Sales Clothes Prediction

Market Basket Analysis

Presented by Group D

Course name: Data Mining
Professors: Dr. Nguyen Thi Thanh Sang, MSc. Nguyen Quang Phu

Members:
Dang Quoc Anh Duy - ITDSIU20015
Doan Huu Nguyen - ITITIU20260
Nguyen Hoang Minh Tuan - ITITIU20343

CONTENTS

01 PREDICTION

01 PREDICTION

INTRODUCTION

- With the exponential growth of online shopping, the vast amounts of data related to products
- Businesses are increasingly reliant on accurate sales predictions to make decisions

→ **The need of robust prediction model**

Objective:

- Perform sales prediction on E-Commerce dataset
- Figure out key features affect the prediction model
- Implement using Weka API and Python to find the best classifier

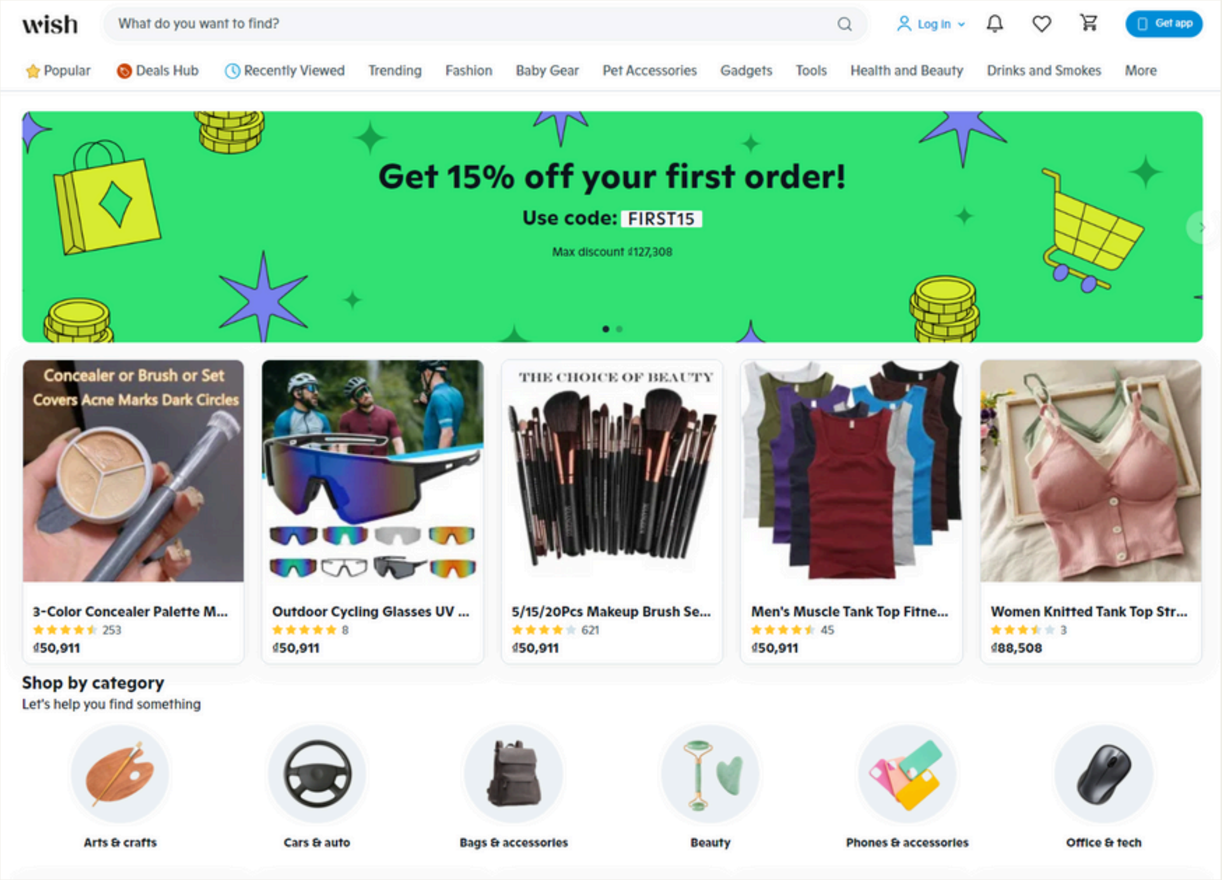
01 PREDICTION

INTRODUCTION

Dataset info:

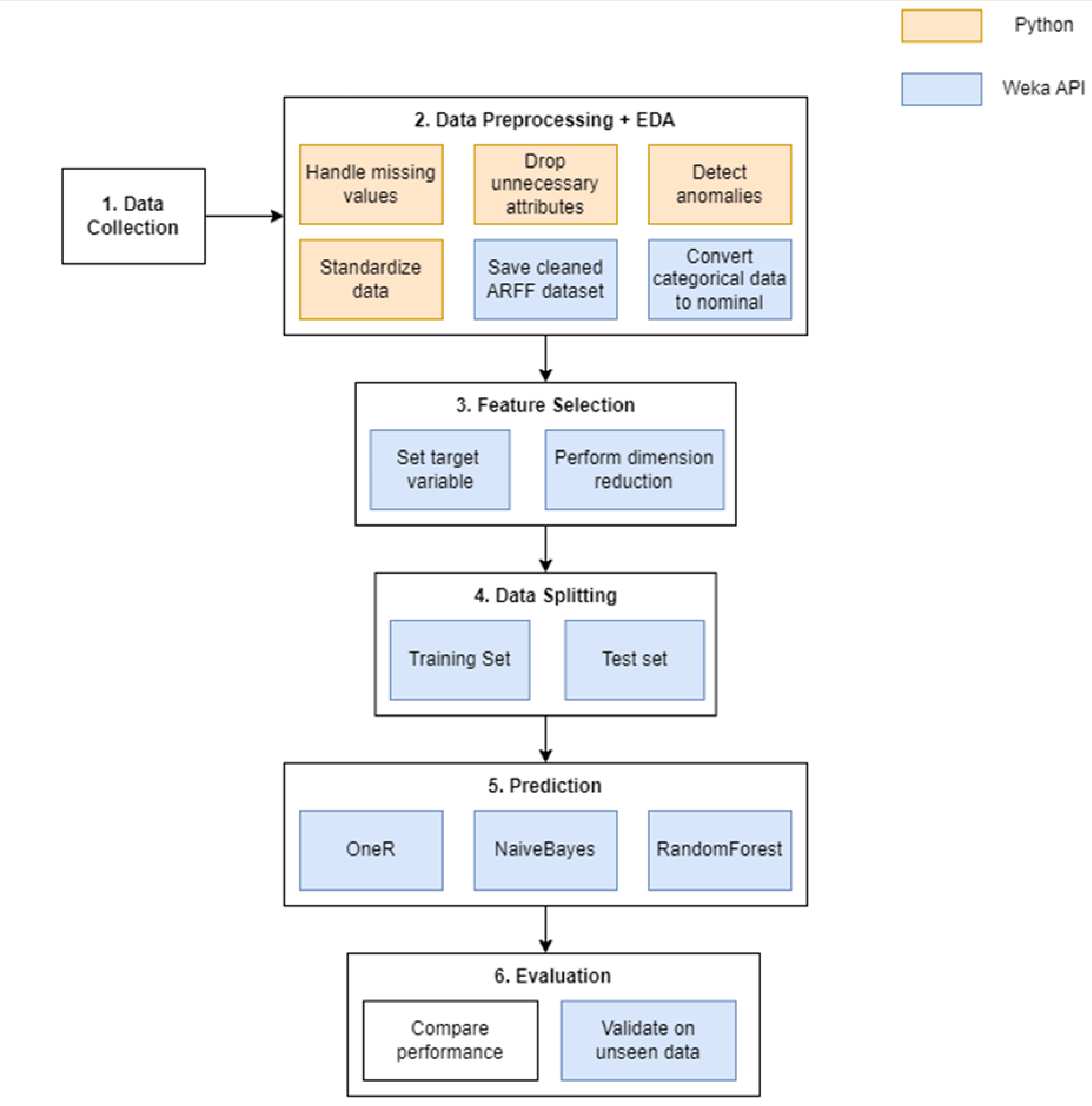
- Summer Sales Clothes in E-Commerce - Kaggle
- Was scrapped from the Wish platform with 1573 observations and 43 variables
- Contain product listings, ratings and sales performance

	title	title_orig	price	retail_price	currency_buyer	units_sold	uses_ad_boosts	rating	rating_count	rating_five_count
0	2020 Summer Vintage Flamingo Print Pajamas Se...	2020 Summer Vintage Flamingo Print Pajamas Se...	16.00	14	EUR	100	0	3.76	54	26.0
1	SSHOUSE Summer Casual Sleeveless Soirée Party ...	Women's Casual Summer Sleeveless Sexy Mini Dress	8.00	22	EUR	20000	1	3.45	6135	2269.0
2	2020 Nouvelle Arrivée Femmes Printemps et Été ...	2020 New Arrival Women Spring and Summer Beach...	8.00	43	EUR	100	0	3.57	14	5.0



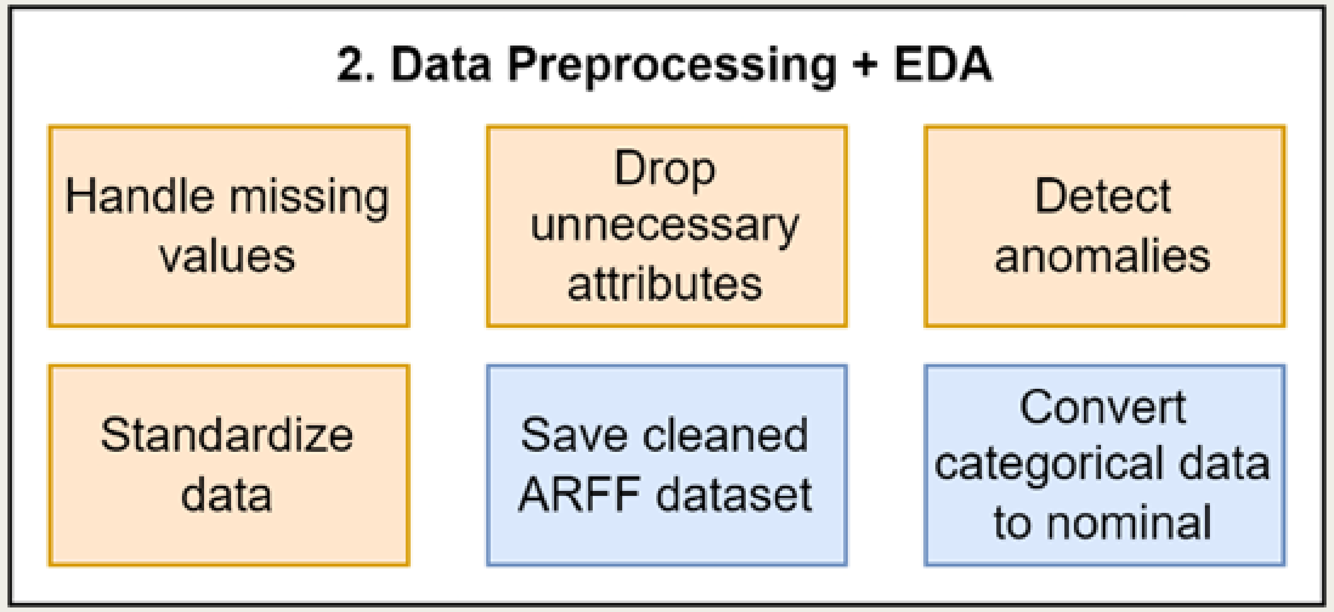
01 PREDICTION

METHODOLOGY



01 PREDICTION

DATA PREPROCESSING




```
[ ] # Fill number of ratings by 0 if the value is None
    data['rating_five_count'] = data['rating_five_count'].replace(np.nan, 0)
    data['rating_four_count'] = data['rating_four_count'].replace(np.nan, 0)
    data['rating_three_count'] = data['rating_three_count'].replace(np.nan, 0)
    data['rating_two_count'] = data['rating_two_count'].replace(np.nan, 0)
    data['rating_one_count'] = data['rating_one_count'].replace(np.nan, 0)
```

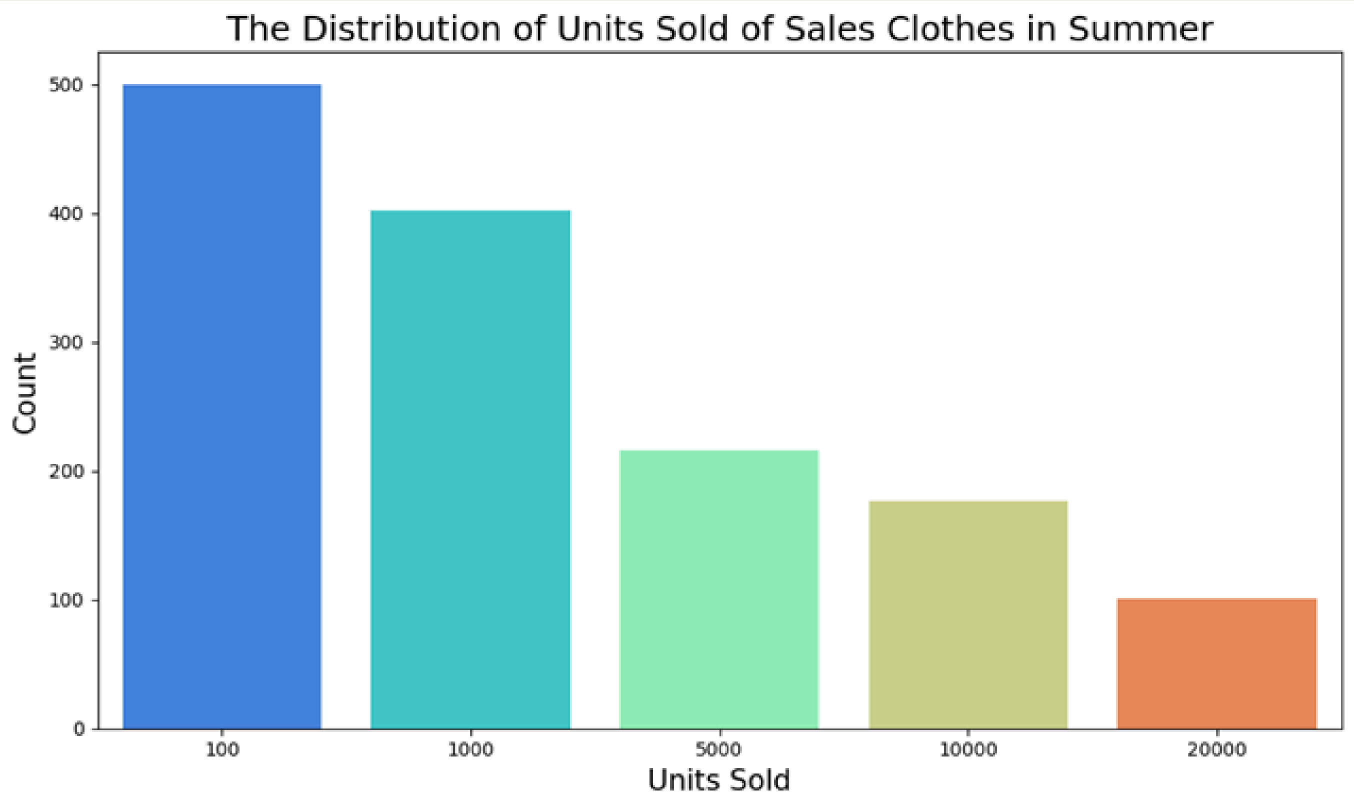
```
def standardize_product_size(name):
    valid_sizes = ['S', 'XS', 'XXS', 'XXXS', 'M', 'L', 'XL', 'XXL', 'XXXL', 'XXXXL', 'XXXXXL']
    return name if name in valid_sizes else 'OTHER'
```

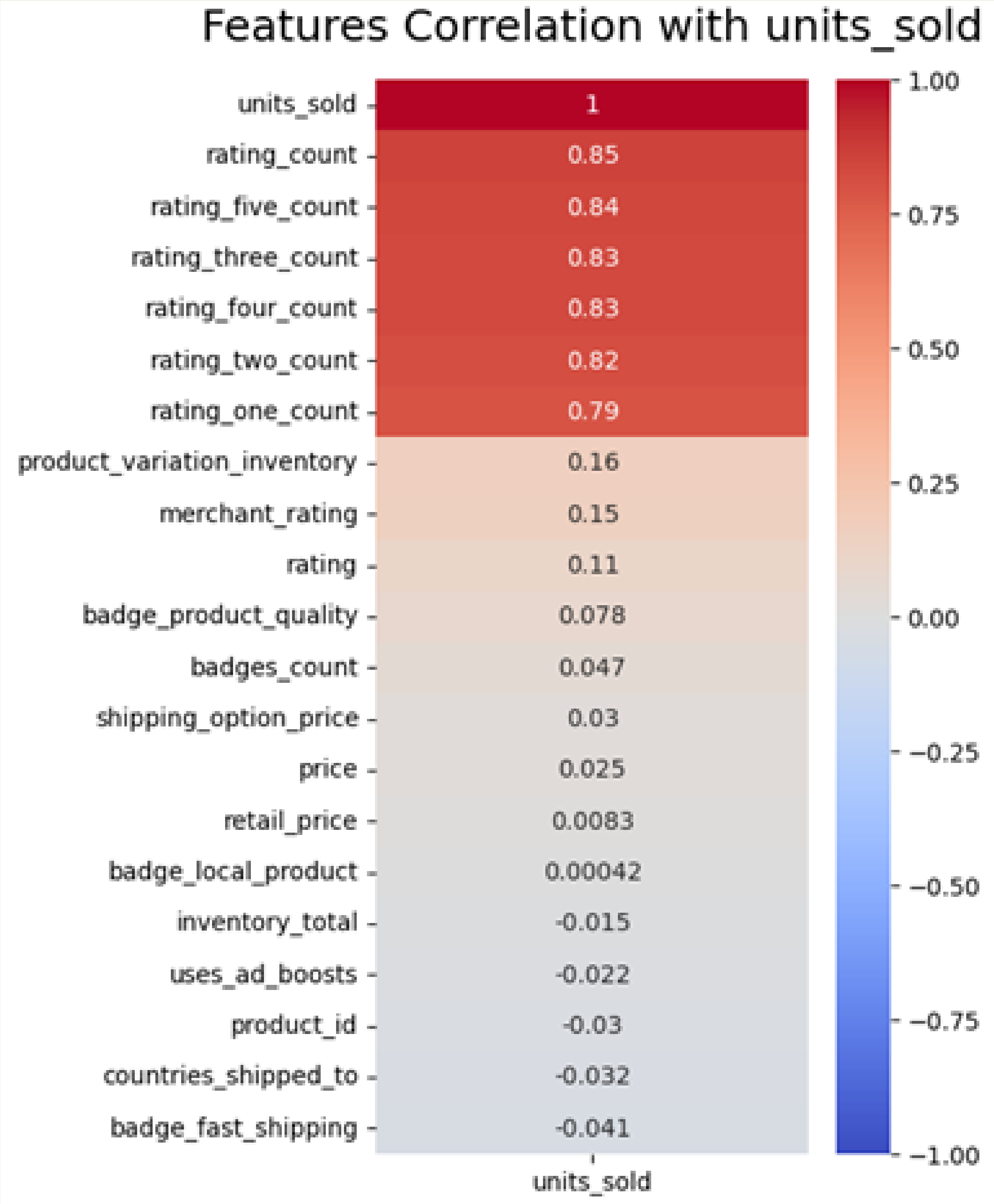
```
data['product_variation_size_id'] = data['product_variation_size_id'].replace(np.nan,'OTHER')
data['product_variation_size_id'] = data['product_variation_size_id'].apply(standardize_product_size)
```

```
# Encode the categorical attributes

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data['product_color'] = le.fit_transform(data['product_color'])
data['product_variation_size_id'] = le.fit_transform(data['product_variation_size_id'])
data['origin_country'] = le.fit_transform(data['origin_country'])
data['units_sold'] = le.fit_transform(data['units_sold'])
```

01 PREDICTION

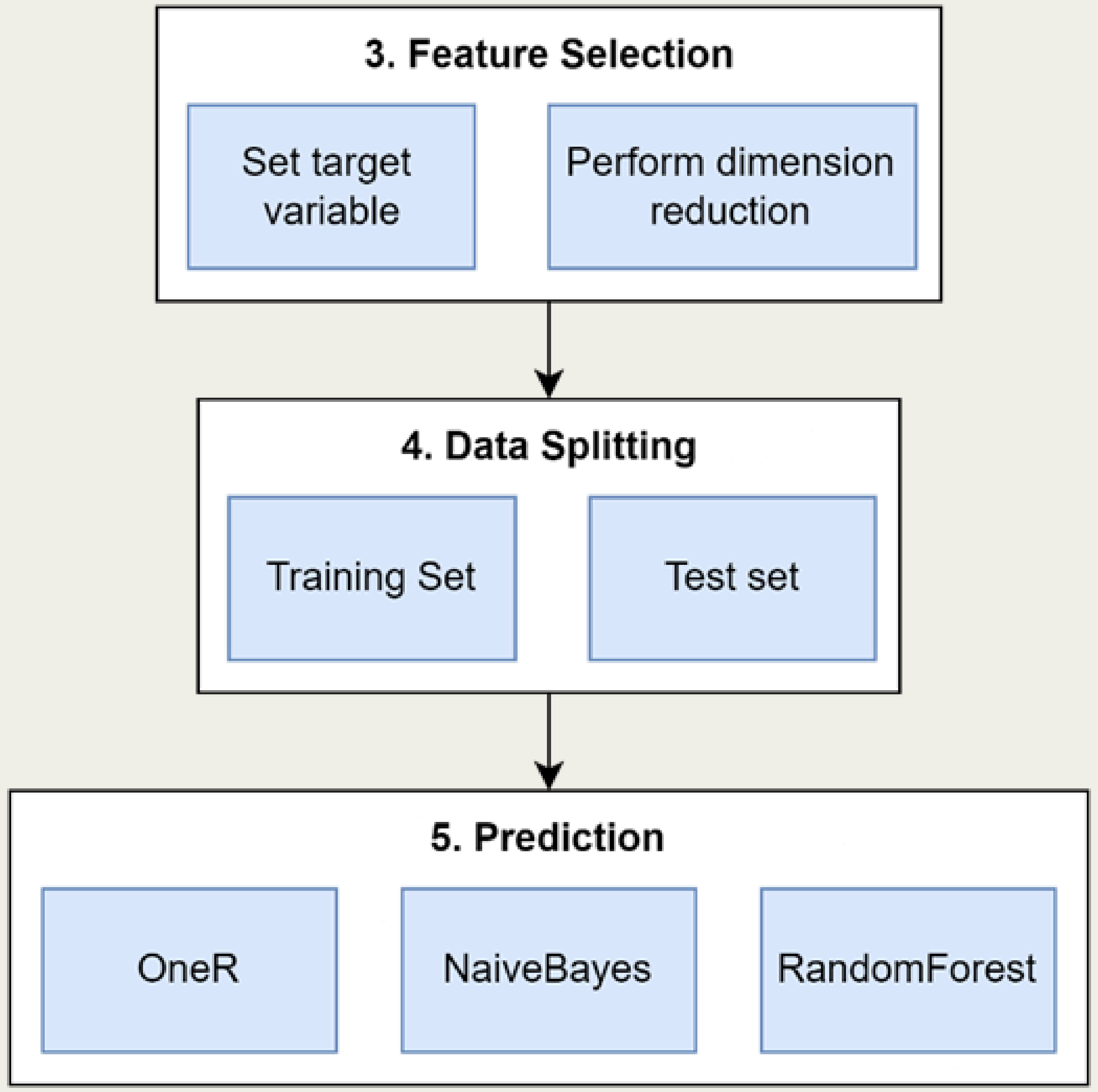
EXPLORATORY DATA ANALYSIS





01 PREDICTION

MODEL IMPLEMENTATION




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

        // Set the index of the target attribute
        int targetAttributeIndex = 2;
        filteredData.setClassIndex(targetAttributeIndex);

        AttributeSelection filter = new AttributeSelection();

        //create evaluator and search algorithm objects
        CfsSubsetEval eval = new CfsSubsetEval();
        GreedyStepwise search = new GreedyStepwise();
        search.setSearchBackwards(true);
        filter.setEvaluator(eval);
        filter.setSearch(search);
        filter.setInputFormat(filteredData);

        Instances newData = Filter.useFilter(filteredData, filter);

        ArffSaver saver = new ArffSaver();
        saver.setInstances(newData);
        saver.setFile(new File("D:\\Year 4\\Data Mining\\Project\\data\\dimension_reduction_sales_clothes.arff"));
        saver.writeBatch();
    }
}
```

01 PREDICTION

MODEL IMPLEMENTATION

```
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);
```

```
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: OneR
        Classifier oneR = (Classifier) SerializationHelper.read("OneR.model");

        // Perform predictions and print actual class and OneR predicted class
        System.out.println("===================");
        System.out.println("Actual Class, OneR 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 predOneR = oneR.classifyInstance(newInst);

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

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

01 PREDICTION

MODEL IMPLEMENTATION

Console

```
<terminated> Classification [Java Application] C:\Program Files\Java\jdk-1.8\bin\javaw.exe  (May 22, 2024, 10:04:49 AM – 10:04:02 AM) [pid: 5112]
=== NaiveBayes Evaluation ===

Correctly Classified Instances         875               78.405  %
Incorrectly Classified Instances       241               21.595  %
Kappa statistic                          0.7079
Mean absolute error                      0.0864
Root mean squared error                  0.281
Relative absolute error                 28.9982 %
Root relative squared error             72.8088 %
Total Number of Instances             1116

=== Confusion Matrix ===

   a   b   c   d   e   <-- classified as
 380  17   0   0   0 |   a = 100
  53 235  34   0   0 |   b = 1000
   1  42 108  21   0 |   c = 5000
   0   4  35  98   4 |   d = 10000
   0   0   3  27  54 |   e = 20000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.957    0.075    0.876      0.957    0.915      0.866    0.981     0.958     100
                 0.730    0.079    0.789      0.730    0.758      0.666    0.932     0.848     1000
                 0.628    0.076    0.600      0.628    0.614      0.542    0.917     0.617     5000
                 0.695    0.049    0.671      0.695    0.683      0.636    0.951     0.664     10000
                 0.643    0.004    0.931      0.643    0.761      0.759    0.982     0.867     20000
Weighted Avg.    0.784    0.068    0.786      0.784    0.782      0.721    0.953     0.830

Training time: 107 milliseconds
```

01 PREDICTION

ANOTHER APPROACH

SVM - Support Vector Machine Apply for both PCA and entire set to compare performance

```
# 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
```

Parameter Selection

```
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
svm_model.fit(X_train_scaled, y_train)

# Make predictions on the testing set
y_pred = svm_model.predict(X_test_scaled)

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

# Evaluate the model
print("Accuracy on the testing set:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Model Training

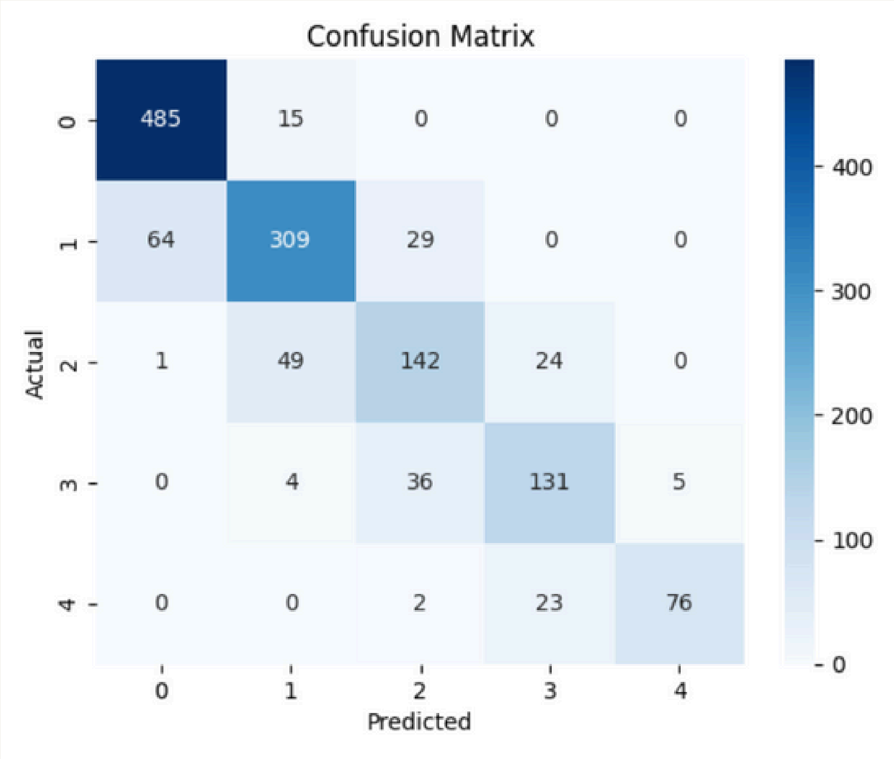
01 PREDICTION

ANOTHER APPROACH

SVM with PCA

Classification Report:

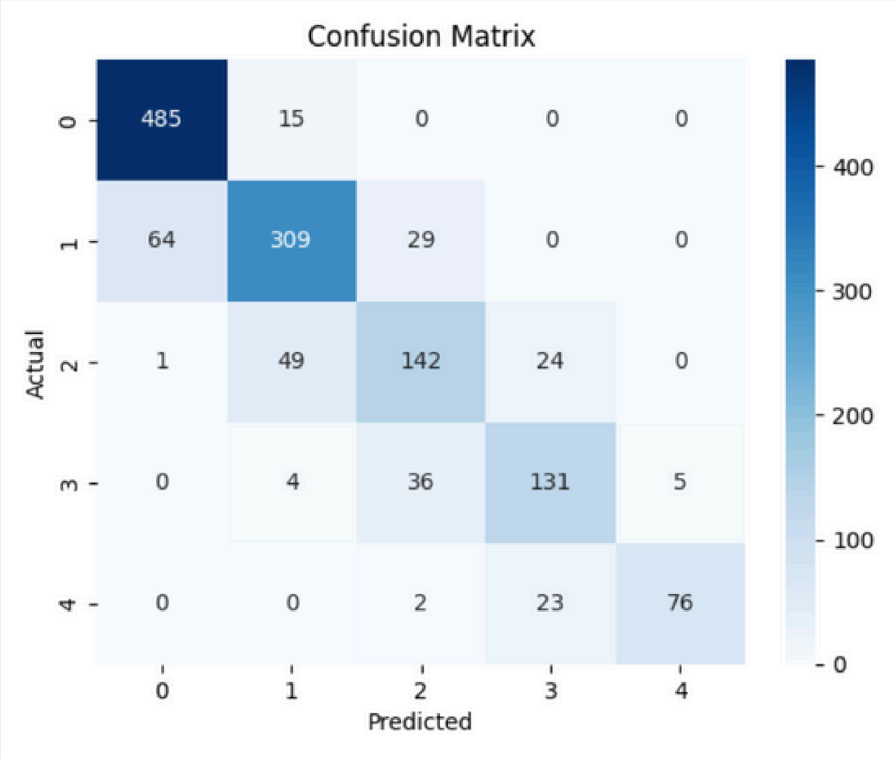
	precision	recall	f1-score	support
0	0.84	0.95	0.89	500
1	0.77	0.69	0.73	402
2	0.63	0.61	0.62	216
3	0.66	0.69	0.68	176
4	0.90	0.62	0.74	101
accuracy			0.77	1395
macro avg	0.76	0.71	0.73	1395
weighted avg	0.77	0.77	0.76	1395



SVM without PCA

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.97	0.92	500
1	0.82	0.77	0.79	402
2	0.68	0.66	0.67	216
3	0.74	0.74	0.74	176
4	0.94	0.75	0.84	101
accuracy			0.82	1395
macro avg	0.81	0.78	0.79	1395
weighted avg	0.82	0.82	0.82	1395



01 PREDICTION

MODEL EVALUATION

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

Remarks:

- Accuracy: SVM
- Run-time: NaiveBayes
- Best Model: SVM

→

1. Performance of Weka prediction model is quite good for the small dataset
2. Not much improvement in Python compared to the Weka models

02 SEQUENCE MINING

02 SEQUENCE MINING

INTRODUCTION

- Sequence mining for recommendation systems addresses the demands of businesses to provide personalized experiences and marketing
- Help to optimize revenue generation by driving sales through targeted suggestions

Objective:

- Analyze the sequential data
- Implement on various sequence mining models to find the best one
- Evaluate on the generated rules

02 SEQUENCE MINING

INTRODUCTION

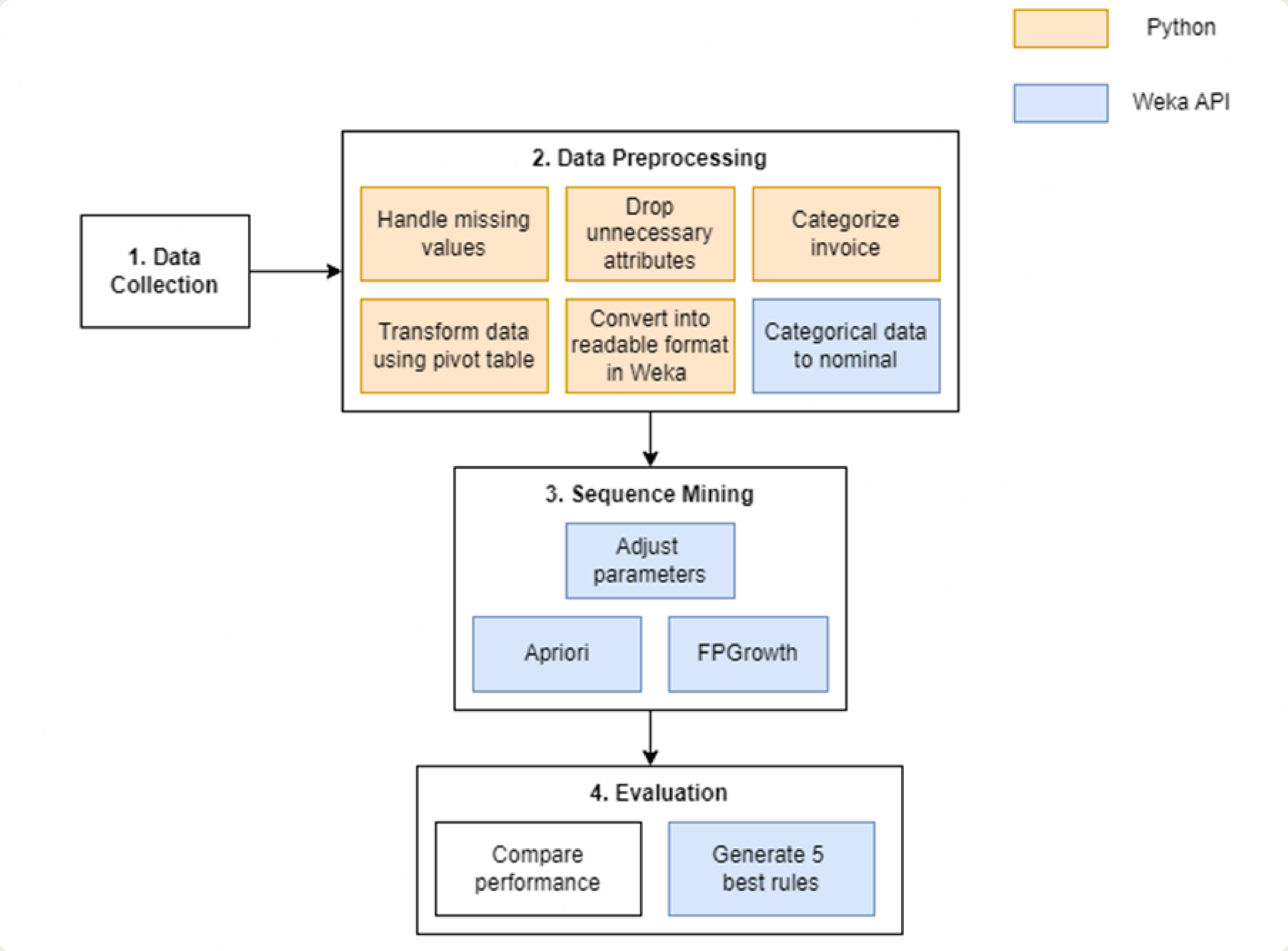
Dataset info:

- The Bread Basket - Kaggle
- Belong to a bakery located in Edinburgh
- Has 20507 entries, over 9000 transactions, and 4 columns

	Transaction	Item	date_time	period_day	weekday_weekend
0	1	Bread	30-10-2016 09:58	morning	weekend
1	2	Scandinavian	30-10-2016 10:05	morning	weekend
2	2	Scandinavian	30-10-2016 10:05	morning	weekend
3	3	Hot chocolate	30-10-2016 10:07	morning	weekend
4	3	Jam	30-10-2016 10:07	morning	weekend

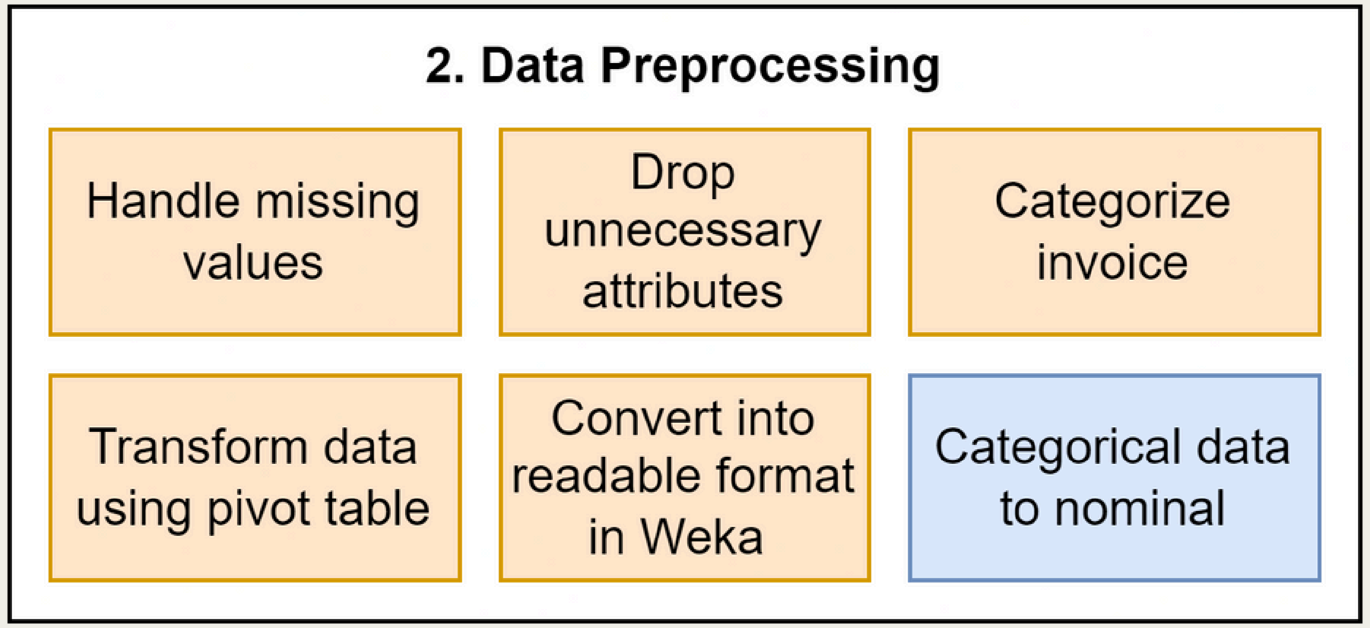
02 SEQUENCE MINING

METHODOLOGY



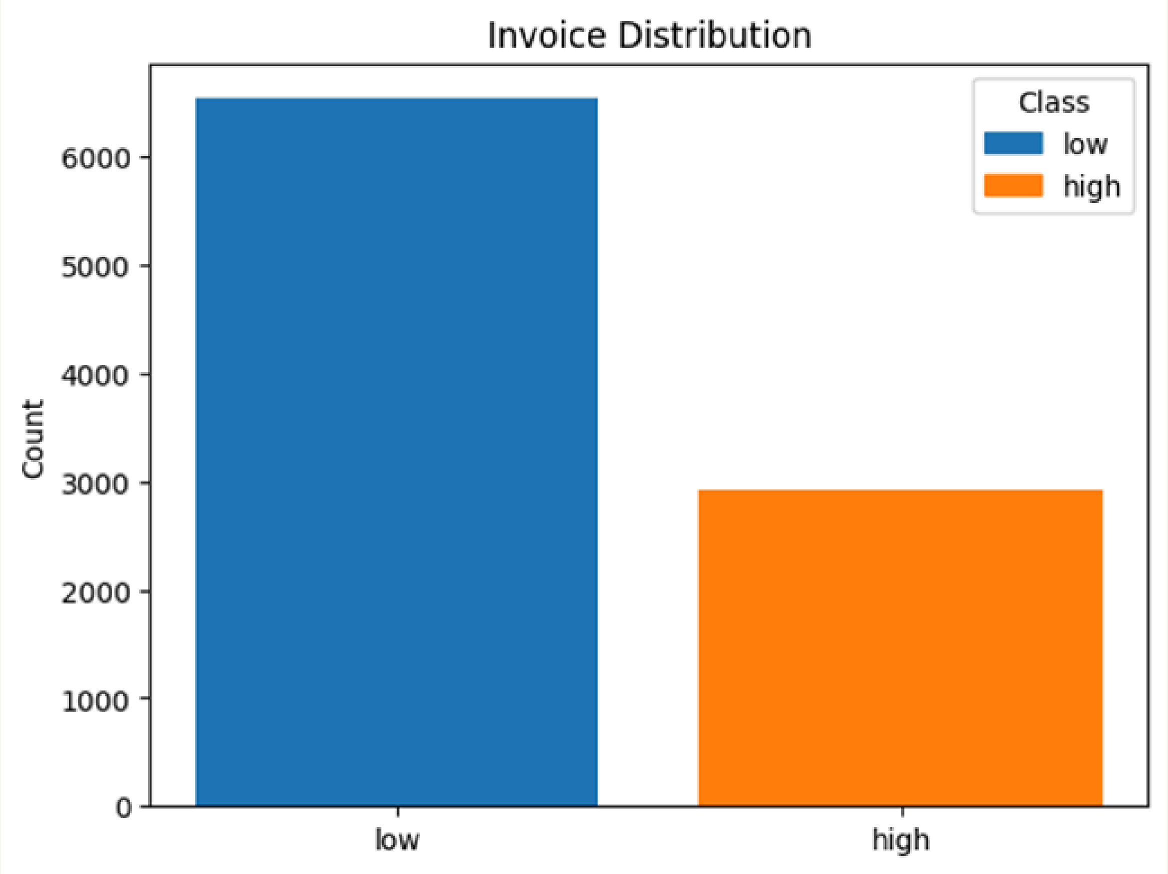
02 SEQUENCE MINING

DATA PREPROCESSING



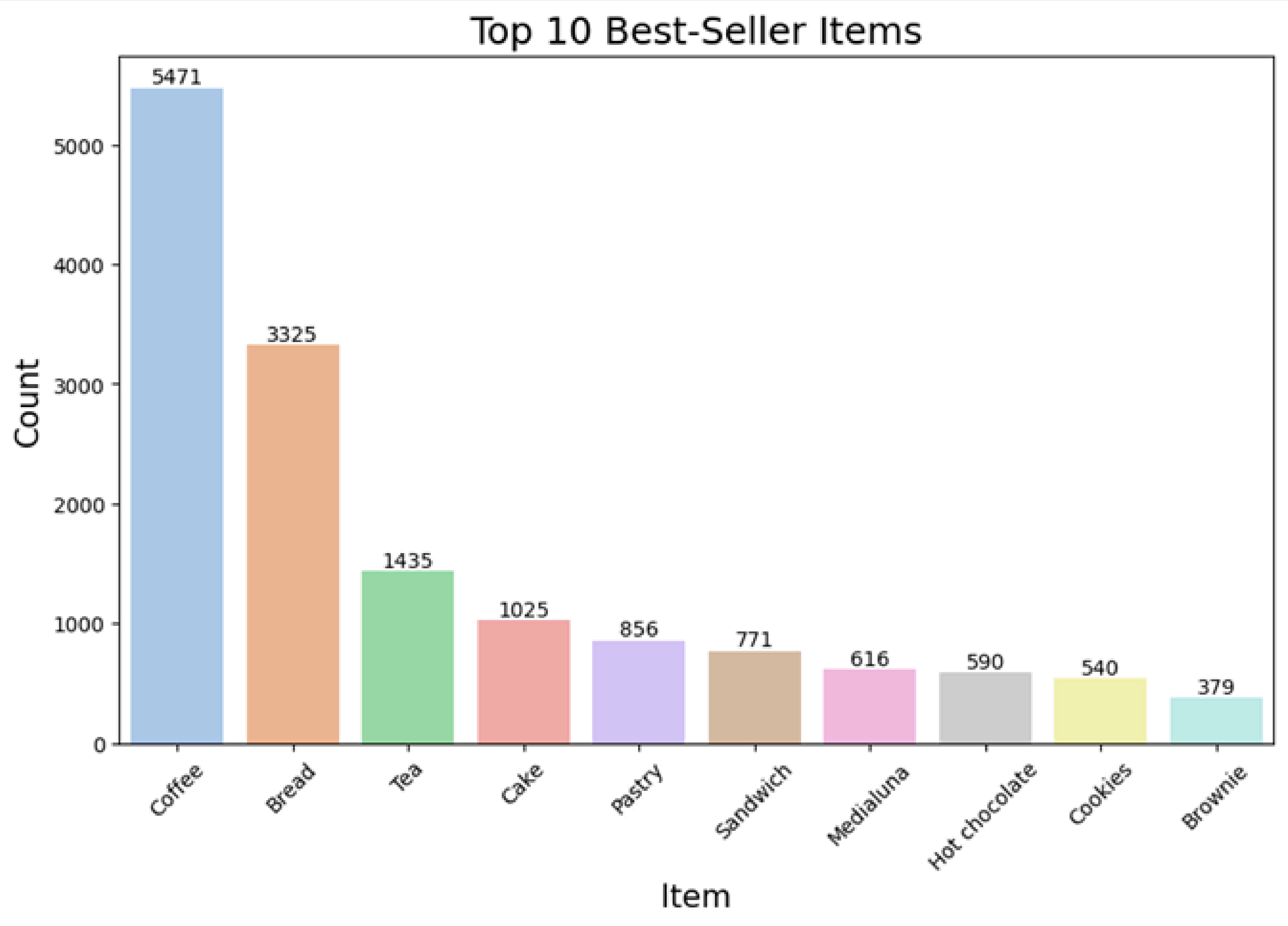
	Transaction	Item
0	1	Bread
1	2	Scandinavian
2	2	Scandinavian
3	3	Hot chocolate
4	3	Jam

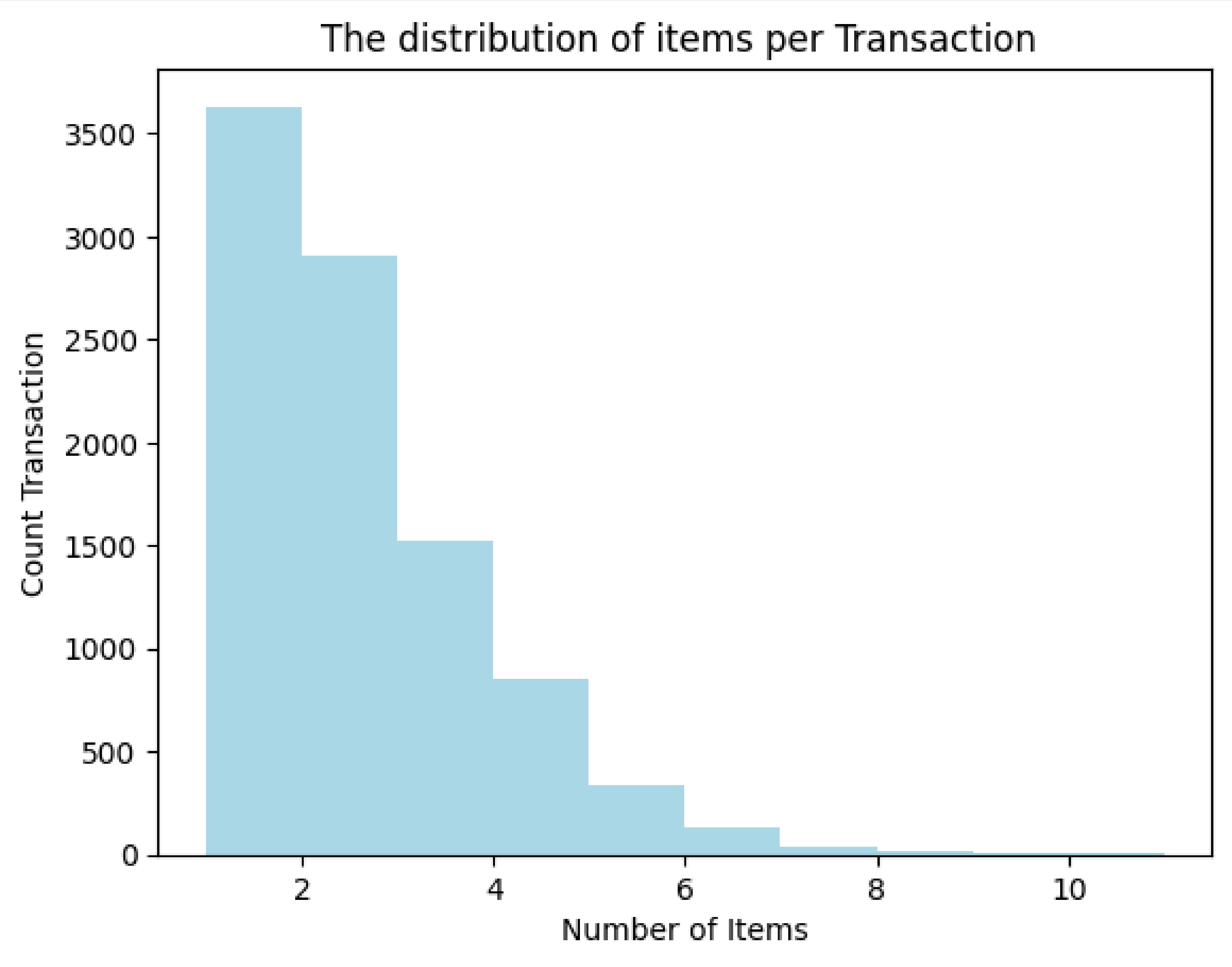
Item	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Bacon	Baguette	Bakewell
Transaction								
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN



02 SEQUENCE MINING

EXPLORATORY DATA ANALYSIS





02 SEQUENCE MINING

MODEL IMPLEMENTATION

```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();
    }
```

02 SEQUENCE MINING

MODEL IMPLEMENTATION

```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);
    }
}
```

02 SEQUENCE MINING

MODEL IMPLEMENTATION

```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);
    }
}
```

02 SEQUENCE MINING

MODEL IMPLEMENTATION

```
Console
<terminated> FPGrowthModel [Java Application] C:\Program Files\Java\jdk-1.8\bin\javaw.exe  (May 21, 2024, 11:38:47 PM – 11:38:48 PM) [pid: 14632]
FPGrowth found 78 rules (displaying top 10)

 1. [toast=1]: 931 ==> [coffee=1]: 679   <conf:(0.73)> lift:(1.3) lev:(0.01) conv:(1.61)
 2. [cake=1, sandwich=1]: 284 ==> [coffee=1]: 205   <conf:(0.72)> lift:(1.28) lev:(0) conv:(1.55)
 3. [salad=1]: 351 ==> [coffee=1]: 242   <conf:(0.69)> lift:(1.23) lev:(0) conv:(1.4)
 4. [cake=1, hot_chocolate=1]: 402 ==> [coffee=1]: 265   <conf:(0.66)> lift:(1.17) lev:(0) conv:(1.28)
 5. [medialuna=1]: 1635 ==> [coffee=1]: 1044   <conf:(0.64)> lift:(1.14) lev:(0.01) conv:(1.21)
 6. [spanish_brunch=1]: 610 ==> [coffee=1]: 389   <conf:(0.64)> lift:(1.13) lev:(0) conv:(1.2)
 7. [tiffin=1]: 466 ==> [coffee=1]: 296   <conf:(0.64)> lift:(1.13) lev:(0) conv:(1.19)
 8. [hearty_&_seasonal=1]: 307 ==> [coffee=1]: 192   <conf:(0.63)> lift:(1.11) lev:(0) conv:(1.16)
 9. [pastry=1]: 2174 ==> [coffee=1]: 1348   <conf:(0.62)> lift:(1.1) lev:(0.01) conv:(1.15)
10. [sandwich=1]: 2017 ==> [coffee=1]: 1241   <conf:(0.62)> lift:(1.09) lev:(0.01) conv:(1.14)

Association Rules:
Rule: [toast=1] => [coffee=1]
Support: 679

Rule: [cake=1, sandwich=1] => [coffee=1]
Support: 205

Rule: [salad=1] => [coffee=1]
Support: 242

Rule: [cake=1, hot_chocolate=1] => [coffee=1]
Support: 265
```

ANOTHER APPROACH

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

02 SEQUENCE MINING

ANOTHER APPROACH

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.

02 SEQUENCE MINING

ANOTHER APPROACH

```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
```

02 SEQUENCE MINING

MODEL EVALUATION

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

CONCLUSION

01

Features

- **Comprehensive Data Processing**: Successfully handled data collection, preprocessing, and feature engineering to prepare data for analysis
- **Model Implementation:** Implemented and compared prediction algorithms using Java WEKA and Python libraries.
- **Performance Evaluation :** Conducted rigorous model evaluation using metrics like MAE, MSE, and RMSE to ensure accuracy.

02

Limitation

- **Scalability and Complexity** :Faced challenges in handling large datasets and computational complexity.
- **Dependency on Data Quality:** Model performance was heavily dependent on the quality and completeness of the data.
- **Tool Limitations (Java WEKA) :** Experienced limitations in flexibility and community support with Java WEKA compared to Python.

03

Future Plans

- **Adopt Advanced Algorithms** : Plan to explore more advanced algorithms like GBM and neural networks for better accuracy.
- **Shift to Python Ecosystem:** Intend to transition to Python libraries for greater flexibility and ease of use in model implementation.

REFERENCES

- https://www.kaggle.com/datasets/mittalvasu95/the-bread-basket
- https://www.kaggle.com/datasets/jmmvutu/summer-products-and-sales-in-ecommerce-wish
- https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/
- https://www.youtube.com/playlist?list=PLea0WJq13cnBVfsPVNyRAus2NK-KhCuzJ
- 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.

Thank you!

FOR YOUR ATTENTION