**Experiment Number : 8 -**  **Data Classification using Decision Tree Algorithm (ID3)**

**Batch: FDS-2 Roll Number: 1601042223 Name: Chandana Ramesh Galgali**

**Aim of the Experiment:** Exploration of data classification using Decision Tree algorithm (ID3) on a Sample Dataset

**Program/ Steps:**

1. Computes the ID3 algorithm to select an attribute subset that best predicts class labels

2. Use Decision Tree Classifiers to classify the Sample Data(e.g.IRIS Sample Data).

3. Manual write should take his own data so that it could be possible to find best class

labels as well as do the classification.

**Code with Output/Result:**

**Problem 2: Attribute subset selection using the ID3 algorithm. (Python Code)**

**import numpy as np**

**import pandas as pd**

**# Sample dataset**

**data = pd.DataFrame({**

**'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],**

**'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],**

**'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'High', 'Normal', 'High'],**

**'Class': ['No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No']**

**})**

**# Function to calculate entropy**

**def entropy(class\_labels):**

**unique\_labels, counts = np.unique(class\_labels, return\_counts=True)**

**prob = counts / len(class\_labels)**

**entropy = -np.sum(prob \* np.log2(prob))**

**return entropy**

**# Function to calculate information gain**

**def information\_gain(data, attribute, class\_label):**

**total\_entropy = entropy(data[class\_label])**

**unique\_values = data[attribute].unique()**

**weighted\_entropy = 0**

**for value in unique\_values:**

**subset = data[data[attribute] == value]**

**subset\_entropy = entropy(subset[class\_label])**

**weight = len(subset) / len(data)**

**weighted\_entropy += weight \* subset\_entropy**

**return total\_entropy - weighted\_entropy**

**# ID3 algorithm for attribute selection**

**def id3(data, class\_label, attributes):**

**if len(attributes) == 0:**

**# If no attributes are left, return the majority class**

**return data[class\_label].mode().iloc[0]**

**unique\_classes = data[class\_label].unique()**

**if len(unique\_classes) == 1:**

**# If all examples have the same class, return that class**

**return unique\_classes[0]**

**best\_attribute = max(attributes, key=lambda attr: information\_gain(data, attr, class\_label))**

**tree = {best\_attribute: {}}**

**for value in data[best\_attribute].unique():**

**subset = data[data[best\_attribute] == value]**

**if len(subset) == 0:**

**# If the subset is empty, return the majority class**

**tree[best\_attribute][value] = data[class\_label].mode().iloc[0]**

**else:**

**new\_attributes = [attr for attr in attributes if attr != best\_attribute]**

**tree[best\_attribute][value] = id3(subset, class\_label, new\_attributes)**

**return tree**

**# Define the class label and attributes**

**class\_label = 'Class'**

**attributes = ['Outlook', 'Temperature', 'Humidity']**

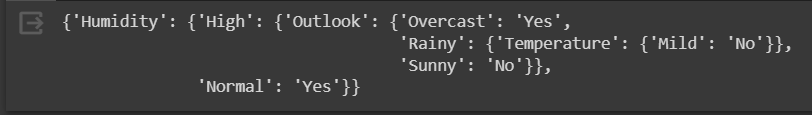
**# Build the ID3 decision tree**

**decision\_tree = id3(data, class\_label, attributes)**

**# Print the decision tree**

**import pprint**

**pprint.pprint(decision\_tree)**

****

**Problem 3: Classification of Data (IRIS SAMPLE DATA) with Decision Tree Algorithm**

**from sklearn import datasets**

**import pandas as pd**

**import numpy as np**

**from sklearn import metrics**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.tree import DecisionTreeClassifier**

**import matplotlib.pyplot as plt**

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

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

**iris = datasets.load\_iris() #Loading the dataset**

**iris.keys()**

**print("Iris Keys:", iris.keys)**

**dict\_keys=(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename', 'data\_module'])**

**print("Dict Keys:", dict\_keys)**

**#convert dataset to pandas df**

**iris = pd.DataFrame(**

**data= np.c\_[iris['data'], iris['target']],**

**columns= iris['feature\_names'] + ['target']**

**)**

**print(iris)**

**print(iris.head())**

**# To give name to target class as species, Species is a list**

**species = []**

**for i in range(len(iris['target'])):**

**if iris['target'][i] == 0:**

**species.append("setosa")**

**elif iris['target'][i] == 1:**

**species.append('versicolor')**

**else:**

**species.append('virginica')**

**iris['species'] = species**

**print("Species have class now", iris['species'])**

**#To check new data set is taken place or not then used once again tail/head command**

**print(iris.tail())**

**# Do observe total sample of each species**

**print( iris.groupby('species').size())**

**# Plotting a dataset is a great way to explore its distribution.**

**# Plotting the iris dataset can be done using matplotlib, a Python library for 2D plotting.**

**import matplotlib**

**setosa = iris[iris.species == "setosa"]**

**versicolor = iris[iris.species=='versicolor']**

**virginica = iris[iris.species=='virginica']**

**fig, ax = plt.subplots()**

**fig.set\_size\_inches(13, 7) # adjusting the length and width of plot, lables and scatter points**

**ax.scatter(setosa['petal length (cm)'], setosa['petal width (cm)'],**

**label="Setosa", facecolor="blue")**

**ax.scatter(versicolor['petal length (cm)'], versicolor['petal width (cm)'],**

**label="Versicolor", facecolor="green")**

**ax.scatter(virginica['petal length (cm)'], virginica['petal width (cm)'],**

**label="Virginica", facecolor="red")**

**ax.set\_xlabel("petal length (cm)")**

**ax.set\_ylabel("petal width (cm)")**

**ax.grid()**

**ax.set\_title("Iris petals")**

**ax.legend()**

**# To display image here (optional)**

**plt.show()**

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

**X = iris.drop('species', axis=1)**

**y = iris['species']**

**# 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)**

**# Create and Train the Decision Tree Classifier:**

**# Create an instance of DecisionTreeClassifier and train it on the training data using the fit() method.**

**# Create a DecisionTreeClassifier instance**

**classifier = DecisionTreeClassifier(random\_state=42)**

**DecisionTreeClassifier**

**# Train the classifier on the training data**

**classifier.fit(X\_train, y\_train)**

**# Make Predictions:**

**# Use the trained classifier to make predictions on new, unseen data (testing set) using the predict() method.**

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

**x\_pred=classifier.predict(X\_train)**

**'''Evaluate the Model:**

**Calculate evaluation metrics to assess the performance of the model. For classification tasks, you can use metrics like accuracy, precision, recall, F1-score, and confusion matrix.**

**'''**

**''' Compute Training time Accuracy'''**

**# Calculate accuracy**

**accuracy = accuracy\_score(y\_train, x\_pred)**

**print("Accuracy:", accuracy)**

**# Display classification report**

**print("Classification Report:")**

**print(classification\_report(y\_train, x\_pred))**

**# Display confusion matrix**

**print("Confusion Matrix:")**

**print(confusion\_matrix(y\_train, x\_pred))**

**''' Compute Testing time Accuracy'''**

**print("Testing Time Accuracy")**

**# Calculate accuracy**

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

**print("Accuracy Testing:", accuracy)**

**# Display classification report**

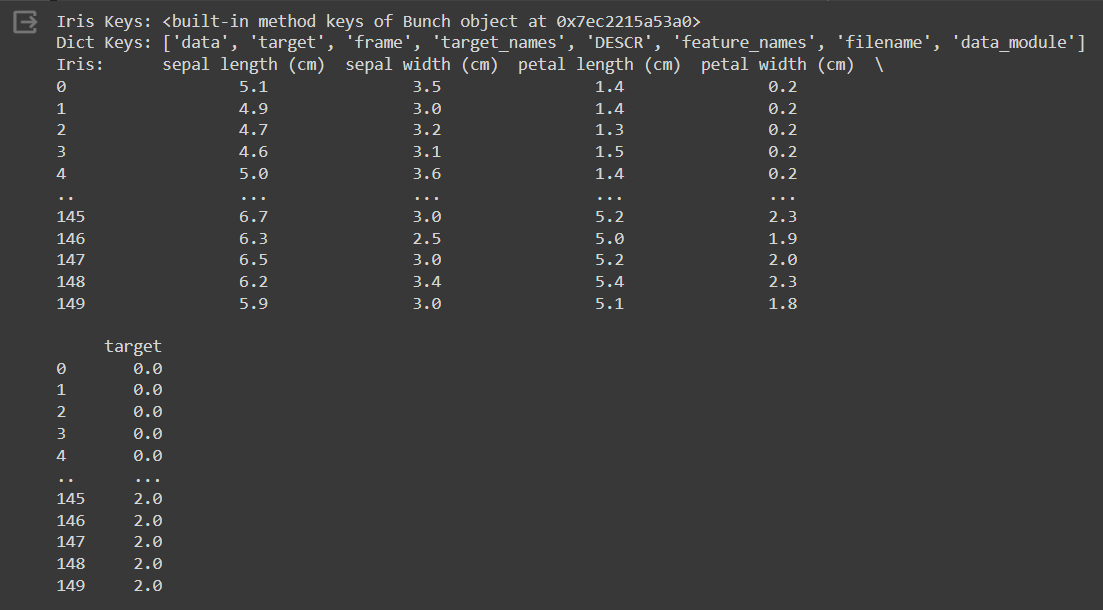
**print("Classification Report Testing:")**

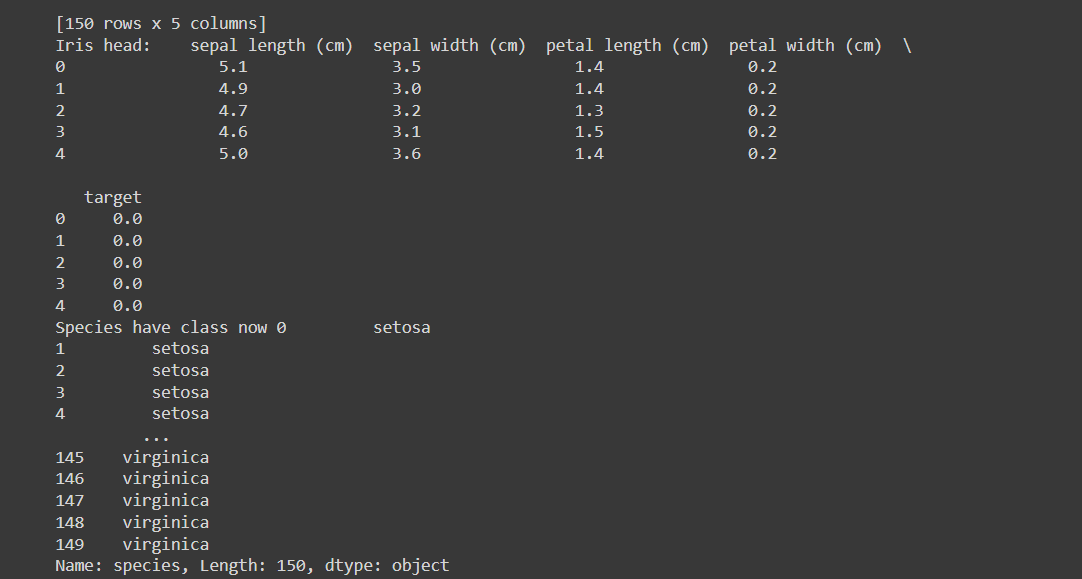
**print(classification\_report(y\_test, y\_pred))**

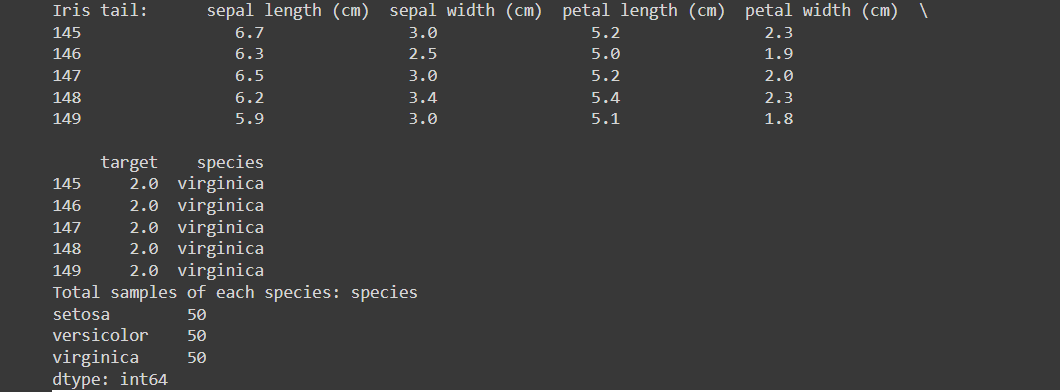
**# Display confusion matrix**

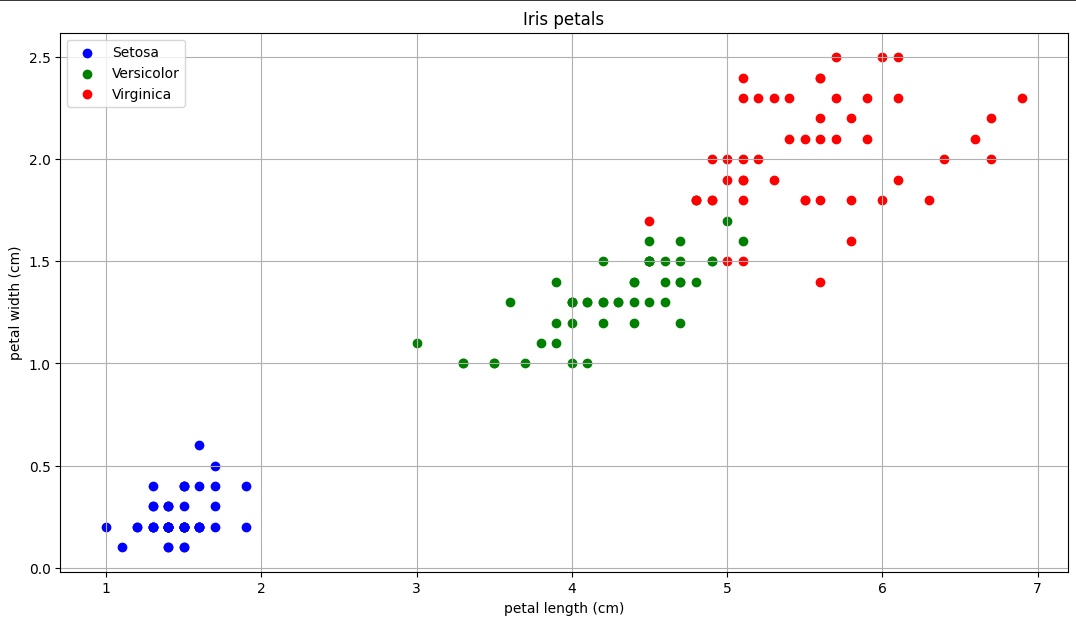
**print("Confusion Matrix:")**

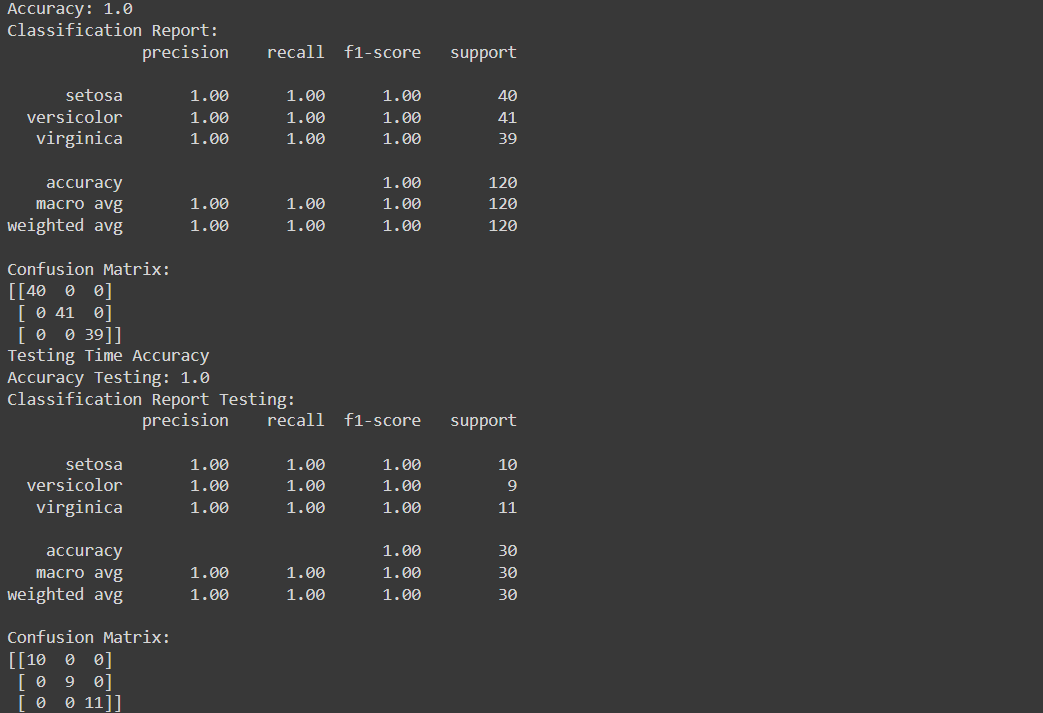
**print(confusion\_matrix(y\_test, y\_pred))**

****

****

****

****

****

**Code:**

**import numpy as np**

**import pandas as pd**

**dataframe = pd.read\_csv(r'/content/Flight\_delay.csv')**

**data\_array = dataframe.to\_numpy()**

**data = pd.DataFrame({**

**'AirTime': data\_array[:100, 11],**

**'TaxiIn': data\_array[:100, 19],**

**'TaxiOut': data\_array[:100, 20],**

**'CarrierDelay': data\_array[:100, 24]**

**})**

**# Function to calculate entropy**

**def entropy(class\_labels):**

**unique\_labels, counts = np.unique(class\_labels, return\_counts=True)**

**prob = counts / len(class\_labels)**

**entropy = -np.sum(prob \* np.log2(prob))**

**return entropy**

**# Function to calculate information gain**

**def information\_gain(data, attribute, class\_label):**

**total\_entropy = entropy(data[class\_label])**

**unique\_values = data[attribute].unique()**

**weighted\_entropy = 0**

**for value in unique\_values:**

**subset = data[data[attribute] == value]**

**subset\_entropy = entropy(subset[class\_label])**

**weight = len(subset) / len(data)**

**weighted\_entropy += weight \* subset\_entropy**

**return total\_entropy - weighted\_entropy**

**# ID3 algorithm for attribute selection**

**def id3(data, class\_label, attributes):**

**if len(attributes) == 0:**

**# If no attributes are left, return the majority class**

**return data[class\_label].mode().iloc[0]**

**unique\_classes = data[class\_label].unique()**

**if len(unique\_classes) == 1:**

**# If all examples have the same class, return that class**

**return unique\_classes[0]**

**best\_attribute = max(attributes, key=lambda attr: information\_gain(data, attr, class\_label))**

**tree = {best\_attribute: {}}**

**for value in data[best\_attribute].unique():**

**subset = data[data[best\_attribute] == value]**

**if len(subset) == 0:**

**# If the subset is empty, return the majority class**

**tree[best\_attribute][value] = data[class\_label].mode().iloc[0]**

**else:**

**new\_attributes = [attr for attr in attributes if attr != best\_attribute]**

**tree[best\_attribute][value] = id3(subset, class\_label, new\_attributes)**

**return tree**

**# Define the class label and attributes**

**class\_label = 'CarrierDelay'**

**attributes = ['AirTime', 'TaxiIn', 'TaxiOut']**

**# Build the ID3 decision tree**

**decision\_tree = id3(data, class\_label, attributes)**

**# Print the decision tree**

**import pprint**

**pprint.pprint(decision\_tree)**

**Result:**

**{'AirTime': {32: 0,**

**34: 3,**

**36: {'TaxiIn': {3: 27, 4: 0, 5: 0}},**

**37: 4,**

**41: {'TaxiIn': {9: 61, 16: 18}},**

**42: {'TaxiIn': {6: 0, 7: 16}},**

**43: {'TaxiIn': {3: 114, 5: 2, 7: 0}},**

**44: 13,**

**45: 15,**

**46: {'TaxiOut': {10: 20, 15: 9, 17: 17, 18: 1}},**

**47: {'TaxiOut': {5: 7, 16: 10, 24: 9}},**

**48: {'TaxiIn': {3: 4, 4: 6}},**

**49: {'TaxiIn': {2: 12, 5: 2, 7: 0}},**

**59: {'TaxiOut': {7: 282, 9: 0}},**

**60: {'TaxiOut': {7: 26, 10: 7, 11: 15}},**

**65: 50,**

**72: {'TaxiIn': {5: {'TaxiOut': {9: 0}}}},**

**73: 9,**

**76: 2,**

**77: {'TaxiOut': {8: 45, 10: 2, 13: 0, 16: 26}},**

**78: {'TaxiIn': {2: 10, 4: {'TaxiOut': {10: 11, 18: 14}}, 6: 7}},**

**80: 2,**

**81: {'TaxiIn': {3: 15, 8: 7}},**

**88: 0,**

**90: 25,**

**91: 3,**

**95: 27,**

**97: 1,**

**106: 24,**

**107: 8,**

**110: 0,**

**112: 12,**

**113: 3,**

**116: 50,**

**118: 3,**

**121: 0,**

**125: {'TaxiIn': {2: 32, 6: 11}},**

**127: 23,**

**130: {'TaxiIn': {4: 59, 5: 4}},**

**131: 14,**

**132: 38,**

**134: 5,**

**139: 1,**

**143: {'TaxiIn': {3: 12, 5: 0, 6: 40}},**

**150: 17,**

**166: 3,**

**168: 13,**

**171: {'TaxiIn': {7: 60, 8: 4}},**

**175: 0,**

**176: 0,**

**177: {'TaxiIn': {5: 0, 6: 4, 62: 6}},**

**178: 2,**

**179: 32,**

**183: 18,**

**195: 10,**

**201: 10,**

**213: 3,**

**221: 48,**

**230: {'TaxiIn': {3: 10, 5: 7}},**

**232: 6,**

**237: 13,**

**243: 8,**

**244: 18,**

**245: 18,**

**253: 19,**

**269: 13}}**

**Code:**

**import pandas as pd**

**from sklearn.tree import DecisionTreeRegressor**

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

**from sklearn.metrics import mean\_squared\_error**

**# Load the dataset from CSV file**

**dataset = pd.read\_csv('/content/Flight\_delay.csv')**

**# Drop non-numeric columns**

**dataset = dataset.select\_dtypes(include='number')**

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

**X = dataset.drop('ActualElapsedTime', axis=1)**

**y = dataset['ActualElapsedTime']**

**# 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 an instance of the DecisionTreeRegressor**

**regressor = DecisionTreeRegressor()**

**# Train the regressor on the training data**

**regressor.fit(X\_train, y\_train)**

**# Make predictions on the testing data**

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

**# Calculate the mean squared error of the regressor**

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

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

**Result:**

****

**Post Lab Question-Answers:**

**1. What are the data filtering techniques available? Explain in brief.**

**Ans:** There are several data filtering techniques available for processing and analyzing data. Here are a few commonly used techniques:

1. Rule-based Filtering: This technique involves applying predefined rules to filter data based on specific conditions or criteria. For example, you can filter data based on a specific range of values or exclude certain categories.

2. Time-based Filtering: This technique involves filtering data based on time-related criteria. You can filter data by specific time periods, such as days, weeks, or months, or by specific dates or time ranges.

3. Frequency-based Filtering: This technique involves filtering data based on the frequency of occurrence. You can filter data to include only the most frequent or least frequent values, or you can set a threshold to include values above or below a certain frequency.

4. Pattern-based Filtering: This technique involves filtering data based on specific patterns or sequences. You can use pattern matching algorithms or regular expressions to identify and filter data that matches a particular pattern.

5. Statistical Filtering: This technique involves filtering data based on statistical measures. You can filter data based on measures such as mean, median, standard deviation, or percentiles to include or exclude values that fall within certain statistical ranges.

6. Text-based Filtering: This technique involves filtering textual data based on specific keywords, phrases, or patterns. You can use techniques like keyword matching, text mining, or natural language processing to filter and extract relevant information from text data.

These are just a few examples of data filtering techniques. The choice of technique depends on the specific requirements and characteristics of the data being analyzed.

**2. What do you mean by sampling a data set? How is sampling done in data science?**

**Ans:** Sampling a dataset refers to the process of selecting a subset of observations or data points from a larger population or dataset. In data science, sampling is commonly used to make inferences or draw conclusions about the entire population based on the analysis of the smaller sample.

Sampling in data science can be done using various techniques, including:

1. Simple Random Sampling: This technique involves randomly selecting observations from the population, where each observation has an equal chance of being selected. It is often used when the population is homogeneous and there are no specific criteria for selection.

2. Stratified Sampling: This technique involves dividing the population into distinct subgroups or strata based on certain characteristics, and then selecting samples from each stratum. It ensures that the sample represents the diversity of the population and can be useful when there are significant differences within the population.

3. Cluster Sampling: This technique involves dividing the population into clusters or groups, and then randomly selecting entire clusters to include in the sample. It is useful when it is difficult or costly to sample individual observations, but clusters can be easily sampled.

4. Systematic Sampling: This technique involves selecting every nth observation from the population after randomly selecting a starting point. It provides a simple and efficient way to sample large populations.

5. Sampling with Replacement: This technique allows selected observations to be included in the sample multiple times. It is useful when the population is small or when the same observation can contribute to multiple samples.

Sampling in data science is crucial because it allows analysts to work with manageable subsets of data, reducing computational complexity and saving time. By analyzing the sample, data scientists can make inferences and draw conclusions about the larger population, assuming the sample is representative and unbiased.

**Outcomes: Apply the transformations required on data to make it suitable for Mining**

**Conclusion (based on the Results and outcomes achieved):**

The experiment demonstrated the effectiveness of the ID3 algorithm for data classification. It showcased the potential of decision tree algorithms in analyzing and categorizing datasets, providing valuable insights for various applications in data science and machine learning.

**References:**

Books/ Journals/ Websites

1. Han, Kamber, "Data Mining Concepts and Techniques", Morgan Kaufmann 3nd Edition