In [472…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

In [473…
```
from sklearn.datasets import fetch_california_housing
# Loading California Housing Dataset from sklearn  in-built Datasets

california  = fetch_california_housing()
```

In [474…
```
#       Convert the dataset into a pandas DataFrame for easier handling
california_df = pd.DataFrame(california.data,
                             columns=california.feature_names)
california_df['target'] = pd.Series(california.target)
california_df.head()
```

Out[474…

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [475…
```
# Check the datatype and also if any NULL values are there in any columns
california_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 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
 8   target      20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

No NULL values found here.

In [477…
```
# Checking if any Duplicate values are there
california_df.duplicated().sum()
```

Out[477…
```
0
```

In [478…
```
california_df.describe()
```

Out[478…

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333

In [479…

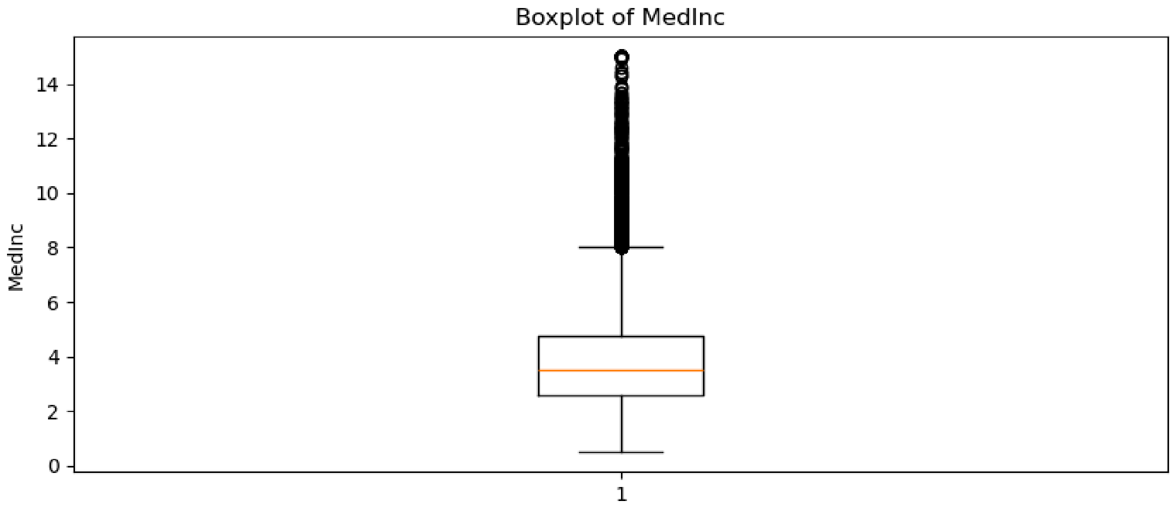
```
california_df.corr()
```

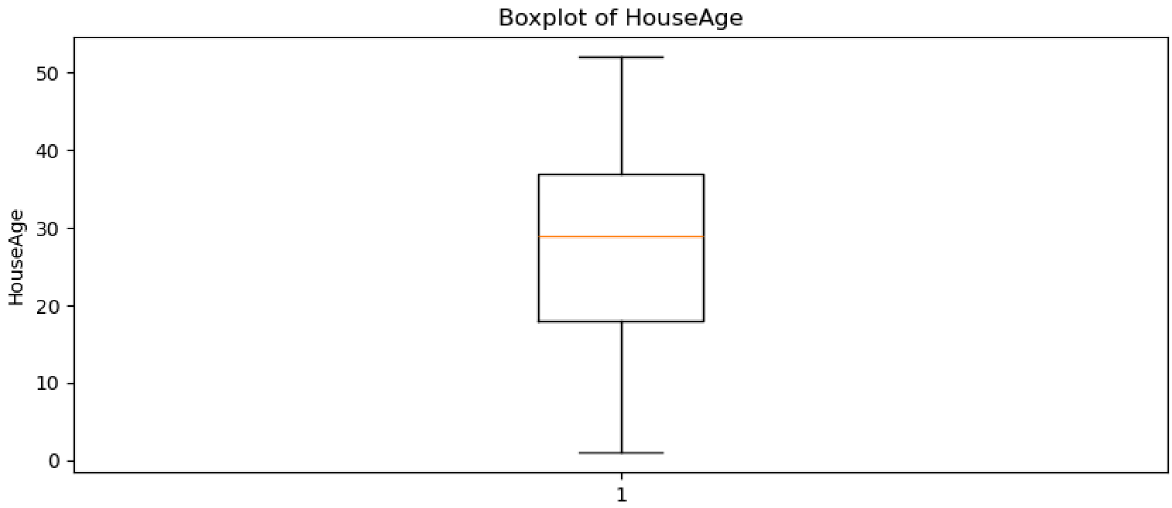
Out[479…

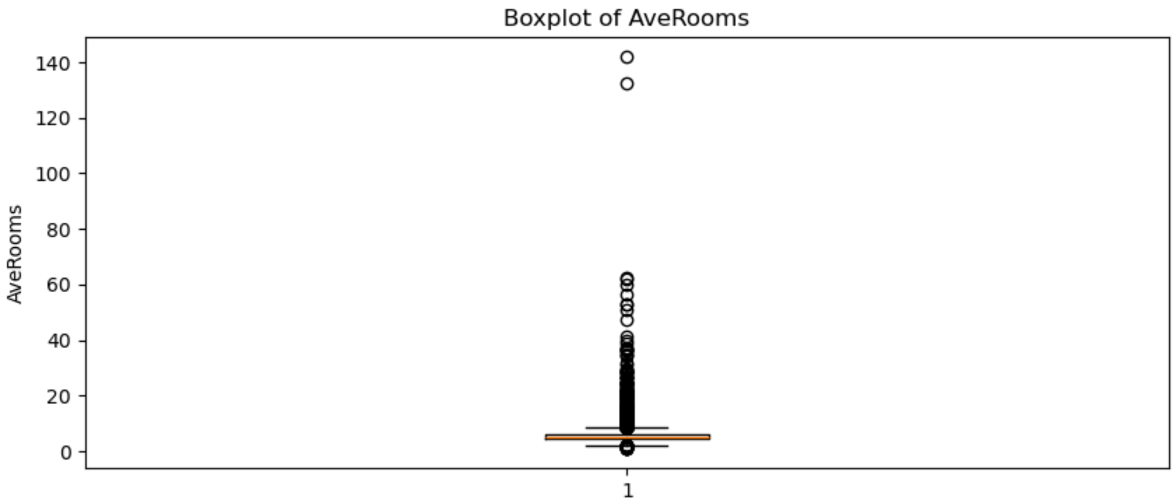
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664
target	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737	-0.144160

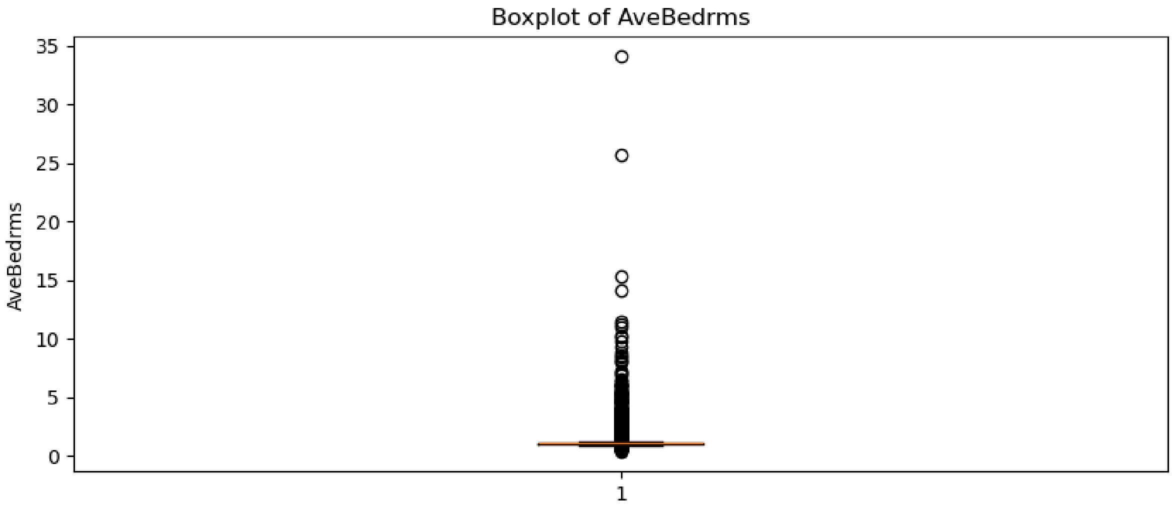
In [480…

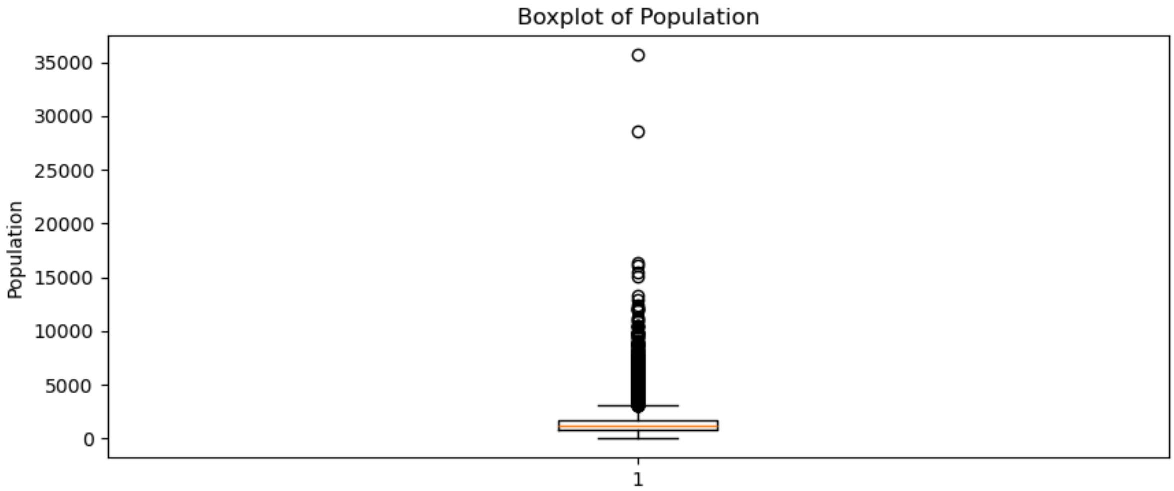
```
# Finding Outliers
numerical_cols=california_df.select_dtypes(include=np.number).columns
for column in numerical_cols:
    plt.figure(figsize=(10,4))
    plt.boxplot(california_df[column])
    plt.title(f'Boxplot of {column}')
    plt.ylabel(column)
    plt.show()
```

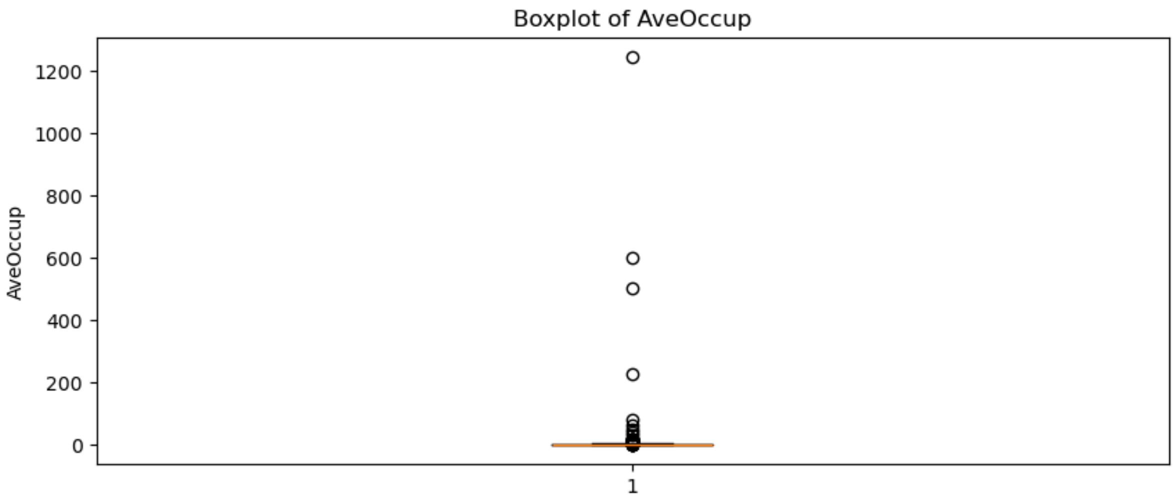

Boxplot of MedInc
MedInc
0
2
4
6
8
10
12
14
1

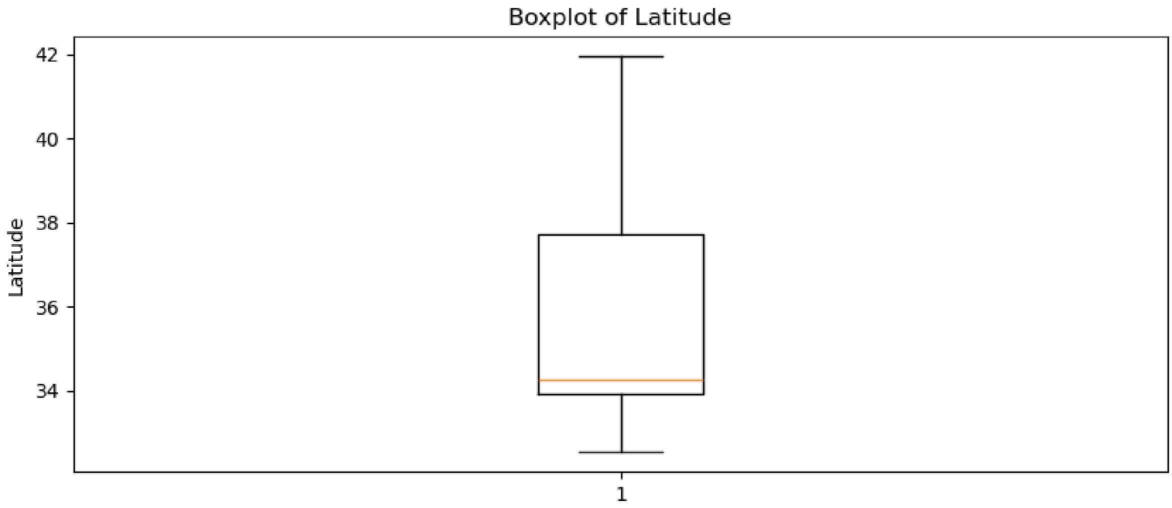

Boxplot of HouseAge
HouseAge
0
10
20
30
40
50
1

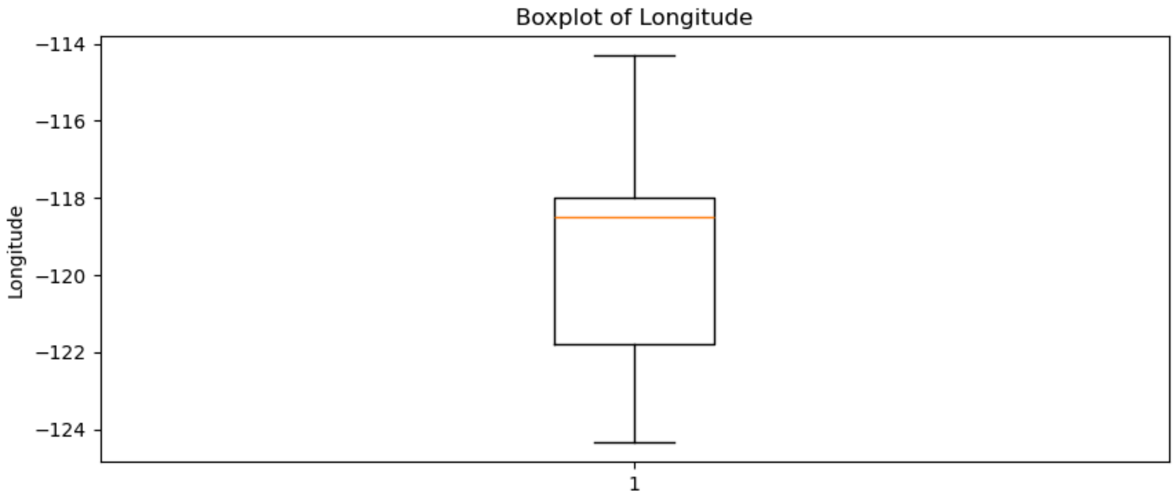

Boxplot of AveRooms
AveRooms
0
20
40
60
80
100
120
140
1

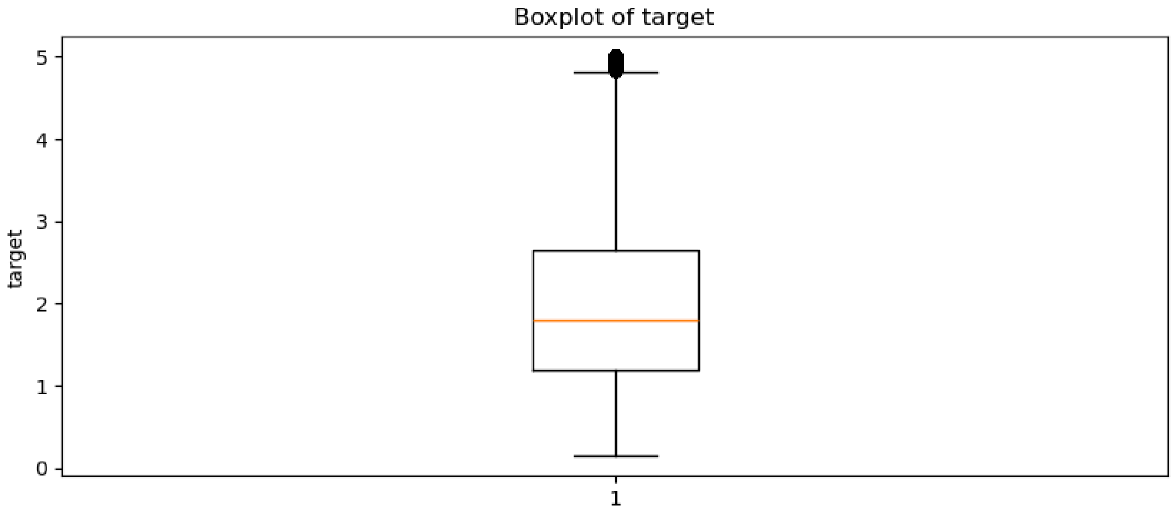

Boxplot of AveBedrms
AveBedrms
0
5
10
15
20
25
30
35
1


Boxplot of Population
Population
0
5000
10000
15000
20000
25000
30000
35000
1


Boxplot of AveOccup
AveOccup
0
200
400
600
800
1000
1200
1







In [481…

```
#nu_train = data_train.select_dtypes(include=['number']).columns  ###########*******
new_df_col = california_df.drop("target",axis=1).columns
new_df_col
```

Out[481…

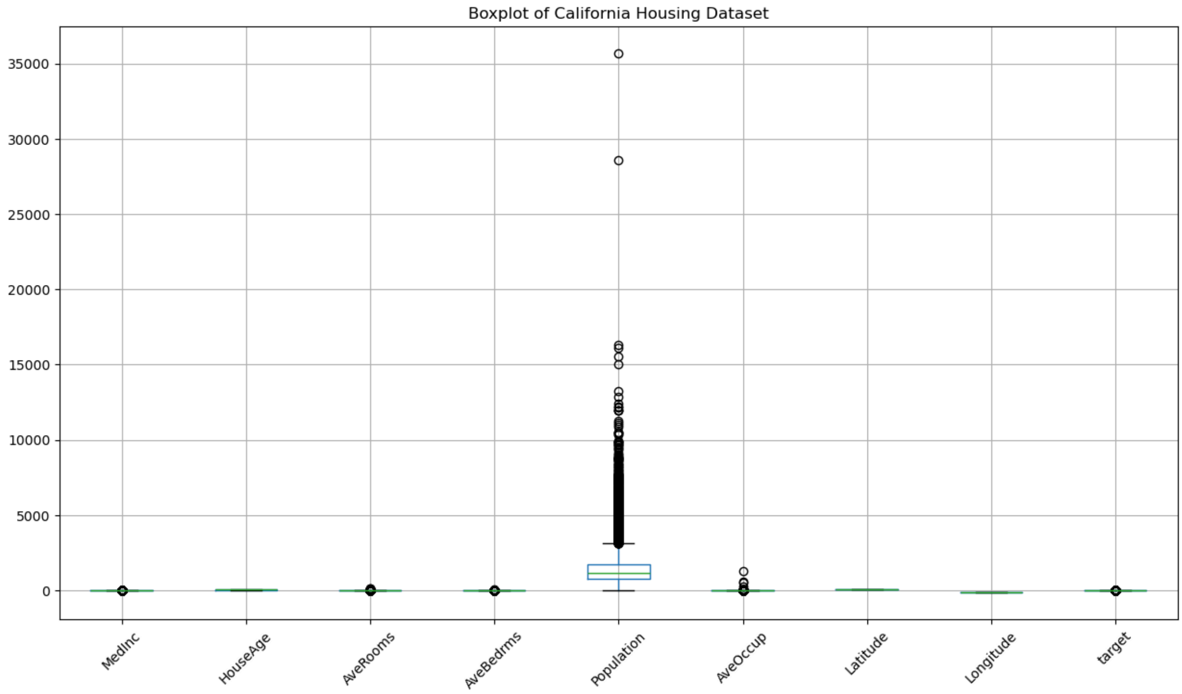
```
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude'],
      dtype='object')
```

Handling Outliers in this dataset

In [483…

```
# Visualising Outliers in this dataset
plt.figure(figsize=(15, 8))
```

```
california_df.boxplot()
plt.xticks(rotation=45)
plt.title("Boxplot of California Housing Dataset")
plt.show()
```



Here outliers are there in the Population column. So we will perform IQR method on this dataset

In [485…

```
# Calculate Q1 (25th percentile) and Q3 (75th percentile) for each column
Q1 = california_df[new_df_col].quantile(0.25)
Q3 = california_df[new_df_col].quantile(0.75)

# Calculate the IQR for each column
IQR = Q3 - Q1

# Determine the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = (california_df[new_df_col] < lower_bound) | (california_df[new_df_col] > u

# Handle outliers (example: removing them)
df_after_iqr = california_df[~outliers.any(axis=1)]

# Print the cleaned data (or you can choose to impute outliers instead of removing th
df_after_iqr
```

Out[485...

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122
...	...	...	...	...	...	...	...	
20634	3.7125	28.0	6.779070	1.148256	1041.0	3.026163	39.27	-121
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121

16842 rows × 9 columns

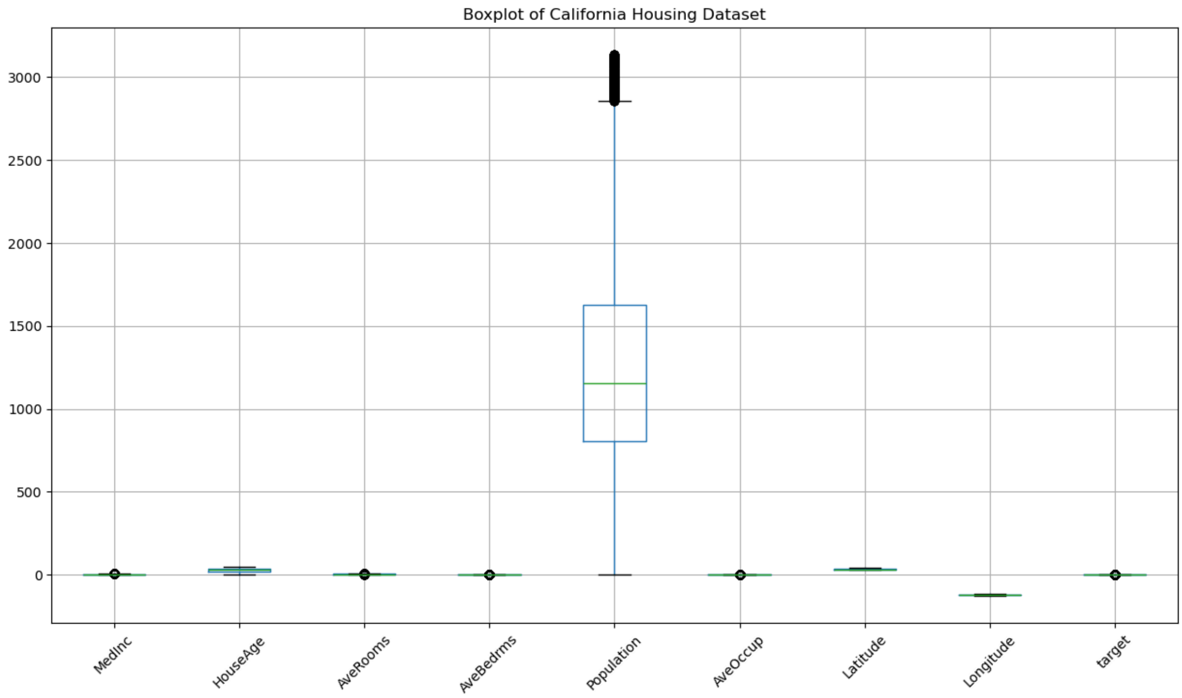
In [486...

```
df_after_iqr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16842 entries, 2 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      16842 non-null  float64
 1   HouseAge    16842 non-null  float64
 2   AveRooms    16842 non-null  float64
 3   AveBedrms   16842 non-null  float64
 4   Population  16842 non-null  float64
 5   AveOccup    16842 non-null  float64
 6   Latitude    16842 non-null  float64
 7   Longitude   16842 non-null  float64
 8   target      16842 non-null  float64
dtypes: float64(9)
memory usage: 1.3 MB
```

In [487...

```
# Visualising Outliers in the new dataset after performing IQR
plt.figure(figsize=(15, 8))
df_after_iqr.boxplot()
plt.xticks(rotation=45)
plt.title("Boxplot of California Housing Dataset")
plt.show()
```

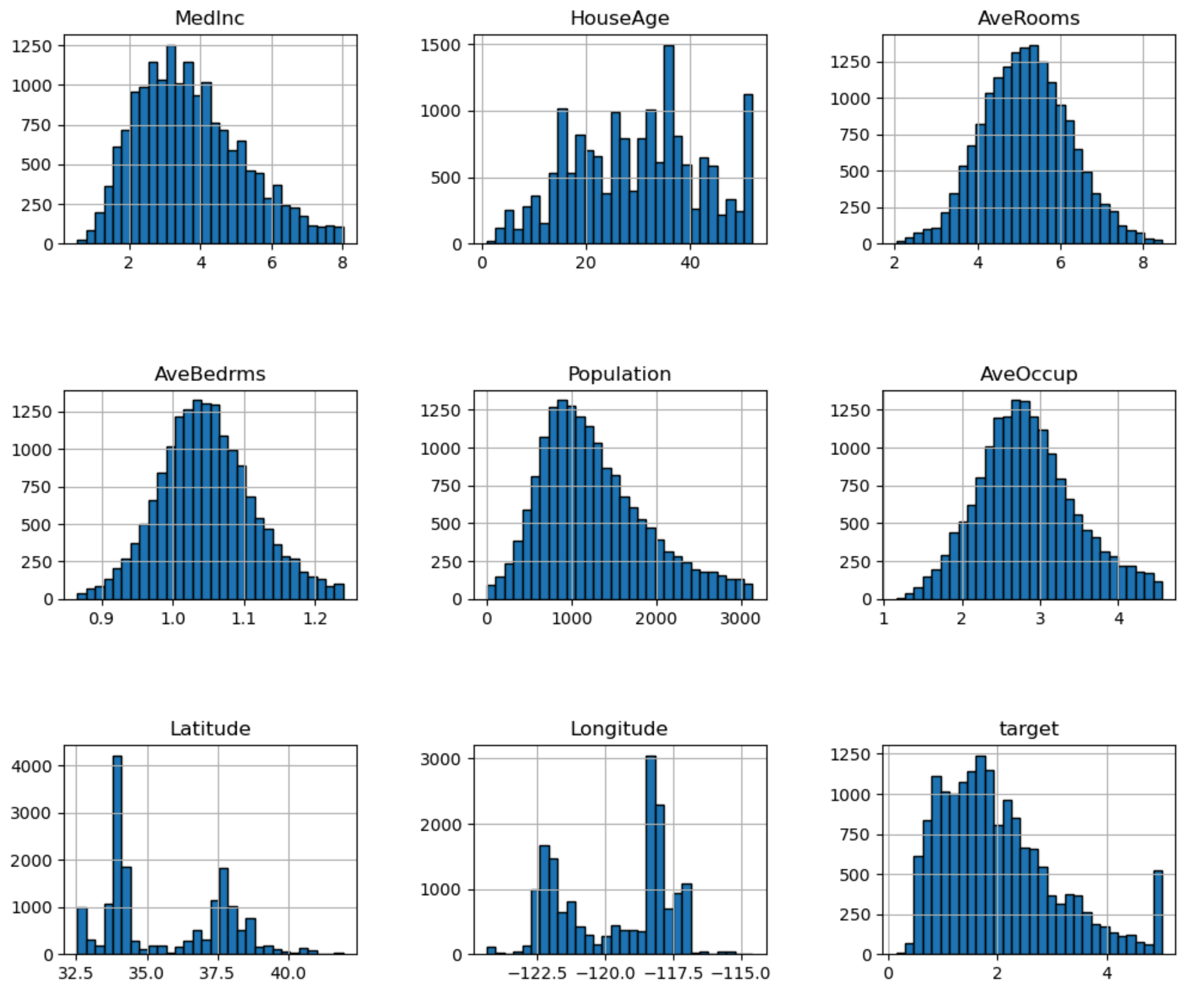


In [ ]:

No Categorical Columns found here. So no Categorical encoding required in this dataset

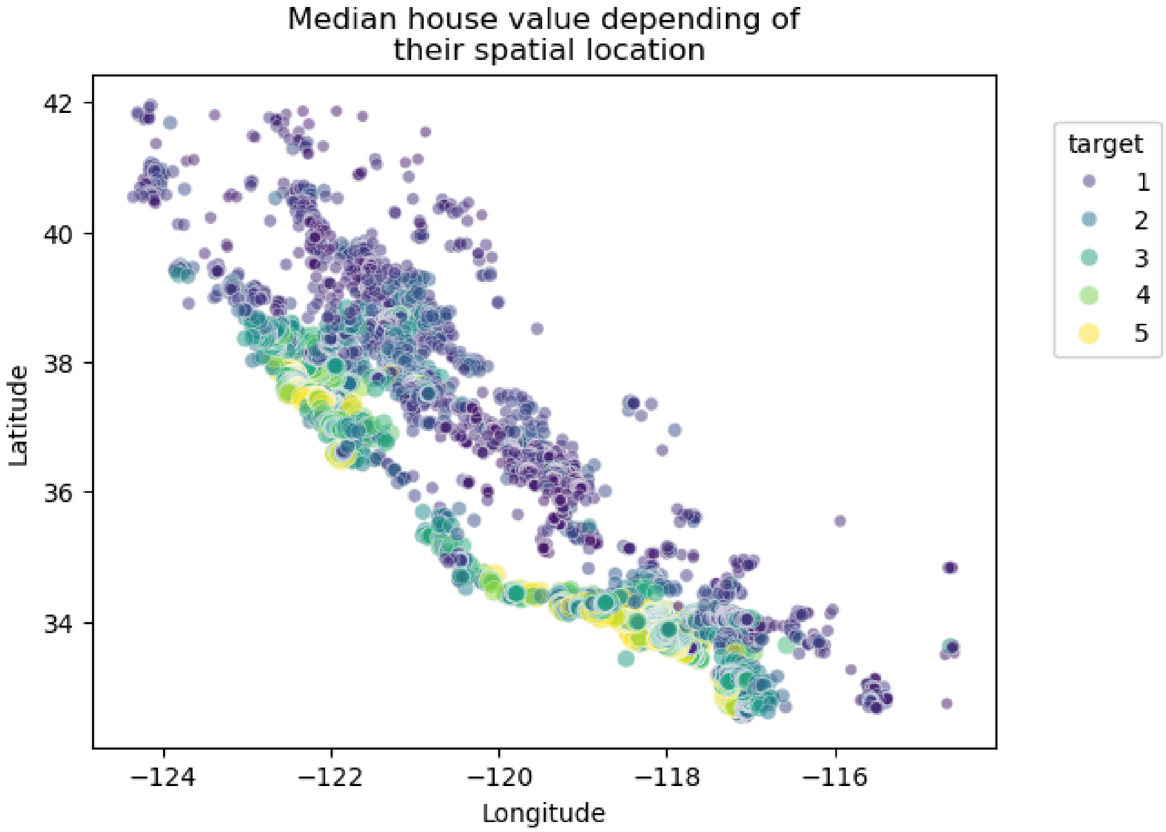
In [489…

```
df_after_iqr.hist(figsize=(12, 10), bins=30, edgecolor="black")
plt.subplots_adjust(hspace=0.7, wspace=0.4)
```

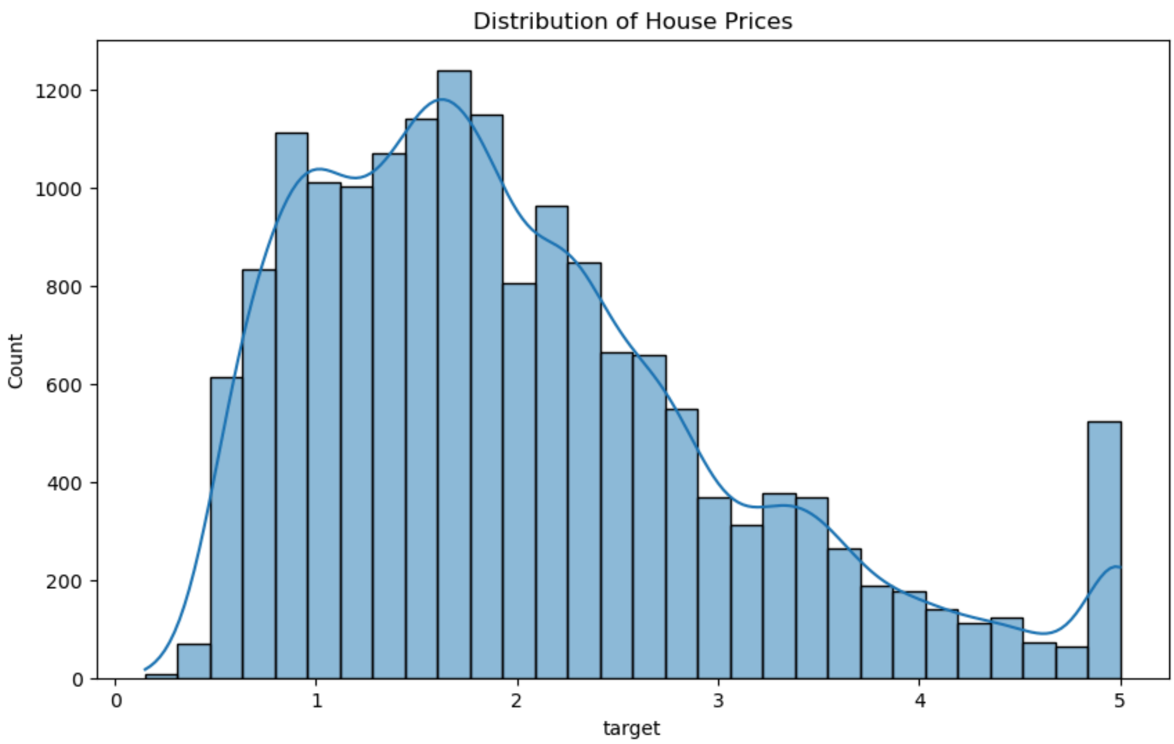


In [490…

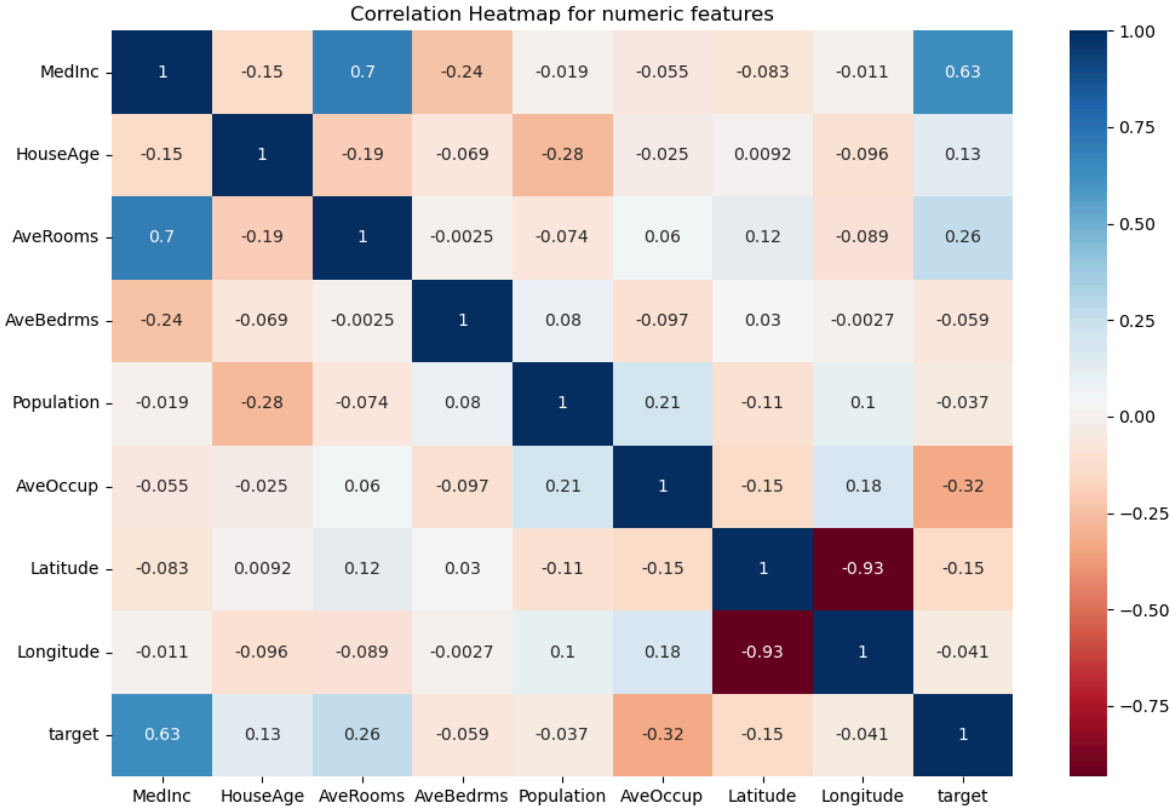
```
sns.scatterplot(
    data=df_after_iqr,
    x="Longitude",
    y="Latitude",
    size="target",
    hue="target",
    palette="viridis",
    alpha=0.5,
)
plt.legend(title="target", bbox_to_anchor=(1.05, 0.95), loc="upper left")
_ = plt.title("Median house value depending of\n their spatial location")
```



In [491…
```
plt.figure(figsize=(10, 6))
sns.histplot(df_after_iqr['target'], bins=30, kde=True)
plt.title('Distribution of House Prices')
plt.show()
```



In [492…
```
# plotting correlation between columns
plt.figure(figsize=(12, 8))
sns.heatmap(df_after_iqr.corr(), annot=True, cmap="RdBu")
plt.title("Correlation Heatmap for numeric features")
plt.show()
```



In [493…

```
# Checking Skewness
df_after_iqr.skew()
```

Out[493…

```
MedInc        0.545038
HouseAge      0.009569
AveRooms      0.113327
AveBedrms     0.242424
Population    0.731394
AveOccup      0.334693
Latitude      0.418345
Longitude    -0.275190
target        0.926549
dtype: float64
```

SKEWNESS seems fine... So now it is ready for Training purposes

2. Regression Algorithm Implementation

As the target value is continuous value, we have to use "Regression model".

a) feature selection

In [497…

```
#Copying the cleaned dataframe to new 'df'
my_df=df_after_iqr
my_df
```

Out[497…

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122
...	...	...	...	...	...	...	...	
20634	3.7125	28.0	6.779070	1.148256	1041.0	3.026163	39.27	-121
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121

16842 rows × 9 columns

In [498…

```
X = my_df.drop("target",axis=1)
Y = my_df['target']
print(X)
print(Y)
```

```
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
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
5      4.0368      52.0  4.761658   1.103627       413.0  2.139896     37.85
6      3.6591      52.0  4.931907   0.951362      1094.0  2.128405     37.84
...       ...       ...       ...        ...         ...       ...       ...
20634  3.7125      28.0  6.779070   1.148256      1041.0  3.026163     39.27
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48
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
2        -122.24
3        -122.25
4        -122.25
5        -122.25
6        -122.25
...          ...
20634    -121.56
20635    -121.09
20637    -121.22
20638    -121.32
20639    -121.24

[16842 rows x 8 columns]
2        3.521
3        3.413
4        3.422
5        2.697
6        2.992
         ...
20634    1.168
20635    0.781
20637    0.923
20638    0.847
20639    0.894
Name: target, Length: 16842, dtype: float64
```

To store performance of each models created below

In [500…
```
results = {}   # all model results will be stored here
```

K-Nearest Neighbors (KNN) REGRESSION

In [502…
```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# 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, random_state

#  Feature Scaling Process
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

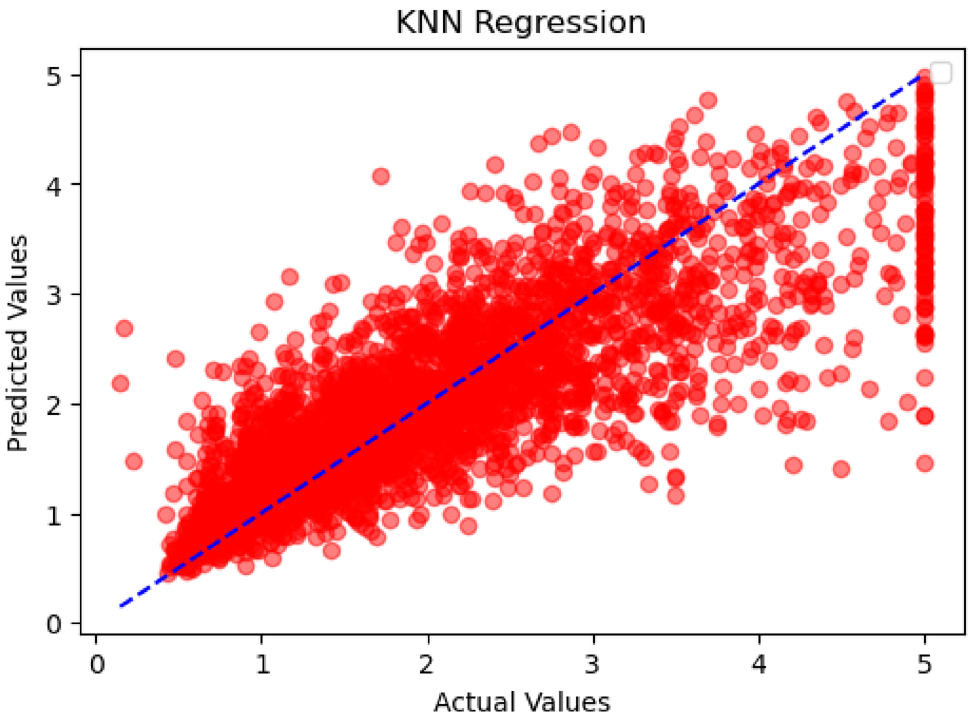
```python
print("X_train size: ", X_train.shape)
print("X_test size: ", X_test.shape)
print("y_train size: ", y_train.shape)
print("y_test size: ", y_test.shape)

# Create and train the KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)   # Mean Squared Error (MSE)
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error (MAE)
r2 = r2_score(y_test, y_pred)              # R2 SCORE

print('\n\tK-Nearest Neighbors (KNN) REGRESSION')
print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')

results['K-Nearest Neighbors (KNN) REGRESSION'] = {"MSE": mse, "MAE": mae, "R2 Score"
```

```
X_train size:  (13473, 8)
X_test size:  (3369, 8)
y_train size:  (13473,)
y_test size:  (3369,)

        K-Nearest Neighbors (KNN) REGRESSION
Mean Squared Error: 0.3872991875535743
Mean Absolute Error: 0.44192728168596024
R-squared: 0.6604499746311885
```

In [503…

```python
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, color='red', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='blue', linest
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title('KNN Regression')
plt.legend()
plt.show()
```

```
C:\Users\NIVED\AppData\Local\Temp\ipykernel_75084\1035173829.py:7: UserWarning: No art
ists with labels found to put in legend.  Note that artists whose label start with an
underscore are ignored when legend() is called with no argument.
  plt.legend()
```



In [504…

```
my_df.isnull().sum()
my_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16842 entries, 2 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      16842 non-null  float64
 1   HouseAge    16842 non-null  float64
 2   AveRooms    16842 non-null  float64
 3   AveBedrms   16842 non-null  float64
 4   Population  16842 non-null  float64
 5   AveOccup    16842 non-null  float64
 6   Latitude    16842 non-null  float64
 7   Longitude   16842 non-null  float64
 8   target      16842 non-null  float64
dtypes: float64(9)
memory usage: 1.3 MB
```

LINEAR REGRESSION

In [506…

```
from sklearn.linear_model import LinearRegression

model=LinearRegression()
#  Split the dataset into training and testing sets has been done earlier for KNN. So
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
model.fit(X_train,y_train)
```

Out[506…

```
▾ LinearRegression
LinearRegression()
```

In [507…

```
y_predict=model.predict(X_test)
y_predict
```

Out[507…

```
array([1.96737949, 3.10852942, 1.09816813, ..., 1.84760252, 3.03743516,
       3.44182787])
```

In [508…

```
Linear_reg_result_df=pd.DataFrame({
    'Actual Values': y_test,
    'Predicted Values': y_predict})

Linear_reg_result_df
```

Out[508…

	Actual Values	Predicted Values
6815	2.13200	1.967379
14560	1.90200	3.108529
6874	1.76800	1.098168
4750	1.72700	1.743760
6273	1.51800	1.525675
...	...	...
20596	0.68500	0.938791
3641	1.81800	2.438722
16446	1.75300	1.847603
13688	1.73200	3.037435
18382	5.00001	3.441828

3369 rows × 2 columns

In [509…

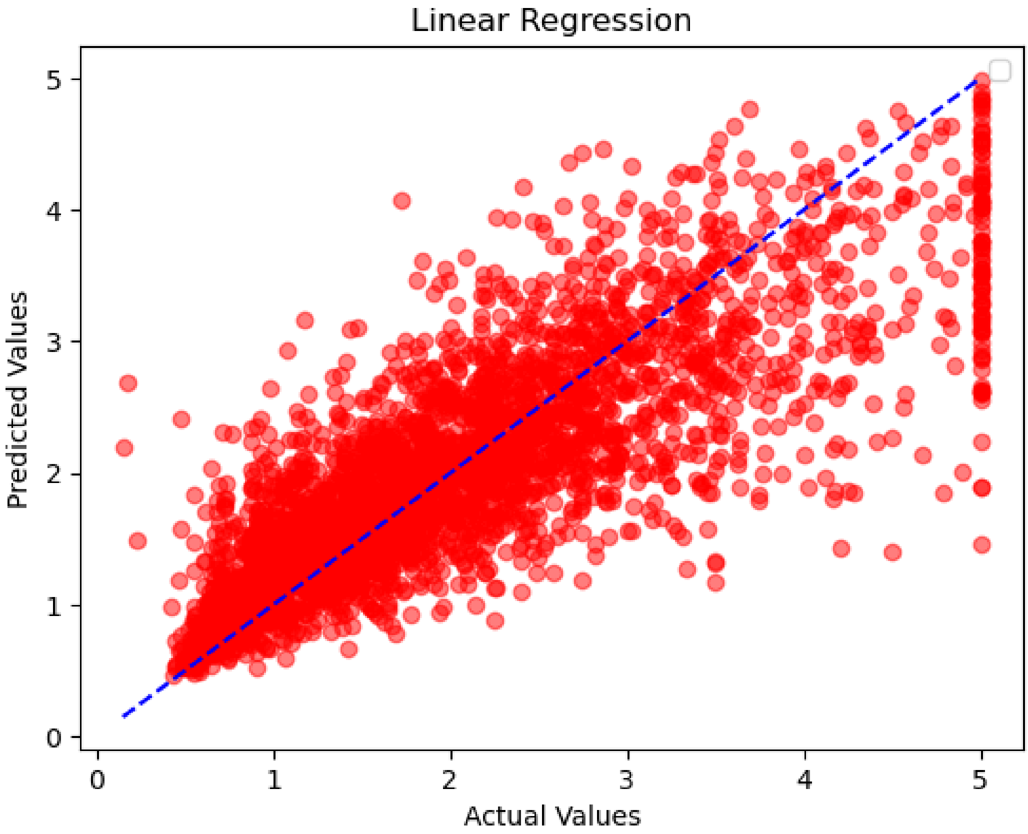
```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)   # Mean Squared Error (MSE)
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error (MAE)
r2 = r2_score(y_test, y_pred)              # R2 SCORE

print('\n\tLINEAR REGRESSION')
print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')

results['LINEAR REGRESSION'] = {"MSE": mse, "MAE": mae, "R2 Score": r2}
```

```
	LINEAR REGRESSION
Mean Squared Error: 0.3872991875535743
Mean Absolute Error: 0.44192728168596024
R-squared: 0.6604499746311885
```

In [510…

```
# Visualize the results
plt.scatter(y_test, y_pred, color='red', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='blue', linest
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title('Linear Regression')
plt.legend()
plt.show()
```

```
C:\Users\NIVED\AppData\Local\Temp\ipykernel_75084\1214985853.py:7: UserWarning: No art
ists with labels found to put in legend.  Note that artists whose label start with an
underscore are ignored when legend() is called with no argument.
  plt.legend()
```



In [ ]:

Decision Tree Regressor

In [512…

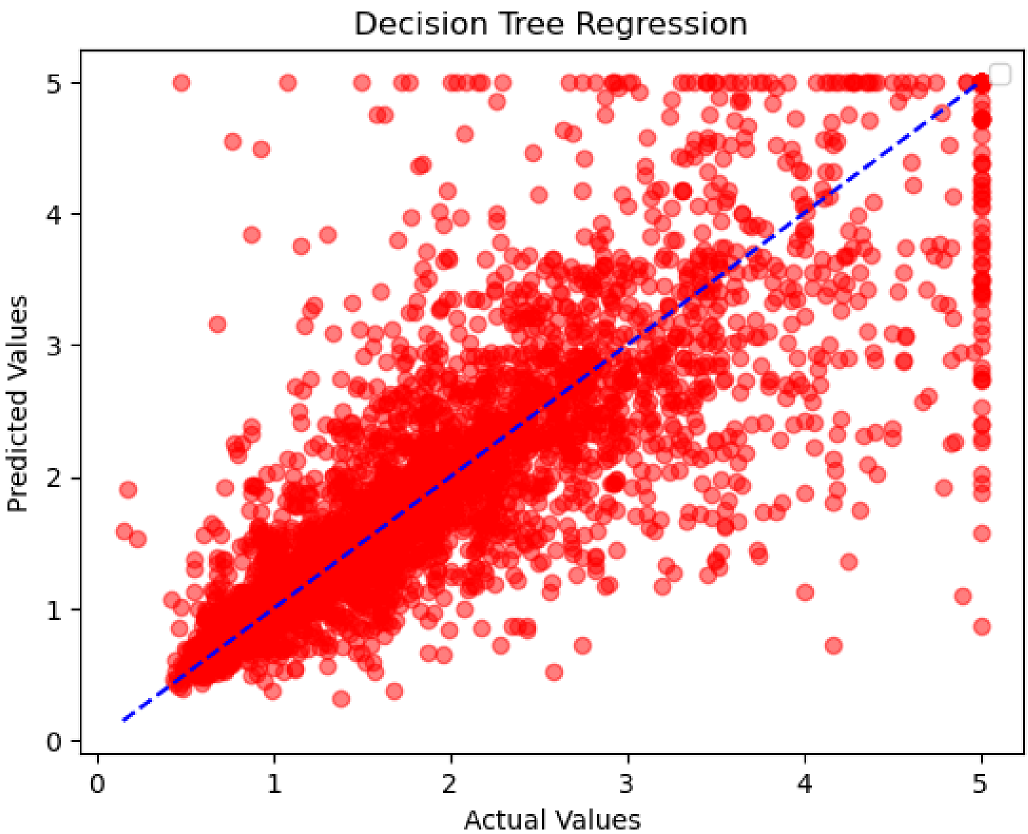
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score

# Initializing the Decision Tree Regression model
model = DecisionTreeRegressor(random_state = 0)
# Fitting the Decision Tree Regression model to the data
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRoot Mean Square Error (RMSE): ", rmse)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)   # Mean Squared Error (MSE)
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error (MAE)
r2 = r2_score(y_test, y_pred)              # R2 SCORE

print('\n\tDECISION TREE REGRESSION')
print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')

results['DECISION TREE REGRESSION'] = {"MSE": mse, "MAE": mae, "R2 Score": r2}
```

```
Root Mean Square Error (RMSE):  0.694

        DECISION TREE REGRESSION
Mean Squared Error: 0.48207709390667847
Mean Absolute Error: 0.4433684060552093
R-squared: 0.5773570027355329
```

In [513…

```python
# Visualize the results
plt.scatter(y_test, y_pred, color='red', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='blue', linest
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title('Decision Tree Regression')
plt.legend()
plt.show()
```

```
C:\Users\NIVED\AppData\Local\Temp\ipykernel_75084\3152475013.py:7: UserWarning: No art
ists with labels found to put in legend.  Note that artists whose label start with an
underscore are ignored when legend() is called with no argument.
  plt.legend()
```



Random Forest Regressor

In [515…

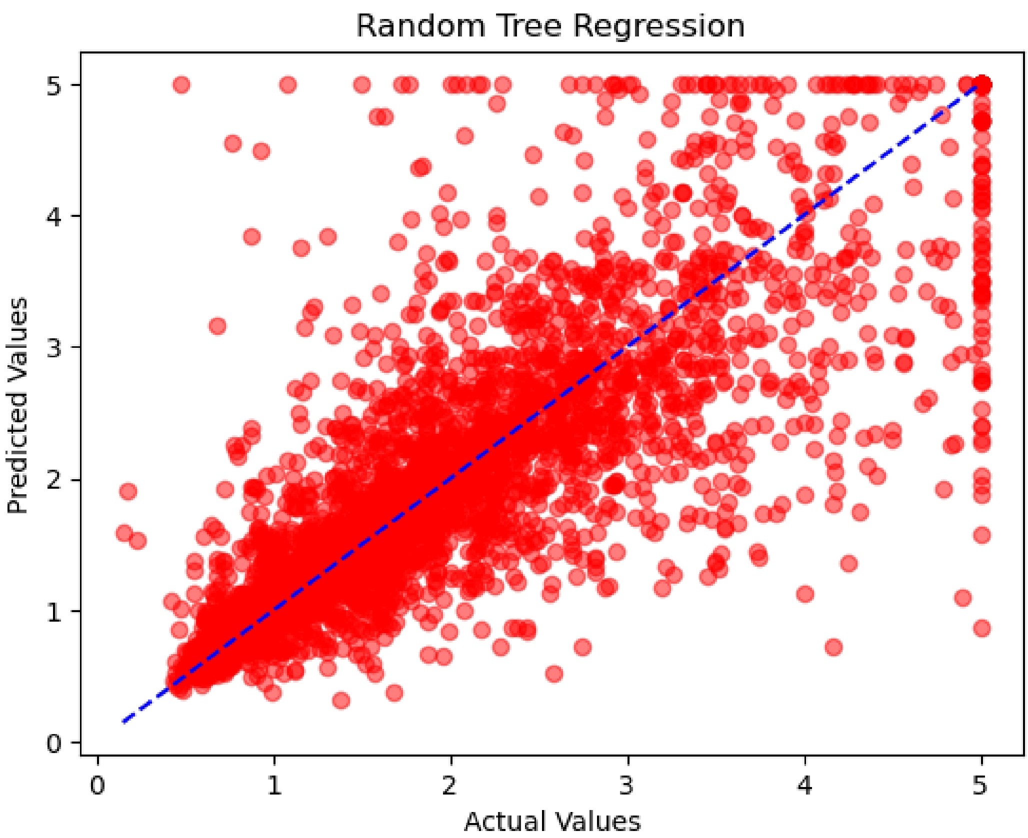
```python
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

# Make predictions
rf_y_pred = rf_model.predict(X_test)

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, rf_y_pred)), '.3f'))
print("\nRoot Mean Square Error (RMSE): ", rmse)

# Evaluate the model
```

```
mse = mean_squared_error(y_test, rf_y_pred)   # Mean Squared Error (MSE)
mae = mean_absolute_error(y_test, rf_y_pred)  # Mean Absolute Error (MAE)
r2 = r2_score(y_test, rf_y_pred)              # R2 SCORE

print('\n\tFOREST TREE REGRESSION')
print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')

results['RANDOM FOREST TREE REGRESSION'] = {"MSE": mse, "MAE": mae, "R2 Score": r2}
```

```
Root Mean Square Error (RMSE):  0.501

        FOREST TREE REGRESSION
Mean Squared Error: 0.25084216579797025
Mean Absolute Error: 0.3292813660136539
R-squared: 0.7800835465256778
```

In [516…

```
# Visualize the results
plt.scatter(y_test, y_pred, color='red', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='blue', linest
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title('Random Tree Regression')
plt.show()
```



Gradient Boosting Regressor

In [518…

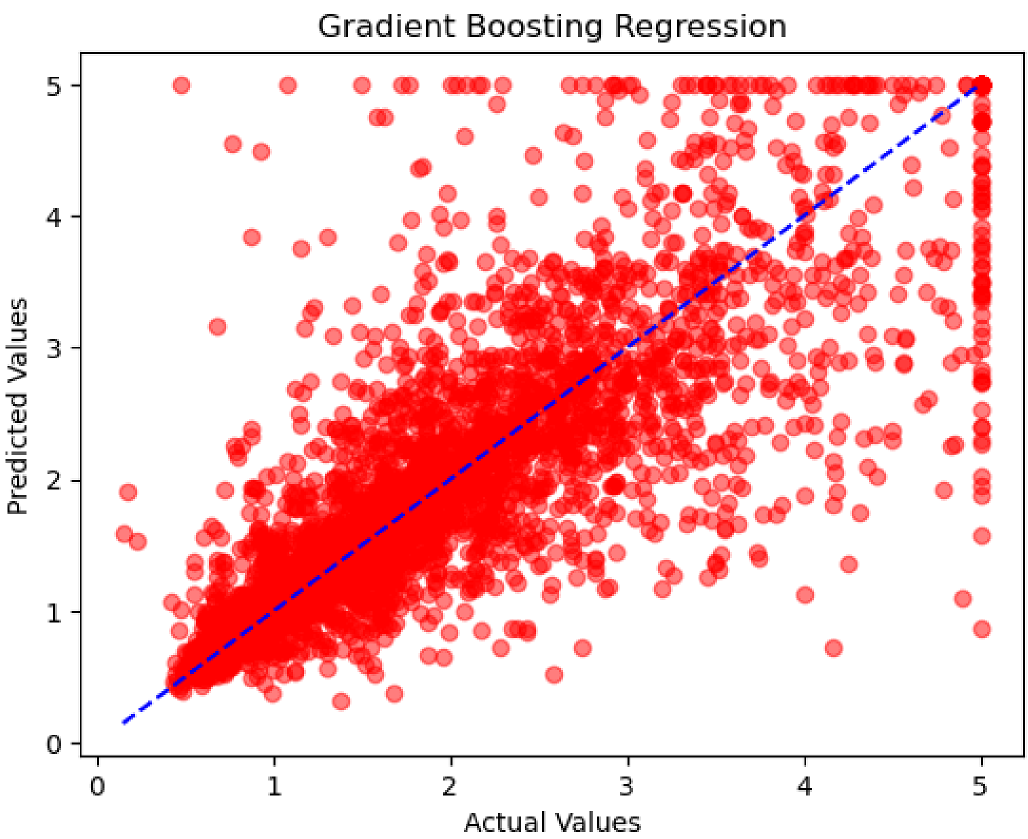
```
from sklearn.ensemble import GradientBoostingRegressor
# Create a Gradient Boosting model
gradient_boosting_model = GradientBoostingRegressor()

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

```
# Make predictions on the test set
grad_y_pred = gradient_boosting_model.predict(X_test)

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, grad_y_pred)), '.3f'))
print("\nRoot Mean Square Error (RMSE): ", rmse)

# Evaluate the model
mse = mean_squared_error(y_test, grad_y_pred)   # Mean Squared Error (MSE)
mae = mean_absolute_error(y_test, grad_y_pred)  # Mean Absolute Error (MAE)
r2 = r2_score(y_test, grad_y_pred)              # R2 SCORE

print('\n\tGRADIENT BOOSTING REGRESSION')
print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')

results['GRADIENT BOOSTING REGRESSION'] = {"MSE": mse, "MAE": mae, "R2 Score": r2}
```

```
Root Mean Square Error (RMSE):  0.533

        GRADIENT BOOSTING REGRESSION
Mean Squared Error: 0.28393981944807456
Mean Absolute Error: 0.3727859199536891
R-squared: 0.7510664210121198
```

In [519…

```
# Visualize the results
plt.scatter(y_test, y_pred, color='red', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='blue', linest
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title('Gradient Boosting Regression')
plt.show()
```



SUPPORT VECTOR MACHINE

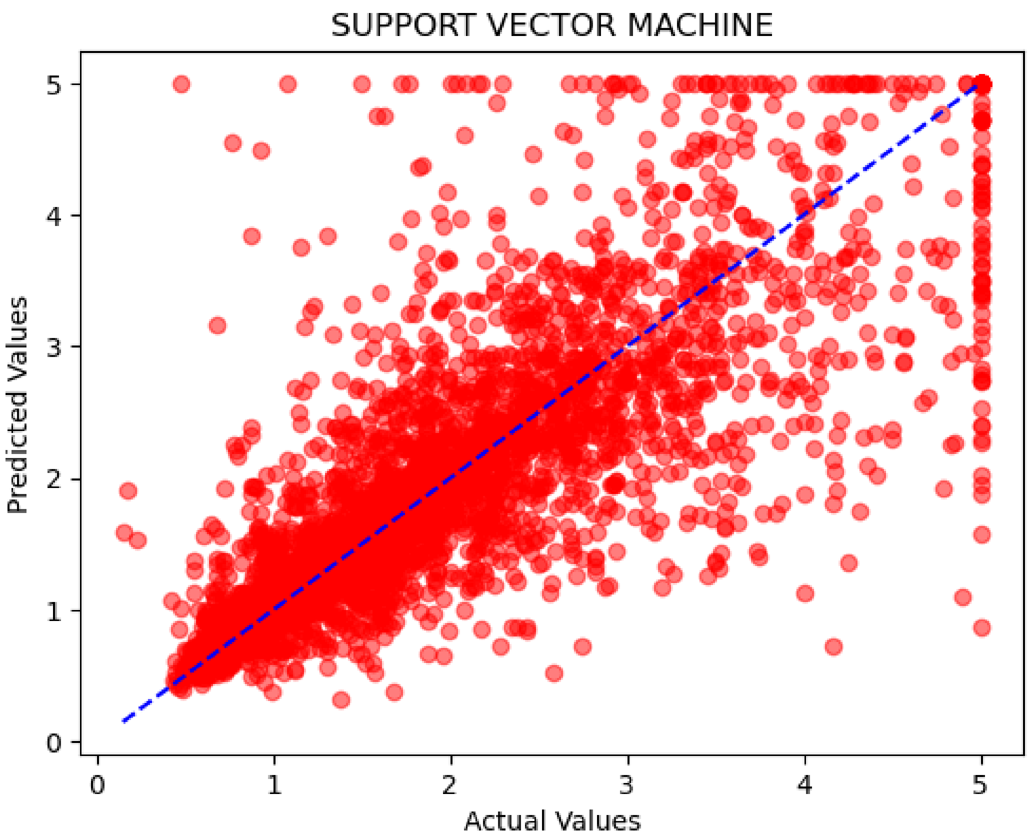
In [521…

```
from sklearn.svm import SVR
# Train SVM classifier
svm_regression_model = SVR()
svm_regression_model.fit(X_train, y_train)

# Make predictions
svm_y_pred = svm_regression_model.predict(X_test)

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, grad_y_pred)), '.3f'))
print("\nRoot Mean Square Error (RMSE): ", rmse)

# Evaluate the model
mse = mean_squared_error(y_test, grad_y_pred)   # Mean Squared Error (MSE)
mae = mean_absolute_error(y_test, grad_y_pred)  # Mean Absolute Error (MAE)
r2 = r2_score(y_test, grad_y_pred)              # R2 SCORE

print('\n\tSUPPORT VECTOR MACHINE')
print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')

results['SUPPORT VECTOR MACHINE REGRESSION'] = {"MSE": mse, "MAE": mae, "R2 Score": r
```

```
Root Mean Square Error (RMSE):  0.533

        SUPPORT VECTOR MACHINE
Mean Squared Error: 0.28393981944807456
Mean Absolute Error: 0.3727859199536891
R-squared: 0.7510664210121198
```

In [522…

```
# Visualize the results
plt.scatter(y_test, y_pred, color='red', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='blue', linest
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title('SUPPORT VECTOR MACHINE')
plt.show()
```



MODEL PERFORMANCE COMPARISON

In [524…

```
results
```

Out[524…

```
{'K-Nearest Neighbors (KNN) REGRESSION': {'MSE': 0.3872991875535743,
  'MAE': 0.44192728168596024,
  'R2 Score': 0.6604499746311885},
 'LINEAR REGRESSION': {'MSE': 0.3872991875535743,
  'MAE': 0.44192728168596024,
  'R2 Score': 0.6604499746311885},
 'DECISION TREE REGRESSION': {'MSE': 0.48207709390667847,
  'MAE': 0.4433684060552093,
  'R2 Score': 0.5773570027355329},
 'RANDOM FOREST TREE REGRESSION': {'MSE': 0.25084216579797025,
  'MAE': 0.3292813660136539,
  'R2 Score': 0.7800835465256778},
 'GRADIENT BOOSTING REGRESSION': {'MSE': 0.28393981944807456,
  'MAE': 0.3727859199536891,
  'R2 Score': 0.7510664210121198},
 'SUPPORT VECTOR MACHINE REGRESSION': {'MSE': 0.28393981944807456,
  'MAE': 0.3727859199536891,
  'R2 Score': 0.7510664210121198}}
```

In [525…

```
df_results = pd.DataFrame(results)
```

In [526…

```
df_results
```

Out[526...

	K-Nearest Neighbors (KNN) REGRESSION	LINEAR REGRESSION	DECISION TREE REGRESSION	RANDOM FOREST TREE REGRESSION	GRADIENT BOOSTING REGRESSION	SUPPORT VECTOR MACHINE REGRESSION
MSE	0.387299	0.387299	0.482077	0.250842	0.283940	0.283940
MAE	0.441927	0.441927	0.443368	0.329281	0.372786	0.372786
R2 Score	0.660450	0.660450	0.577357	0.780084	0.751066	0.751066

In [527...

```
df_results_transpose = pd.DataFrame(results).T

# or we can use transpose()
df_results_transpose = df_results.transpose()
```

In [528...

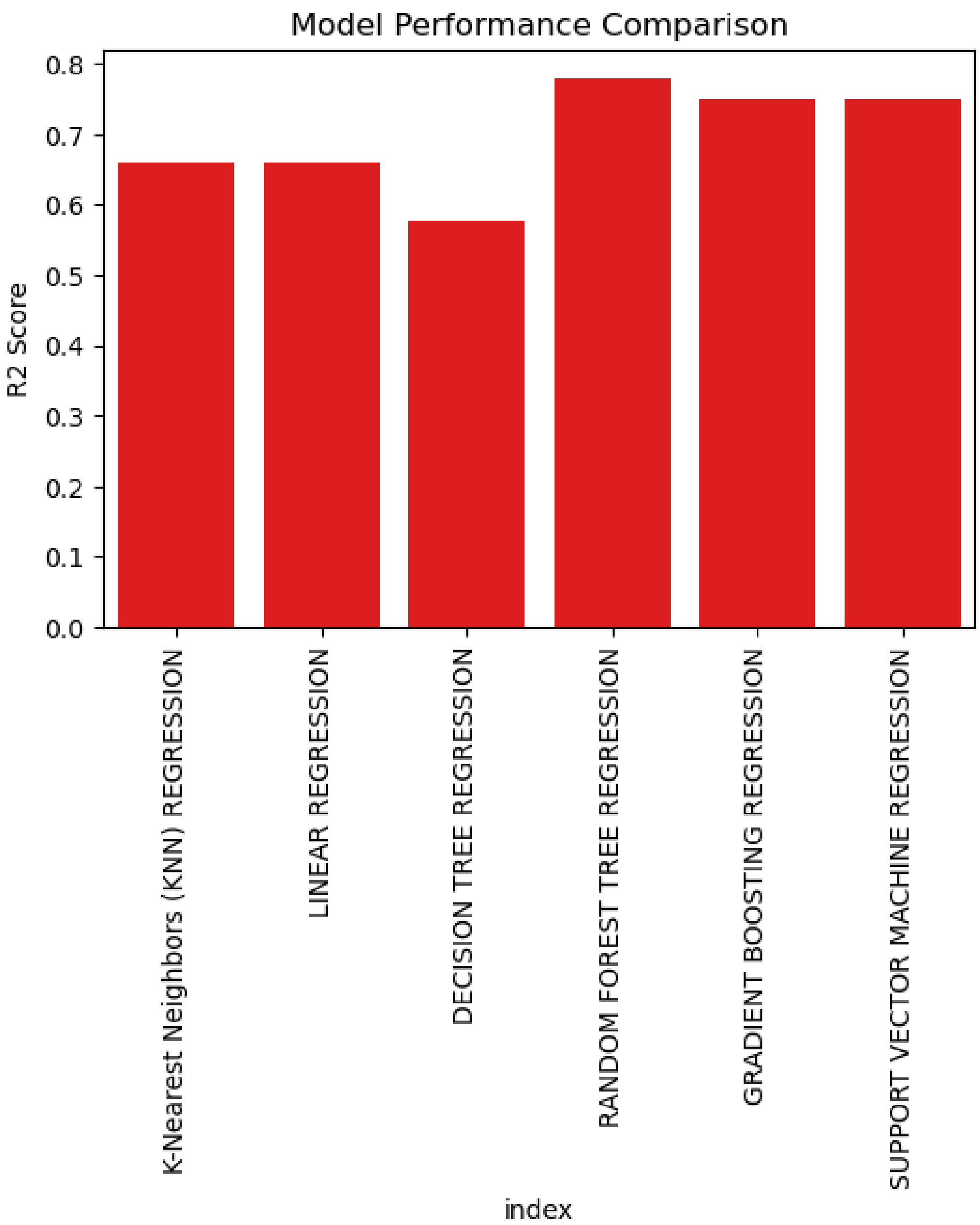
```
df_results_transpose
```

Out[528...

	MSE	MAE	R2 Score
K-Nearest Neighbors (KNN) REGRESSION	0.387299	0.441927	0.660450
LINEAR REGRESSION	0.387299	0.441927	0.660450
DECISION TREE REGRESSION	0.482077	0.443368	0.577357
RANDOM FOREST TREE REGRESSION	0.250842	0.329281	0.780084
GRADIENT BOOSTING REGRESSION	0.283940	0.372786	0.751066
SUPPORT VECTOR MACHINE REGRESSION	0.283940	0.372786	0.751066

In [529...

```
# Visualizing Model Performance plotting for R2_score
plt.figure(figsize=(6, 4))
sns.barplot(data=df_results_transpose.reset_index(), x='index', y='R2 Score', color='
plt.xticks(rotation=90)
plt.title("Model Performance Comparison")
plt.show()
```



FROM ABOVE BARPLOT, WE CAN CLEARLY CONCLUDE THAT "RANDOM FOREST TREE REGRESSION" IS THE BEST MODEL

PREDICTION ACCURACY IS 78%

WORST PERFORMNG ALGORITHM IS - DECISION TREE REGRESSION*

*AS ITS PREDICTION ACCURACY IS THE LOWEST OF ALL - ONLY 57%

In [ ]: