AI_PHASE 3: PREDICTING HOUSE PRICES USING MACHINE LEARNING

Data loading and preprocessing:

Import the required Libraries:

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

%matplotlib inline
```
add Codeadd Markdown

```
import warnings
warnings.filterwarnings('ignore')
```
add Codeadd Markdown

Loading the dataset:

add Codeadd Markdown

```
data = pd.read_csv('/kaggle/input/usa-housing/USA_Housing.csv')
data.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.4586	5.682861	7.009188	4.09	23086.8005	1.06E+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.6425	6.0029	6.730821	3.09	40173.0722	1.51E+06	188 Johnson Views Suite 079\nLake Kathleen, CA...

add Codeadd Markdown
We can interpret that this is a regression problem since the house prise can't be labelled, it is a continuous variable.

The dataset contains the following features:-

- Avg. Area Income: Average Income of residents of the area the house is located in.
- Avg. Area House Age: Average Age of Houses in same area
- Avg. Area Number of Rooms: Average Number of Rooms for Houses in same area
- Avg. Area Number of Bedrooms: Average Number of Bedrooms for Houses in same area
- Area Population: Population of the area the house is located in.
- Price: Price of the house.

Address: Address for a particular house.

add Codeadd Markdown

```
data.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
```

add Codeadd Markdown

```
data.isnull().sum() #The data does not have any null values

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
```

add Codeadd Markdown

```
data.describe()
```

add Codeadd Markdown

```
data.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')
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000	5000	5000	5000	5000	5.00E+03
mean	68583.109	5.977222	6.987792	3.98133	36163.516	1.23E+06
std	10657.9912	0.991456	1.005833	1.234137	9925.65011	3.53E+05
min	17796.6312	2.644304	3.236194	2	172.610686	1.59E+04
25%	61480.5624	5.322283	6.29925	3.14	29403.9287	9.98E+05
50%	68804.2864	5.970429	7.002902	4.05	36199.4067	1.23E+06
75%	75783.3387	6.650808	7.665871	4.49	42861.2908	1.47E+06
max	107701.748	9.519088	10.759588	6.5	69621.7134	2.47E+06

add Codeadd Markdown

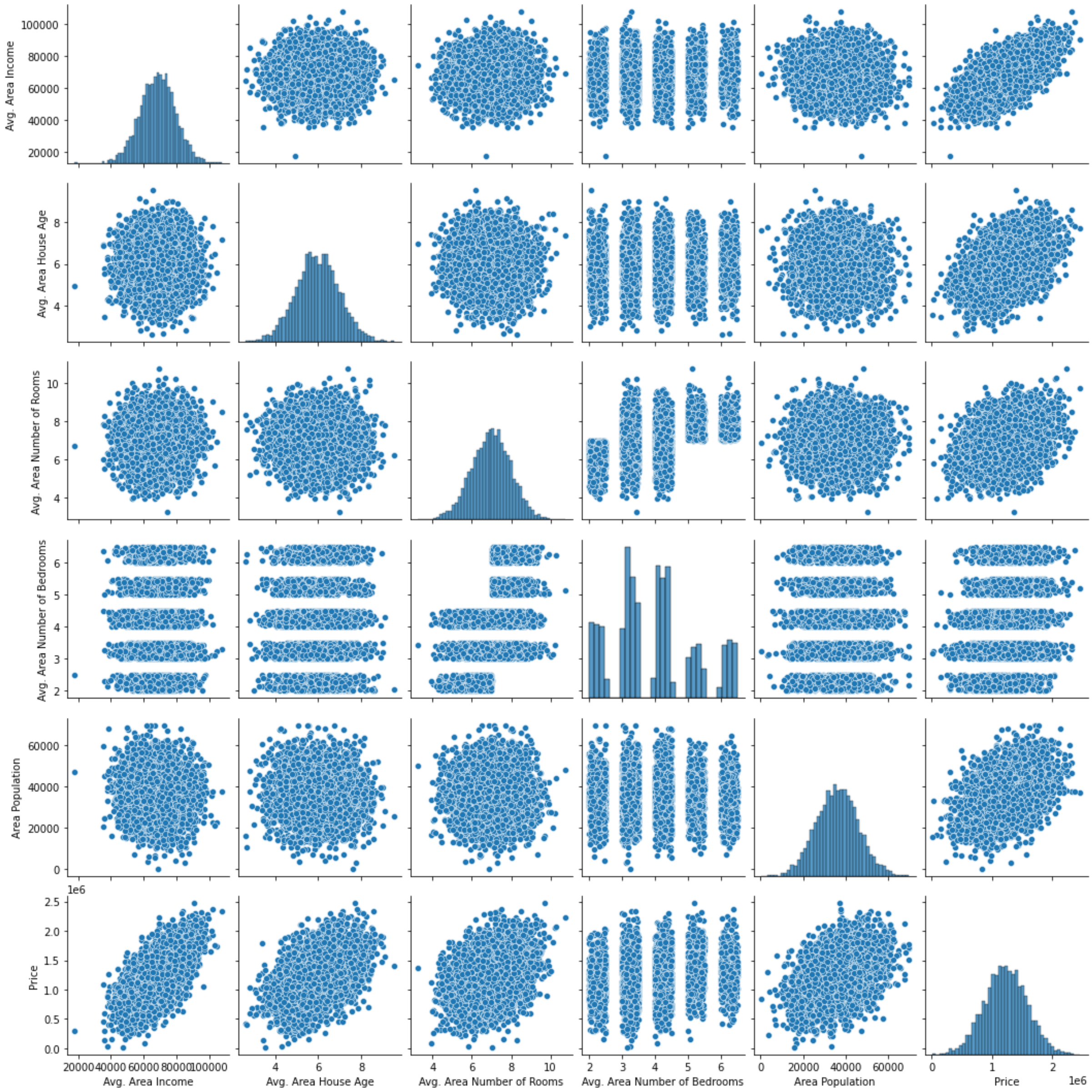
EDA(Exploratory Data Analysis):

Creating plots to analyze our data using different visualization techniques.

add Codeadd Markdown

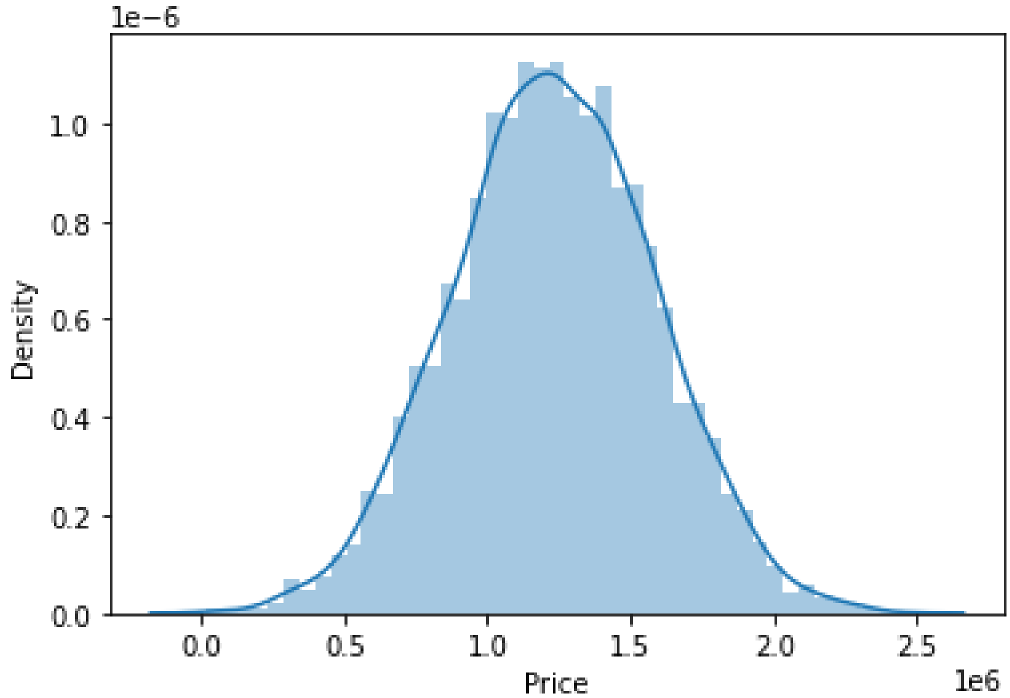
```
sns.pairplot(data)
```

<seaborn.axisgrid.PairGrid at 0x7f95ba648190>



add Codeadd Markdown

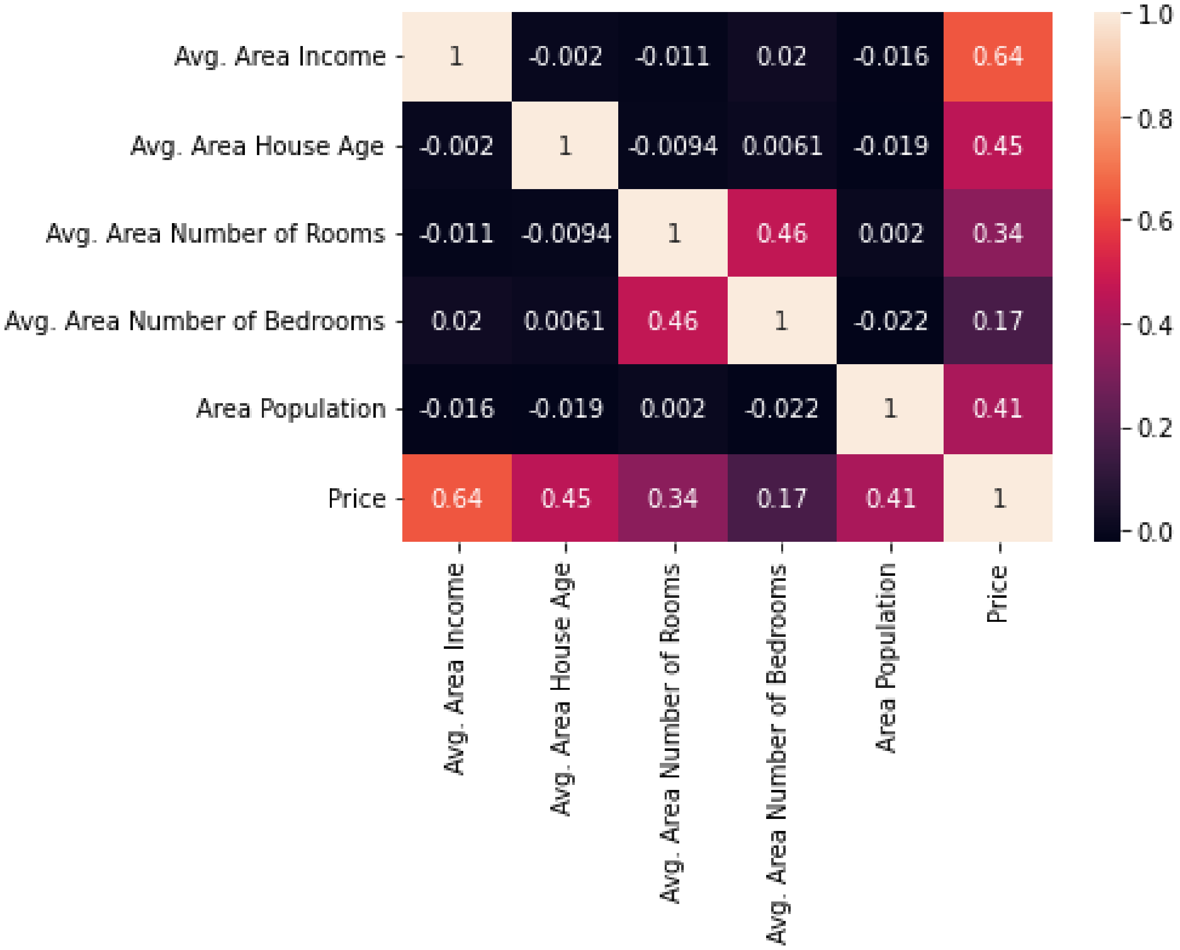
```
sns.distplot(data['Price'])
plt.plot()
```



add Codeadd Markdown

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

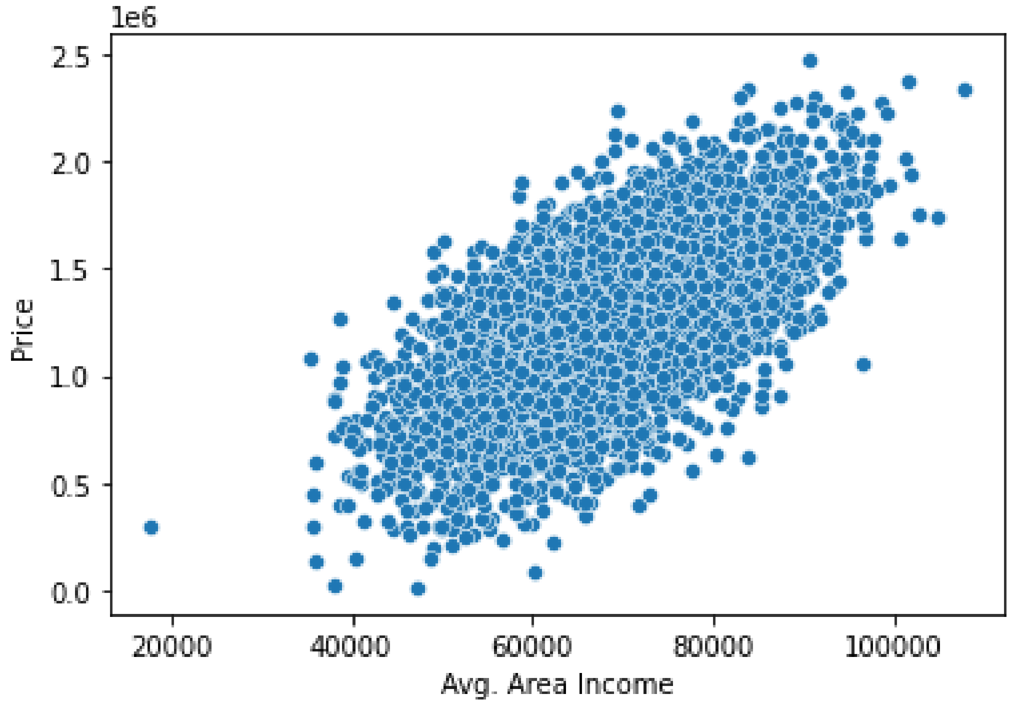
<AxesSubplot:>



add Codeadd Markdown

sns.scatterplot(x=data['Avg. Area Income'], y=data['Price'])

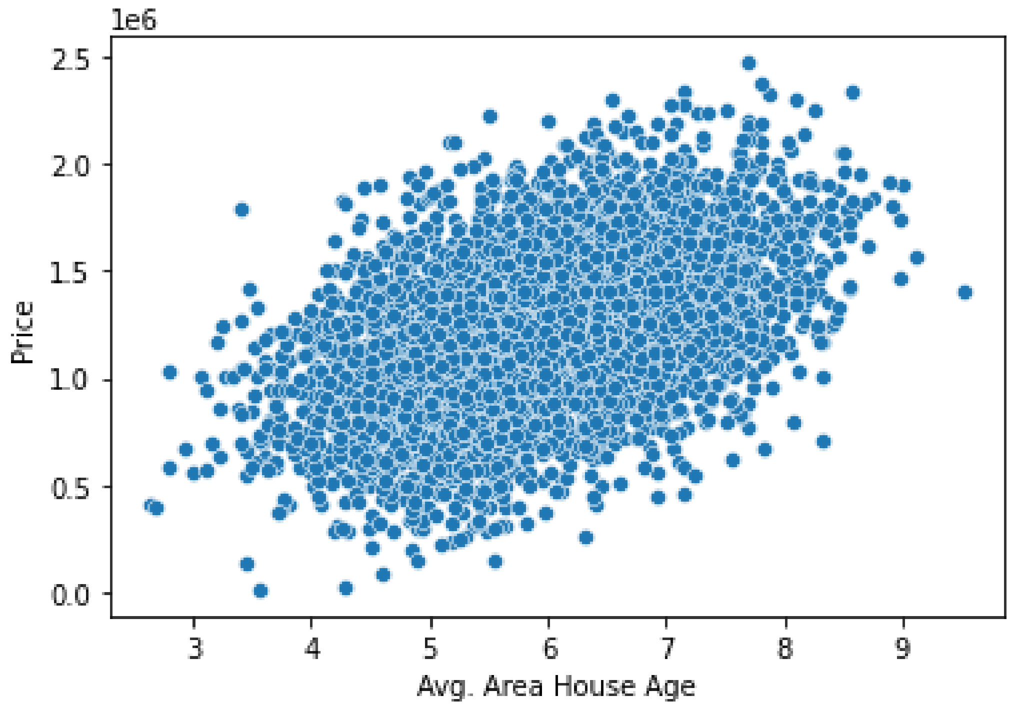
<AxesSubplot:xlabel='Avg. Area Income', ylabel='Price'>



add Codeadd Markdown

sns.scatterplot(x=data['Avg. Area House Age'], y=data['Price'])

<AxesSubplot:xlabel='Avg. Area House Age', ylabel='Price'>



add Codeadd Markdown

Training Our Linear Regression Model:

add Codeadd Markdown

#dividing the data into dependent and independent features

X = data.drop(labels = ['Price', 'Address'], axis=1)

Y = data['Price']

add Codeadd Markdown

X

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
0	79545.4586	5.682861	7.009188	4.09	23086.8005
1	79248.6425	6.0029	6.730821	3.09	40173.0722
2	61287.0672	5.86589	8.512727	5.13	36882.1594
3	63345.24	7.188236	5.586729	3.26	34310.2428
4	59982.1972	5.040555	7.839388	4.23	26354.1095
...	...	...	...	...	...
4995	60567.9441	7.830362	6.137356	3.46	22837.361
4996	78491.2754	6.999135	6.576763	4.02	25616.1155
4997	63390.6869	7.250591	4.805081	2.13	33266.1455

5000 rows × 5 column

add Codeadd Markdown

Y

```
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
```

add Codeadd Markdown

#Shape of X and Y

print(f"X Shape: {X.shape}")

print(X)

print(f"y Shape: {Y.shape}")

print(Y)

```
X Shape: (5000, 5)
     Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0        79545.458574             5.682861                   7.009188
1        79248.642455             6.002900                   6.730821
2        61287.067179             5.865890                   8.512727
3        63345.240046             7.188236                   5.586729
4        59982.197226             5.040555                   7.839388
```

```
...          ...            ...                ...
4995     60567.944140        7.830362            6.137356
4996     78491.275435        6.999135            6.576763
4997     63390.686886        7.250591            4.805081
4998     68001.331235        5.534388            7.130144
4999     65510.581804        5.992305            6.792336

     Avg. Area Number of Bedrooms  Area Population
0                    4.09    23086.800503
1                    3.09    40173.072174
2                    5.13    36882.159400
3                    3.26    34310.242831
4                    4.23    26354.109472
...                  ...          ...
4995                    3.46    22837.361035
4996                    4.02    25616.115489
4997                    2.13    33266.145490
4998                    5.44    42625.620156
4999                    4.07    46501.283803

[5000 rows x 5 columns]
y Shape: (5000,)
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
```

add Codeadd Markdown

Spliting data for our model:

add Codeadd Markdown

```
from sklearn.model_selection import train_test_split

#Train-Test Split to train our model on the training set and then use the test set to evaluate the model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=10)
```

add Codeadd Markdown

Scale/Normalize the Training data:

add Codeadd Markdown

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_norm = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
```

add Codeadd Markdown

```
X_norm
array([[ 0.24082188, -1.0185292 , -0.77505327, -1.34738436, -1.03787896],
       [ 0.98865562,  1.66179379,  3.18432256, -0.45796685, -0.92397092],
       [ 0.14525596, -0.18436613,  1.45749594,  1.78174815,  0.71971398],
       ...,
       [ 0.20142052,  0.59892004, -0.52979241, -0.66819281, -1.2877454 ],
       [ 0.49559505,  0.75068063,  0.60135685,  0.20505347,  0.48354548],
       [-0.4372132 ,  0.40979201,  1.9322583 ,  0.01908436, -2.21714608]])
```

add Codeadd Markdown

Creating and fitting our Linear Regression Model:

add Codeadd Markdown

```
from sklearn.linear_model import LinearRegression


lin_model = LinearRegression(normalize=True)
lin_model.fit(X_norm,y_train)
LinearRegression(normalize=True)
```

add Codeadd Markdown

View Parameters:

add Codeadd Markdown

```
print(f"Model parameters:- w: {lin_model.coef_}, b:{lin_model.intercept_}")
Model parameters:- w: [230102.81587921 164125.74656351 121432.99857953    591.86181588
 149933.87898543], b:1236207.893936157
```

add Codeadd Markdown

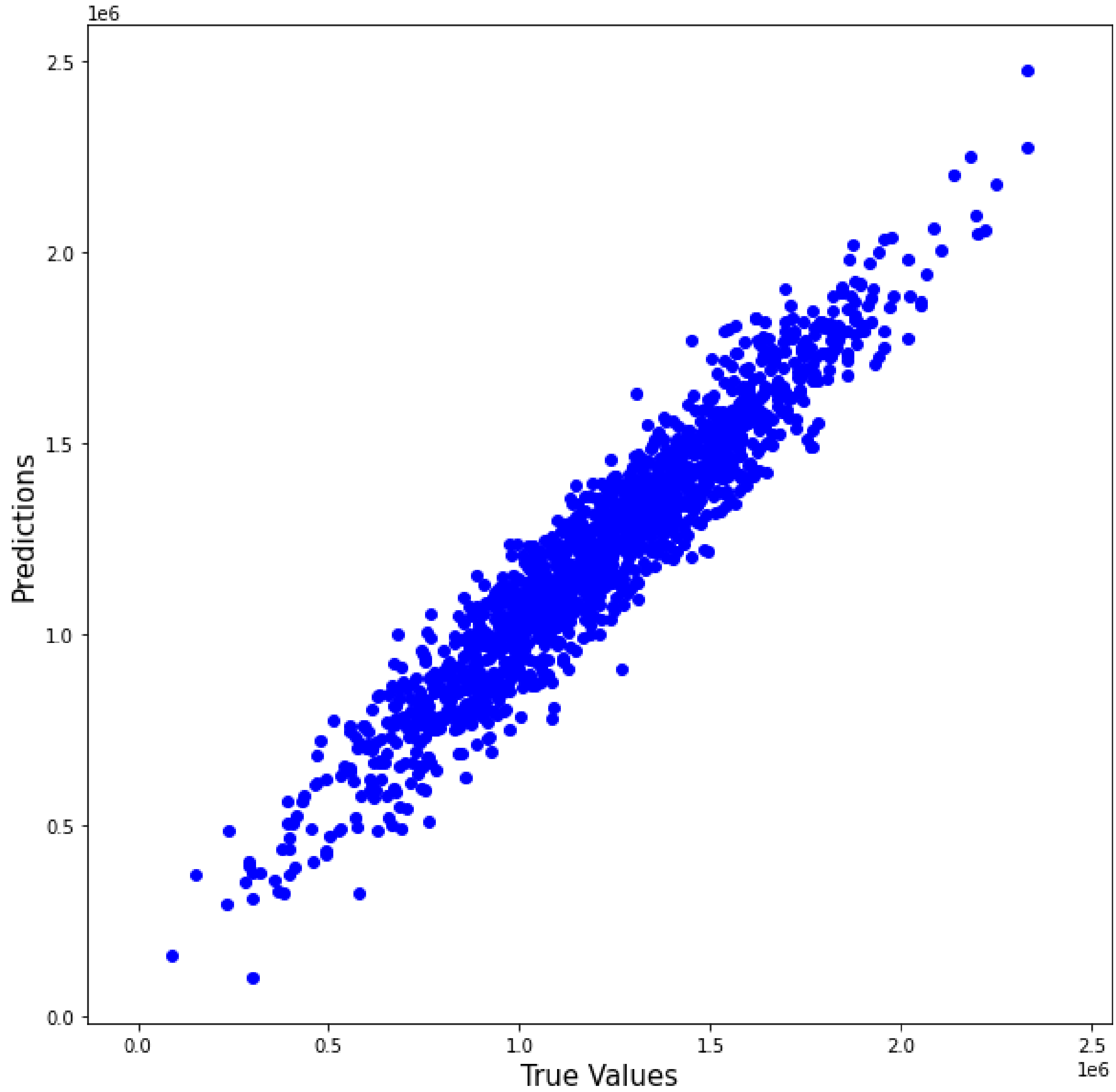
Making predictions:

add Codeadd Markdown

```
# make a prediction using lin_model.predict()
y_pred_linear_model = lin_model.predict(X_test)
```

add Codeadd Markdown

```
plt.figure(figsize=(10,10))
plt.scatter(y_test, y_pred_linear_model, c='blue')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```



add Codeadd Markdown

Regression Evaluation Metrics:

add Codeadd Markdown

The common evaluation metrics for regression problems are:

- Mean Absolute Error(MAE)
- Mean Squared Error(MSE)
- Root Mean Squared Error (RMSE)

add Codeadd Markdown

```
from sklearn import metrics

def evaluated_results(true, predicted):
    print('__________________________________')
```

```
    print('MAE:', metrics.mean_absolute_error(true, predicted))
    print('MSE:', metrics.mean_squared_error(true, predicted))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(true, predicted)))
    print('__________________________________')
```

add Codeadd Markdown

```
y_train_pred = lin_model.predict(X_norm)

print('Test set evaluation:')
evaluated_results(y_test, y_pred_linear_model)
print('Train set evaluation:')
evaluated_results(y_train, y_train_pred)
```

```
Test set evaluation:
__________________________________
MAE: 81349.24091897604
MSE: 10408992254.1173
RMSE: 102024.46889897197
__________________________________
Train set evaluation:
__________________________________
MAE: 81383.52050897086
MSE: 10146811289.400652
RMSE: 100731.38184995107
```