Linear Regression

information about a bunch of houses (500) in regions of the United States,it is all in the data set: USA_Housing.csv.

The data contains the following columns:

- 'Avg. Area Income': Avg. Income of residents of the city house is located in.
- 'Avg. Area House Age': Avg Age of Houses in same city
- 'Avg. Area Number of Rooms': Avg Number of Rooms for Houses in same city
- 'Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in same city
- 'Area Population': Population of city house is located in
- 'Price': Price that the house sold at
- 'Address': Address for the house

1- import libararies

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

2-load the data

In [11]:
```
housing_data = pd.read_csv("data/USA_Housing.csv")
```

2.1-checkout the data

In [18]:
```
housing_data.head(3)
```

Out[18]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Addre
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry A 674\nLaurabury, I 370
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Vie Suite 079\nLa Kathleen, C/
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizab Stravenue\nDanieltov WI 0648

In [16]:
```
housing_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
Avg. Area Income                5000 non-null float64
Avg. Area House Age             5000 non-null float64
Avg. Area Number of Rooms       5000 non-null float64
Avg. Area Number of Bedrooms    5000 non-null float64
Area Population                 5000 non-null float64
Price                           5000 non-null float64
Address                         5000 non-null object
dtypes: float64(6), object(1)
memory usage: 273.5+ KB
```

In [17]:
```
housing_data.describe()
```

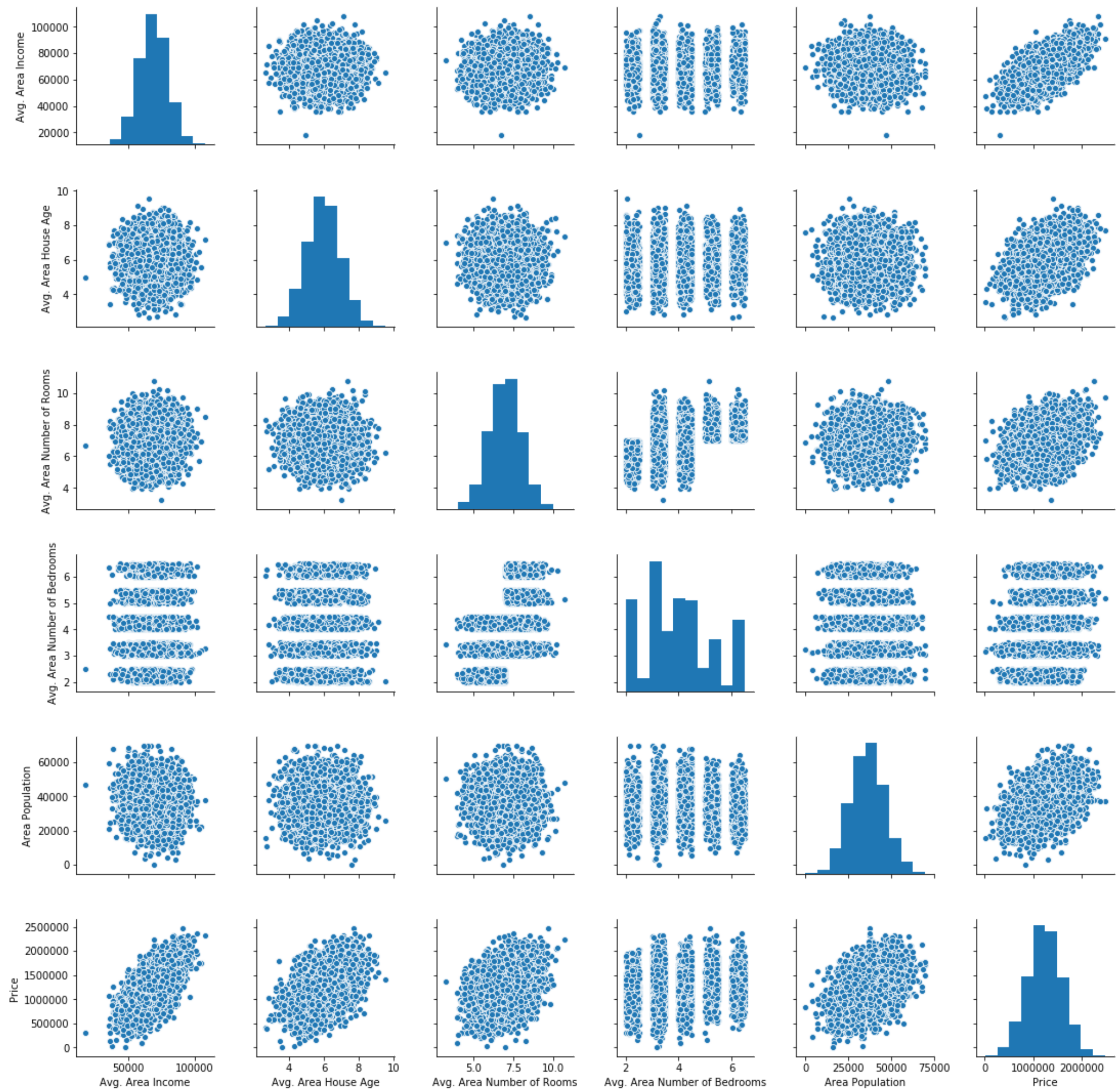
Out[17]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

2.2_visualize the data

In [19]:
```
sns.pairplot(housing_data)
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x2f512193ac8>



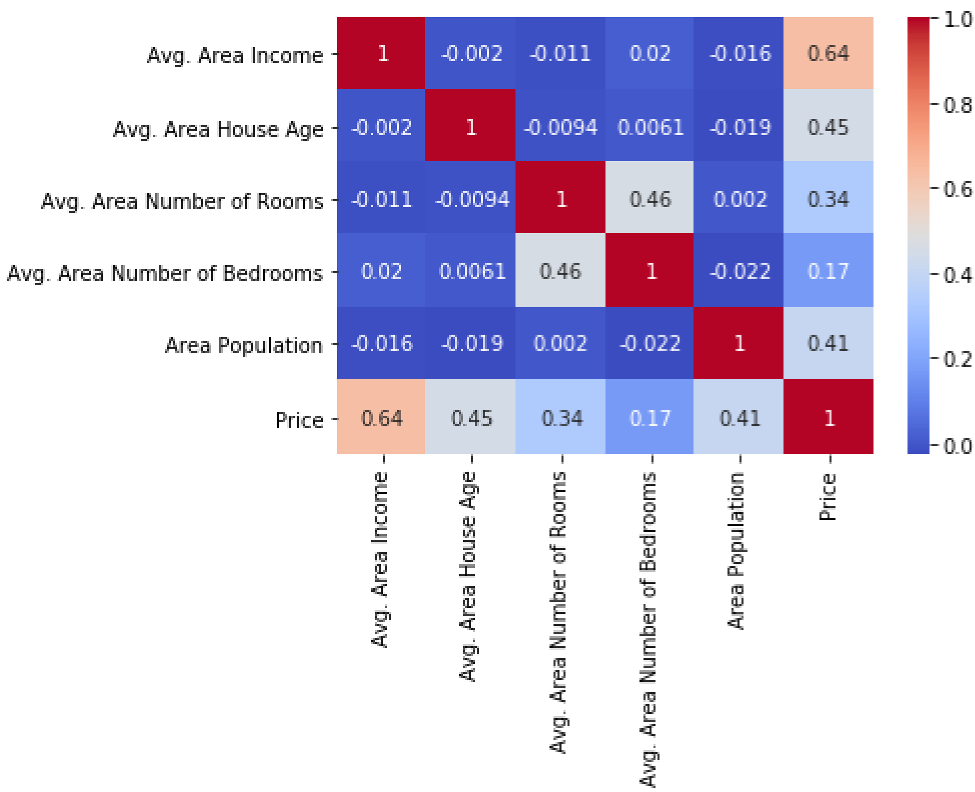
```
In [21]: housing_data.corr()
```

Out[21]:

	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
Avg. Area Number of Bedrooms	0.019788	0.006149	0.462695	1.000000	-0.022168	0.171071
Area Population	-0.016234	-0.018743	0.002040	-0.022168	1.000000	0.408556
Price	0.639734	0.452543	0.335664	0.171071	0.408556	1.000000

```
In [25]: sns.heatmap(housing_data.corr(), cmap = 'coolwarm', annot = True)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x2f51762feb8>



3- train with a linear regression model

In [28]:
```
housing_data.columns
```

Out[28]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Room
s',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Addres
s'],
      dtype='object')
```

In [59]:
```
X = housing_data.drop(columns = ['Price', 'Address'], axis = 1)
y = housing_data['Price']
```

In [60]:
```
# check the dimensions of data
print('housing_data size: ', housing_data.shape)
print('X size: ',X.shape)
print('y size: ',y.shape)
```

```
housing_data size:  (5000, 7)
X size:  (5000, 5)
y size:  (5000,)
```

split housing_data into tarin/ test data

In [61]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
m_state=101)
```

In [62]:
```
# check the dimensions of train/test data
print('X_train size: ', X_train.shape)
print('X_test size: ', X_test.shape)
print('y_train size: ', y_train.shape)
print('y_test size: ', y_test.shape)
```

```
X_train size:  (3500, 5)
X_test size:  (1500, 5)
y_train size:  (3500,)
y_test size:  (1500,)
```

fit the train data with a linear model

In [84]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Out[84]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

```
In [93]: # check the parameters of the model
         #The intercept of the regression line is just the predicted value for y, when
          x is 0.
         print('intercept:',lm.intercept_)
         print('coefficients:\n', lm.coef_)
```

```
intercept: -2641372.667301679
coefficients:
 [2.16176350e+01 1.65221120e+05 1.21405377e+05 1.31871878e+03
 1.52251955e+01]
```

Predict the test data with the trained lm model

```
In [66]: pred = lm.predict(X_test)
```

Evaluate the performance of the model:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
In [81]: from sklearn.metrics import mean_absolute_error, mean_squared_error
         print('MAE:',mean_absolute_error(y_test, pred))
         print('MSE:',mean_squared_error(y_test, pred))
         print('RMSE:',np.sqrt(mean_squared_error(y_test, pred)))
```

```
MAE: 81257.55795856068
MSE: 10169125565.897734
RMSE: 100842.08231635111
```

```
In [83]: print('RMSE(on trained data):',np.sqrt(mean_squared_error(y_train, lm.predict(
         X_train))))
```

```
RMSE(on trained data): 101211.97819208549
```

Visualization of the model

residual error frquency

In [190]:
```
f,axs = plt.subplots(2,2, figsize = (15, 10), )

axs[0,0].scatter(x = y_train, y = lm.predict(X_train))
axs[0,1].scatter(x = y_test, y = pred)

sns.distplot(y_train - lm.predict(X_train), bins = 40, ax = axs[1,0], axlabel=
'y_train - prediction')
sns.distplot(y_test - pred, bins = 40, ax = axs[1,1], axlabel='y_test - predic
tion')

axs[0,0].set_xlabel('y_train')
axs[0,0].set_ylabel('predictions')

axs[0,1].set(xlabel="y_test", ylabel='predictions')

f.subplots_adjust(hspace = 0.3, wspace = 0.3)
```

```
C:\Users\FirouzehPC\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: Fu
tureWarning: Using a non-tuple sequence for multidimensional indexing is depr
ecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result eit
her in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```