 'Percentage': sizes})
ax = sns.barplot(data=df, x='Fruit', y='Percentage', hue='Fruit',␣
 ↪palette=colors, legend=False)
ax.set_title('Fruit Distribution')
```

[31]:
```
Text(0.5, 1.0, 'Fruit Distribution')
```

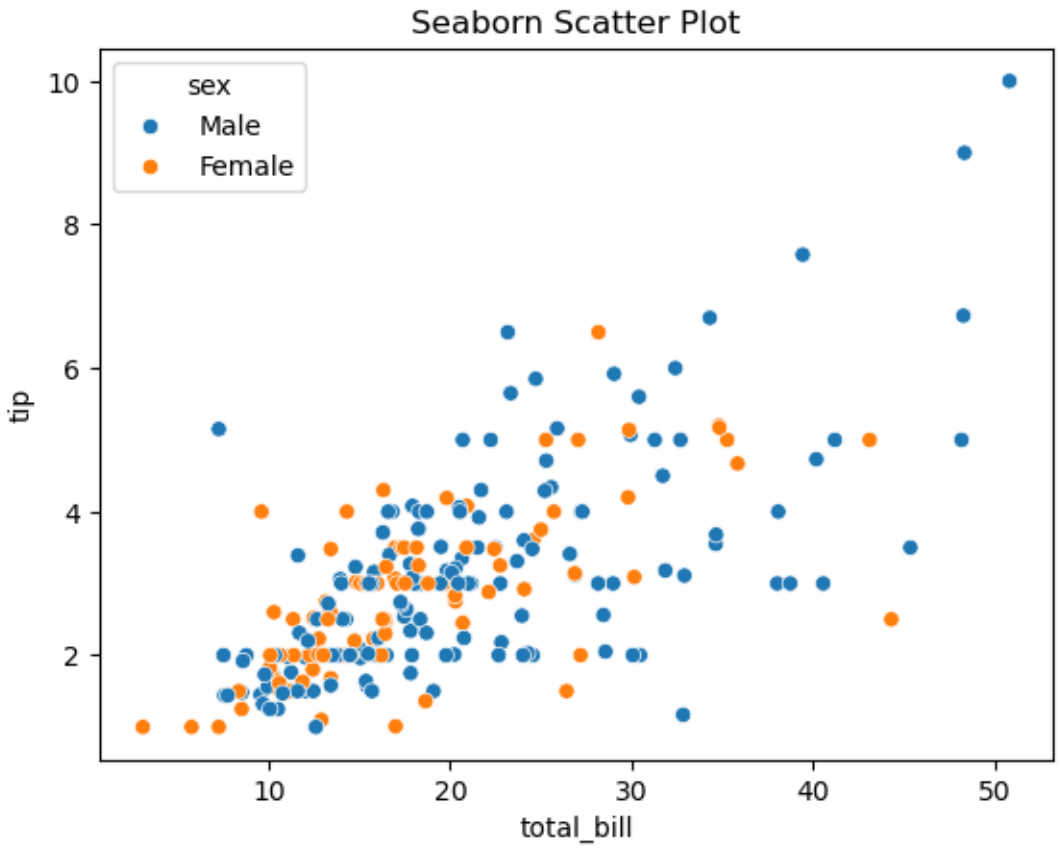


[33]:
```
#seaborn's lineplot() is used to create a line chart that shows the␣
 ↪relationship between two continuous variables.
#A pandas DataFrame is created with two columns: 'x' (independent variable) and␣
 ↪'y' (dependent variable).
#The lineplot() function takes the DataFrame as input and plots 'x' on the␣
 ↪X-axis and 'y' on the Y-axis.
df = pd.DataFrame({
    'x': [1, 2, 3, 4, 5],
    'y': [3, 5, 2, 8, 7]
})
sns.lineplot(data=df, x='x', y='y')
```

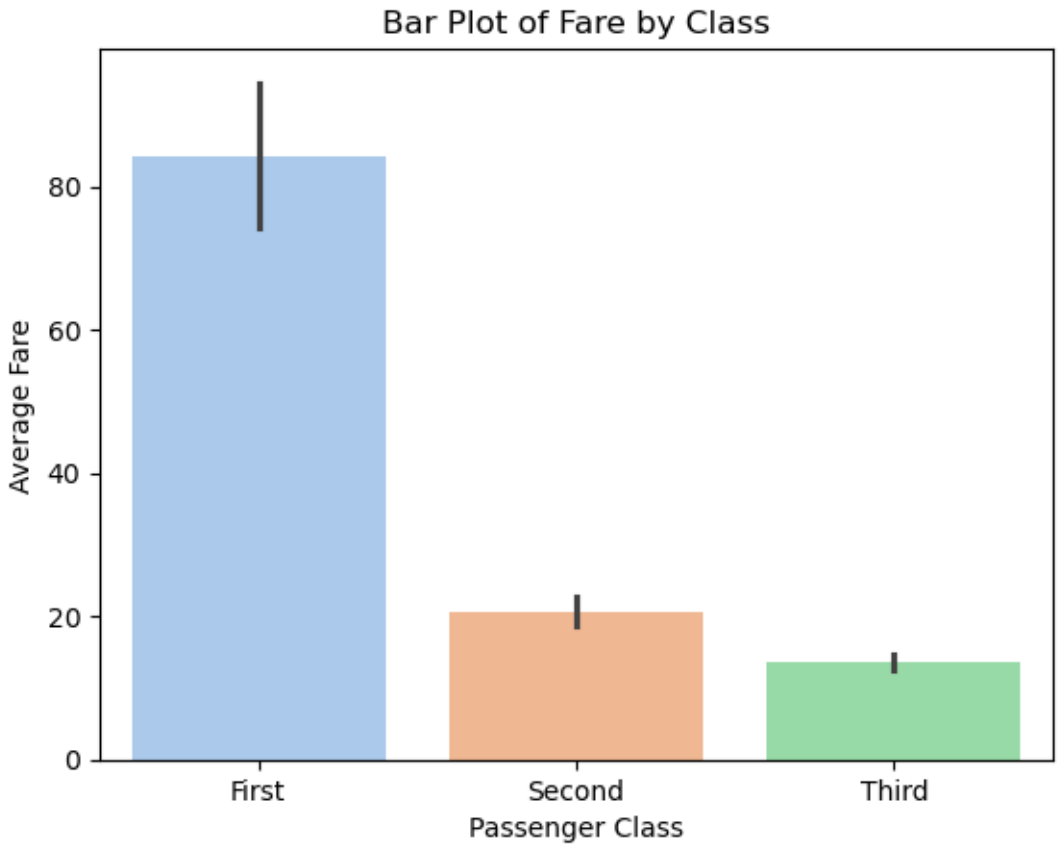
[33]:
```
<Axes: xlabel='x', ylabel='y'>
```



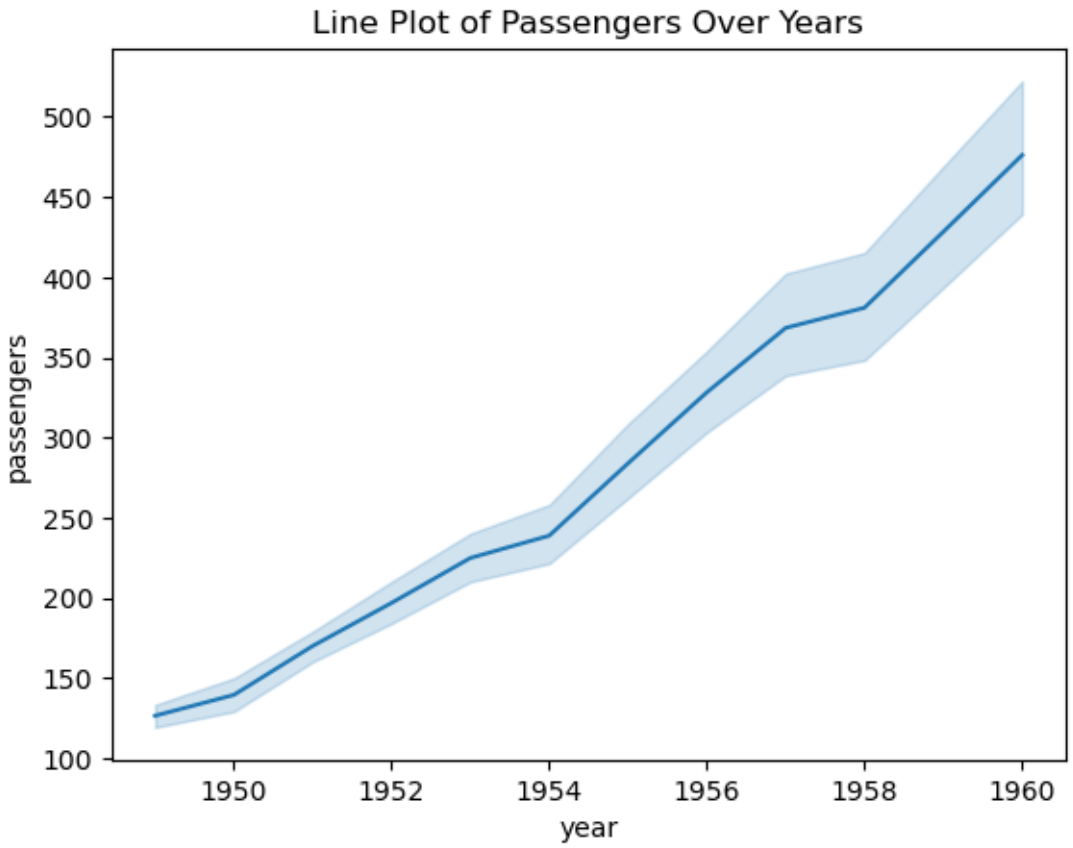
[35]:
```
#Seaborn's scatterplot() is used to create a scatter plot, which shows the␣
 ↪relationship between two continuous variables.
#The built-in "tips" dataset is loaded using sns.load_dataset(), which contains␣
 ↪restaurant bill and tip data.
#The x and y parameters are set to 'total_bill' and 'tip', respectively, to␣
 ↪plot each data point.
import seaborn as sns
df = sns.load_dataset("tips")
sns.scatterplot(x='total_bill', y='tip', hue='sex', data=df)
plt.title('Seaborn Scatter Plot')
plt.show()
```



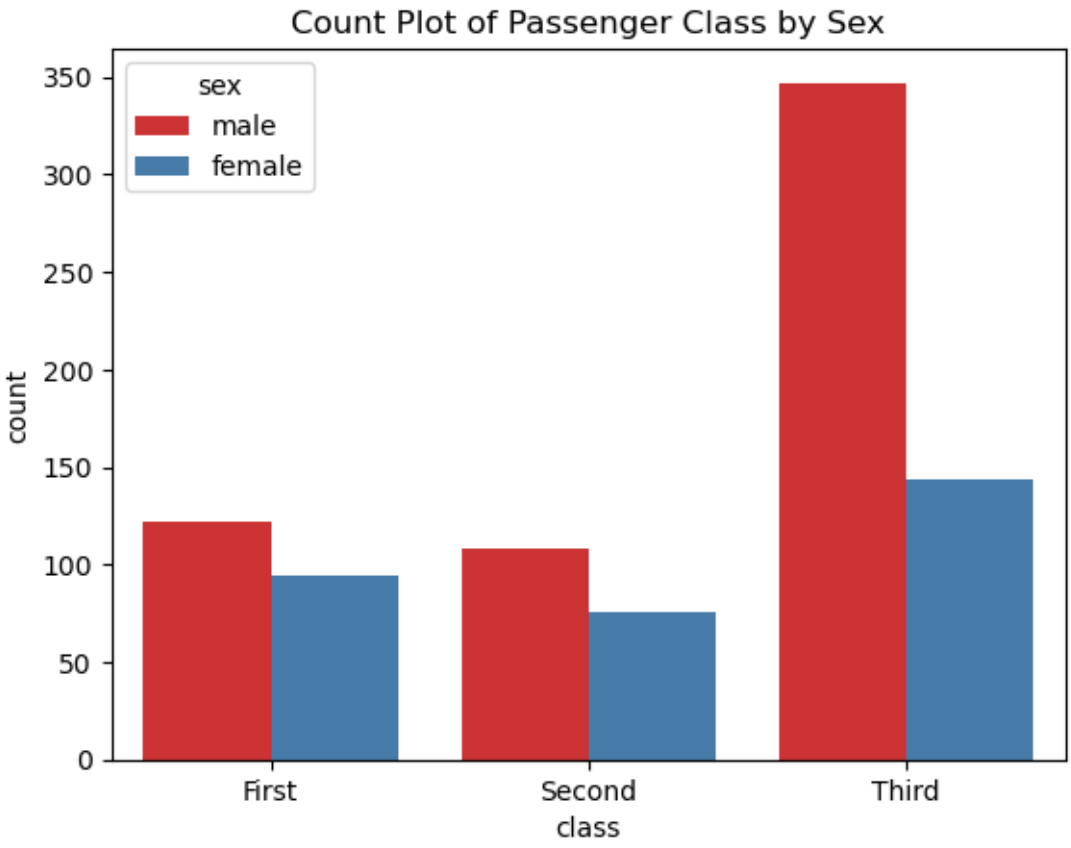
[119]:
```
# Seaborn's barplot() is used to create a bar chart that shows the average fare␣
 ↪paid by passengers in each class from the Titanic dataset.
# The built-in Titanic dataset is loaded using sns.load_dataset("titanic").
# The x and y parameters are set to 'class' and 'fare', which plot passenger␣
 ↪classes on the X-axis and their corresponding average fares on the Y-axis.
# The hue='class' parameter assigns different colors to each bar based on␣
 ↪class, while palette='pastel' applies soft, pleasant colors.
import seaborn as sns
df = sns.load_dataset("titanic")
sns.barplot(x='class', y='fare', hue='class', data=df, palette='pastel',␣
 ↪legend=False)
plt.title('Bar Plot of Fare by Class')
plt.xlabel('Passenger Class')
plt.ylabel('Average Fare')
plt.show()
```



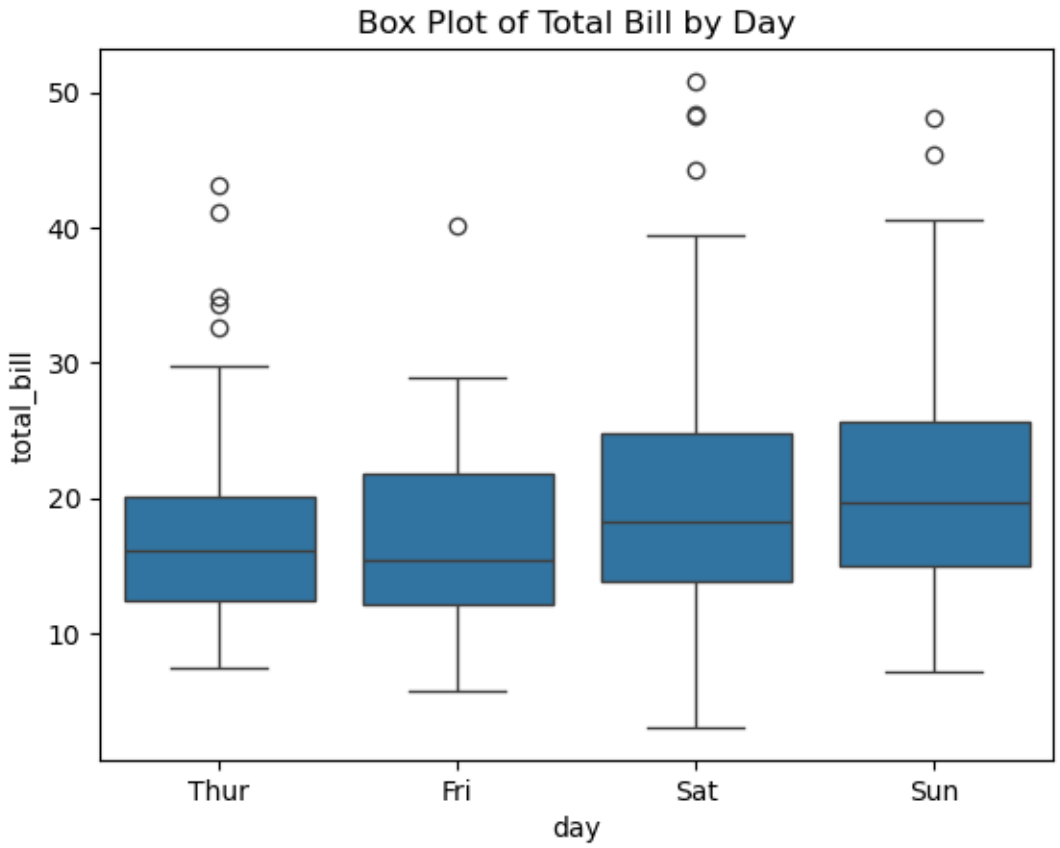
[57]:
```
# Seaborn's lineplot() is used to visualize trends over time or continuous␣
 ↪values.
# The built-in 'flights' dataset is loaded using sns.load_dataset("flights"),
# which contains data about monthly airline passengers over years.
# The x parameter is set to 'year' and the y to 'passengers',
# showing how passenger numbers changed over the years.
df = sns.load_dataset("flights")
sns.lineplot(x='year', y='passengers', data=df)
plt.title('Line Plot of Passengers Over Years')
plt.show()
```



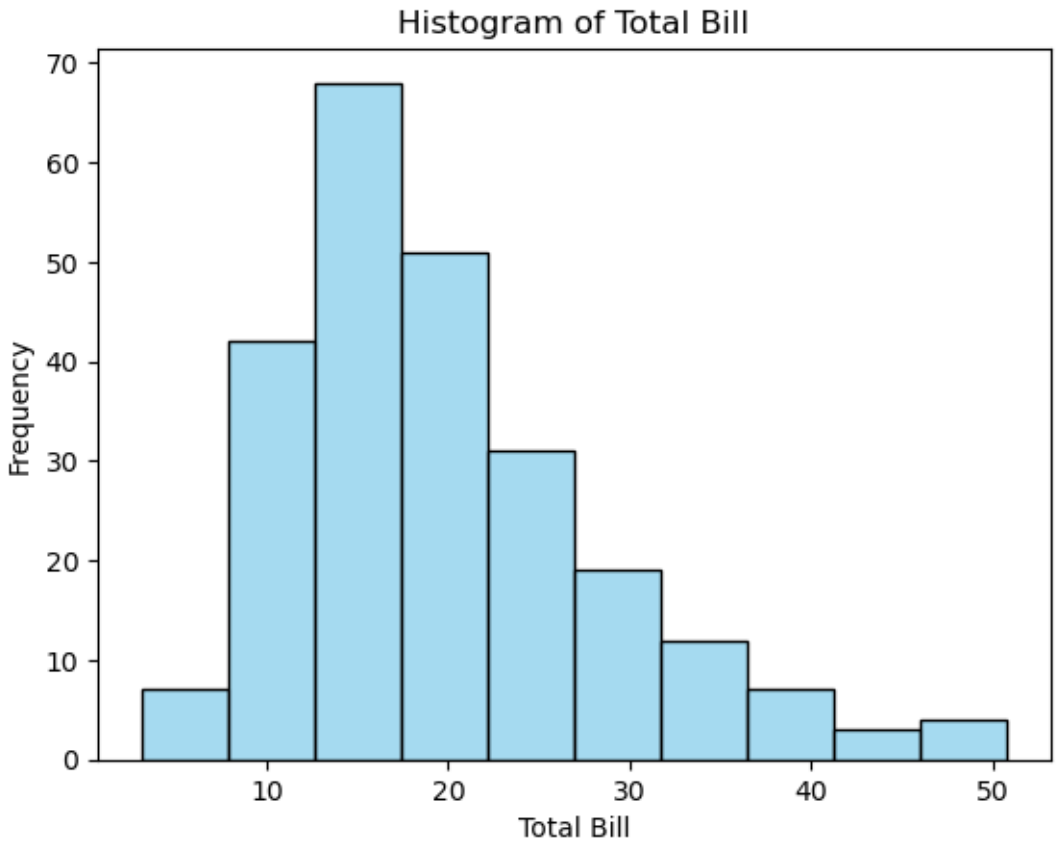
[59]:
```
#Seaborn's countplot() is used to display the count of observations in each␣
 ↪categorical bin using bars.
#The titanic dataset is loaded using sns.load_dataset("titanic"), which␣
 ↪contains passenger data.
#The x='class' parameter sets the passenger class as the category to group by.
#The hue='sex' parameter further separates bars based on passenger sex (male/
 ↪female).
#The palette='Set1' applies a predefined color scheme for better visual␣
 ↪distinction.
df = sns.load_dataset("titanic")
sns.countplot(x='class', hue='sex', palette='Set1', data=df)
plt.title('Count Plot of Passenger Class by Sex')
plt.show()
```



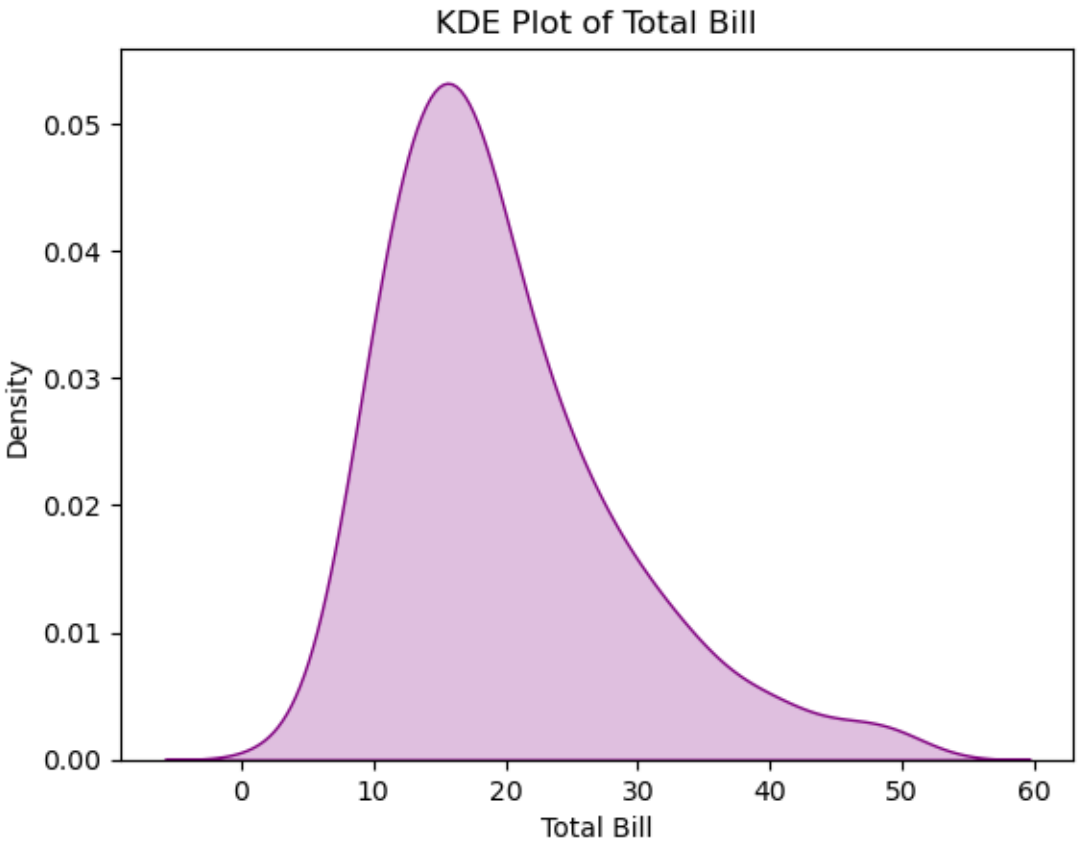
[61]:
```
#Seaborn's boxplot() is used to visualize the distribution, spread, and␣
 ↪outliers of numerical data across different categories.
#The tips dataset is loaded using sns.load_dataset("tips"), which contains data␣
 ↪about restaurant bills, tips, gender, smoking habits, etc.
#The x='day' and y='total_bill' parameters indicate that the plot will compare␣
 ↪total bills for each day of the week.
#This helps identify central tendency, spread, and any outliers in the␣
 ↪total_bill amounts for each day.

df = sns.load_dataset("tips")
sns.boxplot(x='day', y='total_bill', data=df)
plt.title('Box Plot of Total Bill by Day')
plt.show()
```



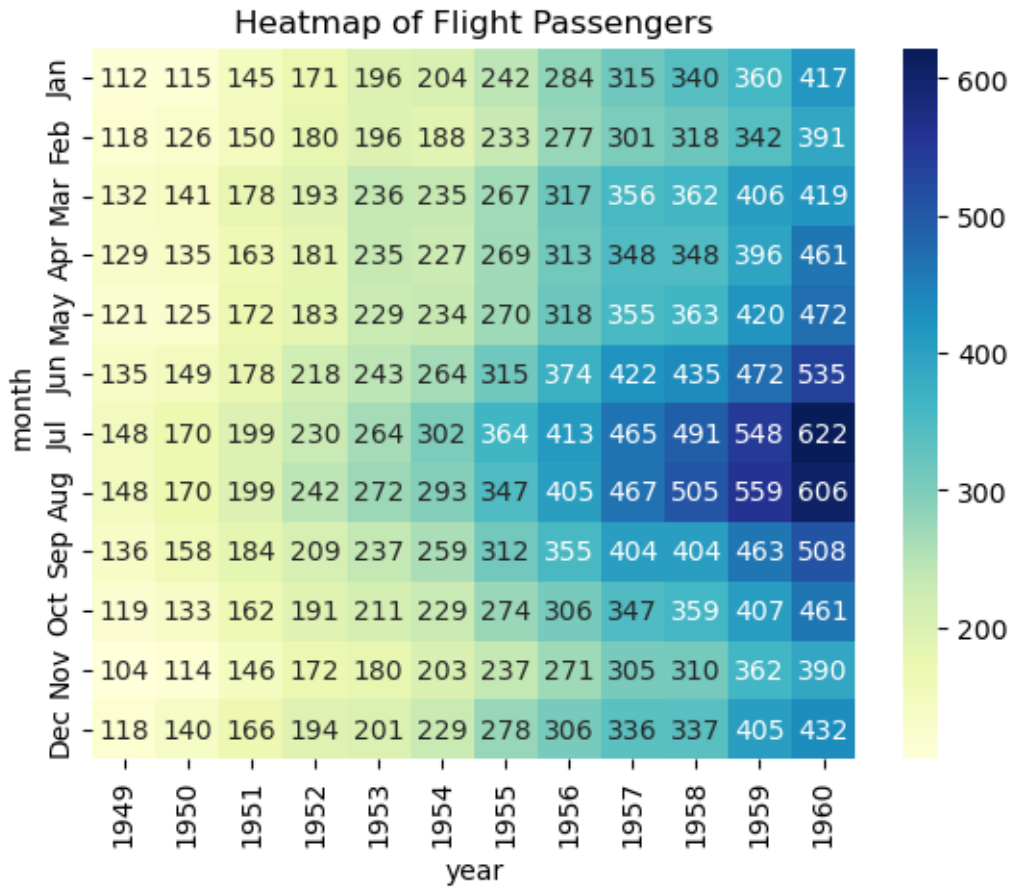
[67]:
```
#The tips dataset is loaded using sns.load_dataset("tips"), which includes␣
 ↪information about restaurant bills and tips.
#sns.histplot() is used to create a histogram of the total_bill column.
#bins=10 divides the total bill values into 10 equal intervals.
#kde=False disables the kernel density estimate line, showing only the␣
 ↪histogram bars.
#color='sky blue' sets the bar color to sky blue.
df = sns.load_dataset("tips")
sns.histplot(df['total_bill'], bins=10, kde=False, color='skyblue',␣
 ↪edgecolor='black')
plt.title('Histogram of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Frequency')
plt.show()
```



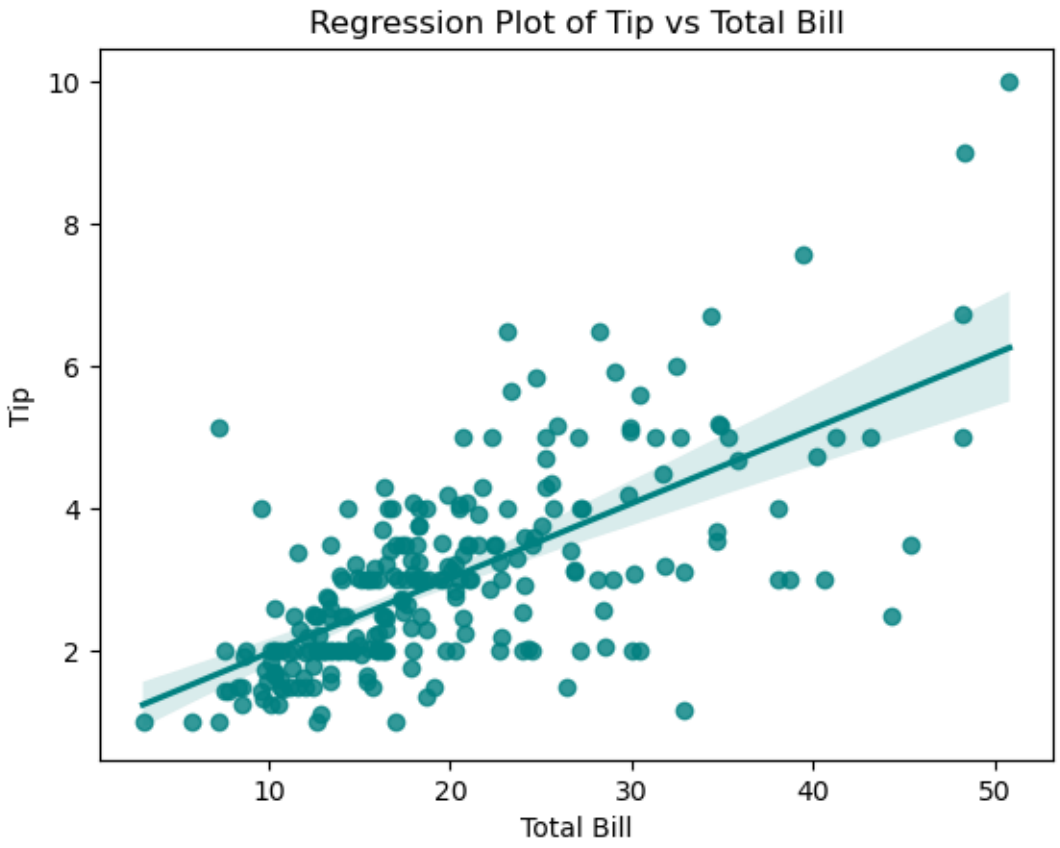
[69]:
```
#The tips dataset is loaded using sns.load_dataset("tips"), which includes␣
 ↪restaurant billing and tipping data.
#sns.kdeplot() is used to draw the Kernel Density Estimate (KDE) plot of the␣
 ↪total_bill column.
#fill=True fills the area under the KDE curve for better visualization.
#color='purple' sets the color of the KDE curve and filled area.
#plt.title(), plt.xlabel(), and plt.ylabel() add a title and axis labels to␣
 ↪describe what the plot represents.
df = sns.load_dataset("tips")
sns.kdeplot(df['total_bill'], fill=True, color='purple')
plt.title('KDE Plot of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Density')
plt.show()
```



[73]:
```
#df = sns.load_dataset("flights"): Loads the flights dataset, which contains␣
 ↪monthly passenger counts for different years.
#df.pivot(index="month", columns="year", values="passengers"): Transforms the␣
 ↪dataset into a pivot table where months are rows, years are columns, and␣
 ↪values are passenger counts.
#sns.heatmap(...): Creates a heatmap where each cell's color represents the␣
 ↪number of passengers.
#data: the pivot table is used as the heatmap input.
#cmap="YlGnBu": applies a yellow-green-blue color gradient.
#annot=True: shows the exact passenger values inside each cell.
#fmt="d": formats annotations as integers.

df = sns.load_dataset("flights")
data = df.pivot(index="month", columns="year", values="passengers")
sns.heatmap(data, cmap="YlGnBu", annot=True, fmt="d")
plt.title("Heatmap of Flight Passengers")
plt.show()
```



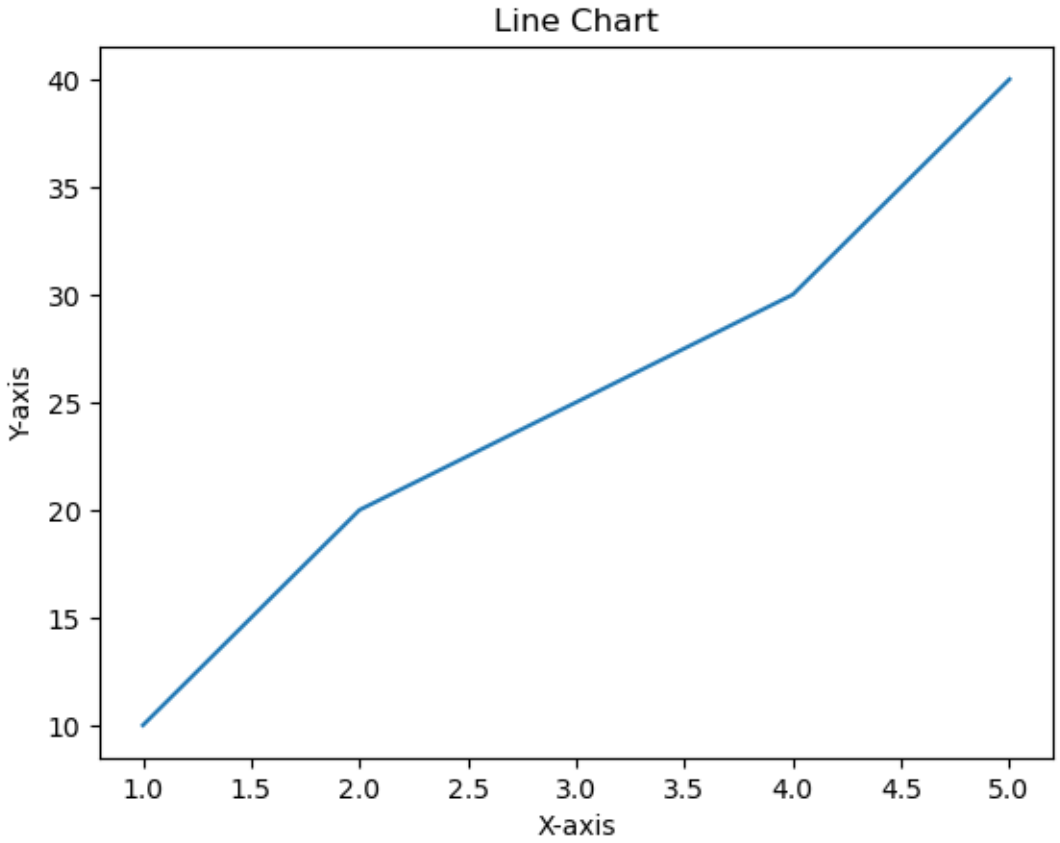
[77]:
```
#df = sns.load_dataset("tips"): Loads the built-in tips dataset, which contains␣
 ↪data about restaurant bills and tips.
#sns.regplot(x='total_bill', y='tip', data=df, color='teal')
#Creates a regression plot to visualize the relationship between the total bill␣
 ↪and the tip amount.
#Plots individual data points and fits a linear regression line.
#color='teal': Sets the color of the plot to teal.
#plt.title('Regression Plot of Tip vs Total Bill'): Adds a title to the plot.
df = sns.load_dataset("tips")
sns.regplot(x='total_bill', y='tip', data=df, color='teal')
plt.title('Regression Plot of Tip vs Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



1 Matplotlib

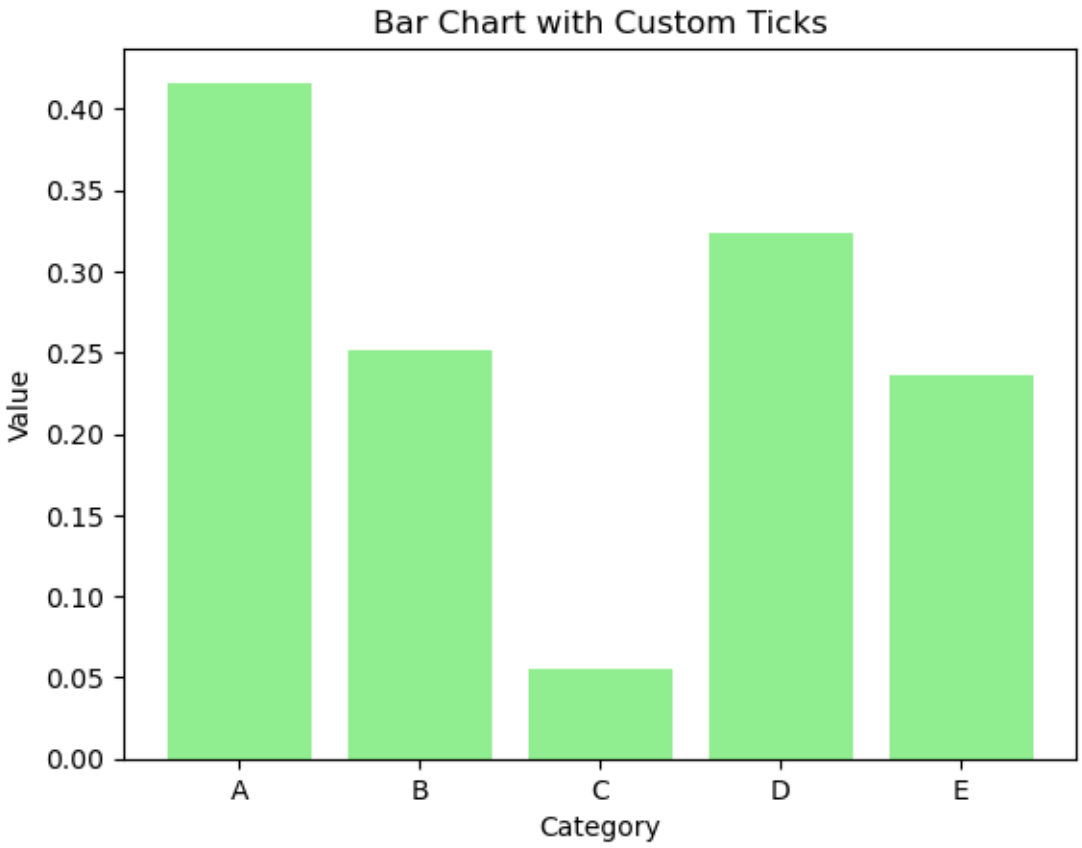
[85]:
```
#import matplotlib.pyplot as plt:
#Imports the Matplotlib plotting library and assigns it the alias plt.
#import numpy as np:
#Imports NumPy, a library used for numerical operations.
#x = np.array([1, 2, 3, 4, 5]):
#Creates a NumPy array x with values [1, 2, 3, 4, 5] representing the x-axis␣
 ↪data.
#y = np.array([10, 20, 25, 30, 40]):
#Creates a NumPy array y with values [10, 20, 25, 30, 40] representing the␣
 ↪y-axis data.
import matplotlib.pyplot as plt
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([10, 20, 25, 30, 40])

plt.plot(x, y)
```

```
plt.title('Line Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

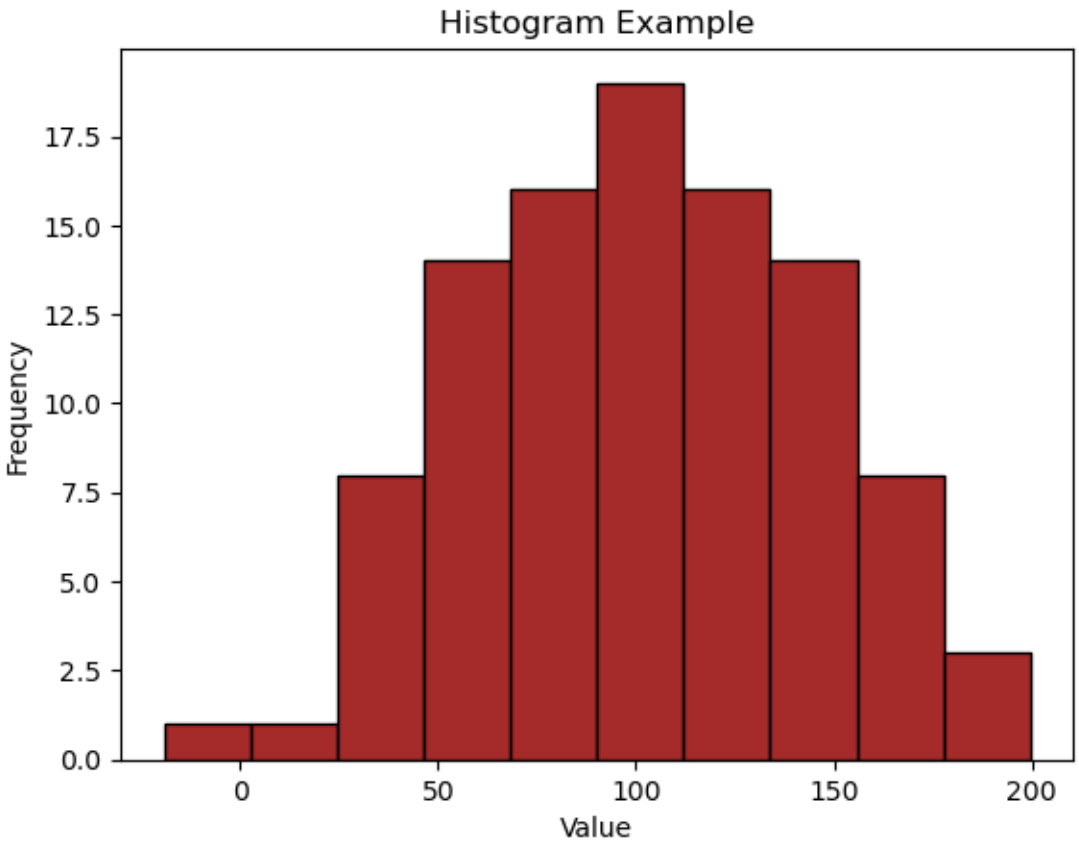


[101]:
```
#Imports Matplotlib for creating visualizations.
#Imports NumPy for generating random numerical data.
#Imports Pandas for handling data in DataFrame format.
#x = np.array(['A', 'B', 'C', 'D', 'E'])
#Defines the category labels for the x-axis.
#y = np.random.rand(5):
#Generates 5 random values between 0 and 1 for the y-axis.
#df = pd.DataFrame({'Category': x, 'Value': y}):
#Creates a DataFrame with two columns: 'Category' and 'Value'.
indices = np.arange(len(df))
plt.bar(indices, df['Value'], color='lightgreen')
plt.xticks(indices, df['Category'])
plt.title('Bar Chart with Custom Ticks')
plt.xlabel('Category')
plt.ylabel('Value')
```

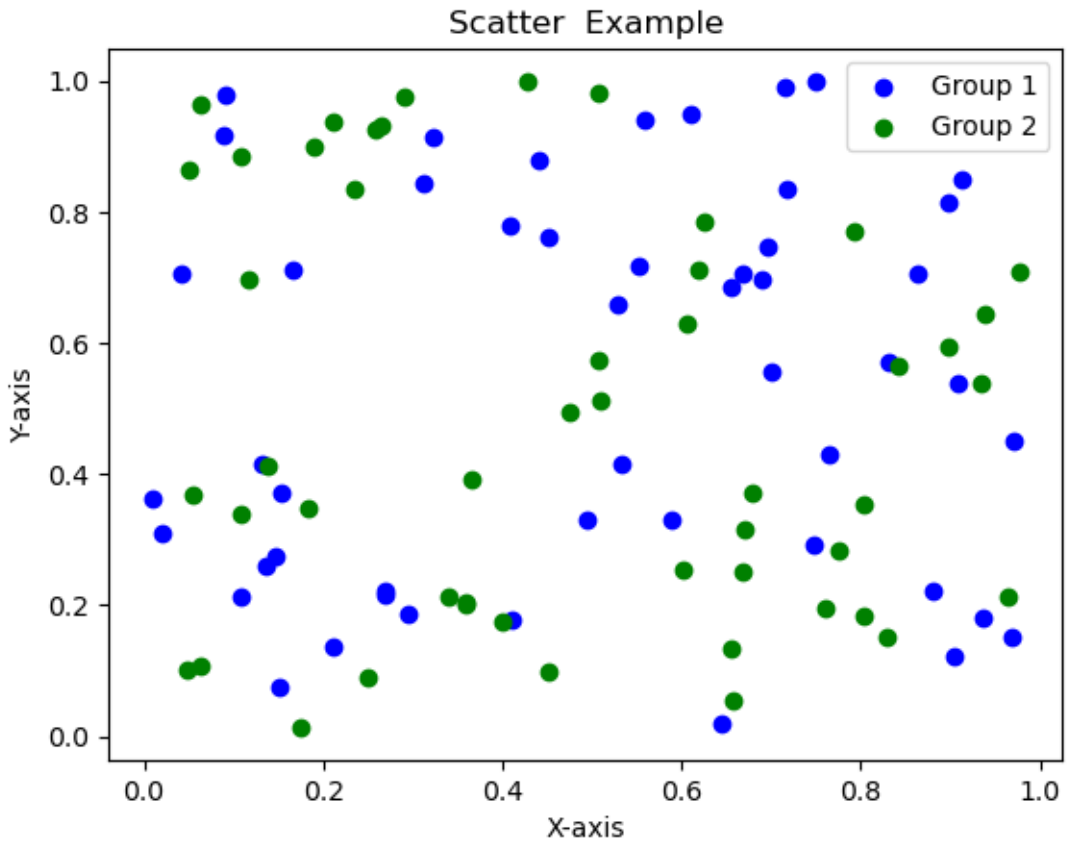
```
plt.show()
```



[109]:
```
#Matplotlib is used for creating visualizations in Python.
#A histogram shows the distribution of numerical data.
#First, data is generated or collected (e.g., random numbers).
#The data is grouped into bins (ranges of values).
#plt.hist() is used to plot the histogram.
#Bars represent how many data points fall into each bin.
data = np.random.normal(100, 50, 100)
df = pd.DataFrame({'Value': data})
# Plot histogram using Matplotlib
plt.hist(df['Value'], bins=10, color='brown', edgecolor='black')
plt.title('Histogram Example')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

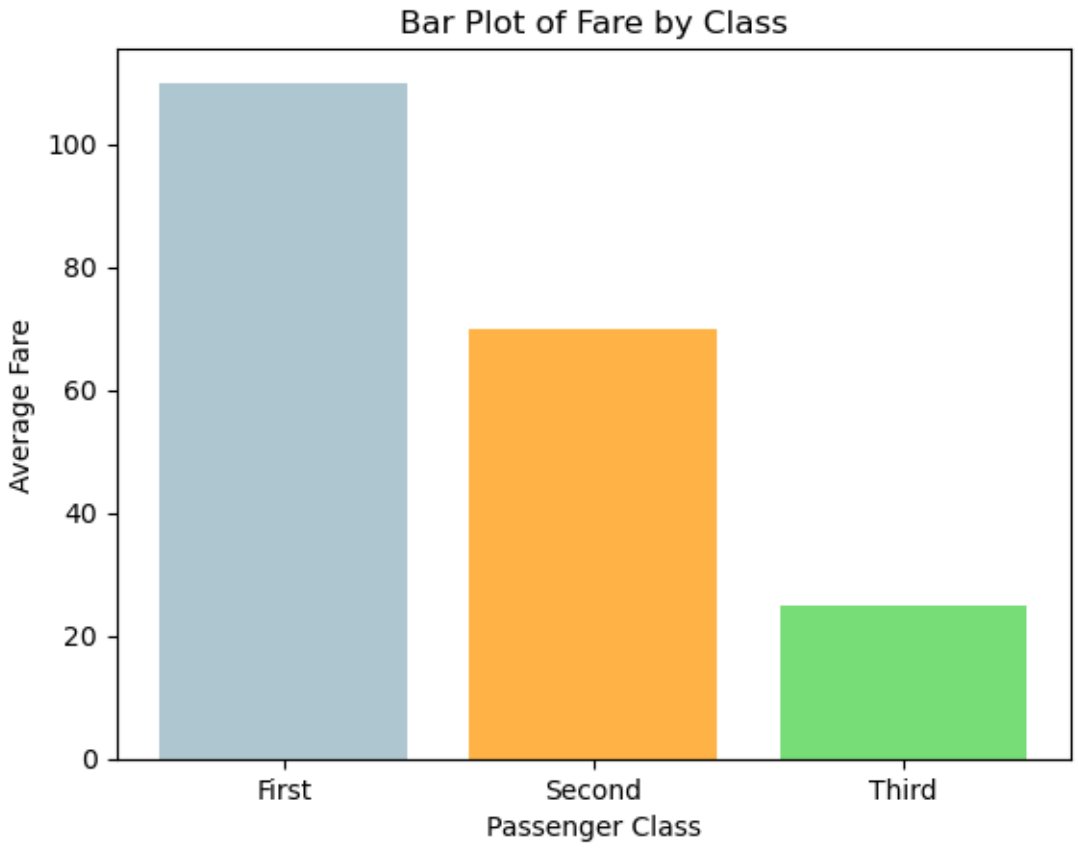


[117]:
```
#Import matplotlib.pyplot and numpy.
#Generate random data using np.random.rand().
#Use plt.scatter() to plot data points.
#Set different colors for different groups.
#Add title, xlabel, and ylabel for context.
#Use plt.legend() to show group labels.
#Use plt.show() to display the plot.
import matplotlib.pyplot as plt
import numpy as np
x1 = np.random.rand(50)
y1 = np.random.rand(50)
x2 = np.random.rand(50)
y2 = np.random.rand(50)
plt.scatter(x1, y1, c='blue', label='Group 1')
plt.scatter(x2, y2, c='green', label='Group 2')
plt.title('Scatter  Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
```



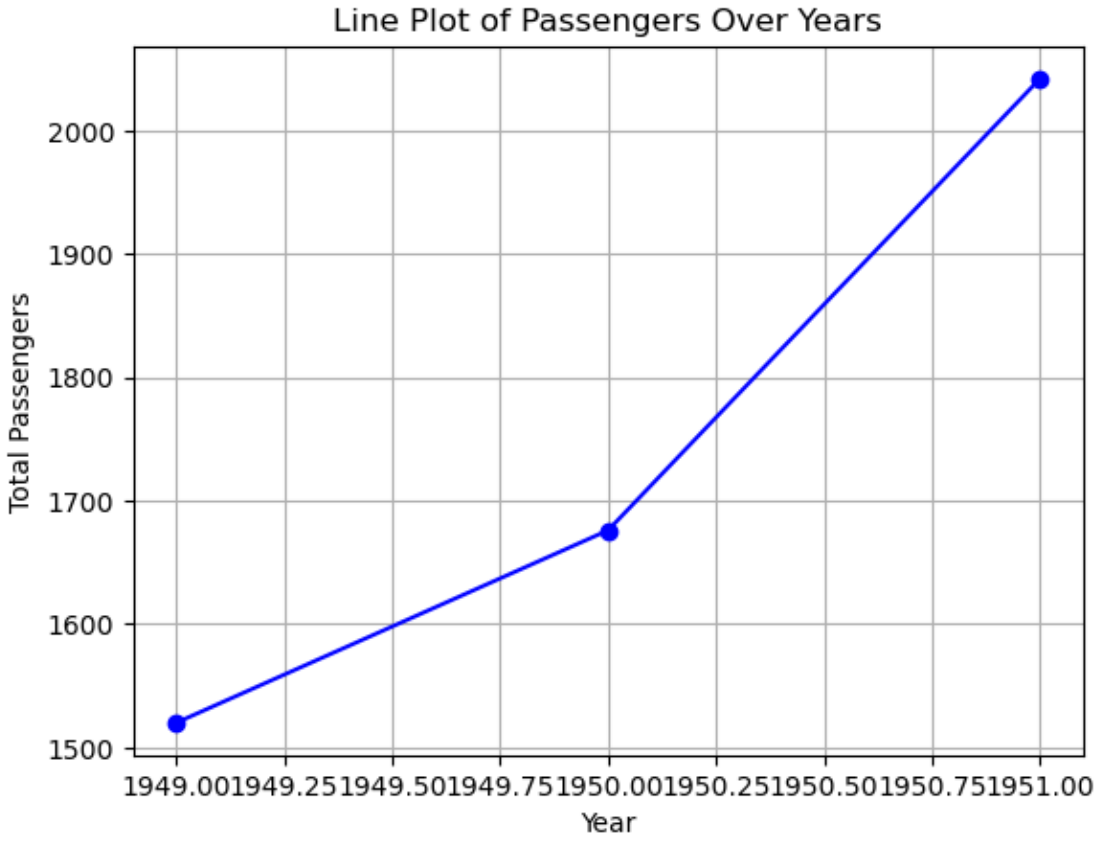
[5]:
```
#A bar plot is used to represent categorical data using rectangular bars, where the height of each bar represents the value associated with that category.
#We manually create a dataset with two columns: class and fare.
#Then we calculate the average fare for each class using groupby() and mean().
#Finally, we plot a bar chart using plt.bar().
import matplotlib.pyplot as plt
import pandas as pd
data = {
    'class': ['First', 'Second', 'Third', 'First', 'Second', 'Third', 'First', 'Second', 'Third'],
    'fare': [100, 70, 30, 120, 60, 25, 110, 80, 20]
}
df = pd.DataFrame(data)
class_fare = df.groupby('class')['fare'].mean().sort_index()
# Plot using Matplotlib
plt.bar(class_fare.index, class_fare.values, color=['#AEC6CF', '#FFB347', '#77DD77'])
plt.title('Bar Plot of Fare by Class')
plt.xlabel('Passenger Class')
plt.ylabel('Average Fare')
```

```
plt.show()
```



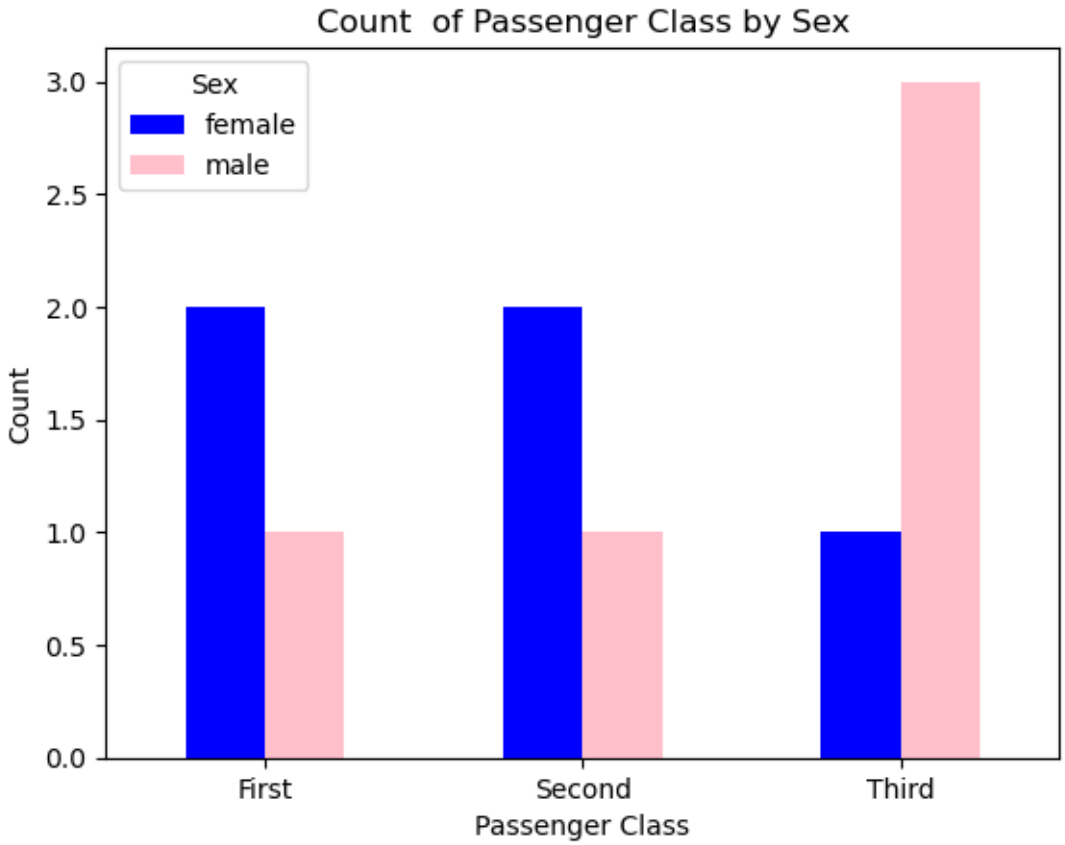
[11]:
```
#A line plot is used to visualize trends over time by connecting data points␣
 ↪with straight lines.
#In this example, we are plotting the number of passengers over the years using␣
 ↪only Matplotlib and Pandas
#The dataset includes the number of airline passengers per month across years.
#We group the data by year and calculate the total passengers per year.
import matplotlib.pyplot as plt
import pandas as pd
data = {
    'year': [1949]*12 + [1950]*12 + [1951]*12,
    'month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
              'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'] * 3,
    'passengers': [
        112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118,
        115, 126, 141, 135, 125, 149, 170, 170, 158, 133, 114, 140,
        145, 150, 178, 163, 172, 178, 199, 199, 184, 162, 146, 166
    ]
}
```

```
df = pd.DataFrame(data)
yearly_data = df.groupby('year')['passengers'].sum()
plt.plot(yearly_data.index, yearly_data.values, marker='o', color='blue')
plt.title('Line Plot of Passengers Over Years')
plt.xlabel('Year')
plt.ylabel('Total Passengers')
plt.grid(True)
plt.show()
```



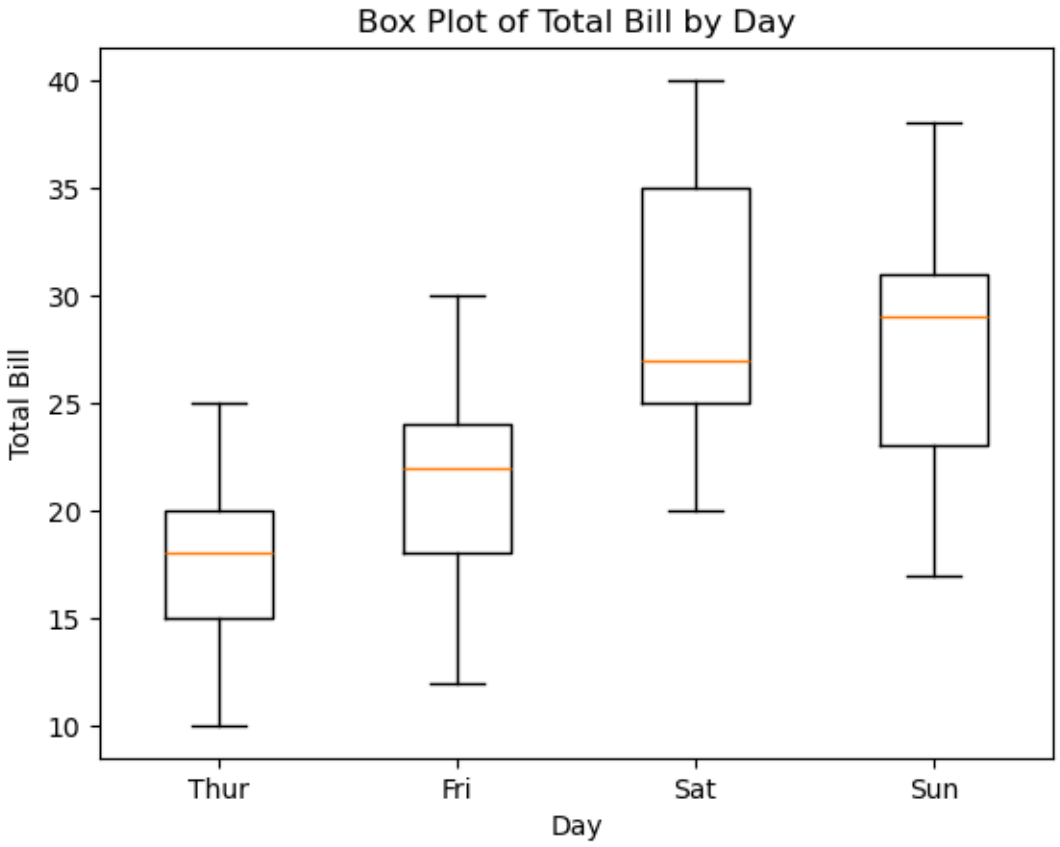
[25]:
```
#plt.bar() is used to plot two bars (male and female) side by side for each␣
 ↪class.
#X-axis shows the passenger classes, and Y-axis shows the count.
#legend() is used to label male and female bars.
#This simple version uses hardcoded counts to demonstrate grouped bar plotting.
data = {
    'class': ['First', 'Second', 'Third', 'First', 'Second', 'Third', 'Third',␣
 ↪'Second', 'First', 'Third'],
    'sex':   ['male', 'female', 'male', 'female', 'female', 'male', 'female',␣
 ↪'male', 'female', 'male']
}
```

```
df = pd.DataFrame(data)
counts = df.groupby(['class', 'sex']).size().unstack(fill_value=0)
counts.plot(kind='bar', stacked=False, color=['blue', 'pink'])
plt.title('Count  of Passenger Class by Sex')
plt.xlabel('Passenger Class')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Sex')
plt.show()
```



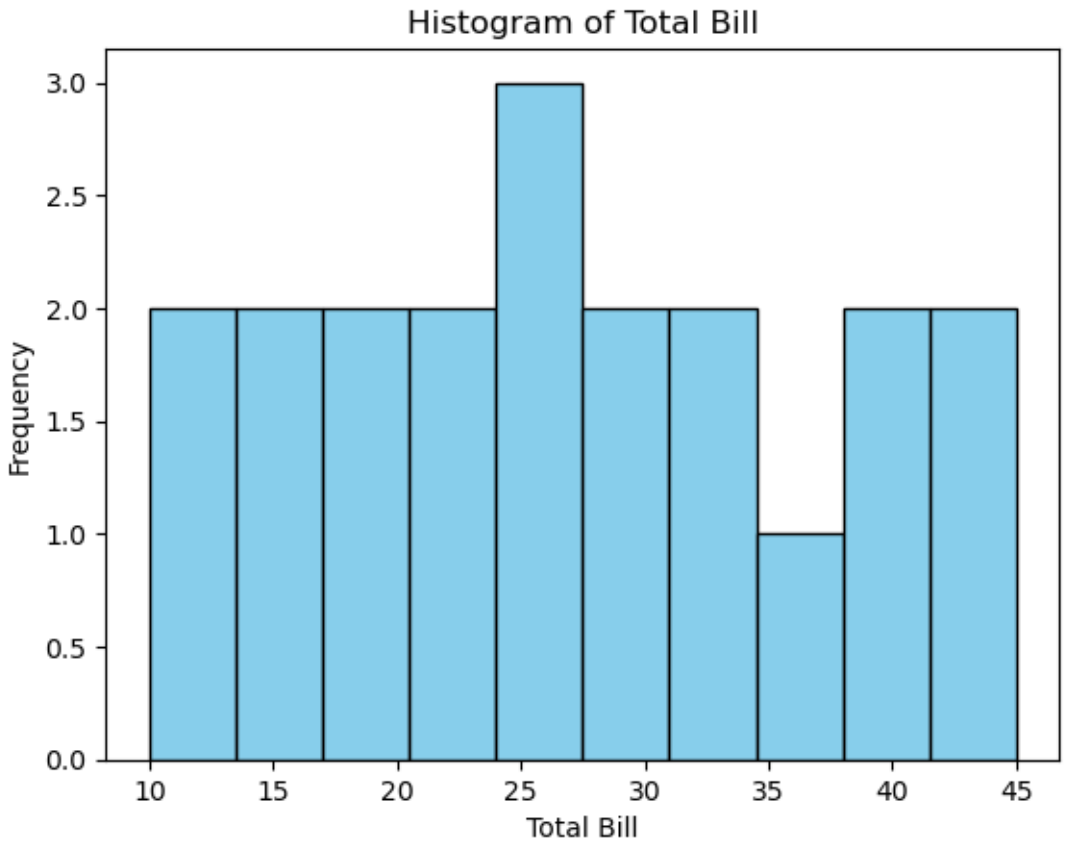
[35]:
```
#A box plot is used to visualize the spread and distribution of numerical data.
#It highlights the median, quartiles, and potential outliers in the dataset.
#We compare the total bill amounts across different days (Thur, Fri, Sat, Sun)␣
 ↪using Matplotlib only.
#Each box represents the distribution of bills on that day.

data = {
    'Thur': [10, 15, 20, 25, 18],
    'Fri': [12, 18, 22, 24, 30],
    'Sat': [20, 25, 27, 35, 40],
```

```
    'Sun': [17, 23, 29, 31, 38]
}
days = list(data.keys())
values = [data[day] for day in days]
plt.boxplot(values, tick_labels=days)
plt.title('Box Plot of Total Bill by Day')
plt.xlabel('Day')
plt.ylabel('Total Bill')
plt.show()
```



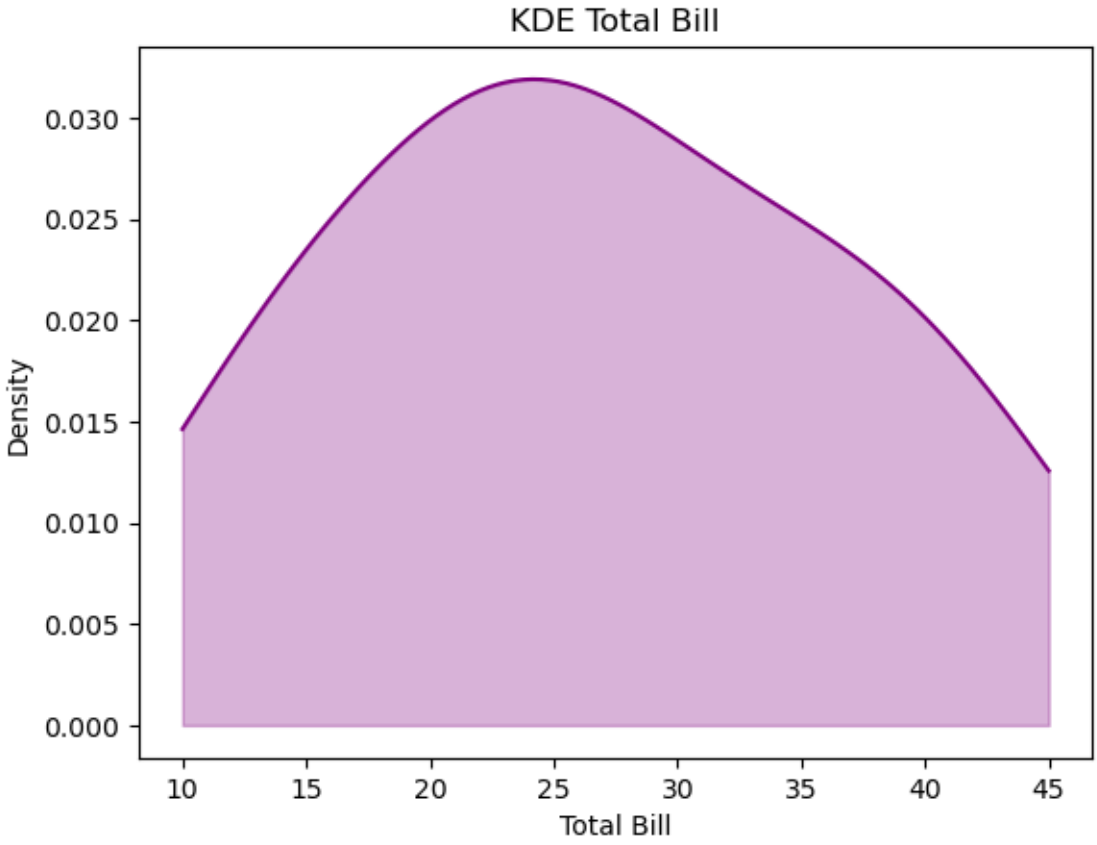
[42]:
```
#A histogram displays the frequency distribution of a numerical variable by␣
 ↪grouping data into bins.
#This example shows how the total_bill values are distributed using Matplotlib␣
 ↪only.
#plt.hist() creates the histogram with 10 bins.
#color sets the bar color and edgecolor outlines the bars.
total_bill = [10, 12, 14, 16, 18, 20, 22, 23, 24, 25,
              26, 28, 30, 32, 34, 36, 38, 40, 42, 45]
plt.hist(total_bill, bins=10, color='skyblue', edgecolor='black')
plt.title('Histogram of Total Bill')
```

```
plt.xlabel('Total Bill')
plt.ylabel('Frequency')
plt.show()
```



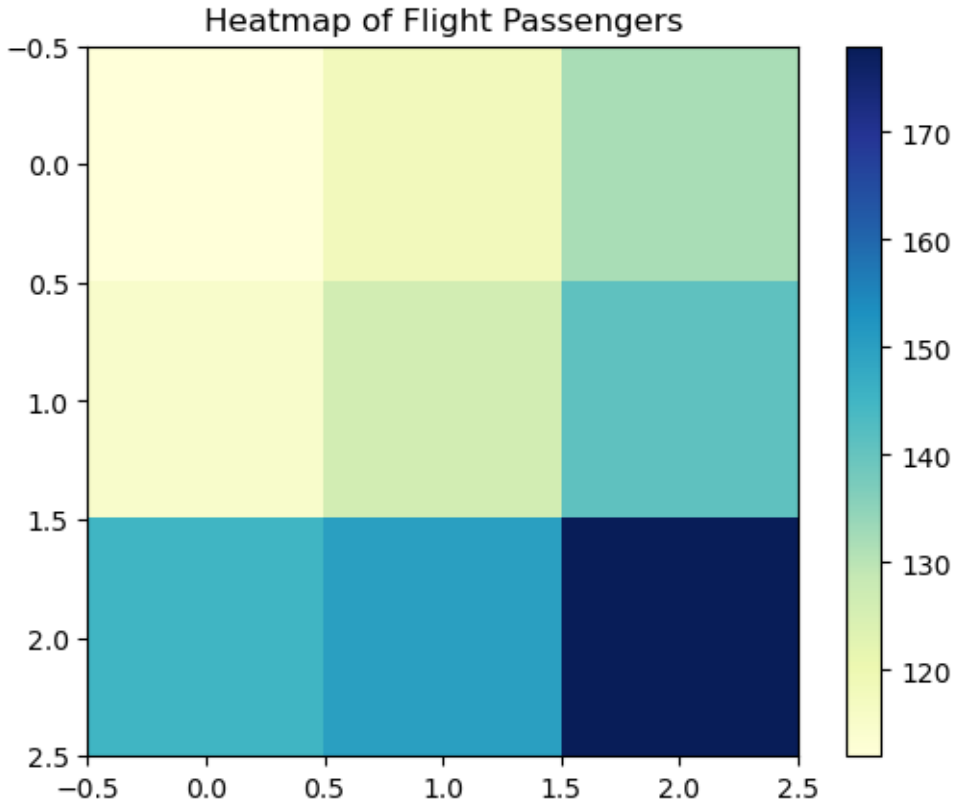
[48]:
```
#A KDE plot (Kernel Density Estimate) is a smooth curve that estimates the␣
 ↪probability density of a variable.
#In this code, we use gaussian_kde() from scipy.stats to compute KDE for the␣
 ↪total_bill data.
#plt.plot() draws the smooth density curve
#plt.fill_between() shades the area under the curve (like fill=True in Seaborn).
#This plot helps us visualize where values are concentrated.
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import gaussian_kde
total_bill = [10, 12, 14, 16, 18, 20, 22, 23, 24, 25,
              26, 28, 30, 32, 34, 36, 38, 40, 42, 45]
kde = gaussian_kde(total_bill)
x = np.linspace(min(total_bill), max(total_bill), 100)
y = kde(x)
plt.plot(x, y, color='purple')
```

```
plt.fill_between(x, y, color='purple', alpha=0.3)
plt.title('KDE Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Density')
plt.show()
```

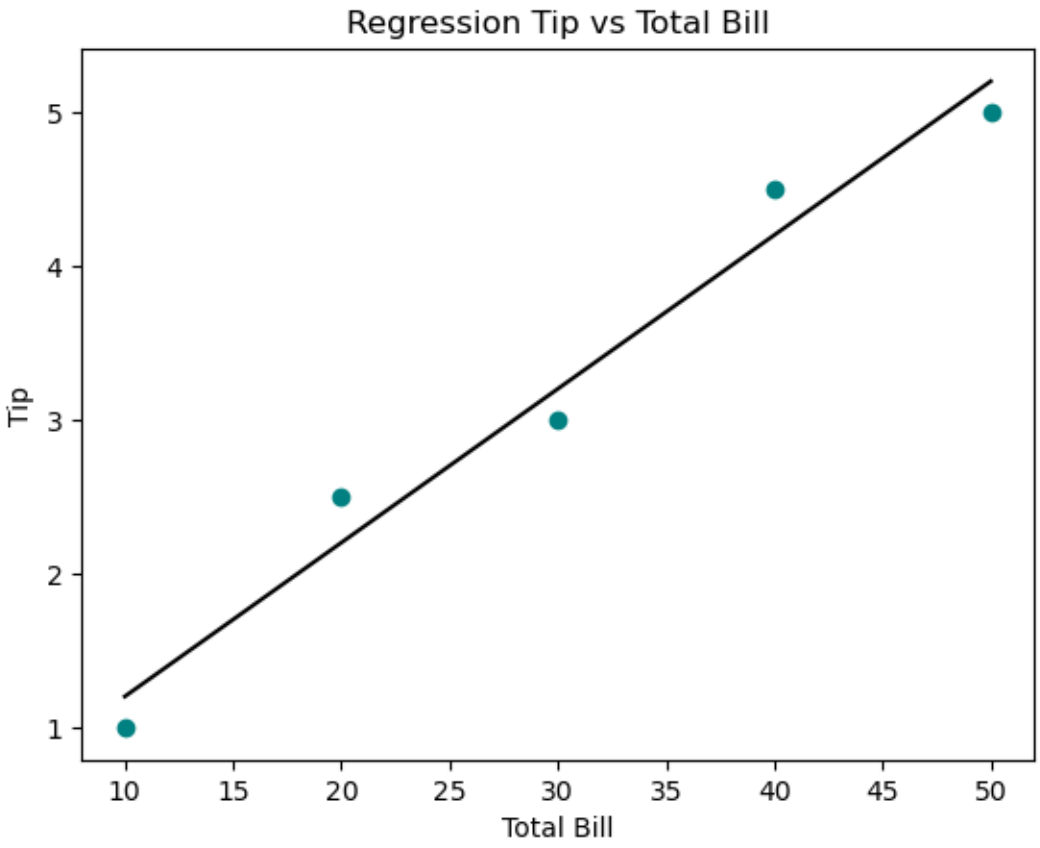


[52]:
```
#A heatmap visualizes 2D data using color intensity.
#plt.imshow() is used in Matplotlib to create a heatmap.
#Each cell's color represents the magnitude of the value.
#plt.colorbar() adds a color scale beside the heatmap.
#Useful for showing patterns, trends, and high/low values in a table-like␣
 ↪format.
#Commonly used in correlation matrices, flight data, and confusion matrices.
import matplotlib.pyplot as plt

data = [[112, 118, 132],
        [115, 126, 141],
        [145, 150, 178]]

plt.imshow(data, cmap='YlGnBu')
```

```
plt.title("Heatmap of Flight Passengers")
plt.colorbar()
plt.show()
```



[60]:
```
#A regression plot shows the relationship between two variables with a best-fit␣
 ↪line.
#plt.scatter() is used to plot the data points.
#np.polyfit() fits a linear regression line.
#plt.plot() draws the regression line.
#Helps to identify correlation and trend between total_bill and tip.
total_bill = np.array([10, 20, 30, 40, 50])
tip = np.array([1, 2.5, 3, 4.5, 5])
coeffs = np.polyfit(total_bill, tip, deg=1)
reg_line = np.poly1d(coeffs)
plt.scatter(total_bill, tip, color='teal')
plt.plot(total_bill, reg_line(total_bill), color='black')
plt.title('Regression Tip vs Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```


Regression Tip vs Total Bill
Tip
Total Bill
1
2
3
4
5
10
15
20
25
30
35
40
45
50

[ ]:

shadowfox-intermediate

August 30, 2025

[50]:
```
#Import Libraries & Load Dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv(r"C:\Users\varsh\Downloads\delhiaqi.csv")
# Preview data
print(df.head())
print(df.info())
```

```
                 date       co     no    no2    o3    so2   pm2_5    pm10  \
0  2023-01-01 00:00:00  1655.58   1.66  39.41  5.90  17.88  169.29  194.64
1  2023-01-01 01:00:00  1869.20   6.82  42.16  1.99  22.17  182.84  211.08
2  2023-01-01 02:00:00  2510.07  27.72  43.87  0.02  30.04  220.25  260.68
3  2023-01-01 03:00:00  3150.94  55.43  44.55  0.85  35.76  252.90  304.12
4  2023-01-01 04:00:00  3471.37  68.84  45.24  5.45  39.10  266.36  322.80

     nh3
0   5.83
1   7.66
2  11.40
3  13.55
4  14.19
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 561 entries, 0 to 560
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    561 non-null    object
 1   co      561 non-null    float64
 2   no      561 non-null    float64
 3   no2     561 non-null    float64
 4   o3      561 non-null    float64
 5   so2     561 non-null    float64
 6   pm2_5   561 non-null    float64
 7   pm10    561 non-null    float64
```

```
 8   nh3     561 non-null    float64
dtypes: float64(8), object(1)
memory usage: 39.6+ KB
None
```

[40]:
```
#Preprocess Dataset
df['date'] = pd.to_datetime(df['date'], errors='coerce')
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
def assign_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Summer'
    elif month in [6, 7, 8, 9]:
        return 'Monsoon'
    else:
        return 'Post-Monsoon'
df['season'] = df['month'].apply(assign_season)
print(df.head())
```

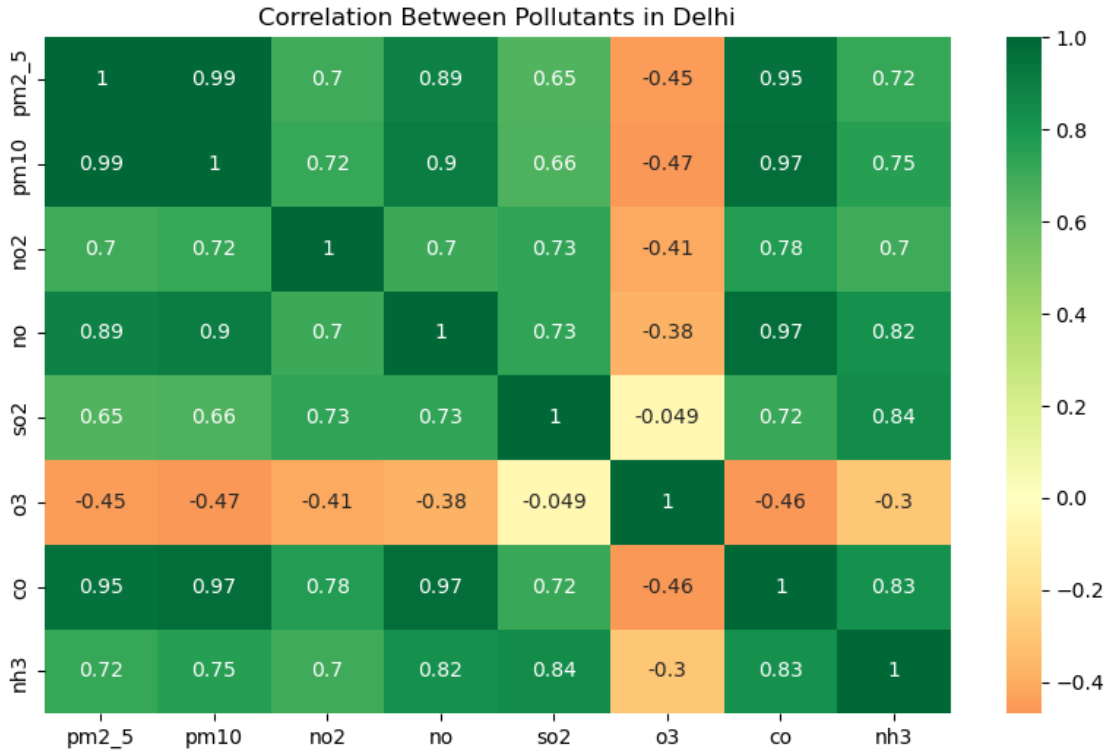
```
                 date       co     no    no2    o3    so2   pm2_5    pm10  \
0 2023-01-01 00:00:00  1655.58   1.66  39.41  5.90  17.88  169.29  194.64
1 2023-01-01 01:00:00  1869.20   6.82  42.16  1.99  22.17  182.84  211.08
2 2023-01-01 02:00:00  2510.07  27.72  43.87  0.02  30.04  220.25  260.68
3 2023-01-01 03:00:00  3150.94  55.43  44.55  0.85  35.76  252.90  304.12
4 2023-01-01 04:00:00  3471.37  68.84  45.24  5.45  39.10  266.36  322.80

     nh3  year  month  season
0   5.83  2023      1  Winter
1   7.66  2023      1  Winter
2  11.40  2023      1  Winter
3  13.55  2023      1  Winter
4  14.19  2023      1  Winter
```

[26]:
```
# Check missing values
print("Missing values before filling:")
print(df.isnull().sum())

# Fill only numeric columns with their median
numeric_cols = df.select_dtypes(include=['number']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())

print("\nMissing values after filling:")
print(df.isnull().sum())
```

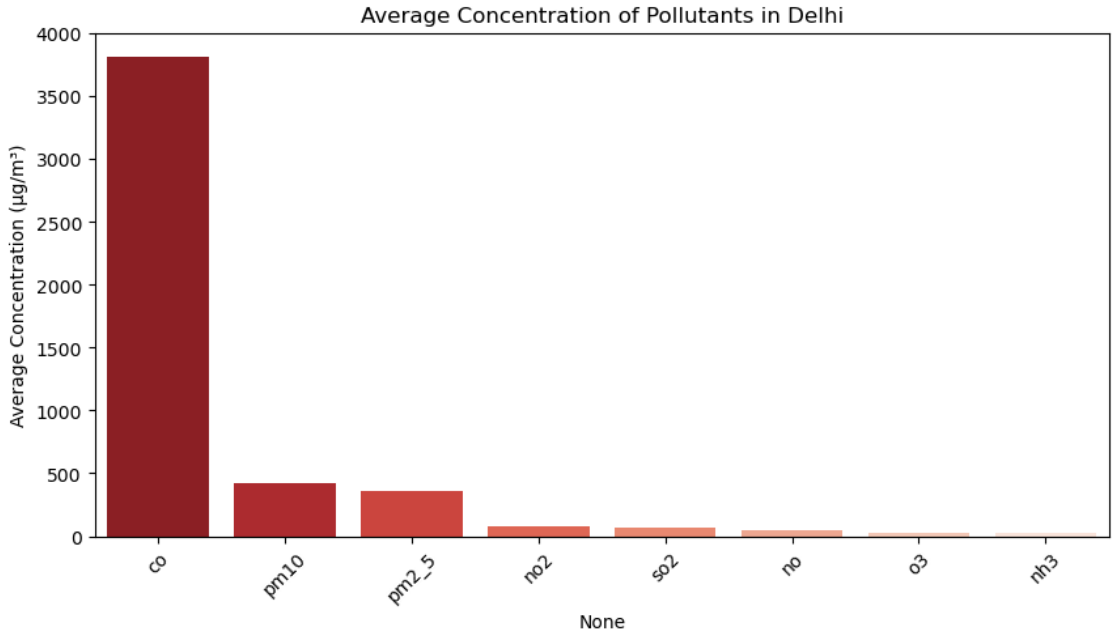
```
Missing values before filling:
date      0
```

```
co        0
no        0
no2       0
o3        0
so2       0
pm2_5     0
pm10      0
nh3       0
year      0
month     0
season    0
dtype: int64

Missing values after filling:
date      0
co        0
no        0
no2       0
o3        0
so2       0
pm2_5     0
pm10      0
nh3       0
year      0
month     0
season    0
dtype: int64
```

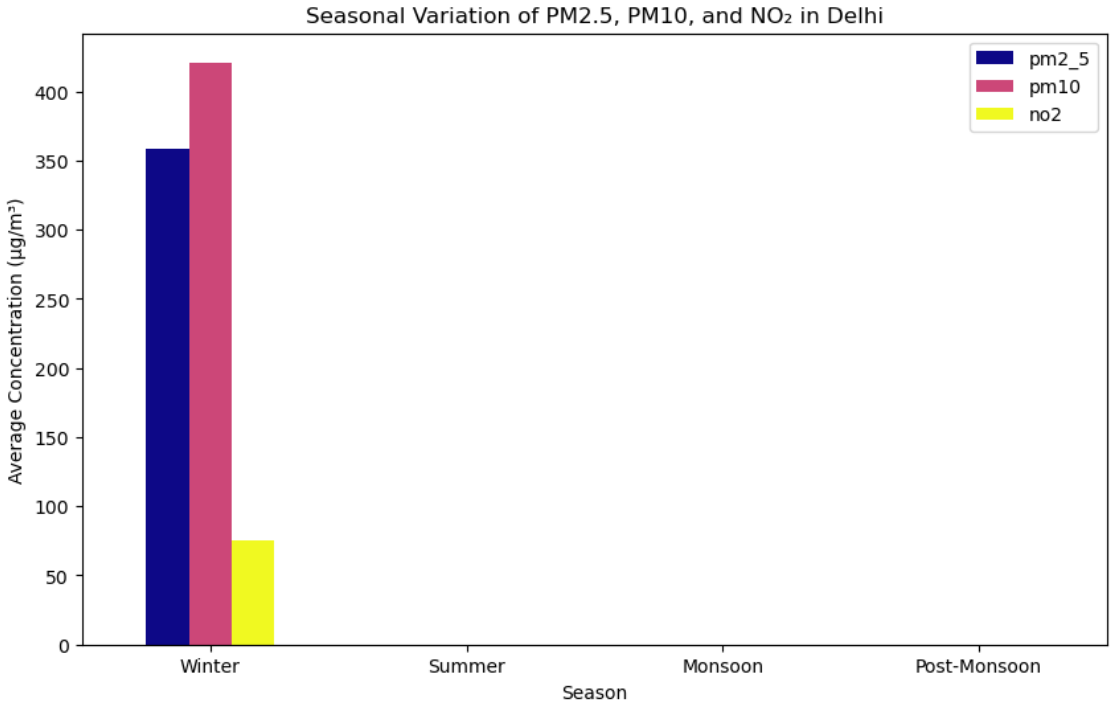
[42]:
```python
# Correlation Analysis of Pollutants
pollutants = ['pm2_5', 'pm10', 'no2', 'no', 'so2', 'o3', 'co', 'nh3']
plt.figure(figsize=(10,6))
sns.heatmap(df[pollutants].corr(), annot=True, cmap="RdYlGn", center=0)
plt.title("Correlation Between Pollutants in Delhi")
plt.show()
```



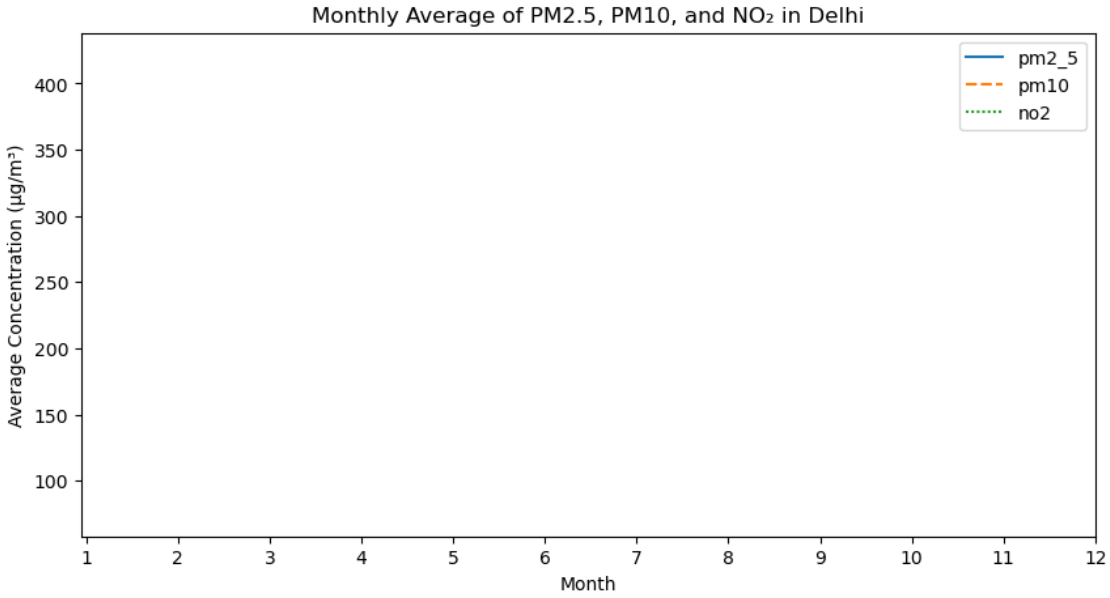
[46]:
```
#Average Concentration of Pollutants
avg_pollutants = df[pollutants].mean().sort_values(ascending=False)
plt.figure(figsize=(10,5))
sns.barplot(
    x=avg_pollutants.index,
    y=avg_pollutants.values,
    hue=avg_pollutants.index,
    palette="Reds_r",
    legend=False
)
plt.ylabel("Average Concentration (µg/m³)")
plt.title("Average Concentration of Pollutants in Delhi")
plt.xticks(rotation=45)
plt.show()
```



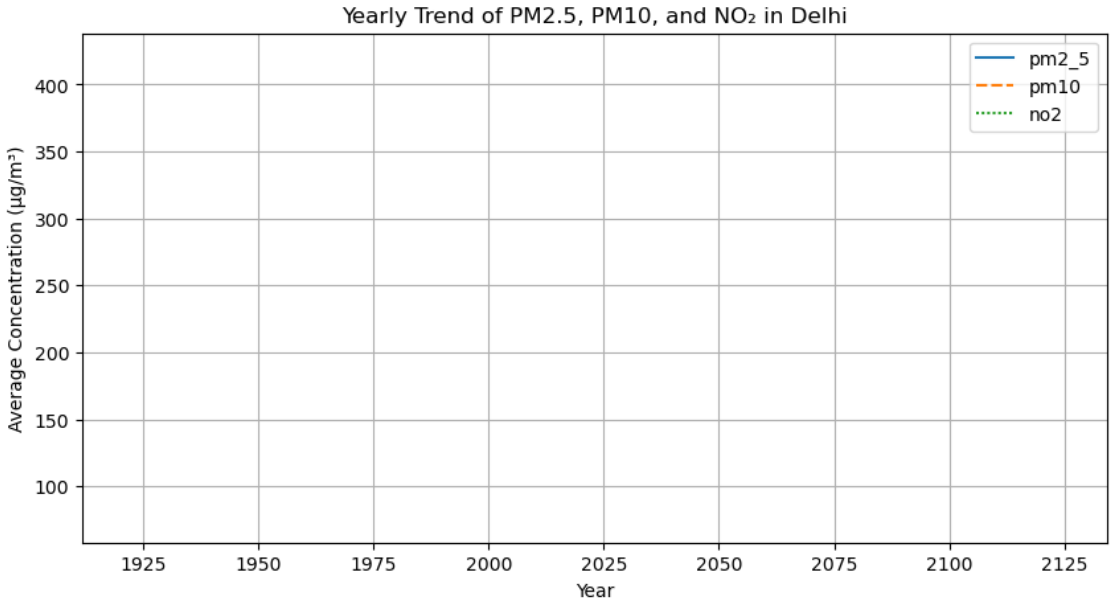
[44]:
```
# Seasonal Variation of Key Pollutants
seasonal_avg = df.groupby("season")[["pm2_5", "pm10", "no2"]].mean()
seasonal_avg = seasonal_avg.reindex(["Winter", "Summer", "Monsoon",␣
 ↪"Post-Monsoon"])

# Bar plot
seasonal_avg.plot(kind="bar", figsize=(10,6), colormap="plasma")
plt.title("Seasonal Variation of PM2.5, PM10, and NO  in Delhi")
plt.ylabel("Average Concentration (µg/m³)")
plt.xlabel("Season")
plt.xticks(rotation=0)
plt.show()
```



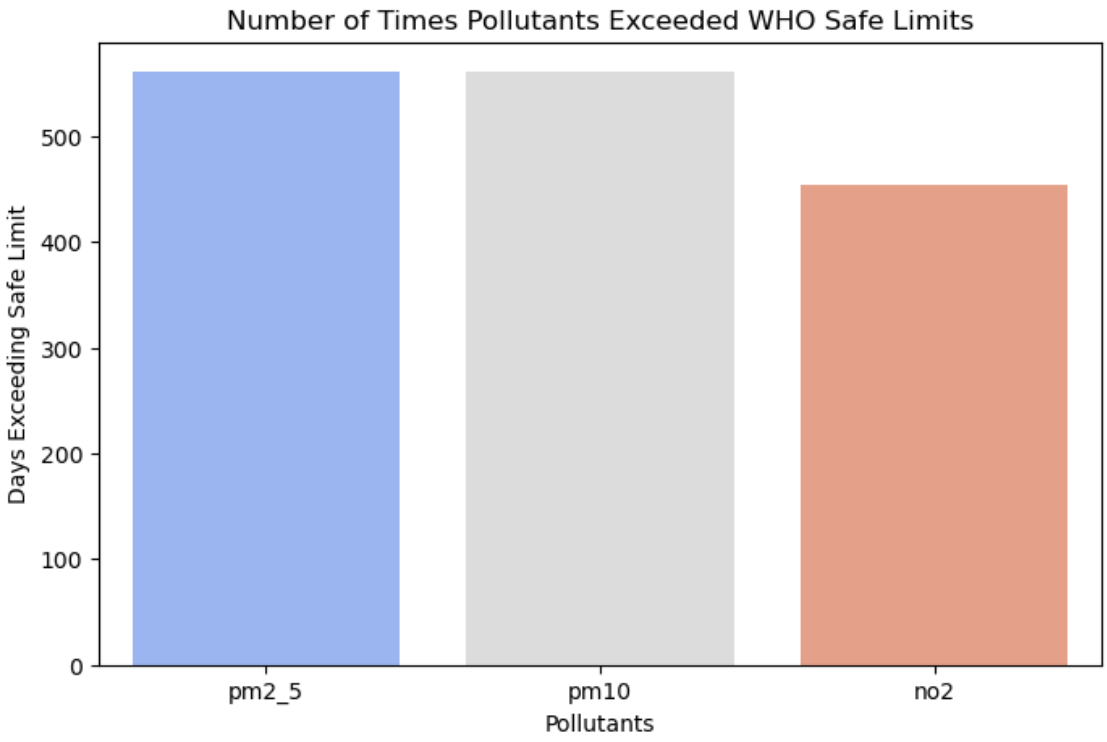
[48]:
```
# Monthly AQI Trend
monthly_avg = df.groupby("month")[["pm2_5", "pm10", "no2"]].mean()

plt.figure(figsize=(10,5))
sns.lineplot(data=monthly_avg)
plt.title("Monthly Average of PM2.5, PM10, and NO  in Delhi")
plt.ylabel("Average Concentration (µg/m³)")
plt.xlabel("Month")
plt.xticks(range(1,13))
plt.show()
```



[68]:
```
# Yearly AQI Trend
df['date'] = pd.to_datetime(df['date'], errors='coerce')
df['year'] = df['date'].dt.year
yearly_avg = df.groupby("year")[["pm2_5", "pm10", "no2"]].mean()
plt.figure(figsize=(10, 5))
sns.lineplot(data=yearly_avg)
plt.title("Yearly Trend of PM2.5, PM10, and NO  in Delhi")
plt.ylabel("Average Concentration (µg/m³)")
plt.xlabel("Year")
plt.grid(True)
plt.show()
```



[64]:
```python
# Pollution Episodes (Exceeding Safe Limits)
safe_limits = {'pm2_5': 25, 'pm10': 50, 'no2': 40}
exceedances = {
    pollutant: (df[pollutant] > safe_limits[pollutant]).sum()
    for pollutant in safe_limits
}

plt.figure(figsize=(8, 5))
sns.barplot(
    x=list(exceedances.keys()),
    y=list(exceedances.values()),
    hue=list(exceedances.keys()),
    palette="coolwarm",
    legend=False
)
plt.title("Number of Times Pollutants Exceeded WHO Safe Limits")
plt.ylabel("Days Exceeding Safe Limit")
plt.xlabel("Pollutants")
plt.show()
```


Number of Times Pollutants Exceeded WHO Safe Limits
Days Exceeding Safe Limit
0
100
200
300
400
500
pm2_5
pm10
no2
Pollutants

[ ]:

shadowfox-advanced

August 30, 2025

[124]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from datetime import timedelta


sns.set(style='whitegrid')
plt.rcParams['figure.figsize'] = (10,5)
pd.options.display.max_columns = 60
os.makedirs('figures', exist_ok=True)
```

[126]:
```
USE_SYNTHETIC = True # set False to load your CSV
USER_CSV_PATH = 'sales_data.csv' # used if USE_SYNTHETIC=False
```

[165]:
```
df = generate_synthetic_sales()
print(df.head())
print(df.shape)
```

```
  TransactionID       Date   Product Region  Quantity  GrossSales  Discount  \
0       T000172 2018-01-02  Widget A  South         4      387.38     30.25
1       T001347 2018-01-03  Widget A   West         3      390.11      8.97
2       T000829 2018-01-04  Widget B  North         4      637.47     42.54
3       T000362 2018-01-05  Gadget X  South         1       55.69      3.89
4       T000825 2018-01-06  Gadget Y   East         3      219.41      4.75

   NetSales    COGS  ManufacturingCost  FreightCost  Profit  FiscalYear
0    357.13  154.06              14.77        17.32  154.83        2017
1    381.15  208.91              24.33         4.87  142.99        2017
2    594.93  310.81              60.09        30.74  186.33        2017
3     51.80   20.89               3.01         1.99   18.88        2017
4    214.66   87.43              18.98         6.39   95.88        2017
(1500, 13)
```

[169]:
```
df.shape
```

[169]: (1500, 13)

[167]:
```
import numpy as np
import pandas as pd
import random

def generate_synthetic_sales(n=1500, start_date='2018-01-01',␣
 ↪end_date='2024-12-31', fiscal_start_month=4, random_seed=42):
    np.random.seed(random_seed)
    random.seed(random_seed)
    date_range = pd.date_range(start_date, end_date, freq='D')
    dates = np.random.choice(date_range, size=n)

    products = ['Widget A', 'Widget B', 'Gadget X', 'Gadget Y']
    regions = ['North', 'South', 'East', 'West']
    price_map = {'Widget A':100,'Widget B':150,'Gadget X':80,'Gadget Y':120}

    rows = []
    for i in range(n):
        d = dates[i]
        prod = np.random.choice(products, p=[0.35,0.25,0.2,0.2])
        qty = np.random.randint(1,6)
        base_price = price_map[prod]
        month = pd.to_datetime(d).month
        season = 1.25 if month in [11,12] else (0.85 if month==6 else 1.0)
        gross = base_price * qty * (1 + np.random.normal(0,0.12)) * season
        discount = gross * np.random.uniform(0,0.08)
        net = gross - discount
        cogs = net * np.random.uniform(0.40,0.65)
        manuf = net * np.random.uniform(0.04,0.12)
        freight = net * np.random.uniform(0.01,0.06)
        profit = net - (cogs + manuf + freight) - np.random.uniform(0,20)
        rows.append({
            'TransactionID': f'T{str(i+1).zfill(6)}',
            'Date': pd.to_datetime(d),
            'Product': prod,
            'Region': np.random.choice(regions),
            'Quantity': qty,
            'GrossSales': round(gross,2),
            'Discount': round(discount,2),
            'NetSales': round(net,2),
            'COGS': round(cogs,2),
            'ManufacturingCost': round(manuf,2),
            'FreightCost': round(freight,2),
            'Profit': round(profit,2)
        })
```

```
    df = pd.DataFrame(rows)
    def fiscal_year(dt):
        dt = pd.to_datetime(dt)
        return dt.year if dt.month >= fiscal_start_month else dt.year - 1
    df['FiscalYear'] = df['Date'].apply(fiscal_year)
    df = df.sort_values('Date').reset_index(drop=True)
    return df
```

[137]:
```
if USE_SYNTHETIC:
    df = generate_synthetic_sales()
else:
    df = pd.read_csv(USER_CSV_PATH, parse_dates=['Date'])

print('Loaded rows:', len(df))
```

```
Loaded rows: 1500
```

[143]:
```
print(df.head())
print('Data types:')
print(df.dtypes)
print('Missing counts:')
print(df.isnull().sum())
```

```
  TransactionID       Date   Product Region  Quantity  GrossSales  Discount  \
0       T000172 2018-01-02  Widget A  South         4      387.38     30.25
1       T001347 2018-01-03  Widget A   West         3      390.11      8.97
2       T000829 2018-01-04  Widget B  North         4      637.47     42.54
3       T000362 2018-01-05  Gadget X  South         1       55.69      3.89
4       T000825 2018-01-06  Gadget Y   East         3      219.41      4.75

   NetSales    COGS  ManufacturingCost  FreightCost  Profit  FiscalYear
0    357.13  154.06              14.77        17.32  154.83        2017
1    381.15  208.91              24.33         4.87  142.99        2017
2    594.93  310.81              60.09        30.74  186.33        2017
3     51.80   20.89               3.01         1.99   18.88        2017
4    214.66   87.43              18.98         6.39   95.88        2017
Data types:
TransactionID                 object
Date                  datetime64[ns]
Product                       object
Region                        object
Quantity                       int64
GrossSales                   float64
Discount                     float64
NetSales                     float64
COGS                         float64
ManufacturingCost            float64
```

```
FreightCost                 float64
Profit                      float64
FiscalYear                    int64
dtype: object
Missing counts:
TransactionID        0
Date                 0
Product              0
Region               0
Quantity             0
GrossSales           0
Discount             0
NetSales             0
COGS                 0
ManufacturingCost    0
FreightCost          0
Profit               0
FiscalYear           0
dtype: int64
```

[179]:
```
if not np.issubdtype(df['Date'].dtype, np.datetime64):
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
df = df.dropna(subset=['Date', 'TransactionID', 'NetSales'])
df = df.drop_duplicates(subset=['TransactionID'])
df = df[(df['NetSales'] > 0) & (df['GrossSales'] > 0)].reset_index(drop=True)

print('After cleaning rows:', len(df))
```

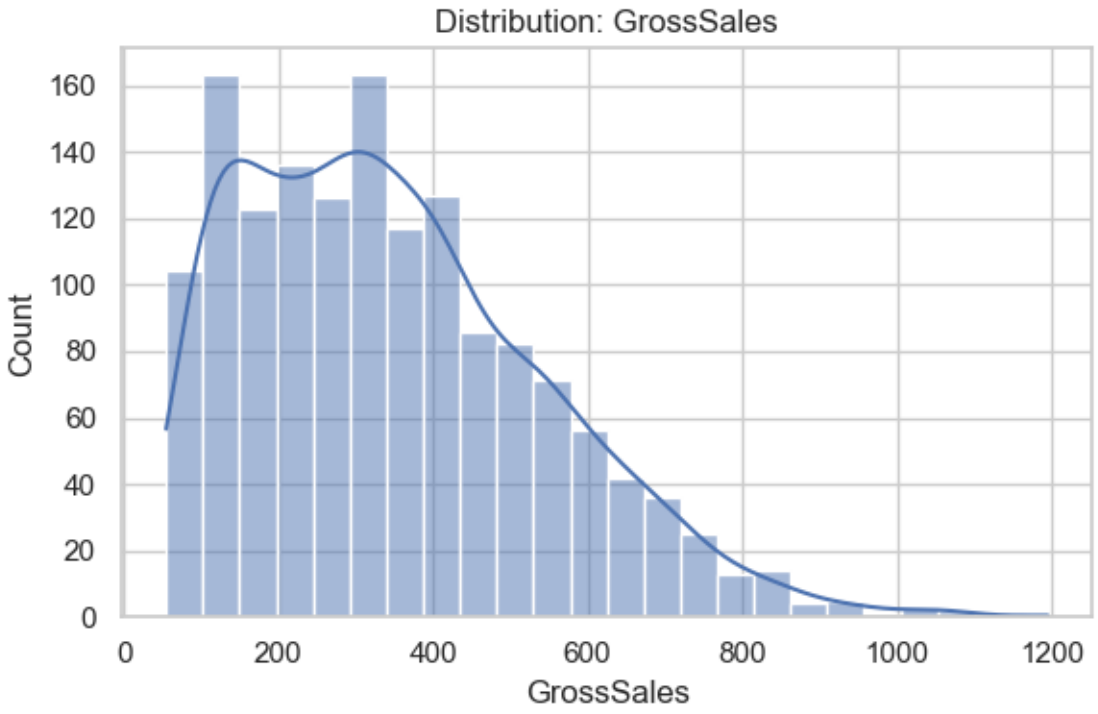
```
After cleaning rows: 1500
```

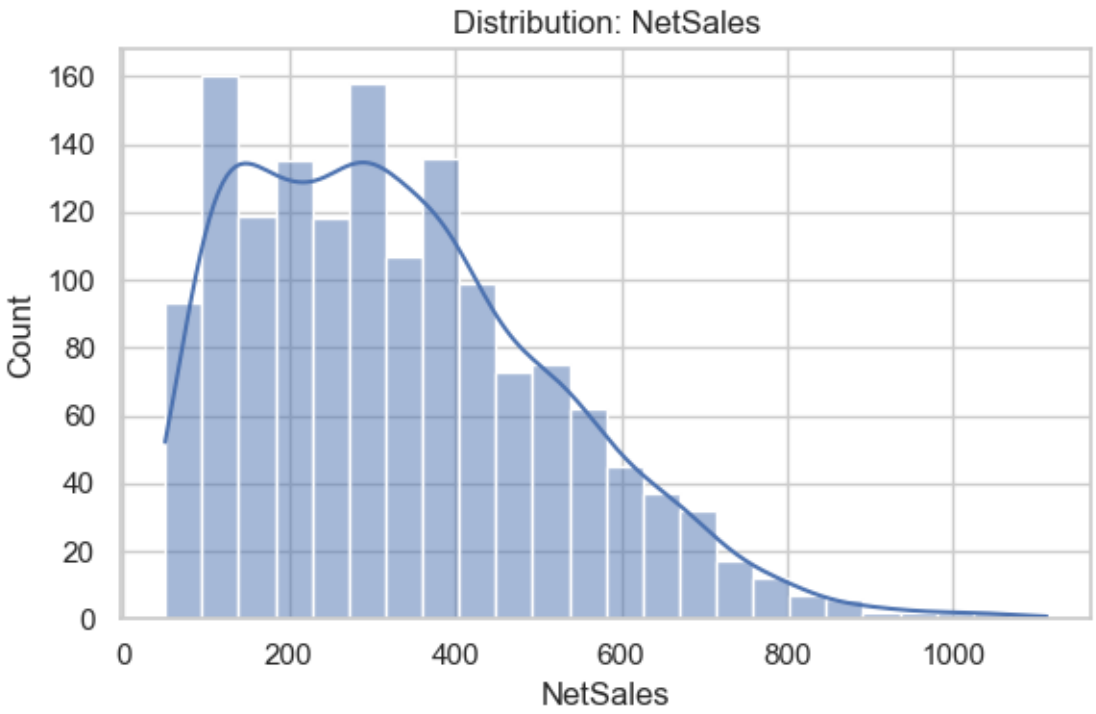
[181]:
```
if not np.issubdtype(df['Date'].dtype, np.datetime64):
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
df = df.dropna(subset=['Date', 'TransactionID', 'NetSales'])
df = df.drop_duplicates(subset=['TransactionID'])
df = df[(df['NetSales'] > 0) & (df['GrossSales'] > 0)].reset_index(drop=True)
print('After cleaning rows:', len(df))
eps = 1e-9
for col in ['COGS', 'ManufacturingCost', 'FreightCost', 'Profit']:
    df[col + '_pct'] = df[col] / (df['NetSales'] + eps)

print(df[['NetSales', 'COGS', 'COGS_pct', 'Profit', 'Profit_pct']].head())
```

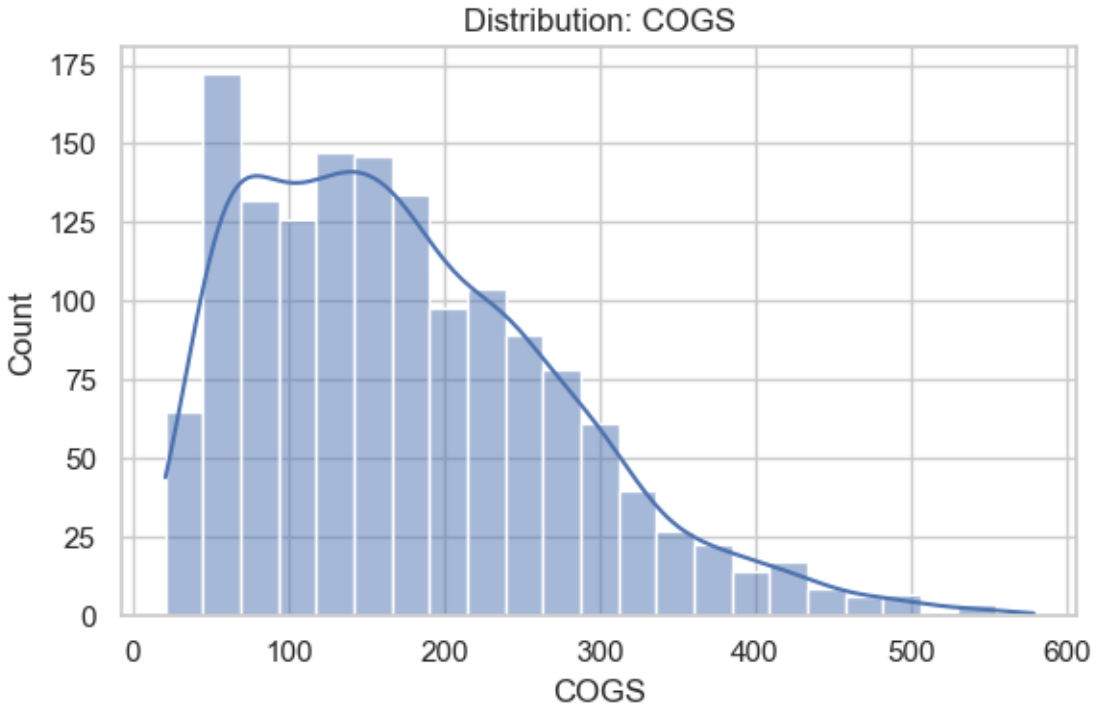
```
After cleaning rows: 1500
   NetSales    COGS  COGS_pct  Profit  Profit_pct
0    357.13  154.06  0.431384  154.83    0.433540
1    381.15  208.91  0.548104  142.99    0.375154
2    594.93  310.81  0.522431  186.33    0.313197
3     51.80   20.89  0.403282   18.88    0.364479
```

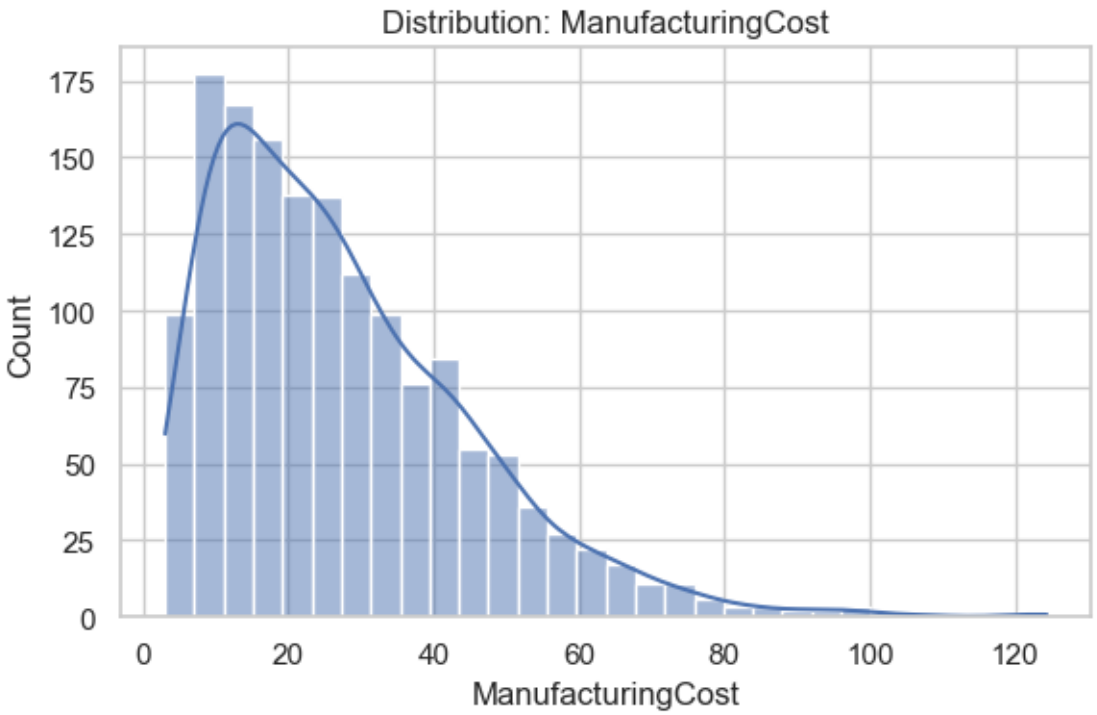
```
4     214.66    87.43  0.407295    95.88     0.446660
```

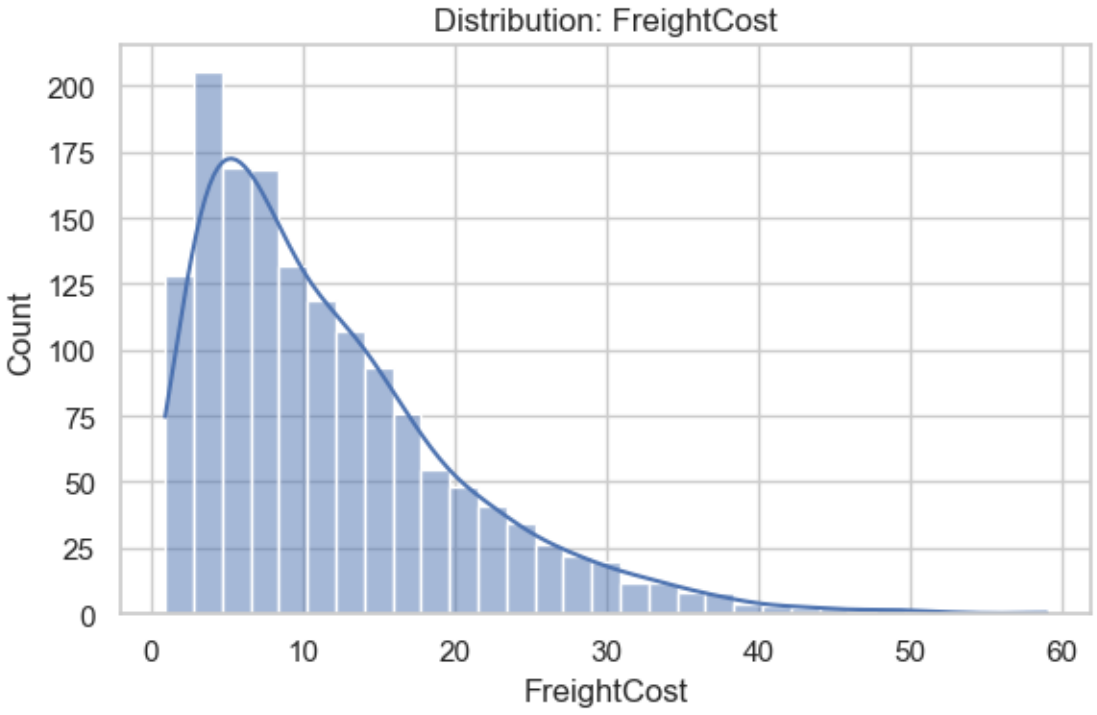
[191]:
```
os.makedirs("figures", exist_ok=True)
numeric_cols = ['GrossSales', 'NetSales', 'COGS', 'ManufacturingCost',␣
 ↪'FreightCost', 'Profit']
for c in numeric_cols:
    plt.figure(figsize=(6,4))
    sns.histplot(df[c], kde=True)
    plt.title(f'Distribution: {c}')
    plt.tight_layout()
    plt.savefig(f'figures/dist_{c}.png')
    plt.show()
```

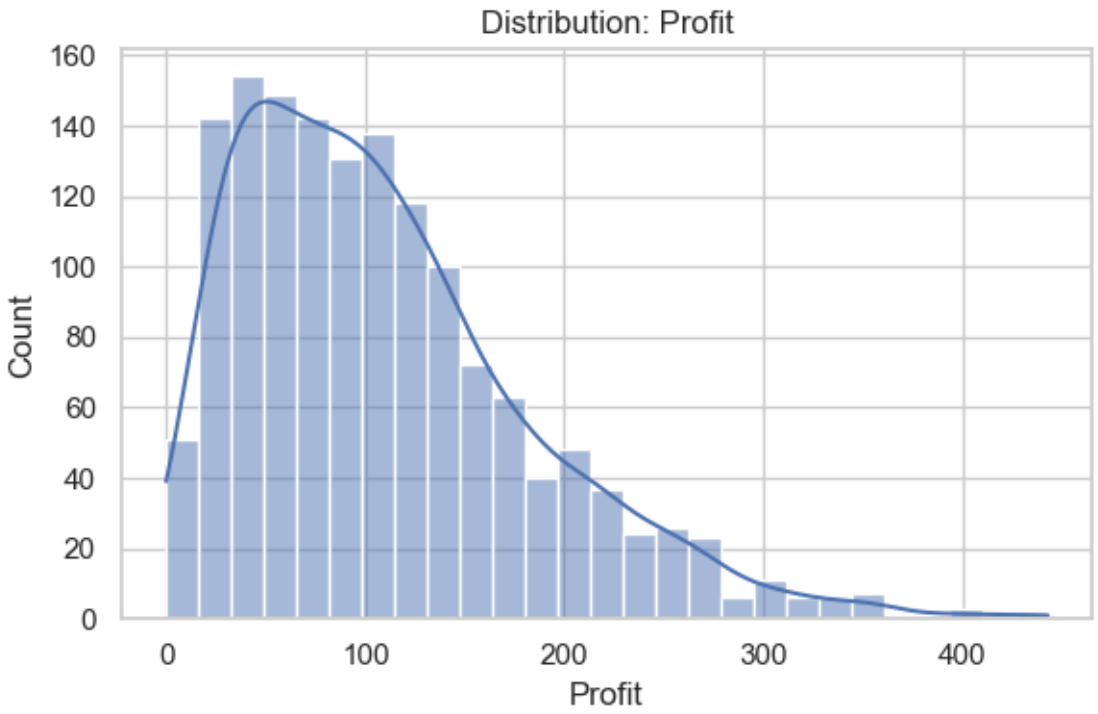



Distribution: NetSales
Count
NetSales
0
20
40
60
80
100
120
140
160
0
200
400
600
800
1000


Distribution: COGS
Count
COGS
0
25
50
75
100
125
150
175
0
100
200
300
400
500
600

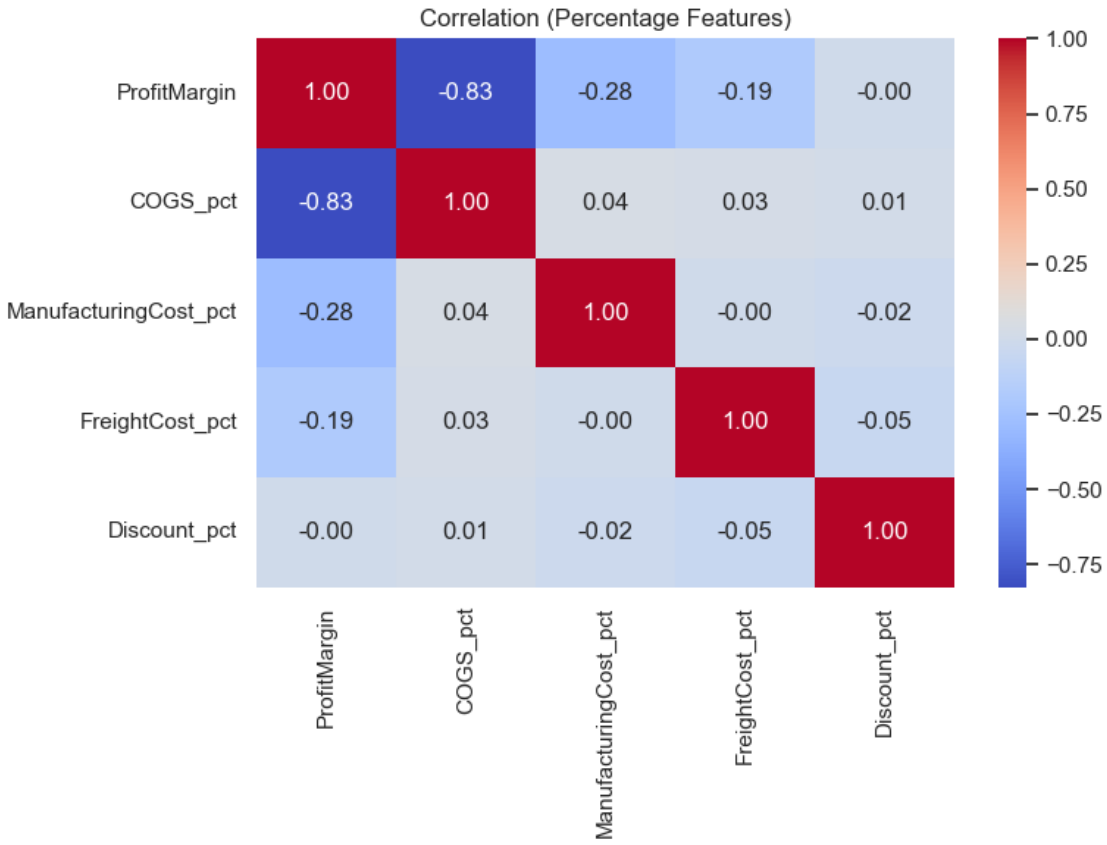

Distribution: ManufacturingCost
Count
ManufacturingCost
0
25
50
75
100
125
150
175
0
20
40
60
80
100
120


Distribution: FreightCost
Count
FreightCost
0
25
50
75
100
125
150
175
200
0
10
20
30
40
50
60

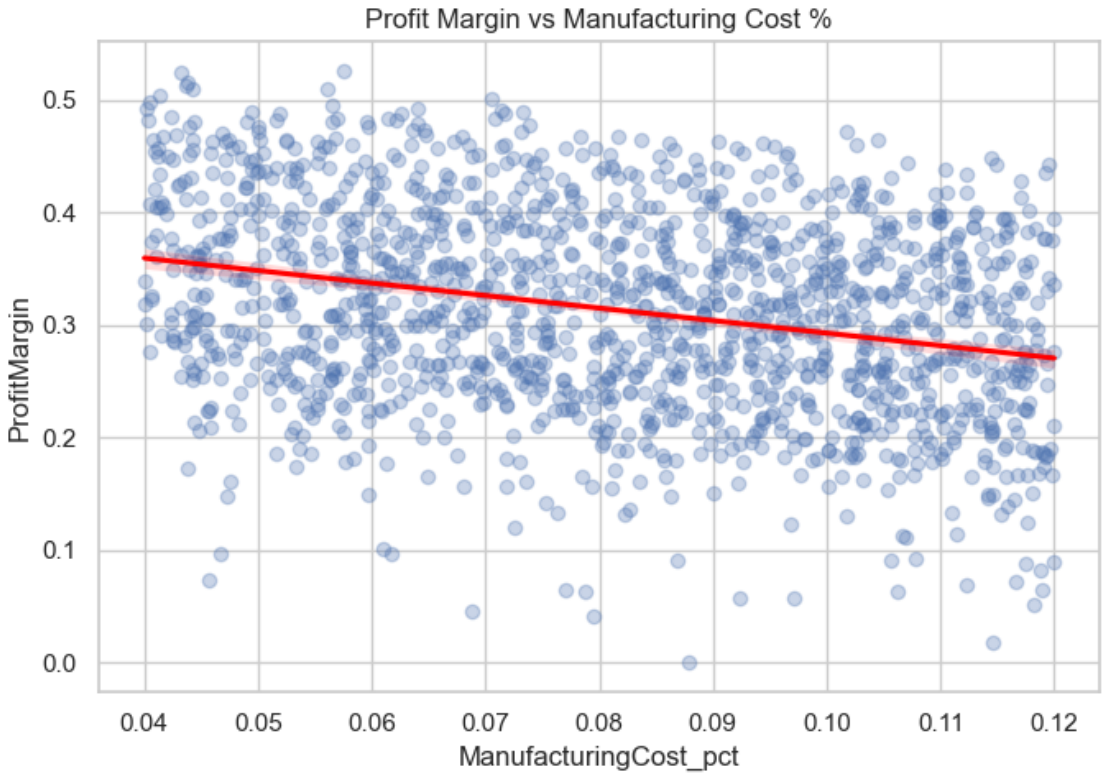


[195]:
```
eps = 1e-9

# Profit Margin = Profit / NetSales
df['ProfitMargin'] = df['Profit'] / (df['NetSales'] + eps)

# Discount percentage = Discount / GrossSales
df['Discount_pct'] = df['Discount'] / (df['GrossSales'] + eps)
```

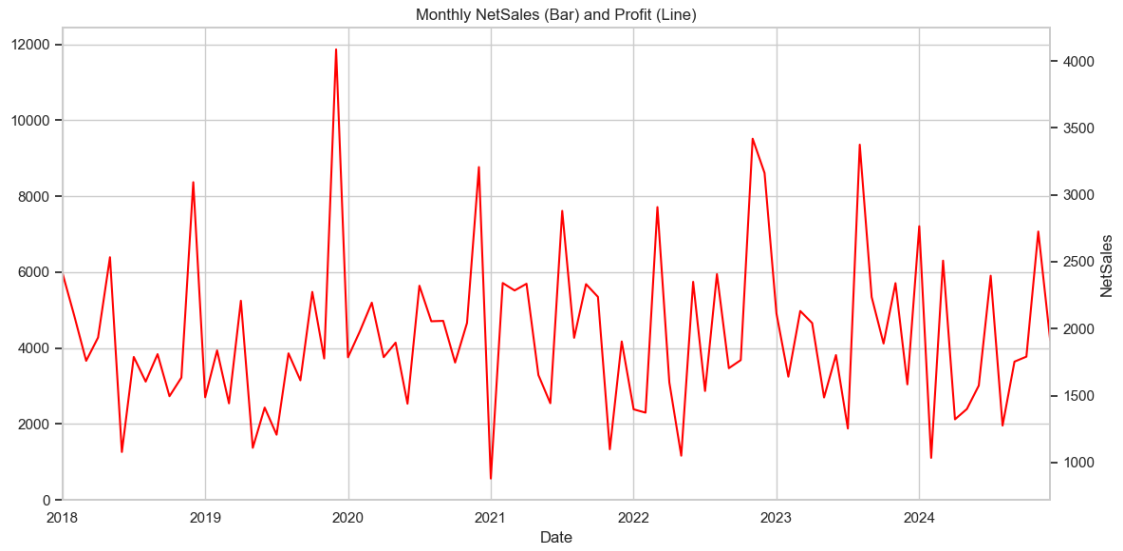
[197]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Ensure figures folder exists
os.makedirs("figures", exist_ok=True)

plt.figure(figsize=(8,6))
corr = df[['ProfitMargin', 'COGS_pct', 'ManufacturingCost_pct',␣
 ↪'FreightCost_pct', 'Discount_pct']].corr()
sns.heatmap(corr, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation (Percentage Features)')
plt.tight_layout()
plt.savefig('figures/corr_pct.png')
plt.show()
```



[199]:
```
plt.figure(figsize=(7,5))
sns.regplot(
    data=df,
    x='ManufacturingCost_pct',
    y='ProfitMargin',
    scatter_kws={'alpha': 0.3},
    line_kws={'color': 'red'}
)
plt.title('Profit Margin vs Manufacturing Cost %')
plt.tight_layout()
plt.savefig('figures/reg_manuf.png')
plt.show()
```



[201]:
```
monthly = df.set_index('Date').resample('ME').agg({'NetSales': 'sum', 'Profit':␣
↪'sum'})
monthly['ProfitMargin'] = monthly['Profit'] / (monthly['NetSales'] + eps)

plt.figure(figsize=(12, 6))
monthly['NetSales'].plot(kind='bar', alpha=0.6, color='skyblue',␣
↪label='NetSales')
monthly['Profit'].plot(secondary_y=True, color='red', label='Profit')

plt.title('Monthly NetSales (Bar) and Profit (Line)')
plt.xlabel('Month')
plt.ylabel('NetSales')
plt.tight_layout()
plt.savefig('figures/monthly_net_profit.png')
plt.show()
```



[209]:
```
def run_ols(df, features, target='ProfitMargin'):
    X = df[features].copy()
    X = sm.add_constant(X)
    y = df[target]
    return sm.OLS(y, X, missing='drop').fit()
features = ['ManufacturingCost_pct', 'FreightCost_pct', 'COGS_pct',␣
 ↪'Discount_pct']
model = run_ols(df, features)
print(model.summary())
with open('figures/ols_summary.txt', 'w') as f:
    f.write(model.summary().as_text())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           ProfitMargin   R-squared:                       0.771
Model:                            OLS   Adj. R-squared:                  0.771
Method:                 Least Squares   F-statistic:                     1261.
Date:                Sun, 31 Aug 2025   Prob (F-statistic):               0.00
Time:                        00:13:07   Log-Likelihood:                 2609.9
No. Observations:                1500   AIC:                            -5210.
Df Residuals:                    1495   BIC:                            -5183.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
=========
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---------
```

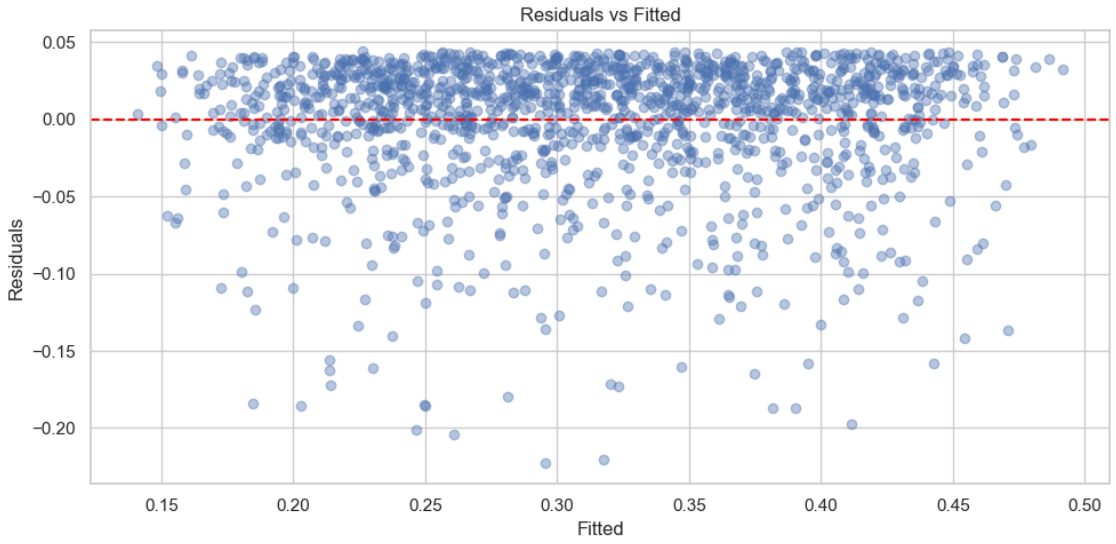
```
const                      0.9554      0.009    101.469      0.000       0.937
0.974
ManufacturingCost_pct     -0.9848      0.048    -20.374      0.000      -1.080
-0.890
FreightCost_pct           -1.0145      0.077    -13.209      0.000      -1.165
-0.864
COGS_pct                  -0.9962      0.015    -65.458      0.000      -1.026
-0.966
Discount_pct              -0.0308      0.048     -0.636      0.525      -0.126
0.064
==============================================================================
Omnibus:                      605.083   Durbin-Watson:                   1.950
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2329.807
Skew:                          -1.973   Prob(JB):                         0.00
Kurtosis:                       7.659   Cond. No.                         79.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[213]:
```
fy_models = {}
for fy_year, grp in df.groupby('FiscalYear'):
    if len(grp) < 30:
        continue
    m = run_ols(grp, features)
    fy_models[fy_year] = m.rsquared
print('Per-FY R-squared:')
for k in sorted(fy_models.keys()):
    print(k, round(fy_models[k], 3))
```

```
Per-FY R-squared:
2017 0.697
2018 0.758
2019 0.798
2020 0.784
2021 0.79
2022 0.788
2023 0.758
2024 0.82
```

[225]:
```
resid = model.resid
fitted = model.fittedvalues

# Residuals vs Fitted Plot
plt.figure()
plt.scatter(fitted, resid, alpha=0.4)
```

```
plt.axhline(0, linestyle='--', color='red')
plt.xlabel('Fitted')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted')
plt.tight_layout()
plt.savefig('figures/resid_vs_fitted.png')
plt.show()
```



[163]:
```
clean_path = 'cleaned_sales_data.csv'
df.to_csv(clean_path, index=False)
print('Saved cleaned dataset to', clean_path)
```

```
Saved cleaned dataset to cleaned_sales_data.csv
```

[ ]:
```

```