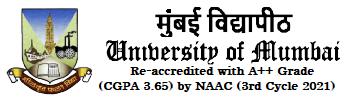
UNIVERSITY OF MUMBAI

**DEPARTMENT OF COMPUTER SCIENCE**



M.Sc. Computer Science – Semester II

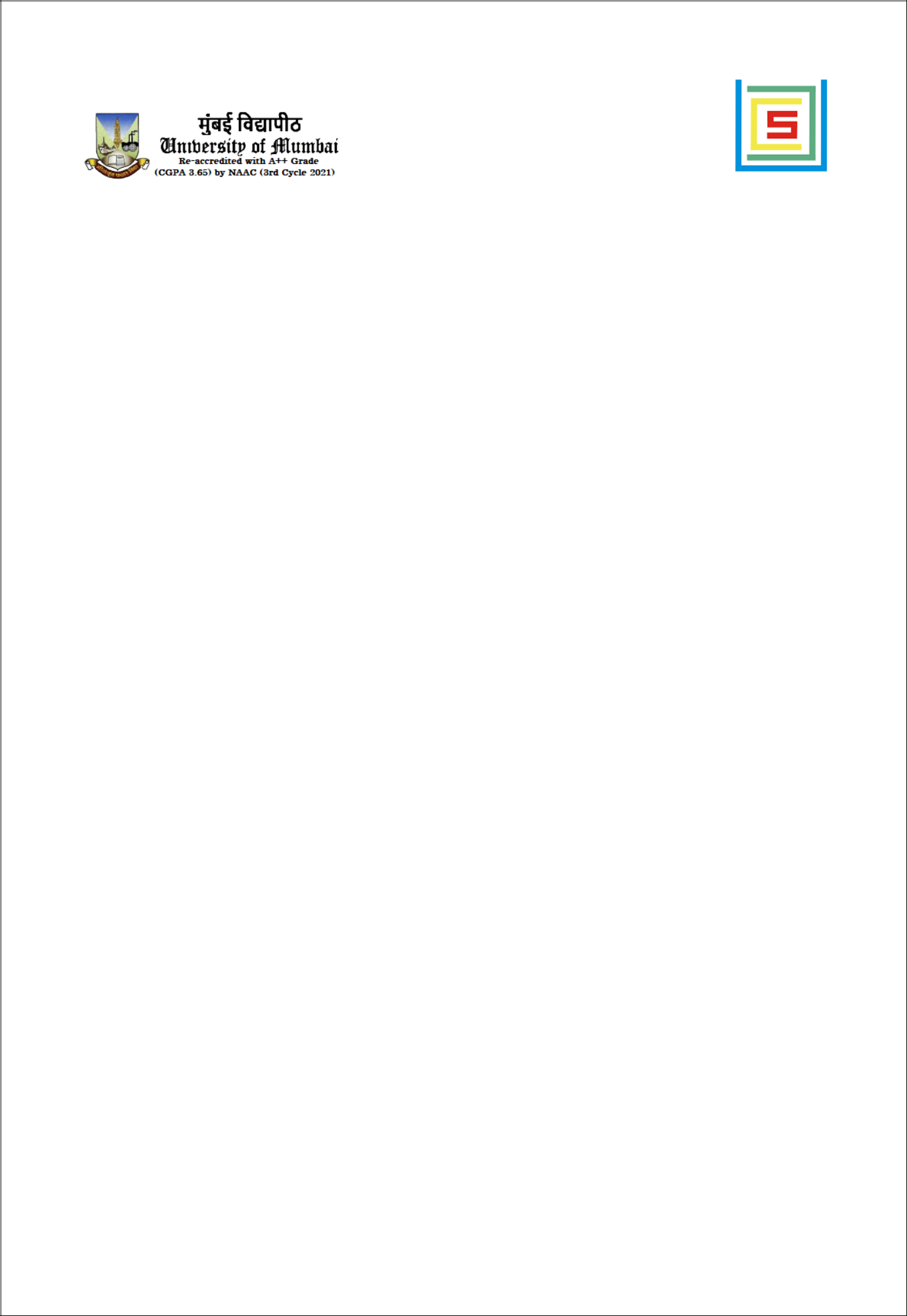
(NEP 2020)

Machine Learning

JOURNAL

2023-2024

Seat No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



UNIVERSITY OF MUMBAI

**DEPARTMENT OF COMPUTER SCIENCE**

**CERTIFICATE**

This is to certify that the work entered in this journal was done in the University

Department of Computer Science laboratory by Mr./Ms.**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Seat No. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

for the course of M.Sc. (Computer Science) - Semester II (NEP 2020) during the academic year 2023- 2024 in a satisfactory manner.

**\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Subject In-charge**

**Head of Department**

**\_\_\_\_\_\_\_\_\_\_\_\_**

**External Examiner**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. no.** | **Name of the practical** | **Page** | **Date** | **Sign** |
|  |  | **No.** |  |  |
|  |  |  |  |  |
| **1** | **Implement Linear** |  |  |  |
|  | **Regression(Diabetes** |  |  |  |
|  | **Dataset).** |  |  |  |
|  |  |  |  |  |
| **2** | **Implement Logistic** |  |  |  |
|  | **Regression. (Iris Dataset)** |  |  |  |
|  |  |  |  |  |
| **3** | **Implement Multinomial** |  |  |  |
|  | **Logistic Regression (Iris** |  |  |  |
|  | **Dataset)** |  |  |  |
| **4** | **Implement SVM Classifier** |  |  |  |
|  | **(Iris Datasets)** |  |  |  |
| **5** | **Train and fine-tune a** |  |  |  |
|  | **Decision Tree for Moon** |  |  |  |
|  | **dataset.** |  |  |  |
| **6** | **Train an SVM regression** |  |  |  |
|  | **on the California Housing** |  |  |  |
|  | **Dataset.** |  |  |  |
|  |  |  |  |  |
| **7** | **Implement Batch Gradient** |  |  |  |
|  | **Descent with Early** |  |  |  |
|  | **Stopping for Softmax** |  |  |  |
|  | **Regression.** |  |  |  |
| **8** | **Implement MLP for** |  |  |  |
|  | **Classification of** |  |  |  |
|  | **Handwritten digits(MNIST** |  |  |  |
|  | **datasets)** |  |  |  |
| **9** | **Classification of Image of** |  |  |  |
|  | **clothing using** |  |  |  |
|  | **Tensorflow(Fashion** |  |  |  |
|  | **MNIST dataset)** |  |  |  |
| **10** | **Implement Regression to** |  |  |  |
|  | **predict fuel efficiency using** |  |  |  |
|  | **TensorFlow (Auto MPG** |  |  |  |
|  | **dataset).** |  |  |  |

1

**PRACTICAL-01**

**Aim: Implement Linear Regression(Diabetes Dataset).**

**Theory:**

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

**Code :**

**import numpy as np**

**import pandas as pd**

**from sklearn import datasets**

**from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_squared\_error, r2\_score import matplotlib.pyplot as plt import seaborn as sns**

* **Load the Diabetes dataset from sklearn diabetes = datasets.load\_diabetes()**
* **Create a DataFrame from the dataset**

**diabetes\_df = pd.DataFrame(data=diabetes.data, columns=diabetes.feature\_names)**

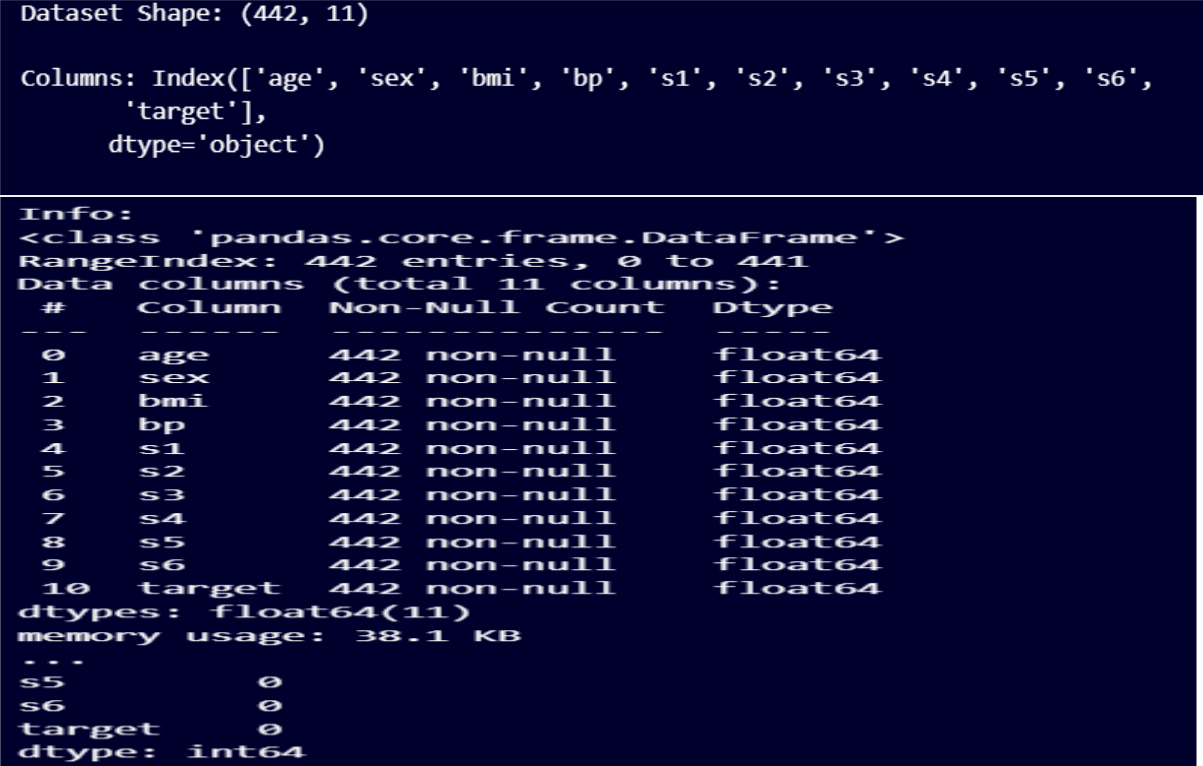
**diabetes\_df['target'] = diabetes.target**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

2

* **Display basic information about the dataset(EDA) print("Dataset Shape:", diabetes\_df.shape) print("\nColumns:", diabetes\_df.columns) print("\nInfo:")**

**print(diabetes\_df.info()) print("\nNull Values:") print(diabetes\_df.isnull().sum())**

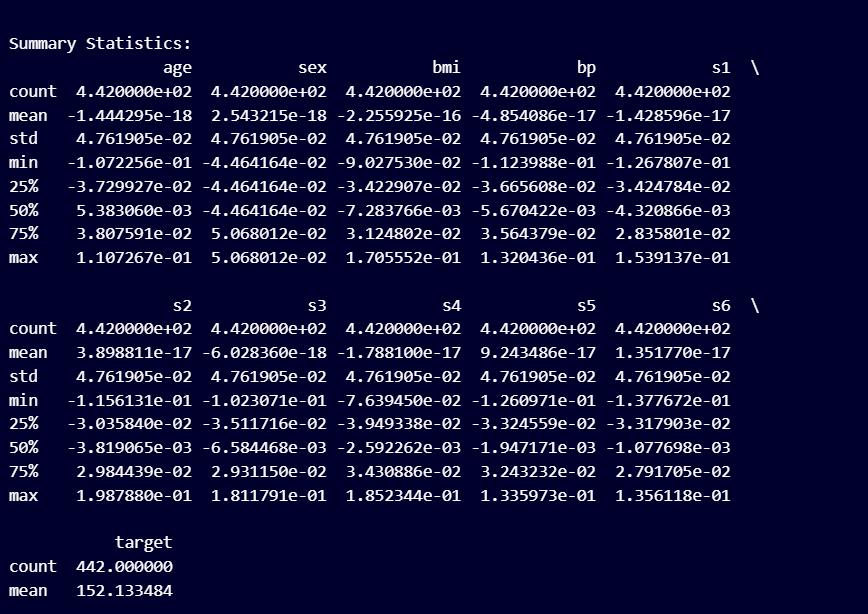


* **Display summary statistics of numerical columns print("\nSummary Statistics:")**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

3

**print(diabetes\_df.describe())**



* **Visualize the distribution of the target variable (diabetes progression) plt.figure(figsize=(8, 6))**

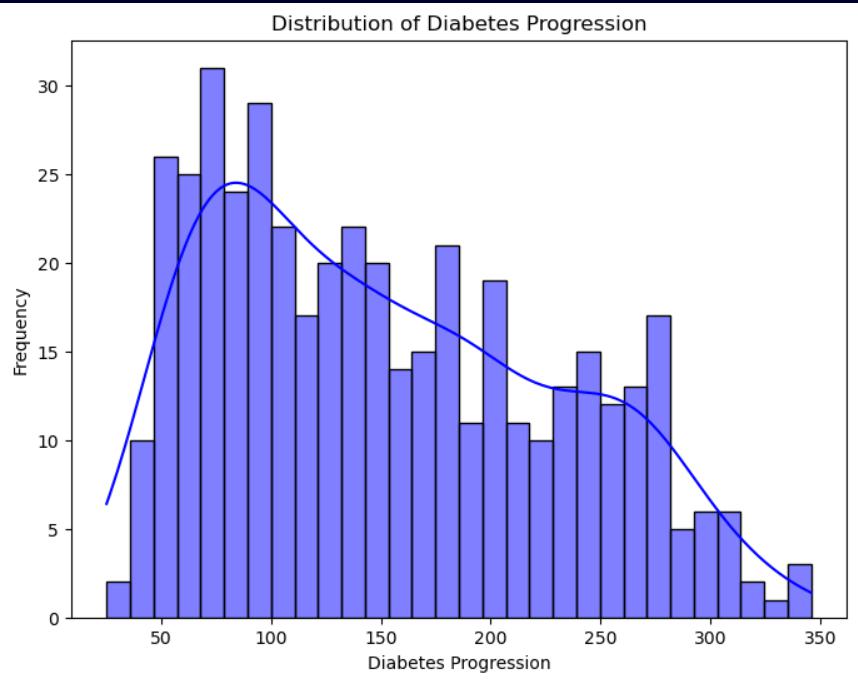
**sns.histplot(diabetes\_df['target'], bins=30, kde=True, color='blue') plt.title('Distribution of Diabetes Progression') plt.xlabel('Diabetes Progression')**

**plt.ylabel('Frequency')**

**plt.show()**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

4



* **Pairplot to visualize relationships between features and the target variable**

**plt.figure(figsize=(12, 10))**

**sns.pairplot(diabetes\_df, diag\_kind='kde')**

**plt.suptitle("Pairplot of Diabetes Dataset", y=1.02)**

**plt.show()**

**#Split the Data into Features and Target**

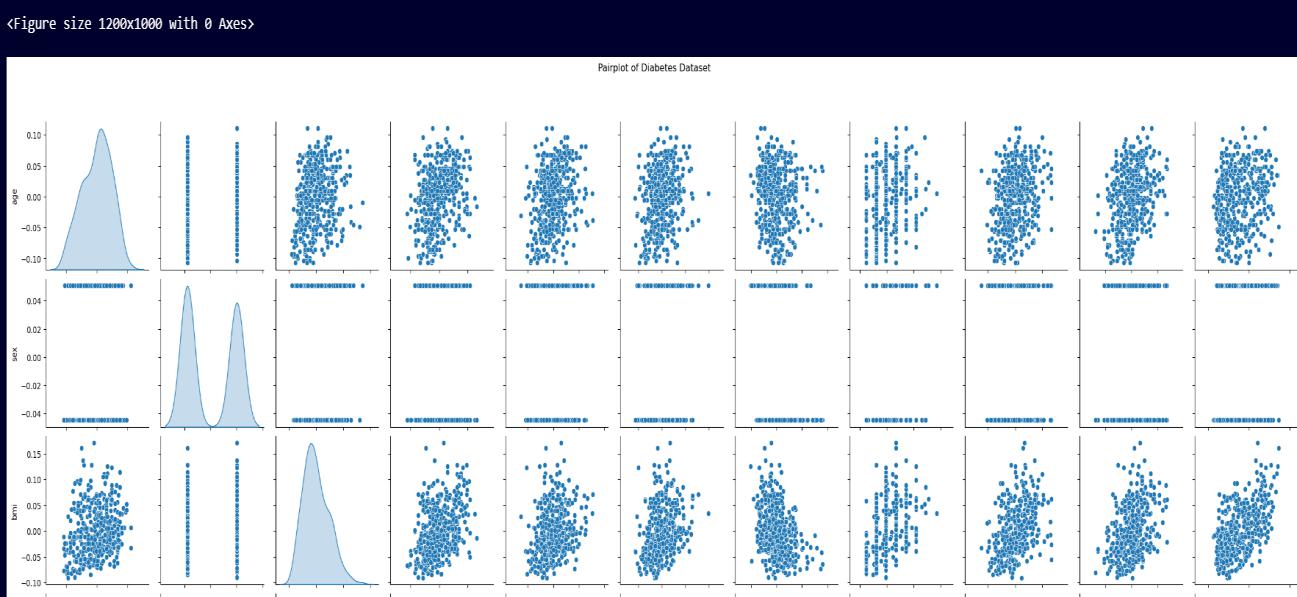
**# Split the dataset into features (X) and target (y)**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

5

**X = diabetes\_df.drop('target', axis=1) # Features**

**y = diabetes\_df['target']** **# Target (continuous variable)**



**# Split the 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)**

* **Initialize the Linear Regression model model = LinearRegression()**
* **Train the model on the training data model.fit(X\_train, y\_train)**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

6

* **Make predictions on the test data y\_pred = model.predict(X\_test)**
* **Evaluate the model's performance**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**r2 = r2\_score(y\_test, y\_pred)**

**print("\nMean Squared Error (MSE):", mse)**

**print("R-squared Score:", r2)**



**print("\nMean Squared Error (MSE):", mse)**

**print("R-squared Score:", r2)**

* **Plot predicted vs actual values plt.figure(figsize=(8, 6)) plt.scatter(y\_test, y\_pred, color='blue')**

**plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], linestyle='--', color='red')**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

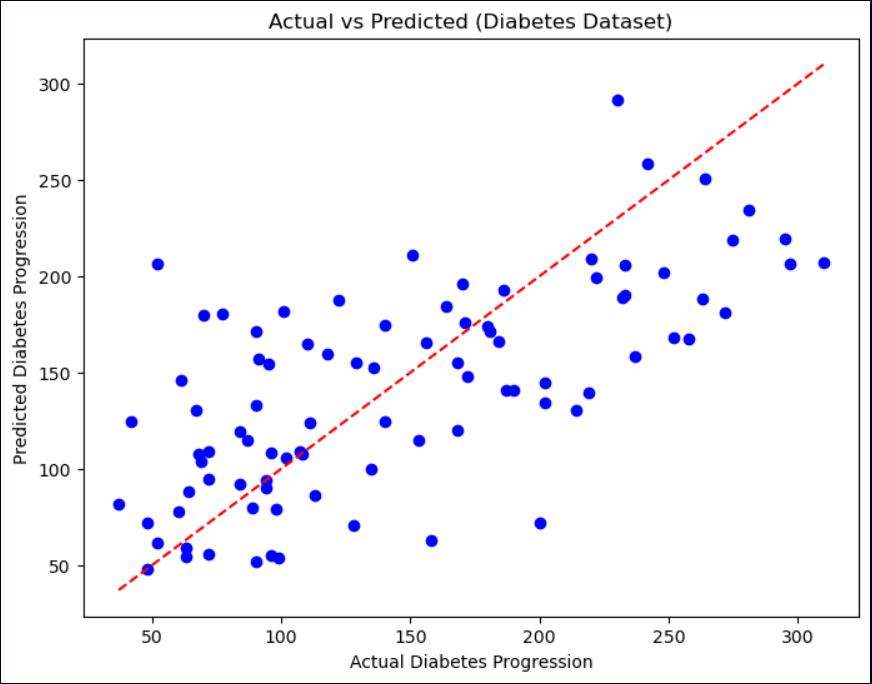
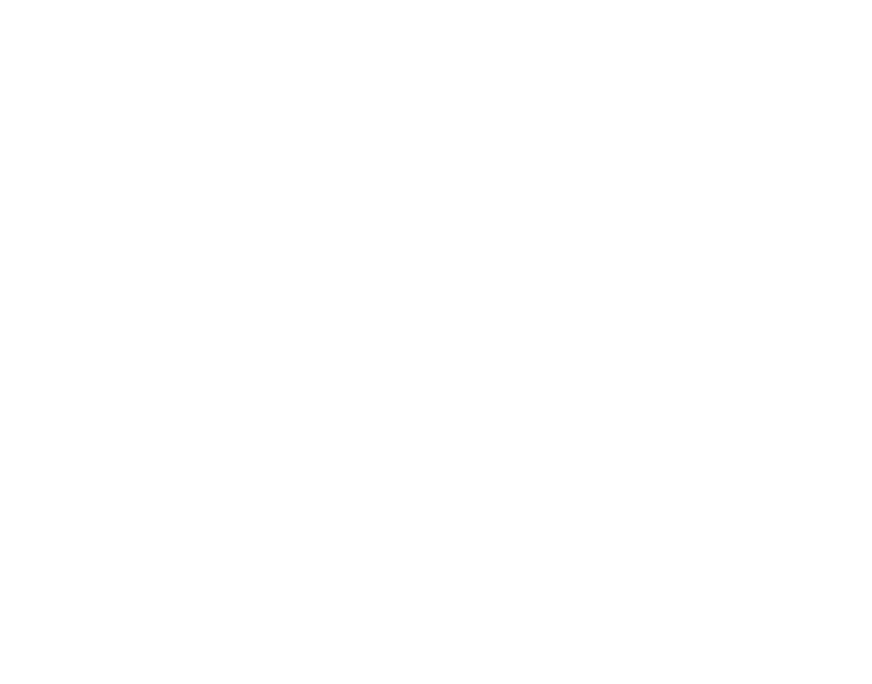
7

**plt.xlabel('Actual Diabetes Progression')**

**plt.ylabel('Predicted Diabetes Progression')**

**plt.title('Actual vs Predicted (Diabetes Dataset)')**

**plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

8

**Practical No 2**

**Aim: Implement Logistic Regression. (Iris Dataset)**

**Theory:**

**Logistic regression** is a supervised machine learning algorithm widely used for binary classification tasks, such as identifying whether an email is spam or not and diagnosing diseases by assessing the presence or absence of specific conditions based on patient test results.

**CODE:**

**import numpy as np**

**import pandas as pd**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn import datasets**

**from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report**

* **Load the Iris dataset from sklearn iris = datasets.load\_iris()**
* **Convert the data into a DataFrame**

**iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names) iris\_df['target'] = iris.target**

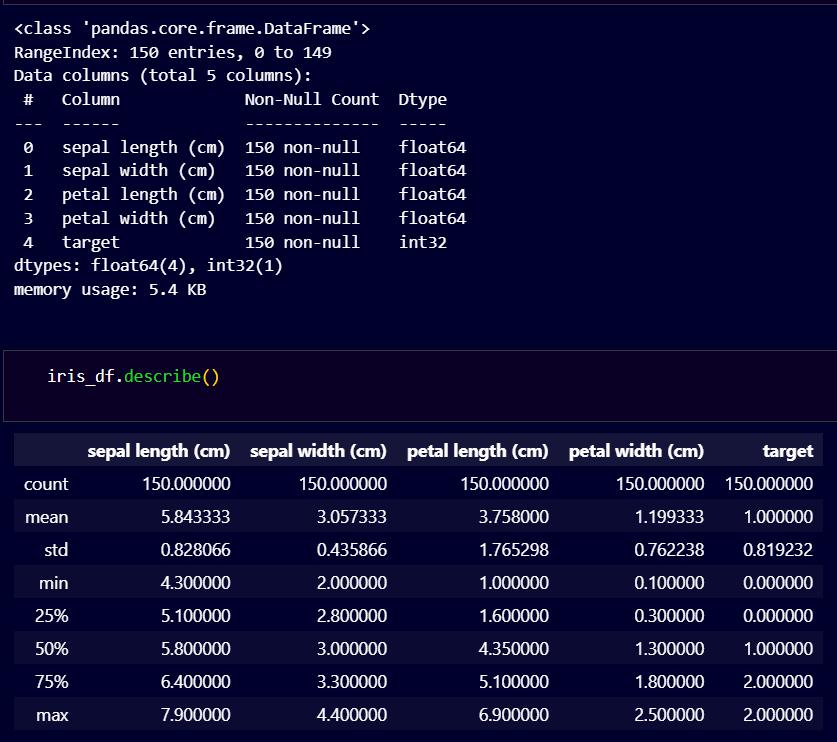
M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

9

**iris\_df.shape**



**iris\_df.info()**

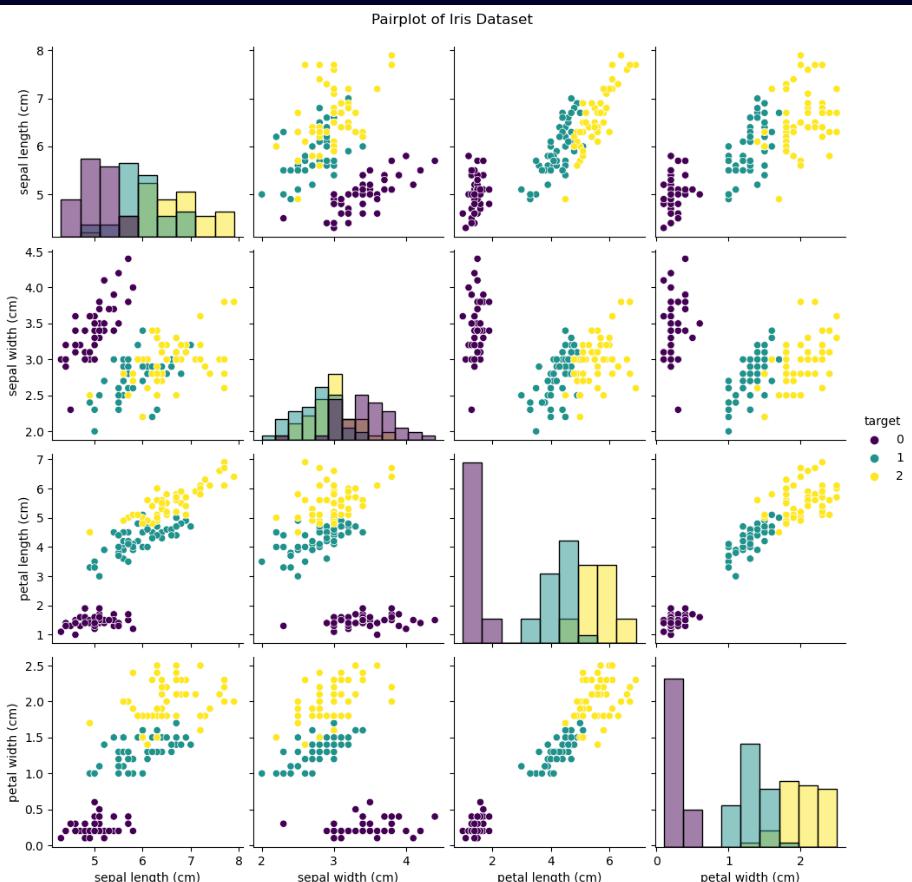


* **EDA: Pairplot to visualize relationships between features sns.pairplot(iris\_df, hue='target', palette='viridis', diag\_kind='hist') plt.suptitle("Pairplot of Iris Dataset", y=1.02)**

**plt.show()**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

10



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

11

* **Other Plots plt.figure(figsize=(12, 6))**
* **Boxplot**

**plt.subplot(1, 2, 1)**

**sns.boxplot(x='target', y='sepal length (cm)', data=iris\_df)**

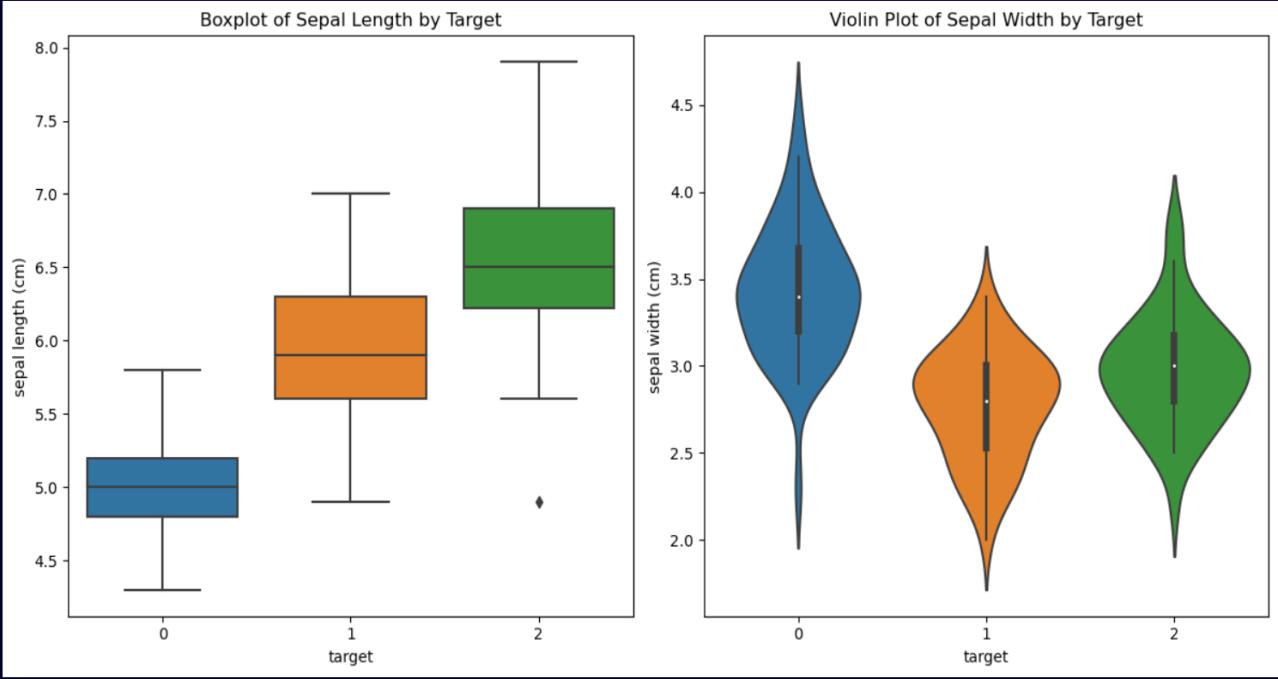
**plt.title('Boxplot of Sepal Length by Target')**

* **Violin Plot plt.subplot(1, 2, 2)**

**sns.violinplot(x='target', y='sepal width (cm)', data=iris\_df) plt.title('Violin Plot of Sepal Width by Target')**

**plt.tight\_layout()**

**plt.show()**



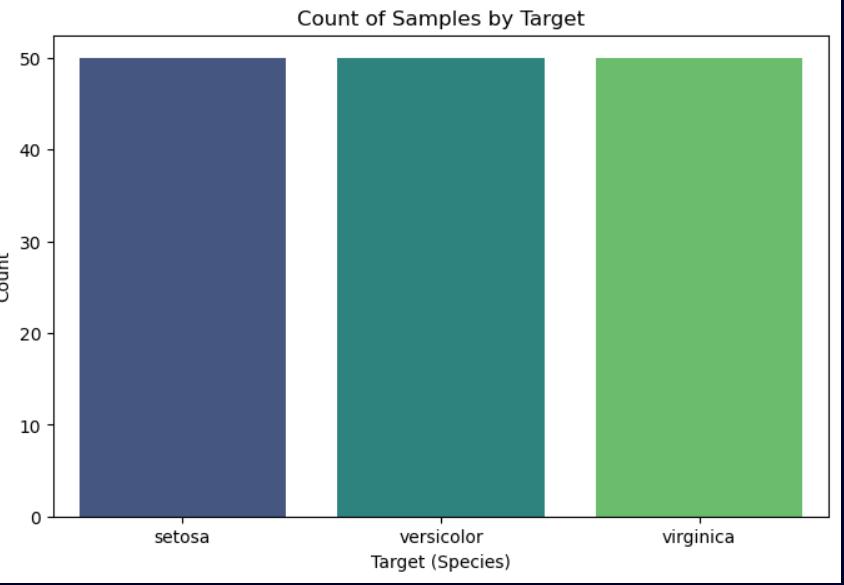
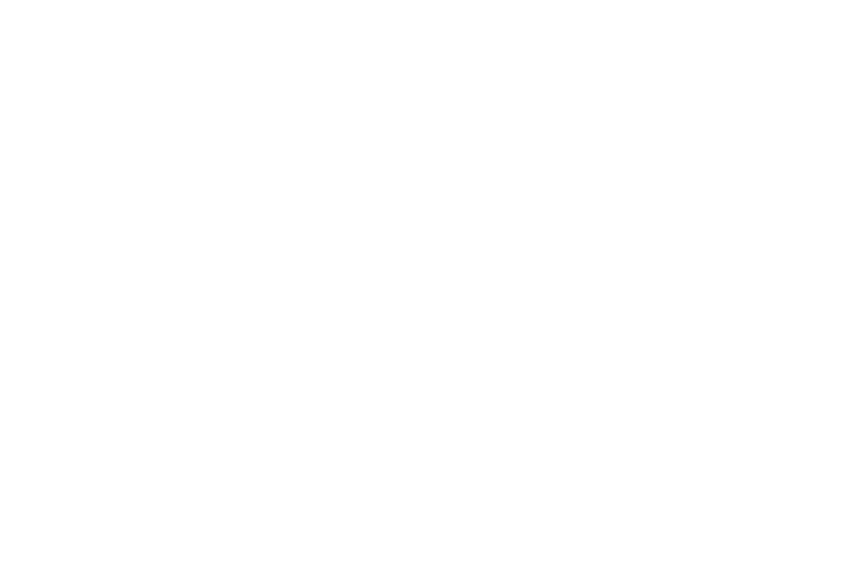
M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

12

* **Count Plot (Bar Plot) plt.figure(figsize=(8, 5)) sns.countplot(x='target', data=iris\_df, palette='viridis') plt.title('Count of Samples by Target') plt.xlabel('Target (Species)')**

**plt.ylabel('Count')**

**plt.xticks(ticks=[0, 1, 2], labels=iris.target\_names) plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

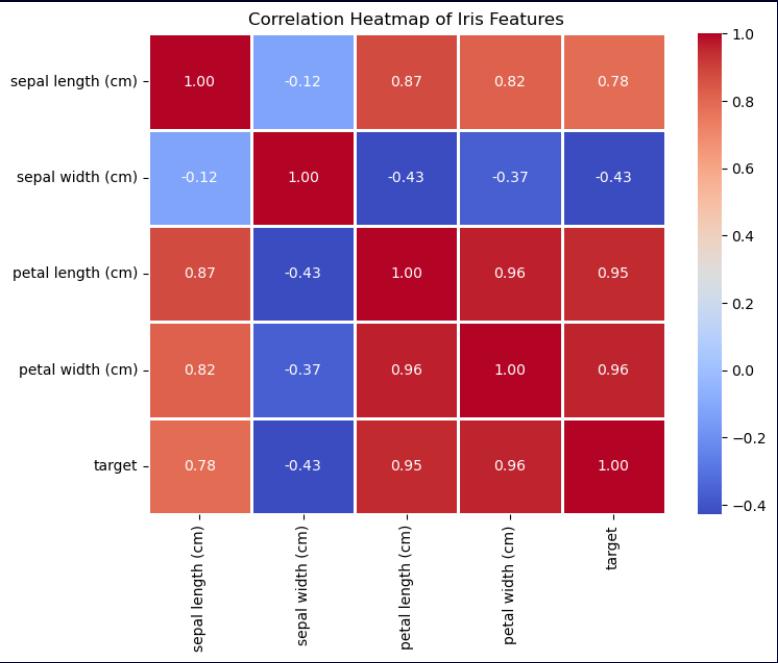
13

* **Correlation Heatmap plt.figure(figsize=(8, 6))**

**sns.heatmap(iris\_df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=1, linecolor='white')**

**plt.title('Correlation Heatmap of Iris Features')**

**plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

14

* **Separate features (X) and target (y) from the DataFrame X = iris\_df.drop('target', axis=1) # Features**

**y = iris\_df['target']** **# Target (labels)**

**# Split the dataset 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 features by removing the mean and scaling to unit variance scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)**

* **Initialize the Logistic Regression model**

**model = LogisticRegression()**

* **Train the model on the training data model.fit(X\_train, y\_train)**
* **Predict on the test data**

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

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

15

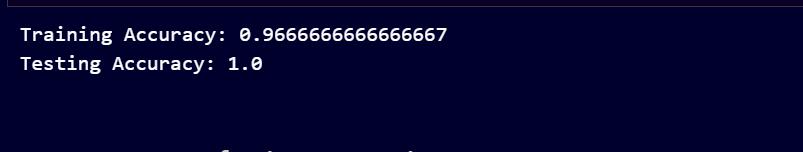
**# Calculate training and testing accuracy**

**train\_accuracy = accuracy\_score(y\_train, model.predict(X\_train))**

**test\_accuracy = accuracy\_score(y\_test, y\_pred)**

**print("Training Accuracy:", train\_accuracy)**

**print("Testing Accuracy:", test\_accuracy)**

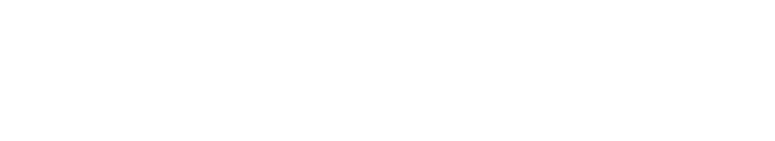


**# Create a confusion matrix**

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

**print("\nConfusion Matrix:")**

**print(conf\_matrix)**

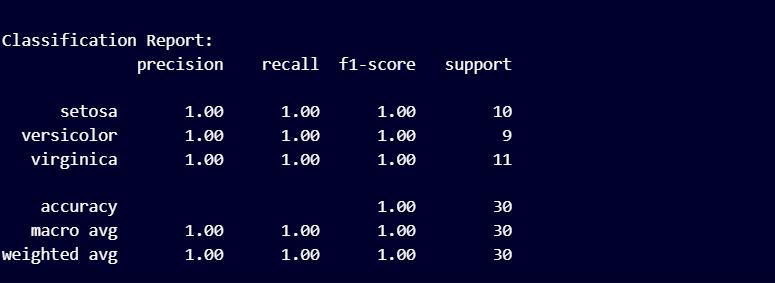


M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

16

* **Print classification report print("\nClassification Report:")**

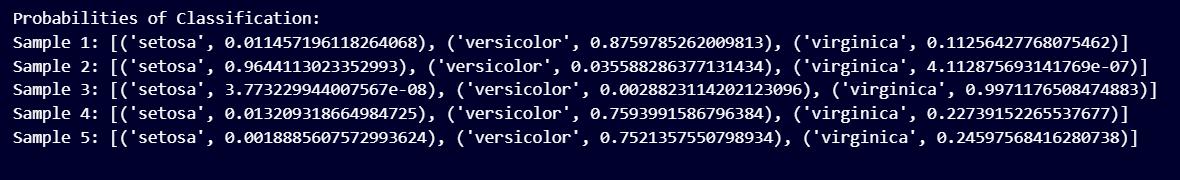
**print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))**



* **Print probabilities of classification for the first few samples in the test set print("\nProbabilities of Classification:")**

**probabilities = model.predict\_proba(X\_test[:5]) for i, prob in enumerate(probabilities):**

**print(f"Sample {i+1}: {list(zip(iris.target\_names, prob))}")**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

17

**Practical No 3**

**Aim:- Implement Multinomial Logistic Regression (Iris Dataset)**

**Theory:**

A **multinomial logistic regression** (or multinomial regression for short) is used when the outcome variable being predicted is nominal and has more than two categories that do not have a given rank or order. This model can be used with any number of independent variables that are categorical or continuous.

**CODE:**

**import numpy as np**

**import pandas as pd**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn import datasets**

**from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report**

* **Load the Iris dataset from sklearn iris = datasets.load\_iris()**
* **Convert the data into a DataFrame**

**iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names) iris\_df['target'] = iris.target**

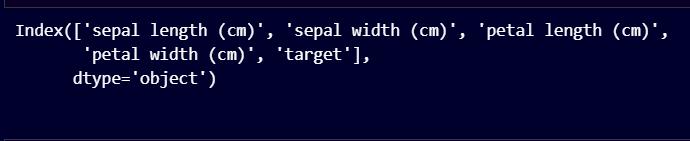
M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

18

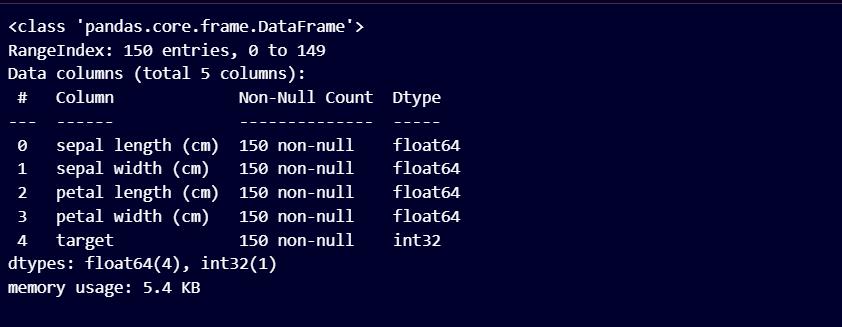
**iris\_df.shape**



**iris\_df.columns**



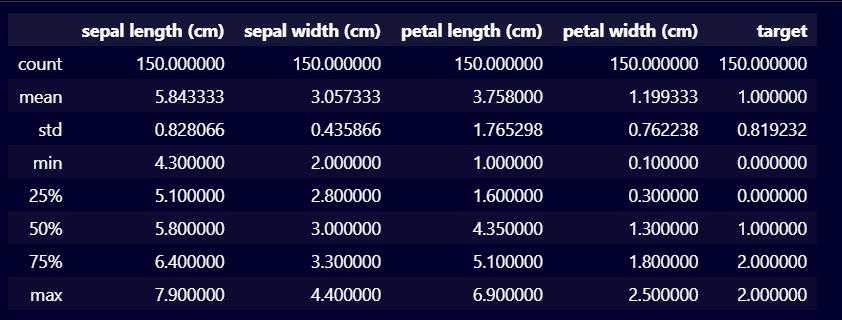
**iris\_df.info()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

19

**iris\_df.describe()**

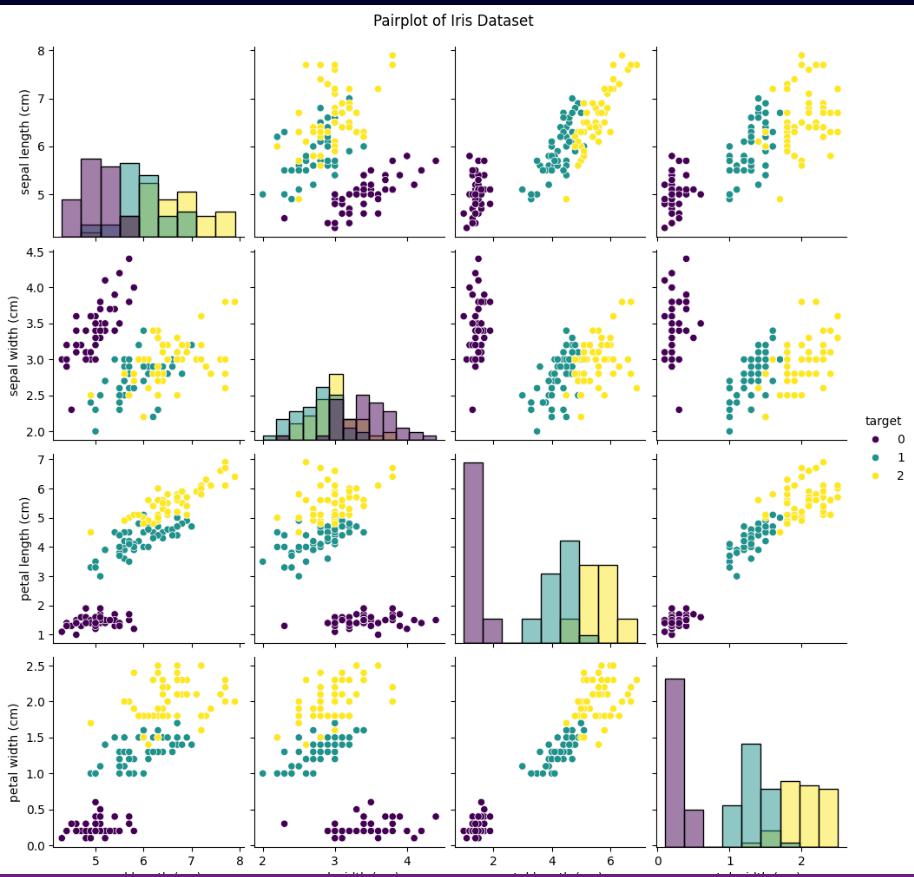


* **EDA: Pairplot to visualize relationships between features sns.pairplot(iris\_df, hue='target', palette='viridis', diag\_kind='hist') plt.suptitle("Pairplot of Iris Dataset", y=1.02)**

**plt.show()**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

20



* + **Other Plots plt.figure(figsize=(12, 6))**
* **Boxplot**

**plt.subplot(1, 2, 1)**

**sns.boxplot(x='target', y='sepal length (cm)', data=iris\_df)**

**plt.title('Boxplot of Sepal Length by Target')**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

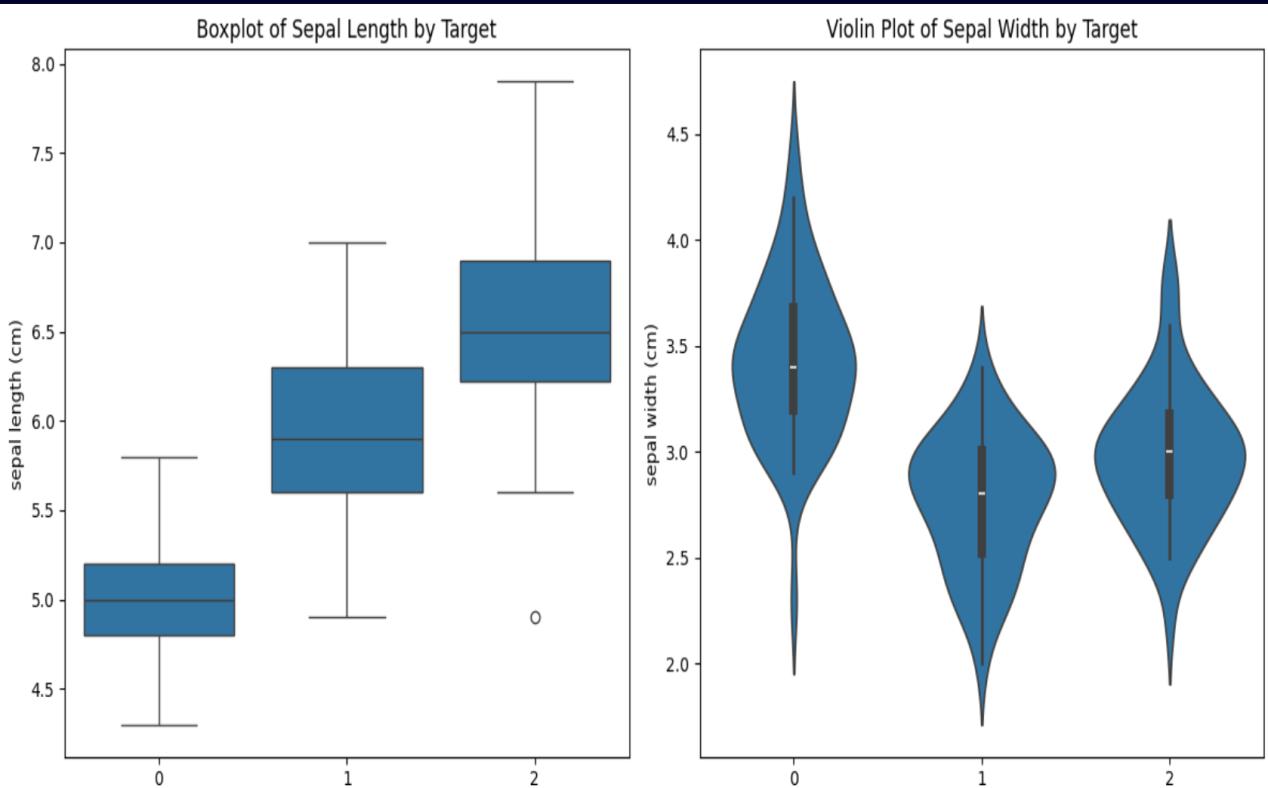
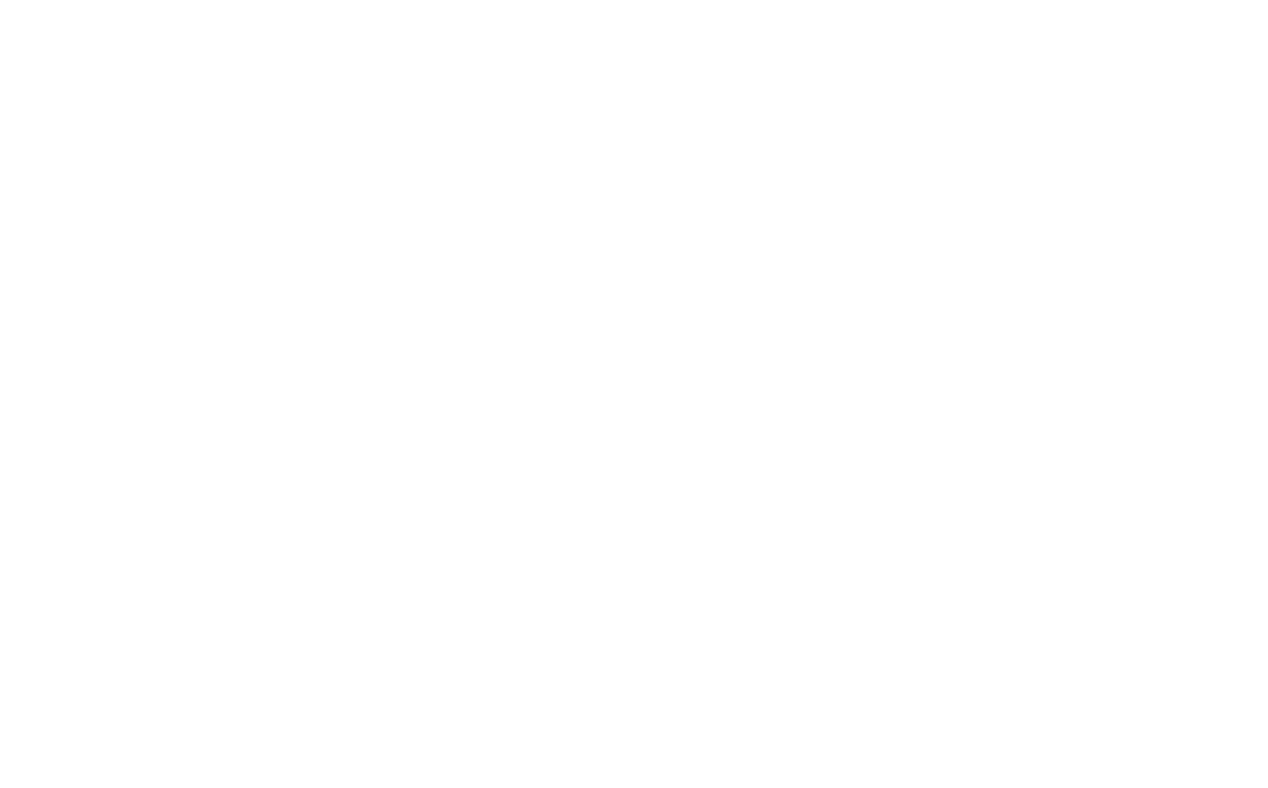
21

* **Violin Plot plt.subplot(1, 2, 2)**

**sns.violinplot(x='target', y='sepal width (cm)', data=iris\_df) plt.title('Violin Plot of Sepal Width by Target')**

**plt.tight\_layout()**

**plt.show()**



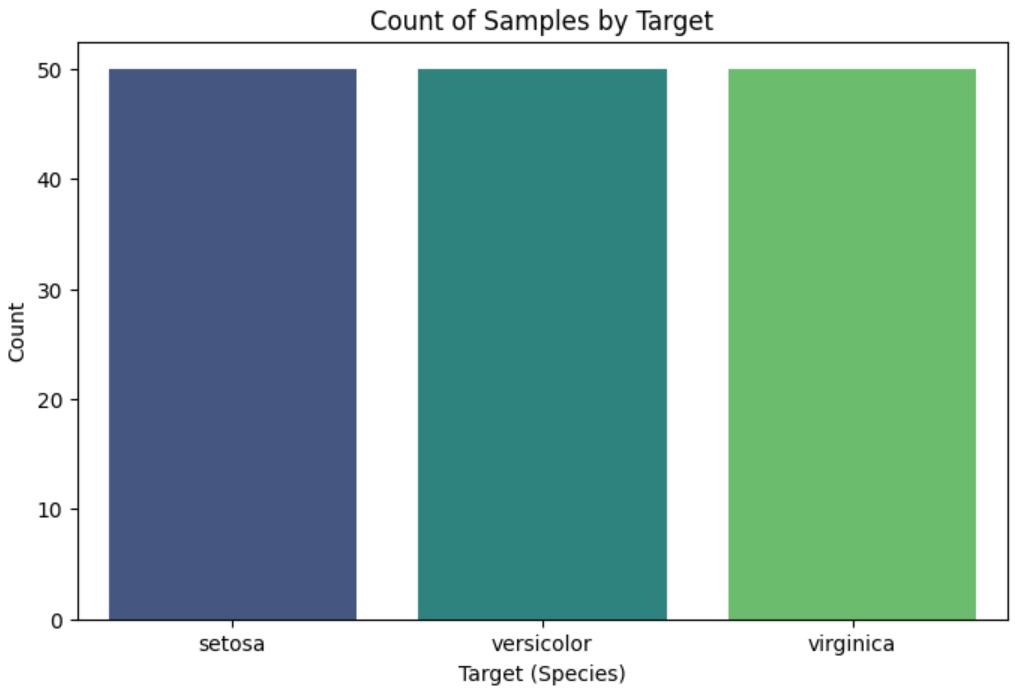
M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

22

* **Count Plot (Bar Plot) plt.figure(figsize=(8, 5)) sns.countplot(x='target', data=iris\_df, palette='viridis') plt.title('Count of Samples by Target') plt.xlabel('Target (Species)')**

**plt.ylabel('Count')**

**plt.xticks(ticks=[0, 1, 2], labels=iris.target\_names) plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

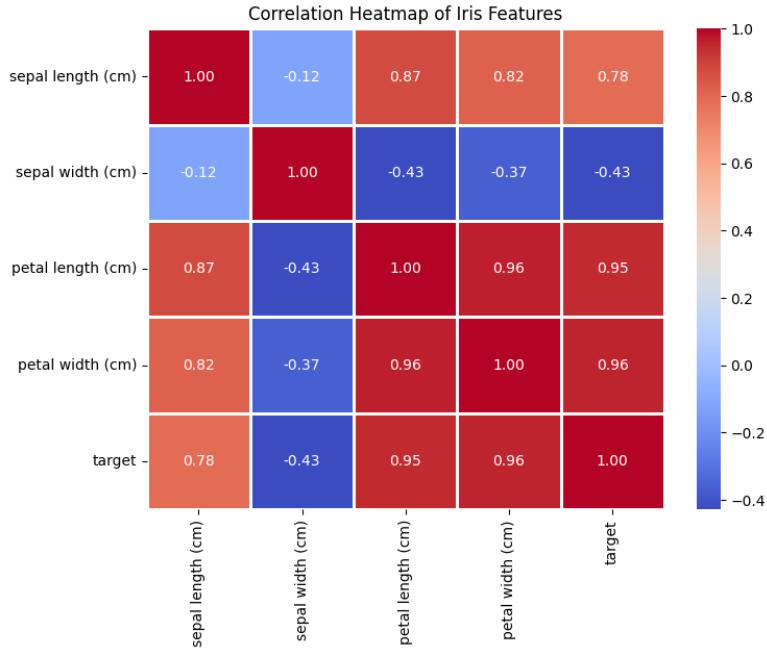
23

* **Correlation Heatmap plt.figure(figsize=(8, 6))**

**sns.heatmap(iris\_df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=1, linecolor='white')**

**plt.title('Correlation Heatmap of Iris Features')**

**plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

24

**# Split the dataset 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 features by removing the mean and scaling to unit variance scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)**

* **Initialize the Logistic Regression model**

**model = LogisticRegression(multi\_class="multinomial")**

* **Train the model on the training data model.fit(X\_train, y\_train)**
* **Predict on the test data**

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

**# Calculate training and testing accuracy**

**train\_accuracy = accuracy\_score(y\_train, model.predict(X\_train))**

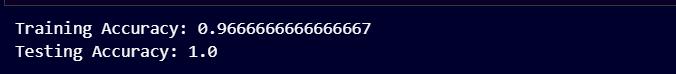
**test\_accuracy = accuracy\_score(y\_test, y\_pred)**

**print("Training Accuracy:", train\_accuracy)**

**print("Testing Accuracy:", test\_accuracy)**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

25



**# Create a confusion matrix**

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

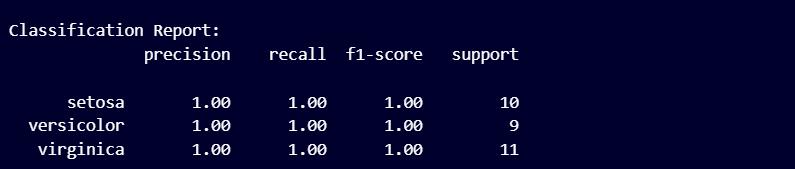
**print("\nConfusion Matrix:")**

**print(conf\_matrix)**



* **Print classification report print("\nClassification Report:")**

**print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

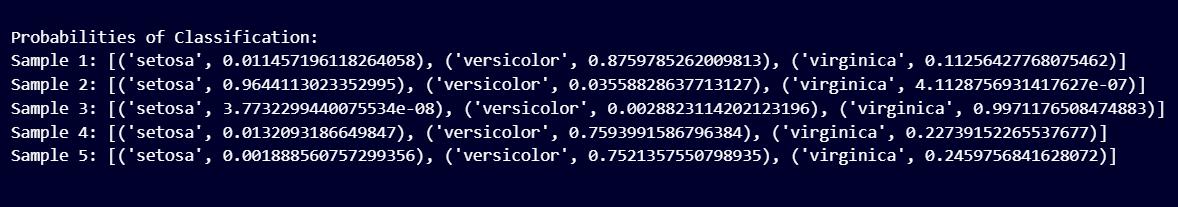
26



* **Print probabilities of classification for the first few samples in the test set print("\nProbabilities of Classification:")**

**probabilities = model.predict\_proba(X\_test[:5]) for i, prob in enumerate(probabilities):**

**print(f"Sample {i+1}: {list(zip(iris.target\_names, prob))}")**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

27

**Practical No 4**

**Aim: Implement SVM Classifier (Iris Datasets)**

**Theory:**

A **support vector machine** (SVM) is defined as a machine learning algorithm that uses supervised learning models to solve complex classification, regression, and outlier detection problems by performing optimal data transformations that determine boundaries between data points based on predefined classes, labels.

**CODE:**

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn import datasets**

**from sklearn.datasets import load\_iris**

**from sklearn.model\_selection import train\_test\_split from sklearn.svm import SVC**

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

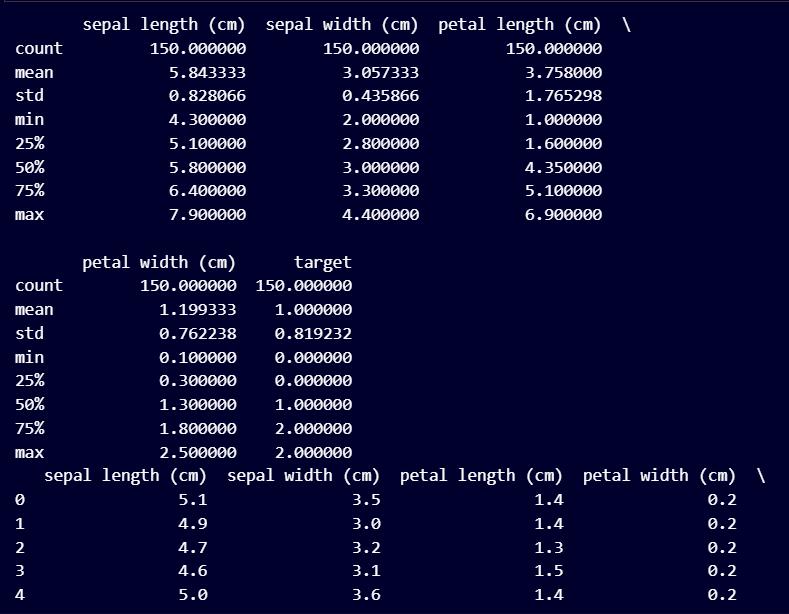
* **Load the Iris dataset from sklearn iris = datasets.load\_iris()**
* **Convert the data into a DataFrame**

**iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names) iris\_df['target'] = iris.target**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

28

**# Exploratory Data Analysis (EDA) print(iris\_df.describe()) # Summary statistics print(iris\_df.head()) # View first few rows**

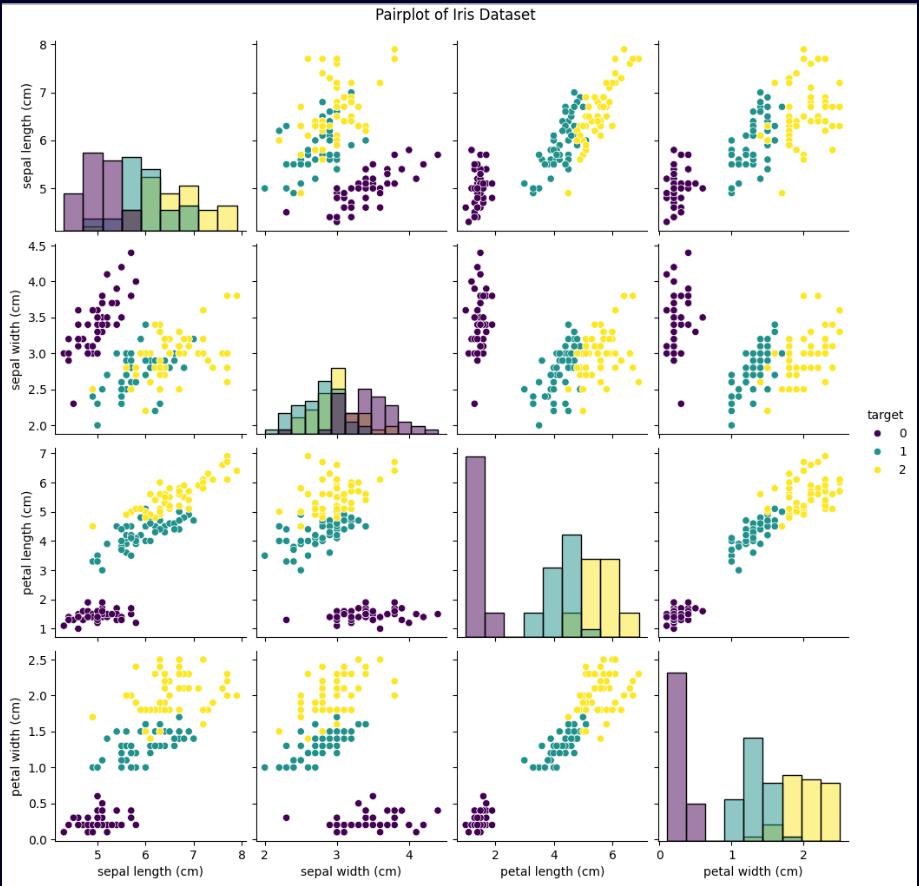


* **Visualization (pairplot for all features) import seaborn as sns**
* **EDA: Pairplot to visualize relationships between features sns.pairplot(iris\_df, hue='target', palette='viridis', diag\_kind='hist') plt.suptitle("Pairplot of Iris Dataset", y=1.02)**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

29

**plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

30

* **Separate features (X) and target (y) from the DataFrame X = iris\_df.drop('target', axis=1) # Features**

**y = iris\_df['target']** **# Target (labels)**

**# Split the dataset 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)**

**# Create the SVM model**

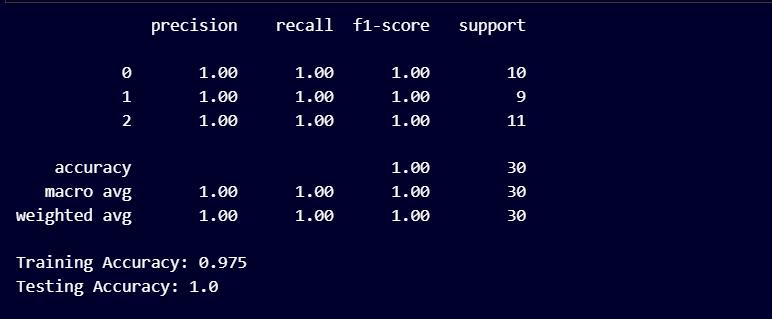
**clf = SVC(kernel='linear') # Experiment with different kernels (e.g., 'rbf')**

* **Train the model clf.fit(X\_train, y\_train)**
* **Make predictions on the testing set y\_pred = clf.predict(X\_test)**
* **Evaluate model performance print(classification\_report(y\_test, y\_pred))**

**print("Training Accuracy:", accuracy\_score(y\_train, clf.predict(X\_train))) print("Testing Accuracy:", accuracy\_score(y\_test, y\_pred))**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

31



**HYPER PARAMETER TUNING**

**from sklearn.model\_selection import GridSearchCV**

* **Define a parameter grid to explore param\_grid = {'kernel': ['linear', 'rbf'],**

**'C': [0.01, 0.1, 1, 10, 100]}**

* **Create the GridSearchCV object**

**grid\_search = GridSearchCV(SVC(), param\_grid, cv=5) # 5-fold cross-validation**

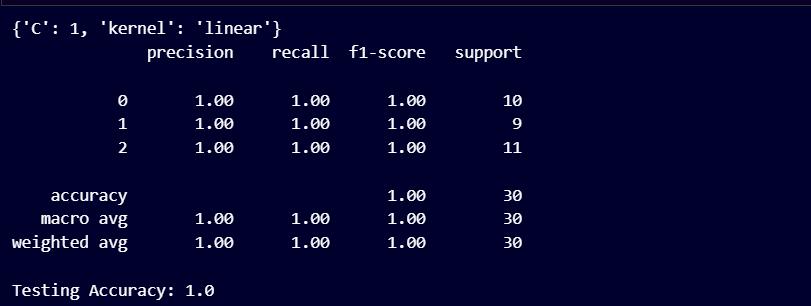
* **Fit the grid search to the training data grid\_search.fit(X\_train, y\_train)**
* **Get the best model and its parameters best\_model = grid\_search.best\_estimator\_ best\_params = grid\_search.best\_params\_ print(best\_params)**
* **Use the best model for prediction and evaluation y\_pred = best\_model.predict(X\_test)**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

32

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

**print("Testing Accuracy:", accuracy\_score(y\_test, y\_pred))**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

33

**Practical No 5**

**Aim: Train and fine-tune a Decision Tree for Moon dataset.**

**Theory:**

A **decision tree** is a decision support hierarchical model that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

**CODE:**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**from sklearn.datasets import make\_moons**

**from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.model\_selection import GridSearchCV from sklearn.tree import plot\_tree**

**# Generate moons data**

**X, y = make\_moons(n\_samples=1000, noise=0.3)**

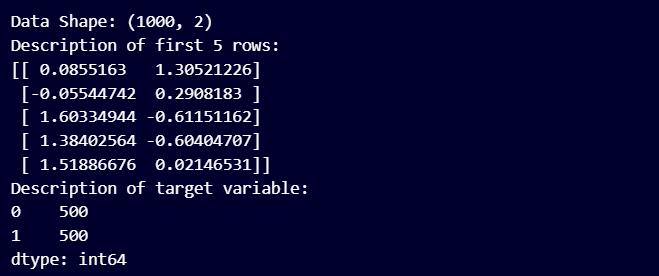
* **1. Data Shape and Description print("Data Shape:", X.shape) print("Description of first 5 rows:") print(X[:5])**

**print("Description of target variable:")**

**print(pd.Series(y).value\_counts()) # Convert y to pandas Series**

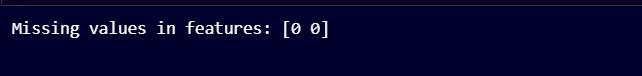
M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

34



* **2. Check for Missing Values import numpy as np**

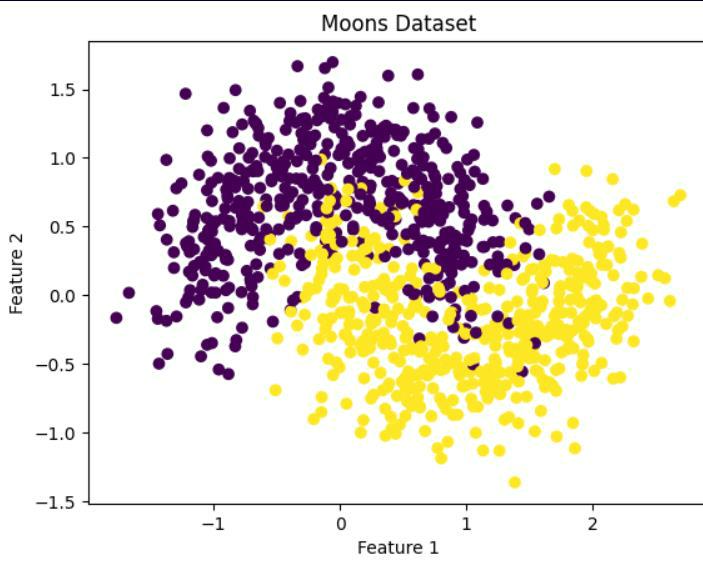
**print("Missing values in features:", np.isnan(X).sum(axis=0))**



* **3. Visualize the moons data plt.scatter(X[:, 0], X[:, 1], c=y) plt.title("Moons Dataset") plt.xlabel("Feature 1") plt.ylabel("Feature 2") plt.show()**

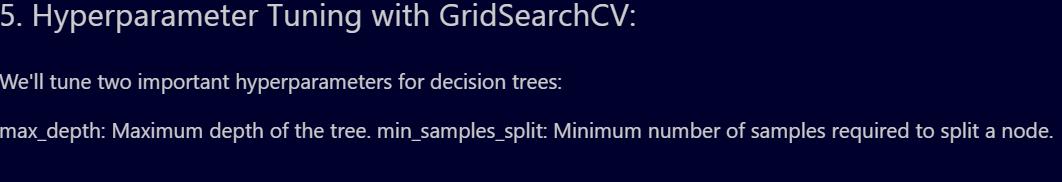
M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

35



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

* **Define the decision tree classifier clf = DecisionTreeClassifier()**



* **Define hyperparameter grid param\_grid = {**

**'max\_depth': [2, 3, 4, 5],**

**'min\_samples\_split': [2, 5, 10]**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

36

**}**

**# Create GridSearchCV object**

**grid\_clf = GridSearchCV(clf, param\_grid, scoring='accuracy')**

**# Train the model**

**grid\_clf.fit(X\_train, y\_train)**

**# Get the best model**

**best\_model = grid\_clf.best\_estimator\_**

**# Print the best hyperparameters**

**print("Best Hyperparameters:", grid\_clf.best\_params\_)**



**# Predict on test set**

**y\_pred = best\_model.predict(X\_test)**

**# Calculate accuracy**

**from sklearn.metrics import accuracy\_score accuracy = accuracy\_score(y\_test, y\_pred) print("Test Accuracy:", accuracy)**

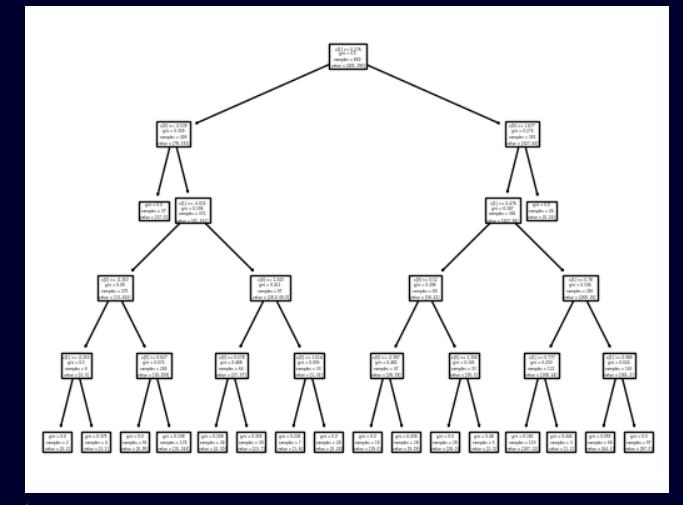
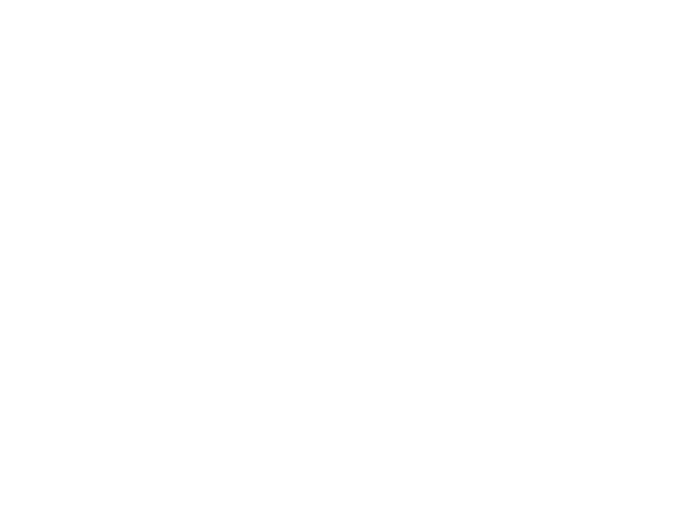


**#**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

37

* **Visualize the decision tree plot\_tree(best\_model) plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

38

**Practical No 6**

**Aim: Train an SVM regression on the California Housing Dataset**

**Theory:**

**Support Vector Regression** is an extension of SVM which introduces a region, named tube, around the function to optimize with the aim of finding the tube that best approximates the continuous-valued function, while minimizing the prediction error, that is, the difference between the predicted and the true class label.

**CODE:**

**import pandas as pd**

**from sklearn.datasets import fetch\_california\_housing from sklearn.preprocessing import StandardScaler from sklearn.model\_selection import train\_test\_split from sklearn.svm import SVR**

**from sklearn.metrics import mean\_squared\_error, r2\_score import seaborn as sns # For visualization**

**import matplotlib.pyplot as plt # For visualization**

**# Load data**

**data = fetch\_california\_housing()**

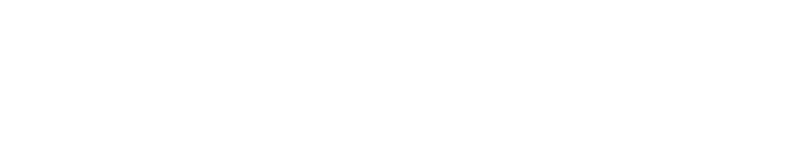
**X = pd.DataFrame(data.data, columns=data.feature\_names)**

**y = data.target**

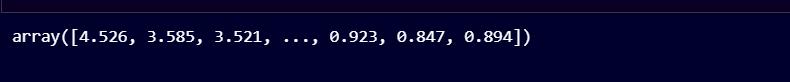
M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

39

**X.columns**

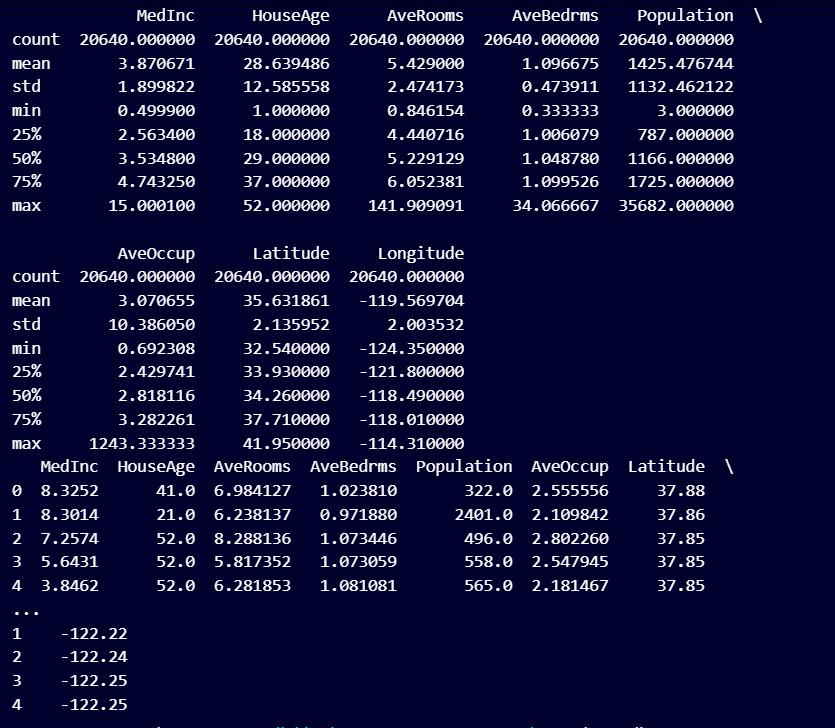


**y**



* **Print basic info about the data print(X.describe()) # Summary statistics**

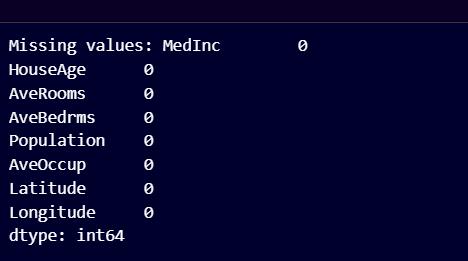
**print(X.head())** **# View first few rows**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

40

* **Check for missing values print("Missing values:", X.isnull().sum())**

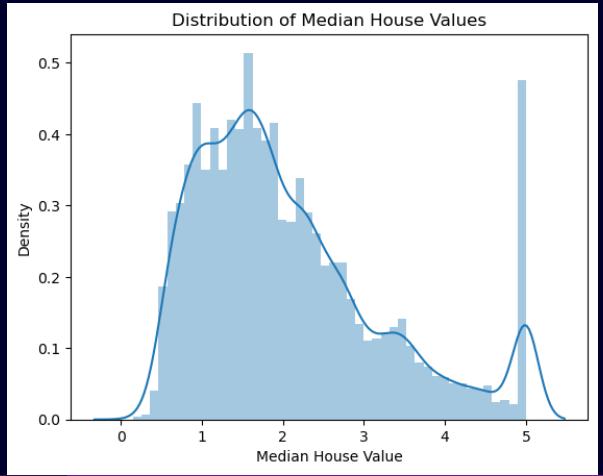


* **Exploratory Data Analysis (EDA)**
* **Visualize the distribution of the target variable (median house value) sns.distplot(y)**

**plt.xlabel("Median House Value") plt.ylabel("Density") plt.title("Distribution of Median House Values") plt.show()**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

41



* **Data Preprocessing**
* **Scale features (SVM regressor is sensitive to feature scales) scaler = StandardScaler()**

**X\_scaled = scaler.fit\_transform(X)**

* **Split Data into Training and Testing Sets**

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

**# Train the SVM Regressor**

**svr = SVR(kernel='rbf') # Experiment with 'linear' or other kernels svr.fit(X\_train, y\_train)**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

42

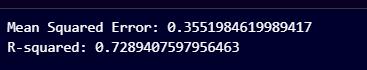
* **Make Predictions and Evaluate Performance y\_pred = svr.predict(X\_test)**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**r2 = r2\_score(y\_test, y\_pred)**

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

**print("R-squared:", r2)**



**from sklearn.model\_selection import GridSearchCV from sklearn.svm import SVR**

* **Define a parameter grid to explore param\_grid = {**

**'kernel': ['linear', 'rbf'], # Experiment with different kernels**

**'C': [0.01, 0.1, 1, 10, 100], # Regularization parameter**

**'gamma': [0.001, 0.01, 0.1, 1], # Gamma for RBF kernel (optional)**

**}**

* **Create the GridSearchCV object**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

43

**grid\_search = GridSearchCV(SVR(), param\_grid, cv=5) # 5-fold cross-validation**

* **Fit the grid search to the training data grid\_search.fit(X\_train, y\_train)**
* **Get the best model and its parameters best\_model = grid\_search.best\_estimator\_ best\_params = grid\_search.best\_params\_**
* **Use the best model for prediction and evaluation y\_pred = best\_model.predict(X\_test)**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**r2 = r2\_score(y\_test, y\_pred)**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

44

**Practical No 7**

**Aim: Implement Batch Gradient Descent with Early Stopping for Softmax Regression.**

**Theory:**

**Batch gradient descent**, also called vanilla gradient descent, calculates the error for each example within the training dataset, but only after all training examples have been evaluated does the model get updated. This whole process is like a cycle and it's called a training epoch.

In machine learning, **early stopping** is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration.

**SoftMax** is particularly suited for multi-class classification problems, as

it provides a clear and normalized probability distribution across all possible classes.

**CODE:**

**import numpy as np**

**from sklearn import datasets**

**from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import accuracy\_score**

* **Load the Iris dataset iris = datasets.load\_iris() X = iris.data**

**y = iris.target**

* **Standardize the features scaler = StandardScaler()**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

45

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

* **Add a bias term (column of ones) to the data X = np.c\_[np.ones(X.shape[0]), X]**
* **Split the dataset 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)**

**def softmax(logits):**

**exp\_logits = np.exp(logits - np.max(logits, axis=1, keepdims=True))**

**return exp\_logits / np.sum(exp\_logits, axis=1, keepdims=True)**

**def compute\_loss\_and\_gradients(X, y, theta):**

**logits = X.dot(theta)**

**y\_proba = softmax(logits)**

**m = X.shape[0]**

**entropy\_loss = -np.mean(np.log(y\_proba[np.arange(m), y])) gradients = (1/m) \* X.T.dot(y\_proba - np.eye(np.max(y) + 1)[y]) return entropy\_loss, gradients**

**def predict(X, theta):**

**logits = X.dot(theta)**

**return np.argmax(softmax(logits), axis=1)**

**def softmax\_regression(X\_train, y\_train, X\_val, y\_val, learning\_rate=0.01, n\_epochs=1000, tol=1e-4, patience=5):**

**n\_inputs = X\_train.shape[1]**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

46

**n\_outputs = np.max(y\_train) + 1**

**theta = np.random.randn(n\_inputs, n\_outputs)**

**best\_loss = np.inf**

**epochs\_without\_improvement = 0**

**for epoch in range(n\_epochs):**

**loss, gradients = compute\_loss\_and\_gradients(X\_train, y\_train, theta) theta = theta - learning\_rate \* gradients**

**val\_loss, \_ = compute\_loss\_and\_gradients(X\_val, y\_val, theta)**

**if val\_loss < best\_loss - tol:**

**best\_loss = val\_loss**

**epochs\_without\_improvement = 0**

**else:**

**epochs\_without\_improvement += 1**

**if epochs\_without\_improvement >= patience: print(f"Early stopping at epoch {epoch}") break**

**return theta**

**# Split the training data into training and validation sets**

**X\_train\_split, X\_val\_split, y\_train\_split, y\_val\_split =**

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

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

47

**# Train the model**

**theta = softmax\_regression(X\_train\_split, y\_train\_split, X\_val\_split, y\_val\_split)**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

48

**Practical No 8**

**Aim: Implement MLP for Classification of Handwritten digits(MNIST datasets)**

**Theory:**

The **MNIST database** (Modified National Institute of Standards and Technology database) is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of 10,000 examples.

**CODE:**

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.datasets import mnist from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense**

**import matplotlib.pyplot as plt**

**# Load and preprocess the MNIST dataset**

**(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()**

**print(x\_train.shape)**

**x\_train = x\_train / 255.0**

**x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)**

* **Define the CNN model architecture model = Sequential([**

**Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

49

**MaxPooling2D((2, 2)),**

**Conv2D(64, (3, 3), activation='relu'),**

**MaxPooling2D((2, 2)),**

**Flatten(),**

**Dense(128, activation='relu'),**

**Dense(10, activation='softmax')**

**])**

* **Compile the model model.compile(optimizer='adam',**

**loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])**

* **Train the model using the training data model.fit(x\_train, y\_train, epochs=5)**
* **Choose a single image from the test set**

**index = 0 # Replace with the index of the image you want to use single\_image = x\_test[index]**

**input\_image = np.expand\_dims(single\_image, axis=0)**

* **Get the predicted probabilities for the single image predicted\_probabilities = model.predict(input\_image)**
* **Display the input image**

**plt.imshow(single\_image, cmap='gray')**

**plt.title('Input Image')**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

50

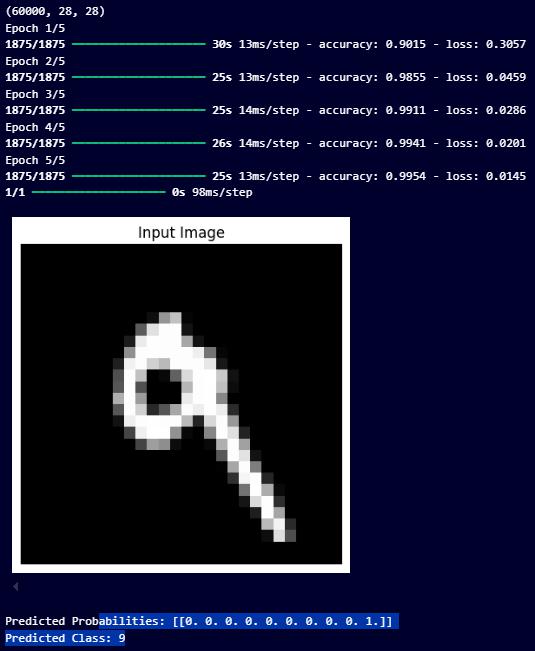
**plt.axis('off')**

**plt.show()**

**# Display the predicted probabilities**

**print("Predicted Probabilities:", predicted\_probabilities)**

* **Get the predicted class (index with highest probability) predicted\_class = np.argmax(predicted\_probabilities) print("Predicted Class:", predicted\_class)**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

51

**Practical No 9**

**Aim: Classification of Image of clothing using Tensorflow(Fashion MNIST dataset)**

**Theory:**

**TensorFlow** is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. It was developed by the Google Brain team for Google's internal use in research and production

**CODE:**

**import tensorflow as tf**

**from tensorflow.keras import datasets, layers, models import matplotlib.pyplot as plt**

**# Load the Fashion MNIST dataset**

**(train\_images, train\_labels), (test\_images, test\_labels) = datasets.fashion\_mnist.load\_data()**

**# Normalize the images to a range of 0 to 1**

**train\_images, test\_images = train\_images / 255.0, test\_images / 255.0**

**model = models.Sequential([**

**layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),**

**layers.MaxPooling2D((2, 2)),**

**layers.Conv2D(64, (3, 3), activation='relu'),**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

52

**layers.MaxPooling2D((2, 2)),**

**layers.Conv2D(64, (3, 3), activation='relu'),**

**layers.Flatten(),**

**layers.Dense(64, activation='relu'),**

**layers.Dense(10, activation='softmax')**

**])**

**model.compile(optimizer='adam',**

**loss='sparse\_categorical\_crossentropy',**

**metrics=['accuracy'])**

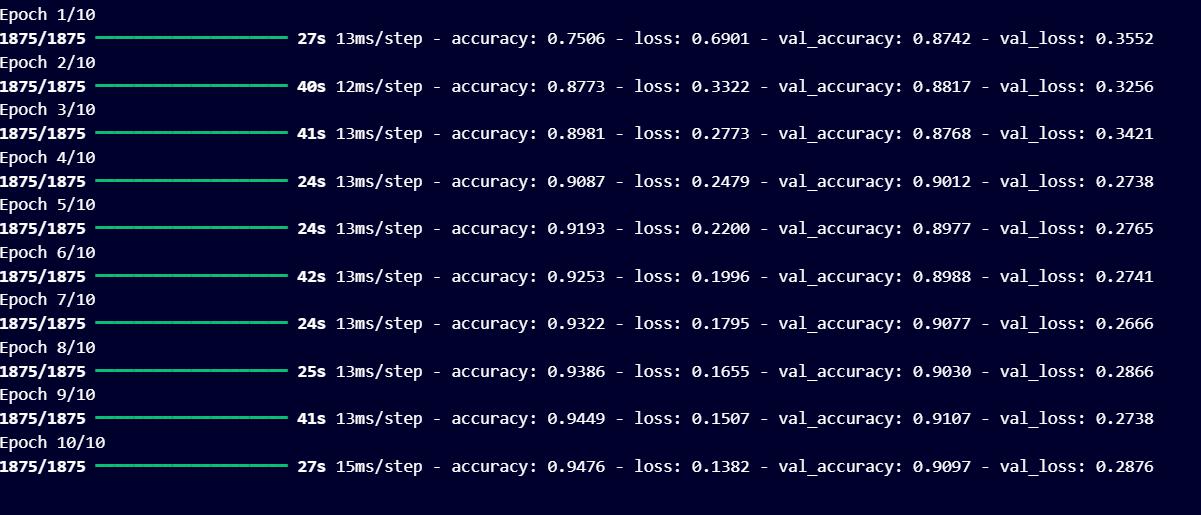
**# Reshape the data to include the channel dimension**

**train\_images = train\_images.reshape((train\_images.shape[0], 28, 28, 1))**

**test\_images = test\_images.reshape((test\_images.shape[0], 28, 28, 1))**

**# Train the model**

**history = model.fit(train\_images, train\_labels, epochs=10, validation\_data=(test\_images, test\_labels))**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

53

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Accuracy')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

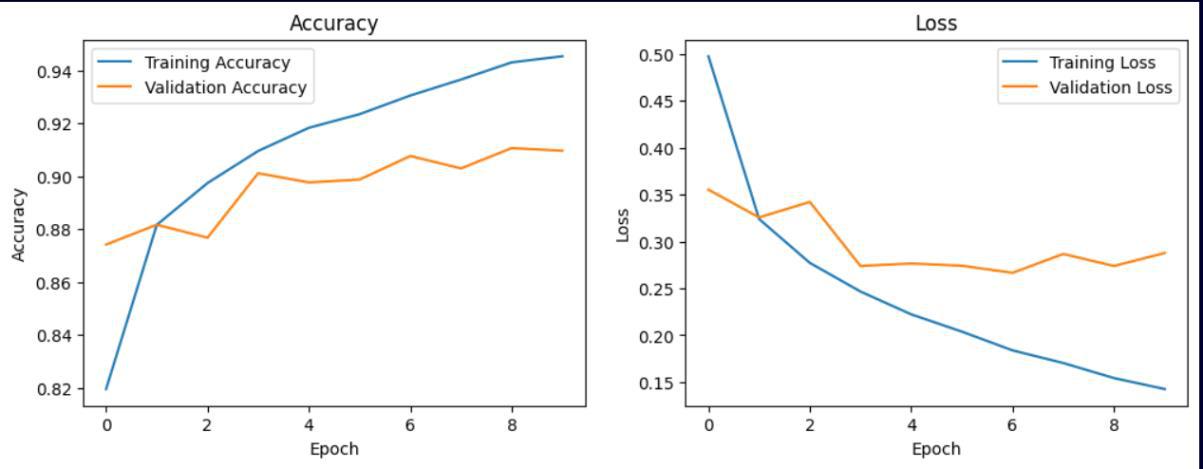
plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Loss')

plt.show()



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

54

**import numpy as np**

**predictions = model.predict(test\_images)**

* **Define a function to plot the images and predictions def plot\_image(predictions\_array, true\_label, img):**

**plt.grid(False)**

**plt.xticks([])**

**plt.yticks([])**

**plt.imshow(img, cmap=plt.cm.binary)**

**predicted\_label = np.argmax(predictions\_array)**

**if predicted\_label == true\_label:**

**color = 'blue'**

**else:**

**color = 'red'**

**plt.xlabel(f"{class\_names[predicted\_label]} {100\*np.max(predictions\_array):2.0f}% ({class\_names[true\_label]})", color=color)**

**def plot\_value\_array(predictions\_array, true\_label):**

**plt.grid(False)**

**plt.xticks(range(10))**

**plt.yticks([])**

**thisplot = plt.bar(range(10), predictions\_array, color="#777777")**

**plt.ylim([0, 1])**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

55

**predicted\_label = np.argmax(predictions\_array)**

**thisplot[predicted\_label].set\_color('red')**

**thisplot[true\_label].set\_color('blue')**

**# Define the class names**

**class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']**

* **Plot the first 15 test images, their predicted labels, and the true labels**
* **Color correct predictions in blue and incorrect predictions in red num\_rows = 5**

**num\_cols = 3**

**num\_images = num\_rows \* num\_cols plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows)) for i in range(num\_images):**

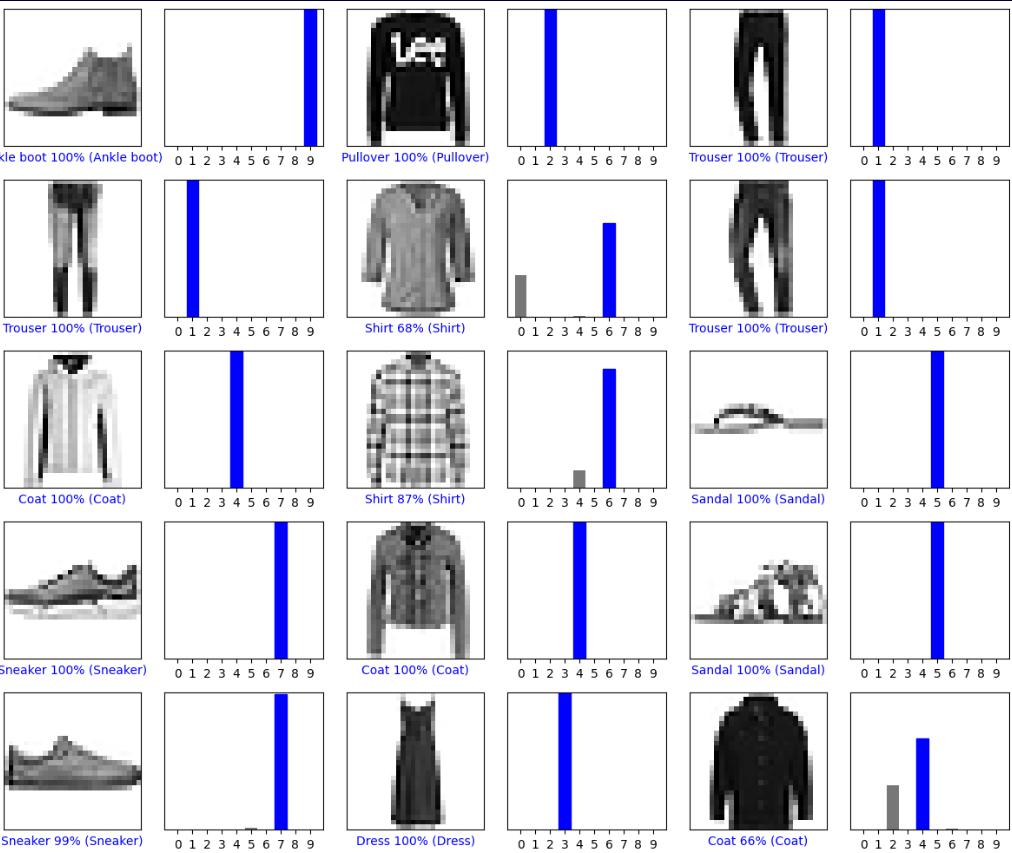
**plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)**

**plot\_image(predictions[i], test\_labels[i], test\_images[i].reshape(28, 28)) plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2) plot\_value\_array(predictions[i], test\_labels[i])**

**plt.tight\_layout() plt.show()**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

56



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

57

**Practical No 10**

**Aim: Implement Regression to predict fuel efficiency using TensorFlow (Auto MPG dataset).**

**Theory:**

**Regression** is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between a dependent variable and one or more independent variables. Linear regression is the most common form of this technique.

**TensorFlow** is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. It was developed by the Google Brain team for Google's internal use in research and production

**CODE:**

**import pandas as pd**

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras import layers, models**

**from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler import matplotlib.pyplot as plt**

**# Load the dataset**

**url = "auto-mpg.csv" # Replace with your CSV file path**

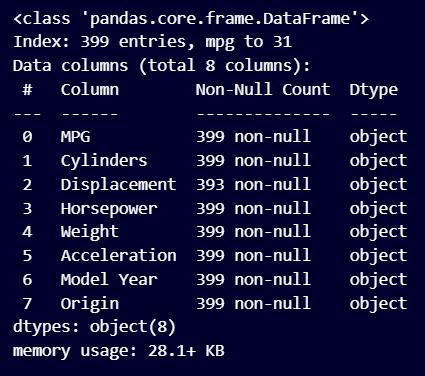
**column\_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight', 'Acceleration', 'Model Year', 'Origin']**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

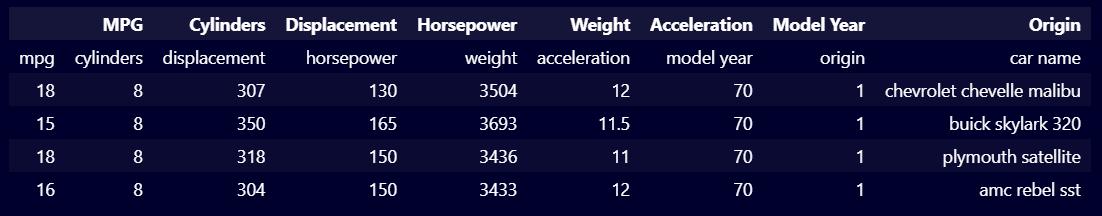
58

**dataset = pd.read\_csv(url, names=column\_names, na\_values='?', comment='\t', sep=',', skipinitialspace=True)**

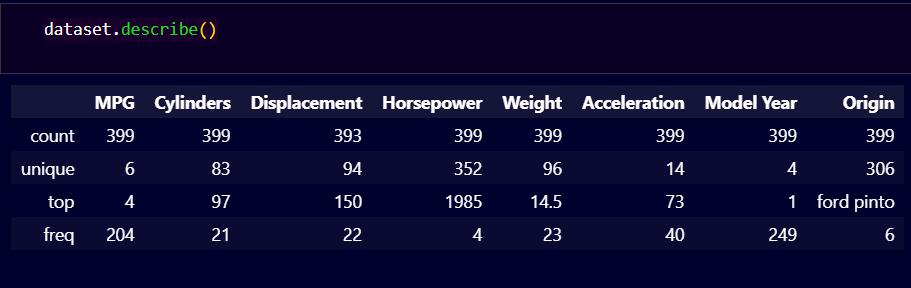
**dataset.info()**



**dataset.head()**



**dataset.describe()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

59

* **Drop rows with missing values dataset = dataset.dropna()**
* **Convert columns to appropriate numeric data types**

**for column in ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight', 'Acceleration', 'Model Year']:**

**dataset[column] = pd.to\_numeric(dataset[column], errors='coerce')**

**# Check for NaN values in the dataset**

**print("NaN values before dropping: \n", dataset.isnull().sum())**

* **Drop any rows with NaN values dataset = dataset.dropna()**
* **Check again for NaN values to confirm**

**print("NaN values after dropping: \n", dataset.isnull().sum())**

* **Convert 'Origin' to string for one-hot encoding dataset['Origin'] = dataset['Origin'].astype(str)**
* **Convert categorical 'Origin' column to one-hot encoding**

**dataset = pd.get\_dummies(dataset, columns=['Origin'], prefix='', prefix\_sep='')**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

60



**# Separate features and labels**

**train\_features = train\_dataset.copy()**

**test\_features = test\_dataset.copy()**

**train\_labels = train\_features.pop('MPG')**

**test\_labels = test\_features.pop('MPG')**

**# Check for NaN values in the dataset**

**assert not train\_features.isnull().any().any(), "There are NaN values in the training features"**

**assert not test\_features.isnull().any().any(), "There are NaN values in the test features"**

**assert not train\_labels.isnull().any(), "There are NaN values in the training labels"**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

61

**assert not test\_labels.isnull().any(), "There are NaN values in the test labels"**

* **Normalize the features scaler = StandardScaler()**

**train\_features = scaler.fit\_transform(train\_features) test\_features = scaler.transform(test\_features)**

**def build\_model():**

**model = models.Sequential([**

**layers.Dense(64, activation='relu', input\_shape=[train\_features.shape[1]]),**

**layers.Dense(64, activation='relu'),**

**layers.Dense(1)**

**])**

**return model**

**model = build\_model()**

**model.compile(optimizer='adam',**

**loss='mse',**

**metrics=['mae', 'mse'])**

**history = model.fit(train\_features, train\_labels, epochs=100, validation\_split=0.2, verbose=0)**

**# Plot training history**

**hist = pd.DataFrame(history.history)**

**hist['epoch'] = history.epoch**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

62

**plt.figure(figsize=(12, 4))**

**plt.subplot(1, 2, 1)**

**plt.xlabel('Epoch')**

**plt.ylabel('Mean Abs Error [MPG]')**

**plt.plot(hist['epoch'], hist['mae'], label='Train Error')**

**plt.plot(hist['epoch'], hist['val\_mae'], label='Val Error')**

**plt.legend()**

**plt.subplot(1, 2, 2)**

**plt.xlabel('Epoch')**

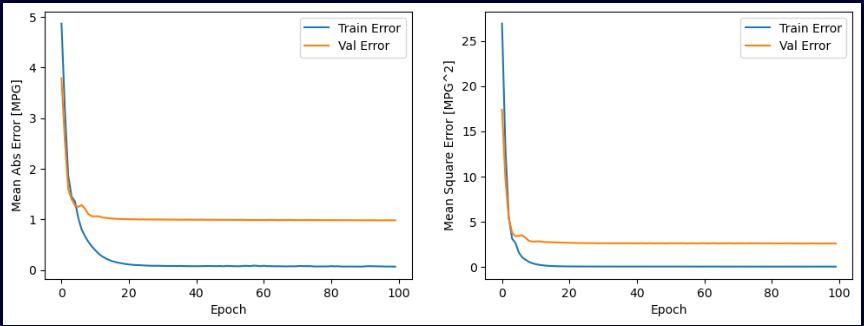
**plt.ylabel('Mean Square Error [MPG^2]')**

**plt.plot(hist['epoch'], hist['mse'], label='Train Error')**

**plt.plot(hist['epoch'], hist['val\_mse'], label='Val Error')**

**plt.legend()**

**plt.show()**

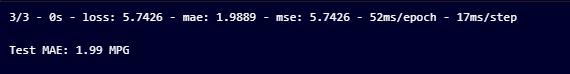


M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

63

**test\_loss, test\_mae, test\_mse = model.evaluate(test\_features, test\_labels, verbose=2)**

**print(f'\nTest MAE: {test\_mae:.2f} MPG')**



**test\_predictions = model.predict(test\_features).flatten()**

**plt.scatter(test\_labels, test\_predictions)**

**plt.xlabel('True Values [MPG]')**

**plt.ylabel('Predictions [MPG]')**

**plt.axis('equal')**

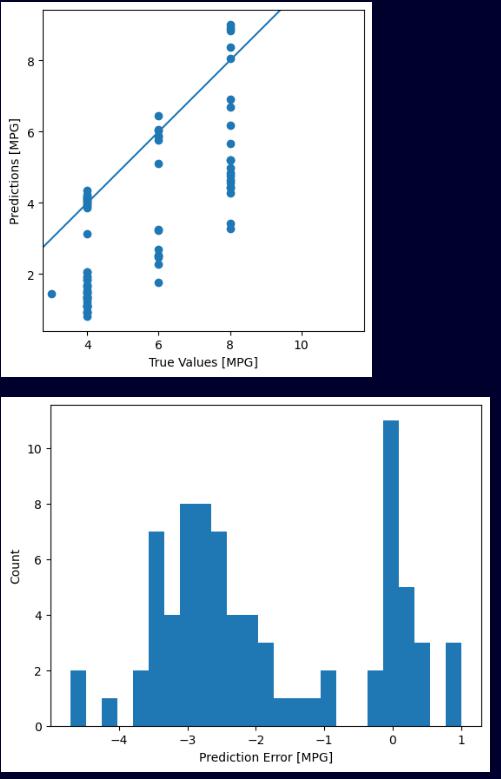
**plt.axis('square')**

**plt.plot([-100, 100], [-100, 100])**

M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

64

**plt.show()**



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024

65

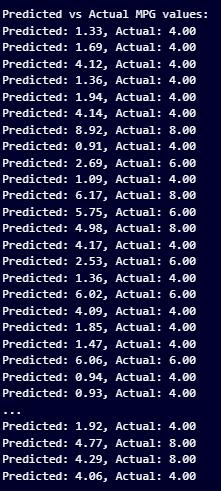
error = test\_predictions - test\_labels

plt.hist(error, bins=25)

plt.xlabel('Prediction Error [MPG]')

plt.ylabel('Count')

plt.show()



M.Sc. Computer Science – Semester II (NEP 2020) Machine Learning JOURNAL-2023-2024