In [ ]:
```
#Python libraries for statistical analyses.
#numpy, pandas, scipy, statsmodels, scikit-learn
#matplotlib, seaborn
```

In [ ]:
```
##############numpy##################
```

In [1]:
```
#Mean and median calculation with numpy
import numpy as np

data = np.array([10, 20, 30, 40, 50, 60])

mean = np.mean(data)
median = np.median(data)

print("Mean:", mean)
print("Median:", median)
```

```
Mean: 35.0
Median: 35.0
```

In [3]:
```
#Standard deviation and variance calculation with numpy
import numpy as np

data = np.array([10, 20, 30, 40, 50, 60])

std_dev = np.std(data)
variance = np.var(data)

print("Standard Deviation:", std_dev)
print("Variance:", variance)
```

```
Standard Deviation: 17.07825127659933
Variance: 291.6666666666667
```

In [14]:
```
#Generating random numbers within a range and perform statistics on that
import numpy as np

# Generate 100 random numbers from a normal distribution with mean 0 and standard devi
data = np.random.normal(0, 1, 100)

# Compute mean and standard deviation of the generated data
mean = np.mean(data)
median = np.median(data)
std_dev = np.std(data)

print("Mean:", mean)
print("Median:", median)
print("Standard Deviation:", std_dev)
```

```
Mean: 0.036363732410518546
Median: 0.03332759549777098
Standard Deviation: 1.0381678236343312
```

In [16]:
```
#Performing Hypothesis Tests
import numpy as np
from scipy.stats import ttest_ind

# Generate two sets of random data
```

```
data1 = np.random.normal(0, 1, 100)
data2 = np.random.normal(1, 1, 100)

# Perform a two-sample t-test
t_stat, p_value = ttest_ind(data1, data2)

print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
T-statistic: -8.122455796698768
P-value: 4.820743545905614e-14
```

In [ ]:

```
##############pandas###################
```

In [17]:

```
#Descriptive statistics
import pandas as pd

# Create a DataFrame with some sample data
data = {'A': [1, 2, 3, 4],
        'B': [10, 20, 30, 40],
        'C': [100, 200, 300, 400]}
df = pd.DataFrame(data)

# Calculate mean, median, and standard deviation
print("Mean:")
print(df.mean())

print("\nMedian:")
print(df.median())

print("\nStandard Deviation:")
print(df.std())
```

```
Mean:
A      3.0
B     30.0
C    300.0
dtype: float64

Median:
A      3.0
B     30.0
C    300.0
dtype: float64

Standard Deviation:
A      1.581139
B     15.811388
C    158.113883
dtype: float64
```

In [18]:

```
#Correlation and Covariance
import pandas as pd

# Create a DataFrame with some sample data
data = {'A': [1, 2, 3, 4],
        'B': [10, 20, 30, 40],
        'C': [100, 200, 300, 400]}
df = pd.DataFrame(data)
```

```
# Calculate correlation and covariance
print("Correlation:")
print(df.corr())

print("\nCovariance:")
print(df.cov())
```

```
Correlation:
     A    B    C
A  1.0  1.0  1.0
B  1.0  1.0  1.0
C  1.0  1.0  1.0

Covariance:
            A            B             C
A    1.666667    16.666667    166.666667
B   16.666667   166.666667   1666.666667
C  166.666667  1666.666667  16666.666667
```

In [21]:

```
#GroupBy and Aggregation
import pandas as pd

# Create a DataFrame with some sample data
data = {'Category': ['A', 'B', 'C', 'B', 'A'],
        'Value': [10, 50, 20, 30, 40]}
df = pd.DataFrame(data)

# Group by 'Category' and calculate the mean
grouped_df = df.groupby('Category')
mean_values = grouped_df.mean()

print(mean_values)
```

```
          Value
Category
A          25.0
B          40.0
C          20.0
```

In [ ]:

```
###############scipy###################
```

In [23]:

```
#Descriptive Statistics
import numpy as np
from scipy import stats

data = np.array([1, 2, 3, 4])

# Compute mean
mean = np.mean(data)

# Compute median
median = np.median(data)

# Compute standard deviation
std_dev = np.std(data)

# Compute variance
variance = np.var(data)

print("Mean:", mean)
```

```python
print("Median:", median)
print("Standard Deviation:", std_dev)
print("Variance:", variance)
```

```
Mean: 2.5
Median: 2.5
Standard Deviation: 1.118033988749895
Variance: 1.25
```

In [26]:

```python
#Hypothesis Testing
from scipy import stats

# Perform a t-test
group1 = [1, 2, 3, 4, 5, 6]
group2 = [2, 4, 6, 8]
t_statistic, p_value = stats.ttest_ind(group1, group2)

print("T-statistic:", t_statistic)
print("P-value:", p_value)
```

```
T-statistic: -1.0733126291998991
P-value: 0.31443616587335843
```

In [27]:

```python
#Probability Distributions
from scipy.stats import norm

# Create a normal distribution with mean 0 and standard deviation 1
distribution = norm(0, 1)

# Compute the probability density function (PDF) at a given value
pdf_value = distribution.pdf(0.5)

# Compute the cumulative distribution function (CDF) at a given value
cdf_value = distribution.cdf(0.5)

print("PDF:", pdf_value)
print("CDF:", cdf_value)
```

```
PDF: 0.3520653267642995
CDF: 0.6914624612740131
```

In [ ]:

```python
###############statsmodels###################
```

In [29]:

```python
#Linear regression
import statsmodels.api as sm
import numpy as np

# Generate some random data
np.random.seed(0)
x = np.random.rand(100)
y = 2 * x + 1 + np.random.randn(100)

# Add constant term to the predictor variable
#sm is short for Statsmodels
X = sm.add_constant(x)

# Fit the linear regression model
#ordinary least squares (OLS) regression model
model = sm.OLS(y, X)
results = model.fit()
```

```
# Print the model summary
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.239
Model:                            OLS   Adj. R-squared:                  0.231
Method:                 Least Squares   F-statistic:                     30.79
Date:                Tue, 06 Jun 2023   Prob (F-statistic):           2.45e-07
Time:                        23:25:27   Log-Likelihood:                -141.51
No. Observations:                 100   AIC:                             287.0
Df Residuals:                      98   BIC:                             292.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.2222      0.193      6.323      0.000       0.839       1.606
x1             1.9369      0.349      5.549      0.000       1.244       2.630
==============================================================================
Omnibus:                       11.746   Durbin-Watson:                   2.083
Prob(Omnibus):                  0.003   Jarque-Bera (JB):                4.097
Skew:                           0.138   Prob(JB):                        0.129
Kurtosis:                       2.047   Cond. No.                         4.30
==============================================================================

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

In [35]:

```
#A fake timeseries dataset creation and saving the dataframe as csv file.
import numpy as np
import pandas as pd

# Set the number of data points and the frequency of the time series
num_points = 100
freq = 'D'  # Daily frequency

# Generate a sequence of dates
dates = pd.date_range(start='2023-04-01', periods=num_points, freq=freq)

# Generate random values for the time series data
values = np.random.randn(num_points)

# Create a DataFrame using the dates and values
df = pd.DataFrame({'Date': dates, 'Value': values})

# Print the generated time series dataset
print(df)

file_path = 'C:/Users/Seema Patel/df.csv'
df.to_csv(file_path, index=False)
```

```
          Date     Value
0  2023-04-01 -1.698106
1  2023-04-02  0.387280
2  2023-04-03 -2.255564
3  2023-04-04 -1.022507
4  2023-04-05  0.038631
..         ...       ...
95 2023-07-05  0.994394
96 2023-07-06  1.319137
97 2023-07-07 -0.882419
98 2023-07-08  1.128594
99 2023-07-09  0.496001

[100 rows x 2 columns]
```

In [36]:

```
#Time series analysis (to analyze and forecast data points collected over time)
#Used the dataset generated above.
#Use the seasonal_decompose function for decomposition of the time series into its tre
import pandas as pd
import statsmodels.api as sm

# Load the time series data
data = pd.read_csv('df.csv', index_col='Date', parse_dates=True)

# Perform seasonal decomposition of the time series
decomposition = sm.tsa.seasonal_decompose(data['Value'], model='additive')

# Print the decomposition results
print(decomposition.trend)
print(decomposition.seasonal)
print(decomposition.resid)
print(decomposition.observed)

# Fit an ARIMA model to the time series data
model = sm.tsa.ARIMA(data['Value'], order=(1, 0, 1))
results = model.fit()

# Print the model summary
print(results.summary())
```

```
Date
2023-04-01         NaN
2023-04-02         NaN
2023-04-03         NaN
2023-04-04   -1.027499
2023-04-05   -0.995174
                ...
2023-07-05    0.473312
2023-07-06    0.583550
2023-07-07         NaN
2023-07-08         NaN
2023-07-09         NaN
Name: trend, Length: 100, dtype: float64
Date
2023-04-01   -0.508611
2023-04-02    0.434586
2023-04-03   -0.133200
2023-04-04    0.339855
2023-04-05    0.134644
                ...
2023-07-05    0.134644
2023-07-06   -0.089452
2023-07-07   -0.177822
2023-07-08   -0.508611
2023-07-09    0.434586
Name: seasonal, Length: 100, dtype: float64
Date
2023-04-01         NaN
2023-04-02         NaN
2023-04-03         NaN
2023-04-04   -0.334863
2023-04-05    0.899161
                ...
2023-07-05    0.386439
2023-07-06    0.825039
2023-07-07         NaN
2023-07-08         NaN
2023-07-09         NaN
Name: resid, Length: 100, dtype: float64
Date
2023-04-01   -1.698106
2023-04-02    0.387280
2023-04-03   -2.255564
2023-04-04   -1.022507
2023-04-05    0.038631
                ...
2023-07-05    0.994394
2023-07-06    1.319137
2023-07-07   -0.882419
2023-07-08    1.128594
2023-07-09    0.496001
Name: Value, Length: 100, dtype: float64
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Value   No. Observations:                  100
Model:                 ARIMA(1, 0, 1)   Log Likelihood                -144.963
Date:                Tue, 06 Jun 2023   AIC                            297.927
Time:                        23:37:24   BIC                            308.348
Sample:                    04-01-2023   HQIC                           302.144
                         - 07-09-2023
```

```
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0885      0.117     -0.755      0.450      -0.318       0.141
ar.L1          0.3475      1.046      0.332      0.740      -1.702       2.398
ma.L1         -0.2666      1.094     -0.244      0.807      -2.410       1.877
sigma2         1.0632      0.158      6.709      0.000       0.753       1.374
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 0.91
Prob(Q):                              0.96   Prob(JB):                         0.63
Heteroskedasticity (H):               1.15   Skew:                            -0.23
Prob(H) (two-sided):                  0.68   Kurtosis:                         2.91
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
C:\Users\Seema Patel\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: No frequency information was provided, so inferred frequency D will
be used.
  self._init_dates(dates, freq)
C:\Users\Seema Patel\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: No frequency information was provided, so inferred frequency D will
be used.
  self._init_dates(dates, freq)
C:\Users\Seema Patel\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: No frequency information was provided, so inferred frequency D will
be used.
  self._init_dates(dates, freq)
```

In [ ]:

```
##############scikit-Learn##################
```

In [39]:

```
#Linear Regression
from sklearn.linear_model import LinearRegression

# Create a linear regression model
model = LinearRegression()

# Prepare the data
X = [[5], [7], [9], [11], [8]]  # Input features
y = [1, 3, 4, 6, 8]  # Target variable

# Fit the model to the data
model.fit(X, y)

# Make predictions
X_new = [[4], [5]]  # New input features
predictions = model.predict(X_new)

# Print the predictions
print(predictions)
```

```
[1.2 2. ]
```

In [43]:

```
#K-Means Clustering
from sklearn.cluster import KMeans
import numpy as np

# Generate random data
```

```
np.random.seed(0)
X = np.random.rand(30, 3)

# Create a K-Means clustering model
model = KMeans(n_clusters=3)

# Fit the model to the data
model.fit(X)

# Get the cluster labels
labels = model.labels_

# Print the cluster labels
print(labels)
```

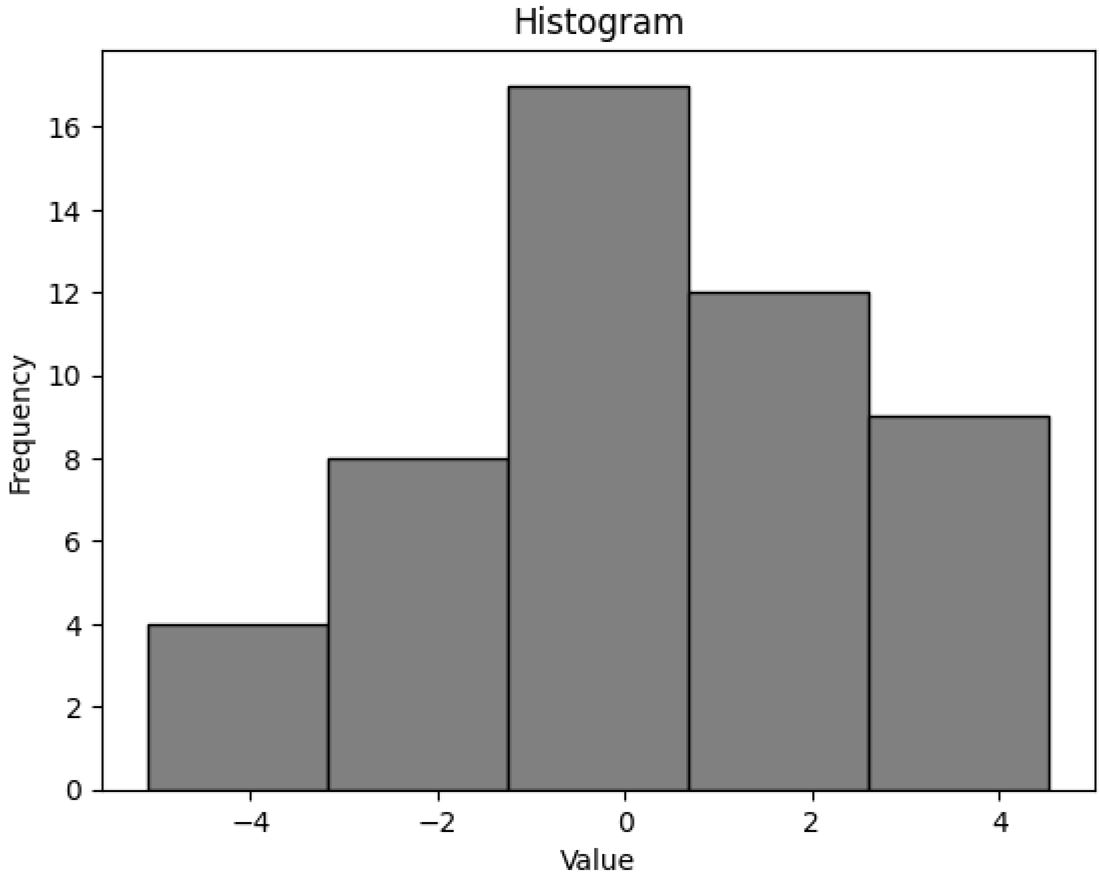
```
[1 1 1 2 2 0 1 1 2 1 2 0 1 0 0 0 0 2 0 2 0 0 0 2 1 2 0 0 0 1]
```

In [ ]:

```
##############matplotlib###################
```

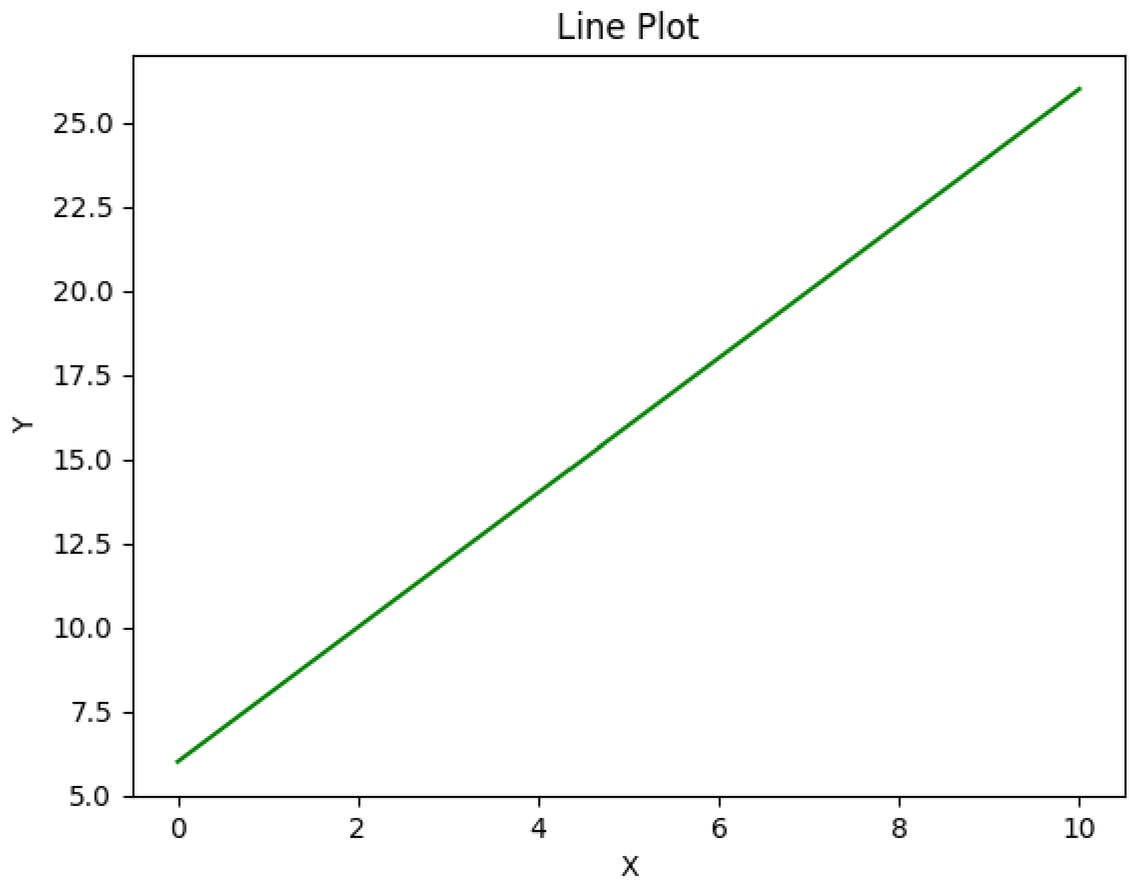
In [56]:

```
#statistical visualization (Histogram)
import matplotlib.pyplot as plt
import numpy as np

# Generate random data
np.random.seed(0)
x = np.random.normal(0, 2, 50)  # Random values from a standard normal distribution

# Plot histogram
plt.hist(x, bins=5, color='grey', edgecolor='black')

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')

# Show the plot
plt.show()
```



In [55]:

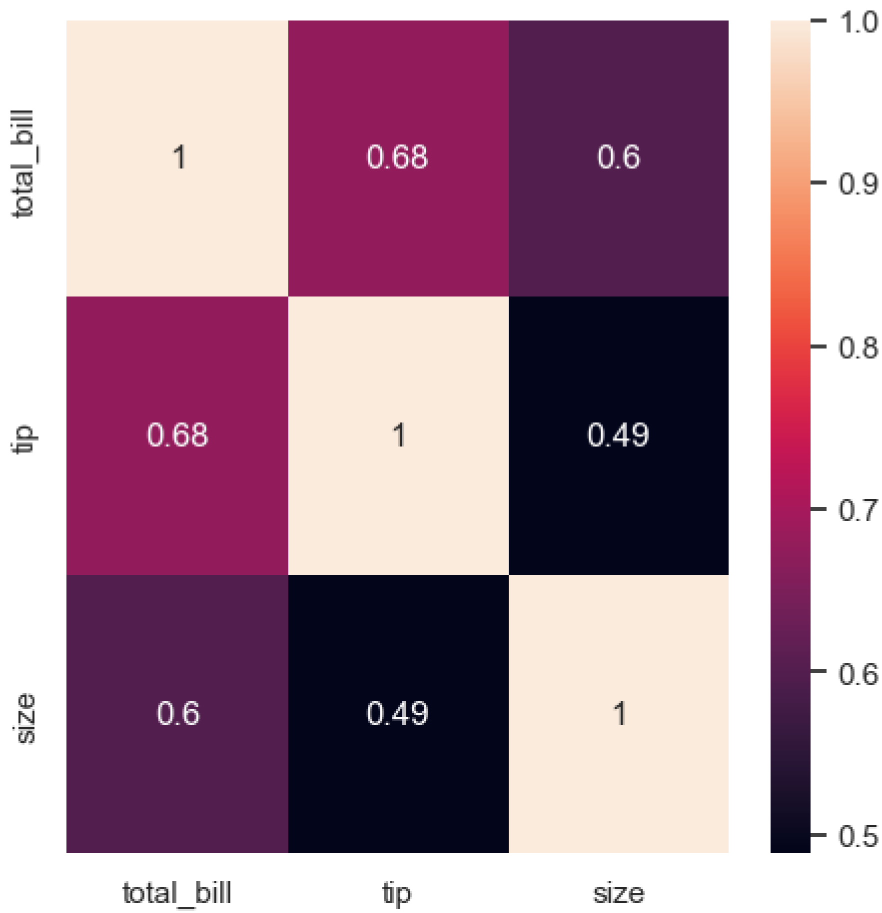
```
import matplotlib.pyplot as plt
import numpy as np

# Generate x values from 0 to 10
x = np.linspace(0, 10, 50)

# Generate y values using a linear function
y = 2 * x + 6

# Plot the line
plt.plot(x, y, color='green')

# Add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Line Plot')

# Show the plot
plt.show()
```

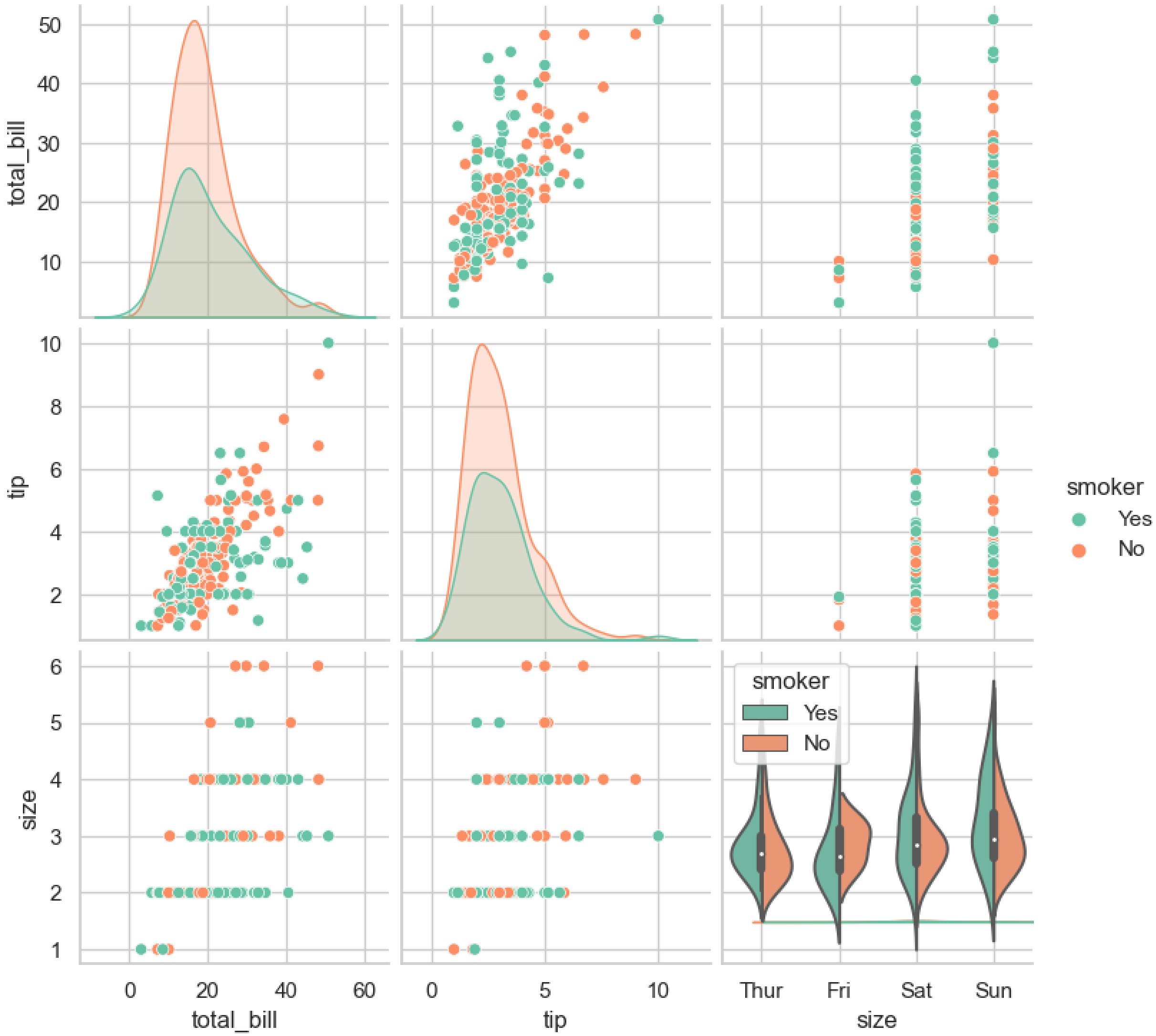


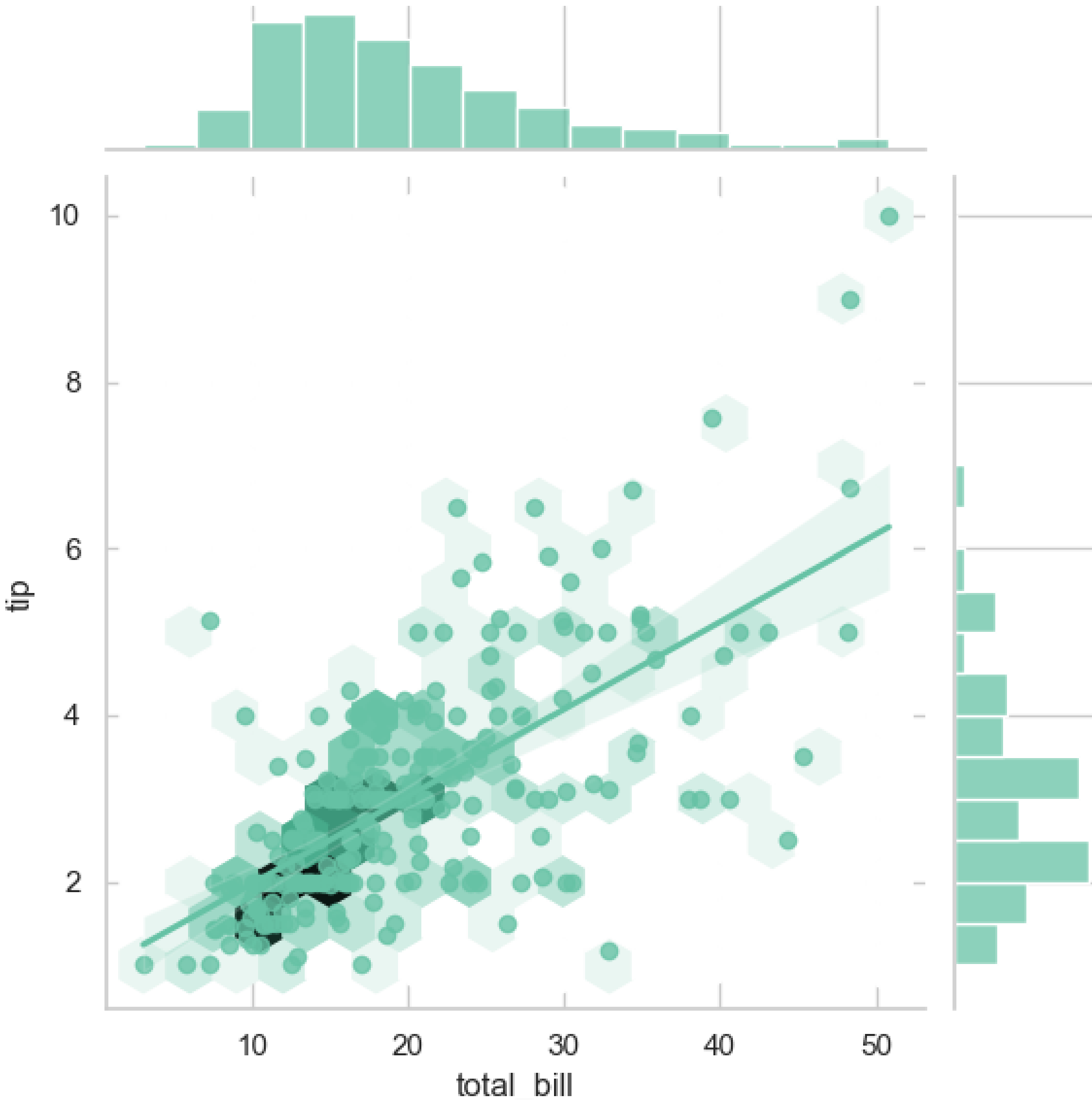
In [ ]:
```
##############seaborn##################
```

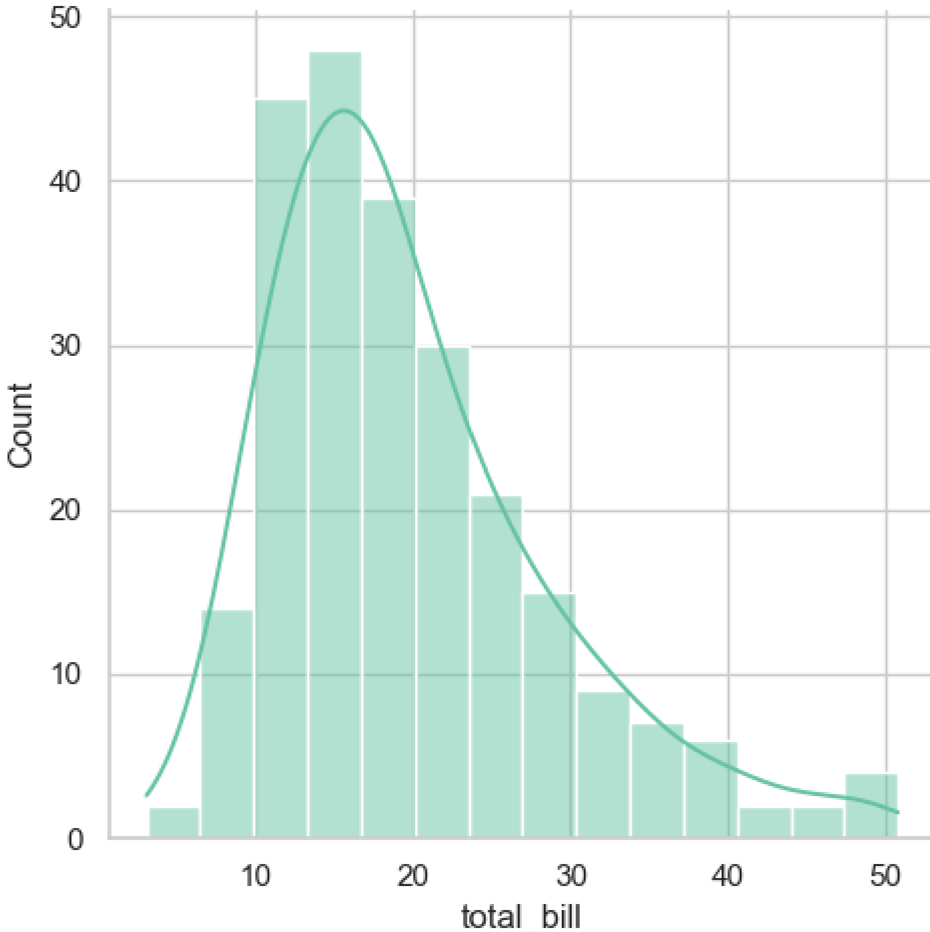
In [61]:
```
import seaborn as sns


# Load example dataset from seaborn
tips = sns.load_dataset("tips")

# Scatter plot with linear regression line
sns.lmplot(x="total_bill", y="tip", data=tips)

# Box plot
sns.boxplot(x="day", y="total_bill", data=tips)

# Histogram with KDE (Kernel Density Estimation)
sns.histplot(data=tips, x="total_bill", kde=True)

# Bar plot
sns.barplot(x="sex", y="total_bill", data=tips)

# Heatmap (correlation matrix)
correlation_matrix = tips.corr()
sns.heatmap(correlation_matrix, annot=True)

# Pairwise scatter plot
sns.pairplot(data=tips, hue="smoker")

# Violin plot
sns.violinplot(x="day", y="total_bill", hue="smoker", split=True, data=tips)
```

```python
# Joint distribution plot
sns.jointplot(data=tips, x="total_bill", y="tip", kind="hex")

# Regression plot
sns.regplot(x="total_bill", y="tip", data=tips)

# Distribution plot
sns.displot(data=tips, x="total_bill", kde=True)

# Cat plot
sns.catplot(x="day", y="total_bill", hue="sex", kind="swarm", data=tips)

# Facet grid plot
g = sns.FacetGrid(tips, col="time", row="sex")
g.map(sns.scatterplot, "total_bill", "tip")

# Customize plot aesthetics
sns.set(style="whitegrid", palette="Set2")

# Show the plots
plt.show()
```

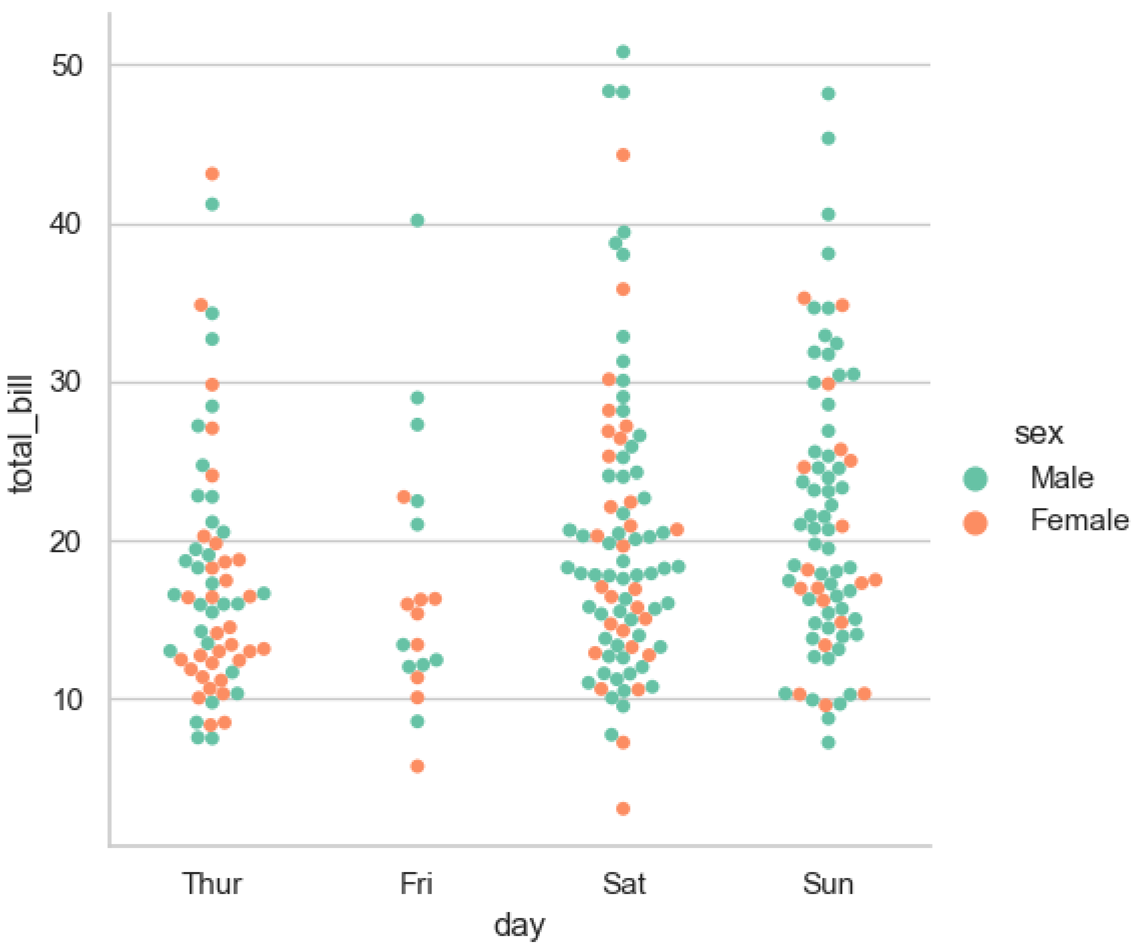



total_bill
tip
size
smoker
Yes
No
Thur
Fri
Sat
Sun
0
20
40
60
5
10

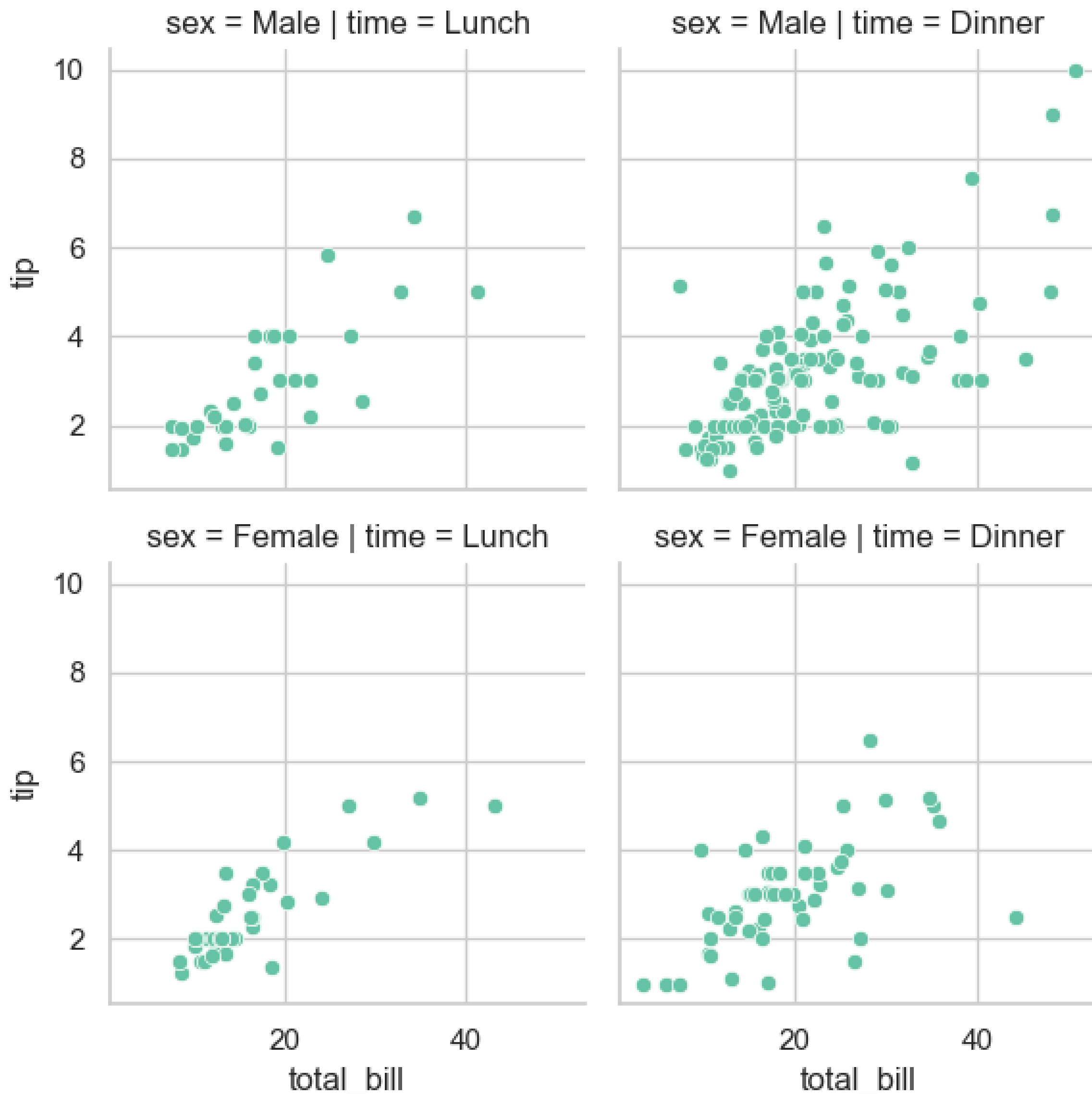

tip
10
8
6
4
2
10
20
30
40
50
total_bill



Count
total_bill
0
10
20
30
40
50



total_bill
day
Thur
Fri
Sat
Sun
10
20
30
40
50
sex
Male
Female


sex = Male | time = Lunch
sex = Male | time = Dinner
sex = Female | time = Lunch
sex = Female | time = Dinner
tip
tip
10
8
6
4
2
20
40
total_bill
total_bill