Data Warehousing and Data Mining Lab

Faculty Name: Ms. Sakshi Jha

Student Name: Vanshik Sanwaria

Roll No.: 01914812721

Semester: 7th

Group: CST I

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

M A I T

उद्यमेन हि सिध्यन्ति
कार्याणि न मनोरथैः

Department of Computer Science and Engineering

Rubrics for Lab Assessment

Rubrics		0	1	2	3
		Missing	Inadequate	Needs Improvement	Adequate
R1	Is able to identify the problem to be solved and define the objectives of the experiment.	No mention is made of the problem to be solved.	An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable.	The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors.	The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors.
R2	Is able to design a reliable experiment that solves the problem.	The experiment does not solve the problem.	The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution.	The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable solution	The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution.
R3	Is able to communicate the details of an experimental procedure clearly and completely.	Diagrams are missing and/or experimental procedure is missing or extremely vague.	Diagrams are present but unclear and/or experimental procedure is present but important details are missing.	Diagrams and/or experimental procedure are present but with minor omissions or vague details.	Diagrams and/or experimental procedure are clear and complete.
R4	Is able to record and represent data in a meaningful way.	Data are either absent or incomprehensible.	Some important data are absent or incomprehensible.	All important data are present, but recorded in a way that requires some effort to comprehend.	All important data are present, organized and recorded clearly.
R5	Is able to make a judgment about the results of the experiment.	No discussion is presented about the results of the experiment.	A judgment is made about the results, but it is not reasonable or coherent.	An acceptable judgment is made about the result, but the reasoning is flawed or incomplete.	An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered.

INDEX

Name: Vanshik Sanwaria

Branch: Computer Science and Technology

Roll No: 01914812721

Group: 7 CST - 1

Exp. No.	Experiment Name	Date of performance	Date of checking	R1 (3)	R2 (3)	R3 (3)	R4 (3)	R5 (3)	Total Marks (15)	Signature
1.	Study of ETL process and its tools.									
2.	Program of Data warehouse cleansing to input names from users (inconsistent) and format them.									
3.	Program of Data warehouse cleansing to remove redundancy in data.									
4.	Introduction to WEKA tool.									
5.	Implementation of Classification technique on ARFF files using WEKA.									
6.	Implementation of Clustering technique on ARFF files using WEKA.									
7.	Implementation of Association Rule technique on ARFF files using WEKA.									
8.	Implementation of Visualization technique on ARFF files using WEKA.									
9.	Perform Data Similarity Measure (Euclidean, Manhattan Distance).									
10.	Perform Apriori algorithm to mine frequent item-sets.									
11.	Develop different clustering algorithms like K-Means, KMedoids Algorithm, Partitioning Algorithm and Hierarchical									
12.	Apply Validity Measures to evaluate the quality of Data									

Experiment – 1

Aim: Study of ETL process and its tools.

Theory:

The ETL (Extract, Transform, Load) process is a critical component in data warehousing and data integration. It enables organizations to gather data from various sources, transform it into a structured format, and load it into a central data warehouse or database, making it accessible for analysis, reporting, and decision-making. Here's a breakdown of each stage in the ETL process:

1. Extract

- **Purpose**: In this step, data is extracted from various sources, such as transactional databases, spreadsheets, legacy systems, or cloud services.
- **Process**: Extraction methods depend on the data source. For databases, it may involve querying data through SQL; for web sources, APIs may be used; and for unstructured data, methods like web scraping may be employed.
- **Challenges**: Extracting data from multiple sources may involve different formats and structures, requiring careful handling to ensure accuracy and completeness.

2. Transform

- **Purpose**: The extracted data is processed and converted into a usable format that aligns with the requirements of the target data warehouse.
- **Process**: Transformation may involve data cleaning (removing duplicates, handling missing values), data mapping, data aggregation, and standardizing formats (like dates or currency).
- **Techniques**: Common transformations include filtering data, merging datasets, calculating new metrics, applying business rules, and reformatting data types.
- **Challenges**: Transformation must ensure data integrity and consistency, requiring a clear understanding of business rules and data relationships.

3. Load

- **Purpose**: The final stage is to load the transformed data into a target system, usually a data warehouse or data mart, where it can be accessed for analysis and reporting.
- **Process**: Data loading can be done in two ways:
    - **Full Load**: Loading all data at once, typically during initial setup or major data refreshes.
    - **Incremental Load**: Loading only new or updated data to maintain an up-to-date dataset without redundancy.
- **Challenges**: Ensuring that data loads are efficient, reliable, and meet performance requirements, especially in large-scale environments with high data volumes.

Common ETL Tools

Several ETL tools are widely used in the industry, each with unique features and capabilities:

1. **Informatica PowerCenter**: A widely used ETL tool known for its robust data integration capabilities, data quality assurance, and support for various data formats.

2. **Apache NiFi**: An open-source tool focused on automating data flows, with a strong focus on security and scalability.
3. **Microsoft SQL Server Integration Services (SSIS)**: A popular ETL tool provided by Microsoft, designed for data migration, integration, and transformation tasks within the SQL Server environment.
4. **Talend**: An open-source ETL tool that offers extensive data integration features and is suitable for data cleansing, transformation, and loading in cloud and on-premises environments.
5. **Pentaho Data Integration (PDI)**: Part of the Pentaho suite, it provides user-friendly data integration and transformation features and supports big data and analytics.

Importance of ETL in Data Warehousing

The ETL process is essential for building reliable and efficient data warehouses. It ensures that data from disparate sources is unified, cleaned, and standardized, providing a single source of truth for analytics and reporting. This consistency is crucial for businesses to make accurate, data-driven decisions, ensuring that all departments rely on the same high-quality data.

The ETL process also enables data governance and compliance, as transformations can enforce data quality standards and regulatory requirements, making it essential for industries with strict compliance regulations.

Experiment – 2

Aim: Program of Data warehouse cleansing to input names from users (inconsistent) and format them.

Theory:

Data cleansing, also called data cleaning or data scrubbing, is a critical step in preparing data for analysis and storage in a data warehouse. It involves detecting, correcting, or removing inaccuracies, inconsistencies, and errors from datasets to ensure high data quality. Clean data is essential in a data warehouse because it directly impacts the accuracy and reliability of any insights derived from the data. Errors in data can lead to misleading conclusions, faulty analysis, and suboptimal decision-making.

Importance of Data Cleansing

1. **Improves Data Quality**: Clean data is accurate, consistent, and complete. Quality data allows for reliable insights and reduces the likelihood of errors in analytical outcomes.
2. **Enhances Decision Making**: When data is accurate and consistent, decisions made based on this data are more likely to be effective, driving better business outcomes.
3. **Increases Efficiency**: Data cleansing helps streamline data processing by reducing redundant data and standardizing data formats, making the data easier to analyze and interpret.
4. **Maintains Consistency**: Standardizing data formats across sources ensures that the data conforms to uniform standards, enabling seamless integration in a data warehouse.
5. **Enables Compliance**: Many industries require data accuracy for compliance with regulations. Data cleansing can ensure that data meets industry and regulatory standards.

Common Issues in Data Cleansing

Inconsistent data can arise from various sources, including human error, different data formats, or legacy systems. Some typical issues include:

1. **Inconsistent Formatting**: Variations in capitalization, spacing, or punctuation (e.g., "john doe" vs. "John Doe").
2. **Duplicate Entries**: Repeated records for the same entity, leading to skewed analysis.
3. **Incomplete Data**: Missing information in one or more fields.
4. **Incorrect Data**: Values that do not match expected patterns or contain obvious errors (e.g., incorrect phone numbers or email formats).

Data Cleansing Process

The data cleansing process typically includes these steps:

1. **Data Profiling**: Analyzing the data to understand its structure, content, and patterns. This helps identify specific areas that require cleansing.
2. **Data Standardization**: Applying uniform formats to data, such as consistent capitalization, removing special characters, or using standardized date formats.
3. **Data Validation**: Checking data against predefined rules or patterns to identify outliers or inaccuracies.
4. **Data Enrichment**: Filling missing information or correcting data using external reference data.

5. **Data Deduplication**: Identifying and removing duplicate records to avoid redundancy.

Tools for Data Cleansing

Many data integration and ETL tools offer data cleansing functionalities. Here are some widely used tools:

- **Informatica Data Quality**: Offers data profiling, cleansing, and standardization features and is particularly popular for enterprise-level data cleansing.
- **Trifacta**: Known for its user-friendly interface, Trifacta provides data profiling, transformation, and visualization for cleansing workflows.
- **OpenRefine**: An open-source tool that allows users to clean and transform data in bulk, with features for clustering similar values and removing duplicates.
- **Python**: Libraries like pandas provide robust data manipulation functions, allowing for custom data cleansing scripts tailored to specific needs.

Code:

```
def cleanse_name(name):
    # Remove leading and trailing spaces
    cleaned_name = name.strip()
    # Replace multiple spaces with a single space
    cleaned_name = " ".join(cleaned_name.split())
    # Capitalize each word in the name
    cleaned_name = cleaned_name.title()
    return cleaned_name

# Input: List of names with inconsistent formatting
names = [
    " john doe ",      # Extra spaces and lowercase
    "MARy AnnE",       # Mixed case
    "   alice   JOHNSON  ",  # Extra spaces
    "pETER  o'CONNOR"   # Mixed case and special character
]
# Apply cleansing function to each name
cleaned_names = [cleanse_name(name) for name in names]

# Display the cleaned names
print("Cleaned Names:")
for original, cleaned in zip(names, cleaned_names):
    print(f"Original: '{original}' -> Cleaned: '{cleaned}'")
```

Output:

```
Cleaned Names:
Original: ' john doe ' -> Cleaned: 'John Doe'
Original: 'MARy AnnE' -> Cleaned: 'Mary Anne'
Original: '   alice   JOHNSON  ' -> Cleaned: 'Alice Johnson'
Original: 'pETER  o'CONNOR' -> Cleaned: 'Peter O'Connor'
```

Viva Questions

1. What is data cleansing, and why is it important in data warehousing?

Answer: Data cleansing is the process of detecting and correcting (or removing) corrupt or inaccurate records from a dataset. In data warehousing, it's important because clean data ensures accuracy and consistency in analytics, reporting, and decision-making. It helps maintain the integrity of the data warehouse by eliminating errors and inconsistencies from various data sources.

2. What are some common issues in inconsistent data, particularly with names, that you handle in this program?

Answer: Inconsistent name data can include extra spaces, incorrect capitalization (e.g., all lowercase or all uppercase), and unwanted characters like numbers or special symbols. The program handles these issues by:

Stripping extra spaces.

Capitalizing the first letter of each word.

Removing non-alphabetical characters.

3. What is the role of the strip() function in this program?

Answer: The strip() function is used to remove any leading and trailing spaces from a string. This is important because users might enter names with extra spaces at the beginning or end, which can cause issues in analysis or storage.

4. How does the program handle multiple spaces between words in names?

Answer: The program uses the re.sub(r'\s+', ' ', name) function from the re (regular expression) module to replace multiple spaces between words with a single space. This ensures that names have consistent spacing.

5. Explain how the title() method is used in the program.

Answer: The title() method converts the first character of each word in the string to uppercase and the rest to lowercase.

Experiment – 3

Aim: Program of Data warehouse cleansing to remove redundancy in data.

Theory:

Redundancy in data warehousing refers to the presence of duplicate records or entries, which can result in inaccurate analysis, increased storage costs, and performance inefficiencies. Redundant data can arise due to data integration from multiple sources, manual data entry errors, or other inconsistencies. Removing redundancy helps maintain data quality, improves efficiency in data processing, and ensures accurate analysis.

Common Techniques for Removing Redundancy

1. **Deduplication**: Identifying and removing duplicate records based on specific columns or combinations of columns.
2. **Primary Key Constraints**: Ensuring unique identifiers (such as IDs) in a database to prevent duplicate entries.
3. **Data Merging and Consolidation**: Aggregating or merging data from multiple sources and applying deduplication rules.
4. **Standardization**: Normalizing data fields (such as name or address) to consistent formats, which helps in identifying duplicates.

Code:

```
import pandas as pd
# Sample dataset with duplicate entries
data = {
   'CustomerID': [101, 102, 103, 104, 101, 102],
   'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Alice', 'Bob'],
   'Email': ['alice@example.com', 'bob@example.com', 'charlie@example.com', 'david@example.com',
'alice@example.com', 'bob@example.com'],
   'PurchaseAmount': [250, 150, 300, 200, 250, 150]
}
# Load data into a DataFrame
df = pd.DataFrame(data)

# Display the original data with duplicates
print("Original Data:")
print(df)

# Remove duplicates based on all columns
df_cleaned = df.drop_duplicates()

# Alternatively, remove duplicates based on specific columns, e.g., 'CustomerID' and 'Name'
# df_cleaned = df.drop_duplicates(subset=['CustomerID', 'Name'])

# Display the cleaned data
print("\nCleaned Data (Duplicates Removed):")
print(df_cleaned)
```

Output:

```
Original Data:
   CustomerID     Name                Email  PurchaseAmount
0         101    Alice    alice@example.com             250
1         102      Bob      bob@example.com             150
2         103  Charlie  charlie@example.com             300
3         104    David    david@example.com             200
4         101    Alice    alice@example.com             250
5         102      Bob      bob@example.com             150

Cleaned Data (Duplicates Removed):
   CustomerID     Name                Email  PurchaseAmount
0         101    Alice    alice@example.com             250
1         102      Bob      bob@example.com             150
2         103  Charlie  charlie@example.com             300
3         104    David    david@example.com             200
```

Viva Questions

1. What is data redundancy, and why is it a problem in data warehousing?

Answer: Data redundancy occurs when the same piece of data is stored in multiple places within a system. In data warehousing, redundancy can lead to inconsistencies, increased storage costs, and inefficient data processing. Removing redundancy improves data quality and ensures that the warehouse contains only unique and relevant information.

2. How does your program remove redundancy in the data?

Answer: The program removes redundancy by converting the list of data entries into a set. Since a set only stores unique elements, any duplicate data is automatically removed. The set is then converted back to a list to maintain the same data structure.

3. What data structures are used in your program to handle redundancy?

Answer: The program uses a Python list to collect the input data and a set to remove duplicates. A set is ideal for this purpose because it automatically removes any duplicate values while maintaining only unique elements.

4. Can your program handle both small and large datasets?

Answer: Yes, the program can handle both small and large datasets. However, for very large datasets, using a set might have performance limitations in terms of memory usage and speed. For large-scale data warehouses, more efficient algorithms and distributed systems might be required to remove redundancy.

5. Why do you convert the data list into a set to remove redundancy?

Answer: A set is a built-in data structure in Python that only stores unique elements, meaning it automatically removes duplicates when data is added to it. This makes it an efficient and easy way to remove redundancy in a list of data entries.

Experiment – 4

Aim: Introduction to WEKA tool.

Theory:

WEKA (Waikato Environment for Knowledge Analysis) is a popular open-source software suite for machine learning and data mining, developed by the University of Waikato in New Zealand. It is written in Java and provides a collection of tools for data pre-processing, classification, regression, clustering, association rules, and visualization, making it highly suitable for educational and research purposes in data mining and machine learning.

Key Features of WEKA

1. **User-Friendly Interface**: WEKA offers a graphical user interface (GUI) that allows users to easily experiment with different machine learning algorithms without extensive programming knowledge. This interface includes several panels, such as Explorer, Experimenter, Knowledge Flow, and Simple CLI (Command Line Interface).
2. **Extensive Collection of Algorithms**: WEKA includes a wide variety of machine learning algorithms, such as decision trees, support vector machines, neural networks, and Naive Bayes. These algorithms can be applied to classification, regression, clustering, and other tasks, making WEKA a versatile tool for different types of data analysis.
3. **Data Pre-processing Tools**: WEKA supports data pre-processing techniques like normalization, attribute selection, data discretization, and handling missing values. These capabilities help prepare raw data for analysis, ensuring more reliable and accurate model training.
4. **File Format Compatibility**: WEKA primarily works with ARFF (Attribute-Relation File Format) files, which is a text format developed for WEKA's datasets. It also supports CSV and other data formats, making it flexible for data import.
5. **Visualization**: WEKA includes data visualization tools that allow users to explore datasets graphically. Users can view scatter plots, bar charts, and other visualizations, which help in understanding data distribution, patterns, and relationships between attributes.
6. **Extendibility**: Since WEKA is open-source, users can add new algorithms or modify existing ones to meet specific needs. Its integration with Java also enables users to incorporate WEKA into larger Java-based applications.

Components of WEKA

1. **Explorer**: The primary GUI in WEKA, which provides a comprehensive environment for loading datasets, pre-processing data, and applying machine learning algorithms. Users can easily evaluate model performance using this component.
2. **Experimenter**: Allows users to perform controlled experiments to compare different algorithms or configurations. This component is ideal for analyzing and optimizing model performance across different datasets.
3. **Knowledge Flow**: Provides a data flow-oriented interface, similar to visual workflow tools, enabling users to create complex machine learning pipelines visually. It is beneficial for designing, testing, and implementing custom workflows.

4. **Simple CLI**: A command-line interface for advanced users to interact with WEKA's functionalities using commands. This component allows users to bypass the GUI for faster, script-based operations.

Common Applications of WEKA

- **Educational Use**: WEKA is widely used for teaching machine learning concepts because it offers an easy-to-understand interface and a rich set of algorithms.
- **Research**: Researchers use WEKA to develop and test new machine learning models or compare the performance of different algorithms.
- **Real-World Data Mining**: WEKA's tools for classification, clustering, and association rule mining make it applicable in real-world tasks, including medical diagnosis, market basket analysis, text classification, and bioinformatics.

Advantages of WEKA

- **Ease of Use**: WEKA's GUI and pre-packaged algorithms make it accessible even to those new to machine learning and data mining.
- **Wide Algorithm Support**: With various machine learning techniques included, WEKA is suitable for a broad range of tasks and applications.
- **Open-Source**: Being open-source, WEKA can be freely used, modified, and integrated into other projects.

Limitations of WEKA

- **Scalability**: WEKA is primarily designed for smaller to medium-sized datasets. For big data applications, WEKA may face performance issues, and other tools, such as Apache Spark, may be more suitable.
- **Limited Real-Time Support**: WEKA is typically used for batch processing, meaning it is not ideal for real-time or streaming data applications.

Overall, WEKA remains a valuable tool for data mining and machine learning, especially in academic and research settings, due to its intuitive interface, comprehensive algorithm collection, and robust pre-processing capabilities.

Experiment – 5

Aim: Implementation of Classification technique on ARFF files using WEKA.

Theory: Classification is a supervised machine learning technique used to categorize data points into predefined classes or labels. Given a labeled dataset, a classification algorithm learns patterns in the data to predict the class labels of new, unseen instances. Classification is essential in applications such as spam detection, medical diagnosis, sentiment analysis, and image recognition.

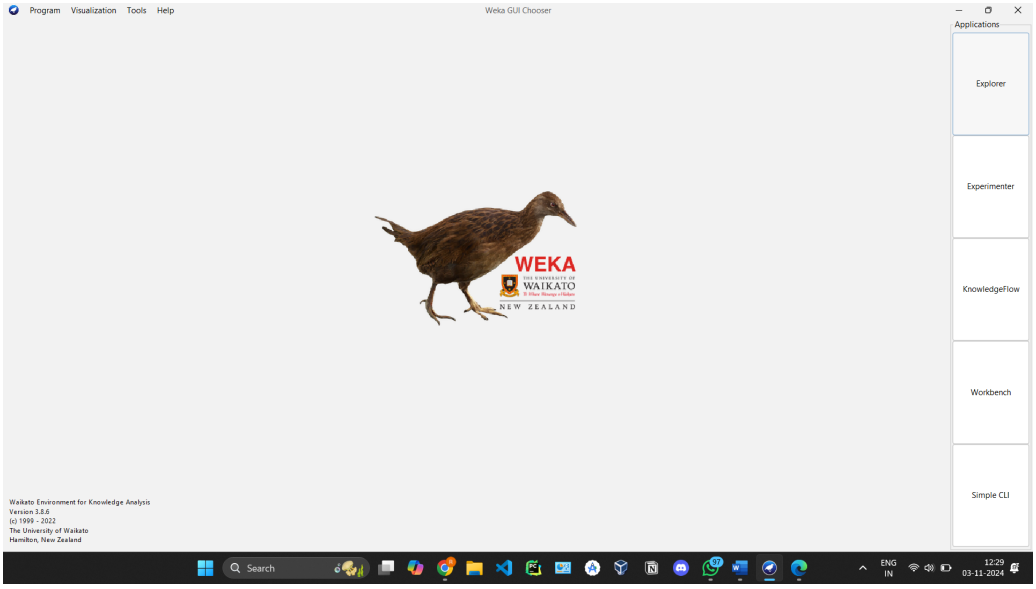
Key Concepts in Classification

1. **Supervised Learning**:
    - In supervised learning, models are trained on a dataset with known labels. Each instance in the training set has features (input variables) and a target class label (output variable), allowing the model to learn from examples.
2. **Types of Classification**:
    - **Binary Classification**: Involves two classes, such as "spam" vs. "not spam."
    - **Multiclass Classification**: Involves more than two classes, like classifying images as "cat," "dog," or "bird."
    - **Multilabel Classification**: Instances may belong to multiple classes at once, such as a news article categorized under "politics" and "finance."
3. **Common Classification Algorithms**:
    - **Decision Trees**: Uses a tree-like model to make decisions based on attribute values, with nodes representing features and branches indicating decision outcomes. Examples include the CART and C4.5 algorithms.
    - **Naive Bayes**: A probabilistic classifier based on Bayes' theorem, assuming feature independence within each class. It's fast and works well for text classification tasks.
    - **k-Nearest Neighbors (k-NN)**: A non-parametric method that classifies a point based on the majority label of its closest k neighbors in the feature space.
    - **Support Vector Machines (SVM)**: Finds a hyperplane that maximally separates classes in a high-dimensional space, often effective for complex and high-dimensional data.
    - **Neural Networks**: Complex models inspired by the human brain, effective for large datasets and able to learn intricate patterns through multiple hidden layers.
4. **Training and Testing**:
    - **Training Set**: The portion of data used to fit the classification model.
    - **Testing Set**: A separate portion of data used to evaluate model performance. Typically, data is split into 70-80% for training and 20-30% for testing.
5. **Evaluation Metrics**:
    - **Accuracy**: Proportion of correctly classified instances out of all instances.

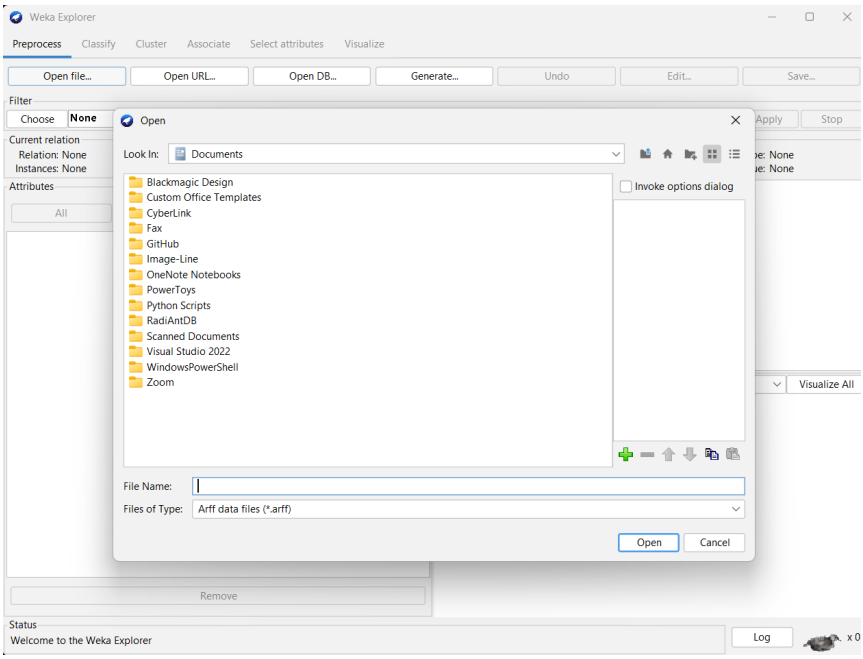
- **Precision**: Fraction of true positive predictions out of all positive predictions made (useful when false positives are costly).
- **Recall (Sensitivity)**: Fraction of true positive predictions out of all actual positives (useful when false negatives are costly).
- **F1 Score**: The harmonic mean of precision and recall, balancing both metrics.
- **Confusion Matrix**: A matrix summarizing correct and incorrect predictions for each class, providing insight into specific errors.

Steps:

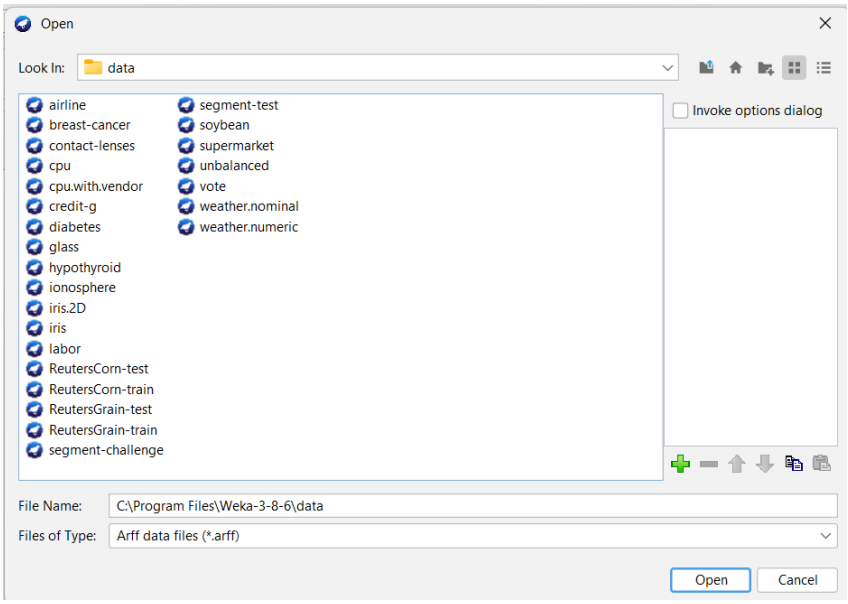
1. Launch the Weka GUI and click on the "Explorer" button to open the Weka Explorer.

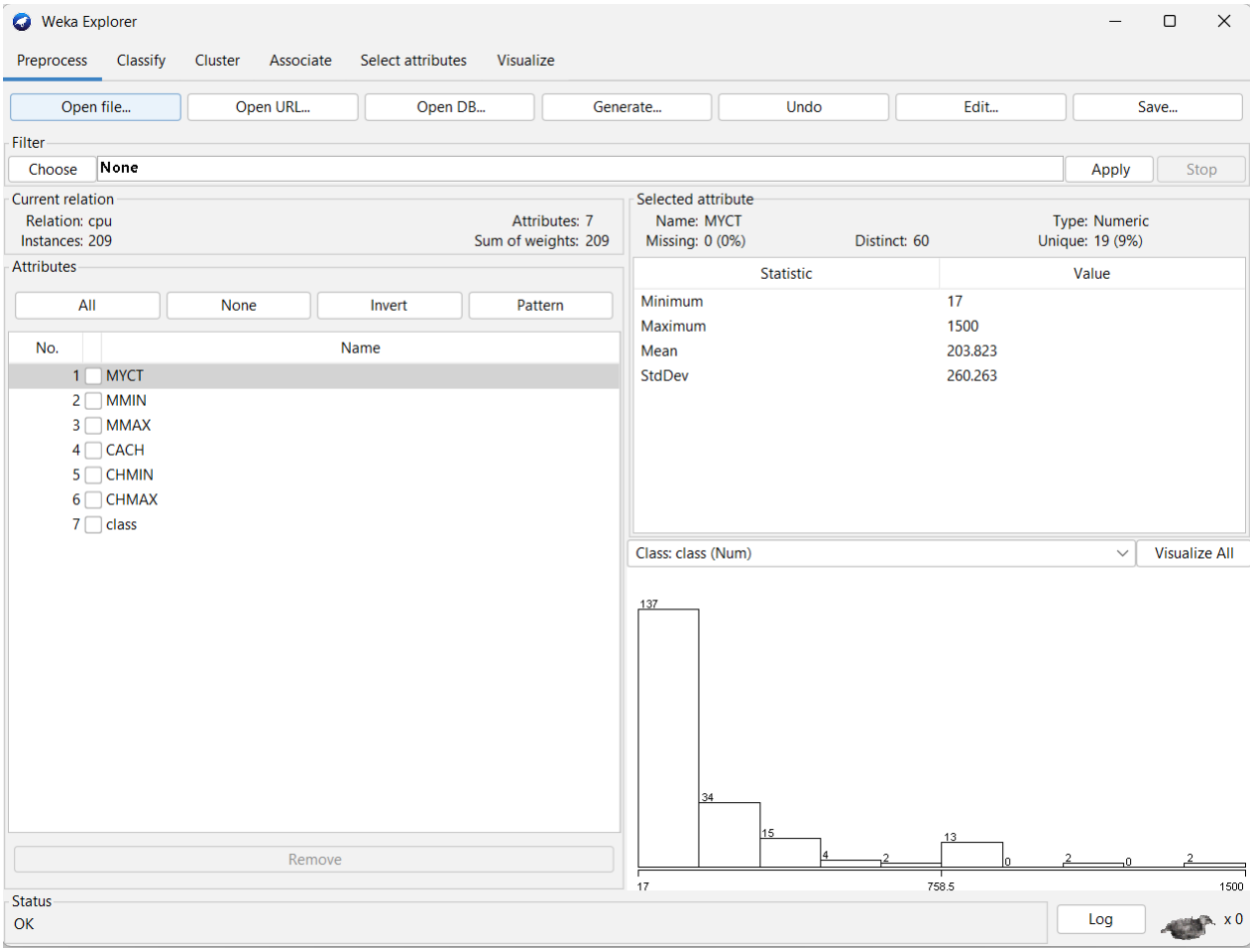


2. Click on the "Open file" button to load your dataset.

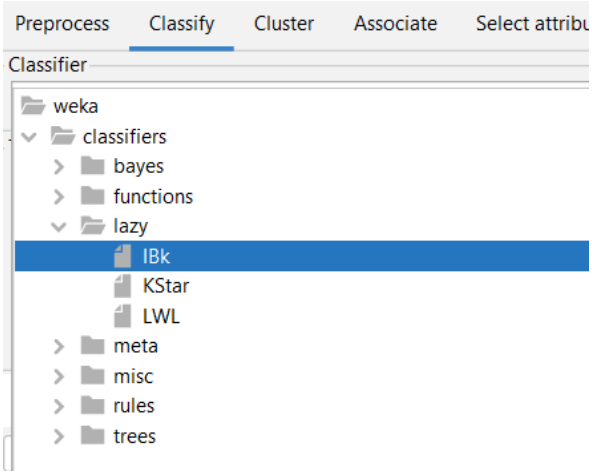


3. Select the dataset and click open.

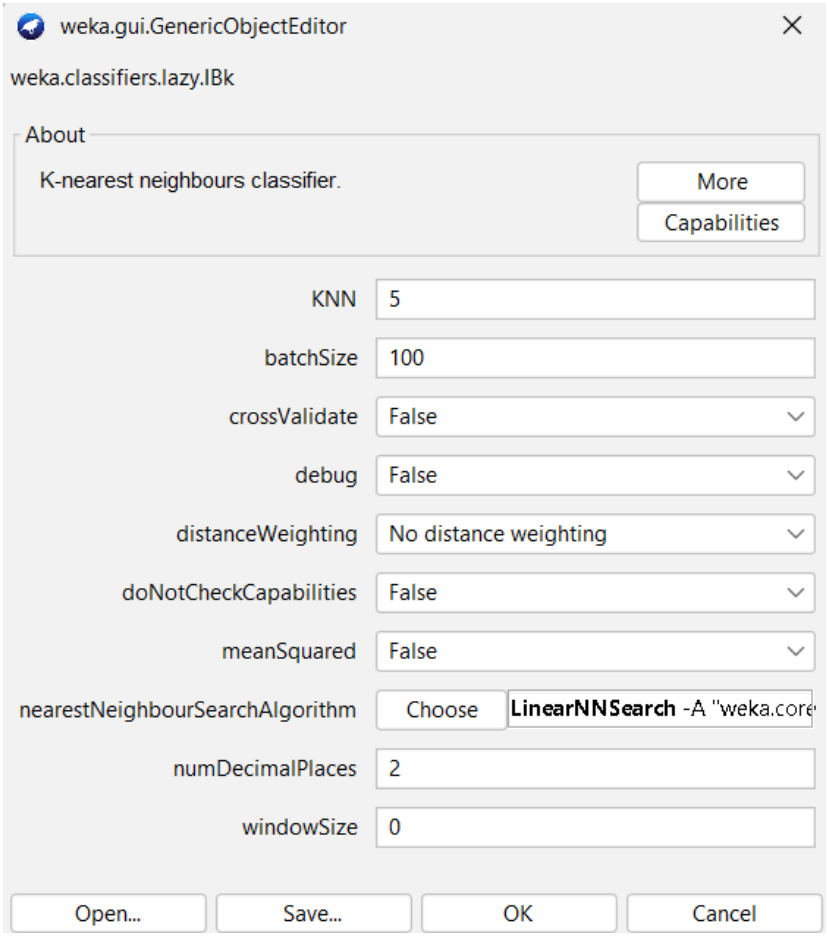




4. Click on “Classify” and choose the "IBK" algorithm from the "lazy" section of the classifiers.



5. Congifure the model by clicking on the “IBK” classifier.

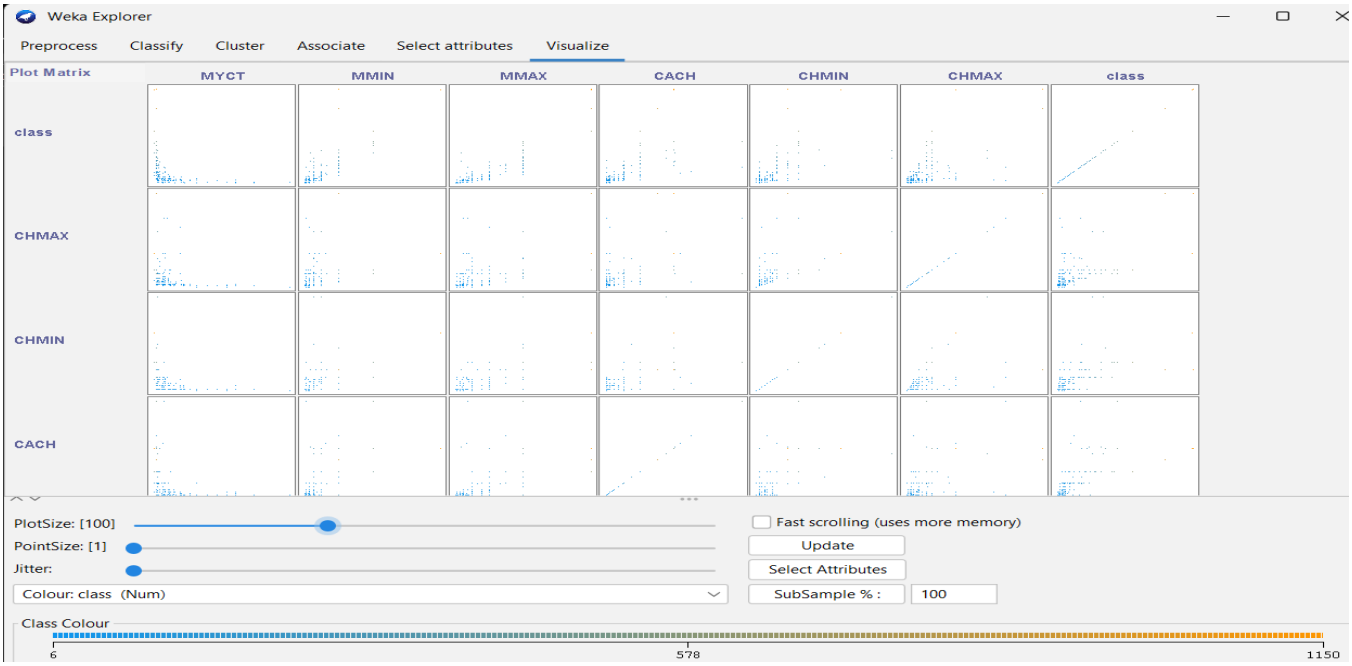


6. Click on “Start” and WEKA will provide K-NN summary.

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.lazy.IBk -K 5 -W 0 -A "weka
Relation:     cpu
Instances:    209
Attributes:   7
              MYCT
              MMIN
              MMAX
              CACH
              CHMIN
              CHMAX
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 5 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8892
Mean absolute error                     32.5101
Root mean squared error                 84.1436
Relative absolute error                 33.7805 %
Root relative squared error             52.3109 %
Total Number of Instances              209
```

7. We can visualize the linear regression model in the visualization tab.



Viva Questions

1. What is classification in machine learning?

Classification is a supervised learning technique used to assign data points into predefined classes or labels by learning patterns from a labeled dataset.

2. What is supervised learning?

Supervised learning involves training a model on a dataset where each instance has input features and a known target label, enabling the model to learn from these examples.

3. What are the types of classification?

Binary classification has two classes (e.g., “spam” vs. “not spam”), multiclass classification involves more than two classes (e.g., “cat,” “dog,” or “bird”), and multilabel classification allows instances to belong to multiple classes simultaneously (e.g., a news article categorized as “politics” and “finance”).

4. What are some common classification algorithms?

Decision Trees use tree-like models to make decisions, Naive Bayes is a probabilistic classifier based on Bayes’ theorem assuming feature independence, k-Nearest Neighbors classifies based on the majority label of the closest k neighbors, Support Vector Machines find a hyperplane that separates classes, and Neural Networks are complex models inspired by the human brain that work well for large datasets.

5. What is the difference between a training set and a testing set?

A training set is used to fit the classification model, while a testing set is a separate portion of data used to evaluate the model’s performance. Typically, 70-80% of the data is used for training and 20-30% for testing.

Experiment – 6

Aim: Implementation of Clustering technique on ARFF files using WEKA.

Theory: Clustering is an unsupervised machine learning technique used to group similar data points into clusters. The objective is to organize a dataset into distinct groups where data points in the same group (or cluster) are more similar to each other than to those in other clusters. Unlike classification, clustering does not rely on labeled data; instead, it explores inherent structures within the data itself.

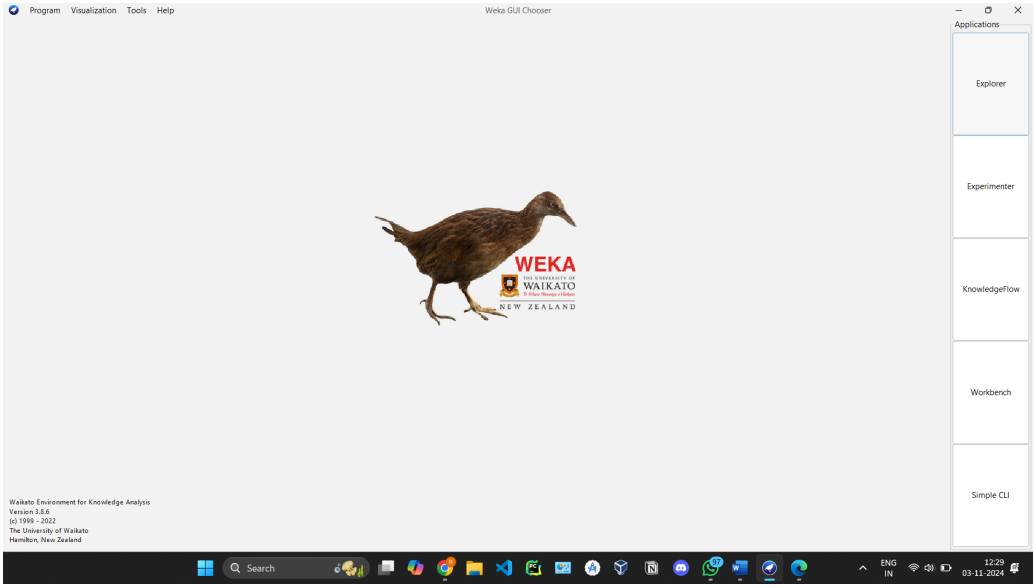
Key Concepts in Clustering

1. **Unsupervised Learning**:
    - Clustering is unsupervised, meaning there is no predefined target variable or label. The algorithm identifies patterns and structures based on feature similarities without external guidance.
2. **Types of Clustering Algorithms**:
    - Clustering methods vary in how they define and identify clusters:
    - **k-Means Clustering**:
        - Partitions data into *k* clusters by minimizing the variance within each cluster.
        - Starts by selecting *k* initial centroids (center points), assigns each data point to the nearest centroid, and iteratively updates centroids based on cluster members.
    - **Hierarchical Clustering**:
        - Builds a tree-like structure of nested clusters (dendrogram) through either:
            - **Agglomerative** (bottom-up): Each data point starts as its own cluster, which gradually merges into larger clusters.
            - **Divisive** (top-down): Starts with all data points in a single cluster and splits them into smaller clusters.
        - Often visualized through dendrograms, making it easy to see how clusters split or merge at each level.
    - **Density-Based Clustering (DBSCAN)**:
        - Groups points that are close to each other (dense regions) while marking points in low-density regions as outliers.
        - Effective for identifying arbitrarily shaped clusters and handling noise, unlike k-means, which assumes spherical clusters.
    - **Gaussian Mixture Models (GMM)**:
        - Assumes data is generated from multiple Gaussian distributions and uses probabilistic methods to assign data points to clusters.
        - Flexible and allows for overlapping clusters, making it useful for complex distributions.

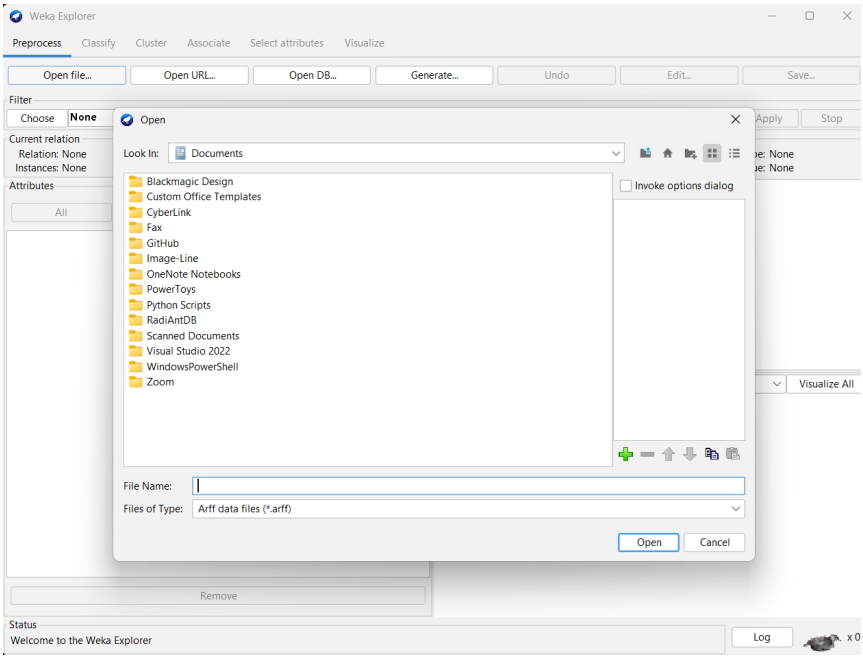
3. **Distance Measures**:
    - Clustering relies on measuring similarity between points, often using:
        - **Euclidean Distance**: Common for k-means, measures straight-line distance.
        - **Manhattan Distance**: Sum of absolute differences, useful in high-dimensional data.
        - **Cosine Similarity**: For text or sparse data, measuring angle between vectors rather than direct distance.

Steps:

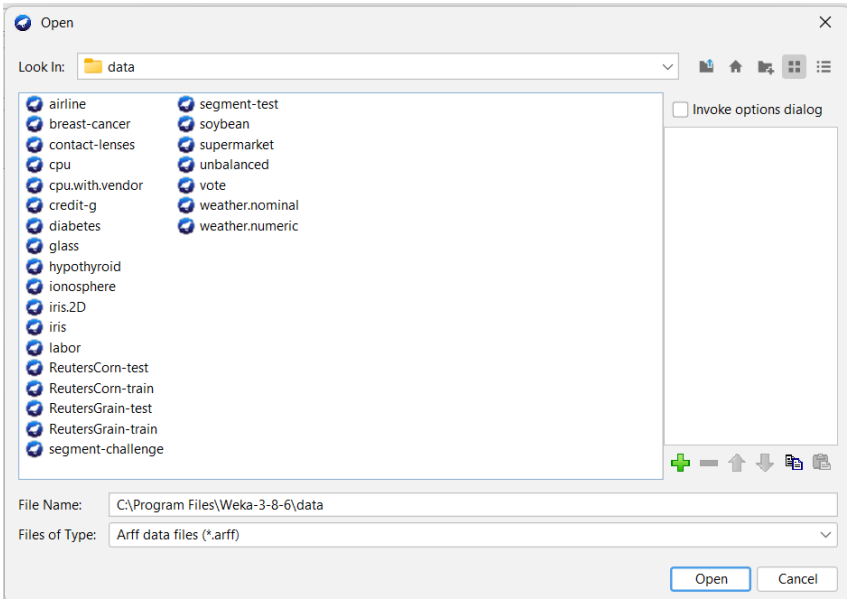
1. Launch the Weka GUI and click on the "Explorer" button to open the Weka Explorer.



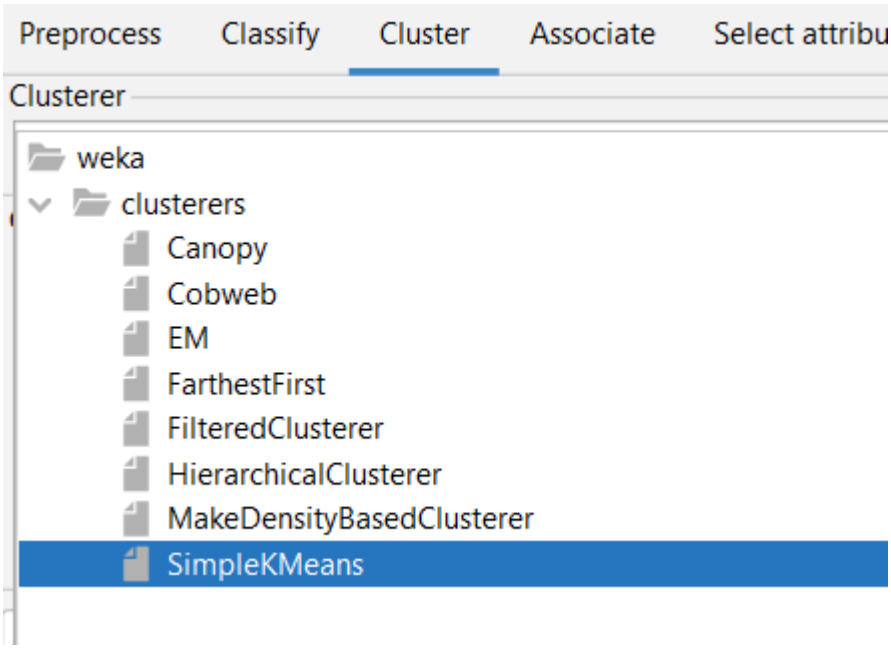
2. Click on the "Open file" button to load your dataset.



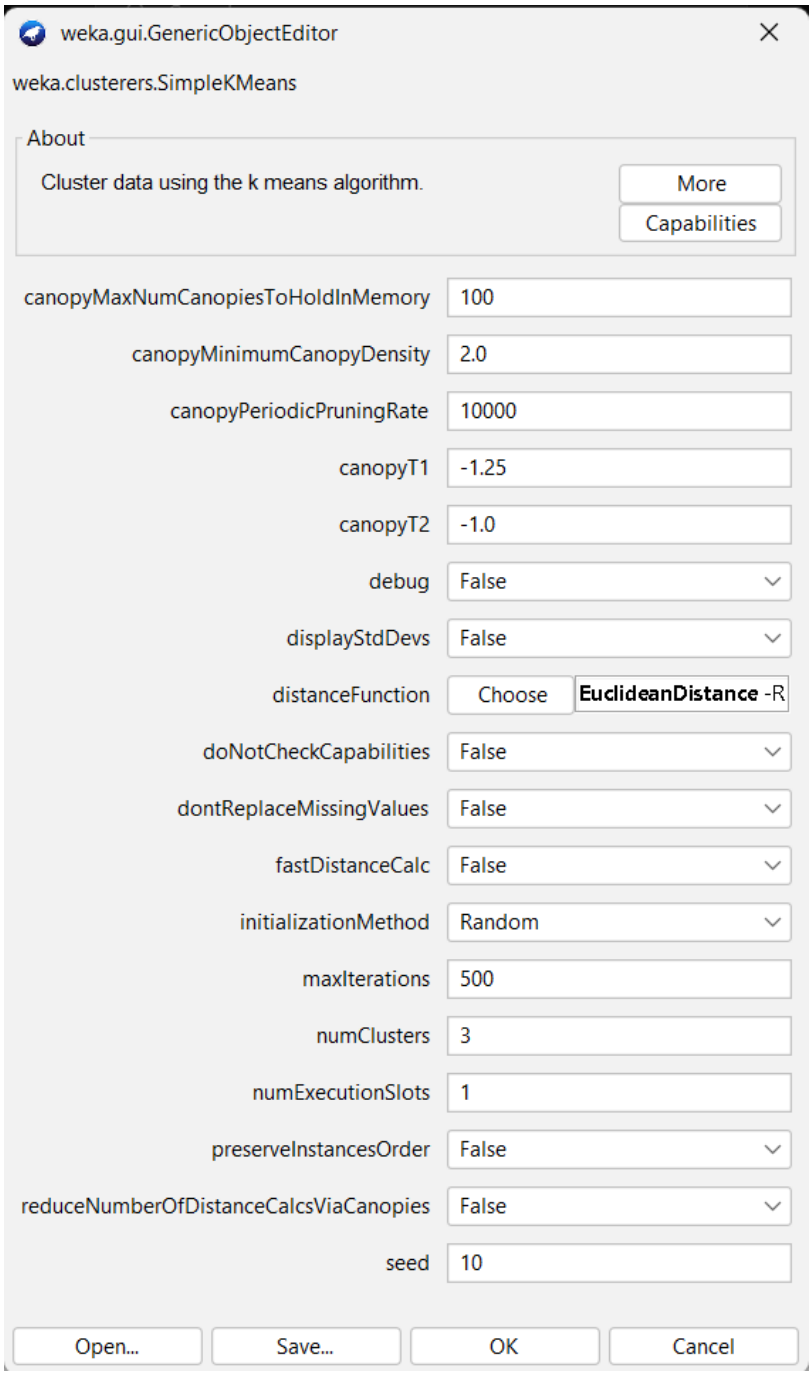
3. Select the dataset and click open.



4. Click on “Cluster” and choose the "SimpleKMeans" algorithm from the "clusterers".



5. Congifure the model by clicking on “SimpleKMeans”.

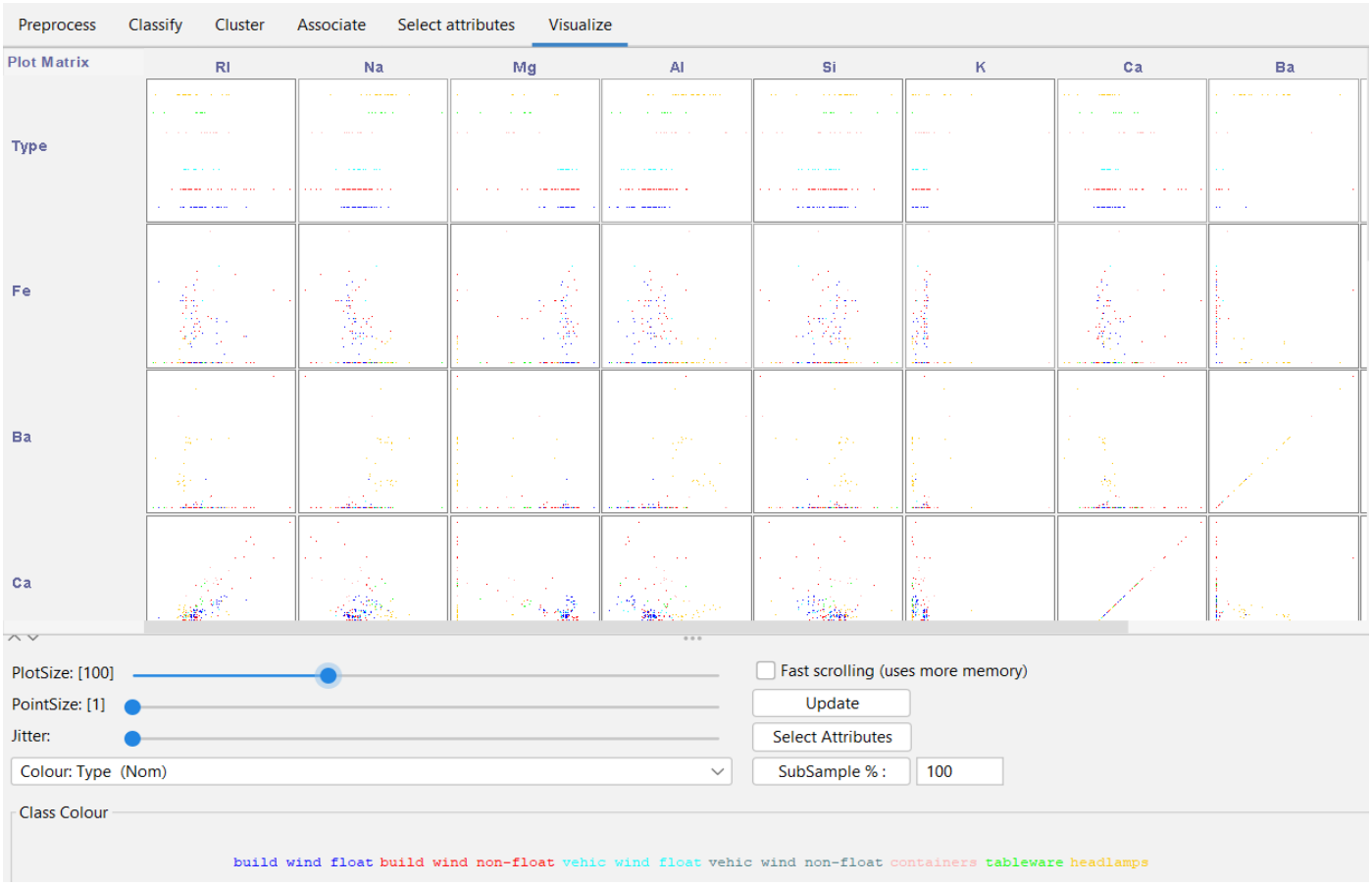


6. Click on “Start” and WEKA will provide K-Means summary.

Clusterer output

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -perio
Relation:     Glass
Instances:    214
Attributes:   10
              RI
              Na
              Mg
              Al
              Si
              K
              Ca
              Ba
              Fe
              Type
Test mode:    evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 77.12426898755668

Initial starting points (random):

Cluster 0: 1.52152,13.05,3.65,0.87,72.32,0.19,9.85,0,0.17,'build wind float'
Cluster 1: 1.51618,13.53,3.55,1.54,72.99,0.39,7.78,0,0,'build wind float'
Cluster 2: 1.51316,13.02,0,3.04,70.48,6.21,6.96,0,0,containers
Missing values globally replaced with mean/mode

Final cluster centroids:
                                                Cluster#
Attribute                           Full Data          0                  1                  2
                                      (214.0)       (87.0)             (84.0)             (43.0)
=============================================================================================================
RI                                     1.5184       1.5187             1.5186             1.5172
Na                                    13.4079      13.1245            13.2686            14.2533
Mg                                     2.6845       2.9314             3.5514             0.4916
Al                                     1.4449        1.431             1.1618              2.026
Si                                    72.6509      72.5814            72.6005            72.8902
K                                      0.4971        0.499             0.4385             0.6077
Ca                                      8.957       9.1571             8.7951             8.8681
Ba                                      0.175       0.0467             0.0124             0.7526
Fe                                      0.057       0.0828             0.0515             0.0156
Type                    build wind non-float build wind non-float  build wind float    headlamps



Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       87 ( 41%)
1       84 ( 39%)
2       43 ( 20%)
```

7. We can visualize the linear regression model in the visualization tab.



Viva Questions

1. What is clustering in machine learning?

Clustering is an unsupervised learning technique used to group similar data points into clusters based on feature similarities without predefined labels.

2. How does unsupervised learning differ from supervised learning?

Unsupervised learning, like clustering, doesn't use labeled data, instead finding patterns and structures in the data, whereas supervised learning requires labeled data for training models.

3. What are the types of clustering algorithms?

Common types include k-Means, which partitions data into k clusters; Hierarchical Clustering, which builds a tree of nested clusters; DBSCAN, which finds dense regions of points and identifies outliers; and Gaussian Mixture Models (GMM), which uses probabilistic models to assign data to clusters.

4. How does k-Means clustering work?

K-Means works by selecting k initial centroids, assigning each data point to the nearest centroid, and iteratively updating the centroids to minimize variance within clusters.

5. What is hierarchical clustering and how does it work?

Hierarchical clustering builds a tree of clusters using either an agglomerative (bottom-up) or divisive (top-down) approach, which is often visualized in a dendrogram to show how clusters merge or split.

Experiment – 7

Aim: Implementation of Association Rule technique on ARFF files using WEKA.

Theory: Association rule mining is a data mining technique used to discover interesting relationships, patterns, or associations among items in large datasets. It is widely used in fields like market basket analysis, where the goal is to understand how items co-occur in transactions. For instance, an association rule might identify that "Customers who buy bread also tend to buy butter." Such insights can support product placements, recommendations, and targeted marketing.

Key Concepts in Association Rule Mining

1. **Association Rules**:
    - An association rule is typically in the form of X→Y, which means "If itemset X is present in a transaction, then itemset Y is likely to be present as well."
    - Each association rule is evaluated based on three main metrics:
        - **Support**: The frequency with which an itemset appears in the dataset. Support for X→Y is calculated as:

$$\text{Support}(X \rightarrow Y) = \frac{\text{Number of transactions containing both } X \text{ and } Y}{\text{Total number of transactions}}$$

          Higher support indicates that the rule is relevant for a larger portion of the dataset.

        - **Confidence**: The likelihood of seeing Y in a transaction that already contains X. Confidence for X→Y is calculated as:

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Number of transactions containing both } X \text{ and } Y}{\text{Number of transactions containing } X}$$

          Higher confidence means the rule is more reliable.

        - **Lift**: Indicates the strength of an association by comparing the confidence of a rule with the expected confidence if X and Y were independent. Lift is calculated as:

$$\text{Lift}(X \rightarrow Y) = \frac{\text{Confidence}(X \rightarrow Y)}{\text{Support of } Y}$$

          A lift value greater than 1 indicates a positive association between X and Y.

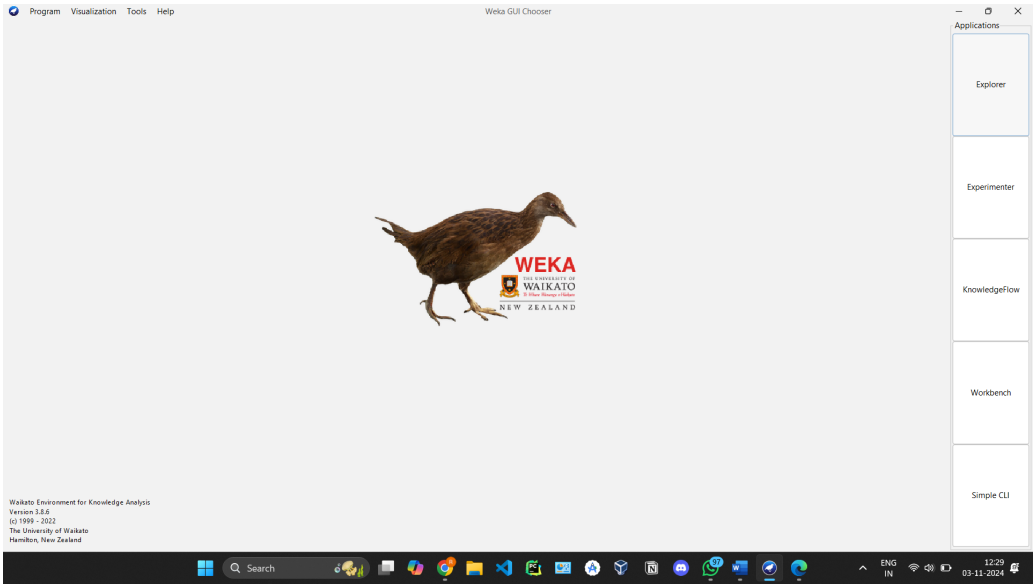
2. **Applications**:
    - **Market Basket Analysis**: Finds products frequently purchased together, guiding product placement and promotions.
    - **Recommendation Systems**: Suggests items based on co-occurrence with previously purchased items, useful in e-commerce.

- **Medical Diagnosis**: Identifies symptom or condition associations that commonly occur together.
- **Fraud Detection**: Detects patterns in fraudulent transactions by examining recurring behaviors or itemsets.

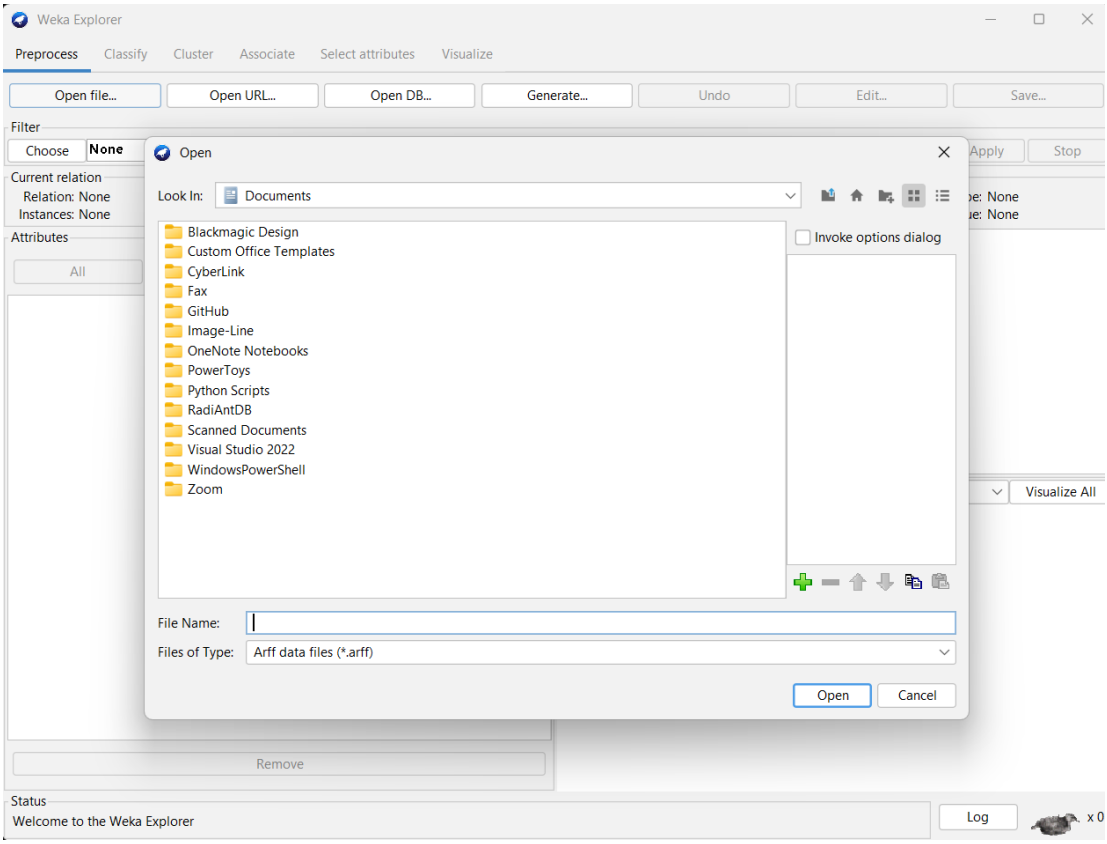
3. **Apriori Algorithm**:
   - The **Apriori algorithm** is a popular method for generating frequent itemsets and association rules.
   - **Frequent Itemset Generation**: Apriori starts by identifying single items that meet the minimum support threshold, then expands to larger itemsets, adding items iteratively while keeping only those itemsets that meet the support threshold.
   - **Rule Generation**: For each frequent itemset, rules are created by dividing the itemset into antecedents (left-hand side) and consequents (right-hand side) and calculating the confidence for each possible rule.
4. **Challenges**:
   - **Large Dataset Size**: Large datasets can generate a massive number of itemsets, making computation and rule filtering challenging.
   - **Choosing Thresholds**: Setting appropriate support and confidence thresholds is crucial, as too high a threshold may yield too few rules, while too low may produce many insignificant rules.
   - **Interpretability**: Association rules must be interpretable to provide value, requiring meaningful thresholds and metrics.

Steps:

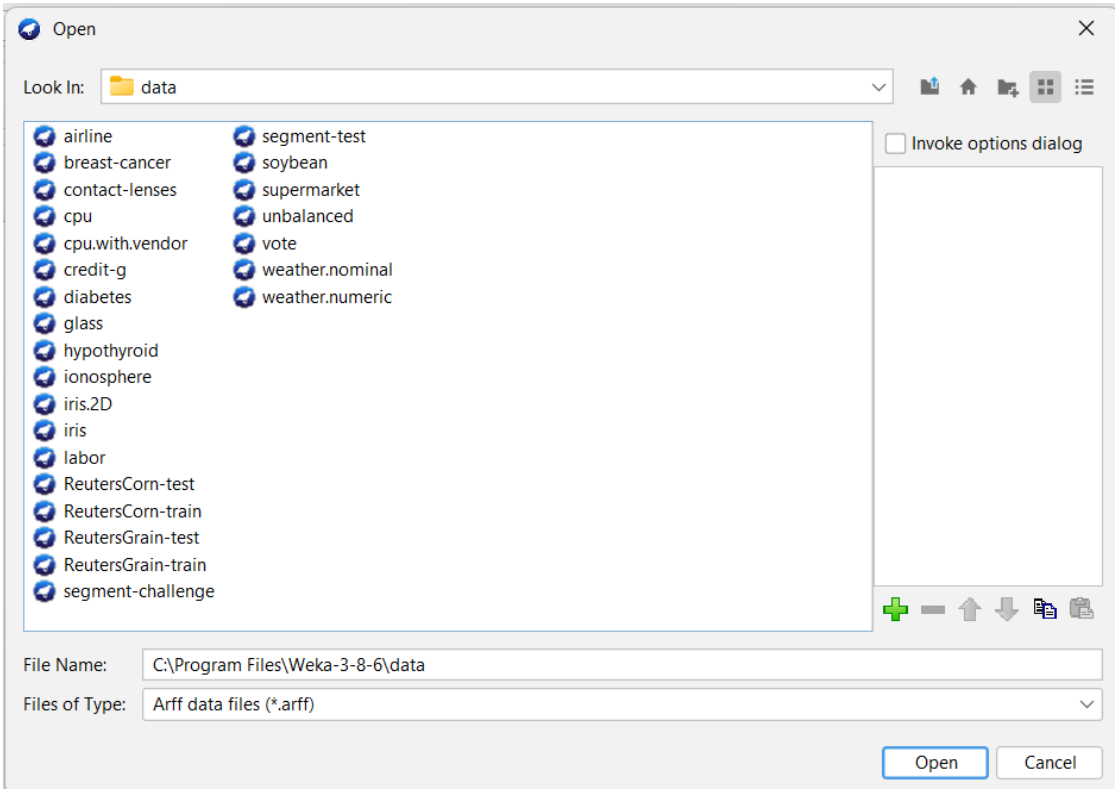
1. Launch the Weka GUI and click on the "Explorer" button to open the Weka Explorer.

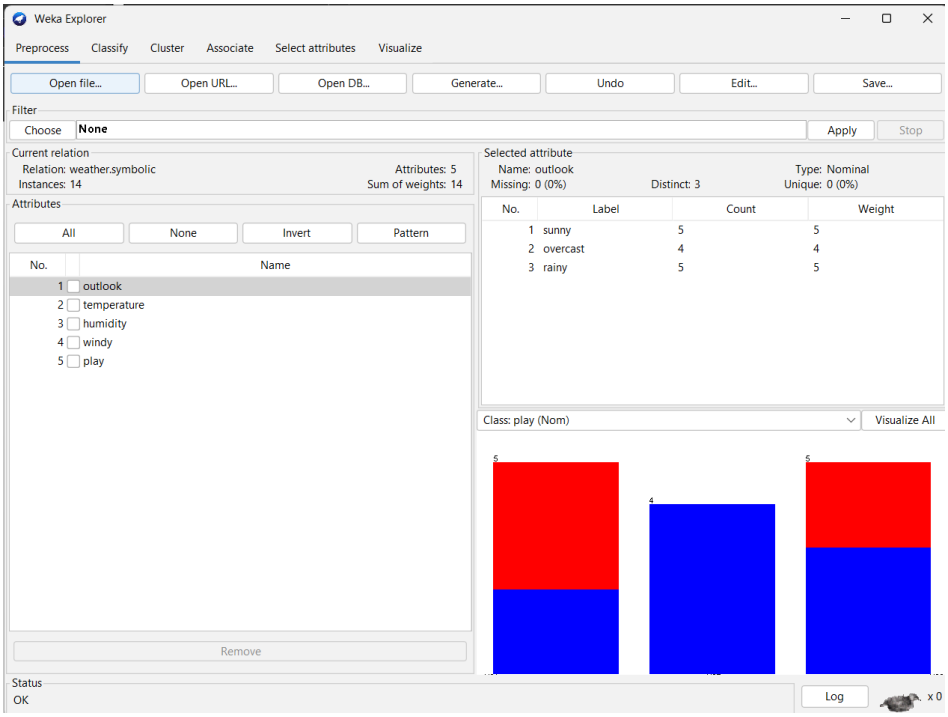


2. Click on the "Open file" button to load your dataset.

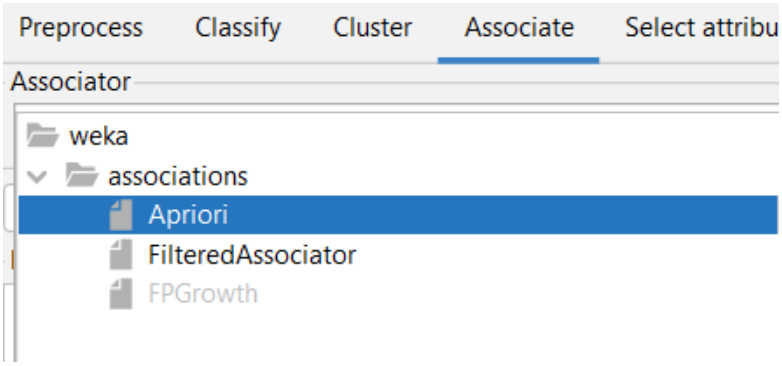


3. Select the dataset and click open.

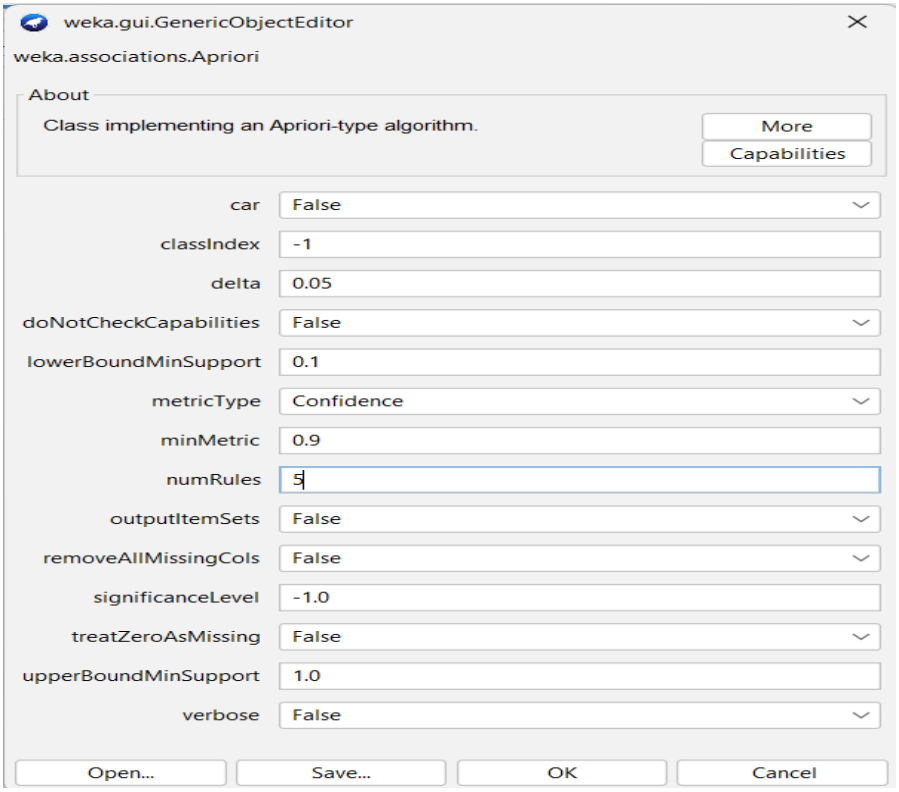




4. Click on “Associate” and choose the "Apriori" algorithm from the "associations" section.



5. Congifure the model by clicking on “Apriori”.



6. Click on “Start” and WEKA will provide Apriori summary.

Associator output

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 5 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     weather.symbolic
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.25 (3 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 15

Generated sets of large itemsets:

Size of set of large itemsets L(1): 12

Size of set of large itemsets L(2): 26

Size of set of large itemsets L(3): 4

Best rules found:

1. outlook=overcast 4 ==> play=yes 4    <conf:(1)> lift:(1.56) lev:(0.1) [1] conv:(1.43)
2. temperature=cool 4 ==> humidity=normal 4    <conf:(1)> lift:(2) lev:(0.14) [2] conv:(2)
3. humidity=normal windy=FALSE 4 ==> play=yes 4    <conf:(1)> lift:(1.56) lev:(0.1) [1] conv:(1.43)
4. outlook=sunny play=no 3 ==> humidity=high 3    <conf:(1)> lift:(2) lev:(0.11) [1] conv:(1.5)
5. outlook=sunny humidity=high 3 ==> play=no 3    <conf:(1)> lift:(2.8) lev:(0.14) [1] conv:(1.93)
```

Viva Questions

1. What is association rule mining?

Association rule mining is a data mining technique used to discover interesting relationships or patterns between items in large datasets, such as identifying which products are frequently purchased together in market basket analysis.

2. What is the form of an association rule?

An association rule is typically in the form X → Y, meaning "If itemset X is present in a transaction, then itemset Y is likely to be present as well."

3. What are the main metrics used to evaluate association rules?

The three main metrics are support, confidence, and lift. Support measures the frequency of itemsets, confidence measures the likelihood of Y occurring given X, and lift measures the strength of the association by comparing the confidence of the rule with the expected confidence if X and Y were independent.

4. What is support in association rule mining?

Support is the frequency with which an itemset appears in the dataset. It is calculated as the ratio of transactions containing both X and Y to the total number of transactions.

5. How is confidence calculated?

Confidence is the likelihood of seeing Y in a transaction that already contains X, calculated as the ratio of transactions containing both X and Y to the number of transactions containing X.

Experiment – 8

Aim: Implementation of Visualization technique on ARFF files using WEKA.

Theory: Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. This technique is essential in data analysis, making complex datasets understandable and actionable.

Importance of Data Visualization

1. **Enhances Understanding**:
    - Visualization helps in simplifying complex data by converting it into a visual format that is easier to interpret. This is particularly important for large datasets with many variables.
2. **Reveals Insights**:
    - Visualizations can uncover hidden insights, correlations, and trends that may not be apparent in raw data. For instance, scatter plots can reveal relationships between two variables, while heatmaps can show patterns across multiple dimensions.
3. **Facilitates Communication**:
    - Effective visualizations convey information clearly and efficiently to stakeholders, enabling better decision-making. Visual aids can enhance presentations and reports, making the data more engaging and comprehensible.
4. **Supports Exploration**:
    - Interactive visualizations allow users to explore data from different angles, facilitating discovery and hypothesis generation. Users can drill down into specific segments of data or adjust parameters to see how results change.

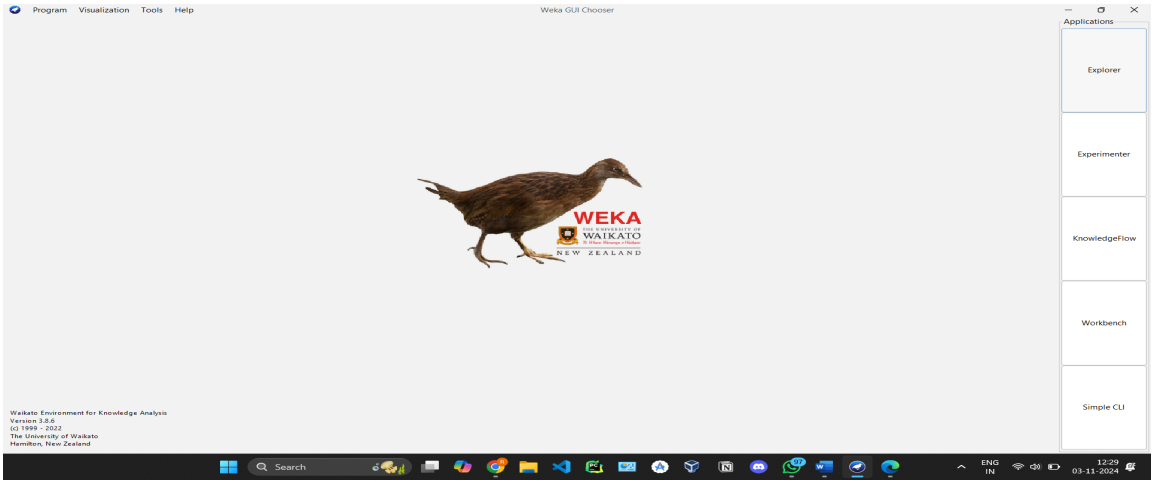
Common Visualization Techniques

1. **Bar Charts**:
    - Bar charts display categorical data with rectangular bars representing the values. They are effective for comparing different categories or showing changes over time.
2. **Line Graphs**:
    - Line graphs are used to display continuous data points over a period. They are ideal for showing trends and fluctuations in data, such as stock prices or temperature changes over time.
3. **Histograms**:
    - Histograms are similar to bar charts but are used for continuous data. They show the frequency distribution of numerical data by dividing the range into intervals (bins) and counting the number of observations in each bin.
4. **Scatter Plots**:
    - Scatter plots display the relationship between two continuous variables, with points plotted on an x and y axis. They are useful for identifying correlations, trends, and outliers.

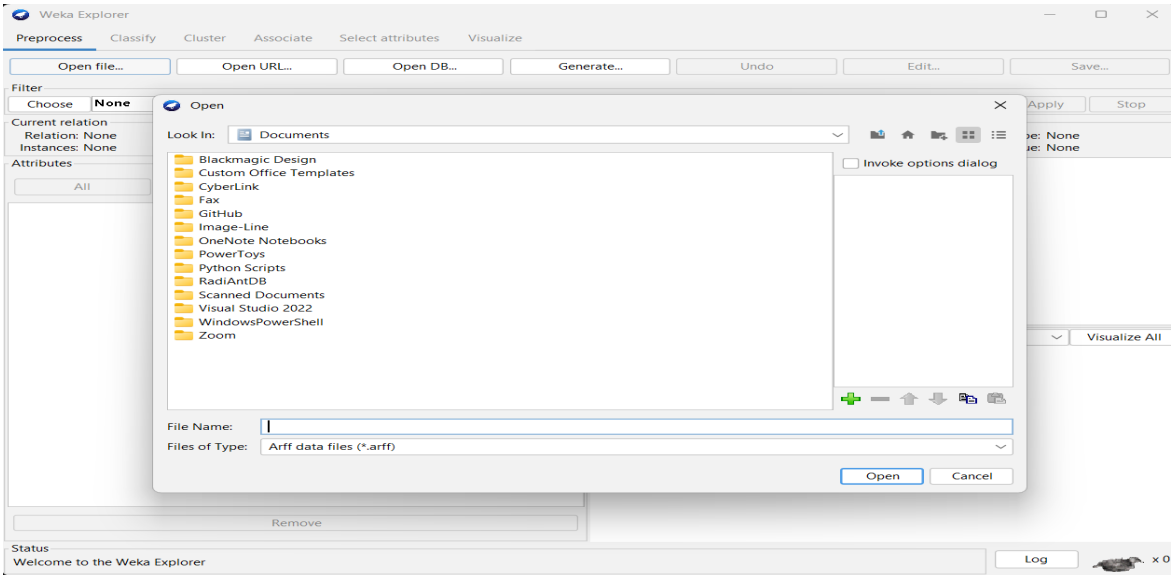
5. **Box Plots**:
    - Box plots summarize the distribution of a dataset by showing its median, quartiles, and potential outliers. They are effective for comparing distributions across categories.
6. **Heatmaps**:
    - Heatmaps display data in matrix form, where individual values are represented by colors. They are useful for visualizing correlations between variables or representing data density in geographical maps.
7. **Pie Charts**:
    - Pie charts represent data as slices of a circle, showing the proportion of each category relative to the whole. They are best for displaying percentage shares but can be less effective for comparing multiple values.
8. **Area Charts**:
    - Area charts display quantitative data visually over time, similar to line graphs, but fill the area beneath the line. They are useful for emphasizing the magnitude of values over time.

Steps:

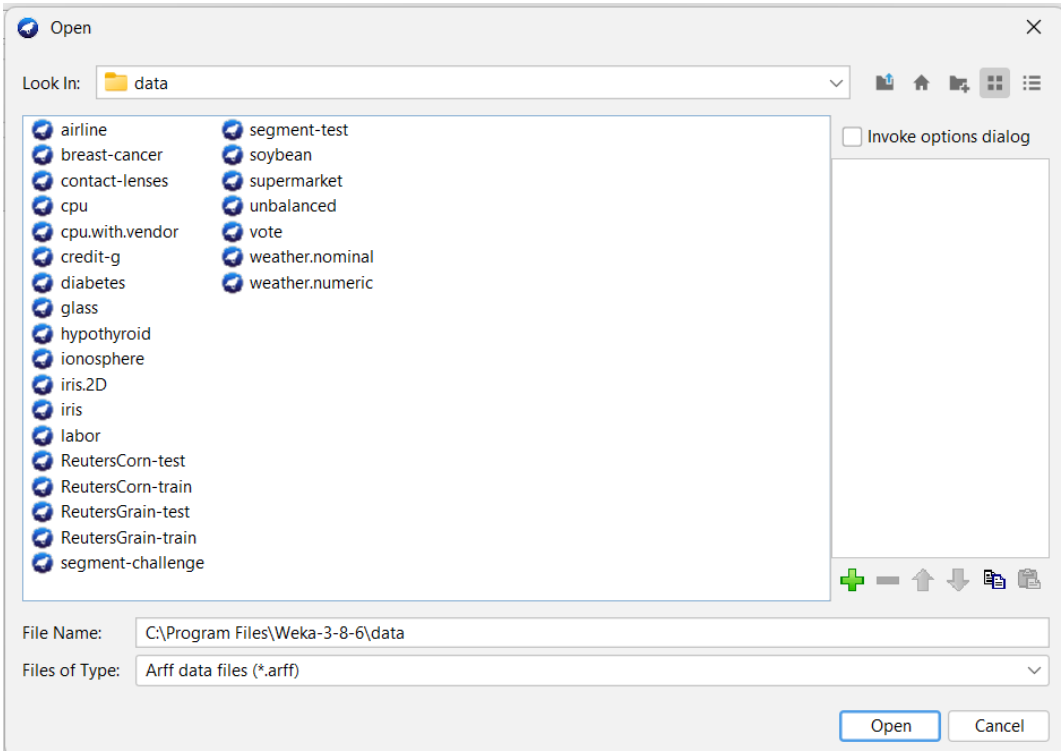
1. Launch the Weka GUI and click on the "Explorer" button to open the Weka Explorer.

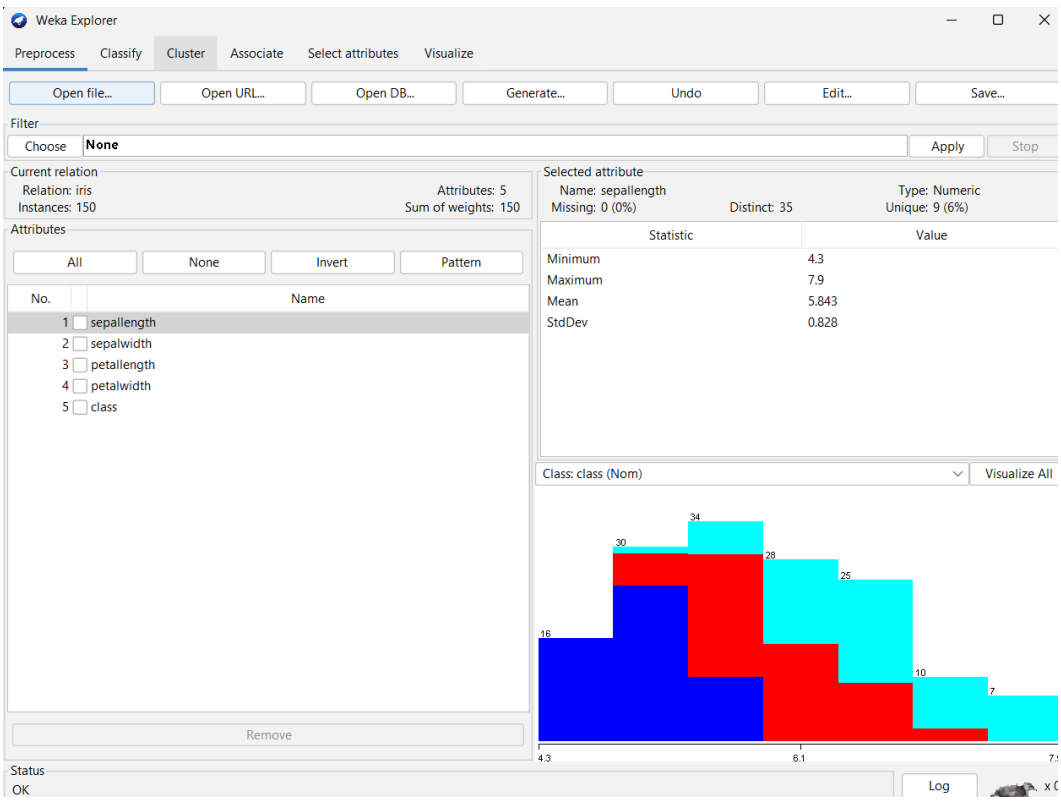


2. Click on the "Open file" button to load your dataset.

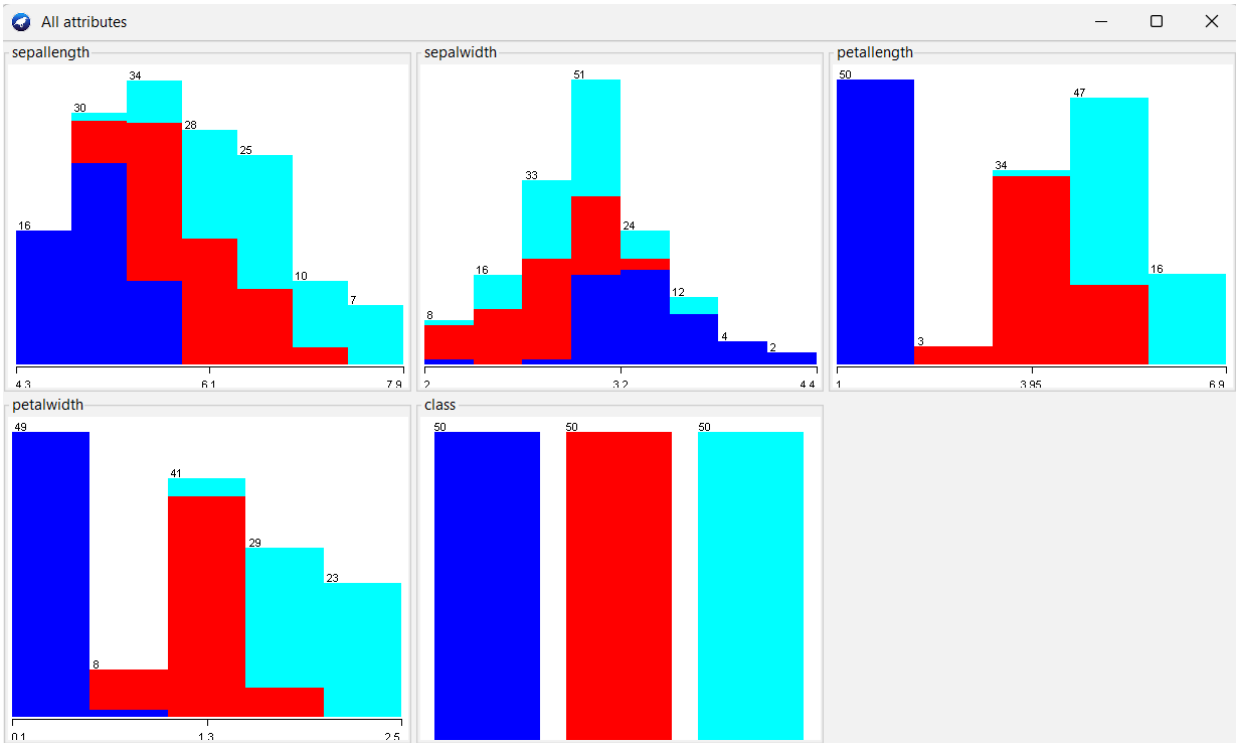


3. Select the dataset and click open.

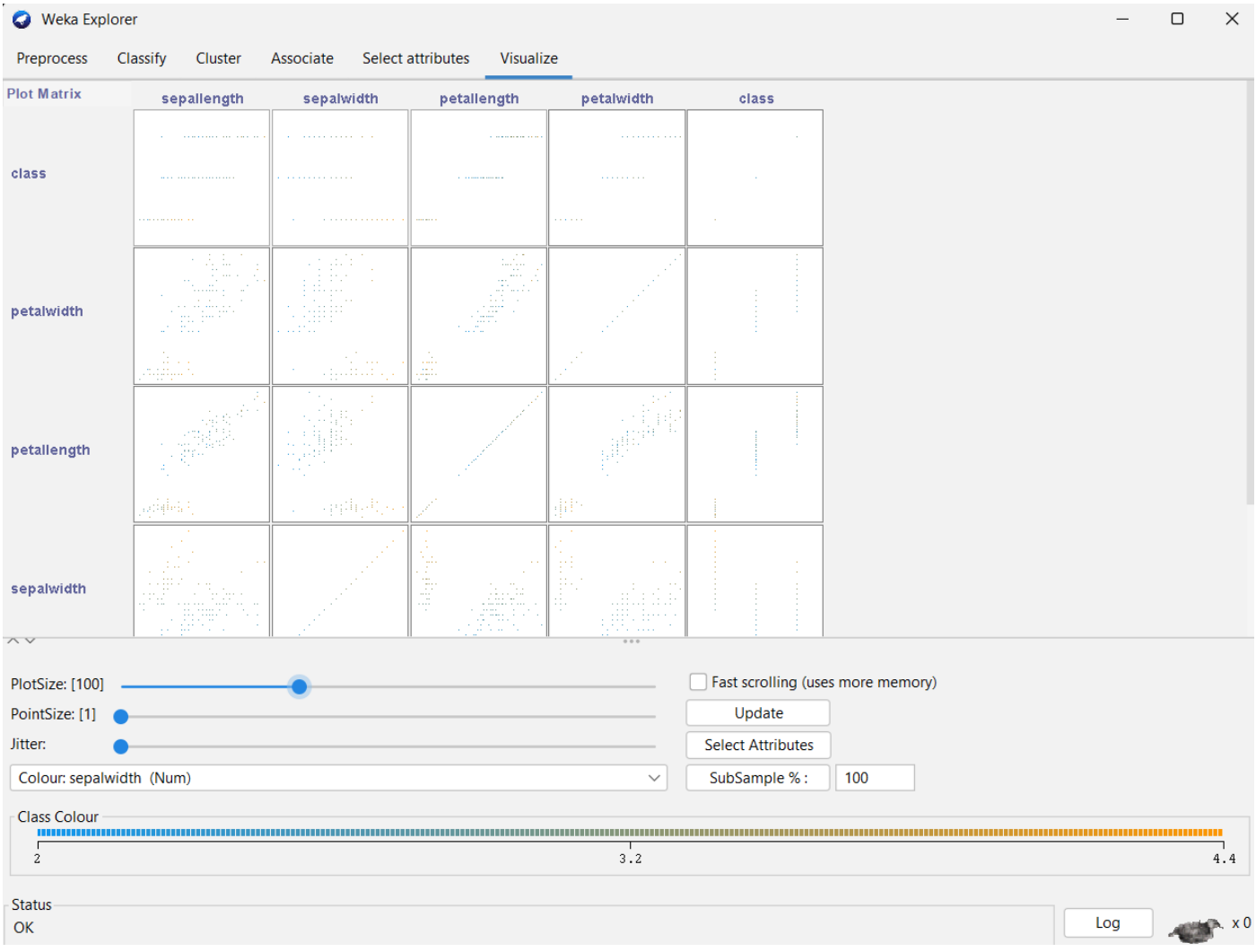




4. Click on “Visualize all” on the bottom right side for viewing all the attributes present in the dataset.

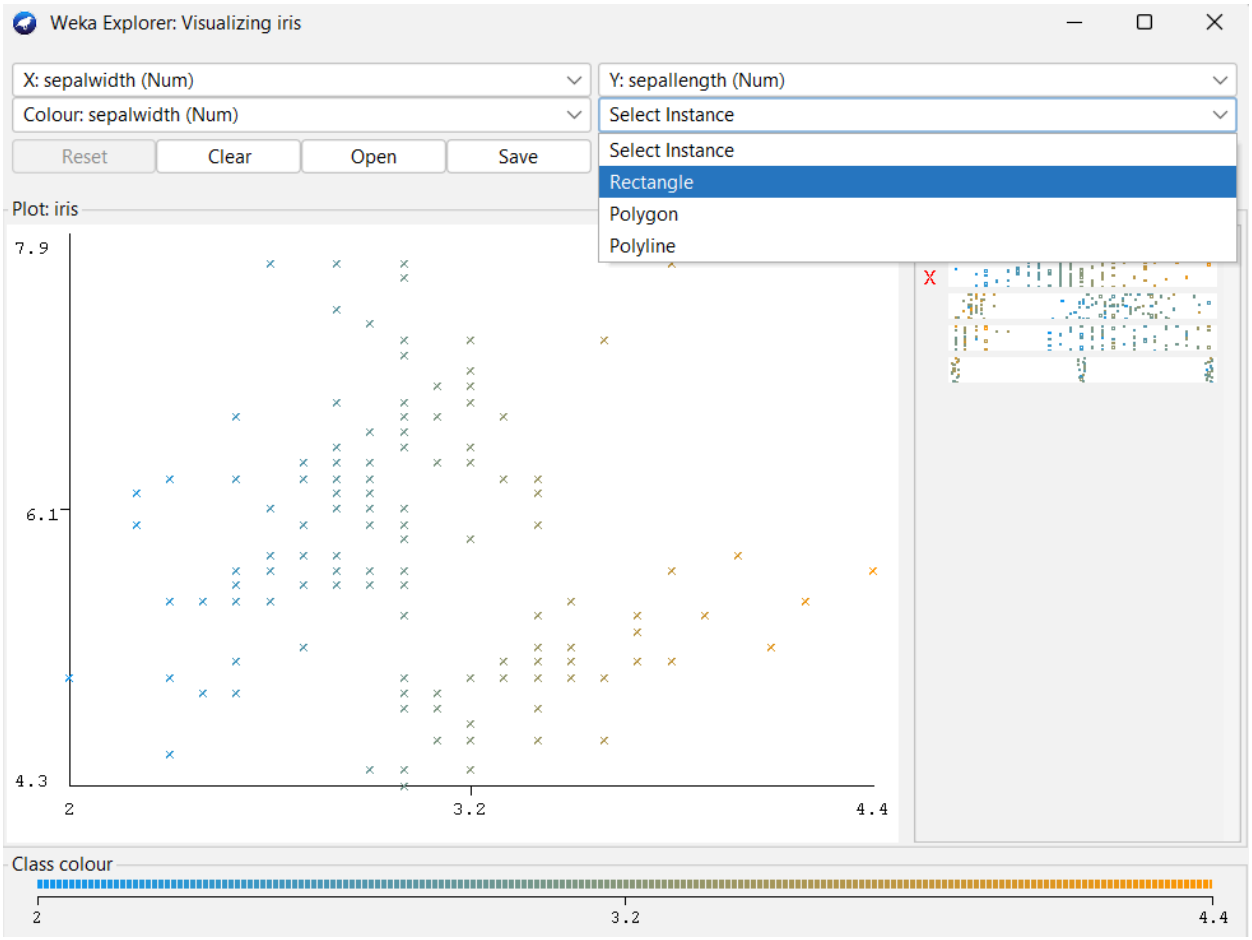


5. We can visualize the linear regression model in the visualization tab.

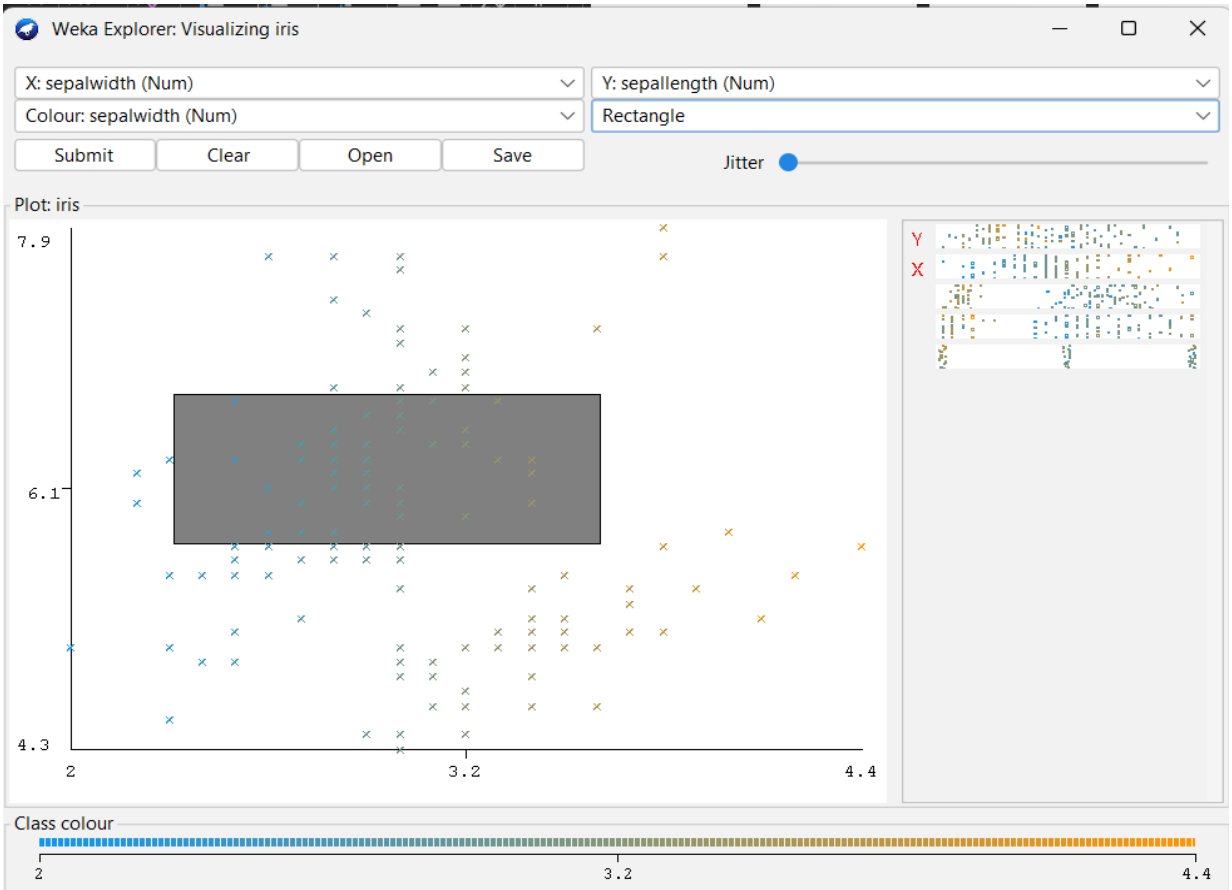


6. We can also visualize the scatter plot between any 2 attributes from the dataset.

7. We can also select a particular section of the scatter plot to visualize separately by clicking on “Select Instance”.



8. Select the area for visualization.



9. Click on “Submit”.

Viva Questions

1. What is data visualization?

Data visualization is the graphical representation of data using charts, graphs, and maps to help identify trends, outliers, and patterns, making complex data more understandable and actionable.

2. Why is data visualization important?

Data visualization helps in simplifying complex data, revealing hidden insights, enhancing communication, supporting exploration, and making data more accessible for decision-making.

3. How does data visualization enhance understanding?

By converting raw data into a visual format, data visualization makes it easier to interpret, especially for large datasets with many variables, allowing users to quickly grasp key trends and patterns.

4. What insights can be revealed through data visualization?

Visualizations can uncover relationships, correlations, trends, and outliers that might not be obvious in raw data. For example, scatter plots can reveal the relationship between two variables, while heatmaps show patterns across multiple dimensions.

5. How does data visualization facilitate communication?

Visual aids such as charts and graphs make it easier to present data to stakeholders, helping to convey complex information clearly and efficiently, leading to better decision-making.

Experiment – 9

Aim: Perform Data Similarity Measure (Euclidean, Manhattan Distance).

Theory: Data similarity measures quantify how alike two data points or vectors are. These measures are fundamental in various fields, including machine learning, data mining, and pattern recognition. Among the most commonly used similarity measures are **Euclidean distance** and **Manhattan distance**, which are often used to assess the closeness of points in a multidimensional space.

1. Euclidean Distance

Definition: Euclidean distance is the straight-line distance between two points in Euclidean space. It is calculated using the Pythagorean theorem and is one of the most commonly used distance measures in mathematics and machine learning.

Formula: For two points P and Q in an n-dimensional space, where P=(p1,p2,…,pn) and Q=(q1,q2,…,qn), the Euclidean distance d is given by:

$$d(P,Q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Properties:

- **Non-negativity**: d(P,Q)≥0
- **Identity**: d(P,Q)=0 if and only if P=Q
- **Symmetry**: d(P,Q)=d(Q,P)
- **Triangle Inequality**: d(P,R)≤d(P,Q)+d(Q,R) for any points P,Q,R

Applications:

- Used in clustering algorithms like K-means to determine the distance between points and centroids.
- Commonly applied in image recognition, pattern matching, and other areas requiring spatial analysis.

2. Manhattan Distance

Definition: Manhattan distance, also known as the "city block" or "taxicab" distance, measures the distance between two points by summing the absolute differences of their coordinates. It reflects the total grid distance one would need to travel in a grid-like path.

Formula: For two points P and Q in an n-dimensional space, the Manhattan distance d is calculated as:

$$d(P,Q) = \sum_{i=1}^{n}|p_i - q_i|$$

Properties:

- **Non-negativity**: d(P,Q)≥0
- **Identity**: d(P,Q)=0 if and only if P=Q

- **Symmetry**: d(P,Q)=d(Q,P)
- **Triangle Inequality**: d(P,R)≤d(P,Q)+d(Q,R)

Applications:

- Useful in scenarios where movement is restricted to a grid, such as in geographical data analysis.
- Often employed in clustering algorithms and machine learning models where linear relationships are more meaningful than straight-line distances.

Comparison of Euclidean and Manhattan Distance

1. **Geometric Interpretation**:
    - Euclidean distance measures the shortest path between two points, while Manhattan distance measures the total path required to travel along axes.
2. **Sensitivity to Dimensions**:
    - Euclidean distance can be sensitive to the scale of data and the number of dimensions, as it tends to emphasize larger values. In contrast, Manhattan distance treats all dimensions equally, summing absolute differences.
3. **Use Cases**:
    - Euclidean distance is preferred in applications involving continuous data and geometric spaces, whereas Manhattan distance is favored in discrete settings, such as grid-based environments.

Code:

```
!pip install liac-arff pandas scipy
import arff
from google.colab import files
import pandas as pd
from scipy.spatial import distance

#Step 1: Upload and Load ARFF File
uploaded =  files.upload()
arff_file = 'diabetes.arff'
#Load the ARFF file
with open(arff_file, 'r') as f:
  dataset =  arff.load(f)

data = pd.DataFrame(dataset ['data'], columns=[attr[0] for attr in dataset ['attributes']])

print("Dataset preview:")
print(data.head())

#Step 2: Select only numeric columns (Exclude categorical/string columns)
numeric_data = data.select_dtypes (include=[float, int])
print("\nNumeric columns:")
print(numeric_data.head())

#Step 3: Compute Euclidean and Manhattan Distances
point1 = numeric_data.iloc[0].values #First numeric data point
```

```
point2 = numeric_data.iloc[1].values #Second numeric data point

euclidean_dist = distance.euclidean(point1, point2)
print(f'\nEuclidean Distance between the first two points: {euclidean_dist}')

manhattan_dist = distance.cityblock(point1, point2)
print(f'Manhattan Distance between the first two points: {manhattan_dist}')

#Step 4: Compute Pairwise Distance Matrices
euclidean_dist_matrix = distance.squareform(distance.pdist(numeric_data.values, metric='euclidean'))
euclidean_dist_df = pd. DataFrame (euclidean_dist_matrix)
print("\nEuclidean Distance Matrix:")
print(euclidean_dist_df)

manhattan_dist_matrix = distance.squareform(distance.pdist(numeric_data.values, metric='cityblock'))
manhattan_dist_df = pd.DataFrame (manhattan_dist_matrix)
print("\nManhattan Distance Matrix:")
print(manhattan_dist_df)
```

Output:

```
Dataset preview:
   preg   plas  pres  skin   insu  mass   pedi   age            class
0   6.0  148.0  72.0  35.0    0.0  33.6  0.627  50.0  tested_positive
1   1.0   85.0  66.0  29.0    0.0  26.6  0.351  31.0  tested_negative
2   8.0  183.0  64.0   0.0    0.0  23.3  0.672  32.0  tested_positive
3   1.0   89.0  66.0  23.0   94.0  28.1  0.167  21.0  tested_negative
4   0.0  137.0  40.0  35.0  168.0  43.1  2.288  33.0  tested_positive

Numeric columns:
   preg   plas  pres  skin   insu  mass   pedi   age
0   6.0  148.0  72.0  35.0    0.0  33.6  0.627  50.0
1   1.0   85.0  66.0  29.0    0.0  26.6  0.351  31.0
2   8.0  183.0  64.0   0.0    0.0  23.3  0.672  32.0
3   1.0   89.0  66.0  23.0   94.0  28.1  0.167  21.0
4   0.0  137.0  40.0  35.0  168.0  43.1  2.288  33.0

Euclidean Distance between the first two points: 66.90348403483932
Manhattan Distance between the first two points: 106.276
```

Euclidean Distance Matrix:

	0	1	2	3	4	5
0	0.000000	66.903484	54.296335	115.730124	172.589133	52.136182
1	66.903484	0.000000	102.518257	94.817107	178.664496	43.405328
2	54.296335	102.518257	0.000000	135.640315	180.556505	67.878655
3	115.730124	94.817107	135.640315	0.000000	94.733830	101.297834
4	172.589133	178.664496	180.556505	94.733830	0.000000	177.173942
..	...	...	...	...	...	...
763	187.028602	185.099223	206.468910	100.663002	63.352361	190.000766
764	36.046114	38.859235	68.770344	100.347994	171.745407	30.404923
765	117.783301	118.021910	130.393989	38.530716	69.884074	114.465549
766	43.581272	53.162487	59.877745	107.004828	174.183121	24.925327
767	61.719829	12.745246	96.319974	94.558510	177.037235	39.925593

	6	7	8	9	...	758	\
0	117.170404	89.208369	545.333687	58.984032	...	60.178027	
1	90.035385	79.105860	555.126334	68.018925	...	39.061922	
2	141.807862	94.219369	545.559441	73.587252	...	79.748766	
3	23.052474	120.804971	463.460635	116.140493	...	99.334591	
4	100.987978	178.085597	381.827575	187.338830	...	178.302362	
..	...	...	...	...	...	...	
763	106.520495	204.542811	375.765565	191.956802	...	190.263398	
764	100.700787	75.817494	549.102653	59.441161	...	31.992350	
765	55.073042	135.653317	438.841085	122.214730	...	116.076235	
766	107.785993	64.444443	549.592755	47.968987	...	33.937930	
767	91.560715	80.535972	553.861025	67.787660	...	35.006056	

Manhattan Distance Matrix:

	0	1	2	3	4	5	6	7
0	0.000	106.276	108.345	210.960	245.161	98.426	212.979	167.193
1	106.276	0.000	140.621	115.684	273.437	74.150	125.503	144.917
2	108.345	140.621	0.000	236.305	303.416	84.771	258.124	149.538
3	210.960	115.684	236.305	0.000	190.121	167.534	51.981	233.233
4	245.161	273.437	303.416	190.121	0.000	285.587	176.140	288.954
..	...	...	...	...	...	...	...	...
763	262.156	272.480	365.101	188.804	149.317	290.330	202.977	354.437
764	66.487	58.211	118.832	150.873	237.248	54.339	164.892	115.706
765	179.782	165.506	213.327	70.978	142.943	142.644	108.803	228.211
766	80.778	95.502	90.123	188.182	263.939	49.648	202.001	103.415
767	96.512	25.836	150.457	114.448	271.673	73.914	129.667	143.081

	8	9	...	758	759	760	761	762	\
0	614.569	121.995	...	114.330	115.249	136.339	48.624	135.585	
1	702.093	155.719	...	76.054	208.973	42.215	132.452	51.309	
2	642.714	135.740	...	116.675	83.594	164.194	86.969	99.330	
3	618.409	251.165	...	158.430	301.511	92.899	237.136	146.625	
4	511.730	345.156	...	285.691	356.610	257.222	260.785	304.746	
..	...	...	...	...	...	...	...	...	
763	488.413	315.961	...	283.626	342.707	271.095	300.332	295.429	
764	668.482	125.908	...	51.843	161.362	76.826	86.263	95.498	
765	561.387	216.213	...	173.348	270.333	159.721	205.958	187.803	
766	676.591	81.217	...	64.552	125.471	110.117	114.954	68.807	
767	692.257	157.483	...	60.218	203.137	42.451	122.688	69.073	

Viva Questions

1. What are data similarity measures?

Data similarity measures quantify how alike two data points or vectors are, and they are widely used in fields like machine learning, data mining, and pattern recognition to assess the closeness between data points.

2. What is Euclidean distance?

Euclidean distance is the straight-line distance between two points in Euclidean space, calculated using the Pythagorean theorem. It is commonly used to measure spatial similarity in multidimensional space.

3. What is the formula for Euclidean distance?

The Euclidean distance d between two points $P = (p_1, p_2, \ldots, p_n)$ and $Q = (q_1, q_2, \ldots, q_n)$ in an n-dimensional space is given by:

$$d(P,Q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

4. What are the properties of Euclidean distance?

The properties include non-negativity (distance is always non-negative), identity (distance is zero if and only if the points are identical), symmetry (distance between points is the same in both directions), and the triangle inequality (the direct distance between two points is less than or equal to the sum of distances through a third point).

5. What are the applications of Euclidean distance?

Euclidean distance is commonly used in clustering algorithms like K-means, image recognition, pattern matching, and other applications involving spatial analysis and geometric spaces.

Experiment – 10

Aim: Perform Apriori algorithm to mine frequent item-sets.

Theory: The Apriori algorithm is a classic algorithm used in data mining for mining frequent itemsets and learning association rules. It was proposed by R. Agrawal and R. Srikant in 1994. The algorithm is particularly effective in market basket analysis, where the goal is to find sets of items that frequently co-occur in transactions.

Key Concepts

1. **Itemsets**:
    - An itemset is a collection of one or more items. For example, in a grocery store dataset, an itemset might include items like {milk, bread}.
2. **Frequent Itemsets**:
    - A frequent itemset is an itemset that appears in the dataset with a frequency greater than or equal to a specified threshold, called **support**.
    - The support of an itemset X is defined as the proportion of transactions in the dataset that contain X.

$$\text{Support}(X) = \frac{\text{Number of transactions containing } X}{\text{Total number of transactions}}$$

3. **Association Rules**:
    - An association rule is an implication of the form X→Y, indicating that the presence of itemset X in a transaction implies the presence of itemset Y.
    - Rules are evaluated based on two main metrics: **support** and **confidence**. The confidence of a rule is the proportion of transactions that contain Y among those that contain X:

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}$$

4. **Lift**:
    - Lift measures the effectiveness of a rule over random chance and is defined as:

$$\text{Lift}(X \rightarrow Y) = \frac{\text{Confidence}(X \rightarrow Y)}{\text{Support}(Y)}$$

A lift greater than 1 indicates a positive correlation between X and Y.

Apriori Algorithm Steps

The Apriori algorithm operates in two main phases: **Frequent Itemset Generation** and **Association Rule Generation**.

Phase 1: Frequent Itemset Generation

1. **Initialization**:
    - Scan the database to count the frequency of each individual item (1-itemsets) and generate a list of frequent 1-itemsets based on the minimum support threshold.
2. **Iterative Process**:
    - Generate candidate itemsets of length k (k-itemsets) from the frequent (k-1)-itemsets found in the previous iteration.
    - Prune the candidate itemsets by removing those that contain any infrequent subsets (based on the Apriori property, which states that all subsets of a frequent itemset must also be frequent).
3. **Count Support**:
    - Scan the database again to count the support of the candidate itemsets.
    - Retain those that meet or exceed the minimum support threshold, forming the set of frequent k-itemsets.
4. **Repeat**:
    - Repeat steps 2 and 3 for increasing values of k until no more frequent itemsets can be found.

Phase 2: Association Rule Generation

1. **Rule Generation**:
    - For each frequent itemset, generate all possible non-empty subsets to create rules of the form $X \rightarrow Y$.
    - Calculate the confidence for each rule and retain those that meet or exceed a specified confidence threshold.
2. **Evaluation**:
    - Evaluate the rules using metrics such as support, confidence, and lift to determine their significance and usefulness.

Code:

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Sample dataset: Transactions with items bought
data = {'Milk': [1, 1, 0, 1, 0],
'Bread': [1, 1, 1, 0, 1],
'Butter': [0, 1, 1, 1, 0],
'Beer': [0, 1, 0, 0, 1],
'Cheese': [1, 0, 0, 1, 1]}

# Convert the dataset into a DataFrame
df = pd.DataFrame(data)

# Display the dataset
print("Transaction Dataset:")
```

```
print(df)

#Step 1: Apply the Apriori algorithm to find frequent item-sets
# Minimum support = 0.6 (changeable)
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
# Display frequent item-sets
print("\nFrequent Item-Sets:")
print(frequent_itemsets)

# Step 2: Generate the association rules based on the frequent item-sets
# Minimum confidence = 0.7 (changeable)
rules = association_rules (frequent_itemsets, metric="confidence", min_threshold=0.7)
# Display the association rules
print("\nAssociation Rules:")
print(rules [['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

Output:

```
Transaction Dataset:
   Milk  Bread  Butter  Beer  Cheese
0     1      1       0     0       1
1     1      1       1     1       0
2     0      1       1     0       0
3     1      0       1     0       1
4     0      1       0     1       1

Frequent Item-Sets:
   support  itemsets
0      0.6    (Milk)
1      0.8   (Bread)
2      0.6  (Butter)
3      0.6  (Cheese)

Association Rules:
Empty DataFrame
Columns: [antecedents, consequents, support, confidence, lift]
Index: []
```

Viva Questions

1. What is lift in association rule mining?

Lift is a metric that measures the strength of an association rule compared to random chance. It is calculated as the ratio of the confidence of the rule to the expected confidence if X and Y were independent. A lift greater than 1 indicates a positive correlation between X and Y.

2. What are the two main phases in the Apriori algorithm?

The Apriori algorithm consists of two main phases: Frequent Itemset Generation and Association Rule Generation.

3. What happens in the Frequent Itemset Generation phase of the Apriori algorithm?

In the Frequent Itemset Generation phase, the algorithm scans the database to count the frequency of individual items (1-itemsets) and iteratively generates larger itemsets (k-itemsets) by extending the frequent (k-1)-itemsets. It prunes candidate itemsets by removing those that contain infrequent subsets.

4. How does the Apriori algorithm handle candidate itemsets?

The algorithm generates candidate itemsets of length k from frequent (k-1)-itemsets and prunes those that have any infrequent subsets. The database is then scanned to count the support of the candidate itemsets, retaining only those that meet or exceed the minimum support threshold.

5. What is the Apriori property?

The Apriori property states that all subsets of a frequent itemset must also be frequent. This property is used to prune candidate itemsets in the Apriori algorithm.

Experiment – 11

Aim: Develop different clustering algorithms like K-Means, KMedoids Algorithm, Partitioning Algorithm and Hierarchical.

Theory: Clustering is a fundamental technique in data mining and machine learning used to group similar data points into clusters based on their features. This unsupervised learning method helps in identifying patterns and structures within datasets. Here, we explore four common clustering algorithms: K-Means, K-Medoids, Partitioning Algorithm, and Hierarchical Clustering.

1. K-Means Clustering

Overview: K-Means is one of the most popular clustering algorithms that partitions a dataset into K distinct, non-overlapping subsets (clusters). It aims to minimize the variance within each cluster while maximizing the variance between clusters.

Algorithm Steps:

1. **Initialization**: Randomly select K initial centroids from the dataset.
2. **Assignment**: Assign each data point to the nearest centroid based on the Euclidean distance, forming K clusters.
3. **Update**: Calculate the new centroids as the mean of all points assigned to each cluster.
4. **Convergence**: Repeat the assignment and update steps until the centroids no longer change significantly or a predetermined number of iterations is reached.

Strengths:

- Simple and easy to implement.
- Efficient for large datasets.
- Scales well with data size.

Weaknesses:

- Requires specifying the number of clusters (K) in advance.
- Sensitive to the initial placement of centroids.
- Prone to converging to local minima.

2. K-Medoids Clustering

Overview: K-Medoids is similar to K-Means but instead of using the mean to represent a cluster, it uses actual data points called medoids. This method is more robust to noise and outliers compared to K-Means.

Algorithm Steps:

1. **Initialization**: Randomly select K medoids from the dataset.
2. **Assignment**: Assign each data point to the nearest medoid based on the chosen distance metric (commonly Manhattan distance).

3. **Update**: For each cluster, choose the data point with the smallest total distance to all other points in the cluster as the new medoid.
4. **Convergence**: Repeat the assignment and update steps until no changes occur in the medoids.

Strengths:

- More robust to outliers than K-Means since it uses medoids.
- Does not require calculating means, which may not be meaningful in some contexts.

Weaknesses:

- Computationally more expensive than K-Means, especially for large datasets.
- Still requires the specification of the number of clusters (K).

3. Partitioning Clustering Algorithms

Overview: Partitioning clustering algorithms divide the dataset into K clusters without any hierarchy. These algorithms can include K-Means and K-Medoids but also extend to other approaches. The goal is to create a partition such that the total intra-cluster variance is minimized.

Common Approaches:

- **K-Means**: As described earlier, partitions data into K clusters based on centroid distances.
- **CLARA (Clustering LARge Applications)**: An extension of K-Medoids that uses a sampling method to find medoids and then scales the results to larger datasets.
- **PAM (Partitioning Around Medoids)**: A specific case of K-Medoids that chooses medoids based on minimizing the total distance.

Strengths:

- Efficient for a wide range of datasets.
- Flexible to different distance metrics.

Weaknesses:

- Requires the number of clusters to be defined a priori.
- May not perform well if clusters are of varying shapes or sizes.

4. Hierarchical Clustering

Overview: Hierarchical clustering builds a hierarchy of clusters either through a bottom-up (agglomerative) or top-down (divisive) approach. This method does not require a predetermined number of clusters and allows for the exploration of data at various levels of granularity.

Types:

1. **Agglomerative Hierarchical Clustering**:

- Starts with each data point as an individual cluster and iteratively merges the closest clusters until one cluster remains or a stopping criterion is met.
- Common linkage criteria include single-linkage (minimum distance), complete-linkage (maximum distance), and average-linkage (average distance).

2. **Divisive Hierarchical Clustering**:
   - Starts with one cluster containing all data points and recursively splits it into smaller clusters based on a chosen criterion.

Strengths:

- Does not require the number of clusters to be specified in advance.
- Produces a dendrogram (tree-like diagram) that visually represents the merging of clusters.

Weaknesses:

- Computationally intensive, especially for large datasets ($O(n^2)$ complexity).
- Sensitive to noise and outliers, which can distort the hierarchy.

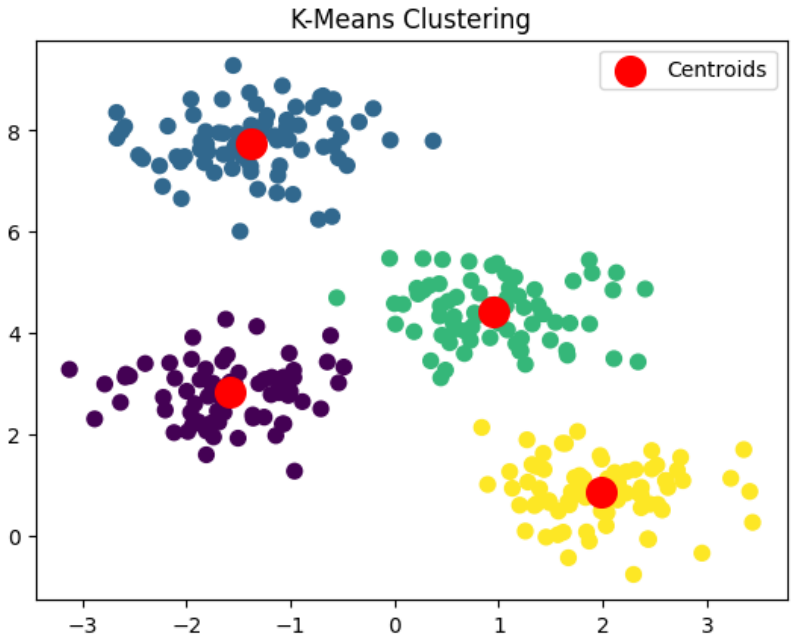
Code:

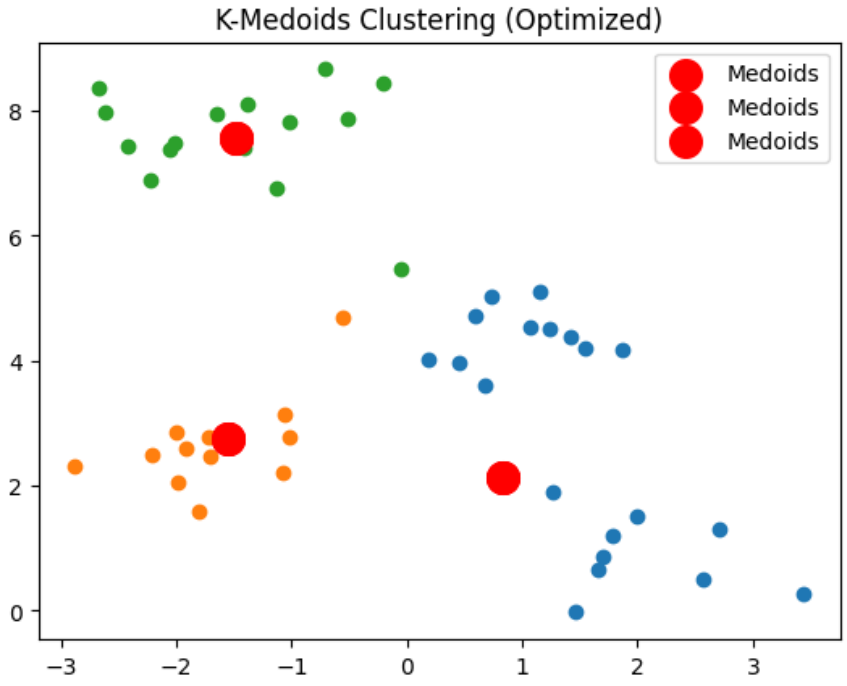
```
!pip install sklearn
!pip install pyclustering
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans, AgglomerativeClustering
from pyclustering.cluster.kmedoids import kmedoids
from pyclustering.cluster.kmeans import kmeans
from pyclustering.cluster.center_initializer import kmeans_plusplus_initializer
from pyclustering.utils import calculate_distance_matrix
from scipy.cluster.hierarchy import dendrogram, linkage

# Generate sample data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Visualize the generated data
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.title("Sample Data for Clustering")
plt.show()


#Apply K-Means algorithm
kmeans_model = KMeans (n_clusters=4)
kmeans_model.fit(X)

#Get cluster labels and centroids
labels_kmeans = kmeans_model.labels_
centroids_kmeans = kmeans_model.cluster_centers_
```

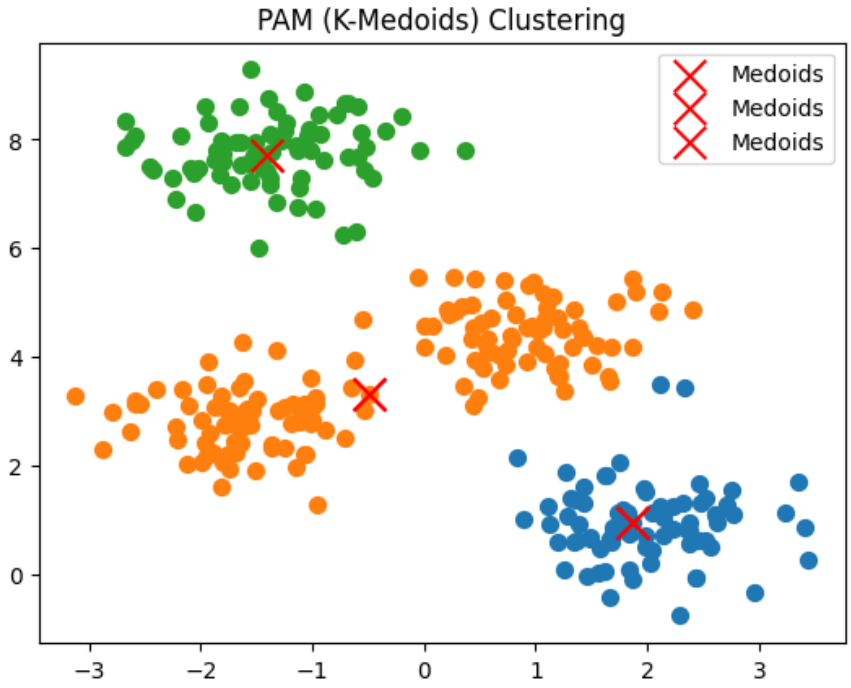
```
#Visualize the K-Means clustering result
plt.scatter(X[:, 0], X[:, 1], c=labels_kmeans, s=50, cmap='viridis')
plt.scatter(centroids_kmeans [:, 0], centroids_kmeans [:, 1], s=200, c='red', label='Centroids')
plt.title("K-Means Clustering")
plt.legend()
plt.show()


#Apply K-Medoid algorithm
from pyclustering.cluster.kmedoids import kmedoids
import numpy as np
import matplotlib.pyplot as plt

#Generate a smaller sample dataset (you can use a subset of your original data)
sampled_data = X[:50] # Use first 50 data points from your dataset

#Initialize the medoid indices (choose random initial medoids from the subset)
initial_medoids = [0, 10, 20] # Choose medoid indices carefully

#Apply K-Medoids to the smaller dataset
kmedoids_instance = kmedoids (sampled_data, initial_medoids, data_type='points')
kmedoids_instance.process()

#Get the resulting clusters and medoids
clusters = kmedoids_instance.get_clusters()
medoids = kmedoids_instance.get_medoids()

#Visualize the clusters and medoids
for cluster in clusters:
  plt.scatter(sampled_data [cluster, 0], sampled_data [cluster, 1])
  plt.scatter(sampled_data [medoids, 0], sampled_data [medoids, 1], c='red', s=200, label='Medoids')
  plt.title("K-Medoids Clustering (Optimized)")
plt.legend()
plt.show()


#Apply the PAM algorithm
pam_instance = kmedoids (X, initial_medoids, data_type='points')
pam_instance.process() # Perform clustering

#Get the resulting clusters and medoids
clusters = pam_instance.get_clusters()
medoids = pam_instance.get_medoids()

#Visualize the PAM clustering result
for cluster in clusters:
  plt.scatter(X [cluster, 0], X [cluster, 1], s=50)
  plt.scatter (X [medoids, 0], X [medoids, 1], s=200, c='red', marker='x', label='Medoids')
plt.title("PAM (K-Medoids) Clustering")
plt.legend()
plt.show()



#Apply Hierarchical Clustering
```

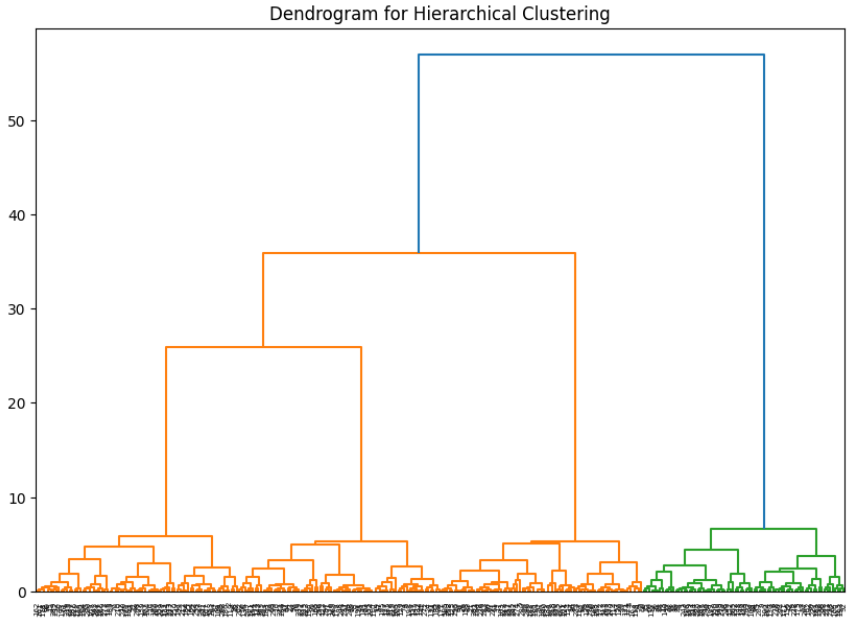
```
linked = linkage(X, method='ward')
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title("Dendrogram for Hierarchical Clustering")
plt.show()

#Perform agglomerative clustering to get cluster labels
hierarchical_model = AgglomerativeClustering(n_clusters=4, metric='euclidean', linkage='ward')
labels_hierarchical = hierarchical_model.fit_predict(X)

#Visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels_hierarchical, s=50, cmap='viridis')
plt.title("Agglomerative Hierarchical Clustering")
plt.show()
```

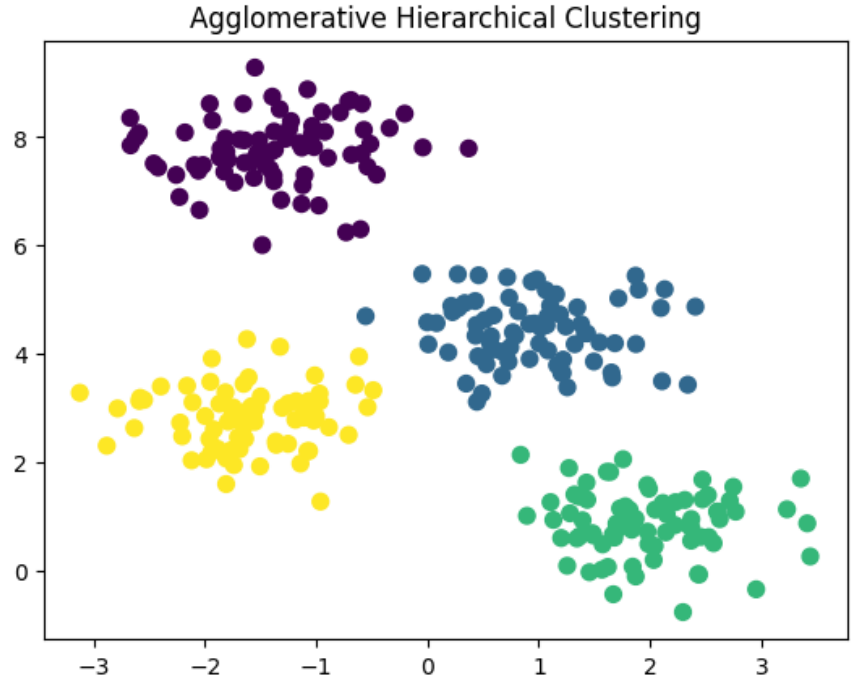
Output:




K-Medoids Clustering (Optimized)
Medoids
Medoids
Medoids
8
6
4
2
0
−3
−2
−1
0
1
2
3


PAM (K-Medoids) Clustering
Medoids
Medoids
Medoids
8
6
4
2
0
−3
−2
−1
0
1
2
3


Dendrogram for Hierarchical Clustering
50
40
30
20
10
0


Agglomerative Hierarchical Clustering
8
6
4
2
0
−3
−2
−1
0
1
2
3

Viva Questions

1. What are the key types of clustering algorithms?

The four key types of clustering algorithms are K-Means, K-Medoids, Partitioning algorithms, and Hierarchical Clustering.

2. What is the K-Means clustering algorithm?

K-Means is a popular clustering algorithm that partitions data into K distinct, non-overlapping clusters by minimizing the variance within each cluster. The algorithm starts by selecting K initial centroids, assigns data points to the nearest centroid, updates centroids, and repeats until convergence.

3. What are the steps in the K-Means algorithm?

The K-Means algorithm consists of four main steps:

- Initialization: Randomly select K centroids.
- Assignment: Assign each data point to the nearest centroid.
- Update: Recalculate centroids as the mean of assigned data points.
- Convergence: Repeat the assignment and update steps until centroids stabilize.

4. What are the strengths and weaknesses of K-Means clustering?

Strengths:

- Simple and easy to implement.
- Efficient for large datasets.
- Scales well with data size.

Weaknesses:

- Requires specifying K (number of clusters) in advance.
- Sensitive to the initial placement of centroids.
- Prone to converging to local minima.

5. What is the difference between K-Means and K-Medoids clustering?

In K-Means, the centroid of each cluster is the mean of the data points, while in K-Medoids, the centroid is replaced by an actual data point called the medoid. K-Medoids is more robust to noise and outliers compared to K-Means.

Experiment – 12

Aim: Apply Validity Measures to evaluate the quality of Data.

Theory: Evaluating data quality is critical in data analysis and machine learning as it directly affects the performance of models and the validity of insights derived from the data. Various validity measures help assess different aspects of data quality. Here, we outline the key validity measures demonstrated in your code, along with their importance.

1. Missing Values

Definition: Missing values occur when data points for certain features are not recorded. They can introduce bias and reduce the quality of the dataset.

Importance:

- A high proportion of missing values can lead to inaccurate models and biased results.
- Different strategies can be applied to handle missing values, including imputation, deletion, or using algorithms that can work with missing data.

Measure:

- The code calculates the number of missing values per column, which provides insight into the extent of the issue.

2. Duplicate Entries

Definition: Duplicate entries refer to identical records in a dataset. They can occur due to errors during data collection or processing.

Importance:

- Duplicates can skew the results of analyses and lead to overfitting in machine learning models.
- Identifying and removing duplicates is crucial for maintaining data integrity.

Measure:

- The code checks for the number of duplicate entries, helping to quantify this issue in the dataset.

3. Outlier Detection

Definition: Outliers are data points that differ significantly from other observations in the dataset. They can arise due to variability in the measurement or may indicate a measurement error.

Importance:

- Outliers can disproportionately affect statistical analyses and model training, leading to inaccurate predictions.
- Identifying outliers allows for the option to investigate them further and decide whether to keep, remove, or adjust them.

Measure:

- The code uses the Isolation Forest algorithm to detect outliers, reporting the number detected. This helps understand the presence of anomalies in the dataset.

4. Multicollinearity

Definition: Multicollinearity occurs when two or more independent variables in a regression model are highly correlated, meaning they contain similar information.

Importance:

- High multicollinearity can inflate the variance of coefficient estimates, making the model unstable and reducing interpretability.
- It complicates the process of determining the importance of predictors.

Measure:

- The correlation matrix generated in the code reveals the relationships between features, allowing for the identification of highly correlated features.

5. Feature Distribution (Skewness)

Definition: Skewness measures the asymmetry of the probability distribution of a real-valued random variable. A skewed distribution can indicate that certain transformations may be needed before modeling.

Importance:

- Features that are heavily skewed can violate the assumptions of certain statistical tests and machine learning algorithms, impacting model performance.
- Understanding skewness helps in selecting appropriate preprocessing methods (e.g., normalization, logarithmic transformation).

Measure:

- The code calculates and prints the skewness of each feature, as well as visualizes the feature distributions, which aids in identifying heavily skewed variables.

6. Class Imbalance

Definition: Class imbalance occurs when the number of instances in each class of a classification problem is not approximately equal. This is common in binary classification tasks.

Importance:

- Imbalanced classes can lead to biased models that perform well on the majority class but poorly on the minority class.
- It's crucial to evaluate the class distribution and apply techniques to handle imbalances, such as resampling methods (oversampling, undersampling) or algorithm adjustments.

Measure:

- The code checks the distribution of the target class, helping to identify any potential imbalance that could affect model training.

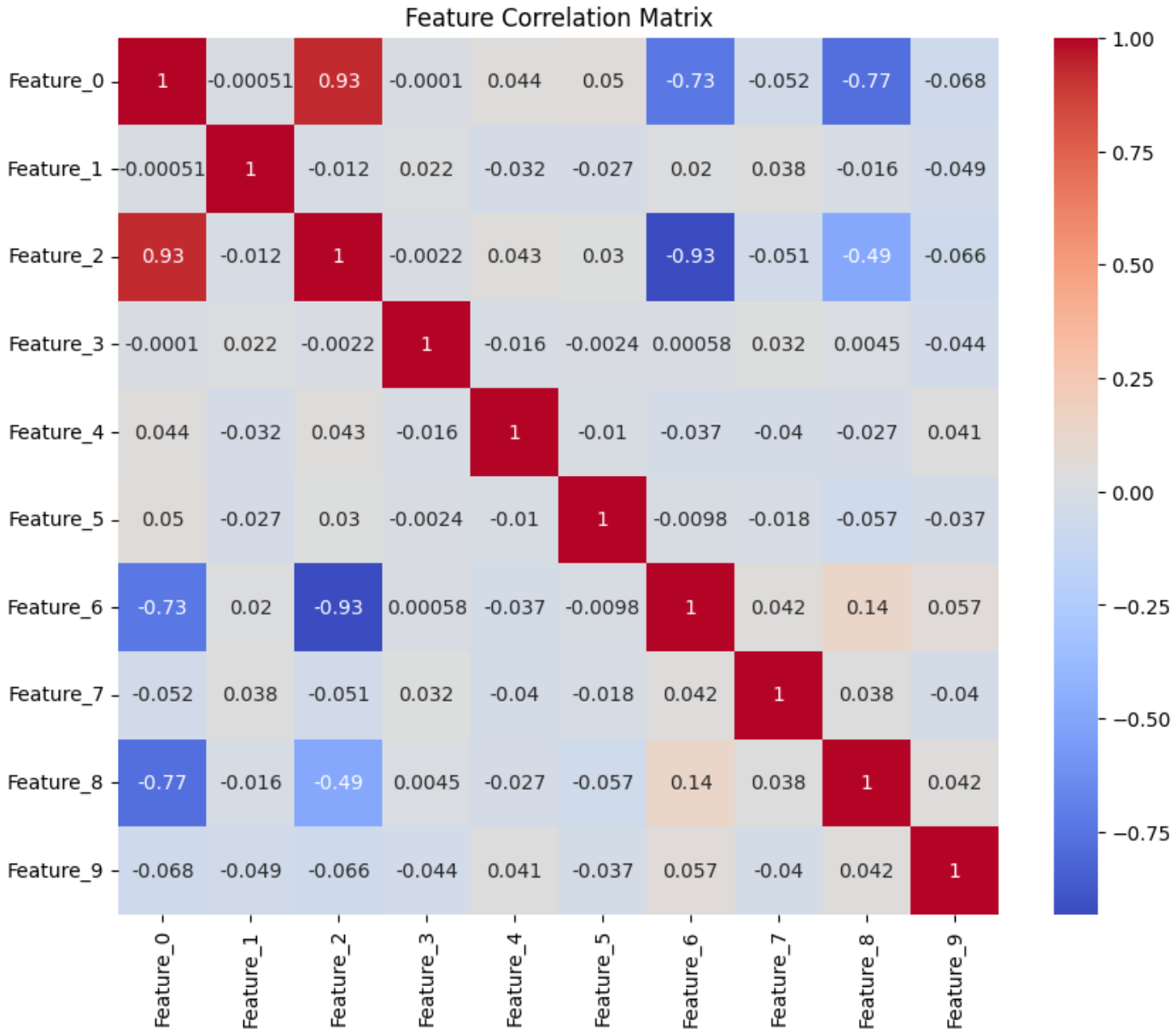
Code:

```
import pandas as pd
import numpy as np
```

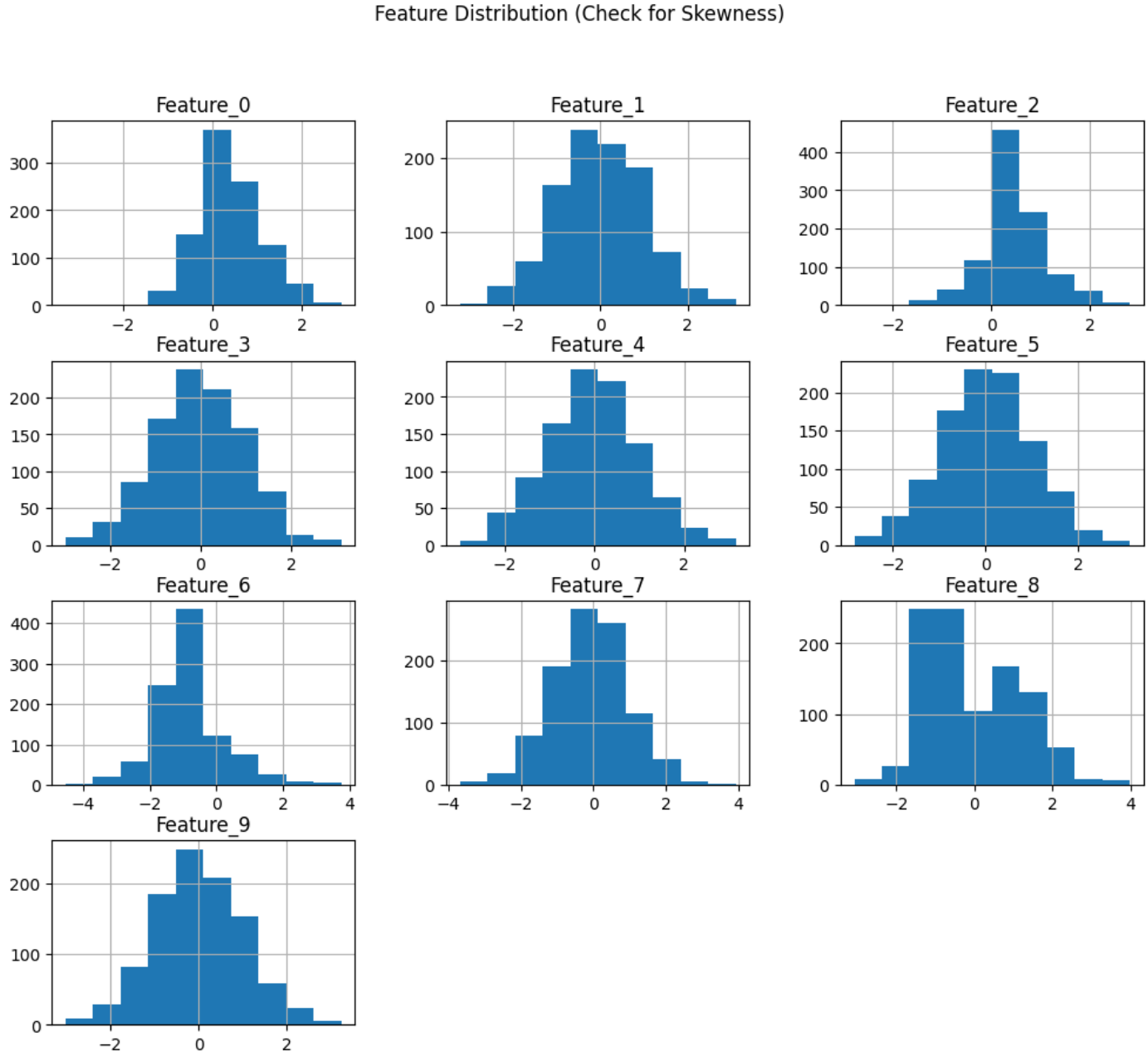
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler

#Step 1: Generate sample data
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2, weights=[0.9, 0.1], random_state=42)
df = pd.DataFrame (X, columns=[f"Feature_{i}" for i in range(10)])
df ['Target'] = y

#Step 2: Introduce missing values for illustration
df.iloc[10:20, 0] = np.nan # Create missing values

#Step 3: Data Quality Validity Measures
#1. Check for Missing Values
missing_values = df.isnull().sum()
print("\nMissing Values per Column:")
print(missing_values)

#2. Check for Duplicate Entries
duplicates = df.duplicated().sum()
print(f"\nNumber of Duplicate Entries: {duplicates}")

#3. Detect Outliers using Isolation Forest (anomaly detection) ) # Assuming 5% of the data is outliers
iso_forest = IsolationForest(contamination=0.05)
outliers = iso_forest.fit_predict(df.drop('Target', axis=1).fillna(df.mean())) # Fill missing values
outlier_count = sum(outliers == -1)
print(f"\nNumber of Outliers Detected: {outlier_count}")

#4. Check for Multicollinearity (Correlation Matrix)
correlation_matrix = df.drop('Target', axis=1).corr()
print("\nCorrelation Matrix (Multicollinearity Check):")
print(correlation_matrix)
# Visualize Correlation Matrix
plt.figure(figsize=(10,8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Feature Correlation Matrix")
plt.show()

#5. Check for Feature Distribution (Skewness)
skewness = df.drop('Target', axis=1).skew()
print("\nSkewness of Features:")
print(skewness)
# Visualize Feature Distributions
df.drop('Target', axis=1).hist(figsize=(12, 10))
plt.suptitle("Feature Distribution (Check for Skewness)")
plt.show()

#6. Check for Class Imbalance
class_distribution = df ['Target'].value_counts (normalize=True)
print("\nClass Distribution (Imbalance Check):")
print(class_distribution)
```

Output:

```
Missing Values per Column:
Feature_0    10
Feature_1     0
Feature_2     0
Feature_3     0
Feature_4     0
Feature_5     0
Feature_6     0
Feature_7     0
Feature_8     0
Feature_9     0
Target        0
dtype: int64

Number of Duplicate Entries: 0

Number of Outliers Detected: 50

Correlation Matrix (Multicollinearity Check):
           Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
Feature_0   1.000000  -0.000511   0.929954  -0.000100   0.043571   0.049565
Feature_1  -0.000511   1.000000  -0.011844   0.022266  -0.031579  -0.026735
Feature_2   0.929954  -0.011844   1.000000  -0.002181   0.042782   0.029734
Feature_3  -0.000100   0.022266  -0.002181   1.000000  -0.016467  -0.002439
Feature_4   0.043571  -0.031579   0.042782  -0.016467   1.000000  -0.010388
Feature_5   0.049565  -0.026735   0.029734  -0.002439  -0.010388   1.000000
Feature_6  -0.730999   0.019942  -0.931190   0.000579  -0.037322  -0.009805
Feature_7  -0.052135   0.037916  -0.051106   0.032063  -0.039584  -0.017981
Feature_8  -0.774910  -0.015513  -0.489943   0.004541  -0.026977  -0.057333
Feature_9  -0.068196  -0.048878  -0.066174  -0.043532   0.041455  -0.036535

           Feature_6  Feature_7  Feature_8  Feature_9
Feature_0  -0.730999  -0.052135  -0.774910  -0.068196
Feature_1   0.019942   0.037916  -0.015513  -0.048878
Feature_2  -0.931190  -0.051106  -0.489943  -0.066174
Feature_3   0.000579   0.032063   0.004541  -0.043532
Feature_4  -0.037322  -0.039584  -0.026977   0.041455
Feature_5  -0.009805  -0.017981  -0.057333  -0.036535
Feature_6   1.000000   0.042084   0.138445   0.057483
Feature_7   0.042084   1.000000   0.038205  -0.039575
Feature_8   0.138445   0.038205   1.000000   0.042318
Feature_9   0.057483  -0.039575   0.042318   1.000000
```




```
Skewness of Features:
Feature_0    0.044845
Feature_1    0.011528
Feature_2   -0.068293
Feature_3   -0.025681
Feature_4    0.076800
Feature_5   -0.012671
Feature_6    0.605986
Feature_7    0.005772
Feature_8    0.481451
Feature_9    0.037996
dtype: float64
```




```
Class Distribution (Imbalance Check):
Target
0    0.897
1    0.103
Name: proportion, dtype: float64
```

Viva Questions

1. What is data quality evaluation, and why is it important?

Data quality evaluation refers to assessing the various aspects of a dataset to ensure its completeness, accuracy, consistency, and relevance for analysis and modeling. It is important because poor data quality can lead to inaccurate models, biased results, and unreliable insights.

2. What are missing values, and why do they matter in data analysis?

Missing values occur when data points for certain features are not recorded or available. They matter because a high proportion of missing values can skew results, introduce bias, and reduce the accuracy of machine learning models. Strategies such as imputation or deletion are used to handle missing data.

3. How are missing values measured and handled?

Missing values are typically measured by calculating the number of missing values per column, which helps assess the extent of the issue. Handling missing values can involve techniques such as imputation (replacing missing values with mean/median values), deletion of rows or columns, or using algorithms that can handle missing data directly.

4. What are duplicate entries, and why is it important to identify them?

Duplicate entries refer to identical records in the dataset, which can occur due to errors during data collection or processing. They are important to identify and remove because they can distort analysis results, skew statistics, and lead to overfitting in machine learning models.

5. How does the code handle duplicate entries?

The code checks for the number of duplicate entries by identifying identical rows in the dataset. This helps to quantify the issue and determine whether any duplicates need to be removed to maintain data integrity.