CSE 412: Machine Learning Lab
Lab Activity 3: Linear Regression

Dr Muhammad Abul Hasan[*]

March 4, 2024

1 Objective

The objective of learning linear regression is to develop an understanding of a fundamental statistical and machine learning technique used for predictive modeling and understanding the relationships between variables. Linear regression is a simple yet powerful method used in various fields, including statistics, economics, finance, and machine learning. Here are the primary objectives of learning linear regression:

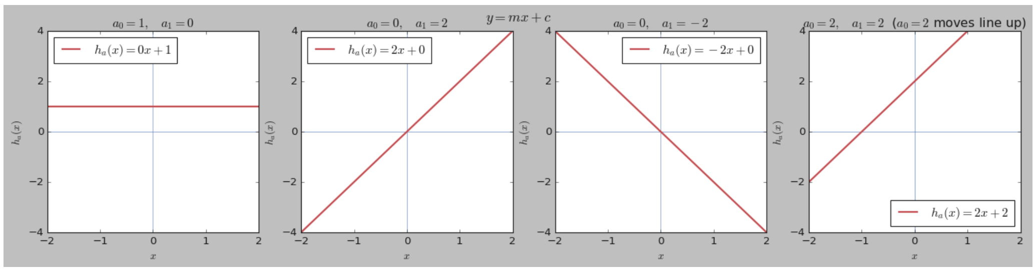
- **Understanding the Basics**: Learn the fundamental concepts of linear regression, including the terminology (dependent and independent variables, coefficients, intercept, etc.) and the mathematical representation of linear regression models.
- **Model Building**: Learn how to build a linear regression model by selecting appropriate independent variables (features) and estimating coefficients that best fit the data.
- **Interpretation**: Develop the ability to interpret the coefficients of a linear regression model. Understand how changes in the independent variables affect the dependent variable.

2 Understanding the Concept of Linear Regression

To understand the concept of linear regression, we will use a small set of "random" data to make the presentation clearer. The hypothesis function $h_a(x)$ can be written as follows:

$$h_a(x) = a_0 + a_1 x.$$

The above equation is an equation for a straight line with a *slope* of a_1 and an *intercept* of a_0. We say that $h_a(x)$ is a linear function of x that is parameterized by a_0 and a_1. The following are a few plots with different values of a_0, and a_1.



[*] Associate Professor, Deptartment of CSE, GUB ⁘ ✉ muhammad.hasan@cse.green.edu.bd

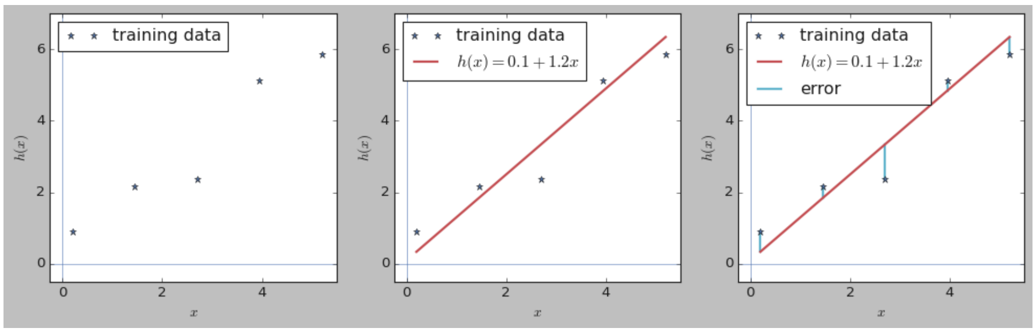
In this case, we define our model as $h_a(x)$ where h represents our predicted outcome as a function of our features, x. Typically, the model of a line is represented by

$$y = mx + b.$$

We use h instead of y to denote that this model is a hypothesis of the *true* trend. In a sense, we are attempting to provide our "best guess" for finding a relationship in the data provided. The *parameters* a_0 and a_1 are coefficients of our linear equation (analogous to b and m in the standard line equation) and will be considered parameters of the model.

2.1 What we would like to achieve?

In the following plot, there are five random "training samples" (left figure), a model function (line) $h_a(x) = 0.1 + 1.2x, (a_0 = 0.1, a_1 = 1.2)$ along with training samples (middle figure) and vertical cyan lines from the model function to the data points (right figure). The lengths of those cyan lines are the errors between the model function $h_a(x)$ and the training set data points.



The goal of linear regression is to make the sum of all of those errors as small as possible by finding the "best" parameters a_0, and a_1 for the hypothesis function $h_a(x)$.

2.2 Least squares error

When we have multiple training samples each sample is represented by $x^{(i)}$ and $y^{(i)}$ where i represents the index of the training sample. Our model is $h_a(x) = a_0 + a_1 x$ can be used for an approximation of $y^{(i)}$ at any given value $x^{(i)}$. Thus, the error for the whole set of training samples can be written as:

$$Error = h_a(x^i) - y^{(i)}$$

That is just the difference between the value or model predicts and the actual value in the training set. These errors can be positive or negative. We want to add them all up and make that sum as small as possible. We want the errors to be positive numbers. A simple way to to that is to square those errors.

$$SquaredError = \left(h_a(x^i) - y^{(i)}\right)^2$$

Now add them up for all training samples and and call it $J(a_0, a_1)$ (why J? ... it is a common choice by the machine learning community).

$$\begin{aligned} J(a_0, a_1) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h_a(x^i) - y^{(i)}\right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left((a_0 + a_1(x^i)) - y^{(i)}\right)^2 \end{aligned}$$

In words, this says: take the summation, (Σ), of the squares of the errors between the model and all m data points in the training set. The constant factor $\frac{1}{m}$ averages the squared error and $\frac{1}{2}$ is in there for convenience when we take derivatives later.

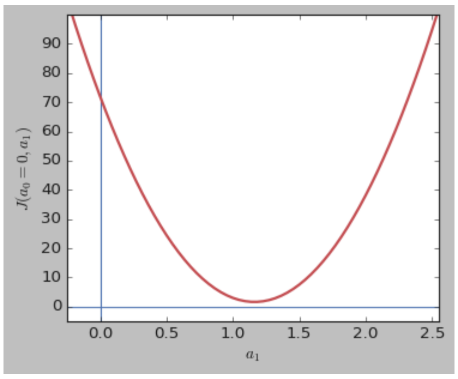
$J(a_0, a_1)$ is a function of the parameters a_0, a_1. The goal is to minimize J with respect to a_0, and a_1. Formally we write this as follows:

$$\min_{a_0, a_1} J(a_0, a_1)$$

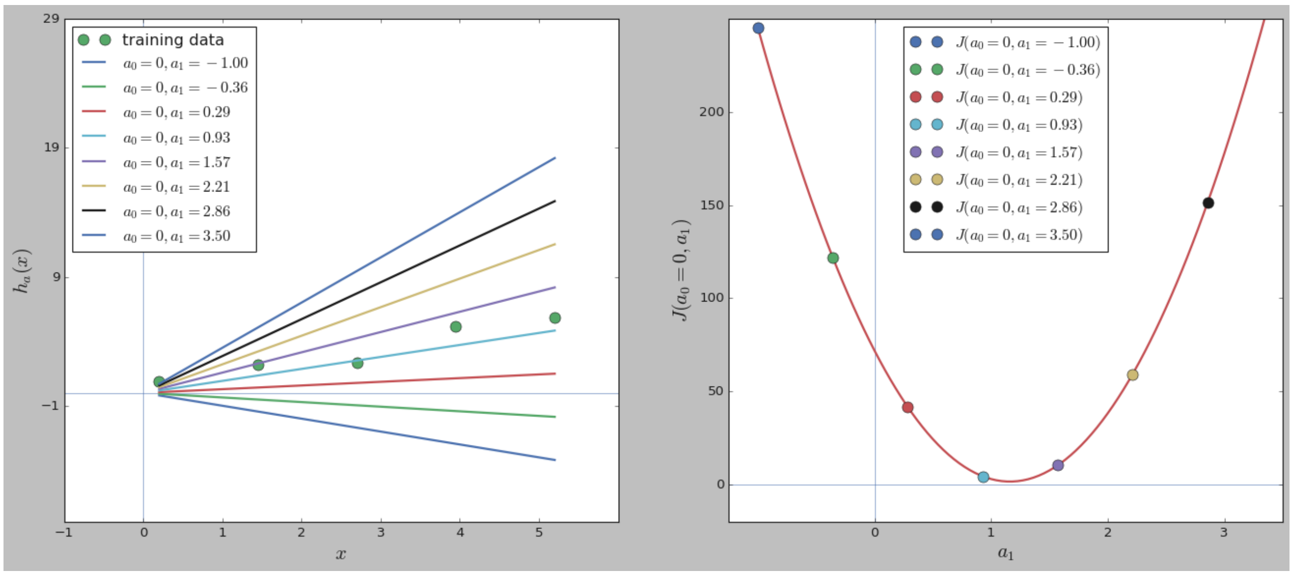
Meaning, we are looking for the parameter values for a_0, and a_1 so that the cost $J(a_0, a_1)$ is minimum (also called optimum). This is an optimization problem and we will see how we can perform such a task.

2.3 Visualizing the Cost Function

J is a sum of squares, a second-order polynomial equation. That means that it is parabolic. If we set $a_0 = 0$ (or some other constant value) and just look at J as a function of a_1 then it is a simple parabola with an obvious minimum.



To make it clear how the regression line relates to the cost function, in the following the test data we used earlier with a set of lines for a range of values of a_1 and $a_0 = 0$ are plotted. Additionally, the cost values $J(a_0 = 0, a_1)$ plotted for different a_1.

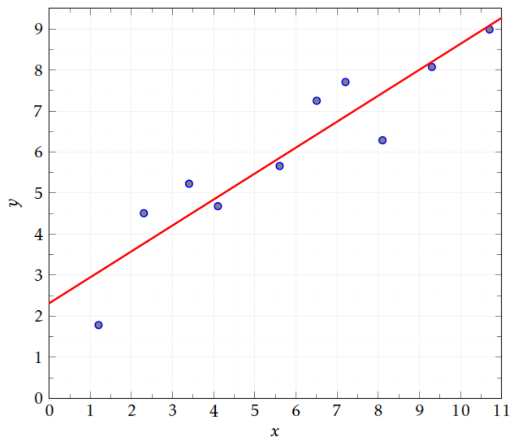


You can see that the ($a_0 = 0$ and $a_1 = 0.93$) line is the closest to the minimum. Our task is to find the best possible parameter a_1 (for this particular problem).

3 Linear regression

Linear regression is used to predict an outcome given some input value(s). While machine learning classifiers use features to predict a discrete label for a given instance or example, machine learning regressors have the ability to use features to predict a continuous outcome for a given instance or example. For example, a classifier might draw a decision boundary that can tell you whether or not a house is likely to sell at a given price (when provided with features of the house) but a regressor can use those same features to predict the market value of the house. Nonetheless, regression is still a supervised learning technique and you will still need to train your model on a set of examples with known outcomes.

The basic premise behind linear regression is to provide a model which can observe linear trends in the data; you have probably encountered linear regression at some point in the past as finding the "line of best fit".



3.1 The model

Our machine learning model is represented by the equation of a line.

$$h_a(x) = a_0 + a_1 x$$

So far, our model has only been capable of capturing one independent variable and mapping its relationship to one dependent variable. We will now extend our linear model to allow for n different independent variables (in our case, features) to predict one dependent variable (an outcome).

3.2 The cost function

In order to find the optimal line to represent our data, we need to develop a cost function that favors model parameters that lead to a better fit. This cost function will approximate the error found in our model, which we can then use to optimize our parameters, a_0 and a_1, by minimizing the cost function. For linear regression, we will use the mean squared error - the average difference between our guess, $h_a(x_i)$, and the true value, y_i - to measure performance.

$$\min \frac{1}{2m} \sum_{i=1}^{m} \left(h_a(x_i) - y_i\right)^2$$

Substituting in our definition of the model, $h_a(x_i)$, we update our cost function as:

$$J(a) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\sum_{j=0}^{n} a_j x_{i,j} \right) - y_i \right)^2$$

4 Finding the best parameters

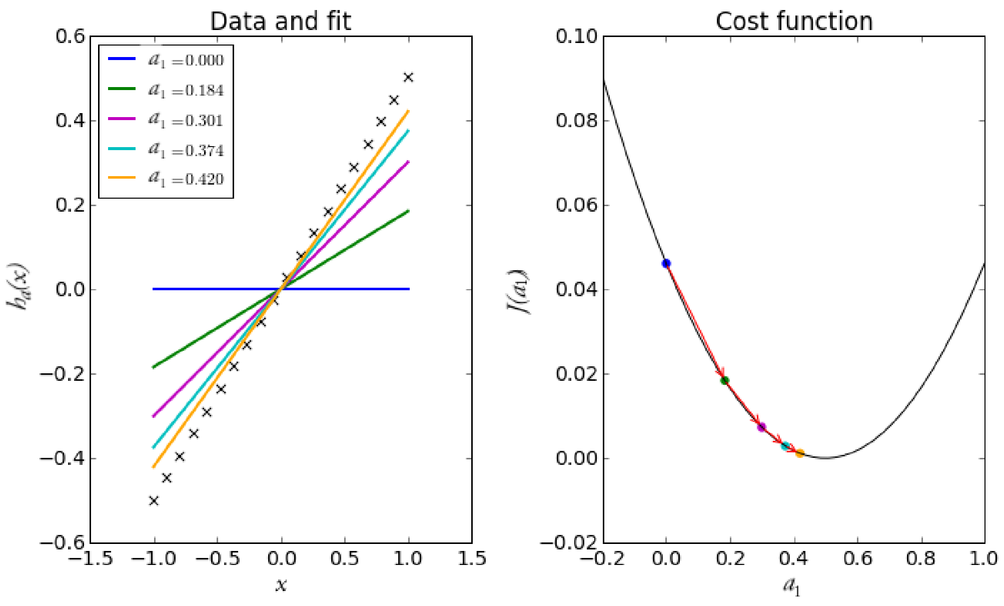
There is a number of ways to find the optimal model parameters. A few common approaches are detailed below.

4.1 Gradient descent

We use gradient descent to perform the parameter optimization in accordance with our cost function. Remember, the cost function essentially will score how well our model is fitting the present data. A smaller cost function means less error, which means we have a better model.

We first start with an initial guess for what our parameter values should be. Gradient descent will then guide us through an iterative process of updating our parameter weights in search of an optimum solution.

Let us first look at the simplest example, where we fix $a_0 = 0$ and find the optimal slope value.



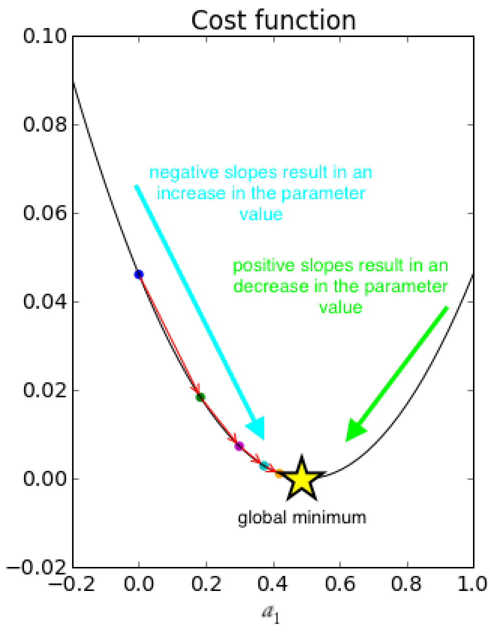
Using gradient descent, the weights will continue to update until we have converged on the optimal value. For linear regression, the cost function is always convex with only one optimum, so we can be confident that our solution is the best possible one. This is not always the case, and we will need to be wary of converging on local optima using other cost functions.

As the picture depicts, gradient descent is an iterative process that guides the parameter values toward the global minimum of our cost function.

$$a_j := a_j - \eta \frac{\partial}{\partial a_j} J(a_0, a_1)$$

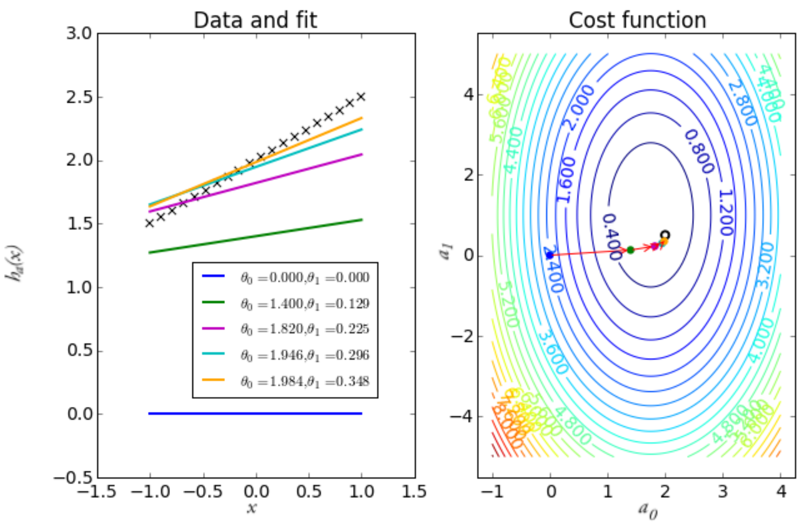
Gradient descent will update your parameter value by subtracting the current value from the slope of the cost function at that point multiplied by a learning rate, which basically dictates how far to step. The learning rate, η, is always positive.

Notice in the picture above how the first attempted value, $a_1 = 0$, has a negative slope on the cost function. Thus, we would update the current value by subtracting a negative number (in other words, addition) multiplied by the learning rate. In fact, every point left of the global minimum has a negative slope and would thus result in increasing our parameter weight. On the flip side, every value to the right of the global minimum has a positive slope and would thus result in decreasing our parameter weight until convergence.



One neat thing about this process is that gradient descent will automatically slow down as we are nearing the optimum. See how the initial iterations take relatively large steps in comparison to the last steps? This is because our slope begins to level off as we approach the optimum, resulting in smaller update steps. Because of this, there is no need to adjust η during the optimization.

We can extend this idea to update both a_0 and a_1 simultaneously. In this case, our 2D curve has become a 3D contour plot.



While the gradient descent algorithm remains largely the same,

$$a_j := a_j - \alpha \frac{\partial}{\partial a_j} J(a)$$

we now, using the chain rule, define the differential as:

$$\frac{\partial}{\partial a_j} J(a) = \frac{1}{m} \sum_{i=1}^{m} \left(\left(\sum_{j=0}^{n} a_j x_{i,j} \right) - y_i \right) x_{i,j}$$

A few notes and practical tips on this process.

- Scaling your features to be on a similar scale can greatly improve the time it takes for your gradient descent algorithm to converge.
- It is important that your learning rate is reasonable. If it is too large it may overstep the global optimum and never converge, and if it is too small it will take a very long time to converge.
- Plot the $J(a_0, \ldots, a_n)$ as a function of the number of iterations you have run gradient descent. If gradient descent is working properly it will continue to decrease steadily until leveling off, hopefully, close to zero. You can also use this plot to visually determine when your cost function has converged and gradient descent may be stopped.
- If $J(a_0, \ldots, a_n)$ begins to increase as the number of iterations increases, something has gone wrong in your optimization. Typically, if this occurs it is a sign that your learning rate is too large.

Gradient descent is often the preferred method of parameter optimization for models which have a large number (think, over $10,000$) of features.

5 Implementation

In this section, we will see how the Python Scikit-Learn library for machine learning can be used to implement regression functions. We will start with simple linear regression involving two variables and then you will move towards linear regression involving multiple variables.

In this regression task, we will predict the percentage of marks that a student is expected to score based on the number of hours they studied. This is a simple linear regression task as it involves just two variables.

To import the necessary libraries for this task, execute the following import statements:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

The following command imports the CSV dataset using pandas and retrieves the first 5 records from the dataset:

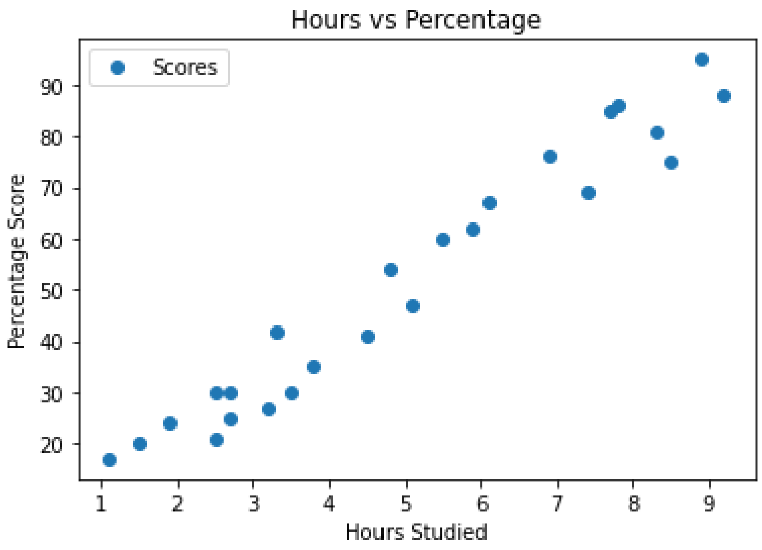
```
# https://drive.google.com/file/d/1oakZCv7g3mlmCSdv9J8kdSaq05_6dI0w/view
# Please download the dataset from the above link

dataset = pd.read_csv('/content/drive/MyDrive/student_scores.csv')
dataset.head()
```

Output:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

Let's plot the data points on 2-D graph to understand the characteristics of the dataset and see if we can manually find any relationship between the data. We can create the plot with the following script:

```
dataset.plot(x='Hours', y='Scores', style='o')
plt.title('Hours␣vs␣Percentage')
plt.xlabel('Hours␣Studied')
plt.ylabel('Percentage␣Score')
plt.show()
```



Now we have an idea about the statistical details of our data. The next step is to divide the data into "attributes" and "labels". Attributes are the independent variables while labels are dependent variables whose values are to be predicted.

In our dataset, we only have two columns. We want to predict the percentage score depending on the hours studied. Therefore our attribute set will consist of the "Hours" column, and the label will be the "Score" column. To extract the attributes and labels, execute the following script:

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

The attributes are stored in the X variable. We specified "-1" as the range for columns since we wanted our attribute set to contain all the columns except the last one, which is "Scores". Similarly, the y variable contains the labels. We specified 1 for the label column since the index for the "Scores" column is 1. Remember, the column indexes start with 0, with 1 being the second column. In the next section, we will see a better way to specify columns for attributes and labels.

Now that we have our attributes and labels, the next step is to split this data into training and test sets. We'll do this by using Scikit-Learn's built-in `train_test_split()` method:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

The above script splits 80% of the data into the training set while 20% of the data to the test set. The `test_size` variable is where we actually specify the proportion of the test set.

Now is the time to train our algorithm. Execute the following command:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

With Scikit-Learn it is extremely straightforward to implement linear regression models, as all you really need to do is import the `LinearRegression` class, instantiate it, and call the `fit()` method along with our training data. This is about as simple as it gets when using a machine-learning library to train your data.

```
print(regressor.intercept_)
print(regressor.coef_)
```

Now that we have trained our algorithm, it's time to make some predictions. To do so, we will use our test data and see how accurately our algorithm predicts the percentage score. To make predictions on the test data, execute the following script:

```
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head()
```

Output:

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

The final step is to evaluate the performance of the algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, the following evaluation metrics are commonly used:

- Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$= \frac{1}{n}\sum_{i=1}^{n}(actual_i - predicted_i)^2$$

We don't have to perform these calculations manually. The Scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Let's find the values for these metrics using our test data. Execute the following code:

```
from sklearn import metrics
print('Mean␣Squared␣Error:', metrics.mean_squared_error(y_test, y_pred))
```

Output:

```
Mean Squared Error: 21.5987693072
```

6 Lab Exercises

Please code yourself and write a report based on your findings:

1. Download the Diabetes dataset from the following link and predict if a person is diabetic using a linear regression algorithm.

   `https://www.kaggle.com/saurabh00007/diabetescsv`

2. Write a report on your experiments and experimental results and submit it to your instructor.

My Notes Date:________________