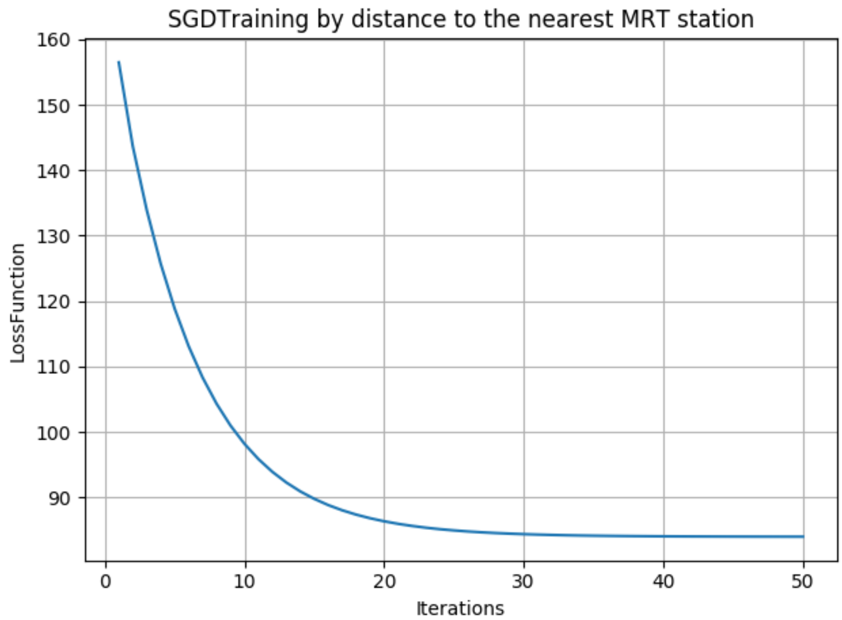
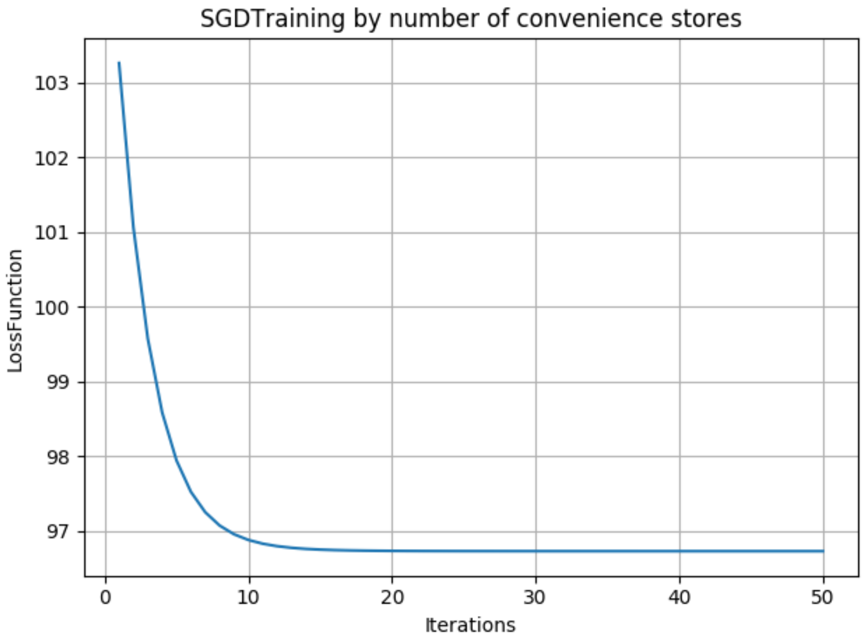
Question 1: The θ parameters (θ_0, θ_1) from step 3 when you are using house age feature. (2 marks)

$$\theta_0 = 42.54078538,\ \theta_1 = -10.31939902$$

Question 2: A plot, which visualises the change in cost function $J(\theta)$ at each iteration. (1 mark)





Question 3: RMSE for your training set when you use house age feature. (0.5 mark)

$$RMSE_{\text{training set/house age feature}} = 12.04551030591235$$

Question 4: RMSE for test set, when you use house age feature. (0.5 mark)

$$RMSE_{\text{test set/house age feature}} = 16.58731450340051$$

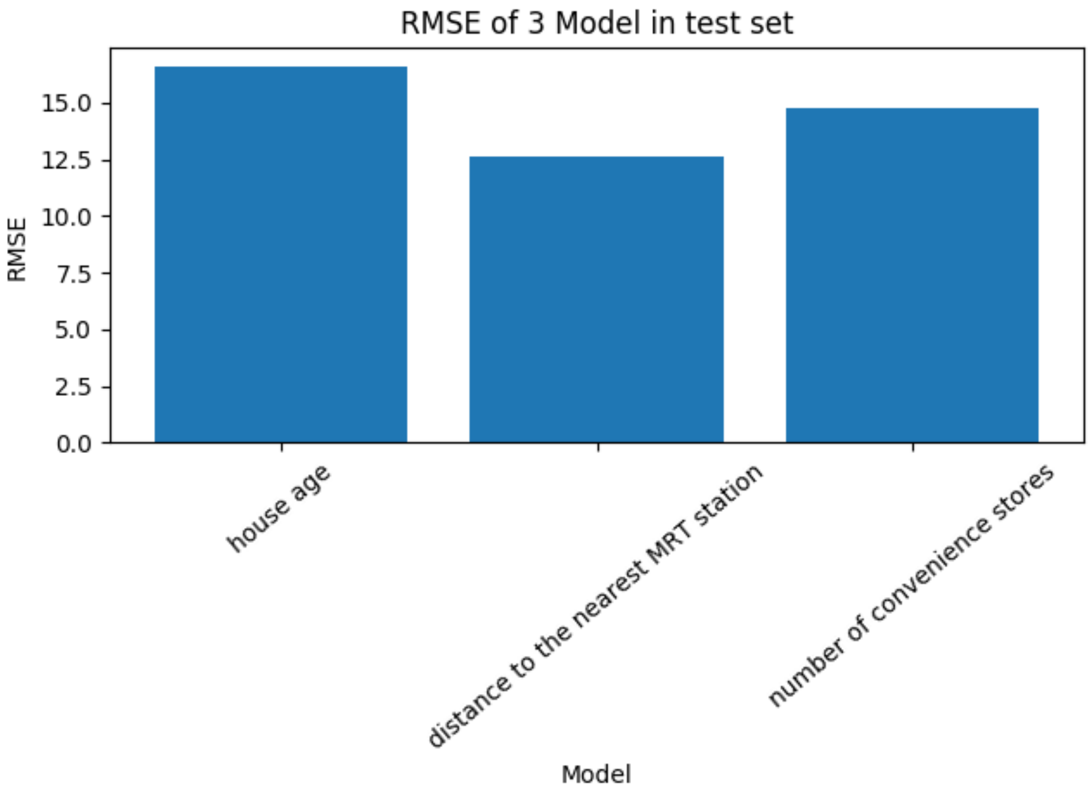
Question 5: RMSE for test set, when you use distance to the station feature. (0.25 mark)

$$RMSE_{\text{test set/distance to the station feature}} = 12.652088009723935$$

Question 6: RMSE for test set, when you use number of stores feature. (0.25 mark)

$$RMSE_{\text{test set/number of stores feature}} = 14.731993508206783$$

Question 7: Compare the performance of your three models and rank them accordingly. (0.5 mark)



The 1st is the model training by distance to the station feature.

The 2nd is the model training by number of stores feature.

The 3rd is the model training by house age feature.

The code

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


def normalize(data):
    return (data - data.min()) / (data.max() - data.min())


def costFunction(x, y, theta):
    return np.dot((np.dot(x, theta) - y).T, (np.dot(x, theta) - y))


def targetFunction(x, theta):
    return theta[0] + x * theta[1]


def SGDTraining(X, Y, parameters, draw = 0):
    iteration_times = 0
    loss_list = []
    theta = np.array(parameters['theta'])
    alpha = parameters['alpha']
    iterations_max = parameters['iterations_max']
    while iteration_times < iterations_max:
        for i in range(X.shape[0]):
            grad = alpha * (Y[i, :] - X[i, :].dot(theta)) * X[i, :]
            theta = theta + np.mat(grad).T
        loss = (np.square(Y - X.dot(theta))).sum() / Y.shape[0]
        loss_list.append(loss)
        iteration_times += 1
    if draw == 1:
        plt.title("SGDTraining by " + parameters['name'])
        plt.grid()
        plt.plot(range(1, iterations_max + 1), loss_list)
        plt.xlabel("Iterations")
        plt.ylabel("LossFunction")
        plt.show()
        SGDPlot(X, Y, theta, parameters['name'])
    return theta


def RMSE(RMSE_Data, name, theta):
    X = RMSE_Data[['One', name]].values
```

```
    Y = RMSE_Data[['house price of unit area']].values
    return np.sqrt((np.square(Y - X.dot(theta))).sum() / Y.shape[0])


def SGDPlot(X, Y, theta, name):
    plt.title("SGDTraining by " + name)
    plt.xlabel(name)
    plt.ylabel('house price of unit area')
    theta = theta.tolist()
    for i in range(300):
        plt.scatter(X[i][1], Y[i], s = 5, c = 'c')
    plt.plot((0, 1), (theta[0][0], theta[0][0] + theta[1][0]))
    plt.show()


def splitTrain(training_Data, parameters):
    X = training_Data[['One', parameters['name']]].values
    Y = training_Data[['house price of unit area']].values
    theta = SGDTraining(X, Y, parameters, draw = 1)
    return theta


def preProcessData(data):
    data_Norm = normalize(data.iloc[:, 1:4])
    data_Norm.insert(0, 'One', 1)
    data_Norm.insert(4, 'house price of unit area', data.iloc[:, 4:])
    return data_Norm

def main():
    data = pd.read_csv("house_prices.csv")
    data_Norm = preProcessData(data)
    training_Data = data_Norm[0:300]
    test_Data = data_Norm[300:]


    training_Parameters = [{"name": "house age",
                            "theta": [[-1], [-0.5]],
                            "alpha": 0.01,
                            "iterations_max" : 50},
                           {"name": "distance to the nearest MRT station",
                            "theta": [[-1], [-0.5]],
                            "alpha": 0.01,
                            "iterations_max" : 50},
                           {"name": "number of convenience stores",
```

```
                                 "theta": [[-1], [-0.5]],
                                 "alpha": 0.01,
                                 "iterations_max" : 50}]


    theta = {}
    RMSE_data = {}
    for data in training_Parameters:
        theta[data['name']] = splitTrain(training_Data, data)
    print(theta)
    for name in theta.keys():
        RMSE_data[name] = RMSE(test_Data, name, theta[name])
    print(RMSE_data)
    plt.bar(*zip(*RMSE_data.items()))
    plt.xlabel("Model")
    plt.ylabel('RMSE')
    plt.title("RMSE of 3 Model in test set")
    plt.xticks(rotation = 40)
    plt.show()



if __name__ == '__main__':
    main()
```