International University, HCMC National University
School of Computer Science and Engineering

**FINAL REPORT**

# Sales Clothes Prediction

# Market Basket Analysis

Submitted by.

| No. | FULL NAME | STUDENT ID |
|---|---|---|
| 1 | Đặng Quốc Anh Duy | ITDSIU20015 |
| 2 | Đoàn Hữu Nguyên | ITITIU20260 |
| 3 | Nguyễn Hoàng Minh Tuấn | ITITIU20343 |

Course name: **Data Mining**

**Professors**: Dr. Nguyen Thi Thanh Sang, MSc. Nguyen Quang Phu

May 22, 2023

## TABLE OF CONTENTS

## List of Figures

# 1. Classification/Prediction Models

## 1.1. Introduction

Recently, with the growth of competitive markets, businesses are increasingly investing in data analysis to gain valuable insights that can drive improvements and sustain advantageous features. This trend is particularly pronounced in the context of product sales on E-Commerce platforms, where understanding sales performance is crucial for business success. Accurate predictions of units sold, based on available metadata, have become a cornerstone of strategic planning and decision-making.

E-Commerce platforms generate vast amounts of data related to products, including metadata such as product descriptions, categories, pricing, customer reviews, and more. Leveraging this data to predict sales performance allows businesses to optimize inventory management, marketing strategies, and customer engagement. By anticipating demand, companies can reduce costs, increase revenue, and enhance customer satisfaction.

The development and application of robust prediction models are essential for these purposes. Machine learning algorithms, such as Support Vector Machines (SVM), Decision Trees, and k-Nearest Neighbors (KNN), offer powerful tools for analyzing complex datasets and making accurate sales predictions. These models can uncover patterns and relationships within the data that may not be immediately apparent, providing a significant competitive advantage.

In this context, the ability to predict units sold based on product metadata plays a crucial role in enabling businesses to make informed decisions, improve operational efficiency, and ultimately succeed in the competitive landscape of E-Commerce. This report discusses the implementation and evaluation of various machine learning models to predict sales performance, providing insights into the most effective approaches and their practical applications.

## 1.2. Objectives

The main goal of this project is to perform the prediction on units sold of products based on various features of the sales data and to figure out the answer to the question that what are the key features that affect the units sold.

Here is the list of objectives to be achieved:

- Data Collection: Select the suitable dataset for exploration
- Data Transformation: Convert data into right format in ARFF/CSV
- Data Preprocessing: Clean the data through many techniques like detecting outliers, handling missing values, standardize the data, or encoding the categorical attributes
- Exploratory Data Analysis: Analyze the raw data, visualize some plots to get brief information about the dataset. This is where we can ask questions for exploration
- Implementation with Weka API and Python: perform the sequence of steps to implement the prediction algorithms.
- Model Evaluation: Compare the performance of the defined algorithms, give remarks based on the predicted results

## 1.3. Resources and Tools

**Resources:**

- Dataset: Sales of Summer Clothes in E-Commerce Wish
- Source: Kaggle
- The dataset contains 1573 observations with 43 variables in total

**Tools:**

- Data Preprocessing: Python
- Data Visualization: matplotlib, seaborn
- Model Building: Java, Python
- Library: sklearn, pandas (Python), weka API (Java)

## 1.4. Project Workflow

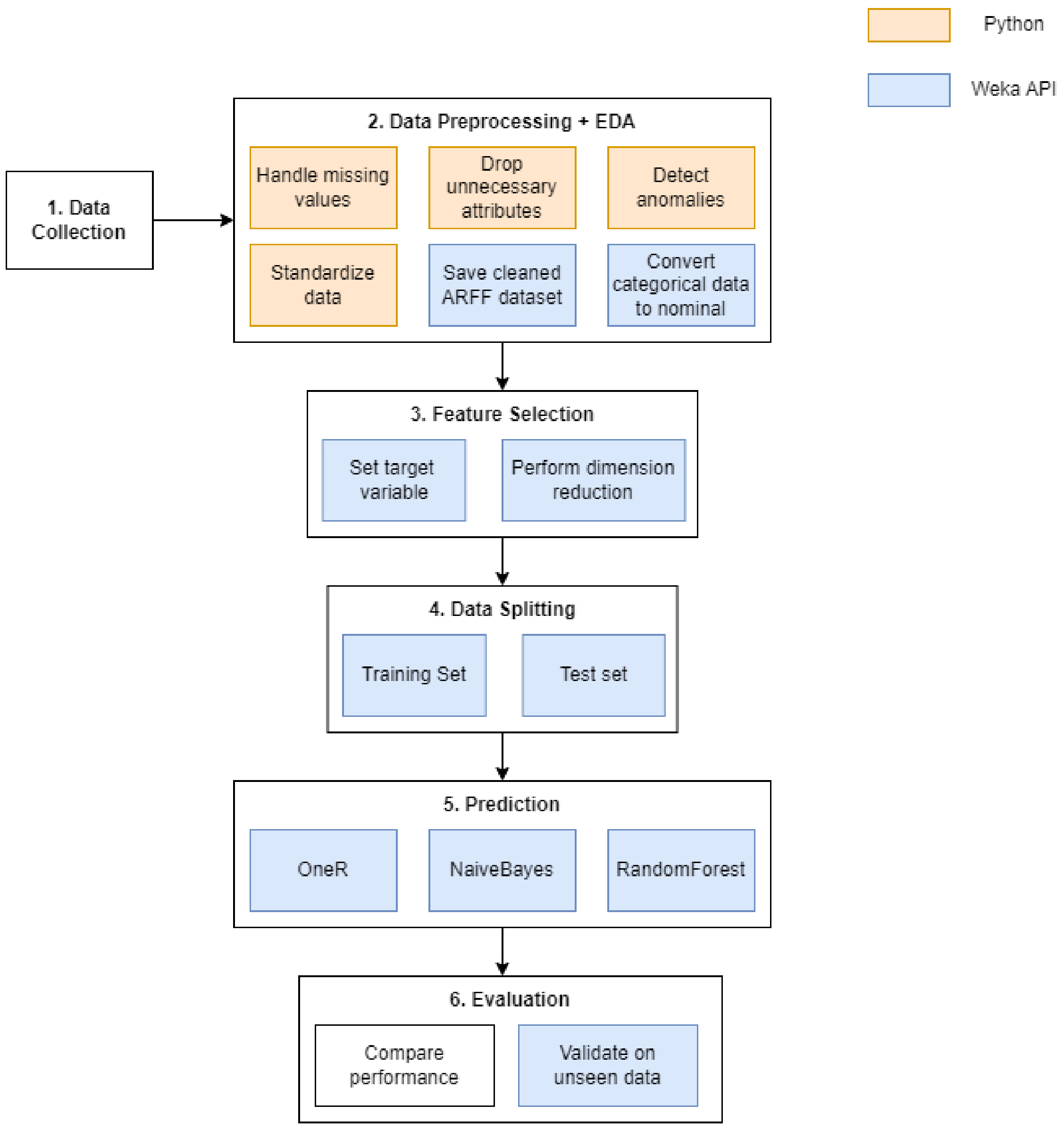


*Figure 1.1: Prediction Workflow*

## 1.5. Data Preprocessing + Exploratory Data Analysis

### Data Preprocessing

To prepare cleaned data for applying prediction models in Weka, the dataset must be preprocessed through many stages in Python.

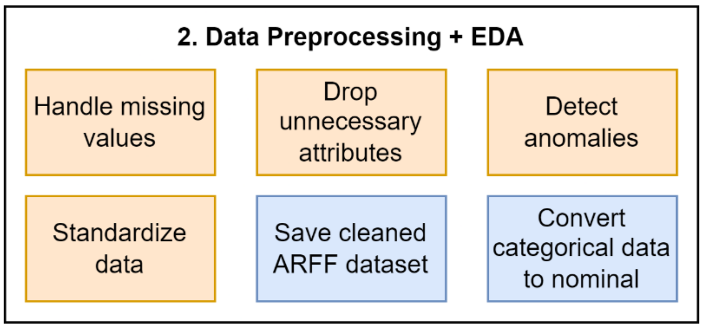

2. Data Preprocessing + EDA
Handle missing values
Drop unnecessary attributes
Detect anomalies
Standardize data
Save cleaned ARFF dataset
Convert categorical data to nominal

*Figure 1.2: Data Preprocessing Process*

First, all the missing values are detected by using pandas' function to check sum of null values for each variable before dropping the unnecessary attributes that contain null like title name, product_url, merchant_id, etc. Now, the important variables with missing values will be handled by many techniques like filling with mean, 0 or drop if the weights of those attributes do not affect the distribution of the dataset. For instance, rating count are converted to 0 if there exists a blank or null values.

Next, the anomalies of the data are found by visualizing the distribution of variables with the big gap between min and max values. These data points are dropped to ensure the normal distribution of variables and no confusion for the prediction models. In the phase of standardization, the data with multiple categories will be grouped into more specific classes. In the context of this project, product color is categorized into specific groups (ex: navyblue → blue). After finishing the preprocessing stage, cleaned data is saved in CSV file before converting to ARFF format using Weka API in Java for model implementation.

However, the identification of Weka can be various with wrong detection of data types. As a result, attributes must be converted to correct format to apply prediction models. Some of the filter functions to perform this transformation in Weka API are NumericalToNominal, StringToNominal, NominalToBinary, etc.

## Exploratory Data Analysis

This part represents some visualizations throughout the EDA stage to see the distribution of the dataset, what are the key takeaways to be achieved and context of the dataset.

The main purpose of the project is to perform predictions on the number of units sold based on the e-commerce features. As you can see from the plot, most of the products are sold around the quantities of 100 and 1000 items.

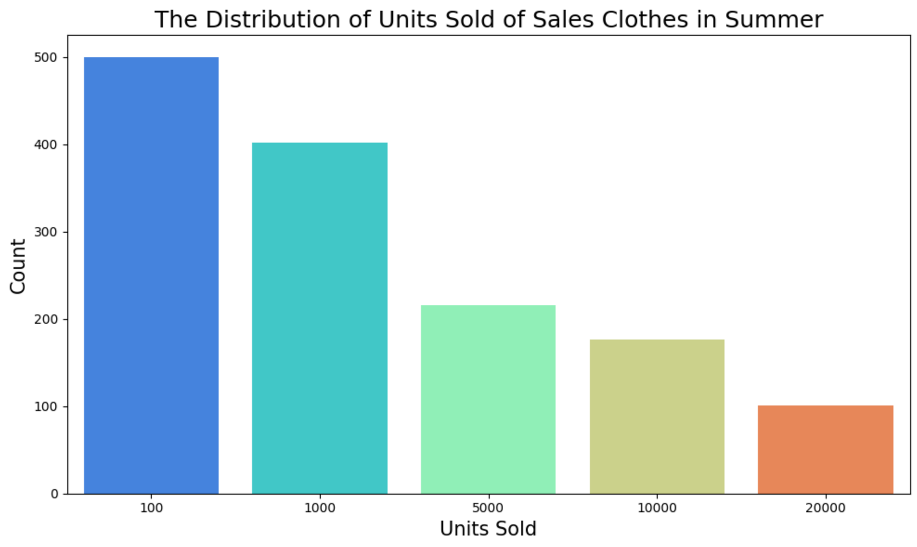


*Figure 1.3: The Distribution of Units Sold*

The tag distribution in the dataset indicates a strong emphasis on seasonal and gender-specific themes, with "summer" and "women's fashion" being the most dominant, each appearing over 1,300 times. General tags like "fashion" and "women" also show significant frequency, reflecting a broad focus on women's apparel. The prominence of the "casual" tag suggests a notable preference for daily, comfortable clothing styles.

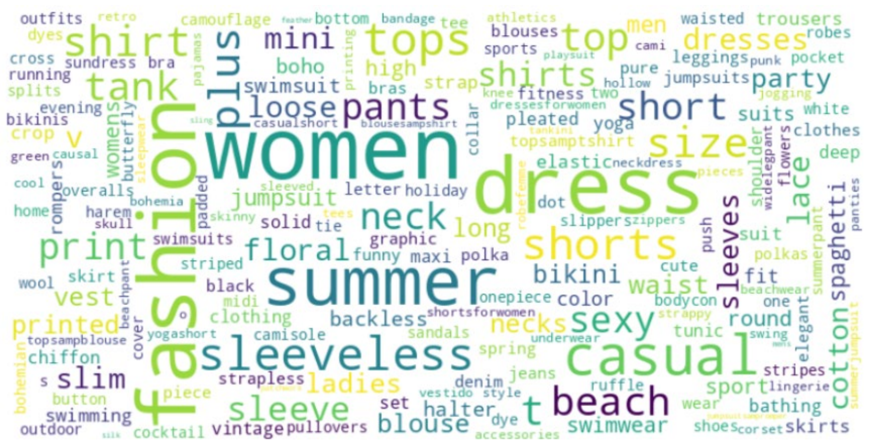


*Figure 1.4: Word Cloud for tags*

The distribution of clothing products by color shows a significant dominance of white, black, and blue classes, indicating a strong preference for these hues. The remaining colors have a relatively equal and minor distribution, suggesting less variation in customer preference outside the main three. This highlights a trend towards classic and versatile colors in the product line.

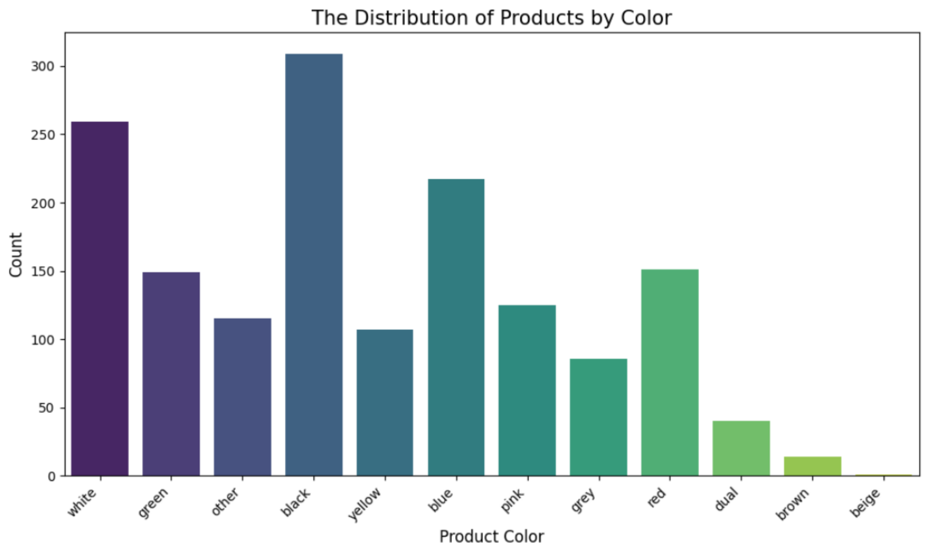


*Figure 1.5: The distribution of clothes by color*

The correlation between features and target labels plays a crucial role in performing feature selection to reduce the noises of training model and the dimensions of the dataset. From the heatmap below, features with high correlation against units sold are related to rating from the customers. These features will be selected as the key attributes to perform model training.

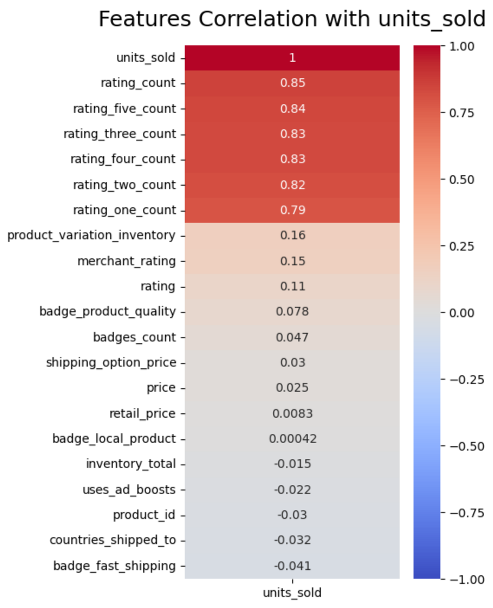
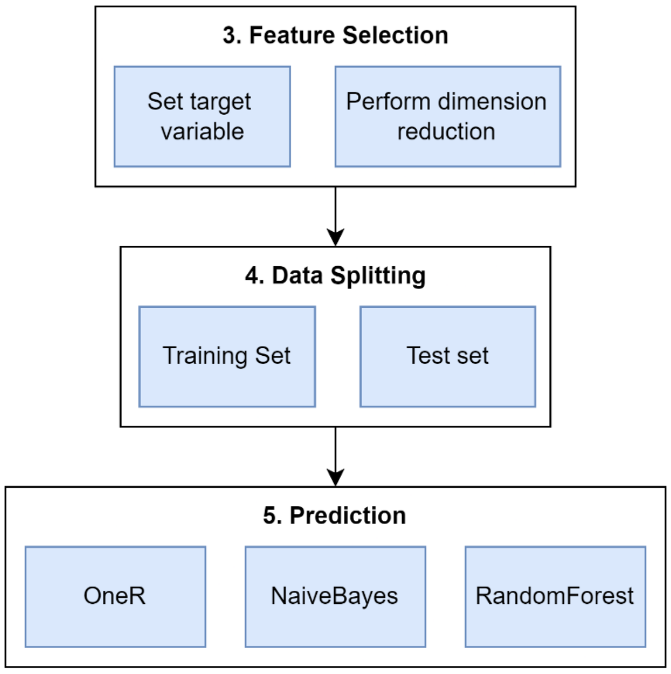


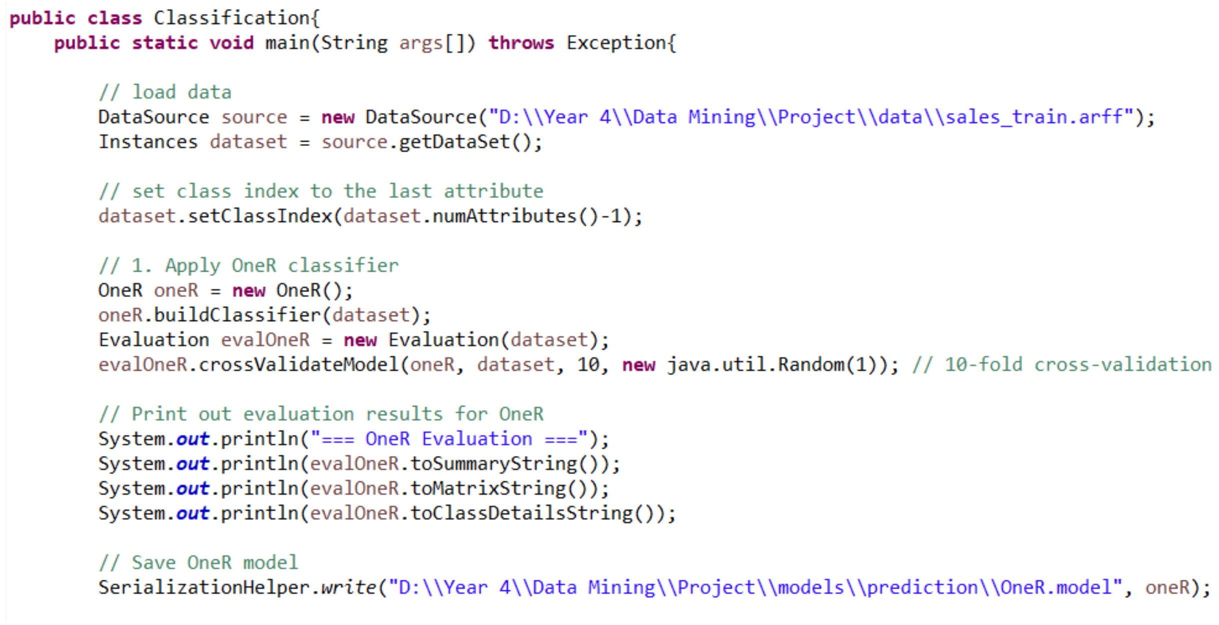
*Figure 1.6: Features Correlation against units sold*

## 1.6. Implement prediction algorithms (Weka API)

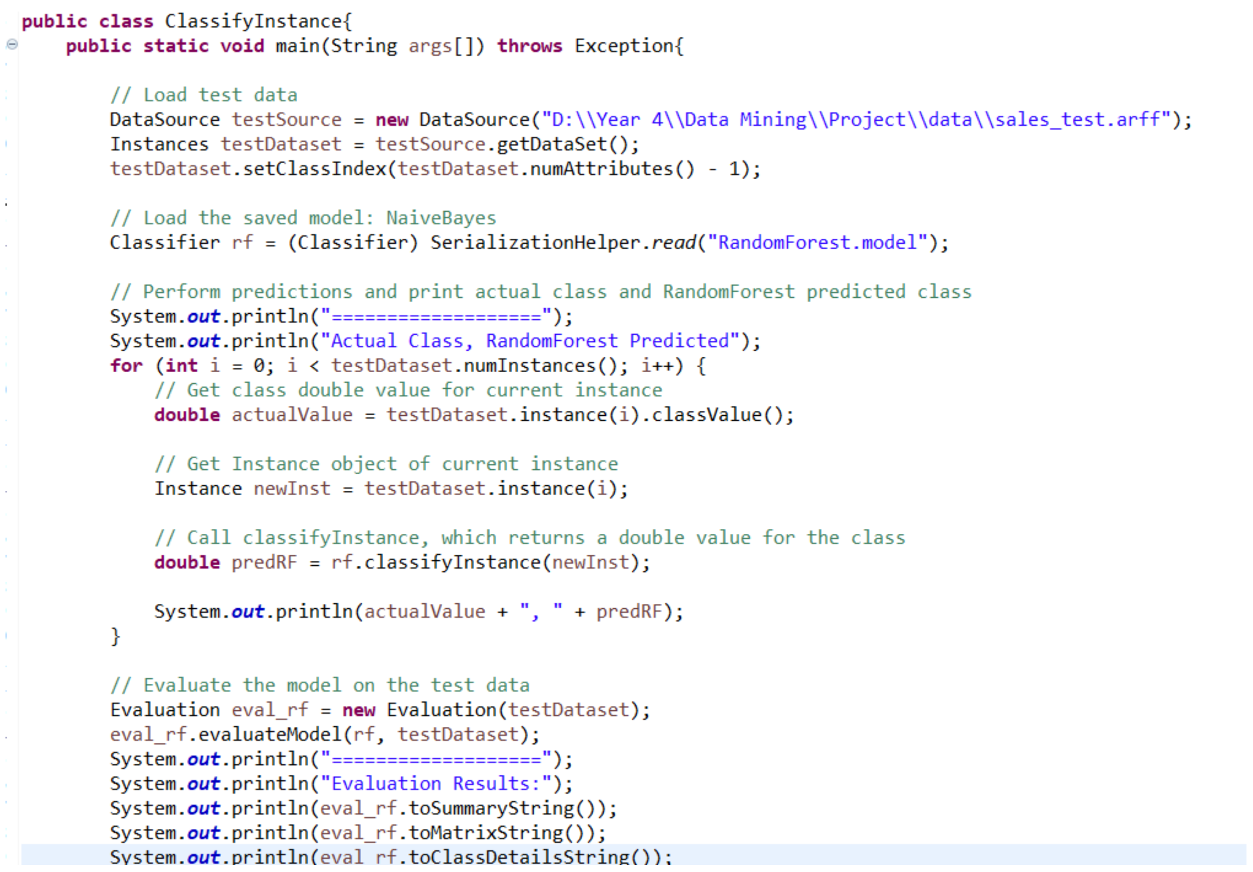
To implement the analysis using the Weka API, the process begins with setting the target variable in the dataset. Following this, dimensionality reduction is performed by applying various attribute evaluators and search methods to identify the most significant features. Once the optimal features are selected, the dataset is split into training and test sets. The dataset is loaded using the "DataSource" class, and the target variable is set to the last attribute of the dataset with the "setClassIndex" method. This setup ensures that the classifiers know which attribute to predict.

The implementation proceeds with the application of three classifiers including OneR, NaiveBayes, and RandomForest, which are generally performed good prediction among other models. After training, these models are evaluated to determine the best-performing ones.



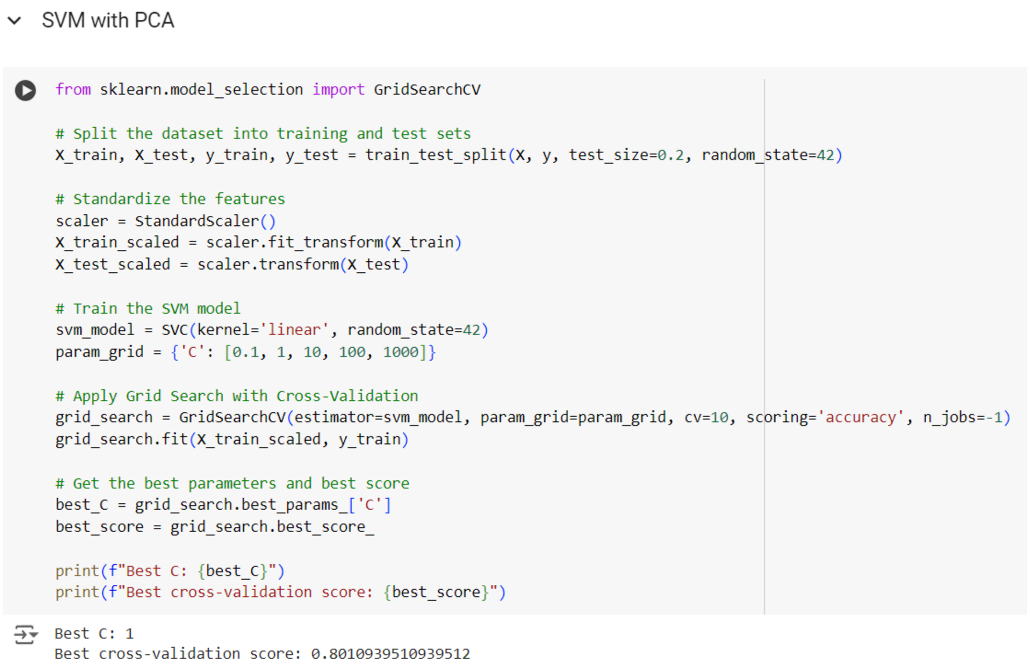


```java
public class Classification{
    public static void main(String args[]) throws Exception{

        // load data
        DataSource source = new DataSource("D:\\Year 4\\Data Mining\\Project\\data\\sales_train.arff");
        Instances dataset = source.getDataSet();

        // set class index to the last attribute
        dataset.setClassIndex(dataset.numAttributes()-1);

        // 1. Apply OneR classifier
        OneR oneR = new OneR();
        oneR.buildClassifier(dataset);
        Evaluation evalOneR = new Evaluation(dataset);
        evalOneR.crossValidateModel(oneR, dataset, 10, new java.util.Random(1)); // 10-fold cross-validation

        // Print out evaluation results for OneR
        System.out.println("=== OneR Evaluation ===");
        System.out.println(evalOneR.toSummaryString());
        System.out.println(evalOneR.toMatrixString());
        System.out.println(evalOneR.toClassDetailsString());

        // Save OneR model
        SerializationHelper.write("D:\\Year 4\\Data Mining\\Project\\models\\prediction\\OneR.model", oneR);
```



```java
public class ClassifyInstance{
    public static void main(String args[]) throws Exception{

        // Load test data
        DataSource testSource = new DataSource("D:\\Year 4\\Data Mining\\Project\\data\\sales_test.arff");
        Instances testDataset = testSource.getDataSet();
        testDataset.setClassIndex(testDataset.numAttributes() - 1);

        // Load the saved model: NaiveBayes
        Classifier rf = (Classifier) SerializationHelper.read("RandomForest.model");

        // Perform predictions and print actual class and RandomForest predicted class
        System.out.println("===================");
        System.out.println("Actual Class, RandomForest Predicted");
        for (int i = 0; i < testDataset.numInstances(); i++) {
            // Get class double value for current instance
            double actualValue = testDataset.instance(i).classValue();

            // Get Instance object of current instance
            Instance newInst = testDataset.instance(i);

            // Call classifyInstance, which returns a double value for the class
            double predRF = rf.classifyInstance(newInst);

            System.out.println(actualValue + ", " + predRF);
        }

        // Evaluate the model on the test data
        Evaluation eval_rf = new Evaluation(testDataset);
        eval_rf.evaluateModel(rf, testDataset);
        System.out.println("===================");
        System.out.println("Evaluation Results:");
        System.out.println(eval_rf.toSummaryString());
        System.out.println(eval_rf.toMatrixString());
        System.out.println(eval_rf.toClassDetailsString());
```

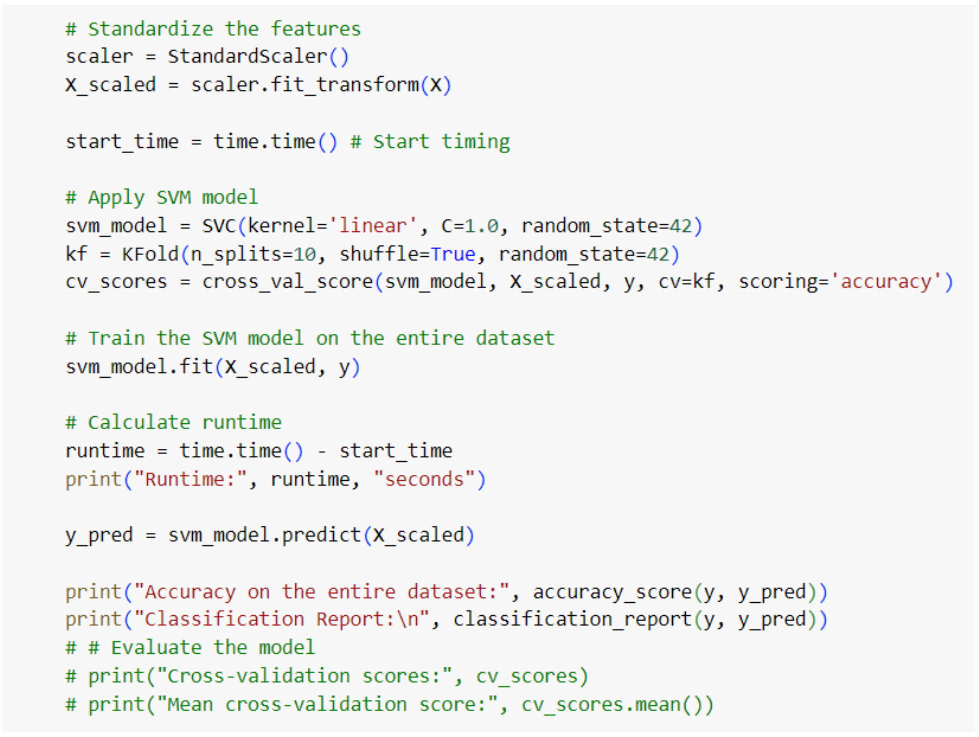
## 1.7. Implement another prediction algorithm (Python)


SVM with PCA

```
from sklearn.model_selection import GridSearchCV

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the SVM model
svm_model = SVC(kernel='linear', random_state=42)
param_grid = {'C': [0.1, 1, 10, 100, 1000]}

# Apply Grid Search with Cross-Validation
grid_search = GridSearchCV(estimator=svm_model, param_grid=param_grid, cv=10, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters and best score
best_C = grid_search.best_params_['C']
best_score = grid_search.best_score_

print(f"Best C: {best_C}")
print(f"Best cross-validation score: {best_score}")
```

```
Best C: 1
Best cross-validation score: 0.8010939510939512
```

To improve the result compared to the previous approaches, we tend to use SVM with 2 options with and without PCA to get the best version of prediction model. The Grid Search is applied to figure out the best value for C before applying SVM model with the specified parameters. After that, we print out the classification report and confusion matrix to show the performance of the model and use those results to compare with others model.



```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

start_time = time.time() # Start timing

# Apply SVM model
svm_model = SVC(kernel='linear', C=1.0, random_state=42)
kf = KFold(n_splits=10, shuffle=True, random_state=42)
cv_scores = cross_val_score(svm_model, X_scaled, y, cv=kf, scoring='accuracy')

# Train the SVM model on the entire dataset
svm_model.fit(X_scaled, y)

# Calculate runtime
runtime = time.time() - start_time
print("Runtime:", runtime, "seconds")

y_pred = svm_model.predict(X_scaled)

print("Accuracy on the entire dataset:", accuracy_score(y, y_pred))
print("Classification Report:\n", classification_report(y, y_pred))
# # Evaluate the model
# print("Cross-validation scores:", cv_scores)
# print("Mean cross-validation score:", cv_scores.mean())
```

## 1.8. Model Evaluation

Model evaluation stage is conducted using 10-fold cross-validation to ensure robust performance metrics. This method splits the dataset into ten parts, using nine for training and one for testing, rotating through all parts. The evaluation will focus on the accuracy, precision, recall, and F1-score of the classification/prediction models. Additionally, the run-time for building models and making predictions will be considered to assess their efficiency. The experimental results are expected to reveal insights into each model's strengths and weaknesses, with particular attention to how well they balance prediction accuracy with computational efficiency.

| Models | Accuracy (%) | MAE | RMSE | Run-time (ms) |
|---|---|---|---|---|
| OneR | 76.13% | 0.0955 | 0.309 | 388 |
| NaiveBayes | 77.92% | 0.0888 | 0.2843 | 103 |
| RandomForest | 80.07% | 0.108 | 0.2385 | 2875 |
| SVM | 81.94% | 0.185 | 0.442 | 1346 |
| SVM (PCA) | 76.71% | 0.2394 | 0.5023 | 610 |

The SVM model achieved the highest accuracy at 81.94% but exhibited higher errors and longer run-time compared to others. Random Forest provided a good balance with 80.07% accuracy and the lowest RMSE of 0.2385, though it had the longest run-time at 2875 ms. NaiveBayes, while slightly less accurate at 77.92%, had the lowest MAE and the fastest run-time at 103 ms. OneR and SVM (PCA) demonstrated lower accuracy and higher errors, making them less suitable for more complex tasks. So, for the selection of stable model, it must be SVM without PCA, but for the requirement of running time, NaiveBayes should be the chosen one.

# 2. Sequence Mining

## 2.1. Introduction

Sequence mining is a technique in data mining that focuses on identifying patterns or regularities within sequential data. This report explores sequence mining using the bread basket dataset from Kaggle. The dataset contains transactional data from a bakery, including the date, time, and items purchased. The goal is to extract meaningful patterns that can help in understanding customer purchasing behavior and optimizing product placements and inventory management.

## 2.2. Objectives

The primary objectives of this report are:

- To understand the fundamental concepts of sequence mining.
- To preprocess data for sequence analysis.
- To implement sequence mining algorithms using the Weka API.
- To develop and compare another sequence mining algorithm using Python.
- To evaluate the performance of the implemented models.

## 2.3. Resources and Tools

**Resources:**

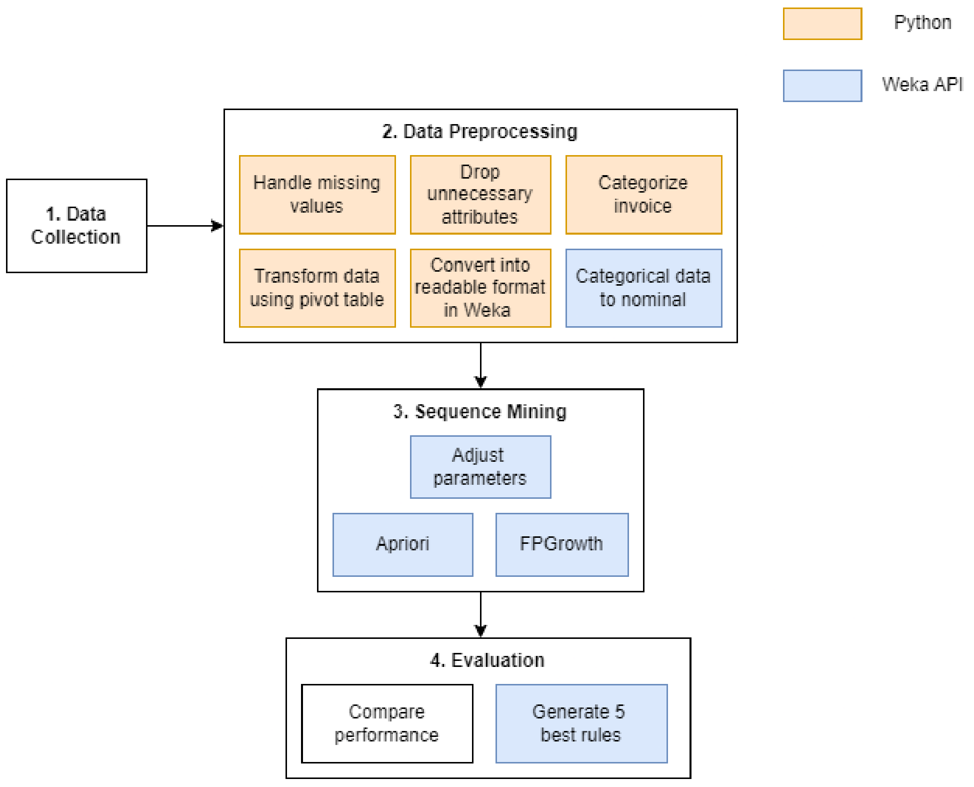
- Dataset: The Bread Basket dataset
- Source: Kaggle
- The dataset contains … observations with … variables in total

**Tools:**

- Data Preprocessing: Python
- Data Visualization: matplotlib, seaborn

- Model Building: Java, Python
- Library: sklearn, pandas (Python), weka API (Java)

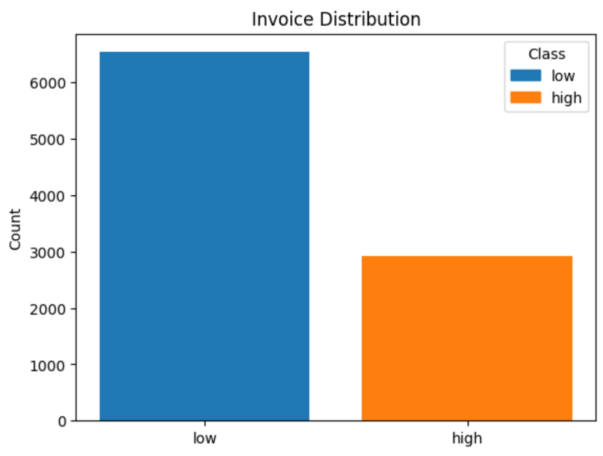
## 2.4. Project Workflow



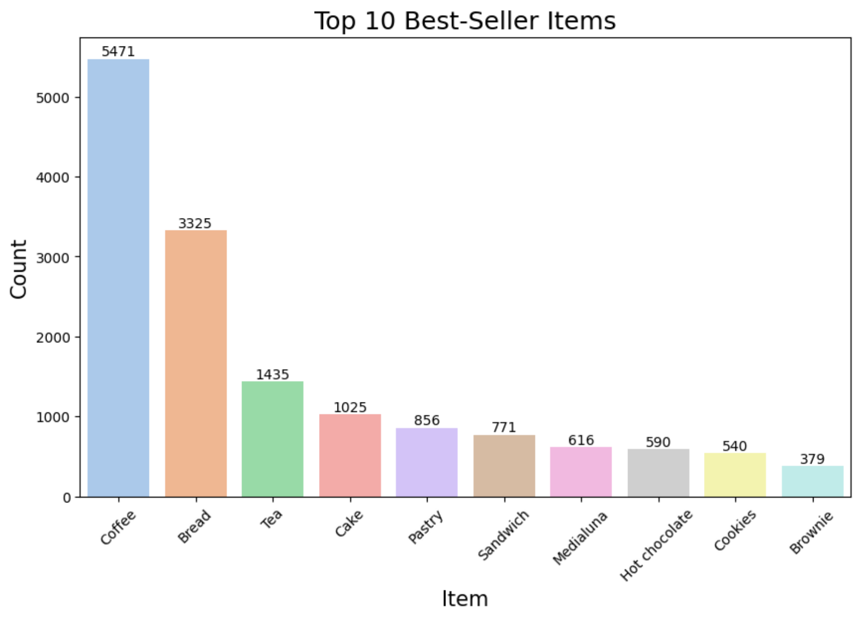
## 2.5. Data Preprocessing + Exploratory Data Analysis

In every data mining process, the preprocessing and EDA plays a crucial role to identify key features of the data and to make the data cleaned before applying the model. The important attributes in our sequence mining task is using the invoice ID with the items along with each invoice ID. So as to group the invoice into 2 groups low and high to improve the decision of the generated rules of the sequence mining models, we divided the invoice into two groups where the number of items that are greater than 2 will be served as group ‘high’ and the rest are belonged to group ‘low’. Next, the transformation of the data has done by using the pivot table in Python to convert the name of item to be the attribute and each row are represented by each Invoice ID. If the invoice include the specified item, the value will be filled with 1 else the blank are filled by np.nan of the numpy library.

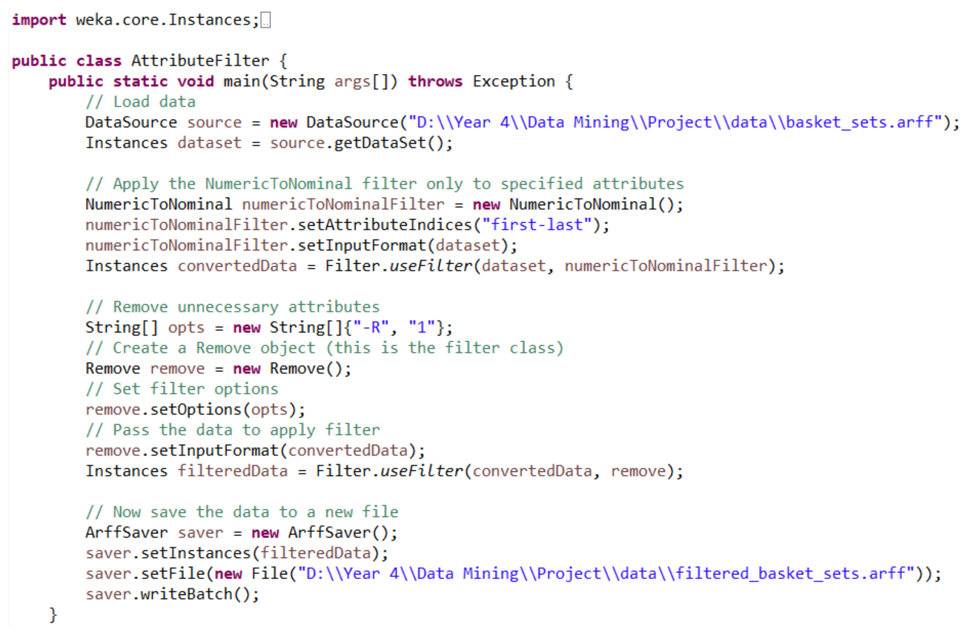
You can see that the distribution of the two class are quite good, with over 6000 observations for class low, and nearly 3000 for the high class. This attribute helps a lot in the generation of rules in sequence mining model.



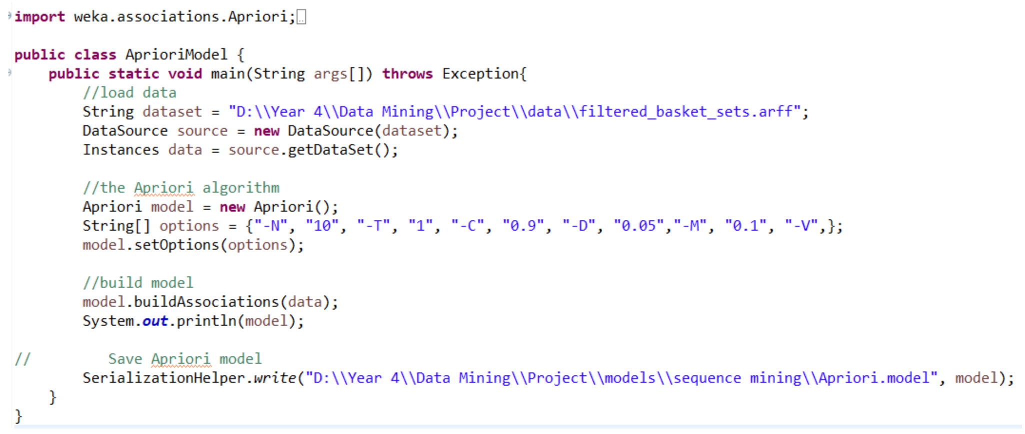
From the visualization of top 10 best-seller items of the bread basket dataset, the plot showcases Coffee as the most prevalent item, with 5471 occurrences, followed by Bread (3325 occurrences) and Tea (1435 occurrences). Additionally, Cake, Pastry, Sandwich, Medialuna, Hot chocolate, Cookies, and Brownie were depicted, highlighting the diversity of products contributing to the dataset.



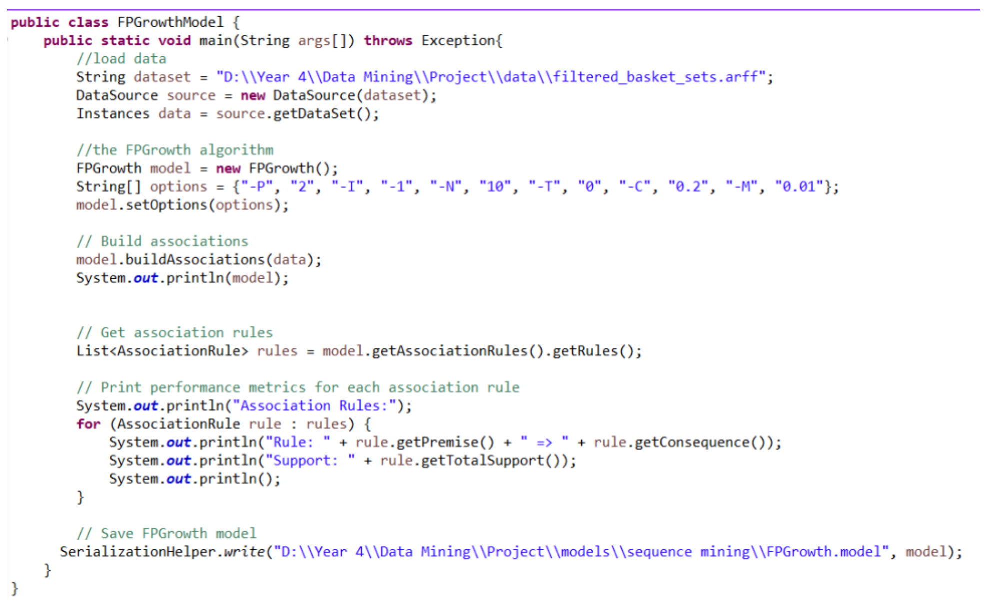
## 2.5. Implement sequence mining algorithms (Weka API)



```java
import weka.core.Instances;

public class AttributeFilter {
    public static void main(String args[]) throws Exception {
        // Load data
        DataSource source = new DataSource("D:\\Year 4\\Data Mining\\Project\\data\\basket_sets.arff");
        Instances dataset = source.getDataSet();

        // Apply the NumericToNominal filter only to specified attributes
        NumericToNominal numericToNominalFilter = new NumericToNominal();
        numericToNominalFilter.setAttributeIndices("first-last");
        numericToNominalFilter.setInputFormat(dataset);
        Instances convertedData = Filter.useFilter(dataset, numericToNominalFilter);

        // Remove unnecessary attributes
        String[] opts = new String[]{"-R", "1"};
        // Create a Remove object (this is the filter class)
        Remove remove = new Remove();
        // Set filter options
        remove.setOptions(opts);
        // Pass the data to apply filter
        remove.setInputFormat(convertedData);
        Instances filteredData = Filter.useFilter(convertedData, remove);

        // Now save the data to a new file
        ArffSaver saver = new ArffSaver();
        saver.setInstances(filteredData);
        saver.setFile(new File("D:\\Year 4\\Data Mining\\Project\\data\\filtered_basket_sets.arff"));
        saver.writeBatch();
    }
```



```java
import weka.associations.Apriori;

public class AprioriModel {
    public static void main(String args[]) throws Exception{
        //load data
        String dataset = "D:\\Year 4\\Data Mining\\Project\\data\\filtered_basket_sets.arff";
        DataSource source = new DataSource(dataset);
        Instances data = source.getDataSet();

        //the Apriori algorithm
        Apriori model = new Apriori();
        String[] options = {"-N", "10", "-T", "1", "-C", "0.9", "-D", "0.05","-M", "0.1", "-V",};
        model.setOptions(options);

        //build model
        model.buildAssociations(data);
        System.out.println(model);

//          Save Apriori model
        SerializationHelper.write("D:\\Year 4\\Data Mining\\Project\\models\\sequence mining\\Apriori.model", model);
    }
}
```

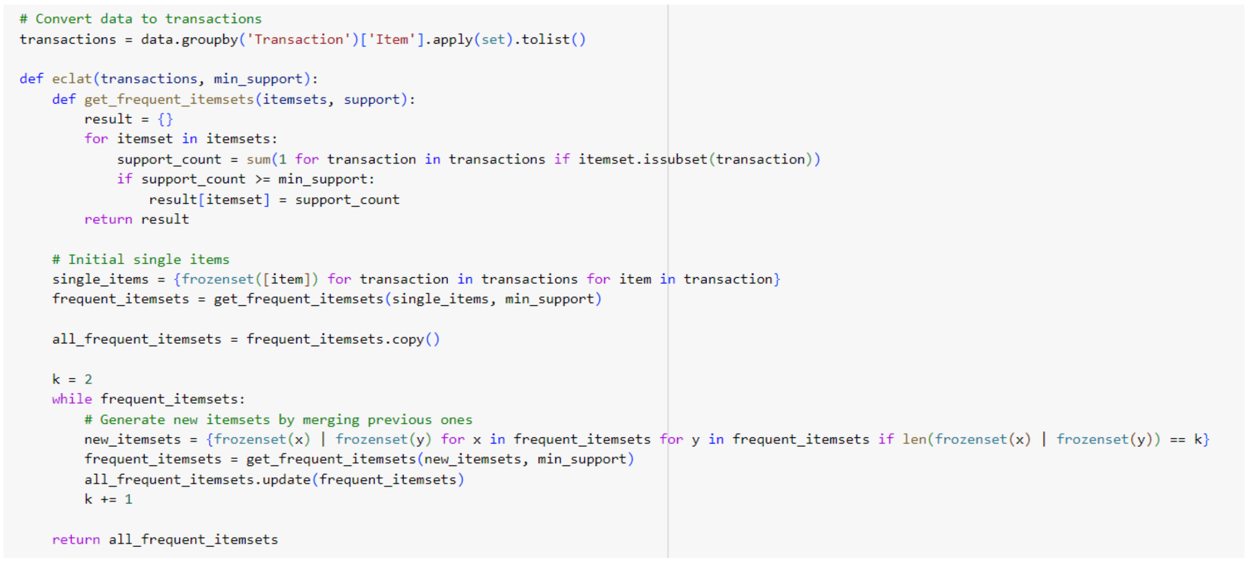


```java
public class FPGrowthModel {
    public static void main(String args[]) throws Exception{
        //load data
        String dataset = "D:\\Year 4\\Data Mining\\Project\\data\\filtered_basket_sets.arff";
        DataSource source = new DataSource(dataset);
        Instances data = source.getDataSet();

        //the FPGrowth algorithm
        FPGrowth model = new FPGrowth();
        String[] options = {"-P", "2", "-I", "-1", "-N", "10", "-T", "0", "-C", "0.2", "-M", "0.01"};
        model.setOptions(options);

        // Build associations
        model.buildAssociations(data);
        System.out.println(model);


        // Get association rules
        List<AssociationRule> rules = model.getAssociationRules().getRules();

        // Print performance metrics for each association rule
        System.out.println("Association Rules:");
        for (AssociationRule rule : rules) {
            System.out.println("Rule: " + rule.getPremise() + " => " + rule.getConsequence());
            System.out.println("Support: " + rule.getTotalSupport());
            System.out.println();
        }

        // Save FPGrowth model
      SerializationHelper.write("D:\\Year 4\\Data Mining\\Project\\models\\sequence mining\\FPGrowth.model", model);
    }
}
```

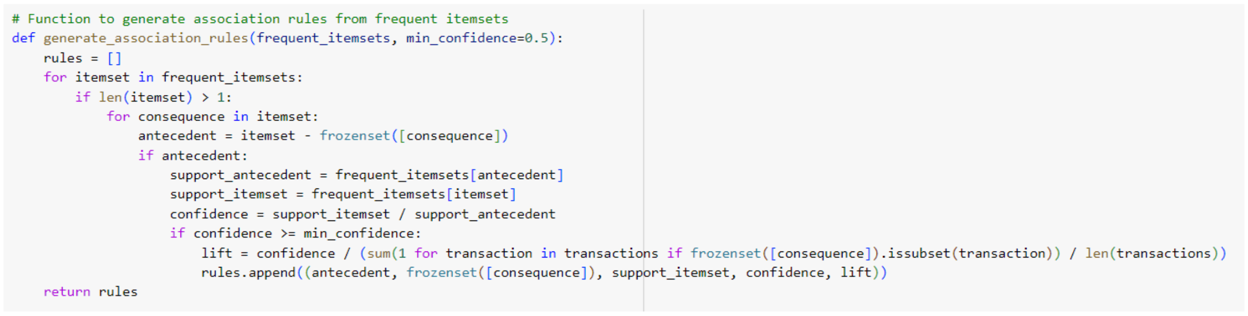
## 2.6. Implement another sequence mining algorithm (Python)

### ECLAT model:

- Stand for Equivalence Class Clustering and bottom-up Lattice Traversal
- A more efficient and scalable version of the Apriori algorithm
- Work in a vertical manner just like the Depth-First Search of a graph



```python
# Convert data to transactions
transactions = data.groupby('Transaction')['Item'].apply(set).tolist()

def eclat(transactions, min_support):
    def get_frequent_itemsets(itemsets, support):
        result = {}
        for itemset in itemsets:
            support_count = sum(1 for transaction in transactions if itemset.issubset(transaction))
            if support_count >= min_support:
                result[itemset] = support_count
        return result

    # Initial single items
    single_items = {frozenset([item]) for transaction in transactions for item in transaction}
    frequent_itemsets = get_frequent_itemsets(single_items, min_support)

    all_frequent_itemsets = frequent_itemsets.copy()

    k = 2
    while frequent_itemsets:
        # Generate new itemsets by merging previous ones
        new_itemsets = {frozenset(x) | frozenset(y) for x in frequent_itemsets for y in frequent_itemsets if len(frozenset(x) | frozenset(y)) == k}
        frequent_itemsets = get_frequent_itemsets(new_itemsets, min_support)
        all_frequent_itemsets.update(frequent_itemsets)
        k += 1

    return all_frequent_itemsets
```



```python
# Function to generate association rules from frequent itemsets
def generate_association_rules(frequent_itemsets, min_confidence=0.5):
    rules = []
    for itemset in frequent_itemsets:
        if len(itemset) > 1:
            for consequence in itemset:
                antecedent = itemset - frozenset([consequence])
                if antecedent:
                    support_antecedent = frequent_itemsets[antecedent]
                    support_itemset = frequent_itemsets[itemset]
                    confidence = support_itemset / support_antecedent
                    if confidence >= min_confidence:
                        lift = confidence / (sum(1 for transaction in transactions if frozenset([consequence]).issubset(transaction)) / len(transactions))
                        rules.append((antecedent, frozenset([consequence]), support_itemset, confidence, lift))
    return rules
```

### Procedure:

- Step 1: Scan the database to create a vertical representation
- Step 2: Generate initial candidate frequent itemsets = 1 by calculating the support
- Step 3: Filter out the items that do not meet the minimum support threshold.
- Step 4: Generate Recursive Frequent Itemset
- Step 5: Repeat process from step 1 to 4 and output a list of all frequent itemsets that meet the minimum support threshold.

## 2.7. Model Evaluation

Model evaluation stage is conducted using 10-fold cross-validation to ensure robust performance metrics. This method splits the dataset into ten parts, using nine for training and one for testing, rotating through all parts. The evaluation will focus on the accuracy, precision, recall, and F1-score of the classification/prediction models. Additionally, the run-time for building models and making predictions will be considered to assess their efficiency. The experimental results are expected to reveal insights into each model's strengths and weaknesses, with particular attention to how well they balance prediction accuracy with computational efficiency.

| **Models** | **Top 5 rules generated by model** | **Run-time** |
|---|---|---|
| Apriori | 1. tea=1 3719 ==> class=low 3708 conf:(1) < lift:(1)> lev:(0) [8] conv:(1.59)<br><br>2. class=low 18790 ==> tea=1 3708 conf:(0.2) < lift:(1)> lev:(0) [8] conv:(1)<br><br>3. bread=1 6627 ==> class=low 6599 conf:(1) < lift:(1)> lev:(0) [6] conv:(1.17)<br><br>4. class=low 18790 ==> bread=1 6599 conf:(0.35) < lift:(1)> lev:(0) [6] conv:(1)<br><br>5. pastry=1 2174 ==> class=low 2164 conf:(1) < lift:(1)> lev:(0) [1] conv:(1.02) | 1095ms |
| FPGrowth | 1. [toast=1]: 931 ==> [coffee=1]: 679 <conf:(0.73)> lift:(1.3) lev:(0.01) conv:(1.61)<br><br>2. [cake=1, sandwich=1]: 284 ==> [coffee=1]: 205 <conf:(0.72)> lift:(1.28) lev:(0) conv:(1.55)<br><br>3. [salad=1]: 351 ==> [coffee=1]: 242 <conf:(0.69)> lift:(1.23) lev:(0) conv:(1.4)<br><br>4. [cake=1, hot_chocolate=1]: 402 ==> [coffee=1]: 265 <conf:(0.66)> lift:(1.17) lev:(0) conv:(1.28)<br><br>5. [medialuna=1]: 1635 ==> [coffee=1]: 1044 <conf:(0.64)> lift:(1.14) lev:(0.01) conv:(1.21) | 740ms |
| ECLAT | 1. Rule: {'Toast'} -> {'Coffee'}, Support: 224, Confidence: 0.70, Lift: 1.47<br><br>2. Rule: {'Spanish Brunch'} -> {'Coffee'}, Support: 103, Confidence: 0.60, Lift: 1.25<br><br>3. Rule: {'Medialuna'} -> {'Coffee'}, Support: 333, Confidence: 0.57, Lift: 1.19<br><br>4. Rule: {'Pastry'} -> {'Coffee'}, Support: 450, Confidence: 0.55, Lift: 1.15<br><br>5. Rule: {'Alfajores'} -> {'Coffee'}, Support: 186, Confidence: 0.54, Lift: 1.13 | 411ms |

# 3. Conclusion

## 3.1. Limitation

Despite the thorough analysis, the study had some limitations:

- Data Constraints: The dataset size and quality could limit the generalizability of the findings. Additionally, external factors affecting wind speed might not be fully captured in the dataset.
- Algorithm Complexity: Some algorithms may have high computational complexity, affecting performance, especially on larger datasets.
- Tool Limitations: Both Weka and Python have their own limitations in terms of capabilities and ease of use, which might impact the implementation and results.

## 3.2. Future Plans

Future work can focus on:

- Enhanced Algorithms: Developing more sophisticated and efficient sequence mining algorithms to handle larger and more complex datasets.
- Real-time Sequence Mining: Implementing algorithms that can handle real-time data streams, which is crucial for timely wind speed predictions.
- Broader Applications: Applying sequence mining techniques to a wider range of domains and datasets to validate their effectiveness and versatility. Additionally, incorporating other meteorological datasets to improve the predictive power.

## 4. Reference Lists

[1] https://www.kaggle.com/datasets/mittalvasu95/the-bread-basket

[2] https://www.kaggle.com/datasets/jmmvutu/summer-products-and-sales-in-ecommerce-wish

[3] https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/

[4] https://www.youtube.com/playlist?list=PLea0WJq13cnBVfsPVNyRAus2NK-KhCuzJ

[5] Jiawei Han, Jian Pei, Hanghang Tong, Data Mining: Concepts and Techniques, 4th Edition, Morgan Kaufmann, 2022. Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman, Mining of Massive Datasets, Cambridge University Press, 3rd Edition, 2020.