

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**CSB4317 – MACHINE LEARNING**

NAME : Constance Xavier S

REGISTER NUMBER : 21113160

SEMESTER : 6th Semester

YEAR : 2024

BRANCH : Computer Science And Technology



**LABORATORY RECORD**

|  |
| --- |
| 21113160 |

**REG NO:**

Name of the lab Machine learning Lab Department Of Computer science and Technology Certified that this is a bonafide record of the work done by SRIDHAR S of 6C Class in the in Machine learning laboratory during the year 2024

……………………………

Signature of

Staff-in-charge

………………………….... …………………………...

Internal Examiner External Examiner

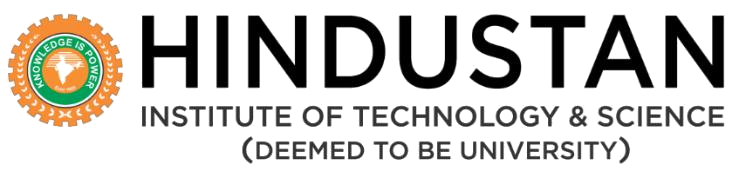
Name of the Examination : Machine Learning Lab Exam

Register No. : 21113160

Date of the Examination : 25.04.2024

**CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| EXP NO | NAME OF THE EXPERIMENT | PAGE NO. | SIGN |
| **1.** | INSTALLATION OF PYTHON LIBRARIES FOR MACHINE LEARNING TOOLS |  |  |
| 2. | DATA PREPROCESSING |  |  |
| 3. | MULTIVARIATE LINEAR REGRESSION |  |  |
| 4. | LOGISTIC REGRESSION |  |  |
| **5.** | KNN |  |  |
| 6. | DECISION TREE |  |  |
| 7. | K-MEAN |  |  |
| 8. | HIERARCHICAL CLUSTER |  |  |
| 9. | SVM |  |  |
| 10. | PCA |  |  |



**PROGRAMME EDUCATIONAL OBJECTIVES**

The B.Tech Programs offered by the Department of Computer Science & Engineering will meet the following objectives:

**PEO 1**. Excel in his/her professional career and/or pursue higher education including research byapplying the knowledge of Computer Science and Engineering

**PEO 2.** Demonstrate the technical skills to analyze and design appropriate solutions for problemswith social consciousness and ethical values.

**PEO 3.** Adapt themselves to organizational needs by understanding the dynamically changingtechnologies.

**PROGRAM SPECIFIC OUTCOMES (PSO):**

On completion of the B.Tech. Computer Science & Engineering degree the graduates will be able to

|  |  |
| --- | --- |
| **PSO1:** | Apply mathematical, conceptual knowledge of computing and analytical skills to solve complex problems. |
| **PSO2:** | Design and develop computer systems based on the domains of Cyber Physical Systems, Algorithm Design Techniques and Enterprise systems security. |
| **PSO3:** | Do innovative system design with analytical knowledge by developing modern tools and techniques. |

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

The B.Tech Programs offered by the Department of Computer Science & Engineering will meet the following outcomes

**PROGRAMME OUTCOMES**

**PO’s 1: Engineering knowledge**: Apply the knowledge of mathematics, science, engineeringfundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO’s 2: Problem analysis**: Identify, formulate, review research literature, and analyse complexengineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO’s 3: Design/development of solutions**: Design solutions for complex engineering problems anddesign system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO’s 4: Conduct investigations of complex problems**: Use research-based knowledge and researchmethods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO’s 5: Modern tool usage**: Create, select, and apply appropriate techniques, resources, modernengineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

**PO’s 6: The engineer and society**: Apply reasoning informed by the contextual knowledge to assesssocietal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO’s 7: Environment and sustainability**: Understand the impact of the professional engineeringsolutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO’s 8: Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and normsof the engineering practice.

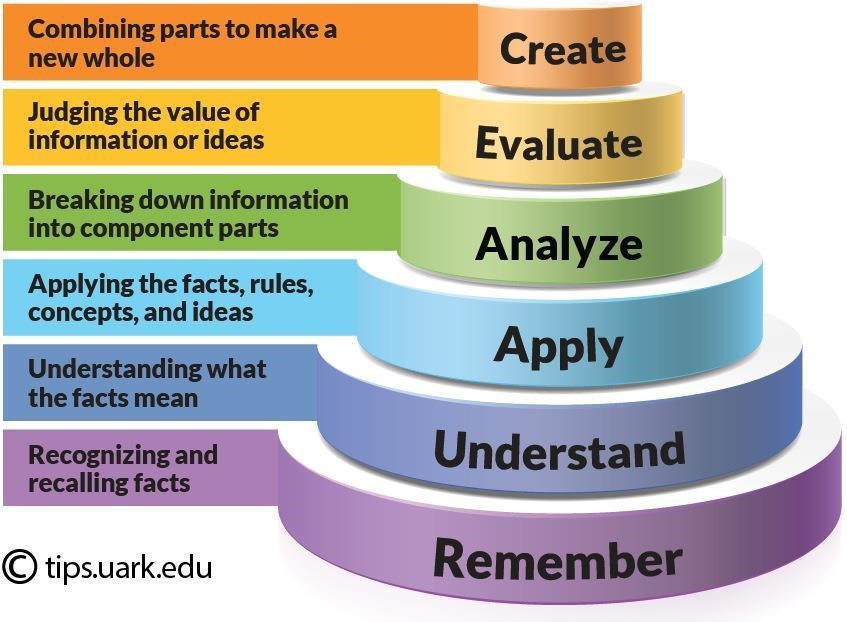
**PO’s 9: Individual and team work**: Function effectively as an individual, and as a member or leader indiverse teams, and in multidisciplinary settings.

**PO’s 10: Communication**: Communicate effectively on complex engineering activities with theengineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO’s 11: Project management and finance**: Demonstrate knowledge and understanding of theengineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO’s 12: Life-long learning**: Recognize the need for, and have the preparation and ability to engage inindependent and life-long learning in the broadest context of technological change.

**THE BLOOM’S TAXONOMY**



**EXP NO:1 INSTALLATION OF PYTHON LIBRARY IN ML**

**DATE:**

**AIM:**

Installation of Python library for machine learning tools.

**PROCEDURE:**

**Step 1:** Download the Anaconda installer.

**Step 2:** Go to your Downloads folder and double-click the installer to launch.

**Step 3:** Click Next.

**Step 4:** Read the licensing terms and click I Agree.

**Step 5:** It is recommended that you install Just Me, which will install Anaconda Distribution to just the current user account. Only select an install for All Users if you need to install for all users’ accounts on the computer (which requires Windows Administrator privileges).

**Step 6:** Click Next.

**Step 7:** Select a destination folder to install Anaconda and click Next. Install Anaconda to a directory path that does not contain spaces or Unicode characters.

**Step 8:** Choose whether to add Anaconda to your PATH environment variable or register Anaconda as your default Python. We don’t recommend adding Anaconda to your PATH environment variable, since this can interfere with other software. Unless you plan on installing and running multiple versions of Anaconda or multiple versions of Python, accept the default and leave this box checked. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.

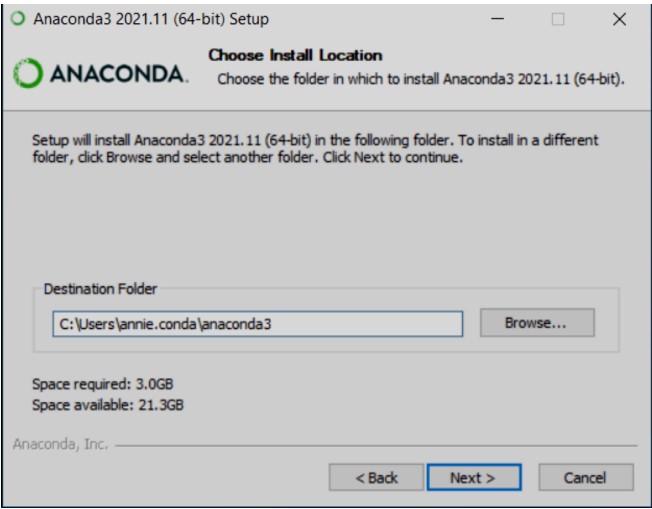
**Step 9:** Click Install. If you want to watch the packages Anaconda is installing, click Show Details.

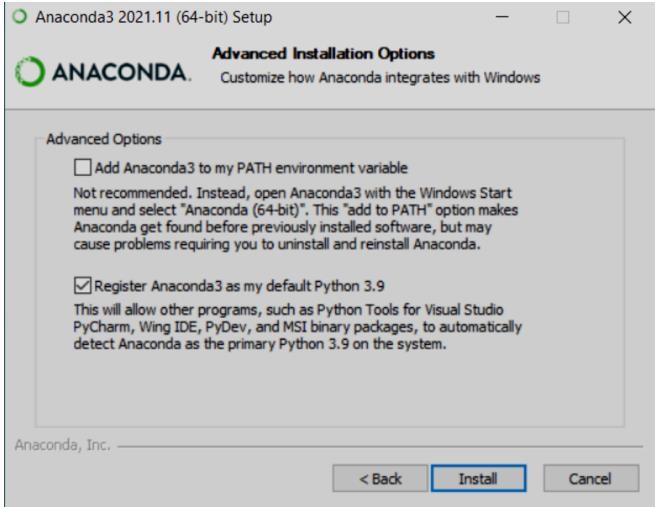
**Step 10:** Click Next.

**Step 11:** After a successful installation you will see the “Thanks for installing Anaconda” dialog box.

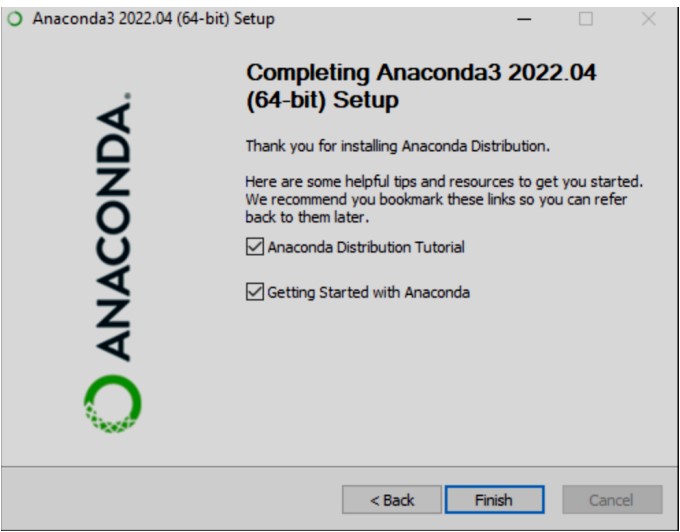
**Step 12:** If you wish to read more about Anaconda.org and how to get started with Anaconda, check the boxes “Anaconda Distribution Tutorial” and “Learn more about Anaconda”. Click the Finish button.

**OUTPUT:**









**RESULT:**

Python libraries for machine learning tools are installed.

**EXP NO:2 DATAPREPROCESSING DATE:**

**AIM:**

The aim is to prepare a dataset for use in a machine learning model. This involves cleaning, transforming, and manipulating the data to make it suitable for the model's needs.

**PROCEDURE:**

1. **Importing Libraries:** Necessary libraries for data manipulation and analysis like Pandas, NumPy, and scikit-learn will be imported.
2. **Data Loading:** The data will be loaded from a source like CSV file, BigQuery, or Google Sheets.
3. **Data Cleaning**: This may involve handling missing values, outliers, and inconsistencies in the data.
4. **Data Transformation:** This could include scaling numerical features, encoding categorical features, and feature engineering (creating new features from existing ones).
5. **Data Splitting:** The data will be split into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate the model's performance**.**

**IMPLEMENTATION CODE:**

import pandas as pd

import numpy as nd

import matplotlib.pyplot as plt

df=pd.read\_csv(r'/content/Heart\_Disease\_Prediction (1).csv)

df

df.head()

df.tail(4)

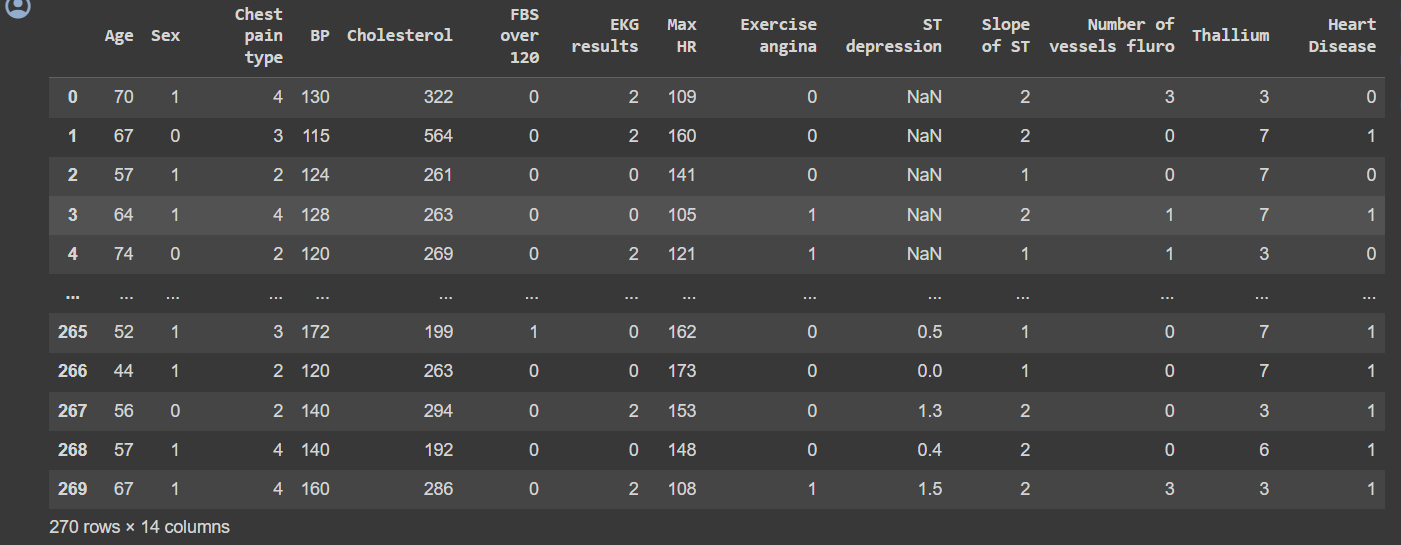
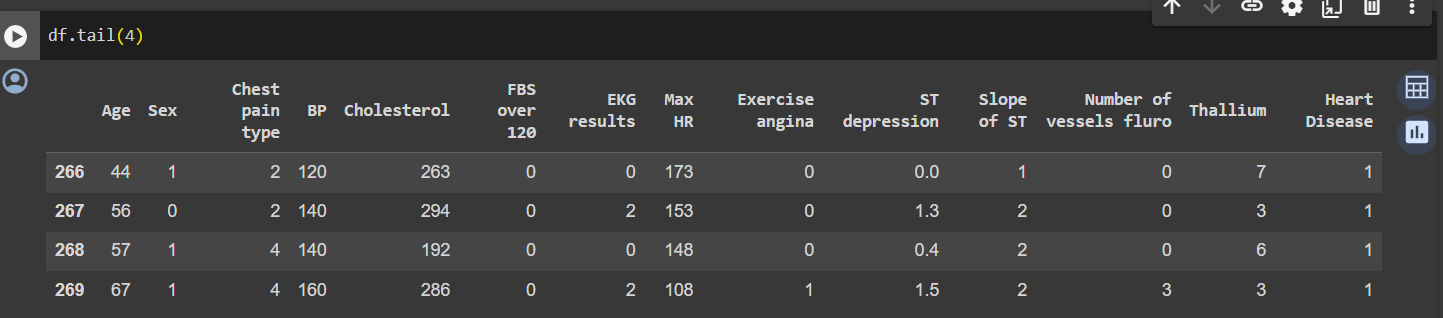
df.info()

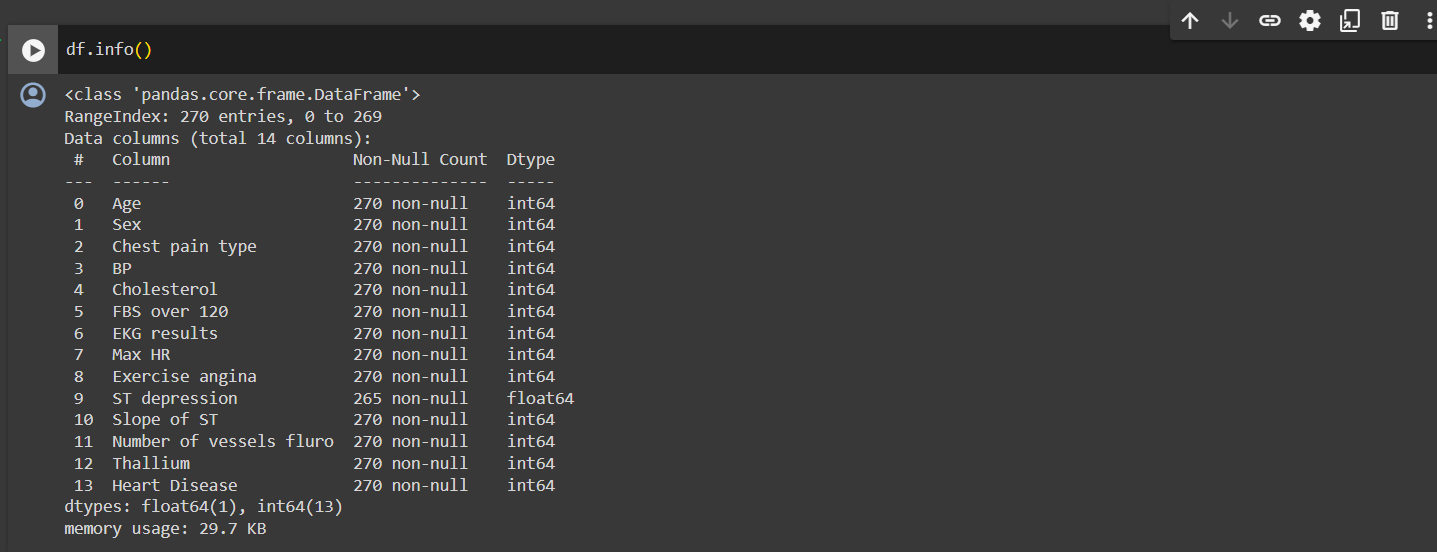
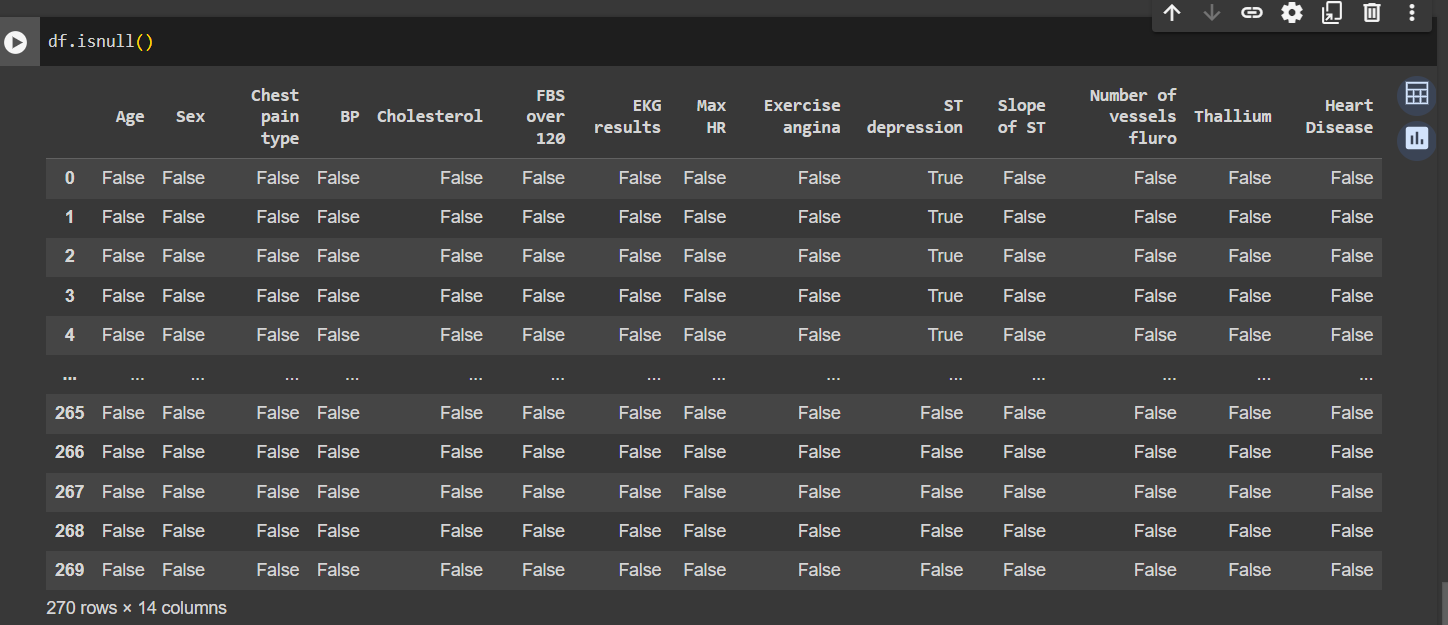
df.isnull()

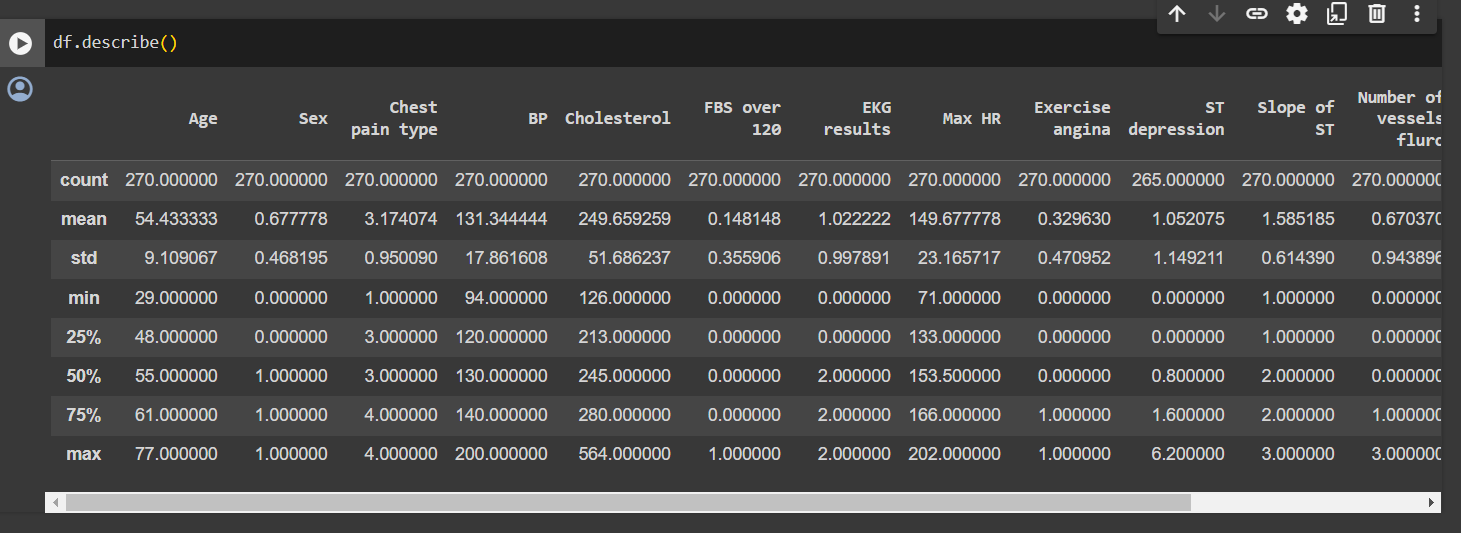
df.describe()

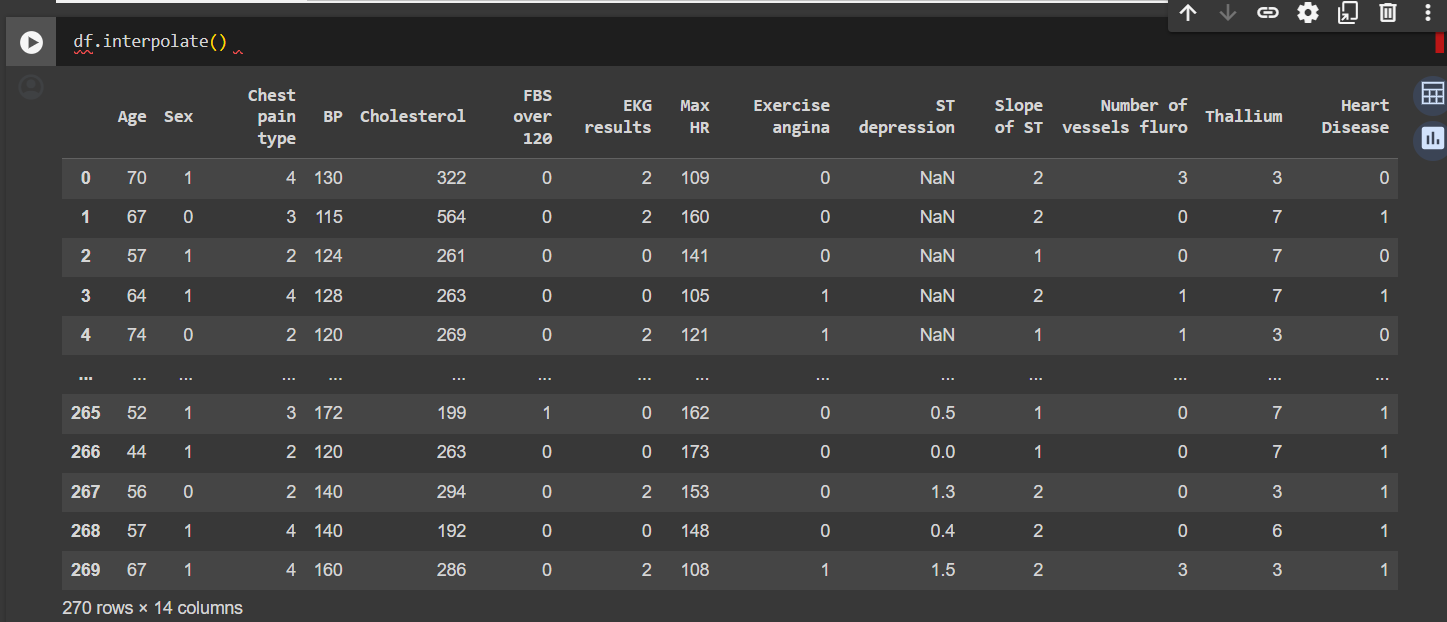
df.interpolate()

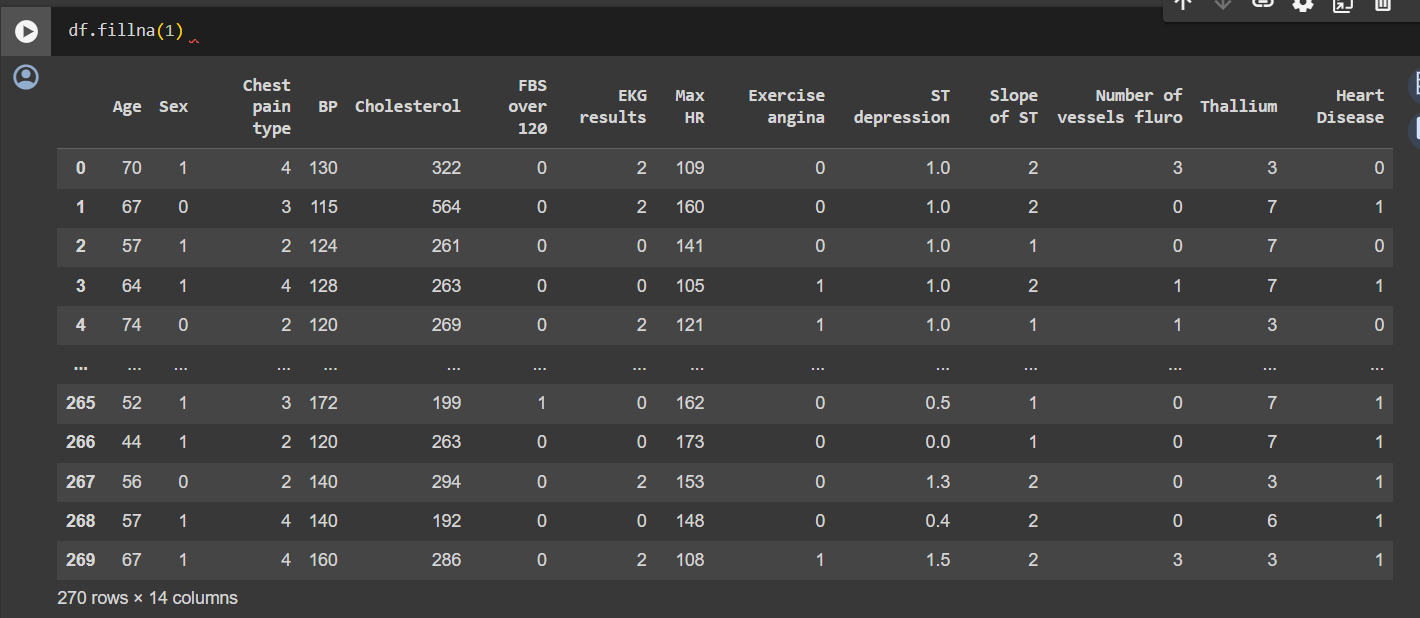
df.fillna(1)











|  |
| --- |
| RESULT:  Above code as been successfully executed and output has been verified and A clean dataset with  missing values addressed the Categorical features converted to numerical representations Numerical  features scaled to a common range Training and testing datasets prepared for model training and  evaluation. |

**EXP NO:3 MULTIVARIATE LINEAR REGRESSION**

**DATE:**

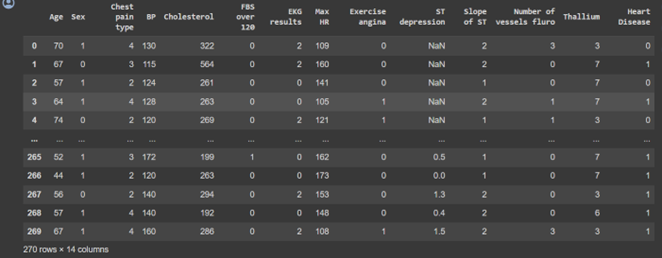
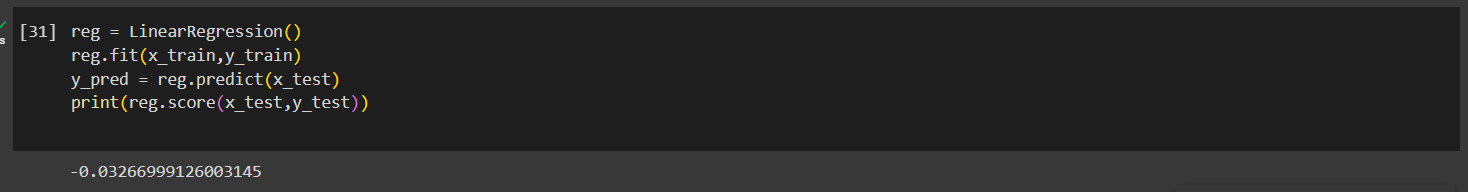
**AIM:**

The aim is likely to demonstrate Multivariate Linear Regression, a statistical method that uses multiple explanatory variables to predict a continuous outcome variable.

**PROCEDURE:**

1. **Import libraries:** The notebook will likely import libraries like pandas for data manipulation, NumPy for numerical computations, and scikit-learn for machine learning tasks.
2. **Load the data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the relationships between features and the target variable, and visualizing data distributions.
4. **Splitting data into training and testing sets:** The data is split into two sets: a training set used to fit the model and a testing set used to evaluate the model's generalizability on unseen data.
5. **Create a Multivariate Linear Regression model:** A Multivariate Linear Regression model is created from scikit-learn's linear\_model module.
6. **Train the model:** The model is trained on the training data. This involves fitting the model coefficients to minimize the difference between the predicted and actual target values.
7. **Evaluate the model's performance:** The model's performance is evaluated on the testing data using metrics like R-squared and mean squared error (MSE). R-squared indicates the proportion of variance in the target variable explained by the model, while MSE measures the average squared difference between predicted and actual values.

|  |
| --- |
| **IMPLEMENTATION CODE:**  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.linear\_model import LinearRegression df = pd.read\_csv(r' /content/Heart\_Disease\_Prediction (1).csv)  df  x = df[['age']]  y = df[['Heart Disease']]  from sklearn.model\_selection import train\_test\_split  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4) x\_train x\_test y\_train y\_test  reg = LinearRegression() reg.fit(x\_train,y\_train) y\_pred = reg.predict(x\_test) print("R-squared score:") print(reg.score(x\_test,y\_test))  from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error r\_squared = reg.score(x\_test, y\_test) print("R-squared score:") print(r\_squared) y\_pred = reg.predict(x\_test) mae = mean\_absolute\_error(y\_test, y\_pred) print("Mean Absolute Error (MAE):", mae) mse = mean\_squared\_error(y\_test, y\_pred) print("Mean Squared Error (MSE):", mse) rmse = mse \*\* 0.5  print("Root Mean Squared Error (RMSE):", rmse) |



**RESULT:**

Above code as been successfully executed and output has been verified and A trained Multivariate Linear Regression model Evaluation metrics to assess how well the model generalizes to unseen data.

**EXP NO: 4 LOGISTIC REGRESSION**

**DATE:**

**AIM:**

The aim of this notebook is to demonstrate how to use Logistic Regression to classify data points into two categories. It will likely guide you through building, training, and evaluating a Logistic Regression model.

**PROCEDURE:**

1. **Import Libraries:** This involves importing libraries like pandas for data manipulation, scikitlearn for machine learning tasks, and matplotlib for visualization (if applicable).
2. **Load Data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the distribution of features, and visualizing relationships between features and the target variable (which is likely binary in this case).
4. **Preprocessing:** The data might need preprocessing steps like handling missing values, encoding categorical features.
5. **Splitting Data into Training and Testing Sets:** The data is split into two sets: a training set used to fit the model and a testing set used to evaluate the model's performance on unseen data.
6. **Create a Logistic Regression Model:** A Logistic Regression model is created from scikit-learn's linear\_model module.
7. **Train the model:** The model is trained on the training data. This involves fitting the model coefficients to minimize the error between predicted probabilities of belonging to a class and the actual class labels.
8. **Evaluate the model's performance:** The model's performance is evaluated on the testing data using metrics like accuracy, precision, recall, and F1-score. These metrics provide insights into how well the model classifies data points into the two categories.

**IMPLEMENTATION CODE:**

|  |
| --- |
| import pandas as pd import numpy as np  import matplotlib.pyplot as plt  from sklearn.linear\_model import LogisticRegression df = pd.read\_csv(r'/content/diabetes (4).csv') df  x = df[['Pregnancies','DiabetesPedigreeFunction']] y = df[['Outcome']]  from sklearn.model\_selection import train\_test\_split  x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.4) x\_train x\_test y\_train y\_test  lr=LogisticRegression() lr.fit(x\_train,y\_train) y\_pred = lr.predict(x\_test) print(lr.score(x\_test,y\_test)) |

**OUTPUT:**

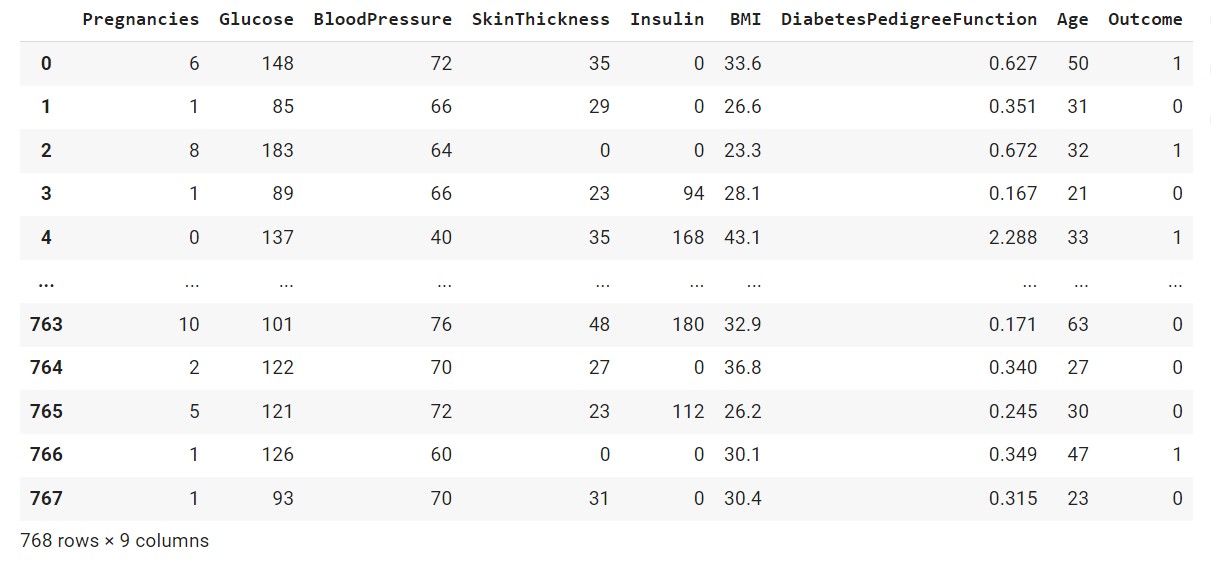
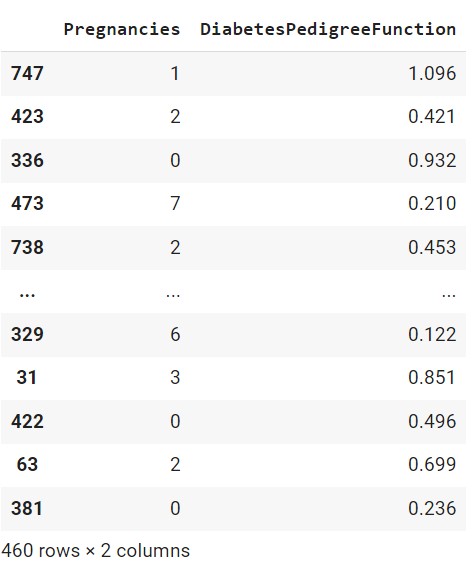
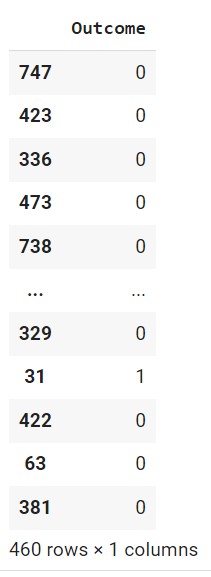


Figure 1- df

 Figure 2 x\_train Figure 3-x\_test Figure 4- y\_train

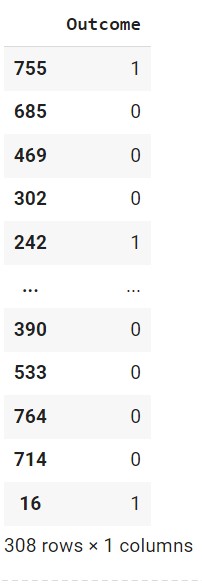
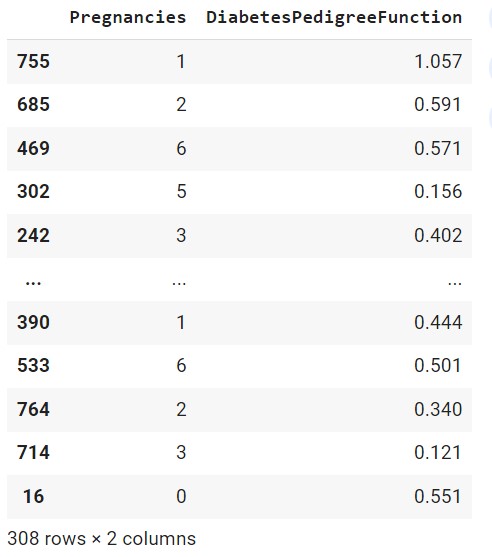


Figure 5- y\_test



Figure 6-Accuracy

**RESULT:**

Above code as been successfully executed and output has been verified A trained Logistic Regression model for binary classification Evaluation metrics to assess the model's performance on unseen data

**EXP NO: 5 DECISION TREE**

**DATE:**

**AIM:**

The aim of this notebook is to introduce Decision Trees, a machine learning model that uses a tree-like structure to make predictions ,training, and evaluating a Decision Tree model.

**PROCEDURE:**

1. **Import Libraries:** This involves importing libraries like pandas for data manipulation, scikitlearn for machine learning tasks, and matplotlib or seaborn for visualization (if applicable).
2. **Load Data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the distribution of features, and visualizing relationships between features and the target variable.
4. **Preprocessing:** The data might need preprocessing steps like handling missing values and encoding categorical features (similar to LogisticRegression.ipynb). Scaling numerical features might be less common with Decision Trees compared to linear models.
5. **Splitting Data into Training and Testing Sets:** The data is split into two sets: a training set used to build the model and a testing set used to evaluate the model's performance on unseen data.
6. **Create a Decision Tree Model:** A Decision Tree model is created from scikit-learn's tree module. The notebook might allow you to experiment with different hyperparameters like maximum depth of the tree or splitting criteria (e.g., Gini impurity).
7. **Train the model:** The model is trained on the training data. This involves recursively splitting the data based on features and building a tree-like structure that predicts the target variable at the leaves.
8. **Evaluate the model's performance:** The model's performance is evaluated on the testing data using metrics like accuracy (for classification) or mean squared error (MSE) (for regression).

The notebook might also introduce metrics specific to Decision Trees, like precision, recall, and F1-score (especially for classification tasks).

1. **Visualize the Decision Tree:** The notebook might include functionalities to visualize the decision tree, allowing you to see the splitting criteria and decision rules at each node.

**IMPLEMENTATION CODE:**

|  |
| --- |
| import pandas as pd import numpy as np from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy\_score df = pd.read\_csv(r'/content/IRIS (1).csv') df  x = df[['sepal\_length','petal\_length','sepal\_width']] y = df[['species']]  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42) x\_train x\_test y\_train y\_test  clf = DecisionTreeClassifier() clf.fit(x\_train, y\_train) y\_pred = clf.predict(x\_test)  accuracy = accuracy\_score(y\_test, y\_pred) print("Accuracy:", accuracy) plt.figure(figsize=(12,8)) from sklearn import tree  tree.plot\_tree(clf\_en.fit(x\_train, y\_train)) |

**OUTPUT:**

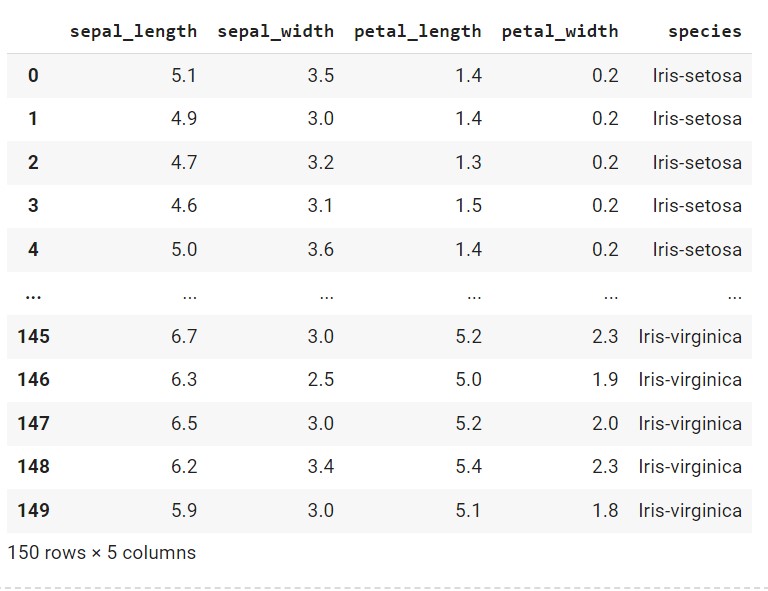


Figure 1 - df

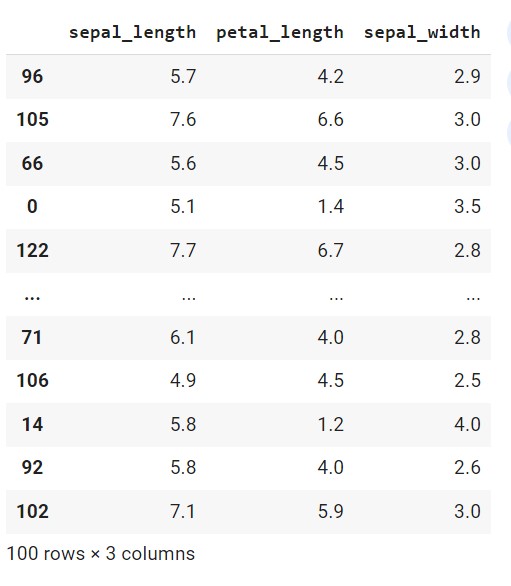


Figure 2 - x\_train



Figure 3- x\_test



Figure 4 - y\_train

Figure 5

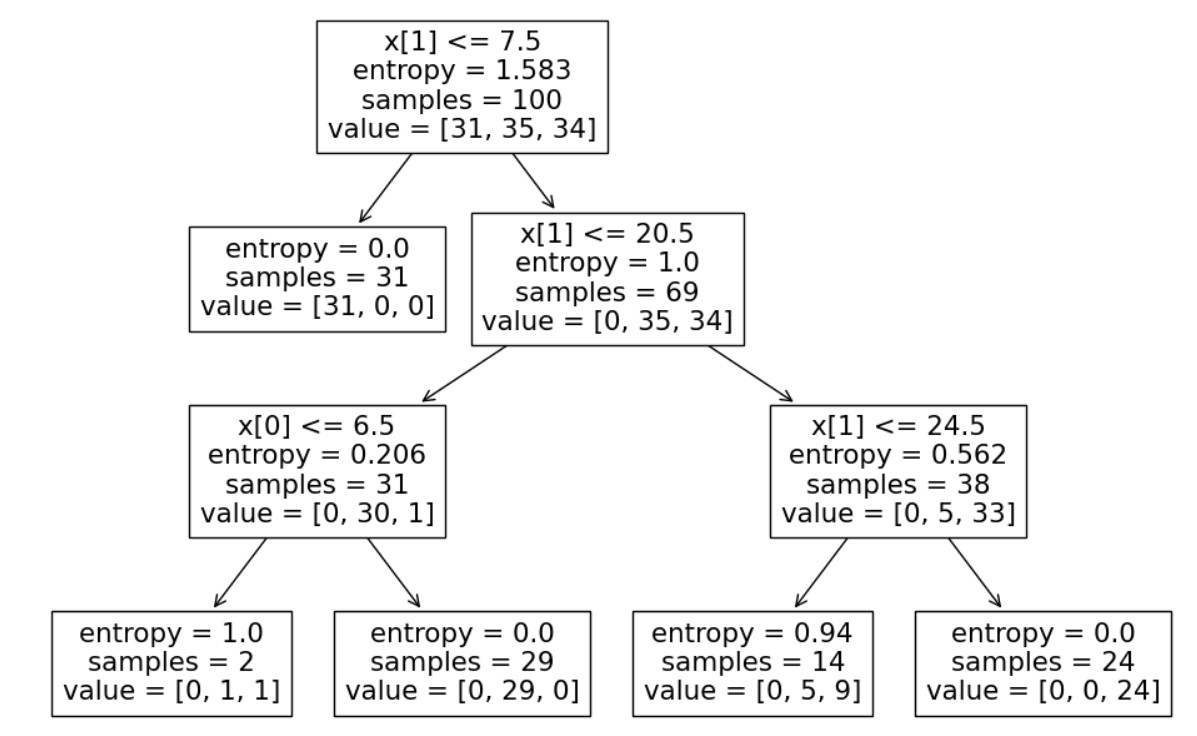
-

y\_test

Figure 6

-

Visualization of Decision Tree



**RESULT:**

Above code as been successfully executed and output has been verified A trained Decision Tree model for classification or regression. Evaluation metrics to assess the model's performance on unseen data. A visualization of the decision tree, aiding in understanding the model's logic.

**EXP NO: 6 K-Nearest Neighbors.**

**DATE:**

**AIM:**

The aim of this notebook is to introduce KNN and demonstrate its application in classification problems. Classifying new data points based on the similarity to existing labeled data Understanding the impact of the number of neighbors (k) on the model's performance

**PROCEDURE:**

1. **Import Libraries:** This involves importing libraries like pandas for data manipulation, scikitlearn for machine learning tasks, and matplotlib for visualization (if applicable).
2. **Load Data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the distribution of features, and visualizing relationships between features and the target variable.
4. **Preprocessing:** The data might need preprocessing steps like handling missing values and encoding categorical features. Scaling numerical features might be beneficial depending on the data and distance metric used.
5. **Splitting Data into Training and Testing Sets:** The data is split into two sets: a training set used to train the KNN model and a testing set used to evaluate its performance on unseen data.
6. **Create a KNN Model:** A KNN model is created from scikit-learn's neighbors module. The notebook might allow you to experiment with the number of neighbors (k) and the distance metric used (e.g., Euclidean distance, Manhattan distance).
7. **Train the model:** In KNN, there's no explicit training phase like other algorithms. The model simply stores the training data.
8. **Make Predictions:** The KNN model is used to predict the class labels for data points in the testing set. It classifies each data point based on the majority vote of its k nearest neighbors in the training data.
9. **Evaluate the model's performance:** The model's performance is evaluated on the testing data using metrics like accuracy, precision, recall, and F1-score.
10. **Visualize the KNN decision boundary:** The notebook might include visualizations to understand how the decision boundary for classification is formed based on the k nearest neighbors.

**IMPLEMENTATION CODE:**

|  |
| --- |
| import pandas as pd import numpy as np  import matplotlib.pyplot as plt  from sklearn.tree import DecisionTreeClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score df = pd.read\_csv(r'/content/IRIS (1).csv') df  X = df[['sepal\_length', 'sepal\_width', 'petal\_length','petal\_width']] y = df[['species']]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0) X\_train X\_test y\_train y\_test  classifier = KNeighborsClassifier(n\_neighbors=3)  classifier.fit(X\_train, y\_train) y\_pred = classifier.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)\*100  print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.') |

**OUTPUT:**

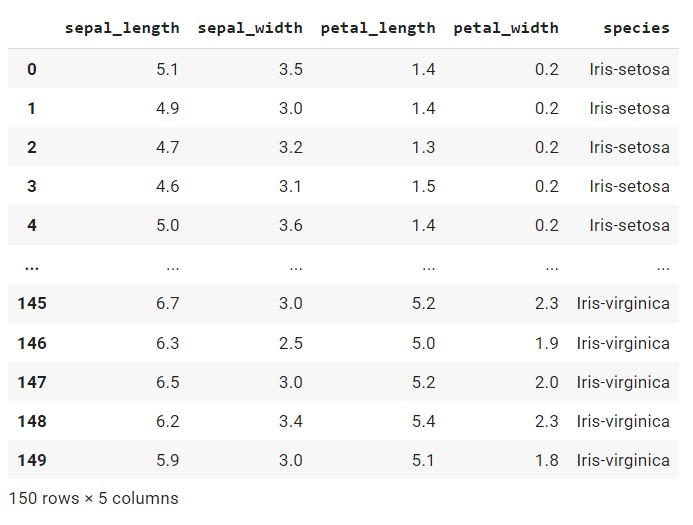


Figure 1 -df

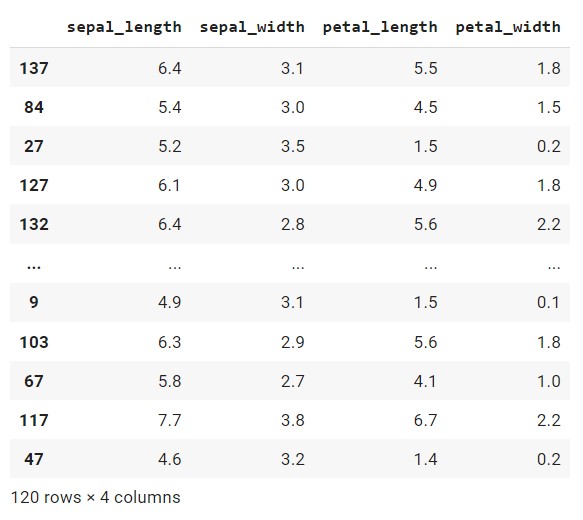


Figure 2- x\_train

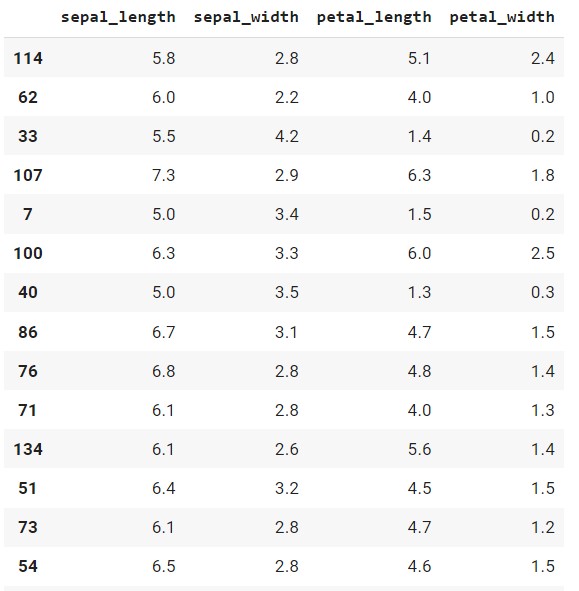


Figure 3-x\_test



*Figure 4-y\_train*



Figure 5-y\_test



Figure 6- Model Accuracy

**RESULT:**

Above code as been successfully executed and output has been verified Evaluation metrics to assess the model's performance on unseen data. An understanding of how the number of neighbors (k) affects the model's performance.

**EXP NO: 7 K-MEANS CLUSTERING**

**DATE:**

**AIM:**

The aim of this notebook is to introduce K-Means clustering and demonstrate its application for grouping data points into meaningful clusters

**PROCEDURE:**

1. **Import Libraries:** This involves importing libraries like pandas for data manipulation, scikitlearn for K-Means clustering, and matplotlib or seaborn for visualization.
2. **Load Data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the distribution of features, and potentially visualizing relationships between features.
4. **Preprocessing:** The data might need preprocessing steps like handling missing values and scaling numerical features (to ensure all features contribute equally during distance calculations).
5. **Determine the number of clusters (k):** This might involve using the Elbow Method. It iteratively trains K-Means models with different k values and evaluates the "within-cluster sum of squares" (WCSS) for each model. The Elbow Method suggests choosing k where the WCSS starts to flatten out significantly, indicating an optimal number of clusters.
6. **Create a K-Means Model:** A K-Means model is created from scikit-learn's cluster module. You might be able to experiment with different parameters like the initialization method (e.g., kmeans++ for better convergence) or the maximum number of iterations.
7. **Train the model:** The K-Means model is trained on the data. It assigns each data point to the closest cluster center (centroid) based on a distance metric (usually Euclidean distance). The centroids are then recalculated based on the assigned data points. This iterative process continues until the centroids stabilize (or a maximum number of iterations is reached).
8. **Predict cluster labels:** The model predicts cluster labels (integers representing cluster membership) for each data point.
9. **Evaluate results :** While K-Means doesn't have a formal evaluation metric, the notebook might provide ways to assess the quality of clusters, like visualizing data points colored by their assigned clusters or calculating silhouette scores (measures how well data points are separated within their clusters).

|  |  |
| --- | --- |
| **IMPLEMENTATION CODE:**  import pandas as pd import numpy as np import matplotlib.pyplot as plt from sklearn.cluster import KMeans from sklearn.metrics import silhouette\_score, davies\_bouldin\_score data = pd.read\_csv("/content/Mall\_Customers (1).csv") data  X = data.iloc[:, [2, 3, 4]].values wcss = [] for i in range(1, 11):  kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0) kmeans.fit(X) wcss.append(kmeans.inertia\_) plt.plot(range(1, 11), wcss) plt.title('Elbow Method') plt.xlabel('Number of clusters') plt.ylabel('WCSS') plt.show()  kmeans = KMeans(n\_clusters=5, init='k-means++', max\_iter=300, n\_init=10, random\_state=0) y\_kmeans = kmeans.fit\_predict(X) fig = plt.figure() ax = fig.add\_subplot(111, projection='3d') ax.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], X[y\_kmeans == 0, 2], s=100, c='red', label='CustomerID') ax.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], X[y\_kmeans == 1, 2], s=100, c='blue', label='Gender') ax.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], X[y\_kmeans == 2, 2], s=100, c='green', label='Age') ax.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], X[y\_kmeans == 3, 2], s=100, c='cyan', label='Annual Income (k$)')  ax.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], X[y\_kmeans == 4, 2], s=100, c='magenta', label='Spending Score (1-100)')  ax.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], kmeans.cluster\_centers\_[:, 2], s=300, c='yellow', label='Centroids') ax.set\_title('Clusters of customers') ax.set\_xlabel('Age')  ax.set\_ylabel('Annual Income (k$)') ax.set\_zlabel('Spending Score (1-100)') plt.legend() plt.show() silhouette\_avg = silhouette\_score(X, y\_kmeans) print("Silhouette Score:", silhouette\_avg) | |
| davies\_bouldin\_index = davies\_bouldin\_score(X, y\_kmeans) print("Davies-Bouldin Index:", davies\_bouldin\_index) | |

**OUTPUT:**

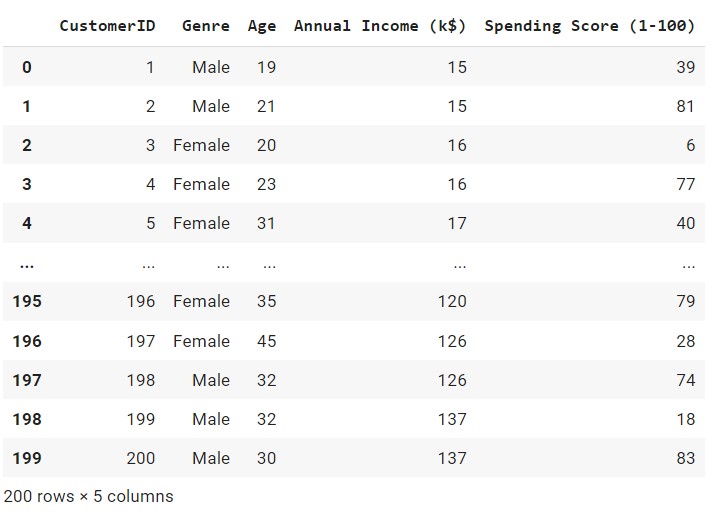


Figure 1 - data

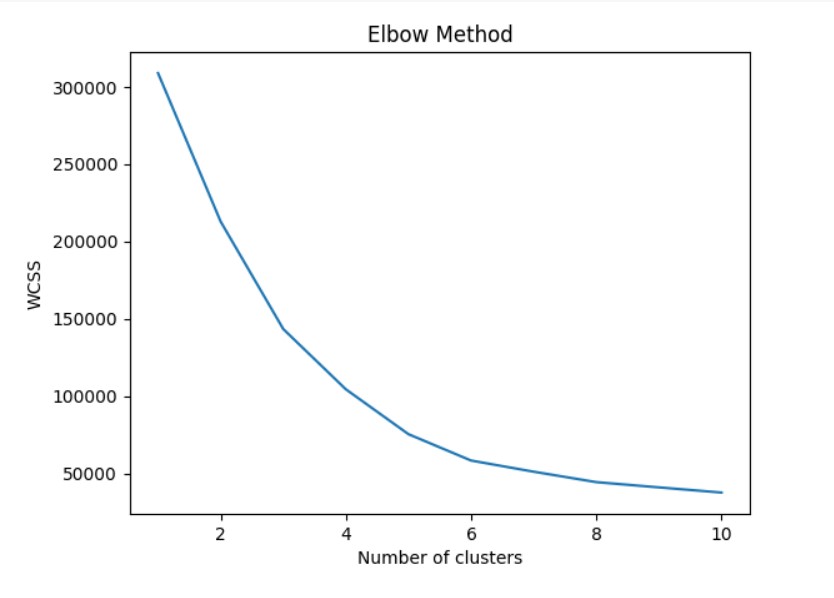


Figure 2 -Elbow Method

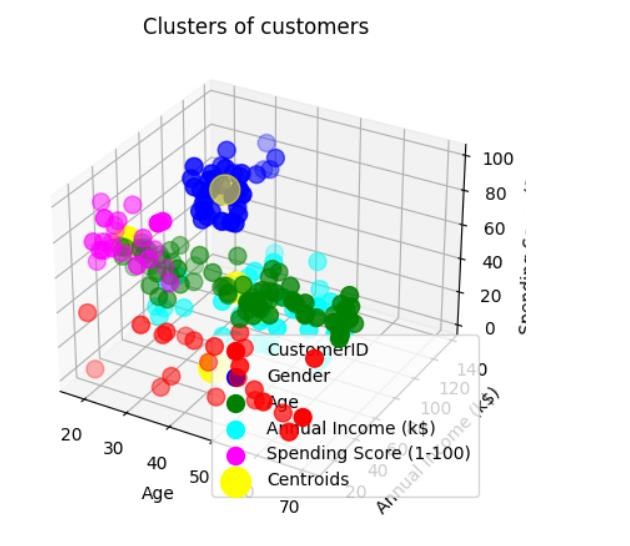


Figure 3 - Clusters of customers



Figure 4- Silhouette Score



Figure 5 - Davies-Bouldin Index

.

**RESULT:**

Above code as been successfully executed and output has been verified .A K-Means model that partitions the data into a specified number of clusters. Cluster labels for each data point, indicating their assigned cluster membership. Visualizations of the clusters, allowing you to analyze how the data has been grouped. Insights into the optimal number of clusters for your data.

**EXP NO: 8 HIERARCHICAL-CLUSTER**

**DATE:**

**AIM:**

The aim of this notebook is to introduce Hierarchical Clustering and demonstrate its application for creating a hierarchical structure of data clusters

**PROCEDURE:**

1. **Import Libraries:** This involves importing libraries like pandas for data manipulation, scikitlearn for Hierarchical Clustering, and matplotlib or seaborn for visualization.
2. **Load Data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the distribution of features, and potentially visualizing relationships between features.
4. **Preprocessing:** The data might need preprocessing steps like handling missing values and scaling numerical features (similar to K-Means).
5. **Fit the model:** The model is not explicitly trained, but it learns the hierarchical relationships between data points during the fitting process.
6. **Determine the number of clusters:** Unlike K-Means, there's no predefined number of clusters. The notebook might introduce techniques like dendrogram cutting (using a threshold on the linkage distance) or silhouette analysis to estimate a suitable number of clusters at a specific level of the hierarchy.
7. **Visualize the results:** The main result is typically a dendrogram, a tree-like structure that depicts the merging process and the relationships between clusters at different levels.

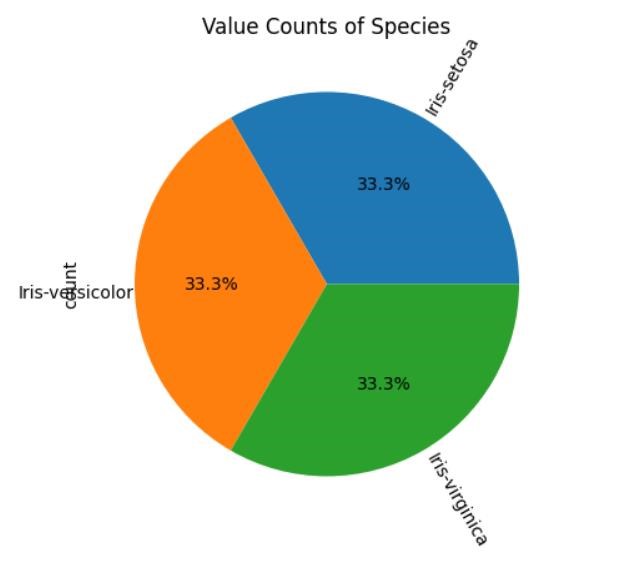
**IMPLEMENTATION CODE:**

|  |
| --- |
| import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.cluster import AgglomerativeClustering from scipy.cluster import hierarchy from sklearn.metrics import silhouette\_score data=pd.read\_csv('/content/Iris.csv') data data[['Species']].describe() data.Species.value\_counts().plot(kind='pie',autopct='%0.1f%%',shadow=False,lab eldistance=1,rotatelabels=True,figsize=(5,5)); plt.title('Value Counts of Species');  sns.pairplot(data,hue='Species',diag\_kind='hist'); X=data.iloc[:,:-1].values plt.figure(figsize=(15,15)) dendrogram=hierarchy.dendrogram(hierarchy.linkage(X,method='ward')) hc=AgglomerativeClustering(n\_clusters=3,metric='euclidean',linkage='ward') pred=hc.fit\_predict(X) plt.scatter(X[pred==0,0],X[pred==0,1],c='green',label='Cluster 1') plt.scatter(X[pred==1,0],X[pred==1,1],c='blue',label='Cluster 2') plt.scatter(X[pred==2,0],X[pred==2,1],c='red',label='Cluster 3') plt.xlabel('Sepal Length') plt.ylabel('Sepal Width') plt.title('Sepal Length vs Sepal Width') plt.legend(); silhouette\_score(X,pred) |

**OUTPUT:**



*Figure 1 - data*



*Figure 2 - Value Counts of Species*

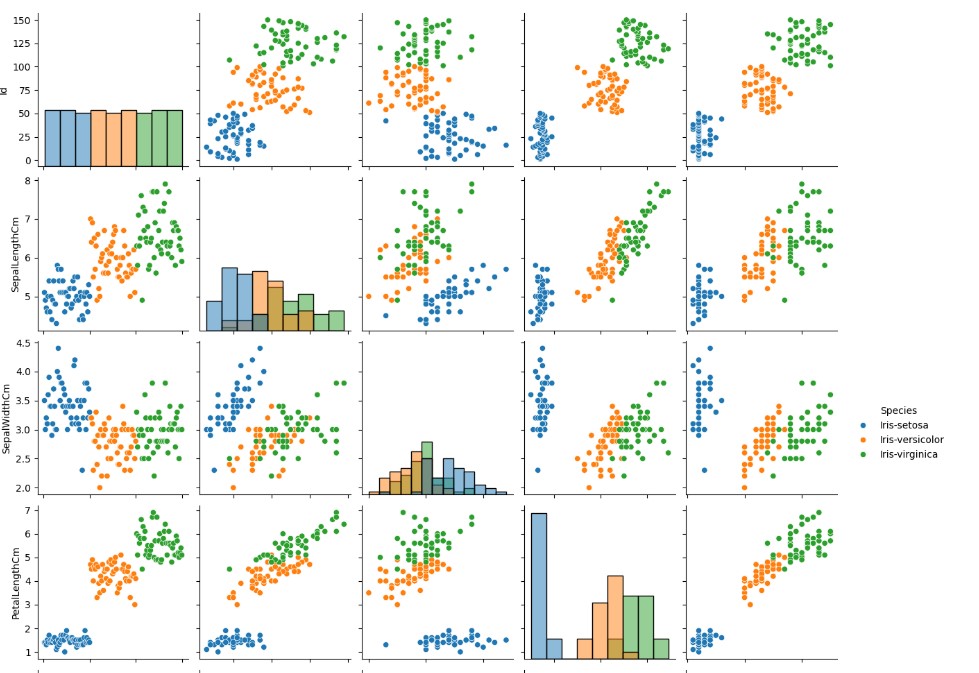
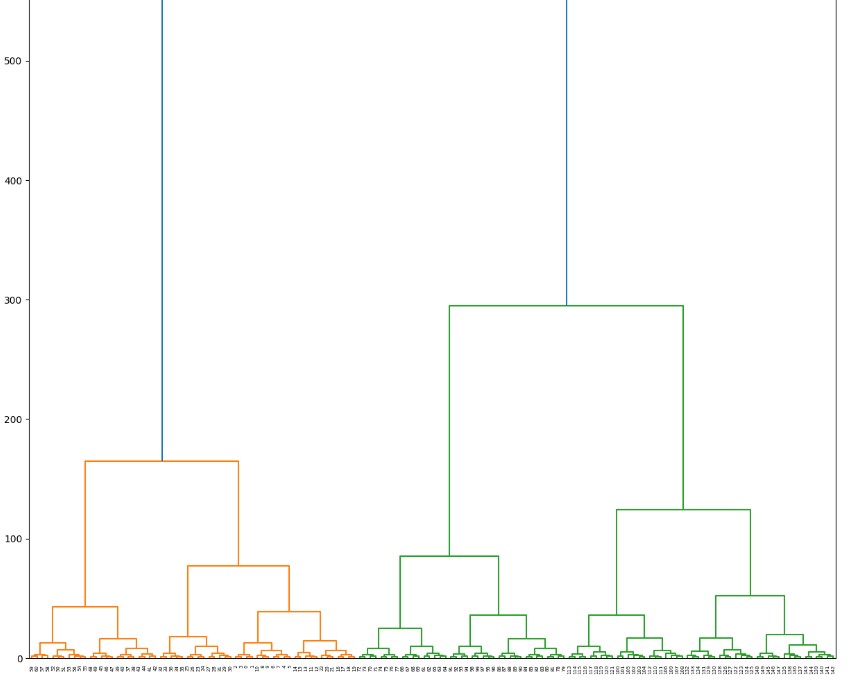
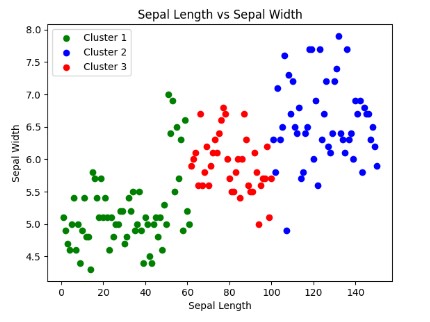


Figure 3 - sns plot



*Figure 4 -Dendrogram*



*Figure 5 -Clusters*



*Figure 6 - silhouette\_score*

**RESULT:**

Above code as been successfully executed and output has been verified. A Hierarchical Clustering model that captures the hierarchical relationships between data points. A dendrogram visualization that shows how data points are progressively merged into clusters. An estimate of the number of clusters at a chosen level of the hierarchy.

**EXP NO: 9 SUPPORT VECTOR MACHINE**

**DATE:**

**AIM:**

The aim of this notebook is to introduce SVMs and demonstrate their application in classification problems. Evaluating the performance of the SVM model on unseen data

**PROCEDURE:**

1. **Import Libraries:** This involves importing libraries like pandas for data manipulation, scikitlearn for SVMs and other functionalities, and matplotlib or seaborn for visualization.
2. **Load Data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the distribution of features, and visualizing relationships between features and the target variable.
4. **Preprocessing:** The data might need preprocessing steps like handling missing values, encoding categorical features, and potentially scaling numerical features (to ensure all features contribute equally during calculations).
5. **Splitting Data into Training and Testing Sets:** The data is split into two sets: a training set used to build the SVM model and a testing set used to evaluate its performance on unseen data.
6. **Create an SVM Model:** An SVM classifier is created from scikit-learn's svm module. You might be able to experiment with different parameters like:
   * Kernel function (e.g., linear, polynomial, RBF) - how the SVM transforms data points to potentially higher dimensions for better separation.
   * Regularization parameter (C) - controls the trade-off between maximizing the margin and allowing for some misclassifications.
7. **Train the model:** The SVM model is trained on the training data. It identifies the support vectors and learns the optimal hyperplane that separates the classes with the maximum margin.
8. **Evaluate the model's performance:** The model's performance is evaluated on the testing data using metrics like accuracy, precision, recall, and F1-score.

.

**IMPLEMENTATION CODE:**

import

seaborn

as

sns

import

pandas

as

pd

import

matplotlib.pyplot

as

plt

%matplotlib

inline

from

sklearn.model\_selection

import

train\_test\_split

from

sklearn.svm

import

SVC

from

sklearn.metrics

import

classification\_report,confusion\_matrix

df = pd.read\_csv(

'/content/Iris.csv'

)

df.head()

sns.pairplot(df,hue=

'Species'

,palette=

'Dark2'

)

X = df.drop(

'Species'

,axis=

1

)

y = df[

'Species'

]

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

0.30

)

svc\_model = SVC()

svc\_model.fit(X\_train,y\_train)

predictions = svc\_model.predict(X\_test)

print

classification\_report(y\_test,predictions

(

))

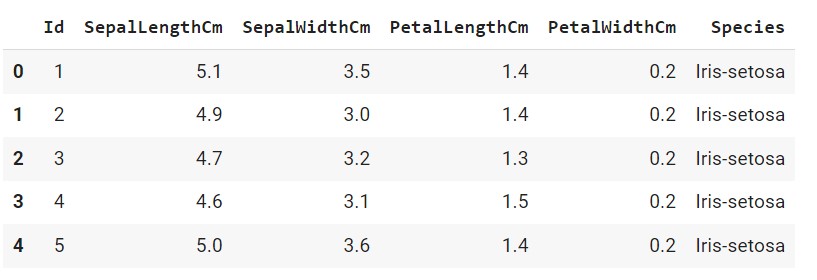
**OUTPUT:**

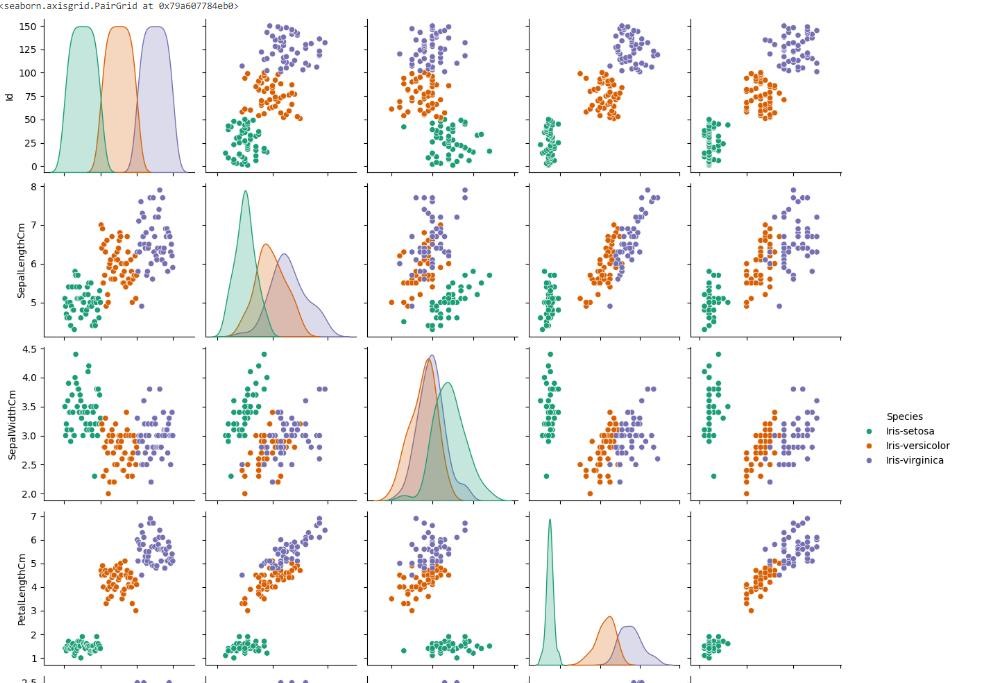
*Figure*

*1*

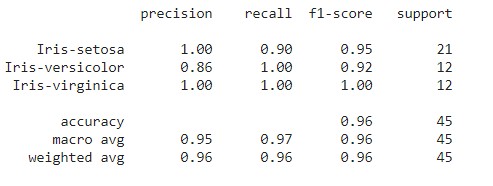
*-*

*df*





*Figure 2 - sns plot*



*Figure 3 -Model Accuracy*

**RESULT:**

Above code as been successfully executed and output has been verified. A trained SVM model for classification. Evaluation metrics to assess the model's performance on unseen data.

**EXP NO: 10 PRINCIPAL COMPONENT ANALYZSIS**

**DATE:**

**AIM:**

The aim of this notebook is to introduce PCA and demonstrate its application for reducing the dimensionality of data while preserving the most important information

**PROCEDURE:**

1. **Import Libraries:** This involves importing libraries like pandas for data manipulation, scikitlearn for PCA and other functionalities, and matplotlib or seaborn for visualization.
2. **Load Data:** The data can be loaded from various sources like CSV files, Google BigQuery, or spreadsheets. The notebook will likely demonstrate how to use appropriate functions for each data source.
3. **Exploratory Data Analysis (EDA):** This might involve checking for missing values, identifying data types, understanding the distribution of features, and potentially visualizing relationships between features.
4. **Preprocessing:** The data might need preprocessing steps like handling missing values and potentially scaling numerical features (to ensure all features contribute equally during calculations).
5. **Create a PCA Model:** A PCA model is created from scikit-learn's decomposition module. You might be able to specify the desired number of components to retain (usually chosen based on the explained variance ratio).
6. **Fit the model:** The PCA model is not explicitly trained, but it learns the principal components by performing a dimensionality reduction on the data.
7. **Transform the data:** The data is transformed into the lower-dimensional space using the principal components identified by the PCA model**.**

**IMPLEMENTATION CODE:**

|  |
| --- |
| from sklearn.datasets import load\_iris from sklearn.decomposition import PCA from sklearn.model\_selection import train\_test\_split iris = load\_iris() X = iris.data y = iris.target  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) n\_components = int(0.9 \* X.shape[1]) |

pca = PCA(n\_components=n\_components)

pca.fit(X\_train)

X\_train\_pca = pca.transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

explained\_variance = pca.explained\_variance\_ratio\_

print

(

"Explained Variance Ratio:"

)

for

i, ev

in

enumerate

explained\_variance

):

(

print

(

f

"Component

i

+

{

1

}

:

{

ev

:.4

f

}

"

)

import

matplotlib.pyplot

as

plt

plt.scatter(X\_train\_pca[:,

0

]

, X\_train\_pca[:,

1

, c=y\_train, cmap

]

=

'plasma'

)

plt.xlabel(

"Principal Component 1"

)

plt.ylabel(

"Principal Component 2"

)

plt.title(

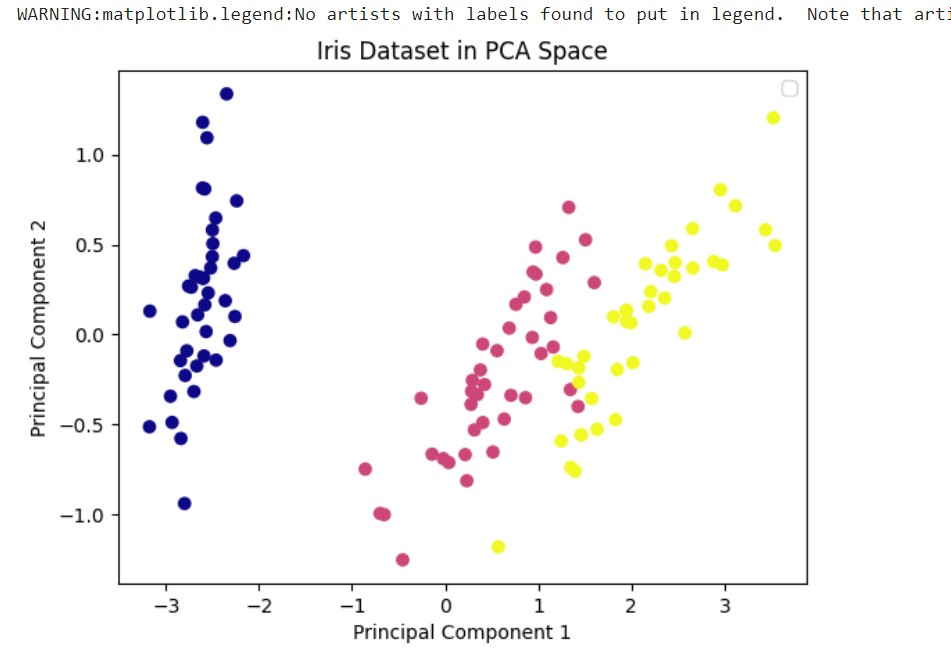
"Iris Dataset in PCA Space"

)

plt.legend()

plt.show()

**OUTPUT:**



**RESULT:**

Above code as been successfully executed and output has been verified. A PCA model that transforms your data into a lower-dimensional space. The transformed data, containing the most important information based on the chosen principal components. An understanding of the explained variance ratio for each principal component.