Name: Chaoran Li
Student ID: (UTD ID) 2021489307; (NET ID) cxl190012
Course Number: CS 6364.002

Homework 4 Writeup Part

Linear Regression, Boston housing dataset
Q1: Use the python library (sklearn.linear model) to train a linear regression model for the Boston housing dataset: https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155.
Split the dataset to a training set (70% samples) and a testing set (30% samples). Report the root mean squared errors (RMSE) on the training and testing sets.

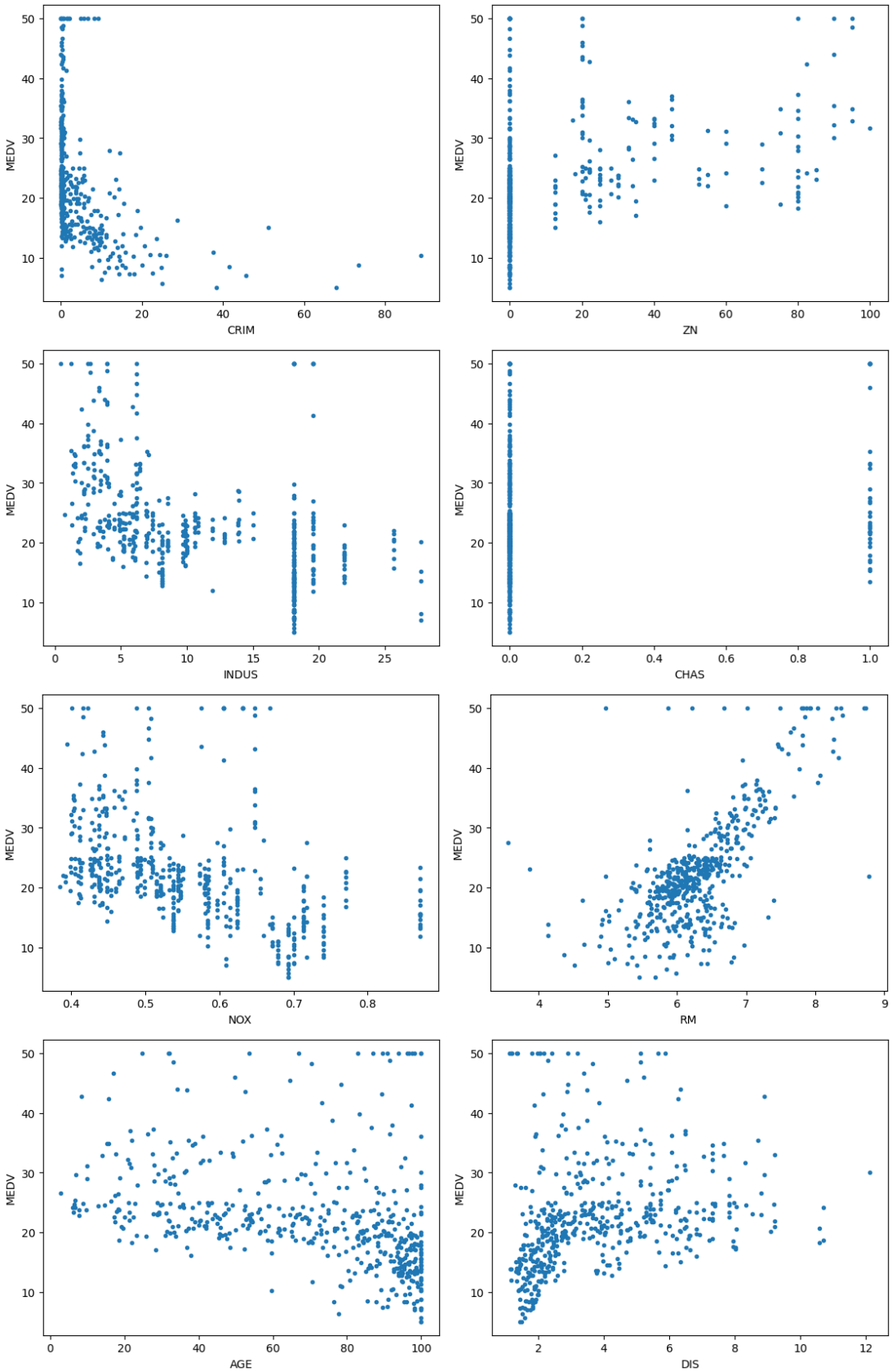
1) Data analysis
To write at first, when I call the boston dataset in sklean, it comes with the following warning.

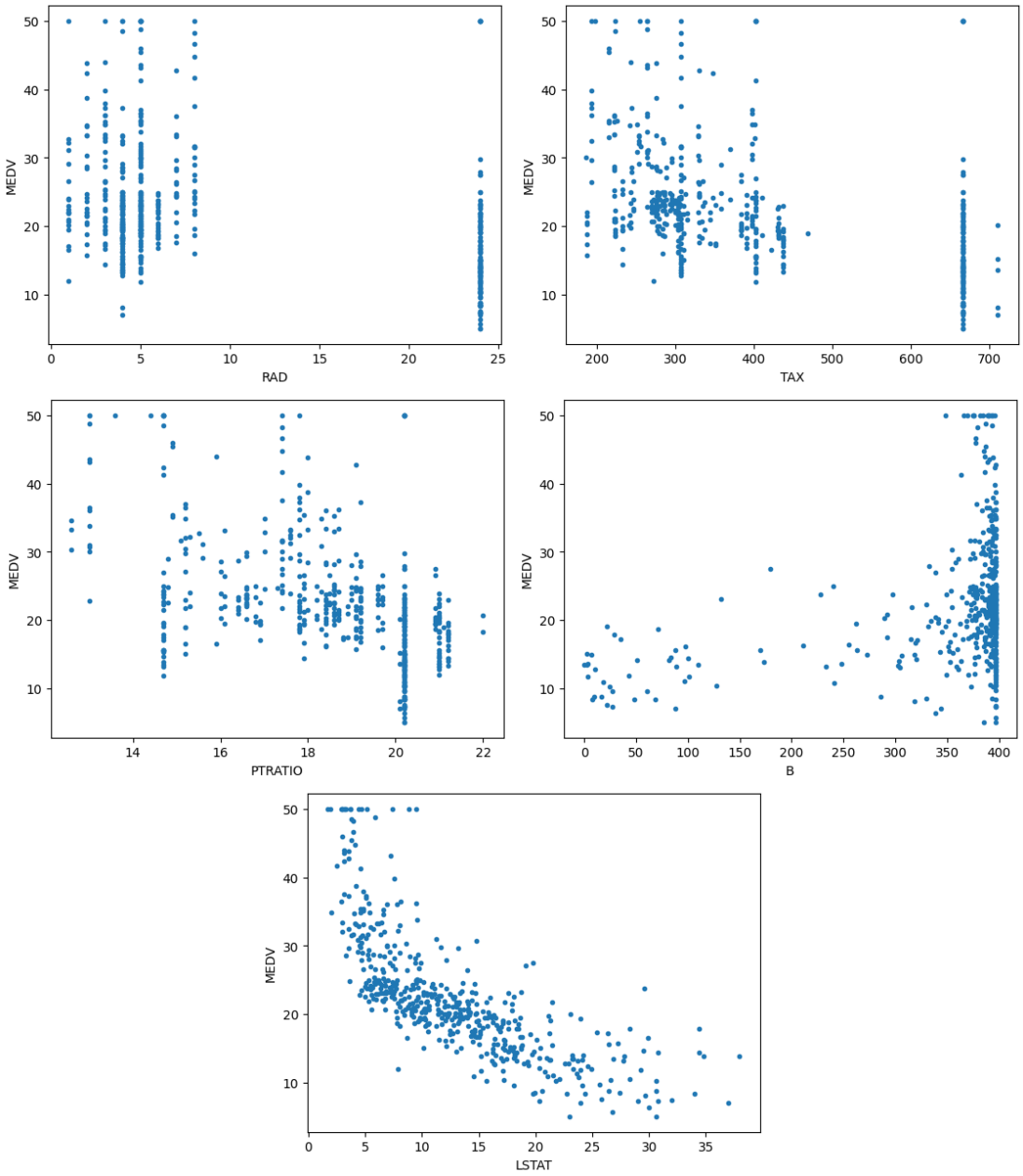
```
The Boston housing prices dataset has an ethical problem.
The scikit-learn maintainers therefore strongly discourage the use of this
dataset unless the purpose of the code is to study and educate about ethical
issues in data science and machine learning. In studying case, you can fetch
the dataset from the original source:
```

For here, I just use the code given by them to fetch the dataset from the original source.
For all features we have, below is the description of them:

```
"CRIM": "Per capita crime rate by town",
"ZN": "Proportion of residential land zoned for lots over 25,000 sq. ft",
"INDUS": "Proportion of non-retail business acres per town",
"CHAS": "Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)",
"NOX": "Nitric oxide concentration (parts per 10 million)",
"RM": "Average number of rooms per dwelling",
"AGE": "Proportion of owner-occupied units built prior to 1940",
"DIS": "Weighted distances to five Boston employment centers",
"RAD": "Index of accessibility to radial highways",
"TAX": "Full-value property tax rate per $10,000",
"PTRATIO": "Pupil-teacher ratio by town",
"B": "1000(Bk — 0.63)², where Bk is the proportion of [people of African
American descent] by town",
"LSTAT": "Percentage of lower status of the population",
"MEDV": "Median value of owner-occupied homes in $1000s"
```

The last row which is “MEDV” is the price which refers to y value. I will treat other features as X values. I checked all dataset and no NaN value found. Hence, we can directly use them. I first drew diagrams to analyze the relation between each feature and “MEDV”.


MEDV
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS



From the diagrams, we could not drop any features. Hence, we will use all the dataset to train our model.

2) Model training:
Below is some screenshot from terminal:

```
'data':
(506, 13)
[[6.3200e-03 1.8000e+01 2.3100e+00 0.0000e+00 5.3800e-01 6.5750e+00
  6.5200e+01 4.0900e+00 1.0000e+00 2.9600e+02 1.5300e+01 3.9690e+02
  4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 6.4210e+00
  7.8900e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9690e+02
  9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 7.1850e+00
  6.1100e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9283e+02
  4.0300e+00]]
...
'target':
(506,)
[24.  21.6 34.7]
...

Check if data needs imputation:
'data': False
'target': False
Training set size:354
Testing set size:152


RMSE of training set: 4.849055005805464
RMSE of testing set: 4.453237437198145

Process finished with exit code 0
```

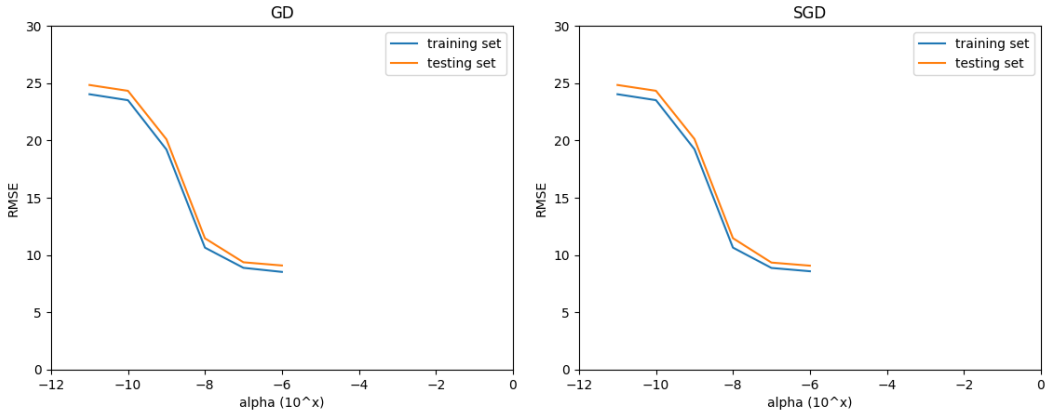
Training set size: 354/(354+152) = 0.6996 = 0.7
With the linear regression model provided in sklearn, we get the RMSE of training set with 4.849 and testing set with 4.453.

Q2: Implement the following five algorithms to train a linear regression model for the Boston housing data set https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155
Split the dataset to a training set (70% samples) and a testing set (30% samples). Report the root mean squared errors (RMSE) on the training and testing sets.
1. The gradient descent algorithm
2. The stochastic gradient descent (SGD) algorithm
3. The SGD algorithm with momentum
4. The SGD algorithm with Nesterov momentum
5. The AdaGrad algorithm

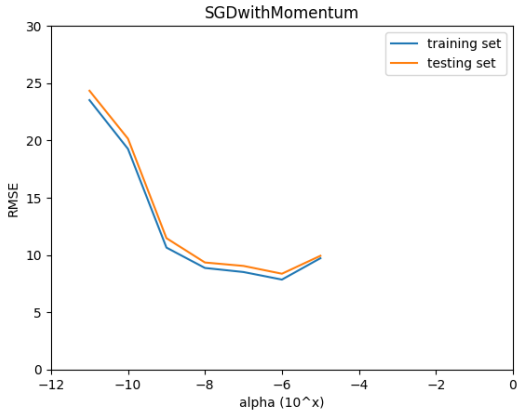
For this question, I implement my own linear regression class.

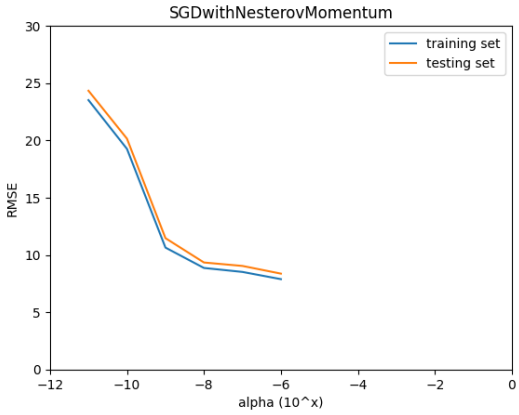
```
89
90  class MyLinearRegression:
91      def __init__(self, _a_id, _alpha=1e-2, _steps=1e3, _batch=50, _eta=0.9, _epsilon=1e-6):
92
93          if a_id not in range(5):
94              print("Unvalid algorithm id.")
95              return
```

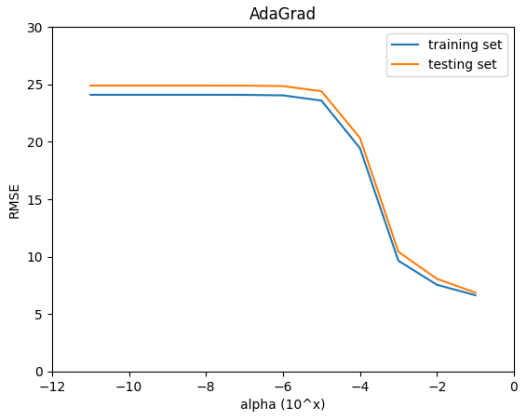
You can check it in code part. I use different a_id which is short for algorithm id to represent different algorithms. When I adjust parameters, I found AdaGrad algorithm needs a much bigger learning rate (α) which is about 0.01 while others need a small learning rate to converge. The following diagrams show this trend. To be clear, the missing points represent that the model cannot converge in that learning rate.


GD
training set
testing set
RMSE
alpha (10^x)

SGD
training set
testing set
RMSE
alpha (10^x)


SGDwithMomentum
training set
testing set
RMSE
alpha (10^x)


SGDwithNesterovMomentum
training set
testing set
RMSE
alpha (10^x)


AdaGrad
training set
testing set
RMSE
alpha (10^x)

```
Algorithm = GD:
Temporary best RMSE for GD:
leaning rate (alpha): 1e-06
training set: 8.519069966605647
testing set: 9.079079972517846

Algorithm = SGD:
Temporary best RMSE for SGD:
leaning rate (alpha): 1e-06
training set: 8.57570639036693
testing set: 9.061947844718818

Algorithm = SGDwithMomentum:
Temporary best RMSE for SGDwithMomentum:
leaning rate (alpha): 1e-06
training set: 7.847651166493731
testing set: 8.36034006398783

Algorithm = SGDwithNesterovMomentum:
Temporary best RMSE for SGDwithNesterovMomentum:
leaning rate (alpha): 1e-06
training set: 7.884938403463604
testing set: 8.367283335108894

Algorithm = AdaGrad:
Temporary best RMSE for AdaGrad:
leaning rate (alpha): 0.1
training set: 6.630060878645883
testing set: 6.856824312306674

Process finished with exit code 0
```

To get the best RMSE value, the AdaGrad algorithm needs a learning rate around 0.1 while other 4 needs a learning rate around 1E-6.

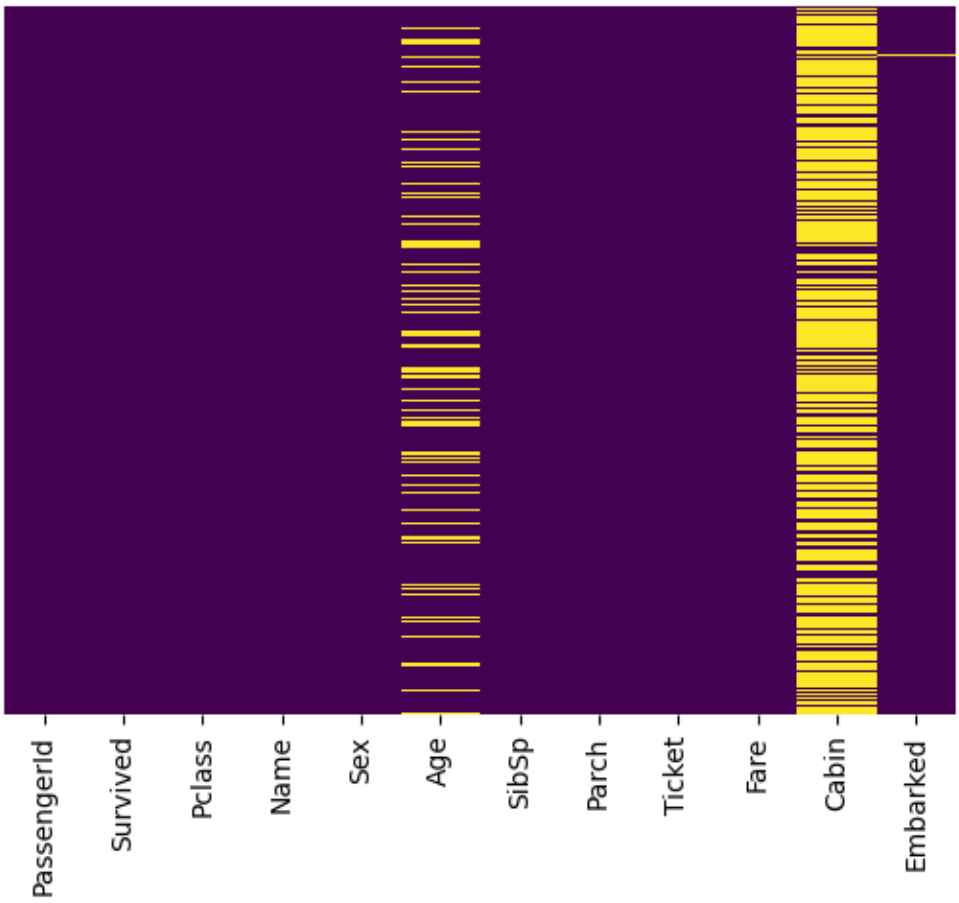
Logistic Regression, Titanic dataset

Q3: Use the python library (sklearn.linear model) to train a logistic regression model for the Titanic dataset: https://blog.goodaudience.com/machine-learning-using-logistic-regression-in-python-with-code-ab3c7f5f3bed. Split the dataset to a training set (80% samples) and a testing set (20% samples). Report the overall classification accuracies on the training and testing sets and report the precision, recall, and F-measure scores for each of the two classes on the training and testing sets.

1) Data analysis:

To be clear, the data provided in the url in our instruction has already divided into training set and testing set. But the instruction of this question requires us to exam the testing set. However, the given testing set does not have values for “Survived”, I decided to only use the training set and divide it into new training set and testing set by 8:2 to finish this work.

For this dataset, we need to evaluate each feature and impute or drop some of them. This is because some of them have NaN values and some of them has no relation with Survival.



Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

First, we use heatmap to view the data we have.
For all features, “Survived” is the target value which can also be saved as y. “PassengerID”, “Name”, “Ticket” has not relation with “Survived” and they are hard to transfer to suitable form at the same time. Hence, I decided to drop them.

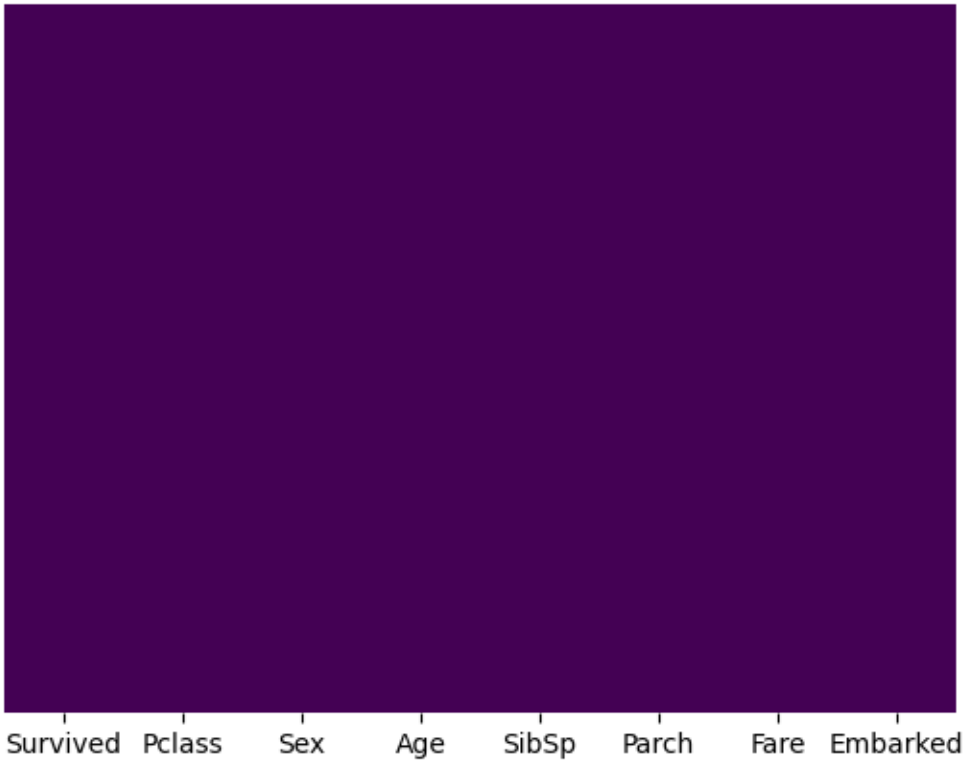
For “Cabin”, it may contain some information, but it has too much NaN values. If we insist to use it, it will make our whole dataset lose more information. Hence, drop it.

```
'Cabin':
NaN percentage in Cabin: 0.7710437710437711
Drop 'Cabin'.
```

For “Age”, it has some NaN values, but not as much as Cabin. I use the method from the given url to impute age by replace NaN with the average age of each group which is grouped by “Pclass”.

```
'Age':
float64
Has NaN?: True
              Age
Pclass
1       38.233441
2       29.877630
3       25.140620
```

For “Embarked”, it might has only one or two NaN data point. I decide to use the mode which is “S” represent for Southampton.
Also, for “Sex” and “Embarked”, we need to transform the string to integer.
“Sex”: “male” --> 0, “female” --> 1
“Embarked”: “C” --> 0, “Q” --> 1, “S” --> 2
After all of these, we get the dataset which can be used for logistic regression.



```
(891, 8)
Survived      int64
Pclass        int64
Sex           int64
Age         float64
SibSp         int64
Parch         int64
Fare        float64
Embarked      int64
dtype: object
     Survived  Pclass  Sex    Age  SibSp  Parch     Fare  Embarked
605         0       3    0  36.00      1      0  15.5500         2
382         0       3    0  32.00      0      0   7.9250         2
313         0       3    0  28.00      0      0   7.8958         2
281         0       3    0  28.00      0      0   7.8542         2
816         0       3    1  23.00      0      0   7.9250         2
880         1       2    1  25.00      0      1  26.0000         2
25          1       3    1  38.00      1      5  31.3875         2
78          1       2    0   0.83      0      2  29.0000         2
511         0       3    0  25.14      0      0   8.0500         2
668         0       3    0  43.00      0      0   8.0500         2
feature_names:
['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
Training set size:712
Testing set size:179
```

Training set size: 354/(354+152) = 0.7991 = 0.8

2) Model training:
With the logistic regression function in sklearn module, we can get the result with accuracy of training set for 0.8104 and testing set for 0.7989. I set the max_iter=200 here.

```
Accuracies of training set: 0.8103932584269663
Accuracies of testing set: 0.7988826815642458
The precision, recall rates, and the F1-score of training set:
(array([0.83189655, 0.77016129]), array([0.87133183, 0.71003717]), array([0.85115766, 0.73887814]), array([443, 269]))
The precision, recall rates, and the F1-score of testing set:
(array([0.81818182, 0.76811594]), array([0.8490566, 0.7260274]), array([0.83333333, 0.74647887]), array([106,  73]))
```

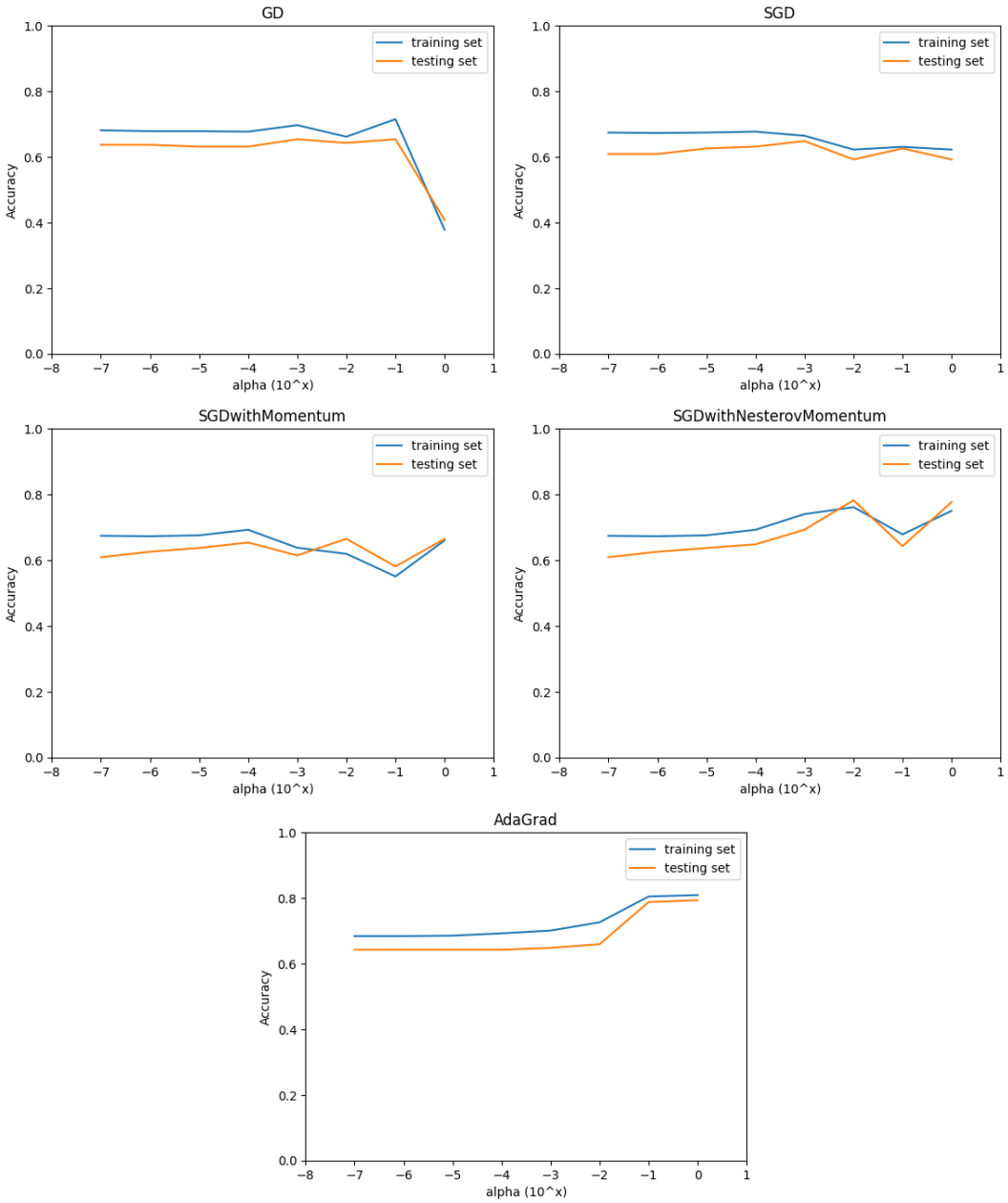
Q4: Implement the following five algorithms to train a logistic regression model for the Titantic dataset: https://blog.goodaudience.com/machine-learning-using-logistic-regression-in-python-with-code-ab3c7f5f3bed. Split the dataset to a training set (80% samples) and a testing set (20% samples). Report the overall classification accuracies on the training and testing sets and report the precision, recall, and F-measure scores for each of the two classes on the training and testing sets.
1. The gradient descent algorithm
2. The stochastic gradient descent (SGD) algorithm
3. The SGD algorithm with momentum
4. The SGD algorithm with Nesterov momentum
5. The AdaGrad algorithm

For this question, I implement my own logistic regression class.

```
class MyLogisticRegression:
    def __init__(self, _a_id, _alpha=1e-2, _steps=1e3, _batch=50, _eta=0.9, _epsilon=1e-6):

        if a_id not in range(5):
            print("Unvalid algorithm id.")
            return
        np.random.seed(SEED)
        self.a_id =  a_id
```

You can check it in code part. I use different a_id which is short for algorithm id to represent different algorithms. It has much similar place with the MyLinearRegression class which is implemented in Q2. I also write new score and predict function for it.

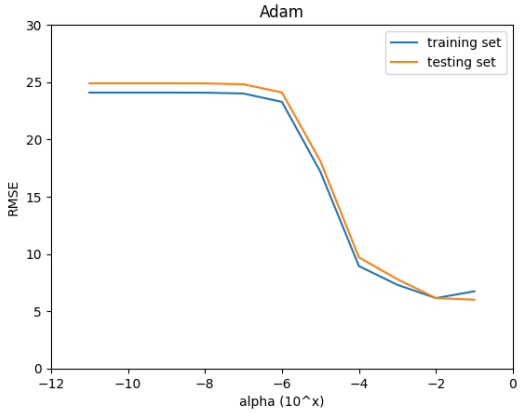
Compared with the result we got in Q2, the learning rate does not affect accuracy much here.



Below is the output for terminal to show the overall classification accuracies on the training and testing sets and report the precision, recall, and F-measure scores. I choose the output from the ones which have the largest accuracies for two sets.

```
Algorithm = GD:
Temporary largest accuracies for GD:
leaning rate (alpha): 0.1
training set: 0.7148876404494382
The precision, recall rates, and the F1-score of training set:
(0.737376473786325, 0.7148876404494382, 0.7190884860182442, None)
testing set: 0.6536312849162011
The precision, recall rates, and the F1-score of training set:
(0.6699572461797055, 0.6536312849162011, 0.6565984142467249, None)

Algorithm = SGD:
Temporary largest accuracies for SGD:
leaning rate (alpha): 0.001
training set: 0.6643258426966292
The precision, recall rates, and the F1-score of training set:
(0.7051401547890311, 0.6643258426966292, 0.583885510821457, None)
testing set: 0.6480446927374302
The precision, recall rates, and the F1-score of training set:
(0.7461640963882291, 0.6480446927374302, 0.5610758613716221, None)


Algorithm = SGDwithMomentum:
Temporary largest accuracies for SGDwithMomentum:
leaning rate (alpha): 0.0001
training set: 0.6924157303370787
The precision, recall rates, and the F1-score of training set:
(0.7257974026369477, 0.6924157303370787, 0.6372834328084233, None)
testing set: 0.6536312849162011
The precision, recall rates, and the F1-score of training set:
(0.6994780946780358, 0.6536312849162011, 0.5871373235128141, None)

Algorithm = SGDwithNesterovMomentum:
Temporary largest accuracies for SGDwithNesterovMomentum:
leaning rate (alpha): 0.01
training set: 0.7612359550561798
The precision, recall rates, and the F1-score of training set:
(0.7846835441689497, 0.7612359550561798, 0.7647913937627228, None)
testing set: 0.7821229050279329
The precision, recall rates, and the F1-score of training set:
(0.7967835237388311, 0.7821229050279329, 0.7839776458811808, None)


Algorithm = AdaGrad:
Temporary largest accuracies for AdaGrad:
leaning rate (alpha): 1
training set: 0.8089887640449438
The precision, recall rates, and the F1-score of training set:
(0.8081984956943733, 0.8089887640449438, 0.8045413233021718, None)
testing set: 0.7932960893854749
The precision, recall rates, and the F1-score of training set:
(0.7954095693277144, 0.7932960893854749, 0.7882913203891924, None)

Process finished with exit code 0
```

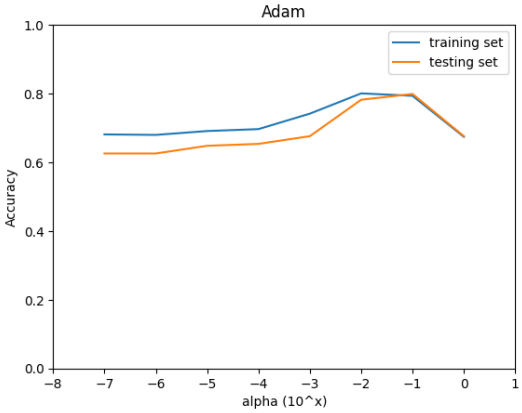
Q5-1: Implement the Adam algorithm to train a linear regression model for the Boston housing data set: https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155.
Split the dataset to a training set (70% samples) and a testing set (30% samples). Report the root mean squared errors (RMSE) on the training and testing sets.




```
Algorithm = Adam:
Temporary best RMSE for Adam:
leaning rate (alpha): 0.01
training set: 6.145470282852703
testing set: 6.150248041480363

Process finished with exit code 0
```

Combined with the results we got in Q1 and Q2, Adam algorithm seems to use large learning rate like AdaGrad among 0.01 while other simple GD or SGD algorithms use smaller learning rate among 1e-6. This might cause by algorithms' basic logic. Adaptive learning rate will use larger learning rate at the beginning and reduce it later. By this, it can use larger learning rate. Other algorithms use constant learning rate, so they should consider the situation when it comes to convergence.
For all RMSEs we got, the one trained by sklearn is the lowest. This also represents that our parameters still need adjust. Besides sklearn, Adam is the best.

Q5-2: Implement the Adam algorithm to train a logistic regression model for the Titantic dataset: https://blog.goodaudience.com/machine-learning-using-logistic-regression-in-python-with-code-ab3c7f5f3bed.
Split the dataset to a training set (80% samples) and a testing set (20% samples). Report the overall classification accuracies on the training and testing sets and report the precision, recall, and F-measure scores for each of the two classes on the training and testing sets.



```
Algorithm = Adam:
Temporary largest accuracies for Adam:
leaning rate (alpha): 0.1
training set: 0.7935393258426966
The precision, recall rates, and the F1-score of training set:
(0.7923660615262236, 0.7935393258426966, 0.7928211145441844, None)
testing set: 0.7988826815642458
The precision, recall rates, and the F1-score of training set:
(0.7976632202117901, 0.7988826815642458, 0.7973406942603805, None)

Process finished with exit code 0
```

The result we got in Q5-2 keep consistent with the results we got in Q3 and Q4. However, in all accuracies we got, the AdaGrad has the highest accuracy.