
Name: Afrida Islam

Student ID: 21221030

✓ EEE385IL (MACHINE LEARNING LABORATORY)

LAB 1: Introduction to Python

Different Types of Data in Python

Variables can store data of different types, and different types can do different things. Python has the following data types built-in by default:

- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: list, tuple, range
- Mapping Type: dict
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview
- None Type: NoneType

✓ Strings

A string is a series of characters, surrounded by single or double quotes

```
print("Hello world!")
```

```
→ Hello world!
```

```
msg = "Hello world!"  
print(msg)
```

```
→ Hello world!
```

Concatenation (combining strings)

```
first_name = 'albert'  
last_name = 'einstein'  
full_name = first_name + ' ' + last_name  
print(full_name)
```

```
→ albert einstein
```

✓ Numeric Data

```
x = 20      #int  
print(x)  
y = 20.5    #float  
print(y)  
z = 1j      #complex  
print(z)
```

```
→ 20  
20.5  
1j
```

✓ Getting the Data Type

```
x = 2 + 3j  
print(type(x))  
a = 5.0
```

```
→ <class 'complex'>
```

```
print(type(a))
```

```
↩ <class 'float'>
```

```
a = 20
b = 30
print(a+b)
```

```
↩ 50
```

```
a = a + a
print(a)
```

```
↩ 200
```

```
a = 50
print(a+b)
```

```
↩ 80
```

✓ Taking User Input

```
name = input("Please input your name: ")
print("Hello " + name + " !")
```

```
↩ Please input your name: Afriiii
Hello Afriiii !
```

```
f_name = input("Please input your first name: ")
l_name = input("Please input your last name: ")
print("Hello " + f_name + " " + l_name + " !")
```

```
↩ Please input your first name: Afrida
Please input your last name: Islam
Hello Afrida Islam !
```

```
text = "The value of the variable a is {} and x is {}".format(a, x)
print(text)
```

```
↩ The value of the variable a is 200 and x is (2+3j)
```

```
print(f"The values are: {a} and {x}")
```

```
↩ The values are: 200 and (2+3j)
```

```
a = int(input("First no: "))
b = int(input("Second no: "))
print (f"The summation is {a+b}")
```

```
↩ First no: 10
Second no: 21
The summation is 31
```

✓ TASK 1

Take input your **Name** and **ID** and then store them in two separate variables. Print your **Name** and **ID** after concatenation.

```
#####
##### Code starts #####
```

```
Name = input("Enter Name:")
ID = input("Enter ID:")
print(Name+ " "+ ID)
##### Code ends #####
#####
```

```
↩ Enter Name:Afrida
Enter ID:21221030
Afrida 21221030
```

```
n1 = input("Enter the first number: ")
n2 = input("Enter the second number: ")
```

```
add = int(n1) + int(n2) # type casting
print(add)
```

```
➦ Enter the first number: 10
Enter the second number: 20
30
```

```
print((type(n1)))
print((type(add)))
```

```
➦ <class 'str'>
<class 'int'>
```

✓ Conditional Statements

Conditional operators:

- equal --> ==
- NOT equal --> !=
- greater than or equal --> >=
- less than or equal --> <=

if-else statements

```
if condition:
```

```
    statements if condition is TRUE
```

```
elif condition:
```

```
    statements
```

```
else:
```

```
    statements
```

```
marks = int(input("Enter: "))
if marks > 90:
    print('Congrats! A!')
elif marks > 80:
    print("Alright!")
else:
    print('Need to do better!')
```

```
➦ Enter: 95
Congrats! A!
```

Multiple Condition checking

and --> TRUE if both conditions are met

or --> TRUE if any one condition is met

✓ TASK 2

```
if cg > 3.8 && credits > 30
scholarship 10%
if cg > 3.5 && credits > 30
scholarship 5%
if cg < 3.5 || credits < 30
no scholarship
```

```
#####
##### Code starts #####
cgpa =float(input("Enter Cgpa:"))
cred = int(input("Enter Cred:"))
if cgpa > 3.8 and cred > 30:
    print("Scholarship 10%")
elif cgpa > 3.5 and cred > 30:
    print("Scholarship 5%")
```

```

else:
    print("No Scholarship")

##### Code ends #####
#####

Enter Cgpa:3.7
Enter Cred:111
Scholarship 5%

```

Loops

for loop

```

for i in range(N):
    statements

for a in range(0,10,2):
    print(a) # indexing starts from 0 in python

0
2
4
6
8

```

while loop

```

while condition:
    statements

a = 5

while a <= 10:
    print(a)
    a = a + 1

5
6
7
8
9
10

```

TASK 3

Take input of the cgpa & credits of 3 students using loop and do the same thing as TASK 2 for each of the students

```

##### Code starts #####
for a in range(1,4):
    cgpa =float(input(f"Enter Cgpa of student {a}:"))
    cred = int(input(f"Enter Cred of student {a}:"))

    if cgpa > 3.8 and cred > 30:
        print("Scholarship 10%")
    elif cgpa > 3.5 and cred > 30:
        print("Scholarship 5%")
    else:
        print("No Scholarship")

##### Code ends #####

Enter Cgpa of student 1:3.8
Enter Cred of student 1:120
Scholarship 5%
Enter Cgpa of student 2:3.2
Enter Cred of student 2:66
No Scholarship
Enter Cgpa of student 3:3.0
Enter Cred of student 3:30

```

No Scholarship

✓ List (like Array in C programming)

```
list_name = [val_1, val_2, val_3, .....]
```

```
marks = [88, 56, 94, 92, 76]
```

```
print(len(marks))
```

```
for mark in marks:
    print("The marks of Student is {}".format(mark)) #{} placeholder in python
```

```
5
The marks of Student is 88
The marks of Student is 56
The marks of Student is 94
The marks of Student is 92
The marks of Student is 76
```

✓ TASK 4

Store the cgpa and completed credits of 5 students in two separate lists and then print their scholarship status.

```
#####
##### Code starts #####
cgpa = [3.6, 2.3, 2.8, 3.9, 4.0]
credits = [120, 65, 130, 25, 100]

min_cgpa = 3.5
min_credits = 30

for i in range(len(cgpa)):
    if cgpa[i] >= min_cgpa and credits[i] >= min_credits:
        print(f"Student {i+1} is eligible for scholarship!")
    else:
        print(f"Student {i+1} is not eligible for scholarship")

##### Code ends #####
#####
```

```
5 Student 1 is eligible for scholarship!
Student 2 is not eligible for scholarship
Student 3 is not eligible for scholarship
Student 4 is not eligible for scholarship
Student 5 is eligible for scholarship!
```

✓ Common List operations

```
marks = [88, 56, 94, 82, 94]
print("Before changing any element --> ", marks)
```

```
marks.append(99)
print("After appending one element --> ", marks)
```

```
marks.remove(94)
print("After removing one element --> ", marks)
```

```
marks.insert(1, 71)
print("After inserting '71' at index '1' --> ", marks)
```

```
marks.pop(2)
print("After popping out the element at index '2' --> ", marks)
```

```
5 Before changing any element --> [88, 56, 94, 82, 94]
After appending one element --> [88, 56, 94, 82, 94, 99]
After removing one element --> [88, 56, 82, 94, 99]
After inserting '71' at index '1' --> [88, 71, 56, 82, 94, 99]
After popping out the element at index '2' --> [88, 71, 82, 94, 99]
```

✓ Downloading file from Colab as PDF

```
# Install the package for Tex and then convert to PDF directly as LaTeX
!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic

# Provide the file path of the notebook file
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/Afrida_Task01.ipynb"
```

```
➔ Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  dvipng fonts-droid-fallback fonts-lato fonts-lmodern fonts-oto-mono
  fonts-texgyre fonts-urw-base35 libapache-pom-java libcommons-logging-java
  libcommons-parent-java libfontbox-java libfontenc1 libgs9 libgs9-common
  libidn12 libijs-0.35 libjbig2dec0 libkpathsea6 libpdfbox-java libptexenc1
  libruby3.0 libsyntax2 libteckit0 libtexlua53 libtexluajit2 libwoff1
  libzip-0-13 lmodern poppler-data preview-latex-style rake ruby
  ruby-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0
  rubygems-integration t1utils teckit tex-common tex-gyre texlive-base
  texlive-binaries texlive-latex-base texlive-latex-extra
  texlive-latex-recommended texlive-pictures tipa xfonts-encodings
  xfonts-utils
Suggested packages:
  fonts-oto fonts-freefont-otf | fonts-freefont-ttf libavalon-framework-java
  libcommons-logging-java-doc libxcalibur-logkit-java liblog4j1.2-java
  poppler-utils ghostscript fonts-japanese-mincho | fonts-ipafont-mincho
  fonts-japanese-gothic | fonts-ipafont-gothic fonts-arphic-ukai
  fonts-arphic-uming fonts-nanum ri ruby-dev bundler debhelper gv
  | postscript-viewer perl-tk xpdf | pdf-viewer xzdec
  texlive-fonts-recommended-doc texlive-latex-base-doc python3-pygments
  icc-profiles libfile-which-perl libspreadsheet-parseexcel-perl
  texlive-latex-extra-doc texlive-latex-recommended-doc texlive-luatex
  texlive-pstricks dot2tex prerex texlive-pictures-doc vprerex
  default-jre-headless tipa-doc
The following NEW packages will be installed:
  dvipng fonts-droid-fallback fonts-lato fonts-lmodern fonts-oto-mono
  fonts-texgyre fonts-urw-base35 libapache-pom-java libcommons-logging-java
  libcommons-parent-java libfontbox-java libfontenc1 libgs9 libgs9-common
  libidn12 libijs-0.35 libjbig2dec0 libkpathsea6 libpdfbox-java libptexenc1
  libruby3.0 libsyntax2 libteckit0 libtexlua53 libtexluajit2 libwoff1
  libzip-0-13 lmodern poppler-data preview-latex-style rake ruby
  ruby-net-telnet ruby-rubygems ruby-webrick ruby-xmlrpc ruby3.0
  rubygems-integration t1utils teckit tex-common tex-gyre texlive-base
  texlive-binaries texlive-fonts-recommended texlive-latex-base
  texlive-latex-extra texlive-latex-recommended texlive-pictures
  texlive-plain-generic texlive-xetex tipa xfonts-encodings xfonts-utils
0 upgraded, 54 newly installed, 0 to remove and 49 not upgraded.
Need to get 182 MB of archives.
After this operation, 571 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-droid-fallback all 1:6.0.1r16-1.1build1 [1,805 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-lato all 2.0-2.1 [2,696 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/main amd64 poppler-data all 0.4.11-1 [2,171 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tex-common all 6.17 [33.7 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-urw-base35 all 20200910-1 [6,367 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libgs9-common all 9.55.0~dfsg1-0ubuntu5.9 [752 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libidn12 amd64 1.38-4ubuntu1 [60.0 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy/main amd64 libijs-0.35 amd64 0.35-15build2 [16.5 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/main amd64 libjbig2dec0 amd64 0.19-3build2 [64.7 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libgs9 amd64 9.55.0~dfsg1-0ubuntu5.9 [5,033 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libkpathsea6 amd64 2021.20210626.59705-1ubuntu0.2 [60.4 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/main amd64 libwoff1 amd64 1.0.2-1build4 [45.2 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy/universe amd64 dvipng amd64 2.13.1-1 [1,221 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy/universe amd64 fonts-lmodern all 2.004.5-6.1 [4,532 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-oto-mono all 20201225-1build1 [397 kB]
```

Name: Afrida Islam

Student ID: 21221030

✓ EEE385IL (MACHINE LEARNING LABORATORY)

LAB 2: Introduction to Python

List in Python

Lists are used to store multiple items in a single variable.

```
list_name = [val_1, val_2, val_3, .....]
```

✓ Common List operations

```
marks = [88, 56, 94, 82, 94]
print("Before changing any element      --> ", marks)
```

```
marks.append(99)
print("After appending one element      --> ", marks)
```

```
marks.remove(94)
print("After removing one element      --> ", marks)
```

```
marks.insert(1, 71)
print("After inserting '71' at index '1' --> ", marks)
```

```
marks.pop(2)
print("After popping out the element at index '2' --> ", marks)
```

```
➦ Before changing any element      --> [88, 56, 94, 82, 94]
  After appending one element      --> [88, 56, 94, 82, 94, 99]
  After removing one element      --> [88, 56, 82, 94, 99]
  After inserting '71' at index '1' --> [88, 71, 56, 82, 94, 99]
  After popping out the element at index '2' --> [88, 71, 82, 94, 99]
```

```
my_list = [10, 'a', "Hello", 20.56, 5.78, 'b']
print(my_list)
print(my_list[2:5:1])
```

```
➦ [10, 'a', 'Hello', 20.56, 5.78, 'b']
  ['Hello', 20.56, 5.78]
```

```
my_list = [[10,20,30], 40, [50,60]]
print(my_list[2][1])
```

```
➦ 60
```

```
my_list = [[[10,20.5,'a']]]
print(my_list[0][0][2])
```

```
➦ a
```

✓ TASK (Practice)

Create a list containing the names of the cars:

- Toyota Corolla
- Mazda RX-8
- Lamborghini Gallardo

Then insert a new car 'Mercedes Benz' at the 2nd position of the list.

```
cars = ['Toyota Corolla', 'Mazda RX-8', 'Lamborghini Gallardo']
print(cars)
```

```

for car in cars:
    print(car)

cars.insert(1, 'Mercedes Benz')
print(cars)

type(cars)

➦ ['Toyota Corolla', 'Mazda RX-8', 'Lamborghini Gallardo']
Toyota Corolla
Mazda RX-8
Lamborghini Gallardo
['Toyota Corolla', 'Mercedes Benz', 'Mazda RX-8', 'Lamborghini Gallardo']
list

```

▼ Tuples

Tuples in Python contain multiple values that are ordered, indexed but **unchangable**.

```

cars = ('Toyota Corolla', 'Mazda RX-8', 'Lamborghini Gallardo')
print(cars[0])

marks = (86, 96, 78, 56)
print(marks[0])

mixed_tup = ('Toyota Corolla', 160)
print(mixed_tup)

type(cars)

➦ Toyota Corolla
86
('Toyota Corolla', 160)
tuple

print(type(cars))
cars_l = list(cars)
print(type(cars_l))
cars_l.append('Tesla')
print(cars_l)

➦ <class 'tuple'>
<class 'list'>
['Toyota Corolla', 'Mazda RX-8', 'Lamborghini Gallardo', 'Tesla']

```

▼ Set

Sets in python can contain multiple values and are **unordered, unindexed** and **unchangable**.

```

cars = {'Toyota Corolla', 'Mazda RX-8', 0}
print(cars)
type(cars)

➦ {0, 'Toyota Corolla', 'Mazda RX-8'}
set

```

▼ Dictionary in Python

Python's dictionaries allow you to connect pieces of related information. Each piece of information in a dictionary is stored as a **key-value pair**. When you provide a **key**, Python returns the **value** associated with that key. You can loop through all the key-value pairs, all the keys, or all the values.

```

cars = {'brand': ['Toyota', 'Mazda', 'Lamborghini'],
        'model': ['Corolla', 'RX-8', 'Gallardo'],
        'year': [1976, 1984, 1998]}

print(cars.items())
print(cars.keys())
print(cars.values())

# for key in cars.keys():
#     print(key)

# m = cars['model']
# print(m)

```

```
#for y in ys:
    #print(y)

cars['speed'] = [150, 180, 220]
print(cars.items())
cars['Mileage'] = [10,6,3]
print(cars.items())

# del cars['year']
# print(cars.items())

dict_items([('brand', ['Toyota', 'Mazda', 'Lamborghini']), ('model', ['Corolla', 'RX-8', 'Gallardo']), ('year', [1976, 1984, 1998])])
dict_keys(['brand', 'model', 'year'])
dict_values(['Toyota', 'Mazda', 'Lamborghini'], ['Corolla', 'RX-8', 'Gallardo'], [1976, 1984, 1998])
dict_items([('brand', ['Toyota', 'Mazda', 'Lamborghini']), ('model', ['Corolla', 'RX-8', 'Gallardo']), ('year', [1976, 1984, 1998])]),
dict_items([('brand', ['Toyota', 'Mazda', 'Lamborghini']), ('model', ['Corolla', 'RX-8', 'Gallardo']), ('year', [1976, 1984, 1998])])
```

✓ TASK-1

Create a dictionary with the "key"s as "ID", "Name", "credits" and input the corresponding information of three students.
Append another "key" named "cgpa" after that.

```
#####
##### code starts here #####
keys = {'ID': [21221030,21221031,21221032],
        'Name': ['A', 'B', 'C'],
        'Credits': [111,30, 78]}
keys['cgpa'] = [3,2.6,3.8]
print(keys.items())

##### code ends here #####
#####

dict_items([('ID', [21221030, 21221031, 21221032]), ('Name', ['A', 'B', 'C']), ('Credits', [111, 30, 78]), ('cgpa', [3, 2.6, 3.8])])
```

```
cars = {'brand': ['Toyota', 'Mazda', 'Lamborghini'],
        'model': ['Corolla', 'RX-8', 'Gallardo'],
        'year': [1976, 1984, 1998]}
```

```
for k, v in cars.items():
    print(k)
    for val in v:
        print(val)
```

```
brand
Toyota
Mazda
Lamborghini
model
Corolla
RX-8
Gallardo
year
1976
1984
1998
```

✓ Function

```
def function_name(input_args):

    statements

    return output_var
```

```
def easy_calc_add(n1, n2):
    return n1 + n2
```

```
a = 'Helloooo'
b = ' Afrida'
result = easy_calc_add(a, b)
print(result)
```

➞ Helloooo Afrida

```
def welcome_msg(names):
    for name in names:
        print("Welcome {}".format(name))
```

```
people = ['ABC', 'XYZ', 'PQR']
welcome_msg(people)
```

➞ Welcome ABC!!
Welcome XYZ!!
Welcome PQR!!

```
import math
def easy_calc_log(n1, eps = 0.0000001):
    if n1 < 0:
        print("MATH ERROR! Log is not defined for negative numbers!")
        return

    return math.log10(n1 + eps)
```

```
number = 10
print(easy_calc_log(number, 0))
```

➞ 1.0

```
import math
def calc_log10(n):
    if n > 0:
        return math.log10(n)
    else:
        print("UNDEFINED!!")
```

```
print(calc_log10(0))
```

➞ UNDEFINED!!
None

```
def std_info(ID, name = 'Bracuian'):
    print("Welcome! {}: {}".format(ID, name))
```

```
id = 28321087
std_info(id, 'ABC')
```

➞ Welcome! 28321087: ABC!

✓ TASK-2

Write a function in Python to determine whether a number is Armstrong or not.

```
#####
##### code starts here #####
import math

def isarmstrong(n):
    m = n
    sum = 0

    while 1:
        dig = n % 10
        n = int(n / 10)
        sum = sum + math.pow(dig, 3)

        if n == 0:
            break

    if sum == m:
        print("ARMSTRONG number!!")
    else:
        print("NOT a Armstrong number!")

##### code ends here #####
```

#####

isarmstrong(153)

➦ ARMSTRONG number!!

```
class Car():
    """ Description of the Car class """

    def __init__(self, brand, model, year):
        """Initializaing the attributes if a car object"""
        self.brand = brand
        self.model = model
        self.year = year

    def show_info(self):
        print('Brand: {}\nModel: {}\nYear: {}'.format(self.brand, self.model, self.year))

    def update_model(self, new_model):
        self.model = new_model

our_car = Car('Toyota', 'Corolla', 1978)
our_car.show_info()
our_car.update_model('Premio')
our_car.show_info()

our_2nd_car = Car('Mazda', 'RX-8', 1984)
```

➦ Brand: Toyota
Model: Corolla
Year: 1978

Brand: Toyota
Model: Premio
Year: 1978

```
class Car():
    """ Description of the Car class """

    def __init__(self, brand, model, year):
        """Initializaing the attributes if a car object"""
        self.brand = brand
        self.model = model
        self.year = year

    def show_info(self):
        print('Brand: {}\nModel: {}\nYear: {}'.format(self.brand, self.model, self.year))

    def update_model(self, new_model):
        self.model = new_model

first_car = Car('Toyota', 'Corolla', 1978)
first_car.show_info()
print(first_car.brand)

second_car = Car('Mazda', 'RX-8', 1996)
second_car.show_info()
second_car.update_model('RX-7')
second_car.show_info()
```

➦ Brand: Toyota
Model: Corolla
Year: 1978

Toyota
Brand: Mazda
Model: RX-8
Year: 1996

Brand: Mazda
Model: RX-7
Year: 1996

```
# Install the package for Tex and then convert to PDF directly as LaTeX
!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic
```

```
# Provide the file path of the notebook file
```

```
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/Afrida Islam_Lab02_21221030.ipynb"
```



Show hidden output

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



Show hidden output

EEE385L: Machine Learning Laboratory

Lab 3: Implementation of Multiple Linear Regression

Name: Afrida Islam

Student ID: 21221030

Import packages

```
import pandas as pd
#MinMaxScaler is for Normalization & StandardScaler is for Standardization
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import numpy as np
import matplotlib.pyplot as plt
```

Load Dataset from Google Drive

```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Wireless_data.csv")
```

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

 Mounted at /content/drive

Explore the Dataset

```
data.head()
```

	Dataset	Bandwidth	TransmissionPower	ChannelGain	Frequency	UnitCost
0	8.517789	8.221591	6.537314	4.294531	1.631189	27.014253
1	9.246335	6.893047	4.565376	13.763571	1.355074	21.211191
2	3.015895	9.057902	7.591142	4.000898	1.997003	28.592177
3	9.307007	7.664128	6.825546	14.521624	1.224171	23.933344
4	7.058874	6.753636	8.175696	14.266931	1.652451	21.655819

Next steps:

[Generate code with data](#)

☒ [View recommended plots](#)

[New interactive sheet](#)

```
data.iloc[5:7,0:3]
```

	Dataset	Bandwidth	TransmissionPower
5	2.780323	9.695008	8.199327
6	4.227986	9.379714	7.831185

```
data["TransmissionPower"]
```

	TransmissionPower
0	6.537314
1	4.565376
2	7.591142
3	6.825546
4	8.175696
...	...
195	6.971062
196	5.138262
197	6.970035
198	4.885649
199	4.329845

200 rows × 1 columns

```
cols = data.columns
print(cols[1])
```

Bandwidth

```
data[cols[-1]][6:10]
```

	UnitCost
6	27.580278
7	29.778232
8	10.010448
9	27.308772

Perform Feature Engineering

```
m = 2 # number of features
X = data.iloc[:, 0:m]
print(X)
```

	Dataset	Bandwidth
0	8.517789	8.221591
1	9.246335	6.893047
2	3.015895	9.057902
3	9.307007	7.664128
4	7.058874	6.753636
...
195	4.450796	5.112563
196	6.068069	7.126297
197	6.086173	6.563594
198	8.541022	5.807424
199	8.358651	5.893831

[200 rows x 2 columns]

```
y = data.iloc[:, -1]
print(y)
```

	UnitCost
0	27.014253
1	21.211191
2	28.592177
3	23.933344
4	21.655819
...	...
195	11.026638
196	11.457706
197	11.770549
198	25.967017
199	28.860163

Name: UnitCost, Length: 200, dtype: float64

```

scaler = StandardScaler()
X_standard = scaler.fit_transform(X)
print(X_standard)

```

 Show hidden output

✓ Define the loss function

Modify the cost function to incorporate the bias term (w_0)

```

def mse(w0, w1, w2, w3, w4, w5, X1, X2, X3, X4, X5, y):
    y_hat = w0 + w1 * X1 + w2 * X2 + w3 * X3 + w4 * X4 + w5 * X5
    error_sq = (y - y_hat) ** 2
    mse_value = (1 / len(y)) * np.sum(error_sq)
    return mse_value

```

✓ Implement Gradient Descent to update model parameters

```

def gradient_descent(w0, w1, w2, w3, w4, w5, X1, X2, X3, X4, X5, y, alpha, epoch):
    train_loss = [0] * epoch

```

```

    for i in range(epoch):
        # Predictions
        y_hat = w0 + w1 * X1 + w2 * X2 + w3 * X3 + w4 * X4 + w5 * X5
        err = y - y_hat

```

```

        # Compute gradients
        w0_gradient = (-2 / len(y)) * np.sum(err)
        w1_gradient = (-2 / len(y)) * np.dot(X1, err)
        w2_gradient = (-2 / len(y)) * np.dot(X2, err)
        w3_gradient = (-2 / len(y)) * np.dot(X3, err)
        w4_gradient = (-2 / len(y)) * np.dot(X4, err)
        w5_gradient = (-2 / len(y)) * np.dot(X5, err)

```

```

        # Update weights and bias
        w0 -= alpha * w0_gradient
        w1 -= alpha * w1_gradient
        w2 -= alpha * w2_gradient
        w3 -= alpha * w3_gradient
        w4 -= alpha * w4_gradient
        w5 -= alpha * w5_gradient

```

```

        # Calculate training loss
        train_loss[i] = mse(w0, w1, w2, w3, w4, w5, X1, X2, X3, X4, X5, y)
        print(f"Epoch {i+1}: Training Loss - {train_loss[i]}")

```

```

    return w0, w1, w2, w3, w4, w5, train_loss

```

✓ Prepare the Dataset for training

```

n = int(0.8 * len(y)) # Number of training samples

```

```

X_train = X_standard[:n, :] # Training features
X_test = X_standard[n:, :] # Testing features

```

```

y_train = y[:n] # Training targets
y_test = y[n:] # Testing targets

```

```

# Separate the individual features for gradient descent

```

```

X1_train = X_train[:, 0]
X2_train = X_train[:, 1]
X3_train = X_train[:, 2]
X4_train = X_train[:, 3]
X5_train = X_train[:, 4]

```

```

X1_test = X_test[:, 0]
X2_test = X_test[:, 1]
X3_test = X_test[:, 2]
X4_test = X_test[:, 3]
X5_test = X_test[:, 4]

```

```
y_train = y[0:n]
print(y_train)
y_test = y[n: ]
print(y_test)
```

Show hidden output

✓ Train the model

```
w0 = 0 # Bias term
w1, w2, w3, w4, w5 = 0, 0, 0, 0, 0
alpha = 0.001
epoch = 1000
```

```
# Run gradient descent to fit the model
```

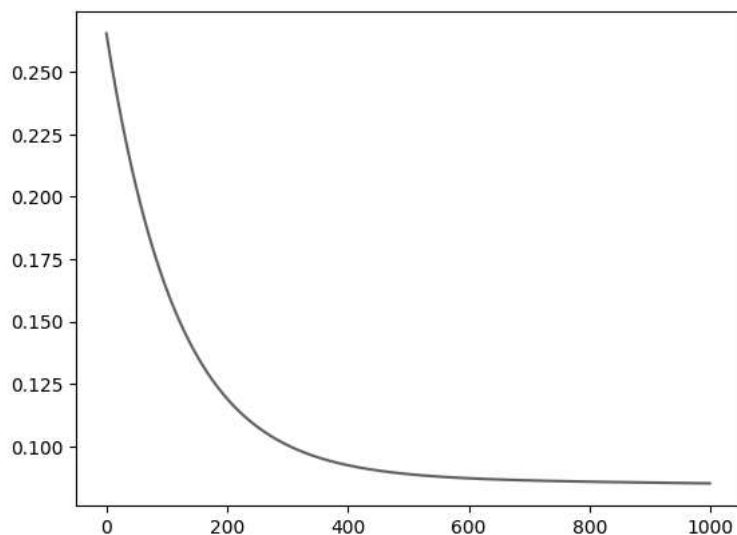
```
updated_w0, updated_w1, updated_w2, updated_w3, updated_w4, updated_w5, loss = gradient_descent(w0, w1, w2, w3, w4, w5, X1_train, X2_train, y_train)
```

Show hidden output

✓ Plot the loss in each epoch

```
plt.plot(loss)
```

[<matplotlib.lines.Line2D at 0x7daa4657b9d0>]



```
# Install the package for Tex and then convert to PDF directly as LaTeX
!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic pandoc
```

```
# Provide the file path of the notebook file
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/21221030_Lab_3.ipynb"
```

Show hidden output

21221030_Lab_4

November 21, 2024

1 EEE385L: Machine Learning Laboratory

2 Lab 4: Implementation of Logistic Regression

3 Name: Afrida Islam

4 Student ID: 21221030

[]:

#Import packages

```
[1]: import pandas as pd
      #MinMaxScaler is for Normalization & StandardScaler is for Standardization
      from sklearn.preprocessing import MinMaxScaler, StandardScaler
      import numpy as np
      import matplotlib.pyplot as plt
```

```
[2]: from google.colab import drive
      drive.mount('/content/drive')
```

Mounted at /content/drive

#Load Dataset from Google Drive

```
[ ]: data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/
      ↪house_rent_data_log_reg.csv")
```

#Explore the Dataset

```
[ ]: data.head()
```

```
[ ]: 
```

	Size (sq. ft.)	Distance (miles)	House Rent (\$)	Possibilty to purchase
0	498	11.9	675	0
1	513	8.6	750	0
2	621	8.3	800	1
3	710	3.4	965	0
4	650	7.5	855	1

```
[ ]: data.iloc[3:6, 1:3]
```

```
[ ]:      Distance (miles)  House Rent ($)
3           3.4           965
4           7.5           855
5           5.1           790
```

```
[ ]: cols = data.columns
print(cols[1])
```

Distance (miles)

```
[ ]: data[cols[-1]][6:10]
```

```
[ ]: 6      1
7      1
8      0
9      0
Name: Possibilty to purchase, dtype: int64
```

#Perform Feature Engineering

```
[ ]: m = 3      # number of features
X = data.iloc[:, 0:m]
print(X)
```

	Size (sq. ft.)	Distance (miles)	House Rent (\$)
0	498	11.9	675
1	513	8.6	750
2	621	8.3	800
3	710	3.4	965
4	650	7.5	855
5	620	5.1	790
6	560	4.6	740
7	780	8.1	810
8	450	5.4	710
9	790	4.5	940

```
[ ]: y = data.iloc[:, -1]
print(y)
```

```
0      0
1      0
2      1
3      0
4      1
5      1
6      1
```

```
7    1
8    0
9    0
Name: Possibility to purchase, dtype: int64
```

```
[ ]: scaler = StandardScaler()
X_standard = scaler.fit_transform(X)
print(X_standard)
```

```
[[-1.09582295  2.10333853 -1.4389012 ]
 [-0.9602013   0.75818017 -0.5990756 ]
 [ 0.0162746   0.63589304 -0.03919186]
 [ 0.82096307 -1.36146331  1.80842447]
 [ 0.27847646  0.30979405  0.57668025]
 [ 0.00723315 -0.66850294 -0.15116861]
 [-0.53525345 -0.87231482 -0.71105234]
 [ 1.45386411  0.55436829  0.07278489]
 [-1.52981224 -0.54621582 -1.04698259]
 [ 1.54427855 -0.91307719  1.5284826  ]]
```

#Define the loss function

```
[ ]: # sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
[ ]: sigmoid(0)
```

```
[ ]: 0.5
```

```
[ ]: # Binary Cross Entropy
def bce(w0,w, X, y):
    y_hat_lin = np.add(np.dot(X, w), w0)
    y_hat = sigmoid(y_hat_lin) #logistic
    y_hat_log = np.log(y_hat) #logarithm

    loss = -(np.dot(y_hat_log, y)+ np.dot((1-y_hat_log), (1-y)))
    return loss
```

#Implement Gradient Descent to update model parameters

```
[ ]: # Gradient descent
def gradient_descent(w0, w, X, y, alpha, epoch):
    train_loss = [0]* epoch

    for i in range(epoch):
        y_hat_lin = np.add(np.dot(X, w), w0)
        y_hat = sigmoid(y_hat_lin)
```

```

err = y-y_hat

gradient = -np.dot(X.T, err)

w0 = w0 - alpha * (-np.sum(err))
w = w - alpha * gradient

train_loss[i] = bce(w0, w, X, y)
print(f"Epoch - {i+1}: Training loss - {train_loss[i]}")

return w0, w, train_loss

```

#Prepare the Dataset for training

```

[ ]: n = int(len(y) * 0.8)    # number of training samples
X_train = X_standard[0:n, 0:m]
print(X_train)
X_test = X_standard[n: , 0:m]
print(X_test)

```

```

[[-1.09582295  2.10333853 -1.4389012 ]
 [-0.9602013   0.75818017 -0.5990756 ]
 [ 0.0162746   0.63589304 -0.03919186]
 [ 0.82096307 -1.36146331  1.80842447]
 [ 0.27847646  0.30979405  0.57668025]
 [ 0.00723315 -0.66850294 -0.15116861]
 [-0.53525345 -0.87231482 -0.71105234]
 [ 1.45386411  0.55436829  0.07278489]]
[[-1.52981224 -0.54621582 -1.04698259]
 [ 1.54427855 -0.91307719  1.5284826 ]]

```

```

[ ]: y_train = y[0:n]
print(y_train)
y_test = y[n: ]
print(y_test)

```

```

0    0
1    0
2    1
3    0
4    1
5    1
6    1
7    1
Name: Possibilty to purchase, dtype: int64
8    0
9    0
Name: Possibilty to purchase, dtype: int64

```

#Train the model

```
[ ]: w0 = 0
      w = [0] * m

      alpha = 0.001
      epoch = 10000

      updated_w0, updated_w, loss = gradient_descent(w0, w, X_train, y_train, alpha,
      ↪epoch)
```

Streaming output truncated to the last 5000 lines.

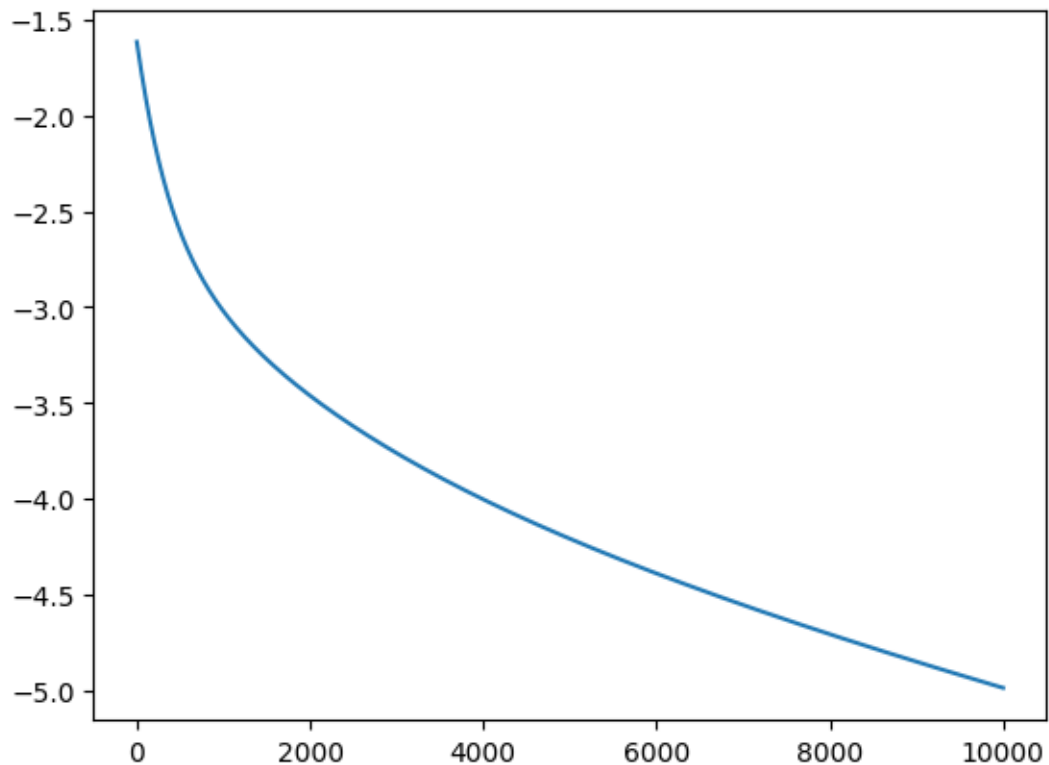
```
Epoch - 5001: Training loss - -4.207433985896637
Epoch - 5002: Training loss - -4.2076255795057085
Epoch - 5003: Training loss - -4.207817149344784
Epoch - 5004: Training loss - -4.208008695422901
Epoch - 5005: Training loss - -4.208200217749095
Epoch - 5006: Training loss - -4.208391716332393
Epoch - 5007: Training loss - -4.208583191181822
Epoch - 5008: Training loss - -4.208774642306403
Epoch - 5009: Training loss - -4.208966069715152
Epoch - 5010: Training loss - -4.209157473417083
Epoch - 5011: Training loss - -4.209348853421204
Epoch - 5012: Training loss - -4.209540209736521
Epoch - 5013: Training loss - -4.209731542372032
Epoch - 5014: Training loss - -4.209922851336735
Epoch - 5015: Training loss - -4.210114136639624
Epoch - 5016: Training loss - -4.210305398289685
Epoch - 5017: Training loss - -4.2104966362958995
Epoch - 5018: Training loss - -4.210687850667252
Epoch - 5019: Training loss - -4.210879041412717
Epoch - 5020: Training loss - -4.211070208541262
Epoch - 5021: Training loss - -4.211261352061859
Epoch - 5022: Training loss - -4.2114524719834705
Epoch - 5023: Training loss - -4.211643568315054
Epoch - 5024: Training loss - -4.211834641065566
Epoch - 5025: Training loss - -4.212025690243959
Epoch - 5026: Training loss - -4.212216715859174
Epoch - 5027: Training loss - -4.212407717920161
Epoch - 5028: Training loss - -4.212598696435854
Epoch - 5029: Training loss - -4.212789651415188
Epoch - 5030: Training loss - -4.212980582867096
Epoch - 5031: Training loss - -4.2131714908005
Epoch - 5032: Training loss - -4.213362375224326
Epoch - 5033: Training loss - -4.21355323614749
Epoch - 5034: Training loss - -4.2137440735789085
Epoch - 5035: Training loss - -4.213934887527488
Epoch - 5036: Training loss - -4.214125678002136
```

```
Epoch - 9981: Training loss - -4.982084072564209
Epoch - 9982: Training loss - -4.982217465566897
Epoch - 9983: Training loss - -4.98235085290434
Epoch - 9984: Training loss - -4.982484234577624
Epoch - 9985: Training loss - -4.982617610587832
Epoch - 9986: Training loss - -4.98275098093605
Epoch - 9987: Training loss - -4.9828843456233605
Epoch - 9988: Training loss - -4.9830177046508455
Epoch - 9989: Training loss - -4.983151058019589
Epoch - 9990: Training loss - -4.983284405730674
Epoch - 9991: Training loss - -4.983417747785181
Epoch - 9992: Training loss - -4.983551084184194
Epoch - 9993: Training loss - -4.983684414928794
Epoch - 9994: Training loss - -4.983817740020058
Epoch - 9995: Training loss - -4.98395105945907
Epoch - 9996: Training loss - -4.984084373246908
Epoch - 9997: Training loss - -4.984217681384656
Epoch - 9998: Training loss - -4.984350983873387
Epoch - 9999: Training loss - -4.9844842807141845
Epoch - 10000: Training loss - -4.984617571908126
```

```
#Plot the loss in each epoch
```

```
[ ]: plt.plot(loss)
```

```
[ ]: [<matplotlib.lines.Line2D at 0x7aaa4a3e3b80>]
```



```
[ ]: print(updated_w0)
      print(updated_w)
```

```
0.9830524444417982
[ 3.48249062 -0.85393737 -2.75766437]
```

```
[3]: data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Rice_Cammeo_Osman.
      ↪csv")
```

```
[4]: data.head()
```

```
[4]:
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	\
0	15231	525.578979	229.749878	85.093788	0.928882	
1	14656	494.311005	206.020065	91.730972	0.895405	
2	14634	501.122009	214.106781	87.768288	0.912118	
3	13176	458.342987	193.337387	87.448395	0.891861	
4	14688	507.166992	211.743378	89.312454	0.906691	

	Convex_Area	Extent	Class
0	15617	0.572896	1
1	15072	0.615436	1
2	14954	0.693259	1

3	13368	0.640669	1
4	15262	0.646024	1

```
[5]: m = 7    # number of features
      X = data.iloc[:, 0:m]
      print(X)
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	\
0	15231	525.578979	229.749878	85.093788	
1	14656	494.311005	206.020065	91.730972	
2	14634	501.122009	214.106781	87.768288	
3	13176	458.342987	193.337387	87.448395	
4	14688	507.166992	211.743378	89.312454	
...	
3805	11441	415.858002	170.486771	85.756592	
3806	11625	421.390015	167.714798	89.462570	
3807	12437	442.498993	183.572922	86.801979	
3808	9882	392.296997	161.193985	78.210480	
3809	11434	404.709992	161.079269	90.868195	

	Eccentricity	Convex_Area	Extent
0	0.928882	15617	0.572896
1	0.895405	15072	0.615436
2	0.912118	14954	0.693259
3	0.891861	13368	0.640669
4	0.906691	15262	0.646024
...
3805	0.864280	11628	0.681012
3806	0.845850	11904	0.694279
3807	0.881144	12645	0.626739
3808	0.874406	10097	0.659064
3809	0.825692	11591	0.802949

[3810 rows x 7 columns]

```
[6]: y = data.iloc[:, -1]
      print(y)
```

0	1
1	1
2	1
3	1
4	1
...	
3805	0
3806	0
3807	0
3808	0

```
3809    0
Name:    Class, Length: 3810, dtype: int64
```

```
[7]: scaler = StandardScaler()
X_standard = scaler.fit_transform(X)
print(X_standard)
```

```
[[ 1.47982953  2.0043543  2.34854657 ...  2.01833746  1.49965944
 -1.15292093]
 [ 1.14787029  1.12585309  0.98839042 ...  0.41001813  1.19291767
 -0.60207876]
 [ 1.13516924  1.31721425  1.45190846 ...  1.21295648  1.12650386
  0.405611   ]
 ...
 [-0.13320373 -0.32985087 -0.29824512 ... -0.27509915 -0.17306812
 -0.45573108]
 [-1.60825742 -1.74032002 -1.58097116 ... -0.59882135 -1.60715621
 -0.03716757]
 [-0.71225612 -1.39156604 -1.58754648 ... -2.93916012 -0.76628981
  1.82594693]]
```

```
[8]: # sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
[9]: # Binary Cross Entropy
def bce(w0,w, X, y):
    y_hat_lin = np.add(np.dot(X, w), w0)
    y_hat = sigmoid(y_hat_lin) #logistic
    y_hat_log = np.log(y_hat) #logarithm

    loss = -(np.dot(y_hat_log, y)+ np.dot((1-y_hat_log), (1-y)))
    return loss
```

```
[10]: # Gradient descent
def gradient_descent(w0, w, X, y, alpha, epoch):
    train_loss = [0]* epoch

    for i in range(epoch):
        y_hat_lin = np.add(np.dot(X, w), w0)
        y_hat = sigmoid(y_hat_lin)
        err = y-y_hat

        gradient = -np.dot(X.T, err)

        w0 = w0 - alpha * (-np.sum(err))
        w = w - alpha * gradient
```

```

train_loss[i] = bce(w0, w, X, y)
print(f"Epoch - {i+1}: Training loss - {train_loss[i]}")

return w0, w, train_loss

```

```

[11]: n = int(len(y) * 0.8)    # number of training samples
X_train = X_standard[0:n, 0:m]
print(X_train)
X_test = X_standard[n: , 0:m]
print(X_test)

```

```

[[ 1.47982953  2.0043543  2.34854657 ...  2.01833746  1.49965944
 -1.15292093]
 [ 1.14787029  1.12585309  0.98839042 ...  0.41001813  1.19291767
 -0.60207876]
 [ 1.13516924  1.31721425  1.45190846 ...  1.21295648  1.12650386
  0.405611  ]
 ...
 [-1.49221601 -1.44152009 -1.13200576 ...  0.16304503 -1.50247004
 -0.2243131  ]
 [ 0.06481717 -0.48353536 -0.78293572 ... -2.0139966  -0.00703359
  0.57462398]
 [ 0.07751822  0.09613933  0.18839148 ...  0.40691406  0.02729898
  1.22256639]]
[[-1.20009188 -0.40826605  0.16786614 ...  1.97639208 -1.20191939
 -1.95498146]
 [-0.67299833 -0.68826966 -0.60255873 ... -0.13189842 -0.69649902
  1.44601405]
 [-0.14590478 -0.42827047 -0.4394798  ... -0.65208636 -0.20627503
 -0.78099929]
 ...
 [-0.13320373 -0.32985087 -0.29824512 ... -0.27509915 -0.17306812
 -0.45573108]
 [-1.60825742 -1.74032002 -1.58097116 ... -0.59882135 -1.60715621
 -0.03716757]
 [-0.71225612 -1.39156604 -1.58754648 ... -2.93916012 -0.76628981
  1.82594693]]

```

```

[12]: y_train = y[0:n]
print(y_train)
y_test = y[n: ]
print(y_test)

```

```

0      1
1      1
2      1

```

```

3      1
4      1
..
3043   0
3044   0
3045   0
3046   0
3047   0
Name: Class, Length: 3048, dtype: int64
3048   0
3049   0
3050   0
3051   0
3052   0
..
3805   0
3806   0
3807   0
3808   0
3809   0
Name: Class, Length: 762, dtype: int64

```

```

[26]: w0 = 0
      w = [0] * m

      alpha = 0.001
      epoch = 1000

      updated_w0, updated_w, loss = gradient_descent(w0, w, X_train, y_train, alpha,
      ↪epoch)

```

```

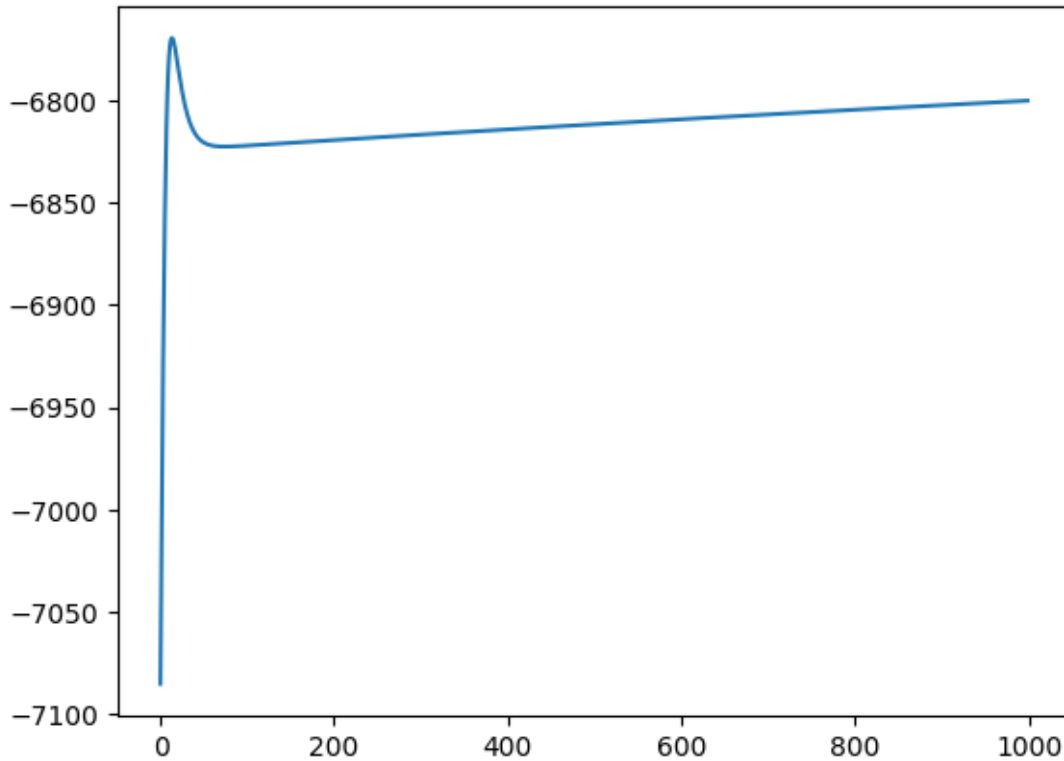
Epoch - 1: Training loss - -7085.123568728295
Epoch - 2: Training loss - -7029.884965773944
Epoch - 3: Training loss - -6977.566729160317
Epoch - 4: Training loss - -6931.047876789875
Epoch - 5: Training loss - -6891.387807400673
Epoch - 6: Training loss - -6858.680661065756
Epoch - 7: Training loss - -6832.5015860635285
Epoch - 8: Training loss - -6812.160166241046
Epoch - 9: Training loss - -6796.855427471239
Epoch - 10: Training loss - -6785.773189726914
Epoch - 11: Training loss - -6778.145283849048
Epoch - 12: Training loss - -6773.282106007351
Epoch - 13: Training loss - -6770.586667531899
Epoch - 14: Training loss - -6769.556431869829
Epoch - 15: Training loss - -6769.777750471592
Epoch - 16: Training loss - -6770.916396013069

```

```
Epoch - 977: Training loss - -6800.728911277302
Epoch - 978: Training loss - -6800.707120454835
Epoch - 979: Training loss - -6800.685334703136
Epoch - 980: Training loss - -6800.6635540191755
Epoch - 981: Training loss - -6800.641778399913
Epoch - 982: Training loss - -6800.620007842332
Epoch - 983: Training loss - -6800.598242343408
Epoch - 984: Training loss - -6800.576481900119
Epoch - 985: Training loss - -6800.554726509448
Epoch - 986: Training loss - -6800.532976168387
Epoch - 987: Training loss - -6800.511230873921
Epoch - 988: Training loss - -6800.489490623045
Epoch - 989: Training loss - -6800.467755412756
Epoch - 990: Training loss - -6800.446025240057
Epoch - 991: Training loss - -6800.424300101949
Epoch - 992: Training loss - -6800.402579995439
Epoch - 993: Training loss - -6800.38086491754
Epoch - 994: Training loss - -6800.35915486526
Epoch - 995: Training loss - -6800.337449835622
Epoch - 996: Training loss - -6800.3157498256405
Epoch - 997: Training loss - -6800.294054832341
Epoch - 998: Training loss - -6800.272364852757
Epoch - 999: Training loss - -6800.250679883908
Epoch - 1000: Training loss - -6800.22899992283
```

```
[27]: plt.plot(loss)
```

```
[27]: [<matplotlib.lines.Line2D at 0x7b79ca267a00>]
```



```
[28]: print(updated_w0)
      print(updated_w)
```

```
-0.3027361096625728
[ 0.44613794  1.006069    0.53811413 -0.49905008  1.32479183  2.41981724
 0.04347242]
```

```
[ ]: # Install the package for Tex and then convert to PDF directly as LaTeX
      !sudo apt-get install texlive-xetex texlive-fonts-recommended
      ↪texlive-plain-generic pandoc

      # Provide the file path of the notebook file
      !jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/
      ↪21221030_Lab_4.ipynb"
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  dvismgm fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono
  fonts-texgyre fonts-urw-base35 libapache-pom-java
  libcmark-gfm-extensions0.29.0.gfm.3 libcmark-gfm0.29.0.gfm.3
```