Bharati Vidyapeeth's College of Engineering
New Delhi

LAB MANUAL

Department	**Computer Science and Engineering**
Academic Year	**2024-25**
Semester	**7th**
Subject Name	**Machine Learning Lab**
Subject Code	**ML-407P**
Faculty Name	**Dr. Rakhi**

INSTITUTE VISION

To be an institute of excellence that provides quality technical education and research to create competent graduates for serving industry and society.

INSTITUTE MISSION

M1: To impart quality technical education through dynamic teaching-learning environment

M2: To promote research and innovations activities which gives opportunities for life-long learning in context of academic and industry.

M3: To build up links with industry-institute through partnerships and collaborative developmental works.

M4: To inculcate work ethics and commitment in graduates for their future endeavors to serve the society.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION OF DEPARTMENT

To develop as a centre of excellence in computer science and engineering education and research so as to produce globally competent professionals with a sense of social responsibility.

MISSION OF DEPARTMENT

Mission: The mission of the CSE Department is to:

M1: Impart technical knowledge in Computer Science and Engineering with the state-of-art infrastructure.
M2: Provide a conducive environment for the holistic development of graduates.
M3: Inculcate leadership qualities, teamwork and strong ethical values among the graduates. **M4**: Promote a research culture and industry-academia collaboration to strengthen innovation.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

Following are the three CSE Department Program Educational Objectives (PEOs):

PEO1: To produce graduates with in-depth knowledge of Computer Science and Engineering to contribute towards innovation, research and excellence in higher studies.
PEO2: To inculcate life-long learning skills in graduates enabling them to adapt to changing technologies, modern tools and work in teams.
PEO3: To produce ethically responsible graduates who are involved in transforming the society by providing suitable engineering solutions.

The undergraduate program of CSE is having the following Program Outcomes and Program Specific Outcomes (POs).

PROGRAM OUTCOMES (PO)

1. **Engineering knowledge**: Apply the knowledge acquired in mathematics, science, engineering for the solution of complex engineering problems.
2. **Problem analysis**: Identify research gaps, formulate and analyze complex engineering problems drawing substantiated conclusions using basic knowledge of mathematics, natural sciences and engineering sciences.
3. **Design/development of solutions**: Design solutions for the identified complex engineering problems as well as develop solutions that meet the specified needs for the public health and safety, and the cultural, societal and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods, including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Work on the latest technologies, resources and software tools including prediction and modelling to complex engineering activities with an understanding of their limitations.
6. **The engineer and society:** Apply the basic acquired knowledge to measure societal, health, safety, legal and cultural issues and identifying the consequential responsibilities relevant to the professional engineering practice.
7. **Environment and Sustainability:** Comprehend the impact of the professional engineering solutions in context of society and environment and demonstrate the need and knowledge for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to apply fundamentals of computational mathematics and algorithmic formulations to solve the real- time challenges of computer engineering encountered in research and industry.

PSO2: Capability to design and develop software and hardware applications using logical, analytical, and programming skills learnt while also following professional and social ethics.

TABLE OF CONTENTS

1. COURSE DETAILS

1.1 COURSE OBJECTIVES

- To understand the need of machine learning.
- To learn about regression and feature selection.
- To understand about classification algorithms.
- To learn clustering algorithms.

1.2 COURSE OUTCOMES

At the end of the course student will be able to:		Bloom Level
ML-407P.1	To formulate machine learning problems	Remember, Analyze,
ML-407P.2	Learn about regression and feature selection techniques and develop applications based on the same.	Remember, Understand, Apply, Evaluate, create
ML-407P.3	Apply machine learning techniques such as classification to practical applications.	Understand, Remember, Apply, Analyze
ML-407P.4	Apply Clustering algorithms to develop various practical applications.	Understand, Remember, Apply, Create

1.3 MAPPING COURSE OUTCOMES (CO) AND PROGRAM OUTCOMES (PO)/ PROGRAM SPECIFIC OUTCOME (PSO)

CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
CO1	**3**	**3**	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**2**	**2**	**2**
CO2	**3**	**3**	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**2**	**2**	**2**
CO3	**3**	**3**	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**2**	**2**	**2**
CO4	**3**	**3**	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**2**	**2**	**2**

1.4 EVALUATION SCHEME

	Laboratory	
Components	**Internal**	**External**
Marks	40	60
Total Marks	100	

1.5 GUIDELINES FOR CONTINUOUS ASSESSMENT FOR EACH EXPERIMENT

- **Attendance and performance in minimum eight experiments and Content Beyond syllabus – 20 marks**

 Each Experiment will carry a weight of 20 marks

  - Experiment performance [5 Marks]
  - File [10 Marks]
  - Attendance [5 Marks]

- **Internal Lab examination Viva-Voce of 20 marks.**

The Rubrics for Experiment execution and Lab file+ viva voce is given below:

Experiment Marks details:

Status	*Completed and Executed perfectly*	*Completed but partially Executing*	*Logically Incorrect Program or errors*	*Unacceptable efforts/Absent*
Marks	4-5	2-3	1	0

File Marks Details:

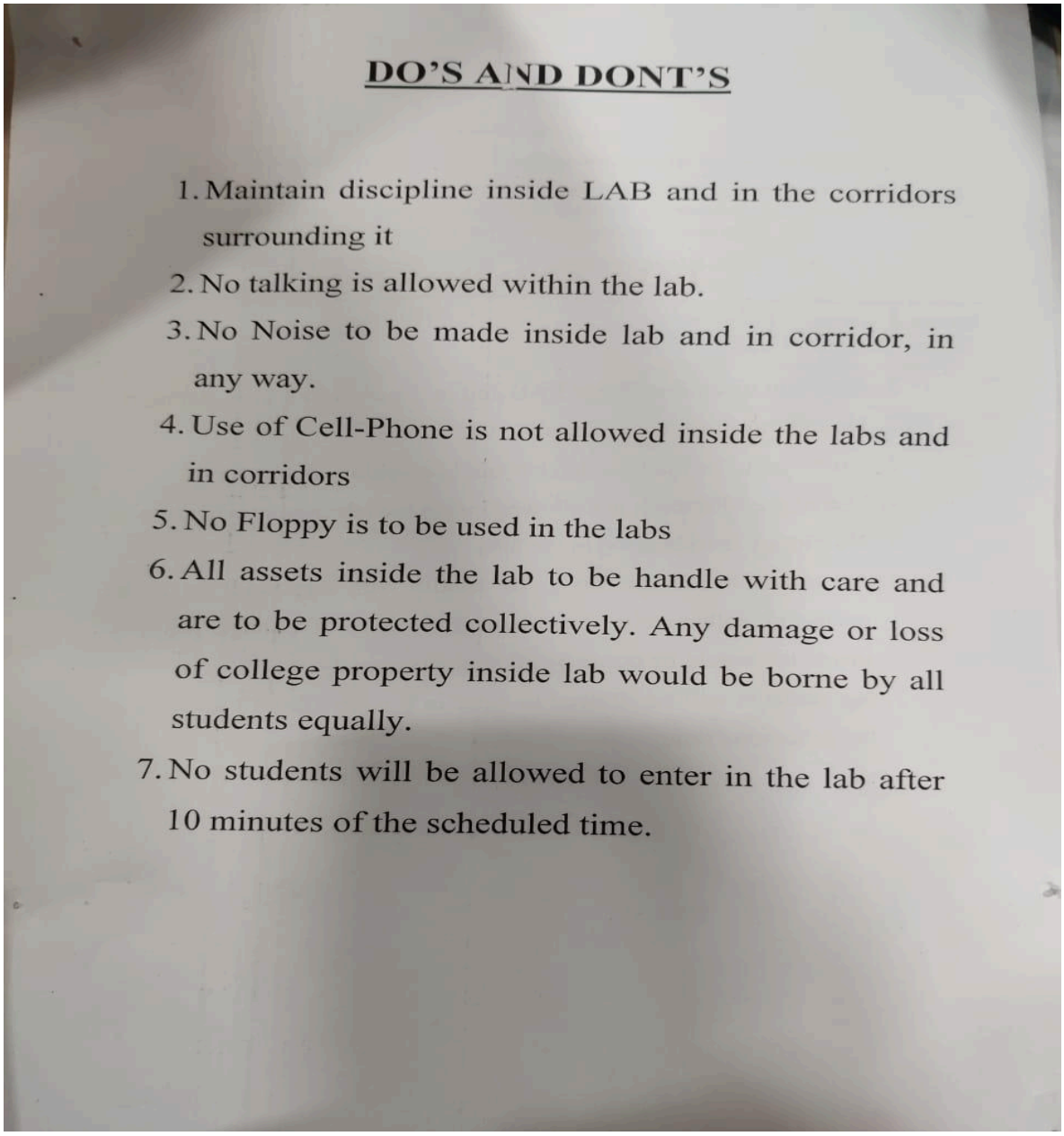
Status	*File Contents &Checked Timely*	*File Contents & Checked not Timely (after one week)*	*File Contents & Checked After two weeks*
Marks	9-10	7-8	0-2

Attendance/Viva-Voce Marks details:

Status	*Viva (Good)*	*Viva (Average)*	*Viva (Unsatisfactory)*
Marks	4-5	1-3	0

Note: Viva Voce Questions for each experiment should be related to Course Outcomes.

1.4 Safety Guidelines/Rules for laboratory



DO'S AND DONT'S

1. Maintain discipline inside LAB and in the corridors surrounding it
2. No talking is allowed within the lab.
3. No Noise to be made inside lab and in corridor, in any way.
4. Use of Cell-Phone is not allowed inside the labs and in corridors
5. No Floppy is to be used in the labs
6. All assets inside the lab to be handle with care and are to be protected collectively. Any damage or loss of college property inside lab would be borne by all students equally.
7. No students will be allowed to enter in the lab after 10 minutes of the scheduled time.

1.6 Format for students while writing Experiment in Lab file.

Experiment No:

Aim:

Course Outcome:

Software used:

Theory:

Flowchart/Algorithm/Code:

Results:

Expected Outcome attained: YES/NO

2. LIST OF EXPERIMENTS AS PER GGSIPU

Sr. No.	Title of Lab Experiments	CO
1	Introduction to JUPYTER IDE and its libraries Pandas and NumPy	CO1
2	Program to demonstrate Simple Linear Regression	CO2
3	Program to demonstrate Logistic Regression	CO2
4	Program to demonstrate Decision Tree – ID3 Algorithm	CO2
5	Program to demonstrate k-Nearest Neighbor flowers classification	CO3
6	Program to demonstrate Naïve- Bayes Classifier	CO3
7	Program to demonstrate PCA and LDA on Iris dataset	CO3
8	Program to demonstrate DBSCAN clustering algorithm	CO4

CONTENT BEYOND SYLLABUS

Sr. No	Name of Experiment	CO
1	Program to demonstrate K-Medoid clustering algorithm	CO4
2	Program to demonstrate K-Means Clustering Algorithm on Handwritten Dataset	CO4

3. EXPERIMENTAL SETUP DETAILS FOR THE COURSE

Software Requirements:

- Python
- Anaconda

Minimum Hardware Requirements

Dual Core based PC with 2 GB RAM

4. EXPERIMENTAL DETAILS

Experiment No.-1

Introduction to JUPYTER IDE and its libraries Pandas and NumPy

Jupyter Notebook

The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects. This article will walk you through how to use Jupyter Notebooks for data science projects and how to set it up on your local machine.

First, though: **what is a "notebook"?**

A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media. In other words: it's a single document where you can run code, display the output, and also add explanations, formulas, charts, and make your work more transparent, understandable, repeatable, and shareable.

Using Notebooks is now a major part of the data science workflow at companies across the globe. If your goal is to work with data, using a Notebook will speed up your workflow and make it easier to communicate and share your results.

Best of all, as part of the open source Project Jupyter, Jupyter Notebooks are completely free. You can download the software on its own, or as part of the Anaconda data science toolkit.

Although it is possible to use many different programming languages in Jupyter Notebooks, but we will use Python for this lab.

Python and pandas specifically. That said, if you have experience with another language, the Python in this article shouldn't be too cryptic, and will still help you get Jupyter Notebooks set up locally. Jupyter Notebooks can also act as a flexible platform for getting to grips with pandas and Python. We will cover the basics of installing Jupyter and creating your first notebook along with pandas and numpy.

Installation

The easiest way for a beginner to get started with Jupyter Notebooks is by installing Anaconda. Anaconda is the most widely used Python distribution for data science and comes pre-loaded with all the most popular libraries and tools. Some of the biggest Python libraries included in Anaconda include NumPy, pandas, and Matplotlib, though the full 1000+ list is exhaustive. Anaconda thus lets us hit the ground running with a fully stocked data science workshop without the hassle of managing countless installations or worrying about dependencies and OS-specific (read: Windows-specific) installation issues.

To get Anaconda, simply:

1. Download the latest version of Anaconda for Python 3.8.
2. Install Anaconda by following the instructions on the download page and/or in the executable. If you are a more advanced user with Python already installed and prefer to manage your packages manually, you can just use pip:

   pip3 install jupyter

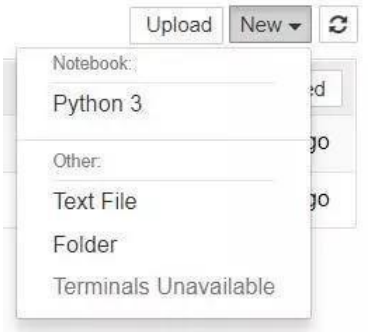
Creating Your First Notebook

In this section, we're going to learn to run and save notebooks, familiarize ourselves with their structure, and understand the interface. We'll become intimate with some core terminology that will steer you

towards a practical understanding of how to use Jupyter Notebooks by yourself and set us up for the next section, which walks through an example data analysis and brings everything we learn here to life.

Running Jupyter

On Windows, you can run Jupyter via the shortcut Anaconda adds to your start menu, which will open new tab in your default web browser that should look something like the following screenshot. This isn't a notebook just yet, but don't panic! There's not much to it. This is the Notebook Dashboard, specifically designed for managing your Jupyter Notebooks. Think of it as the launchpad for exploring, editing and creating your notebooks. Be aware that the dashboard will give you access only to the files and sub-folders contained within Jupyter's start-up directory (i.e., where Jupyter or Anaconda is installed). However, the start-up directory can be changed. It is also possible to start the dashboard on any system via the command prompt (or terminal on Unix systems) by entering the command jupyter notebook; in this case, the current working directory will be the start-up directory. With Jupyter Notebook open in your browser, you may have noticed that the URL for the dashboard is something like https://localhost:8888/tree. Localhost is not a website, but indicates that the content is being served from your *local* machine: your own computer. Jupyter's Notebooks and dashboard are web apps, and Jupyter starts up a local Python server to serve these apps to your web browser, making it essentially platform-independent and opening the door to easier sharing on the web. (If you don't understand this yet, don't worry — the important point is just that although Jupyter Notebooks opens in your browser, it's being hosted and run on your local machine. Your notebooks aren't actually on the web until you decide to share them.) The dashboard's interface is mostly self-explanatory — though we will come back to it briefly later. So what are we waiting for? Browse to the folder in which you would like to create your first notebook, click the "New" drop-down button in the top-right and select "Python 3":



Hey presto, here we are! Your first Jupyter Notebook will open in new tab — each notebook uses its own tab because you can open multiple notebooks simultaneously. If you switch back to the dashboard, you will see the new file Untitled.ipynb and you should see some green text that tells you your notebook is running.

What is an ipynb File?

The short answer: each .ipynb file is one notebook, so each time you create a new notebook, a new .ipynb file will be created. The longer answer: Each .ipynb file is a text file that describes the contents of your notebook in a format called JSON. Each cell and its contents, including image attachments that have been converted into strings of text, is listed therein along with some metadata. You can edit this yourself — if you know what you are doing! — by selecting "Edit > Edit Notebook Metadata" from the menu bar in the notebook. You can also view the contents of your notebook files by selecting "Edit" from the controls on the dashboard However, the key word there is *can.* In most cases, there's no reason you should ever need to edit your notebook metadata manually.

The Notebook Interface

Now that you have an open notebook in front of you, its interface will hopefully not look entirely alien.

After all, Jupyter is essentially just an advanced word processor.

Why not take a look around? Check out the menus to get a feel for it, especially take a few moments to scroll down the list of commands in the command palette, which is the small button with the keyboard icon (or Ctrl + Shift + P). There are two fairly prominent terms that you should notice, which are probably new to you: *cells* and *kernels* are key both to understanding Jupyter and to what makes it more than just a word processor. Fortunately, these concepts are not difficult to understand.

- A **kernel** is a "computational engine" that executes the code contained in a notebook document.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.

Cells

We'll return to kernels a little later, but first let's come to grips with cells. Cells form the body of a notebook. In the screenshot of a new notebook in the section above, that box with the green outline is an empty cell. There are two main cell types that we will cover:

A **code cell** contains code to be executed in the kernel. When the code is run, the notebook displays the output below the code cell that generated it.
A Markdown cell contains text formatted using Markdown and displays its output in-place when the Markdown cell is run.
The first cell in a new notebook is always a code cell.
Let's test it out with a classic hello world example: Type print('Hello World!') into the cell and click the run button in the toolbar above or press Ctrl + Enter.
The result should look like this:

```
print('HelloWorld!')
```

Hello World!

When we run the cell, its output is displayed below and the label to its left will have changed from In [ ] to In [1]. The output of a code cell also forms part of the document, which is why you can see it in this article. You can always tell the difference between code and Markdown cells because code cells have that label on the left and Markdown cells do not. The "In" part of the label is simply short for "Input," while the label number indicates *when* the cell was executed on the kernel — in this case the cell was executed first. Run the cell again and the label will change to In [2] because now the cell was the second to be run on the kernel. It will become clearer why this is so useful later on when we take a closer look at kernels. From the menu bar, click *Insert* and select *Insert Cell Below* to create a new code cell underneath your first and try out the following code to see what happens. Do you notice anything different?

```
import time
time.sleep(3)
```

This cell doesn't produce any output, but it does take three seconds to execute. Notice how Jupyter signifies when the cell is currently running by changing its label to In [*]. In general, the output of a cell comes from any text data specifically printed during the cell's execution, as well as the value of the last line in the cell, be it a lone variable, a function call, or something else. For example:

```
def say_hello(recipient):
return 'Hello, {}!'.format(recipient)
say_hello('Tim')
```

'Hello, Tim!'

NumPy

Numpy is the backbone of Machine Learning in Python. It is one of the most important libraries in Python for numerical computations. It adds support to core Python for multi-dimensional arrays (and matrices) and fast vectorized operations on these arrays. The present day NumPy library is a successor of an early library, Numeric, which was created by Jim Hugunin and some other developers. Travis Oliphant, Anaconda's president and co-founder, took the Numeric library as a base and added a lot of modifications, to launch the present day NumPy library in 2005. It is a major open source project and is one of the most popular Python libraries. It's used in almost all Machine Learning and scientific computing libraries. The extent of popularity of NumPy is verified by the fact that major OS distributions, like Linux and MacOS, bundle NumPy as a default package instead of considering it as an add-on package.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Installation of NumPy

If you have Python and PIP already installed on a system, then installation of NumPy is very easy. Install it using this command:

C:\Users\Your Name>`pip install numpy`

If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Import NumPy

Once NumPy is installed, import it in your applications by adding the import keyword:

```
import numpy
```

Now NumPy is imported and ready to use.

Example:

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
  print(arr)
```

Output:

[1 2 3 4 5]

NumPy as np

NumPy is usually imported under the np alias.

Example:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
  print(arr)
```

Output:

```
[1 2 3 4 5]
```

Pandas

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

Is there a correlation between two or more columns? What is average value? Max value? Min value? Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data. Where is the Pandas Codebase?

The source code for Pandas is located at this github repository https://github.com/pandas-dev/pandas

Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy. Install it using this command:

C:\Users*Your Name*>pip install pandas

If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

import pandas

Example:

```
import pandas
mydataset = { 'cars': ["BMW", "Volvo", "Ford"],  'passings': [3, 7, 2] }
myvar = pandas.DataFrame(mydataset)
  print(myvar)
```

Output:

	cars	passings
0	BMW	3
1	Volvo	7
2	Ford	2

Sample Viva Questions:

PANDAS

Q1. What are Pandas?

Q2. What are the Different Types of Data Structures in Pandas?
Q3. List Key Features of Pandas.
Q4. What is Series in Pandas?
Q5. What are the Different Ways to Create a Series?
Q6. How can we Create a Copy of the Series?
Q7. What is a DataFrame in Pandas?
Q8. What are the Different ways to Create a DataFrame in Pandas?
Q9. How to Read Data into a DataFrame from a CSV file?
Q10. How to access the first few rows of a dataframe?
Q11. What is Reindexing in Pandas?
Q12. How to Select a Single Column of a DataFrame?
Q13. How to Rename a Column in a DataFrame?
Q14. How to add an Index, Row, or Column to an Existing Dataframe?
Q15. How to Delete an Index, Row, or Column from an Existing DataFrame?
Q16. How to set the Index in a Panda dataFrame?
Q17. How to Reset the Index of a DataFrame?
Q18. How to Find the Correlation Using Pandas?
Q19. How to Iterate over Dataframe in Pandas?
Q20. What are the Important Conditions to keep in mind before Iterating?

NUMPY
Q1. What is NumPy? Why should we use it?
Q2. How do you convert Pandas DataFrame to a NumPy array?
Q3. How do you concatenate 2 NumPy arrays?
Q4. How do you multiply 2 NumPy array matrices?
Q5. How is arr[:,0] different from arr[:
Q6. How do we check for an empty array (or zero elements array)?
Q7. How do you count the frequency of a given positive value appearing in the NumPy array?
Q8. How is np.mean() different from np.average() in NumPy?
Q9. How can you reverse a NumPy array?
Q10. How do you find the data type of the elements stored in the NumPy arrays?
Q11. What are ways of creating 1D, 2D and 3D arrays in NumPy?
Q12. What are ndarrays in NumPy?
Q13. How are NumPy arrays better than Python's lists?
Q14. Why is NumPy preferred over Matlab, Octave, Idl or Yorick?

Experiment No.-2
Program to demonstrate Simple Linear Regression

To code a simple linear regression model using StatsModels we will require NumPy, pandas, matplotlib, and statsmodels.

Here is a quick overview of the following libraries:

- *NumPy* — used to perform mathematical operations mainly using multi-dimensional arrays.
- *pandas* — used for data manipulation and analysis.
- *matplotlib* — it is a plotting library as a component of NumPy
- *statsmodels* — it is used to explore data, estimate statistical models and perform statistical test.

Import the relevant libraries

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
sns.set()
```

Load the data

In [ ]:

```
data = pd.read_csv('1.01. Simple linear regression.csv')
```

In [ ]:

```
data
```

In [ ]:

```
data.describe()
```

Define the dependent and the independent variables

In [ ]:

```
y = data ['GPA']
x1 = data ['SAT'
```

Explore the data

In [ ]:

```
plt.scatter(x1,y)
```

```
plt.xlabel('SAT', fontsize = 20)
plt.ylabel('GPA', fontsize = 20)
plt.show()
```

Regression itself

In [ ]:

```
x = sm.add_constant(x1)
results = sm.OLS(y,x).fit()
results.summary()
```

In [ ]:

```
plt.scatter(x1,y)
yhat = 0.0017*x1 + 0.275
fig = plt.plot(x1,yhat, lw=4, c='orange', label ='regression line')
plt.xlabel('SAT', fontsize = 20)
plt.ylabel('GPA', fontsize = 20)
plt.show()
```

In [ ]:

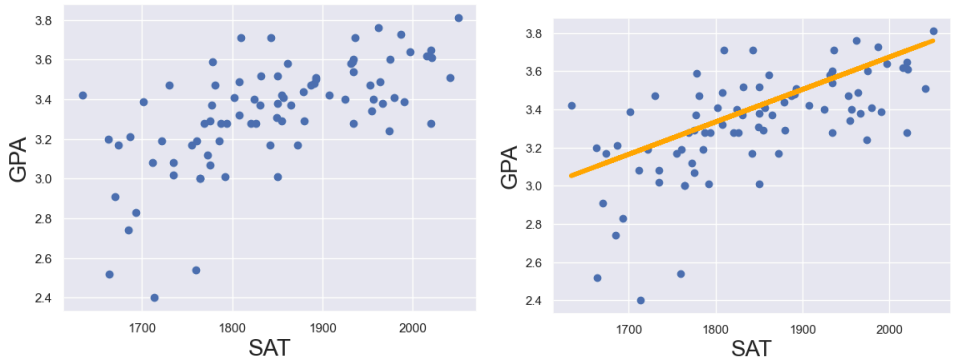
```
plt.scatter(x1,y)
yhat = 0.0017*x1 + 0
fig = plt.plot(x1,yhat, lw=4, c='green', label ='regression line')
plt.xlabel('SAT', fontsize = 20)
plt.ylabel('GPA', fontsize = 20)
plt.xlim(0)
plt.ylim(0)
plt.show()
```

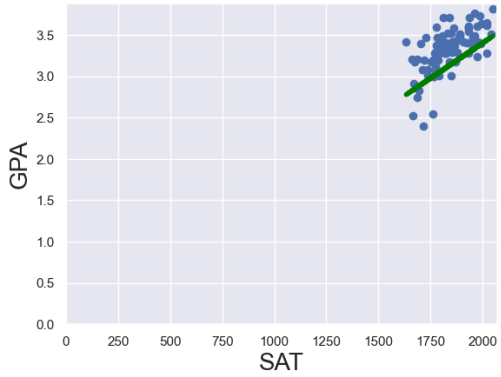
In [ ]:

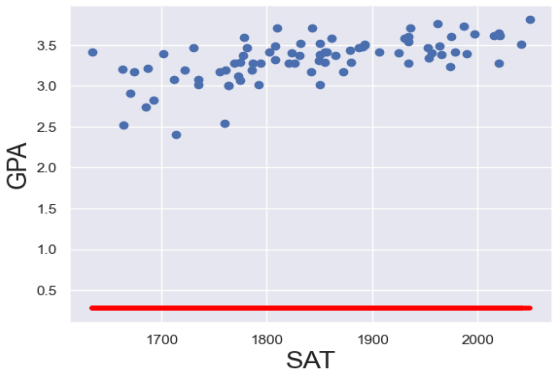
```
plt.scatter(x1,y)
yhat = 0*x1 + 0.275
fig = plt.plot(x1,yhat, lw=4, c='red', label ='regression line')
plt.xlabel('SAT', fontsize = 20)
plt.ylabel('GPA', fontsize = 20)
plt.show()
```

In [ ]:

Output:







Sample viva questions:

1. What is the difference between a population regression line and a sample regression line?
2. What are the assumptions of a linear regression model?
3. What are outliers? How do you detect and treat them? How do you deal with outliers in a linear regression model?
4. How do you determine the best fit line for a linear regression model?
5. What is the difference between simple and multiple linear regression?
6. What is linear Regression Analysis?
7. What is multicollinearity and how does it affect linear regression analysis?
8. What is the difference between linear regression and logistic regression?
9. What are the common types of errors in linear regression analysis?
10. What is the difference between a dependent and independent variable in linear regression?
11. What is an interaction term in linear regression and how is it used?
12. What is the difference between biased and unbiased estimates in linear regression?
13. How do you measure the strength of a linear relationship between two variables?
14. What is linear regression, and how does it work?
15. What is the difference between linear regression and non-linear regression?
16. What are the common techniques used to improve the accuracy of a linear regression model?
17. What is a residual in linear regression and how is it used in model evaluation?
18. What is the difference between a parametric and non-parametric regression model?
19. What are the assumptions of the ordinary least squares method for linear regression?

Ref:

https://medium.com/geekculture/simple-linear-regression-bd4348e1ee62

Experiment No.-3
Program to demonstrate Logistic Regression

Experiment No.- 4
Program to demonstrate Decision Tree – ID3 Algorithm

Experiment No.- 5
Program to demonstrate k‑Nearest Neighbor flowers classification

Experiment No.- 6
Program to demonstrate Naïve- Bayes Classifier

Experiment No.- 7
Program to demonstrate PCA and LDA on Iris dataset

Experiment No.- 8
Program to demonstrate DBSCAN clustering algorithm

Experiment No.- 9
Program to demonstrate K‑Medoid clustering algorithm

Experiment No.- 10
Program to demonstrate K‑Means Clustering Algorithm on Handwritten Dataset

COURSE EXIT SURVEY

BHARATI VIDYAPEETH COLLEGE OF ENGINEERING, NEW DELHI

Department of Computer Science and Engineering
Course Exit Survey
2023- 2024

Subject Name: Machine Learning Lab **Subject Code: ML-407P**
Semester: 6th

Please rate how well you understood the course (Tick the most appropriate option)
(1 – Poor, 2- Good, 3- Excellent)

ML-407P.1 What is your Level of formulating machine learning problems?
1. ☐ **2.** ☐ **3.**

ML-407P.2 To what extent you have learned about regression and feature selection techniques and develop applications based on the same?
1. ☐ **2.** ☐ **3.** ☐

ML-407P.3 To what extent you can apply machine learning techniques such as classification to practical applications?
1. ☐ **2.** ☐ **3.** ☐

ML-407P.4 To what extent you can apply Clustering algorithms to develop various practical applications?
1. ☐ **2.** ☐ **3.** ☐

Suggestions to improve the teaching methodology:

Overall, how do you rate your understanding of the subject (tick whichever is applicable)
1. Below 50%. 2. 50%-70%. 3.70%-90% 4. Above 90%

Name of student
Enrolment number **Signature**