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

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
# Checkout the data
df = pd.read_csv('USA_Housing.csv')
df
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Mich 674\nl
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 J Su K
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9 Stravenue
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Bar
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Ra

```
df.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
```

```
 6   Address                        5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

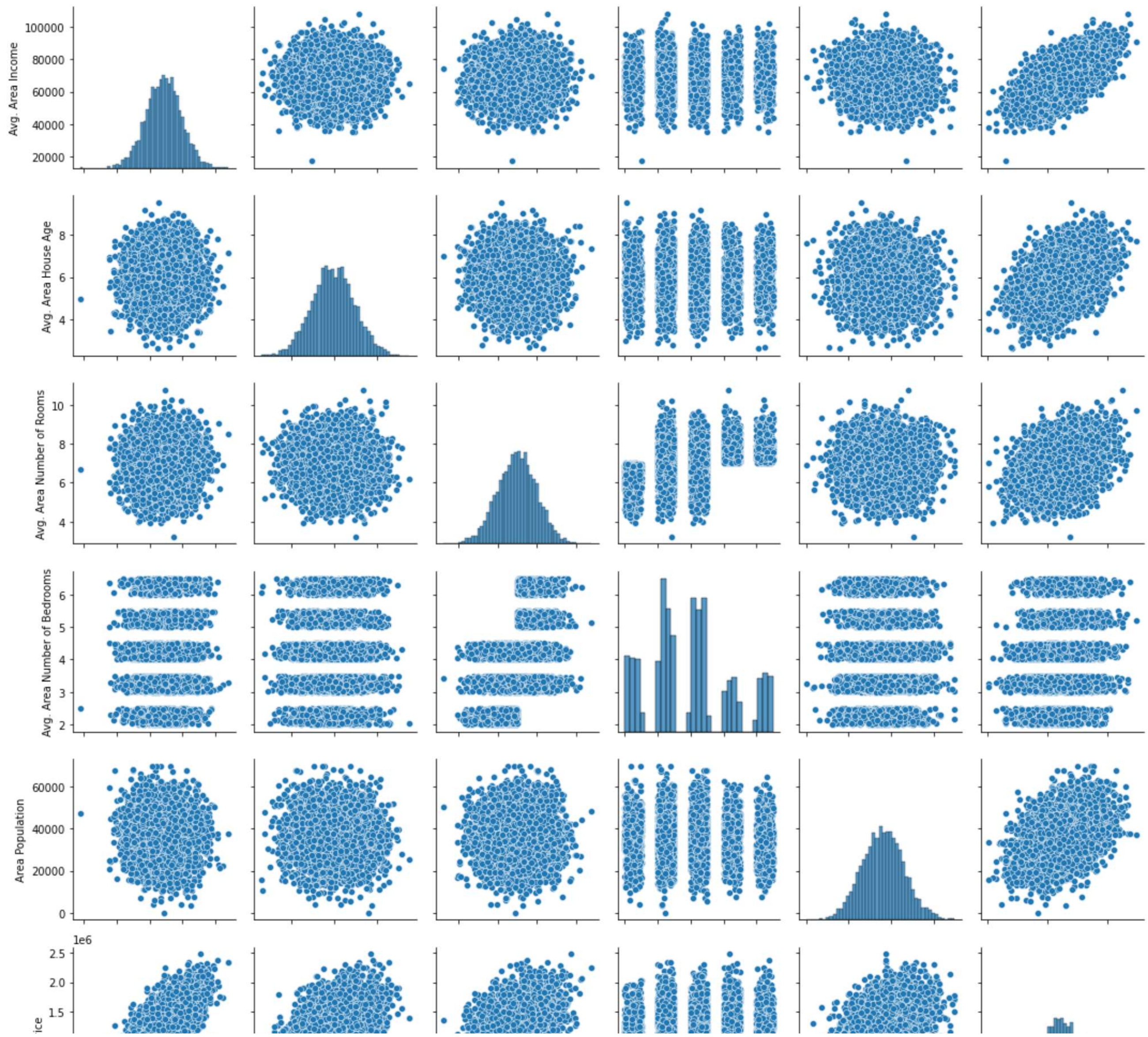
```
df.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Pric
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+0
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+0
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+0
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+0
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+0
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+0
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+0

```
df.columns
```

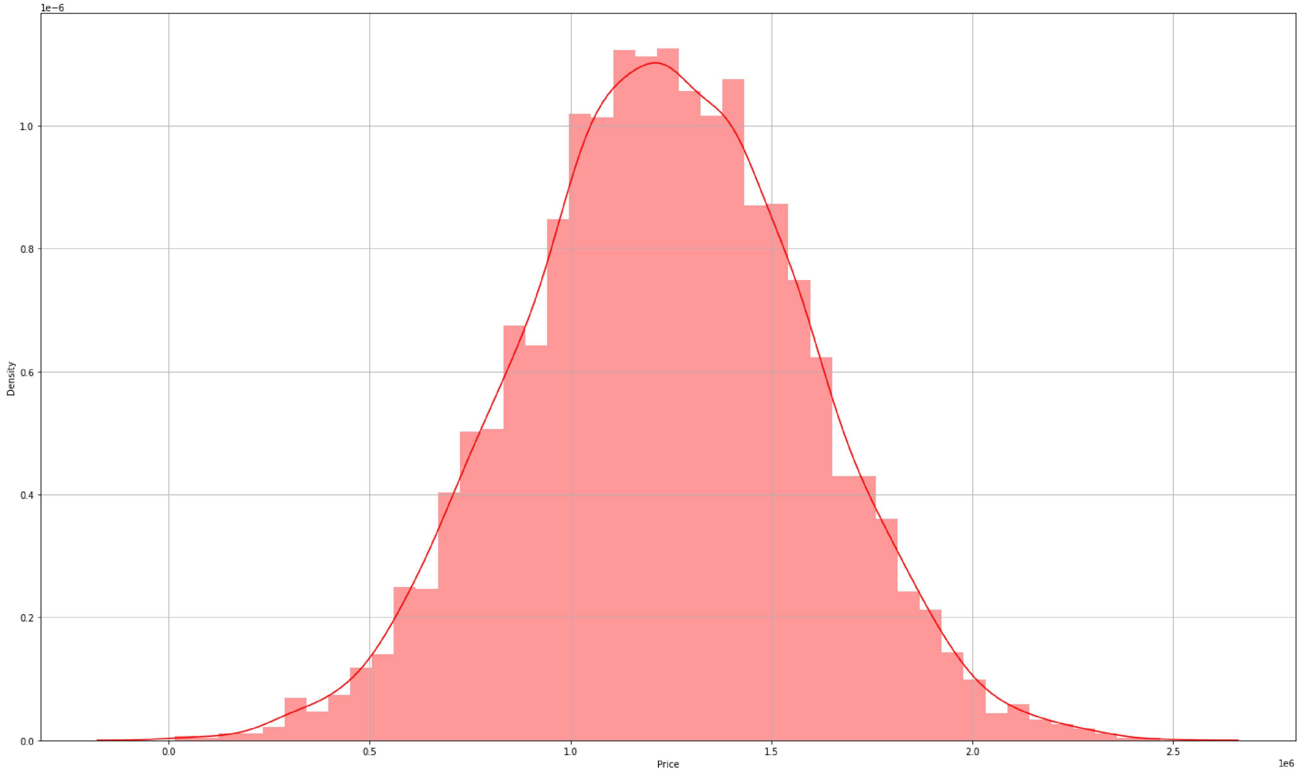
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

```
# Draw the pairplot
sns.pairplot(df)
plt.tight_layout()
```




```
# Now checkout the distribution of the price
plt.figure(figsize=(20,12))
sns.distplot(df['Price'], color='red')
plt.grid()
plt.tight_layout()
```

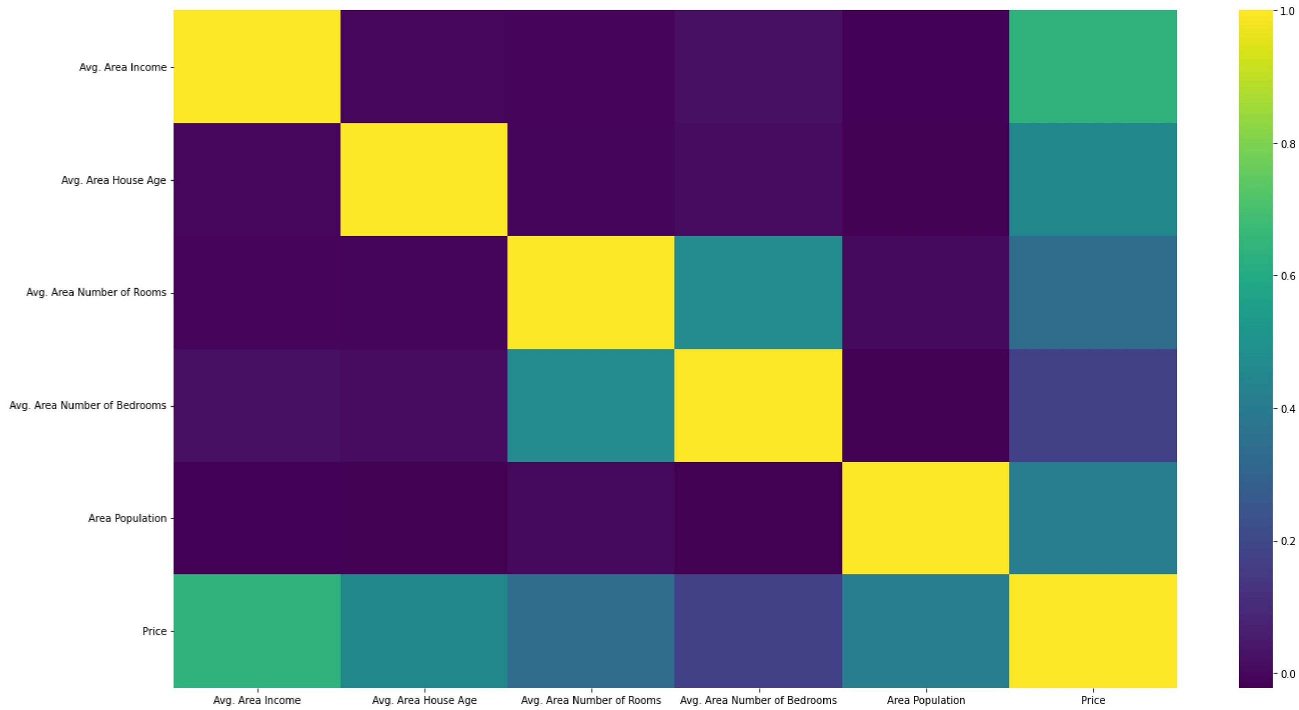
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



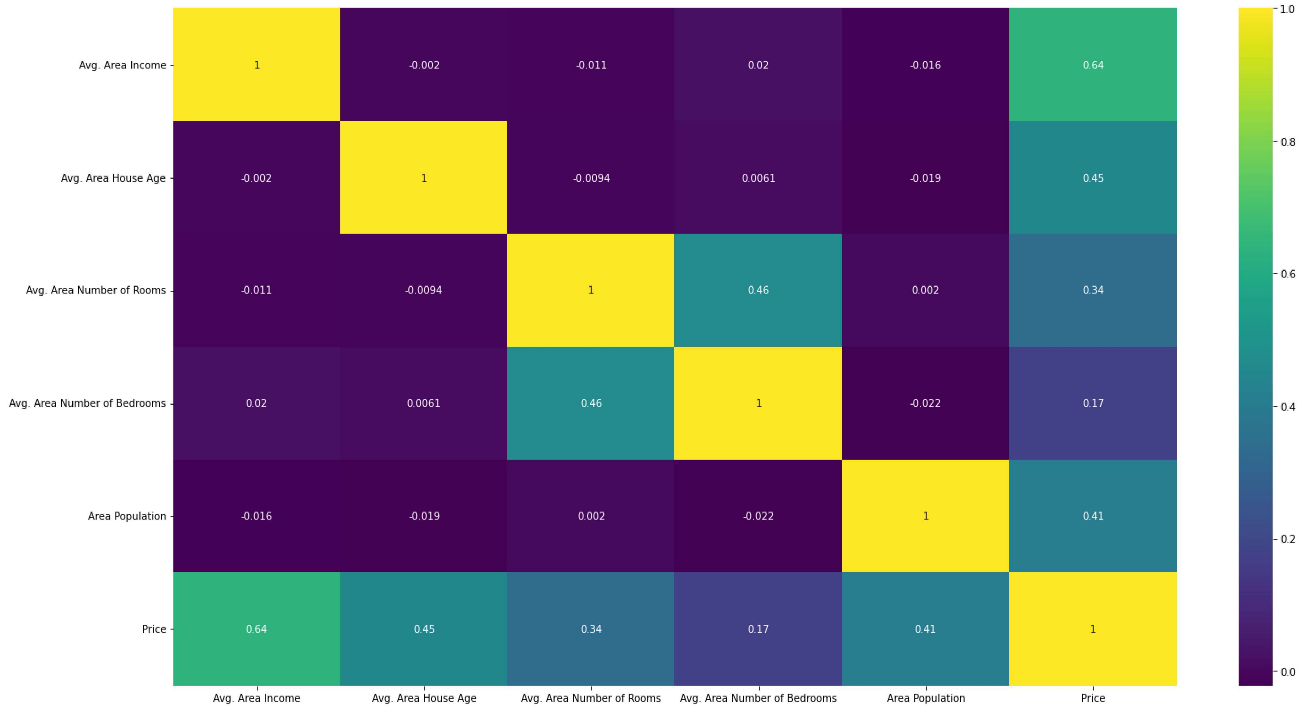

```
# Correalation Matrix
df.corr()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1.000000	-0.002007	-0.011032	0.019788	-0.016234	0.639734
Avg. Area House Age	-0.002007	1.000000	-0.009428	0.006149	-0.018743	0.452543
Avg. Area Number of Rooms	-0.011032	-0.009428	1.000000	0.462695	0.002040	0.335664

```
# Heat Map
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), cmap = 'viridis')
plt.tight_layout()
```




```
# Heat Ma, annot = True
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), cmap = 'viridis', annot = True)
plt.tight_layout()
```




```
# total colums in DataFrame
df.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')
```

```
# Define 'Input Matrix'
X = df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area
```

```
# Check 'Input Matrix'
# Dimension = 5000x5
X
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
0	79545.458574	5.682861	7.009188	4.09	23086.800503
1	79248.642455	6.002900	6.730821	3.09	40173.072174
2	61287.067179	5.865890	8.512727	5.13	36882.159400
3	63345.240046	7.188236	5.586729	3.26	34310.242831

```
# Define the 'Output Vector'
y = df['Price']
```

4995	60567.944140	7.830362	6.137356	3.46	22837.361035

```
# Check our 'Output Vector'
# Dimension = 5000x1
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
```

```
# import the 'train_test_split model' from 'sklearn.model_selection'
from sklearn.model_selection import train_test_split

# Split the data into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)

# Data = (X, y)
# Training Data = (X_train, y_train)
# Testing Data = (X_test, y_test)

# test_size = 0.4 (40% of the whole data)
# random_state 101 (Specific set of random split on our data)
```

```
# Check 'training inputs matrix'
# Dimension: 3000x5
X_train
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
1303	68091.179676	5.364208	7.502956	3.10	44557.379656
1051	75729.765546	5.580599	7.642973	4.21	29996.018448
4904	70885.420819	6.358747	7.250241	5.42	38627.301473
931	73386.407340	4.966360	7.915453	4.30	38413.490484
4976	75046.313791	5.351169	7.797825	5.23	34107.888619
...	...	...	...	...	...
4171	56610.642563	4.846832	7.558137	3.29	25494.740298
599	70596.850945	6.548274	6.539986	3.10	51614.830136

```
# Check 'training outputs vector'
# Dimension: 3000x1
y_train
```

```
1303    1.489648e+06
1051    1.183015e+06
4904    1.547889e+06
931     1.186442e+06
4976    1.340344e+06
            ...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 3000, dtype: float64
```

```
# Check 'test inputs matrix'
# Dimension: 2000x5
X_test
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
1718	66774.995817	5.717143	7.795215	4.32	36788.980327

```
# Check 'test outputs vector'
# Dimension: 2000x1
y_test
```

```
1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
            ...
1776    1.489520e+06
4269    7.777336e+05
1661    1.515271e+05
2410    1.343824e+06
2302    1.906025e+06
Name: Price, Length: 2000, dtype: float64
```

```
# Now import the 'LinearRegression' model from 'sklearn.linear_model'
from sklearn.linear_model import LinearRegression
```

```
# Now define the instances of the LinearRegression model
lm = LinearRegression() # creating a LinearRegression object
```

```
print(dir(lm)) # print all the available methods on 'lm' (LinearRegression object)
```

```
['__abstractmethods__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
```

```
# fit (train) my model on my training data
lm.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# print the intercept
# intercep is the constant term of our hypothesis
print(lm.intercept_)
```

```
-2640159.7968526958
```

```
X_train.columns
```

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

```
# data = lm.coef_
# indices = X_train.columns
# columns ['Parameters']
cdf = pd.DataFrame(lm.coef_, X_train.columns, columns = ['Parameters'])


# Now check our 'Parameter Vector'
# Dimension: 5x1
cdf
```

	Parameters
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

```
from sklearn.datasets import load_boston
boston = load_boston()
boston.keys()
print(boston.DESCR)
boston_df = boston.data
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attrib

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 othe
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None
```

```
This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Me

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that add
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proc

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarn

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
```

```
# For pridictions we use the predict() method and pass X_test (test_set) as argument
# X_test Dimension: 2000x5
# predictions Dimension: 2000x1
predictions = lm.predict(X_test)
```

```
# Check the predictions
# 'predictions' will return predicted prices of the house according to our linear-fit-mode
# it will return an array which size of as input size (size of X_test)
predictions
```

```
array([1260960.70567627,  827588.75560329, 1742421.24254344, ...,
        372191.40626917, 1365217.15140898, 1914519.5417888 ])
```

```
# Convert the predictions from array to Series
predicted_values = pd.Series(data = predictions)
```

```
predicted_values
```

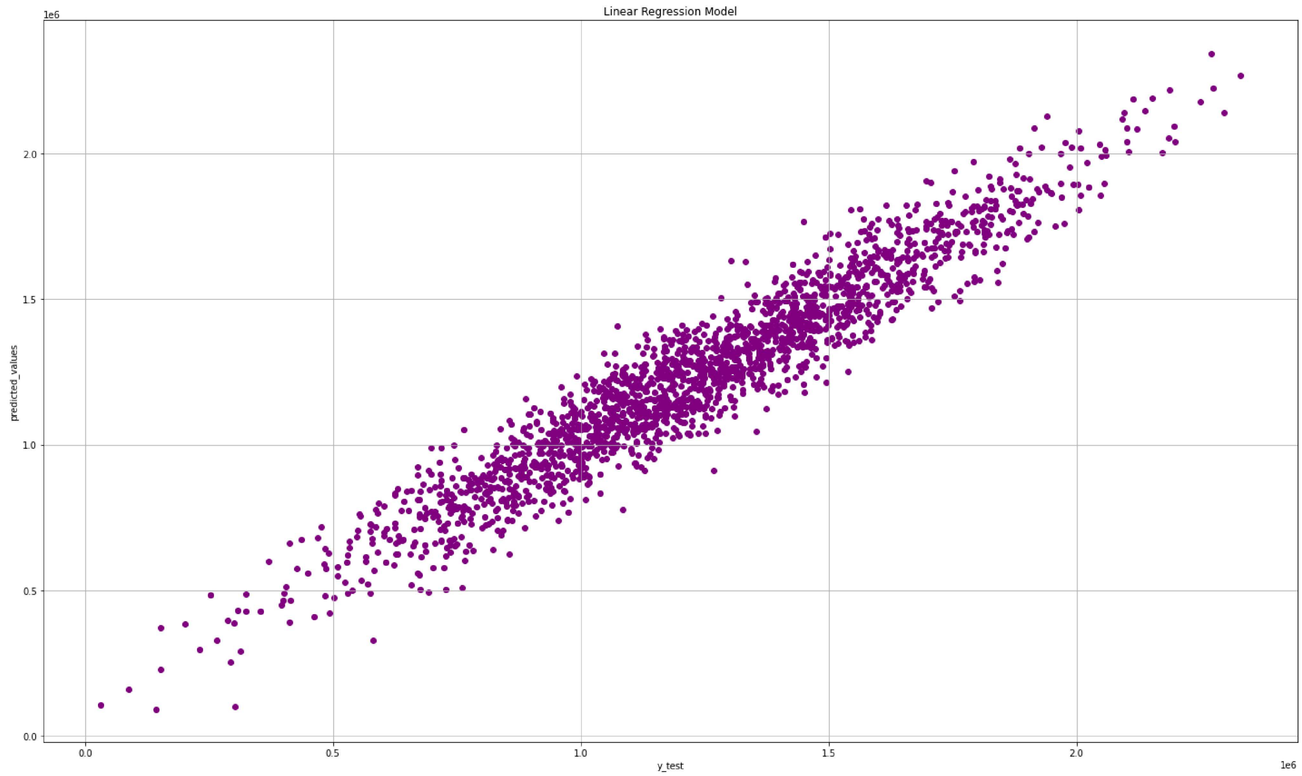
```
0       1.260961e+06
1       8.275888e+05
2       1.742421e+06
3       9.746254e+05
4       9.987178e+05
            ...
1995    1.515043e+06
```

```
1998    1.365217e+06
1999    1.914520e+06
Length: 2000, dtype: float64
```

```
# Check actual values of houses
y_test
```

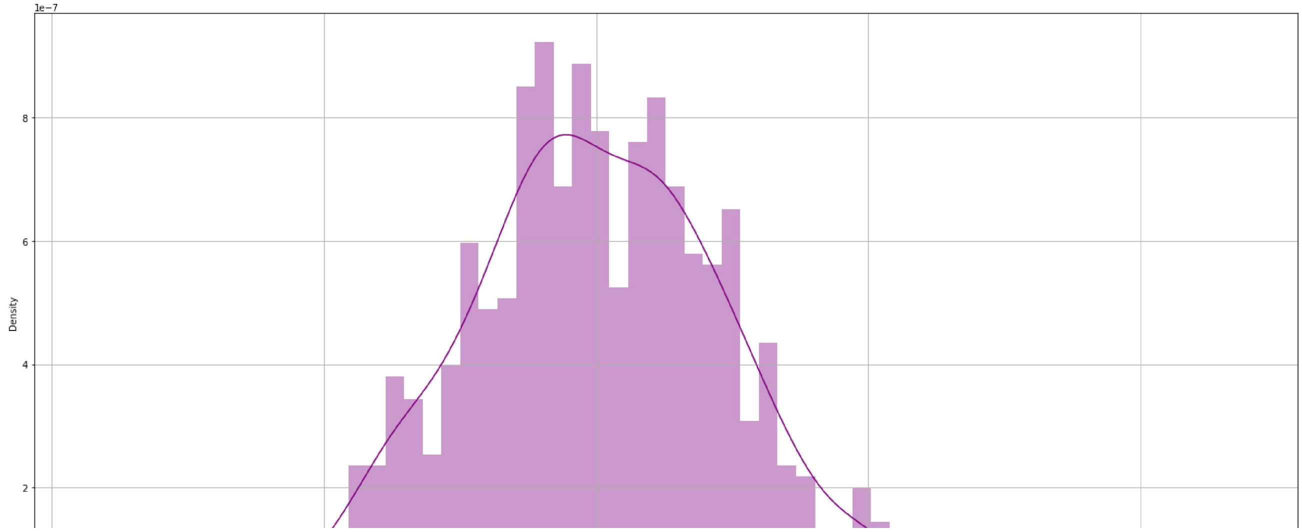
```
1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
            ...
1776    1.489520e+06
4269    7.777336e+05
1661    1.515271e+05
2410    1.343824e+06
2302    1.906025e+06
Name: Price, Length: 2000, dtype: float64
```

```
# Visualizing our predictions
# Draw a scatter plot between 'y_test' and 'predicted_values'
plt.figure(figsize=(20,12))
plt.xlabel('y_test')
plt.ylabel('predicted_values')
plt.title('Linear Regression Model')
plt.scatter(y_test, predicted_values, color='purple')
plt.grid()
plt.tight_layout()
```




```
# Now we are going to create a histogram of the distribution of ore residues
# residues are the differnce between the actual values (y_test) and the predicted values (
residues = y_test - predicted_values # it will give the "absolute error"
plt.figure(figsize = (20,10))
sns.distplot((residues), bins=50, color='purple')
plt.grid()
plt.tight_layout()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```




```
# Import 'matrics' module from sklearn
from sklearn import metrics


# Now calulating errors
print('MAE:', metrics.mean_absolute_error(y_test, predicted_values)) # Mean absolute error
print('MSE:', metrics.mean_squared_error(y_test, predicted_values)) # Mean squared error
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predicted_values))) # Root mean
```

```
MAE: 82288.22251914942
MSE: 10460958907.208977
RMSE: 102278.82922290897
```

```
# Now check the accuracy of our model
# Explained variance regression score function: Best possible score is 1.0, lower values a
metrics.explained_variance_score(y_true=y_test, y_pred=predicted_values)
```

```
0.9178179926151839
```

```
accuracy = metrics.explained_variance_score(y_true=y_test, y_pred=predicted_values)
print("Our model is : ",accuracy * 100,"%", " accurate.")
```

```
Our model is :  91.78179926151839 %  accurate.
```