BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

Laboratory Assignment Submission

Session - 2024 - 2025

Name of the Department:-

Programme Name: -

Semester / Year:-

Course Code: -

Course Name: -

Name of the Student:-

Roll No :-

Registration No :-

Student Code : -

INDEX

	number of measures taken from a photograph.]					
11	**Assignment-11** [Solve the Regression Problem using Swedish Auto Insurance Dataset involves predicting the total payment for all claims in thousands of Swedish Kronor, given the total number of clamis.]	59-61	28/04/2025			
12	**Assignment-12** [A multi-class classification problem, but can also be framed as a regression on Abalone Dataset involves predicting the age of abalone given objective measures of individuals.]	62-66	01/05/2025			
13	**Assignment-13** [A binary (2-class) classification problem Ionosphere Dataset requires the prediction of structure in the atmosphere given radar returns targeting free electrons in the ionosphere]	67-69	121/05/2025			
14	**Assignment-14** [A binary (2-class) classification problem , Wheat Seeds Dataset prediction species given measurements of seeds from different varieties of wheat.]	70-72	15/05/2025			
15	**Assignment-15** [Solve the regression problem on Boston House Price Dataset involves the prediction of a house price in thousands of dollars given details of the house and its neighborhood]	73	19/05/2025			

Assignment-1

[Pandas]

Task 1:

1. Create a Data Frame with 3 columns (**Product, Price, Quantity**) and 5 rows.

Code-

```
df = pd.DataFrame({
   'product' : ['Tab','TV','Gamepad','Mobile','Laptop'],
   'price' : [20000,30000,2000,25000,50000],
   'quantity' : [100,20,10,50,30]
   })

print(df)
```

Output-

	product	price	quantity
0	Tab	20000	100
1	TV	30000	20
2	Gamepad	2000	10
3	Mobile	25000	50
4	Laptop	50000	30

Task 2:

1. Load a dataset.

Code-

```
import pandas as pd

data1=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/pandas.csv")

print(data1.head(10))
```

Output-

	Sl_no	Name	Age	Salary
0	1	A	25	32000
1	2	B	35	36000
2	3	C	45	40000
3	4	D	27	42000
4	5	E	19	48000
5	6	F	35	55000
6	7	G	49	62000
7	8	H	55	68000
8	9	J	52	70000
9	10	K	35	80000

2. Filter employees earning **more than $60,000**.

Code-

```
data1[data1['Salary']> 60000]
```

Output-

	Sl_no	Name	Age	Salary
6	7	G	49	62000
7	8	H	55	68000
8	9	J	52	70000
9	10	K	35	80000

3. Select only the columns **Name & Salary**.

Code-

```
data1[['Name','Salary']]
```

Output-

	Name	Salary
0	A	32000
1	B	36000
2	C	40000
3	D	42000
4	E	48000
5	F	55000
6	G	62000
7	H	68000
8	J	70000
9	K	80000

Task 3:

1. Create a new column **Tax (15% of Salary)**.

Code-

```
data1['Tax']=data1['Salary']*0.15
print(data1)
```

Output-

	Sl_no	Name	Age	Salary	Tax
0	1	A	25	32000	4800.0
1	2	B	35	36000	5400.0
2	3	C	45	40000	6000.0
3	4	D	27	42000	6300.0
4	5	E	19	48000	7200.0
5	6	F	35	55000	8250.0
6	7	G	49	62000	9300.0
7	8	H	55	68000	10200.0
8	9	J	52	70000	10500.0
9	10	K	35	80000	12000.0

2. Sort employees by **Salary in descending order**.

Code-

```
data1.sort_values(by = 'Salary',ascending=False, inplace=True)

print(data1)
```

Output-

	Sl_no	Name	Age	Salary	Tax
9	10	K	35	80000	12000.0
8	9	J	52	70000	10500.0
7	8	H	55	68000	10200.0
6	7	G	49	62000	9300.0
5	6	F	35	55000	8250.0
4	5	E	19	48000	7200.0
3	4	D	27	42000	6300.0

2	3	C	45	40000	6000.0
1	2	B	35	36000	5400.0
0	1	A	25	32000	4800.0

3. Find the **average Salary by Age**.

Code-

```
data1.groupby('Age')['Salary'].mean()
```

Output-

Age	Salary
19	48000.0
25	32000.0
27	42000.0
35	57000.0
45	40000.0
49	62000.0
52	70000.0
55	68000.0

Task 3:

1. Extract employees who earn more than the **average salary**.

Code-

```
data[data['Salary']>data['Salary'].mean()]
```

Output-

	Sl_no	Bonus	Dept	Name	Age	Salary
5	6	35000	BBA	F	35	55000
6	7	12345	MCA	G	49	62000
7	8	63542	BSC	H	55	68000

8	9	52120	MCA	J	52	70000
9	10	20000	MSC	K	35	80000

2. Group by department and calculate **total salary expenditure.**

Code-

```
data.groupby('Dept')['Salary'].sum()
```

Output-

Dept	Salary
BBA	55000
BCA	74000
BSC	108000
MCA	216000
MSC	80000

3. Merge the data with another dataset containing **bonus information**.

Code-

```
import pandas as pd
data1=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Pandas_T3.csv")
data2=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/pandas.csv")
data=data1.merge(data2, on='Sl_no')
print(data)
```

Output-

	Sl_no	Bonus	Dept	Name	Age	Salary
0	1	45000	BCA	A	25	32000
1	2	56555	MCA	B	35	36000

2	3	65466	BSC	C	45	40000
3	4	22000	BCA	D	27	42000
4	5	42000	MCA	E	19	48000
5	6	35000	BBA	F	35	55000
6	7	12345	MCA	G	49	62000
7	8	63542	BSC	H	55	68000
8	9	52120	MCA	J	52	70000
9	10	20000	MSC	K	35	80000

Assignment-2

[NumPy]

Task-1: create a 1d array and a 2d NumPy array with your own values and print them in google Colab.

Code-

```
import NumPy as np

array_1d = np. array ([1, 2, 3, 4, 5])

print ("1D Array:")

print(array_1d)

array_2d = np. array ([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print ("\n2D Array:")

print(array_2d)
```

Output-

```
1D Array:

[1 2 3 4 5]

2D Array:

[[1 2 3]

 [4 5 6]

 [7 8 9]]
```

Task-2: Perform element wise addition and multiplication on a NumPy array of your choice. Use at least two universal functions and print the results.

Code-

```
import NumPy as np

array = np. array ([2, 4, 6, 8, 10])

added array = np.add (array, 5)

multiplied array = unmultiplied (array, 2)

print ("Original Array:")

print(array)

print ("\nAfter Element-wise Addition (Adding 5):")

print (added array)
```

```
print ("\after Element-wise Multiplication (Multiplying by 2):")
print (multiplied array)
```

Output-

```
Original Array:
[ 2 4 6 8 10]
After Element-wise Addition (Adding 5):
[ 7 9 11 13 15]
After Element-wise Multiplication (Multiplying by 2):
[ 4 8 12 16 20]
```

Task-3: create a 2d array and demonstrate indexing, slicing and Boolean indexing. print the results of each operation.

Code-

```
import NumPy as np
array_2d = np. array ([
    [10, 20, 30, 40],
    [50, 60, 70, 80],
    [90, 100, 110, 120]
])
print ("Original 2D Array:")
print(array_2d)
indexed element = array_2d [1, 2]
print ("\indexed Element (Row 1, Column 2):", indexed element)
sliced array = array_2d [0:2, 1:3]
print ("\sliced Sub-array (Rows 0-1, Columns 1-2):")
print (sliced array)
Boolean condition = array_2d > 50
filtered array = array_2d [Boolean condition]
print ("\elements Greater Than 50:")
print (filtered array)
```

Output-

Original 2D Array:

[[10 20 30 40]

[50 60 70 80]

[90 100 110 120]]

Indexed Element (Row 1, Column 2): 70

Sliced Sub-array (Rows 0-1, Columns 1-2):

[[20 30]

[60 70]]

Elements Greater Than 50:

[60 70 80 90 100 110 120]

Task-4: create two NumPy arrays, concatenate, and then split the resulting array into two parts. print the results

Code-

```
import NumPy as np
array1 = np. array ([1, 2, 3, 4, 5])
array2 = np. array ([6, 7, 8, 9, 10])
concatenated array = np. concatenate ((array1, array2))
print ("Concatenated Array:")
print (concatenated array)
split arrays = np. array split (concatenated array, 2)
print ("\first Split Array:")
print (split arrays [0])
print ("\second Split Array:")
print (split arrays [1])
```

Output-

Concatenated Array:

[1 2 3 4 5 6 7 8 9 10]

First Split Array:

[1 2 3 4 5]

Second Split Array:

[6 7 8 9 10]

Assignment-3

[Matplotlib]

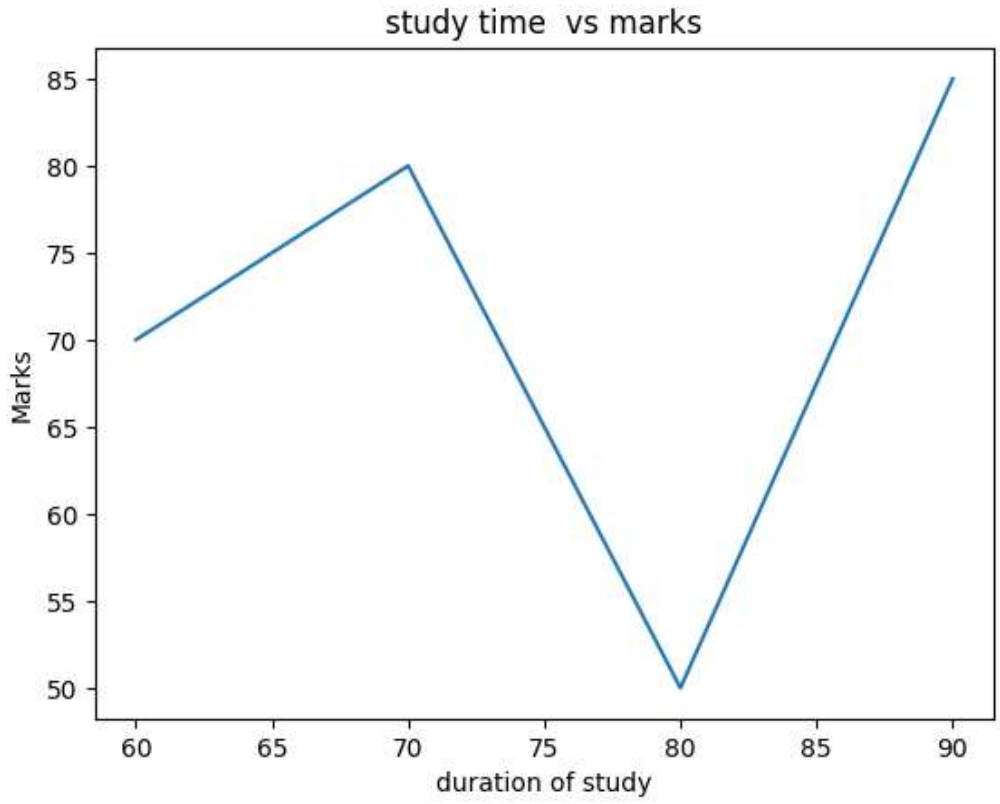
1.1Simple line plot.

Code-

```
import matplotlib.pyplot as plt

x = [60, 70, 80, 90]

y = [70, 80, 50, 85]

plt.plot(x, y)

plt.title("study time  vs marks")

plt.xlabel("duration of study")

plt.ylabel("Marks")
```

Output-

Text(0, 0.5, 'Marks')



1.2Simple line plot.

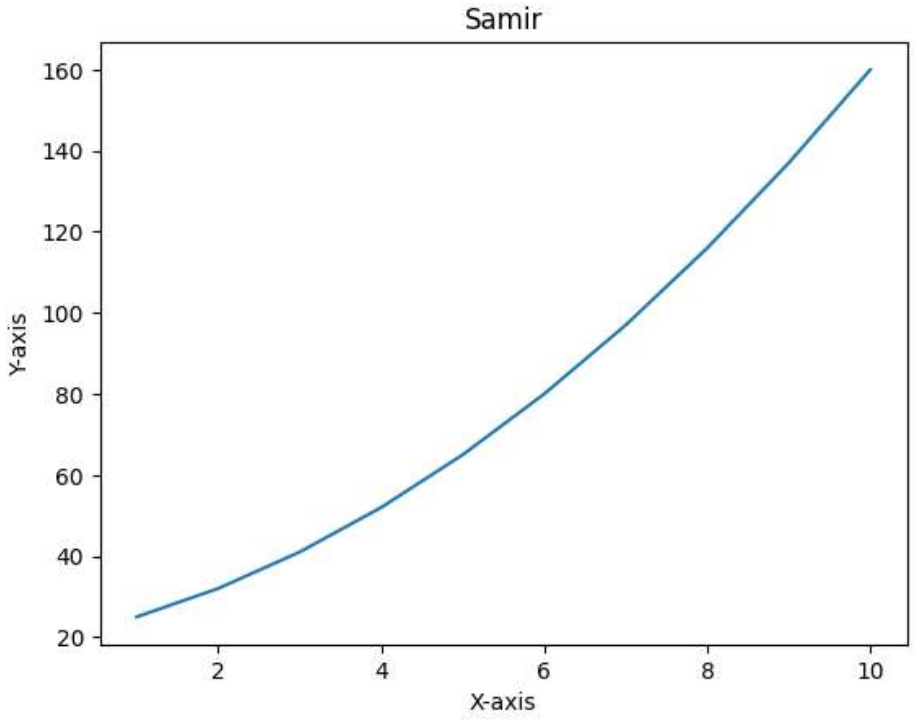
Code-

```
import numpy as np

x = np.arange(1, 11)

y = x**2 + 4*x + 20

plt.plot(x, y)

print(x)

print(y)

plt.title("Samir")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.show()
```

Output-

[1 2 3 4 5 6 7 8 9 10]

[25 32 41 52 65 80 97 116 137 160]

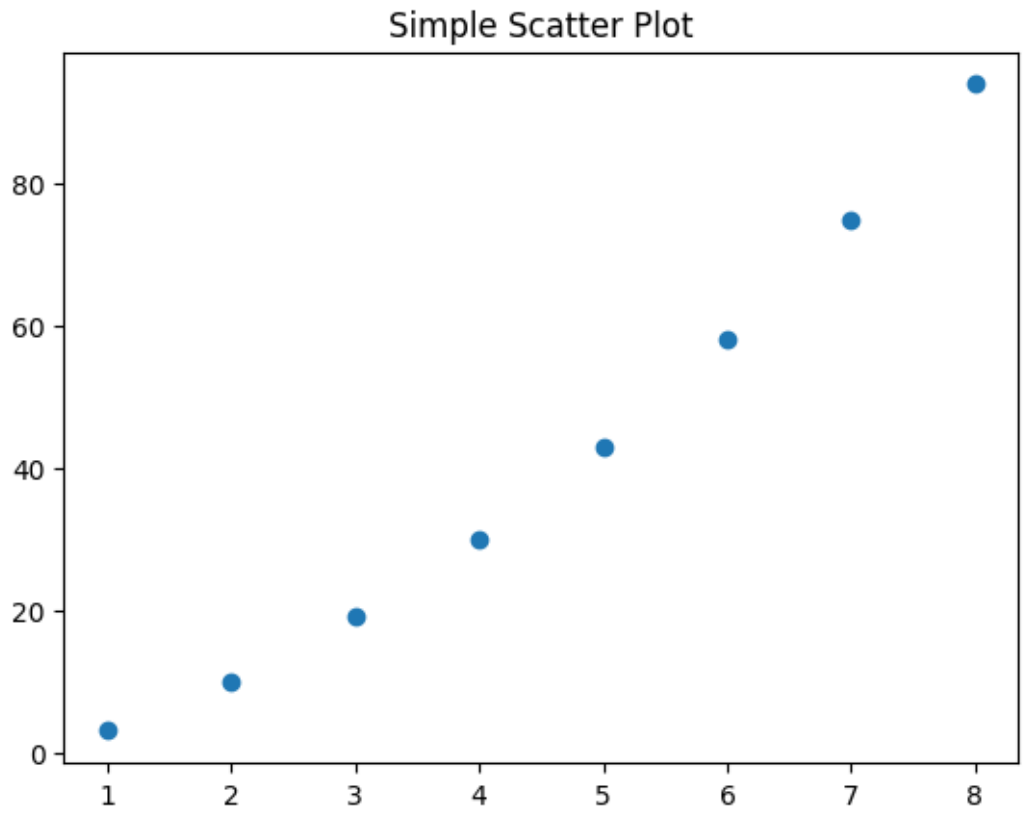


2.Simple Scatter Plot.

Code-

```
plt.scatter(x, y)

plt.title("Simple Scatter Plot")

plt.show()
```

Output-



3. Customized Line Plot.

Code-

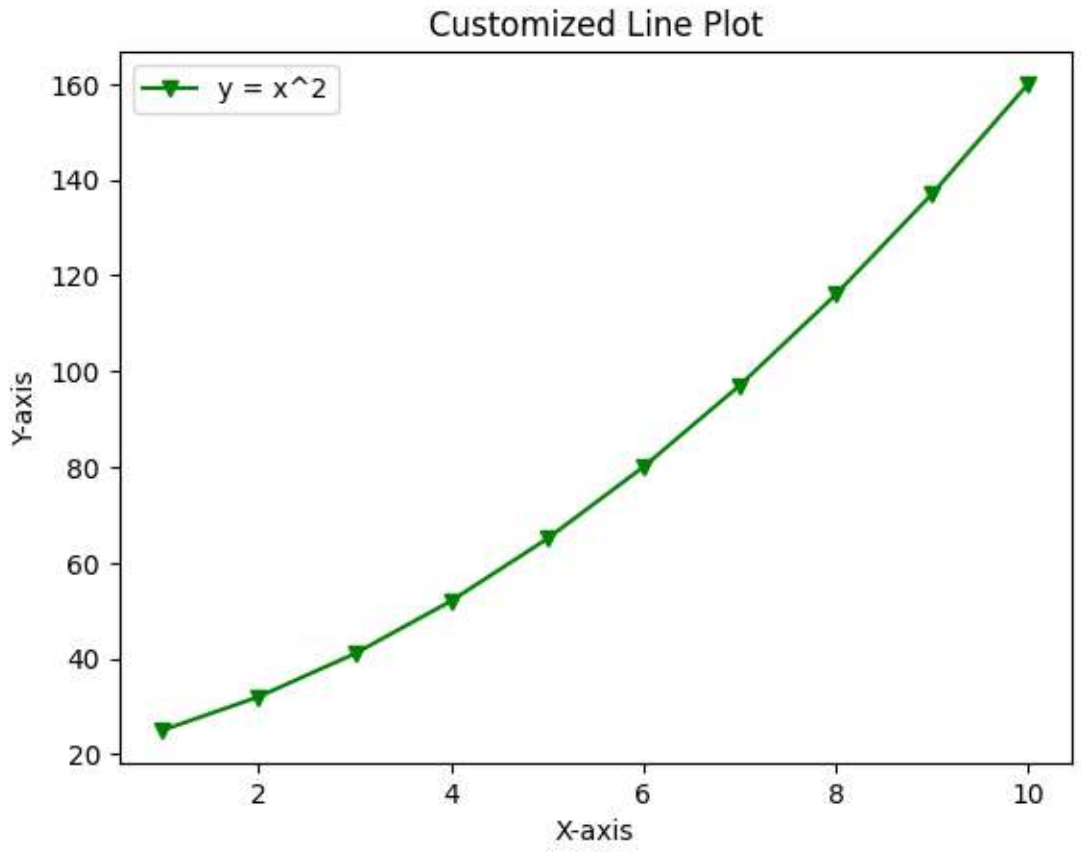
```
plt.plot(x, y, color='green', marker='v', label='y = x^2')

plt.title("Customized Line Plot")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")
```

plt.legend()

plt.grid(False)

plt.show()

Output-



4. Subplot.

Code-

fig, axs = plt.subplots(3, 3)

axs[0, 0].plot(x, y)

axs[0, 1].scatter(x, y)

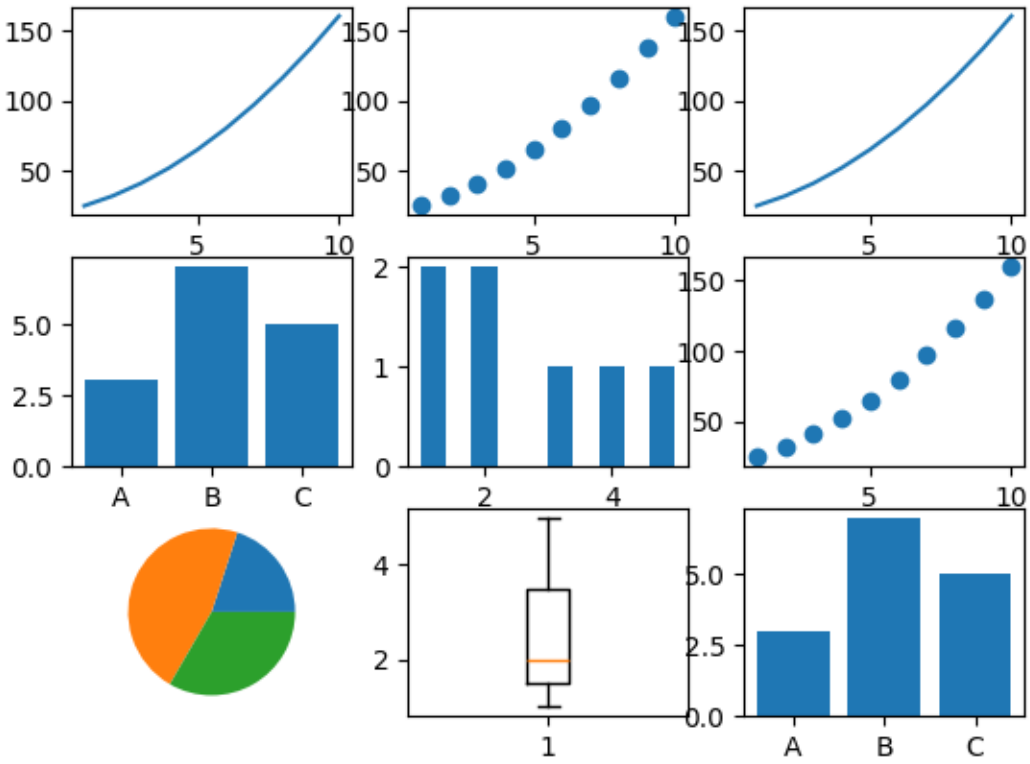
axs[1, 0].bar(['A', 'B', 'C'], [3, 7, 5])

axs[1, 1].hist([1, 2, 1, 2, 3, 4, 5])

axs[2, 0].pie([3, 7, 5])

```
axs[2, 1].boxplot([1, 2, 1, 2, 3, 4, 5])

axs[0, 2].plot(x, y)

axs[1, 2].scatter(x, y)

axs[2, 2].bar(['A', 'B', 'C'], [3, 7, 5])

plt.show()

plt.tight_layout()

plt.show()

plt.savefig("3_3.png", dpi=300, bbox_inches='tight')
```

Output-



<Figure size 640x480 with 0 Axes>

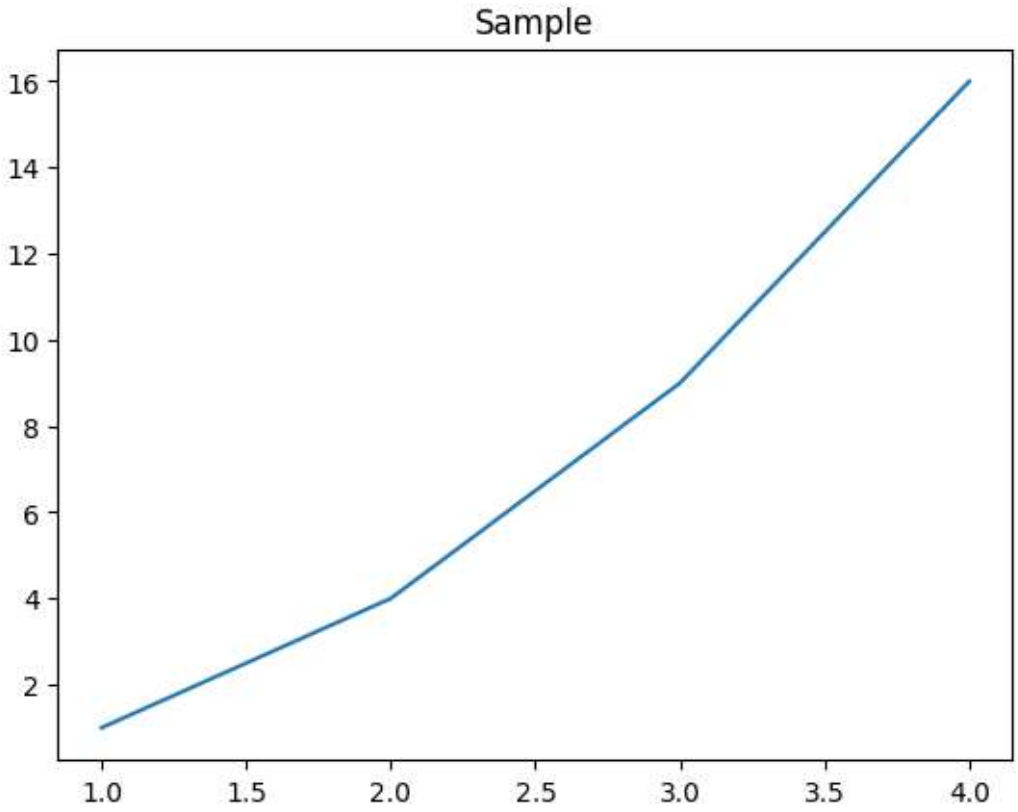
<Figure size 640x480 with 0 Axes>

4.Sample.

Code-

```
plt.plot(x, y)

plt.title("Sample")

plt.savefig("my_plot.png", dpi=300, bbox_inches='tight')
```

Output-

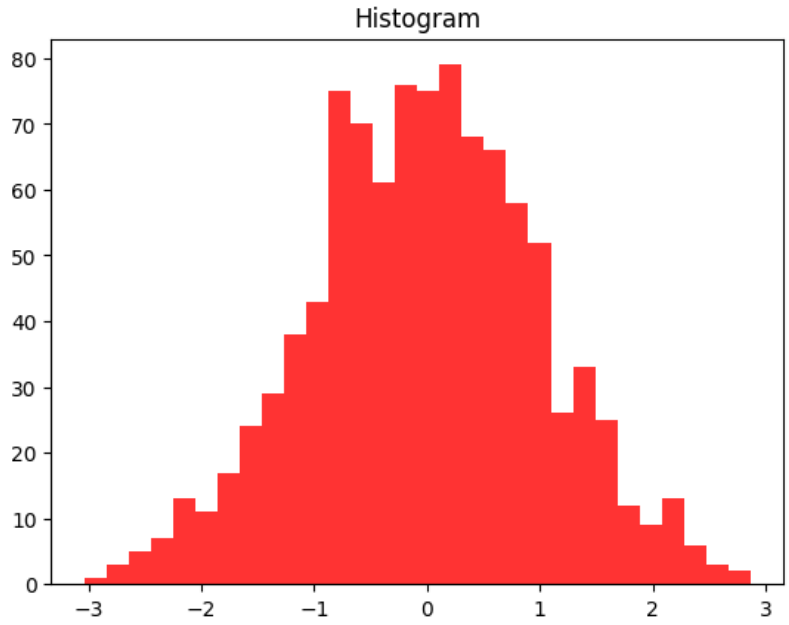


5. Histogram.

Code-

```
import numpy as np

data = np.random.randn(1000)

plt.hist(data, bins=30, alpha=0.8,color='red')

plt.title("Histogram")

plt.show()
```

Output-



6.Box Plot.

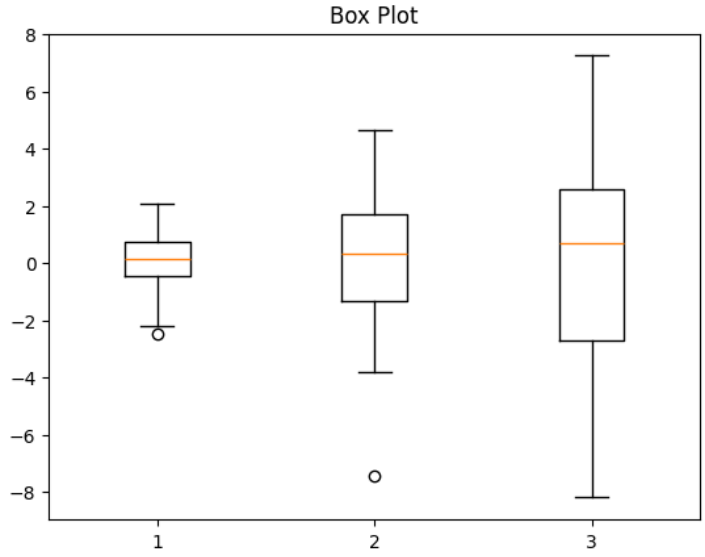
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

plt.boxplot(data)

plt.title("Box Plot")

plt.show()

Output-

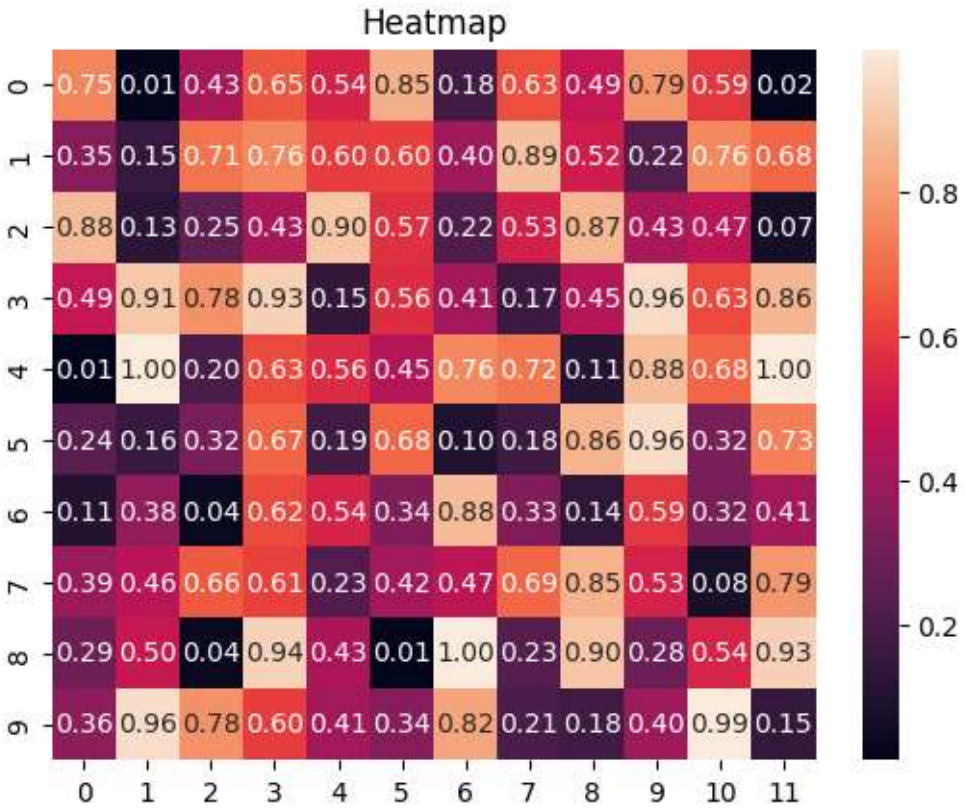


7. Heatmap.

Code-

```
import seaborn as sns

data = np.random.rand(10, 12)

sns. heatmap(data, annot=True, fmt=".2f")

plt.title("Heatmap")

plt.show()
```

Output-

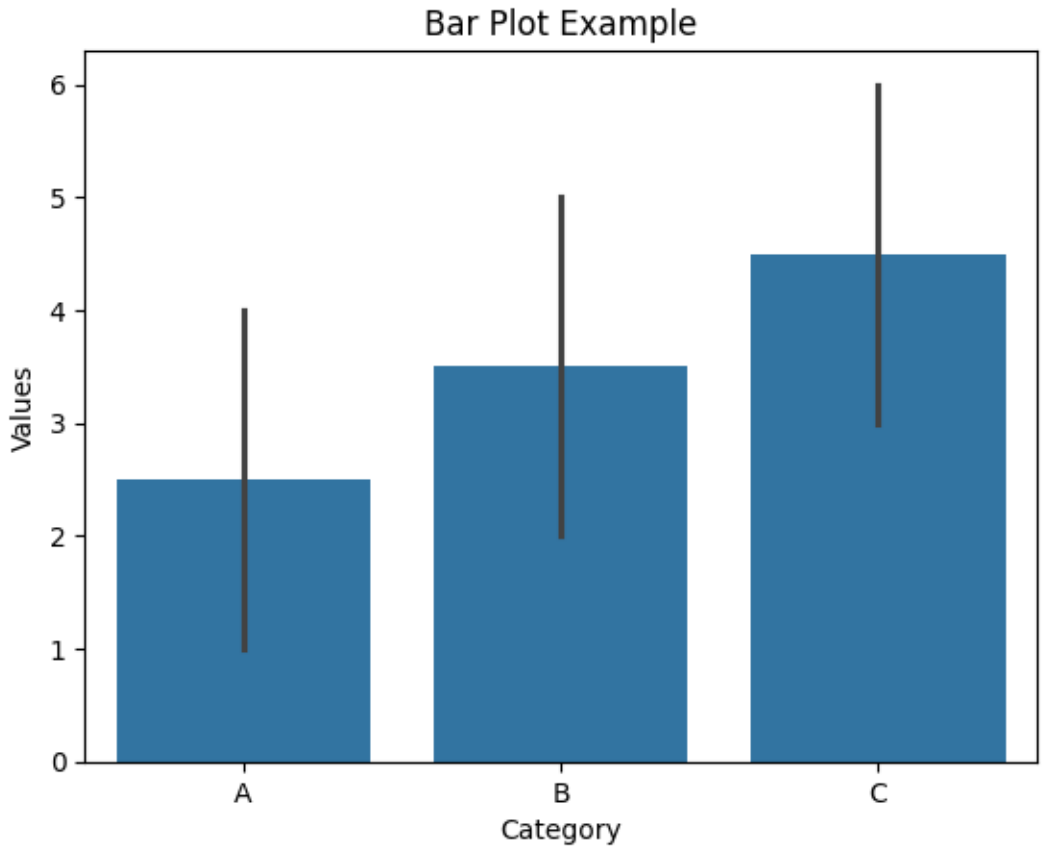


7.1 Bar Plot.

Code-

```
import seaborn as sns

import pandas as pd

data = pd.DataFrame({

    'Category': ['A', 'B', 'C', 'A', 'B', 'C'],

    'Values': [1, 2, 3, 4, 5, 6]

})

sns.barplot(x='Category', y='Values', data=data)

plt.title("Bar Plot Example")

plt.show()
```

Output-

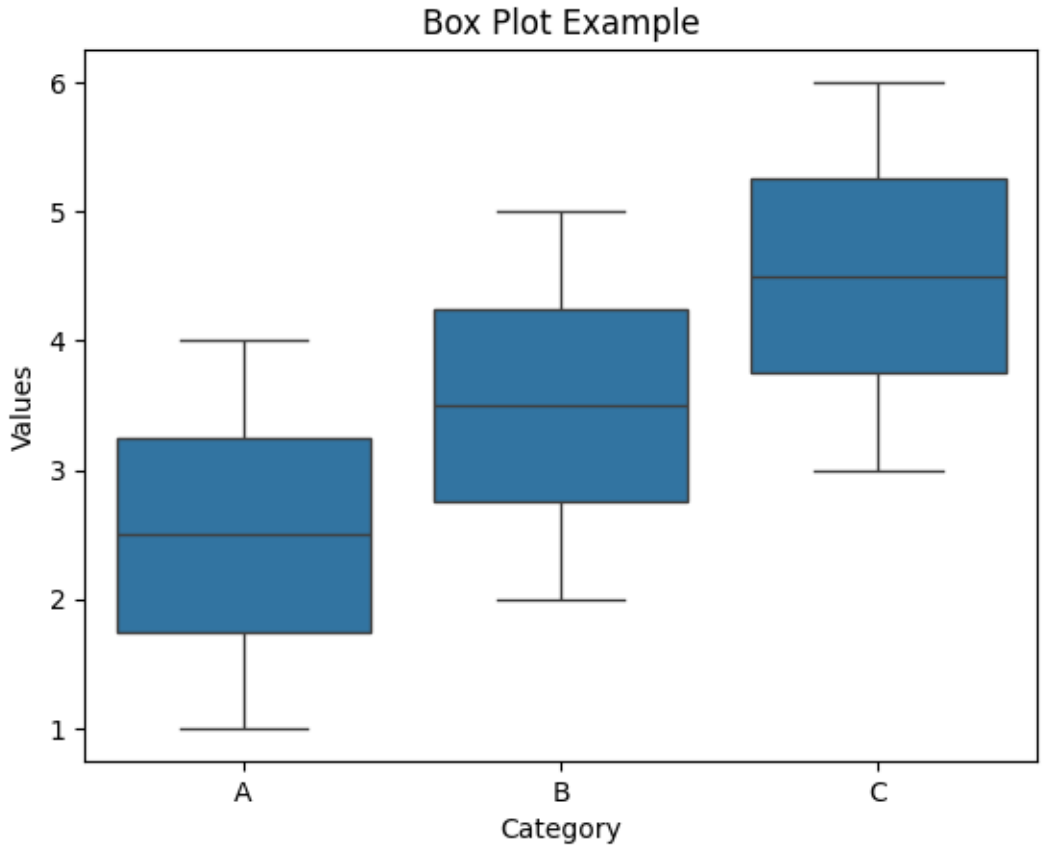


7.2 Box Plot.

Code-

```
sns.boxplot(x='Category', y='Values', data=data)

plt.title("Box Plot Example")

plt.show()
```

Output-

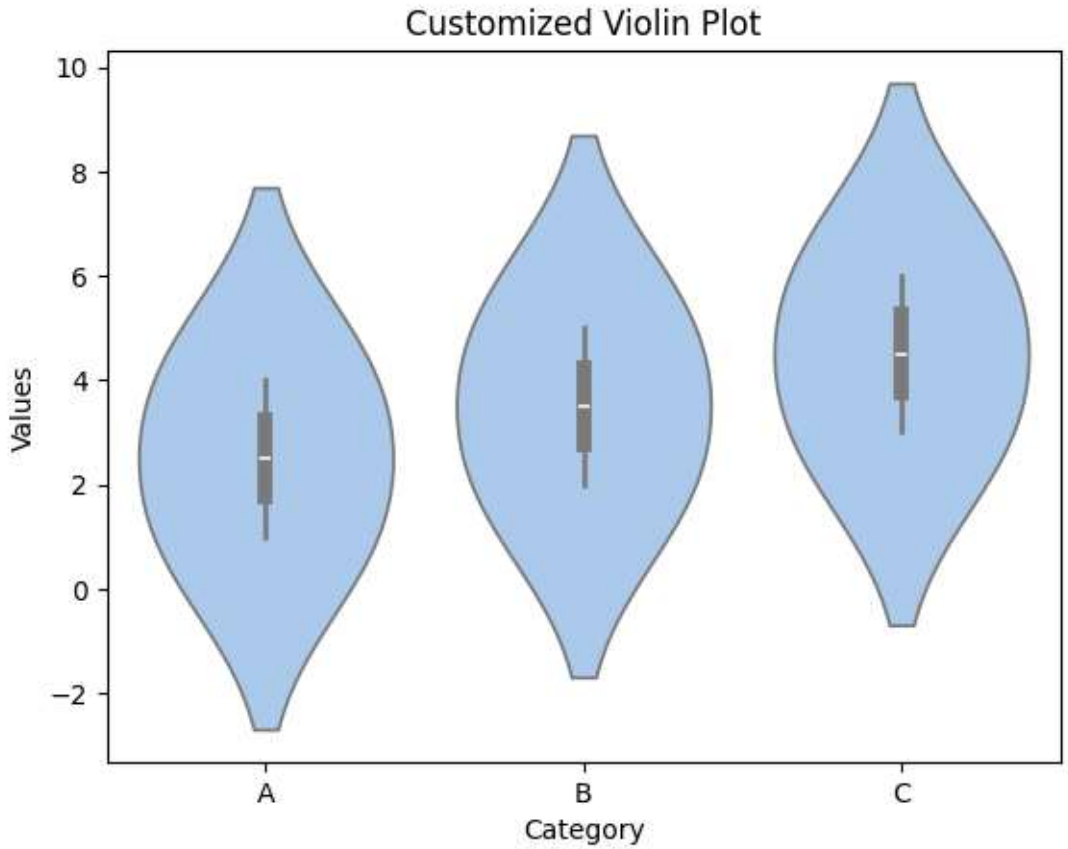


8.Customized Violin Plot.

Code-

```
sns.set_palette("pastel")

sns.violinplot(x='Category', y='Values', data=data)

plt.title("Customized Violin Plot")

plt.show()
```

Output-

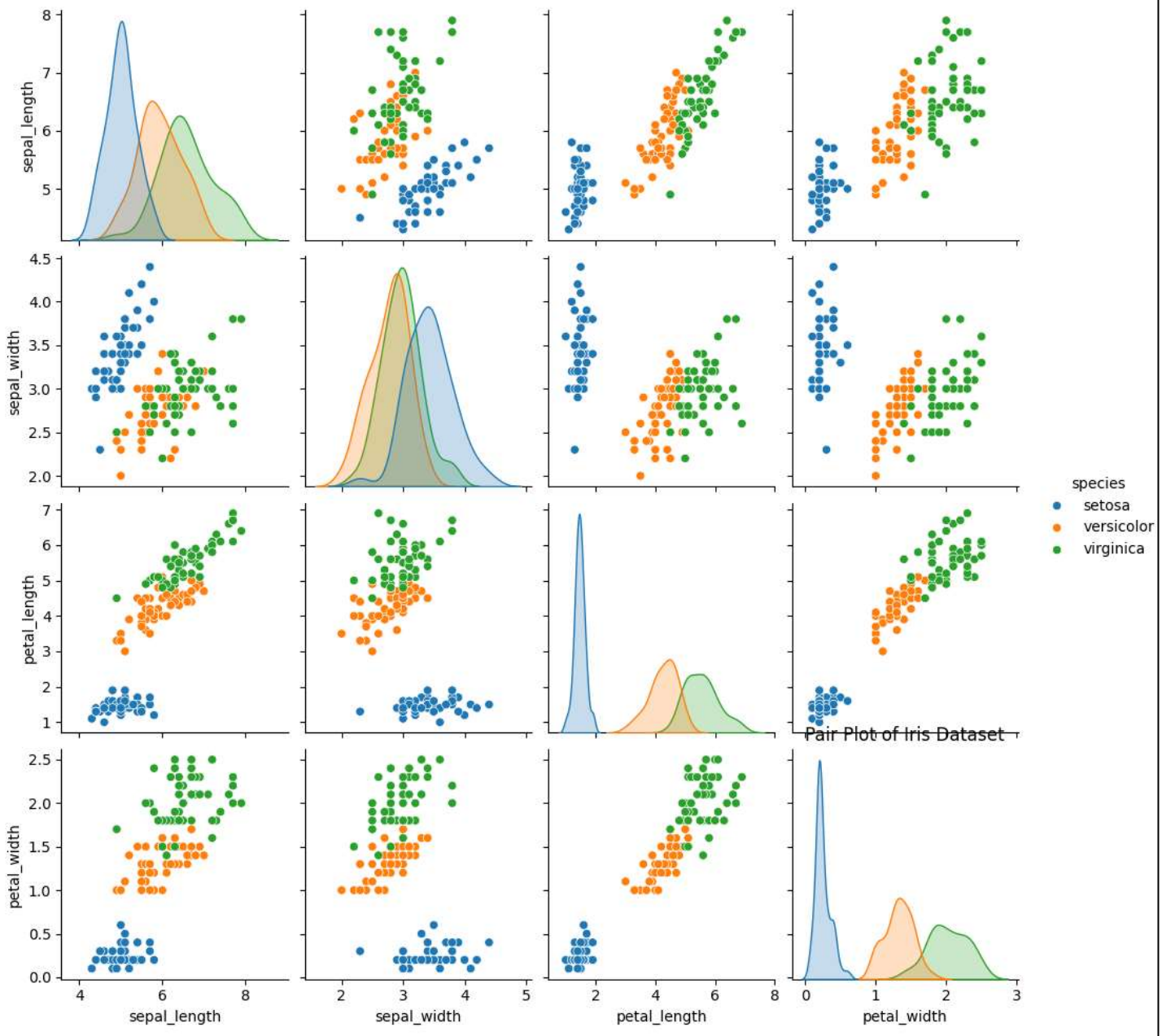


9. Pair Plot of Iris Dataset.

Code-

```
iris = sns.load_dataset("iris")

sns.pairplot(iris, hue='species')

plt.title("Pair Plot of Iris Dataset")

plt.show()
```

Output-



10. 1 Read Iris Dataset.

Code-

iris.head(100)

Output-

	sepal_length	sepal_width	petal_length	petal_width	species
0	**5.1**	**3.5**	**1.4**	**0.2**	**setosa**
1	**4.9**	**3.0**	**1.4**	**0.2**	**setosa**
2	**4.7**	**3.2**	**1.3**	**0.2**	**setosa**
3	**4.6**	**3.1**	**1.5**	**0.2**	**setosa**
4	**5.0**	**3.6**	**1.4**	**0.2**	**setosa**
...	**...**	**...**	**...**	**...**	**...**
95	**5.7**	**3.0**	**4.2**	**1.2**	**versicolor**
96	**5.7**	**2.9**	**4.2**	**1.3**	**versicolor**
97	**6.2**	**2.9**	**4.3**	**1.3**	**versicolor**
98	**5.1**	**2.5**	**3.0**	**1.1**	**versicolor**
99	**5.7**	**2.8**	**4.1**	**1.3**	**versicolor**

100 rows × 5 columns

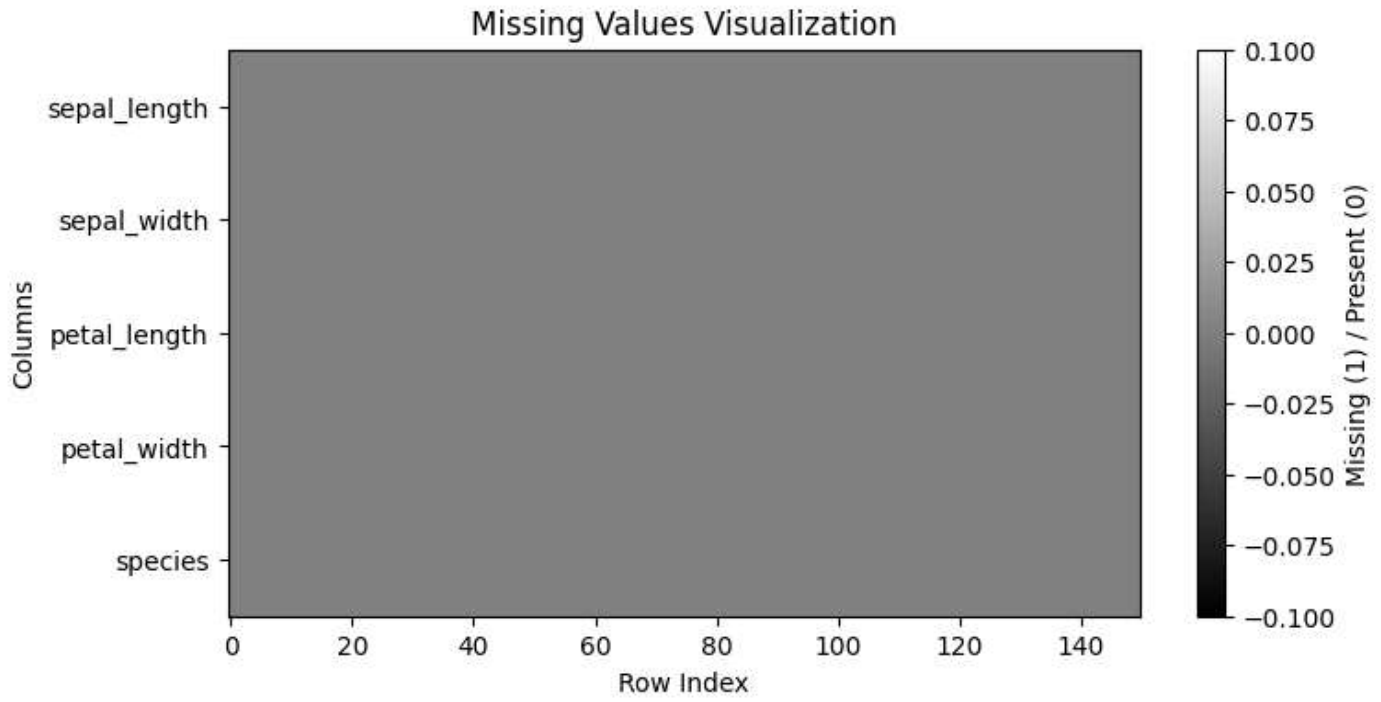
10.2 Missing Values Visualization.

Code-

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

iris = sns.load_dataset("iris")

df = pd.DataFrame(iris)

missing_mask = df.isnull()

plt.figure(figsize=(8, 4))

plt.imshow(missing_mask.T, aspect='auto', cmap='gray', interpolation='none')

plt.title('Missing Values Visualization')

plt.xlabel('Row Index')
```

```
plt.ylabel('Columns')

plt.yticks(ticks=range(df.shape[1]), labels=df.columns)

plt.colorbar(label='Missing (1) / Present (0)')

plt.show()
```

Output-



11.California housing value.

Code-

```
import dask.dataframe as dd

ddf = dd.read_csv('/content/sample_data/california_housing_train.csv')

print(ddf.head())
```

Output-

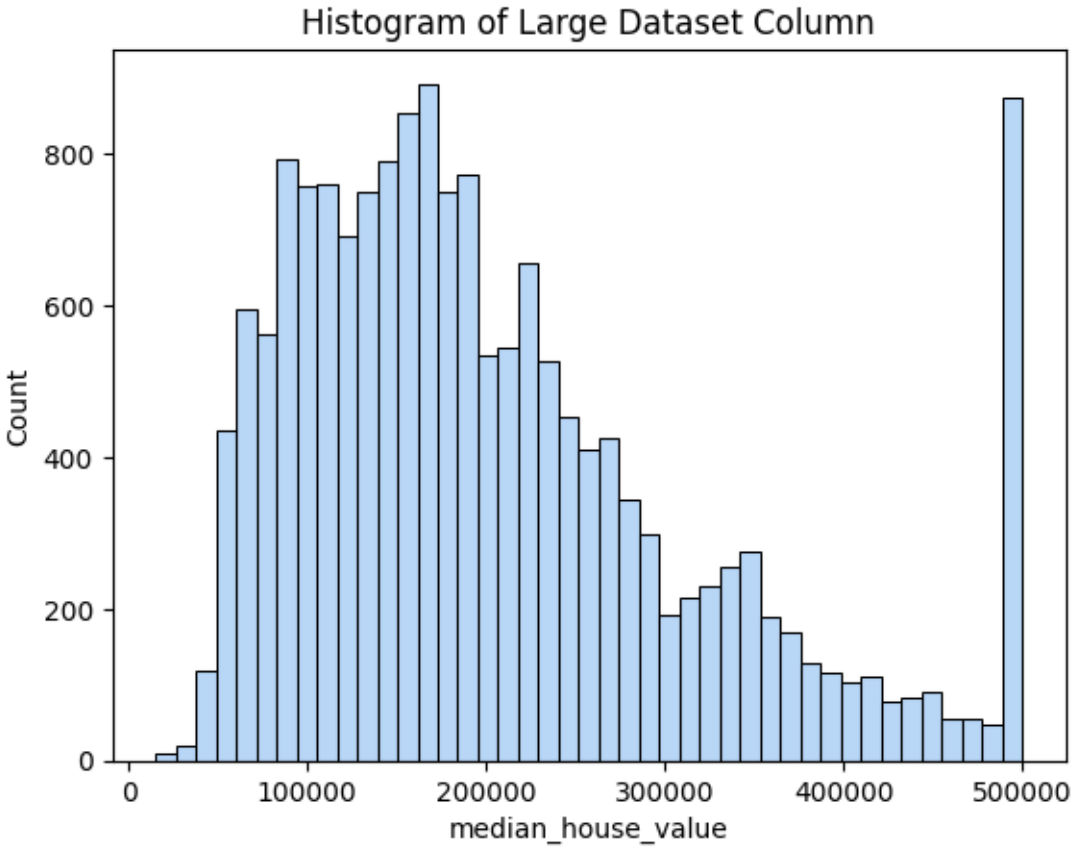
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -114.31     34.19                15.0       5612.0          1283.0
1    -114.47     34.40                19.0       7650.0          1901.0
2    -114.56     33.69                17.0        720.0           174.0
3    -114.57     33.64                14.0       1501.0           337.0
4    -114.57     33.57                20.0       1454.0           326.0

   population  households  median_income  median_house_value
0      1015.0       472.0         1.4936             66900.0
1      1129.0       463.0         1.8200             80100.0  3
2       333.0       117.0         1.6509             85700.0
3       515.0       226.0         3.1917             73400.0
4       624.0       262.0         1.9250             65500.0
```

11.2 Histogram of Large Dataset Column.

Code-

```
ddf['median_house_value'].mean().compute()

sns.histplot(ddf['median_house_value'].compute())

plt.title("Histogram of Large Dataset Column")

plt.show()
```

Output-



Assignment-4

[Sklearn]

1. Read iris flower datasets.

Code-

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
print(iris_df.head())
```

Output-

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

2. Titanic Dataset.

Code-

```
import pandas as pd

from sklearn.model_selection import train_test_split

titanic_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/train.csv')

titanic_df['Age'].fillna(titanic_df['Age'].median(), inplace=True)

titanic_df.drop(columns=['Cabin', 'Ticket'], inplace=True)

X = titanic_df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]

y = titanic_df['Survived']

X = pd.get_dummies(X, columns=['Sex'], drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Output-

```
<ipython-input-3-086bc33daa93>:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

titanic_df['Age'].fillna(titanic_df['Age'].median(), inplace=True)
```

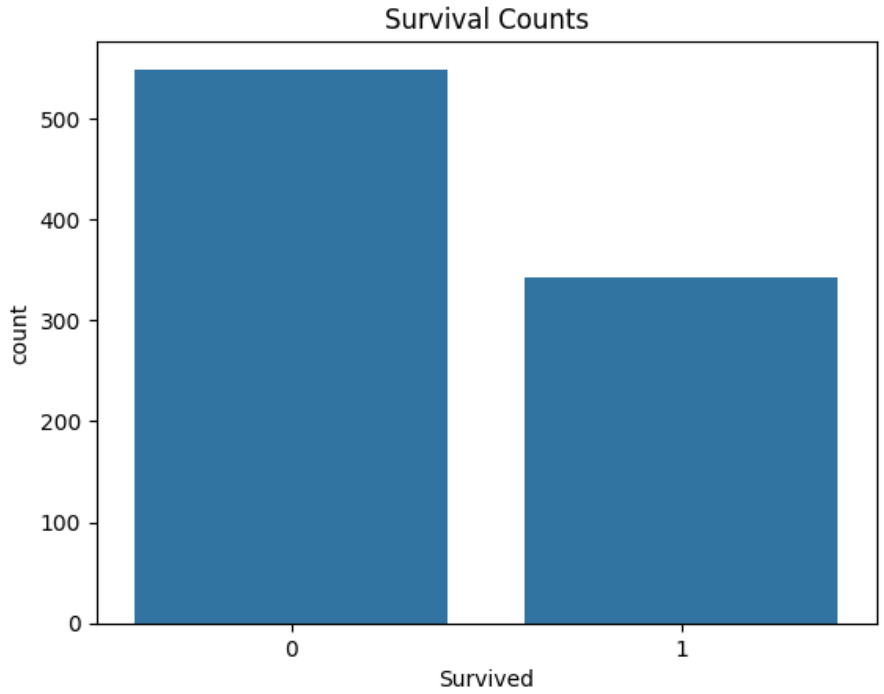
3.Srvival Counts and Age Distribution.

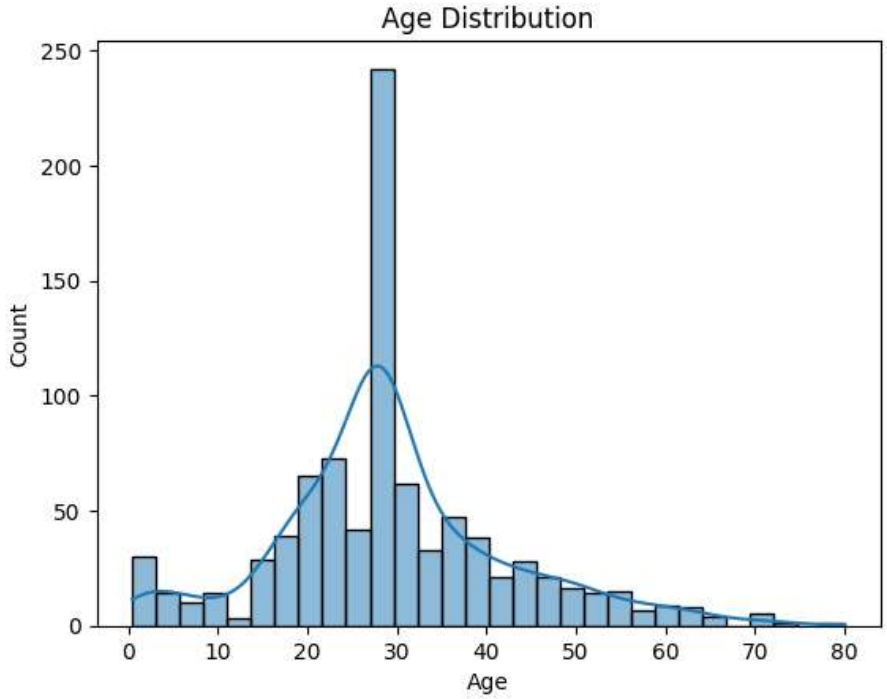
Code-

```
import seaborn as sns

import matplotlib.pyplot as plt

sns.countplot(x='Survived', data=titanic_df)

plt.title('Survival Counts')

plt.show()

sns.histplot(titanic_df['Age'], bins=30, kde=True)

plt.title('Age Distribution')

plt.show()

sns.heatmap(titanic_df.corr(), annot=True, cmap='coolwarm')
```

```
plt.title('Correlation Heatmap')
plt.show()
```

Output-





4. BostonHousing Prediction.

Code-

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
boston = pd.read_csv("/content/drive/MyDrive/BostonHousing.csv")
data=boston.drop(['medv'],axis=1)
target=boston['medv']
print(data)
X = data
y = target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
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'MSE: {mse}, R²: {r2}')
```

Output-

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273
```

	ptratio	b	lstat
0	15.3	396.90	4.98
1	17.8	396.90	9.14
2	17.8	392.83	4.03
3	18.7	394.63	2.94
4	18.7	396.90	5.33
..	...	...	...
501	21.0	391.99	9.67
502	21.0	396.90	9.08
503	21.0	396.90	5.64
504	21.0	393.45	6.48
505	21.0	396.90	7.88

5.Iris Dataset Using LogisticRegression.

Code-

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Output-

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

	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

6. GridSearch Using RandomForestClassifier.

Code-

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rf = RandomForestClassifier()
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
```

Output-

```
Best parameters: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 200}
```

Assignment-5

[Cricket Visualization]

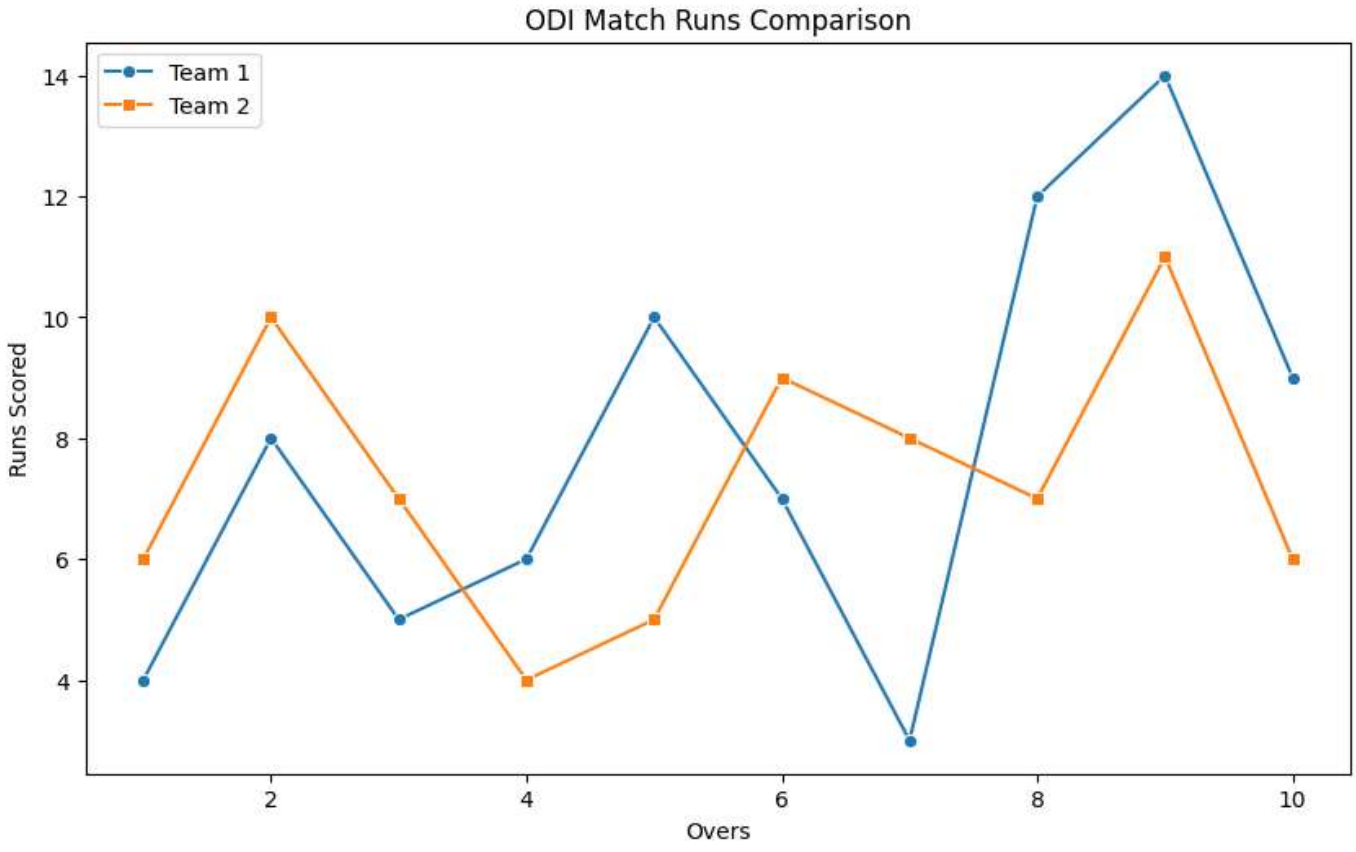
Task 1: ODI Match Comparison Using Matplotlib & Seaborn

1. Plot line charts for runs scored over overs for both teams.

Code-

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/odi_match_data.csv")
print(df.columns)
plt.figure(figsize=(10, 6))
sns.lineplot(x=df["Over"], y=df["Team_A_Runs"], label="Team 1", marker="o")
sns.lineplot(x=df["Over"], y=df["Team_B_Runs"], label="Team 2", marker="s")
plt.xlabel("Overs")
plt.ylabel("Runs Scored")
plt.title("ODI Match Runs Comparison")
plt.show()
```

Output-

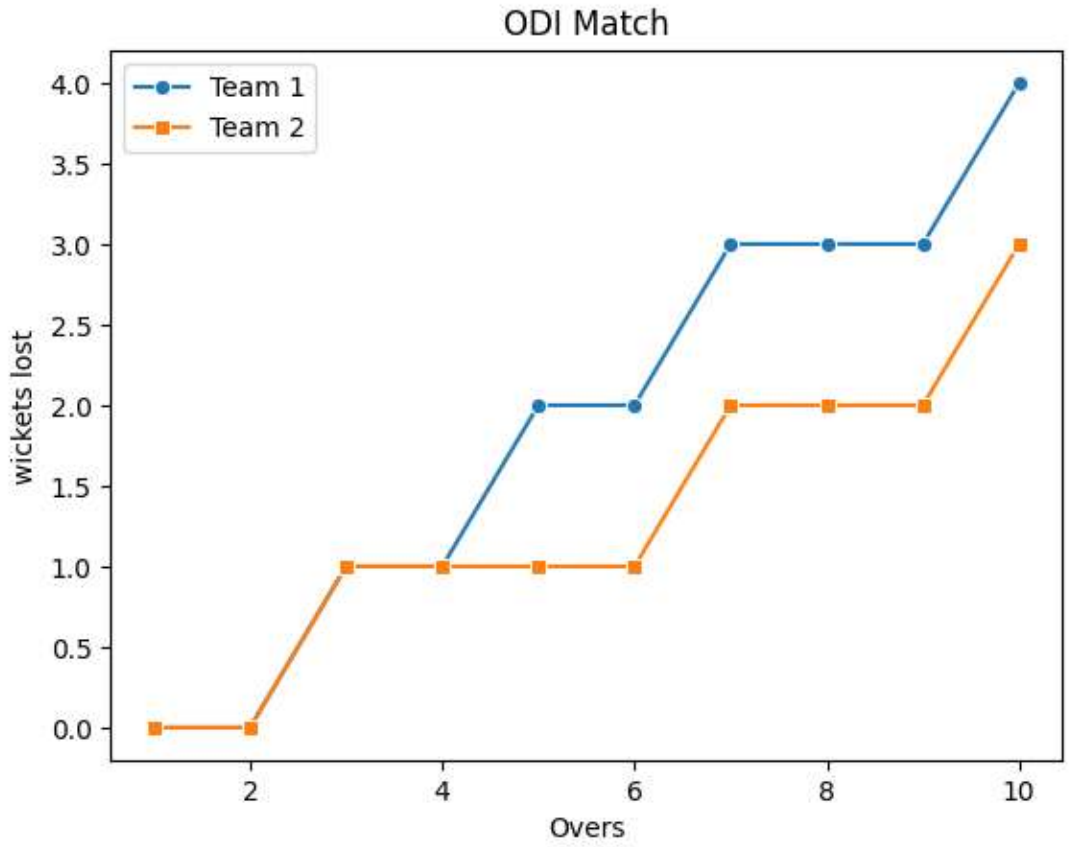


2. Plot line charts for wickets lost over overs.

Code-

```
sns.lineplot(x=df["Over"], y=df["Team_A_Wickets"], label="Team 1", marker="o")
sns.lineplot(x=df["Over"], y=df["Team_B_Wickets"], label="Team 2", marker="s")
plt.xlabel("Overs")
plt.ylabel("wickets lost")
plt.title("ODI Match")
plt.show()
```

Output-

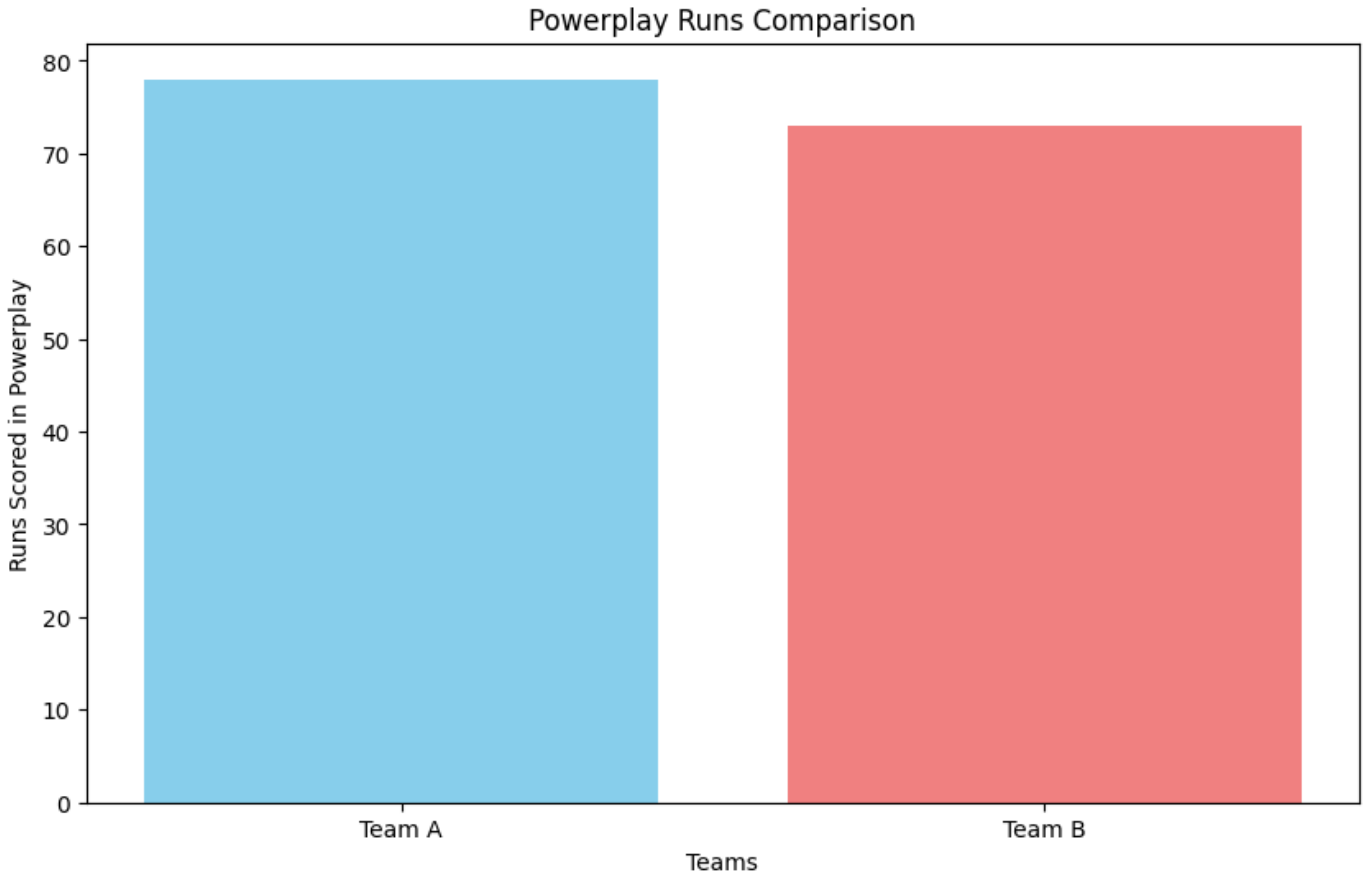


3. Create bar charts for runs scored in powerplays.

Code-

```
plt.figure(figsize=(10, 6))
powerplay_runs_a = df[df['Over'] <= 10]['Team_A_Runs'].sum()
powerplay_runs_b = df[df['Over'] <= 10]['Team_B_Runs'].sum()
plt.bar(['Team A', 'Team B'], [powerplay_runs_a, powerplay_runs_b], color=['skyblue', 'lightcoral'])
plt.xlabel('Teams')
plt.ylabel('Runs Scored in Powerplay')
plt.title('Powerplay Runs Comparison')
plt.show()
```

Output-

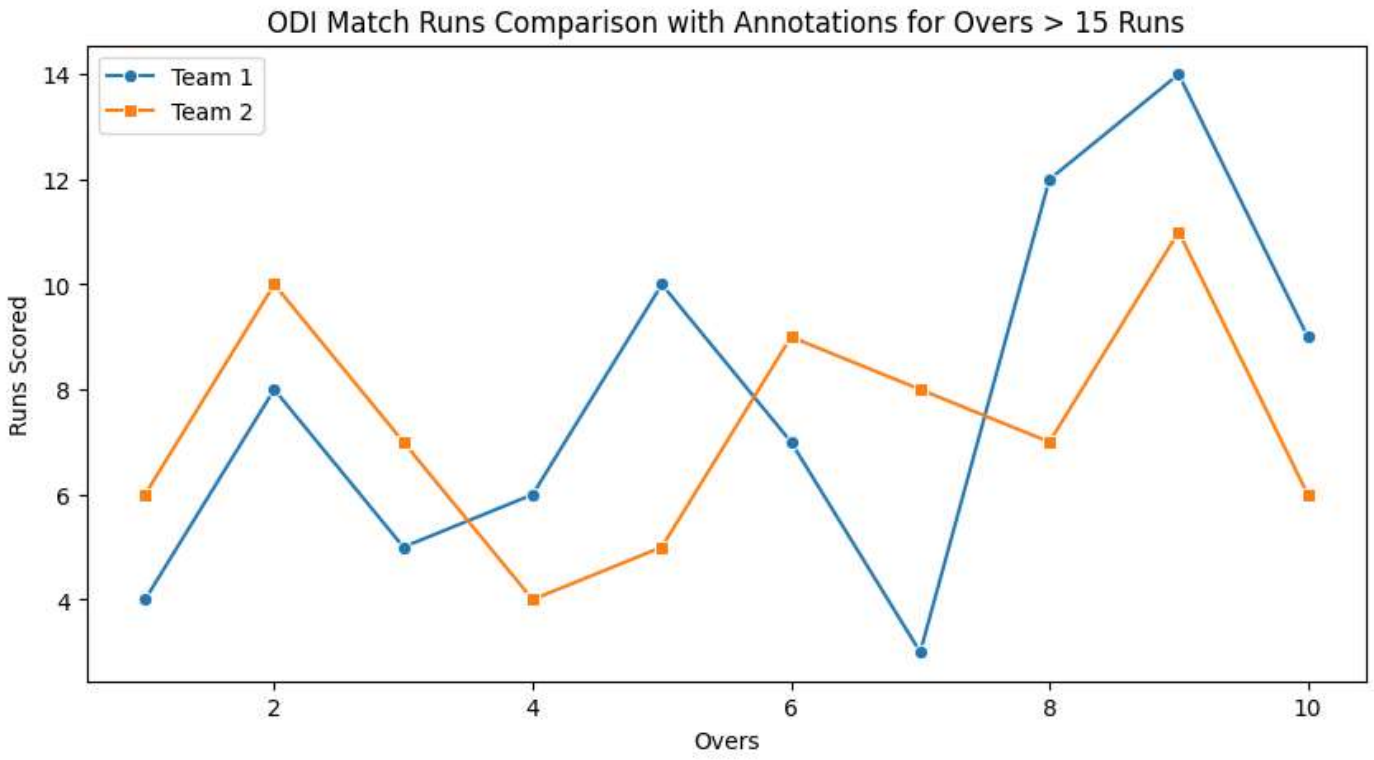


4. Annotate overs where more than 15 runs were scored.

Code-

```
df['Total_Runs'] = df['Team_A_Runs'] + df['Team_B_Runs']
overs_over_15 = df[df['Total_Runs'] > 15]['Over'].tolist()
print("Overs where more than 15 runs were scored:", overs_over_15)
plt.figure(figsize=(10, 5))
sns.lineplot(x=df["Over"], y=df["Team_A_Runs"], label="Team 1", marker="o")
sns.lineplot(x=df["Over"], y=df["Team_B_Runs"], label="Team 2", marker="s")
for over in overs_over_15:
    plt.annotate(f"{over}", xy=(over, df[df['Over'] == over]['Total_Runs'].iloc[0]), xytext=(over + 0.2,
df[df['Over'] == over]['Total_Runs'].iloc[0] + 2),
                 arrowprops=dict(facecolor='black', shrink=0.05))


plt.xlabel("Overs")
plt.ylabel("Runs Scored")
plt.title("ODI Match Runs Comparison with Annotations for Overs > 15 Runs")
plt.show()
```

Output-

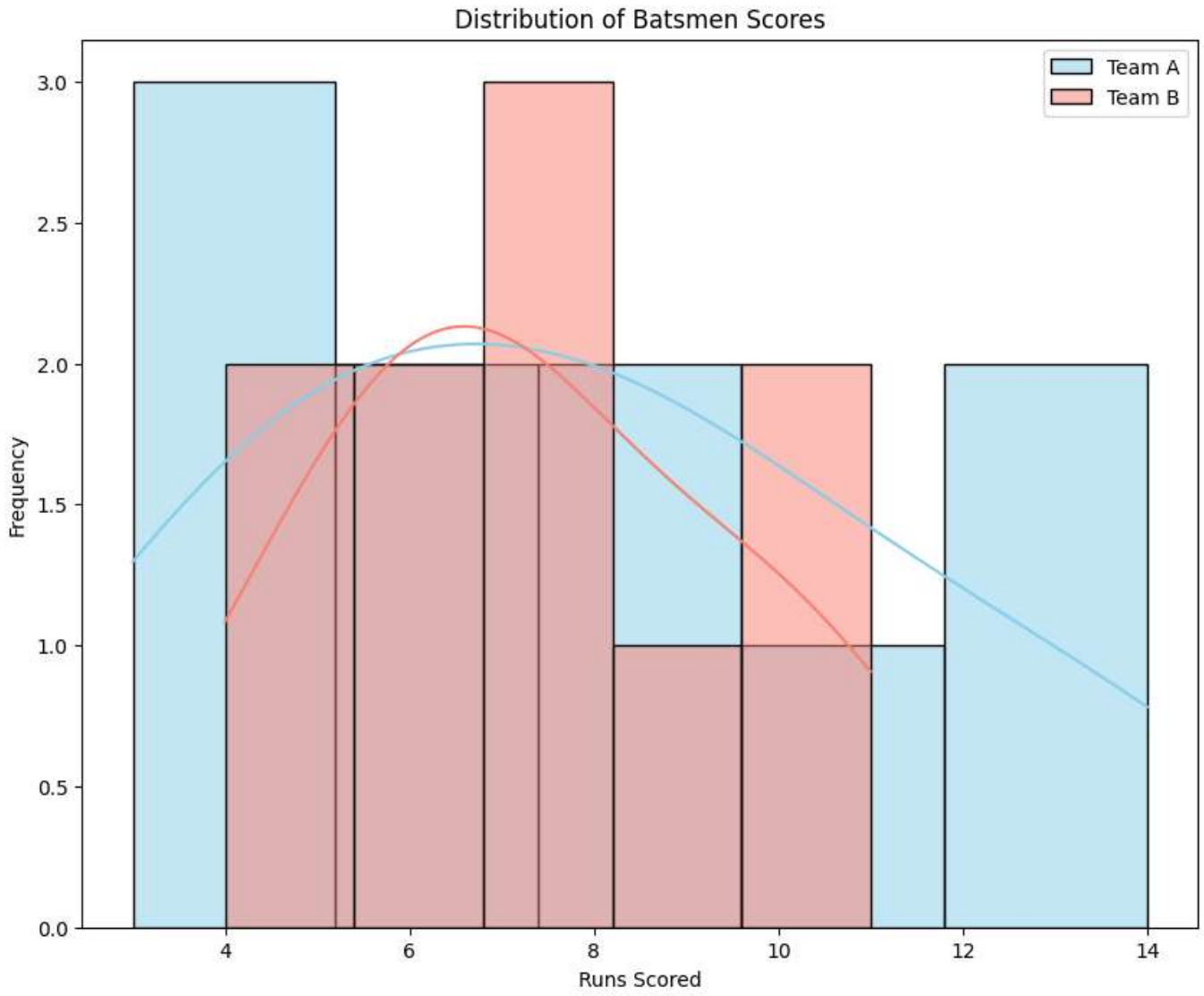


5. Create Seaborn histplots of batsmen scores (assumed data).

Code-

```
plt.figure(figsize=(10, 8))
sns.histplot(df['Team_A_Runs'], kde=True, label='Team A', color='skyblue')
sns.histplot(df['Team_B_Runs'], kde=True, label='Team B', color='salmon')
plt.xlabel('Runs Scored')
plt.ylabel('Frequency')
plt.title('Distribution of Batsmen Scores')
plt.legend()
plt.show()
```

Output-

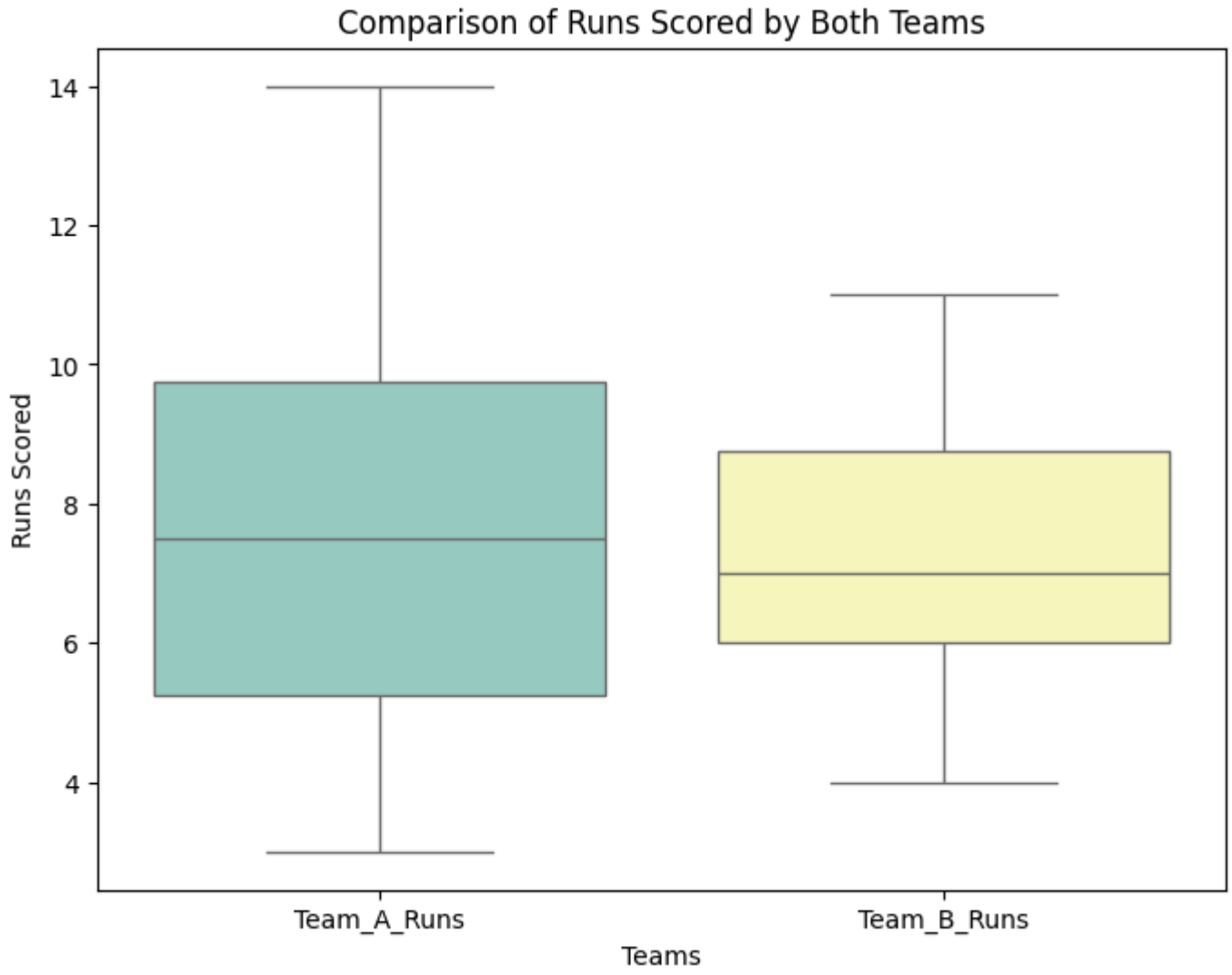


6. Create boxplots comparing both teams' runs.

Code-

```
plt.figure(figsize=(8, 6))
sns.boxplot(data=df[['Team_A_Runs', 'Team_B_Runs']], palette="Set3")
plt.xlabel("Teams")
plt.ylabel("Runs Scored")
plt.title("Comparison of Runs Scored by Both Teams")
plt.show()
```

Output-



7. Write a brief analysis comparing both teams.

Analysis of Team Performance-

Based on the provided code and visualizations, here's a brief comparison of the two teams:

1. Run Scoring Patterns:
   - The line plots show the cumulative runs scored by each team over the course of the match.
   - Visual inspection of these plots helps to understand if one team had a more dominant scoring phase or if the scores were closely matched.
   - By observing the slopes and overall trends, one can get an idea of how each team performed throughout the match.

2. Wicket Loss Patterns:
- Similarly, analyzing the wicket loss patterns can reveal if one team struggled more in maintaining wickets.

3. Powerplay Performance:
- The bar plot comparing the runs scored during the powerplay (first 10 overs) helps to quickly determine which team had a better start to their innings.

4. Overs with High Run Scoring:
- The annotated line plot highlights overs where the combined runs scored by both teams exceeded 15. These overs might indicate periods of particularly aggressive batting or effective bowling.

5. Run Distribution:
- Histograms provide insight into the distribution of runs scored by both teams over the innings. This helps to determine the overall scoring style and consistency of the teams. A skewed distribution suggests a tendency for high or low scores in some overs.

6. Run Comparison:
- Box plots provide a statistical comparison of the runs scored by each team. The box plot shows quartiles, median, and outliers, enabling a comparison of the central tendency and spread of scores, highlighting differences in overall performance and consistency.

Further Analysis Recommendations:

To deepen the analysis, consider these points:

- Calculate key performance indicators (KPIs): Calculate the run rate, average wickets lost per over, and other relevant metrics for each team to quantify performance objectively.
- Analyze individual player contributions: Analyze the performance of individual players using additional data such as batting scores, wickets taken, and strike rates.
- Consider external factors: Factor in external factors like pitch conditions, weather, toss, and team composition.
- Statistical Tests: Apply statistical tests to compare teams significantly, to determine the significance of observed differences between runs and wickets. For example use t-tests to see if the difference in average runs is statistically significant.
- Model Prediction: Build a predictive model to forecast match outcomes based on historical data.

These considerations provide a more comprehensive view of the teams' relative strengths and weaknesses in this ODI match.

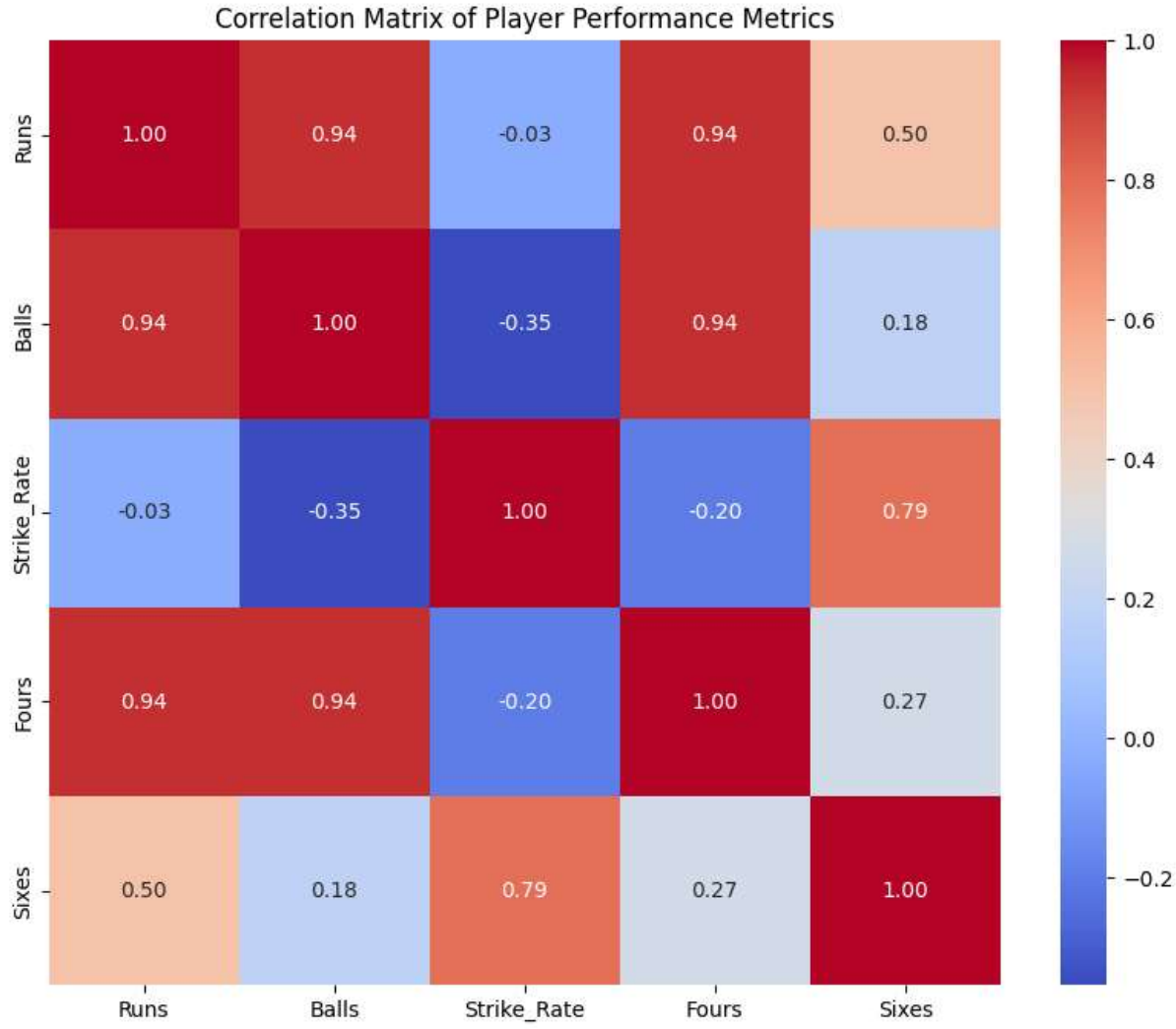
Task 2: Player Performance Heatmap & Analysis Using Seaborn.

1. Generate a Seaborn heatmap showing correlations among Runs, Balls, Strike Rate, Fours, and Sixes.

Code-

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df_player = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/player_performance.csv')
print(df_player.columns)
player_performance_subset = df_player[['Runs', 'Balls', 'Strike_Rate', 'Fours', 'Sixes']]
correlation_matrix = player_performance_subset.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Player Performance Metrics')
plt.show()
```

Output-

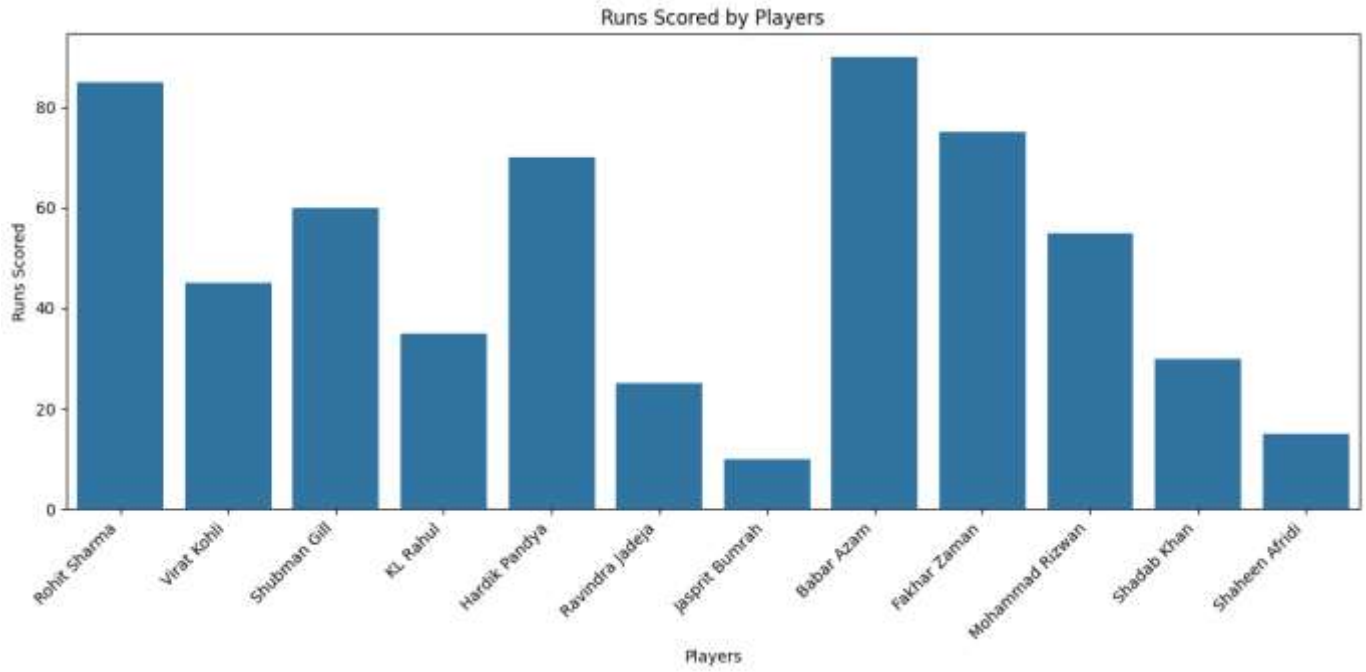


2. Create barplots for Runs scored by players.

Code-

```
plt.figure(figsize=(12, 6))
sns.barplot(x='Player', y='Runs', data=df_player)
plt.xlabel('Players')
plt.ylabel('Runs Scored')
plt.title('Runs Scored by Players')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Output-

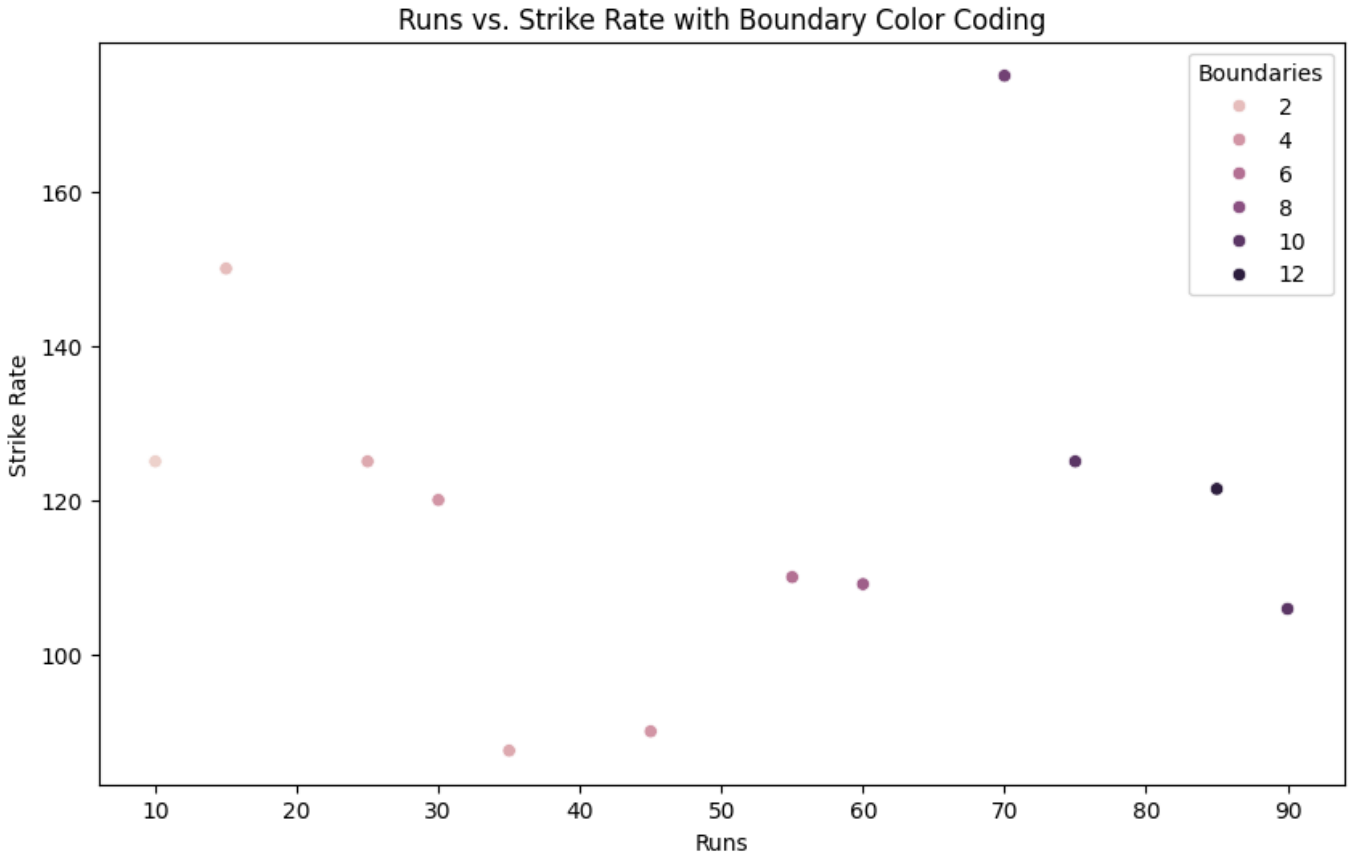


3. Create scatter plots for Runs vs Strike Rate with color coding based on boundaries.

Code-

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Runs', y='Strike_Rate', hue='Boundaries', data=df_player)
plt.xlabel('Runs')
plt.ylabel('Strike Rate')
plt.title('Runs vs. Strike Rate with Boundary Color Coding')
plt.show()
```

Output-

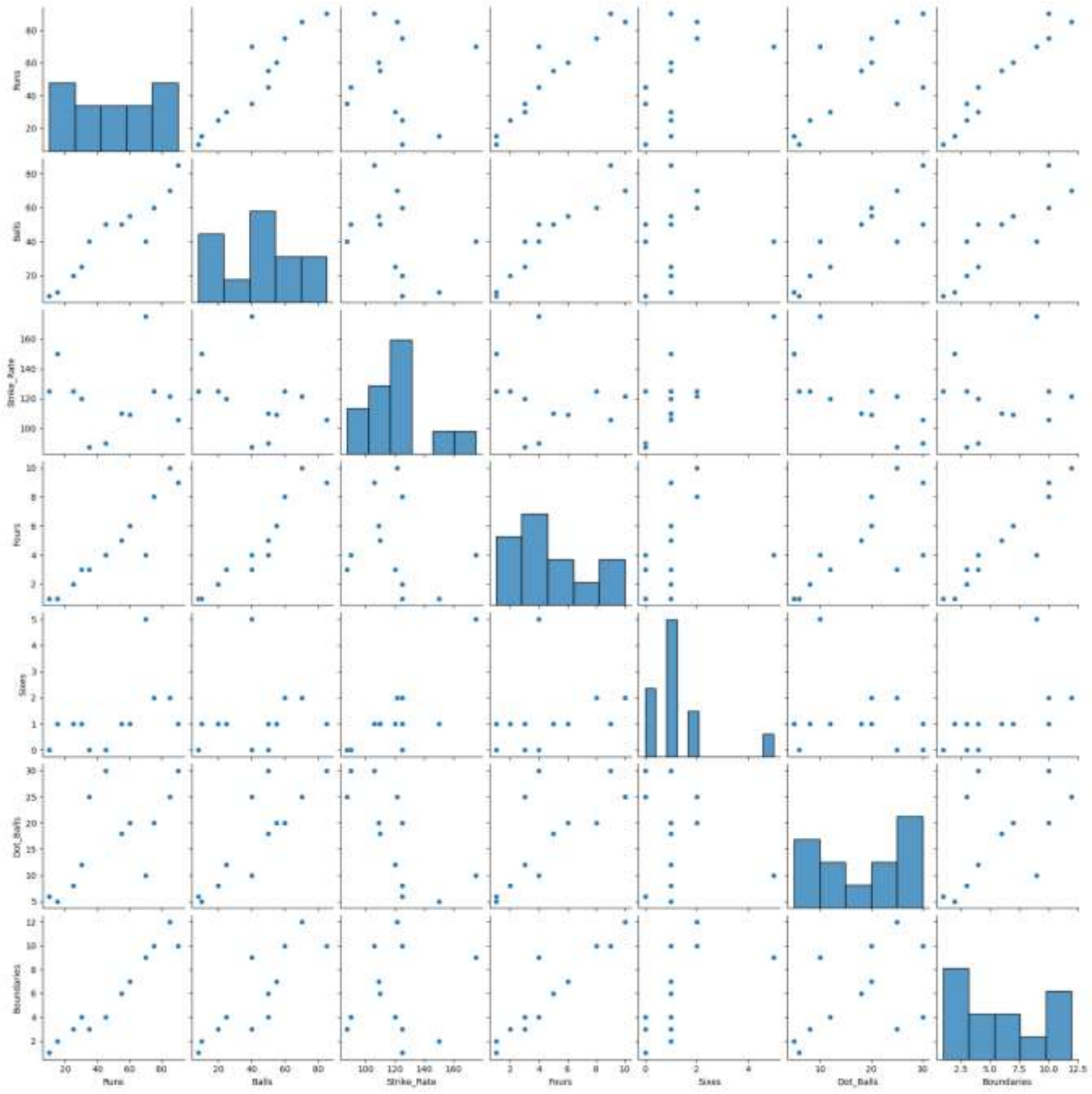


4. Create Seaborn pairplots for all numerical variables.

Code-

```
sns.pairplot(df_player.select_dtypes(include=np.number))
plt.show()
```

Output-



Assignment-6

Solve the multi-class classification problem at Iris Flowers Dataset involves predicting the flower species given measurements of Iris Flowers.

Code-

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score


iris = pd.read_csv('/content/drive/MyDrive/Iris.csv')

iris.head()

# Corrected column names

x = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]

y = iris['Species']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

model = RandomForestClassifier()

model.fit(x_train, y_train)

y_pred = model.predict(x_test)

new_data = [[6.1, 3.5, 5.4, 3.2]]

prediction = model.predict(new_data)

print("Prediction species for the new data:", [prediction][0])
```

Output-

```
Prediction species for the new data: ['Iris-virginica']

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names

warnings.warn(
```

Assignment-7

Solve the multi-class classification problem,using Wine Quality Dataset involves predicting the quality of white wines on a scale given chemical measures of each wine.

Code-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
file_path = '/content/drive/MyDrive/Colab Notebooks/WineQT.csv'
df = pd.read_csv(file_path)
print("First few rows of the dataset:")
print(df.head())
print("\nMissing values in the dataset:")
print(df.isnull().sum())
x = df.drop('quality', axis=1)
y = df['quality']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model = SVC(kernel='linear', random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy * 100:2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Output-

First few rows of the dataset:

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality  Id
0      9.4        5   0
1      9.8        5   1
2      9.8        5   2
3      9.8        6   3
4      9.4        5   4
```

Missing values in the dataset:

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
```

free sulfur dioxide 0

total sulfur dioxide 0

density 0

pH 0

sulphates 0

alcohol 0

quality 0

Id 0

dtype: int64

Accuracy: 63.755459%

Classification Report:

	precision	recall	f1-score	support
4	0.00	0.00	0.00	6
5	0.69	0.76	0.72	96
6	0.61	0.67	0.64	99
7	0.47	0.27	0.34	26
8	0.00	0.00	0.00	2
accuracy			0.64	229
macro avg	0.35	0.34	0.34	229
weighted avg	0.61	0.64	0.62	229

Confusion Matrix:

[[0 3 3 0 0]

[0 73 22 1 0]

[0 28 66 5 0]

```
 [ 0  2 17  7  0]
 [ 0  0  0  2  0]]
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Assignment-8

A binary (2-class) classification problem, using Pima Indians Diabetes Dataset involves predicting the onset of diabetes within 5 years in Pima Indians given medical details.

Code-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
url = "/content/drive/MyDrive/Colab Notebooks/diabetes.csv"
data = pd.read_csv(url)
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
print(data.isnull().sum())
x = data.drop('Outcome', axis=1)
y = data['Outcome']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}%")
print('Confusion Matrix:')
print(conf_matrix)print('Classification Report:')
```

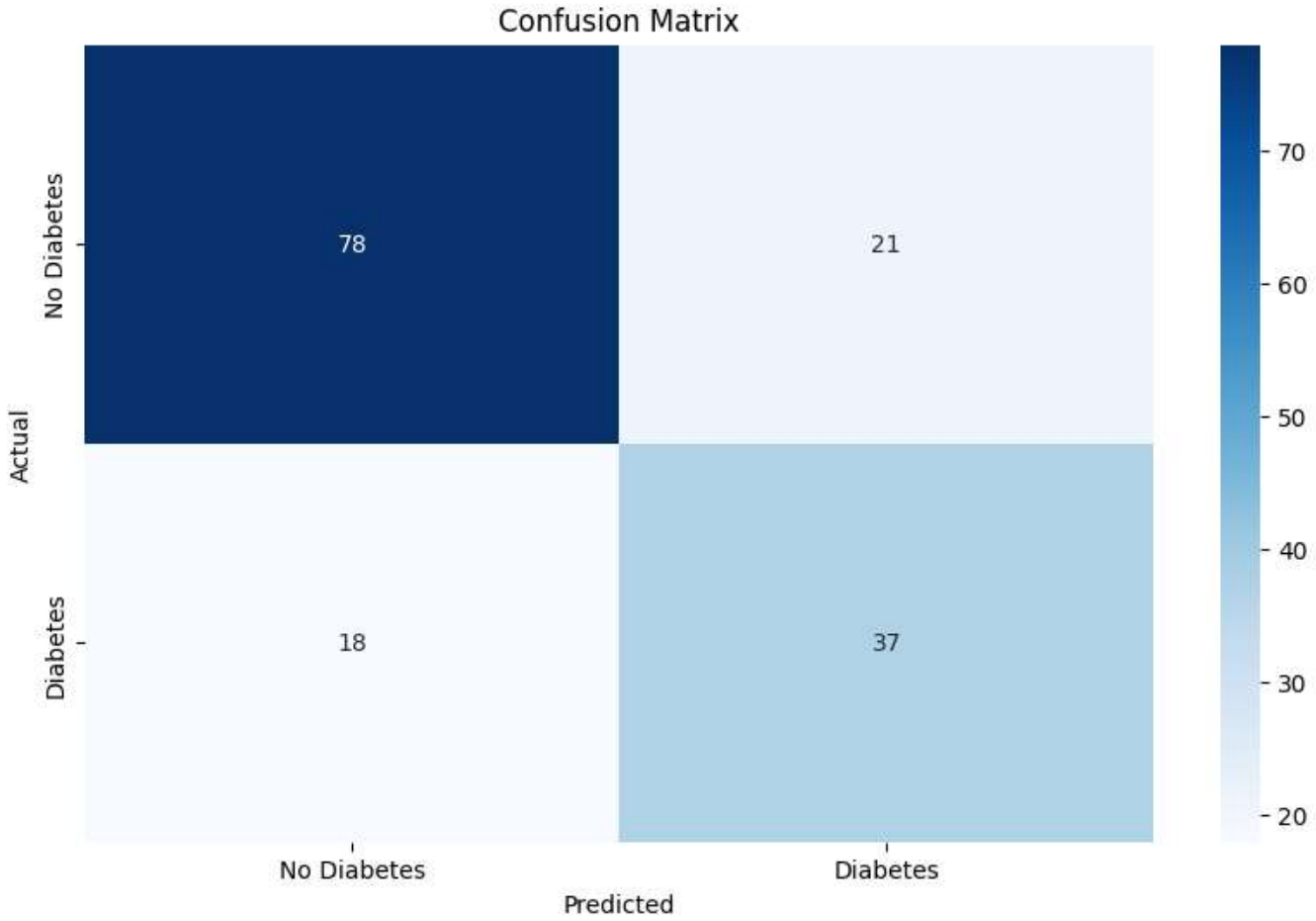
```
print (class_report)

plt.figure(figsize=(10, 6))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Diabetes', 'Diabetes'], yticklabels=['No Diabetes', 'Diabetes'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()
```

Output-

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
Accuracy: 0.75%
Confusion Matrix:
[[78 21]
 [18 37]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.81	0.79	0.80	99
1	0.64	0.67	0.65	55
accuracy			0.75	154
macro avg	0.73	0.73	0.73	154
weighted avg	0.75	0.75	0.75	154



Assignment-9

A Binary (2-class) classification problem on Sonar Dataset prediction of whether or not an object is a mine or a rock 4 given the strength of sonar returns at different angles.

Code-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
file_path = '/content/drive/MyDrive/Colab Notebooks/sonar.csv'
df = pd.read_csv(file_path, header=None)
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
y = y.map({'R': 0, 'M': 1})
sealer = StandardScaler()
x_scaled = sealer.fit_transform(x)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Output-

Accuracy: 78.571429%

Classification Report:

	precision	recall	f1-score	support
0	0.67	0.88	0.76	16
1	0.90	0.73	0.81	26
accuracy			0.79	42
macro avg	0.79	0.80	0.78	42
weighted avg	0.81	0.79	0.79	42

Confusion Matrix:

[[14 2]

[7 19]]

Assignment-10

A binary (2-class) classification problem on the Banknote Dataset involves predicting whether a given banknote is authentic given a number of measures taken from a photograph.

Code-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
file_path = '/content/drive/MyDrive/Colab Notebooks/BankNote_Authentication.csv'
df = pd.read_csv(file_path)
print("First few rows of the dataset:")
print(df.head())
print("\nMissing values in the dataset:")
print(df.isnull().sum())
sealer = StandardScaler()
x_scaled = sealer.fit_transform(df.drop('class', axis=1))
x_train, x_test, y_train, y_test = train_test_split(x_scaled, df['class'], test_size=0.2, random_state=42)
model = DecisionTreeClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy * 100:2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Output-

First few rows of the dataset:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

Missing values in the dataset:

variance 0

skewness 0

curtosis 0

entropy 0

class 0

dtype: int64

Accuracy: 97.818182%

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.99	0.98	148
1	0.99	0.96	0.98	127
accuracy			0.98	275
macro avg	0.98	0.98	0.98	275
weighted avg	0.98	0.98	0.98	275

Assignment-11

Solve the Regression Problem using Swedish Auto Insurance Dataset involves predicting the total payment for all claims in thousands of Swedish Kronor, given the total number of claims.

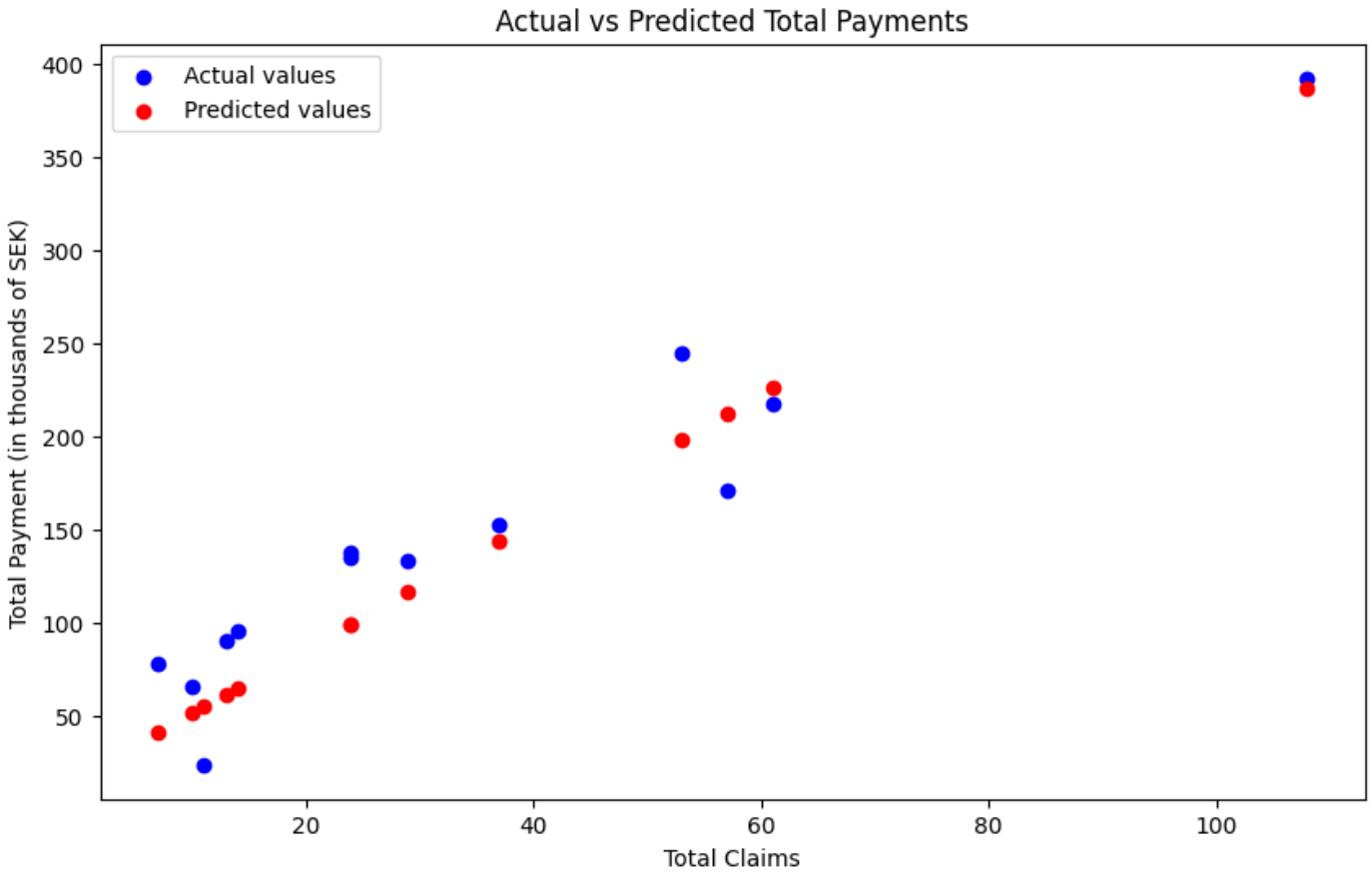
Code-

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/swedish_insurance.csv')
print(data.isnull().sum())
print(data.columns)
print(data.describe())
X = data[['X']]  # Feature
y = data['Y']    # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
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'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual values')
```

```
plt.scatter(X_test, y_pred, color='red', label='Predicted values')
plt.title('Actual vs Predicted Total Payments')
plt.xlabel('Total Claims')
plt.ylabel('Total Payment (in thousands of SEK)')
plt.legend()
```

Output-

```
X  0
Y  0
dtype: int64
Index(['X', 'Y'], dtype='object')
             X          Y
count   63.000000   63.000000
mean    22.904762   98.187302
std     23.351946   87.327553
min      0.000000    0.000000
25%      7.500000   38.850000
50%     14.000000   73.400000
75%     29.000000  140.000000
max    124.000000  422.200000
Mean Squared Error: 875.0434234424163
R-squared: 0.8950819493918402
Out[9]:
<matplotlib.legend.Legend at 0x7c03ab2c8810>
```


Actual vs Predicted Total Payments
Actual values
Predicted values
Total Payment (in thousands of SEK)
Total Claims
400
350
300
250
200
150
100
50
20
40
60
80
100

Assignment-12

A multi-class classification problem, but can also be framed as a regression on Abalone Dataset involves predicting the age of abalone given objective measures of individuals.

Code-

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error, r2_score
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Dataset/abalone.csv")
print("Data Preview:")
print(df.head())
```

Output-

```
Data Preview:
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
```

2 0.210 9

3 0.155 10

4 0.055 7

❖ Abalone Age Class Distribution , classification report, Confusion Matrix,Regression Metrics: MSE,R2 Score.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error, r2_score
df['Age'] = df['Rings'] + 1.5
le = LabelEncoder()
df['Sex'] = le.fit_transform(df['Sex'])
bins = [0, 8, 11, 30]
labels = ['Young', 'Middle', 'Old']
df['AgeClass'] = pd.cut(df['Age'], bins=bins, labels=labels)
target_regression = df['Age']
target_class = df['AgeClass']
df.dropna(inplace=True)
features = df[['Sex', 'Length', 'Diameter', 'Height', 'Whole weight',
        'Shucked weight', 'Viscera weight', 'Shell weight']]
X = features
y_class = target_class
y_reg = target_regression
sns.countplot(x='AgeClass', data=df, palette='Set2')
plt.title("Abalone Age Class Distribution")plt.show()X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X, y_class, test_size=0.2, random_state=42)
```

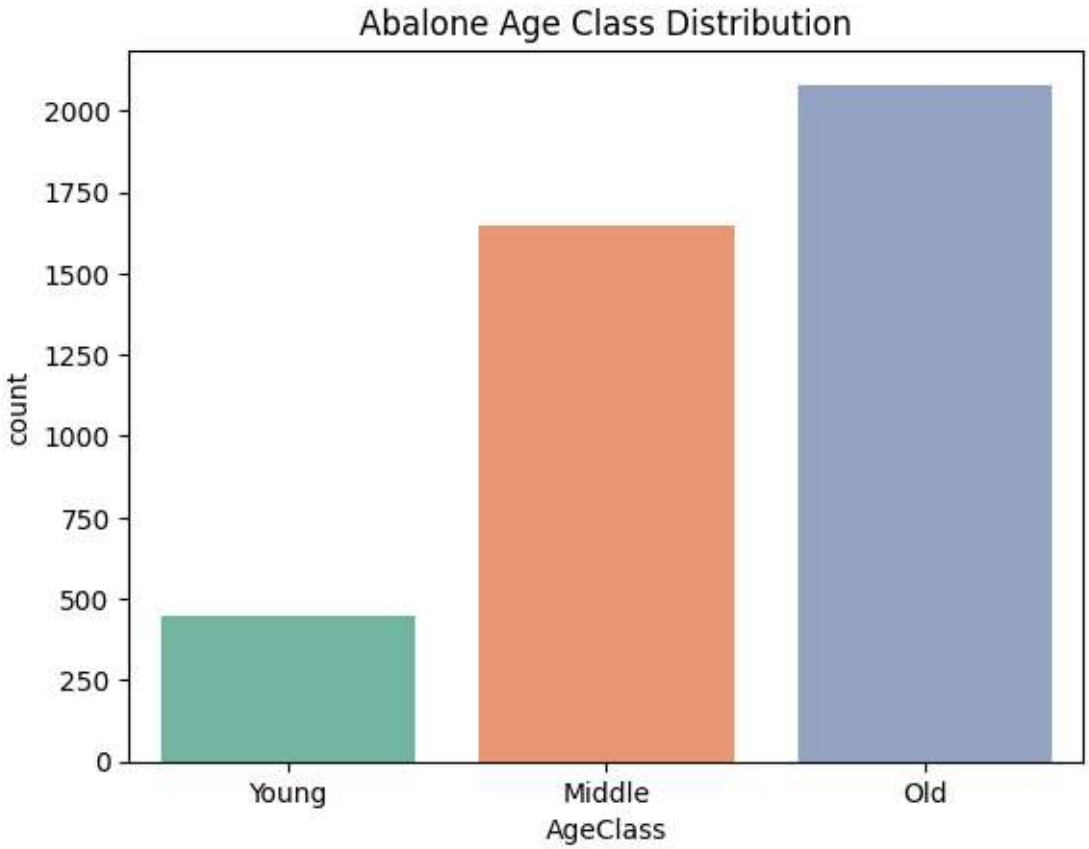
```
X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X, y_reg, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_c = scaler.fit_transform(X_train_c)
X_test_c = scaler.transform(X_test_c)
X_train_r = scaler.fit_transform(X_train_r)
X_test_r = scaler.transform(X_test_r)
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train_c, y_train_c)
y_pred_c = clf.predict(X_test_c)
print("\nClassification Report:")
print(classification_report(y_test_c, y_pred_c))
print("\nConfusion Matrix:")
sns.heatmap(confusion_matrix(y_test_c, y_pred_c), annot=True, fmt="d", cmap="Blues",
        xticklabels=labels, yticklabels=labels)
plt.title("Confusion Matrix - Age Class")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train_r, y_train_r)
y_pred_r = model.predict(X_test_r)
print("\nRegression Metrics:")
print("MSE:", mean_squared_error(y_test_r, y_pred_r))
print("R2 Score:", r2_score(y_test_r, y_pred_r))
plt.figure(figsize=(7, 5))
sns.scatterplot(x=y_test_r, y=y_pred_r, alpha=0.6)
plt.xlabel("Actual Age")
plt.ylabel("Predicted Age")
plt.title("Regression: Actual vs Predicted Age")
plt.plot([y_test_r.min(), y_test_r.max()], [y_test_r.min(), y_test_r.max()], 'r--')
```

plt.show()

Output-

<ipython-input-10-6427c20c4ca5>:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect. sns.countplot(x='AgeClass', data=df, palette='Set2')



📋 Classification Report:

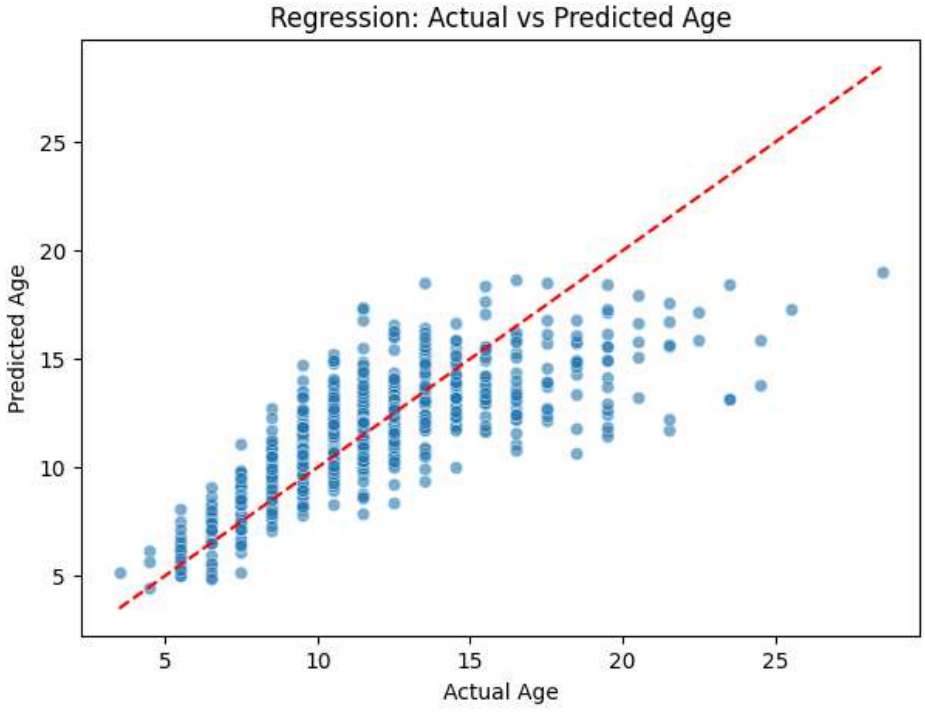
	precision	recall	f1-score	support
Middle	0.66	0.66	0.66	308
Old	0.79	0.84	0.82	425
Young	0.85	0.65	0.74	103
accuracy			0.75	836
macro avg	0.77	0.72	0.74	836
weighted avg	0.75	0.75	0.75	836

📊 Confusion Matrix:

📉 Regression Metrics:

MSE: 4.912791387559809

R2 Score: 0.5770191353452927



Assignment-13

A binary (2-class) classification problem Ionosphere Dataset requires the prediction of structure in the atmosphere given radar returns targeting free electrons in the ionosphere.

Code-

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Output-

Feature_0	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	Feature_9	...	Feature_25	Feature_26	Feature_27	Feature_28	Feature_29	Feature_30	Feature_31	Feature_32	Feature_33	Target	
0	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	column_i	column_j	...	column_z	column_aa	column_ab	column_ac	column_ad	column_ae	column_af	column_ag	column_ah	column_ai
1	true	false	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1	0.03760	...	-0.51171	0.41078	-0.46168	0.21266	-0.34090	0.42267	-0.54487	0.18641	-0.45300	g
2	true	false	1	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1	-0.04549	...	-0.26569	-0.20468	-0.18401	-0.19040	-0.11593	-0.16626	-0.06288	-0.13738	-0.02447	b
3	true	false	1	-0.03365	1	0.00485	1	-0.12062	0.88965	0.01198	...	-0.40220	0.58984	-0.22145	0.43100	-0.17365	0.60436	-0.24180	0.56045	-0.38238	g
4	true	false	1	-0.45161	1	1	0.71216	-1	0	0	...	0.90695	0.51613	1	1	-0.20099	0.25682	1	-0.32382	1	b

5 rows × 35 columns

```
df['Target'] = df['Target'].map({'g': 1, 'b': 0})
X = df.drop(columns=['Target'])
y = df['Target']
y = y.fillna(y.mode()[0])
for column in X.select_dtypes(include=['object']).columns:
    label_encoder = LabelEncoder()
    X[column] = label_encoder.fit_transform(X[column])
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)
# Train a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))

# Visualize feature importance
feature_importance = pd.Series(clf.feature_importances_, index=X.columns)
plt.figure(figsize=(10, 5))
feature_importance.nlargest(10).plot(kind='barh')
```

```
plt.title("Top 10 Important Features")
plt.show()
```

Outout-

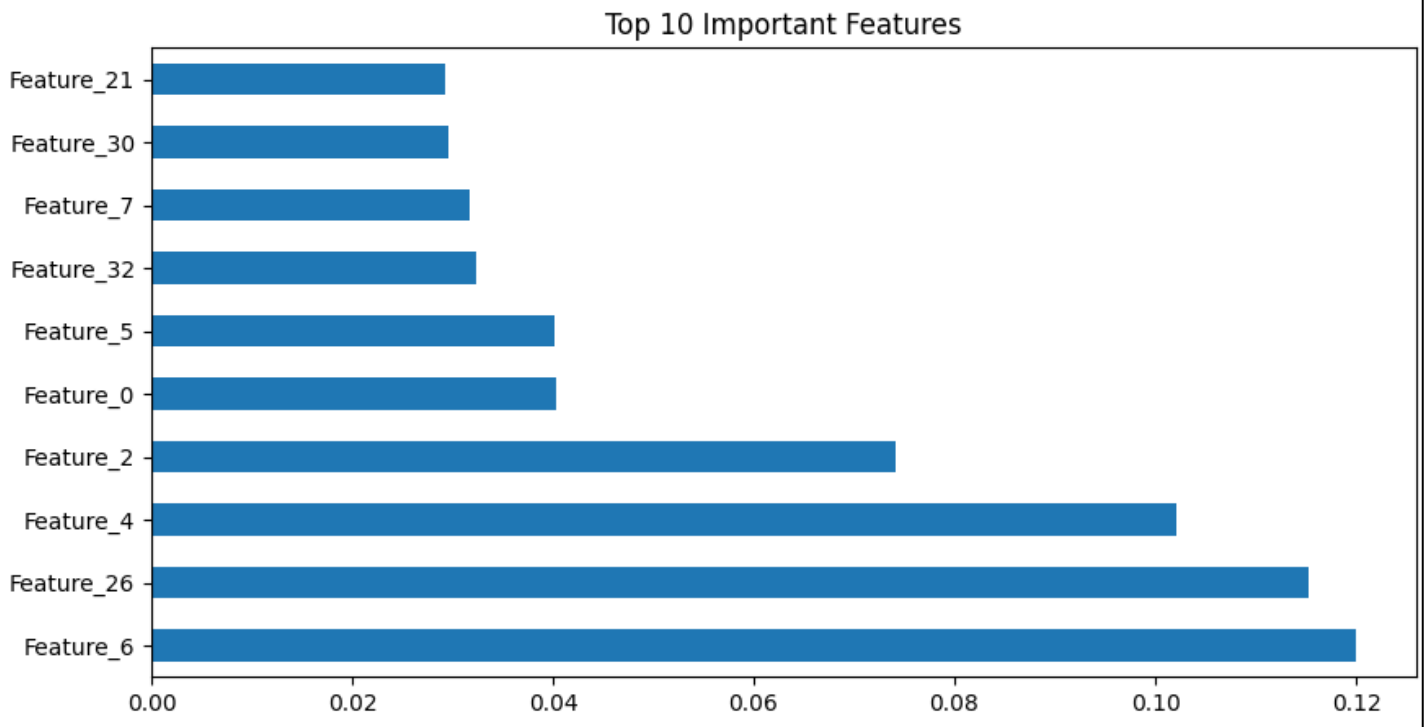
Classification Report:

	precision	recall	f1-score	support
0.0	0.83	0.96	0.89	25
1.0	0.98	0.89	0.93	46
accuracy			0.92	71
macro avg	0.90	0.93	0.91	71
weighted avg	0.92	0.92	0.92	71

Confusion Matrix:

```
[[24  1]
 [ 5 41]]
```

Accuracy Score: 0.9154929577464789



Assignment-14

A binary (2-class) classification problem , Wheat Seeds Dataset prediction of species given measurements of seeds from different varieties of wheat.

Code-

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Load the Wheat Seeds dataset

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00236/seeds_dataset.txt"

columns = ["Area", "Perimeter", "Compactness", "Kernel_Length", "Kernel_Width", "Asymmetry_Coeff",
"Kernel_Groove", "Target"]

df = pd.read_csv(url, delim_whitespace=True, names=columns)


# Convert target labels (1, 2, 3) to binary classification (1 vs. not 1)

df['Target'] = (df['Target'] == 1).astype(int)


# Split data into features (X) and target (y)

X = df.drop(columns=['Target'])

y = df['Target']


# Standardize features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)

# Train a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))

# Visualize feature importance
feature_importance = pd.Series(clf.feature_importances_, index=X.columns)
plt.figure(figsize=(10, 5))
feature_importance.nlargest(7).plot(kind='barh')
plt.title("Feature Importance in Wheat Seed Classification")
plt.show()
```

Output-

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

           0       0.93      1.00      0.97        28
```

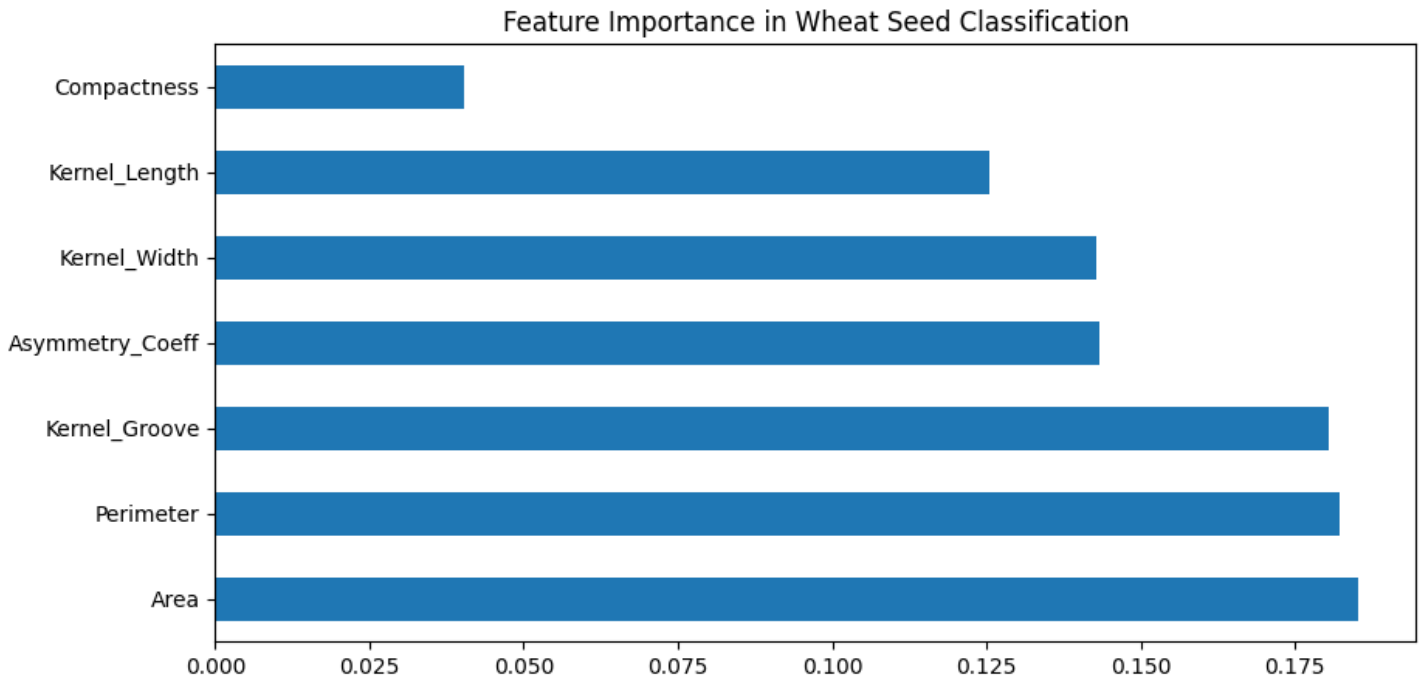
1	1.00	0.86	0.92	14
accuracy			0.95	42
macro avg	0.97	0.93	0.94	42
weighted avg	0.96	0.95	0.95	42

Confusion Matrix:

[[28 0]

[2 12]]

Accuracy Score: 0.9523809523809523



Assignment-15

Solve the regression problem on Boston House Price Dataset involves the prediction of a house price in thousands of dollars given details of the house and its neighborhood.

Code-

```
import pandas as pd
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, r2_score
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
model = LinearRegression()
from sklearn.datasets import fetch_california_housing
california = fetch_california_housing()
data = pd.DataFrame(california.data, columns=california.feature_names)
data['PRICE'] = california.target
X = data.drop(columns=['PRICE'])
y = data['PRICE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mse, r2
```

Output: (0.5558915986952442, 0.575787706032451)