****

**VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS**

**Grade A++ Accredited Institution by NAAC**

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;

Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE

An ISO 9001:2015 Certified Institution

**SCHOOL OF ENGINEERING & TECHNOLOGY**

**Sixth Semester (2022-2026)**

**B.Tech Programme: CSE**

**MACHINE LEARNING LAB**

**(ML – 342P)**

**Submitted To: Submitted By:**

**Ms. Anjani Gupta Name: Ishaan Jain**

**Assistant Professor Enrollment No: 06117702722**

**Branch & Section: CSE B**

**VISION OF INSTITUTE**

To be an educational institute that empowers the field of engineering to build a sustainable future by providing quality education with innovative practices that supports people, planet and profit.

**MISSION OF INSTITUTE**

To groom the future engineers by providing value-based education and awakening students' curiosity, nurturing creativity and building  
capabilities to enable them to make significant contributions to the world.

**INDEX**

**Practical’s as per GGSIPU**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Practical Name** | **Date** | **Marks** | | | **Remark** | **Updated Marks** | **Faculty Signature** |
| **Laboratory Assessment (15 Marks)** | **Class Participation (5 Marks)** | **Viva (5 Marks)** |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

**INDEX**

**Beyond Curriculum**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Practical Name** | **Date** | **Marks** | | | **Remark** | **Updated Marks** | **Faculty Signature** |
| **Laboratory Assessment (15 Marks)** | **Class Participation (5 Marks)** | **Viva (5 Marks)** |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

**PRACTICA L NO. 1**

**Aim:** Introduction to JUPYTER IDE and its libraries Pandas and NumPy

**Theory:** Jupyter Notebook is an open-source web-based interactive development environment that supports multiple programming languages, primarily Python. It allows users to create and share documents that contain live code, equations, visualizations, and explanatory text. Jupyter is widely used in data science, machine learning, and scientific computing due to its flexibility and ease of use.

Features of Jupyter Notebook:

1. **Interactive Coding –** Users can write and execute code in cells, making debugging and experimentation easier.
2. **Markdown Support –** Supports formatted text, images, and LaTeX equations for documentation.
3. **Visualization Support –** Integrates with Matplotlib, Seaborn, and other libraries for data visualization.
4. **Code Execution and Kernel Management –** The kernel executes code and retains variable states across cells.
5. **Cross-Language Support –** Supports languages like Python, R, and Julia via different kernels.

Pandas is an open-source Python library used for data analysis and manipulation. It provides data structures like Series (one-dimensional array) and DataFrame (two-dimensional table) that simplify data handling and processing.

Features of Pandas:

1. **Data Manipulation –** Supports filtering, grouping, and aggregation of data.
2. **Handling Missing Data –** Provides methods to handle NaN (Not a Number) values.
3. **Data Cleaning –** Allows renaming, replacing, and reindexing of data.
4. **File Handling –** Supports reading and writing data in CSV, Excel, JSON, and databases.
5. **Integration with Other Libraries –** Works with NumPy, Matplotlib, and SciPy for scientific computing.

NumPy (Numerical Python) is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions for performing operations efficiently.

Features of NumPy:

1. **Efficient Array Operations –** Supports element-wise operations on large datasets.
2. **Broadcasting –** Performs arithmetic operations on arrays of different shapes without explicit looping.
3. **Mathematical Functions –** Includes trigonometric, statistical, and algebraic functions.
4. **Linear Algebra Support –** Provides matrix operations like inversion, dot product, and eigenvalues.
5. **Random Number Generation –** Generates random numbers for simulations and testing.

Jupyter Notebook provides an interactive platform for coding, documentation, and visualization, making it ideal for data science and machine learning applications. Pandas simplifies data manipulation and analysis, while NumPy enhances numerical computing with efficient array operations. Together, these tools form a powerful ecosystem for scientific computing and data-driven applications.

**Learning Outcomes:**

**PRACTICAL NO. 2**

**Aim:** Program to demonstrate Simple Linear Regression.

**Theory:** Linear Regression is a fundamental statistical and machine learning algorithm used to model the relationship between a dependent variable (Y) and one or more independent variables (X). When there is only one independent variable, it is called Simple Linear Regression. The goal of Simple Linear Regression is to find the best-fitting straight line, also known as the regression line, that minimizes the difference between the actual and predicted values.

The equation of a simple linear regression model is:

𝑌 = w𝑋+ b

Where:

Y = Dependent variable (target/output)

X = Independent variable (input)

w = weight

b = bias (constant term)

The values of w and b are determined using the Least Squares Method, which minimizes the sum of squared differences between actual and predicted values.

**Algorithm:**

The working of Simple Linear Regression involves the following steps:

1. **Data Collection –** Gather historical data containing input-output pairs.
2. **Data Preprocessing –** Handle missing values, remove outliers, and normalize data if necessary.
3. **Splitting Data –** Divide the dataset into training and testing sets to evaluate model performance.
4. **Model Training –** Fit the Simple Linear Regression model to the training data using the Least Squares Method.
5. **Prediction –** Use the trained model to predict new values.
6. **Performance Evaluation –** Measure model accuracy using metrics like Mean Squared Error (MSE) and R-squared (R2) score.

**Dataset Description:** The dataset consists of 10,000 records and contains information about students' study hours and their corresponding exam scores. **It is a** synthetic dataset designed for analyzing the relationship between study hours and exam performance. Its attributes are:

1. **Total\_Hours\_Studied –** Represents the number of hours a student has studied before the exam. The values are continuous.
2. **Exam\_Score –** Represents the score obtained by the student in the exam. The values are continuous and likely range between 0 and 100.

The characteristics of the dataset are:

1. **Number of Observations:** 10,000
2. **Number of Features:** 2 (Total\_Hours\_Studied, Exam\_Score)
3. **Missing Values:** None (all values are present)
4. **Data Type:** Both columns are of type float64, indicating numerical continuous data.

**Programming Code:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error,r2\_score

dataset = pd.read\_csv("synthetic\_exam\_dataset.csv")

X = dataset["Total\_Hours\_Studied"].values.reshape(-1,1)

y = dataset["Exam\_Score"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

prediction = model.predict(X\_test)

mse = mean\_squared\_error(y\_test,prediction)

r2 = r2\_score(y\_test,prediction)

print("Model Performance")

print("Mean Squared Error is: ", mse)

print(r2)

plt.scatter(X\_test, y\_test, color='blue', label='Actual Data')

plt.plot(X\_test, prediction, color='red', linewidth=2, label='Regression Line')

plt.xlabel("Total Hours Studied")

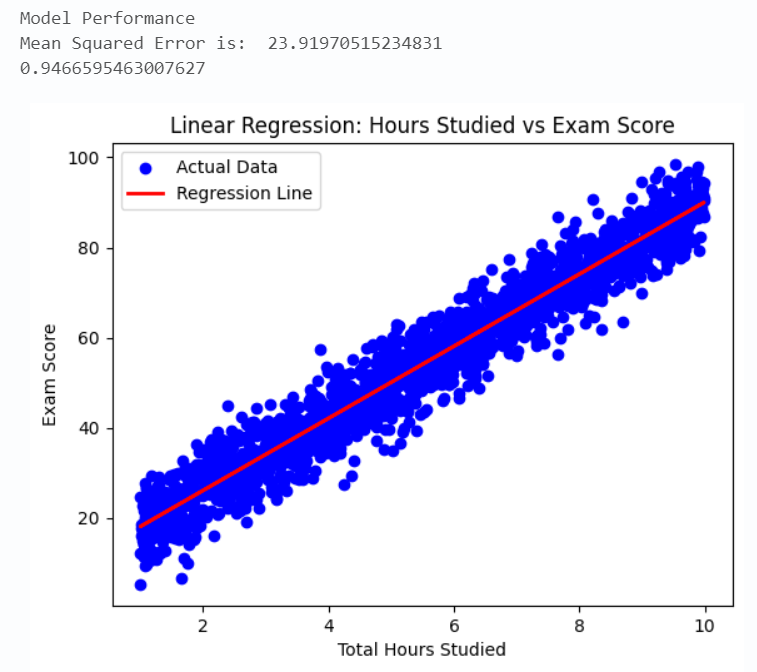
plt.ylabel("Exam Score")

plt.title("Linear Regression: Hours Studied vs Exam Score")

plt.legend()

plt.show()

**Output:**



**Learning Outcomes:**

**PRACTICAL NO. 3**

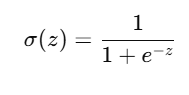
**AIM-** Program to demonstrate Logistic Regression

**THEORY-**

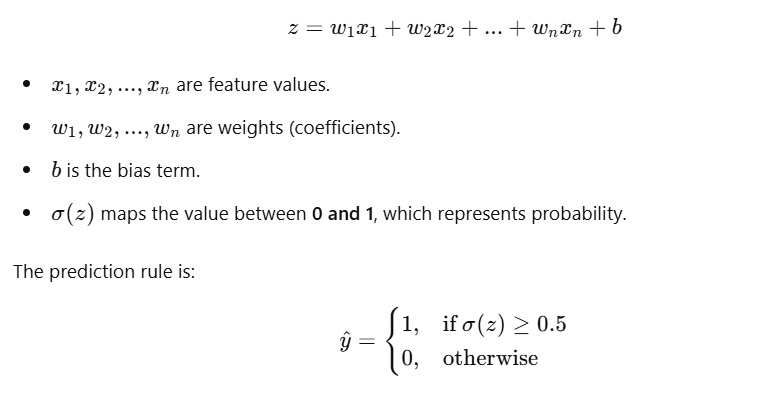
Logistic Regression is a classification algorithm that predicts categorical outcomes using a logistic (sigmoid) function. Unlike linear regression, which predicts continuous values, logistic regression estimates the probability of an event occurring.

**Mathematical Formulation-**

Logistic Regression models the relationship between independent variables (features) and a dependent variable (label) using the logistic function (sigmoid function):



Where:



**ALGORITHM-**

The steps involved in implementing **Logistic Regression** are:

**Step 1: Load the Dataset**

* Import the **Breast Cancer dataset** from sklearn.datasets.
* Extract **features (X)** and **target labels (y)**.

**Step 2: Data Preprocessing**

* Split the dataset into **training (80%)** and **testing (20%)** subsets using train\_test\_split.
* Normalize the feature values using **StandardScaler** to improve convergence.

**Step 3: Model Training**

* Train the **Logistic Regression model** using LogisticRegression().
* Set max\_iter=300 to ensure sufficient iterations for convergence.

**Step 4: Make Predictions**

* Use the trained model to predict tumor classes on the test set.

**Step 5: Model Evaluation**

* Compute **Accuracy**.
* Generate a **Classification Report** (Precision, Recall, F1-score).
* Plot the **Confusion Matrix** using **Seaborn**.

**DATASET EXPLANATION-**

The **Breast Cancer Wisconsin dataset**, available in sklearn.datasets, is commonly used for binary classification tasks. It consists of **569 samples**, each representing a tumor mass. The dataset includes **30 numerical attributes** that describe various characteristics of the tumor, such as its **shape, size, and texture**. These features provide detailed information about the cell nuclei in images. Some key attributes include **mean radius, mean texture, mean perimeter, mean area, and mean smoothness**, along with worst-case values such as **worst radius and worst texture**. The target variable classifies tumors into two categories: **0 (Malignant - cancerous)** and **1 (Benign - non-cancerous)**, making this dataset ideal for developing machine learning models to aid in early cancer detection.

**CODE-**

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, accuracy\_score

# Load the Breast Cancer dataset

cancer = load\_breast\_cancer()

X, y = cancer.data, cancer.target

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train the Logistic Regression model

model = LogisticRegression(max\_iter=300)

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

print(f'Accuracy: {accuracy\_score(y\_test, y\_pred) \* 100:.2f}%')

print(classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Compute confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Plot confusion matrix using seaborn

plt.figure(figsize=(6,5))

sns.heatmap(conf\_matrix, annot=True, cmap="Blues", fmt='d', xticklabels=['Malignant', 'Benign'], yticklabels=['Malignant', 'Benign'])

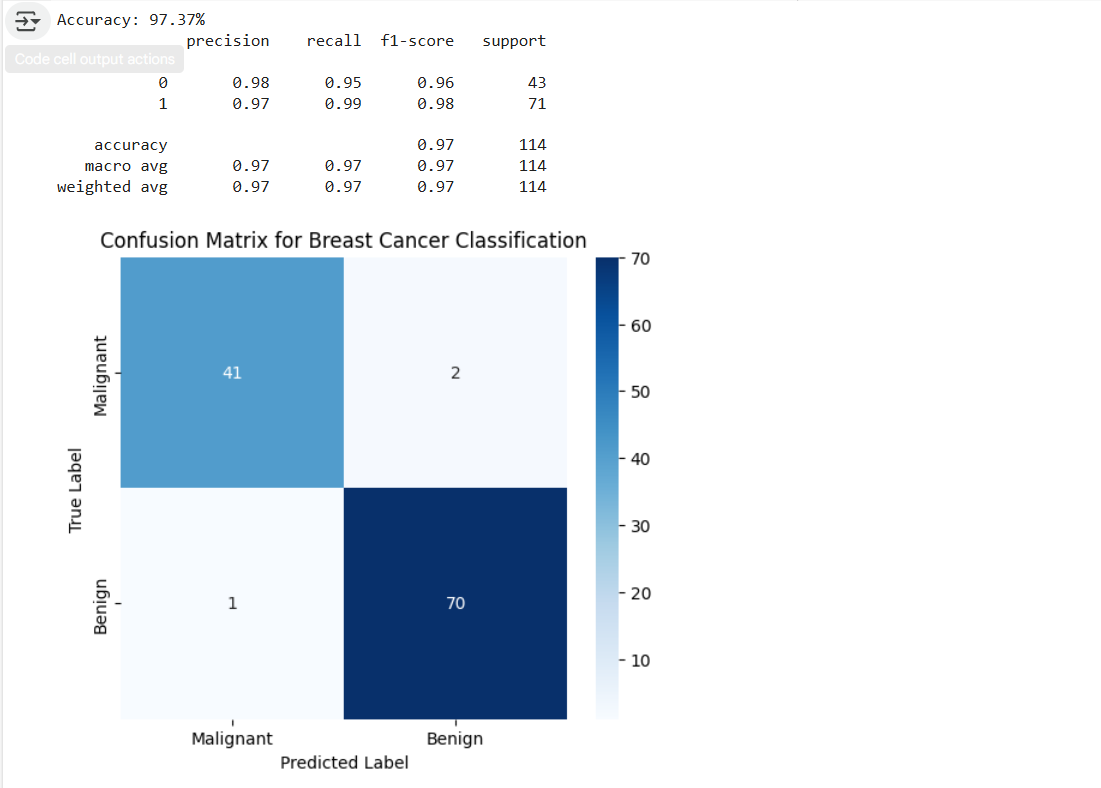
plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix for Breast Cancer Classification')

plt.show()

**OUTPUT-**

****

**LEARNING OUTCOMES-**

**PRACTICAL NO. 4**

**Aim:** Study and Implement classification using SVM.

**Theory:**

Support Vector Machine (SVM) is a powerful supervised machine learning algorithm primarily used for classification tasks, but it can also be applied to regression. SVM works by finding the optimal hyperplane that best separates the classes in the feature space.

The core idea of SVM is to identify a hyperplane that maximizes the margin between different classes in the dataset. The margin is defined as the distance between the hyperplane and the nearest data points from each class, known as support vectors.

**1. Hyperplane and Margin**

* **Hyperplane**: In two dimensions, it’s a line; in three dimensions, it’s a plane; and in higher dimensions, it’s a hyperplane.
* **Margin**: The width of the gap between the two classes. The goal of SVM is to maximize this margin.

**2. Linear SVM**

In linearly separable datasets, SVM finds the optimal hyperplane that separates the classes without any errors. The SVM optimization problem can be formulated as follows:

* Maximize the margin subject to the constraints that all points are classified correctly.

**3. Non-Linear SVM**

Many real-world datasets are not linearly separable. To address this, SVM uses the **kernel trick**, which transforms the input data into a higher-dimensional space where a hyperplane can be used to separate the classes. Common kernels include:

* **Linear Kernel**: No transformation; used for linearly separable data.
* **Polynomial Kernel**: Captures non-linear relationships.
* **Radial Basis Function (RBF) Kernel**: A popular kernel that can handle complex boundaries.

**Code:**

import pandas as pd

from sklearn.datasets import load\_iris

iris = load\_iris()

iris.feature\_names

iris.target\_names

df = pd.DataFrame(iris.data,columns=iris.feature\_names)

df.head()

df['target'] = iris.target

df.head()

df[df.target==1].head()

df[df.target==2].head()

df['flower\_name'] =df.target.apply(lambda x: iris.target\_names[x])

df.head()

df[45:55]

df0 = df[:50]

df1 = df[50:100]

df2 = df[100:]

import matplotlib.pyplot as plt

%matplotlib inline

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')

plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')

plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')

from sklearn.model\_selection import train\_test\_split

X = df.drop(['target','flower\_name'], axis='columns')

y = df.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

len(X\_train)

len(X\_test)

from sklearn.svm import SVC

model = SVC()

model.fit(X\_train, y\_train)

model.score(X\_test, y\_test)

model.predict([[4.8,3.0,1.5,0.3]])

*#Tune parameters*

*#1. Regularization (C)*

model\_C = SVC(C=1)

model\_C.fit(X\_train, y\_train)

model\_C.score(X\_test, y\_test)

model\_C = SVC(C=10)

model\_C.fit(X\_train, y\_train)

model\_C.score(X\_test, y\_test)

*#2. Gamma*

model\_g = SVC(gamma=10)

model\_g.fit(X\_train, y\_train)

model\_g.score(X\_test, y\_test)

model\_linear\_kernal = SVC(kernel='linear')

model\_linear\_kernal.fit(X\_train, y\_train)

model\_linear\_kernal.score(X\_test, y\_test)

**Output:**

**Learning Outcome:**

* Learnt the basics of SVM algorithm and how to implement it.
* Learnt how to plot it graphically.
* Learnt how to predict and know the model score correctly.