WorkFlow

1. House Price dataset
2. Data preprocessing
3. Data Analysis
4. Train and Test Spitting
5. XGBoost Regression Algorithm
6. Evaluation (By Testing Dataset)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

```
# importing Boston House Price Dataset
house_price_dataset = sklearn.datasets.fetch_california_housing()
```

```
# printing the california house price dataset
print(house_price_dataset)
```

```
ame': None, 'target_names': ['MedHouseVal'], 'feature_names': ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup
```

```
# Loading the dataset to pandas dataframe
house_price_dataframe = pd.DataFrame(house_price_dataset.data)
house_price_dataframe.head()
```

	0	1	2	3	4	5	6	7
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

Next steps: Generate code with `house_price_dataframe` View recommended plots New interactive sheet

Suggested code may be subject to a license | showMiko/Projects-Hacktober

```
# if we use "house_price_dataframe = pd.DataFrame(house_price_dataset.data)" we cant get the columns names to get the name use
house_price_dataframe = pd.DataFrame(house_price_dataset.data, columns = house_price_dataset.feature_names)
print(house_price_dataframe)
```

```
       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
```

```
0        -122.23
1        -122.22
2        -122.24
3        -122.25
4        -122.25
...          ...
20635    -121.09
20636    -121.21
20637    -121.22
20638    -121.32
20639    -121.24

[20640 rows x 8 columns]
```

```
# here price of the column as not included bcz we just included the features name to include price just import the target column
house_price_dataframe['price'] = house_price_dataset.target
house_price_dataframe.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	price
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Next steps: Generate code with house_price_dataframe | View recommended plots | New interactive sheet

```
# checking the numbers of rows and columns in our dataset
house_price_dataframe.shape
```

```
(20640, 9)
```

```
# handling missing values
house_price_dataframe.isnull().sum()
```

	0
MedInc	0
HouseAge	0
AveRooms	0
AveBedrms	0
Population	0
AveOccup	0
Latitude	0
Longitude	0
price	0

dtype: int64

Stasticial Measures of the Dataset

1. Count - No of data points we have (rows).
2. Mean - value for the respective column(Average).
3. std - standard deviation.

```
# stasticial measures of the dataset
house_price_dataframe.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	price
count	20640.000000	20640.000000	20640.000000	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	35.631861	-119.569704	2.068558
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532	1.153956
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000	0.149990
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000	1.196000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000	1.797000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000	2.647250
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000	5.000010

Check for the correlation of the dataset

1. +ve - correlation. (one inc other also inc.)
2. -ve - correlation. (one dec other also dec.)

Suggested code may be subject to a license | ajaydurgam85/HousePrice_Prediction_2

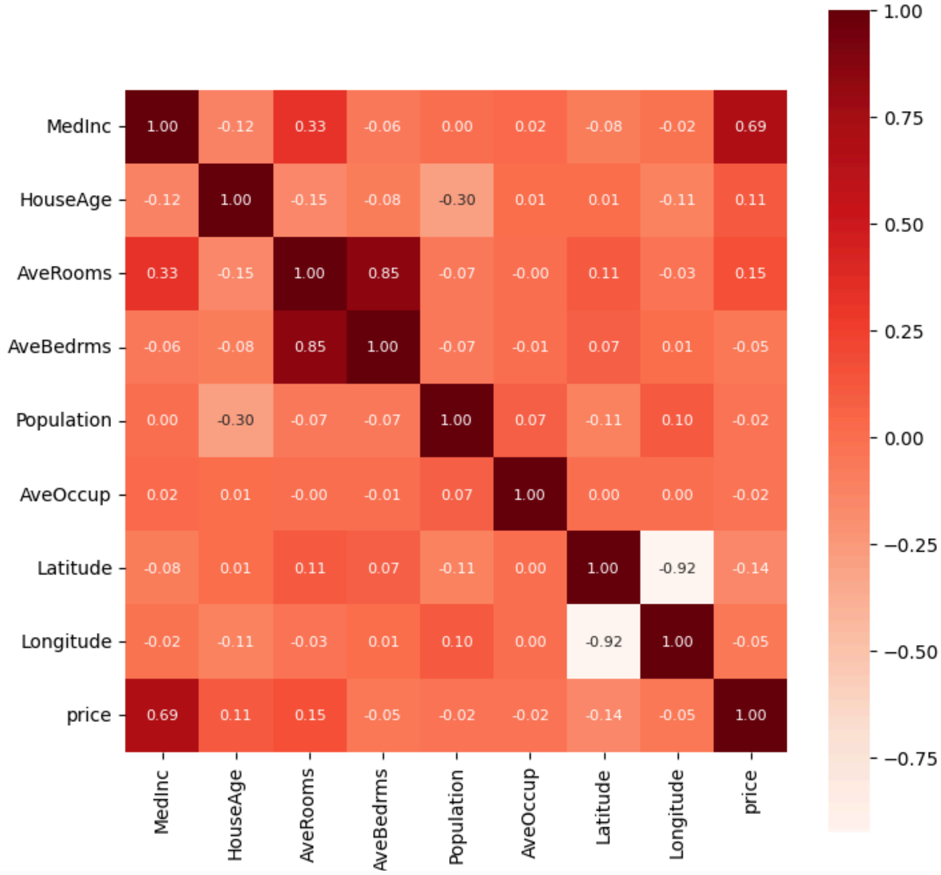
```
correlation = house_price_dataframe.corr()
```

commands explanation

1. fmt='.1f' = 0.0, 0.1 (if fmt='.2f' = 0.00, 0.11).
2. square=True plot in square shape.
3. cbar =True for a bar in the right side.
4. annot = to print the numbers inside the box.
5. annot_kws = for sizes of the numbers inside the box.
6. cmap for color of the map

```
plt.figure(figsize = (8,8))
sns.heatmap(correlation, cbar =True, square=True, fmt='.2f', annot = True, annot_kws ={'size':8}, cmap='Reds' )
```

<Axes: >



```
# spliting the features and Target
x = house_price_dataframe.drop(['price'], axis=1) # axis 1 for representing the columns
y = house_price_dataframe['price']
```

```
print(x)
print(y)
```

```
       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
0        -122.23
1        -122.22
2        -122.24
3        -122.25
4        -122.25
...          ...
20635    -121.09
20636    -121.21
20637    -121.22
20638    -121.32
20639    -121.24

[20640 rows x 8 columns]
0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         ...
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: price, Length: 20640, dtype: float64
```

```
# splitting the dataset for testing and training
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
print(x.shape, x_train.shape, x_test.shape)
```

```
(20640, 8) (16512, 8) (4128, 8)
```

Model Training

XGBoost Regressor

1. It is an Ensemble model
2. this basicially incorporate the one or two model to achieve the result.

```
model = XGBRegressor()
```

```
model.fit(x_train, y_train)
```

```
                                  XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```
train_data_prediction = model.predict(x_train)
```

```
print(train_data_prediction)
```

```
[0.5523039 3.0850039 0.5835302 ... 1.9204227 1.952873  0.6768683]
```

```
# R squared error
error_score1 = metrics.r2_score(y_train, train_data_prediction)
print("R squared error : ", error_score1)

# Mean square error
error_score2 = metrics.mean_squared_error(y_train, train_data_prediction)
print("Mean square error : ", error_score2)
```

```
R squared error :  0.943650140819218
Mean square error :  0.0748112971690747
```

```
# let visualy see the results in graphy using plt(Matplotlib)
plt.scatter(y_train, train_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Price vs Predicted Price")
plt.show()
```

Evaluation discuss

Mean square error : 0.0748112971690747

It does not higher **(like 5 or 6)** *as it is less than 1, means that our model is performing good on training dataset.*

The result we got is only for **Training Data**.

```
test_data_prediction = model.predict(x_test)
```

```
print(test_data_prediction)
```

```
[2.8649795  1.790346   0.92074925 ... 1.5385513  0.92647874 2.043316  ]
```

```
# R squared error
error_score1 = metrics.r2_score(y_test, test_data_prediction)
print("R squared error : ", error_score1)

# Mean square error
error_score2 = metrics.mean_squared_error(y_test, test_data_prediction)
print("Mean square error : ", error_score2)
```

```
R squared error :  0.8338000331788725
Mean square error :  0.22387540906811954
```

Evaluation discuss

Mean square error : 0.22387540906811954

It does not higher **(like 5 or 6)** *as it is less than 1, means that our model is performing good on training dataset.*

The result we got is only for **TEST Data**.

Suggested code may be subject to a license | PSaiRahul/House-Price-Prediction

```
# let visualy see the results in graphy using plt(Matplotlib)
plt.scatter(y_test, test_data_prediction)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual Price vs Predicted Price")
plt.show()
```