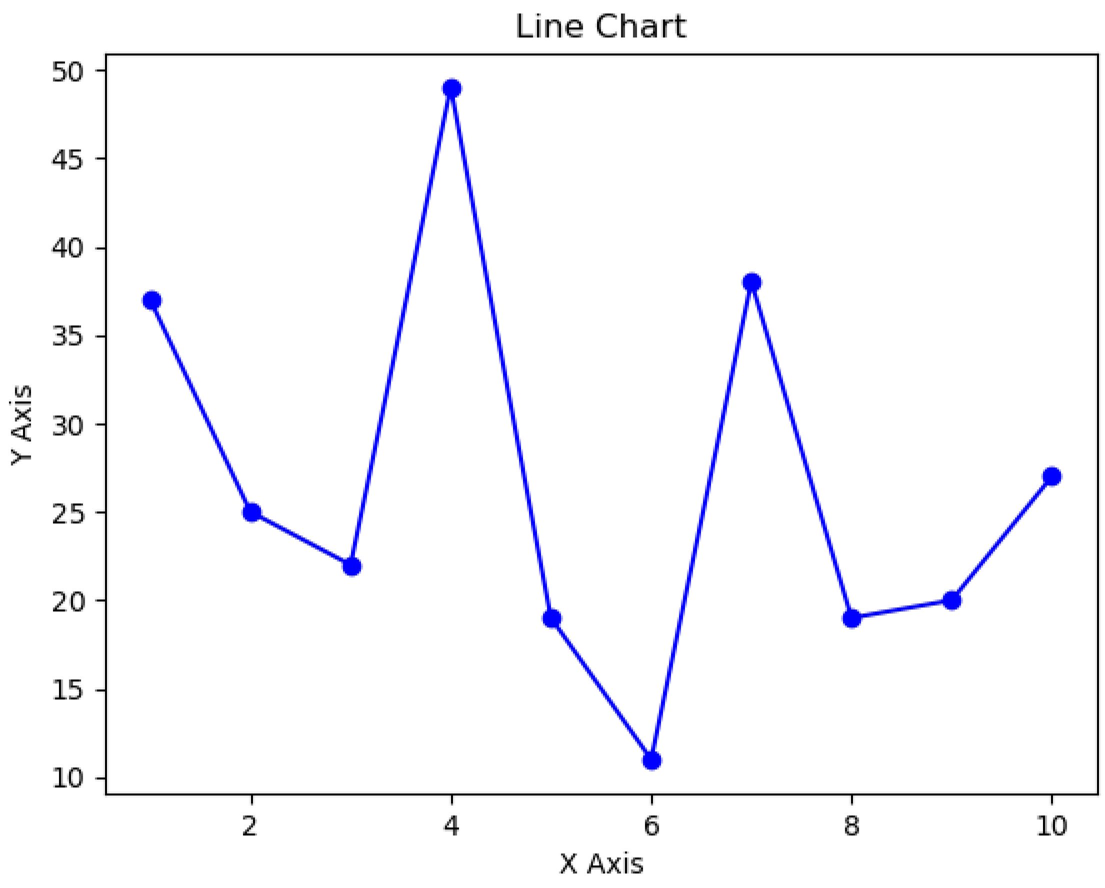
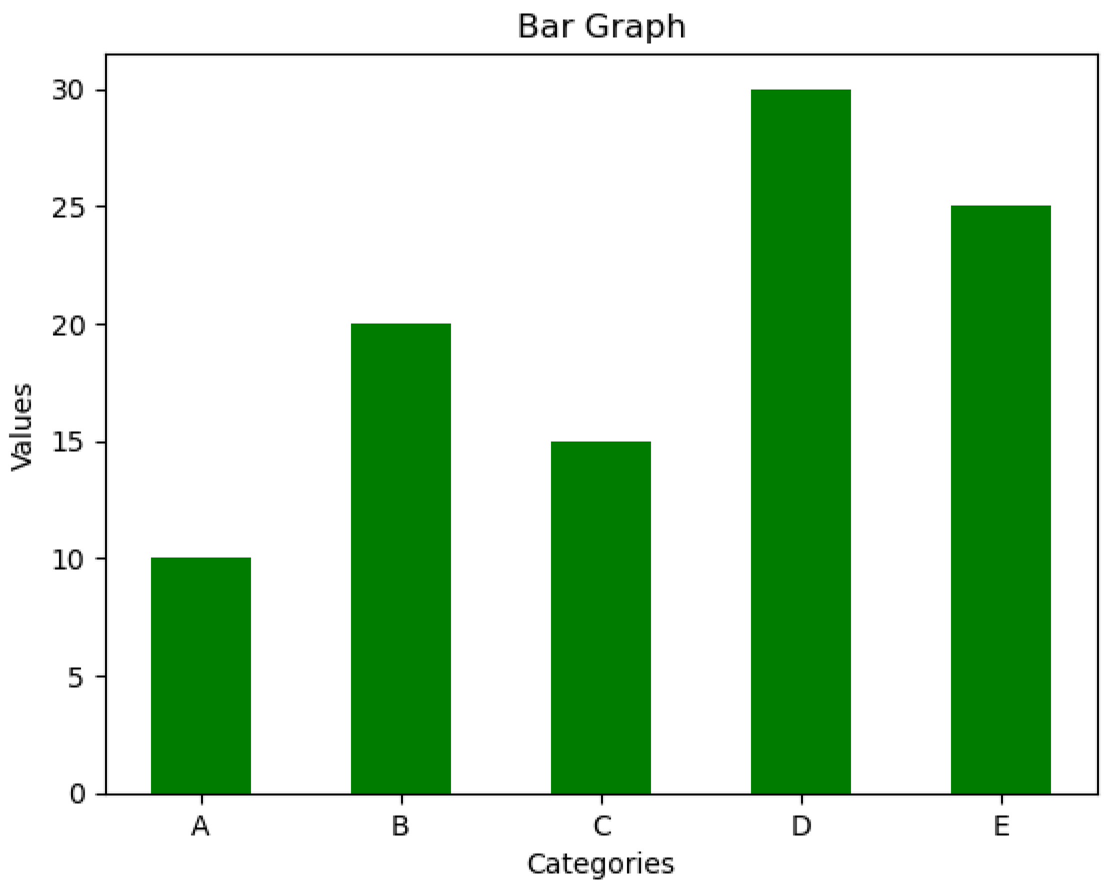
In [19]:
```
import pandas as pd
df = pd.read_excel("IPL sample data.xlsx")
#  first few rows
print(df.head())
```

```
    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  
```

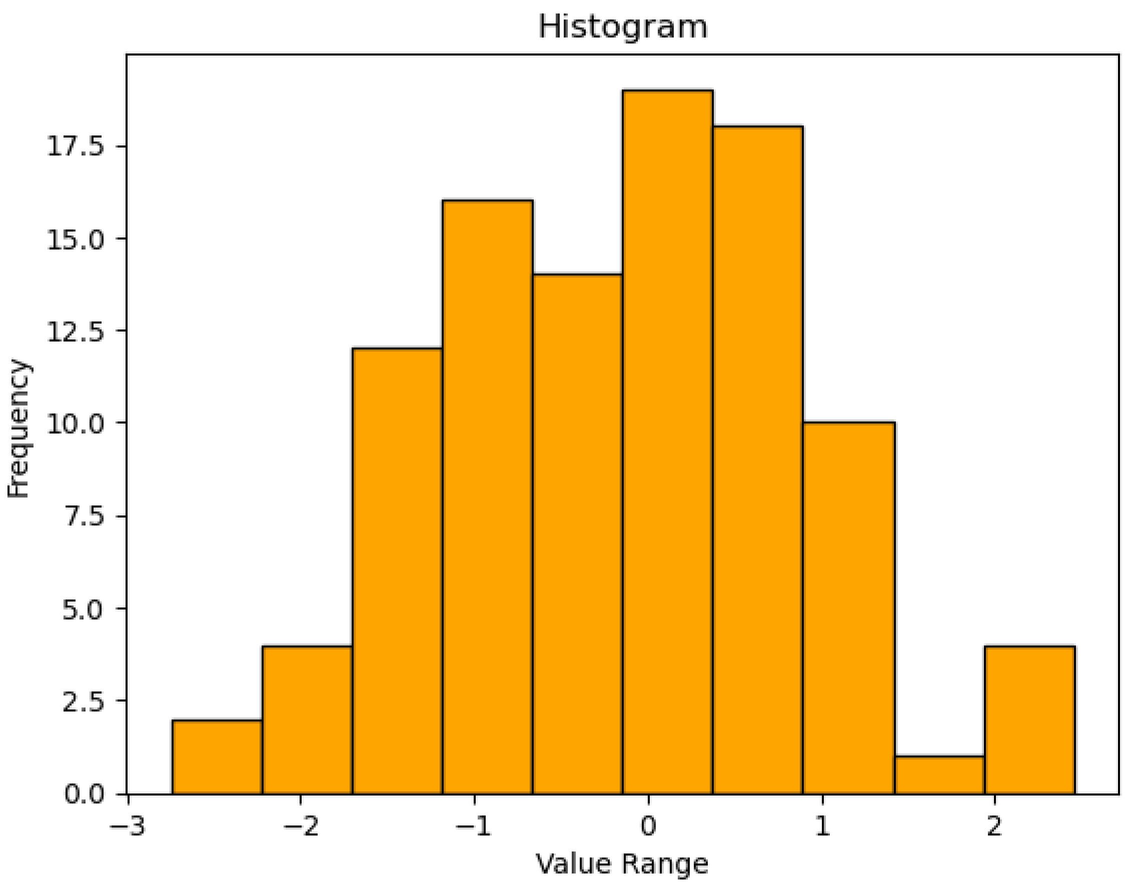
In [6]:
```
# LINE CHART
## Description
# Clearly shows trends and patterns over continuous data.
# Can plot multiple lines on the same graph for comparison.
# Supports custom markers, line styles, and colors.

import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 11)
y = np.random.randint(10, 50, 10)
plt.plot(x, y, color='blue', marker='o')
plt.title("Line Chart")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.show()
```



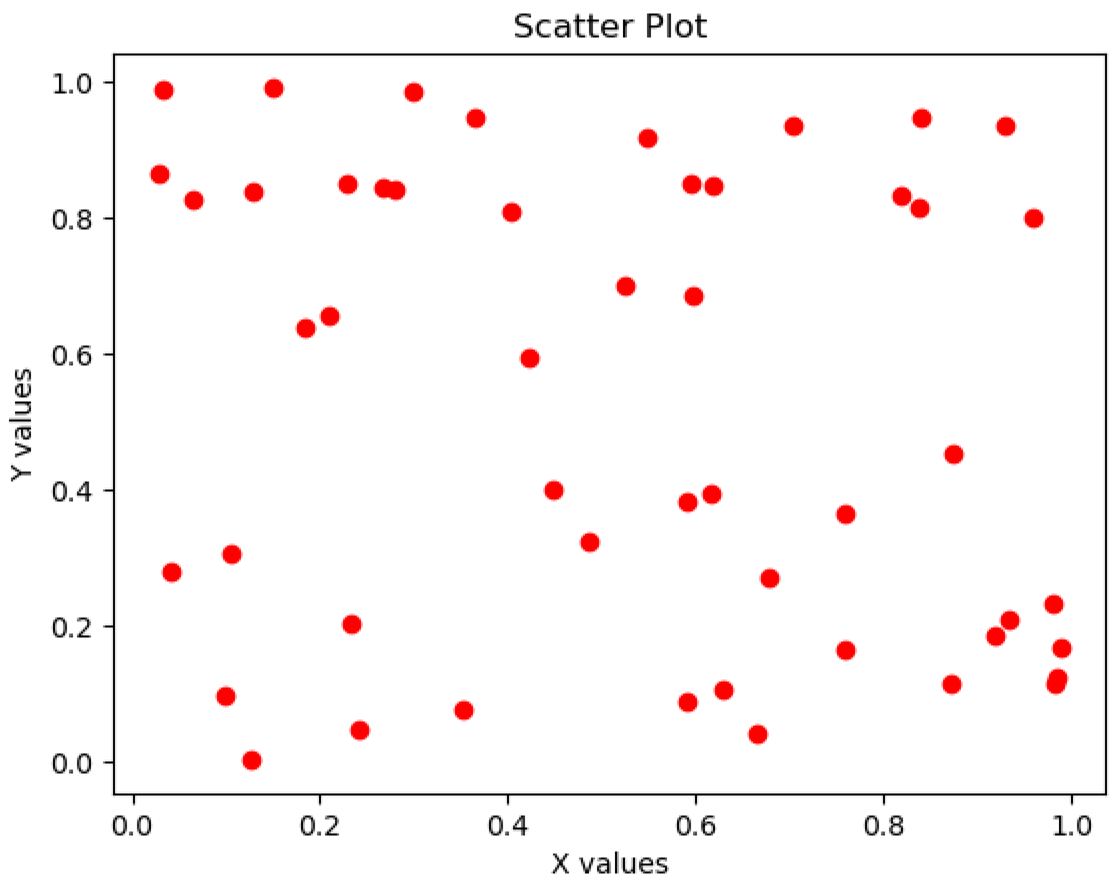
In [7]:
```
# BARGRAPH
## Decription
# Makes it easy to compare different groups or items.
# You can change the bar colors, width, and even make them sideways.
# Useful when you want to show which category is bigger or smaller.

import matplotlib.pyplot as plt
categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 15, 30, 25]
plt.bar(categories, values, color='green', width=0.5)
plt.title("Bar Graph")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
```



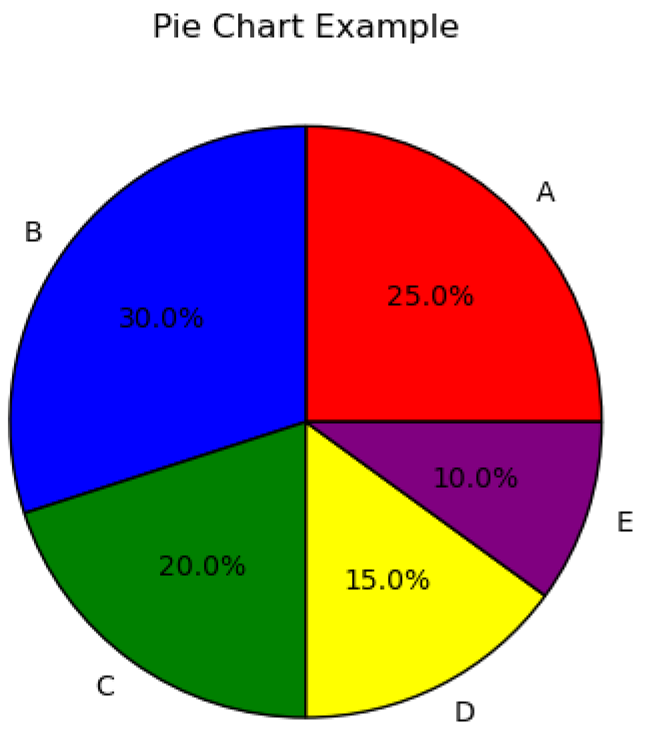
In [8]:
```
# HISTOGRAM
## Description
# Shows how your data is spread out.
# You can divide it into parts (bins) to see details clearly.
# Helps to quickly spot common values or unusual ones.

import matplotlib.pyplot as plt
import numpy as np
data = np.random.randn(100)
plt.hist(data, bins=10, color='orange', edgecolor='black')
plt.title("Histogram")
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.show()
```



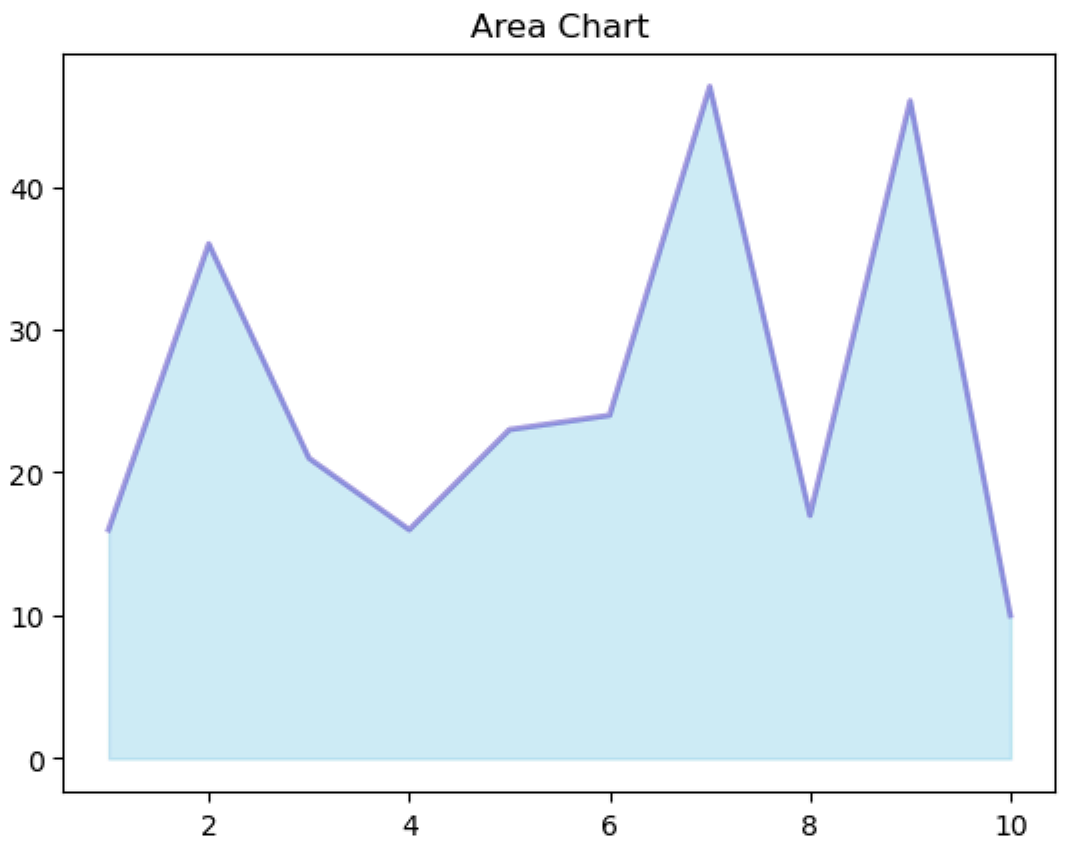
In [9]:
```
# SCATTERPLOT
## Description
# Good for showing how two things are related.
# Each dot shows a pair of values.
# Lets you see patterns, clusters, or points that don’t fit in.

import matplotlib.pyplot as plt
import numpy as np
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x, y, color='red')
plt.title("Scatter Plot")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.show()
```



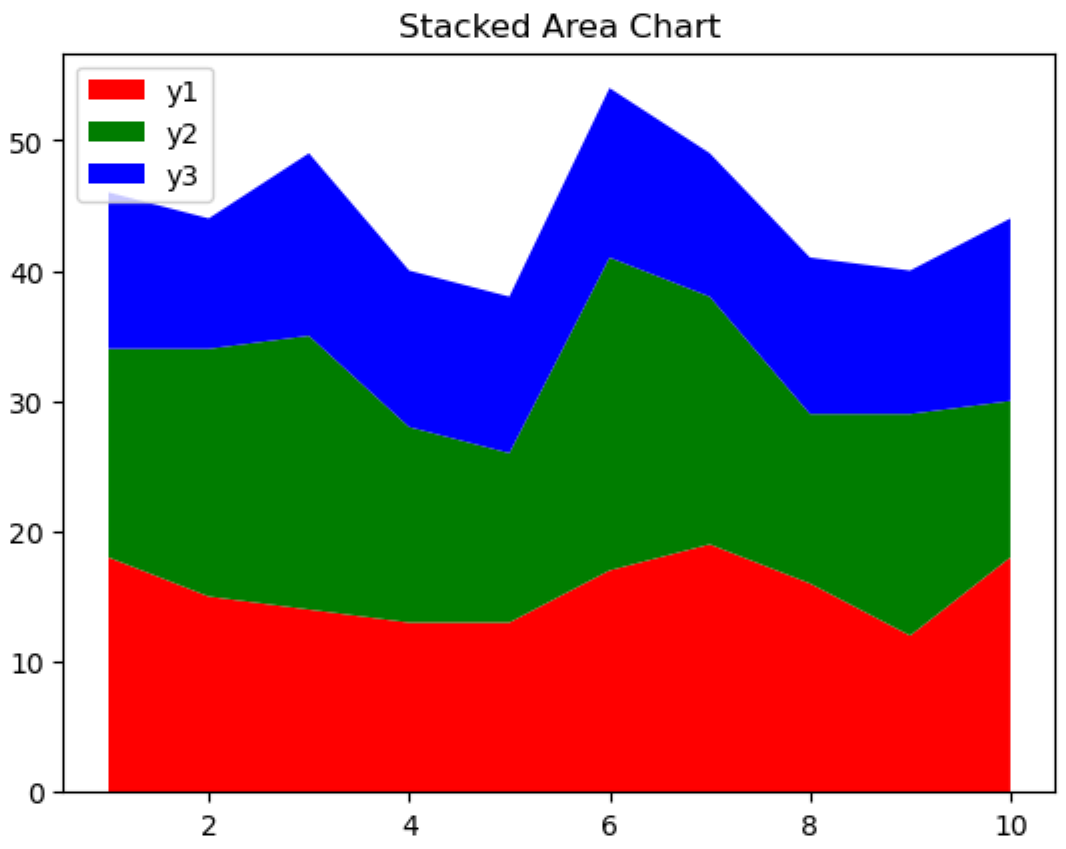
In [10]:
```
# PIECHART
## Description
# Shows how a whole is divided into parts.
#Percentages are displayed inside for easy understanding.
#Colors make it easy to compare different sections.

import matplotlib.pyplot as plt
sizes = [25, 30, 20, 15, 10]
labels = ['A', 'B', 'C', 'D', 'E']
colors = ['red', 'blue', 'green', 'yellow', 'purple']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors, wedgeprops={'edgeco
plt.title("Pie Chart Example")
plt.show()
```



In [11]:
```python
# AREA CHART
## Description
# Looks like a line chart but with the space under the line filled in.
# Good for showing growth or totals.
# More eye-catching when you want to highlight volume.

import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 11)
y = np.random.randint(10, 50, 10)
plt.fill_between(x, y, color="skyblue", alpha=0.4)
plt.plot(x, y, color="Slateblue", alpha=0.6, linewidth=2)
plt.title("Area Chart")
plt.show()
```

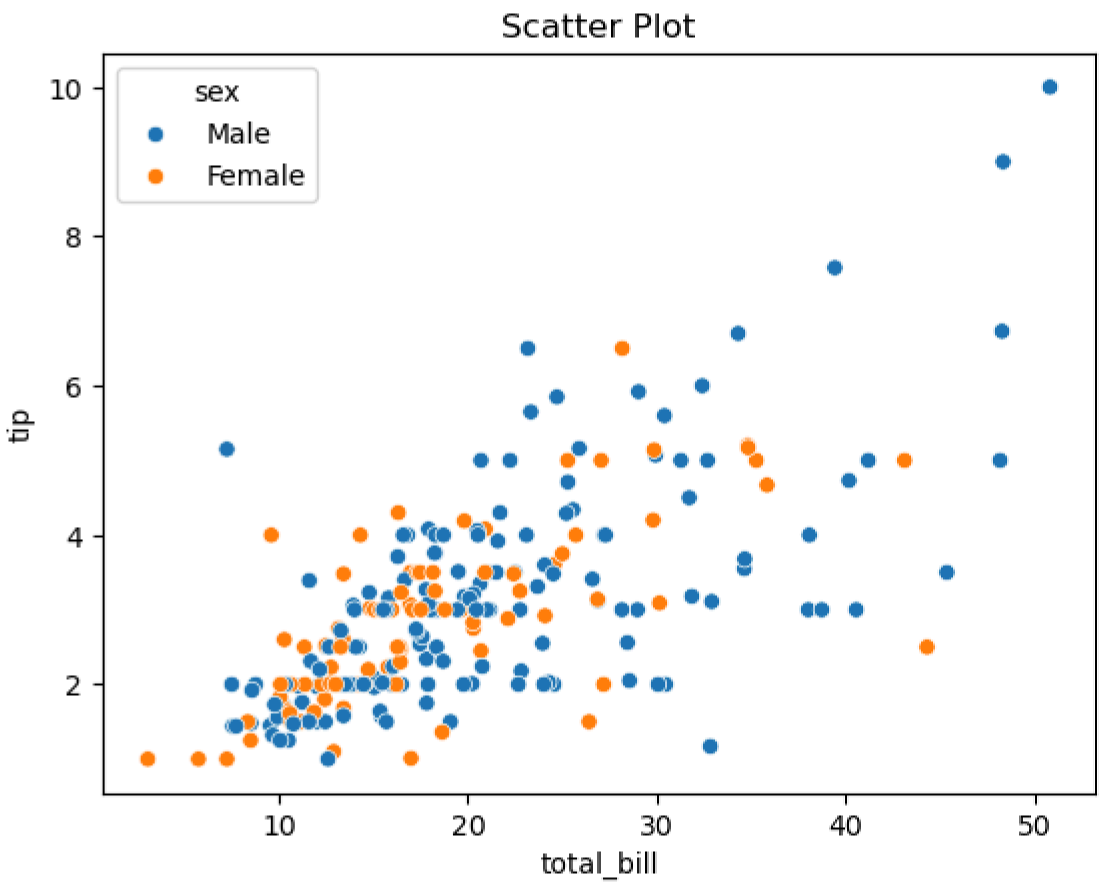


```
In [12]: # STACK PLOT AREA CHART
## Description
# Adds up different data groups into one chart.
# Shows both total and individual contributions.
# Helpful when you want to compare parts to the whole over time.

import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 11)
y1 = np.random.randint(10, 20, 10)
y2 = np.random.randint(10, 30, 10)
y3 = np.random.randint(10, 15, 10)
plt.stackplot(x, y1, y2, y3, labels=['y1','y2','y3'], colors=['red','green','blue']
plt.legend(loc='upper left')
plt.title("Stacked Area Chart")
plt.show()
```

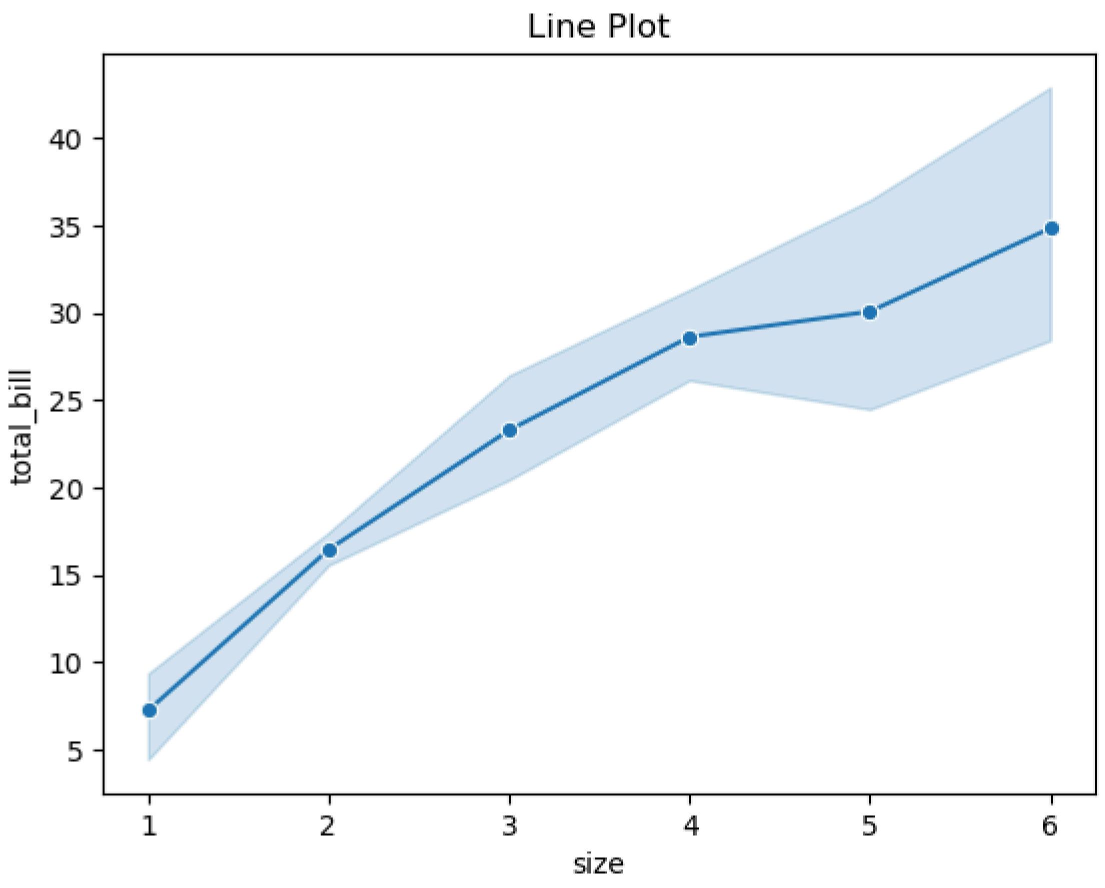


In [13]:
```python
# SEABORN PLOTS
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample Data
df = sns.load_dataset("tips")
```

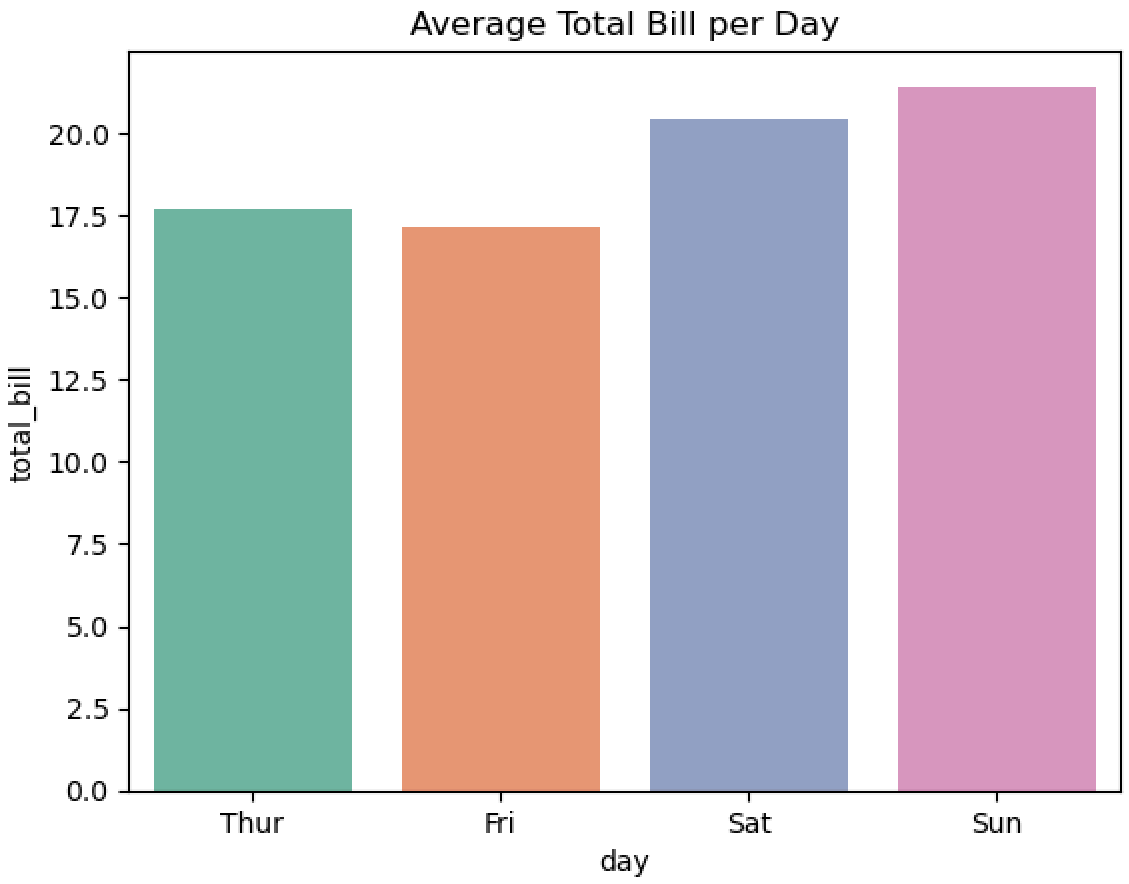
In [14]:
```python
# SCATTER PLOT
## Description
# Shows the relationship between two things, like bill amount and tips.
# Different colors can show categories like male/female.
# Easy to read because it automatically adds a legend.

sns.scatterplot(x="total_bill", y="tip", hue="sex", data=df)
plt.title("Scatter Plot")
plt.show()
```



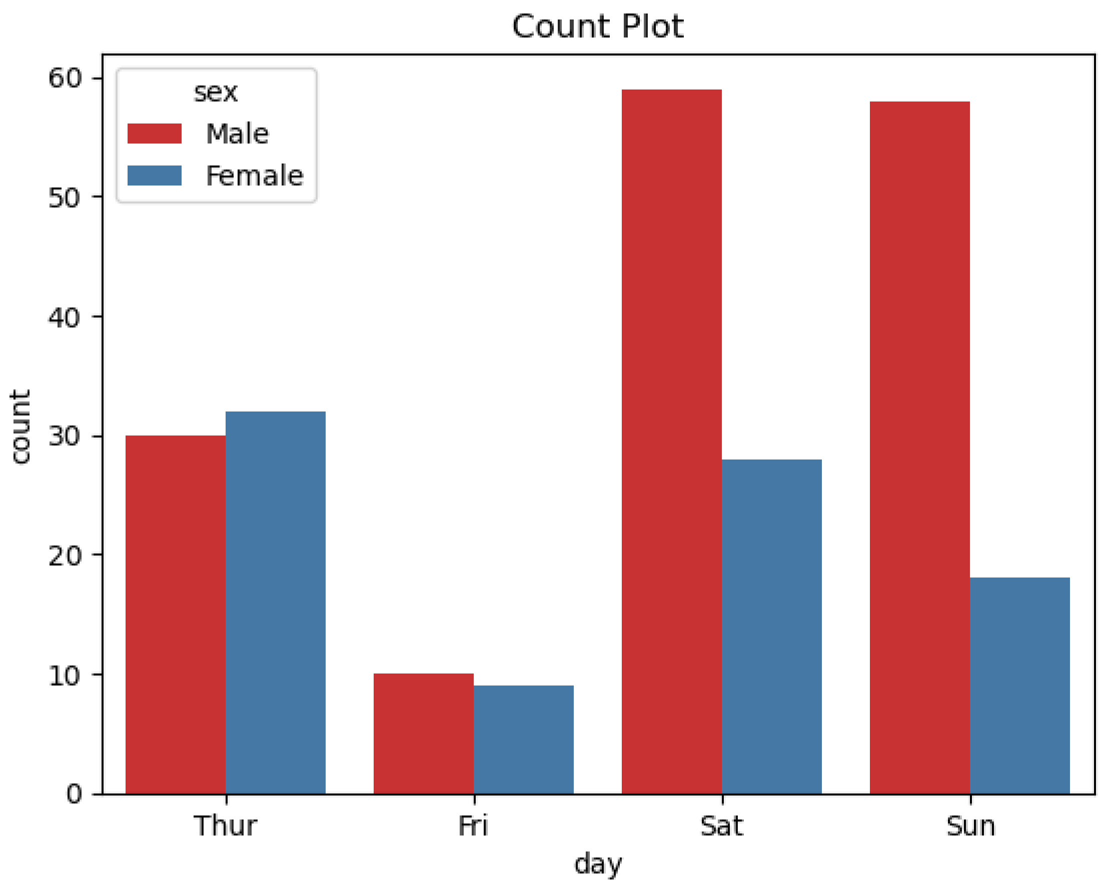
In [15]:
```
# LINEPLOT
## Description
# Simple way to see trends as data changes.
# It can even show a shaded area to suggest confidence levels.
# Great for showing patterns without too much code.

sns.lineplot(x="size", y="total_bill", data=df, marker="o")
plt.title("Line Plot")
plt.show()
```



In [31]:
```
# BAR PLOT
## Description
# Automatically shows the average value for each category.
# Can include error bars to indicate variation in data.
# Easy to customize with colors (palette) and groups (hue).

import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("tips")
sns.barplot(x="day", y="total_bill", hue="day", data=df, palette="Set2", errorbar=N
plt.title("Average Total Bill per Day")
plt.show()
```

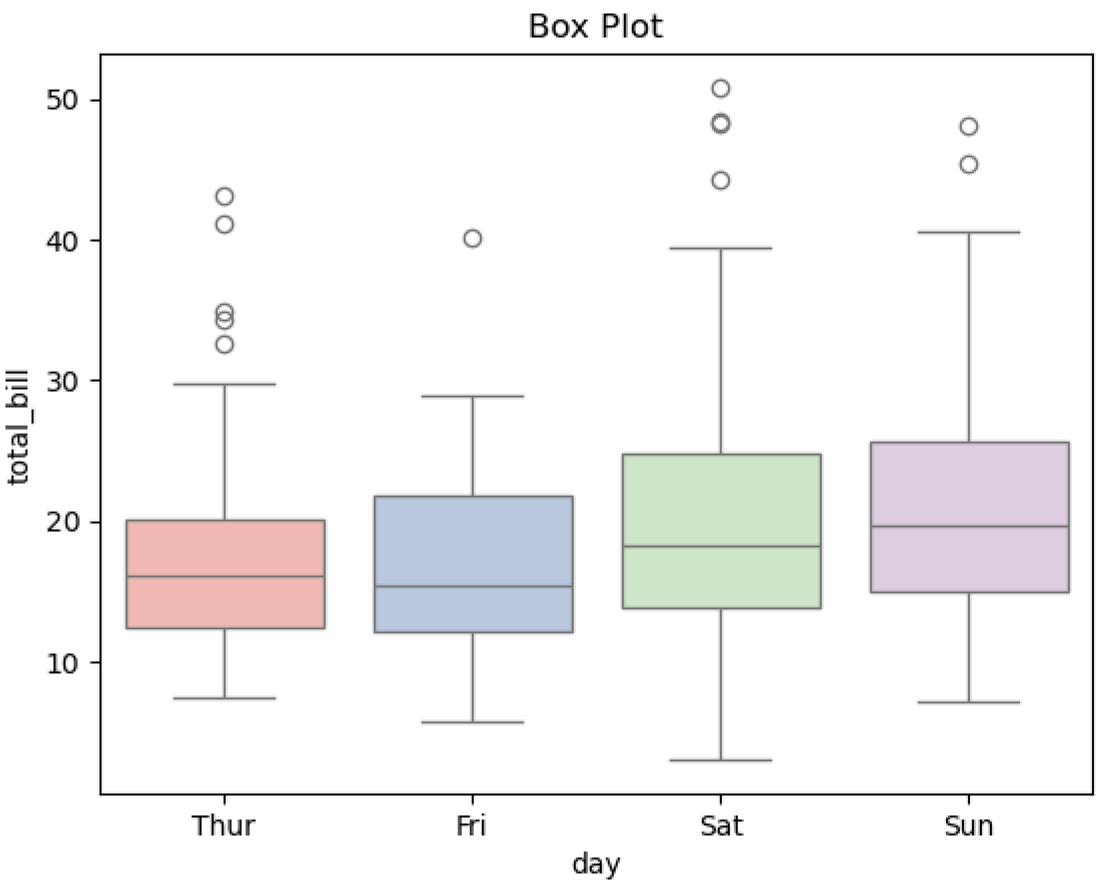


In [16]:
```
# COUNT PLOT
## Description
# Quickly shows how many times something happens.
# Can split the counts by another category (like male/female).
# Perfect for seeing which group is most common.

sns.countplot(x="day", hue="sex", data=df, palette="Set1")
plt.title("Count Plot")
plt.show()
```

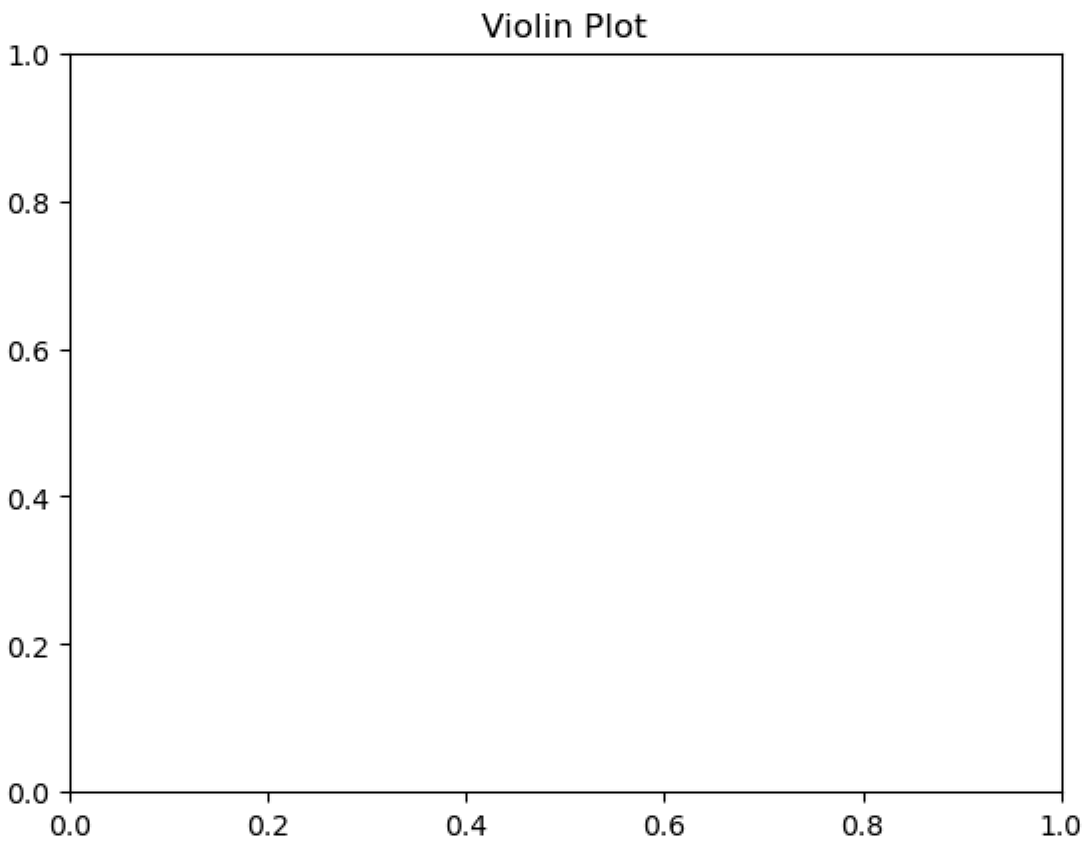


In [33]:
```python
# BOX PLOT
## Description
# Shows the middle value, spread, and outliers.
# You can easily spot if data is evenly spread or not.
# Good for comparing across categories.

sns.boxplot(x="day", y="total_bill", data=df, palette="Pastel1")
plt.title("Box Plot")
plt.show()
```

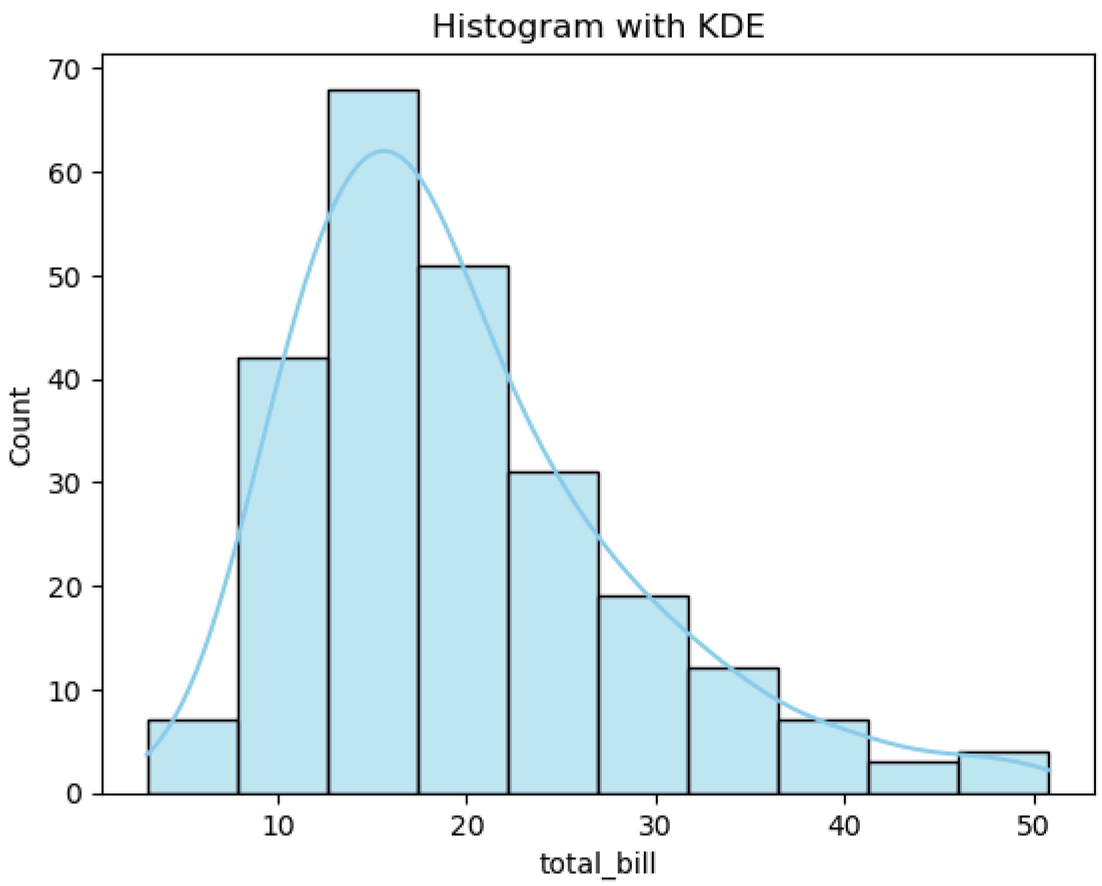
```
C:\Users\KOUSHITHA KETHINENI\AppData\Local\Temp\ipykernel_33000\365244794.py:7: Futu
reWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="day", y="total_bill", data=df, palette="Pastel1")
```



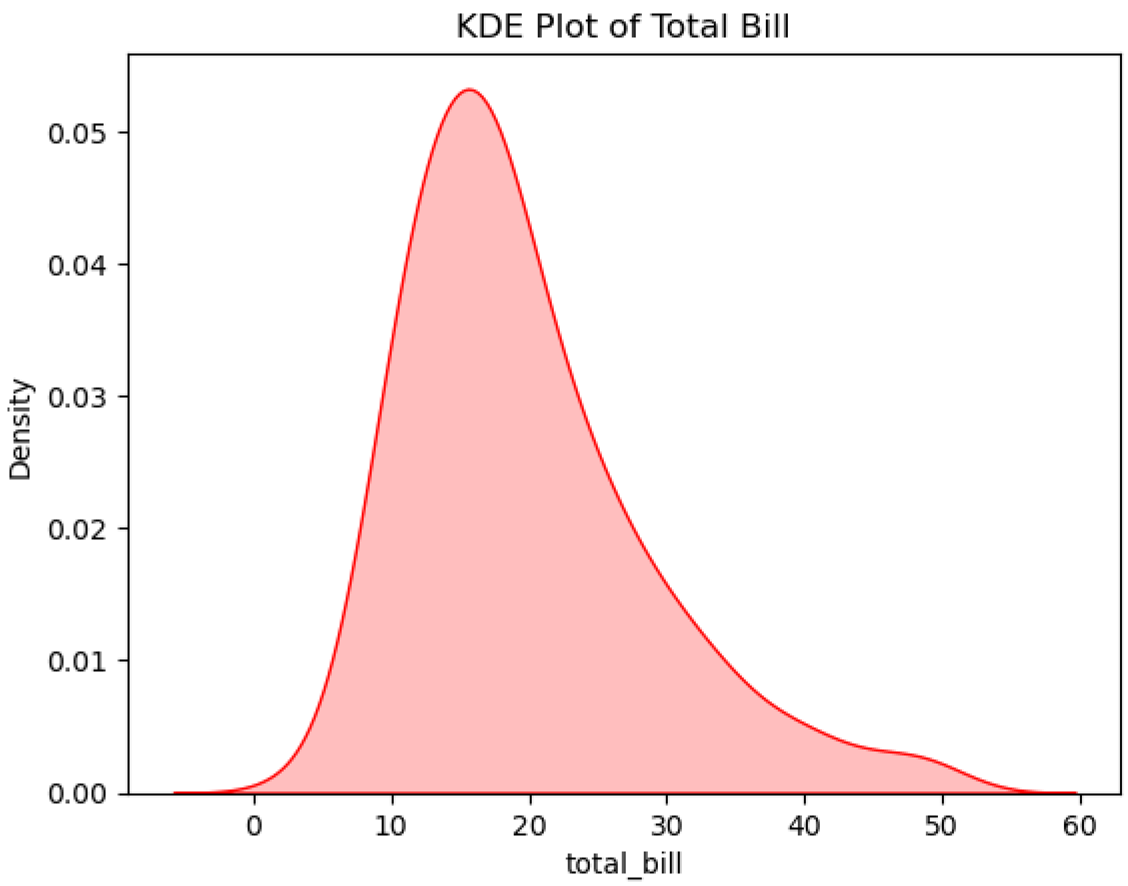
In [19]:
```
# violin plot
## Description
# Similar to a box plot, but also shows the shape of the data.
# Lets you see how the values are spread out.
# Helpful when data has more than one peak.

plt.title("Violin Plot")
plt.show()
```



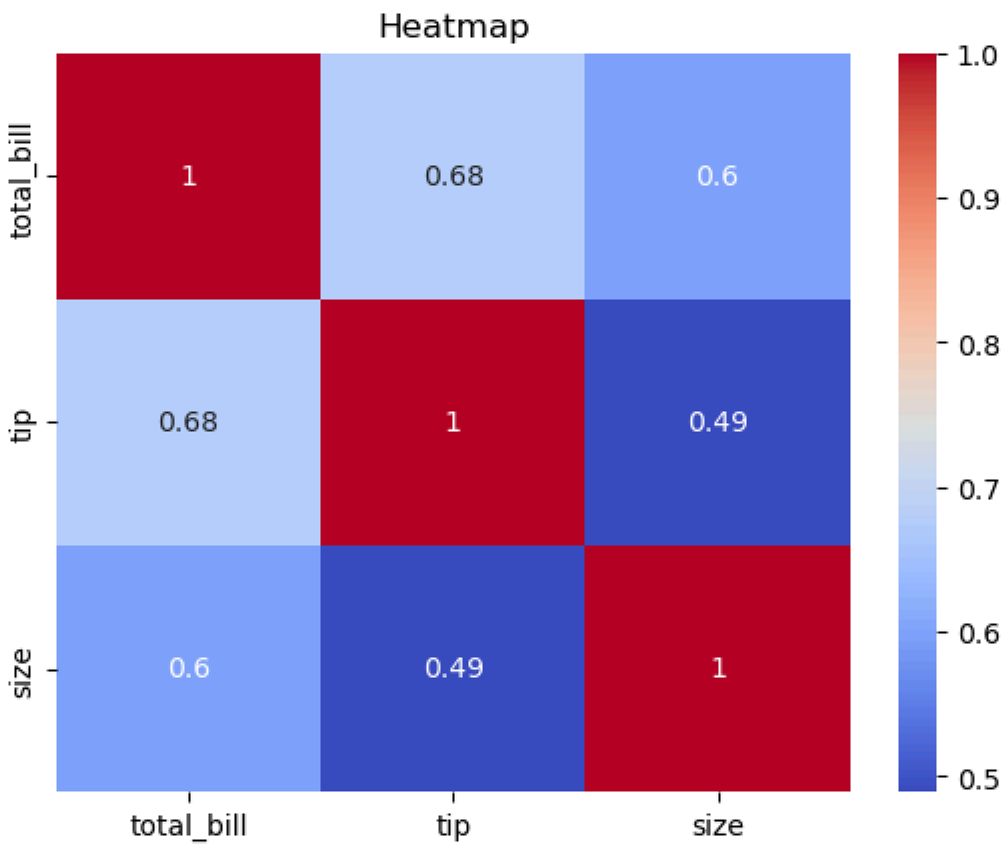
In [20]:
```
# HISTPLOT
## Description
# Shows how often values appear.
# Can also draw a smooth line (KDE) on top.
# Looks neat and clean by default.

sns.histplot(df["total_bill"], bins=10, kde=True, color="skyblue")
plt.title("Histogram with KDE")
plt.show()
```



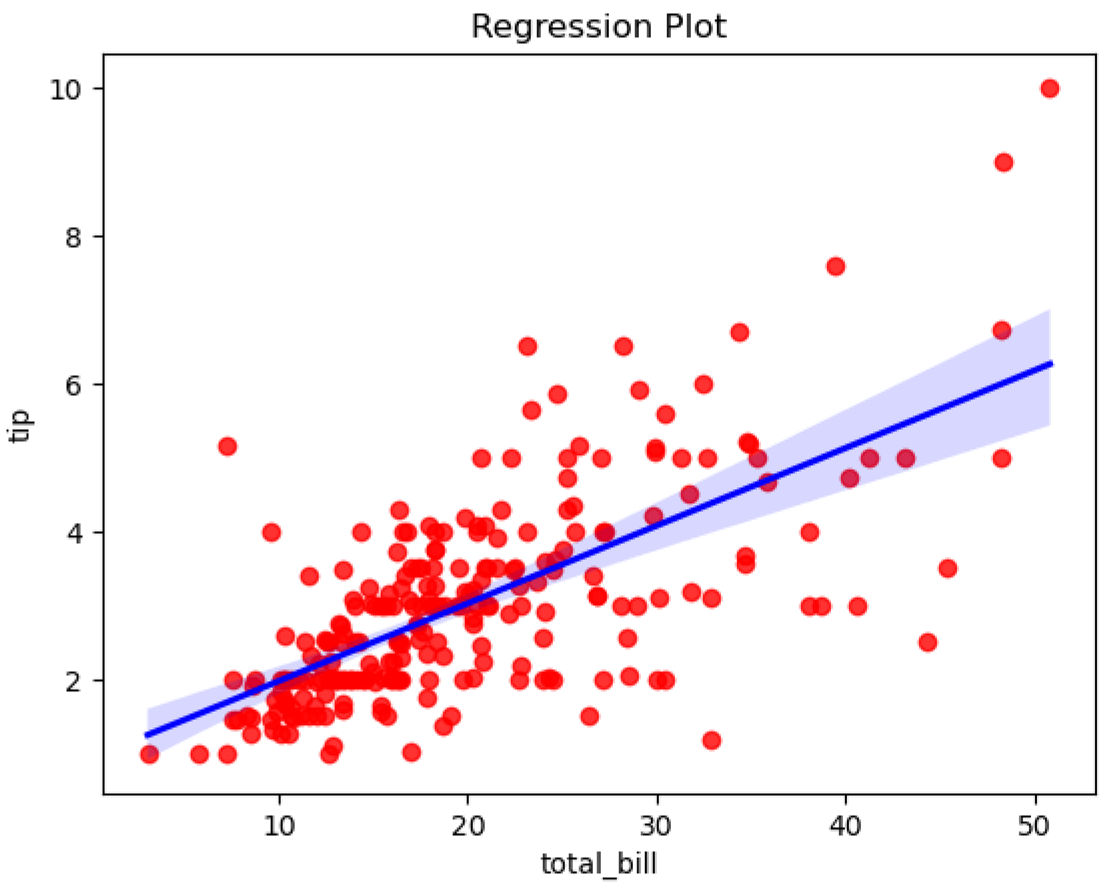
```
In [30]: # KDE PLOT
         ## Description
         # Draws a smooth curve to show data distribution.
         # Can show multiple groups on the same chart.
         # Great for comparing how two or more sets of data behave.

         import seaborn as sns
         import matplotlib.pyplot as plt
         df = sns.load_dataset("tips").dropna(subset=["total_bill"])
         sns.kdeplot(x="total_bill", data=df, fill=True, color="red")
         plt.title("KDE Plot of Total Bill")
         plt.show()
```



In [22]:
```
# HEAT MAP
## Description
# Uses colors to show how strongly things are related.
# Bright and dark shades make differences stand out.
# Very easy to understand at a glance.

corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.title("Heatmap")
plt.show()
```



In [23]:
```
# REGRESSION PLOT
## Description
# Combines a scatter plot with a best-fit line.
# Shows how one thing affects another.
# Great for predicting trends.

sns.regplot(x="total_bill", y="tip", data=df, scatter_kws={"color":"red"}, line_kws
plt.title("Regression Plot")
plt.show()
```



In [ ]:
```
# Comparison between matplotlib and Seaborn
| Feature          | Matplotlib                                | Seaborn
|------------------|-------------------------------------------|---------------------
| Ease of Use      | Needs more code, but gives full control.  | Very simple, works w
| Customization    | You can change almost everything.         | Less customizable, b
| Look & Feel      | Plain style unless you design it.         | Colorful and attract
| Best For         | Professional and detailed charts.         | Quick and beautiful
```

In [ ]:
```
## Resources links:
## Resources

- [Matplotlib Documentation](https://matplotlib.org/stable/users/explain/quick_star
- [Seaborn Documentation](https://seaborn.pydata.org/tutorial/introduction.html)
```