Martinez Assn 1 DSC 540

September 6, 2020

1 Martinez Assignment 1

1.1 Part 1 - Tools Readiness

1.1.1 Add the following libraries: Numpy, Pandas, Matplotlib, and Scikit-Learn

Please note that in part two all of the libraries are imported and work successfully.

1.2 Part 2 - Review Predictive Models and Python Proficiency

I'll build a simple linear regression for the USA housing dataset from kaggle. The methodology will be as follows:

- Import the data
- Format any incorrect datatypes
- Check for NAs
- Verify distributions of random variables with pyplot
- Ensure dependent variable is normally distributed. If not, log-transform it.
- Scale the data
- Split dataset in to train and test sets
- Build LinearRegression object and fit the data
- Predict on the test dataset

[1]:
```
# Import needed libraries.

import pandas as pd                                            # pandas
import numpy as np                                             # numpy. Here mainly to␣
 ↪demonstrate it's installed.
from matplotlib import pyplot as plt                           # plotting
import seaborn as sns                                          # for the correlation␣
 ↪heatmap
from scipy import stats
import statsmodels.api as sm                                   # for a coefficient␣
 ↪summary
from sklearn.model_selection import train_test_split    # easy data splitting
from sklearn.linear_model import LinearRegression       # Linear model class
from sklearn.preprocessing import StandardScaler        # Scale the data using␣
 ↪the x - mu/sigma
from sklearn.metrics import mean_squared_error, r2_score # R2 and MSE
```

```
from math import sqrt
import statistics
%matplotlib inline
```

[80]:
```
data = pd.read_csv('USA_Housing.csv')
```

1.3 Exploratory Data Analysis

[81]:
```
data.head()
```

[81]:
```
   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

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09     23086.800503  1.059034e+06
1                          3.09     40173.072174  1.505891e+06
2                          5.13     36882.159400  1.058988e+06
3                          3.26     34310.242831  1.260617e+06
4                          4.23     26354.109472  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701…
1  188 Johnson Views Suite 079\nLake Kathleen, CA…
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482…
3                          USS Barnett\nFPO AP 44820
4                         USNS Raymond\nFPO AE 09386
```

[82]:
```
data.shape
```

[82]:
```
(5000, 7)
```

[83]:
```
data.columns
```

[83]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

[84]:
```
# With the exception of address, all datatypes are numeric, so we can proceed␣
↪without any data cleanup.
data.describe()
```

[84]:
```
         Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
count         5000.000000          5000.000000                5000.000000
mean         68583.108984             5.977222                   6.987792
std          10657.991214             0.991456                   1.005833
min          17796.631190             2.644304                   3.236194
25%          61480.562388             5.322283                   6.299250
50%          68804.286404             5.970429                   7.002902
75%          75783.338666             6.650808                   7.665871
max         107701.748378             9.519088                  10.759588

         Avg. Area Number of Bedrooms  Area Population         Price
count                    5000.000000      5000.000000  5.000000e+03
mean                        3.981330     36163.516039  1.232073e+06
std                         1.234137      9925.650114  3.531176e+05
min                         2.000000       172.610686  1.593866e+04
25%                         3.140000     29403.928702  9.975771e+05
50%                         4.050000     36199.406689  1.232669e+06
75%                         4.490000     42861.290769  1.471210e+06
max                         6.500000     69621.713378  2.469066e+06
```

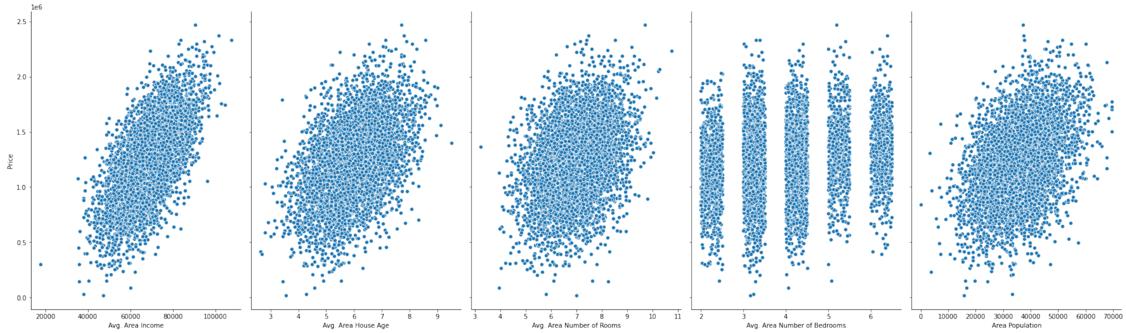
[85]:
```
# Address isn't something we really need for this exercise. However, in the␣
↪real world it would be best to parse
# out the address into geographic regions in order to get a better sense of␣
↪what areas have higher incomes and higher
# home prices. Obviously not all of the variance of home prices can be␣
↪explained by a few factors about the house itself
# or the population numbers.

data = data.drop('Address', axis =1)
```

[86]:
```
data.shape
```

[86]: (5000, 6)

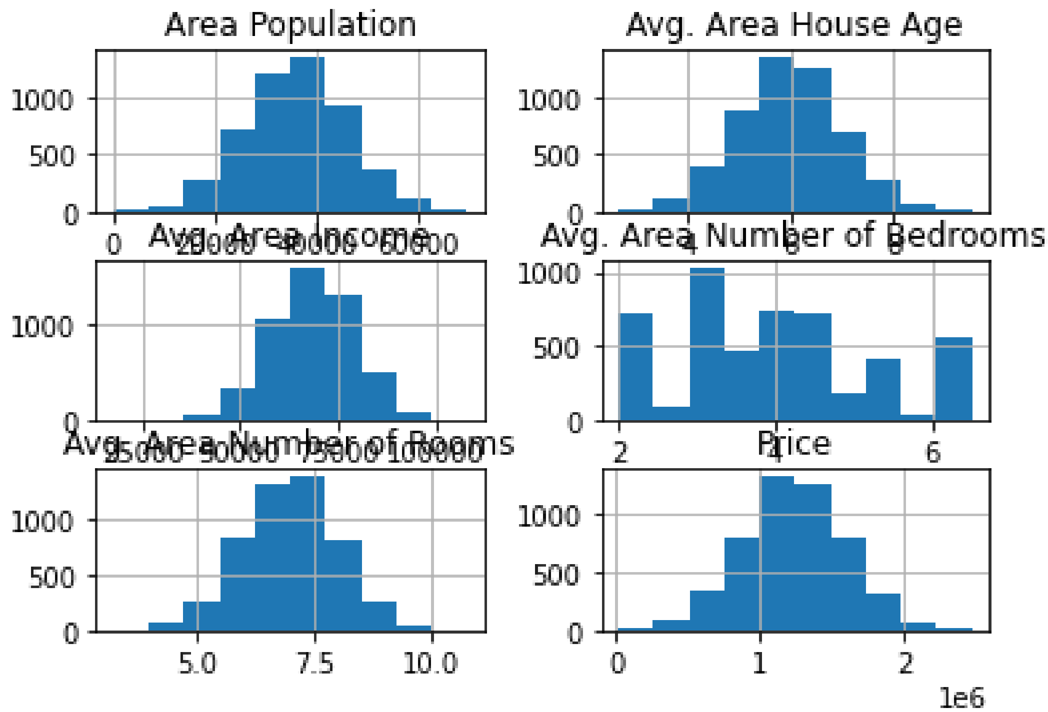
[87]:
```
# From this graph, we an see that there are some good indicators of price.␣
↪Clearly there is positive correlation
# between income and population. This makes sense since people with larger␣
↪incomes can buy larger houses. Also, prices
# tend to increase in cities with high populations or aggressive housing␣
↪restrictions.
sns.pairplot(data, x_vars=['Avg. Area Income','Avg. Area House Age','Avg. Area␣
↪Number of Rooms','Avg. Area Number of Bedrooms','Area Population'],␣
↪y_vars='Price',height=7, aspect=0.7, kind='scatter')
```

[87]: <seaborn.axisgrid.PairGrid at 0x7f6643c207d0>



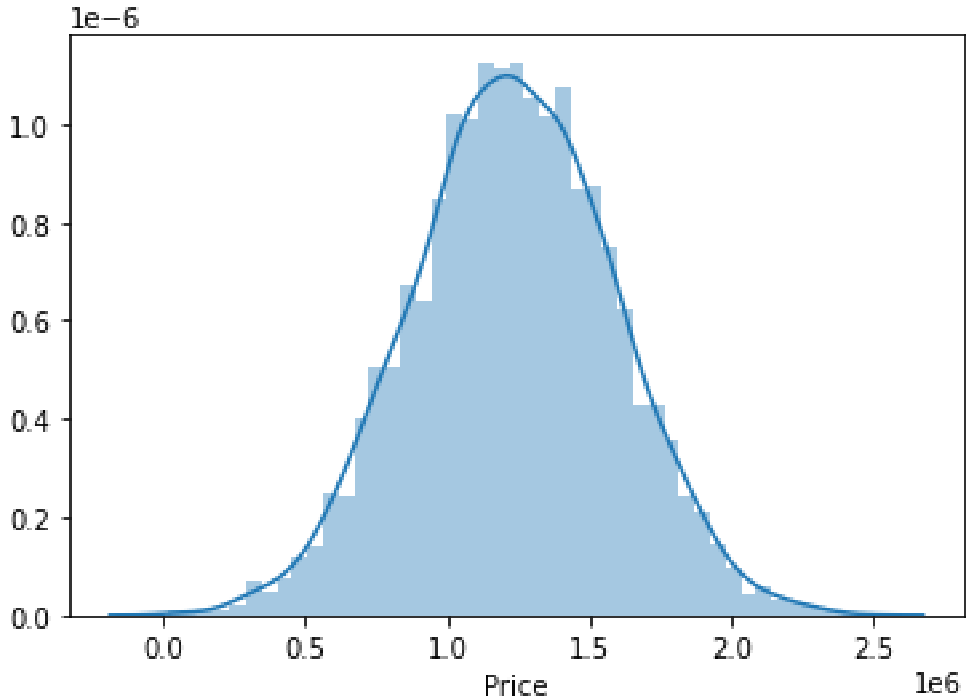
[88]:
```
# Works with pyplot here as well.

data.hist()
plt.show()
```

```
/home/smartinez/.local/lib/python3.7/site-
packages/pandas/plotting/_matplotlib/tools.py:298: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().rowspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
/home/smartinez/.local/lib/python3.7/site-
packages/pandas/plotting/_matplotlib/tools.py:298: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().colspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
/home/smartinez/.local/lib/python3.7/site-
packages/pandas/plotting/_matplotlib/tools.py:304: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().rowspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
/home/smartinez/.local/lib/python3.7/site-
packages/pandas/plotting/_matplotlib/tools.py:304: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().colspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
```



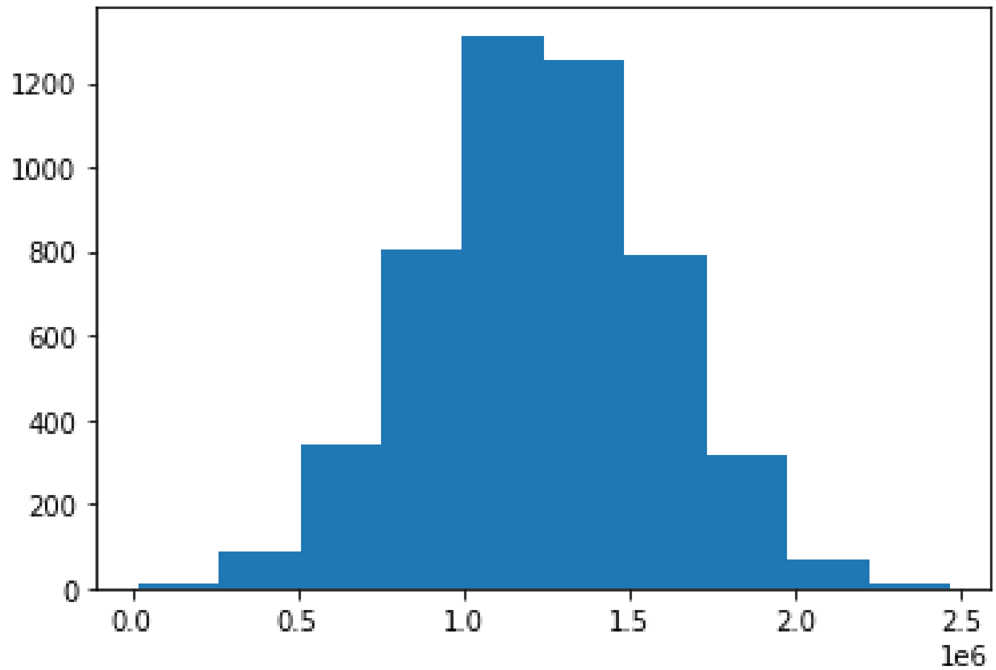
[89]:
```
# We can see from the distplot that our price is fairly normal and there aren't␣
↪any concerning outliers. We don't need
# to log-transform and can simply move forward with the model.

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

[89]: <matplotlib.axes._subplots.AxesSubplot at 0x7f664c765350>



[90]:
```
# This can be done with plt too:

plt.hist(data['Price'])
```

[90]:
```
(array([  11.,   88.,  344.,  804., 1313., 1252.,  793.,  314.,   68.,
          13.]),
 array([  15938.65792329,  261251.35154843,  506564.04517357,
         751876.73879871,  997189.43242385, 1242502.126049  ,
        1487814.81967414, 1733127.51329928, 1978440.20692442,
        2223752.90054956, 2469065.5941747 ]),
 <a list of 10 Patch objects>)
```



[91]:
```
# There are no missing values, so no need to impute values or drop rows.

data.isnull().values.any()
```

[91]: False

[92]:
```
# Price and income are the highest correlated values, but still not too bad.␣
 ↪This is irrelevant since price is our dependent
# variable. As expected, this will likely be the largest contributor in terms␣
 ↪of coefficient preidictors.

sns.heatmap(data.corr())
```

[92]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6643d22d10>

Because the variation is so different between home price, income, and the other continuous variables, it's best to scale this data. The dependent variable doesn't need to be scaled because the model will set the parameters to get the minimum cost function. We'd only need to scale y_test if y_train had been scaled.

The standard scaler function is:

$$\frac{x_i - \mu}{\sigma}$$

[93]:
```
X = data.drop('Price', axis=1)
y = data['Price']

# The standard scaler puts the data into a numpy array which is a bit less␣
↪flexible than the dataframe.
scl = StandardScaler()
X = pd.DataFrame(scl.fit_transform(X))

# 20% test, 80% train
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    random_state = 69)
```

1.4 Model Building and Fitment

Now we will build a linear model. This will be the de facto ordinary least squares regression.

The function is as follows:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \epsilon$$

In order to ensure model accuracy, I'll use the coefficient of determination, or R^2

The function is as follows:

$$SS_{tot} = \sum_i (y_i - \bar{y})^2$$

$$SS_{res} = \sum_i e_i^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

[94]:
```
model = LinearRegression()

model.fit(X_train, y_train)
```

[94]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[95]:
```
# These are the scaled coefs. While they are correct, they are a bit difficult␣
↪to interpret.

model.coef_
```

[95]:
```
array([228853.64224356, 164435.84671342, 121054.20052859,   1961.41011388,
       151825.60512927])
```

[96]:
```
# We can unscale them for a more business-friendly explanation.

coefs2 = pd.DataFrame(model.coef_/scl.scale_)
coefs2
```

[96]:
```
               0
0      21.474641
1  165869.451531
2  120364.195574
```

```
3    1589.455516
4      15.297818
```

With our coefficients now computed, we can view the actual equation that we have built:

$\hat{y} = 1231277.0248333374 + 228853.64224356x_1 + 164435.84671342x_2 + 121054.20052859x_3 + 1961.41011388x_4 + 151825$

```
[97]: coefficients = np.true_divide(model.coef_,  scl.scale_)
intercept = model.intercept_ - np.dot(coefficients, scl.mean_)
```

```
[98]: # The intercept is what we expect mean price to be, or when X_i = 0. This is␣
↪also scaled so we can also print the non-scaled

print(f'Scaled int: {model.intercept_}')
print(f'Raw int: {intercept}')
```

```
Scaled int: 1231277.0248333374
Raw int: -2633590.115971513
```

```
[99]: y_hat = model.predict(X_test)
```

1.5 Model Testing

First I'll do it programatically using the functions for MSE and R2.

```
[100]: mse = mean_squared_error(y_test, y_hat)
r2 = r2_score(y_test, y_hat)

print(f'Mean Square error: {mse}')
print(f'R2 score: {r2}')
```

```
Mean Square error: 9943170477.493637
R2 score: 0.9198377011002263
```

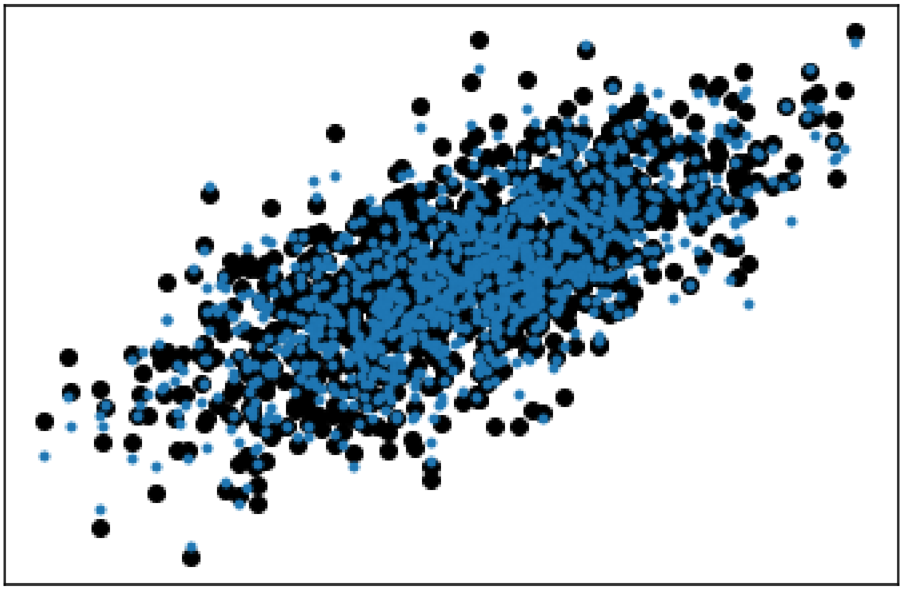
```
[105]: print(f'RMSE Error: {sqrt(mse)}')
```

```
RMSE Error: 99715.44753694703
```

1.6 Plot the predictions

```
[104]: # plot the graph. This example refers to the income.

plt.scatter(X_test[0], y_test, color = "black")



plt.plot(X_test[0], y_hat, '.')
plt.xticks(())
```

```
plt.yticks(())
plt.show()
```



Here the R^2 and mse can be calculated manually:

[103]:
```
SSR = ((y_test - y_hat) ** 2).sum()
SST = ((y_test - y_test.mean()) ** 2).sum()
```

[30]:
```
R_manual = 1-(SSR/SST)
```

[31]:
```
r2_adjusted = 1 - (1-R_manual)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

[32]:
```
print(f'R-squared by hand: {R_manual}')
print(f'Adjusted R2: {r2_adjusted}')
```

```
R-squared by hand: 0.9198377011002263
Adjusted R2: 0.9194344702204488
```

1.7 Reference model with statsmodels

Stats models provides a nice R-like summary of the coefficients so we'll take a look at that for p-values.

[112]:
```
X_const = sm.add_constant(X_train) # adding a constant

mod = sm.OLS(y_train, X_train).fit()
predictions = mod.predict(X_test)
```

```
print(mod.summary())
```

```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                  Price   R-squared (uncentered):
0.071
Model:                            OLS   Adj. R-squared (uncentered):
0.070
Method:                 Least Squares   F-statistic:
60.91
Date:                Sun, 06 Sep 2020   Prob (F-statistic):
2.35e-61
Time:                        20:12:50   Log-Likelihood:
-61783.
No. Observations:                4000   AIC:
1.236e+05
Df Residuals:                    3995   BIC:
1.236e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
0           2.315e+05   1.96e+04     11.787      0.000    1.93e+05     2.7e+05
1           1.656e+05   1.96e+04      8.461      0.000    1.27e+05    2.04e+05
2           1.143e+05   2.22e+04      5.148      0.000    7.08e+04    1.58e+05
3           5099.5414   2.21e+04      0.231      0.817   -3.82e+04    4.84e+04
4           1.567e+05   1.93e+04      8.127      0.000    1.19e+05    1.95e+05
==============================================================================
Omnibus:                        4.121   Durbin-Watson:                   0.013
Prob(Omnibus):                  0.127   Jarque-Bera (JB):                3.722
Skew:                           0.010   Prob(JB):                        0.156
Kurtosis:                       2.852   Cond. No.                         1.64
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

2 Conclusion

Based on our model, we found that the coefficient of determination was around 91% which indicated quite a good fit.

For a single unit increase in income, the house price increases by 21.4. Number of rooms had a huge coefficient, indicating that for every additional room, the house price increased by 120,364 dollars. This makes sense since, aside from the regular rooms (kitchen, bedroom, dining room, etc), specialty rooms are only for luxury homes. Things such as wine cellars, second kitchens, game rooms, etc are likely to make up this large value.

While the mean squared error of 99715 seems a bit high, the cost of the homes is quite large, so this MSE is a pretty good fit, as is confirmed by our R^2_{adj} value.

Surprisingly to me, all regressors were significant except number of bedrooms. This did not really contribute to the model and could be removed. It's generally accepted that this is important since usually larger houses have more bedrooms. However, some of this impact could be within the "number of rooms" variable. It would be interesting to review the model variance without this value and try again.

```
[ ]:
```