| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| 1. **Install and set up Python and essential libraries like NumPy and pandas.**   **Installing Python:**   * 1. **Download Python**: Go to the official Python website download the latest version suitable for your operating system (Windows, macOS, or Linux).   2. **Install Python**:      + For Windows: Run the downloaded installer and make sure to check the box that says "Add Python x.x to PATH" during installation.      + For Linux: Python might already be installed. If not, use your package manager to install it (e.g., **sudo apt-get install python3** for Ubuntu).   3. **Verify Installation**:      + Open a command prompt (Windows) or terminal (macOS/Linux).      + Type **python --version** or **python3 --version** and press Enter. You should see the installed Python version.   **Installing NumPy and pandas:**   1. **Using pip**:    * Pip is Python's package manager. It usually comes installed with Python. Open a terminal/command prompt. 2. **Install NumPy**:    * Type **pip install numpy** and press Enter. This command will download and install NumPy. 3. **Install pandas**:    * Type **pip install pandas** and press Enter. This will download and install pandas. 4. **Verify installations**:    * Open a Python interpreter by typing **python** or **python3** in the terminal.    * Inside the interpreter, import NumPy and pandas: import numpy   import pandas  If no errors occur, both libraries are installed correctly. | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 1 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| 1. **Introduce scikit-learn as a machine learning library.**   Scikit-learn is a widely-used Python library that provides a comprehensive suite of tools and functionalities for machine learning tasks.   * 1. **Versatility**: Scikit-learn offers a wide range of tools and functionalities for various machine learning tasks, including but not limited to:      + **Supervised Learning**: Classification, Regression      + **Unsupervised Learning**: Clustering, Dimensionality Reduction      + **Model Selection and Evaluation**: Cross-validation, Hyperparameter Tuning      + **Preprocessing**: Data cleaning, Feature Engineering   2. **Consistent Interface**: It provides a consistent and user-friendly API, making it easy to experiment with different algorithms and techniques without needing to learn new syntax for each.   3. **Integration with Other Libraries**: Scikit-learn seamlessly integrates with other Python libraries like NumPy, pandas, and Matplotlib, allowing smooth data manipulation, preprocessing, and visualization.   4. **Ease of Learning**: Its well-documented and straightforward interface makes it suitable for both beginners and experienced machine learning practitioners. It's often recommended for educational purposes due to its simplicity.   5. **Performance and Scalability**: While focusing on simplicity, scikit-learn also emphasizes performance. It's optimized for efficiency and scalability, making it suitable for handling large datasets and complex models.   6. **Community and Development**: As an open-source project, scikit-learn benefits from a vibrant community of developers and contributors. Regular updates, bug fixes, and enhancements ensure it stays relevant and up-to-date with the latest advancements in machine learning.   7. **Application in Industry and Academia**: Scikit-learn's robustness and ease of use have made it a go-to choose in various domains, including finance, healthcare, natural language processing, and more. It's widely used in research and production environments. | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 2 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| **Example:**  from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier from sklearn import metrics  # Load Iris dataset (a popular example dataset in machine learning) iris = load\_iris()  X = iris.data # Features  y = iris.target # Target variable  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3) # Initialize the model (K-Nearest Neighbors Classifier in this case) model = KNeighborsClassifier(n\_neighbors=3)  # Train the model model.fit(X\_train, y\_train) # Make predictions  predictions = model.predict(X\_test) # Evaluate model accuracy  accuracy = metrics.accuracy\_score(y\_test, predictions) print(f"Accuracy: {accuracy}")  OUTPUT:  Accuracy: 0.9333333333333333 | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 3 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| 1. **Install and set up scikit-learn and other necessary tools. Procedure**   **Install scikit-learn and other libraries:**   * 1. **Install NumPy and pandas** (if not installed):      + Open a terminal/command prompt.      + Type **pip install numpy pandas** and press Enter. This will install NumPy and pandas, essential libraries for data manipulation and computation.   2. **Install scikit-learn**:      + In the same terminal/command prompt, type **pip install scikit-learn** and press Enter. This will install scikit-learn, the machine learning library.   **Test the installations:**   * Open a Python interpreter by typing **python** or **python3** in the terminal. * Import scikit-learn: **import sklearn** * If there are no errors, scikit-learn is successfully installed and ready for use.   **Simple Example**  from sklearn import datasets  from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier # Load an example dataset (iris dataset)  iris = datasets.load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3)  # Initialize and train a classifier (K-Nearest Neighbors) clf = KNeighborsClassifier(n\_neighbors=3) clf.fit(X\_train, y\_train)  # Evaluate the classifier  accuracy = clf.score(X\_test, y\_test) print(f"Accuracy: {accuracy}")  OUTPUT:  Accuracy: 0.9777777777777777 | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 4 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| 1. **Write a program to Load and explore the dataset of .CVS and excel files using pandas.**   **Load and Explore CSV File:**  *import pandas as pd # Load CSV file*  *csv\_data = pd.read\_csv('train.csv')*  *# Display the first few rows of the CSV file print("First few rows of CSV file:") print(csv\_data.head())*  *# Summary statistics*  *print("\nSummary statistics of CSV file:") print(csv\_data.describe())*  *# Information about columns*  *print("\nInformation about columns in CSV file:") print(csv\_data.info())*  *Output:*  *First few rows of CSV file: PassengerId Survived Pclass \*  *0 1 0 3*  *1 2 1 1*  *2 3 1 3*  *3 4 1 1*  *4 5 0 3*  Name Sex Age SibSp \   * 1. Braund, Mr. Owen Harris male 22.0 1   2. Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   3. Heikkinen, Miss. Laina female 26.0 0   4. Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   5. Allen, Mr. William Henry male 35.0 0   Parch Ticket Fare Cabin Embarked  0 0 A/5 21171 7.2500 NaN S  1 0 PC 17599 71.2833 C85 C  2 0 STON/O2. 3101282 7.9250 NaN S  3 0 113803 53.1000 C123 S  4 0 373450 8.0500 NaN S | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 5 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| Summary statistics of CSV file: | | |
| PassengerId Survived Pclass Age SibSp \ | | |
| count 891.000000 891.000000 891.000000 714.000000 891.000000 | | |
| mean 446.000000 0.383838 2.308642 29.699118 0.523008 | | |
| std 257.353842 0.486592 0.836071 14.526497 1.102743 | | |
| min 1.000000 0.000000 1.000000 0.420000 0.000000 | | |
| 25% 223.500000 0.000000 2.000000 20.125000 0.000000 | | |
| 50% 446.000000 0.000000 3.000000 28.000000 0.000000 | | |
| 75% 668.500000 1.000000 3.000000 38.000000 1.000000 | | |
| max 891.000000 1.000000 3.000000 80.000000 8.000000 | | |
| Parch Fare | | |
| count 891.000000 891.000000 | | |
| mean 0.381594 32.204208 | | |
| std 0.806057 49.693429 | | |
| min 0.000000 0.000000 | | |
| 25% 0.000000 7.910400 | | |
| 50% 0.000000 14.454200 | | |
| 75% 0.000000 31.000000 | | |
| max 6.000000 512.329200 | | |
| Information about columns in CSV file: | | |
| <class 'pandas.core.frame.DataFrame'> | | |
| RangeIndex: 891 entries, 0 to 890 | | |
| Data columns (total 12 columns): | | |
| # Column Non-Null Count Dtype | | |
| 0 PassengerId 891 non-null int64 | | |
| 1 Survived 891 non-null int64 | | |
| 2 Pclass 891 non-null int64 | | |
| 3 Name 891 non-null object | | |
| 4 Sex 891 non-null object | | |
| 5 Age 714 non-null float64 | | |
| 6 SibSp 891 non-null int64 | | |
| 7 Parch 891 non-null int64 | | |
| 8 Ticket 891 non-null object | | |
| 9 Fare 891 non-null float64 | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 6 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| 1. Cabin 204 non-null object 2. Embarked 889 non-null object dtypes: float64(2), int64(5), object(5) memory usage: 83.7+ KB   None  **Load and Explore Excel File:**  import pandas as pd # Load Excel file  excel\_data = pd.read\_excel('Sample - Superstore.xlsx', sheet\_name='Orders') # Display the first few rows of the Excel file  print("First few rows of Excel file:") print(excel\_data.head())  # Summary statistics  print("\nSummary statistics of Excel file:") print(excel\_data.describe())  # Information about columns  print("\nInformation about columns in Excel file:") print(excel\_data.info())  Output:  First few rows of Excel file:  Row ID Order ID Order Date Ship Date Ship Mode Customer ID \  0 1 CA-2016-152156 2016-11-08 2016-11-11 Second Class CG-  12520  1 2 CA-2016-152156 2016-11-08 2016-11-11 Second Class CG-  12520  2 3 CA-2016-138688 2016-06-12 2016-06-16 Second Class DV-  13045 | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 7 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| 3 4 US-2015-108966 2015-10-11 2015-10-18 Standard Class SO-  20335  4 5 US-2015-108966 2015-10-11 2015-10-18 Standard Class SO-  20335  Customer Name Segment Country City ... \   1. Claire Gute Consumer United States Henderson ... 2. Claire Gute Consumer United States Henderson ... 3. Darrin Van Huff Corporate United States Los Angeles ... 4. Sean O'Donnell Consumer United States Fort Lauderdale ... 5. Sean O'Donnell Consumer United States Fort Lauderdale ...   Postal Code Region Product ID Category Sub-Category \   1. 42420 South FUR-BO-10001798 Furniture Bookcases 2. 42420 South FUR-CH-10000454 Furniture Chairs 3. 90036 West OFF-LA-10000240 Office Supplies Labels 4. 33311 South FUR-TA-10000577 Furniture Tables 5. 33311 South OFF-ST-10000760 Office Supplies Storage   Product Name Sales Quantity \   1. Bush Somerset Collection Bookcase 261.9600 2 2. Hon Deluxe Fabric Upholstered Stacking Chairs,... 731.9400 3 3. Self-Adhesive Address Labels for Typewriters b... 14.6200 2 4. Bretford CR4500 Series Slim Rectangular Table 957.5775 5 5. Eldon Fold 'N Roll Cart System 22.3680 2   Discount Profit  0 0.00 41.9136  1 0.00 219.5820  2 0.00 6.8714  3 0.45 -383.0310  4 0.20 2.5164  [5 rows x 21 columns] Summary statistics of Excel file: | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 8 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| Row ID Order Date \ | | |
| count 9994.000000 9994 | | |
| mean 4997.500000 2016-04-30 00:07:12.259355648 | | |
| min 1.000000 2014-01-03 00:00:00 | | |
| 25% 2499.250000 2015-05-23 00:00:00 | | |
| 50% 4997.500000 2016-06-26 00:00:00 | | |
| 75% 7495.750000 2017-05-14 00:00:00 | | |
| max 9994.000000 2017-12-30 00:00:00 | | |
| std 2885.163629 NaN | | |
| Ship Date Postal Code Sales Quantity \ | | |
| count 9994 9994.000000 9994.000000 9994.000000 | | |
| mean 2016-05-03 23:06:58.571142912 55190.379428 229.858001 | | |
| 3.789574 | | |
| min 2014-01-07 00:00:00 1040.000000 0.444000 1.000000 | | |
| 25% 2015-05-27 00:00:00 23223.000000 17.280000 2.000000 | | |
| 50% 2016-06-29 00:00:00 56430.500000 54.490000 3.000000 | | |
| 75% 2017-05-18 00:00:00 90008.000000 209.940000 5.000000 | | |
| max 2018-01-05 00:00:00 99301.000000 22638.480000 | | |
| 14.000000 | | |
| std NaN 32063.693350 623.245101 2.225110 | | |
| Discount Profit | | |
| count 9994.000000 9994.000000 | | |
| mean 0.156203 28.656896 | | |
| min 0.000000 -6599.978000 | | |
| 25% 0.000000 1.728750 | | |
| 50% 0.200000 8.666500 | | |
| 75% 0.200000 29.364000 | | |
| max 0.800000 8399.976000 | | |
| std 0.206452 234.260108 | | |
| Information about columns in Excel file: | | |
| <class 'pandas.core.frame.DataFrame'> | | |
| RangeIndex: 9994 entries, 0 to 9993 | | |
| Data columns (total 21 columns): | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 9 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| # Column Non-Null Count Dtype     1. Row ID 9994 non-null int64 2. Order ID 9994 non-null object 3. Order Date 9994 non-null datetime64[ns] 4. Ship Date 9994 non-null datetime64[ns] 5. Ship Mode 9994 non-null object 6. Customer ID 9994 non-null object 7. Customer Name 9994 non-null object 8. Segment 9994 non-null object 9. Country 9994 non-null object 10. City 9994 non-null object 11. State 9994 non-null object 12. Postal Code 9994 non-null int64 13. Region 9994 non-null object 14. Product ID 9994 non-null object 15. Category 9994 non-null object 16. Sub-Category 9994 non-null object 17. Product Name 9994 non-null object 18. Sales 9994 non-null float64 19. Quantity 9994 non-null int64 20. Discount 9994 non-null float64 21. Profit 9994 non-null float64   dtypes: datetime64[ns](2), float64(3), int64(3), object(13) memory usage: 1.6+ MB  None | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 10 | 23 |

| BCA VI SEM |  | ML-LAB MANUAL (NEP) |
| --- | --- | --- |
| **5. Write a program to Visualize the dataset to gain insights using Matplotlib or Seaborn by plotting scatter plots, bar charts.**  import matplotlib.pyplot as plt import seaborn as sns  from sklearn.datasets import load\_iris import pandas as pd  # Load the Iris dataset iris = load\_iris()  # Convert the dataset to a pandas DataFrame  iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names) iris\_df['target'] = iris.target  # Scatter plot using Matplotlib plt.figure(figsize=(8, 6))  plt.scatter(iris\_df['sepal length (cm)'], iris\_df['sepal width (cm)'], c=iris\_df['target'], cmap='viridis', s=80, alpha=0.7) plt.xlabel('Sepal Length (cm)')  plt.ylabel('Sepal Width (cm)')  plt.title('Scatter Plot of Sepal Length vs Sepal Width') plt.colorbar(label='Species')  plt.show()  # Bar chart using Seaborn plt.figure(figsize=(8, 6))  sns.countplot(x='target', data=iris\_df, palette='viridis') plt.xlabel('Species')  plt.ylabel('Count')  plt.title('Bar Chart: Count of Each Species') plt.show() | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | P a g e 11 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| Output: |
| S H I V A S W A M Y D S  A S S I S T A N T P R O F E S S O R  D E P A R T M E N T O F C O M P U E T R S C I E N C E  S H E S H A D R I P U R A M C O L L E G E B - 2 0 P a g e 12 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| **6. Write a program to Handle missing data, encode categorical variables, and perform feature scaling.**  import pandas as pd  from sklearn.impute import SimpleImputer  from sklearn.preprocessing import LabelEncoder, OneHotEncoder,  StandardScaler  # Sample DataFrame with missing values and categorical variables data = {  'A': [1, 2, None, 4, 5],  'B': ['X', None, 'Y', 'Z', 'X'], 'C': [7, 8, 9, None, 11]  }  df = pd.DataFrame(data) print("DataSet:\n",df)  # Handling missing values using SimpleImputer from scikit-learn print("\nHandling missing Values\n") print(" \n")  imputer = SimpleImputer(strategy='mean') # Other strategies: median, most\_frequent, constant  df[['A', 'C']] = imputer.fit\_transform(df[['A', 'C']])  print("DataSet after handling Missing Values of A and C Columns:\n",df[['A', 'C']])  # Encoding categorical variables using LabelEncoder and OneHotEncoder print("\nEncoding\n") print(" \n")  label\_encoder = LabelEncoder()  df['B'] = df['B'].fillna('Unknown') # Handle NaNs before label encoding print("\nDataSet after handling Missing Values of B Before Label encoding:\n", df['B'])  df['B\_encoded'] = label\_encoder.fit\_transform(df['B']) print("\nDataSet after handling Missing Values of B After Label encoding:\n", df['B\_encoded']) |
| P a g e 13 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| one\_hot\_encoder = OneHotEncoder()  encoded\_data = one\_hot\_encoder.fit\_transform(df[['B\_encoded']]).toarray() encoded\_df = pd.DataFrame(encoded\_data, columns=[f'B\_{i}' for i in range(encoded\_data.shape[1])])  df1 = pd.concat([df, encoded\_df], axis=1) print("DataSet after handling Missing Values of B After one\_hot\_encoder:\n",df1)  # Feature scaling using StandardScaler from scikit-learn print("\nFeature scaling\n") print(" \n")  scaler = StandardScaler()  scaled\_data = scaler.fit\_transform(df[['A', 'C']])  scaled\_df = pd.DataFrame(scaled\_data, columns=['A\_scaled', 'C\_scaled']) df2 = pd.concat([df, scaled\_df], axis=1)  print("Feature Scaling using Standard scaler\n", df2)  Output:  DataSet:  A B C  0 1.0 X 7.0  1 2.0 None 8.0   1. NaN Y 9.0 2. 4.0 Z NaN 3. 5.0 X 11.0   Handling missing Values  ....................................................................  DataSet after handling Missing Values of A and C Columns: A C  0 1.0 7.00  1 2.0 8.00  2 3.0 9.00  3 4.0 8.75  4 5.0 11.00 |
| P a g e 14 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| Encoding  ....................................................................  DataSet after handling Missing Values of B Before Label encoding:   1. X 2. Unknown 3. Y 4. Z 5. X   Name: B, dtype: object  DataSet after handling Missing Values of B After Label encoding:  0 1  1 0  2 2  3 3  4 1  Name: B\_encoded, dtype: int32  DataSet after handling Missing Values of B After one\_hot\_encoder: A B C B\_encoded B\_0 B\_1 B\_2 B\_3  0 1.0 X 7.00 1 0.0 1.0 0.0 0.0  1 2.0 Unknown 8.00 0 1.0 0.0 0.0 0.0  2 3.0 Y 9.00 2 0.0 0.0 1.0 0.0  3 4.0 Z 8.75 3 0.0 0.0 0.0 1.0  4 5.0 X 11.00 1 0.0 1.0 0.0 0.0  Feature scaling  ....................................................................  Feature Scaling using Standard scaler  A B C B\_encoded A\_scaled C\_scaled  0 1.0 X 7.00 1 -1.414214 -1.322876  1 2.0 Unknown 8.00 0 -0.707107 -0.566947  2 3.0 Y 9.00 2 0.000000 0.188982  3 4.0 Z 8.75 3 0.707107 0.000000  4 5.0 X 11.00 1 1.414214 1.700840 |
| P a g e 15 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| **7. Write a program to implement a k-Nearest Neighbours (k-NN) classifier using scikitlearn and Train the classifier on the dataset and evaluate its performance.**  from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier from sklearn import metrics  # Load the Iris dataset (or any other dataset you want to use) iris = load\_iris()  X = iris.data y = iris.target  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  # Initialize the k-NN classifier  k = 3 # Set the number of neighbors  knn\_classifier = KNeighborsClassifier(n\_neighbors=k)  # Train the classifier on the training data knn\_classifier.fit(X\_train, y\_train)  # Make predictions on the testing data predictions = knn\_classifier.predict(X\_test)  # Evaluate the performance of the classifier  accuracy = metrics.accuracy\_score(y\_test, predictions) print(f"Accuracy: {accuracy}")  # You can also print other evaluation metrics if needed  # For example, classification report and confusion matrix print("Classification Report:") print(metrics.classification\_report(y\_test, predictions)) print("Confusion Matrix:") print(metrics.confusion\_matrix(y\_test, predictions)) |
| P a g e 16 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| Output:  Accuracy: 1.0 Classification Report:  precision recall f1-score support  0 1.00 1.00 1.00 19  1 1.00 1.00 1.00 13  2 1.00 1.00 1.00 13  accuracy 1.00 45  macro avg 1.00 1.00 1.00 45  weighted avg 1.00 1.00 1.00 45  Confusion Matrix: [[19 0 0]  [ 0 13 0]  [ 0 0 13]] |
| P a g e 17 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| **8. Write a program to implement a linear regression model for regression tasks and Train the model on a dataset with continuous target variables.**  import numpy as np  from sklearn.linear\_model import LinearRegression from sklearn.datasets import make\_regression  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_squared\_error, r2\_score import matplotlib.pyplot as plt  # Generate a synthetic dataset  X,y = make\_regression(n\_samples=1000, n\_features=1, noise=20, random\_state=42)  # 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)  # Initialize the Linear Regression model linear\_reg = LinearRegression()  # Train the model on the training data linear\_reg.fit(X\_train, y\_train)  # Make predictions on the testing data predictions = linear\_reg.predict(X\_test)  # Evaluate the model's performance  mse = mean\_squared\_error(y\_test, predictions) r2 = r2\_score(y\_test, predictions)  print("Mean Squared Error (MSE):\n",mse) print("R-squared:\n",r2)  # Plotting the regression line (optional) plt.scatter(X\_test, y\_test, color='blue') plt.plot(X\_test, predictions, color='red', linewidth=3) plt.xlabel('X')  plt.ylabel('y') plt.title('Linear Regression') plt.show() |
| P a g e 18 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| Output:  Mean Squared Error (MSE):  431.59967479663896  R-squared:  0.375734632146025 |
| P a g e 19 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| **9. Write a program to implement a decision tree classifier using scikit- learn and visualize the decision tree and understand its splits.**  from sklearn.datasets import load\_iris  from sklearn.tree import DecisionTreeClassifier, plot\_tree import matplotlib.pyplot as plt  # Load the Iris dataset iris = load\_iris()  X = iris.data y = iris.target  class\_names = [str(name) for name in iris.target\_names]  # Initialize the Decision Tree Classifier decision\_tree = DecisionTreeClassifier()  # Train the classifier on the entire dataset decision\_tree.fit(X, y)  # Visualize the Decision Tree plt.figure(figsize=(12, 8))  plot\_tree(decision\_tree, feature\_names=iris.feature\_names, class\_names=class\_names, filled=True, rounded=True) plt.title("Decision Tree Visualization")  plt.show() |
| P a g e 20 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
|  |
| S H I V A S W A M Y D S  A S S I S T A N T P R O F E S S O R  D E P A R T M E N T O F C O M P U E T R S C I E N C E  S H E S H A D R I P U R A M C O L L E G E B - 2 0 P a g e 21 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
| **10. Write a program to Implement K-Means clustering and Visualize clusters.**  import matplotlib.pyplot as plt  from sklearn.datasets import make\_blobs from sklearn.cluster import KMeans  # Generating synthetic data  X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=1.0, random\_state=42)  # Initialize K-Means with the number of clusters kmeans = KMeans(n\_clusters=4)  # Fit the K-Means model to the data kmeans.fit(X)  # Predict cluster labels cluster\_labels = kmeans.predict(X) # Visualize the clusters plt.figure(figsize=(7,5))  plt.scatter(X[:, 0], X[:, 1], c=cluster\_labels, cmap='viridis', edgecolors='k')  plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], marker='o', s=200, color='red', label='Centroids')  plt.title('K-Means Clustering') plt.xlabel('X')  plt.ylabel('Y') plt.legend() plt.show() |
| P a g e 22 | 23 |

| BCA VI SEM ML-LAB MANUAL (NEP) |
| --- |
|  |
| P a g e 23 | 23 |