Introduction

Project Overview

providing businesses with invaluable insights into consumer behavior. Leveraging this data, businesses can refine their sales strategies to stay competitive in the market. For our project, we chose to analyze sales the realm of a Scottish bakery, utilizing association analysis techniques such as Support, Confidence, Lift metrics, and FP-Growth algorithm to uncover patterns among products in a dataset belonging to a bakery in Edinburgh, Scotland, spanning from December 2003 to January 2011."

Dataset Overview

The dataset our group chose contains transaction details of customers who ordered various items online from the bakery. Acquired via Kaggle.com, this data set serves as the foundation for our analysis and gives us information such as the number of products purchased, which products were purchased, and at what days and times they were purchased. With over 20,000 rows and 9,000 transactions, the dataset offers a variety of customer purchasing preferences and behaviors, without being overly-complex in how descriptive the specific sales were, when they are recorded.

Motivation Overview

In the fast-paced bakery industry, understanding product associations is paramount for optimizing sales strategies, since pastry-related foods can expire so rapidly, with some items within our dataset needing to sell the day that it's made. This makes concepts such as shrinkage, the term used within the grocery industry representing how much percent of your product is expiring out, rather than selling out, extra important within a bakery-setting, and thus exciting to choose to focus on. Through discerning which products tend to sell together, businesses can restructure production schedules, selling techniques, minimize their shrink, and enhance customer satisfaction. For instance, the urgency to sell perishable items like donuts and cakes requires thoughtful planning of production and sales, which can be driven by datamining sales datasets such as our own.

Basic Objectives

In our project, we will be addressing two fundamental questions through data mining techniques:

1. Which products exhibit strong associations in sales?

2. Which products should be grouped in combinations and sold together to maximize revenue and minimize waste?

Challenges/Technical Approach Overview

Navigating through our chosen data set presents several challenges which we faced, including a lack of detailed product categorization and inconsistencies in recording similar items. To mitigate these challenges, we employed data preprocessing techniques using Python's Pandas library. We were required to use preprocessing techniques in extracting attributes such as date, time, and day of the week into our program, and created visuals that help us break down the results of our dataset analysis easier, with attributes such as date, time, and day labeled in the charts. Our approach involves using a variety of methods to dig into our data to find relevant information. The data is preprocessed, and then we subsequently employ association analysis techniques such as Support, Confidence, Lift metrics, and an FP-Growth algorithm to uncover associations and patterns among items sold from our Scottish bakery. In clarifying the connections between bakery products and understanding how consumers behave when choosing to purchase them, we can optimize the sales strategies and minimize waste in the landscape of a bakery. Moreover, this same approach could be expanded to other industries of modern commerce as well with few adjustments needed, showing that our approach of Bakery Sales Analysis can be used for other commerce industries with similar sales techniques, even if those industries and businesses are not necessarily bakery themselves.

Preprocessing

Our retail dataset had the following columns:

- Date/Time for a date and time stamp of the transactions made.
- Daypart represents if a transaction is made in the morning, afternoon, evening, or night.
- DayType classifying whether a transaction was made on a weekend or weekday and
- Transaction Numbers (ID) as the unique identifiers for every transaction.
- Item

Given the high number of results we intended to produce with this dataset, it made little sense to preprocess the data the same way for each step of the technical approach. A quick audit of the dataset revealed that it was clean to begin with, requiring little preprocessing for calculating Support, Confidence, and Lift.

In order to run the FP-Growth algorithm, we had to create a dictionary of transactions, each containing all items in that transaction. Given that the original dataset contained purchased items correlated simply with a transaction ID, we had to iterate over the dataset and add each transaction ID to a dictionary, adding any items found in that transaction ID to the dictionary as well. After this was done, we had the dictionary required to run the FP-Growth algorithm.

In terms of the performance of the Association Rule, we evaluated it using a couple metrics such as accuracy, precision, recall and F-1 score. These were calculated based on the predicted top 10 itemset and the actual top 10 itemset for Confidence, Lift, and Support. In the next few paragraphs, we'll take a look at the results and provide an interpretation of the predictions as well. We will provide more information on the findings, interesting trends from the data and analyze the algorithm performance. Some recommendations on future work and ways to improve results from similar work will also be briefly discussed.

Support and Confidence

Support is a key metric in identifying which items are most popular within a given dataset. The motivation behind using such a metric was to learn what items should be stocked to what amounts on any given day. Further, finding the support of each item during a given season, day of the week, or part of a day gave further insight into what items should be stocked to what amounts in any specific case. Calculating support was simple; we counted the number of entries matching the criteria we were looking for. We once again used python to calculate both the support and confidence.

Confidence is another key metric in identifying which items in a dataset may be bought together. The motivation behind using such a metric is to find what items can be offered grouped as packages to increase sales. Calculating confidence was similarly simple; we created a dataset consisting of all items that fit the antecedent, then counted the proportion of rows which contained the consequent item.

FP-Growth

We then used FP-Growth to find rule associations in our dataset. In order to run the FP-Growth algorithm, we used the MLX tend python library. However, for the technical approach, a summary of the algorithm is as follows. FP-Growth involved two main parts: the construction of the FP-tree and the mining of frequent itemset. Constructing an FP-tree involves building an FP-tree consisting of nodes, each of which represents an item. The paths from the root to the leaf nodes represent transactions. The primary goal of this data structure is to compactly store the items in the dataset and their relationships. The algorithm scans the dataset only once to construct the FP-tree. Transaction reduction is then performed by merging transactions with identical itemset, thereby reducing the memory required for the FP-tree. After the FP-Tree has been created, it is mined for association rules. The tree is then recursively traversed, finding the most associated items using conditional pattern bases. The motivation behind using FP-Growth, and, more broadly the use of association rules, is to find items that are highly correlated to offer them as sales together. This is similar to the confidence metric; however, this approach is far more targeted.

Lift

Lift measures the predictive power of a rule association generated by FP-Growth by comparing the observed support of the combined items in the rule with the support expected if those items were independent of each other. For our purposes, lift helps in determining the strength and direction of association between items. While rule associations between the most popular items and every other item will be prominently shown as results of the FP-Growth algorithm, these results are not particularly interesting. Since we already know that coffee is the most popular item at a cafe, it is important to find items that have strong correlations, even if they are less popular themselves, in order to find the best targeted advertisements and deals.

Naives Bayes

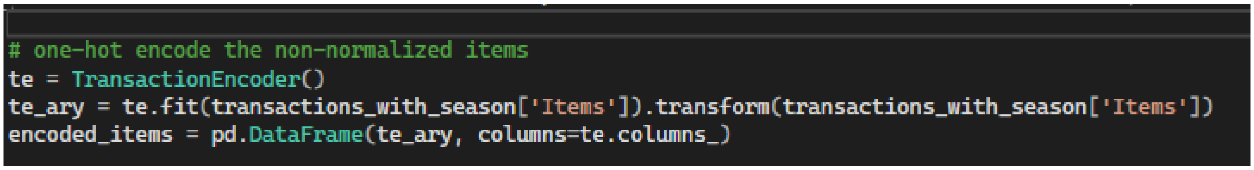
Another aspect of our dataset that we wished to reveal was which items are most frequently purchased together during each season. Our first attempt at uncovering this was to train a Naive bayes classifier on our data. Though we used the scikit-learn MultinomialNB classifier to make our predictions, for the purpose of explaining our technical approach, we will briefly describe how a Multinomial Naive Bayes classifier works. Bayes classifiers are a probabilistic framework for classification based on Thomas Baye’s logical theorem commonly known as Baye’s law. Naive Bayes classifiers, as opposed to Bayesian networks, assume that features are all conditionally independent of one another given the classification label they are assigning. This assumption is the source of the Naive prefix in Naive Bayes as, while it may allow for more efficient modeling, its lack of accounting for conditional dependence can cause the model to miss connections between features and give less accurate results when used on correlated attributes. Despite this shortcoming, Naive Bayes classifiers still tend to perform well at predicting classification of items, but more specifically, Multinomial Naive Bayes classifiers tend to excel at classifying text when using discrete features or features that can be easily encoded into integers.

The MultinomialNB assumes that its features follow a multinomial distribution, which is then used alongside a manipulated equation from Bayes law to estimate the probability of observing specific features for each possible classification of its target variable, which, in our case, was the season in which the transaction took place. The scikit-learn MultinomialNB also uses Laplace smoothing to handle features that aren't seen in a sample of training data, and finally uses logarithmic probabilities in the last step of the prediction to avoid underflow errors due to the multiplication of many very small probabilities. In the end, due to issues with the distribution of our data and the fact that items purchased together are likely to be correlated, we were unable to bring the accuracy score of our MultinomialNB Classifier to a level that we found sufficient, so we decided to use a different method of classification.

Decision Tree Classifiers

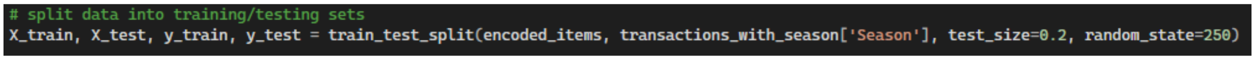
Our next attempt at predicting the season of transactions to estimate which items will sell together most frequently was to use a decision tree classifier. For this method, we again used a scikit-learn class, DecisionTreeClassifier. A decision tree is a model that simulates decisions and the possible outcomes in which they result. Inside a decision tree model, internal nodes represent tests on an attribute, branches between nodes represent an outcome of said tests, and leaf nodes represent class labels or classifications. Because the number of decision trees that can be built from a set of attributes rises exponentially as the attribute count increases, one must use 'greedy strategies' to prioritize attribute tests that maximize information gain and/or minimize node impurity. Scikit-Learn's DecisionTreeClassifier uses an optimized version of the CART tree building algorithm, which selects the next attribute to be tested at each level based on which attribute will result in the lowest amount of entropy or Gini impurity. Branches that do not significantly improve performance are then pruned, and the tree-building continues. Once the tree is built, test data can be passed into it and classified based on its feature values.

We found much better results with the decision tree classifier, which we used by first encoding our categorical data of items purchased in a transaction into an array of binary values:



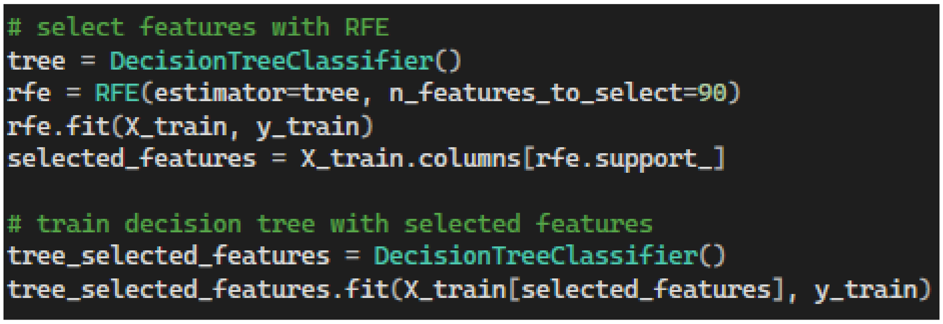
```
# one-hot encode the non-normalized items
te = TransactionEncoder()
te_ary = te.fit(transactions_with_season['Items']).transform(transactions_with_season['Items'])
encoded_items = pd.DataFrame(te_ary, columns=te.columns_)
```

We then pulled the season of each transaction based on the date it took place and split this data at an 80:20 ratio of training to test data:



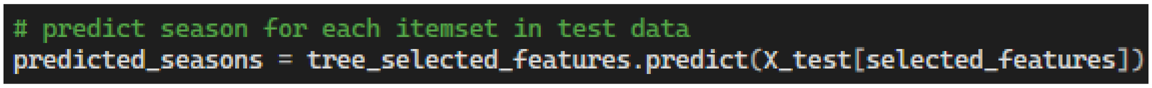
```
# split data into training/testing sets
X_train, X_test, y_train, y_test = train_test_split(encoded_items, transactions_with_season['Season'], test_size=0.2, random_state=250)
```

Next, we used recursive feature elimination to find the most important attributes for classification and fit our decision tree model to our training data:



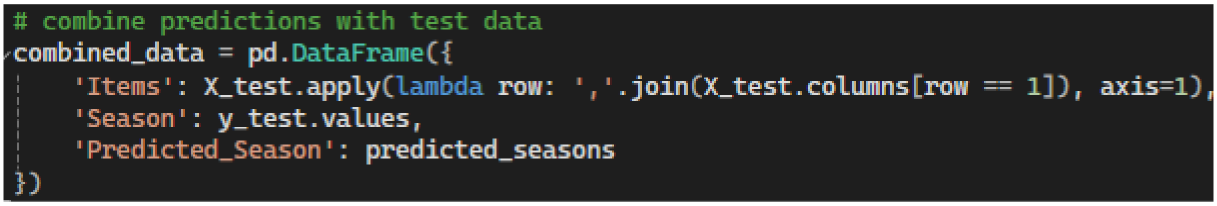
```
# select features with RFE
tree = DecisionTreeClassifier()
rfe = RFE(estimator=tree, n_features_to_select=90)
rfe.fit(X_train, y_train)
selected_features = X_train.columns[rfe.support_]

# train decision tree with selected features
tree_selected_features = DecisionTreeClassifier()
tree_selected_features.fit(X_train[selected_features], y_train)
```

Then, using this model, we predicted the season for each transaction in our test data, achieving more than double the accuracy we had with the Naive Bayes classifier.

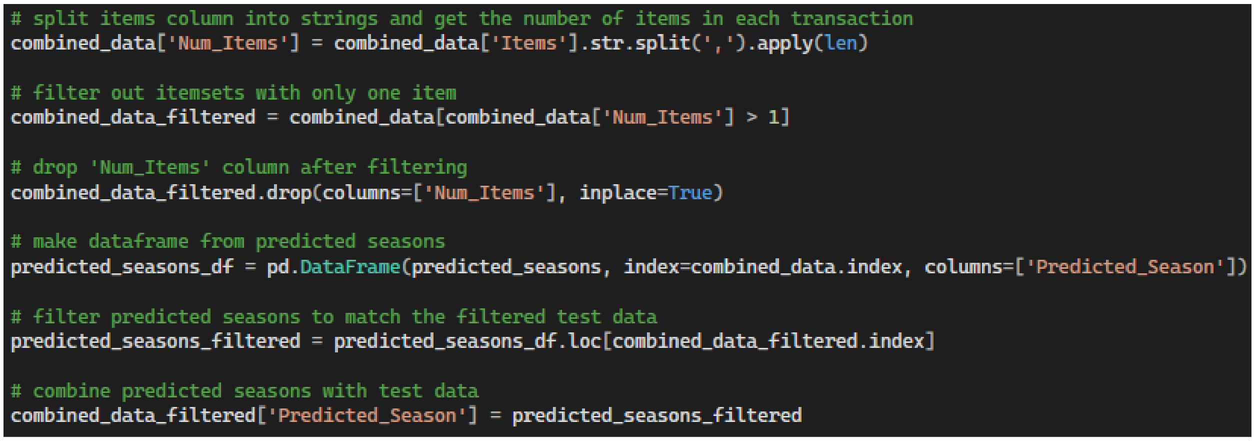


```
# predict season for each itemset in test data
predicted_seasons = tree_selected_features.predict(X_test[selected_features])
```

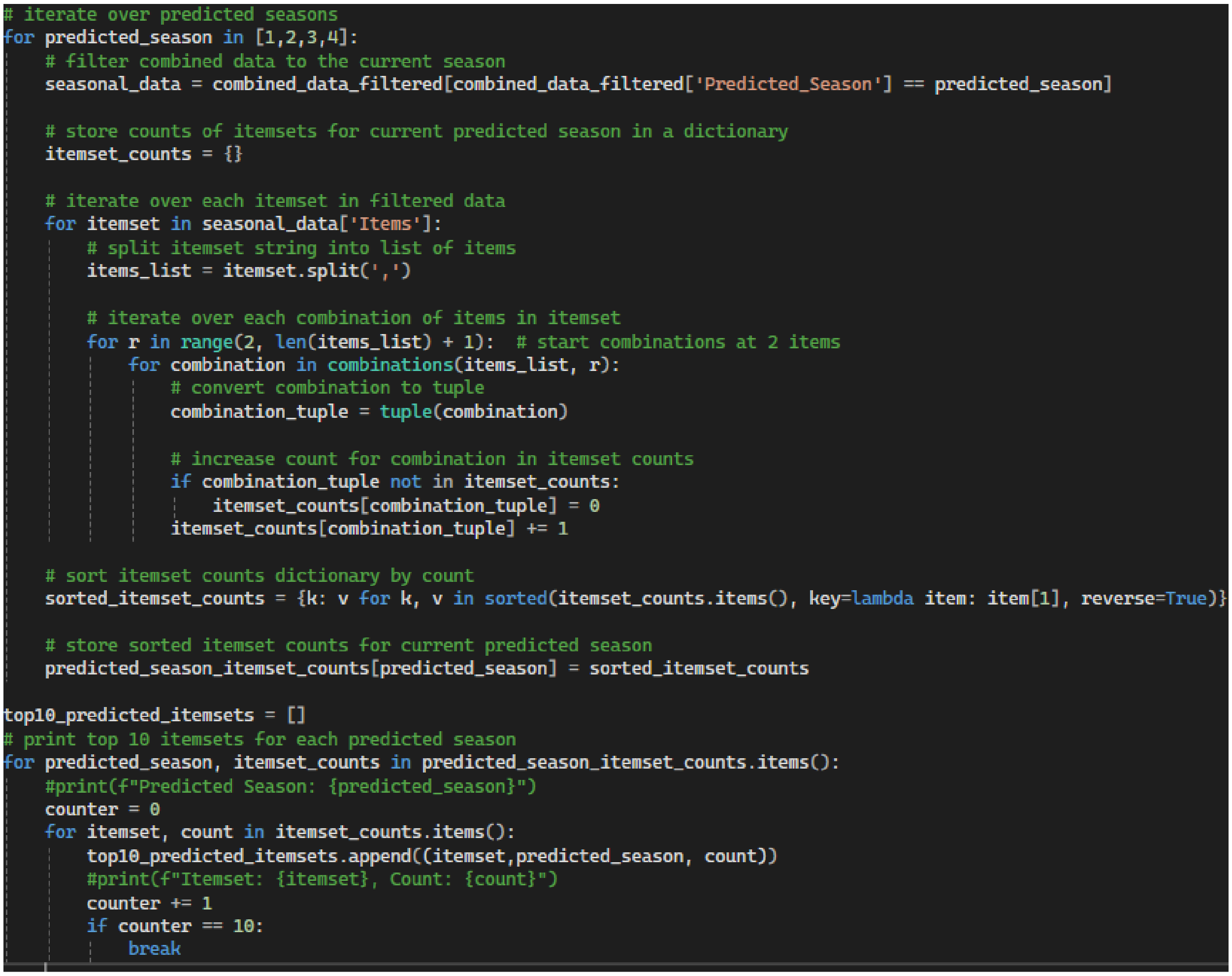
From there, we took these predicted seasons and recombined them with the test data to predict item combinations with the highest probability of being purchased together for each season.



```
# combine predictions with test data
combined_data = pd.DataFrame({
    'Items': X_test.apply(lambda row: ','.join(X_test.columns[row == 1]), axis=1),
    'Season': y_test.values,
    'Predicted_Season': predicted_seasons
})
```

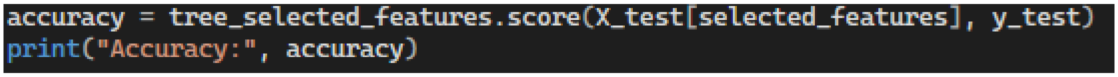


```
# split items column into strings and get the number of items in each transaction
combined_data['Num_Items'] = combined_data['Items'].str.split(',').apply(len)

# filter out itemsets with only one item
combined_data_filtered = combined_data[combined_data['Num_Items'] > 1]

# drop 'Num_Items' column after filtering
combined_data_filtered.drop(columns=['Num_Items'], inplace=True)

# make dataframe from predicted seasons
predicted_seasons_df = pd.DataFrame(predicted_seasons, index=combined_data.index, columns=['Predicted_Season'])

# filter predicted seasons to match the filtered test data
predicted_seasons_filtered = predicted_seasons_df.loc[combined_data_filtered.index]

# combine predicted seasons with test data
combined_data_filtered['Predicted_Season'] = predicted_seasons_filtered
```

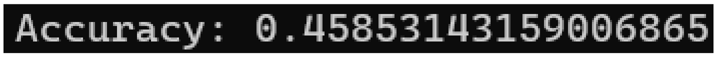


```python
# iterate over predicted seasons
for predicted_season in [1,2,3,4]:
    # filter combined data to the current season
    seasonal_data = combined_data_filtered[combined_data_filtered['Predicted_Season'] == predicted_season]

    # store counts of itemsets for current predicted season in a dictionary
    itemset_counts = {}

    # iterate over each itemset in filtered data
    for itemset in seasonal_data['Items']:
        # split itemset string into list of items
        items_list = itemset.split(',')

        # iterate over each combination of items in itemset
        for r in range(2, len(items_list) + 1):  # start combinations at 2 items
            for combination in combinations(items_list, r):
                # convert combination to tuple
                combination_tuple = tuple(combination)

                # increase count for combination in itemset counts
                if combination_tuple not in itemset_counts:
                    itemset_counts[combination_tuple] = 0
                itemset_counts[combination_tuple] += 1

    # sort itemset counts dictionary by count
    sorted_itemset_counts = {k: v for k, v in sorted(itemset_counts.items(), key=lambda item: item[1], reverse=True)}

    # store sorted itemset counts for current predicted season
    predicted_season_itemset_counts[predicted_season] = sorted_itemset_counts

top10_predicted_itemsets = []
# print top 10 itemsets for each predicted season
for predicted_season, itemset_counts in predicted_season_itemset_counts.items():
    #print(f"Predicted Season: {predicted_season}")
    counter = 0
    for itemset, count in itemset_counts.items():
        top10_predicted_itemsets.append((itemset,predicted_season, count))
        #print(f"Itemset: {itemset}, Count: {count}")
        counter += 1
        if counter == 10:
            break
```

After doing all of this, we still had a relatively low accuracy score:

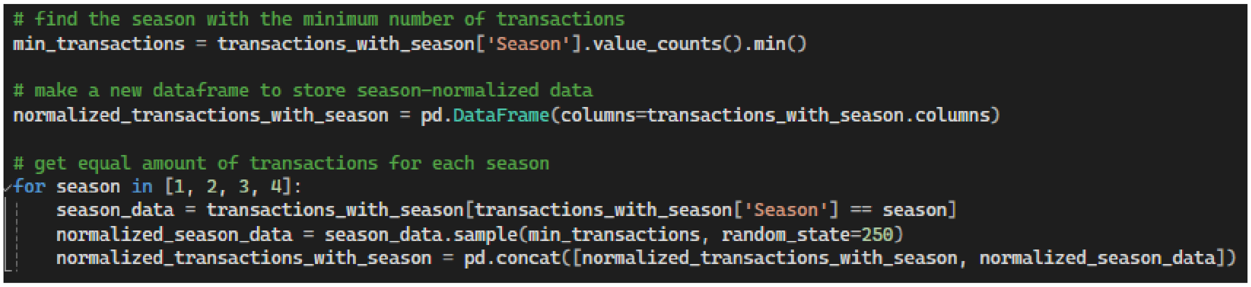


```python
accuracy = tree_selected_features.score(X_test[selected_features], y_test)
print("Accuracy:", accuracy)
```



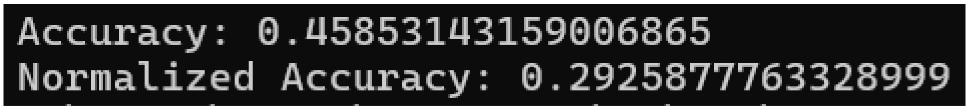
```
Accuracy: 0.45853143159006865
```

We believed this low score may have come from the uneven distribution of our data, as~77% of the data was for the seasons of winter and spring, leaving only ~23% of the data for the entirety of summer and fall. To see if this imbalance was the problem, we attempted to train a decision tree on normalized data with the same number of transactions for each season:



```python
# find the season with the minimum number of transactions
min_transactions = transactions_with_season['Season'].value_counts().min()

# make a new dataframe to store season-normalized data
normalized_transactions_with_season = pd.DataFrame(columns=transactions_with_season.columns)

# get equal amount of transactions for each season
for season in [1, 2, 3, 4]:
    season_data = transactions_with_season[transactions_with_season['Season'] == season]
    normalized_season_data = season_data.sample(min_transactions, random_state=250)
    normalized_transactions_with_season = pd.concat([normalized_transactions_with_season, normalized_season_data])
```

But unfortunately, even with many adjustments to the hyperparameters of our decision tree, we consistently got significantly lower accuracy from the normalized data:



```
Accuracy: 0.45853143159006865
Normalized Accuracy: 0.2925877763328999
```

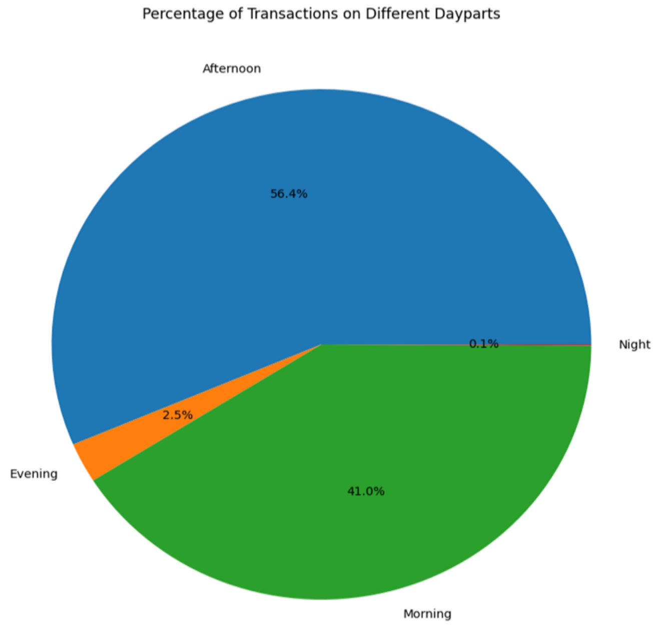
This left us with the conclusion that we simply did not have enough data to classify summer and fall transactions using items as features.

Results

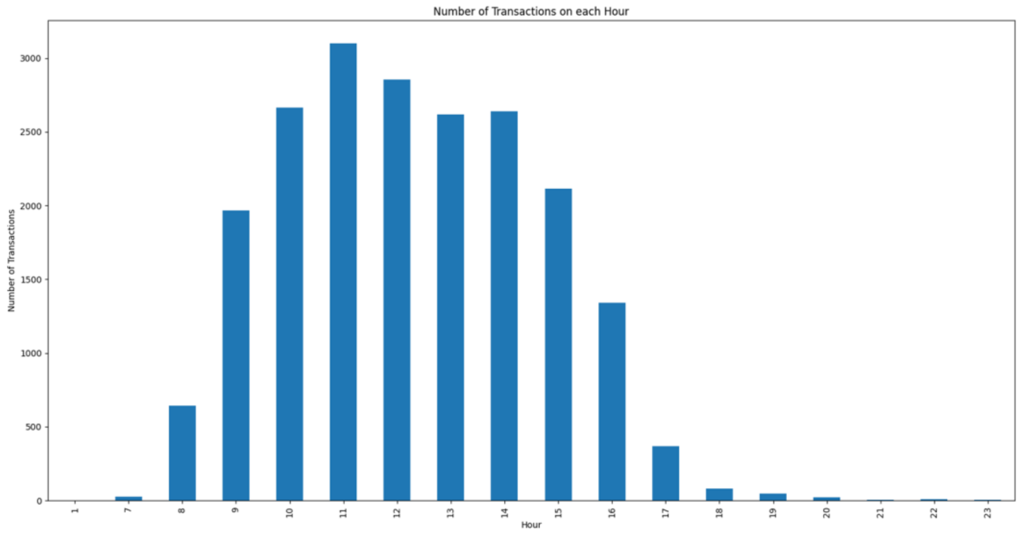
In the field of hospitality and food service, there are two ways to approach preparing customer orders. One option is to make items to-order, meaning that they aren’t created until a customer place an order. Alternatively, preparing items ahead of time allows an order to be instantly fulfilled, leaving customers satisfied. However, this introduces the possibility of food waste. When an item is made, it can only be held on to for as long as its shelf life allows. If a pastry is created and no customer orders it, the pastry will expire. This is undesirable for business as it wastes not only the potential sale of the item, but the waste extends to the value of its ingredients and the employee labor spent to create it.

Since baked items take a long time to create, baking a cake to-order would be unreasonable. However, baking the cake ahead of time introduces the possibility of it expiring on the shelf. Creating a model of customer behavior allows employees to identify trends and predict when items will be sold. Analyzing sales data is an extremely important resource in creating these models of consumer behavior. Popular items are allowed to be prepared freely while less popular items can be made less frequently, minimizing the risk of waste.
Item sales can be divided by time of day, days of the week, by month, and more. Each category offers unique insight into customer behavior that can be tapped by businesses.

For example: Item sales can be divided by the time of day the sale took place in. With this information, we can see that 97.4% of all transactions occurred in the morning and evening. According to the second figure, after 4pm sales take a sharp decline. Realizing this, a business can determine that it would not be economically viable to stay open past then.

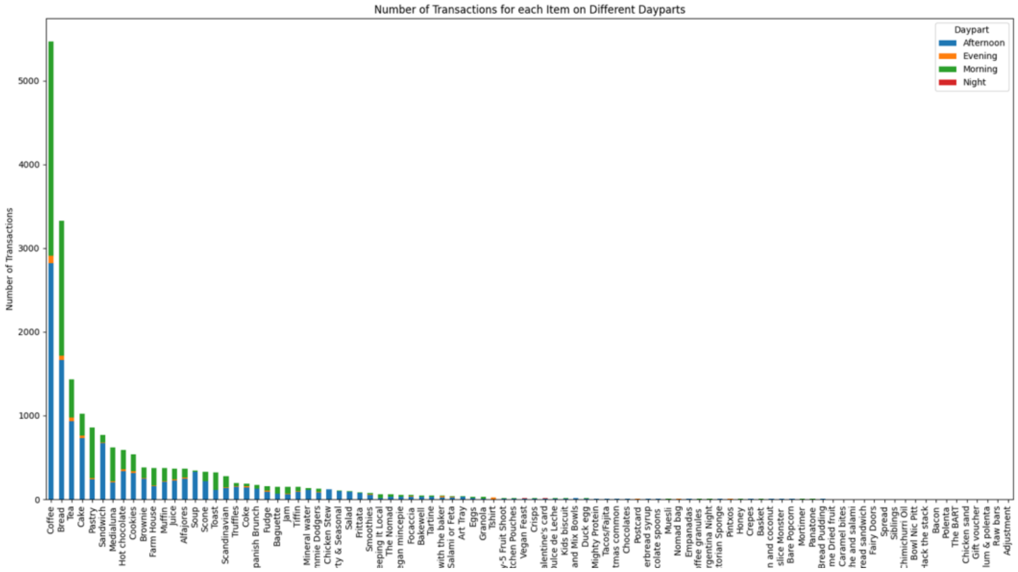


%

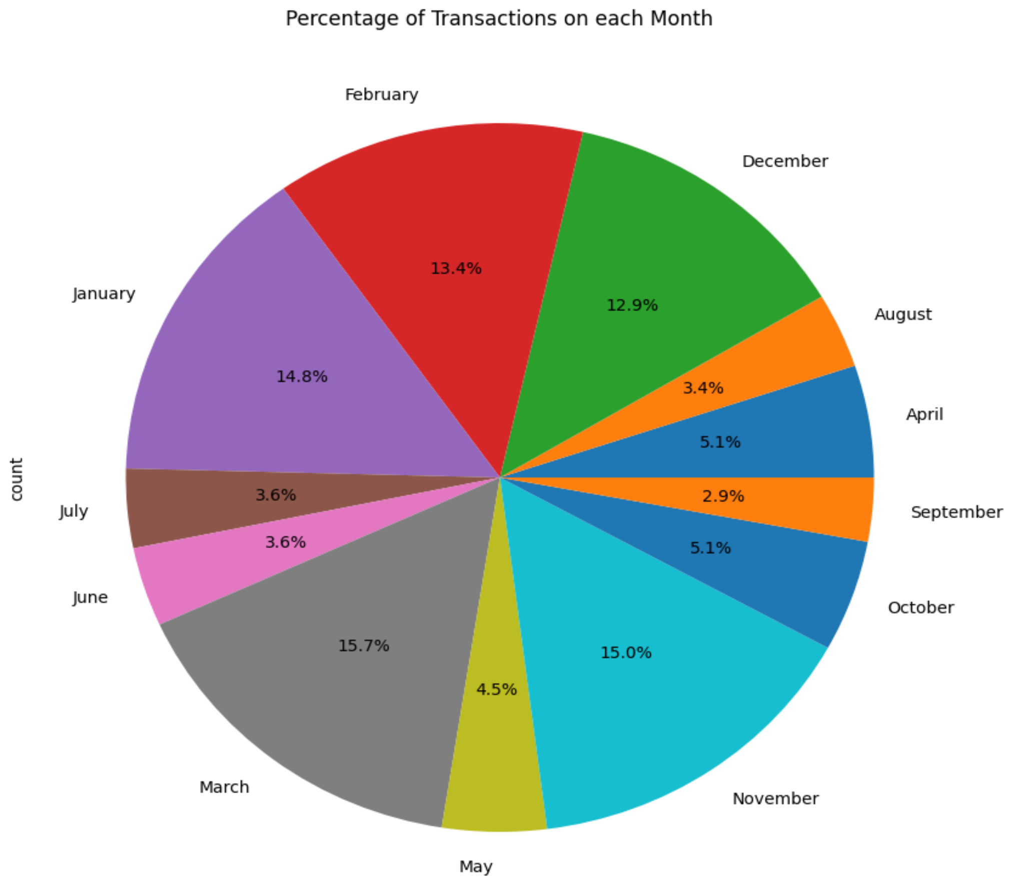


When sorted by hour, the number of transactions in our dataset illustrate this trend. The number of transactions rises dramatically between the hours of 8am and 11am, before dropping off just as sharply between 3pm and 5pm. When deciding business hours, the data supports a clear conclusion: for peak operating efficiency, operate no earlier than 7pm and close no later than 5pm. Additionally, ensure that the staff are prepared for peak operating hours by preparing inventory, scheduling additional staff, and baking popular items before customers are predicted to order them.

Going further, transactions can be further analyzed by looking at the sales data of each individual item for part of the day.



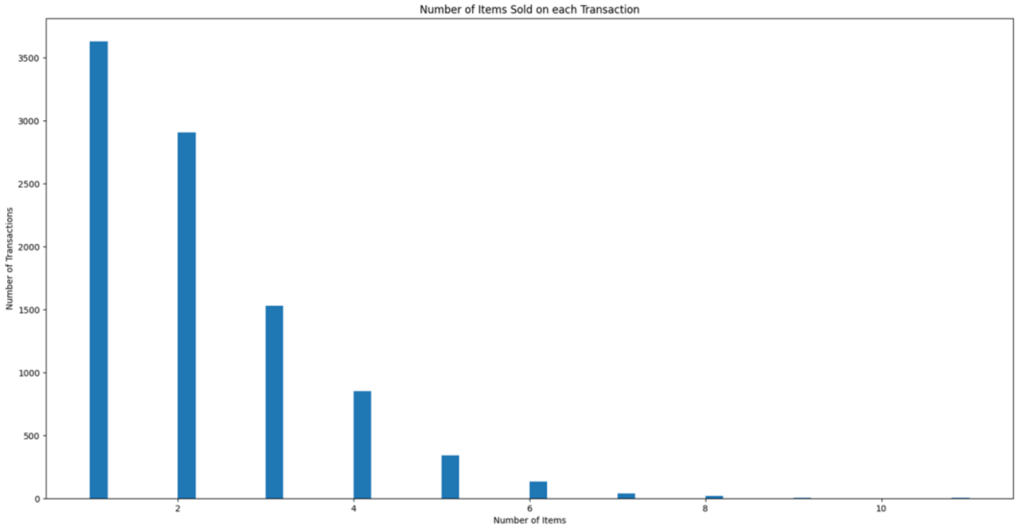
While sales data alone would tell you that cake is a more popular item than pastries, when noting the time of each transaction it is revealed that there is more information to be gleaned. Cake sells overwhelmingly more in the afternoon compared to the morning, while pastries are the inverse. This added specificity allows us to model consumers even more accurately, allowing the correct number of items to be prepared without wasting cakes in the morning and pastries in the evening, for example.



On an individual business sale, sales analysis reduces waste and ensures a smooth functioning environment. On a larger corporate scale, trends in sales throughout the year can inform staffing decisions as well. As illustrated in the chart below, sales are much higher in the winter months. According to the dataset, 71.8% of all yearly sales occurred in the five-month period between November and March, with no month outside of that accounting for more than 5.1%.

Staffing must be higher to accommodate the high volume of customers from November through March, while retaining that many employees could be needlessly expensive in the summer. Many businesses employ seasonal employees for only a few months out of the year, taking on extra hands when they're needed and parting ways once the high-demand months are over.

Analysis of sales data is especially important in a bakery setting, since expirable items such as pastries and bread items are often sold in combination with other items. If a business realizes that two items are often sold together, they can incentivize customers to purchase multiple items by marketing deals on popular item combinations.



While the plurality of transactions is for individual items, the majority of transactions include more than one item.

Seasonal Analysis Results

To further illustrate the importance of analyzing sales data in bakery settings and the impact of ebbs and flows of business throughout the year, our analysis also focused on the intra-relatedness of assorted items within single transactions in order to optimize seasonal performance. The ability to algorithmically predict item groupings that are frequently purchased together offers bakeries a modern avenue to optimize profits and streamline their operations. By leveraging these insights, bakeries can implement targeted promotions and sales strategies aimed at companion items that are frequently bought in the same transaction.

For instance, identifying that cakes and coffee are frequently purchased together could prompt the bakery to offer a bundled discount on this pairing, enticing customers who might initially come in with the sole intent of purchasing a pre-ordered cake to make a spontaneous coffee purchase and boost overall sales revenue.

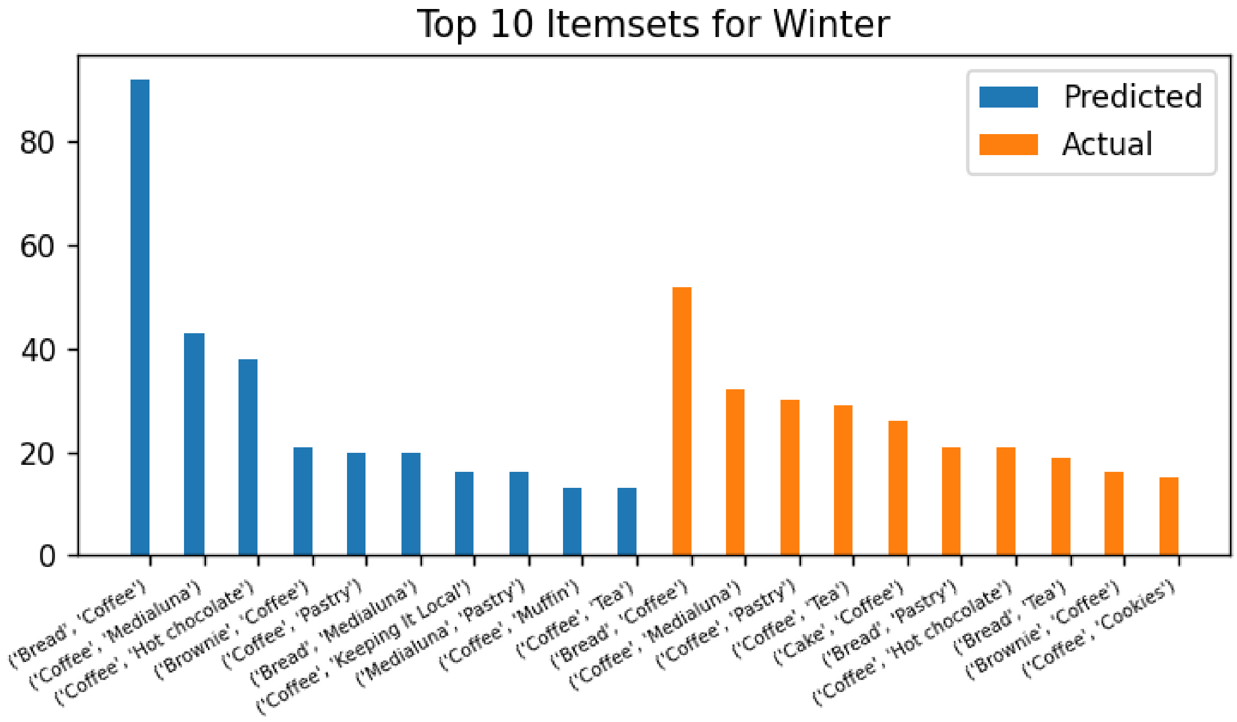
Additionally, understanding the seasonal dynamics of popular item pairings enables bakeries to make informed decisions about inventory management. By anticipating which materials will be in high demand during specific seasons or promotional periods, bakeries can optimize their ingredient purchasing schedules, minimize overstocking, and reduce potential losses due to excess inventory.

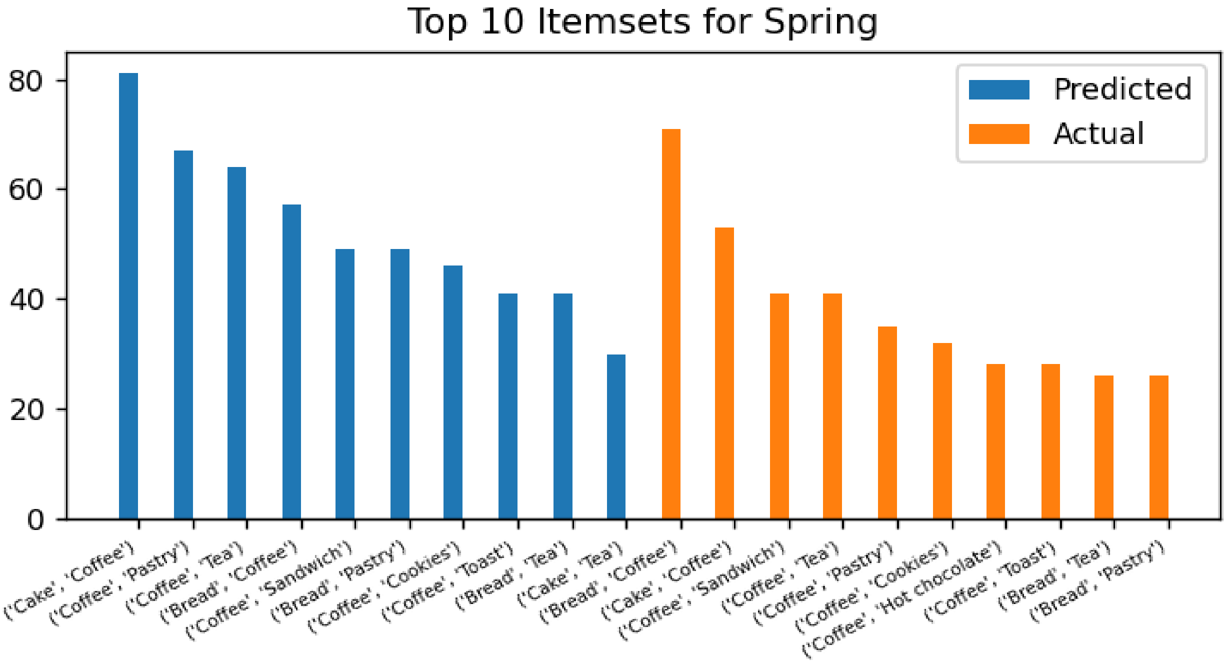
In terms of marketing, predictive insights derived from transactional data can inform strategic decisions on when and how to prioritize promotional campaigns. For instance, through predicting which item groupings will be popular during specific seasons, bakeries can align advertising efforts to capitalize on these trends, maximizing the impact of marketing investments.

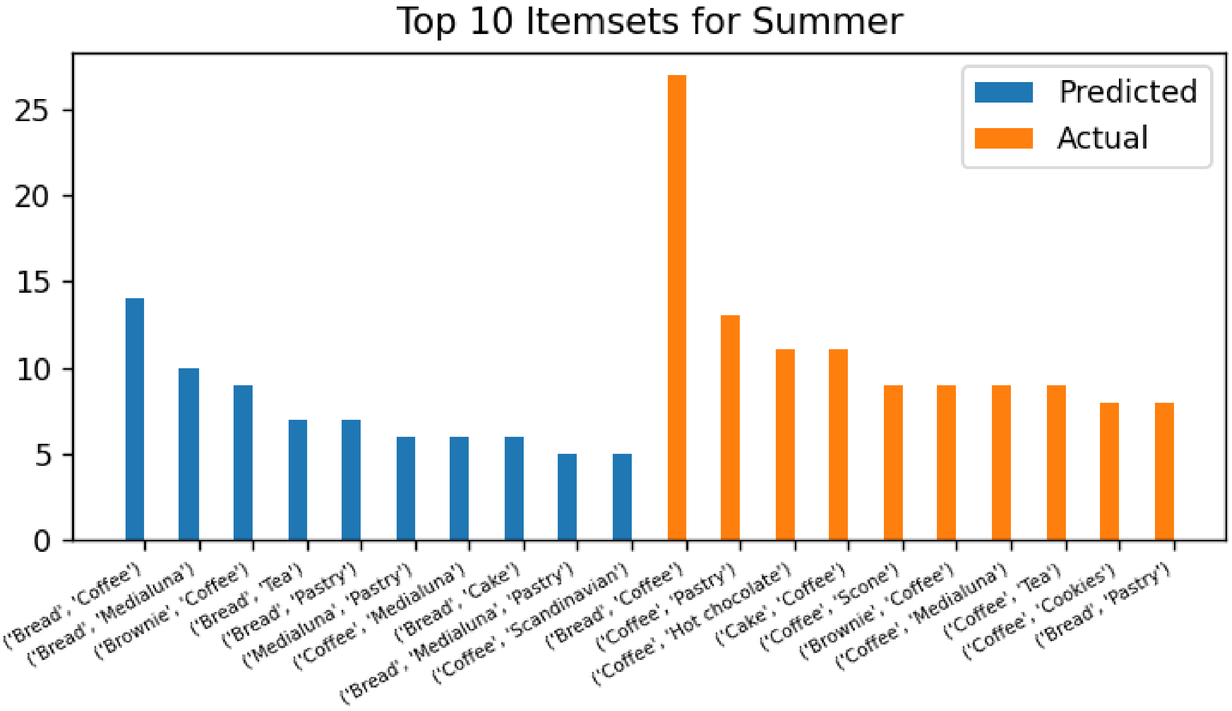
When it comes to predictions made during this analysis, the optimal strategy that emerged was to employ a decision tree classifier trained via attributes refined through recursive feature elimination. This approach excelled in uncovering and using correlations within transactional data that had hindered other classification methods but resulted in more accurate predictions of consumer preferences based on the records of previous purchases.

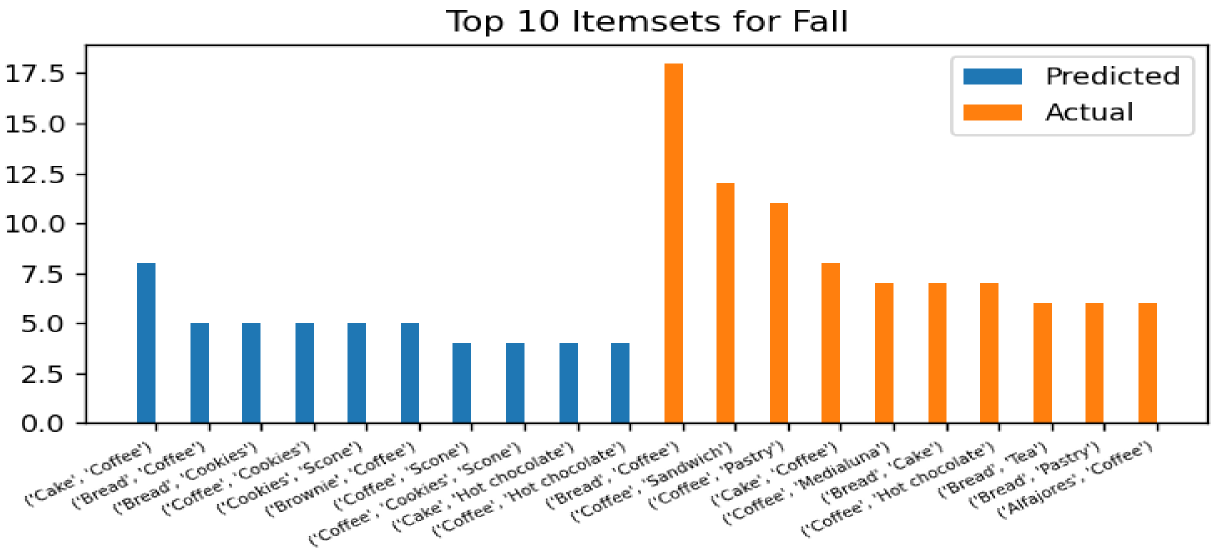
By contrast, the Multinomial Naive Bayes classifier faced even greater challenges in predicting the season in which transactions took place. Because Naïve Bayes models rely on the assumption of conditional independence of features, they often fail to properly represent the complex and correlated nature of customer behavior. This faulty reliance underscores the need for more adaptable classification techniques in this context, such as the decision tree classifier.

The following graphs show the areas where the model succeeded and where it struggled due to uneven distribution of the pre-normalized data:

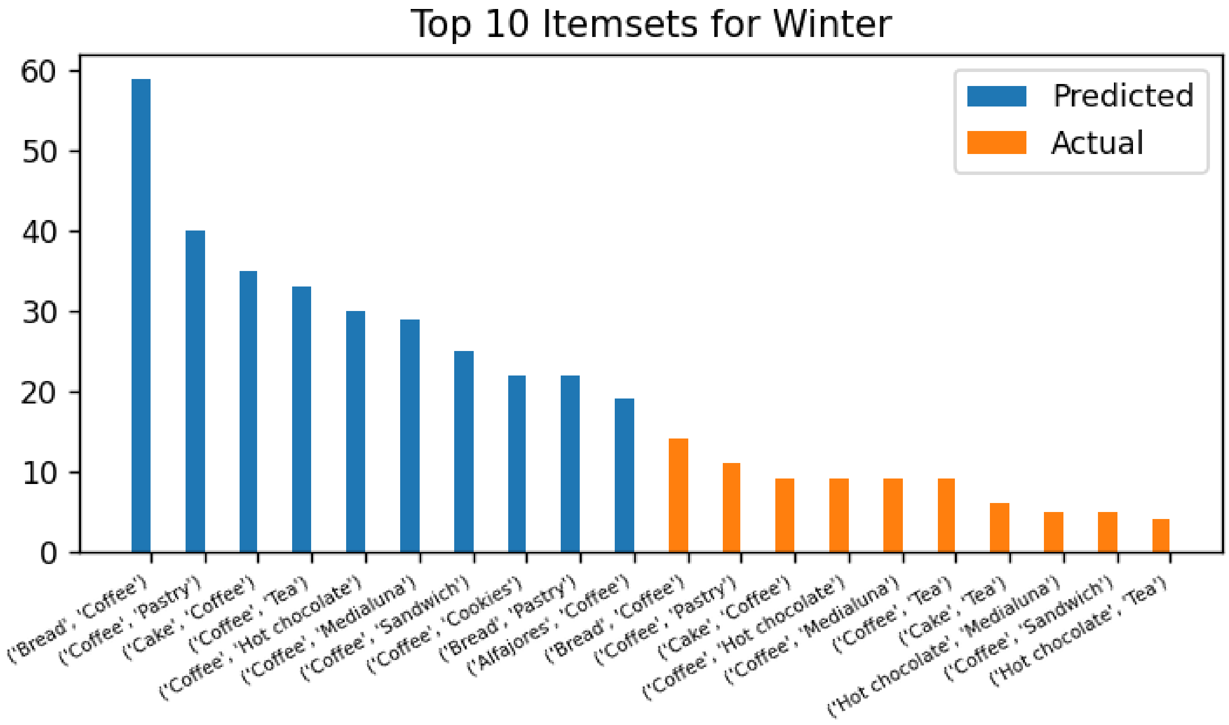


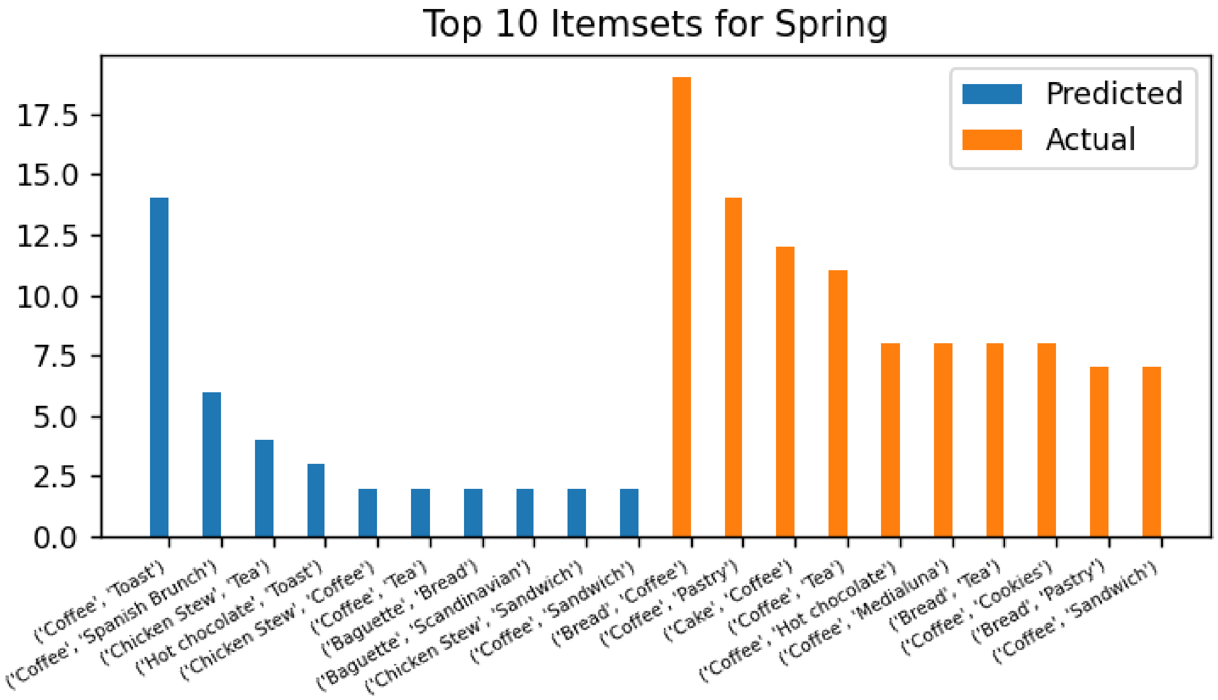

Top 10 Itemsets for Spring
Predicted
Actual
0
20
40
60
80
('Cake', 'Coffee')
('Coffee', 'Pastry')
('Coffee', 'Tea')
('Bread', 'Coffee')
('Coffee', 'Sandwich')
('Bread', 'Pastry')
('Coffee', 'Cookies')
('Coffee', 'Toast')
('Bread', 'Tea')
('Cake', 'Tea')
('Bread', 'Coffee')
('Cake', 'Coffee')
('Coffee', 'Sandwich')
('Coffee', 'Tea')
('Coffee', 'Pastry')
('Coffee', 'Cookies')
('Coffee', 'Hot chocolate')
('Coffee', 'Toast')
('Bread', 'Tea')
('Bread', 'Pastry')

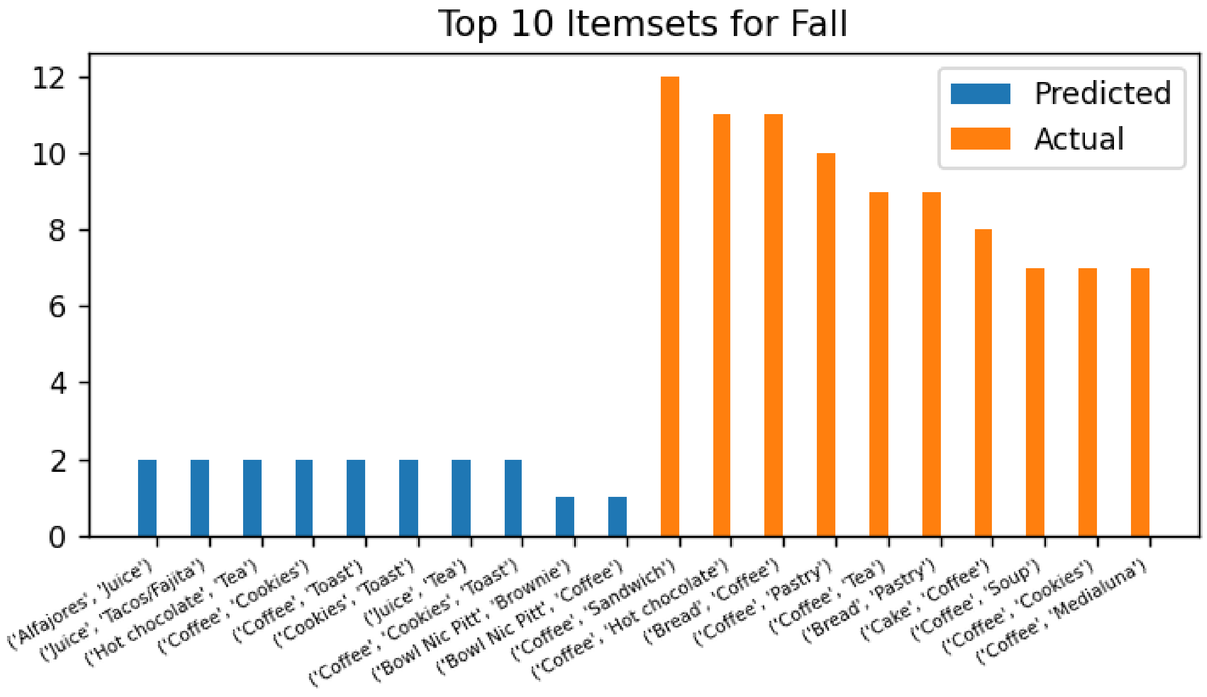

Top 10 Itemsets for Summer
Predicted
Actual
0
5
10
15
20
25
('Bread', 'Coffee')
('Bread', 'Medialuna')
('Brownie', 'Coffee')
('Bread', 'Tea')
('Bread', 'Pastry')
('Medialuna', 'Pastry')
('Coffee', 'Medialuna')
('Bread', 'Cake')
('Bread', 'Medialuna', 'Pastry')
('Coffee', 'Scandinavian')
('Bread', 'Coffee')
('Coffee', 'Pastry')
('Coffee', 'Hot chocolate')
('Cake', 'Coffee')
('Coffee', 'Scone')
('Brownie', 'Coffee')
('Coffee', 'Medialuna')
('Coffee', 'Tea')
('Coffee', 'Cookies')
('Bread', 'Pastry')

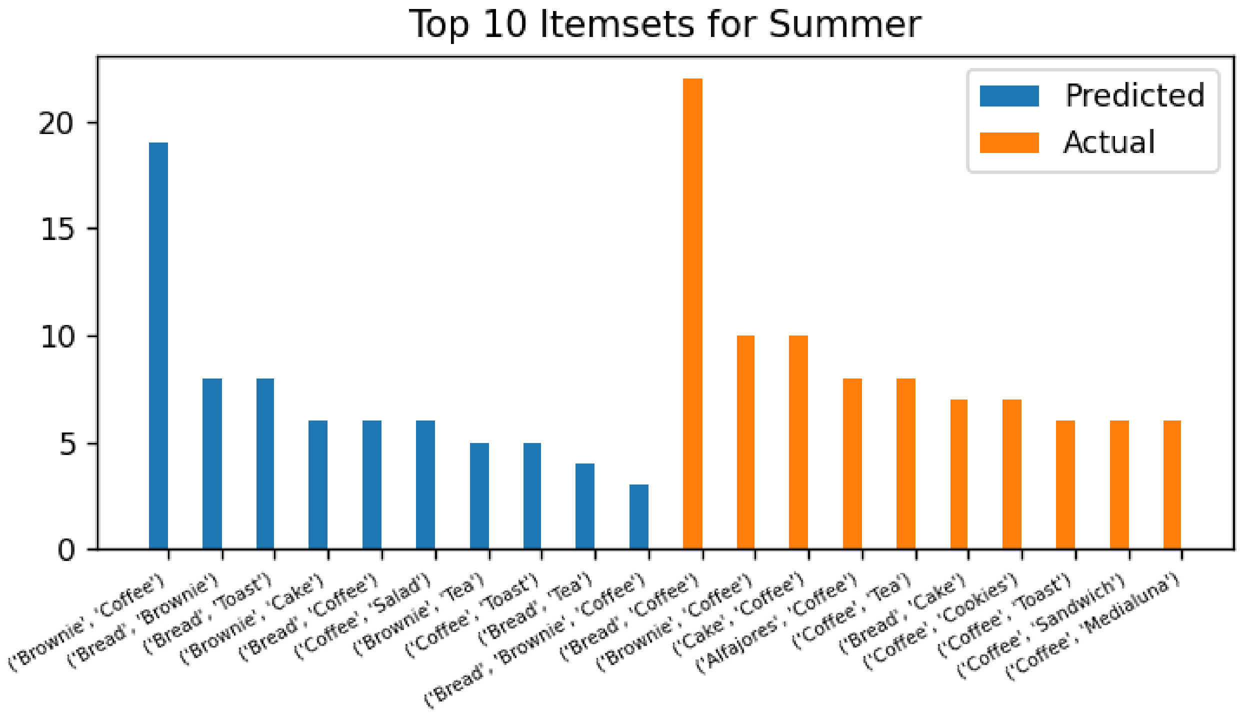


It is clear from reviewing these graphs that while the predicted counts of transactions may be slightly off during winter and spring, the itemset which were predicted to be most popular were fairly close to the actual itemset. As mentioned previously, the uneven distribution of our data between winter/spring and summer/fall created difficulties when it came to predicting itemset during the latter seasons. When we attempted to normalize this imbalance, we ended up with even lower accuracy than before normalization, which can be seen in the following graphs:




Top 10 Itemsets for Spring


Top 10 Itemsets for Fall



We believe that our decision tree requires model requires more transactions than our dataset contained for the summer and fall seasons in order to properly classify them, as even though the pre-normalized predictions for the winter and spring seasons were not perfect, they vastly outperformed the summer and fall seasons. Also, because bread and coffee are one of the most popular combinations for each season, it is likely that too many of these interactions were classified as occurring in the winter due to overfitting in our tree. If we had access to more transactions for summer/fall, our predictions would likely be more accurate and therefore more useful for bakery businesses.

Steps to Reproduce:

To reproduce our results, visit our github and follow the steps in the readme.

Recommendations for future work:

When working towards similar goals in the future with such datasets, we recommend incorporating additional data resources such as customer demographics, weather patterns, and social media, among other categories to help improve the accuracy and performance of the predictions. Having more categories to classify the data caters for random error or sampling variability where the results would be subject to random fluctuations due to limited classification or sampling variability and may not be representative of the data. With this too, the FP-Growth algorithm can better identify frequent association rules and make more accurate predictions.