Name : Arun M Register No : 23122110 Class : 3MScDS B

A. Data Exploration and Preprocessing

```
from sklearn.datasets import fetch_california_housing
import pandas as pd

# Load the California housing dataset
housing = fetch_california_housing()

# Create a DataFrame from the dataset using the data and feature names
df = pd.DataFrame(data=housing.data, columns=housing.feature_names)

# Print the DataFrame to view the data
print(df)

# Print the keys of the housing dataset to see available information
print(housing.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names',
'DESCR'])
```

```
#Finding the First five Rows
df.head()
```

```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467
37.85

   Longitude  House Price
0    -122.23        4.526
1    -122.22        3.585
2    -122.24        3.521
3    -122.25        3.413
4    -122.25        3.422
```

```
#Finding the descriptive Statistics of the All Vaiables
df.describe()
```

```
             MedInc      HouseAge      AveRooms     AveBedrms
Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000
```

```
20640.000000
mean        3.870671     28.639486      5.429000      1.096675
1425.476744
std         1.899822     12.585558      2.474173      0.473911
1132.462122
min         0.499900      1.000000      0.846154      0.333333
3.000000
25%         2.563400     18.000000      4.440716      1.006079
787.000000
50%         3.534800     29.000000      5.229129      1.048780
1166.000000
75%         4.743250     37.000000      6.052381      1.099526
1725.000000
max        15.000100     52.000000    141.909091     34.066667
35682.000000

           AveOccup      Latitude     Longitude
count  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704
std       10.386050      2.135952      2.003532
min        0.692308     32.540000   -124.350000
25%        2.429741     33.930000   -121.800000
50%        2.818116     34.260000   -118.490000
75%        3.282261     37.710000   -118.010000
max     1243.333333     41.950000   -114.310000
```

```
#DataTypes of the Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
dtypes: float64(8)
memory usage: 1.3 MB
```

```
#Descriptive Statistics of All Features including all Categorical
Variables
df.describe(include="all")
```

```
                 MedInc     HouseAge     AveRooms     AveBedrms
Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000
20640.000000
mean       3.870671     28.639486      5.429000      1.096675
1425.476744
std        1.899822     12.585558      2.474173      0.473911
1132.462122
min        0.499900      1.000000      0.846154      0.333333
3.000000
25%        2.563400     18.000000      4.440716      1.006079
787.000000
50%        3.534800     29.000000      5.229129      1.048780
1166.000000
75%        4.743250     37.000000      6.052381      1.099526
1725.000000
max       15.000100     52.000000    141.909091     34.066667
35682.000000

               AveOccup      Latitude     Longitude
count  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704
std       10.386050      2.135952      2.003532
min        0.692308     32.540000   -124.350000
25%        2.429741     33.930000   -121.800000
50%        2.818116     34.260000   -118.490000
75%        3.282261     37.710000   -118.010000
max     1243.333333     41.950000   -114.310000

#Adding the target variable to the Dataset
df['House Price'] = housing.target
df

       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude  \
0      8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88
1      8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
2      7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3      5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4      3.8462      52.0  6.281853   1.081081       565.0  2.181467
37.85
...       ...       ...       ...        ...         ...       ...
...
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606
39.48
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807
```

```
39.49
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635
39.43
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209
39.43
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981
39.37

       Longitude  House Price
0        -122.23        4.526
1        -122.22        3.585
2        -122.24        3.521
3        -122.25        3.413
4        -122.25        3.422
...          ...          ...
20635    -121.09        0.781
20636    -121.21        0.771
20637    -121.22        0.923
20638    -121.32        0.847
20639    -121.24        0.894

[20640 rows x 9 columns]
```

```
#Checking is there any Missing Values
missing=df.dropna(inplace = True)
print(missing)
```

```
None
```

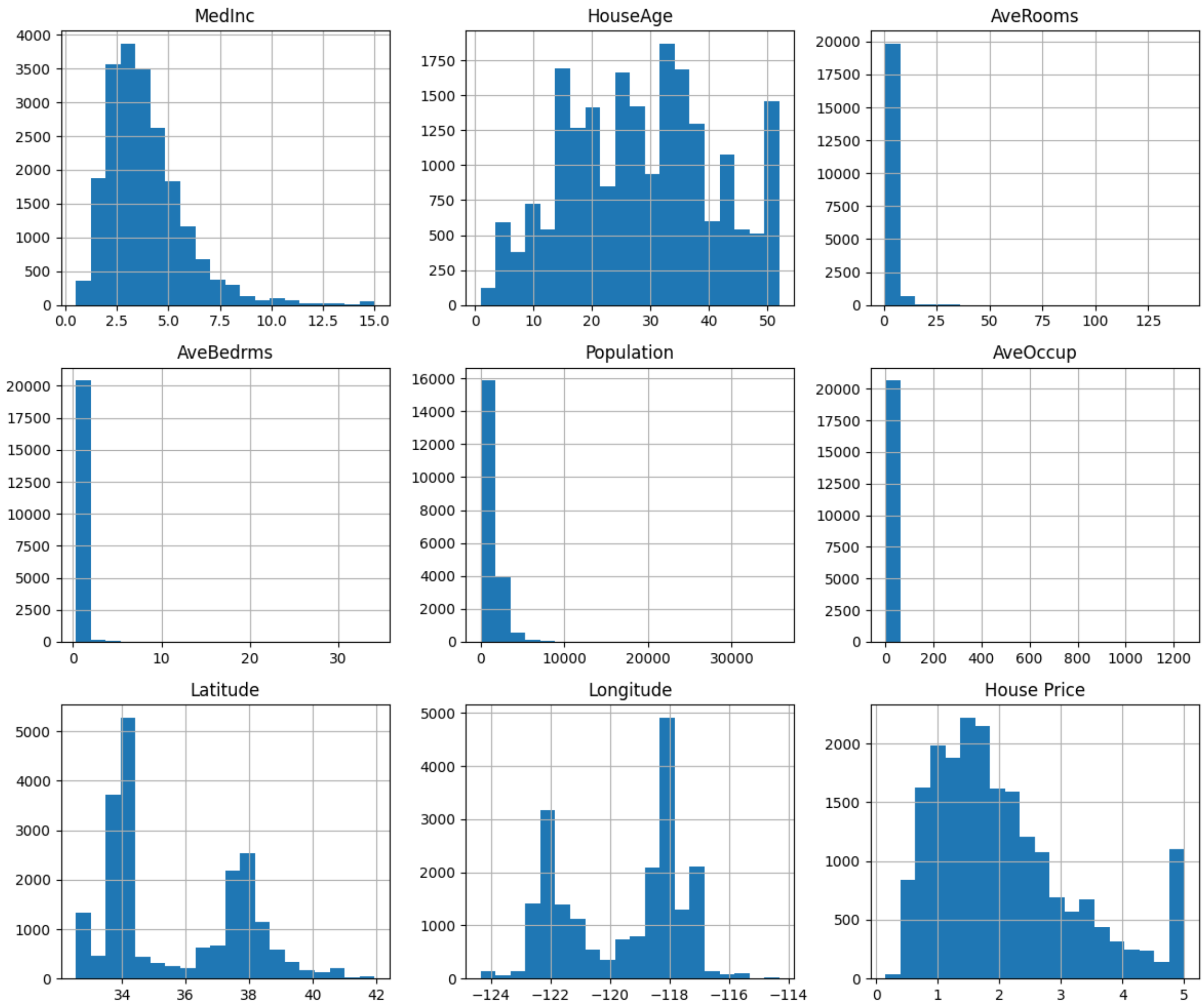
```
#Checking for Duplicate Values
dup=df.drop_duplicates(inplace = True)
print(dup)
```

```
None
```

```
#Checking the Null Values in the Dataset
null_counts = df.isnull().sum()
print("Null values in each column:")
print(null_counts)
```

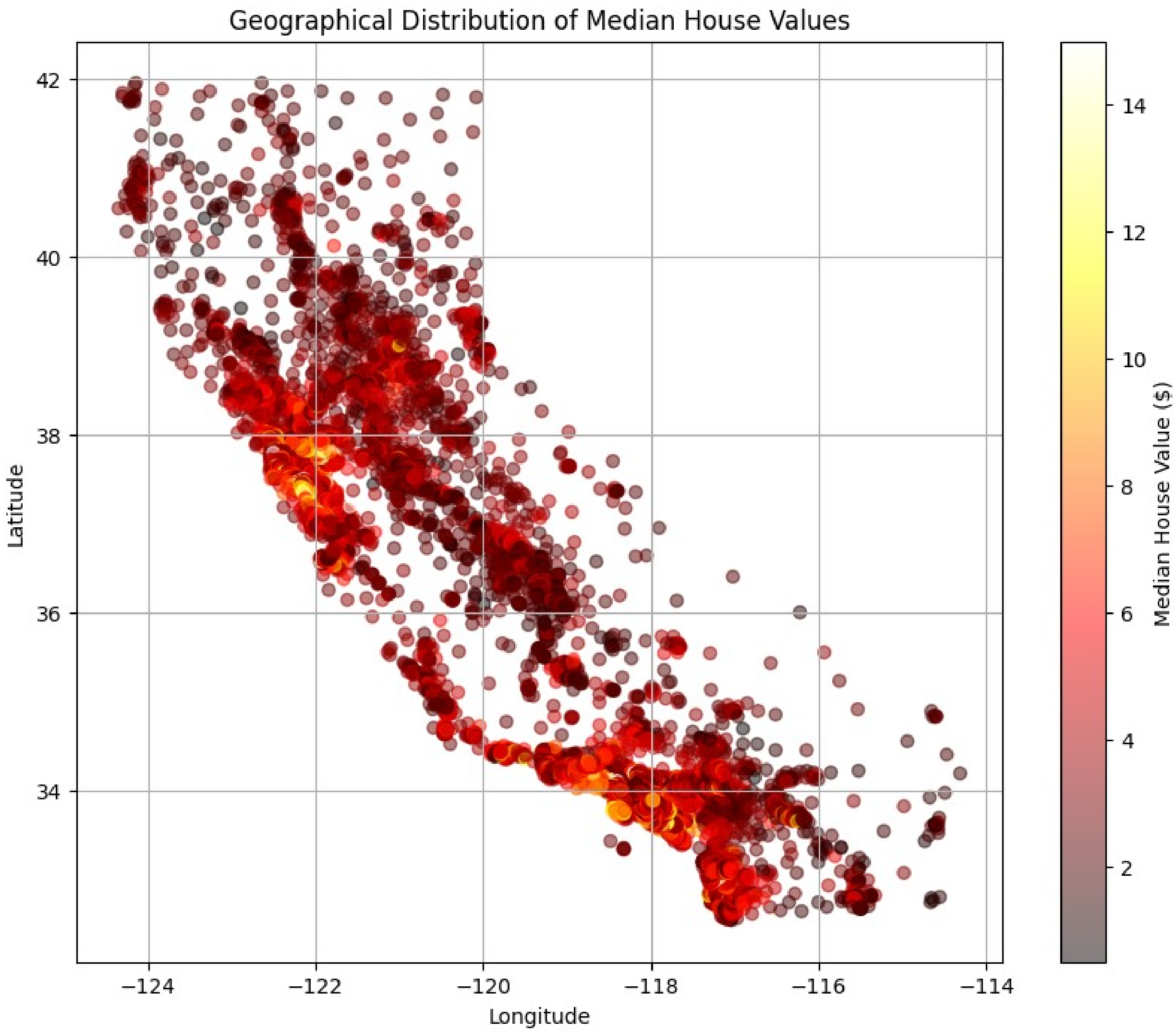
```
Null values in each column:
MedInc         0
HouseAge       0
AveRooms       0
AveBedrms      0
Population     0
AveOccup       0
Latitude       0
Longitude      0
House Price    0
dtype: int64
```

```python
#Histogram of all Features
import matplotlib.pyplot as plt

df.hist(figsize=(12,10),bins=20)
plt.tight_layout()
plt.show()
```




```python
#This is the Scatter Plot Geographical Distribution of Median House Values
plt.figure(figsize=(10, 8))
plt.scatter(df['Longitude'], df['Latitude'], c=df['MedInc'],
cmap='hot', alpha=0.5)
plt.colorbar(label='Median House Value ($)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Geographical Distribution of Median House Values')
```

```
plt.grid(True)
plt.show()
```



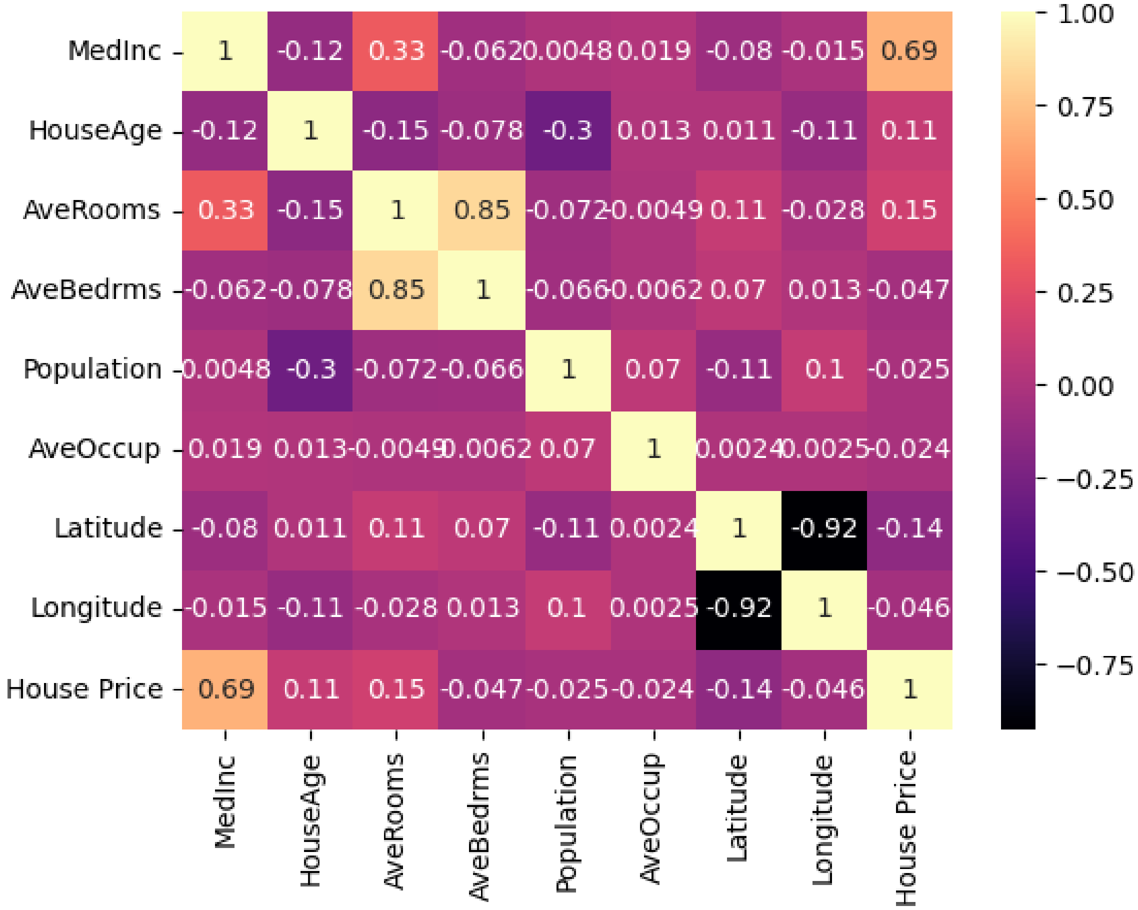
B. Model Building

```
#Finding the Correlation to Extract the Best Input Variables
corr=df.corr()
corr

                MedInc  HouseAge  AveRooms  AveBedrms  Population
AveOccup  \
MedInc        1.000000 -0.119034  0.326895  -0.062040    0.004834
0.018766
HouseAge     -0.119034  1.000000 -0.153277  -0.077747   -0.296244
0.013191
AveRooms      0.326895 -0.153277  1.000000   0.847621   -0.072213 -
0.004852
```

```
AveBedrms    -0.062040 -0.077747  0.847621   1.000000   -0.066197 -
0.006181
Population    0.004834 -0.296244 -0.072213  -0.066197    1.000000
0.069863
AveOccup      0.018766  0.013191 -0.004852  -0.006181    0.069863
1.000000
Latitude     -0.079809  0.011173  0.106389   0.069721   -0.108785
0.002366
Longitude    -0.015176 -0.108197 -0.027540   0.013344    0.099773
0.002476
House Price   0.688075  0.105623  0.151948  -0.046701   -0.024650 -
0.023737

             Latitude  Longitude  House Price
MedInc      -0.079809  -0.015176     0.688075
HouseAge     0.011173  -0.108197     0.105623
AveRooms     0.106389  -0.027540     0.151948
AveBedrms    0.069721   0.013344    -0.046701
Population  -0.108785   0.099773    -0.024650
AveOccup     0.002366   0.002476    -0.023737
Latitude     1.000000  -0.924664    -0.144160
Longitude   -0.924664   1.000000    -0.045967
House Price -0.144160  -0.045967     1.000000
```

```
import seaborn as sns
sns.heatmap(corr,annot=True,cmap='magma')
```

```
<Axes: >
```




```
#Feature Selection
import numpy as np
x=np.array(df)
Y=x[:,8]
x=x[:,0:7]

#Splitting the Dataset into Training and Testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(x,Y,random_state=42,test_size=0.25)
```

C. Linear Regression Model & E. Evaluation Metrics

```
#Fitting the Dataset into Liner Regression Model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)

LinearRegression()
```

```
#Predicting the X_test
y_pred = model.predict(X_test)
print(y_pred)
```

```
[1.00061777 1.57882678 2.59058544 ... 1.76171598 2.76006792
 3.60760481]
```

```
#Finding the r2 Score and Mean Squared Error
from sklearn.metrics import mean_squared_error, r2_score
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 Score: {r2}")
```

```
Mean Squared Error: 0.6232181213603569
R-squared Score: 0.5290133054812776
```

D. Hyperparameter Tuning and Model Evaluation & E. Evaluation Metrics:

```
from sklearn.preprocessing import StandardScaler

# Standardization using StandardScaler
scaler_standard = StandardScaler()
standardized_data = scaler_standard.fit_transform(df)
standardized_df = pd.DataFrame(standardized_data, columns=df.columns)


model = LinearRegression(fit_intercept=False)
model.fit(X_train, y_train)
```

```
LinearRegression(fit_intercept=False)
```

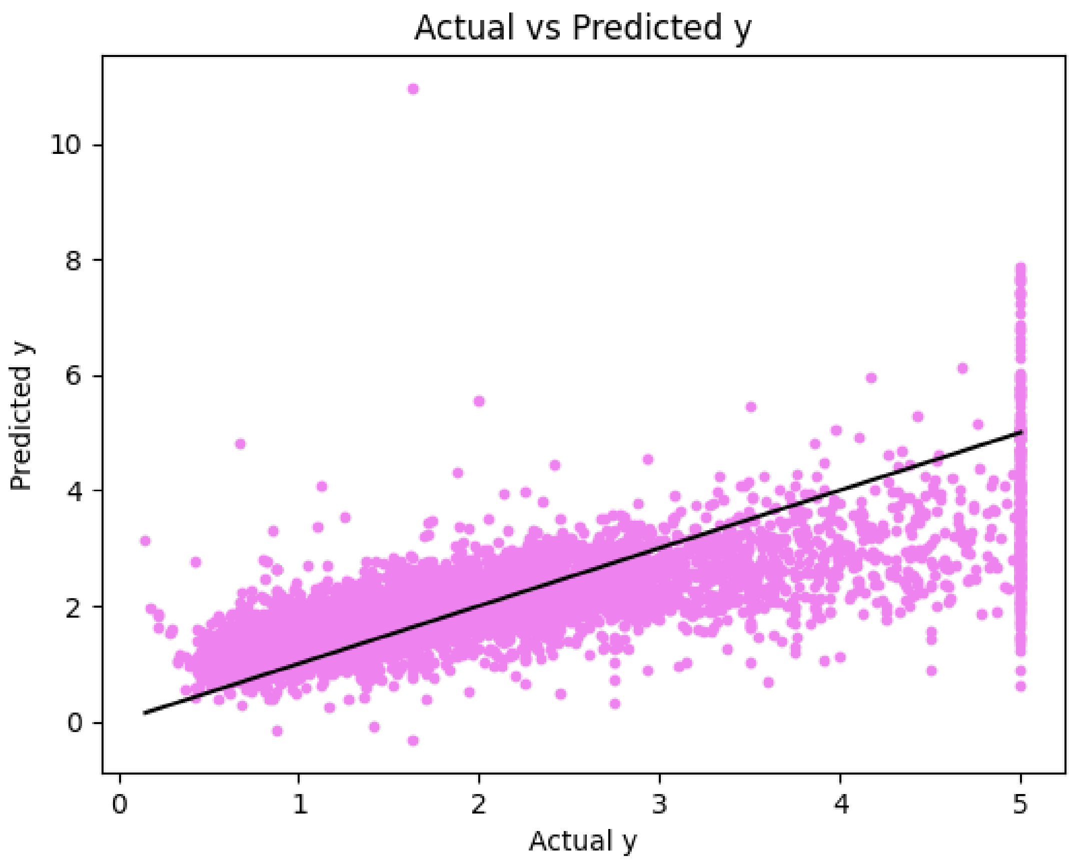
```
#predicting the X_test
y_pred = model.predict(X_test)
print(y_pred)
```

```
[2.89636454 1.10469645 3.06203802 ... 1.6674325  2.6980249
 1.87189299]
```

```
#Calculating and comparing the Root Mean Square Error (RMSE) and r2
Score with Hyperparameters
from sklearn.metrics import mean_squared_error, r2_score
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 Score: {r2}")
```

```
Mean Squared Error: 0.666295686924439
R-squared Score: 0.5264305608038502
```

```
plt.scatter(y_test, y_pred,color='violet',s=10)
plt.plot([min(Y), max(Y)], [min(Y), max(Y)], color='black')
plt.xlabel('Actual y')
plt.ylabel('Predicted y')
plt.title('Actual vs Predicted y')
plt.show()
```



INTERPRETATION

Accuracy without Hyperparameter :- 52% Accuracy with Hyperparameter :- 52%

The consistent R2 score between models with and without hyperparameter tuning suggests that either the default hyperparameters are already optimal for the dataset, the dataset itself is not highly complex, or the chosen hyperparameter range may not encompass the optimal values. It's crucial to consider expanding the hyperparameter search space or exploring alternative metrics to ensure thorough model optimization and performance evaluation.