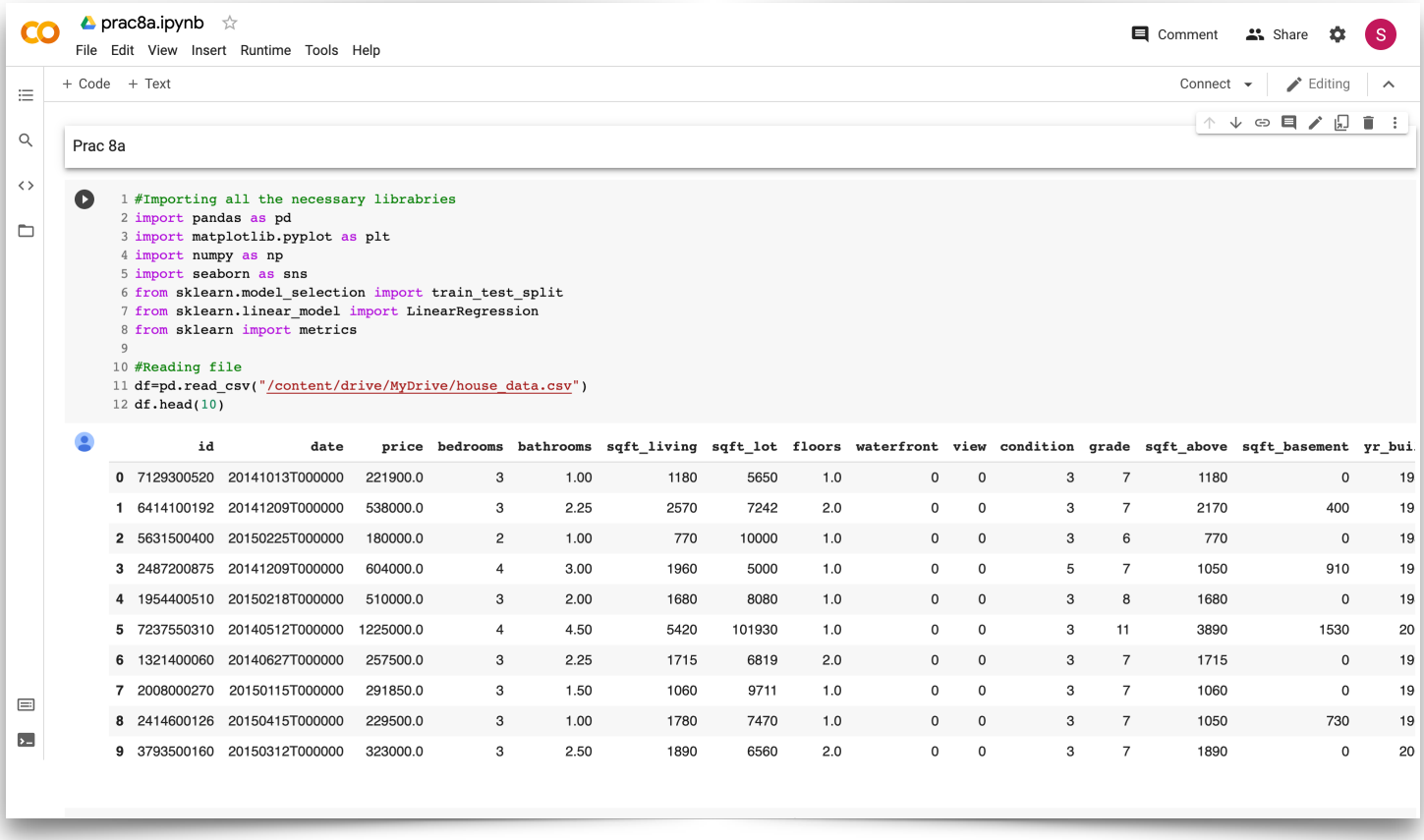
Aayush Shah
D1 - 19BCE245
30 April 2021

Practical 8

Statistical data analysis

- Design a python program which performs the linear regression operation on the given data to predict the house price. Also, visualize the data for different attributes.
- Design a python program which implements the bisection and false positioning method.
- Design a python program that generates the 100 random variables and finds out the mean, median and mode for the same.

8a Code and output :



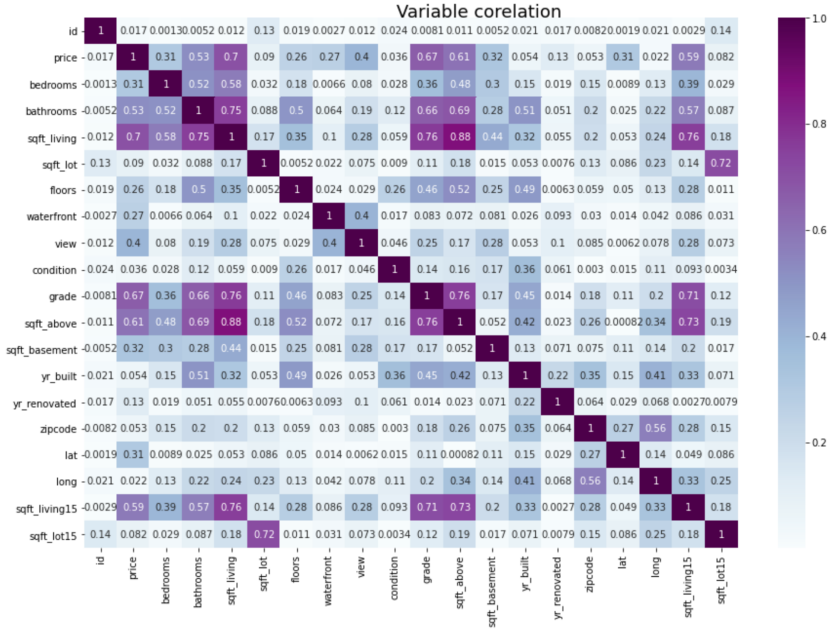
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_bui
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	19
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	19
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	3	6	770	0	19
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	5	7	1050	910	19
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	3	8	1680	0	19
5	7237550310	20140512T000000	1225000.0	4	4.50	5420	101930	1.0	0	0	3	11	3890	1530	20
6	1321400060	20140627T000000	257500.0	3	2.25	1715	6819	2.0	0	0	3	7	1715	0	19
7	2008000270	20150115T000000	291850.0	3	1.50	1060	9711	1.0	0	0	3	7	1060	0	19
8	2414600126	20150415T000000	229500.0	3	1.00	1780	7470	1.0	0	0	3	7	1050	730	19
9	3793500160	20150312T000000	323000.0	3	2.50	1890	6560	2.0	0	0	3	7	1890	0	20

```
[ ]  1 #Printing datatypes of all columns
     2 print(df.dtypes)
     3 #Plotting a pairplot between all variables to check which variables have a good linear regression curve
     4 sns.pairplot(df)
```

```
id                int64
date             object
price           float64
bedrooms          int64
bathrooms       float64
sqft_living       int64
sqft_lot          int64
floors          float64
waterfront        int64
view              int64
condition         int64
grade             int64
sqft_above        int64
sqft_basement     int64
yr_built          int64
yr_renovated      int64
zipcode           int64
lat             float64
long            float64
sqft_living15     int64
sqft_lot15        int64
dtype: object
<seaborn.axisgrid.PairGrid at 0x7f446d8ebc90>
```

```
[ ]  1 #Plotting a heatmap
     2 c=df.corr().abs()
     3 figure, axes=plt.subplots(figsize=(15,10))
     4 figure.suptitle('Variable corelation', fontsize=18,y=.90)
     5 heatmap=sns.heatmap(c,cmap="BuPu",annot=True)
     6 heatmap
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f445375dd50>
```



```
[ ]
alising model for Linear Regression
earRegression()

alising x and y
"sqft_above","sqft_living","sqft_living15","sqft_lot","waterfront","sqft_basement","bathrooms","bedrooms","view","grade","condition","sqft_lot15","long","
price']

ng values of x and y the linear Regression Model
(x,y)
acy score
"Accuracy score: ",lr.score(x,y))

g train_test_split function to split the data into two subsets: test and train
x_te, y_tr, y_te = train_test_split(x, y, test_size=0.1, random_state=1)

"Total number of Samples",x.shape[0])
"Number of test samples : ", x_te.shape[0])
"Number of training samples: ",x_tr.shape[0])
```

```
Accuracy score:  0.6998463505666741
Total number of Samples 21613
Number of test samples :  2162
Number of training samples:  19451
```

```
[ ]
#Re-initialising the Model
lr=LinearRegression()
lr.fit(x_tr, y_tr)

# Evaluation for training set
y_train_predict = lr.predict(x_tr)
mse = metrics.mean_squared_error(y_tr, y_train_predict)
rmse = (np.sqrt(metrics.mean_squared_error(y_tr, y_train_predict)))
r2 = metrics.r2_score(y_tr, y_train_predict)
mae = metrics.mean_absolute_error(y_tr, y_train_predict)
print("Performance for training set:")
print('Mean Squared Error: ',mse)
print('Root Mean Squared Error: ',rmse)
print('R2 value : ',r2)
print('Mean Absolute Error: ',mae)

# Evaluation for testing set
y_test_predict = lr.predict(x_te)
mse = metrics.mean_squared_error(y_te, y_test_predict)
rmse = (np.sqrt(metrics.mean_squared_error(y_te, y_test_predict)))
r2 = metrics.r2_score(y_te, y_test_predict)
mae = metrics.mean_absolute_error(y_te, y_test_predict)
print("\n\nPerformance for testing set:")
print('Mean Squared Error: ',mse)
print('Root Mean Squared Error: ',rmse)
print('R2 value : ',r2)
print('Mean Absolute Error: ',mae)
```

```
Performance for training set:
Mean Squared Error:  39275697159.625984
Root Mean Squared Error:  198180.9707303554
R2 value :  0.7022613191454392
Mean Absolute Error:  124887.39125709626


Performance for testing set:
Mean Squared Error:  51240382072.119606
Root Mean Squared Error:  226363.38500764561
R2 value :  0.6805568369162995
Mean Absolute Error:  132127.29403099057
```

8b Code :

Bisection :

```
1. #!/usr/bin/env python3
2.
3. # Design a python program which implements the bisection and
   false positioning method.
4.
5. def calculate_value(x):
```

```
6.    return sum([float(a)*pow(float(x),float(b)) for a,b in
   zip(coeff_list,power_list)])
7.
8. def bisection(a,b):
9.    if calculate_value(a)*calculate_value(b)>=0:
10.        return False,0
11.
12.   c = a
13.   while (b-a)>=0.01:
14.        c = (a+b)/2
15.        if calculate_value(c)==0.0:
16.             return True,c
17.        if calculate_value(c)*calculate_value(a)<0:
18.             b=c
19.        else:
20.             a=c
21.   return True,c
22.
23.number_of_terms = int(input("How many terms are there in
   the equation : "))
24.coeff_list = []
25.power_list = []
26.for index in range(number_of_terms):
27.   coeff = float(input(f"\n\tEnter coefficient for term
   {index+1} : "))
28.   power = float(input(f"\tEnter power for term {index+1} :
   "))
29.   coeff_list.append(coeff)
30.   power_list.append(power)
31.print("\nEntered formula : ",end="")
32.for index in range(number_of_terms-1):
33.   print(f"{coeff_list[index]}x^({power_list[index]})",end="
   + ")
34.print(f"{coeff_list[number_of_terms-1]}
   x^({power_list[number_of_terms-1]})")
35.
36.while True:
37.   a = float(input("Enter a : "))
38.   b = float(input("Enter b : "))
39.
40.   choice,answer = bisection(a, b)
41.   if choice:
42.        print("Solution is :",round(answer,5))
```

```
43.         break
44.   else:
45.         print("You have not assumed right a and b")
46.         a = float(input("Enter a : "))
47.         b = float(input("Enter b : "))
48.
49.
50.
51.
52.
53.#def calculate_value(x,coeff_list,power_list):
54.# counter = 0
55.# for i in range(len(coeff_list)):
56.#       counter += coeff_list[index] *
   pow(float(x),float(power_list[index]))
57.# return counter
```

Output :

```
How many terms are there in the equation : 3

    Enter coefficient for term 1 : 1
    Enter power for term 1 : 3

    Enter coefficient for term 2 : -1
    Enter power for term 2 : 2

    Enter coefficient for term 3 : 2
    Enter power for term 3 : 0

Entered formula : 1.0x^(3.0) + -1.0x^(2.0) + 2.0x^(0.0)
Enter a : -200
Enter b : 300
Solution is : -1.0025
```

Run Succeeded | Time 74 ms | Peak Memory 7.5M | Symbol | Tabs: 4 | 58 Lines, 1513 Characters

False Position :

```
#!/usr/bin/env python3

# Design a python program which implements the bisection and false positioning method.

def calculate_value(x):
   return sum([float(a)*pow(float(x),float(b)) for a,b in zip(coeff_list,power_list)])

def false_position(a,b):
   if calculate_value(a)*calculate_value(b)>=0:
        return False,0

   for i in range(MAX_ITER=1000000):
        c = (a * calculate_value(b) - b * calculate_value(a))/(calculate_value(b) - calculate_value(a))

        if calculate_value(c)==0:
             return True,c
        elif calculate_value(c)*calculate_value(a)<0:
             b = c
        else:
             a = c

   return True,c

number_of_terms = int(input("How many terms are there in the equation : "))
coeff_list = []
power_list = []
for index in range(number_of_terms):
   coeff = float(input(f"\n\tEnter coefficient for term {index+1} : "))
   power = float(input(f"\tEnter power for term {index+1} : "))
   coeff_list.append(coeff)
   power_list.append(power)
print("\nEntered formula : ",end="")
for index in range(number_of_terms-1):
```

```
34.   print(f"{coeff_list[index]}x^({power_list[index]})",end="
   + ")
35.print(f"{coeff_list[number_of_terms-1]}
   x^({power_list[number_of_terms-1]})")
36.
37.while True:
38.   a = float(input("Enter a : "))
39.   b = float(input("Enter b : "))
40.
41.   choice,answer = false_position(a, b)
42.   if choice:
43.         print("Solution is :",round(answer,5))
44.         break
45.   else:
46.         print("You have not assumed right a and b")
47.         a = float(input("Enter a : "))
48.         b = float(input("Enter b : "))
49.
50.#def calculate_value(x,coeff_list,power_list):
51.# counter = 0
52.# for i in range(len(coeff_list)):
53.#       counter += coeff_list[index] *
   pow(float(x),float(power_list[index]))
54.# return counter
```

Output :

```
How many terms are there in the equation : 3

    Enter coefficient for term 1 : 1
    Enter power for term 1 : 3

    Enter coefficient for term 2 : -1
    Enter power for term 2 : 2

    Enter coefficient for term 3 : 2
    Enter power for term 3 : 0

Entered formula : 1.0x^(3.0) + -1.0x^(2.0) + 2.0x^(0.0)
Enter a : -200
Enter b : 300
Solution is : -1.0

Run Succeeded | Time 0:00:12 | Peak Memory 7.3M | false_position | Tabs: 4 | Line 42, Column 11
```

8c Code :

```python
#!/usr/bin/env python3

# Design a python program that generates the 100 random variables and finds out the mean, median and mode for the same.
import random
import statistics

randomlist = random.sample(range(100000000), 100)
print("Random generated list : ",randomlist)
print("Mean : ",statistics.mean(randomlist))
print("Median : ",statistics.median(randomlist))
print("Mode : ",statistics.mode(randomlist))
```

Output :

```
Random generated list :  [11027479, 83648441, 8249896, 33281212,
    94374492, 2813753, 65113437, 35565492, 30878045, 64428301,
    4424312, 36275609, 23806164, 73508951, 35792559, 59349840,
    43944462, 6488383, 8778659, 30840269, 31633674, 60097610,
    43740373, 42763567, 24017123, 36083891, 24235888, 92458400,
    7398595, 59266034, 96344322, 46629309, 61023809, 20837426,
    48739260, 13477024, 71825963, 29211485, 91041226, 44350520,
    29739931, 73844047, 86223589, 7107940, 8267147, 90416258,
    70217114, 868910, 48700957, 1382113, 58286691, 21617708,
    92420575, 74178352, 77375490, 37171277, 61672382, 90533917,
    2417581, 78156340, 1892024, 84210522, 71626341, 99486759,
    37521917, 40567425, 98863612, 3674578, 25080083, 85057254,
    16246447, 30320895, 45901780, 37932328, 29151680, 76277622,
    68014433, 51666125, 20501113, 67891369, 72872380, 49250712,
    83081930, 41463753, 43359038, 13818538, 49127335, 55353921,
    82438971, 55222911, 79514559, 71568902, 2466083, 57846271,
    86152139, 64801101, 61454459, 24959512, 90437140, 19937683]
Mean :  48053752.19
Median :  46265544.5
Mode :  11027479

Run Succeeded | Time 75 ms | Peak Memory 7.8M | Symbol | Tabs: 4 | 12 Lines, 410 Characters
```

Conclusion :

Here we have learned about Linear Regression, Bisection and false Position method for finding roots of an equation and some statistical data analysis from mean, median and mode.

- **Linear Regression** : which is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables.
- **Bisection** : The method is also called the interval halving method, the binary search method or the dichotomy method. This method is used to find root of an equation in a given interval that is value of 'x' for which f(x) = 0 .
- **False Position** :
  - This method is also known as Regula Falsi or The Method of Chords. It is some what similar to bisection method like
    - Same Assumptions: This method also assumes that function is continuous in [a, b] and given two numbers 'a' and 'b' are such that f(a) * f(b) < 0.
    - Always Converges: like Bisection, it always converges, usually considerably faster than Bisection–but sometimes very much more slowly than Bisection.
  - But It differs in the fact that
    - we make a chord joining the two points [a, f(a)] and [b, f(b)]. We consider the point at which the chord touches the x axis and named it as c.