EXERCISE 1 : DATA PRE-PROCESSING AND DATA CUBE : DATA PREPROCESSING METHODS ON STUDENT AND LABOR DATASETS IMPLEMENT DATA CUBE FOR DATA WAREHOUSE ON 3-DIMENSIONAL DATA

To achieve data preprocessing on student and labor datasets and to implement a data cube for data warehousing on 3-dimensional data in Orange, follow these steps:

Data Preprocessing

Data preprocessing involves cleaning, transforming, and organizing data to prepare it for analysis. We'll use Orange's widgets to perform data preprocessing on student and labor datasets.

Steps for Data Preprocessing

1. **Load Data**:
   - Use the 'File' widget to load your student and labor datasets.
2. **Handle Missing Values**:
   - Use the 'Edit Domain' widget to impute missing values or remove instances with missing values.
3. **Normalize Data**:
   - Use the 'Normalize' widget to scale the data to a standard range.
4. **Discretize Data**:
   - Use the 'Discretize' widget to convert continuous variables into categorical bins if necessary.
5. **Feature Selection**:
   - Use the 'Select Columns' widget to choose the relevant features for analysis.

Example Workflow for Student Dataset

1. **Load Data**:
   - Drag the 'File' widget to the canvas.
   - Load your student dataset file (e.g., `students.csv`).
2. **Handle Missing Values**:
   - Drag the 'Edit Domain' widget to the canvas.
   - Connect the 'File' widget to the 'Edit Domain' widget.
   - Configure the 'Edit Domain' widget to handle missing values (e.g., impute with mean/median).
3. **Normalize Data**:
   - Drag the 'Normalize' widget to the canvas.
   - Connect the 'Edit Domain' widget to the 'Normalize' widget.
   - Configure the 'Normalize' widget to standardize the data.
4. **Discretize Data** (if necessary):
   - Drag the 'Discretize' widget to the canvas.
   - Connect the 'Normalize' widget to the 'Discretize' widget.
   - Configure the 'Discretize' widget to bin continuous variables.
5. **Feature Selection**:
   - Drag the 'Select Columns' widget to the canvas.
   - Connect the final preprocessing widget to the 'Select Columns' widget.
   - Select the relevant features for analysis.

Implementing a Data Cube for Data Warehousing

A data cube allows you to visualize and analyze data across multiple dimensions. In Orange, you can simulate a data cube using the 'Pivot Table' widget for 3-dimensional data.

Steps to Implement a Data Cube

1. **Load Data**:
   - Use the 'File' widget to load your dataset with at least three dimensions.
2. **Pivot Table**:
   - Use the 'Pivot Table' widget to create and visualize the data cube.

Example Workflow for a 3-Dimensional Dataset

1. **Load Data**:
   - Drag the 'File' widget to the canvas.
   - Load your 3-dimensional dataset file (e.g., `3d_dataset.csv`).
2. **Create Data Cube**:
   - Drag the 'Pivot Table' widget to the canvas.
   - Connect the 'File' widget to the 'Pivot Table' widget.
   - Configure the 'Pivot Table' widget to set rows, columns, and layer attributes to the desired dimensions.

Detailed Steps for Implementing the Data Cube

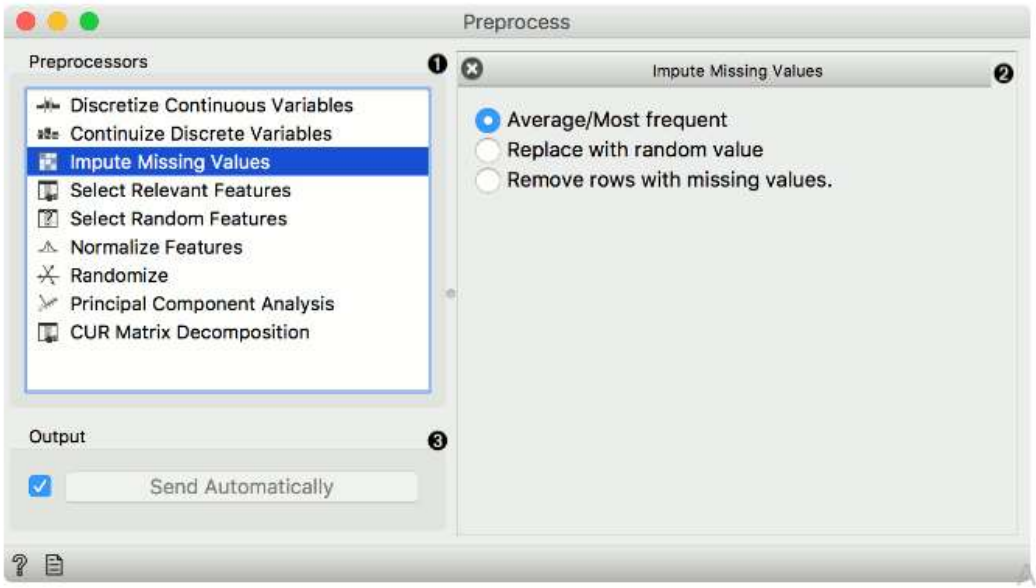
1. **Load and Preprocess Data**:
   - Use the steps described in the data preprocessing section to load and preprocess your dataset.
2. **Create Pivot Table for Data Cube**:
   - Drag the 'Pivot Table' widget to the canvas.
   - Connect the preprocessed data widget to the 'Pivot Table' widget.
   - Configure the rows, columns, and layers to represent the three dimensions of your data.
3. **Analyze Data Cube**:
   - Use the pivot table to analyze data across the selected dimensions, such as aggregating by sum, average, count, etc.

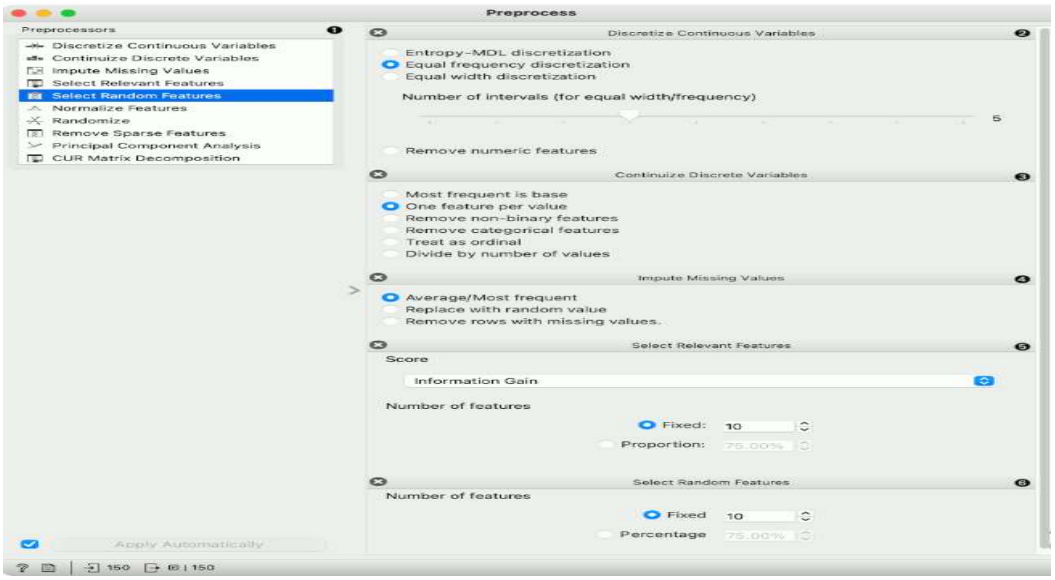
Example Implementation Preprocessing Student Dataset

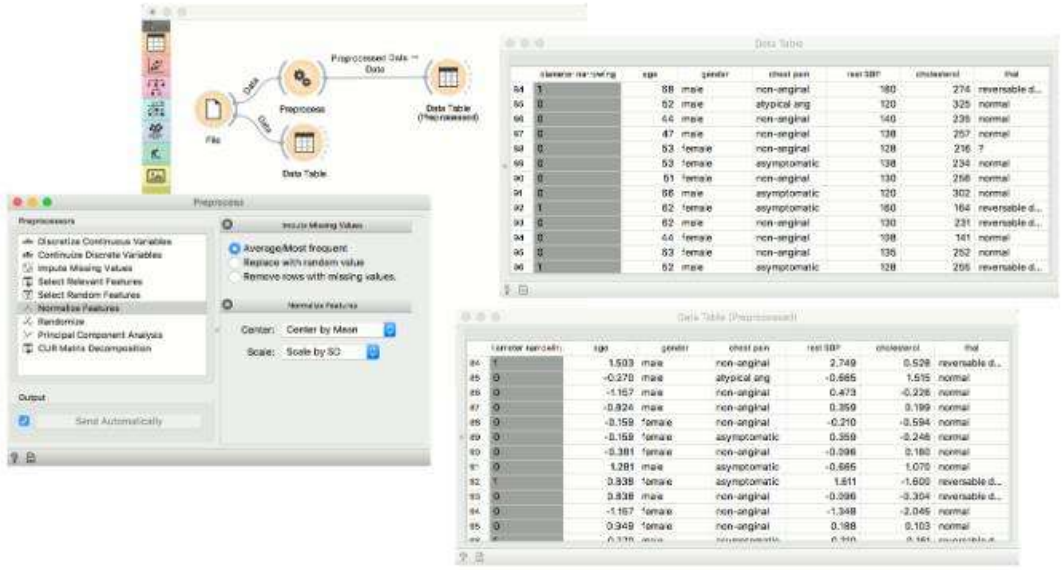
1. **Load Data**:
   - Drag the 'File' widget to the canvas and load `students.csv`.
2. **Handle Missing Values**:
   - Connect 'File' to 'Edit Domain'.
   - Impute missing values with the median.
3. **Normalize Data**:
   - Connect 'Edit Domain' to 'Normalize'.
   - Standardize the data.
4. **Feature Selection**:
   - Connect 'Normalize' to 'Select Columns'.
   - Select relevant features like `Age`, `Grade`, `Hours of Study`.

Creating a Data Cube

1. **Load Data**:
   - Drag the 'File' widget to the canvas and load `3d_dataset.csv`.
2. **Create Pivot Table**:
   - Connect 'File' to 'Pivot Table'.
   - Configure rows to `Product`, columns to `Region`, and layers to `Time`.
3. **Analyze**:
   - Use the pivot table to aggregate data (e.g., sum of sales).







Conclusion

By following these steps, you can perform data preprocessing on student and labor datasets and implement a data cube for data warehousing in Orange. Preprocessing ensures your data is clean and ready for analysis, while the pivot table simulates a data cube, allowing multi-dimensional analysis.

EXERCISE 2 DATA CLEANING: IMPLEMENT VARIOUS MISSING HANDLING MECHANISMS, IMPLEMENT VARIOUS NOISY HANDLING MECHANISMS

data cleaning, particularly handling missing and noisy data, is essential for ensuring accurate analysis and results. Here's how you can implement various mechanisms for handling missing and noisy data using Orange.

Handling Missing Data

1. **Load Data**:
   - Use the 'File' widget to load your dataset.
2. **Data Imputation**:
   - Use the 'Impute' widget to handle missing values.
   - Connect the 'File' widget to the 'Impute' widget.

   Options in the Impute Widget:
   - **Mean/Median Imputation**: Replace missing values with the mean or median of the column.
   - **Most Frequent**: Replace missing values with the most frequent value (mode) in the column.
   - **Model-Based Imputation**: Use a machine learning model to predict and replace missing values based on other features.
   - **User-Defined**: Specify a custom value to replace missing values.
3. **Removing Rows/Columns with Missing Data**:
   - Use the 'Select Columns' widget to remove columns with missing values if they are not essential.
   - Use the 'Edit Domain' widget to remove rows with missing values.

Handling Noisy Data

1. **Load Data**:
   - Use the 'File' widget to load your dataset.
2. **Data Smoothing**:
   - Use the 'Smoothing' widget to handle noisy data.
   - Connect the 'File' widget to the 'Smoothing' widget.

   Options in the Smoothing Widget:
   - **Binning**: Group continuous data into bins to reduce noise.
   - **Moving Average**: Use moving averages to smooth time-series data.
   - **Gaussian Smoothing**: Apply Gaussian smoothing to reduce noise.
3. **Outlier Detection and Removal**:
   - Use the 'Outliers' widget to detect and handle outliers.
   - Connect the 'File' widget to the 'Outliers' widget.

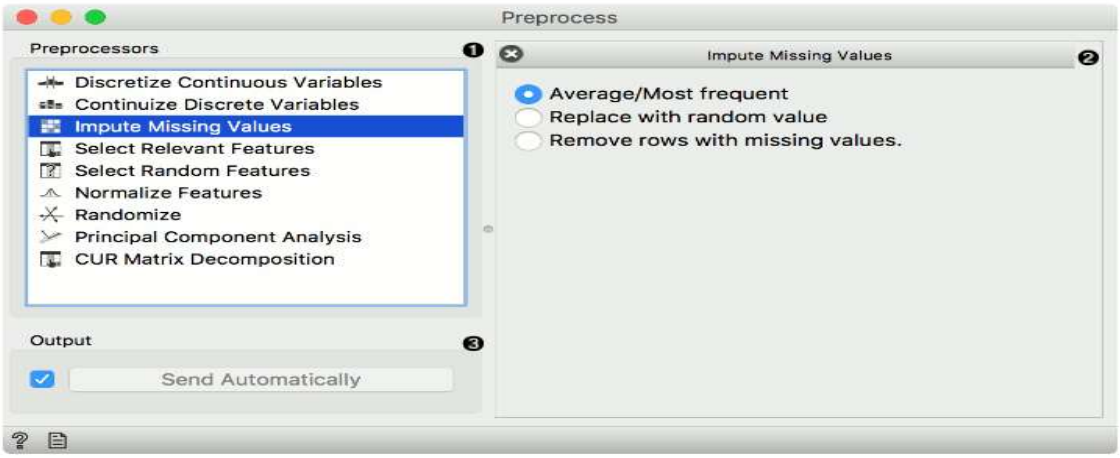
   Options in the Outliers Widget:
   - **Z-Score**: Detect outliers based on the Z-score. Values above or below a certain threshold (e.g., 3 standard deviations from the mean) are considered outliers.
   - **IQR Method**: Detect outliers using the Interquartile Range (IQR). Values outside 1.5 times the IQR above the third quartile or below the first quartile are considered outliers.
   - **Isolation Forest**: Use an Isolation Forest model to detect outliers.

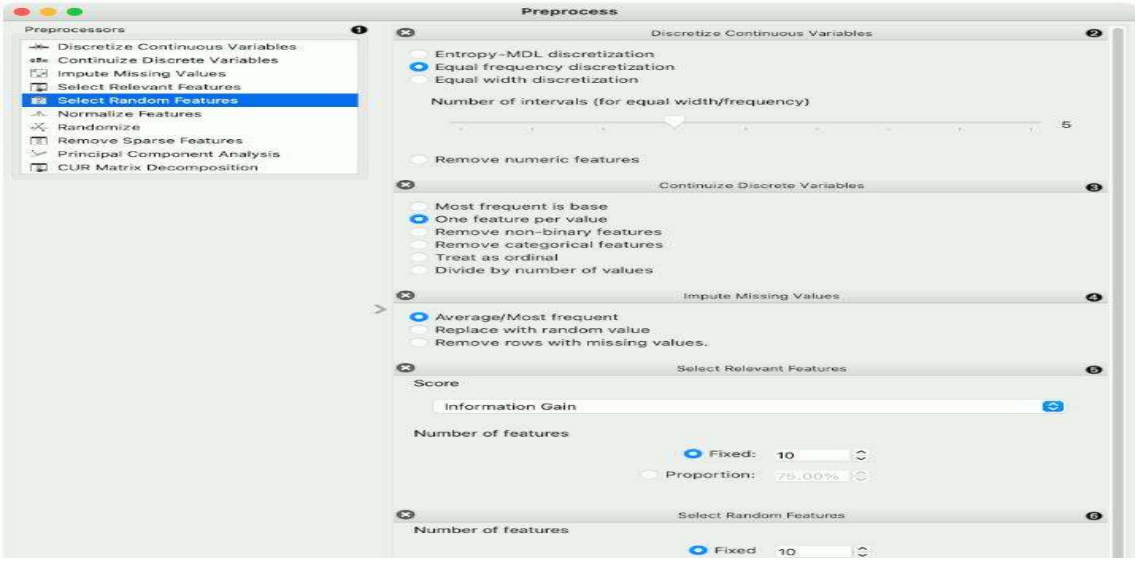
Example Workflow in Orange

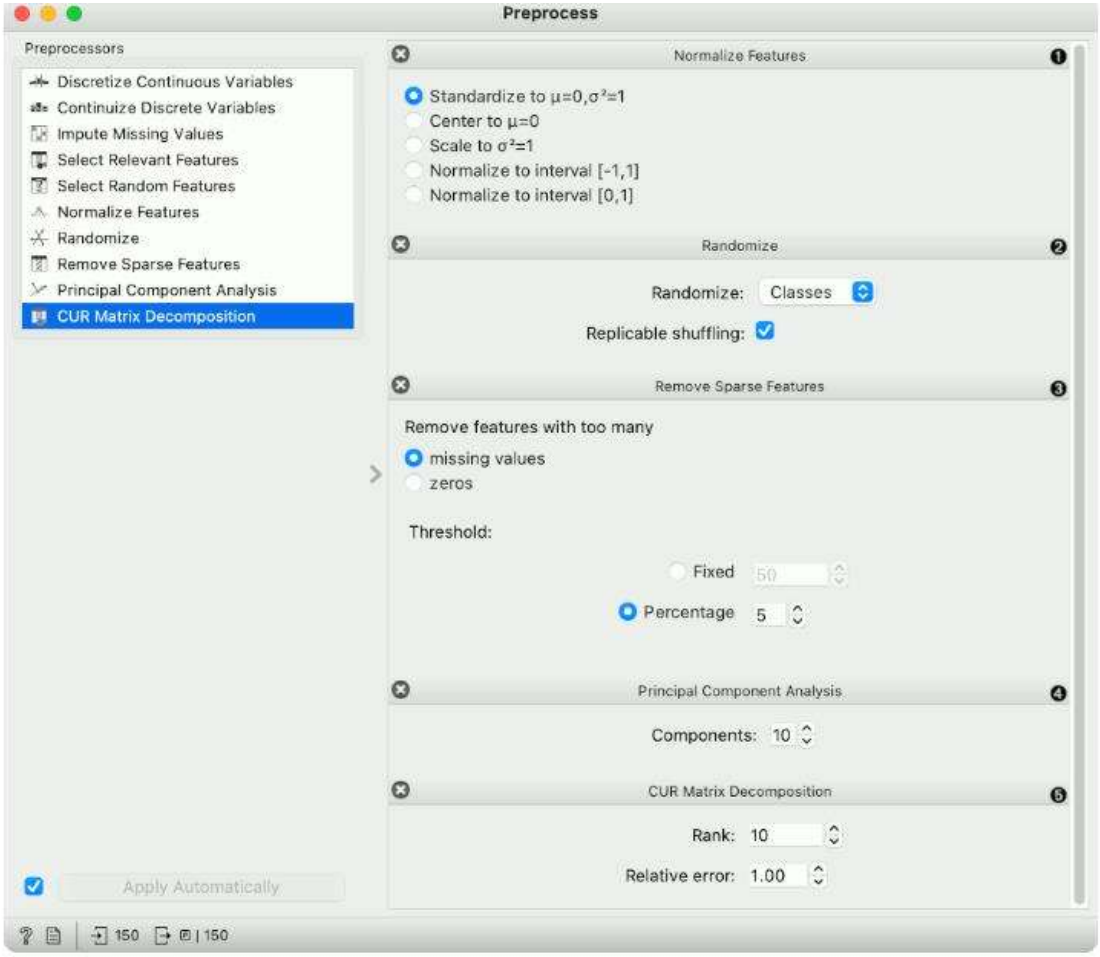
1. **Load and Inspect Data**:
   - Use the 'File' widget to load `dataset.csv`.
   - Connect it to the 'Data Table' widget to inspect the data.
2. **Handle Missing Data**:
   - Connect the 'File' widget to the 'Impute' widget.
   - In the 'Impute' widget, choose 'Mean/Median' for numeric features and 'Most Frequent' for categorical features.

- Connect the 'Impute' widget to a 'Data Table' widget to inspect the imputed data.

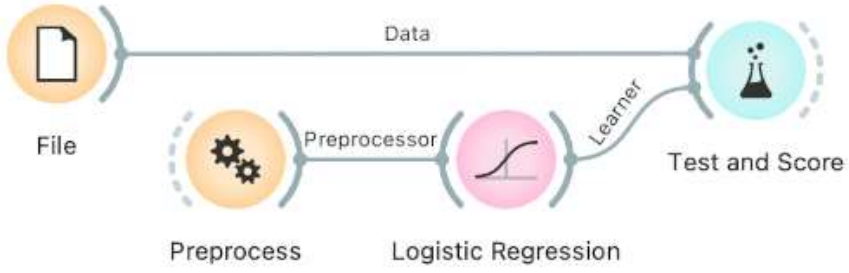
3. **Remove Columns with Excessive Missing Data**:
   - Connect the 'Impute' widget to the 'Select Columns' widget.
   - In the 'Select Columns' widget, manually remove columns with a high percentage of missing values.
4. **Handle Noisy Data**:
   - Connect the 'Select Columns' widget to the 'Smoothing' widget.
   - In the 'Smoothing' widget, apply 'Binning' for continuous variables.
   - Connect the 'Smoothing' widget to a 'Data Table' widget to inspect the smoothed data.
5. **Detect and Remove Outliers**:
   - Connect the 'Smoothing' widget to the 'Outliers' widget.
   - In the 'Outliers' widget, choose the 'Z-Score' method and set a threshold (e.g., 3).
   - Connect the 'Outliers' widget to a 'Data Table' widget to inspect the data with outliers removed.



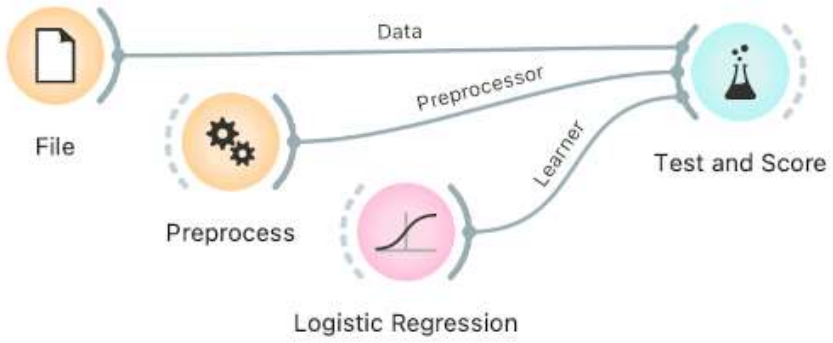




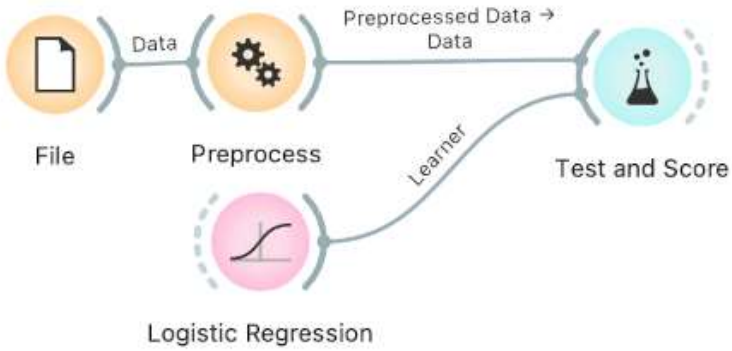
1. Connect **Preprocess** to the learner. This will override the default preprocessing pipeline for the learner and apply only custom preprocessing pipeline (default preprocessing steps are described in each learner's documentation).



2.Connect **Preprocess** to Test and Score. This will apply the preprocessors to each batch within cross-validation. Then the learner's preprocessors will be applied to the preprocessed subset.

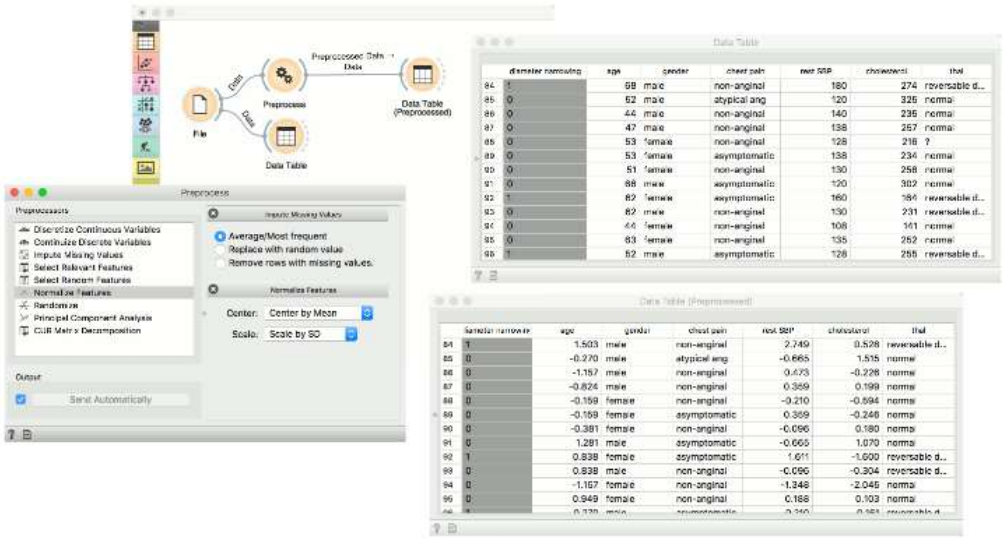


Finally, there's a wrong way to do it. Connecting **Preprocess** directly to the original data and outputting preprocessed data set will likely overfit the model. Don't do it.



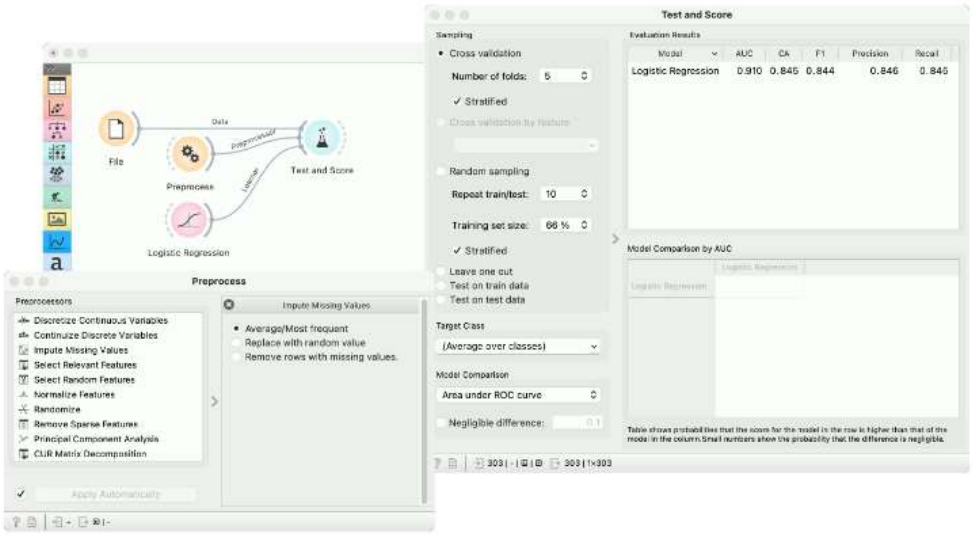
Examples

In the first example, we have used the heart_disease.tab dataset available in the dropdown menu of the File widget. then we used **Preprocess** to impute missing values and normalize features. We can observe the changes in the Data Table and compare it to the non-processed data.



Preprocess for predictive modeling.
This time we are using the heart_disease.tab data from the File widget. You can access the data in the dropdown menu. This is a dataset with 303 patients that came to the doctor suffering from a chest pain. After the tests were done, some patients were found to have diameter narrowing and others did not (this is our class variable).

Some values are missing in our data set, so we would like to impute missing values before evaluating the model. We do this by passing a preprocessor directly to Test and Score. In **Preprocess**, we set the correct preprocessing pipeline (in our example only a single preprocessor with Impute missing values), then connect it to the Preprocessor input of Test and Score.
We also pass the data and the learner (in this case, a Logistic Regression). This is the correct way to pass a preprocessor to cross-validation as each fold will independently get preprocessed in the training phase. This is particularly important for feature



Conclusion

By following these steps, you can effectively clean your data in Orange, handling both missing and noisy data. These preprocessing steps are crucial for ensuring that your data is of high quality, leading to more accurate and reliable analysis and modeling results.

EXCERCISE3 EXPLORATORY ANALYSIS : DEVELOP K-MEANS AND MST BASED CLUSTERING TECHNIQUES, DEVELOP THE METHODOLOGY FOR ASSESSMENT OF CLUSTERS FOR GIVEN DATASET

To perform exploratory analysis using k-means and MST (Minimum Spanning Tree) based clustering techniques in Orange, and to develop a methodology for assessing clusters, follow these steps:

K-Means Clustering in Orange

1. **Load Data**:
   - Use the 'File' widget to load your dataset.
2. **Data Preprocessing**:
   - Use the 'Preprocess' widget to handle any missing values or normalize/standardize the data.
3. **K-Means Clustering**:
   - Drag the 'K-Means' widget into the workspace.
   - Connect the 'File' widget to the 'K-Means' widget.
   - Configure the 'K-Means' widget by setting the number of clusters (k).
4. **Inspect Clusters**:
   - Connect the 'K-Means' widget to the 'Scatter Plot' widget to visualize the clusters.
   - Optionally, connect the 'K-Means' widget to the 'Data Table' widget to inspect the clustered data.

MST (Minimum Spanning Tree) Based Clustering in Orange

Orange does not directly provide an MST-based clustering widget, but you can implement a similar approach using hierarchical clustering or custom scripts.

1. **Load Data**:
   - Use the 'File' widget to load your dataset.
2. **Data Preprocessing**:
   - Use the 'Preprocess' widget to handle any missing values or normalize/standardize the data.
3. **Hierarchical Clustering** (as a proxy for MST-based clustering):
   - Drag the 'Hierarchical Clustering' widget into the workspace.
   - Connect the 'File' widget to the 'Hierarchical Clustering' widget.
   - Configure the 'Hierarchical Clustering' widget by selecting an appropriate linkage method (e.g., single linkage for an MST-like effect).
4. **Inspect Dendrogram**:
   - Connect the 'Hierarchical Clustering' widget to the 'Dendrogram' widget to visualize the hierarchical structure.
   - Use the 'Silhouette Plot' widget to evaluate the clustering quality.

Methodology for Assessment of Clusters

1. **Load and Cluster Data**:
   - Perform clustering using the 'K-Means' and 'Hierarchical Clustering' widgets as described above.
2. **Cluster Evaluation Metrics**:
   - **Silhouette Score**:
     - Drag the 'Silhouette Plot' widget into the workspace.
     - Connect it to the 'K-Means' or 'Hierarchical Clustering' widget to evaluate the silhouette score of each cluster.
   - **Elbow Method**:
     - Use the 'Elbow Plot' to determine the optimal number of clusters for k-means. This can be done by manually setting different values of k and plotting the within-cluster sum of squares.

- **Dunn Index** and **Davies-Bouldin Index**:
  - Use custom scripts or external tools to compute these indices if not directly available in Orange.

3. **Visualization**:
   - Use the 'Scatter Plot', 'Silhouette Plot', and 'Dendrogram' widgets to visualize the clusters and their quality.
   - For multi-dimensional data, use the 't-SNE' or 'PCA' widgets to reduce dimensions and visualize clusters.

Example Workflow in Orange

1. **Load and Preprocess Data**:
   - Use the 'File' widget to load `dataset.csv`.
   - Connect it to the 'Preprocess' widget to handle missing values and normalize data.
   - Connect the 'Preprocess' widget to a 'Data Table' widget to inspect the preprocessed data.
2. **Perform K-Means Clustering**:
   - Connect the 'Preprocess' widget to the 'K-Means' widget.
   - Set the number of clusters (e.g., k=3).
   - Connect the 'K-Means' widget to the 'Scatter Plot' widget to visualize clusters.
3. **Evaluate Clusters**:
   - Connect the 'K-Means' widget to the 'Silhouette Plot' widget to evaluate the silhouette scores.
   - Manually run k-means for different k values and plot the within-cluster sum of squares to create an Elbow Plot.
4. **Perform Hier

Example Workflow in Orange (continued)

4. **Perform Hierarchical Clustering**:
   - Connect the 'Preprocess' widget to the 'Hierarchical Clustering' widget.
   - Configure the widget with the desired linkage method (e.g., single linkage for MST-like clustering).
   - Connect the 'Hierarchical Clustering' widget to the 'Dendrogram' widget to visualize the dendrogram.
   - Optionally, connect to the 'Silhouette Plot' widget to evaluate cluster quality.

Detailed Steps

K-Means Clustering

1. **Load Data**:
   - Drag the 'File' widget to the canvas.
   - Select your dataset file (e.g., `dataset.csv`).
2. **Preprocess Data**:
   - Drag the 'Preprocess' widget to the canvas.
   - Connect the 'File' widget to the 'Preprocess' widget.
   - Configure preprocessing steps such as imputing missing values, normalizing, or standardizing data.
3. **K-Means Clustering**:
   - Drag the 'K-Means' widget to the canvas.
   - Connect the 'Preprocess' widget to the 'K-Means' widget.
   - Set the number of clusters (k) in the 'K-Means' widget.
   - Optionally, connect to the 'Data Table' widget to inspect clustered data.
4. **Inspect Clusters**:
   - Drag the 'Scatter Plot' widget to the canvas.
   - Connect the 'K-Means' widget to the 'Scatter Plot' widget to visualize clusters.
5. **Cluster Evaluation**:
   - Drag the 'Silhouette Plot' widget to the canvas.

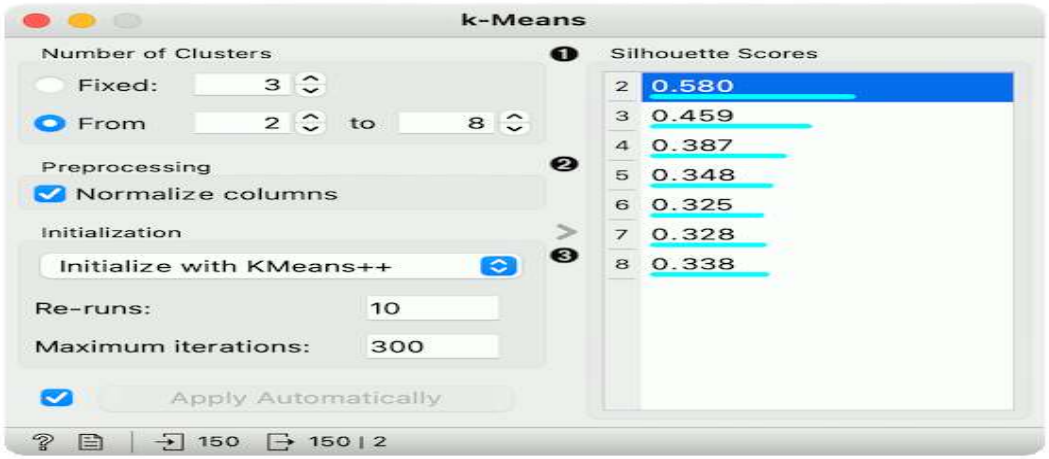
- Connect the 'K-Means' widget to the 'Silhouette Plot' widget to evaluate silhouette scores.
- For the Elbow Method, manually adjust k and observe the within-cluster sum of squares.

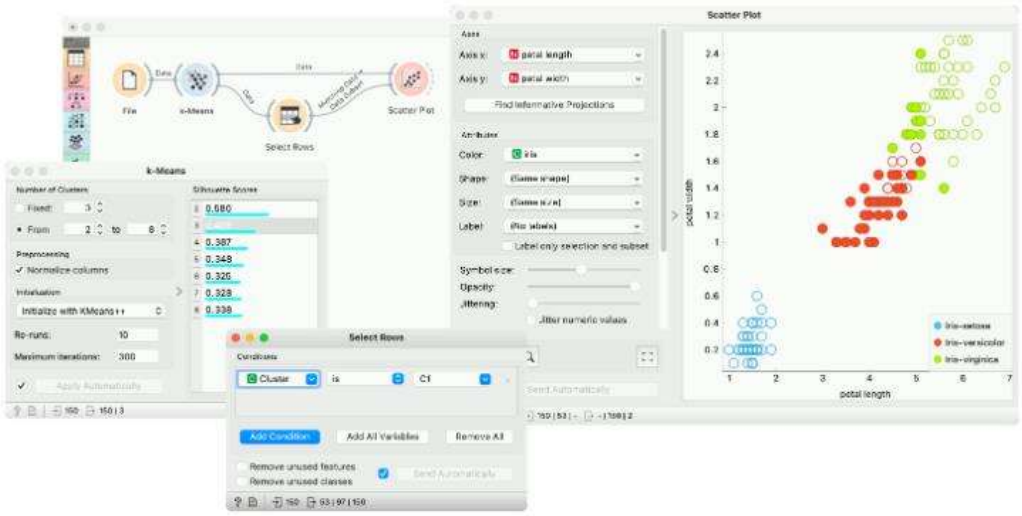
Hierarchical Clustering (MST-like)

1. **Load Data**:
   - Follow the same steps as in K-Means to load and preprocess the data.
2. **Hierarchical Clustering**:
   - Drag the 'Hierarchical Clustering' widget to the canvas.
   - Connect the 'Preprocess' widget to the 'Hierarchical Clustering' widget.
   - Configure linkage method, such as single linkage.
3. **Inspect Dendrogram**:
   - Drag the 'Dendrogram' widget to the canvas.
   - Connect the 'Hierarchical Clustering' widget to the 'Dendrogram' widget.
4. **Cluster Evaluation**:
   - Drag the 'Silhouette Plot' widget to the canvas.
   - Connect the 'Hierarchical Clustering' widget to the 'Silhouette Plot' widget to evaluate clusters.

Assessment Methodology

1. **Silhouette Score**:
   - Measures how similar an object is to its own cluster compared to other clusters.
   - Use the 'Silhouette Plot' widget to assess cluster quality.
2. **Elbow Method**:
   - Plot within-cluster sum of squares for different values of k.
   - Look for an "elbow point" where the decrease in sum of squares becomes linear.
3. **Additional Metrics**:
   - **Dunn Index**: Measures compactness and separation of clusters.
   - **Davies-Bouldin Index**: Measures average similarity ratio of each cluster with its most similar cluster.
   - Use custom scripts or external tools for these metrics if necessary.





Conclusion

By following these steps in Orange, you can effectively perform k-means and MST-based clustering, and assess the quality of your clusters using various evaluation metrics. This structured approach ensures thorough exploratory analysis and meaningful insights from your data.

EXERCISE 4 ASSOCIATION ANALYSIS :DESIGN ALGORITHMS FOR ASSOCIATION RULE MINING ALGORITHMS

Designing algorithms for association rule mining involves several key steps: preparing the data, generating frequent itemsets, and creating association rules. Orange uses established algorithms like Apriori or FP-Growth for these tasks. Below is a detailed description of how you can implement and understand these algorithms within the Orange framework:

Step-by-Step Design

Step 1: Data Preparation

1. **Import Dataset**: Start by loading your dataset into Orange using the 'File' widget. Ensure that the dataset is in the correct format (transactions in rows and items in columns).
2. **Preprocessing**: If necessary, preprocess your data using the 'Preprocess' widget. This might include discretizing continuous variables, handling missing values, or encoding categorical variables.

Step 2: Generating Frequent Itemsets

Orange typically uses the Apriori algorithm to generate frequent itemsets. Here's how it works:

1. **Set Support Threshold**: Define the minimum support threshold. This is the minimum frequency with which an itemset must appear in the transactions to be considered frequent.
2. **Initial Pass**:
    - Count the frequency of each item in the dataset.
    - Identify individual items that

meet the minimum support threshold.

3. **Subsequent Passes**:
    - Generate candidate itemsets of length k from frequent itemsets of length k-1.
    - Count the frequencies of these candidate itemsets in the dataset.
    - Prune itemsets that do not meet the minimum support threshold.
    - Repeat until no more frequent itemsets can be generated.

Step 3: Generating Association Rules

Once frequent itemsets are identified, association rules can be generated. Orange uses the following steps:

1. **Generate Rules**:
    - For each frequent itemset, generate all non-empty subsets.
    - For each non-empty subset, create a rule of the form antecedent→consequent\text{antecedent} \rightarrow \text{consequent}antecedent→consequent, where the antecedent and consequent are disjoint subsets of the itemset.
2. **Calculate Confidence**:
    - For each rule A→BA \rightarrow BA→B, calculate the confidence as Confidence(A→B)=Support(A∪B)Support(A)\text{Confidence}(A \rightarrow B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)}Confidence(A→B)=Support(A)Support(A∪B).
3. **Set Confidence Threshold**:
    - Retain rules that meet or exceed the minimum confidence threshold.
4. **Additional Metrics** (optional):
    - Calculate additional metrics such as lift, leverage, and conviction to evaluate the strength of the rules.

Implementation in Orange

1. **Load Data**:

- Use the 'File' widget to load your dataset.
- Connect the 'File' widget to a 'Data Table' widget to inspect the data.

2. **Preprocess Data** (if necessary):
   - Use the 'Preprocess' widget for any required data transformations.
3. **Generate Frequent Itemsets**:
   - Drag the 'Association Rules' widget to the canvas.
   - Connect the 'File' widget to the 'Association Rules' widget.
   - Set the support threshold in the 'Association Rules' widget.
4. **Generate Association Rules**:
   - In the 'Association Rules' widget, set the confidence threshold.
   - Run the algorithm to generate rules.
5. **Inspect Results**:
   - Connect the 'Association Rules' widget to a 'Data Table' widget to inspect the generated rules.
   - Optionally, use visualization widgets like 'Scatter Plot' or 'Heat Map' to explore the rules.

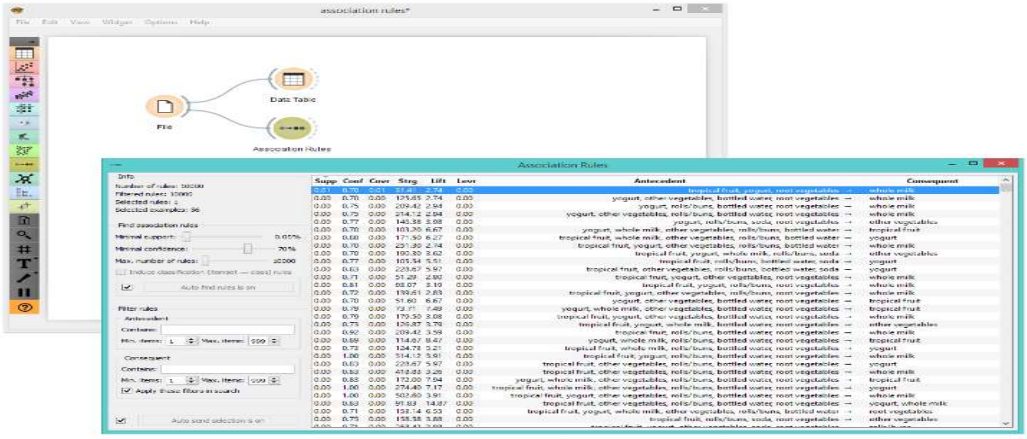
Example

Let's say you have a dataset of retail transactions, and you want to find associations between products:

1. **Load Data**:
   - Use the 'File' widget to load the dataset (e.g., `transactions.csv`).
2. **Set Parameters**:
   - In the 'Association Rules' widget, set a minimum support threshold of 0.01 (1%).
   - Set a minimum confidence threshold of 0.6 (60%).
3. **Generate Rules**:
   - The widget will output rules such as:
     - {bread}→{butter}\{bread\} \rightarrow \{butter\}{bread}→{butter} with confidence 0.8 and support 0.05.
     - {milk,bread}→{eggs}\{milk, bread\} \rightarrow \{eggs\}{milk,bread}→{eggs} with confidence 0.7 and support 0.03.

Example

Association Rules can be used directly with the File widget.



Conclusion

By following these steps, you can effectively design and implement association rule mining algorithms in Orange. Adjusting the support and confidence thresholds allows you to fine-tune the analysis to discover the most significant and interesting rules in your dataset.

EX 5 HYPOTHESIS GENERATION: DERIVE THE HYPOTHESIS FOR ASSOCIATION RULES TO DISCOVERY OF STRONG ASSOCIATION RULES; USE CONFIDENCE AND SUPPORT THRESHOLDS.

Hypothesis Generation

1. **Define the Objective**: Determine what you aim to find with association rule mining. This could be identifying products frequently bought together, common sequences of events, etc.
2. **Select Data**: Choose the dataset that contains the items or events you want to analyze. This dataset should be formatted appropriately for association rule mining, typically in a transactional format.
3. **Identify Items and Transactions**: Define what constitutes an item and a transaction in your dataset. An item could be a product, an event, etc., and a transaction could be a purchase instance, a session, etc.

Discovering Strong Association Rules

To discover strong association rules using Orange, you need to set up and apply association rule mining algorithms, adjusting the confidence and support thresholds to filter out the most significant rules.

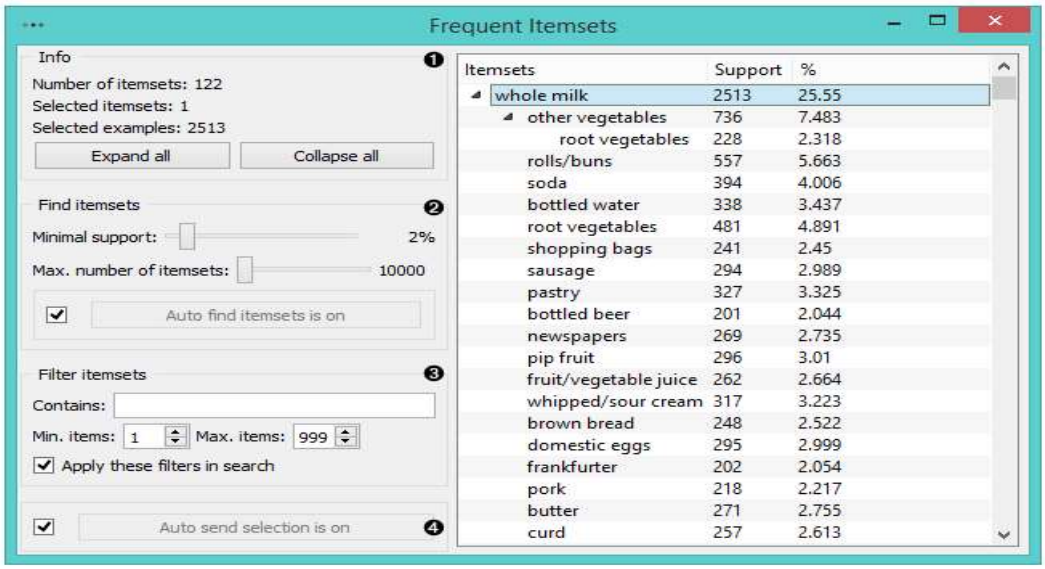
Steps in Orange:

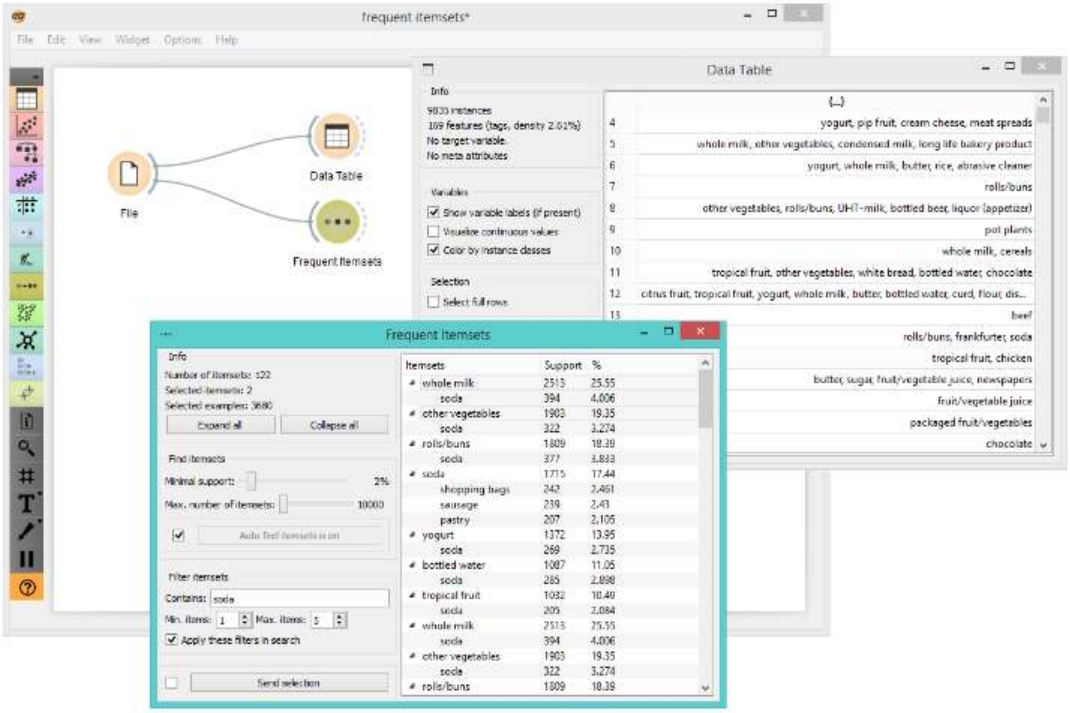
1. **Load Data**: Import your dataset into Orange. You can do this by dragging the 'File' widget and connecting it to the 'Data Table' widget to view the data.
2. **Preprocess Data**: Ensure your data is in the correct format. If needed, use the 'Preprocess' widget to transform your data.
3. **Apply Association Rule Mining**:
    - Drag the 'Association Rules' widget into the workspace.
    - Connect it to the 'File' widget to feed your dataset into the Association Rules widget.
4. **Set Parameters**: In the 'Association Rules' widget:
    - **Support Threshold**: This parameter determines the minimum frequency at which an itemset appears in the dataset. Set a value that is suitable for your dataset size and the rarity of the combinations you are interested in. For example, a support threshold of 0.01 means that the itemset must appear in at least 1% of all transactions.
    - **Confidence Threshold**: This parameter measures the likelihood that a rule is true for a transaction. Set a confidence level that reflects the strength of the rules you are interested in. For example, a confidence threshold of 0.7 means the rule must be true in at least 70% of the cases where the antecedent occurs.
5. **Generate Rules**: Run the algorithm to generate association rules. The output will be a list of rules that meet the specified support and confidence thresholds.
6. **Evaluate and Filter Rules**: Examine the generated rules. You can further filter them by adjusting the thresholds or by using additional metrics such as lift or leverage.

Example of Hypothesis and Rules:

- **Hypothesis**: Customers who buy bread also buy butter.
    - **Support**: The proportion of transactions that include both bread and butter.
    - **Confidence**: The proportion of transactions that include butter among those that include bread.
- **Rule**:
    - **IF** {bread} **THEN** {butter}
    - Support: 0.05 (5% of all transactions include both bread and butter)
    - Confidence: 0.8 (80% of transactions that include bread also include butter)

The widget finds frequent items in a data set based on a measure of support for the rule.





Conclusion:

Using Orange for association rule mining involves setting appropriate support and confidence thresholds to discover meaningful patterns in your data. By adjusting these thresholds and examining the generated rules, you can derive valuable insights and strong association rules from your dataset.

EXERCISE 6
TRANSFORMATION TECHNIQUES: CONSTRUCT HAARWAVELET TRANSFORMATION FOR NUMERICAL DATA, CONSTRUCT PRINCIPAL COMPONENT ANALYSIS (PCA) FOR 5-DIMENSIONAL DATA.

a)Implementing the Haar wavelet transformation for numerical data involves the following steps:

- **Divide the data into pairs**.
- **Calculate the averages and differences** for each pair.
- **Repeat the process** on the averages until you are left with a single value (this process can be stopped earlier if needed).
- Step-by-Step Haar Wavelet Transformation
- **Divide Data into Pairs**:

Suppose you have a list of data points [x1,x2,x3,x4,…,xn] where n is a power of 2. If n is not a power of 2, zero-padding can be applied to the data.

- **Calculate Averages and Differences**:
- For each pair (x2i−1,x2i), calculate:
- Average: ai=2x2i−1+x2i
- Difference: di=2x2i−1−x2i
- The averages will form a new sequence of length n/2, and the differences will form another sequence of length n/2.
- **Repeat the Process**:
- Apply the same process to the new sequence of averages until only one value remains.

```
import numpy as np

def haar_wavelet_transform(data):
    n = len(data)
    output = np.copy(data)
    step = 1

    while step < n:
        for i in range(0, n, 2 * step):
            for j in range(step):
                avg = (output[i + j] + output[i + j + step]) / 2
                diff = (output[i + j] - output[i + j + step]) / 2
                output[i + j] = avg
                output[i + j + step] = diff
        step *= 2

    return output

# Example usage
data = [4, 6, 10, 12, 14, 16, 18, 20]
transformed_data = haar_wavelet_transform(data)
print("Transformed Data:", transformed_data)
```

- output:

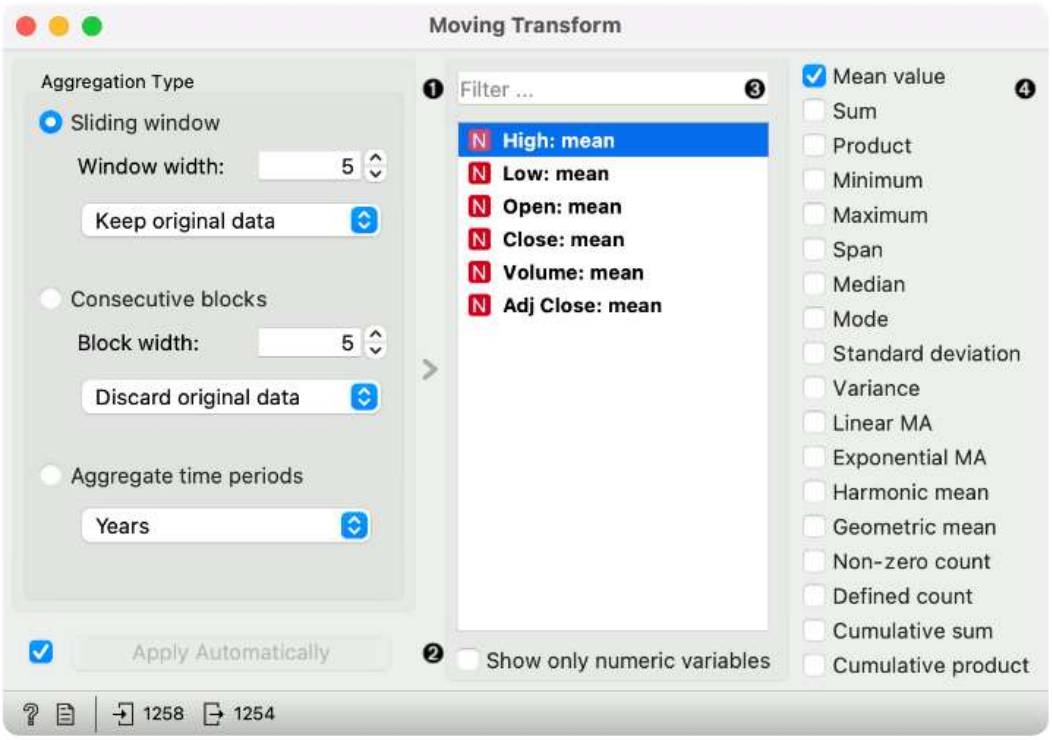
```
Transformed Data: [12 -1 -2 0 -4 0 0 0]
```

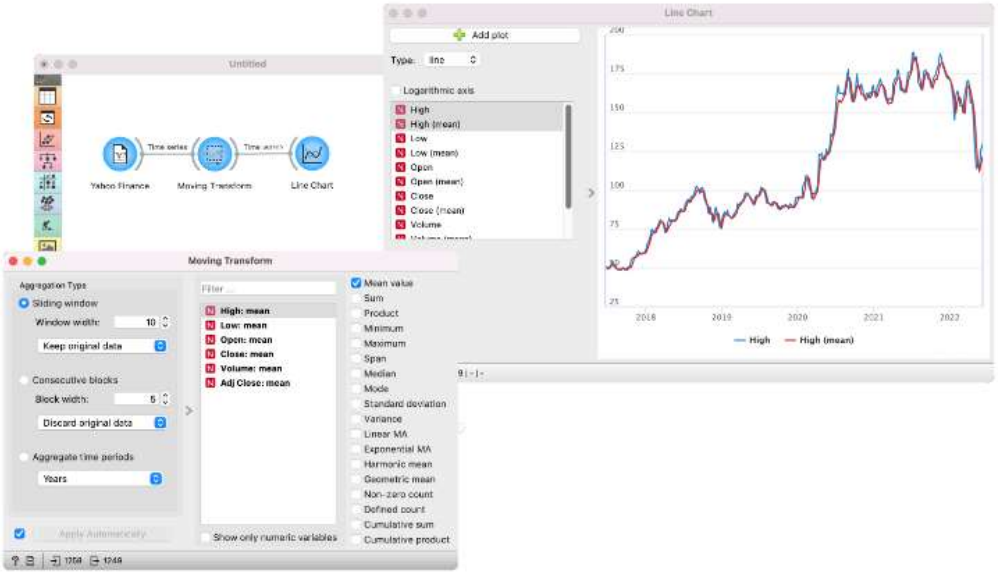
Constructing a Principal Component Analysis (PCA) for 5-dimensional data using the Orange tool can be accomplished through the following steps:

- **Install Orange**: Ensure you have Orange installed. If not, you can install it via pip:
- **Load Orange**:
- Open the Orange application.
- **Import Data**:
- Drag and drop the **'File'** widget to the canvas.
- Double-click the **'File'** widget and load your dataset (ensure it has 5 dimensions).
- **Data Table**:
- Connect the **'File'** widget to the **'Data Table'** widget to inspect your data and ensure it is loaded correctly.
- **PCA**:
- Drag and drop the **'PCA'** widget to the canvas.
- Connect the **'File'** widget to the **'PCA'** widget.
- **PCA Settings**:
- Double-click the **'PCA'** widget to configure it.
- You can choose how many components you want to retain. For visualization purposes, retaining 2 or 3 components is often useful.
- Click **'Apply'** to perform PCA.
- **Visualize**:
- To visualize the results, you can connect the **'PCA'** widget to the **'Scatter Plot'** widget.
- Double-click the **'Scatter Plot'** widget to configure and visualize your data in the new principal component space.

b)Principal Component Analysis (PCA) in Orange

- **Open Orange**: Launch the Orange application.
- **Load Your Data**:
- Drag and drop the **'File'** widget to the canvas.
- Load your dataset.
- **PCA Widget**:
- Drag and drop the **'PCA'** widget to the canvas.
- Connect the **'File'** widget to the **'PCA'** widget.
- Double-click the **'PCA'** widget to configure it.
- **Configure PCA**:
- In the PCA widget, you can specify the number of components you want to retain. For 5-dimensional data, you might want to reduce it to 2 or 3 components for visualization purposes.
- Apply the transformation.
- **Visualize PCA Results**:
- Connect the **'PCA'** widget to the **'Scatter Plot'** widget.
- Double-click the **'Scatter Plot'** widget to configure and visualize the PCA results.





Conclusion

By following these steps, you can effectively Construct Haarwavelet transformation for numerical data and principal component analysis (PCA) for 5-dimensional data

EX 7 :VISUALIZATION: IMPLEMENT BINNING VISUALIZATIONS FOR ANY REAL TIME DATASET, IMPLEMENT LINEAR REGRESSION TECHNIQUES

Binning is a data preprocessing step where continuous data is divided into intervals or "bins." This can be useful for visualizing data distributions or simplifying models. In Orange, you can implement binning visualizations with the following steps:

Steps to Implement Binning Visualization in Orange

1. **Load Data**:
   - Use the 'File' widget to load your dataset.
2. **Binning**:
   - Use the 'Continuize' widget to bin continuous data.
   - Alternatively, use the 'Discretize' widget for more control over the binning process.
3. **Visualize Binned Data**:
   - Use the 'Box Plot', 'Histogram', or 'Distributions' widget to visualize the binned data.

Example Workflow

1. **Load Data**:
   - Drag the 'File' widget to the canvas.
   - Select your dataset file (e.g., `real_time_data.csv`).
2. **Binning Data**:
   - Drag the 'Discretize' widget to the canvas.
   - Connect the 'File' widget to the 'Discretize' widget.
   - Configure the 'Discretize' widget to specify the number of bins or the binning method (e.g., equal-width, equal-frequency).
3. **Visualize Binned Data**:
   - Drag the 'Histogram' widget to the canvas.
   - Connect the 'Discretize' widget to the 'Histogram' widget.
   - Configure the 'Histogram' widget to select the binned variable and visualize its distribution.

Visualizing Binned Data with Box Plot

1. **Load Data**:
   - Follow the steps to load the data using the 'File' widget.
2. **Binning Data**:
   - Follow the steps to bin the data using the 'Discretize' widget.
3. **Box Plot Visualization**:
   - Drag the 'Box Plot' widget to the canvas.
   - Connect the 'Discretize' widget to the 'Box Plot' widget.
   - Configure the 'Box Plot' widget to visualize the distribution of the binned variable across different categories or bins.

Implementing Linear Regression in Orange

Linear regression is a statistical method for modeling the relationship between a dependent variable and one or more independent variables. In Orange, you can implement linear regression as follows:

Steps to Implement Linear Regression in Orange

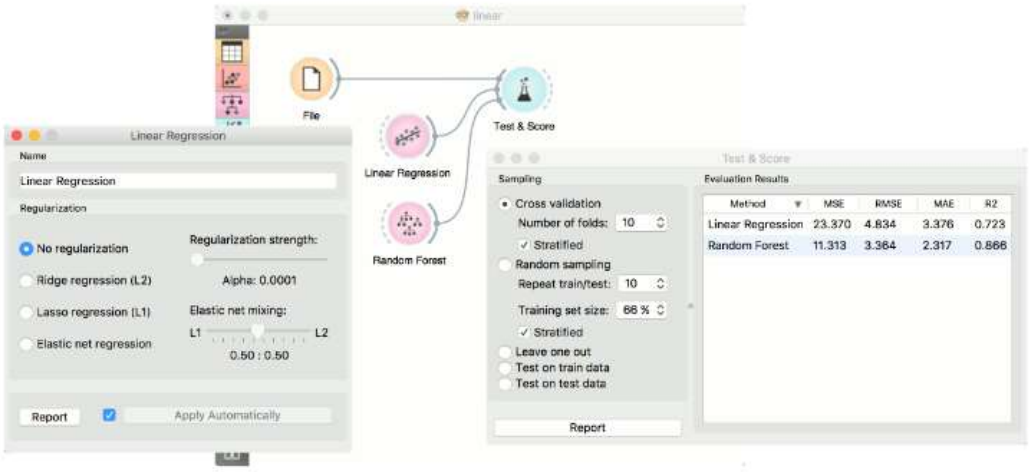
1. **Load Data**:
   - Use the 'File' widget to load your dataset.
2. **Data Preprocessing**:
   - Use the 'Select Columns' widget to select the dependent (target) and independent (predictor) variables.
   - Optionally, use the 'Preprocess' widget to handle missing values, normalize data, or encode categorical variables.

3. **Linear Regression**:
    - Use the 'Linear Regression' widget to perform linear regression analysis.
    - Connect the 'Select Columns' widget to the 'Linear Regression' widget.
4. **Evaluate Model**:
    - Use the 'Test & Score' widget to evaluate the linear regression model.
    - Connect the 'Linear Regression' widget to the 'Test & Score' widget.
    - Connect the 'File' widget (with the data split if necessary) to the 'Test & Score' widget.
5. **Inspect Results**:
    - Use the 'Scatter Plot' widget to visualize the regression line.
    - Connect the 'Linear Regression' widget to the 'Scatter Plot' widget.
    - Configure the 'Scatter Plot' widget to display the regression line and the data points.

Example Workflow

1. **Load Data**:
    - Drag the 'File' widget to the canvas.
    - Select your dataset file (e.g., `real_time_data.csv`).
2. **Select Columns**:
    - Drag the 'Select Columns' widget to the canvas.
    - Connect the 'File' widget to the 'Select Columns' widget.
    - Select the dependent variable (e.g., `y`) and independent variables (e.g., `x1`, `x2`).
3. **Linear Regression**:
    - Drag the 'Linear Regression' widget to the canvas.
    - Connect the 'Select Columns' widget to the 'Linear Regression' widget.
4. **Evaluate Model**:
    - Drag the 'Test & Score' widget to the canvas.
    - Connect the 'Linear Regression' widget to the 'Test & Score' widget.
    - Connect the 'File' widget to the 'Test & Score' widget for evaluation.
5. **Visualize Results**:
    - Drag the 'Scatter Plot' widget to the canvas.
    - Connect the 'Linear Regression' widget to the 'Scatter Plot' widget.
    - Configure the 'Scatter Plot' widget to visualize the regression line and the data points.

	name	coef
6	NOX	-17.7666
9	DIS	-1.47557
12	PTRATIO	-0.952747
14	LSTAT	-0.524758
2	CRIM	-0.108011
11	TAX	-0.0123346
8	AGE	0.000692225
13	B	0.00931168
4	INDUS	0.0205586
3	ZN	0.0464205
10	RAD	0.306049
5	CHAS	2.68673
7	RM	3.80987
1	intercept	36.4595



Conclusion

By following these steps in Orange, you can effectively implement binning visualizations and linear regression analysis. These workflows allow you to preprocess your data, perform exploratory analysis, and build predictive models, ensuring a comprehensive data analysis process.

EXERCISE 8

CLUSTERS ASSESSMENT: VISUALIZE THE CLUSTERS FOR ANY SYNTHETIC DATASET, IMPLEMENT THE PROGRAM FOR CONVERTING THE CLUSTERS INTO HISTOGRAMS

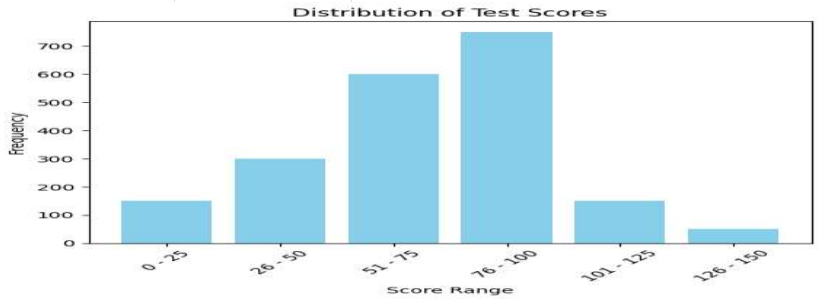
The histogram works by organizing and visualizing the distribution of data into intervals or bins along a continuous scale.

- The range of data values is divided into intervals called "bins." The number of bins and their widths can be predefined or determined algorithmically based on the range and distribution of the data.
- Each data point in the dataset is assigned to a corresponding bin based on its value. As data points are assigned to bins, the frequency or count of data points falling within each bin is calculated.
- The histogram is constructed by plotting the bins along the x-axis and the frequencies (or densities) along the y-axis. Each bin is represented by a bar, and the height of the bar corresponds to the frequency of data points in that bin.

By examining the histogram, you can gain insights into the distribution of the data. You can identify patterns, trends, central tendencies, variability, outliers, and other characteristics of the dataset. For example, a **symmetric bell-shaped histogram** suggests a , while skewed histograms indicate asymmetry in the data.

Suppose you're analyzing the distribution of scores on a standardized test. You have data for 2000 students, and you want to visualize how many students scored within different score ranges. For this you can create a histogram using the following data.

Score Range	Frequency
0-25	150
26-50	300
51-75	600
76-100	750
101-125	150
126-150	50



Histogram

The histogram show that the data is normally distributed, and the students have mostly score between 76-100. This histogram displays the frequency of students falling within different score ranges on the standardized test. Each bar represents a score range, and the height of the bar represents the frequency of students in that range. By customizing the x-axis intervals and the labels, you can effectively visualize the distribution of test scores. Additionally, you can further customize the histogram by changing the y-axis to display percentages or density if needed.

Histogram and its Interpretation

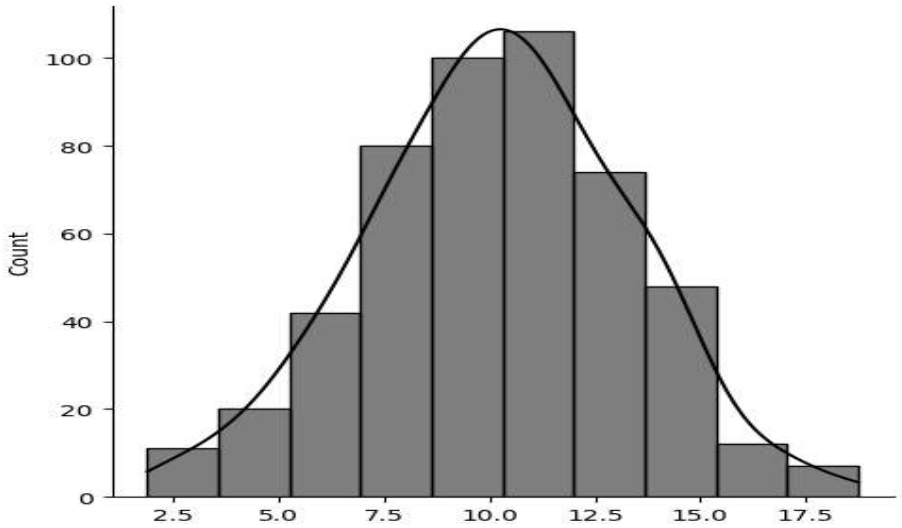
Normal Histogram

Normal histogram is a classical bell-shaped histogram with most of the frequency counts focused on the middle with diminishing tails and there is symmetry with respect to the median. Since the normal distribution is most commonly observed in real-world scenarios, you are most likely to find these. In Normally distributed histogram mean is almost equal to median.

- Python3

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Normal histogram plot
data = np.random.normal(10.0, 3, 500)
sns.displot(data, kde= True, bins=10, color='black')
```

Output:



Normal Distribution Graph

We have plotted a normal distribution graph.

- The peak of the curve represents the mean of the dataset.
- The normal distribution graph is symmetric.

Non-normal Short-tailed/ long-tailed histogram

In short-tailed distribution tail approaches 0 very fast, as we move from the median of data, In the long-tailed histogram, the tail approaches 0 slowly as we move far from the . Here, we refer tail as the extreme regions in the histogram where most of the data is not concentrated and this is on both sides of the peak.

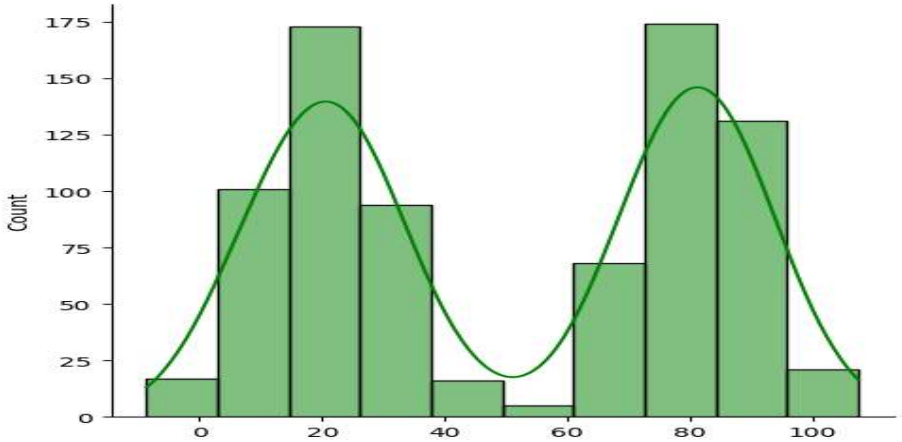
Bimodal Histogram

Aof data represents the most common values in the histogram (i.e. peak of the histogram. A bimodal histogram represents that there are two peaks in the histogram. The histogram can be used to test the unimodality of data. The bimodality (or for instance non-unimodality) in the dataset represents that there is something wrong with the process. Bimodal histogram many one or both of two characters: Bimodal normal distribution and symmetric distribution.

- Python3

```
# Bi-modal histogram
N=400
mu_1, sigma_1 = 80, 10
mu_2, sigma_2 = 20, 10
# Generate two normal distributions of given mean sdand concatenate
X_1 = np.random.normal(mu_1, sigma_1, N)
X_2 = np.random.normal(mu_2, sigma_2, N)
X = np.concatenate([X_1, X_2])
sns.displot(X,bins=10,kde=True , color='green')
```

Output:



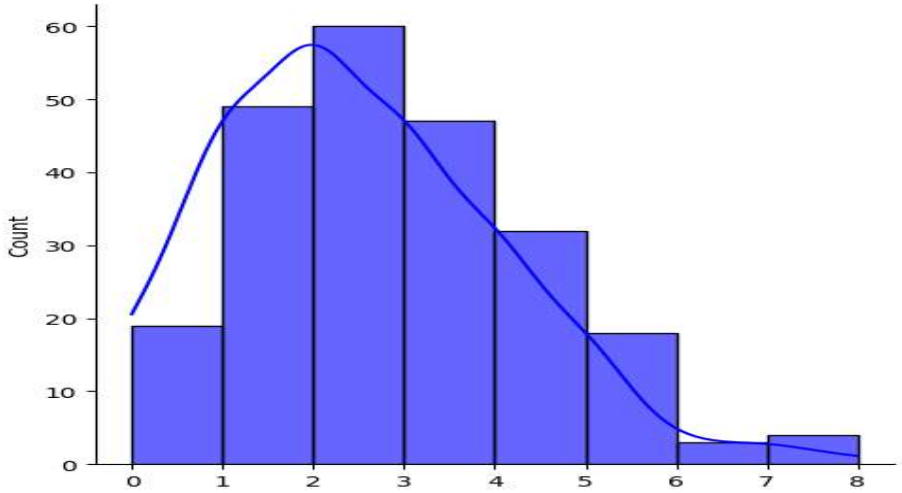
Skewed Left/Right Histogram

is those where the one-side tail is quite clearly longer than the other-side tail. A right-skewed histogram means that the right-sided tail of the peak is more stretched than its left and vice-versa for the left-sided. In a left-skewed histogram, the mean is always lesser than the median, while in a right-skewed histogram mean is greater than the histogram.

Right-skewed Histogram

```
# Right-skewed Histogram
rdata = [0] * 19 + [1]*49 + [2]*60 + [3] * \
    47 + [4]*32 + [5] * 18 + [6]*3 + [7]*3 + [8]
sns.displot(rdata, bins=8, kde=True, alpha=0.6, color='blue')
```

Output:

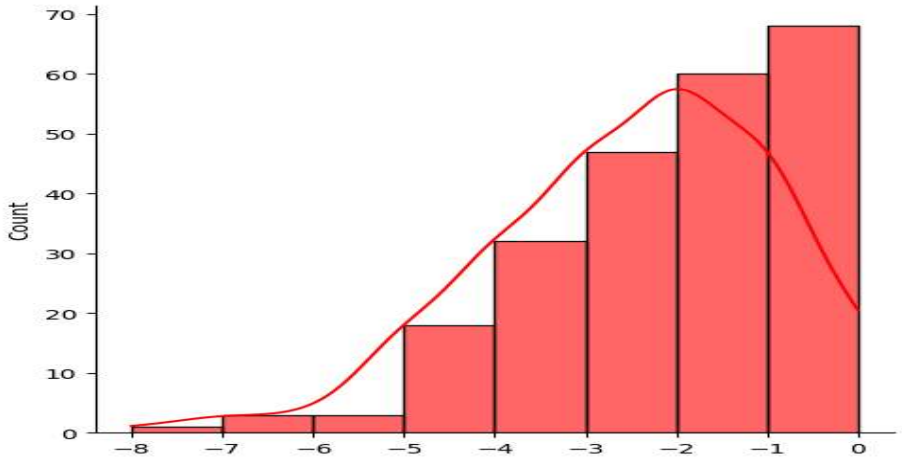


Right Skewed Histogram
Left-skewed Histogram

- Python3

```
# Left-skewed Histogram
ldata = [0]* 19 + [-1]*49 + [-2]*60 + [-3] *47 + [-4]*32 + [-5]* 18+
[-6]*3 + [-7]*3 + [-8]
sns.displot(ldata, kde = True,bins=8,  alpha=0.6, color='red')
```

Output:



Left Skewed Histogram

Uniform Histogram

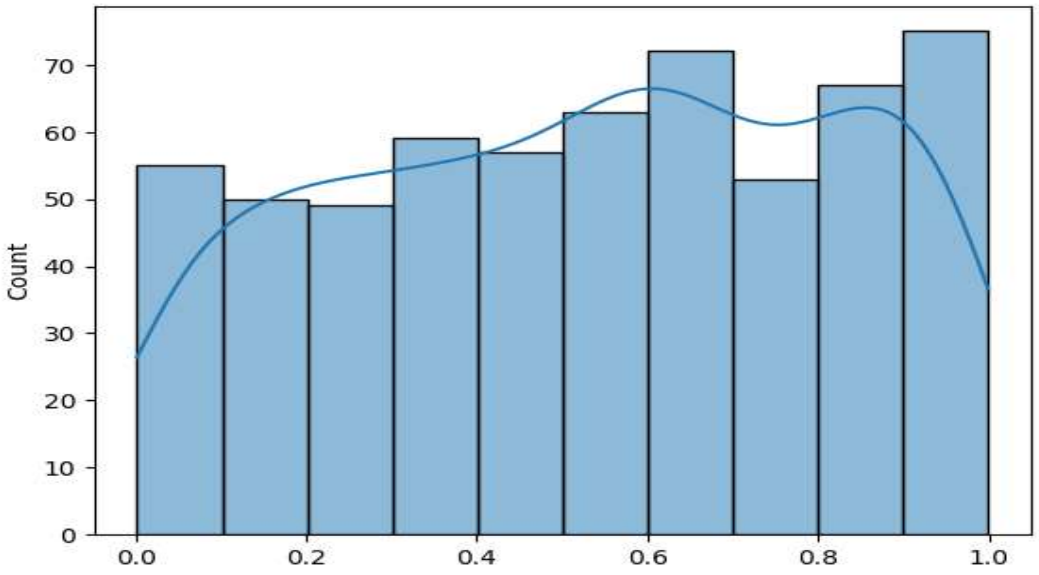
In uniform histogram, each bin contains approximately the same number of counts (frequency). The example of uniform histogram is such as a die is rolled n (n>>30) number of times and record the frequency of different outcomes.

- Python3

```
# Generate random data following a uniform distribution
data = np.random.uniform(low=0, high=1, size=600)
sns.histplot(data, kde =True, bins =10 )
plt.show()
```

Output:

Uniform Distribution



Conclusion;

By following these steps, you can effectively converting the clusters into histograms

EX NO 9 HIERARCHICAL CLUSTERING: WRITE A PROGRAM TO IMPLEMENT AGGLOMERATIVE CLUSTERING TECHNIQUE ,WRITE A PROGRAM TO IMPLEMENT DIVISIVE HIERARCHICAL CLUSTERING TECHNIQUE

Agglomerative Clustering in Orange

Agglomerative clustering is a bottom-up approach where each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

Steps to Implement Agglomerative Clustering

1. **Load Data**:
   - Use the 'File' widget to load your dataset.
2. **Agglomerative Clustering**:
   - Use the 'Hierarchical Clustering' widget to perform agglomerative clustering.
   - Visualize the clustering using the 'Dendrogram' widget.

Example Workflow

1. **Load Data**:
   - Drag the 'File' widget to the canvas.
   - Load your dataset file (e.g., `dataset.csv`).
2. **Agglomerative Clustering**:
   - Drag the 'Hierarchical Clustering' widget to the canvas.
   - Connect the 'File' widget to the 'Hierarchical Clustering' widget.
   - Configure the widget to use agglomerative clustering (default behavior).
3. **Visualize Clustering**:
   - Drag the 'Dendrogram' widget to the canvas.
   - Connect the 'Hierarchical Clustering' widget to the 'Dendrogram' widget.

Divisive Hierarchical Clustering in Orange

Divisive clustering is a top-down approach where all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy. Orange does not provide a built-in widget for divisive clustering, but you can implement it using a Python script.

Python Script for Divisive Hierarchical Clustering

Here is a Python script to implement divisive hierarchical clustering using Orange's scripting capabilities:

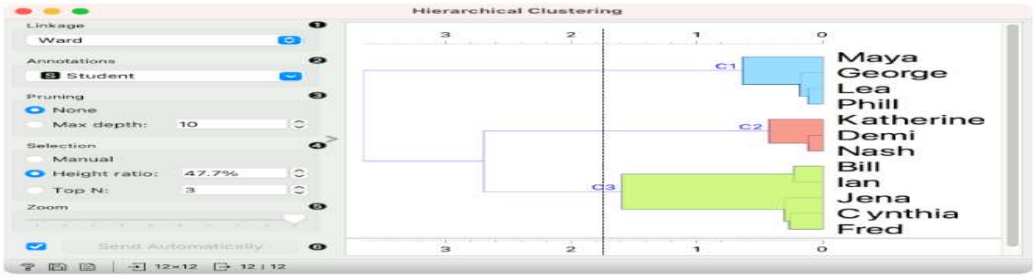
```
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from Orange.data import Table, Domain
from Orange.clustering import hierarchical
from Orange.widgets.utils.plot import OWScatterPlotBase
import matplotlib.pyplot as plt

# Load data
data = Table("your_dataset.csv")

# Convert Orange Table to numpy array
X = np.array([list(row) for row in data])
```

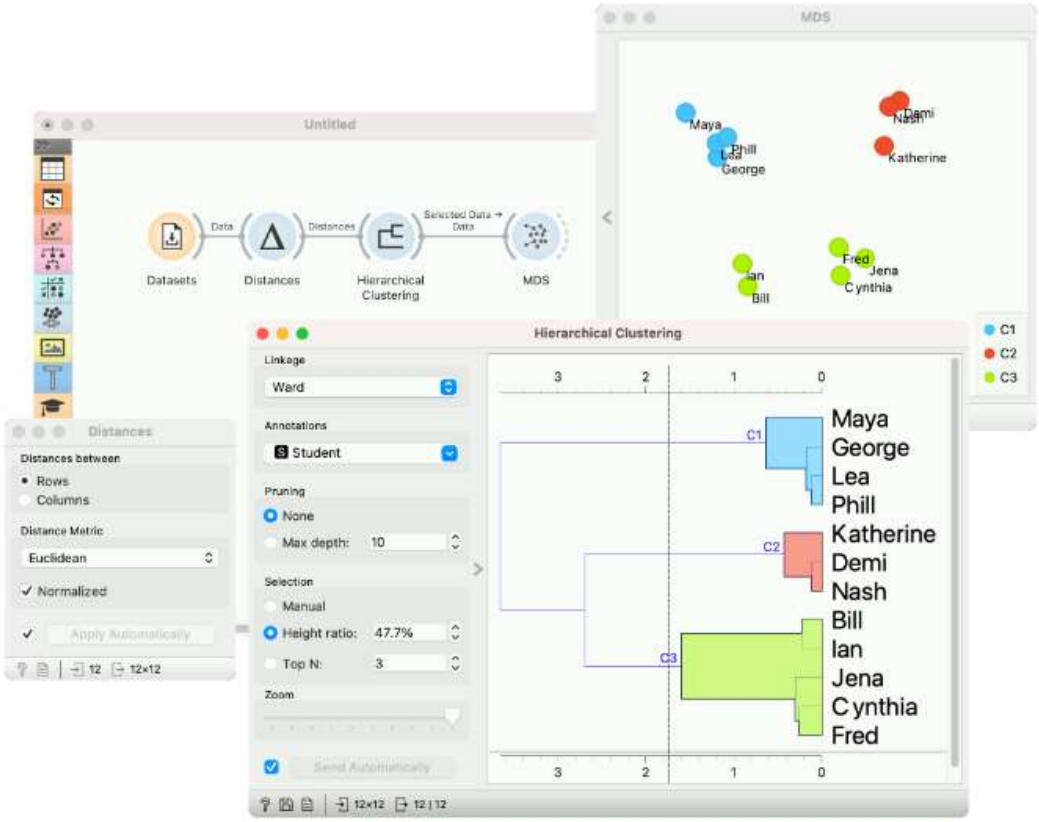
```
# Function to perform divisive hierarchical clustering
def divisive_clustering(X, n_clusters=2):
    # Initially, all points are in one cluster
    clusters = [X]

    while len(clusters) < n_clusters:
        # Find the cluster to split
        cluster_to_split = max(clusters, key=len)
        clusters.remove(cluster_to_split)

        # Perform agglomerative clustering on the chosen cluster
        clustering = AgglomerativeClustering(n_clusters=2)
        labels = clustering.fit_predict(cluster_to_split)

        # Split the chosen cluster into two sub-clusters
        sub_cluster_1 = cluster_to_split[labels == 0]
        sub_cluster_2 = cluster_to_split[labels == 1]

        # Add the sub-clusters to the list of clusters
        clusters.append(sub_cluster_1)
        clusters.append(sub_cluster_2)

    return clusters

# Perform divisive clustering
clusters = divisive_clustering(X, n_clusters=3)

# Visualize the clusters
colors = ['red', 'green', 'blue']
for cluster, color in zip(clusters, colors):
    plt.scatter(cluster[:, 0], cluster[:, 1], c=color)

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Divisive Hierarchical Clustering')
plt.show()
```

How to Run the Script in Orange

1. **Load Data**:
    - Use the 'File' widget to load your dataset.
2. **Python Script**:
    - Drag the 'Python Script' widget to the canvas.
    - Connect the 'File' widget to the 'Python Script' widget.
    - Copy and paste the provided Python script into the 'Python Script' widget.
    - Ensure that the dataset file name in the script matches the file you loaded.
3. **Run Script**:

- Execute the script to perform divisive hierarchical clustering and visualize the results.





Conclusion

By following these steps, you can implement agglomerative and divisive hierarchical clustering techniques in Orange. Agglomerative clustering can be directly performed using the 'Hierarchical Clustering' and 'Dendrogram' widgets, while divisive clustering requires a custom Python script. These clustering techniques provide valuable insights into the structure and relationships within your data.

EX 10 SCALABILITY ALGORITHMS :DEVELOP SCALABLE CLUSTERING ALGORITHMS ,DEVELOP SCALABLE APRIORI ALGORITHM

To develop scalable clustering algorithms and a scalable Apriori algorithm in Orange, we'll focus on techniques and optimizations that can handle large datasets efficiently. While Orange's graphical interface is user-friendly for small to medium-sized datasets, scalable implementations often require custom scripting and leveraging optimized libraries.

Scalable Clustering Algorithms

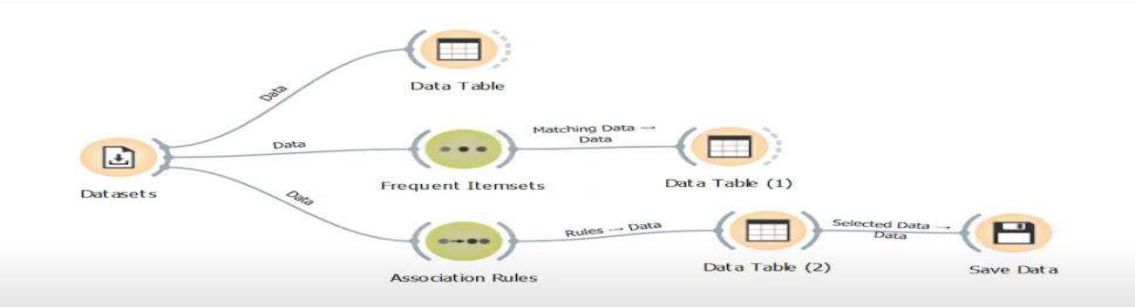
1. **MiniBatch K-Means**: This is a scalable version of the K-Means algorithm that processes mini-batches of the dataset to reduce computational load.
2. **DBSCAN with KD-Tree**: Using a KD-Tree for neighborhood searches can make DBSCAN more scalable.
3. **Hierarchical Clustering with Sparse Data Structures**: Optimizing hierarchical clustering using sparse data structures for large datasets.

Example: MiniBatch K-Means in Orange

Orange does not natively support MiniBatch K-Means via its GUI widgets, but you can use the `Python Script` widget to implement it.

```
import numpy as np
from sklearn.cluster import MiniBatchKMeans
from Orange.data import Table

# Load data
data = Table("your_large_dataset.csv")

# Convert Orange Table to numpy array
X = np.array([list(row) for row in data])

# Define MiniBatch K-Means
mbk = MiniBatchKMeans(init='k-means++', n_clusters=10, batch_size=100, n_init=10, max_no_improvement=10, verbose=0)

# Fit the model
mbk.fit(X)

# Get the cluster centers and labels
cluster_centers = mbk.cluster_centers_
labels = mbk.labels_

# Add the cluster labels back to the Orange Table
data[:, 'Cluster'] = labels
```

```
# Output the table with cluster labels
output_data = data
```



Scalable Apriori Algorithm

The Apriori algorithm can be made scalable by using data reduction techniques, transaction reduction, and parallel processing.

Example: Scalable Apriori in Orange

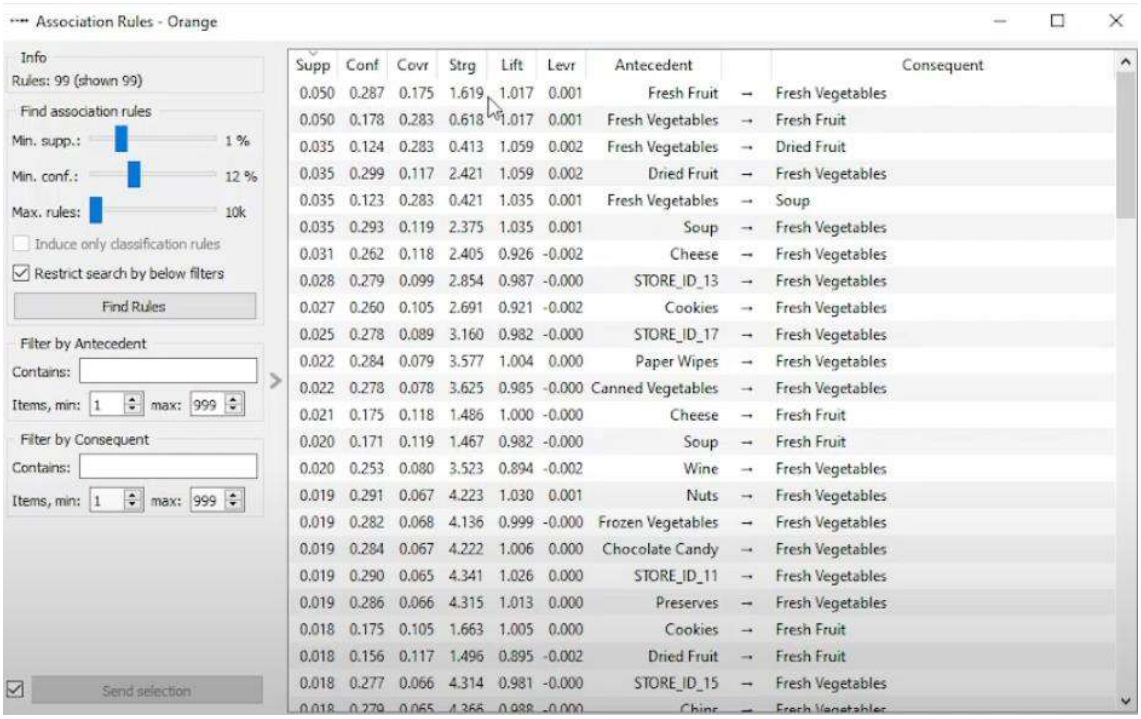
Orange does not provide a scalable version of Apriori directly. However, you can use a Python script with optimized libraries like `apyori` or `mlxtend`.

```
from mlxtend.frequent_patterns import apriori, association_rules
import pandas as pd
from Orange.data import Table, Domain, DiscreteVariable

# Load data
data = Table("your_large_dataset.csv")

# Convert Orange Table to pandas DataFrame
df = pd.DataFrame(data)

# Assume the dataset is in the transaction format with boolean values
# Apply Apriori
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

# Convert results back to Orange Table
domain = Domain([DiscreteVariable.make("itemsets", values=[str(i) for i in range(len(rules))])])
orange_table = Table(domain, np.array(rules))
```

```
# Output the association rules
output_data = orange_table
```

How to Run the Scripts in Orange

1. **Load Data**:
   - Use the 'File' widget to load your large dataset.
2. **Python Script**:
   - Drag the 'Python Script' widget to the canvas.
   - Connect the 'File' widget to the 'Python Script' widget.
   - Copy and paste the provided Python scripts into the 'Python Script' widget.
   - Ensure that the dataset file name in the script matches the file you loaded.



Conclusion:

Scalability in data mining involves using optimized algorithms and data structures to handle large datasets efficiently. In Orange, you can extend its capabilities by using custom Python scripts and leveraging powerful libraries such as `scikit-learn` and `mlxtend`. This approach allows you to implement scalable clustering algorithms and the Apriori algorithm, making it possible to work with large datasets effectively.