WEEK 03 ¶

Import CSV File

In [6]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\Anusha V\Desktop\diabetes.csv")
```

In [7]:
```
df.head(5)
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

In [5]:
```
df.tail(5)
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFur
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

In [10]:
```
df.describe()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

Aggregation functions

In [14]:
```
sum = df['Age'].sum()
sum
```

Out[14]: 25529

In [16]:
```
mean = df['Age'].mean()
mean
```

Out[16]: 33.240885416666664

meadian = df[

In [17]:
```
median = df['Age'].median()
median
```

Out[17]: 29.0

In [20]:
```
sum_result = df['Age'].max()
sum_result
```

Out[20]: 81

In [22]:
```
count = df['Age'].count()
count
```

Out[22]: 768

In [23]:
```
std = df['Age'].std()
std
```

Out[23]: 11.76023154067868

In [24]:
```
var = df['Age'].var()
var
```

Out[24]: 138.30304589037365

In [26]:
```
mode = df['Age'].mode()
mode
```

Out[26]:
```
0    22
Name: Age, dtype: int64
```

In [32]:
```
pivot_table = df.pivot_table(index = 'Pregnancies', aggfunc='mean')
pivot_table
```

Out[32]:

Pregnancies	Age	BMI	BloodPressure	DiabetesPedigreeFunction	Glucose	
0	27.603604	34.290090	67.153153	0.520838	123.000000	81
1	27.370370	31.372593	67.792593	0.486496	112.748148	98
2	27.194175	30.583495	63.252427	0.491660	110.796117	85
3	29.026667	30.425333	66.586667	0.432147	123.586667	87
4	32.779412	32.141176	70.029412	0.446353	125.117647	69
5	39.035088	33.192982	76.210526	0.396421	118.859649	57
6	39.340000	30.290000	68.420000	0.429520	120.800000	63
7	41.111111	32.631111	70.777778	0.443622	136.444444	84
8	45.368421	31.568421	75.184211	0.504711	131.736842	92
9	44.178571	31.707143	77.892857	0.550679	131.392857	62
10	42.666667	30.641667	70.208333	0.454167	120.916667	34
11	44.545455	38.563636	74.181818	0.522545	126.454545	65
12	47.444444	32.344444	76.333333	0.444333	113.555556	112
13	44.500000	35.000000	73.800000	0.463300	125.500000	27
14	42.000000	35.100000	70.000000	0.312000	137.500000	92
15	43.000000	37.100000	70.000000	0.153000	136.000000	110
17	47.000000	40.900000	72.000000	0.817000	163.000000	114

In [31]:
```
melt_df = pd.melt(df,id_vars=['Pregnancies'], var_name='Glucose')
melt_df
```

Out[31]:

	Pregnancies	Glucose	value
0	6	Glucose	148.0
1	1	Glucose	85.0
2	8	Glucose	183.0
3	1	Glucose	89.0
4	0	Glucose	137.0
...	...	...	...
6139	10	Outcome	0.0
6140	2	Outcome	0.0
6141	5	Outcome	0.0
6142	1	Outcome	1.0
6143	1	Outcome	0.0

6144 rows × 3 columns

In [38]:
```python
from functools import reduce
def multiply(x, y):
 return x * y
column_to_reduce = 'Age'
result = reduce(multiply, df['Age'])
print(result)

```

```
20284665723245228193614084253999352807096216644787475165960597286437491826
50300860084146912743176432386896437949417552224754694918461030985875746723
98861742653274477350388439246827486367077339417991021756600722205480679213
11671684090579329329110825129371520332552503575827136280712038222090754435
03799774450803521372950519677096295689479819338309114217989778217355422683
40007959721326102629538713795103425181897954976607943161161287640755717132
48746070625494923755473821992520837058256513531257073395395073035515688420
83700442608312853888292127104953038346548772126166864716538691286019578335
52144307059156342353204555425553233311103489466311898891770287197801503795
23779495151246885384963479960894623529017671991428989945284986654754171595
04246494418359162311592122264218027987193226096942549151333400687518924025
10315055938255571344987708175965377091184351788423290840943234386642681799
91729657197336948175755950788401393415928976925328426550984285391528707479
30624000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000
```

In [39]:

```
def map_grades(grade):
 grade_maping = {
 'A':90,
 'B':80,
 'C':70,
 'D':60,
 'E':50,
 'F':40
}
LeagueIndex = 'Grade'
df['Glucose'] = df['BloodPressure'].map(map_grades)
print(df)
```

```
     Pregnancies Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    None             72             35        0  33.6
1              1    None             66             29        0  26.6
2              8    None             64              0        0  23.3
3              1    None             66             23       94  28.1
4              0    None             40             35      168  43.1
..           ...     ...            ...            ...      ...   ...
763           10    None             76             48      180  32.9
764            2    None             70             27        0  36.8
765            5    None             72             23      112  26.2
766            1    None             60              0        0  30.1
767            1    None             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

In [41]:
```
1 def score_condition(score):
2  return score > 5
3 Likes = 'BloodPressure'
4 filtered_df = df[df['Glucose'].apply(score_condition)]
5 print(filtered_df)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_6124\1532437874.py in <module>
      2  return score > 5
      3 Likes = 'BloodPressure'
----> 4 filtered_df = df[df['Glucose'].apply(score_condition)]
      5 print(filtered_df)

~\anaconda3\lib\site-packages\pandas\core\series.py in apply(self, func, c
onvert_dtype, args, **kwargs)
   4431         dtype: float64
   4432         """
-> 4433         return SeriesApply(self, func, convert_dtype, args, kwarg
s).apply()
   4434
   4435     def _reduce(

~\anaconda3\lib\site-packages\pandas\core\apply.py in apply(self)
   1086             return self.apply_str()
   1087
-> 1088         return self.apply_standard()
   1089
   1090     def agg(self):

~\anaconda3\lib\site-packages\pandas\core\apply.py in apply_standard(self)
   1141                 # List[Union[Callable[..., Any], str]]]]"; expect
ed
   1142                 # "Callable[[Any], Any]"
-> 1143                 mapped = lib.map_infer(
   1144                     values,
   1145                     f,  # type: ignore[arg-type]

~\anaconda3\lib\site-packages\pandas\_libs\lib.pyx in pandas._libs.lib.map
_infer()

~\AppData\Local\Temp\ipykernel_6124\1532437874.py in score_condition(scor
e)
      1 def score_condition(score):
----> 2  return score > 5
      3 Likes = 'BloodPressure'
      4 filtered_df = df[df['Glucose'].apply(score_condition)]
      5 print(filtered_df)

TypeError: '>' not supported between instances of 'NoneType' and 'int'
```

In [43]:
```
modify_score = lambda x: x * 2
LeagueIndex = 'BloodPressure'
df['Glucose'] = df['Age'].apply(modify_score)
print(df)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      100             72             35        0  33.6
1              1       62             66             29        0  26.6
2              8       64             64              0        0  23.3
3              1       42             66             23       94  28.1
4              0       66             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      126             76             48      180  32.9
764            2       54             70             27        0  36.8
765            5       60             72             23      112  26.2
766            1       94             60              0        0  30.1
767            1       46             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

In [45]:
```
sorted_df = df.sort_values(by='Age', ascending=True)
print(sorted_df)

```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
255            1       42             64             35        0  33.6   
60             2       42              0              0        0   0.0   
102            0       42             96              0        0  22.5   
182            1       42             74             20       23  27.7   
623            0       42             70             27      115  43.5   
..           ...      ...            ...            ...      ...   ...   
123            5      138             80              0        0  26.8   
684            5      138             82              0        0   0.0   
666            4      140             82             18        0  32.5   
453            2      144              0              0        0  19.6   
459            9      162             74             33       60  25.9   

     DiabetesPedigreeFunction  Age  Outcome  
255                     0.543   21        1  
60                      0.304   21        0  
102                     0.262   21        0  
182                     0.299   21        0  
623                     0.347   21        0  
..                        ...  ...      ...  
123                     0.186   69        0  
684                     0.640   69        0  
666                     0.235   70        1  
453                     0.832   72        0  
459                     0.460   81        0  

[768 rows x 9 columns]
```

In [46]:
```
grouped_data = df.groupby('Age')
result = grouped_data['Age'].mean()
print(result)
```

```
Age
21    21.0
22    22.0
23    23.0
24    24.0
25    25.0
26    26.0
27    27.0
28    28.0
29    29.0
30    30.0
31    31.0
32    32.0
33    33.0
34    34.0
35    35.0
36    36.0
37    37.0
38    38.0
39    39.0
40    40.0
41    41.0
42    42.0
43    43.0
44    44.0
45    45.0
46    46.0
47    47.0
48    48.0
49    49.0
50    50.0
51    51.0
52    52.0
53    53.0
54    54.0
55    55.0
56    56.0
57    57.0
58    58.0
59    59.0
60    60.0
61    61.0
62    62.0
63    63.0
64    64.0
65    65.0
66    66.0
67    67.0
68    68.0
69    69.0
70    70.0
72    72.0
81    81.0
Name: Age, dtype: float64
```

In [47]:
```
1 df.head(2)
```

Out[47]:

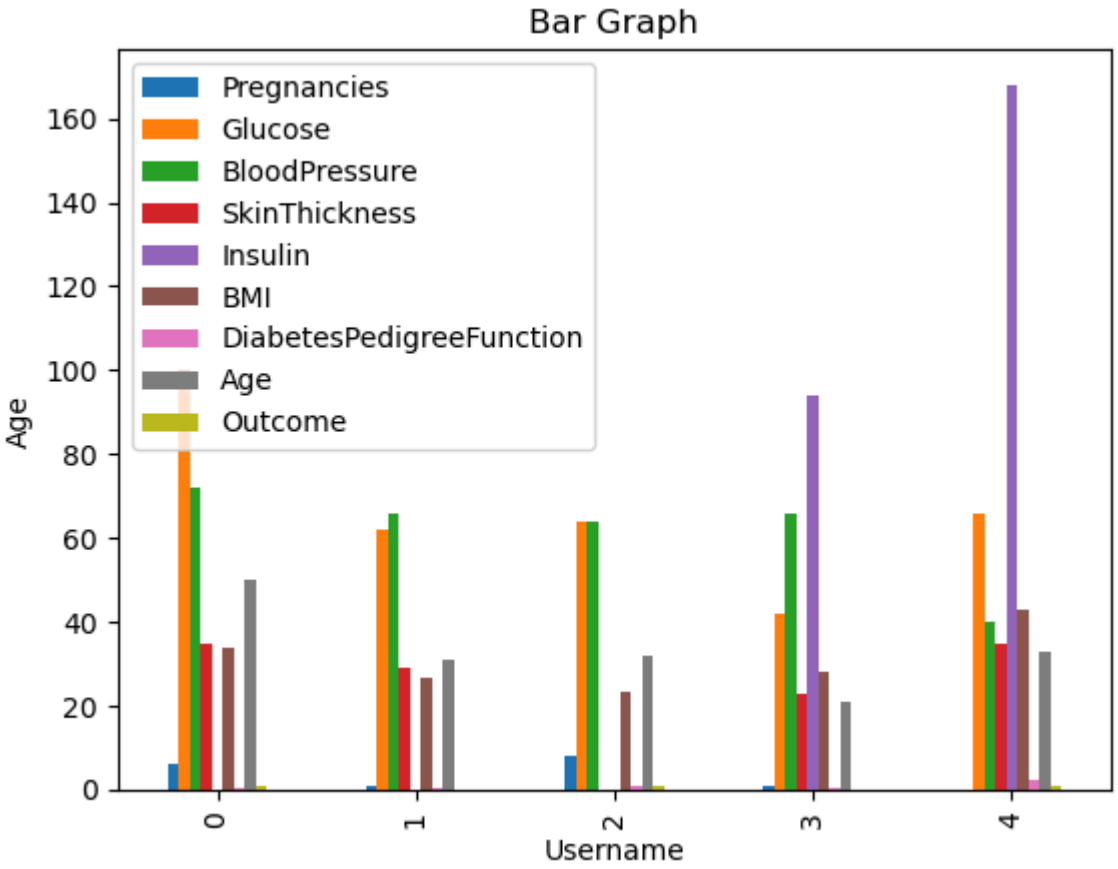
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	100	72	35	0	33.6	0.
1	1	62	66	29	0	26.6	0.

VISUALIZATION

Bar Chat

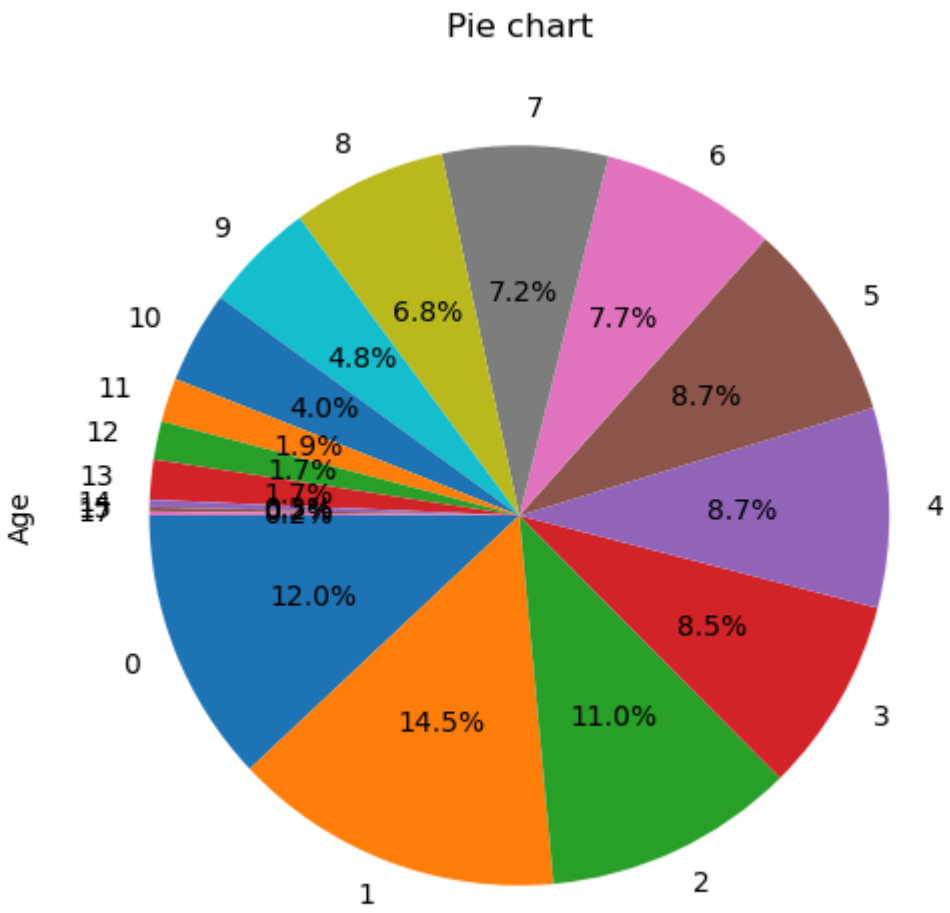
In [51]:
```
1 import matplotlib.pyplot as plt
2 bar_data = df.head(5)
3 plt.figure(figsize=(8, 6))
4 bar_data.plot(kind='bar')
5 plt.xlabel('Username')
6 plt.ylabel('Age')
7 plt.title('Bar Graph')
8 plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



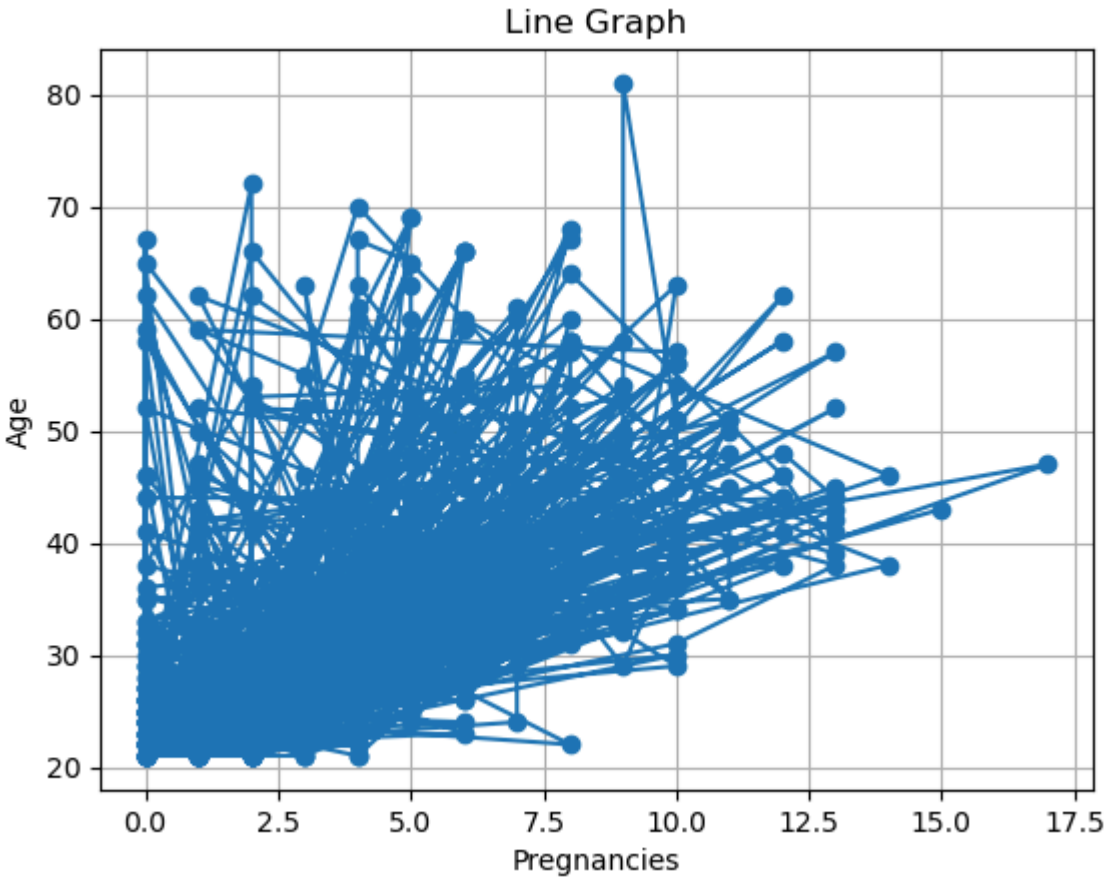
Pie Chart

In [53]:
```
pie_data = df.groupby('Pregnancies')['Age'].sum()
plt.figure(figsize=(8, 6))
pie_data.plot(kind='pie', autopct='%1.1f%%', startangle=180)
plt.title('Pie chart')
plt.show()

```



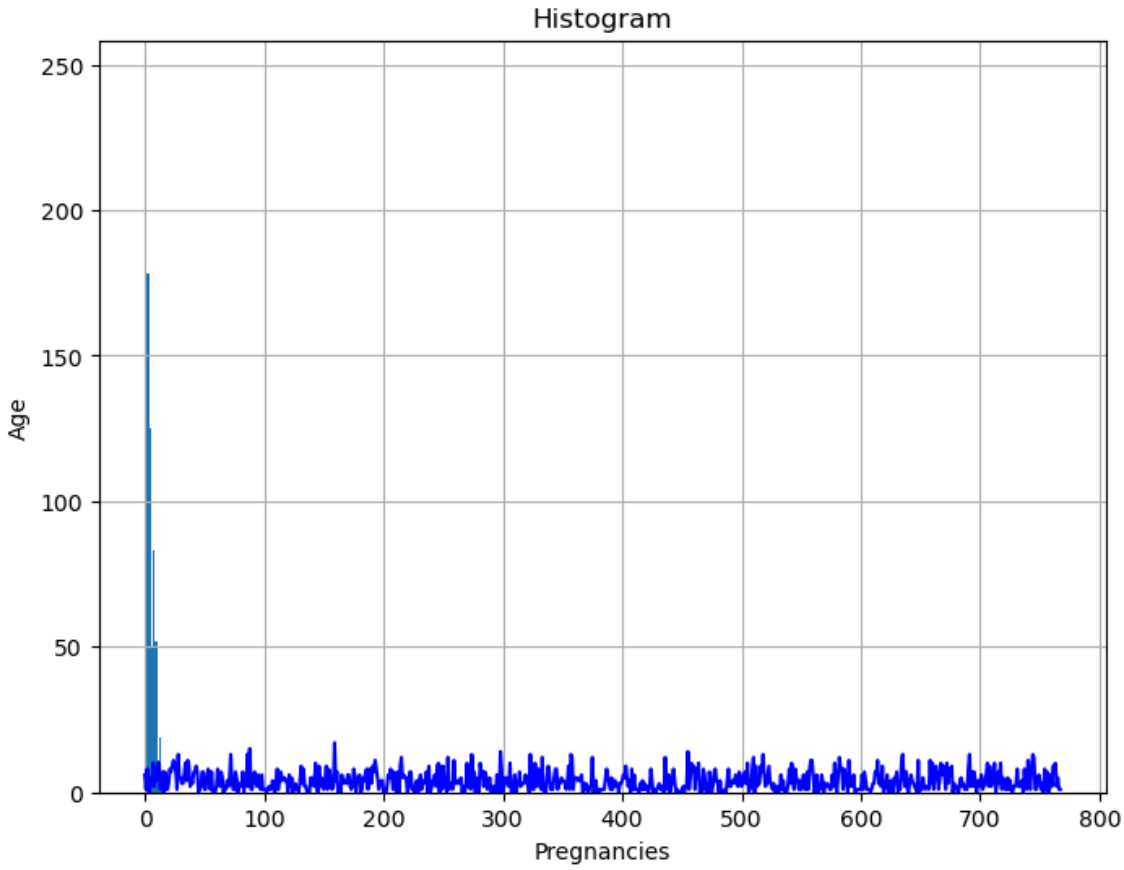
LINE GRAPH

In [56]:
```
line_chart = df.head(2)
x_line = df['Pregnancies']
y_line = df['Age']
plt.plot(x_line, y_line, marker='o', linestyle='-')
plt.xlabel('Pregnancies')
plt.ylabel('Age')
plt.title('Line Graph')
plt.grid(True)
plt.show()
```



HISTOGRAM

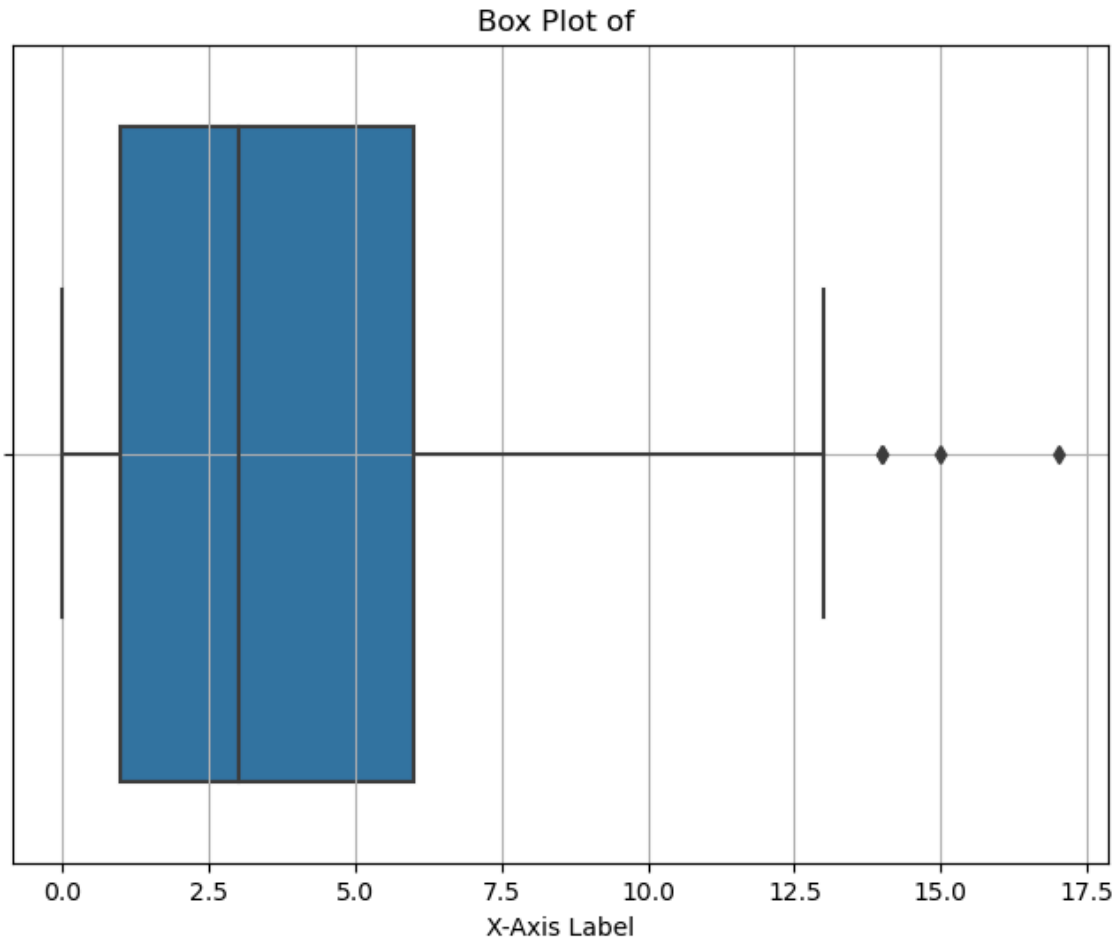
In [58]:
```
data_hist =df['Pregnancies']
plt.figure(figsize=(8, 6))
plt.hist(data_hist, bins=10)
plt.xlabel('Pregnancies')
plt.ylabel('Age')
plt.plot(df.index, df['Pregnancies'], label='Age', color='b')
# Display the plot
plt.grid(True) # Optional: Add grid lin')
plt.title('Histogram')
plt.show()
```



BOX PLOT

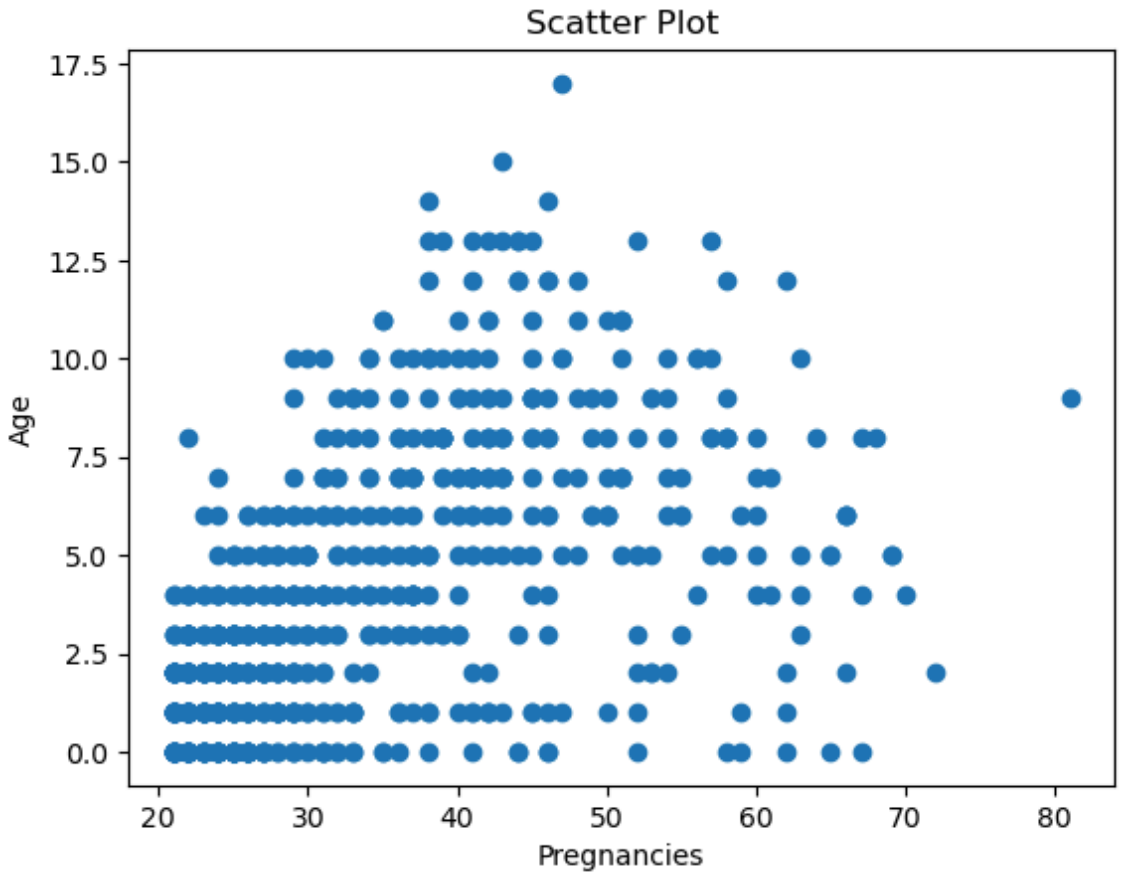
In [59]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Create a box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='Pregnancies', data=df)
plt.xlabel('X-Axis Label')
plt.title('Box Plot of ')
plt.grid(True)
plt.show()
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```



SCATTER PLOT

In [62]:
```
import pandas as pd
import matplotlib.pyplot as plt
x_column = 'Pregnancies'
y_column = 'Age'
# Create a scatter plot
plt.scatter(df['Age'], df['Pregnancies'])
# Add labels and title
plt.xlabel('Pregnancies')
plt.ylabel('Age')
plt.title('Scatter Plot')
# Show the plot
plt.show()

```



BUBBLE CHART

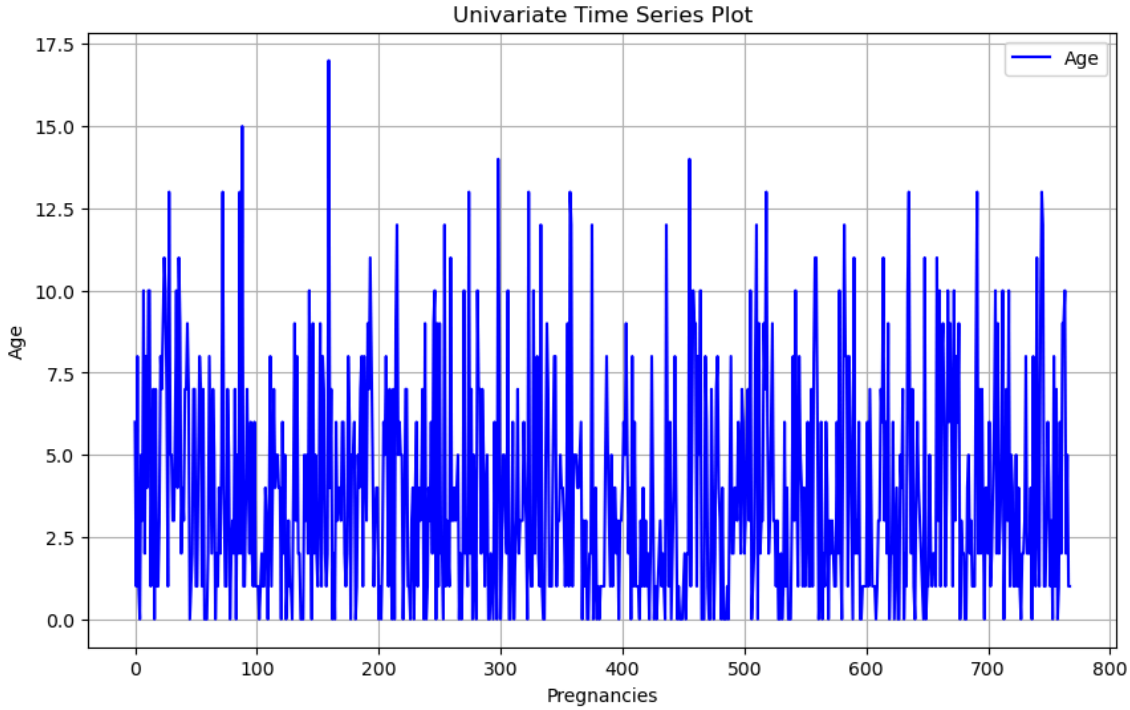
In [67]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Define the columns for x, y, and bubble size
x_column = 'Pregnancies' # Replace with the actual column name
y_column = 'Age' # Replace with the actual column name
size_column = 'Size_Column' # Replace with the actual column name
# Create a bubble chart
plt.figure(figsize=(10, 6))
sns.scatterplot(x=x_column, y=y_column, data=df, sizes=(20, 200), alpha
# Customize the plot properties (title, labels, etc.)
```

```
  File "C:\Users\Anusha V\AppData\Local\Temp\ipykernel_6124\1800711684.p
y", line 13

    ^
SyntaxError: positional argument follows keyword argument
```

Univariate Time Series Plot

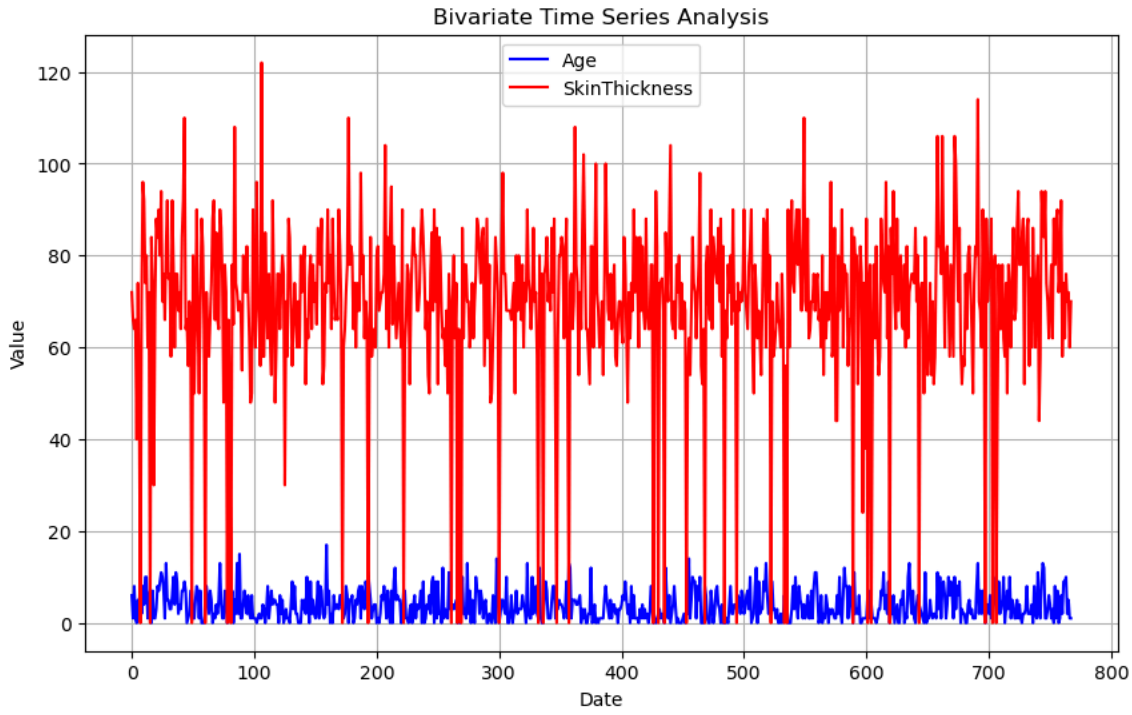
In [68]:
```
plt.figure(figsize=(10, 6)) # Optional: Set the figure size
# Plot the time series data
plt.plot(df.index, df['Pregnancies'], label='Age', color='b')
# Optional: Add labels and a title
plt.xlabel('Pregnancies')
plt.ylabel('Age')
plt.title('Univariate Time Series Plot')
# Optional: Customize other plot properties
# Display the plot
plt.grid(True) # Optional: Add grid lines
plt.legend() # Optional: Show the legend if you have multiple series
plt.show()
1

```



Out[68]: 1

Bivariate Time Series Analysis

In [69]:
```
import pandas as pd
import matplotlib.pyplot as plt
# Plot the time series data
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Pregnancies'], label='Age', color='blue')
plt.plot(df.index, df['BloodPressure'], label='SkinThickness', color='r
# Customize the plot
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Bivariate Time Series Analysis')
plt.legend()
plt.grid(True)
# Show the plot
plt.show()
```



In [71]:
```
df.head(1)
```

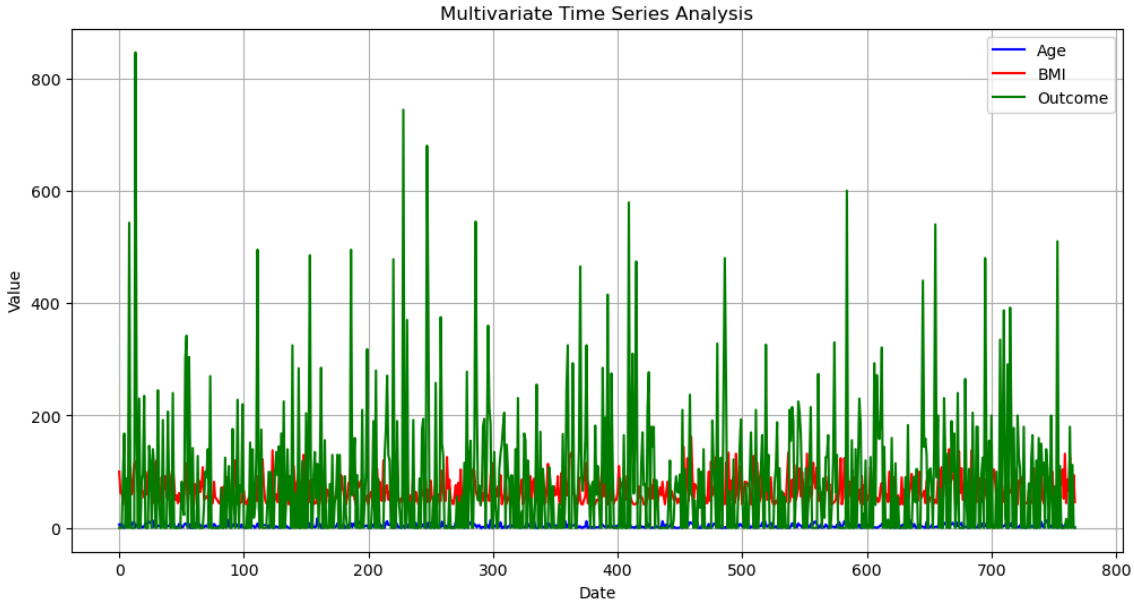
Out[71]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	100	72	35	0	33.6	0.

Multivariate Time Series Analysis

In [73]:

```
import matplotlib.pyplot as plt
# Plot the multivariate time series data
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Pregnancies'], label='Age', color='blue')
plt.plot(df.index, df['Glucose'], label='BMI', color='red')
plt.plot(df.index, df['Insulin'], label='Outcome', color='green')
# Customize the plot
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Multivariate Time Series Analysis')
plt.legend()
plt.grid(True)
# Show the plot
plt.show()
```



In [1]:
```
import pandas as pd
import numpy as np

# Create a Series from a list
my_list = [1, 2, 3, 4, 5]
series_from_list = pd.Series(my_list)
print(series_from_list)

# Create a Series from a Numpy array
my_array = np.array([10, 20, 30, 40, 50])
series_from_array = pd.Series(my_array)
print(series_from_array)

```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
0    10
1    20
2    30
3    40
4    50
dtype: int32
```

In [2]:
```
# Create a Series from a dictionary
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
series_from_dict = pd.Series(my_dict)
print(series_from_dict)


```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

In [3]:
```
# You can also provide custom indices
custom_index_series = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(custom_index_series)

```

```
a    10
b    20
c    30
dtype: int64
```

In [5]:
```
import pandas as pd
import numpy as np

# Create a DataFrame from a dictionary of lists
data_dict = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'San Francisco', 'Los Angeles']
}

df_from_dict = pd.DataFrame(data_dict)
print(df_from_dict)
```

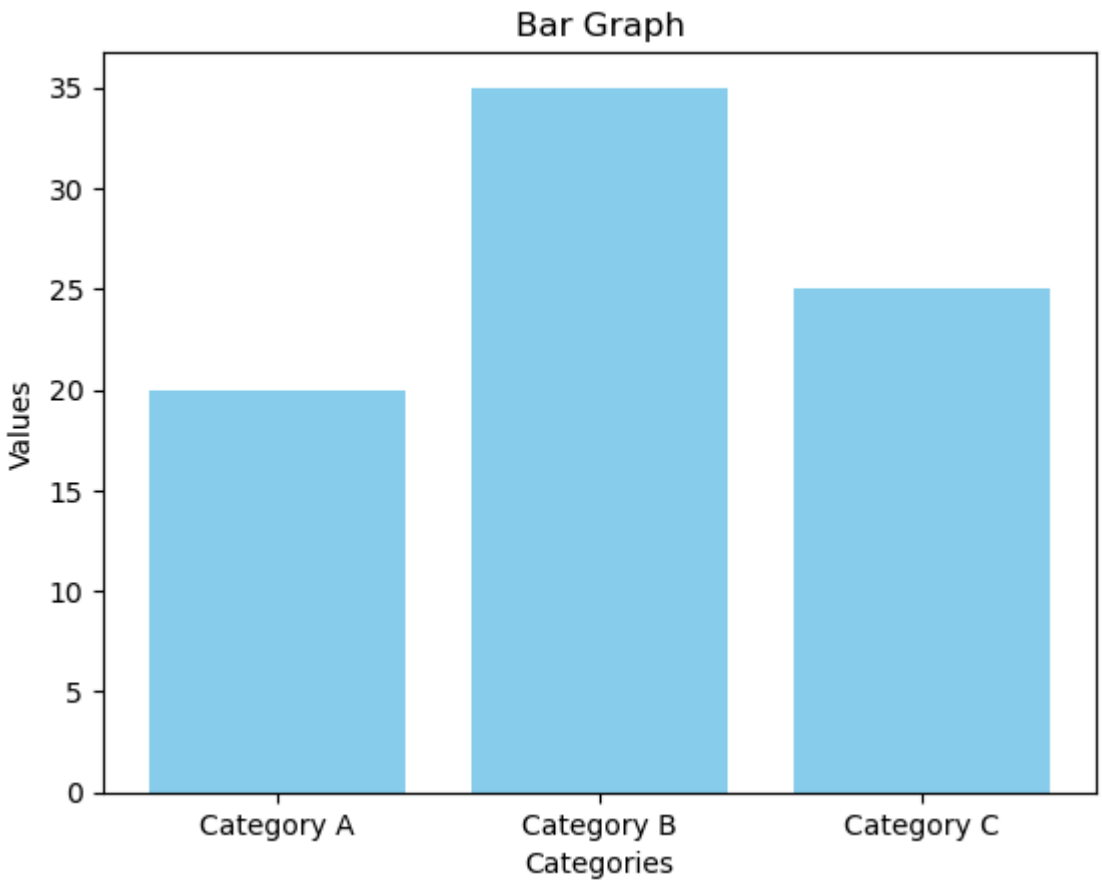
```
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles
```

In [6]:
```
# Create a DataFrame from a dictionary of NumPy arrays
data_dict_np = {
    'A': np.array([1,bnn b  2, 3]),
    'B': np.array([4, 5, 6]),
    'C': np.array([7, 8, 9])
}

df_from_dict_np = pd.DataFrame(data_dict_np)
print(df_from_dict_np)
```

```
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9
```

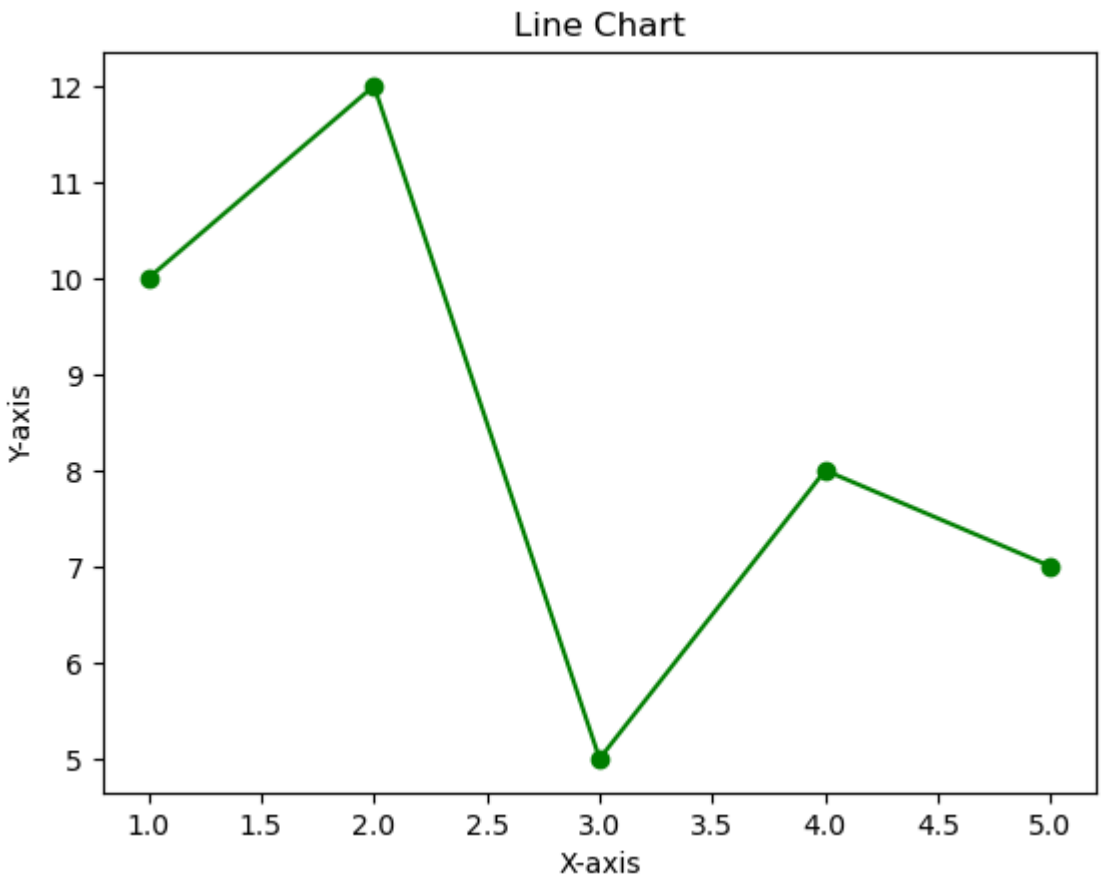
In [7]:

```
import matplotlib.pyplot as plt

categories = ['Category A', 'Category B', 'Category C']
values = [20, 35, 25]

plt.bar(categories, values, color='skyblue')
plt.title('Bar Graph')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()

```



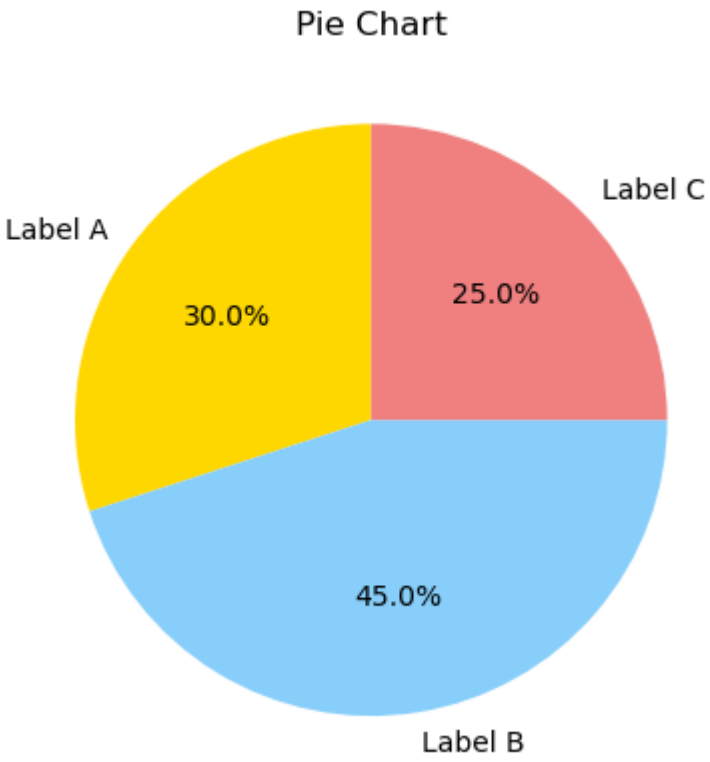
In [8]:

```
import matplotlib.pyplot as plt

x_values = [1, 2, 3, 4, 5]
y_values = [10, 12, 5, 8, 7]

plt.plot(x_values, y_values, marker='o', linestyle='-', color='green')
plt.title('Line Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

```



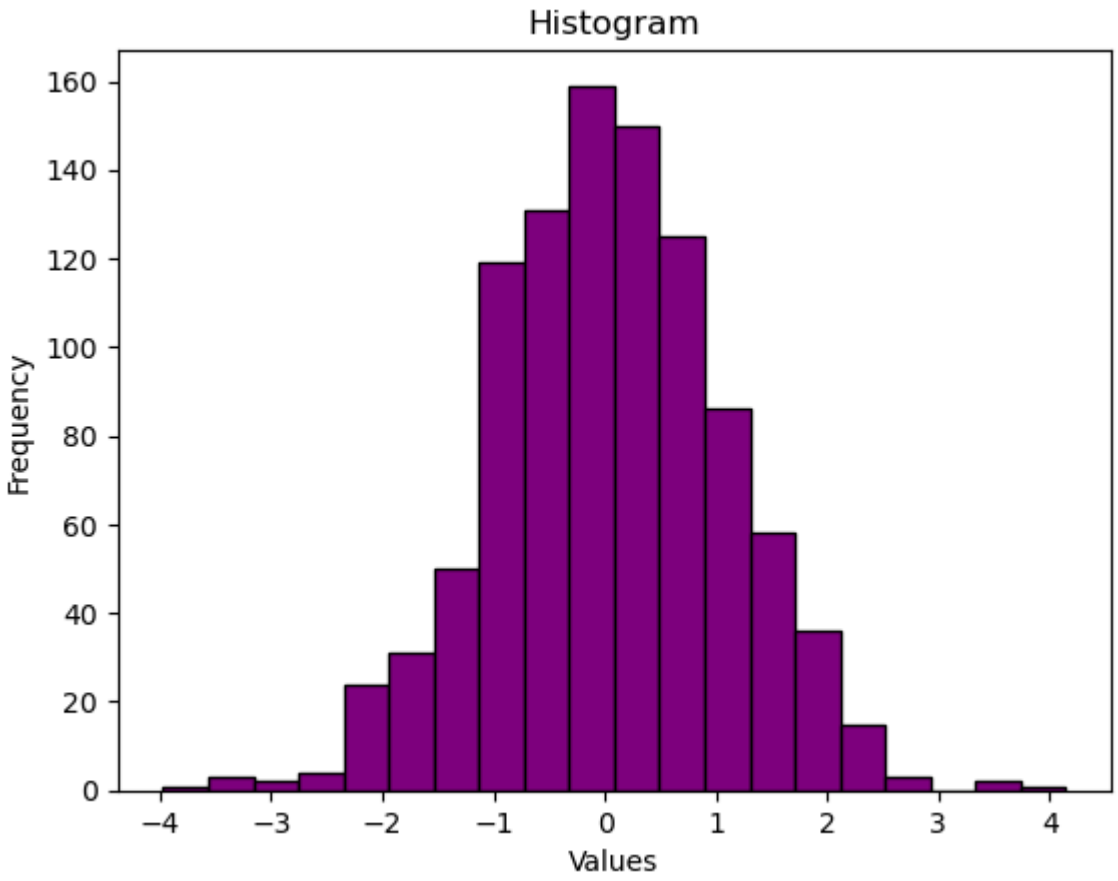
In [9]:

```
import matplotlib.pyplot as plt

labels = ['Label A', 'Label B', 'Label C']
sizes = [30, 45, 25]

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, colors=
plt.title('Pie Chart')
plt.show()

```



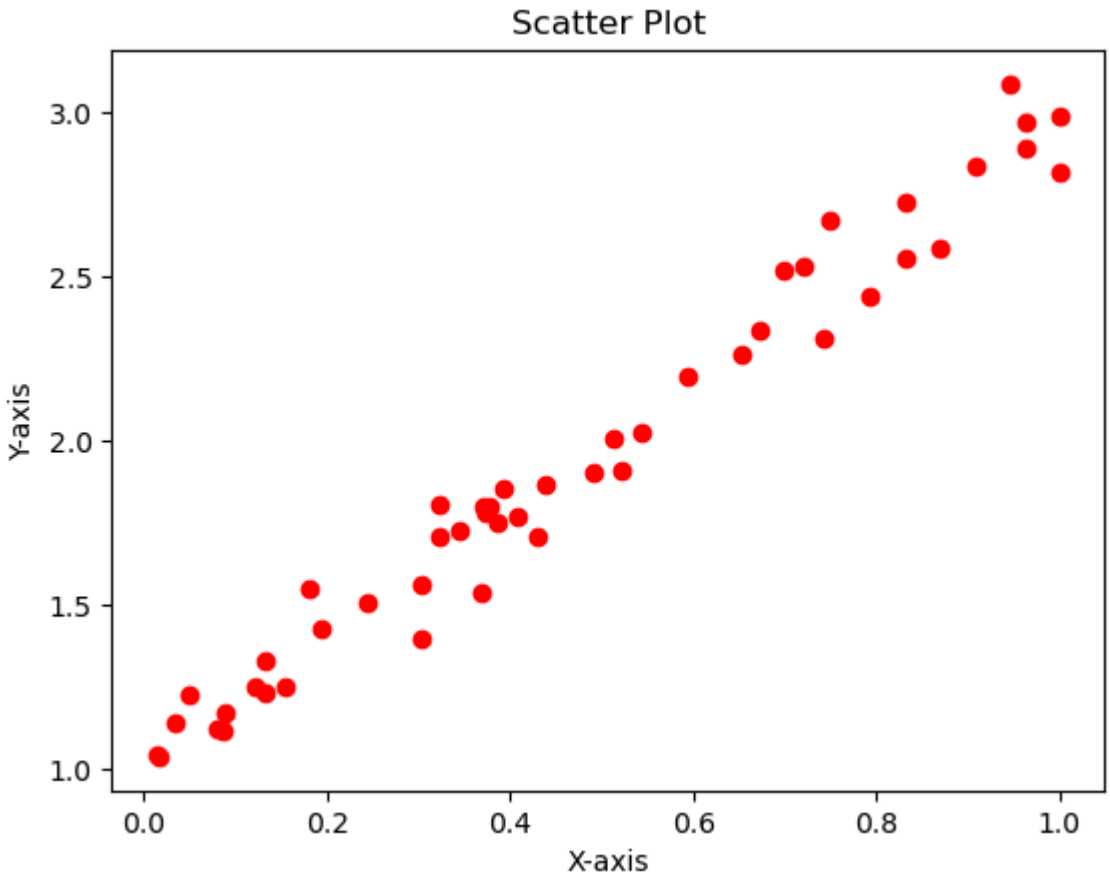
In [10]:

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

data = np.random.randn(1000)

plt.hist(data, bins=20, color='purple', edgecolor='black')
plt.title('Histogram')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()

```



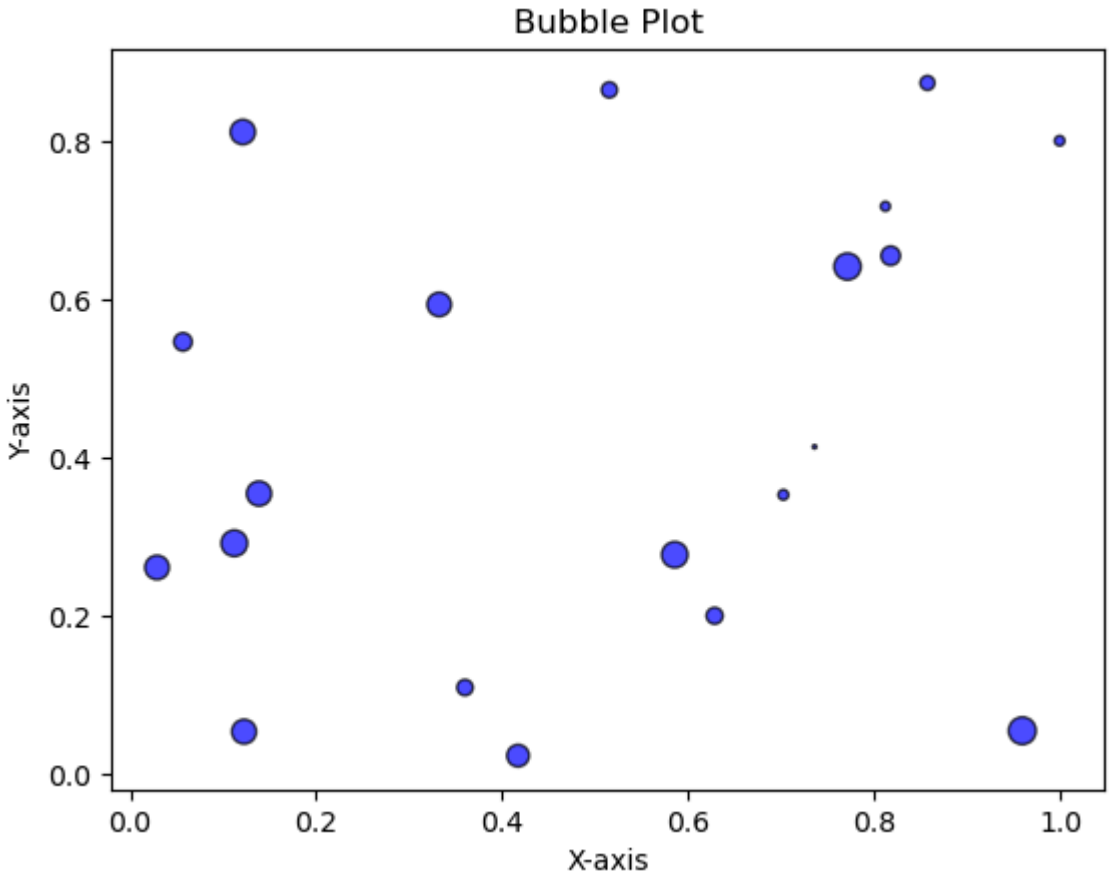
In [11]:

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

x_values = np.random.rand(50)
y_values = 2 * x_values + 1 + 0.1 * np.random.randn(50)

plt.scatter(x_values, y_values, color='red')
plt.title('Scatter Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

```



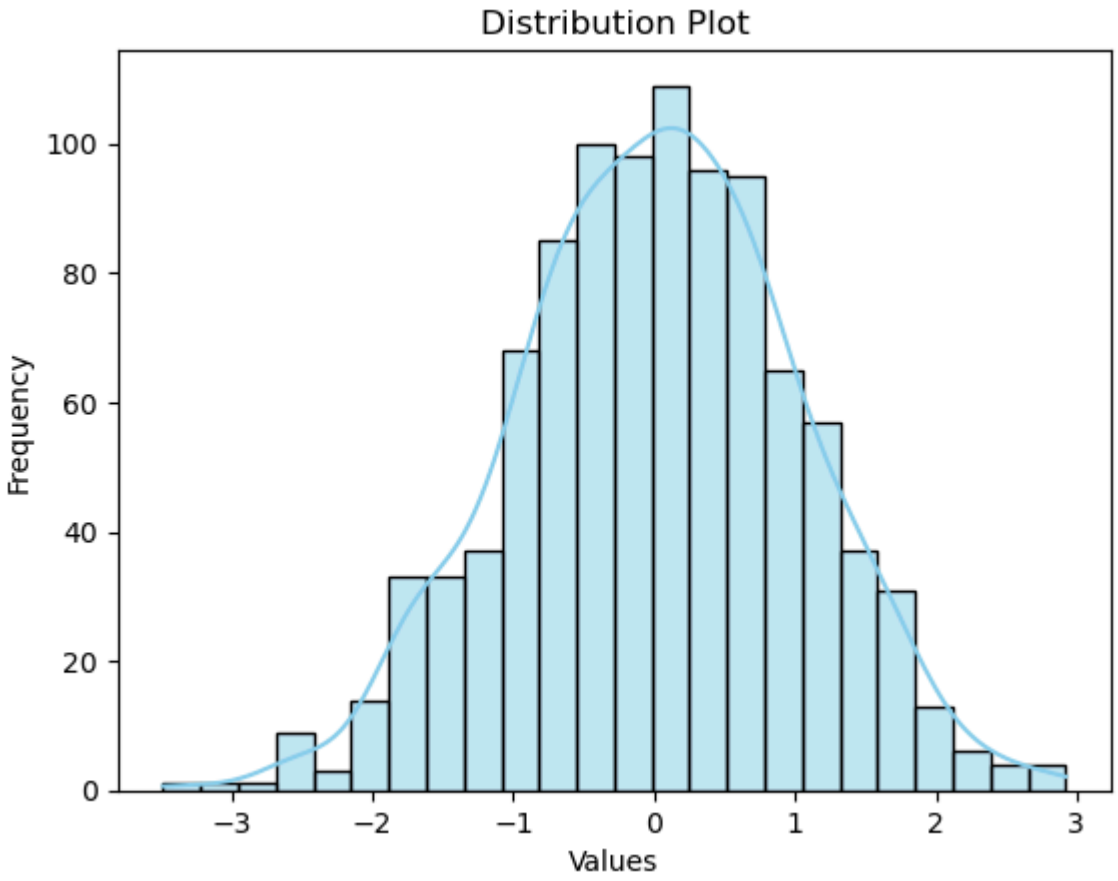
In [12]:

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

# Generate random data
x_values = np.random.rand(20)
y_values = np.random.rand(20)
sizes = np.random.rand(20) * 100  # Random sizes for bubbles

# Scatter plot with bubbles
plt.scatter(x_values, y_values, s=sizes, alpha=0.7, c='blue', edgecolor

plt.title('Bubble Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

```



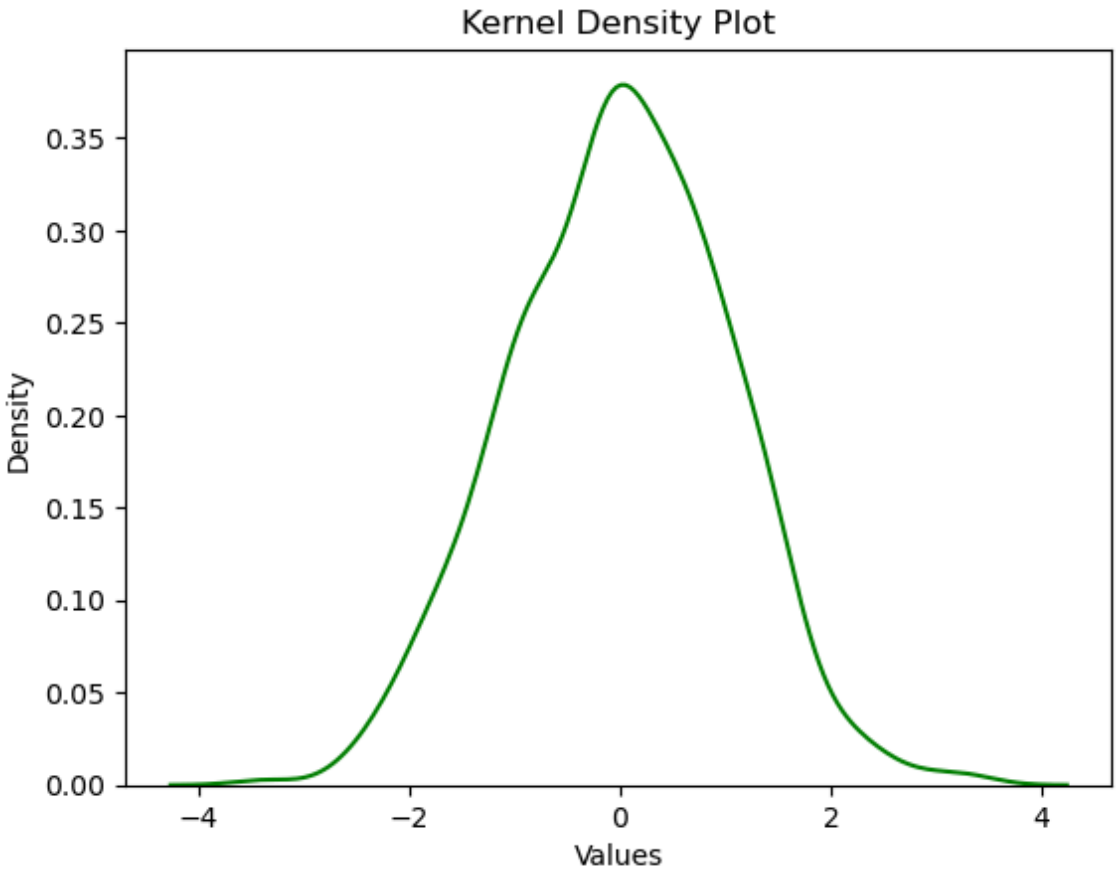
In [13]:
```
import seaborn as sns
import numpy as np

data = np.random.randn(1000)

sns.histplot(data, kde=True, color='skyblue')
plt.title('Distribution Plot')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()

```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

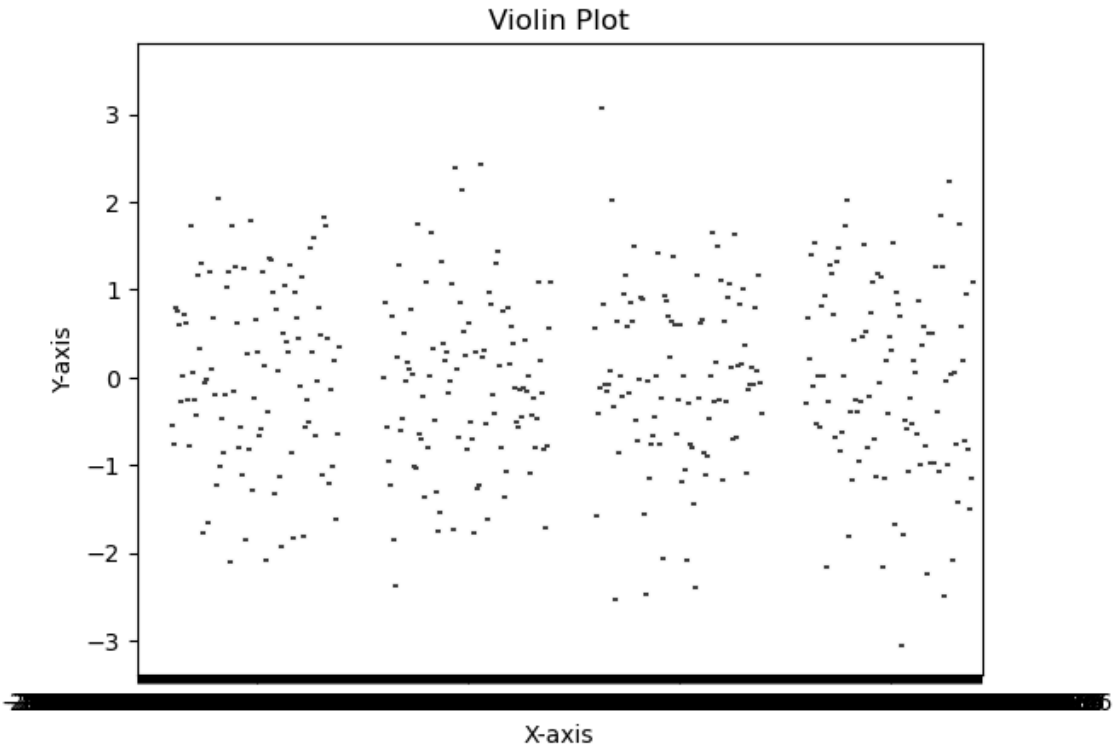


In [14]:

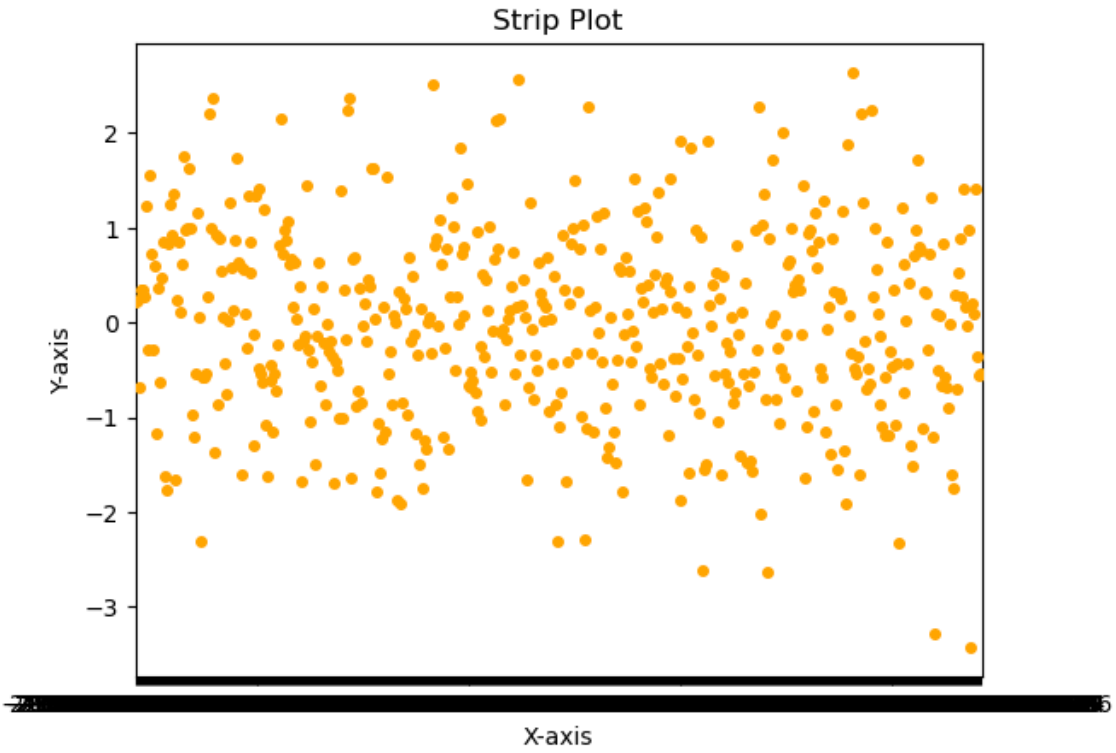
```
import seaborn as sns
import numpy as np

data = np.random.randn(1000)

sns.kdeplot(data, color='green')
plt.title('Kernel Density Plot')
plt.xlabel('Values')
plt.ylabel('Density')
plt.show()

```



In [15]:
```
import seaborn as sns
import numpy as np

data = np.random.randn(500, 2)

sns.violinplot(x=data[:, 0], y=data[:, 1], palette='muted')
plt.title('Violin Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

```



In [16]:
```
import seaborn as sns
import numpy as np

data = np.random.randn(500, 2)

sns.stripplot(x=data[:, 0], y=data[:, 1], jitter=True, color='orange')
plt.title('Strip Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

```



DATA SPLITTING

In [5]:
```
from sklearn.model_selection import train_test_split
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [15]:
```
import numpy as np
X = np.random.rand(100, 5)
y = np.random.randint(0, 2, size=100)
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, tes

```

In [12]:
```
print(f"Training set size: {len(X_train)}")
print(f"Validation set size: {len(X_val)}")
print(f"Test set size: {len(X_test)}")
```

```
Training set size: 491
Validation set size: 123
Test set size: 154
```

In [13]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\Anusha V\Desktop\diabetes.csv")
df.head(5)
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

In [18]:
```
X = df.drop('Outcome', axis=1)  # Features
y = df['Outcome']  # Labels
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, tes

print(f"Training set size: {len(X_train)}")
print(f"Validation set size: {len(X_val)}")
print(f"Test set size: {len(X_test)}")
```

```
Training set size: 491
Validation set size: 123
Test set size: 154
```

OUTLIERS

In [21]:
```
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
```

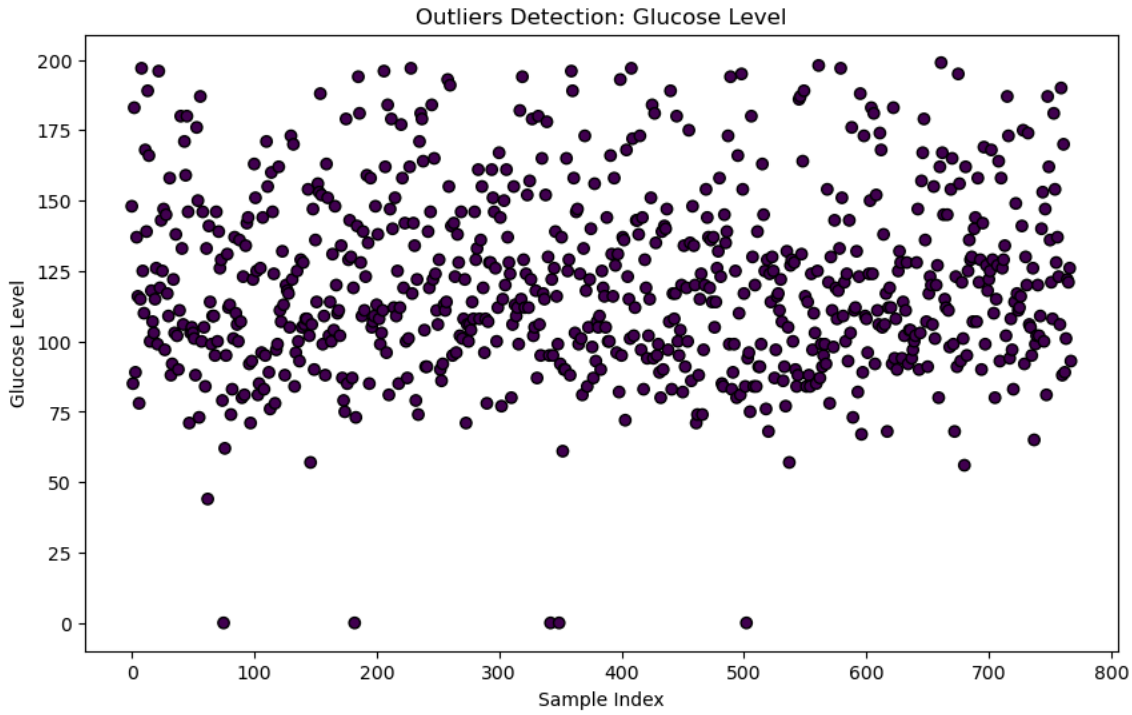
In [20]:

```
z_scores = np.abs(stats.zscore(df))

# Define a threshold for identifying outliers (commonly 3)
threshold = 3

# Find and print the indices of outliers
outlier_indices = np.where(z_scores > threshold)
outliers = set(zip(outlier_indices[0], outlier_indices[1]))

print("Indices of outliers:")
print(outliers)
```

```
Indices of outliers:
{(459, 7), (535, 2), (145, 5), (484, 2), (673, 5), (584, 4), (81, 5), (39
5, 6), (298, 0), (706, 5), (336, 2), (697, 2), (75, 1), (589, 2), (159,
0), (695, 4), (579, 3), (604, 2), (182, 1), (619, 2), (8, 4), (60, 5), (66
6, 7), (300, 2), (49, 2), (153, 4), (455, 0), (9, 5), (222, 2), (266, 2),
(332, 2), (15, 2), (81, 2), (453, 2), (370, 4), (753, 4), (706, 2), (468,
2), (621, 6), (370, 6), (371, 5), (426, 5), (494, 2), (60, 2), (655, 4),
(415, 4), (430, 2), (269, 2), (7, 2), (522, 5), (342, 1), (111, 4), (4,
6), (502, 1), (645, 4), (45, 6), (593, 6), (684, 5), (347, 2), (435, 2),
(286, 4), (13, 4), (445, 6), (426, 2), (703, 2), (409, 4), (684, 7), (349,
1), (88, 0), (228, 4), (330, 6), (522, 2), (533, 2), (643, 2), (601, 2),
(228, 6), (193, 2), (247, 4), (371, 6), (58, 6), (261, 2), (220, 4), (453,
7), (123, 7), (486, 4), (494, 5), (78, 2), (177, 5), (186, 4), (357, 2),
(49, 5), (445, 5), (172, 2)}
```

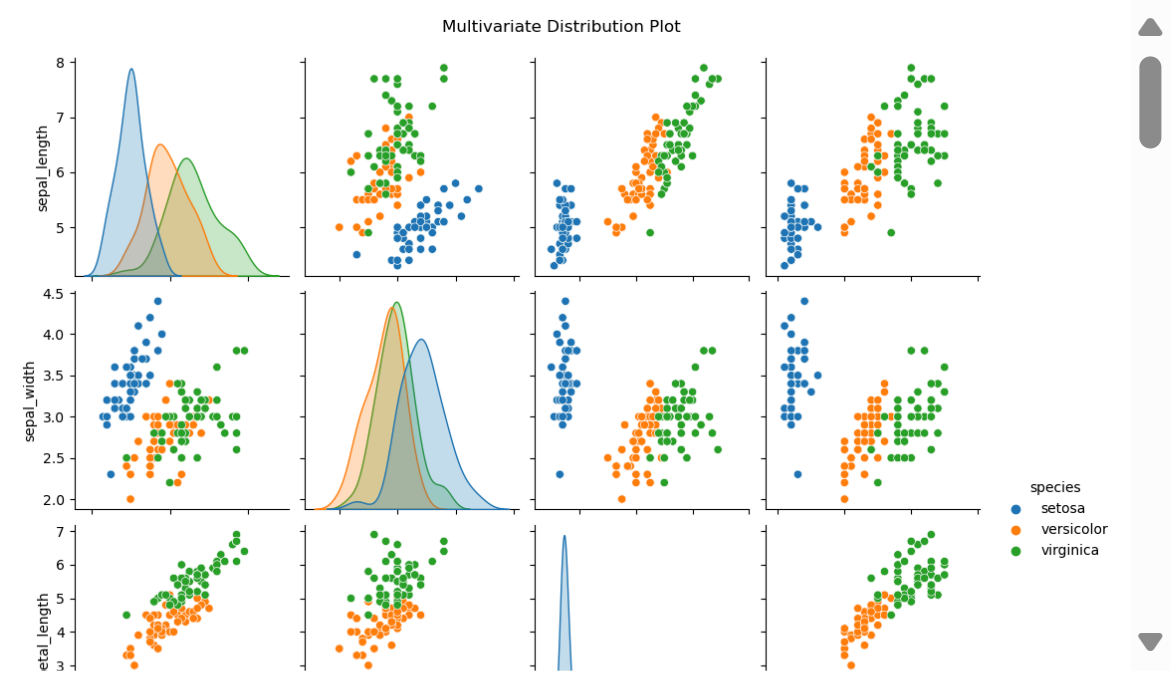
In [22]:

```
df = df.dropna()

# Function to detect outliers using Z-score
def detect_outliers_zscore(data_frame):
    z_scores = np.abs(stats.zscore(data_frame))
    outliers = (z_scores > 3).all(axis=1)
    return outliers

# Identify outliers
outliers = detect_outliers_zscore(df[['Pregnancies', 'Glucose', 'BloodP

# Plot a graph of the outliers
plt.figure(figsize=(10, 6))
plt.scatter(df.index, df['Glucose'], c=outliers, cmap='viridis', marker
plt.title('Outliers Detection: Glucose Level')
plt.xlabel('Sample Index')
plt.ylabel('Glucose Level')
plt.show()
```



In [29]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample dataset
df = sns.load_dataset('iris')  # You can replace this with your own dat

# Multivariate Distribution Plot
sns.pairplot(df, hue='species', diag_kind='kde')
plt.suptitle('Multivariate Distribution Plot', y=1.02)
plt.show()

# Multivariate Comparison Plot
sns.violinplot(x='species', y='sepal_length', data=df)
plt.title('Multivariate Comparison Plot')
plt.show()

# Multivariate Relationship Plot
sns.scatterplot(x='sepal_length', y='petal_length', hue='species', data
plt.title('Multivariate Relationship Plot')
plt.show()

# Multivariate Composition Plot
sns.jointplot(x='sepal_length', y='petal_length', kind='hex', data=df)
plt.suptitle('Multivariate Composition Plot', y=1.02)
plt.show()


```



In [30]:

```
# Initialize the model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the evaluation metrics
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')
print('Confusion Matrix:')
print(conf_matrix)

```

```
Accuracy: 0.75
Precision: 0.64
Recall: 0.67
F1 Score: 0.65
Confusion Matrix:
[[78 21]
 [18 37]]

C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\linear_model\_logist
ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [32]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2

# Assuming 'Outcome' is the target variable
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Assuming Linear Regression as the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate MAE (Mean Absolute Error)
mae = mean_absolute_error(y_test, y_pred)
print(f"MAE: {mae}")

# Calculate RMSE (Root Mean Squared Error)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"RMSE: {rmse}")

# Calculate R2 score
r2 = r2_score(y_test, y_pred)
print(f"R2 Score: {r2}")
0
```

```
Accuracy: 0.7467532467532467
Precision: 0.64
Recall: 0.67
F1 Score: 0.65
Confusion Matrix:
[[78 21]
 [18 37]]

C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\linear_model\_logist
ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [38]:
```
from sklearn.metrics import confusion_matrix
```

In [39]:
```
import numpy as np
```

In [42]:
```
actual = np.array([1,0,1,1,0,1,0,1,1])
predicted = np.array([1,0,1,0,1,1,0,1,0])
cm = confusion_matrix(actual, predicted)
print("confusion matrix")
print(cm)
```

```
confusion matrix
[[2 1]
 [2 4]]
```

In [43]:
```
from sklearn.metrics import classification_report

# Example true labels and predicted labels
y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]
y_pred = [1, 0, 1, 1, 0, 0, 1, 0, 1, 1]

# Generate a classification report
report = classification_report(y_true, y_pred)

# Print the classification report
print(report)

```

```
              precision    recall  f1-score   support

           0       0.75      0.60      0.67         5
           1       0.67      0.80      0.73         5

    accuracy                           0.70        10
   macro avg       0.71      0.70      0.70        10
weighted avg       0.71      0.70      0.70        10
```

In [48]:

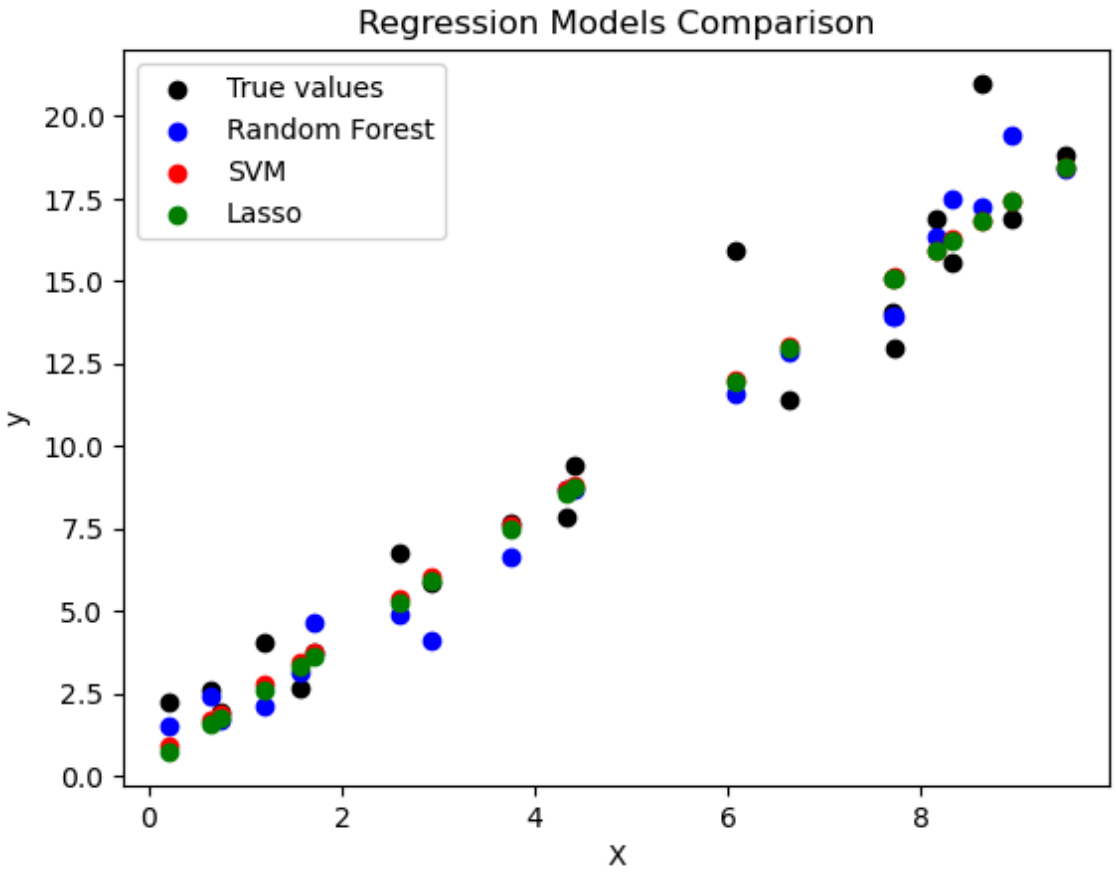
```
 import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Generate sample data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X.squeeze() + np.random.randn(100) * 2  # True relationship: y

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# SVM Regression
svm_model = SVR(kernel='linear')
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# Lasso Regression
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
lasso_pred = lasso_model.predict(X_test)

# Evaluate models
def evaluate_model(model_name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name} - Mean Squared Error: {mse:.2f}, R2 Score: {r2

# Print evaluation metrics for each model
evaluate_model("Random Forest", y_test, rf_pred)
evaluate_model("SVM", y_test, svm_pred)
evaluate_model("Lasso", y_test, lasso_pred)


```

```
Random Forest - Mean Squared Error: 3.02, R2 Score: 0.92
SVM - Mean Squared Error: 2.52, R2 Score: 0.93
Lasso - Mean Squared Error: 2.60, R2 Score: 0.93
```

In [49]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Generate sample data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X.squeeze() + np.random.randn(100) * 2  # True relationship: y

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# SVM Regression
svm_model = SVR(kernel='linear')
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# Lasso Regression
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
lasso_pred = lasso_model.predict(X_test)

# Evaluate models
def evaluate_model(model_name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name} - Mean Squared Error: {mse:.2f}, R2 Score: {r2

# Print evaluation metrics for each model
evaluate_model("Random Forest", y_test, rf_pred)
evaluate_model("SVM", y_test, svm_pred)
evaluate_model("Lasso", y_test, lasso_pred)

# Plot the true vs predicted values for each model
plt.scatter(X_test, y_test, color='black', label='True values')
plt.scatter(X_test, rf_pred, color='blue', label='Random Forest')
plt.scatter(X_test, svm_pred, color='red', label='SVM')
plt.scatter(X_test, lasso_pred, color='green', label='Lasso')
plt.legend()
plt.xlabel('X')
plt.ylabel('y')
plt.title('Regression Models Comparison')
plt.show()

```

```
Random Forest - Mean Squared Error: 3.02, R2 Score: 0.92
SVM - Mean Squared Error: 2.52, R2 Score: 0.93
Lasso - Mean Squared Error: 2.60, R2 Score: 0.93
```


Regression Models Comparison
True values
Random Forest
SVM
Lasso
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
y
0
2
4
6
8
X

In [50]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score

# Generate sample data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X.squeeze() + np.random.randn(100) * 2  # True relationship: y

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Define and train models
models = {
    "Random Forest": RandomForestRegressor(n_estimators=100, random_sta
    "SVM": SVR(kernel='linear'),
    "Lasso": Lasso(alpha=0.1)
}

# Evaluate models
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"{model_name} - Mean Squared Error: {mse:.2f}, R2 Score: {r2

```

```
Random Forest - Mean Squared Error: 3.02, R2 Score: 0.92
SVM - Mean Squared Error: 2.52, R2 Score: 0.93
Lasso - Mean Squared Error: 2.60, R2 Score: 0.93
```

In [51]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Generate sample data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X.squeeze() + np.random.randn(100) * 2  # True relationship: y

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# SVM Regression
svm_model = SVR(kernel='linear')
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# Lasso Regression
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
lasso_pred = lasso_model.predict(X_test)

# Evaluate models
def evaluate_model(model_name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name} - Mean Squared Error: {mse:.2f}, R2 Score: {r2

# Print evaluation metrics for each model
evaluate_model("Random Forest", y_test, rf_pred)
evaluate_model("SVM", y_test, svm_pred)
evaluate_model("Lasso", y_test, lasso_pred)


```

```
Random Forest - Mean Squared Error: 3.02, R2 Score: 0.92
SVM - Mean Squared Error: 2.52, R2 Score: 0.93
Lasso - Mean Squared Error: 2.60, R2 Score: 0.93
```

In [52]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_scc

# Load the Iris dataset (binary classification for simplicity)
iris = load_iris()
X, y = iris.data, iris.target
# Convert the problem into binary classification by considering only th
X, y = X[y != 2], y[y != 2]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create and train a Support Vector Machine (SVM) classifier
classifier = SVC(kernel='linear', C=1.0)
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print("Confusion Matrix:")
print(conf_matrix)

```

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
Confusion Matrix:
[[12  0]
 [ 0  8]]
```

In [53]:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import SMOTE

# Generate sample data for binary classification
X_binary, y_binary = make_classification(n_samples=1000, n_features=20,

# Generate sample data for multi-class classification
X_multiclass, y_multiclass = make_classification(n_samples=1000, n_feat

# Generate sample data for multi-label classification
X_multilabel, y_multilabel = make_classification(n_samples=1000, n_feat

# Make a binary classification problem imbalanced
X_imbalanced, y_imbalanced = make_classification(n_samples=1000, n_feat

# Apply SMOTE for over-sampling in imbalanced classification
smote = SMOTE(random_state=42)
X_imbalanced_resampled, y_imbalanced_resampled = smote.fit_resample(X_i

# Define and train classifiers
classifiers = {
    "Binary Classification": RandomForestClassifier(random_state=42),
    "Multi-Class Classification": OneVsRestClassifier(RandomForestClass
    "Multi-Label Classification": RandomForestClassifier(random_state=4
    "Imbalanced Classification": RandomForestClassifier(random_state=42
}

# Train and evaluate classifiers
for problem_type, clf in classifiers.items():
    if "Binary" in problem_type:
        X_train, X_test, y_train, y_test = train_test_split(X_binary, y
    elif "Multi-Class" in problem_type:
        X_train, X_test, y_train, y_test = train_test_split(X_multiclas
    elif "Multi-Label" in problem_type:
        X_train, X_test, y_train, y_test = train_test_split(X_multilabe
    elif "Imbalanced" in problem_type:
        X_train, X_test, y_train, y_test = train_test_split(X_imbalance

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    print(f"\n{problem_type} Evaluation:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

```

```
Binary Classification Evaluation:
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.89      0.92       104
           1       0.89      0.95      0.92        96

    accuracy                           0.92       200
   macro avg       0.92      0.92      0.92       200
weighted avg       0.92      0.92      0.92       200


Multi-Class Classification Evaluation:
Accuracy: 0.84
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.85      0.80        61
           1       0.89      0.83      0.86        81
           2       0.88      0.84      0.86        58

    accuracy                           0.84       200
   macro avg       0.84      0.84      0.84       200
weighted avg       0.85      0.84      0.84       200


Multi-Label Classification Evaluation:
Accuracy: 0.65
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.56      0.61        43
           1       0.62      0.70      0.66        33
           2       0.62      0.55      0.58        38
           3       0.68      0.68      0.68        40
           4       0.66      0.76      0.71        46

    accuracy                           0.65       200
   macro avg       0.65      0.65      0.65       200
weighted avg       0.65      0.65      0.65       200


Imbalanced Classification Evaluation:
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       180
           1       0.98      0.98      0.98       179

    accuracy                           0.98       359
   macro avg       0.98      0.98      0.98       359
weighted avg       0.98      0.98      0.98       359
```

In [1]:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Binary Classification
X_bin, y_bin = make_classification(n_samples=1000, n_features=20, n_inf
X_bin_train, X_bin_test, y_bin_train, y_bin_test = train_test_split(X_b

binary_classifier = LogisticRegression()
binary_classifier.fit(X_bin_train, y_bin_train)
y_bin_pred = binary_classifier.predict(X_bin_test)

print("Binary Classification Accuracy:", accuracy_score(y_bin_test, y_b
print("Binary Classification Report:\n", classification_report(y_bin_te

# Multi-class Classification
X_multi, y_multi = make_classification(n_samples=1000, n_features=20, n
X_multi_train, X_multi_test, y_multi_train, y_multi_test = train_test_s

multi_classifier = OneVsRestClassifier(LogisticRegression())
multi_classifier.fit(X_multi_train, y_multi_train)
y_multi_pred = multi_classifier.predict(X_multi_test)

print("\nMulti-class Classification Accuracy:", accuracy_score(y_multi_
print("Multi-class Classification Report:\n", classification_report(y_m

# Multi-label Classification
X_multilabel, y_multilabel = make_classification(n_samples=1000, n_feat
X_multilabel_train, X_multilabel_test, y_multilabel_train, y_multilabel

multilabel_classifier = RandomForestClassifier()
multilabel_classifier.fit(X_multilabel_train, y_multilabel_train)
y_multilabel_pred = multilabel_classifier.predict(X_multilabel_test)

print("\nMulti-label Classification Accuracy:", accuracy_score(y_multil
print("Multi-label Classification Report:\n", classification_report(y_m

# Imbalanced Classification
from imblearn.datasets import make_imbalance
from collections import Counter
from sklearn.svm import SVC

X_imbalanced, y_imbalanced = make_classification(n_samples=1000, n_feat
X_imbalanced_train, X_imbalanced_test, y_imbalanced_train, y_imbalanced

print("\nOriginal Class Distribution:", Counter(y_imbalanced_train))

imbalanced_classifier = SVC()
imbalanced_classifier.fit(X_imbalanced_train, y_imbalanced_train)
y_imbalanced_pred = imbalanced_classifier.predict(X_imbalanced_test)

print("Imbalanced Classification Accuracy:", accuracy_score(y_imbalance
print("Imbalanced Classification Report:\n", classification_report(y_im
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

Binary Classification Accuracy: 0.82
Binary Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.78      0.82       104
           1       0.78      0.86      0.82        96

    accuracy                           0.82       200
   macro avg       0.82      0.82      0.82       200
weighted avg       0.82      0.82      0.82       200


Multi-class Classification Accuracy: 0.755
Multi-class Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.67      0.64        61
           1       0.87      0.80      0.83        81
           2       0.78      0.78      0.78        58

    accuracy                           0.76       200
   macro avg       0.75      0.75      0.75       200
weighted avg       0.76      0.76      0.76       200


---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_6100\3238744082.py in <module>
     29
     30 # Multi-label Classification
---> 31 X_multilabel, y_multilabel = make_classification(n_samples=1000, n
_features=20, n_informative=10, n_classes=5, random_state=42, n_labels=3)
     32 X_multilabel_train, X_multilabel_test, y_multilabel_train, y_multi
label_test = train_test_split(X_multilabel, y_multilabel, test_size=0.2, r
andom_state=42)
     33

TypeError: make_classification() got an unexpected keyword argument 'n_lab
els'
```

In [6]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import make_classification

# Generate a hypothetical binary classification dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative
                           n_redundant=10, n_classes=2, random_state=42

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Binary classification model (Logistic Regression)
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)

print(f'Accuracy: {accuracy}')
print('Classification Report:\n', report)

```

```
Accuracy: 0.875
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.89      0.89       111
           1       0.86      0.85      0.86        89

    accuracy                           0.88       200
   macro avg       0.87      0.87      0.87       200
weighted avg       0.87      0.88      0.87       200
```

In [3]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Sample data for multi-class classification (Iris dataset)
data = load_iris()
X, y = data.data, data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Multi-class classification model (Logistic Regression)
model = LogisticRegression(multi_class='auto', max_iter=1000)
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)

print(f'Accuracy: {accuracy}')
print('Classification Report:\n', report)

```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

In [7]:

```
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import make_multilabel_classification

# Generate a hypothetical multi-label classification dataset
X, y = make_multilabel_classification(n_samples=1000, n_features=20, n_

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Multi-label classification model (Random Forest)
model = MultiOutputClassifier(RandomForestClassifier(n_estimators=100))
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)

print(f'Accuracy: {accuracy}')
print('Classification Report:\n', report)


```

```
Accuracy: 0.69
Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.64      0.74        85
           1       0.93      0.89      0.91       126
           2       0.88      0.88      0.88       111

   micro avg       0.90      0.82      0.86       322
   macro avg       0.90      0.80      0.84       322
weighted avg       0.90      0.82      0.85       322
 samples avg       0.79      0.72      0.74       322


C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in samples with no predicted labels. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defined an
d being set to 0.0 in samples with no true labels. Use `zero_division` par
ameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [5]:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Generate imbalanced binary classification data
X, y = make_classification(n_samples=1000, n_features=20, n_informative
                           n_redundant=10, weights=[0.9, 0.1], random_s

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Imbalanced classification model (Random Forest)
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)

print(f'Accuracy: {accuracy}')
print('Classification Report:\n', report)

```

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       184
           1       1.00      0.75      0.86        16

    accuracy                           0.98       200
   macro avg       0.99      0.88      0.92       200
weighted avg       0.98      0.98      0.98       200
```

vectroized operations

In [2]:

```python
import numpy as np

# Creating two arrays
array1 = np.array([1, 2, 3, 4])
array2 = np.array([5, 6, 7, 8])

# Vectorized addition
result_addition = array1 + array2

# Vectorized multiplication
result_multiplication = array1 * array2

# Vectorized square root
result_sqrt = np.sqrt(array1)

# Print results
print("Vectorized Addition:", result_addition)
print("Vectorized Multiplication:", result_multiplication)
print("Vectorized Square Root:", result_sqrt)

```

```
Vectorized Addition: [ 6  8 10 12]
Vectorized Multiplication: [ 5 12 21 32]
Vectorized Square Root: [1.         1.41421356 1.73205081 2.        ]
```

In [3]:

```
import pandas as pd
from functools import reduce

# Creating a sample DataFrame
data = {'numbers': [1, 2, 3, 4, 5]}
df = pd.DataFrame(data)

# Using map with lambda to square each element in the 'numbers' column
df['squared'] = df['numbers'].map(lambda x: x ** 2)

# Using filter with lambda to keep only even numbers in the 'numbers' c
df_filtered = df[df['numbers'].apply(lambda x: x % 2 == 0)]

# Using reduce with lambda to find the product of all elements in the '
product_of_numbers = reduce(lambda x, y: x * y, df['numbers'])

# Print results
print("Original DataFrame:")
print(df)

print("\nDataFrame with Squared Numbers:")
print(df)

print("\nDataFrame with Only Even Numbers:")
print(df_filtered)

print("\nProduct of Numbers using Reduce:", product_of_numbers)

```

```
Original DataFrame:
   numbers  squared
0        1        1
1        2        4
2        3        9
3        4       16
4        5       25

DataFrame with Squared Numbers:
   numbers  squared
0        1        1
1        2        4
2        3        9
3        4       16
4        5       25

DataFrame with Only Even Numbers:
   numbers  squared
1        2        4
3        4       16

Product of Numbers using Reduce: 120
```

In [5]:
```
pip install autograd

```

```
Collecting autograd
  Downloading autograd-1.6.2-py3-none-any.whl (49 kB)
     ---------------------------------------- 49.3/49.3 kB 357.9 kB/s eta 0:
00:00
Requirement already satisfied: future>=0.15.2 in c:\users\anusha v\anacond
a3\lib\site-packages (from autograd) (0.18.2)
Requirement already satisfied: numpy>=1.12 in c:\users\anusha v\anaconda3
\lib\site-packages (from autograd) (1.26.2)
Installing collected packages: autograd
Successfully installed autograd-1.6.2
Note: you may need to restart the kernel to use updated packages.
```

In [6]:

```python
import numpy as np
import autograd.numpy as ag_np
from autograd import grad

# Scalars
scalar = 5

# Vectors
vector = np.array([1, 2, 3])

# Matrices
matrix = np.array([[1, 2, 3], [4, 5, 6]])

# Tensors
tensor = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

# Gradient Computation using autograd
def scalar_function(x):
    return 3 * x**2 + 2 * x + 1

# Compute the gradient of the scalar function
grad_scalar_function = grad(scalar_function)

# Test gradient computation
point = 2.0
gradient_at_point = grad_scalar_function(point)

# Print results
print("Scalar:", scalar)
print("Vector:", vector)
print("Matrix:")
print(matrix)
print("Tensor:")
print(tensor)

print("\nGradient of the scalar function f(x) = 3x^2 + 2x + 1 at x =",

```

```
Scalar: 5
Vector: [1 2 3]
Matrix:
[[1 2 3]
 [4 5 6]]
Tensor:
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]

Gradient of the scalar function f(x) = 3x^2 + 2x + 1 at x = 2.0 is 14.0
```

In [7]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

# Sample data
data = {'Feature1': [1, 2, 3, 4],
        'Feature2': ['A', 'B', 'A', 'B'],
        'Target': [0, 1, 0, 1]}

df = pd.DataFrame(data)

# Display original data
print("Original DataFrame:")
print(df)

# Handling missing values (if any)
df.fillna(value=0, inplace=True)

# Encoding categorical variables
label_encoder = LabelEncoder()
df['Feature2'] = label_encoder.fit_transform(df['Feature2'])

# Splitting the data into features (X) and target variable (y)
X = df.drop('Target', axis=1)
y = df['Target']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Standardizing features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Display preprocessed data
print("\nPreprocessed DataFrame:")
print("X_train_scaled:")
print(pd.DataFrame(X_train_scaled, columns=X.columns))
print("y_train:")
print(y_train)
print("X_test_scaled:")
print(pd.DataFrame(X_test_scaled, columns=X.columns))
print("y_test:")
print(y_test)

```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
Original DataFrame:
   Feature1 Feature2  Target
0         1        A       0
1         2        B       1
2         3        A       0
3         4        B       1

Preprocessed DataFrame:
X_train_scaled:
   Feature1  Feature2
0  1.069045  1.414214
1 -1.336306 -0.707107
2  0.267261 -0.707107
y_train:
3    1
0    0
2    0
Name: Target, dtype: int64
X_test_scaled:
   Feature1  Feature2
0 -0.534522  1.414214
y_test:
1    1
Name: Target, dtype: int64
```

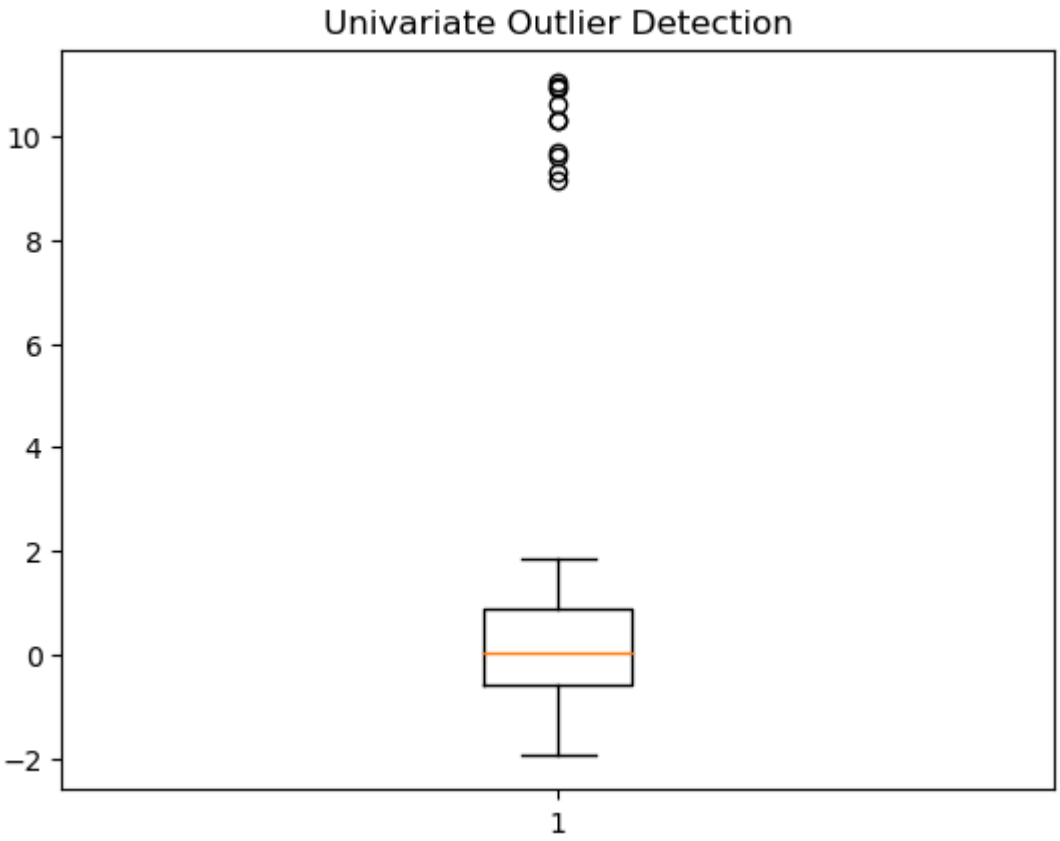
In [10]:

```
import pandas as pd

# Sample data with missing values
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Emily'],
        'Age': [25, 30, None, 22, 35],
        'Salary': [50000, None, 60000, 45000, 70000]}

df = pd.DataFrame(data)

# Display information about the DataFrame, including non-null counts
print("Original DataFrame Info:")
print(df.info())

# Check for missing values
print("\nMissing Values:")
print(df.isnull())

# Handling missing values
# Option 1: Drop rows with missing values
df_dropped = df.dropna()

# Option 2: Fill missing values with a specific value (e.g., mean)
df_filled = df.fillna(df.mean())

# Display processed DataFrames
print("\nDataFrame after Dropping Missing Values:")
print(df_dropped)

print("\nDataFrame after Filling Missing Values:")
print(df_filled)

```

```
Original DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Name    5 non-null      object 
 1   Age     4 non-null      float64
 2   Salary  4 non-null      float64
dtypes: float64(2), object(1)
memory usage: 248.0+ bytes
None

Missing Values:
    Name    Age  Salary
0  False  False   False
1  False  False    True
2  False   True   False
3  False  False   False
4  False  False   False

DataFrame after Dropping Missing Values:
    Name   Age   Salary
0  Alice  25.0  50000.0
3  David  22.0  45000.0
4  Emily  35.0  70000.0

DataFrame after Filling Missing Values:
      Name   Age   Salary
0    Alice  25.0  50000.0
1      Bob  30.0  56250.0
2  Charlie  28.0  60000.0
3    David  22.0  45000.0
4    Emily  35.0  70000.0

C:\Users\Anusha V\AppData\Local\Temp\ipykernel_7028\26514610.py:23: Future
Warning: Dropping of nuisance columns in DataFrame reductions (with 'numer
ic_only=None') is deprecated; in a future version this will raise TypeErro
r.  Select only valid columns before calling the reduction.
  df_filled = df.fillna(df.mean())
```

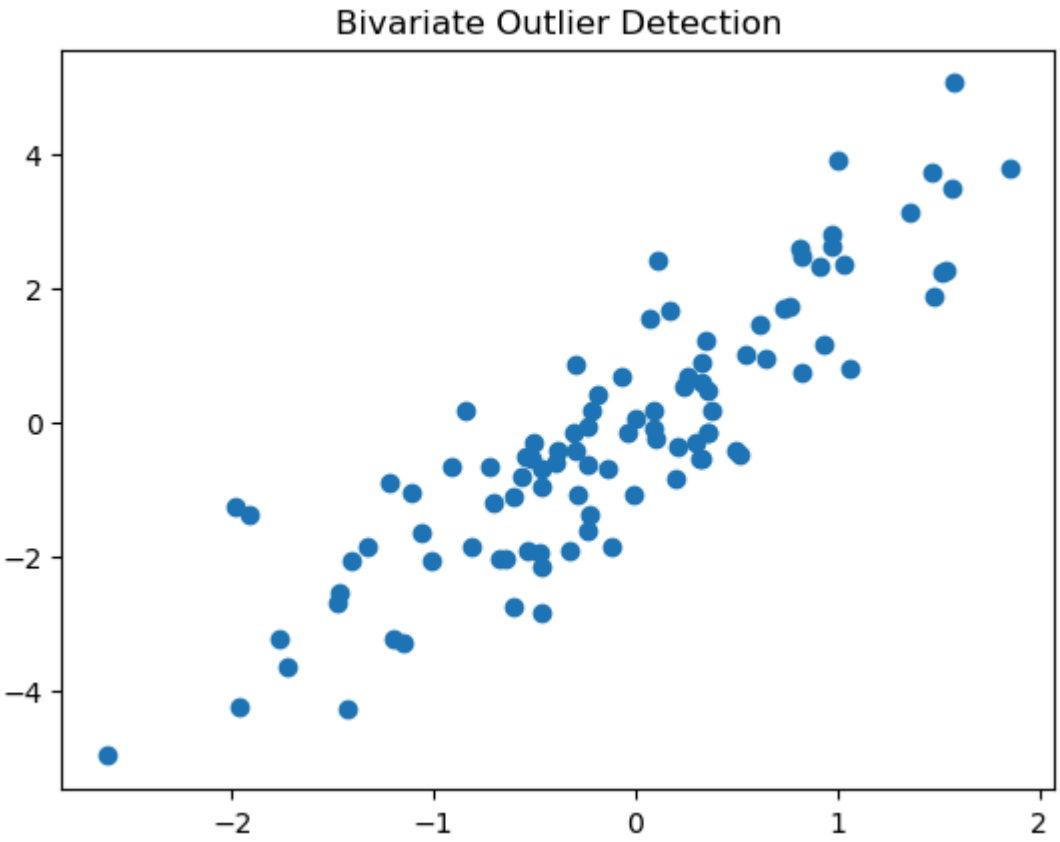
In [11]:

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

# Generating a sample univariate dataset
np.random.seed(42)
data_univariate = np.concatenate([np.random.normal(0, 1, 50), np.random

# Plotting the univariate data
plt.boxplot(data_univariate)
plt.title("Univariate Outlier Detection")
plt.show()

```

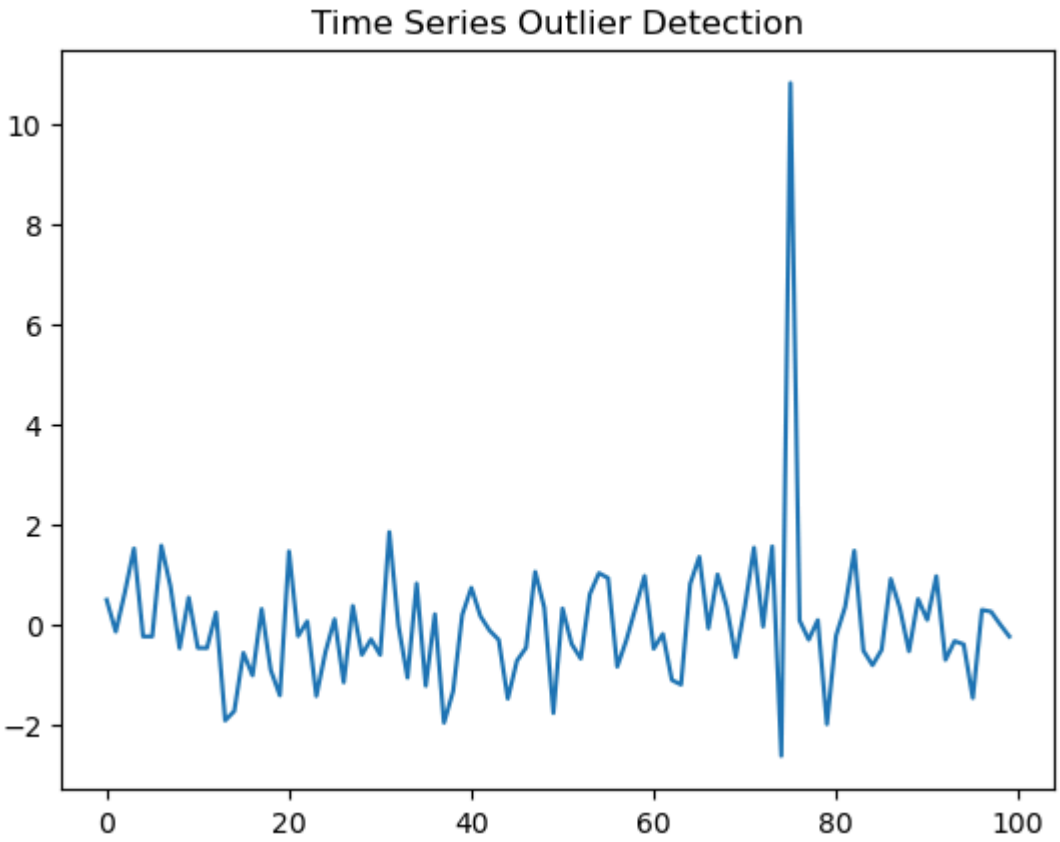


In [12]:

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

# Generating a sample bivariate dataset
np.random.seed(42)
data_bivariate_x = np.random.normal(0, 1, 100)
data_bivariate_y = data_bivariate_x * 2 + np.random.normal(0, 1, 100)

# Plotting the bivariate data
plt.scatter(data_bivariate_x, data_bivariate_y)
plt.title("Bivariate Outlier Detection")
plt.show()

```



In [13]:
```
import numpy as np
import matplotlib.pyplot as plt

# Generating a sample time series dataset with an outlier
np.random.seed(42)
data_timeseries = np.random.normal(0, 1, 100)
data_timeseries[75] += 10  # Introducing an outlier

# Plotting the time series data
plt.plot(data_timeseries)
plt.title("Time Series Outlier Detection")
plt.show()

```



Numerosity Data Reduction:

In [14]:
```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans

# Generating a sample dataset for numerosity reduction
X, _ = make_classification(n_samples=1000, n_features=10, random_state=

# Applying K-Means clustering for numerosity reduction
kmeans = KMeans(n_clusters=5, random_state=42)
X_reduced = kmeans.fit_predict(X)

# Print the reduced dataset
print("Original Data Shape:", X.shape)
print("Reduced Data Shape:", X_reduced.shape)

```

```
Original Data Shape: (1000, 10)
Reduced Data Shape: (1000,)
```

Dimensionality Data Reduction:

In [15]:
```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA

# Generating a sample dataset for dimensionality reduction
X, _ = make_classification(n_samples=1000, n_features=10, random_state=

# Applying Principal Component Analysis (PCA) for dimensionality reduct
pca = PCA(n_components=3)
X_reduced = pca.fit_transform(X)

# Print the reduced dataset
print("Original Data Shape:", X.shape)
print("Reduced Data Shape:", X_reduced.shape)

```

```
Original Data Shape: (1000, 10)
Reduced Data Shape: (1000, 3)
```

Data transformation in python code

In [16]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Sample data
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 22, 35],
        'Gender': ['Female', 'Male', 'Male', 'Female'],
        'Salary': [50000, 60000, 45000, 70000]}

df = pd.DataFrame(data)

# Display original data
print("Original DataFrame:")
print(df)

# Define columns for different transformations
numeric_features = ['Age', 'Salary']
categorical_features = ['Gender']

# Create transformers
numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder()

# Combine transformers into a preprocessor using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Define a pipeline with the preprocessor
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Fit and transform the data
transformed_data = pipeline.fit_transform(df)

# Display transformed data
transformed_df = pd.DataFrame(transformed_data, columns=['Age_scaled',
print("\nTransformed DataFrame:")
print(transformed_df)

```

```
Original DataFrame:
      Name  Age  Gender  Salary
0    Alice   25  Female   50000
1      Bob   30    Male   60000
2  Charlie   22    Male   45000
3    David   35  Female   70000

Transformed DataFrame:
   Age_scaled  Salary_scaled  Gender_Female  Gender_Male
0   -0.606092      -0.650945            1.0          0.0
1    0.404061       0.390567            0.0          1.0
2   -1.212183      -1.171700            0.0          1.0
3    1.414214       1.432078            1.0          0.0
```

Data transformation with binary coding,ranking transformation, discretization in python

In [17]:

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer, MinMaxScaler, KBinsDi

# Sample data
data = {'Category': ['A', 'B', 'A', 'C', 'B'],
        'Value': [10, 20, 15, 5, 25]}

df = pd.DataFrame(data)

# Display original data
print("Original DataFrame:")
print(df)

# Binary coding for categorical variable 'Category'
lb = LabelBinarizer()
binary_coding_result = lb.fit_transform(df['Category'])
binary_coding_df = pd.DataFrame(binary_coding_result, columns=[f'{col}_

# Ranking transformation for numerical variable 'Value'
df['Value_rank'] = df['Value'].rank()

# Discretization for numerical variable 'Value'
discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='ur
df['Value_discretized'] = discretizer.fit_transform(df[['Value']])

# Display transformed data
print("\nDataFrame after Transformation:")
print(binary_coding_df)
print("\nDataFrame with Ranking Transformation:")
print(df[['Value', 'Value_rank']])
print("\nDataFrame with Discretization:")
print(df[['Value', 'Value_discretized']])

```

```
Original DataFrame:
  Category  Value
0        A     10
1        B     20
2        A     15
3        C      5
4        B     25

DataFrame after Transformation:
   A_binary  B_binary  C_binary
0         1         0         0
1         0         1         0
2         1         0         0
3         0         0         1
4         0         1         0

DataFrame with Ranking Transformation:
   Value  Value_rank
0     10         2.0
1     20         4.0
2     15         3.0
3      5         1.0
4     25         5.0

DataFrame with Discretization:
   Value  Value_discretized
0     10                0.0
1     20                2.0
2     15                1.0
3      5                0.0
4     25                2.0
```

Data splitting using train_test_split

In [19]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Sample data
data = {'Feature1': [1, 2, 3, 4, 5],
        'Feature2': [10, 20, 30, 40, 50],
        'Target': [0, 1, 0, 1, 1]}

df = pd.DataFrame(data)

# Splitting the data into features (X) and target variable (y)
X = df[['Feature1', 'Feature2']]
y = df['Target']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Display the split datasets
print("X_train:")
print(X_train)
print("\ny_train:")
print(y_train)
print("\nX_test:")
print(X_test)
print("\ny_test:")
print(y_test)

```

```
X_train:
   Feature1  Feature2
4         5        50
2         3        30
0         1        10
3         4        40

y_train:
4    1
2    0
0    0
3    1
Name: Target, dtype: int64

X_test:
   Feature1  Feature2
1         2        20

y_test:
1    1
Name: Target, dtype: int64
```

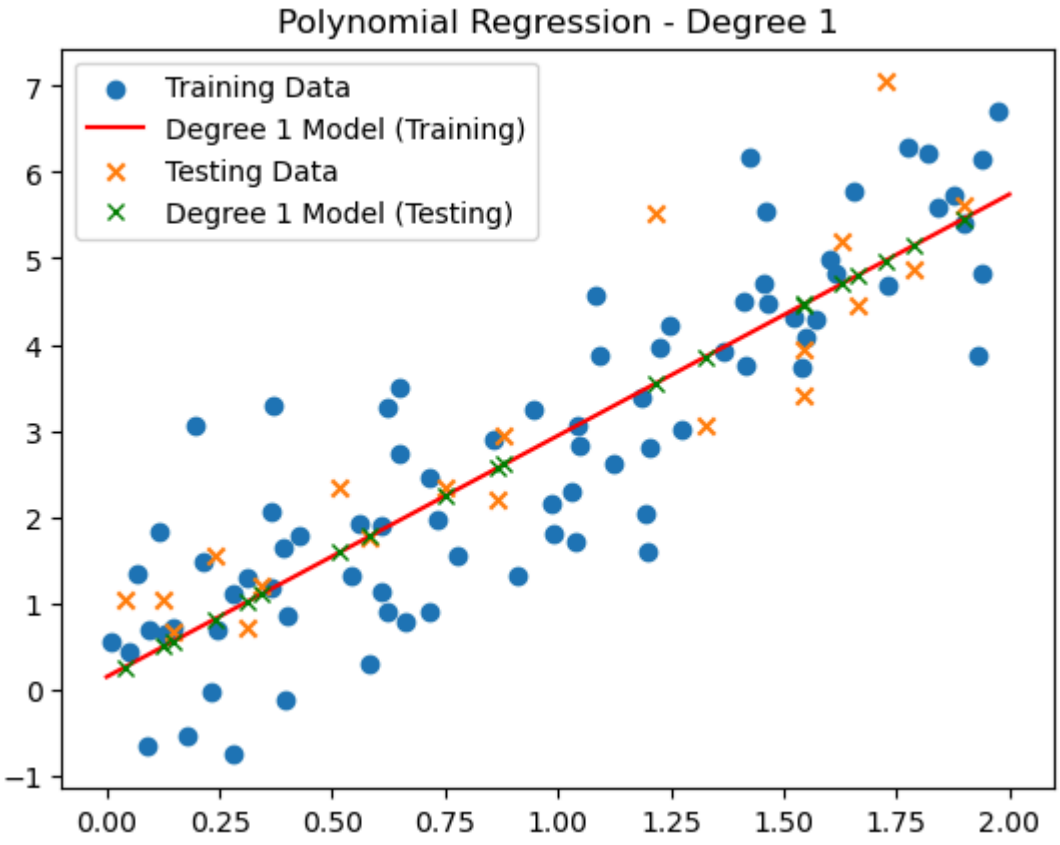
Python code for underfitting and overfitting

In [30]:

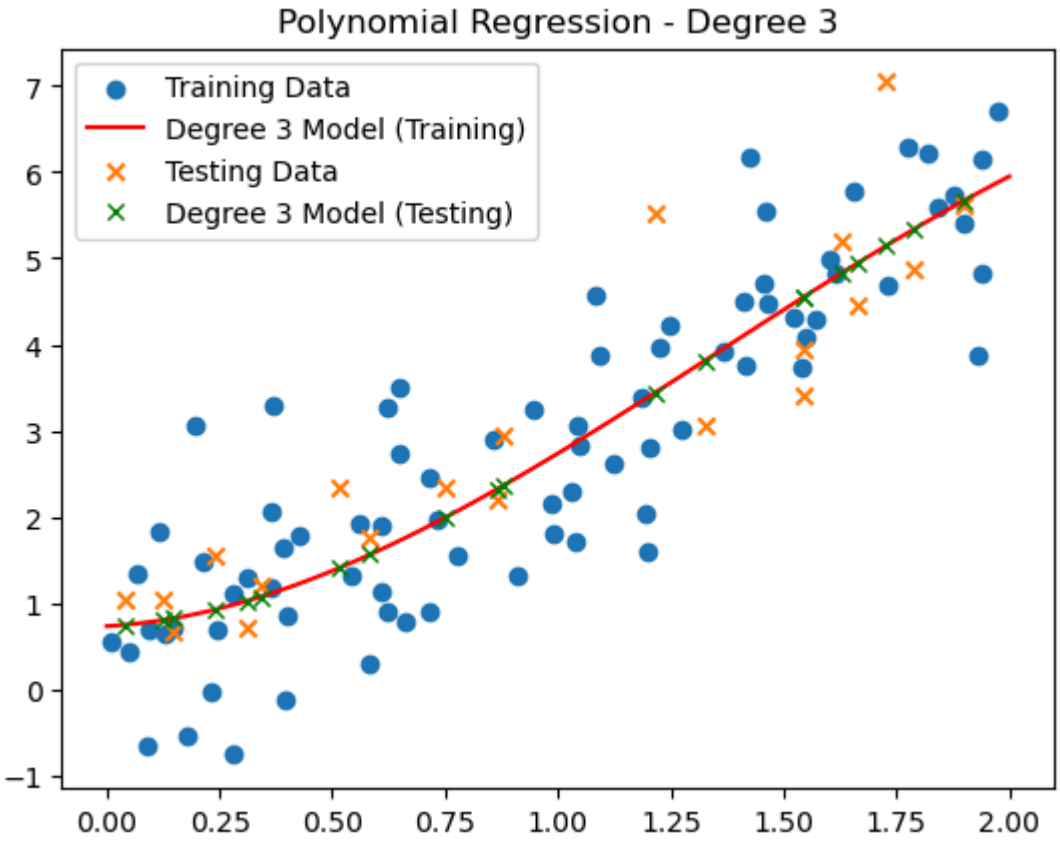
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Generate synthetic data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 3 * X + np.random.randn(100, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Function to plot the data and model predictions
def plot_model(degree, X, y, X_test, y_test):
    poly_features = PolynomialFeatures(degree=degree, include_bias=Fals
    X_poly = poly_features.fit_transform(X)

    model = LinearRegression()
    model.fit(X_poly, y)

    # Training data
    plt.scatter(X, y, label='Training Data')
    x_range = np.linspace(0, 2, 100).reshape(-1, 1)
    x_range_poly = poly_features.transform(x_range)
    plt.plot(x_range, model.predict(x_range_poly), label=f'Degree {degr

    # Testing data
    plt.scatter(X_test, y_test, label='Testing Data', marker='x')
    x_test_poly = poly_features.transform(X_test)
    plt.plot(X_test, model.predict(x_test_poly), 'x', label=f'Degree {d

    plt.title(f'Polynomial Regression - Degree {degree}')
    plt.legend()
    plt.show()

# Try different polynomial degrees to observe underfitting and overfitt
degrees = [1, 3, 10]
for degree in degrees:
    poly_features = PolynomialFeatures(degree=degree, include_bias=Fals
    X_poly = poly_features.fit_transform(X_train)

    model = LinearRegression()
    model.fit(X_poly, y_train)

    # Training error
    y_train_pred = model.predict(X_poly)
    train_error = mean_squared_error(y_train, y_train_pred)

    # Testing error
    X_test_poly = poly_features.transform(X_test)
    y_test_pred = model.predict(X_test_poly)
    test_error = mean_squared_error(y_test, y_test_pred)

    print(f'Degree {degree} - Training Error: {train_error:.3f}, Testin

    # Plot the model
    plot_model(degree, X_train, y_train, X_test, y_test)
```

```
61
```

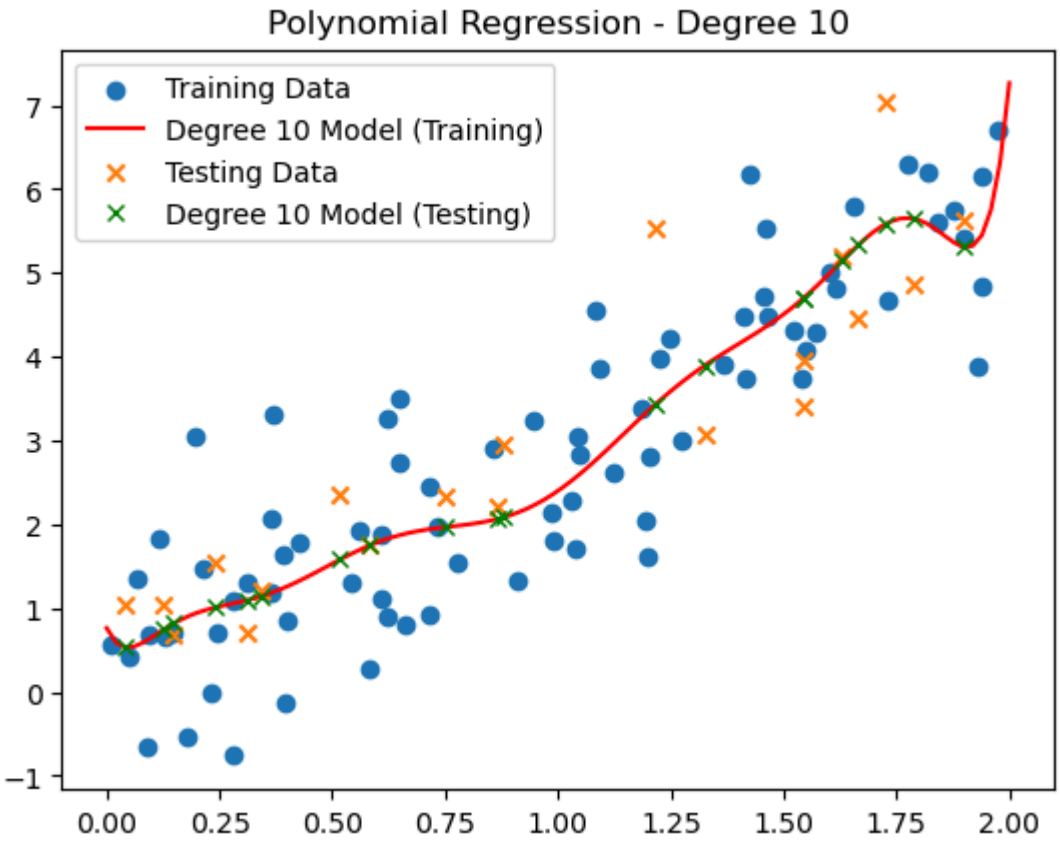
```
Degree 1 - Training Error: 0.848, Testing Error: 0.654
```




```
Degree 3 - Training Error: 0.808, Testing Error: 0.642
```



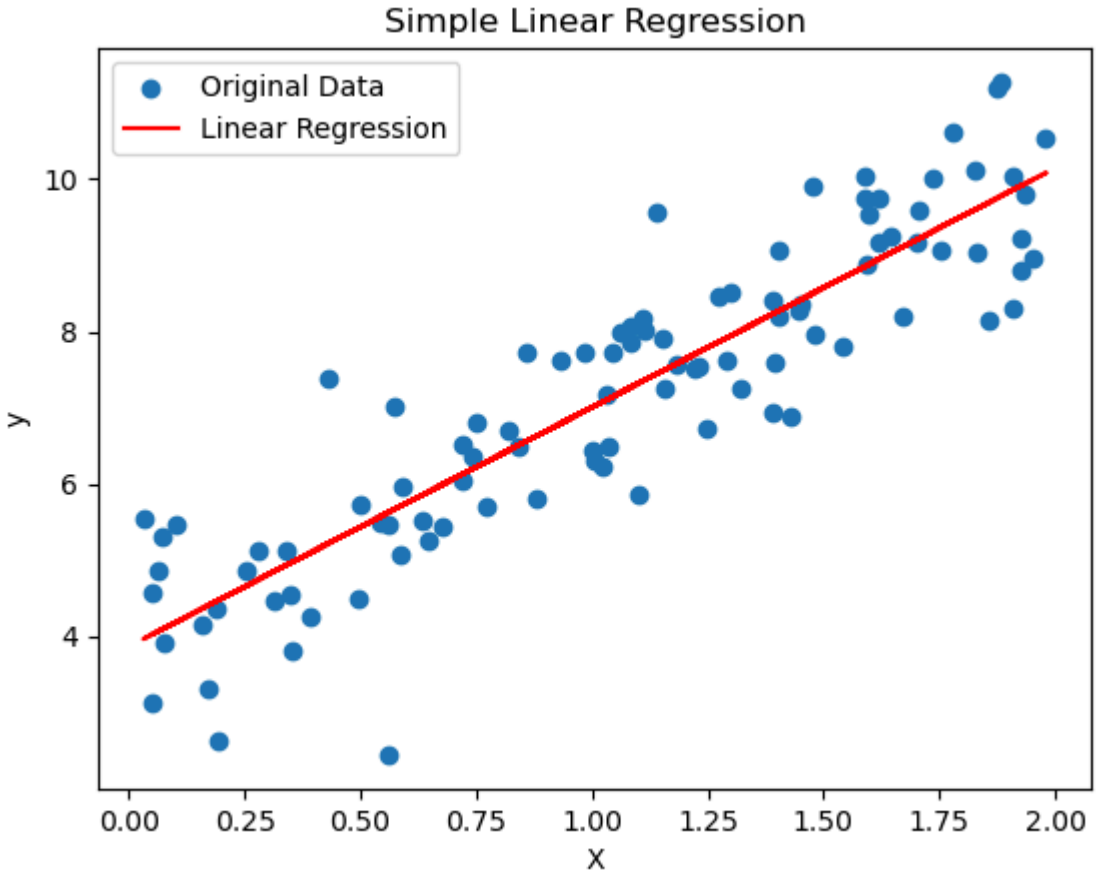

```
Degree 10 - Training Error: 0.770, Testing Error: 0.661
```



simple linear regression model

In [21]:

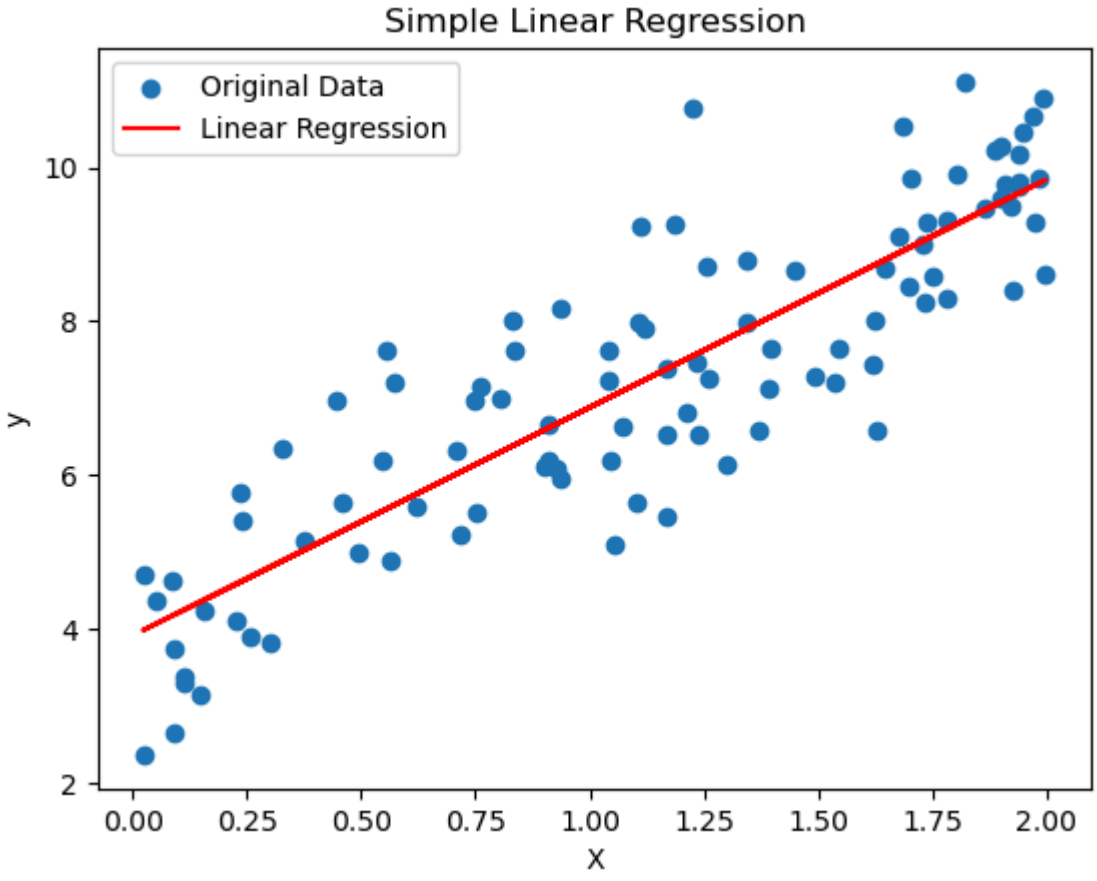
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate sample data
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Train a simple linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Plot the data and the linear regression line
plt.scatter(X, y, label='Original Data')
plt.plot(X, lin_reg.predict(X), 'r', label='Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()

```



Simple Linear Regression:

In [22]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate sample data
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Train a simple linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Plot the data and the linear regression line
plt.scatter(X, y, label='Original Data')
plt.plot(X, lin_reg.predict(X), 'r', label='Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()

```



Multiple Linear Regression:

In [23]:

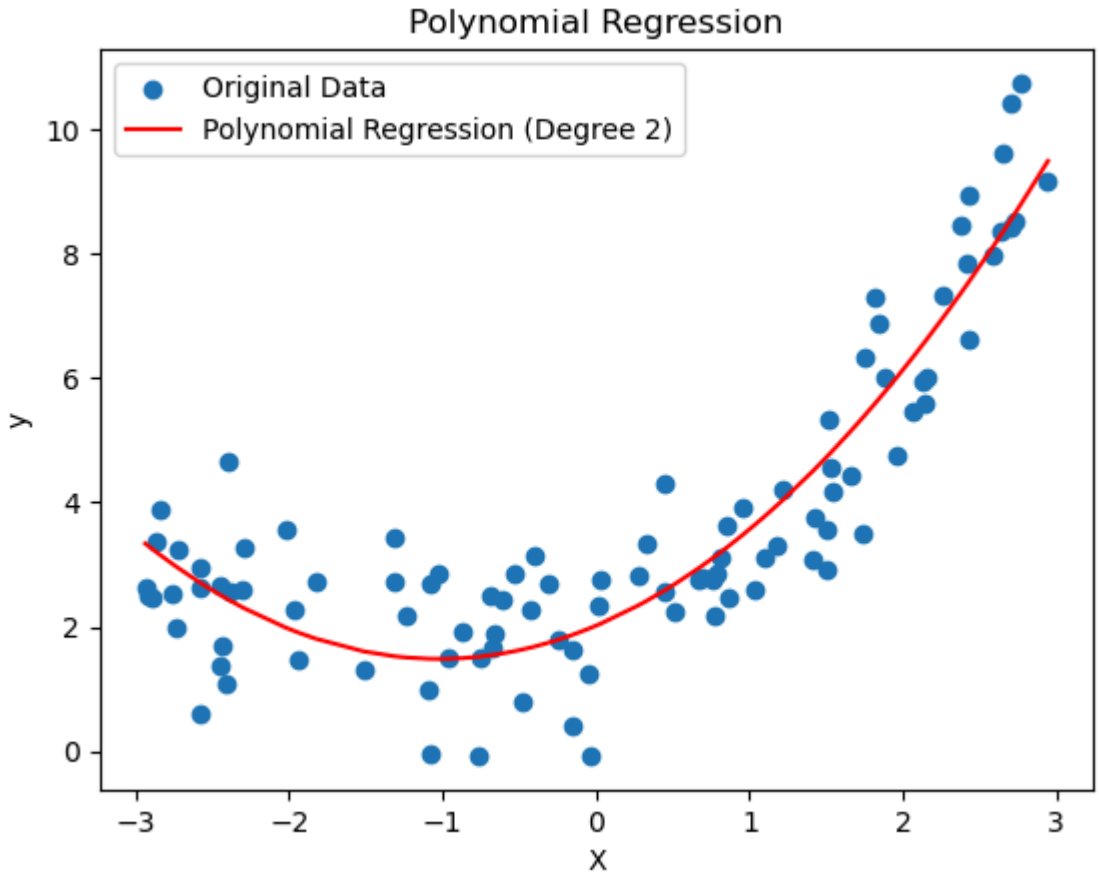
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate sample data with multiple features
np.random.seed(42)
X_mult = 2 * np.random.rand(100, 2)
y_mult = 4 + 3 * X_mult[:, 0] + 5 * X_mult[:, 1] + np.random.randn(100)

# Train a multiple linear regression model
lin_reg_mult = LinearRegression()
lin_reg_mult.fit(X_mult, y_mult)

# Display the coefficients and intercept
print("Coefficients:", lin_reg_mult.coef_)
print("Intercept:", lin_reg_mult.intercept_)

# Note: Visualization is not straightforward in multiple dimensions.

```

```
Coefficients: [3.16933339 5.17747302]
Intercept: 3.7722722641017086
```

Polynomial Regression:

In [24]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Generate sample data
X_poly = 6 * np.random.rand(100, 1) - 3
y_poly = 0.5 * X_poly**2 + X_poly + 2 + np.random.randn(100, 1)

# Transform features to include polynomial terms up to degree 2
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly_transformed = poly_features.fit_transform(X_poly)

# Train a polynomial regression model
lin_reg_poly = LinearRegression()
lin_reg_poly.fit(X_poly_transformed, y_poly)

# Sort the data for better visualization
X_poly_sorted = np.sort(X_poly, axis=0)
y_poly_pred = lin_reg_poly.predict(poly_features.transform(X_poly_sorte

# Plot the data and the polynomial regression line
plt.scatter(X_poly, y_poly, label='Original Data')
plt.plot(X_poly_sorted, y_poly_pred, 'r', label='Polynomial Regression
plt.xlabel('X')
plt.ylabel('y')
plt.title('Polynomial Regression')
plt.legend()
plt.show()

```

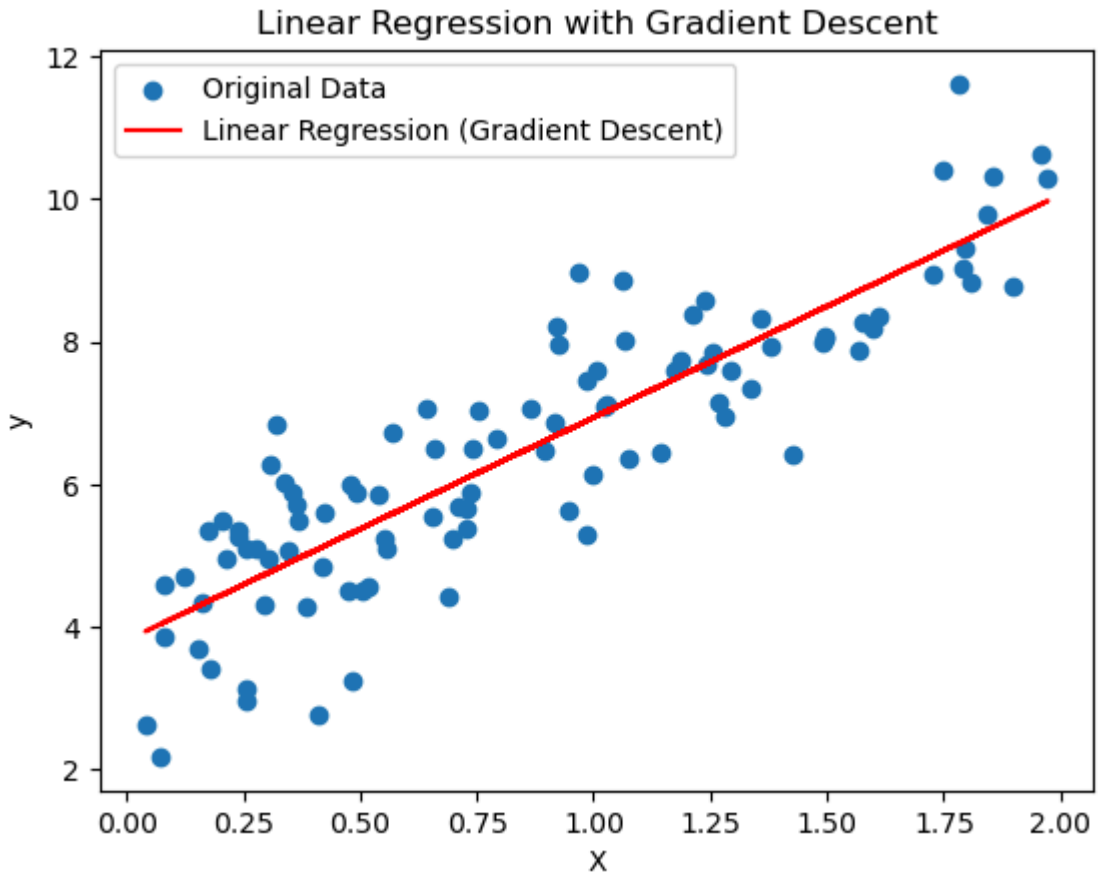


Linear Regression with Gradient Descent:

In [25]:

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

# Generate sample data
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Add bias term to features
X_b = np.c_[np.ones((100, 1)), X]

# Set hyperparameters
learning_rate = 0.01
n_iterations = 1000

# Initialize coefficients randomly
theta = np.random.randn(2, 1)

# Perform gradient descent
for iteration in range(n_iterations):
    gradients = 2/100 * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - learning_rate * gradients

# Display the final coefficients
print("Intercept:", theta[0][0])
print("Coefficient:", theta[1][0])

# Plot the data and the linear regression line
plt.scatter(X, y, label='Original Data')
plt.plot(X, X_b.dot(theta), 'r', label='Linear Regression (Gradient Des
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression with Gradient Descent')
plt.legend()
plt.show()

```

```
Intercept: 3.8131886040073186
Coefficient: 3.12272055734439
```

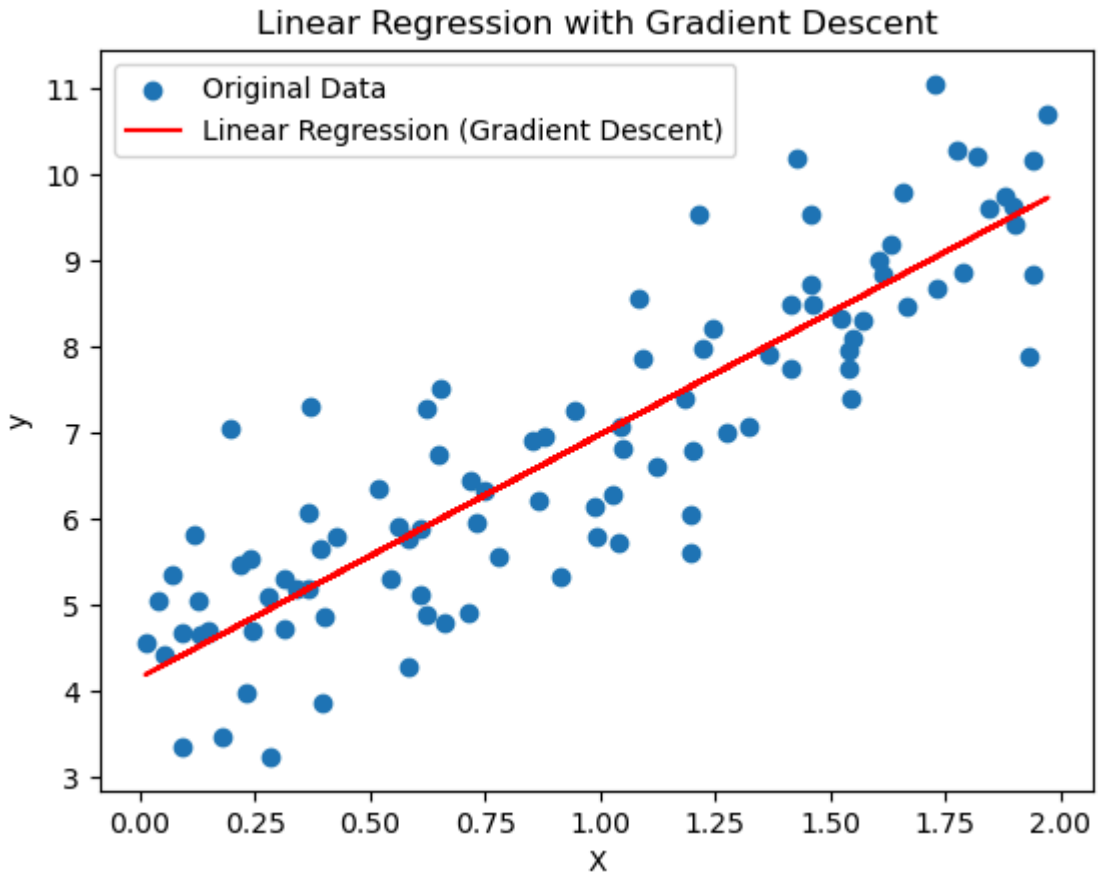


Python code for R-square,RMSE, Gradient descent,optimize regression model

In [26]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression

# Generate sample data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Add bias term to features
X_b = np.c_[np.ones((100, 1)), X]

# Initialize coefficients randomly
theta = np.random.randn(2, 1)

# Set hyperparameters
learning_rate = 0.01
n_iterations = 1000

# Perform gradient descent
for iteration in range(n_iterations):
    gradients = 2/100 * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - learning_rate * gradients

# Calculate predictions
y_pred = X_b.dot(theta)

# Calculate R-squared
r2 = r2_score(y, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mean_squared_error(y, y_pred))

# Display results
print("R-squared:", r2)
print("RMSE:", rmse)

# Plot the data and the optimized linear regression line
plt.scatter(X, y, label='Original Data')
plt.plot(X, y_pred, 'r', label='Linear Regression (Gradient Descent)')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression with Gradient Descent')
plt.legend()
plt.show()

```

```
R-squared: 0.7689928256499341
RMSE: 0.8986467066353333
```



Hold-out Method:

In [27]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate sample data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Split the data into training and testing sets using hold-out method
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Train a linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lin_reg.predict(X_test)

# Evaluate the model
mse_holdout = mean_squared_error(y_test, y_pred)
print("Hold-out Method MSE:", mse_holdout)

```

```
Hold-out Method MSE: 0.6536995137170021
```

Leave-One-Out Cross-Validation (LOOCV):

In [28]:
```
from sklearn.model_selection import LeaveOneOut

# Perform leave-one-out cross-validation
loo = LeaveOneOut()

mse_loocv = 0

for train_index, test_index in loo.split(X):
    X_train_loo, X_test_loo = X[train_index], X[test_index]
    y_train_loo, y_test_loo = y[train_index], y[test_index]

    lin_reg_loo = LinearRegression()
    lin_reg_loo.fit(X_train_loo, y_train_loo)

    y_pred_loo = lin_reg_loo.predict(X_test_loo)
    mse_loocv += mean_squared_error(y_test_loo, y_pred_loo)

mse_loocv /= len(X)
print("LOOCV MSE:", mse_loocv)

```

```
LOOCV MSE: 0.8389654996853929
```

K-Fold Cross-Validation:

In [29]:
```
from sklearn.model_selection import KFold

# Perform k-fold cross-validation (k=5 in this example)
kf = KFold(n_splits=5, shuffle=True, random_state=42)

mse_kfold = 0

for train_index, test_index in kf.split(X):
    X_train_kfold, X_test_kfold = X[train_index], X[test_index]
    y_train_kfold, y_test_kfold = y[train_index], y[test_index]

    lin_reg_kfold = LinearRegression()
    lin_reg_kfold.fit(X_train_kfold, y_train_kfold)

    y_pred_kfold = lin_reg_kfold.predict(X_test_kfold)
    mse_kfold += mean_squared_error(y_test_kfold, y_pred_kfold)

mse_kfold /= kf.get_n_splits()
print("K-Fold Cross-Validation MSE:", mse_kfold)

```

```
K-Fold Cross-Validation MSE: 0.8364353497704167
```

A python code on data exploration, preprocessing and splitting on Boston housing price from sci-kit learn

In [30]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Boston Housing dataset
boston = load_boston()

# Create a DataFrame for easier exploration
boston_df = pd.DataFrame(data=boston.data, columns=boston.feature_names
boston_df['TARGET'] = boston.target

# Explore the dataset
print("Head of the DataFrame:")
print(boston_df.head())

print("\nSummary Statistics:")
print(boston_df.describe())

# Visualize the correlation matrix
correlation_matrix = boston_df.corr()
plt.figure(figsize=(12, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none', a
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns,
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.title("Correlation Matrix")
plt.show()

# Preprocessing: Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(boston.data)
y = boston.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Display the shape of the training and testing sets
print("\nShape of Training Set (X_train, y_train):", X_train.shape, y_t
print("Shape of Testing Set (X_test, y_test):", X_test.shape, y_test.sh

```

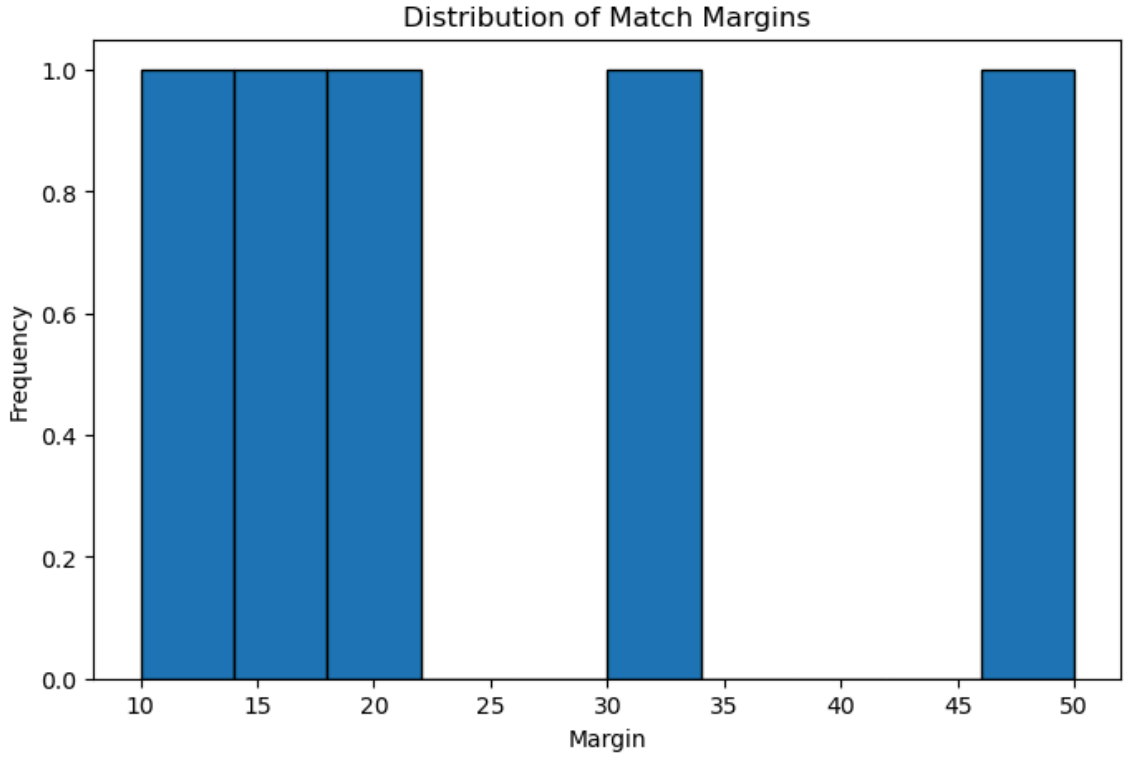
```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\utils\deprecatio
n.py:87: FutureWarning: Function load_boston is deprecated; `load_bosto
n` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can r
efer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use
of this
    dataset unless the purpose of the code is to study and educate abou
t
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np
```

A python code on data exploration, preprocessing and splitting on cricket match result-past data from sci-kit learn

In [31]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Sample cricket match dataset (hypothetical)
data = {
    'Team1': ['India', 'Australia', 'England', 'India', 'Australia'],
    'Team2': ['Australia', 'England', 'India', 'Australia', 'England'],
    'Winner': ['India', 'Australia', 'India', 'India', 'England'],
    'Margin': [50, 20, 30, 10, 15],
    'Toss_Winner': ['India', 'Australia', 'England', 'Australia', 'Engl
    'Toss_Decision': ['Bat', 'Field', 'Bat', 'Field', 'Field']
}

cricket_df = pd.DataFrame(data)

# Explore the dataset
print("Head of the DataFrame:")
print(cricket_df.head())

print("\nSummary Statistics:")
print(cricket_df.describe())

# Visualize the match margins
plt.figure(figsize=(8, 5))
plt.hist(cricket_df['Margin'], bins=10, edgecolor='black')
plt.title("Distribution of Match Margins")
plt.xlabel("Margin")
plt.ylabel("Frequency")
plt.show()

# Preprocessing: Convert categorical features to numerical using one-ho
cricket_df_encoded = pd.get_dummies(cricket_df, columns=['Team1', 'Team

# Split the data into features (X) and target variable (y)
X = cricket_df_encoded.drop('Winner', axis=1)
y = cricket_df_encoded['Winner']

# Standardize the features
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_standardized, y,

# Display the shape of the training and testing sets
print("\nShape of Training Set (X_train, y_train):", X_train.shape, y_t
print("Shape of Testing Set (X_test, y_test):", X_test.shape, y_test.sh

```

```
Head of the DataFrame:
       Team1      Team2     Winner  Margin Toss_Winner Toss_Decision
0      India  Australia      India      50       India           Bat
1  Australia    England  Australia      20   Australia         Field
2    England      India      India      30     England           Bat
3      India  Australia      India      10   Australia         Field
4  Australia    England    England      15     England         Field

Summary Statistics:
          Margin
count   5.000000
mean   25.000000
std    15.811388
min    10.000000
25%    15.000000
50%    20.000000
75%    30.000000
max    50.000000
```



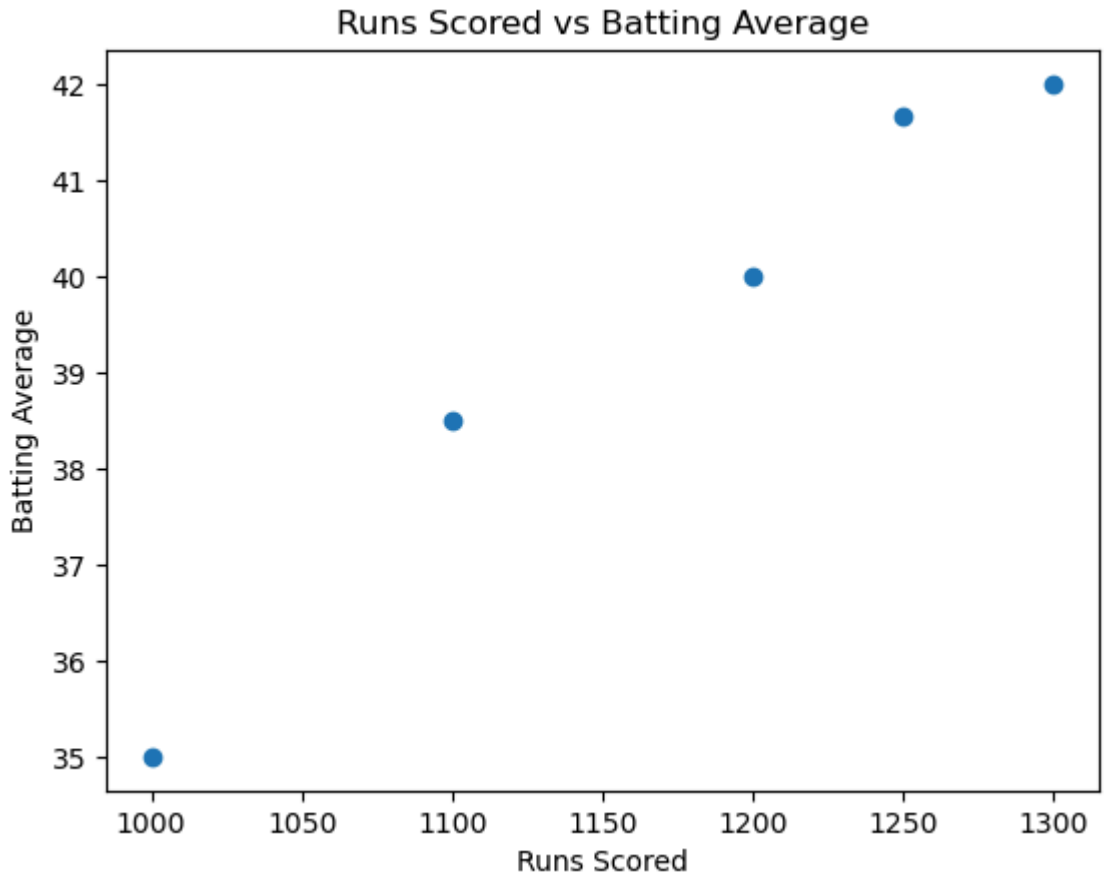

```
Shape of Training Set (X_train, y_train): (4, 8) (4,)
Shape of Testing Set (X_test, y_test): (1, 8) (1,)
```

A python code on data exploration, preprocessing and splitting on performance of a cricket player from sci-kit learn

In [32]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Sample cricket player performance dataset (hypothetical)
data = {
    'Matches_Played': [50, 48, 52, 45, 51],
    'Runs_Scored': [1200, 1100, 1300, 1000, 1250],
    'Batting_Average': [40.0, 38.5, 42.0, 35.0, 41.67],
    'Strike_Rate': [85.0, 82.5, 87.5, 75.0, 80.0],
    'Centuries': [5, 4, 6, 3, 5],
    'Wickets_Taken': [20, 18, 22, 15, 21],
    'Bowling_Average': [25.0, 27.0, 23.5, 30.0, 24.5],
    'Player_Rating': [8.5, 8.2, 8.8, 7.9, 8.3]
}

cricket_player_df = pd.DataFrame(data)

# Explore the dataset
print("Head of the DataFrame:")
print(cricket_player_df.head())

print("\nSummary Statistics:")
print(cricket_player_df.describe())

# Visualize Runs Scored vs Batting Average
plt.scatter(cricket_player_df['Runs_Scored'], cricket_player_df['Battin
plt.title("Runs Scored vs Batting Average")
plt.xlabel("Runs Scored")
plt.ylabel("Batting Average")
plt.show()

# Preprocessing: Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(cricket_player_df.drop('Player_Rating', axis=1
y = cricket_player_df['Player_Rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Display the shape of the training and testing sets
print("\nShape of Training Set (X_train, y_train):", X_train.shape, y_t
print("Shape of Testing Set (X_test, y_test):", X_test.shape, y_test.sh

```

```
Head of the DataFrame:
   Matches_Played  Runs_Scored  Batting_Average  Strike_Rate  Centuries  \
0              50         1200            40.00         85.0          5
1              48         1100            38.50         82.5          4
2              52         1300            42.00         87.5          6
3              45         1000            35.00         75.0          3
4              51         1250            41.67         80.0          5

   Wickets_Taken  Bowling_Average  Player_Rating
0             20             25.0            8.5
1             18             27.0            8.2
2             22             23.5            8.8
3             15             30.0            7.9
4             21             24.5            8.3

Summary Statistics:
       Matches_Played  Runs_Scored  Batting_Average  Strike_Rate  Centurie
s  \
count        5.000000     5.000000          5.00000     5.000000   5.00000
0
mean        49.200000  1170.000000         39.43400    82.000000   4.60000
0
std          2.774887   120.415946          2.84768     4.808846   1.14017
5
min         45.000000  1000.000000         35.00000    75.000000   3.00000
0
25%         48.000000  1100.000000         38.50000    80.000000   4.00000
0
50%         50.000000  1200.000000         40.00000    82.500000   5.00000
0
75%         51.000000  1250.000000         41.67000    85.000000   5.00000
0
max         52.000000  1300.000000         42.00000    87.500000   6.00000
0

       Wickets_Taken  Bowling_Average  Player_Rating
count       5.000000         5.000000       5.000000
mean       19.200000        26.000000       8.340000
std         2.774887         2.573908       0.336155
min        15.000000        23.500000       7.900000
25%        18.000000        24.500000       8.200000
50%        20.000000        25.000000       8.300000
75%        21.000000        27.000000       8.500000
max        22.000000        30.000000       8.800000
```



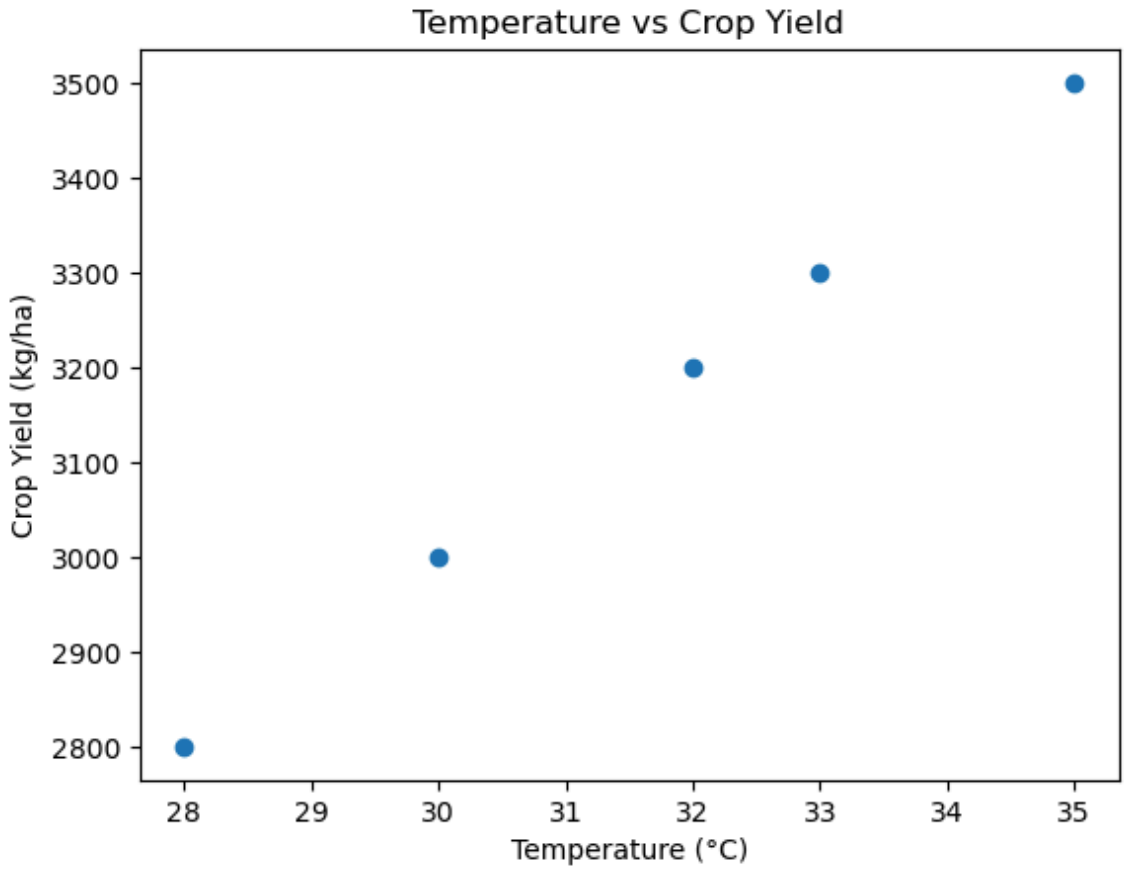

```
Shape of Training Set (X_train, y_train): (4, 7) (4,)
Shape of Testing Set (X_test, y_test): (1, 7) (1,)
```

A python code on data exploration, preprocessing and splitting on crop yield from sci-kit learn

In [33]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Sample crop yield dataset (hypothetical)
data = {
    'Temperature': [30, 32, 28, 35, 33],
    'Rainfall': [100, 80, 120, 90, 110],
    'Fertilizer_Use': [20, 25, 18, 30, 22],
    'Crop_Yield': [3000, 3200, 2800, 3500, 3300]
}

crop_yield_df = pd.DataFrame(data)

# Explore the dataset
print("Head of the DataFrame:")
print(crop_yield_df.head())

print("\nSummary Statistics:")
print(crop_yield_df.describe())

# Visualize the relationship between Temperature and Crop Yield
plt.scatter(crop_yield_df['Temperature'], crop_yield_df['Crop_Yield'])
plt.title("Temperature vs Crop Yield")
plt.xlabel("Temperature (°C)")
plt.ylabel("Crop Yield (kg/ha)")
plt.show()

# Preprocessing: Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(crop_yield_df.drop('Crop_Yield', axis=1))
y = crop_yield_df['Crop_Yield']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Display the shape of the training and testing sets
print("\nShape of Training Set (X_train, y_train):", X_train.shape, y_t
print("Shape of Testing Set (X_test, y_test):", X_test.shape, y_test.sh

```

```
Head of the DataFrame:
   Temperature  Rainfall  Fertilizer_Use  Crop_Yield
0           30       100              20        3000
1           32        80              25        3200
2           28       120              18        2800
3           35        90              30        3500
4           33       110              22        3300

Summary Statistics:
       Temperature    Rainfall  Fertilizer_Use   Crop_Yield
count     5.000000    5.000000        5.000000     5.000000
mean     31.600000  100.000000       23.000000  3160.000000
std       2.701851   15.811388        4.690416   270.185122
min      28.000000   80.000000       18.000000  2800.000000
25%      30.000000   90.000000       20.000000  3000.000000
50%      32.000000  100.000000       22.000000  3200.000000
75%      33.000000  110.000000       25.000000  3300.000000
max      35.000000  120.000000       30.000000  3500.000000
```



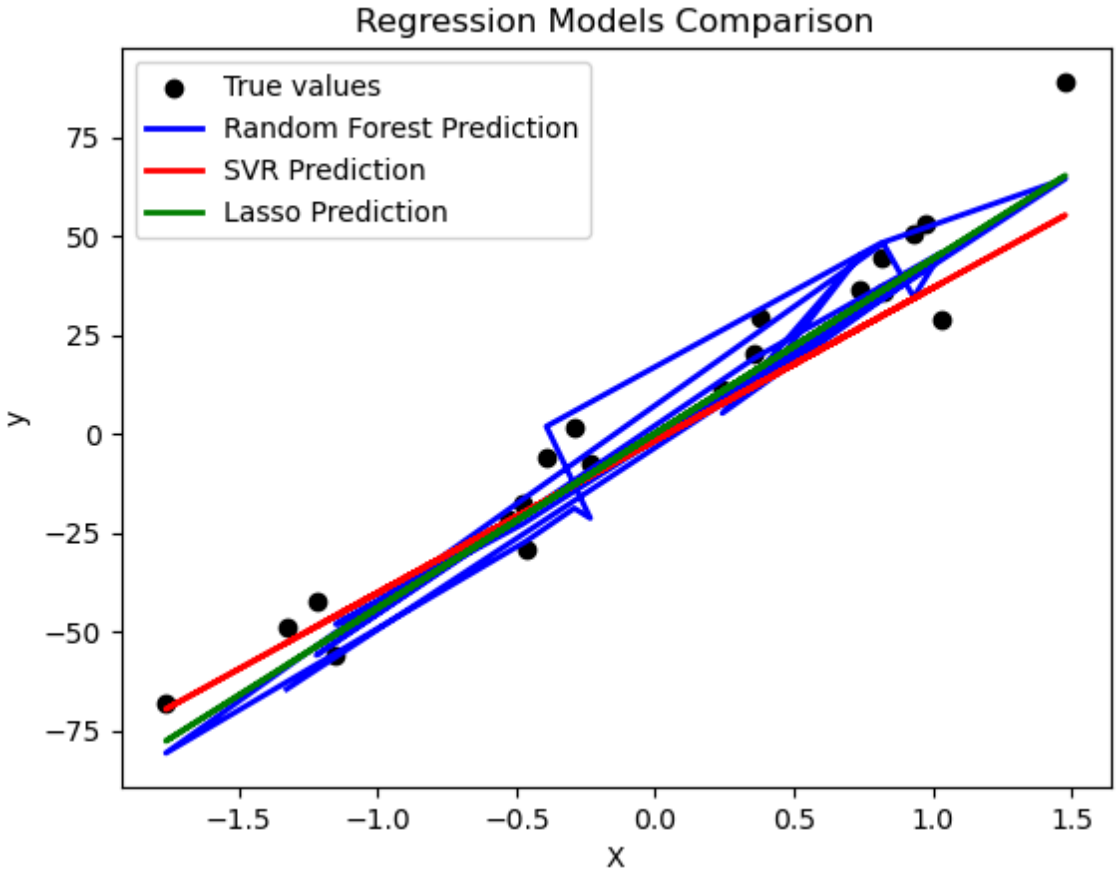

```
Shape of Training Set (X_train, y_train): (4, 3) (4,)
Shape of Testing Set (X_test, y_test): (1, 3) (1,)
```

In [35]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Generate a synthetic dataset
X, y = make_regression(n_samples=100, n_features=1, noise=10, random_st

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Random Forest Regression
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Support Vector Regression (SVR)
svr_regressor = SVR(kernel='linear')
svr_regressor.fit(X_train_scaled, y_train)

# Lasso Regression
lasso_regressor = Lasso(alpha=0.1, random_state=42)
lasso_regressor.fit(X_train_scaled, y_train)

# Predictions
y_pred_rf = rf_regressor.predict(X_test)
y_pred_svr = svr_regressor.predict(X_test_scaled)
y_pred_lasso = lasso_regressor.predict(X_test_scaled)

# Evaluate the models
mse_rf = mean_squared_error(y_test, y_pred_rf)
mse_svr = mean_squared_error(y_test, y_pred_svr)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)

print("Mean Squared Error (Random Forest):", mse_rf)
print("Mean Squared Error (SVR):", mse_svr)
print("Mean Squared Error (Lasso):", mse_lasso)

# Plot the results
plt.scatter(X_test, y_test, color='black', label='True values')
plt.plot(X_test, y_pred_rf, color='blue', linewidth=2, label='Random Fo
plt.plot(X_test, y_pred_svr, color='red', linewidth=2, label='SVR Predi
plt.plot(X_test, y_pred_lasso, color='green', linewidth=2, label='Lasso

plt.title('Regression Models Comparison')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
59
```

```
Mean Squared Error (Random Forest): 160.99855696169863
Mean Squared Error (SVR): 154.7063747913098
Mean Squared Error (Lasso): 104.41512794678081
```



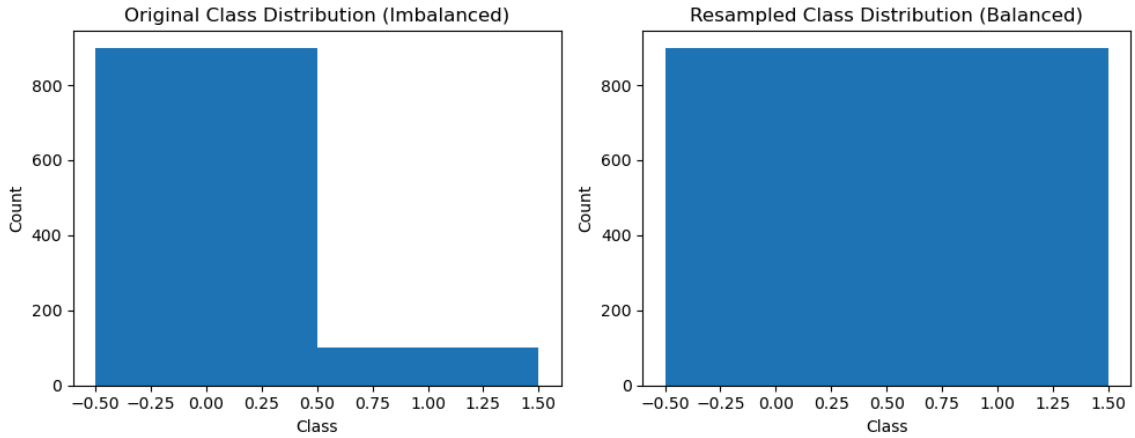
Python code for binary classification,multi-label classification,multi-class classification and imbalanced classification

In [36]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, conf
from imblearn.over_sampling import SMOTE

# Generate synthetic datasets for binary and multiclass classification
X_bin, y_bin = make_classification(n_samples=1000, n_features=20, n_inf
X_multi, y_multi = make_classification(n_samples=1000, n_features=20, r

# Binary Classification
X_bin_train, X_bin_test, y_bin_train, y_bin_test = train_test_split(X_b

# Train a Logistic Regression model for binary classification
logreg_bin = LogisticRegression()
logreg_bin.fit(X_bin_train, y_bin_train)
y_bin_pred = logreg_bin.predict(X_bin_test)

# Evaluate the model for binary classification
print("Binary Classification:")
print("Accuracy:", accuracy_score(y_bin_test, y_bin_pred))
print("Classification Report:\n", classification_report(y_bin_test, y_b
print("Confusion Matrix:\n", confusion_matrix(y_bin_test, y_bin_pred))

# Multiclass Classification
X_multi_train, X_multi_test, y_multi_train, y_multi_test = train_test_s

# Train a Logistic Regression model for multiclass classification
logreg_multi = LogisticRegression(multi_class='ovr')  # One-vs-Rest str
logreg_multi.fit(X_multi_train, y_multi_train)
y_multi_pred = logreg_multi.predict(X_multi_test)

# Evaluate the model for multiclass classification
print("\nMulticlass Classification:")
print("Accuracy:", accuracy_score(y_multi_test, y_multi_pred))
print("Classification Report:\n", classification_report(y_multi_test, y
print("Confusion Matrix:\n", confusion_matrix(y_multi_test, y_multi_pre

# Imbalanced Classification (using SMOTE for oversampling)
X_imbalanced, y_imbalanced = make_classification(n_samples=1000, n_feat
                                                 weights=[0.9, 0.1], r

# Apply SMOTE to handle class imbalance
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_imbalanced, y_imbalance

# Split the resampled data into training and testing sets
X_resampled_train, X_resampled_test, y_resampled_train, y_resampled_tes
    X_resampled, y_resampled, test_size=0.2, random_state=42)

# Train a Random Forest model for imbalanced classification
rf_imbalanced = RandomForestClassifier(random_state=42)
rf_imbalanced.fit(X_resampled_train, y_resampled_train)
y_resampled_pred = rf_imbalanced.predict(X_resampled_test)

# Evaluate the model for imbalanced classification
```

```
print("\nImbalanced Classification:")
print("Accuracy:", accuracy_score(y_resampled_test, y_resampled_pred))
print("Classification Report:\n", classification_report(y_resampled_tes
print("Confusion Matrix:\n", confusion_matrix(y_resampled_test, y_resam

# Plot the original and resampled class distribution
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Class Distribution (Imbalanced)")
plt.hist(y_imbalanced, bins=[0, 1, 2], align='left')
plt.xlabel("Class")
plt.ylabel("Count")

plt.subplot(1, 2, 2)
plt.title("Resampled Class Distribution (Balanced)")
plt.hist(y_resampled, bins=[0, 1, 2], align='left')
plt.xlabel("Class")
plt.ylabel("Count")

plt.tight_layout()
plt.show()

```

```
Binary Classification:
Accuracy: 0.81
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.77      0.82       112
           1       0.75      0.86      0.80        88

    accuracy                           0.81       200
   macro avg       0.81      0.82      0.81       200
weighted avg       0.82      0.81      0.81       200

Confusion Matrix:
 [[86 26]
 [12 76]]

Multiclass Classification:
Accuracy: 0.6
Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.55      0.59        66
           1       0.55      0.65      0.60        63
           2       0.61      0.61      0.61        71

    accuracy                           0.60       200
   macro avg       0.60      0.60      0.60       200
weighted avg       0.60      0.60      0.60       200

Confusion Matrix:
 [[36 15 15]
 [10 41 12]
 [10 18 43]]

Imbalanced Classification:
Accuracy: 0.9777777777777777
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       179
           1       0.97      0.99      0.98       181

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360

Confusion Matrix:
 [[173   6]
 [  2 179]]
```



Python code for evaluation matrices for classification, confusion matrix, accuracy, precession and recall,specifity,f1-score,AUC-ROC

In [41]:
```
pip install scikit-learn matplotlib

```

```
Requirement already satisfied: scikit-learn in c:\users\anusha v\anaconda3
\lib\site-packages (1.0.2)
Requirement already satisfied: matplotlib in c:\users\anusha v\anaconda3\l
ib\site-packages (3.5.2)
Requirement already satisfied: scipy>=1.1.0 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-learn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\anusha v\a
naconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: numpy>=1.14.6 in c:\users\anusha v\anaconda
3\lib\site-packages (from scikit-learn) (1.24.4)
Requirement already satisfied: joblib>=0.11 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\anusha v\a
naconda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\anusha v\anaconda
3\lib\site-packages (from matplotlib) (9.2.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\anusha v\anac
onda3\lib\site-packages (from matplotlib) (1.4.2)
Requirement already satisfied: cycler>=0.10 in c:\users\anusha v\anaconda3
\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\anusha v\anac
onda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\anusha v\anaco
nda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: packaging>=20.0 in c:\users\anusha v\anacon
da3\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: six>=1.5 in c:\users\anusha v\anaconda3\lib
\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.

WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
```

In [27]:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision

# Generate synthetic dataset for demonstration
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Initialize and train a classification model (Random Forest in this ex
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Precision
precision = precision_score(y_test, y_pred)
print(f"Precision: {precision:.4f}")

# Recall
recall = recall_score(y_test, y_pred)
print(f"Recall: {recall:.4f}")

# Specificity (True Negative Rate)
specificity = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1
print(f"Specificity: {specificity:.4f}")

# F1-score
f1 = f1_score(y_test, y_pred)
print(f"F1-score: {f1:.4f}")

# AUC-ROC
y_pred_prob = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"AUC-ROC: {roc_auc:.4f}")

```

```
Confusion Matrix:
[[88  5]
 [15 92]]
Accuracy: 0.9000
Precision: 0.9485
Recall: 0.8598
Specificity: 0.9462
F1-score: 0.9020
AUC-ROC: 0.9379
```

Python code for Hyper parameter tuning for decesion tree classifier

In [39]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Decision Tree Classifier
dt_classifier = DecisionTreeClassifier()

# Define the hyperparameters to tune
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Use GridSearchCV to perform hyperparameter tuning
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_gr
grid_search.fit(X_train, y_train)

# Display the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:")
print(best_params)

# Evaluate the model with the best hyperparameters on the test set
best_dt_model = grid_search.best_estimator_
y_pred = best_dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy on Test Set:", accuracy)

```

```
Best Hyperparameters:
{'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 4, 'min_sa
mples_split': 2}

Accuracy on Test Set: 1.0
```

Logistic regression model in python

In [42]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, conf

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Logistic Regression model
logreg_model = LogisticRegression()

# Train the model on the training set
logreg_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logreg_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display the results
print("Logistic Regression Model Evaluation:")
print("Accuracy:", accuracy)
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

```

```
Logistic Regression Model Evaluation:
Accuracy: 1.0

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\linear_model\_logist
ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Python code for basic ensemble technique- Max voting, Averaging,Weighted average

In [43]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create three different models
model_rf = RandomForestClassifier(random_state=42)
model_lr = LogisticRegression(random_state=42)
model_svm = SVC(probability=True, random_state=42)  # SVC requires prob

# Train the models
model_rf.fit(X_train, y_train)
model_lr.fit(X_train, y_train)
model_svm.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = model_rf.predict(X_test)
y_pred_lr = model_lr.predict(X_test)
y_pred_svm = model_svm.predict(X_test)

# Ensembling techniques:

# Max Voting (Hard Voting)
y_pred_max_voting = np.array([np.argmax(np.bincount([pred_rf, pred_lr,

# Averaging
y_pred_averaging = (y_pred_rf + y_pred_lr + y_pred_svm) // 3

# Weighted Average
weights = {'rf': 2, 'lr': 1, 'svm': 1}  # Adjust weights based on model
y_pred_weighted_average = (weights['rf'] * y_pred_rf + weights['lr'] *

# Evaluate individual models
acc_rf = accuracy_score(y_test, y_pred_rf)
acc_lr = accuracy_score(y_test, y_pred_lr)
acc_svm = accuracy_score(y_test, y_pred_svm)

# Evaluate ensemble methods
acc_max_voting = accuracy_score(y_test, y_pred_max_voting)
acc_averaging = accuracy_score(y_test, y_pred_averaging)
acc_weighted_average = accuracy_score(y_test, y_pred_weighted_average)

# Display results
print("Accuracy of Individual Models:")
print("Random Forest:", acc_rf)
print("Logistic Regression:", acc_lr)
print("Support Vector Machine:", acc_svm)

print("\nAccuracy of Ensemble Techniques:")
print("Max Voting (Hard Voting):", acc_max_voting)
print("Averaging:", acc_averaging)
```

```
print("Weighted Average:", acc_weighted_average)

```

```
Accuracy of Individual Models:
Random Forest: 1.0
Logistic Regression: 1.0
Support Vector Machine: 1.0

Accuracy of Ensemble Techniques:
Max Voting (Hard Voting): 1.0
Averaging: 1.0
Weighted Average: 1.0

C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\linear_model\_logist
ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Python code for advanced ensemble technique-stacking, blending,bagging and boosting

In [45]:
```
pip install scikit-learn

```

```
Requirement already satisfied: scikit-learn in c:\users\anusha v\anaconda3
\lib\site-packages (1.0.2)Note: you may need to restart the kernel to use
updated packages.

Requirement already satisfied: joblib>=0.11 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\anusha v\a
naconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: scipy>=1.1.0 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-learn) (1.9.1)
Requirement already satisfied: numpy>=1.14.6 in c:\users\anusha v\anaconda
3\lib\site-packages (from scikit-learn) (1.24.4)

WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\anusha v\anaconda3
\lib\site-packages)
```

In [26]:

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import StackingClassifier, VotingClassifier, Bagg
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load or generate your dataset
# For simplicity, let's use the famous Iris dataset
from sklearn.datasets import load_iris
data = load_iris()
X, y = data.data, data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Base models
knn_model = KNeighborsClassifier(n_neighbors=3)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
svm_model = SVC(kernel='linear', probability=True)

# Stacking
stacking_model = StackingClassifier(
    estimators=[('knn', knn_model), ('rf', rf_model), ('svm', svm_model
    final_estimator=LogisticRegression()
)

# Blending (Voting Classifier)
blending_model = VotingClassifier(
    estimators=[('knn', knn_model), ('rf', rf_model), ('svm', svm_model
    voting='soft'
)

# Bagging
bagging_model = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    n_estimators=50,
    random_state=42
)

# Boosting
boosting_model = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=50,
    random_state=42
)

# Train models
stacking_model.fit(X_train, y_train)
blending_model.fit(X_train, y_train)
bagging_model.fit(X_train, y_train)
boosting_model.fit(X_train, y_train)

# Predictions
stacking_pred = stacking_model.predict(X_test)
blending_pred = blending_model.predict(X_test)
bagging_pred = bagging_model.predict(X_test)
boosting_pred = boosting_model.predict(X_test)
```

```
62
63 # Evaluate models
64 stacking_accuracy = accuracy_score(y_test, stacking_pred)
65 blending_accuracy = accuracy_score(y_test, blending_pred)
66 bagging_accuracy = accuracy_score(y_test, bagging_pred)
67 boosting_accuracy = accuracy_score(y_test, boosting_pred)
68
69 # Print accuracies
70 print(f"Stacking Accuracy: {stacking_accuracy}")
71 print(f"Blending Accuracy: {blending_accuracy}")
72 print(f"Bagging Accuracy: {bagging_accuracy}")
73 print(f"Boosting Accuracy: {boosting_accuracy}")
74
```

```
Stacking Accuracy: 1.0
Blending Accuracy: 1.0
Bagging Accuracy: 1.0
Boosting Accuracy: 1.0
```

Python code for random forest on breast cancer

In [47]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, conf

# Load the Breast Cancer dataset
breast_cancer = load_breast_cancer()
X, y = breast_cancer.data, breast_cancer.target

# Convert the data to a DataFrame for better visualization (optional)
feature_names = breast_cancer.feature_names
df = pd.DataFrame(data=np.c_[X, y], columns=np.append(feature_names, 't

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=4

# Train the model on the training set
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display the results
print("Random Forest Classifier Evaluation:")
print("Accuracy:", accuracy)
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

```

```
Random Forest Classifier Evaluation:
Accuracy: 0.9649122807017544

Confusion Matrix:
[[40  3]
 [ 1 70]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Evaluation matrices on inertia and dunn index

Inertia (Within-Cluster Sum of Squares):

In [24]:
```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate sample data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Fit a KMeans model
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)

# Get the inertia
inertia = kmeans.inertia_
print(f"Inertia: {inertia}")

```

```
Inertia: 564.9141808210252
```

Dunn Index:

In [25]:

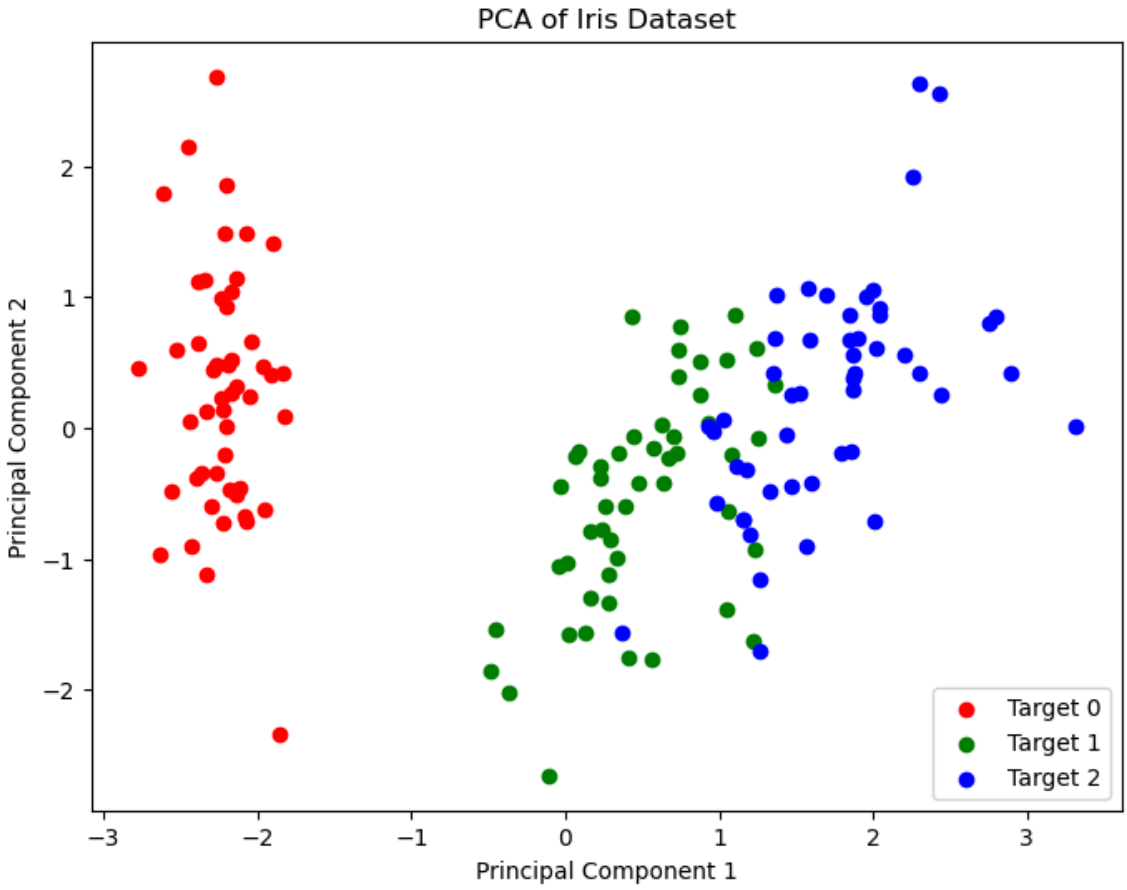
```
from sklearn.metrics import pairwise_distances
from sklearn.metrics import pairwise_distances_argmin_min
from sklearn.cluster import KMeans

def dunn_index(X, labels):
    # Compute pairwise distances between points
    dists = pairwise_distances(X)

    # Compute minimum distance between points of different clusters
    min_inter_cluster_distance = float('inf')
    for i in range(len(labels)):
        for j in range(i + 1, len(labels)):
            if labels[i] != labels[j]:
                min_inter_cluster_distance = min(min_inter_cluster_dist

    # Compute maximum intra-cluster distance
    cluster_distances = []
    for label in set(labels):
        cluster_points = X[labels == label]
        cluster_dists = pairwise_distances(cluster_points)
        cluster_distances.append(cluster_dists.max())

    # Compute Dunn Index
    dunn_index = min_inter_cluster_distance / max(cluster_distances)
    return dunn_index

# Fit a KMeans model
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)

# Get the Dunn Index
labels = kmeans.labels_
dunn_index_value = dunn_index(X, labels)
print(f"Dunn Index: {dunn_index_value}")

```

```
Dunn Index: 0.22935905312943683
```

Python code for dimensionality reduction using PCA

In [1]:

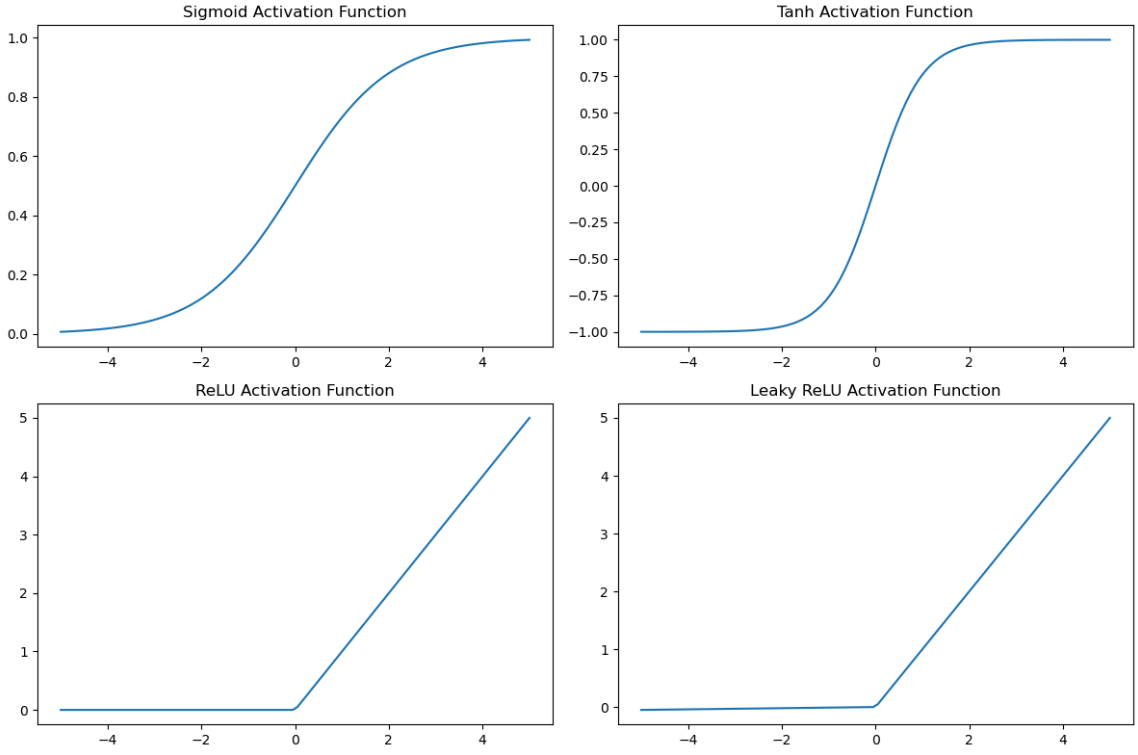
```
 import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Standardize the features (mean=0 and variance=1)
X_std = StandardScaler().fit_transform(X)

# Apply PCA with 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_std)

# Create a DataFrame for visualization
df_pca = pd.DataFrame(data=X_pca, columns=['Principal Component 1', 'Pr
df_pca['Target'] = y

# Visualize the PCA result
plt.figure(figsize=(8, 6))
targets = np.unique(y)
colors = ['r', 'g', 'b']

for target, color in zip(targets, colors):
    indices_to_keep = df_pca['Target'] == target
    plt.scatter(df_pca.loc[indices_to_keep, 'Principal Component 1'],
                df_pca.loc[indices_to_keep, 'Principal Component 2'],
                c=color, label=f'Target {target}')

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.show()

```



Python code for activation function using sigmoid,Tanh,Relu,leaky relu

In [2]:

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

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def tanh(x):
    return np.tanh(x)

def relu(x):
    return np.maximum(0, x)

def leaky_relu(x, alpha=0.01):
    return np.where(x > 0, x, alpha * x)

# Generate input values
x_values = np.linspace(-5, 5, 100)

# Apply activation functions
sigmoid_values = sigmoid(x_values)
tanh_values = tanh(x_values)
relu_values = relu(x_values)
leaky_relu_values = leaky_relu(x_values)

# Plot the activation functions
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(x_values, sigmoid_values, label='Sigmoid')
plt.title('Sigmoid Activation Function')

plt.subplot(2, 2, 2)
plt.plot(x_values, tanh_values, label='Tanh')
plt.title('Tanh Activation Function')

plt.subplot(2, 2, 3)
plt.plot(x_values, relu_values, label='ReLU')
plt.title('ReLU Activation Function')

plt.subplot(2, 2, 4)
plt.plot(x_values, leaky_relu_values, label='Leaky ReLU')
plt.title('Leaky ReLU Activation Function')

plt.tight_layout()
plt.show()

```



Python code for any data set dealing with missing values, category values, labelled encoding,one -hot coding

In [3]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Load the Titanic dataset (a simplified version)
titanic_data = {
    'Name': ['John', 'Jane', 'Bob', 'Alice', 'Charlie'],
    'Age': [22, 30, None, 25, 32],
    'Sex': ['male', 'female', 'male', 'female', 'male'],
    'Survived': [1, 1, 0, 1, 0]
}

df = pd.DataFrame(titanic_data)

# Display the original dataset
print("Original Dataset:")
print(df)
print("\n")

# Dealing with missing values (replace with mean or median in a real sc
df['Age'].fillna(df['Age'].median(), inplace=True)

# Dealing with categorical values using Label Encoding
label_encoder = LabelEncoder()
df['Sex_LabelEncoded'] = label_encoder.fit_transform(df['Sex'])

# Dealing with categorical values using One-Hot Encoding
one_hot_encoder = OneHotEncoder(drop='first', sparse=False)
sex_one_hot_encoded = one_hot_encoder.fit_transform(df[['Sex']])
df_encoded = pd.concat([df, pd.DataFrame(sex_one_hot_encoded, columns=[

# Display the preprocessed dataset
print("Preprocessed Dataset:")
print(df_encoded)

```

```
Original Dataset:
      Name   Age     Sex  Survived
0     John  22.0    male         1
1     Jane  30.0  female         1
2      Bob   NaN    male         0
3    Alice  25.0  female         1
4  Charlie  32.0    male         0


Preprocessed Dataset:
      Name   Age     Sex  Survived  Sex_LabelEncoded  Sex_Female
0     John  22.0    male         1                 1         1.0
1     Jane  30.0  female         1                 0         0.0
2      Bob  27.5    male         0                 1         1.0
3    Alice  25.0  female         1                 0         0.0
4  Charlie  32.0    male         0                 1         1.0
```

Python code for n-grams

In [7]:
```
import nltk
from nltk import ngrams

# Sample text
text = "This is a sample sentence for generating n-grams."

# Tokenize the text into words
words = nltk.word_tokenize(text)

# Function to generate n-grams
def generate_ngrams(input_list, n):
    return list(ngrams(input_list, n))

# Generate n-grams
ngram_size = 2# Change this value to generate different n-grams (e.g.,
result_ngrams = generate_ngrams(words, ngram_size)

# Display the generated n-grams
print(f"{ngram_size}-grams:")
for ngram in result_ngrams:
    print(ngram)

```

```
2-grams:
('This', 'is')
('is', 'a')
('a', 'sample')
('sample', 'sentence')
('sentence', 'for')
('for', 'generating')
('generating', 'n-grams')
('n-grams', '.')
```

sentiment analysis on Spacey,rensflow and keras

In [10]:
```
import spacy

# Load spaCy model
nlp = spacy.load("en_core_web_sm")

# Sample text
text = "I love using spaCy for NLP tasks."

# Analyze sentiment
doc = nlp(text)
sentiment = doc.sentiment

# Display sentiment score
print("Sentiment Score:", sentiment)

```

```
Sentiment Score: 0.0
```

In [13]:
```
pip install spacy tensorflow keras

```

```
Requirement already satisfied: spacy in c:\users\anusha v\anaconda3\lib
\site-packages (3.5.3)Note: you may need to restart the kernel to use u
pdated packages.

Requirement already satisfied: tensorflow in c:\users\anusha v\anaconda
3\lib\site-packages (2.14.0)
Requirement already satisfied: keras in c:\users\anusha v\anaconda3\lib
\site-packages (2.14.0)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in c:\users\anus
ha v\anaconda3\lib\site-packages (from spacy) (2.0.7)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in c:\users\anusha
v\anaconda3\lib\site-packages (from spacy) (0.9.1)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\anusha
v\anaconda3\lib\site-packages (from spacy) (3.0.6)
Requirement already satisfied: numpy>=1.15.0 in c:\users\anusha v\anaco
nda3\lib\site-packages (from spacy) (1.24.4)
Requirement already satisfied: thinc<8.2.0,>=8.1.8 in c:\users\anusha v
\anaconda3\lib\site-packages (from spacy) (8.1.10)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in c:\users
```

In [ ]:
```

```

In [22]:

```
import spacy
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load SpaCy model
nlp = spacy.load("en_core_web_sm")

# Sample text for sentiment analysis
text = "ChatGPT is an amazing language model. I love using it!"

# SpaCy for sentiment analysis
doc = nlp(text)
sentiment_score = doc.sentiment

print(f"SpaCy Sentiment Score: {sentiment_score}")

# Define a simple LSTM model for sentiment analysis
model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=16, input_length=100))
model.add(LSTM(64))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['a

# Tokenize and pad the text
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
tokenizer.fit_on_texts([text])
sequences = tokenizer.texts_to_sequences([text])
padded_sequences = pad_sequences(sequences, maxlen=100, truncating='pos

# Make a prediction with the model
prediction = model.predict(padded_sequences)

print(f"Keras/TensorFlow Sentiment Prediction: {prediction[0][0]}")


```

```
SpaCy Sentiment Score: 0.0
1/1 [==============================] - 1s 617ms/step
Keras/TensorFlow Sentiment Prediction: 0.500891923904419
```

In [ ]:

```
1
```