02_multiple_regression

February 3, 2021

1 1) Importing all the libraries required

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

2 2) Exploring the data

[2]:
```
df_test=pd.read_csv('kc_house_test_data.csv')
df_train=pd.read_csv('kc_house_train_data.csv')
```

[3]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17384 entries, 0 to 17383
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             17384 non-null  int64
 1   date           17384 non-null  object
 2   price          17384 non-null  float64
 3   bedrooms       17384 non-null  int64
 4   bathrooms      17384 non-null  float64
 5   sqft_living    17384 non-null  int64
 6   sqft_lot       17384 non-null  int64
 7   floors         17384 non-null  float64
 8   waterfront     17384 non-null  int64
 9   view           17384 non-null  int64
 10  condition      17384 non-null  int64
 11  grade          17384 non-null  int64
 12  sqft_above     17384 non-null  int64
 13  sqft_basement  17384 non-null  int64
 14  yr_built       17384 non-null  int64
```

```
 15  yr_renovated   17384 non-null  int64
 16  zipcode        17384 non-null  int64
 17  lat            17384 non-null  float64
 18  long           17384 non-null  float64
 19  sqft_living15  17384 non-null  int64
 20  sqft_lot15     17384 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 2.8+ MB
```

[4]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4229 entries, 0 to 4228
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             4229 non-null   int64
 1   date           4229 non-null   object
 2   price          4229 non-null   float64
 3   bedrooms       4229 non-null   int64
 4   bathrooms      4229 non-null   float64
 5   sqft_living    4229 non-null   int64
 6   sqft_lot       4229 non-null   int64
 7   floors         4229 non-null   float64
 8   waterfront     4229 non-null   int64
 9   view           4229 non-null   int64
 10  condition      4229 non-null   int64
 11  grade          4229 non-null   int64
 12  sqft_above     4229 non-null   int64
 13  sqft_basement  4229 non-null   int64
 14  yr_built       4229 non-null   int64
 15  yr_renovated   4229 non-null   int64
 16  zipcode        4229 non-null   int64
 17  lat            4229 non-null   float64
 18  long           4229 non-null   float64
 19  sqft_living15  4229 non-null   int64
 20  sqft_lot15     4229 non-null   int64
dtypes: float64(5), int64(15), object(1)
memory usage: 693.9+ KB
```

[5]:
```
df_train.describe()
```

[5]:
```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  1.738400e+04  1.738400e+04  17384.000000  17384.000000  17384.000000
mean   4.574349e+09  5.393666e+05      3.369363      2.115048   2080.029510
std    2.872356e+09  3.696912e+05      0.906468      0.771783    921.630888
min    1.000102e+06  7.500000e+04      0.000000      0.000000    290.000000
```

```
25%    2.124087e+09  3.200000e+05      3.000000      1.750000   1420.000000
50%    3.892800e+09  4.500000e+05      3.000000      2.250000   1910.000000
75%    7.304301e+09  6.400000e+05      4.000000      2.500000   2550.000000
max    9.900000e+09  7.700000e+06     10.000000      8.000000  13540.000000

           sqft_lot        floors    waterfront          view     condition  \
count  1.738400e+04  17384.000000  17384.000000  17384.000000  17384.000000
mean   1.509191e+04      1.494248      0.007651      0.236079      3.410780
std    4.145927e+04      0.539443      0.087136      0.768008      0.649792
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000
25%    5.049500e+03      1.000000      0.000000      0.000000      3.000000
50%    7.616000e+03      1.500000      0.000000      0.000000      3.000000
75%    1.066525e+04      2.000000      0.000000      0.000000      4.000000
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  17384.000000  17384.000000   17384.000000  17384.000000  17384.000000
mean       7.655028   1787.844512     292.184998   1971.152727     83.107973
std        1.169818    827.107595     444.404136     29.328722    398.692283
min        1.000000    290.000000       0.000000   1900.000000      0.000000
25%        7.000000   1200.000000       0.000000   1952.000000      0.000000
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000

            zipcode           lat          long  sqft_living15     sqft_lot15
count  17384.000000  17384.000000  17384.000000   17384.000000   17384.000000
mean   98077.936896     47.559313   -122.213281    1985.994995   12776.380867
std       53.525617      0.138703      0.140906     686.512835   27175.730523
min    98001.000000     47.159300   -122.519000     399.000000     651.000000
25%    98033.000000     47.468650   -122.328000    1490.000000    5100.000000
50%    98065.000000     47.571400   -122.229000    1840.000000    7620.000000
75%    98117.000000     47.677625   -122.125000    2360.000000   10065.250000
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000
```

[6]:
```
df_train.head()
```

[6]:
```
           id             date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900.0         3       1.00         1180
1  6414100192  20141209T000000  538000.0         3       2.25         2570
2  5631500400  20150225T000000  180000.0         2       1.00          770
3  2487200875  20141209T000000  604000.0         4       3.00         1960
4  1954400510  20150218T000000  510000.0         3       2.00         1680

   sqft_lot  floors  waterfront  view  ...  grade  sqft_above  sqft_basement  \
0      5650     1.0           0     0  ...      7        1180              0
1      7242     2.0           0     0  ...      7        2170            400
```

```
2     10000     1.0           0      0  ...      6         770              0
3      5000     1.0           0      0  ...      7        1050            910
4      8080     1.0           0      0  ...      8        1680              0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

3 3) Added 4 new combination of existing features

[7]:
```
# a) 'bedrooms_squared' = 'bedrooms'*'bedrooms'
df_train['bedrooms_squared']=df_train['bedrooms']*df_train['bedrooms']
df_test['bedrooms_squared']=df_test['bedrooms']*df_test['bedrooms']
# b) 'bed_bath_rooms' = 'bedrooms'*'bathrooms'
df_train['bed_bath_rooms']=df_train['bedrooms']*df_train['bathrooms']
df_test['bed_bath_rooms']=df_test['bedrooms']*df_test['bathrooms']
# c) 'log_sqft_living' = log('sqft_living')
df_train['log_sqft_living']=df_train['sqft_living'].apply( lambda  x : np.
↪log(x))
df_test['log_sqft_living']=df_test['sqft_living'].apply( lambda  x : np.log(x))
# d)  'lat_plus_long' =  'lat' + 'long'
df_train['lat_plus_long']=df_train['lat']+df_train['long']
df_test['lat_plus_long']=df_test['lat']+df_test['long']
```

[8]:
```
print(df_test['bedrooms_squared'].mean())
print(df_test['bed_bath_rooms'].mean())
print(df_test['log_sqft_living'].mean())
print(df_test['lat_plus_long'].mean())
```

```
12.4466777015843
7.5039016315913925
7.550274679645921
-74.65333355403185
```

4 4) Fitting a multiple features regression model

[9]:
```
model_1=LinearRegression().
 ↪fit(df_train[['sqft_living','bedrooms','bathrooms','lat','long']],df_train['price'])
print(model_1.intercept_)
print(model_1.coef_)
```

```
-69075726.79256989
[ 3.12258646e+02 -5.95865332e+04  1.57067421e+04  6.58619264e+05
 -3.09374351e+05]
```

[10]:
```
model_2=LinearRegression().
 ↪fit(df_train[['sqft_living','bedrooms','bathrooms','lat','long','bed_bath_rooms']],df_train
print(model_2.intercept_)
print(model_2.coef_)
```

```
-66867968.871078975
[ 3.06610053e+02 -1.13446368e+05 -7.14613083e+04  6.54844630e+05
 -2.94298969e+05  2.55796520e+04]
```

[11]:
```
model_3=LinearRegression().
 ↪fit(df_train[['sqft_living','bedrooms','bathrooms','lat','long','bed_bath_rooms','bedrooms_
print(model_2.intercept_)
print(model_2.coef_)
```

```
-66867968.871078975
[ 3.06610053e+02 -1.13446368e+05 -7.14613083e+04  6.54844630e+05
 -2.94298969e+05  2.55796520e+04]
```

[12]:
```
def cost_RSS(Y_train,Y_predicted):
  error=Y_train-Y_predicted
  error_squared=error**2
  RSS=error_squared.sum()
  return RSS
```

[13]:
```
X1=df_train[['sqft_living','bedrooms','bathrooms','lat','long']]
X2=df_train[['sqft_living','bedrooms','bathrooms','lat','long','bed_bath_rooms']]
X3=df_train[['sqft_living','bedrooms','bathrooms','lat','long','bed_bath_rooms','bedrooms_squa
Y=df_train['price']
Y1_pred=model_1.predict(X1)
Y2_pred=model_2.predict(X2)
Y3_pred=model_3.predict(X3)
```

[14]:
```
print(cost_RSS(Y,Y1_pred))
print(cost_RSS(Y,Y2_pred))
print(cost_RSS(Y,Y3_pred))
```

```
967879963049545.8
```

958419635074068.8
903436455050478.0

[15]:
```
X1=df_test[['sqft_living','bedrooms','bathrooms','lat','long']]
X2=df_test[['sqft_living','bedrooms','bathrooms','lat','long','bed_bath_rooms']]
X3=df_test[['sqft_living','bedrooms','bathrooms','lat','long','bed_bath_rooms','bedrooms_squar
Y=df_test['price']
Y1_pred=model_1.predict(X1)
Y2_pred=model_2.predict(X2)
Y3_pred=model_3.predict(X3)
```

[16]:
```
print(cost_RSS(Y,Y1_pred))
print(cost_RSS(Y,Y2_pred))
print(cost_RSS(Y,Y3_pred))
```

225500469795489.56
223377462976467.2
259236319207179.3

5 5) Implementing Multiple Linear Regression

[17]:
```
df_train['constant-feature']=1
H=df_train[['constant-feature','sqft_living','bedrooms','bathrooms','lat','long']]␣
↪# Feature Matrix (with n data points and 6 features -> n * 6
```

[18]:
```
H
```

[18]:

	constant-feature	sqft_living	bedrooms	bathrooms	lat	long
0	1	1180	3	1.00	47.5112	-122.257
1	1	2570	3	2.25	47.7210	-122.319
2	1	770	2	1.00	47.7379	-122.233
3	1	1960	4	3.00	47.5208	-122.393
4	1	1680	3	2.00	47.6168	-122.045
…	…	…	…	…	…	…
17379	1	3510	4	3.50	47.5537	-122.398
17380	1	1310	3	2.50	47.5773	-122.409
17381	1	1530	3	2.50	47.6993	-122.346
17382	1	1600	3	2.50	47.5345	-122.069
17383	1	1020	2	0.75	47.5941	-122.299

[17384 rows x 6 columns]

[19]:
```
Y=df_train['price'] # y_actual is what we ahave in dataset
                    # (y_predicted = w0*1 + w1*h1(x) + w2*h2(x) + w3*h3(x) +␣
↪w4*h4(x) + w5*h5(x)
                    # (since y_pred = H(n*6) * w(6*1) so y_pred(n*1) )
```

[20]:
```
# To minimize cost_RSS we find the best w (6 *1)
```

[21]:
```
# We find the gradient of RSS in terms of w
# gradient of RSS = -2 * H_transpose * [Y - H * w]  ( Here , Y , H and w are
↪all matrices)
```

5.1 a) Closed Form Solution

[22]:
```
# We make gradient=0
# Then we get : w = (H_tanspose * H)^-1 * H_tanspose*Y
```

[23]:
```
H_transpose=np.transpose(H)
H_transpose
```

[23]:
```
                          0         1         2          3          4      \
constant-feature     1.0000     1.000    1.0000     1.0000     1.0000
sqft_living       1180.0000  2570.000  770.0000  1960.0000  1680.0000
bedrooms             3.0000     3.000    2.0000     4.0000     3.0000
bathrooms            1.0000     2.250    1.0000     3.0000     2.0000
lat                 47.5112    47.721   47.7379    47.5208    47.6168
long              -122.2570  -122.319 -122.2330  -122.3930  -122.0450

                          5          6          7          8          9     …  \
constant-feature     1.0000     1.0000     1.0000     1.0000     1.0000  …
sqft_living       5420.0000  1715.0000  1060.0000  1780.0000  1890.0000  …
bedrooms             4.0000     3.0000     3.0000     3.0000     3.0000  …
bathrooms            4.5000     2.2500     1.5000     1.0000     2.5000  …
lat                 47.6561    47.3097    47.4095    47.5123    47.3684  …
long              -122.0050  -122.3270  -122.3150  -122.3370  -122.0310  …

                      17374      17375      17376      17377      17378  \
constant-feature     1.0000     1.0000     1.0000     1.0000     1.0000
sqft_living       4470.0000  1425.0000  1500.0000  2270.0000  1490.0000
bedrooms             5.0000     3.0000     3.0000     3.0000     3.0000
bathrooms            3.7500     2.5000     1.7500     2.5000     2.0000
lat                 47.6321    47.6963    47.3095    47.5389    47.5699
long              -122.2000  -122.3180  -122.0020  -121.8810  -122.2880

                      17379      17380      17381      17382      17383
constant-feature     1.0000     1.0000     1.0000     1.0000     1.0000
sqft_living       3510.0000  1310.0000  1530.0000  1600.0000  1020.0000
bedrooms             4.0000     3.0000     3.0000     3.0000     2.0000
bathrooms            3.5000     2.5000     2.5000     2.5000     0.7500
lat                 47.5537    47.5773    47.6993    47.5345    47.5941
long              -122.3980  -122.4090  -122.3460  -122.0690  -122.2990

[6 rows x 17384 columns]
```

[24]:
```
M=H_transpose@H
```

[25]:
```
M
```

[25]:
```
                   constant-feature   sqft_living      bedrooms     bathrooms  \
constant-feature       1.738400e+04  3.615923e+07  5.857300e+04  3.676800e+04
sqft_living            3.615923e+07  8.997745e+10  1.304171e+08  8.580820e+07
bedrooms               5.857300e+04  1.304171e+08  2.116370e+05  1.303208e+05
bathrooms              3.676800e+04  8.580820e+07  1.303208e+05  8.812025e+04
lat                    8.267711e+05  1.719835e+09  2.785670e+06  1.748711e+06
long                  -2.124556e+06 -4.418603e+09 -7.158096e+06 -4.493119e+06

                            lat          long
constant-feature   8.267711e+05 -2.124556e+06
sqft_living        1.719835e+09 -4.418603e+09
bedrooms           2.785670e+06 -7.158096e+06
bathrooms          1.748711e+06 -4.493119e+06
lat                3.932100e+07 -1.010425e+08
long              -1.010425e+08  2.596493e+08
```

[26]:
```
try:
    M_inverse = np.linalg.inv(M)
except:
    print("Inverse couldn't be found")
```

[27]:
```
M_inverse
```

[27]:
```
array([[ 4.87041429e+01, -7.96390688e-06,  2.23124770e-05,
        -7.32275902e-03, -8.88562076e-02,  3.63677063e-01],
       [-7.96390688e-06,  1.81926478e-10, -4.92259344e-08,
        -1.29334271e-07, -6.51079434e-08, -9.09998623e-08],
       [ 2.23124774e-05, -4.92259344e-08,  1.10683577e-04,
        -2.51719035e-05,  3.15493908e-05,  1.42380741e-05],
       [-7.32275902e-03, -1.29334271e-07, -2.51719035e-05,
         2.31035834e-04,  5.17712972e-06, -5.68000355e-05],
       [-8.88562076e-02, -6.51079434e-08,  3.15493908e-05,
         5.17712971e-06,  3.07998777e-03,  4.71370372e-04],
       [ 3.63677063e-01, -9.09998623e-08,  1.42380741e-05,
        -5.68000355e-05,  4.71370372e-04,  3.15705189e-03]])
```

[28]:
```
def closed_form_solution(Y,H):
    H_transpose=np.transpose(H)
    M=H_transpose@H
    try:
        M_inverse = np.linalg.inv(M)
    except:
        print("Inverse couldn't be found")
```

```
        return -1
    return M_inverse @ H_transpose @ Y
```

[29]:
```
w_best=closed_form_solution(Y,H)
```

[30]:
```
if(type(w_best)=='int'):
    print("Not possible to find closed form solution")
else:
    print(w_best)
```

```
0   -6.907573e+07
1    3.122586e+02
2   -5.958653e+04
3    1.570674e+04
4    6.586193e+05
5   -3.093744e+05
dtype: float64
```

[31]:
```
print(model_1.intercept_)
print(model_1.coef_)
```

```
-69075726.79256989
[ 3.12258646e+02 -5.95865332e+04  1.57067421e+04  6.58619264e+05
 -3.09374351e+05]
```

Note : you can see here that our values have matched the output previously done using there library

5.2 b) Gradient Descent Solution

[114]:
```
def gradientDescent(max_iterations,step_size,tolerance,Y,H):
    w_transpose=np.array(np.zeros(len(H.columns)),ndmin=2)
    w=np.transpose(w_transpose)
    H=np.array(H)
    H_transpose=np.transpose(H)
    Y_transpose=np.array(Y,ndmin=2)
    Y=np.transpose(Y_transpose)
    gradient_RSS = -2 * np.matmul(H_transpose,(np.subtract(Y,np.matmul(H,w))))
    converged=False
    count=0
    while(not converged and count<max_iterations):
        if(all(x <= tolerance for x in gradient_RSS)):
            converged=True
        else:
            w = w-  step_size * gradient_RSS
        count+=1
    print(w)
```

[115]:
```
gradientDescent(2000,7e-12,0.5e-7,Y,H)
```

```
[[ 2.62537785e+02]
 [ 6.62655192e+05]
 [ 9.35612039e+02]
 [ 6.27958386e+02]
 [ 1.24938408e+04]
 [-3.20850637e+04]]
```

[116]:

```python
print(model_1.intercept_)
print(model_1.coef_)
```

```
-69075726.79256989
[ 3.12258646e+02 -5.95865332e+04  1.57067421e+04  6.58619264e+05
 -3.09374351e+05]
```

Note : You can see that the best values of weights tried to reach the solution as close as possible using gradient descent but the challenge here is to select a appropriate value of step_size,tolerance

[ ]:

```python

```