Importing the dependencies

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

Check out the data

```
house_price_dataset = pd.read_csv("/content/drive/MyDrive/USA_Housing.csv")
```

```
house_price_dataset.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Ad
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Fer 674\nLaurabu
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Suite 079 Kathleen
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Eli Stravenue\nDanie WI 0
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nF
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond AE

New section

```
house_price_dataset.shape
```

```
(5000, 7)
```

```
house_price_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

```
house_price_dataset.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.00000
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.23207
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.53117
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.59386
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.97577
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.23266
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.47121
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.46906

```
house_price_dataset.columns
```

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

```
#check for missing values
house_price_dataset.isnull().sum()
```

```
Avg. Area Income               0
Avg. Area House Age            0
Avg. Area Number of Rooms      0
Avg. Area Number of Bedrooms   0
Area Population                0
Price                          0
Address                        0
dtype: int64
```

Exploratory Data Analysis (EDA)

```
sns.pairplot(house_price_dataset)
```

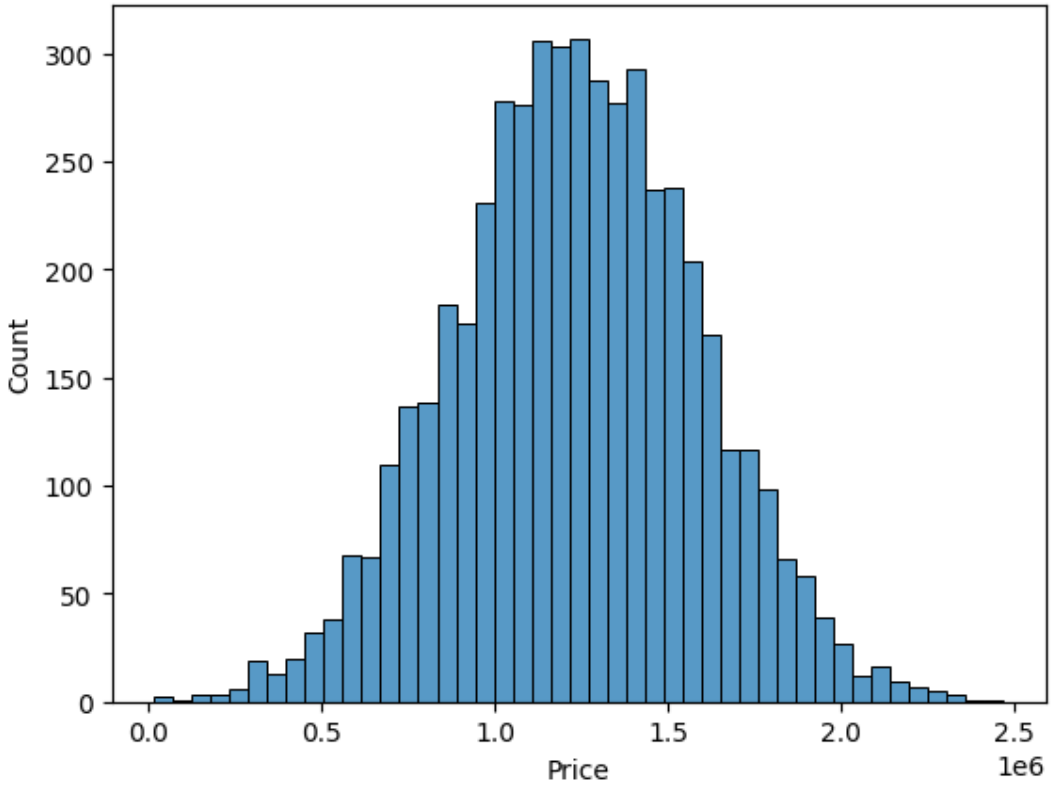
```
<seaborn.axisgrid.PairGrid at 0x7bf665140f70>
```

```
house_price_dataset.hvplot.hist(by='Price',subplots=False,width=1000)
```

```
house_price_dataset.hvplot.hist("Price")
```

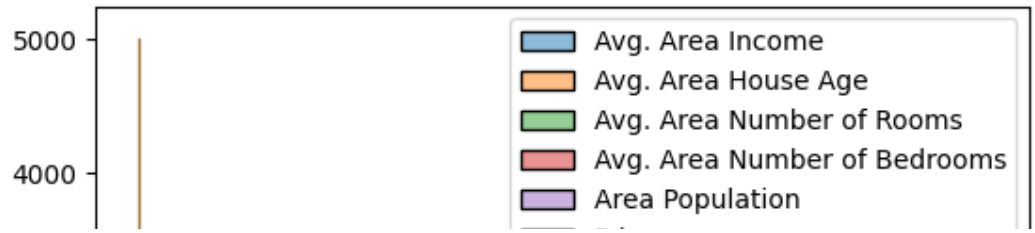
```
sns.histplot(data=house_price_dataset.Price)
```

```
<Axes: xlabel='Price', ylabel='Count'>
```




```
#sns.set(rc={'figure.figsize':(5,3)})
sns.histplot(house_price_dataset)
```

```
<Axes: ylabel='Count'>
```




```
house_price_dataset.Price
```

```
0       1.059034e+06
1       1.505891e+06
2       1.058988e+06
3       1.260617e+06
4       6.309435e+05
            ...
4995    1.060194e+06
4996    1.482618e+06
4997    1.030730e+06
4998    1.198657e+06
4999    1.298950e+06
Name: Price, Length: 5000, dtype: float64
```

1e6

```
sns.heatmap(house_price_dataset.corr(), annot=True)
```

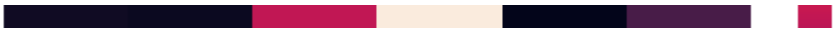
```
<ipython-input-15-7098d4588721>:1: FutureWarning: The default value of numeric_only in Data
  sns.heatmap(house_price_dataset.corr(), annot=True)
```

Training a Linear Regression Model

X and Y arrays

```
X=house_price_dataset[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', '
y=house_price_dataset['Price']
```

Train Test Split



```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train, y_tes=train_test_split(X,y,test_size=0.3,random_state=42)
```


```
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def cross_val(model):
  pred=cross_val_score(model,X,y,cv=10)
  return pred.mean()
def print_evaluate(true, predicted):
  mae=metrics.mean_absolute_error(true, predicted)
  mse=metrics.mean_squared_error(true, predicted)
  rmse=np.sqrt(metrics.mean_squared_error(true, predicted))
  r2_square=metrics.r2_score(true, predicted)
  print('MAE:', mae)
  print('MSE:', mse)
  print('RMSE:', rmse)
  print('R2 Square', r2_square)
  print('_____________________________')
def evaluate(true, predicted):
  mae=metrics.mean_absolute_error(true, predicted)
  mse=metrics.mean_squared_error(true,predicted)
  rmse=np.sqrt(metrics.mean_squared_error(true,predicted))
  r2_square=metrics.r2_score(true,predicted)
  return mae,mse,rmse,r2_square
```

Preparing Data For Linear Regression

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

pipeline = Pipeline([ ('std_scaler', StandardScaler())])

X_train=pipeline.fit_transform(X_train)
X_test=pipeline.transform(X_test)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression

lin_reg=LinearRegression()
lin_reg.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Model Evalution Let's evalute the model by checking out it's coefficients and how we can intercept them.

```
#print the intercept
print(lin_reg.intercept_)
```

```
1228219.1492415662
```

```
coeff_df=pd.DataFrame(lin_reg.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

	Coefficient
Avg. Area Income	232679.724643
Avg. Area House Age	163841.046593
Avg. Area Number of Rooms	121110.555478
Avg. Area Number of Bedrooms	2892.815119
Area Population	151252.342377

```
pred=lin_reg.predict(X_test)
```

```
pd.DataFrame({'True Values': y_tes, 'Predicted Values': pred}).hvplot.scatter(x='True Values',y=
```

```
pd.DataFrame({'Error Values': (y_tes-pred)}).hvplot.kde()
```

```
test_pred=lin_reg.predict(X_test)
train_pred=lin_reg.predict(X_train)

print('Test set evalution:\n_____________________________________________')
print_evaluate(y_tes,test_pred)
print('Train set evaluton:\n_____________________________________________')
print_evaluate(y_train,train_pred)
results_df=pd.DataFrame(data=[["Linear Regression", *evaluate(y_tes, test_pred), cross_val(Linea
```

```
Test set evalution:
_____________________________________________
MAE: 81135.56609336878
MSE: 10068422551.40088
```

```
RMSE: 100341.52954485436
R2 Square 0.9146818498754016

__________________________
Train set evaluton:

__________________________________________
MAE: 81480.49973174892
MSE: 10287043161.197224
RMSE: 101425.06180031257
R2 Square 0.9192986579075526

__________________________
```

Robust Regression

```
from sklearn.linear_model import RANSACRegressor

model= RANSACRegressor (base_estimator=LinearRegression(), max_trials=100)
model.fit(X_train, y_train)

test_pred = model.predict(X_test)
train_pred=model.predict(X_train)

print('Test set evalution:\n_____________________________________________________')
print_evaluate(y_tes, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________________________')
print_evaluate(y_train,train_pred)

results_df_2=pd.DataFrame(data=[["Robust Regression", *evaluate (y_tes, test_pred), cross_val(RA
results_df=results_df.append(results_df_2, ignore_index=True)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ransac.py:343: FutureWarning:
  warnings.warn(
Test set evalution:

_____________________________________________
MAE: 83461.11441172272
MSE: 10826948773.617777
RMSE: 104052.62502031257
R2 Square 0.9082542239220648
__________________________
====================================
Train set evaluation:

_____________________________________________
MAE: 84215.14195015775
MSE: 10919364262.715544
RMSE: 104495.76193662375
R2 Square 0.9143381303073199
__________________________
<ipython-input-27-9889222c76f5>:16: FutureWarning: The frame.append method is deprecated ar
  results_df=results_df.append(results_df_2, ignore_index=True)
```

Ridge Regression

```
from sklearn.linear_model import Ridge

model=Ridge(alpha=100, solver='cholesky', tol=0.0001, random_state=42)
model.fit(X_train, y_train)
```

```
pred=model.predict(X_test)

test_pred=model.predict(X_test)
train_pred=model.predict(X_train)

print('Test set evaluation:\n_____________________________________________')
print_evaluate(y_tes, test_pred)
print('==================================')
print('Train set evaluation:\n_____________________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["Ridge Regression", *evaluate(y_tes, test_pred) , cross_val(R
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', "Cross Validati
results_df = results_df.append(results_df_2, ignore_index=True)
```

```
Test set evaluation:
_____________________________________________
MAE: 81428.64835535336
MSE: 10153269900.892609
RMSE: 100763.43533689494
R2 Square 0.9139628674464607
__________________________________
==================================
Train set evaluation:
_____________________________________________
MAE: 81972.39058585507
MSE: 10382929615.143456
RMSE: 101896.66145239232
R2 Square 0.9185464334441484
__________________________________
<ipython-input-29-358ef1b6c489>:18: FutureWarning: The frame.append method is deprecated ar
  results_df = results_df.append(results_df_2, ignore_index=True)
```

LASSO Regresssion

```
from sklearn.linear_model import Lasso

model = Lasso(alpha=0.1,
              precompute=True,
#               warm_start=True,
              positive=True,
              selection='random',
              random_state=42)
model.fit(X_train, y_train)

test_pred = model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_tes, test_pred)
print('==================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["Lasso Regression", *evaluate(y_tes, test_pred) , cross_val(L
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', "Cross Validati
results_df = results_df.append(results_df_2, ignore_index=True)
```

```
Test set evaluation:
_____________________________________
MAE: 81135.6985172622
MSE: 10068453390.364523
RMSE: 100341.68321472648
R2 Square 0.914681588551116
_____________________________
==================================
Train set evaluation:
_____________________________________
MAE: 81480.63002185506
MSE: 10287043196.634295
RMSE: 101425.0619750084
R2 Square 0.9192986576295505
_____________________________
<ipython-input-33-0f60150cf142>:22: FutureWarning: The frame.append method is deprecated ar
  results_df = results_df.append(results_df_2, ignore_index=True)
```

```
results_df
```

R2 Cross

```
results_df.columns
```

```
Index(['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', 'Cross Validation',
       'Cross validation'],
      dtype='object')
```

```
results_df=results_df.reset_index()
```

Model Comparison

Lasso

```
results_df.set_index('Model', inplace=True)
results_df['R2 Square'].plot(kind='barh', figsize=(12, 8))
```

```
<Axes: ylabel='Model'>
```


```
model.predict(X_test)
```

```
1308533.8356186203
```

```
model.predict([[90499.05745,  6.384358921,  4.242191302 ,3.04,  33970.16499]])
```

✓ 0s completed at 02:43

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.