- Name: Dipak Shingade
- Roll No.:21102B0029
- SEM-7 ML Lab5 GitHub

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

headers = ['ID', 'diagnosis', 'radius', 'texture', 'perimeter',
'area', 'smoothness', 'compactness', 'concavity', 'concave_points',
'symmetry', 'fractal_dimension',
           'radius2', 'texture2', 'perimeter2', 'area2',
'smoothness2', 'compactness2', 'concavity2', 'concave_points2',
'symmetry2', 'fractal_dimension2',
           'radius3', 'texture3', 'perimeter3', 'area3',
'smoothness3', 'compactness3', 'concavity3', 'concave_points3',
'symmetry3', 'fractal_dimension3']
data = pd.read_csv("/content/sample_data/wdbc.data", delimiter=",",
names = headers)

data

{"type":"dataframe","variable_name":"data"}

data.shape

(569, 32)

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  ------
 0   ID                  569 non-null    int64
 1   diagnosis           569 non-null    object
 2   radius              569 non-null    float64
 3   texture             569 non-null    float64
 4   perimeter           569 non-null    float64
 5   area                569 non-null    float64
 6   smoothness          569 non-null    float64
 7   compactness         569 non-null    float64
 8   concavity           569 non-null    float64
 9   concave_points      569 non-null    float64
 10  symmetry            569 non-null    float64
 11  fractal_dimension   569 non-null    float64
 12  radius2             569 non-null    float64
```

```
 13  texture2              569 non-null    float64
 14  perimeter2            569 non-null    float64
 15  area2                 569 non-null    float64
 16  smoothness2           569 non-null    float64
 17  compactness2          569 non-null    float64
 18  concavity2            569 non-null    float64
 19  concave_points2       569 non-null    float64
 20  symmetry2             569 non-null    float64
 21  fractal_dimension2    569 non-null    float64
 22  radius3               569 non-null    float64
 23  texture3              569 non-null    float64
 24  perimeter3            569 non-null    float64
 25  area3                 569 non-null    float64
 26  smoothness3           569 non-null    float64
 27  compactness3          569 non-null    float64
 28  concavity3            569 non-null    float64
 29  concave_points3       569 non-null    float64
 30  symmetry3             569 non-null    float64
 31  fractal_dimension3    569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in
/usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
from sklearn.model_selection import train_test_split

X = data.drop(['diagnosis','ID'], axis=1)
# Map 'B' to 0 and 'M' to 1
y = data['diagnosis'].map({'B': 0, 'M': 1})


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

RANDOM FOREST

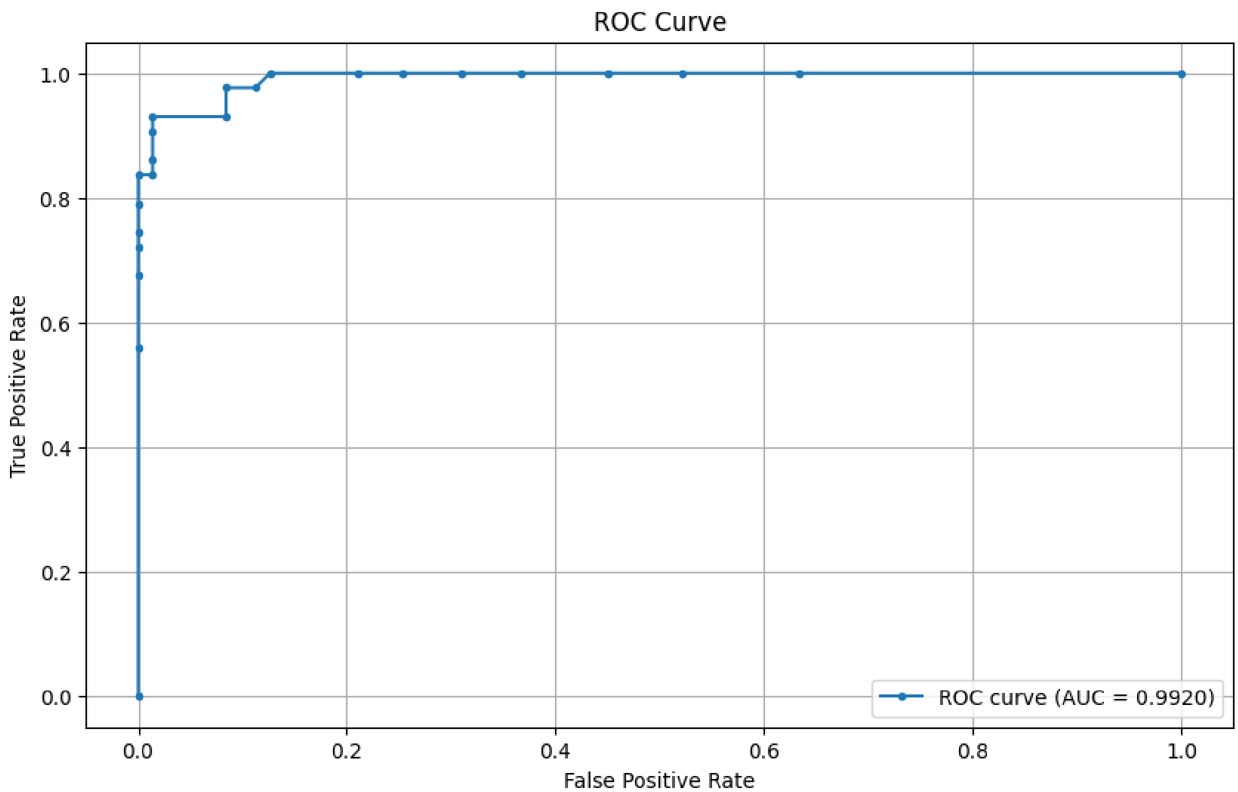
```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier()
```

```
rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, precision_score, recall_score, f1_score, roc_curve,
auc

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label=1)
recall = recall_score(y_test, y_pred, pos_label=1)
f1 = f1_score(y_test, y_pred, pos_label=1)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-score: {f1}')

print(classification_report(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
Precision: 0.975609756097561
Recall: 0.9302325581395349
F1-score: 0.9523809523809523
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        71
           1       0.98      0.93      0.95        43

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

[[70  1]
 [ 3 40]]
```

```
y_probs = rf_classifier.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_probs, pos_label=1)

auc_value = auc(fpr, tpr)
print(f'AUC: {auc_value:.4f}')

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, marker='.', label=f'ROC curve (AUC =
{auc_value:.4f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
```

```
plt.legend()
plt.grid()
plt.show()

AUC: 0.9920
```



XGBOOST

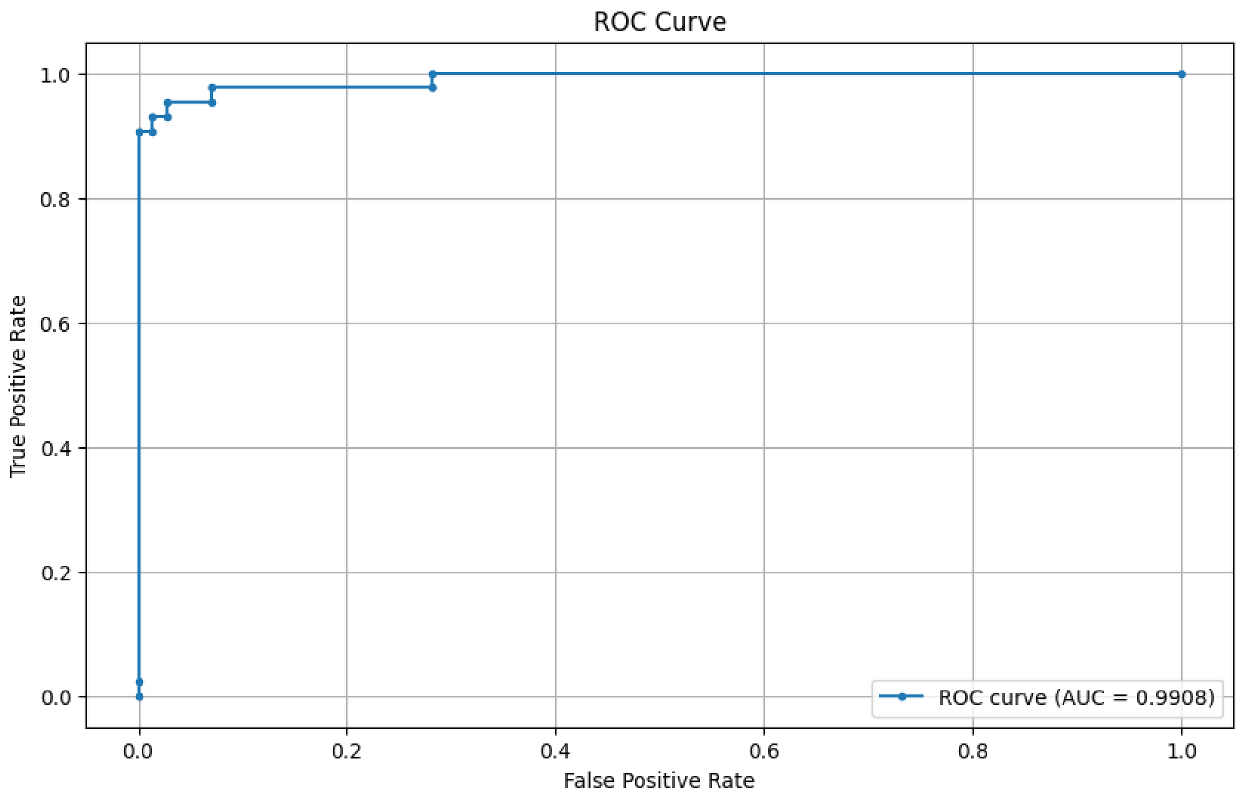
```
from xgboost import XGBClassifier

# Initialize XGBoost Classifier
xgb_classifier = XGBClassifier()

# Fit the model
xgb_classifier.fit(X_train, y_train)

# Make predictions
y_pred = xgb_classifier.predict(X_test)

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, precision_score, recall_score, f1_score, roc_curve,
auc

accuracy = accuracy_score(y_test, y_pred)
```

```
precision = precision_score(y_test, y_pred, pos_label=1)
recall = recall_score(y_test, y_pred, pos_label=1)
f1 = f1_score(y_test, y_pred, pos_label=1)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-score: {f1}')

print(classification_report(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.956140350877193
Precision: 0.9523809523809523
Recall: 0.9302325581395349
F1-score: 0.9411764705882353
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        71
           1       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

[[69  2]
 [ 3 40]]
```

```
y_probs = xgb_classifier.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_probs, pos_label=1)

auc_value = auc(fpr, tpr)
print(f'AUC: {auc_value:.4f}')

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, marker='.', label=f'ROC curve (AUC =
{auc_value:.4f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()
```

```
AUC: 0.9908
```



#Regression

```
data = pd.read_csv("/content/sample_data/housing.csv")
data
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 20640,\n
\"fields\": [\n   {\n   \"column\": \"longitude\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
2.0035317235025882,\n   \"min\": -124.35,\n   \"max\": -
114.31,\n   \"num_unique_values\": 844,\n   \"samples\": [\n
-118.63,\n   -119.86,\n   -121.26\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"latitude\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
2.1359523974571153,\n   \"min\": 32.54,\n   \"max\": 41.95,\
n   \"num_unique_values\": 862,\n   \"samples\": [\n
33.7,\n   34.41,\n   38.24\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"housing_median_age\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
12.58555761211165,\n   \"min\": 1.0,\n   \"max\": 52.0,\n
\"num_unique_values\": 52,\n   \"samples\": [\n   35.0,\n
25.0,\n   7.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"total_rooms\",\n   \"properties\": {\n   \"dtype\":

\"number\",\n   \"std\": 2181.615251582795,\n   \"min\":
2.0,\n   \"max\": 39320.0,\n   \"num_unique_values\": 5926,\
n   \"samples\": [\n   699.0,\n   1544.0,\n
3966.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"total_bedrooms\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 421.3850700740323,\n   \"min\":
1.0,\n   \"max\": 6445.0,\n   \"num_unique_values\": 1923,\n
\"samples\": [\n1538.0,\n 1298.0,\n
1578.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"population\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1132.462121765341,\n   \"min\":
3.0,\n   \"max\": 35682.0,\n   \"num_unique_values\": 3888,\
n   \"samples\": [\n   4169.0,\n   636.0,\n
3367.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"households\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 382.32975283161073,\n   \"min\":
1.0,\n   \"max\": 6082.0,\n   \"num_unique_values\": 1815,\n
\"samples\": [\n   21.0,\n   750.0,\n   1447.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"median_income\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1.8998217179452688,\n   \"min\": 0.4999,\n   \"max\":
15.0001,\n   \"num_unique_values\": 12928,\n   \"samples\":
[\n   5.0286,\n   2.0433,\n   6.1228\n],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"median_house_value\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
115395.61587441387,\n   \"min\": 14999.0,\n   \"max\":
500001.0,\n   \"num_unique_values\": 3842,\n   \"samples\":
[\n   194300.0,\n   379000.0,\n   230100.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"ocean_proximity\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"<1H
OCEAN\",\n   \"ISLAND\",\n   \"INLAND\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"data"}

```
data.shape
```

```
(20640, 10)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
```

```
 #   Column              Non-Null Count  Dtype
---  -------             --------------
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```python
data.isnull().sum()
```

```python
data.dropna(inplace=True)
```

```python
data.isnull().sum()
```

```python
data.reset_index(inplace=True,drop=True)
```

```python
data['ocean_proximity'].value_counts()
```

```
ocean_proximity
<1H OCEAN     9034
```

```
INLAND         6496
NEAR OCEAN     2628
NEAR BAY       2270
ISLAND            5
Name: count, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
data['ocean_proximity']=le.fit_transform(data['ocean_proximity'])
```

```
data["rooms_per_household"] = data["total_rooms"]/data["households"]
data["bedrooms_per_room"] = data["total_bedrooms"]/data["total_rooms"]
data["population_per_household"]=data["population"]/data["households"]
```

```
data.corr()
```


```
{"summary":"{\n \"name\": \"data\",\n  \"rows\": 13,\n  \"fields\":
[\n    {\n      \"column\": \"longitude\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.4064425796486295,\n
\"min\": -0.9246161131160101,\n        \"max\": 1.0,\n
\"num_unique_values\": 13,\n        \"samples\": [\n
0.09265683306977554,\n          -0.28953010139097807,\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"latitude\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.4047506201096927,\n        \"min\": -0.9246161131160101,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          -0.11381506848357399,\n
0.2008010727260355,\n          -0.9246161131160101\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"housing_median_age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.3533184865983313,\n        \"min\": -0.36062829984244227,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          0.13608923857356492,\n
0.11233014464562725,\n          -0.10935654863027307\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_rooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.474090511200691,\n        \"min\": -0.36062829984244227,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          -0.18790000413461336,\n          -
0.015363166414694703,\n          0.0454801674218395\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_bedrooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.47841935648102935,\n        \"min\": -0.32045104175060396,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          0.08423813762384522,\n          -
0.014767943833213214,\n          0.06960802175408133\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"population\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.4678296793254769,\n        \"min\": -0.2957872971044803,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          0.03531932605444879,\n          -
0.06962966726072072,\n          0.10027030094083503\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"households\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.48289530521105845,\n        \"min\": -0.30276797344891637,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          0.06508685650819257,\n          -
0.018250608808529776,\n          0.056512772430637834\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"median_income\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.3946564083548571,\n        \"min\": -0.6156606088731494,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          -0.6156606088731494,\n          -
0.014678593813623618,\n          -0.015550150379729375\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"median_house_value\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.33973304362366785,\n        \"min\": -0.25588014941949866,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          -0.25588014941949866,\n
0.08048786002048676,\n          -0.04539821933443104\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ocean_proximity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.2997284541302523,\n        \"min\": -0.28953010139097807,\n
\"max\": 1.0,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          0.002106848712775119,\n          1.0,\n
-0.28953010139097807\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rooms_per_household\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.32868946618396233,\n        \"min\": -
0.41695223734049447,\n        \"max\": 1.0,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          -
0.41695223734049447,\n          -0.0026413047510256043,\n          -
0.027306806809850443\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"bedrooms_per_room\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3768344965853683,\n        \"min\": -
0.6156606088731494,\n\"max\": 1.0,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          1.0,\n
0.002106848712775119,\n          0.09265683306977554\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

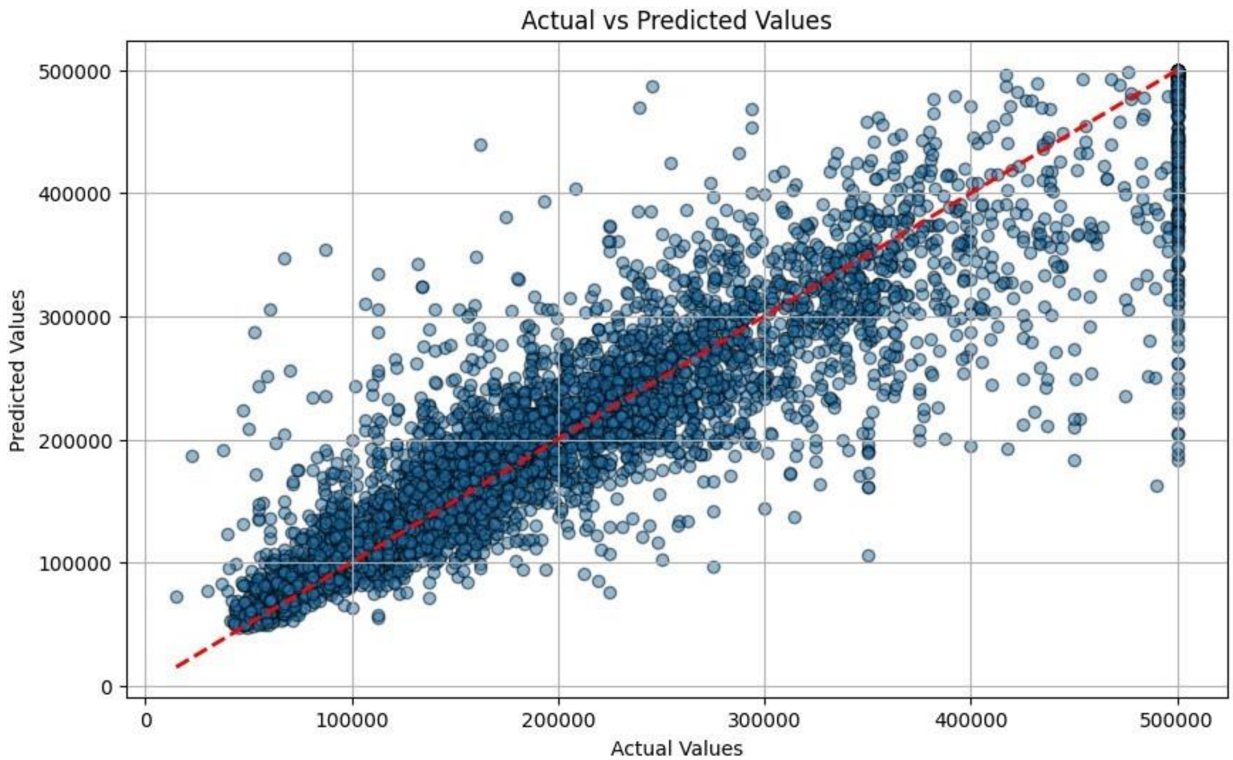
```
n    },\n    {\n      \"column\": \"population per household\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.27877451835446043,\n        \"min\": -0.02835486367066539,\n
\"max\": 1.0,\n        \"num unique values\": 13,\n
\"samples\": [\n          0.00293785249526524,\n          -
0.009151877026194142,\n          0.002303875515041799\n        ],\n
\"semantic type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```



```
x_data = data.drop(["median_house_value"], axis = 1).values
y_data = data["median_house_value"].values

from sklearn.model_selection import train_test_split
x_train, x_test, y_train , y_test = train_test_split(x_data,y_data,
test_size=0.25 , random_state=42)
```

RANDOM FOREST

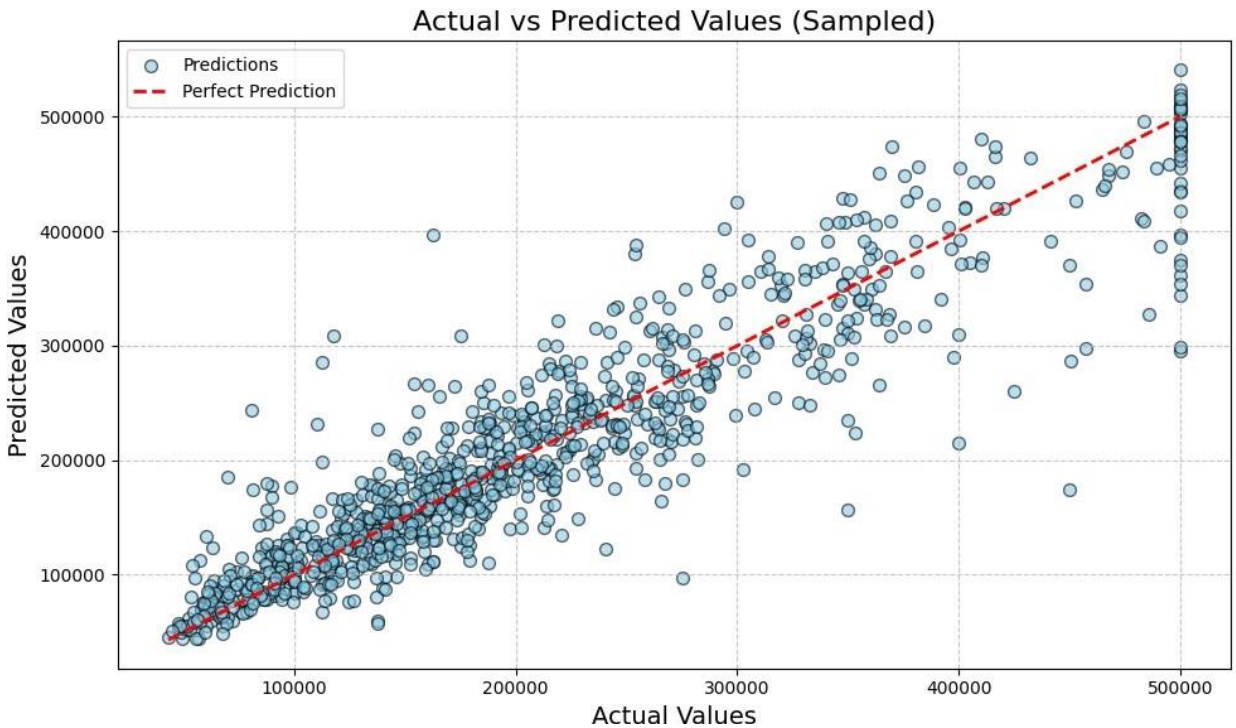
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

rf_regressor = RandomForestRegressor()

rf_regressor.fit(x_train, y_train)

y_pred = rf_regressor.predict(x_test)

from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_test, y_pred)
print(f'R-squared: {r2}')

Mean Squared Error: 2631339567.8051205
R-squared: 0.8055012570233084

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5, edgecolor='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.grid(True)
plt.show()
```



XGBOOST

```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize XGBoost Regressor
xgb_regressor = XGBRegressor()

# Fit the model
xgb_regressor.fit(x_train, y_train)

# Make predictions
y_pred = xgb_regressor.predict(x_test)

from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y_test, y_pred)
print(f'R-squared: {r2}')

Mean Squared Error: 2161493666.7455506
R-squared: 0.8402305022590588
```

```
# Randomly sample 1000 points or fewer if your data has fewer points
sample_size = min(1000, len(y_test))
indices = np.random.choice(len(y_test), size=sample_size,
replace=False)

# Sampled data
y_test_sample = y_test[indices]
y_pred_sample = y_pred[indices]

# Create the plot
plt.figure(figsize=(10, 6))

# Scatter plot
plt.scatter(y_test_sample, y_pred_sample, alpha=0.6, edgecolor='k',
s=50, color='skyblue', label='Predictions')

# Line of perfect prediction
plt.plot([y_test_sample.min(), y_test_sample.max()],
[y_test_sample.min(), y_test_sample.max()], 'r--', lw=2,
label='Perfect Prediction')

# Adding labels and title
plt.xlabel('Actual Values', fontsize=14)
plt.ylabel('Predicted Values', fontsize=14)
plt.title('Actual vs Predicted Values (Sampled)', fontsize=16)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)

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


Actual vs Predicted Values (Sampled)
Predictions
Perfect Prediction
Predicted Values
Actual Values
500000
400000
300000
200000
100000
100000
200000
300000
400000
500000