EAST WEST UNIVERSITY

Lab Report-07

Course Title: Artificial Intelligence

Course Code: CSE 366

Semester: Fall 2021

Section No: 01

Submitted By

Name: Syeda Tamanna Sheme

ID:2018-2-60-010

Submitted To

Md Al-Imran

Lecturer

Department of Computer Science & Engineering

Date of Submission: 25 December, 2021

Lab-07: Application of Gradient Descent Algorithm

Theory

Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

Optimization algorithm refers to the task of minimizing or maximizing an objective function f(x) parameterized by x. Similarly, in machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters. The main objective of gradient descent is to minimize the convex function using iteration of parameter updates. Once these machine learning models are optimized, these models can be used as powerful tools for Artificial Intelligence and various computer science applications.

Gradient descent was initially discovered by **"Augustin-Louis Cauchy"** in mid of 18th century. Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.
- Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.

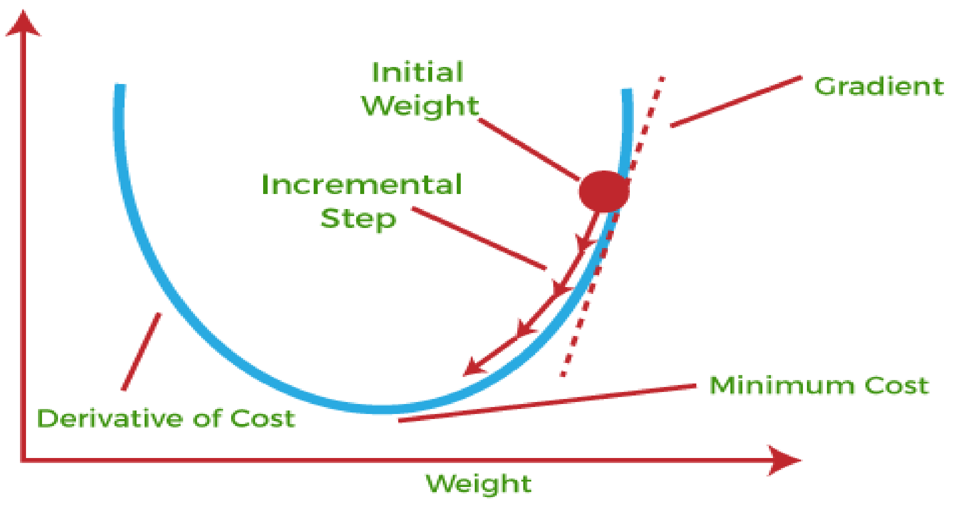


Figure: Gradient Descent

The main objective of using a gradient descent algorithm is to minimize the cost function using iteration. To achieve this goal, it performs two steps iteratively:

- Calculates the first-order derivative of the function to compute the gradient or slope of that function.
- Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.

Gradient Descent Working Procedure

Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as: **Y=mX+c**

Where 'm' represents the slope of the line, and 'c' represents the intercepts on the y-axis.

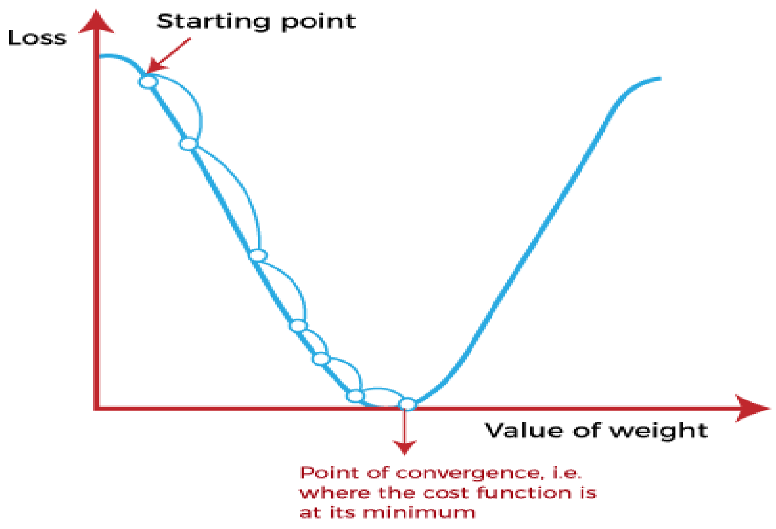


Figure: Gradient Descent

The starting point (shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias).

The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called **a point of convergence.**

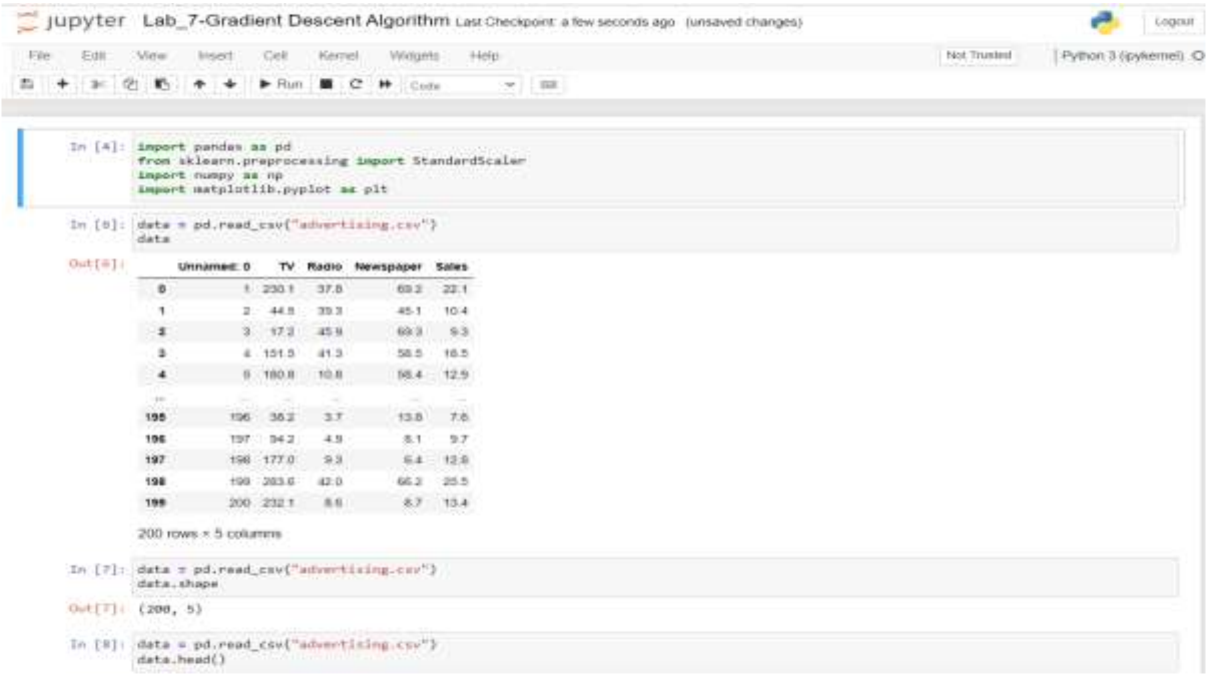
Types of gradient Descent:

- ❖ **Batch Gradient Descent:** This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.
- ❖ **Stochastic Gradient Descent:** This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.
- ❖ **Mini Batch gradient descent:** This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here b examples where b<m are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

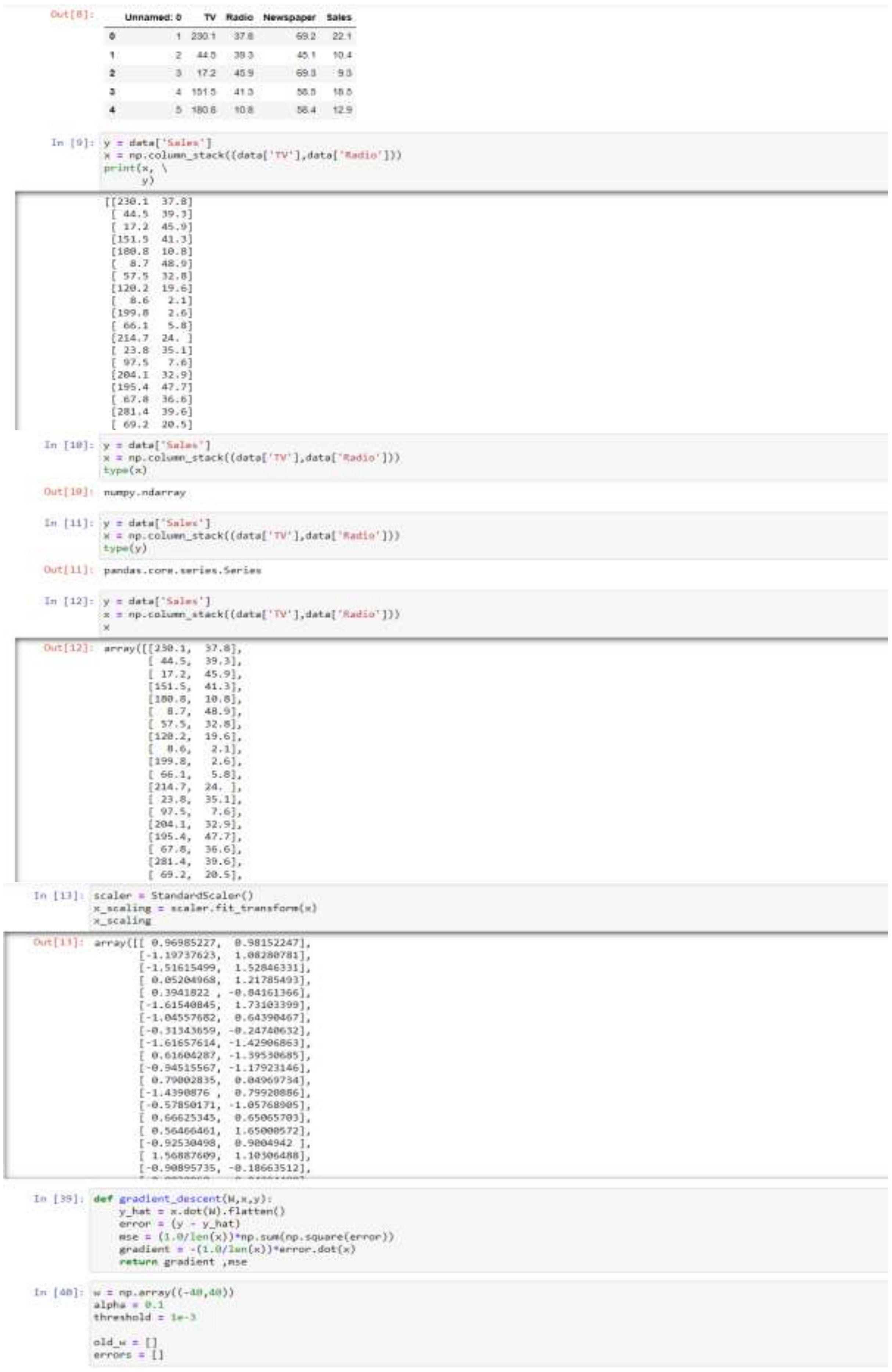
Convergence trends in different variants of Gradient Descents:
In case of Batch Gradient Descent, the algorithm follows a straight path towards the minimum. If the cost function is convex, then it converges to a global minimum and if the cost function is not convex, then it converges to a local minimum. Here the learning rate is typically held constant.

In case of stochastic gradient Descent and mini-batch gradient descent, the algorithm does not converge but keeps on fluctuating around the global minimum. Therefore, in order to make it converge, we have to slowly change the learning rate. However, the convergence of Stochastic gradient descent is much noisier as in one iteration, it processes only one training example.

<u>Code Screenshot</u>


jupyter
Lab_7-Gradient Descent Algorithm Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
In [4]: import pandas as pd
        from sklearn.preprocessing import StandardScaler
        import numpy as np
        import matplotlib.pyplot as plt

In [6]: data = pd.read_csv("advertising.csv")
        data

Out[6]:
        Unnamed: 0    TV  Radio  Newspaper  Sales
    0            1 230.1   37.8       69.2   22.1
    1            2  44.5   39.3       45.1   10.4
    2            3  17.2   45.9       69.3    9.3
    3            4 151.5   41.3       58.5   18.5
    4            5 180.8   10.8       58.4   12.9
  ...          ...   ...    ...        ...    ...
  195          196  38.2    3.7       13.8    7.6
  196          197  94.2    4.9        8.1    9.7
  197          198 177.0    9.3        6.4   12.8
  198          199 283.6   42.0       66.2   25.5
  199          200 232.1    8.6        8.7   13.4

200 rows × 5 columns

In [7]: data = pd.read_csv("advertising.csv")
        data.shape

Out[7]: (200, 5)

In [8]: data = pd.read_csv("advertising.csv")
        data.head()
```



Out[8]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

```
In [9]: y = data['Sales']
        x = np.column_stack((data['TV'],data['Radio']))
        print(x, \
              y)
```

```
[[230.1  37.8]
 [ 44.5  39.3]
 [ 17.2  45.9]
 [151.5  41.3]
 [180.8  10.8]
 [  8.7  48.9]
 [ 57.5  32.8]
 [120.2  19.6]
 [  8.6   2.1]
 [199.8   2.6]
 [ 66.1   5.8]
 [214.7  24. ]
 [ 23.8  35.1]
 [ 97.5   7.6]
 [204.1  32.9]
 [195.4  47.7]
 [ 67.8  36.6]
 [281.4  39.6]
 [ 69.2  20.5]
```

```
In [10]: y = data['Sales']
         x = np.column_stack((data['TV'],data['Radio']))
         type(x)

Out[10]: numpy.ndarray
```

```
In [11]: y = data['Sales']
         x = np.column_stack((data['TV'],data['Radio']))
         type(y)

Out[11]: pandas.core.series.Series
```

```
In [12]: y = data['Sales']
         x = np.column_stack((data['TV'],data['Radio']))
         x

Out[12]: array([[230.1,  37.8],
                [ 44.5,  39.3],
                [ 17.2,  45.9],
                [151.5,  41.3],
                [180.8,  10.8],
                [  8.7,  48.9],
                [ 57.5,  32.8],
                [120.2,  19.6],
                [  8.6,   2.1],
                [199.8,   2.6],
                [ 66.1,   5.8],
                [214.7,  24. ],
                [ 23.8,  35.1],
                [ 97.5,   7.6],
                [204.1,  32.9],
                [195.4,  47.7],
                [ 67.8,  36.6],
                [281.4,  39.6],
                [ 69.2,  20.5],
```

```
In [13]: scaler = StandardScaler()
         x_scaling = scaler.fit_transform(x)
         x_scaling

Out[13]: array([[ 0.96985227,  0.98152247],
                [-1.19737623,  1.08280781],
                [-1.51615499,  1.52846331],
                [ 0.05204968,  1.21785493],
                [ 0.3941822 , -0.84161366],
                [-1.61540845,  1.73103399],
                [-1.04557682,  0.64390467],
                [-0.31343659, -0.24740632],
                [-1.61657614, -1.42906863],
                [ 0.61604287, -1.39530685],
                [-0.94515567, -1.17923146],
                [ 0.79002835,  0.04969734],
                [-1.4390876 ,  0.79920886],
                [-0.57850171, -1.05768905],
                [ 0.66625345,  0.65065703],
                [ 0.56466461,  1.65000572],
                [-0.92530498,  0.9004942 ],
                [ 1.56887609,  1.10306488],
                [-0.90895735, -0.18663512],
```

```
In [39]: def gradient_descent(W,x,y):
             y_hat = x.dot(W).flatten()
             error = (y - y_hat)
             mse = (1.0/len(x))*np.sum(np.square(error))
             gradient = -(1.0/len(x))*error.dot(x)
             return gradient ,mse
```

```
In [40]: w = np.array((-40,40))
         alpha = 0.1
         threshold = 1e-3

         old_w = []
         errors = []
```

In [41]:
```python
##Gradient descent Loop
iterations = 1
for i in range(200):
    gradient, error = gradient_descent(w, x_scaling , y)
    new_w = w - alpha*gradient

    if iterations%10 == 0: ##Printing error every 10 iters
        print("Iteration: %d)- Error: %.4f" % (iterations,error))
        old_w.append(new_w)
        errors.append(error)

    #Stoping Criterion

    if np.sum(abs(new_w - w)) < threshold:
        print("Converged")
        break

    iterations += 1
    w = new_w
    ##print(w)

print("W = ",w)
```

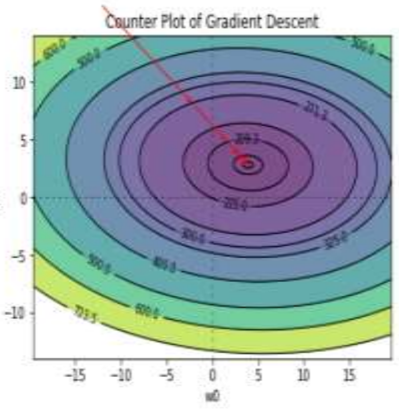
```
Iteration: 10)- Error: 723.4665
Iteration: 20)- Error: 271.2587
Iteration: 30)- Error: 209.2671
Iteration: 40)- Error: 200.7664
Iteration: 50)- Error: 199.6005
Iteration: 60)- Error: 199.4405
Iteration: 70)- Error: 199.4186
Iteration: 80)- Error: 199.4156
Iteration: 90)- Error: 199.4151
Converged
W =  [3.91344732 2.78882878]
```

In [43]:
```python
ws = np.array(old_w)

levels = np.sort(np.array(errors))
```

In [44]:
```python
w0 = np.linspace(-w[0]*5, w[0]*5, 100)
w1 = np.linspace(-w[1]*5, w[1]*5, 100)
e_vals = np.zeros(shape = (w0.size, w1.size))
```

In [45]:
```python
for i , val1 in enumerate(w0):
    for j, val2 in enumerate(w1):
        temp = np.array((val1,val2))
        e_vals[i, j] = gradient_descent(temp, x_scaling, y)[1]
```

In [47]:
```python
plt.contourf(w0, w1, e_vals, levels, alpha = 0.7)
plt.axhline(0, color = 'black', alpha = 0.5 , dashes = [2,4], linewidth = 1)
plt.axvline(0, color = 'black', alpha = 0.5 , dashes = [2,4], linewidth = 1)

for i in range(len(old_w) - 1):
    plt.annotate('', xy = ws[i +1 , :], xytext = ws[i,:],
                 arrowprops = {'arrowstyle': '->', 'color': 'r', 'lw':1},
                 va = 'center', ha = 'center')

CS = plt.contour(w0, w1, e_vals, levels, linewidths = 1, colors='black')
plt.clabel(CS, inline =1 , fontsize = 8)
plt.title("Counter Plot of Gradient Descent")
plt.xlabel("w0")
plt.ylabel("w1")
plt.show()
```



Results

After completing the lab task and activity, we learn the Gradient Descent Algorithm. We also learn how to use the Gradient Descent Algorithm to add data, read data, and make graphics. We can now develop shows around these topics.

Discussion

Python is a general-purpose, versatile and popular programming language. It's great as a first language because it is concise and easy to read, and it is also a good language to have in any programmer's stack as it can be used for everything from web development to software development and data science applications.

This lab task is a great introduction to both fundamental programming concepts and the Python programming language. Python 3 is the most up-to-date version of the language with many improvements made to increase the efficiency and simplicity of the code that we write. Day by-day, python new version are realising and all the new versions have new features and these new features are better than previous one. So, finally I can say that python will be more user friendly in future.