Assignment 2

Tanmoy Das [redacted]

www.tanmoyie.com
https://www.kaggle.com/tanmoyie/kernels
https://github.com/tanmoyie

Double-click (or enter) to edit

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Question 1

Question 1(a)

Explain briefly what k-fold cross validation is and what it is used for.

K fold cross validation

Definition:

The k fold cross validation is an approach in machine learning where the dataset is splitted into k number of folds. devided into 4 sets. In first iteration, set 1 is used for testing, & other 3 are used for training, In second time, set 2 is as training set. This method is repeated until all sets are used 4 times.

Utilization of K fold:

1. Machine Learning models can be evaluated using k fold cross validation.
2. k fold can be utilized also for setting or tuning the best parameters for the machine learning models to perfo

---

Question 1(b)

Write a script that does a k-fold cross-validation without using the cross_val_score function or another cross-valida the results with the sklearn function.

```
# k fold cross validation
# load libraries and import the dataset
import numpy as np
#import matplotlib.pyplot as plt
## iris_data = np.loadtxt('/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments csci-
iris_data = np.loadtxt('iris.data',delimiter=',')

# /content/drive/My Drive/Colab Notebooks/iris.data
```

```
# /content/drive/My Drive/Colab Notebooks/iris.data
iris_data[1:5,:] # just to check the first few rows of the data
```

```
array([[4.9, 3. , 1.4, 0.2, 0. ],
       [4.7, 3.2, 1.3, 0.2, 0. ],
       [4.6, 3.1, 1.5, 0.2, 0. ],
       [5. , 3.6, 1.4, 0.2, 0. ]])
```

```
# Developing the data model
x_train = iris_data[1:-1:2,0:4]
y_train = np.int32(iris_data[1:-1:2,4])
x_test = iris_data[0:-1:2,0:4]
y_test = np.int32(iris_data[0:-1:2,4])

iris_feature = iris_data[:,0:4] #  would be useful for spliting
#print(iris_feature)
iris_target = iris_data[:, 4]
#print(x_train)


# Cross validation from sklearn
# SVC classifier
from sklearn import svm
from sklearn.model_selection import cross_val_score
clf_svc = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf_svc, iris_data, iris_target, cv=15)
print('The scores of cross_val_score from sklearn: \n', scores)
#iris.target
```

```
The scores of cross_val_score from sklearn:
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
# Customized k fold cross validation function
# k_fold = 15
from sklearn import svm
model = svm.SVC(kernel='linear')
k_fold = 15
k = np.int32(np.linspace(0,140,15))
# will build a for-loop here
score_customized = []
for i in range(len(k)):
 # print(type(i))
  x_test = iris_feature[k[i]:k[i]+10, :]
  idx_to_del = list(np.int32(np.linspace((i-1)*10, i*10-1, 10)))
  #1 0:9 (i-1)*10, i*10-1    2 10:19   3 20:29
  x_train = np.delete(iris_feature, idx_to_del, axis=0) # axis = 0 for row-wise
 # print(y_train) # Instances should be splitted into 140-10 records
  y_test = iris_target[k[i]:k[i]+10]
  y_train = np.delete(iris_target, idx_to_del, axis=0)
  model.fit(x_train,y_train)
  y_predicted=model.predict(x_test)
  score = np.mean(y_test == y_predicted)
  #print(score)
```

```
    #print(score)
    score_customized.append(score)
    #print('score_customized: ', score_customized)
print('The scores obtained from the customized Cross Validation function \n', score_customize
print('--------------')
```

```
The scores obtained from the customized Cross Validation function
 [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.8, 0.9, 1.0, 1.0, 1.0, 1.0, 0.8, 1.0]
--------------
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: FutureWarning: in the f
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: FutureWarning: in the f
  from ipykernel import kernelapp as app
```

Compare the result with sklearn cross_val_score

From the SKLEARN cross_val_score, we have obtained the accuracy of 100% for each cases in validation consideri

However, my customized cross validation program is a bit under-performing, eventually obtaining the accuracy of 8

Question 1(c)

Compare the cross-validated results of the SVM and RF and comment on which method is better.

```
# Developing the data model
x_train = iris_data[1:-1:2,0:4]
y_train = np.int32(iris_data[1:-1:2,4])
x_test = iris_data[0:-1:2,0:4]
y_test = np.int32(iris_data[0:-1:2,4])

# Machine learning models developed in sklearn (we are considering svm & RF)
# SVM
from sklearn import svm
model = svm.SVC(kernel='linear')
model.fit(x_train,y_train)
y_predicted=model.predict(x_test)
print('Percentage correct (accuracy) of SVM : ', np.mean(y_test == y_predicted))
# print('Mean?? ', np.mean(y_test == y_predicted)*75)

# RF
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=10)
model.fit(x_train,y_train)
y_predicted=model.predict(x_test)
print('Percentage correct (accuracy) of RFC : ', np.mean(y_test == y_predicted))

# ?? need to integrate cross validation developed earlier & obtained from sklearn
```

```
Percentage correct (accuracy) of SVM :  0.9733333333333334
```

Comparison of SVM vs RF model:

SVM outperforms RF considering the accuracy, i.e. SVM provides us an accuracy of 97.33% whereas RFC is giving t
Nevertheless, we must acknowledge that without knowing the underlying assumption behind the performance mea
should not generalize the performance measure for all real-life problems in Data Science.

Question 2

Please download the wine.zip file and extract it to the directory for this assignment. Read through the wine_names.
the problem and the wine data contained in the wine.train dataset. Train one of the models SVM, MLP, or RF to deve
classifying the wine data in the hold-out test data set of 58 records in the wine.test file given the training data. In ot
of 58 classifications (as a separate *.csv file) for the hold-out test set in the same order as received. We will use yo
model performs.

```
# Load libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Import the dataset
##wine_train = np.loadtxt('/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments csci-
##wine_test = np.loadtxt('/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments csci-6

wine_train = np.loadtxt('wine.train',delimiter=',')
wine_test = np.loadtxt('wine.test',delimiter=',')


# Develop the data model
x_train_original = wine_train[:,1:]
y_train_original = wine_train[:,0]
print('x_train_original[1:4,:] \n', x_train_original[1:4,:]) # check data type of the element
print('y_train_original \n', y_train_original)
# Hold out test set: x_test includes all features
x_test_original = wine_test[:,1:]
# we dont know y_test. We will determine y_predicted
#print(x_train_original.shape, y_train_original.shape, x_test_original.shape)
# Devite the training set into two set to check the training accuracy of the model
# Random spliting of the training dataset
x_train, x_test, y_train, y_test = train_test_split(x_train_original, y_train_original, test_
```

```
x_train_original[1:4,:]
 [[ 13.2    1.78   2.14  11.2  100.     2.65   2.76   0.26   1.28   4.38
    1.05   3.4 ]
 [ 14.37   1.95   2.5   16.8  113.     3.85   3.49   0.24   2.18   7.8
    0.86   3.45]
 [ 14.2    1.76   2.45  15.2  112.     3.27   3.39   0.34   1.97   6.75
    1.05   2.85]]
y_train_original
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.]
```

```
# Implement SVM, MLP, RF
# Checking default parameters first
#SVM
from sklearn import svm
model = svm.SVC(kernel='linear', C = 0.001)
model.fit(x_train,y_train)
y_predicted_svm=model.predict(x_test)
print('Percentage correct (accuracy) of SVM : ', np.mean(y_test == y_predicted_svm))

# MLP
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=1)
clf.fit(x_train, y_train)
y_predicted = clf.predict(x_test)
print('Percentage correct (accuracy) of MLP : ', np.mean(y_test == y_predicted))

# RF
#  the number of trees (ntree) and the number of features in each split (mtry).
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=10)
model.fit(x_train,y_train)
y_predicted=model.predict(x_test)
print('Percentage correct (accuracy) of RFC : ', np.mean(y_test == y_predicted))
```

```
Percentage correct (accuracy) of SVM :  0.7
Percentage correct (accuracy) of MLP :  0.8666666666666667
Percentage correct (accuracy) of RFC :  0.9666666666666667
```

Based on the default parameters, **Random Forest seems the best performer**. Anyway, lets perform some analysis b
input parameters.

```
"""
There are two parameters that need to be set when applying the SVM classifier with RBF kernel
the optimum parameters of cost (C) and the kernel width parameter (γ).
The C parameter decides the size of misclassification allowed for non-separable training data
which makes the adjustment of the rigidity of training data possible .
The kernel width parameter (γ) affects the smoothing of the shape of the class-dividing hyper
```

```
Larger values of C may lead to an over-fitting model,
whereas increasing the γ value will affect the shape of the class-dividing hyperplane,
which may affect the classification accuracy results
"""
# Tune the models Hyper-parameter

# Implement SVM, MLP, RF
# Checking different values of the parameters
#SVM
from sklearn import svm
c = np.array([.000001, .001 , 1 , 10 , 100 ])
accuracy_svm = []
for id in range (5):
    model = svm.SVC(kernel='linear', C = c[id])
    model.fit(x_train,y_train)
    y_predicted_svm =model.predict(x_test)
    accuracy_svm.append(np.mean(y_test == y_predicted_svm))
#print('The Learning Rate, c = .000001, .001 , 1 , 10 , 100')
print('Accuracy of SVM for different learning rates: \n', accuracy_svm)

# MLP
from sklearn.neural_network import MLPClassifier
hidden_layer = np.array([1, 5 , 10 , 50 , 100 ])
accuracy_mlp = []
for id in range (5):
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(hidden_layer[id], 2),
    clf.fit(x_train, y_train)
    y_predicted_mlp = clf.predict(x_test)
    accuracy_mlp.append(np.mean(y_test == y_predicted_mlp))
print('Accuracy of MLP for different number of hidden layers: \n', accuracy_mlp)

# RF
#  the number of trees (ntree) and the number of features in each split (mtry).
from sklearn.ensemble import RandomForestClassifier
no_of_estimator = np.array([1, 5 , 10 , 20, 40 ])
accuracy_rf = []
for id in range (5):
    model = RandomForestClassifier(n_estimators=no_of_estimator[id])
    model.fit(x_train,y_train)
    y_predicted_rf = model.predict(x_test)
    accuracy_rf.append(np.mean(y_test == y_predicted_rf))
print('Accuracy of RFC for different number of estimators: \n', accuracy_rf)
```

```
Accuracy of SVM for different learning rates:
 [0.36666666666666664, 0.7, 0.9333333333333333, 0.9333333333333333, 0.9333333333333333]
Accuracy of MLP for different number of hidden layers:
 [0.3, 0.8666666666666667, 0.3, 0.3, 0.3]
Accuracy of RFC for different number of estimators:
 [0.9666666666666667, 0.9666666666666667, 1.0, 1.0, 1.0]
```

Considering various combinations of input parameters of each machine learning algorithms, it is reasonable to con

performing much better than the other algorithms.

```
#
# Confusion table considering the last hyper-parameters of the previous section
from sklearn.metrics import classification_report
print('Classification Report of SVM')
print(classification_report(y_test, y_predicted_svm))
print('------------------')

print('Classification Report of MLP')
print(classification_report(y_test, y_predicted_mlp))
print('------------------')

print('Classification Report of RFC')
print(classification_report(y_test, y_predicted_rf))
print('------------------')
```

Classification Report of SVM

Describe briefly your methodology for determining the best model

2.0 1.00 1.00 1.00 9

Implementing default parameters, Random Forest has performed better. Even after some variations in the input par Random Forest is still out-performing other algorithms (in our case, MLP & SVM). Therefore, it is resonable that Ran learning model for the problem at hand: classifing the wine dataset.

macro avg 0.97 0.97 0.97 58

Submit your final prediction program as well as the .csv file with the labels.

```
# final prediction program: choosen from previous experimentation
selected_model = RandomForestClassifier(n_estimators=10)
model.fit(x_train,y_train)
# y_predicted is the class label
wine_test1 = wine_test[:,1:]
y_predicted_rfc = np.int32(model.predict(wine_test1))
#final_prediction = np.reshape(y_predicted_rfc, (58, 1))
final_prediction_list = list(y_predicted_rfc)
print(final_prediction_list)
np.savetxt("final_prediction.csv", final_prediction_list,delimiter=',')
#print(final_prediction)
```

```
[3, 2, 2, 1, 1, 2, 2, 2, 3, 3, 1, 1, 3, 3, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 1,
```

3.0 1.00 1.00 1.00 12

Question 3

Question 3.1

Implement a multi-layer perceptron (MLP) by modifying the MLP program from the class to solve the XOR problem handwritten numbers of MNIST into their corresponding ASCII representation. Plot a training curve and interpret yo

```
# Implement MLP
import numpy as np
import matplotlib.pyplot as plt
from keras import models, layers, optimizers, datasets, utils

(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()

x_train = x_train.reshape(60000, 784)/255
x_test = x_test.reshape(10000, 784)/255
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)

inputs = layers.Input(shape=(784,))
```

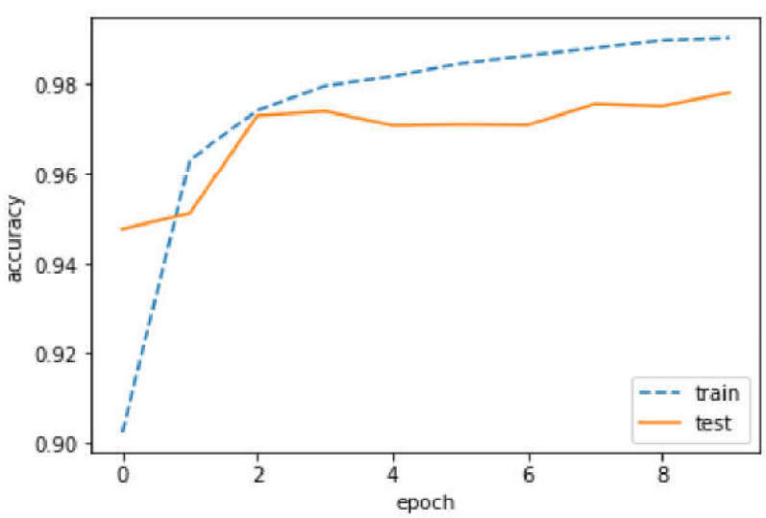
```
inputs = layers.Input(shape=(784,))
x = layers.Dense(128, activation='relu')(inputs)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dense(128, activation='relu')(x)
outputs= layers.Dense(10, activation='softmax')(x)

model = models.Model(inputs=inputs, outputs=outputs)

model.compile(loss='categorical_crossentropy',
              optimizer='Nadam', metrics=['accuracy'])

history=model.fit(x_train, y_train,
            batch_size=128,
            epochs=10,
            validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0],'Test accuracy:', score[1])
```

```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793:

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

Train on 60000 samples, validate on 10000 samples
Epoch 1/10
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

60000/60000 [==============================] - 7s 109us/step - loss: 0.3158 - acc: 0.902
Epoch 2/10
60000/60000 [==============================] - 5s 85us/step - loss: 0.1204 - acc: 0.9632
Epoch 3/10
60000/60000 [==============================] - 5s 84us/step - loss: 0.0844 - acc: 0.9743
Epoch 4/10
60000/60000 [==============================] - 5s 84us/step - loss: 0.0670 - acc: 0.9796
Epoch 5/10
60000/60000 [==============================] - 5s 86us/step - loss: 0.0571 - acc: 0.9818
Epoch 6/10
60000/60000 [==============================] - 5s 87us/step - loss: 0.0493 - acc: 0.9846
Epoch 7/10
60000/60000 [==============================] - 5s 84us/step - loss: 0.0444 - acc: 0.9864
Epoch 8/10
60000/60000 [==============================] - 5s 86us/step - loss: 0.0389 - acc: 0.9881
Epoch 9/10
60000/60000 [==============================] - 5s 85us/step - loss: 0.0337 - acc: 0.9898
Epoch 10/10
60000/60000 [==============================] - 5s 89us/step - loss: 0.0319 - acc: 0.9903
10000/10000 [==============================] - 0s 45us/step
Test loss: 0.08151330696701188 Test accuracy: 0.9782
```

```
c = '2'
print("The ASCII value of '" + c + "' is",ord(c))
```

```
The ASCII value of '2' is 50
```

```
# Plot a training curve and interpret your results.
# Plotting learning curves
plt.plot(history.history['acc'],'--')
plt.plot(history.history['val_acc'])
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```



Interpret the results

Question 3.2

Investigate how much noise the MLP can tolerate in the pattern before being unable to recognize a letter. Explain yo report your results.

```
# Salt & pepper
# Skip
```

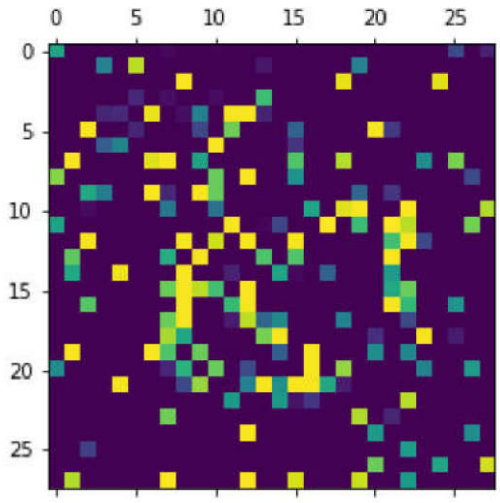
Explain your implementation of noise and report your results.

Question 3.3

Which letter is represented in file pattern.txt?

```
## input_pattern = np.loadtxt('/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments c

input_pattern = np.loadtxt('pattern.txt')

input_pattern[1:5]
np.max(input_pattern)
letter_data = input_pattern.reshape(28,28)
letter_data
plt.matshow(letter_data) # Show the input data as an image
```

```
<matplotlib.image.AxesImage at 0x7f34d72eef98>
```




```
from sklearn.neural_network import MLPClassifier
mlp_q3 = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)
mlp_q3.fit(x_train, y_train)
```

```
Iteration 1, loss = 0.68406271
Iteration 2, loss = 0.37303567
Iteration 3, loss = 0.31256934
Iteration 4, loss = 0.27838517
Iteration 5, loss = 0.25614642
Iteration 6, loss = 0.24135793
Iteration 7, loss = 0.22529110
Iteration 8, loss = 0.21521206
Iteration 9, loss = 0.20508928
Iteration 10, loss = 0.19783897
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:5
  % self.max_iter, ConvergenceWarning)
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(50,), learning_rate='constant',
              learning_rate_init=0.1, max_iter=10, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=1, shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=10, warm_start=False)
```

Question 3.4

Investigate the network performance when training on noisy patterns. Also, how does the number of hidden nodes

```
# Skip
```

A large number of hidden nodes will introduce model complexity

---

Question 4

Question 4.1

Load the House sales dataset from the houses.csv file and place them in a data-frame df. (Hint: You can use panda
pandas? Click here). Then generate and show various statistic summary using pandas.DataFrame.describe metho
Using pandas.DataFrame methods split the dataset into target value Y (price) and feature matrix X (all feature colu
sqft_living column into a feature vector name X_1.

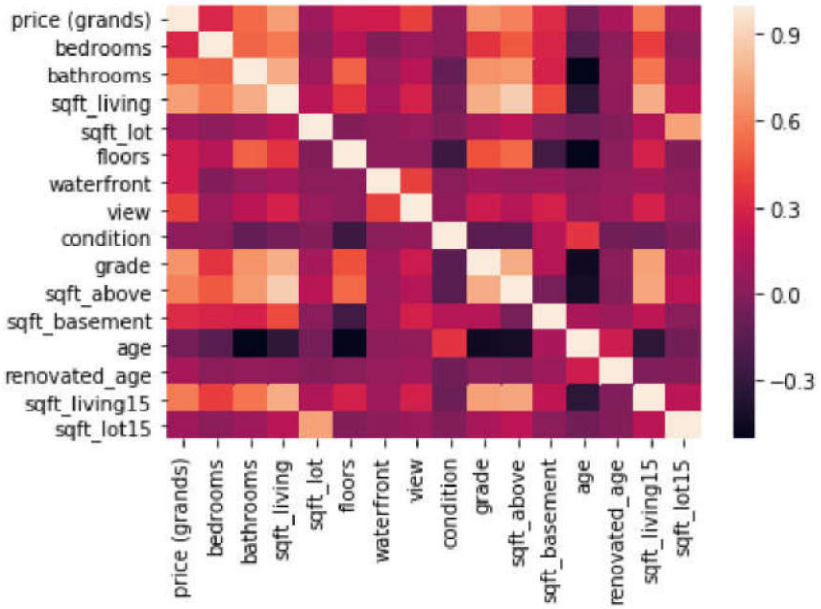
```
# Read the house data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
##df = pd.read_csv('/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments csci-6505/As
df = pd.read_csv('houses1.csv',delimiter=',')


summary_statistics = df.describe()
summary_statistics
```

	price (grands)	bedrooms	bathrooms	sqft_living	sqft_lot	floors
count	21613.000000	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000
mean	540.088142	3.370842	2.114757	2079.899736	1.510697e+04	1.494309
std	367.127196	0.930062	0.770163	918.440897	4.142051e+04	0.539989
min	75.000000	0.000000	0.000000	290.000000	5.200000e+02	1.000000
25%	321.950000	3.000000	1.750000	1427.000000	5.040000e+03	1.000000
50%	450.000000	3.000000	2.250000	1910.000000	7.618000e+03	1.500000
75%	645.000000	4.000000	2.500000	2550.000000	1.068800e+04	2.000000
max	7700.000000	33.000000	8.000000	13540.000000	1.651359e+06	3.500000

Pandas dataframe is a two-deminsional data structure in the Pandas package.

```
import seaborn as sns
corr = df.corr()
sns.heatmap(corr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f6e03080>
```




```
import pandas as pd
##df = pd.read_csv('/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments csci-6505/As

df = pd.read_csv('houses1.csv',delimiter=',')

print(df.describe(include = 'all'))

X = np.array(df[df.columns[1:16]].values, dtype = 'int')
X_1 = np.array(df[['sqft_living']].values)
y = np.array(df[['price (grands)']].values, dtype = 'int')
print(X.shape)
print(X_1.shape)
print(y.shape)
```

```
       price (grands)      bedrooms  ...  sqft_living15     sqft_lot15
count    21613.000000  21613.000000  ...   21613.000000   21613.000000
mean       540.088142      3.370842  ...    1986.552492   12768.455652
std        367.127196      0.930062  ...     685.391304   27304.179631
min         75.000000      0.000000  ...     399.000000     651.000000
25%        321.950000      3.000000  ...    1490.000000    5100.000000
50%        450.000000      3.000000  ...    1840.000000    7620.000000
75%        645.000000      4.000000  ...    2360.000000   10083.000000
max       7700.000000     33.000000  ...    6210.000000  871200.000000

[8 rows x 16 columns]
(21613, 15)
(21613, 1)
(21613, 1)
```

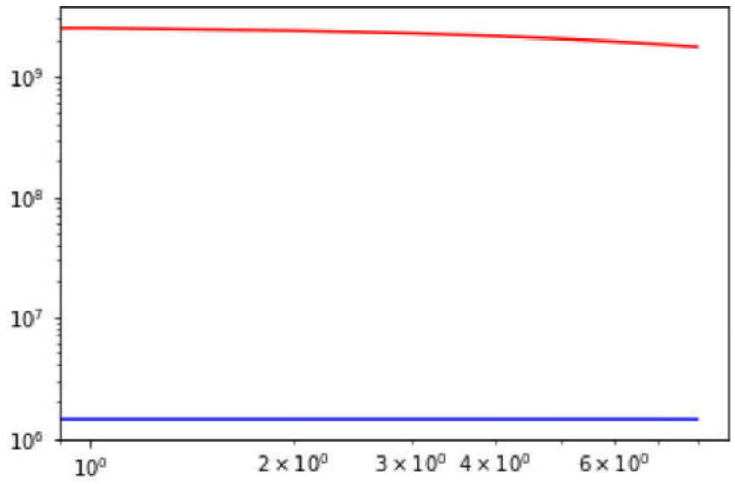
Question 4.2

Write a function named linear_regression to implement Linear Regression without using public libraries related to re
function should be predictor values (X or X_1), a target value (Y), a learning rate (lr), and the number of iterations (re
a linear model using gradient descent and output the model (params) and loss values per iteration (loss). Set the ite
show the mean squared error (MSE) for the models obtained from both X and X_1 predictors (hint: you might write
predict the values based on X or X_1 and params) and plot the learning curve (loss) for both models in one figure (h
different learning rates and show the results.

```
# Write the linear regression model

def linear_regression(X, price, lr = 0.00000000001, repetition = 10):
    count_of_columns = len(X[1, :])
    count_of_rows = len(price)

    w0 = np.array([[1]]).T
    w = np.array([[1 for i in range(count_of_columns)]]).reshape(count_of_columns,1)
    loss = np.array([])

    for no_iteration in range(repetition - 1):
        y = np.dot(X, w[:, -1].reshape(count_of_columns, 1))+ w0[-1]
        w = np.concatenate((w, w[:, -1].reshape(count_of_columns, 1) - (lr*sum(np.dot((y - pr
        w0 = np.append(w0, w0[-1] - (lr*sum((y - price)) / count_of_rows))
        loss = np.append(loss, sum((y - price)**2) / (2*count_of_rows))

    return (w0, w, loss, y)


# you may want to write another function
def predict(w0, w, test_x):
    count_of_columns = len(test_x[1, :])
    y_pred = np.dot(test_x, w[:, -1].reshape(count_of_columns, 1)) + w0[-1]
    return (y_pred)


import pylab
```

```
import pylab

# Regression for all features in X
w0_X, w_X, loss_X, y_X = linear_regression(X, y, 0.00000000001, 10)
pylab.loglog(loss_X, 'r')
print("Mean Squared Error for all features in X")
print(np.mean(loss_X))

# Regression for  X_1
w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression(X_1, y, 0.00000000001, 10)
pylab.loglog(loss_X_1, 'b')
print("Mean Squared Error for X_1 (sqft_living)")
print(np.mean(loss_X_1))
pylab.show()
#y_predicted = predict(w0_X_1, w_X_1, test_x_1)
```

```
Mean Squared Error for all features in X
2167361949.98522
Mean Squared Error for X_1 (sqft_living)
1439112.6307965359
```



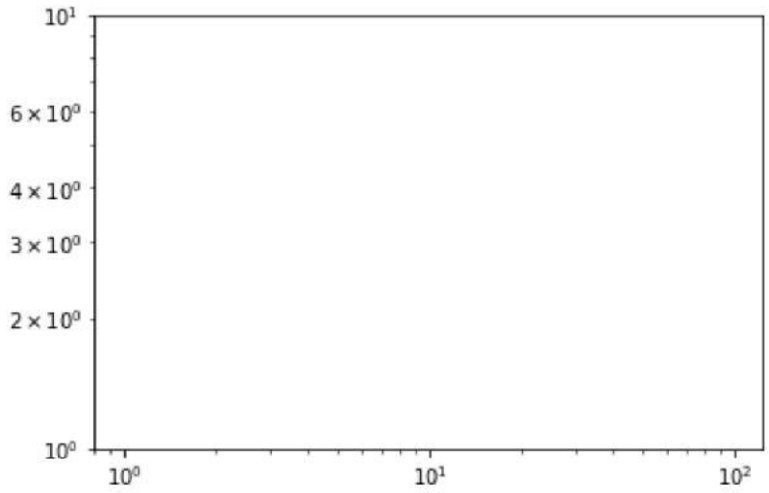

```
# show the mean squared error (MSE) for the models obtained from both X and X_1 predictors
#(hint: you might write another function named predict to predict the values based on X or X_
```

A higher value of the number of iterations gives us "runtime overflow" error, so a lesser number of iteration has beer

```
import pylab

# Regression for all features in X
w0_X, w_X, loss_X, y_X = linear_regression(X, y, 0.0000001, 100)
pylab.loglog(loss_X, 'r')
print("Mean Squared Error for all features in X")
print(np.mean(loss_X))

# Regression for  X_1
w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression(X_1, y, 0.0000001, 100)
pylab.loglog(loss_X_1, 'b')
print("Mean Squared Error for X_1 (sqft_living)")
```

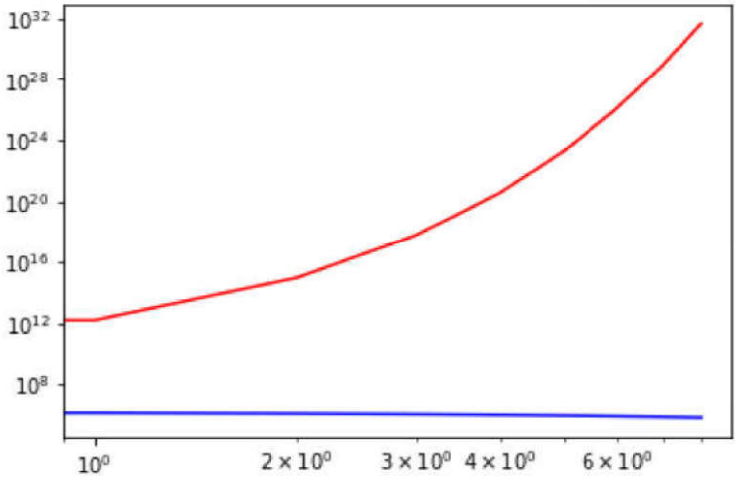
```
print(np.mean(loss_X_1))
pylab.show()
#y_predicted = predict(w0_X_1, w_X_1, test_x_1)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: RuntimeWarning: overflo

/usr/local/lib/python3.6/dist-packages/numpy/ma/core.py:6666: RuntimeWarning: overflow e
  result = np.where(m, fa, umath.power(fa, fb)).view(basetype)
Mean Squared Error for all features in X
inf
Mean Squared Error for X_1 (sqft_living)
52861.61219779472
```




```
# Try different learning rates and show the results.
# Regression for all features in X
w0_X, w_X, loss_X, y_X = linear_regression(X, y, 0.00000001, 10)
pylab.loglog(loss_X, 'r')
print("Mean Squared Error for all features in X")
print(np.mean(loss_X))

# Regression for  X_1
w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression(X_1, y, 0.00000001, 10)
pylab.loglog(loss_X_1, 'b')
print("Mean Squared Error for X_1 (sqft_living)")
print(np.mean(loss_X_1))
pylab.show()
#y_predicted = predict(w0_X_1, w_X_1, test_x_1)
```

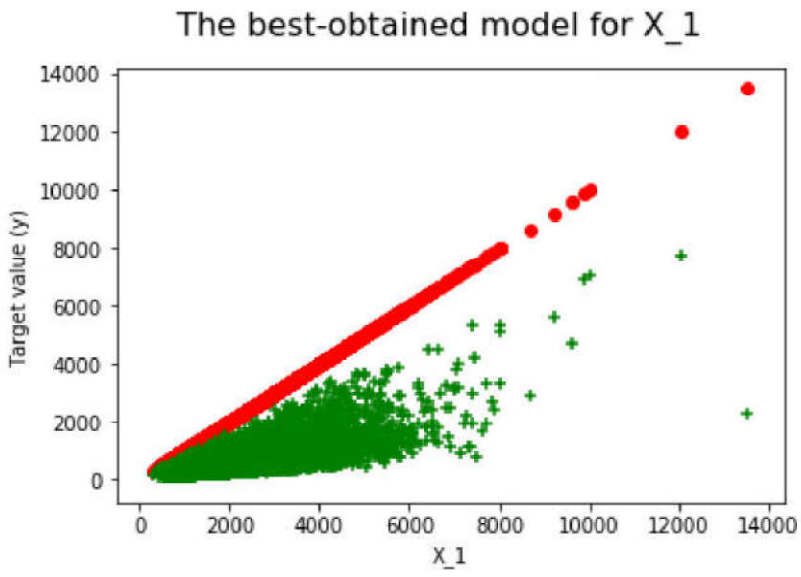
```
Mean Squared Error for all features in X
5.186345701149292e+30
Mean Squared Error for X_1 (sqft_living)
988369.8376383682
```



Question 4.3

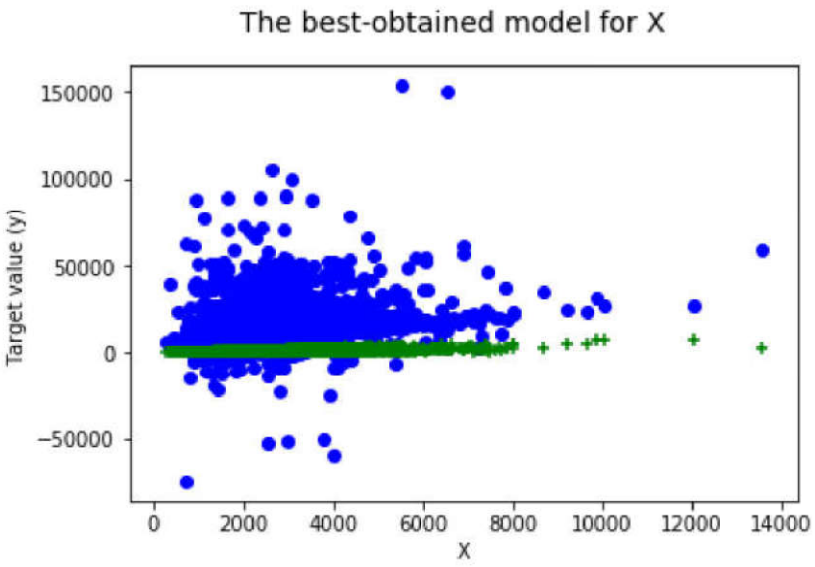
Visualize the best-obtained model for X_1 using a scatter plot to show price vs area and plot the linear model. Then model for all features (X) using a scatter plot to show the predicted vs actual target values.

```
w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression(X_1, y, 0.00000000001, 100)
plt.scatter(X_1,predict(w0_X_1, w_X_1, X_1), marker = "o", color = 'r')
plt.scatter(X_1,y, marker = "+", color = 'g')
plt.suptitle('The best-obtained linear regression model for X_1', fontsize=16)
plt.xlabel('X_1')
plt.ylabel('Target value (y)')
plt.show()
```




```
# visualize the best-obtained model for all features (X) using a scatter plot to show the pre
```

```
# visualize the best-obtained model for all features (X) using a scatter plot to show the pre
w0_X, w_X, loss_X, y_X = linear_regression(X, y, 0.00000000001, 100)
plt.scatter(X_1,predict(w0_X, w_X, X), marker = "o", color = 'b')
plt.scatter(X_1,y, marker = "+", color = 'g') # trying to understand how to use X not X_1
plt.suptitle('The best-obtained model for X', fontsize=14)
plt.xlabel('X')
plt.ylabel('Target value (y)')
plt.show()
# ??
```



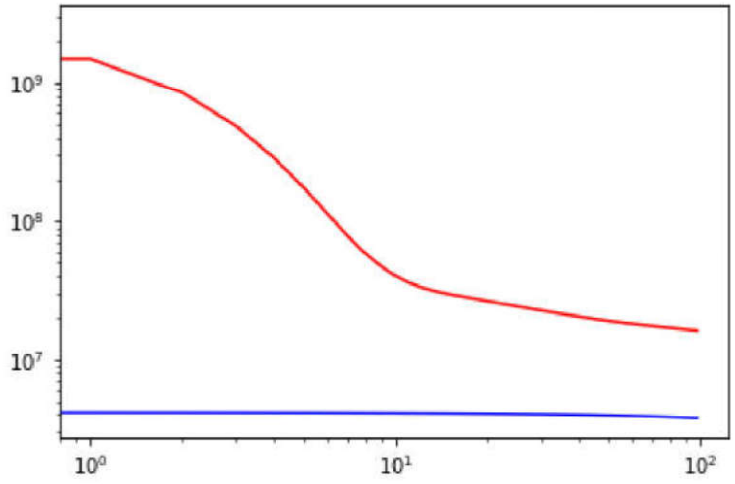
Question 4.4

Modify the linear_regression function in a way that applies Ridge regression, and name them linear_regression_Ridg
1.2 and 1.3. You can thereby use a fixed learning rate that you find appropriate, but you should try and plot different
penalty alpha.

```
def linear_regression_Ridge(X, price, lr = 0.0000000001, repetition = 10, lambdaa = 0.5):
    number_of_column = len(X[1, :])
    number_of_rows = len(price)
    w0 = np.array([[-1]]).T
    w = np.array([[-1 for i in range(number_of_column)]]).reshape(number_of_column,1)
    loss = np.array([])
    for no_iteration in range(repetition - 1):
        y = np.dot(X, w[:, -1].reshape(number_of_column, 1))+ w0[-1]
        w = np.concatenate((w, w[:, -1].reshape(number_of_column, 1) - (lr*sum(np.dot((y - pr
        w0 = np.append(w0, w0[-1] - (lr*sum((y - price)) / number_of_rows))
        loss = np.append(loss, (sum((y - price)**2) / (2*number_of_rows)) + (lambdaa * (sum(w

    return (w0, w, loss, y)


#
w0_X, w_X, loss_X, y_X = linear_regression_Ridge(X, y, 0.0000000001, 100, lambdaa = 0.5)
pylab.loglog(loss_X, 'r')
```

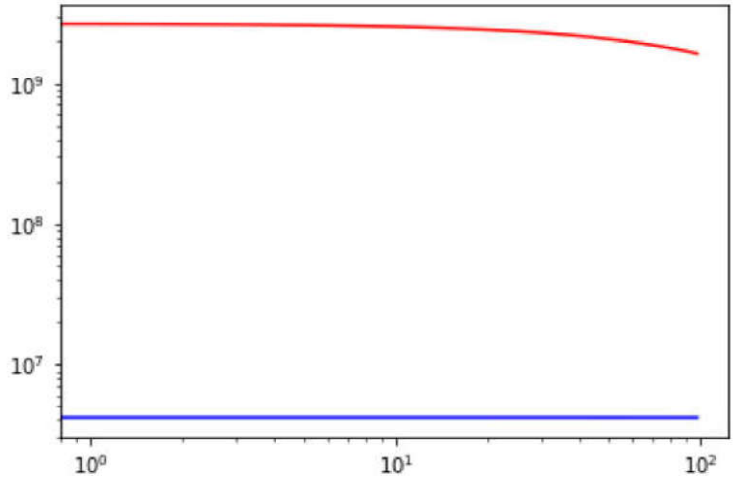
```
print("Mean Squared Error for in X: Linear Ridge")
print(np.mean(loss_X))

w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression_Ridge(X_1, y, 0.0000000001, 100, lambdaa =
pylab.loglog(loss_X_1, 'b')
print("Mean Squared Error in (sqft_living): Linear Ridge")
print(np.mean(loss_X_1))
pylab.show()
```

```
Mean Squared Error for all features in X
81605749.11976719
Mean Squared Error considering X_1 (sqft_living)
3958130.254671199
```



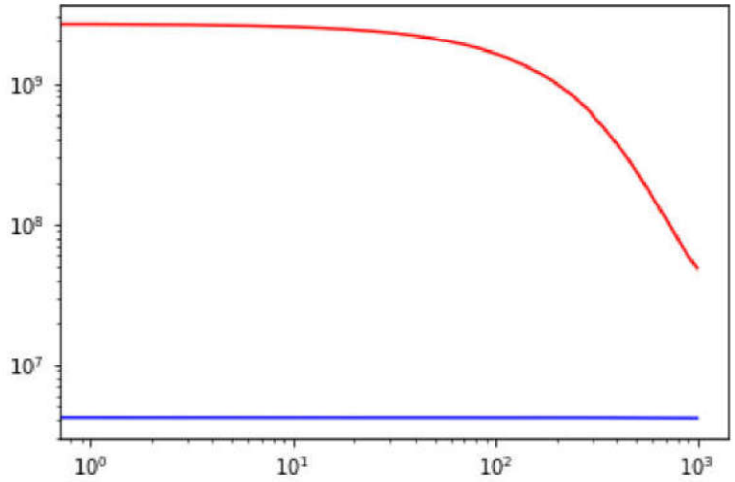

```
# We need to take a smaller learning rate... otherwise throwing run-time error
w0_X, w_X, loss_X, y_X = linear_regression_Ridge(X, y, 0.000000000001, 100, lambdaa = 0.5)
pylab.loglog(loss_X, 'r')
print("Mean Squared Error for all features - X")
print(np.mean(loss_X))

w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression_Ridge(X_1, y, 0.000000000001, 100, lambdaa
pylab.loglog(loss_X_1, 'b')
print("Mean Squared Error considering X_1")
print(np.mean(loss_X_1))
pylab.show()
```

```
Mean Squared Error for all features - X
2114576941.7446153
Mean Squared Error considering X_1
4158197.2015402094
```




```
# We need to take a smaller learning rate... otherwise throwing run-time error
w0_X, w_X, loss_X, y_X = linear_regression_Ridge(X, y, 0.000000000001, 1000, lambdaa = 0.5)
pylab.loglog(loss_X, 'r')
print("MSE in X")
print(np.mean(loss_X))

w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression_Ridge(X_1, y, 0.000000000001, 1000, lambda
pylab.loglog(loss_X_1, 'b')
print("MSE in X1")
print(np.mean(loss_X_1))
pylab.show()
```

```
Mean Squared Error for all features - X
550691244.6379845
Mean Squared Error for X_1 (sqft_living)
4139073.3723179996
```

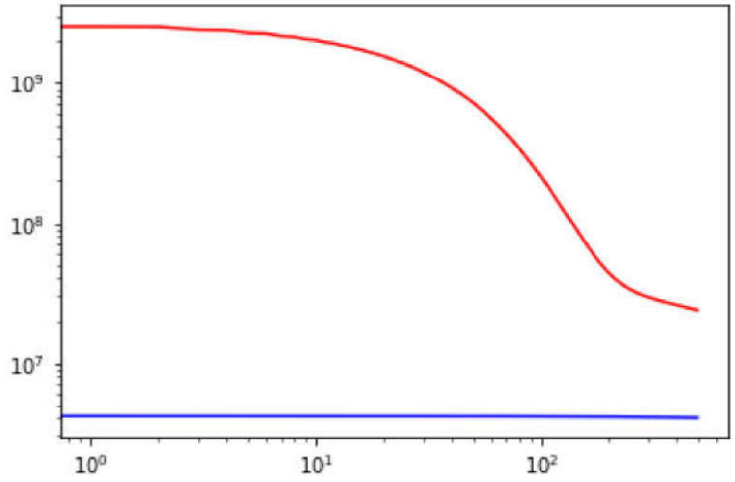


Question 4.5

Use linear_regression_Ridge and write a function named linear_regression_Ridge_momentum in which you add a m
momenta and plot the learning curves with and without momentum for a fixed learning rate.

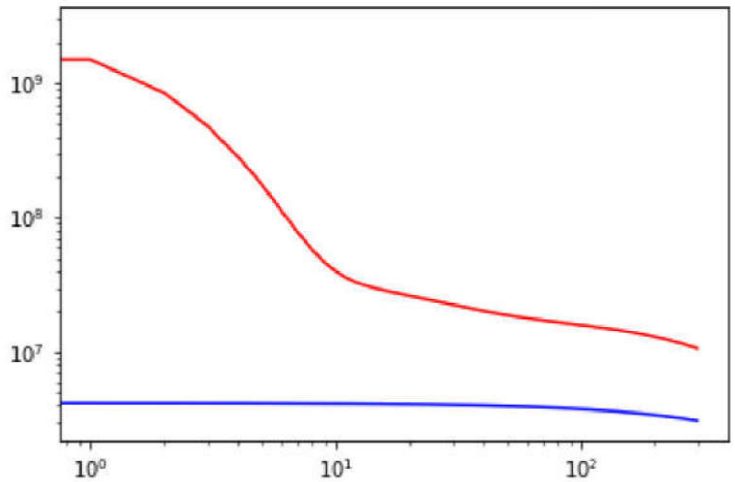
```
# momentum
def linear_regression_Ridge_momentum(X, price, lr = 0.0000000001, repetition = 10, momenta =

    num_of_columns = len(X[1, :]) # reshaping the y!
    num_of_rows = len(price)
    w0 = np.array([[-1]]).T
    w = np.array([[-1 for i in range(num_of_columns)]]).reshape(num_of_columns,1)
    loss = np.array([])
    # write a for loop
    for no_iteration in range(repetition - 1):
        two_step_back = no_iteration - 2
        y = np.dot(X, w[:, -1].reshape(num_of_columns, 1))+ w0[-1]
        w = np.concatenate((w, w[:, -1].reshape(num_of_columns, 1) -
                            ((lr*sum(np.dot((y - price).reshape(1, num_of_rows), X)) / num_of
                             (momenta*(w[:, -1] - w[:, no_iteration - 1]))).reshape(num_of_co
        w0 = np.append(w0, w0[-1] - (lr*sum((y - price)) / num_of_rows))
        loss = np.append(loss, (sum((y - price)**2) / (2*num_of_rows)) + lambdaa * (sum(w[:,

    return (w0, w, loss, y)


# Try different momenta and plot the learning curves with momentum for a fixed learning rate
# write a for loop ; 10000 is not working, why?
w0_X, w_X, loss_X, y_X = linear_regression_Ridge_momentum(X, y, 0.00000000001, 500, momenta =
pylab.loglog(loss_X, 'r')
print("Ridge Momentum: Mean Squared Error on X")
print(np.mean(loss_X))

w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression_Ridge_momentum(X_1, y, 0.00000000001, 500,
pylab.loglog(loss_X_1, 'b')
print("Ridge Momentum: Mean Squared Error on sqft_living")
print(np.mean(loss_X_1))
pylab.show()
```

```
Ridge Momentum: Mean Squared Error on X
227611002.8367987
Ridge Momentum: Mean Squared Error on sqft_living
4104778.6507310914
```




```
# Try different momenta and plot the learning curves without momentum for a fixed learning ra
# write a for loop ??
w0_X, w_X, loss_X, y_X = linear_regression_Ridge_momentum(X, y, 0.0000000001, 300, momenta =
pylab.loglog(loss_X, 'r')
print("Mean Squared Error on X")
print(np.mean(loss_X))

w0_X_1, w_X_1, loss_X_1, y_X_1 = linear_regression_Ridge_momentum(X_1, y, 0.0000000001, 300,
pylab.loglog(loss_X_1, 'b')
print("Mean Squared Error on sqft_living")
print(np.mean(loss_X_1))
pylab.show()
```

```
Mean Squared Error on X
35777507.453453206
Mean Squared Error on sqft_living
3585184.2183669554
```



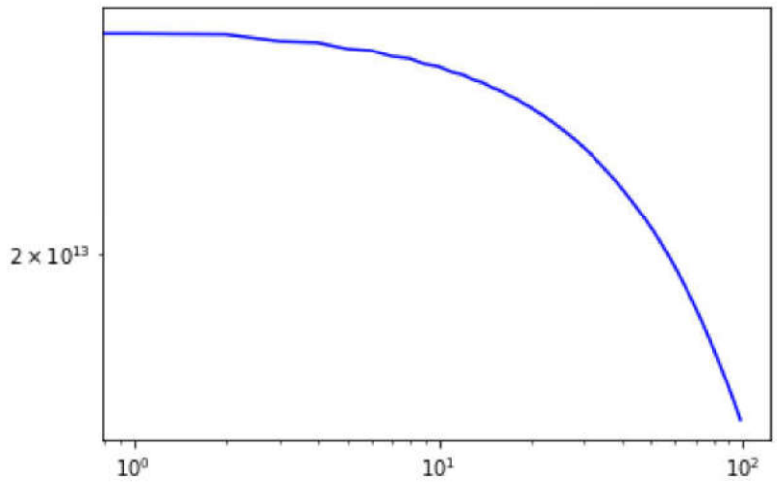
Question 4.6

Modify the linear_regression_Ridge_momentum function in a way that it fits the feature vector X_1 with a polynomi
polynomial_regression_optimized. Calculate the MSE, plot the learning curve and show the quadratic model on the

```
# polynomial
def polynomial_regression_optimized(X, price, lr = 0.0001, repetition = 10, momenta = 0.9, la
    X = np.concatenate((X, X*X), axis = 1)
    count_of_columns = len(X[1, :])
    num_of_rows = len(price)
    w0 = np.array([[-1]]).T
    w = np.array([[-1 for i in range(count_of_columns)]]).reshape(count_of_columns,1)
    loss = np.array([]) # appending purpose
    # write a for loop
    for no_iteration in range(repetition - 1):
        two_step_back = no_iteration - 2 #
        y = np.dot(X, w[:, -1].reshape(count_of_columns, 1))+ w0[-1]
        w = np.concatenate((w, w[:, -1].reshape(count_of_columns, 1) -
                            ((lr*sum(np.dot((y - price).reshape(1, num_of_rows), X)) / num_of
        w0 = np.append(w0, w0[-1] - (lr*sum((y - price)) / num_of_rows))
        loss = np.append(loss, (sum((y - price)**2) / (2*num_of_rows)) + lambdaa * (sum(w[:,

    return (w0, w, loss, y)


the MSE
1, loss_X_1, y_X_1 = polynomial_regression_optimized(X_1, y, 0.0000000000000001, 100, momenta
Squared Error regarding X_1 (sqft_living)")
n(loss_X_1))
```

```
Mean Squared Error regarding X_1 (sqft_living)
21217440233151.234
```

```
# plot the learning curve
pylab.loglog(loss_X_1, 'b')
pylab.show()
```




```
# Show the quadratic model in the scatter plot
w0_X_1, w_X_1, loss_X_1, y_X_1 = polynomial_regression_optimized(X_1, y, 0.0000000000000001,
plt.scatter(X_1,predict(w0_X_1, w_X_1, np.concatenate((X_1, X_1*X_1), axis = 1)), marker = "*
plt.scatter(X_1,y, marker = "o", color = 'b')
plt.suptitle('Polynomial Best models for X_1', fontsize=13)
plt.xlabel('X_1')
plt.ylabel('Target value (y)')
plt.show()
```