PROGRAMMING REPORT

NUMBER 1

- **Step 1**: Use Dataframe.info and Dataframe.describe functions to check the dataset. Briefly summarize the information of the dataset.

```
data.info()
```

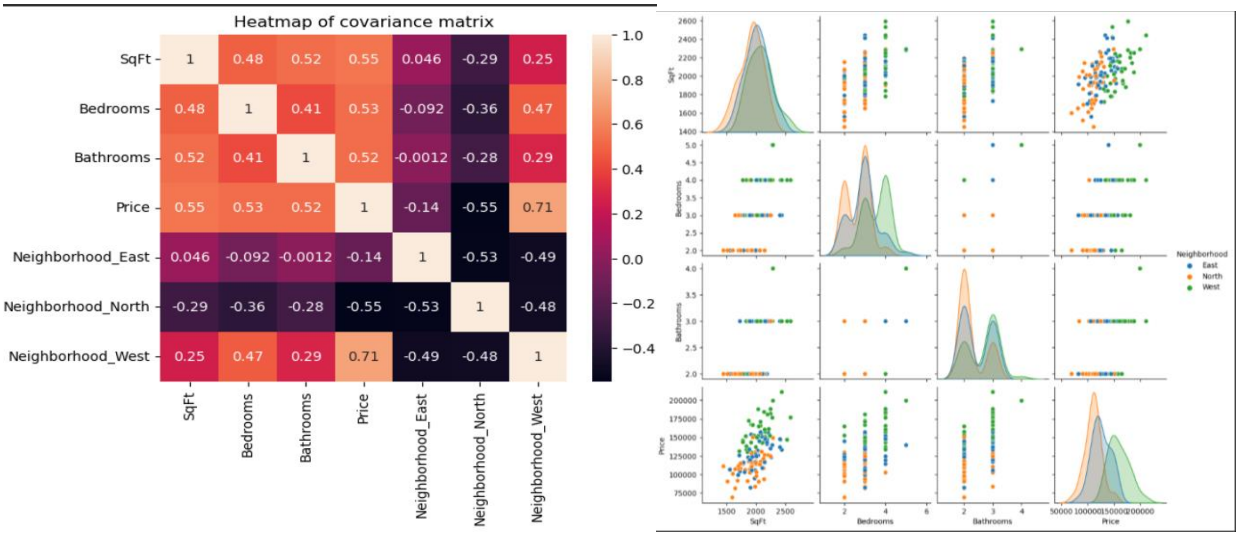
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128 entries, 0 to 127
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   SqFt          128 non-null    int64
 1   Bedrooms      128 non-null    int64
 2   Bathrooms     128 non-null    int64
 3   Neighborhood  128 non-null    category
 4   Price         128 non-null    int64
dtypes: category(1), int64(4)
memory usage: 4.4 KB
```

```
data.describe()
```

	SqFt	Bedrooms	Bathrooms	Price
count	128.000000	128.000000	128.000000	128.000000
mean	2000.937500	3.023438	2.445312	130427.343750
std	211.572431	0.725951	0.514492	26868.770371
min	1450.000000	2.000000	2.000000	69100.000000
25%	1880.000000	3.000000	2.000000	111325.000000
50%	2000.000000	3.000000	2.000000	125950.000000
75%	2140.000000	3.000000	3.000000	148250.000000
max	2590.000000	5.000000	4.000000	211200.000000

There is no incomplete data (NaN or Null) in the dataset. All datatypes are int64, except for "Neighborhood", which has been changed into category. The data.describe() method generate descriptive statistics of the data frame, including count, mean, standard deviation, minimum, first quantile, second quantile, third quantile, and the maximum. The prices of the house range from 69100 to 211200 with a mean price of 130427 and median 125950. The standard deviation in SqFt and Price indicates spreads in the features.

- **Step 2**: Use seaborn.heatmap function to plot the pairwise correlation on data. Briefly analyze the potential patterns between "Price" and other attributes.



Overall, the price in the north ranks the lowest since they have lower SqFt and the housing price starts at below 75,000. Prices are way expensive in the West, since they have more SqFt and the price starts at 125,000. Similarly, the numbers of bedrooms and bathrooms are fewer in the North compared to the West. Overall, in terms of the relationship between "Price"

and ”Neighborhood”, the ranking starting from the lowest is the following: North, East, West. The darker color in the heatmap indicates strong correlation, such as the correlation between ”Price” and ”Neighborhood_North”.

- **Step 3**: We could use ColumnTransformer function in sklearn.compose and OneHotEncoder function in sklearn.preprocessing to convert the category column into a one-hot numeric matrix in the dataset. Use sklearn library to split data into train and test subset.

	location__Neighborhood_East	location__Neighborhood_North	location__Neighborhood_West	r
0	1.0	0.0	0.0	1
1	1.0	0.0	0.0	2
2	1.0	0.0	0.0	1
3	1.0	0.0	0.0	1
4	1.0	0.0	0.0	2
...	...	...	...	...
123	1.0	0.0	0.0	1
124	1.0	0.0	0.0	2
125	0.0	1.0	0.0	2
126	0.0	0.0	1.0	2
127	0.0	1.0	0.0	2

- **Step 4**: Report the training error and testing error in terms of RMSE.

After making predictions in the fitted model:

```
train_pred_y = linear_reg.predict(train_X)

test_pred_y = linear_reg.predict(test_X)
```

We can import mse from sklearn.metrics and obtain rmse_train and rmse_test.

Rmse_train = 14067.643277743977

Rmse_test = 16099.659294811521

The higher rmse on the testing data compared to the training data may suggest that the model is overfitting to the training data and might not perform as well on new data.

NUMBER 2

- **Step 1**: Use numpy library to conduct the training of linear regression model. We use matrix operations in numpy to write the codes of learning the parameters with gradient descent methods. (Notice: Do not use the linear regression packages of Sklearn).
- **Step 2**: Randomly split the data into two parts, one contains 80% of the samples and the other contains 20% of the samples. Use the first part as training data and train a linear

regression model and make prediction on the second part. Report the training error and testing error in terms of RMSE. Plot the loss curves in the training process.

The cost function formula is given by:

$$(\frac{1}{n})\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

Where the derivative of the cost function is:

$$\frac{dJ}{dw} = (\frac{-2}{n})\sum_{i=1}^{n} x_i * (y_i - (wx_i + b))$$

$$\frac{dJ}{db} = (\frac{-2}{n})\sum_{i=1}^{n}(y_i - (wx_i + b))$$

Hence, the gradient descent calculation will be repeated until convergence:

w = w - (learning rate * (dJ/dw))

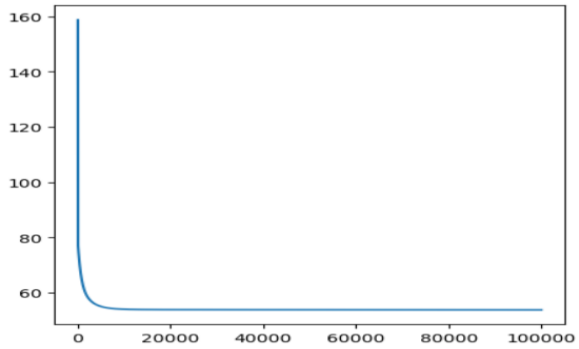
w = w - (learning rate * (dJ/db))

From the gradient descent method, we can get this output:

```
Epoch 0: 158.797897
Epoch 5000: 54.980427
Epoch 10000: 53.988021
Epoch 15000: 53.859765
Epoch 20000: 53.833280
Epoch 25000: 53.823230
Epoch 30000: 53.816667
Epoch 35000: 53.811032
Epoch 40000: 53.805721
Epoch 45000: 53.800579
Epoch 50000: 53.795561
Epoch 55000: 53.790655
Epoch 60000: 53.785853
Epoch 65000: 53.781152
Epoch 70000: 53.776550
Epoch 75000: 53.772045
Epoch 80000: 53.767634
Epoch 85000: 53.763314
Epoch 90000: 53.759085
Epoch 95000: 53.754944
```

The learning rate is 0.05 while the number of iterations is 100,000. Epoch refers to one complete pass through the entire training dataset during the training process. The output shows the progression of the gradient descent over multiple epochs. The loss is decreasing over time, which is a positive sign since the goal of gradient descent is to minimize the loss.

The loss curve is given by the graph below:



Which is converging as the number of epochs increases. The optimization process is approaching a minimum of the loss function.

The calculation of errors in RMSE are the following:

Testing error = 53.72013399961999

Training error = 53.750889098110086

The slightly lower testing error might be a positive indicator of good generalization. However, it is crucial to consider other evaluation metrics and be cautious for potential overfitting and underfitting issues.

- **Step 3**: Repeat the splitting, training, and testing 10 times with different parameters such as step size, iterations, etc. Use a loop and print the RMSE in each trial. Analyze the influence of different parameters on RMSE.

Learning Rate	Iteration numbers	RMSE train	RMSE test
0.05	1,000	63.31242155982301	59.194811217175825
0.05	20,000	53.83329139723261	53.66755620149614
0.0001	1,000	148.01971542832746	138.5458051787916
0.0001	10,000	80.16077819264989	73.42127124356294
0.0001	20,000	76.97172874005221	71.94488431645017
0.000001	1,000	172.09338463358046	162.66090667655982
0.000001	10,000	169.65595523145714	160.21799908959775
0.000001	20,000	167.01135208049402	157.56473432840252
0.5	1,000	53.987492102856834	53.54394715969
0.5	50,000	53.594912112261774	53.765916863237784

Analysis:

Different parameters of learning rate and number of iterations influenced the RMSE test and train, which corresponds to the overall performance of the model.

1. Effect on learning rate

- A higher learning rate (e.g., 0.5) tends to converge faster, however, this can lead to poor generalization performance of new data. In some cases (0.5, 50,000), the RMSE of training continues to decrease, but the RMSE of testing experienced a slight increase, resulting in a slightly higher RMSE than that of the training set. This indicates that overfitting may occur if the learning rate is too high.
- A smaller learning rate (e.g., 0.000001) results in a larger training and testing error, indicating its precision on obtaining minimum loss function. However, this may take a lot of time to converge for a too small learning rate.

2. Effect on number of iterations

- Increasing the number of iterations generally improves model performance, resulting in lower RMSE train and error.
- If the dataset is large, it will take longer time to train the number.