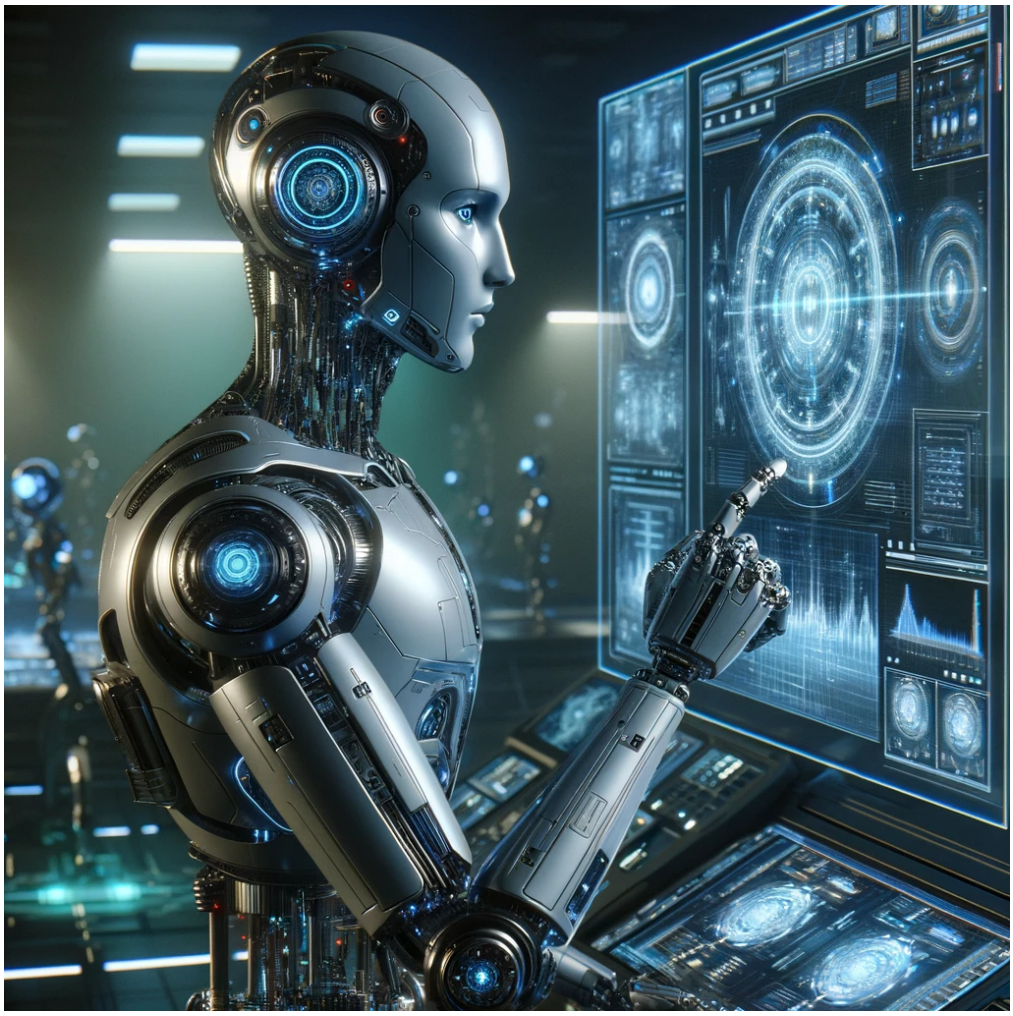
ONLINE RETAIL MARKET BASKET ANALYSIS AND CUSTOMER SEGMENTATION, 2009-2011, UK



Per Idar Rød

Table of Contents

Cover image generated using Dall-E 3

1. Summary

This project aims to gain valuable insights into customer behavior, extract “hidden” information from the data, and predict customers’ future spending.

In this study, several key insights were obtained:

- Cross-Selling Opportunities: By identifying frequently bought item sets, the project revealed cross-selling opportunities, which can help in making targeted product suggestions and promotions.
- Customer Segmentation: Both RFM (recency, frequency, monetary) analysis and unsupervised clustering (grouping) revealed distinct customer segments. These segments can inform targeted marketing campaigns, improve customer retention efforts, and identify potential churn risks.
- Predictive Models: The predictive models developed provide valuable insights for future planning and resource allocation, helping the business anticipate long-term revenue from each customer.

The detailed results can be found in Chapter 11: Discussion.

2. Introduction

Have you ever wondered if you could group your customers in a better way or thought about what combinations of products they put into their shopping baskets? Are there connections to be made based on their purchasing history? Are you at risk of losing customers, or do you want to find good opportunities for cross-selling products? Perhaps you struggle with targeting the correct customers with tailor-made marketing strategies? Would you like to find out how much your customers will spend at your business in the future?

This study leverages AI techniques, more spesifically machine learning to address these issues. Enhancing customer insights by analyzing frequently bought items, segmenting customers with similar characteristics and predicting customer lifetime value, we aim to

utilize the power of computers to analyse and interpret data, revealing valuable insights from the data related to your customers.

3. Background

Online Retail

The online retail domain involves selling goods and services directly to consumers or other businesses via the internet, hence the term “online.” This market has grown exponentially in recent years, becoming a critical component of the global economy. The convenience and accessibility of online shopping have transformed consumer behavior, enabling people to purchase a wide variety of products from the comfort of their homes. Additionally, retailers can display their products in a virtual environment, reaching a global market without the need for physical stores (Brenner, 2023).

Online retail offers several advantages over traditional brick-and-mortar stores. One significant benefit is the ability to operate 24/7, providing customers with the flexibility to shop at any time. This continuous availability has been a key driver in the growth of the online retail sector. Furthermore, online platforms can offer a broader range of products than physical stores, as they are not limited by shelf space. This extensive product availability can attract a diverse customer base and increase sales opportunities.

Another crucial aspect of online retail is the potential for personalized shopping experiences. By leveraging customer data and advanced analytics, online retailers can offer personalized recommendations, targeted promotions, and tailored content. This level of customization enhances the customer experience, fosters loyalty, and can lead to higher conversion rates.

Moreover, the online retail market has seen significant advancements in technology, such as the integration of artificial intelligence (AI) and machine learning. These technologies help retailers optimize their supply chain, manage inventory more efficiently, and provide better customer service through chatbots and automated systems.

The global reach of online retail is another transformative factor. Retailers can easily expand their market presence across different countries and regions without the need for physical stores. This global accessibility not only increases potential customer bases but also allows retailers to tap into emerging markets with growing internet penetration and consumer spending.

Overall, the online retail domain continues to evolve rapidly, driven by technological innovations and changing consumer preferences. Retailers that adapt to these changes and leverage the unique advantages of online platforms can achieve significant growth and success in this competitive landscape.

Market Basket Analysis

The need for online and physical stores to understand their customers better and target them with appropriate marketing strategies is greater than ever before. Over the past 20 years, modern companies have increasingly collected and analyzed customer data to gain insights into purchasing behavior, payment methods, and other relevant patterns. Market Basket Analysis involves analyzing what customers put into their baskets, whether as consumers or businesses. It aims to answer what sets of items customers are likely to purchase on a given trip to the store (Jiawei, Jian, & Hanghang, 2022).

Market Basket Analysis uses data mining techniques to uncover patterns and correlations in transaction data. For instance, it can reveal that customers who buy bread are also likely to buy butter, which can inform product placement and promotional strategies. This analysis is crucial for identifying cross-selling opportunities, optimizing inventory management, and enhancing the overall shopping experience.

Advanced algorithms, such as association rule mining, are employed to identify these relationships. The most famous of these algorithms is the Apriori algorithm, which efficiently finds frequent itemsets and generates association rules. These insights enable retailers to

create more effective marketing campaigns, bundle products strategically, and improve customer satisfaction.

Integrating Market Basket Analysis with other data analytics techniques, such as predictive analytics and customer segmentation, provides a comprehensive view of customer behavior. This integration helps businesses to not only understand current purchasing patterns but also to predict future trends and customer needs, thereby staying ahead in a competitive market.

Customer Segmentation

The need to segment customers is also increasing for online retailers. They must do this to provide tailor-made marketing strategies for the right customers. “Customer segmentation involves grouping existing and potential customers based on shared characteristics” (Jolaoso, 2024). For example, grouping customers can help identify those likely to churn, known as customer churn analysis, or determine who the best customers are.

Customer segmentation allows businesses to personalize their marketing efforts, resulting in higher engagement and conversion rates. By understanding the specific needs and behaviors of different customer groups, retailers can design targeted campaigns that resonate more effectively with each segment.

Customer segmentation offers several benefits, including improved customer retention, enhanced marketing efficiency, and increased profitability. By focusing on the most profitable segments, businesses can allocate their resources more effectively and develop products and services that meet the specific needs of each group.

Superviced vs unsuperviced learning

Superviced learning:

Supervised learning, or supervised machine learning, refers to the process where a computer program is trained using a predefined, labeled dataset. The training dataset includes inputs and their corresponding correct outputs, enabling the model to learn and make accurate predictions on new, unseen data. The model adjusts its parameters through a process called fitting, which is part of cross-validation. Supervised learning problems are categorized into classification (predicting categories) and regression (predicting continuous values), using various algorithms such as linear classifiers, support vector machines, decision trees, and regression models.

Unsuperviced learning:

Unsupervised learning, or unsupervised machine learning, involves a computer program autonomously finding patterns and relationships in an unlabeled dataset to solve clustering and association problems. It primarily encompasses three tasks:

- Association: This technique finds relationships between variables in a dataset using rule-based methods. Examples include recommendation systems like “customers who bought this also bought that.” Apriori algorithms are commonly used for this purpose.
- Clustering: Clustering groups unlabeled data into clusters based on similarities or differences. Various clustering algorithms serve different applications.
- Dimensionality Reduction: This reduces the number of features in a dataset to a more manageable size while preserving data integrity. Common methods include Principal Component Analysis (PCA), singular value decomposition, and autoencoders.

Unsupervised learning has numerous business applications, such as:

- Anomaly Detection: Identifying atypical data points for detecting human errors, security breaches, or faulty equipment.
- Customer Personas: Creating profiles to understand buying habits, aiding in targeted marketing.
- Computer Vision: Object recognition in machine vision systems.

- Medical Imaging: Enhancing image detection, classification, and segmentation in radiology and pathology.
- News Categorization: Grouping related news stories and personalizing reader experiences.
- Recommendation Engines: Uncovering data trends for cross-selling strategies and making relevant product recommendations.

Classification vs. Regression

- Classification: Used for predicting categorical outcomes. The model assigns inputs to one of several predefined classes (labels).
- Regression: Used for predicting continuous outcomes. The model estimates the relationship between input variables and a continuous target variable.

Some abbreviations and technical terms used in this report:

- Python - refers to the programming language python. Python files have .py as the file extension.
- ML - Machine Learning, refers to having the computer(machine) learn from the data we select.
- Model - Refers to a machine learning model (there are many) which we can train to output predictions, or an appropriate label when given properly cleaned and prepared data.
- Code or script - refers to the .py(python) files where the data cleaning, data analysis, model training/evaluation/classification/predictions is done.
- NaN - pythons way of saying that this is Not a Number.
- MBA - Market Basket Analysis.
- RFM - Recency, Frequency, Monetary (analysis)
- CLV – Customer lifetime value
- Hyperparameters – ML models have internal parameters that they use, the term hyperparameter refers to the parameters the developer can tune to get the best results from the model.

To be able to read this report effectively here is the flow of the work done in order to reach the overall goal of the study:

- Data cleaning

- Market Basket Analysis
- Customer segmentation
- Predicting CLV (customer lifetime value)

4. Problem statement

It is challenging to accurately group customers and target them with efficient and effective marketing strategies. This difficulty can lead to missed opportunities, wasted resources, and decreased customer satisfaction.

5. Project objectives

“Enhance customer insights and sales strategy through market basket analysis and customer segmentation in online retail.”

6. Data warehouse details

The data used in this study was procured from the website Kaggle. A search was conducted for datasets in the domain of online retail, leading to the discovery of the dataset available at Online Retail II Data Set from ML Repository.

In this study the "Online Retail II" dataset are used, available under the Open Data Commons, Database Contents License (DbCL) v1.0 and Open Database License (ODbL). This dataset encompasses transaction records from a UK-based online retail between 2009 and 2011, freely available for academic and commercial use. The project involves conducting market basket analysis and customer segmentation to enhance marketing strategies. The dataset's usage is in full compliance with the mentioned licenses, ensuring respect for legal standards and ethical data usage.

This dataset consists of two parts, both of which were downloaded as .csv files. These parts were then merged into a single dataset using Python for the purposes of this study.

7. Interesting insights

Table 1 highlights a selection of rows that illustrate some issues in the original data. There is a wealth of information present. In the "StockCode" column, certain entries stand out because they contain letters alongside numbers, such as "POST" and "DOT." Similarly, in the Invoice column, some invoice numbers are prefixed with the letter "C." According to the dataset provider, the "C" prefix denotes a cancelled order, further evidenced by negative numbers in the Quantity column.

The Description column is highly effective for identifying which customers purchase specific products. The Quantity and Price columns are crucial for data analysis as they provide insights into purchasing behaviour and revenue generation.

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
489439	POST	POSTAGE	3	12/01/2009 09:28	18	12682	France
489597	DOT	DOTCOM POSTAGE	1	12/01/2009 14:28	647.19		United Kingdom
C565382	CRUK	CRUK Commission	-1	09/02/2011 15:45	13.01	14096	United Kingdom
C489535	D	Discount	-1	12/01/2009 12:11	9	15299	United Kingdom
C565428	M	Manual	-1	09/04/2011 12:53	4.25	12867	United Kingdom
489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	12/01/2009 07:45	6.95	13085	United Kingdom

Table 1

8. Data pre-processing

General data cleaning and removal of missing entries.

To train machine learning models on the data, it is essential to ensure the dataset is properly cleaned. This involves making sure there are no missing entries in the dataset. Data pre-processing is carried out at various stages throughout the code. The initial pre-processing step is a general data cleaning task applied to the entire dataset. Image 1 illustrates how this data cleaning was implemented in the code.

After the merge of the two original files the following steps were taken:

- Dropping duplicate entries(there were some due to the overlap in December)
- Converting “InvoiceDate” to datetime objects. This is done so python can work with the dates effectively.
- Creating time features(Month, Day of week and Hour) This is for the models to be able to capture seasonality in the data.
- Create column “TotalSpend” by multiplying quantity with price on every transaction. The main reason for this is to be able to aggregate the data on each customer, since each customer might have multiple transactions connected to them.
- Split the letters from the numbers in “StockCode” column. This is done to be able to handle the column as numerical.
- Split the letters from the numbers in “Invoice” column, and drop the letter column. The same reason as for the “StockCode” column. It eases the use of the column if it holds numbers only.
- Filter and remove “Description” column for the words: Manual and Postage as these are not descriptions of items. They will not add value for the goal of this study.
- Filter and remove “StockCodeVariation” for the words: Adjust, D, DOT and CRUK. These will not add value to this study, but rather add noise to the data.

- Deal with negative values in columns “Quantity” and “Price”. I’ve only included positive values. One could argue that to gain further customer insight they should have been included, however in the scope of this study, the focus is on actual buying of goods, and not returns. The returns i.e. negative values, would need an investigation of its own.

- Dropping rows where the “Customer ID” are empty. This is done to ensure we can connect insights gained to each individual customer.

- Filling all empty rows in “StockCode” with 0. This is done to avoid getting NaN warning when working with the data.

```
    # Remove duplicates
    data.drop_duplicates(keep='first', inplace=True)
    print(
        "Dropped duplicates. Combined data (no duplicates) shape:", data.shape)

    # Converting from string to datetime
    data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
    # Create time features
    data['Month'] = data['InvoiceDate'].dt.month
    data['DayOfWeek'] = data['InvoiceDate'].dt.dayofweek
    data['TimeOfDay'] = data['InvoiceDate'].dt.hour

    # Add total spend column
    data['TotalSpend'] = data['Quantity'] * data['Price']

    # Display unique customers
    unique_customers = data['Customer ID'].nunique()
    print("Unique customers:", unique_customers)

    # Split letter codes from numbers in 'StockCode' column
    data['StockCodeVariation'] = data['StockCode'].str.extract(
        '([A-Za-z]+)', expand=False)
    data['StockCode'] = data['StockCode'].str.extract(r'(\d+)', expand=False)

    # Extract letter from 'Invoice' column and drop it
    data['CancelledOrders'] = data['Invoice'].str.extract(
        '([A-Za-z]+)', expand=False)
    data.drop('CancelledOrders', axis=1, inplace=True)
    data['Invoice'] = data['Invoice'].str.extract(r'(\d+)', expand=False)

    # Remove specific entries and filter out negative quantities and prices
    data = data[~data['Description'].isin(['Manual', 'POSTAGE'])]
    data = data[~data['StockCodeVariation'].isin(
        ['ADJUST', 'D', 'DOT', 'CRUK'])]
    data = data[data['Quantity'] > 0]
    data = data[data['Price'] > 0]

    # Handle missing 'Customer ID' values
    data.dropna(subset=['Customer ID'], inplace=True)
    data['Customer ID'] = data['Customer ID'].astype(int)

    # Handle missing 'StockCode' values
    data['StockCode'] = data['StockCode'].fillna(0).astype(int)
```

Image 1

Furter adjustments are shown in Image 2:

- Handling missing values in “Description” column, by checking the “StockCode” and assigning correct descriptions with same StockCode value to the missing descriptions.
- Removing rows where “Description” is empty to avoid NaN, and to be sure all rows holds the description of an item.
- Fill NaNs in “StockCodeVariation” column with the string “None”. This means the cell holds information, rather than nothing, even if there is no value there.
- Set the column order in a systematic way for further work with the dataset.

```
    # Handle missing 'Description' values
    description_dict = data.dropna(
        subset=['Description']).set_index('StockCode')[
        'Description'].to_dict()
    data['Description'] = data.apply(
        lambda row: description_dict[row['StockCode']] if pd.isnull(
            row['Description']) and row['StockCode'] in
        description_dict else row['Description'],
        axis=1
    )
    data.dropna(subset=['Description'], inplace=True)

    # Fill missing 'StockCodeVariation' with 'None'
    data['StockCodeVariation'] = data['StockCodeVariation'].fillna('None')

    # Display the shape of the cleaned data
    print("Shape of cleaned data:", data.shape)

    # Display missing entries in the combined dataset
    missing_entries = data.isnull().sum()
    print("Missing Entries in combined dataset:\n", missing_entries)

    # Set the column order for the DataFrame
    column_order = [
        'Customer ID', 'Country',
        'InvoiceDate', 'Invoice',
        'StockCode', 'StockCodeVariation', 'Description',
        'Month', 'DayOfWeek', 'TimeOfDay',
        'Quantity', 'Price', 'TotalSpend'
    ]

    data = data[column_order]
```

Image 2

Data pre-processing for Market Basket Analysis

The only extra step in pre-processing specifically for MBA is shown in Image 3.

- Creating a list of all descriptions per invoice.

```
    # Converting grouped items to a list of lists,
    # where each list contains items from a single invoice.
    transactions = data.groupby('Invoice')['Description'].apply(list).tolist()
```

Image 3

Data pre-processing for Customer Segmentation

Image 4 depicts the pre-processing steps undertaken prior to performing an RFM analysis, which involved creating new columns to assist in the customer segmentation clustering task. While an RFM analysis is informative on its own, in this context it was specifically used to enhance the clustering process, ensuring the creation of meaningful customer segments.

```
    # Adding one day to the end of the 'InvoiceDate' to avoid getting 0s.
    current_date = data['InvoiceDate'].max() + pd.Timedelta(days=1)

    # Grouping data and aggregating new columns
    rfm = data.groupby('Customer ID').agg({
        'InvoiceDate': lambda x: (current_date - x.max()).days,
        'Invoice': 'count',
        'Price': lambda x: x.sum()
    }).rename(columns={'InvoiceDate': 'R_Score',
                       'Invoice': 'F_Score',
                       'Price': 'M_Score'
                       })

    # Define levels based on raw scores
    def rfm_level(df):
        if df['R_Score'] <= rfm['R_Score'].quantile(0.33) and \
                df['F_Score'] > rfm['F_Score'].quantile(0.66) and \
                df['M_Score'] > rfm['M_Score'].quantile(0.66):
            return 'High Value'
        elif df['R_Score'] > rfm['R_Score'].quantile(0.33) and \
                df['R_Score'] <= rfm['R_Score'].quantile(0.66) and \
                df['F_Score'] > rfm['F_Score'].quantile(0.33) and \
                df['M_Score'] > rfm['M_Score'].quantile(0.33):
            return 'Medium Value'
        else:
            return 'Low Value'

    # Setting labels on every entry
    rfm['RFM_Level'] = rfm.apply(rfm_level, axis=1)

    # Merging the data frames
    data = data.merge(rfm[['R_Score', 'F_Score', 'M_Score',
                           'RFM_Level']], on='Customer ID', how='left')

    print("\nCustomer Segmentation results:\n", data.shape)
    print(data.head())
```

Image 4

Further pre-processing were done before running the kMeans clustering on the data as can be seen in Image 5.

- Setting the order of the “RFM_Level” to get the heatmap displayed in the correct way.
- Selecting which columns to include in the clustering.
- Applying scaling and OneHotEncoder to the columns to make the data suitable for the clustering model. One-hot encoding is used to convert categorical variables into a format that can be provided to machine learning algorithms to improve predictions.

```
413     # Setting order in rfm_level to get the heatmap correct
  1     rfm_order = ['High Value', 'Medium Value', 'Low Value']
  2     data['RFM_Level'] = pd.Categorical(
  3         data['RFM_Level'], categories=rfm_order, ordered=True)
  4
  5     print("Starting clustering pipeline...")
  6     try:
  7         # Defining columns for different preprocessing.
  8         categorical_cols = ['Country']
  9         numeric_cols = ['Quantity', 'Price', 'Month', 'DayOfWeek',
 10                         'TimeOfDay', 'R_Score', 'F_Score', 'M_Score',
 11                         'TotalSpend']
 12         rule_cols = [col for col in data.columns if 'Rule_' in col]
 13
 14         required_columns = numeric_cols + categorical_cols + rule_cols
 15         missing_columns = [
 16             col for col in required_columns if col not in data.columns]
 17         if missing_columns:
 18             print(f"Missing columns in kmeans:\n{missing_columns}")
 19             return None
 20
 21         # Preproccessing pipeline
 22         preprocessor = ColumnTransformer(
 23             transformers=[
 24                 ('num', StandardScaler(), numeric_cols + rule_cols),
 25                 ('cat', OneHotEncoder(), categorical_cols),
 26             ], verbose=True
 27         )
```

Image 5

Data pre-processing for Predicting Customer Lifetime Value

Before moving on to the script where the prediction models are implemented, as shown in Image 6, an important step involves aggregating the data by customer. This aggregation ensures that each row in the dataset represents an individual customer. The methods used

for aggregation are crucial for preparing the data for accurate and effective predictive modelling.

The methods used for aggregation:

- First: Selects the first entry found.
- Sum: Sums up the column.
- Mean: Calculates the mean for the column.

The creation of the column “Has_Rule” is for storing a binary indicator if the customer is associated with any of the rules generated in the MBA. It uses the .max() because the rule columns contains 0 or 1 depending on whether a rule is associated with the row.

```
    # Identify all rule columns
    rule_columns = [col for col in data.columns if col.startswith('Rule_')]

    # Define aggregation methods
    aggregation_methods = {
        'Country': 'first',
        'Quantity': 'sum',
        'TotalSpend': 'sum',
        'R_Score': 'mean',
        'F_Score': 'mean',
        'M_Score': 'mean',
        'RFM_Level': 'first',
        'Month': 'first',
        'DayOfWeek': 'first',
        'TimeOfDay': 'first'
    }

    # Add rule columns to the aggregation methods
    # Using 'max' to see if any transactions matches
    # the rule.
    for rule in rule_columns:
        aggregation_methods[rule] = 'max'

    # Including any other columns that should be retained
    other_columns = [
        col for col in data.columns if col not
        in aggregation_methods and col not in
        rule_columns and col != 'Customer ID']
    for col in other_columns:
        aggregation_methods[col] = 'first'

    # Aggregating data
    aggregated_data = data.groupby(
        'Customer ID').agg(aggregation_methods).reset_index()

    # Create a new column to indicate if any rules are connected to customer
    aggregated_data['Has_Rule'] = aggregated_data[rule_columns].max(axis=1)
```

Image 6

Image 7 shows how the data was split into training and validation data sets. Her a cut-off date is set at December 1. By visually inspecting the two original datasets the overlap seems to start at this date.

```
    data = data.copy(deep=True)

    cutoff_date = pd.to_datetime('2010-12-01')
    train_data = data[data['InvoiceDate'] < cutoff_date]
    validation_data = data[data['InvoiceDate'] >= cutoff_date]

    return train_data, validation_data

```

Image 7

The last step before saving the split data into two files is to ensure that the validation set includes only customers present in both years of data (both original files). This process is shown in Image 8.

```
    # Rounding decimals to 2
    numeric_cols_train = train_agg_data.select_dtypes(
        include=['number']).columns
    train_agg_data.loc[:, numeric_cols_train] = train_agg_data[
            numeric_cols_train].round(
        2)
    numeric_cols_val = validation_agg_data.select_dtypes(
        include=['number']).columns
    validation_agg_data.loc[:, numeric_cols_val] = validation_agg_data[
            numeric_cols_val].round(
        2)

    # Identify common unique customers from both original datasets
    unique_customers_train = train_agg_data['Customer ID'].unique()
    unique_customers_validation = validation_agg_data['Customer ID'].unique()

    common_customers = set(unique_customers_train).intersection(
        unique_customers_validation)
    validation_agg_data = validation_agg_data[validation_agg_data[
        'Customer ID'].isin(common_customers)]

    # Save aggregated data
    train_agg_data.to_csv(f'{csv_dir}train_data.csv', index=False)
    validation_agg_data.to_csv(f'{csv_dir}validation_data.csv', index=False)

```

Image 8

The predictive models are found in script 2. Image 9 shows how the columns were selected for the model training and predictions. The target column: “Actual_CLV” is not included in this list. The steps here are:

- Calculate “Actual_CLV”, multiplying “TotalSpend” with 5. This could be any number, but 5 was chosen to represent 5 years of total customer spending. This is however based on one year of data. Two years could have been aggregated for those customers present in both years, but that would further limit the training data set. A decision was made to use the first year of data only. Another reason for this is to be able to directly compare the predicted CLV against the calculated Actual CLV using the same customers data for the validation dataset.

- Applying log transformation to “Price” and “Quantity” columns to normalize data distribution. By applying the logarithm to each data point, this method reduces the variability and compresses the range of values. It is particularly useful for datasets where the variance grows with the mean, making the data more symmetrical and improving the performance of regression models.

- Select the columns wanted to train the models.

```
    # Add actual_clv column (target label)
    train_data['Actual_CLV'] = train_data['TotalSpend'] * 5

    # Remove outliers
    train_data['Price_log'] = np.log1p(train_data['Price'])
    train_data['Quantity_log'] = np.log1p(train_data['Quantity'])

    validation_data['Price_log'] = np.log1p(validation_data['Price'])
    validation_data['Quantity_log'] = np.log1p(validation_data['Quantity'])

    # Target feature not included here.
    base_columns = [
        'RFM_Level', 'R_Score', 'F_Score',
        'M_Score', 'Price_log', 'Quantity_log',
        'Month', 'DayOfWeek', 'TimeOfDay',
        'Has_Rule'
    ]

    categorical_features = [
        'RFM_Level',
    ]

    rule_columns = [col for col in train_data.columns if 'Rule_' in col]

    feature_columns = base_columns + rule_columns
```

Image 9

```python
def preprocess_data(data, feature_columns, categorical_features):
    """Function for preprocessing of data.
    Returns preprocessor
    """
    numerical_features = [
        col for col in feature_columns if col not in categorical_features]

    preprocessor = ColumnTransformer(
        transformers=[
            ('scaler', StandardScaler(), numerical_features),
            ('cat', OneHotEncoder(), categorical_features)
        ], remainder='passthrough'
    )

    return preprocessor
```

Image 10

Image 10 shows the last step as far as pre-processing is concerned. The same steps were taken here as in the pre-processing before the kMeans clustering.

9. Measuring performance

kMeans clustering

For this model, the silhouette score metric was chosen to evaluate the clustering quality. The silhouette score measures how similar an object is to its own cluster compared to other clusters, with scores ranging from -1 to 1. Higher scores indicate well-defined clusters, while scores around 0 suggest overlapping clusters. This metric ensures meaningful and well-separated clusters.

The KMeans algorithm clusters data by separating samples into *n* groups of equal variance, minimizing the inertia or within-cluster sum-of-squares. This algorithm requires specifying the number of clusters and scales well to large datasets, making it widely applicable across various fields. (SciKit-Learn, D. 2024c)

Regression Models Evaluated:

- Linear Regression
- Random Forest Regressor

- Gradient Boosting Regressor.

Independent vs dependent variables:
A variable in computer programming is an abstract storage location with an associated name that holds a value. It can store various types of data, such as strings, integers, and Booleans. Variables have a scope that determines their accessibility within a program. In machine learning, variables correspond to columns in a dataset. Independent variables (features) are the inputs controlled in an analysis, while dependent variables are the outputs that change in response to the independent variables.

For regression tasks, accuracy is not measured as in classification tasks. Regression is different from classification, which involves predicting a category or a class label. Instead, we measure how close predictions are to actual values using the following metrics:

- Mean Squared Error (MSE): represents the mean of the squared differences between predicted and actual values. It measures the average squared difference, with a lower MSE indicating better model performance, as predictions are closer to the actual values. This makes MSE useful for identifying models with larger errors due to its penalization of larger discrepancies. (SciKit-Learn, D. 2024f)

- R-squared (R^2): It indicates the proportion of variance in the dependent variable (y) explained by the independent variables in the model. This metric provides a measure of goodness of fit, showing how well the model is likely to predict unseen samples through the proportion of explained variance. (SciKit-Learn, D. 2024e)

- Mean Absolute Error (MAE): measures the average absolute difference between the actual and predicted values. It gives an idea of how much the predictions deviate from the actual values on average, with lower values indicating better performance. (SciKit-Learn, D. 2024d)

- Feature Importances: highlight significant features contributing to the prediction. They are inherently applicable to tree-based models and can be inferred from the

coefficients in linear models, although not done in this study as the linear regressor serves as a baseline model. Recursive Feature Elimination (RFE) selects features by recursively considering smaller sets. An external estimator assigns weights to features, and the least important features are pruned. This process repeats until the desired number of features is reached, refining the model by focusing on the most significant predictors. (SciKit-Learn, D. 2024b)

Other metrics exist but were not used in this study.

Why these metrics were chosen:

- Mean Squared Error (MSE): This method penalizes larger errors more than smaller ones by squaring the differences, making it useful for identifying models with a few large errors. It's widely used for its clarity in measuring model error and ease of computation. (SciKit-Learn, D. 2024f)

- R-squared (R^2): Measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It provides insight into the goodness of fit of the model, showing how well the model explains the variability of the response data around its mean. (SciKit-Learn, D. 2024e)

- Mean Absolute Error (MAE): Provides the average magnitude of errors in a set of predictions, without considering their direction (i.e., it treats all errors equally). It's straightforward to interpret because it gives a clear measure of the average prediction error in the same units as the response variable. (SciKit-Learn, D. 2024d)

- Feature Importances: In models like Random Forest and Gradient Boosting, feature importance scores indicate which features have the most influence on the predictions. This helps in understanding the model and selecting important features to improve model performance. (SciKit-Learn, D. 2024b)

10.Algorithms applied

10.1 Customer Segmentation:

- Consists of the clustering model applied in script 1.

10.2-4 Predictive Models:

- Consists of the predictive models in script 2.

10.1 k-Means Clustering

K-Means clustering is an unsupervised learning algorithm used to partition data into k clusters, where each data point belongs to the cluster with the nearest mean, or centroid. The algorithm works by minimizing the within-cluster sum-of-squares, known as inertia.

The process involves:

- Initialization: Choosing initial centroids, often using k-means++ to ensure they are far apart.
- Assignment: Assigning each data point to the nearest centroid.
- Update: Recalculating centroids as the mean of assigned points.

This iterative process continues until centroids stabilize. While K-Means scales well to large datasets and is widely applicable, it assumes clusters are convex and isotropic, which might not always be the case. Preprocessing steps like dimensionality reduction (e.g., PCA) can improve performance in high-dimensional spaces.

K-Means clustering can reveal valuable patterns in data, such as customer segments in marketing. Despite potential convergence to local minima, methods like k-means++

initialization help mitigate this by ensuring better starting points. The algorithm's flexibility and efficiency make it a powerful tool for various clustering tasks (SciKit-Learn, 2024c).

Why Selected: It was chosen to identify distinct customer segments based on their transactional and behavioural data. Clustering helps in understanding different customer groups and tailoring marketing strategies accordingly.

An example of how k-means clustering was applied to the data can be seen in Image 11. The hyperparameters are set as follows:

- n_clusters=3: This parameter specifies that the algorithm should attempt to find 3 distinct clusters in the data.
- random_state=1: This ensures that the results are reproducible when the same random state is used.
- verbose=1: This setting enables the algorithm to print progress updates and other relevant information to the terminal during execution.

```
    print("Starting clustering pipeline...")
    try:
        # Defining columns for different preprocessing.
        categorical_cols = ['Country']
        numeric_cols = ['Quantity', 'Price', 'Month', 'DayOfWeek',
                        'TimeOfDay', 'R_Score', 'F_Score', 'M_Score',
                        'TotalSpend']
        rule_cols = [col for col in data.columns if 'Rule_' in col]

        required_columns = numeric_cols + categorical_cols + rule_cols
        missing_columns = [
            col for col in required_columns if col not in data.columns]
        if missing_columns:
            print(f"Missing columns in kmeans:\n{missing_columns}")
            return None

        # Preproccessing pipeline
        preprocessor = ColumnTransformer(
            transformers=[
                ('num', StandardScaler(), numeric_cols + rule_cols),
                ('cat', OneHotEncoder(), categorical_cols),
            ], verbose=True
        )
        print("before fit_transform:\n", data.head())

        data_processed = preprocessor.fit_transform(
            data[numeric_cols + rule_cols + categorical_cols])

        # Clustering
        kmeans = KMeans(n_clusters=3, random_state=1, verbose=1)
        kmeans.fit(data_processed)
        cluster_labels = kmeans.labels_
        data['ClusterGroup'] = cluster_labels

        # Evaluating clusters
        print("Starting Evaluation using silhouette.")
        silhouette = silhouette_score(
            data_processed,
            cluster_labels,
            sample_size=70000,
            random_state=1,
            n_jobs=-1
        )
        print("Silhouette Score:", silhouette)
```

Image 11

Performance:

- Silhouette Score: This metric measures how similar an object is to its own cluster compared to other clusters. Values range from -1 to 1, with higher values indicating better-defined clustering. A silhouette score around 0.10, as shown in Image 12, suggests some overlap between clusters, but also indicates that distinct grouping is still present. This score helps evaluate the effectiveness of the clustering model in distinguishing between different clusters.

```
Silhoutte:
 0.10381205744143966

Clustering executed successfully.
```

Image 12

10.2-4 Overview

The system is designed to train all three models and automatically select the best one based on the Mean Squared Error (MSE) score before moving on to predict values in the validation dataset. The choice of which model to use can be set manually in the main function, with a default setting of None. Additionally, whether to use RandomSearch can also be configured in the main function, with a default setting of True. Image 13 illustrates the beginning of this function.

```
def run_models(
        data,
        feature_columns,
        categorical_features,
        use_randomsearch=True,
        best_model=None):
    """
```

Image 13

Further steps taken can be seen in image 14:

- Split into train and test data.
- Scale the target columns: y.
- Setup of preprocessor.
- Model setup.
- Check if model is passed as an argument to the function.
- Setup parameters for the applicable models.

```
    # Split the data
    X = data[feature_columns]
    y = data['Actual_CLV'].values.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=1)

    # Scale the target variable
    y_scaler = StandardScaler()
    y_train_scaled = y_scaler.fit_transform(y_train)
    y_test_scaled = y_scaler.transform(y_test)

    # Preproccess the features
    preprocessor = preprocess_data(data, feature_columns, categorical_features)

    # Define models
    models = {
        'Linear Regression': LinearRegression(n_jobs=-1),
        'Random Forest': RandomForestRegressor(
            random_state=1,
            n_jobs=-1
        ),
        'Gradient Boosting': GradientBoostingRegressor(
            random_state=1,
        )}

    if best_model:
        models = {best_model: models[best_model]}

    # Hyperparameter grids
    param_grids = {
        'Random Forest': {
            'model__n_estimators': [200, 250, 300],
            'model__min_samples_split': [2, 3, 4],
            'model__max_depth': [5, 10, 15],
        },
        'Gradient Boosting': {
            'model__n_estimators': [150, 200, 250, 300, 350],
            'model__min_samples_split': [2, 3, 4, 5],
            'model__max_depth': [4, 5, 6],
            'model__learning_rate': [0.04, 0.05, 0.09, 0.1, 0.11],

        }
    }
```

Image 14

The next steps are shown in Image 15:

- Check if model exists, load it if yes.
- Check feature importance and move on to setup pipeline if no.

```
    results = []
    # Pipeline setup and running models
    for name, model in models.items():
        model_path = f'{model_dir}{name}.pkl'
        if os.path.exists(model_path):
            with open(model_path, 'rb') as f:
                pipeline, y_scaler = pickle.load(f)
            print(f"Loaded {name} and scaler from pickle.\n")
        else:
            preprocessor.fit(X_train)
            X_train_transformed = preprocessor.transform(X_train)

            # Get feature names after preprocessing
            num_feature_names = preprocessor.transformers_[0][2]
            cat_feature_names = preprocessor.transformers_[
                1][1].get_feature_names_out(categorical_features).tolist()
            transformed_feature_names = num_feature_names + cat_feature_names

            # Define the RFE(Recursive Feature Elimination) step
            rfe = RFE(estimator=model, n_features_to_select=10, step=1)
            rfe.fit(X_train_transformed, y_train_scaled.ravel())

            # Print the 10 best features selected by RFE
            selected_features = [feature for feature, support in zip(
                transformed_feature_names, rfe.support_) if support]
            print(
                f"Top 10 features selected by RFE for {name}: {selected_features}")
            selected_features = pd.DataFrame(selected_features)
            selected_features.to_csv(
                f"{csv_dir}{name}_rfe_selected_features.csv", index=False)
            print(f"Saved top 10 features selected by RFE for {name} to csv!")

            pipeline = Pipeline([
                ('preprocessor', preprocessor),
                ('rfe', rfe),
                ('model', model)
            ])
```

Image 15

```
        if use_randomsearch and name in param_grids:
            random_search = RandomizedSearchCV(
                pipeline, param_grids[name],
                cv=5, scoring='neg_mean_squared_error',
                n_jobs=-1)

            random_search.fit(X_train, y_train_scaled.ravel())
            pipeline = random_search.best_estimator_
            best_params = random_search.best_params_
            try:
                best_params_df = pd.DataFrame([best_params])
                best_params_df.to_csv(
                    f"{csv_dir}best_parameters_{name}.csv")
            except Exception as e:
                print(f"Saving csv failed..{e}")

            print(
                f"Best params for {name}: {random_search.best_params_}")

        else:
            pipeline.fit(X_train, y_train_scaled.ravel())

        with open(model_path, 'wb') as f:
            pickle.dump((pipeline, y_scaler), f)
        print(f"Saved {name} model and scaler to pickle.")

    # Predictions and evaluations
    y_pred_scaled = pipeline.predict(X_test)
    y_pred = y_scaler.inverse_transform(
        y_pred_scaled.reshape(-1, 1)).flatten()
    y_test_original = y_scaler.inverse_transform(
        y_test_scaled.reshape(-1, 1)).flatten()
```

Image 16

In image 16 the following takes place:

- If RandomSearchCV is set to True, run RandomSearchCV. This will check what the best hyperparameters are. Then it will fit the model using the best hyperparameters found.
- If set to False in main function, move on to fit(train) the model without using RandomSearch, default internal hyperparameters will then be used.
- Save the model and scaler to pickle.
- Predict and evaluate on the test set.

```
        mse = mean_squared_error(y_test_original, y_pred)
        mae = mean_absolute_error(y_test_original, y_pred)
        r2 = r2_score(y_test_original, y_pred)
        result = {'Model': name,
                  'MSE': mse,
                  'R-squared': r2,
                  'MAE': mae
                  }

        # Feature importances for applicable models
        if hasattr(pipeline.named_steps['model'], 'feature_importances_'):
            feature_importances = pipeline.named_steps['model']\
                .feature_importances_
            feature_names = [col for col in feature_columns
                             if col not in categorical_features] + \
                list(pipeline.named_steps['preprocessor'].named_transformers_[
                    'cat'].get_feature_names_out())
            result['Feature Importances'] = dict(
                zip(feature_names, feature_importances))

        results.append(result)

    training_results_df = pd.DataFrame(results)
    columns_to_print = ['Model', 'MSE', 'R-squared', 'MAE']
    print("\nTest results DF:\n", training_results_df[columns_to_print])
    training_results_df.to_csv(
        f"{csv_dir}model_evaluation_metrics.csv", index=False)
    print(
        f"\nModel eval metrics saved to {csv_dir}model_evaluation_metrics.csv")

    # Identify the best model based on Test MSE
    best_model_row = training_results_df.loc[training_results_df['MSE'].idxmin(
    )]
    best_model = best_model_row['Model']
    print(f"Best model based on Test MSE: {best_model}")

    for index, row in training_results_df.iterrows():
        if row['Model'] in ['Random Forest', 'Gradient Boosting']:
            plot_feature_importances(
                row['Feature Importances'],
                f"{row['Model']} Feature Importances")

    return training_results_df, best_model
```

Image 17

The final steps in the run_model function (Image 17):

- Measure MSE
- Measure MAE
- Measure R^2-Score
- Extract feature importances
- Print test results to the terminal, and save them to csv.
- Return the training dataframe and the best_model.

Validation

The validation dataset consists of “new unseen” data for the model to make predictions on. In this case, it is data from the second year of the original dataset used in this study. Image 18 demonstrates the function that predicts Customer Lifetime Value (CLV) using the validation dataset as the input.

```
def make_predictions(data, feature_columns, best_model):
    """
    Make predictions using the pre-trained best model and inverse
    transform the scaled predictions.

    Args:
        data (pd.DataFrame): The input data.
        feature_columns (list): List of feature columns to be used
        for making predictions.
        best_model_name (str): The name of the best model to be used for predictions.

    Returns:
        pd.DataFrame: DataFrame with predictions added.
    """

    model_path = f'{model_dir}{best_model}.pkl'
    if os.path.exists(model_path):
        with open(model_path, 'rb') as f:
            pipeline, y_scaler = pickle.load(f)
        print(f"Loaded {best_model} and scaler from pickle.\n")

        X = data[feature_columns]
        y_pred_scaled = pipeline.predict(X).reshape(-1, 1)
        y_pred = y_scaler.inverse_transform(y_pred_scaled).flatten()

        # Clip negative predictions
        y_pred = [max(0, pred) for pred in y_pred]

        data["CLV_Predictions"] = y_pred

    return data
```

Image 18

Image 19 shows how to set model manually, and if RandomSearchCV should be enabled.

```
    # Initialize model manually, None selects the best model automatically.
    # Uncomment the model wanted to run only that.
    model = None
    # model = 'Linear Regression'
    # model = 'Random Forest'
    # model = 'Gradient Boosting'

    # Run models with or without RandomSearchCV.
    # RandomSearchCV has an integral Cross-Validation.
    # Set use_randomsearch to True to use it.
    training_results_df, model = run_models(
        train_data, feature_columns, categorical_features,
        use_randomsearch=True, best_model=model)

    # Predict Customer Lifetime Value
    predictions = make_predictions(
        validation_data, feature_columns, model)
```

Image 19

10.2 Linear Regression

Linear regression is a fundamental technique in machine learning used for predictive analysis and solving regression problems. It models the relationship between a dependent variable and one or more independent variables using a linear equation, assuming a linear relationship between the input and output variables.

Key Assumptions of Linear Regression:

- Linearity: The relationship between the dependent variable (y) and independent variables (x) must be linear. A scatter plot can be used to verify this.
- Homoscedasticity: The spread of residuals (errors) should be constant across all values of the independent variables. If the residuals form a consistent pattern, rather than a funnel shape, this assumption holds.
- Normality: Both the independent and dependent variables should be normally distributed, which can be checked using histograms and Q-Q plots.

- No Multicollinearity/Independence: Independent variables should not be highly correlated with each other. This can be checked using a correlation matrix or the Variance Inflation Factor (VIF) score, where a VIF above 5 indicates high correlation.

Application and Evaluation:

Linear regression is used to understand and predict the relationship between continuous variables. It helps in uncovering how one or more independent variables influence a dependent variable. Evaluating regression analysis involves checking these assumptions and using various performance metrics to ensure the model's reliability and accuracy.

This straightforward yet powerful algorithm is widely used across different fields for its ability to model complex relationships with relatively simple implementation and interpretation.

Why Selected:

It serves as a baseline model due to its simplicity, ease of interpretation, and quick implementation. It helps to set a benchmark to compare more complex models. The performance of the model can be seen in table 2.

Example of how the 3 different models were applied to the data can be seen in Images 13-18.

Performance (Test metrics):

Model	MSE	R-squared	MAE
Linear Regression	4247895166.41	0.15	10379.19

Table 2

10.3 Random Forest Regressor

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mean prediction of the individual trees. It operates by creating a 'forest' of trees, each trained on a random subset of the data and features.

In Random Forests, each tree is built from a bootstrap sample drawn with replacement from the training set. When splitting each node, the best split is found through an exhaustive search of the feature values, either from all input features or a random subset of size max_features. This randomness helps to decrease the variance of the forest estimator, addressing the high variance and tendency to overfit seen in individual decision trees.

By combining diverse trees, Random Forests reduce variance, sometimes at the cost of a slight increase in bias. The scikit-learn implementation of Random Forests combines classifiers by averaging their probabilistic predictions, yielding an overall better model. This method is effective in practice due to its significant variance reduction.

Comparatively, Histogram-Based Gradient Boosting (HGBT) models offer an alternative with efficient binning algorithms and sequential boosting to correct errors iteratively. HGBT models often build shallow trees, which are faster to fit and predict, whereas Random Forests rely on deep trees, requiring more computational resources.

In summary, Random Forests are robust and versatile, performing well across a range of applications by averaging multiple decision trees to improve prediction accuracy and reduce overfitting (SciKit-Learn, 2024a).

Why Selected:
It reduces overfitting by averaging multiple decision trees and can handle large datasets with higher dimensionality. Random Forest is robust to noise and can model complex relationships in the data. Performance can be seen in table 3.

Performance (Test metrics):

Model	MSE	R-squared	MAE
Random Forest	2456354735.16	0.51	4815.68

Table 3

Best features selected:

R_Score	0.02285
F_Score	0.02605
M_Score	0.06383
Price_log	0.01226
Quantity_log	0.80665
Month	0.00658
DayOfWeek	0.01429
TimeOfDay	0.00936
Has_Rule	0.01853
Rule_6	0.01958

Table 4

The numbers in table 4 shows the scores given to the features by the model. It is setup to use the 10 best features avaliable.

10.4 Gradient Boosting Regressor

Gradient Boosting is a sequential ensemble learning technique used to build a strong predictive model by combining multiple weak learners, usually decision trees. Each new model in the sequence attempts to correct the errors of the previous models, minimizing the loss function.

Key Steps in Gradient Boosting:

- Initialization: All data points are assigned equal weights initially, and a subset of the dataset is used to create a base model.
- Error Correction: The base model makes predictions on the entire dataset, and the errors are calculated by comparing actual and predicted values.
- Weight Adjustment: Observations that are incorrectly predicted are assigned higher weights.
- Subsequent Models: New models are created sequentially, each focusing on correcting the errors of the previous model.
- Final Model: The final strong learner is the weighted mean of all preceding models (weak learners).

Gradient Boosting works by building a series of regression trees, where each subsequent tree in the series is built on the errors calculated by the previous tree. This iterative process continues until the model's performance is optimized. The technique is effective because it focuses on improving areas where previous models performed poorly, ultimately boosting the overall predictive power of the ensemble.

Why Selected:
It excels in capturing complex patterns in data and often provides higher accuracy compared to other models. Gradient Boosting is particularly effective for problems with non-linear relationships and interactions between features. Table 5 shows the performance.

Performance (Test metrics):

Model	MSE	R-squared	MAE
Gradient Boosting	2374825324.38	0.53	4493.69

Table 5

R_Score	0.02434
F_Score	0.00604
M_Score	0.04923
Price_log	0.00746
Quantity_log	0.77597
Month	0.00272
DayOfWeek	0.03733
TimeOfDay	0.07969
Has_Rule	0.01698
Rule_6	0.00024

Table 6

In table 6 the numbers are the scores given by the model when selecting the 10 best features, similar to the setup in Random Forest.

10.5 Comparison

To compare which model is the best one, we should look more closely at the metrics. In the current setup the MSE is used to select the best model. The reason for this is that it penalizes large error more heavily. Table 7 shows the metrics selected for comparison of the models.

Model	MSE	R-squared	MAE
Linear Regression	4247895166.41	0.15	10379.19
Random Forest	2456354735.16	0.51	4815.68
Gradient Boosting	2374825324.38	0.53	4493.69

Table 7

MSE: The lowest MSE indicates that Gradient Boosting has the smallest average squared error, meaning its predictions are closest to the actual values.

R-squared: With an R^2 of 0.53, this model explains 53% of the variance in the data, which is a relatively good fit.

MAE: The lowest MAE of 4493.69 suggests it has the smallest average error magnitude, further confirming its accuracy.

For the feature importances the results can be seen in table 8. As we can see, both RandomForest and Gradient Boosting selected the same features as the most important ones.

RandomForest	Score	Gradient Boosting	Score
R_Score	0.02285	R_Score	0.02434
F_Score	0.02605	F_Score	0.00604
M_Score	0.06383	M_Score	0.04923
Price_log	0.01226	Price_log	0.00746
Quantity_log	0.80665	Quantity_log	0.77597
Month	0.00658	Month	0.00272
DayOfWeek	0.01429	DayOfWeek	0.03733
TimeOfDay	0.00936	TimeOfDay	0.07969
Has_Rule	0.01853	Has_Rule	0.01698
Rule_6	0.01958	Rule_6	0.00024

Table 8

Best model: Gradient Boosting Regressor.

11.Discussion

This study's findings can significantly impact marketing strategies by targeting customer groups based on cluster groups or RFM levels and providing insights into customer lifetime value (CLV). The analysis results are generated through scripts, which must be run sequentially (script 1 followed by script 2) with the relevant CSV files in the same directory. The true power of these insights emerges when the findings are combined, offering a comprehensive understanding of customer behavior and enhancing strategic decision-making.

Findings:

- Association rules
- Customer segments based on RFM
- Customer segments based on clustering
- Predicted customer lifetime value

Association Rules

The top ten association rules found in the data can be seen in Plot 1. Based on this data, items can be suggested to customers for cross-selling opportunities. These rules are generated using the Apriori analysis method, an algorithm for finding boolean association rules by mining frequent itemsets. We can think of it this way: in a spreadsheet, all items purchased will be marked with 1, while all items not purchased will be marked with 0.

An example of an association rule is:
Item1 => Item2 [support = 1% (0.01), confidence = 67% (0.67), lift = 105% (1.05)]

When identifying these patterns, we need to measure “interestingness.” These can be quantified using support, confidence, and lift:

- Support: Indicates the percentage of transactions where item1 and item2 are purchased together.
- Confidence: Indicates the percentage of customers who purchased item1 and also purchased item2.
- Lift: Measures how much the occurrence of both items together exceeds what would be expected if they were independent.

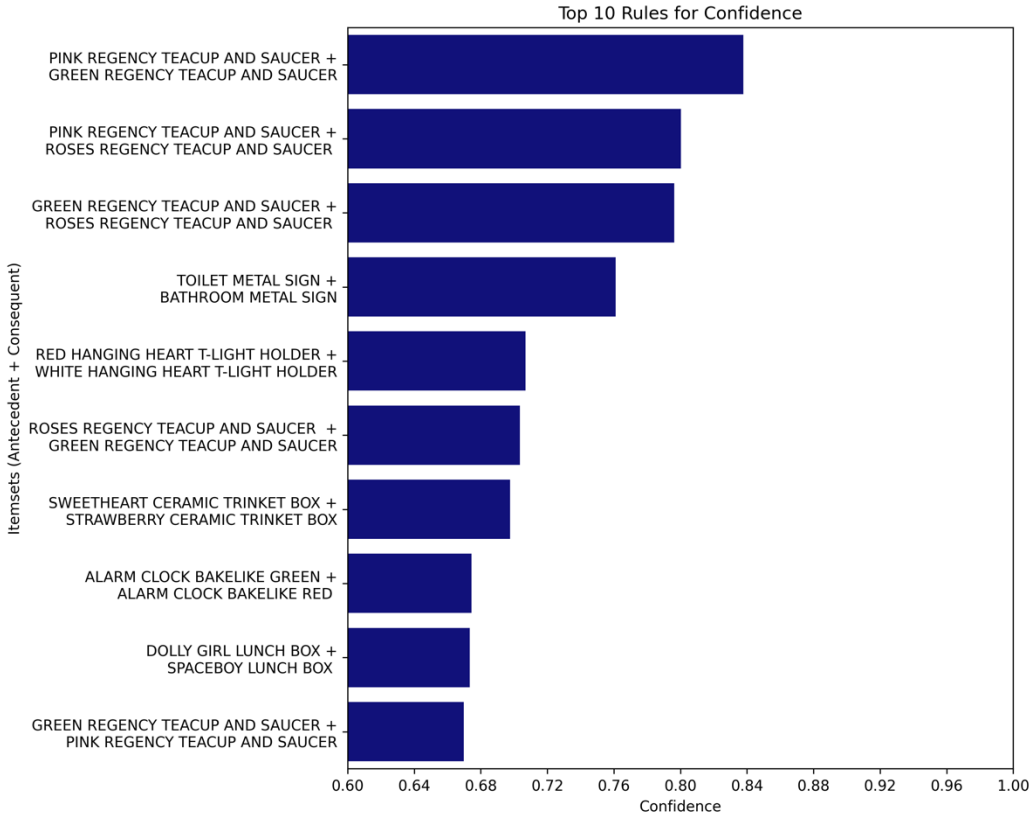
The name “apriori” comes from the algorithm’s use of prior knowledge of frequent itemset properties. It starts by identifying the smallest frequent itemsets that exceed a minimum support threshold, known as L1. This set is then used to find larger itemsets, L2, and so forth, up to Lk. Each new level of itemsets, Lk, is used to find the next level, Lk+1, by joining itemsets from Lk-1. These are referred to as candidate itemsets, Ck. During this step, pruning is done to narrow down the itemsets to those with a support count higher than the minimum required for Lk.

The set of rules can be found in “confidence_rules.csv.” These rules have a minimum support of 0.015 and a confidence threshold set to 0.6.

An itemset consists of an antecedent and a consequent, both of wich are a list of items. (Garg, A. 2019)

A simple example of this:

{Bread, Egg}=Antecedent => {Milk}=Consequent
Itemset = {Bread, Egg, Milk}



Plot 1

The file “clustered_data.csv” (not aggregated by customer) shows which transactions are part of any Apriori rules found. Details of these transactions can be seen in Table 9.

Customer segments based on RFM

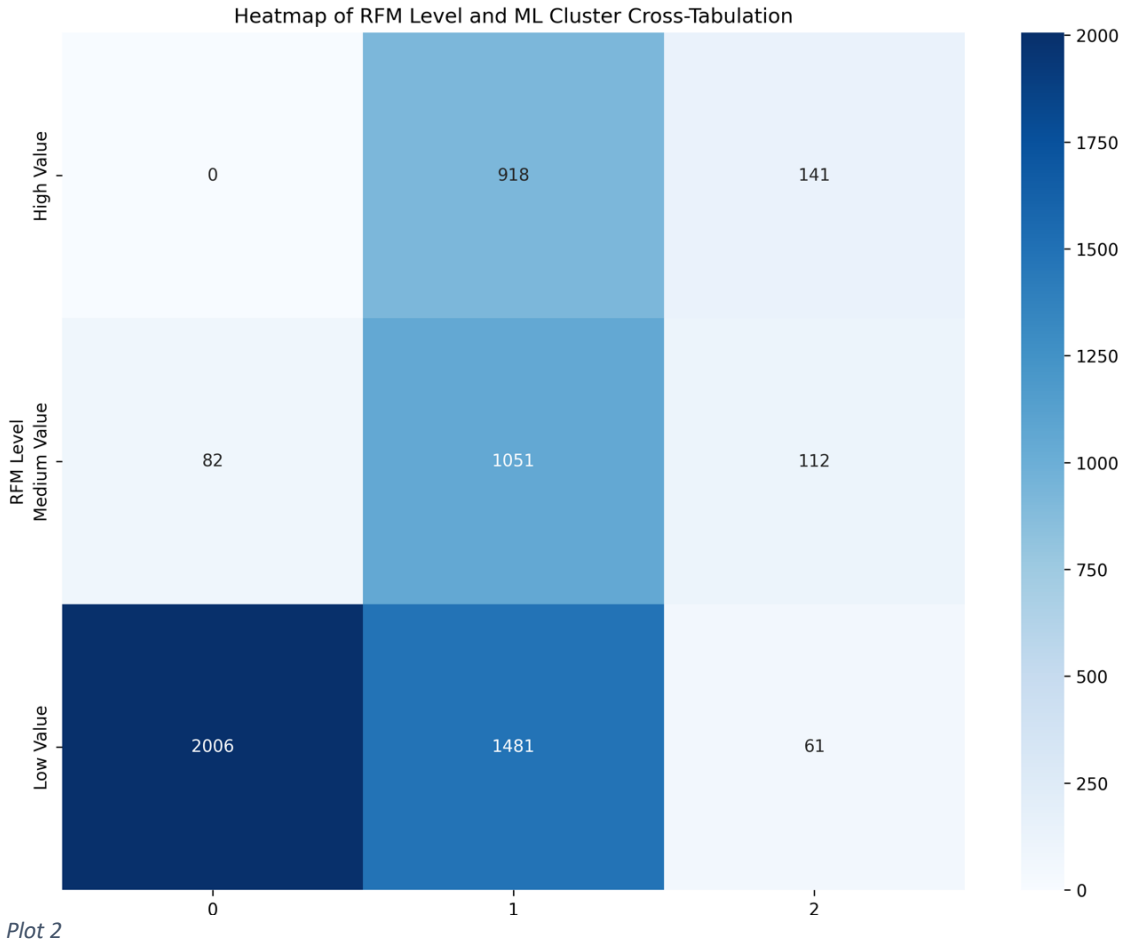
The complete findings can also be seen with the entire dataset in “clustered_data.csv.” Table 9 shows some of the columns present in this file. The segments derived from this analysis indicate how recent (recency), how often (frequency), and how much (monetary) each customer interacts with the company. The RFM Level was determined by taking the average of the RFM Scores for each customer and mapping them into three bins: “Low value,” “Medium value,” and “High value.” The RFM metrics are important indicators of a customer’s behavior because frequency and monetary value affect a customer’s lifetime value, while recency affects retention, a measure of engagement (Makhija, 2024).

Quantity	Price	TotalSpend	R_Score	F_Score	M_Score	RFM_Level	ClusterGroup	Rule_0	Rule_10
74285	4.5	77556.46	326.0	34.0	207.4	Low Value	1	0	0
2940	2.95	4851.48	2.0	218.0	541.67	High Value	2	1	1
2704	0.55	1658.4	75.0	46.0	32.1	Low Value	1	0	0
1621	1.65	3678.69	19.0	172.0	730.44	High Value	1	0	0

Table 9

Customer segments based on clustering

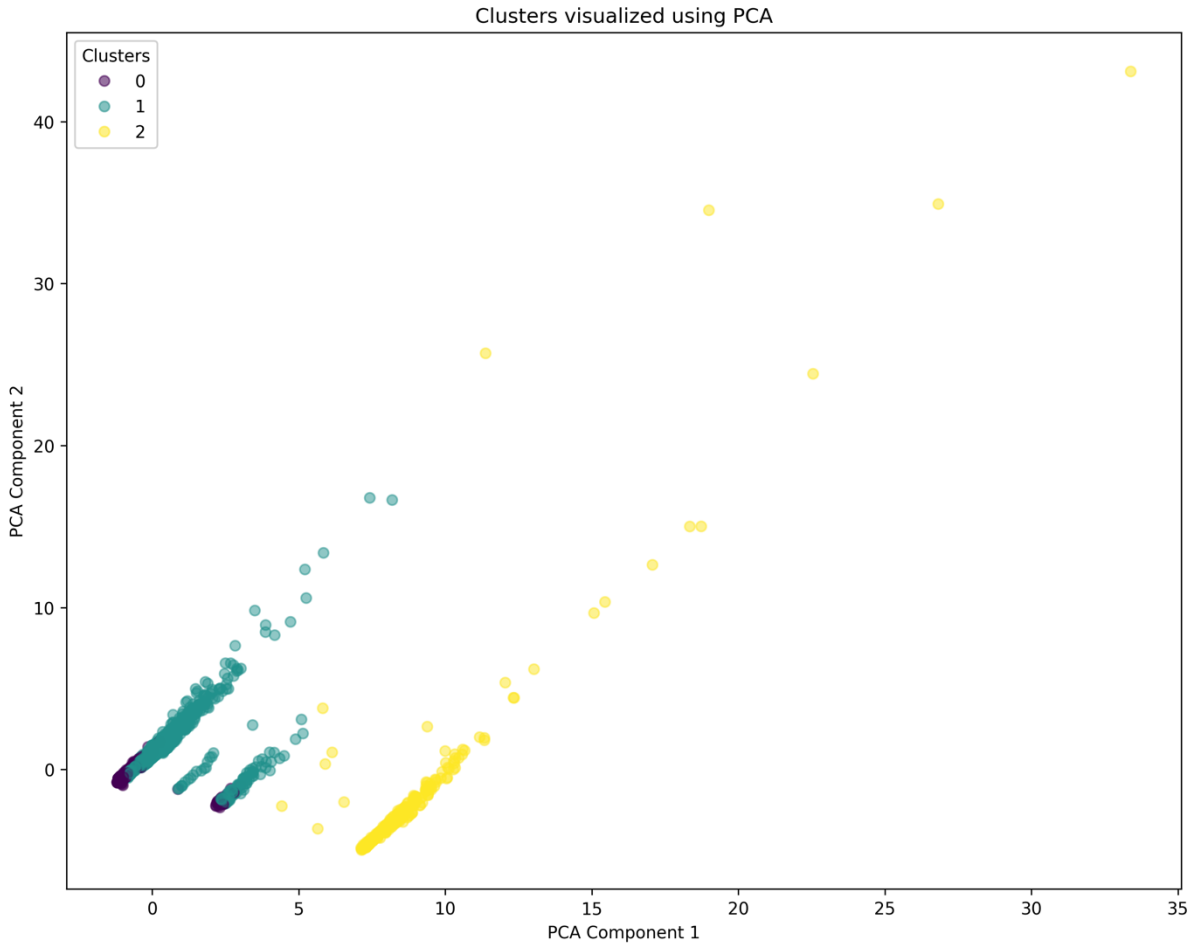
The cluster groups can be seen in table 9. Plot 2 displays the cluster groups along with the RFM Levels.



Plot 2

From the plot, one can observe how many customers from each RFM group belong to each cluster. This can provide insights into customer churn, for example. Notably, there are quite a few customers in the three clusters labeled as Low Value in their RFM Level. This information should be further investigated to develop strategies accordingly. The data can be used to target these customers with tailored marketing specifically designed for them, which could lead to retaining these customers instead of losing them.

The same principle can be applied to other customer segments as well. This knowledge can be leveraged to create marketing campaigns specifically for the appropriate customers. Plot 3 displays the clusters found in the data. Although the clusters show some overlap, they still provide valuable new information that can be used in conjunction with the RFM-segmented customers.

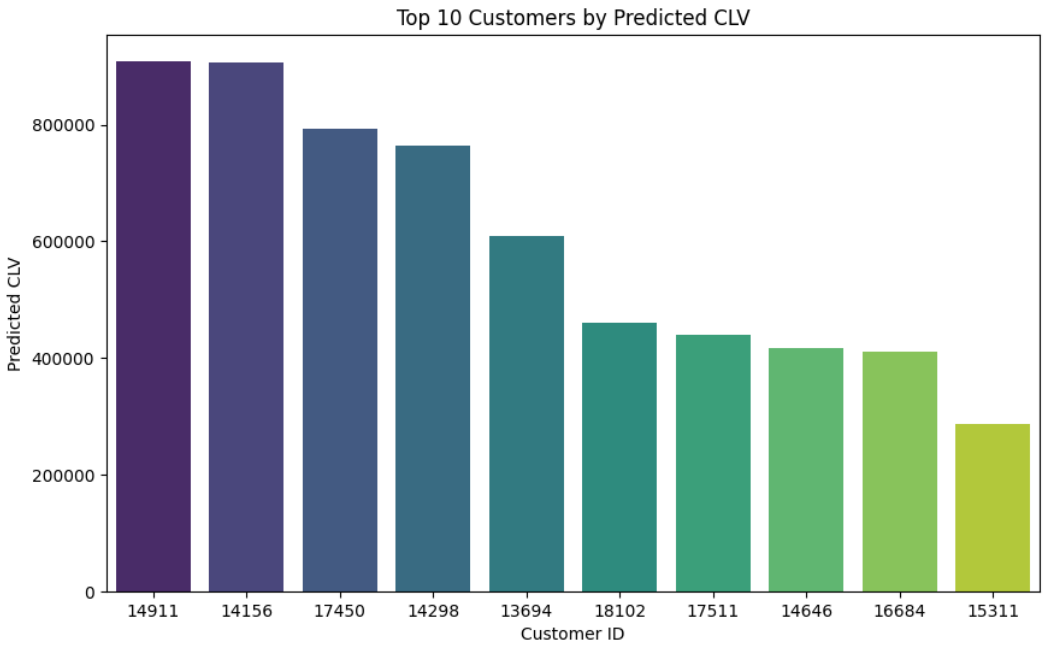


Plot 3

Predicted customer lifetime value

Predicting customer lifetime value (CLV) is crucial for understanding how much revenue a customer will generate for the company over their lifetime. In this study, the “Actual CLV” and predicted CLV are based on a 5-year period. This insight allows the business to make informed decisions about customer acquisition, retention strategies, and resource allocation. By accurately predicting CLV, the company can tailor marketing efforts to high-value customers, optimize customer service, and enhance overall profitability. The model leverages historical data and various features such as purchase frequency, monetary value, and recency to forecast future contributions to the company’s revenue stream.

Plot 4 shows the top 10 customers based on predicted CLV.



Plot 4

The data for these findings can be found in the file “customer_insights_CLV.csv”. Table 10 shows some of the rows from this file.

Customer ID	Country	R_Score	F_Score	M_Score	RFM_Level	Has_Rule	Quantity	Price	Actual_CLV	CLV_Predictions
12346	United Kingdom	326.0	34.0	207.4	Low Value	0	74215	1.04	385918.0	211016.91
12347	Iceland	2.0	218.0	541.67	High Value	1	2449	2.1	21363.25	19271.18
12348	Finland	75.0	46.0	32.1	Low Value	0	2332	0.55	7186.2	8526.34
12349	Italy	19.0	172.0	730.44	High Value	0	630	7.5	7287.75	6607.12
12352	Norway	36.0	95.0	368.41	High Value	0	526	2.95	6928.7	5727.33
12353	Bahrain	204.0	24.0	63.08	Low Value	0	20	9.95	445.0	508.63

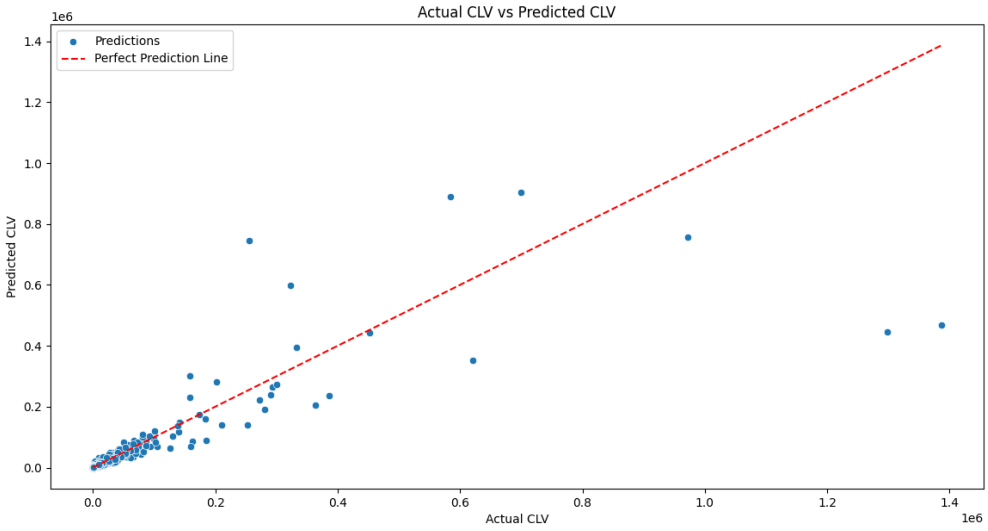
Table 10

An important note regarding the predicted values:

The “Actual_CLV” column holds values calculated by multiplying the price by the quantity for each customer. However, the models have been trained on a broader set of features including RFM_Level, R_Score, F_Score, M_Score, Price_log, Quantity_log, Month, DayOfWeek, TimeOfDay, Has_Rule, and all the “rules” columns generated from the Apriori algorithm. These features capture a variety of nuances in customer behaviour and interactions. Therefore, directly comparing the “Actual_CLV” column with “CLV_Predictions” is not appropriate, as the model uses complex patterns in the data to predict CLV. This complexity is why we train models rather than relying on manual calculations. The “Actual_CLV” is included to display the impact of the model predictions.

The predictions on the validation dataset can be seen in Plot 5. The red line in the plot represents the “perfect prediction” from the “Actual_CLV,” so a strict comparison should not be done. Nevertheless, it shows that predictions are indeed mostly gathered around the “perfect prediction,” especially for lower values. The higher the values, the more they deviate. This is expected, given that most data points the model has been trained on are in the lower

range. There are fewer high values in the data compared to the lower ones. A decision was made not to limit outliers in this prediction task, as it would severely reduce the amount of data available.



Plot 5

12.Conclusion

The objectives of the study:
“Enhance customer insights and sales strategy through market basket analysis and customer segmentation in online retail.”

This project has been a highly educational journey, and I feel that the objectives of the study have been met to a strong degree. Throughout the project, valuable customer insights have been gained. Some of these insights were obtained through strong association rules generated by applying the Apriori algorithm, and others through the grouping (segmenting) of customers.

For the sales strategy part, the focus was not on creating the strategies themselves but rather on providing the data needed to formulate them. The strategies can utilize the findings of the rules where itemsets are frequently bought together. By combining these rules with the correct customer segments, it is likely to result in a highly efficient strategy targeted at the appropriate customers.

Additionally, using the data from the RFM analysis, the retail company can easily identify customers likely to churn or high-value customers who might spend even more if suggested additional products based on association rules. This approach can significantly increase revenue.

The customer lifetime value (CLV) predictions will also help the business to plan better for the future, as they can now more accurately predict customer spending over a five-year period. This foresight enables better resource allocation, targeted marketing efforts, and overall improved business planning.

Reflections

The most important lesson learned through this project has been the necessity of thoroughly planning before implementing a solution. This lesson applies especially to the Python scripts written but also to the overall project. On multiple occasions, I had to revisit and redo parts of the project due to inadequate upfront planning. This is a valuable lesson for future projects.

One specific example was forgetting to multiply the quantity by the price per transaction before aggregating customer data. This oversight led to models predicting unrealistically high values, and it took a long time to identify the root cause. Had I thought this through in detail beforehand, I could have saved many hours spent on model hyperparameter tuning to achieve realistic predictions. Lesson learned.

Throughout the project, I often found that errors in “easy” tasks were causing problems in “hard” tasks. This highlighted the importance of paying close attention to the basics early on to avoid errors later. For example, neglecting simple calculations or data manipulations led to significant issues in model accuracy and performance. Lesson learned.

Another issue was the selection of features or columns to retain for further analysis. Initially, I did not think this through, resulting in unnecessary columns such as “StockCode” and “StockCodeVariation” being included in the data processing. These columns did not contribute to the study’s objectives and should have been excluded from the start. Lesson learned.

Additionally, I realized that the study’s objectives might have been too broad. This left me with insufficient time to delve into each topic as deeply as I would have liked. For instance, I felt constrained in the k-means clustering part of the study, where the presence of overlapping clusters bothered me. I am still uncertain whether I used the best visualization methods for these clusters. Lesson learned.

Despite these challenges and setbacks, the project has left me with a confident feeling of “I can do this,” which is a very positive outcome.

References

Brenner, M. (2023) *Online retailing*, *Adogy*. Available at: https://www.adogy.com/terms/online-retailing/ (Accessed: 24 May 2024).

Garg, A. (2019) *Complete Guide to Association rules (1/2)*, *Medium*. Available at: https://towardsdatascience.com/association-rules-2-aa9a77241654 (Accessed: 30 May 2024).

Jiawei Han, Jian Pei, Hanghang Tong (2022) *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems) 4th Edition*

Jolaoso, C. (2024) *Customer segmentation: The ultimate guide*, *Forbes*. Available at: https://www.forbes.com/advisor/business/customer-segmentation/ (Accessed: 24 May 2024).

Makhija, P. (2024) *RFM analysis for Customer Segmentation*, *CleverTap*. Available at: https://clevertap.com/blog/rfm-analysis/ (Accessed: 30 May 2024).

SciKit-Learn, D. (2024a) *1.11. ensembles: Gradient boosting, random forests, bagging, voting, stacking*, *scikit*. Available at: https://scikit-learn.org/stable/modules/ensemble.html#forest (Accessed: 30 May 2024).

SciKit-Learn, D. (2024b) *1.13. feature selection*, *scikit*. Available at: https://scikit-learn.org/stable/modules/feature_selection.html#rfe (Accessed: 31 May 2024).

SciKit-Learn, D. (2024c) *2.3. clustering*, *scikit*. Available at: https://scikit-learn.org/stable/modules/clustering.html#k-means (Accessed: 30 May 2024).

SciKit-Learn, D. (2024d) *3.4. metrics and scoring: Quantifying the quality of predictions*, *scikit*. Available at: https://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error (Accessed: 31 May 2024).

SciKit-Learn, D. (2024e) *3.4. metrics and scoring: Quantifying the quality of predictions*, *scikit*. Available at: https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score (Accessed: 31 May 2024).

SciKit-Learn, D. (2024f) *3.4. metrics and scoring: Quantifying the quality of predictions*, *scikit*. Available at: https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error (Accessed: 31 May 2024).

Appendix

Log while working on the exam project.

22.04.24

Started examining the datasets I've chosen. A decision was made to concatenate the two separate years into one csv file for easier handling. 2009-2010 and 2010-2011. There were some overlaps in the files, so I ran drop duplicates on it after the concatenation.

23.04.24

Examining the dataset. I found that some of the stock codes contains letters in addition to a 5-digit number. From what I see it's used for variations within the same product.

The plan is to split the letter/number combination and create a new column containing the letter. Name of column should be StockCodeVariation. I am going to compare the models with or without these letters. One-hot encoding might be applied here.

Another issue I've found is the term 'POST' in the stock code column. This needs to be addressed properly. For market basket analysis, it is not required and can be dropped from the dataset, but for the customer segmentation part it could provide more insight into customer behavior.

24.04.24

Day spent thinking about how to proceed. How should the data be dealt with.

25.04.24

Examining the dataset further. I found that the invoice column as well holds the letter 'C' on some occasions. From Kaggle.com, where I found the dataset, this is a code for cancelled orders. A decision was made to split this as well. The 'C' will be put into its own column.

Today I found out there were quite many missing entries of customer ID. Of 1 033 036 rows, there are 235 151 rows with missing customer Id data. Since there are a significant amount of data, I decided to set missing customer ID to -1. It could be that there has been a malfunction in the logging of customer ID in the system that exported the data, or it could be anonymous transactions. Either way I want to keep the data and try to work around it.

When splitting the StockCode into letters and numbers, some entries which had POST as the code were moved to the StockCodeVariation column, this was expected, but left 5472 cells with NaN. A decision was made to fill these with 0.

Dates were converted to datetime as well.

26.04.24

The description column also had 4275 missing values. When checking the 'StockCode' column visually in excel, I found that the 'StockCode' elsewhere held the 'Description'. From this I created a dictionary from 'StockCode' and 'Description' and filled in the missing in 'Description'. After this there are still 267 missing in 'Description'. Since this is a very small number in comparison to the complete dataset, I remove all 267 rows with missing data. The columns 'StockCodeVariation' and 'CancelledOrders' I filled missing values with 'None'.

I now consider dealing with missing values complete. I have also altered the columns to my liking in terms of placement.

29.04.24

Started looking into what ML-algorithm should be used to perform MBA (Market Basket Analysis). On this site I found information regarding this: https://www.turing.com/kb/market-basket-analysis#

There are three types of MBAs. Descriptive, predictive and differential MBA. I will focus on Descriptive MBA to gain insight into which products should always be in inventory, and which products are frequently bought together. This data could then be used to drive decisions on product suggestions on the retails online store, to increase profitability.

I have found two libraries for python suited for this kind of application. Mlxtend and efficient apriori. I've decided to use mlxtend, the main reason for this is a more streamlined use with pandas DataFrames, and good documentation. Although this is not regarded as a ML-algorithm, it certainly could add value to the dataset.

30.04.24

Ran the mlxtend apriori on the dataset. Determining threshold levels and what columns are vital to reach my objectives. For MBA the focus is sales strategy, so I will include support, confidence and lift.

01.05.24

Working on the visualizations. Displaying the top 10 most common itemsets. (Support). The plan regarding confidence/lift is to create a scatterplot, showing where the 'gold' is in terms of strongest associations. This will guide promotional and cross-selling strategies.

02.05.24

Working on the visualizations to get them to display correctly. Figuring out how to best display support, confidence and lift. The decision was made to create bar plots for all three metrics, as this best represents the data. In the report there will be a table alongside the plots, displaying the raw data as well, or at least a reference to it.

03.05.24

Looking through the code I have so far, adjusting comments, etc.

06.05.24

At first, I started with separate files for preprocessing, MBA and customer segmentation. Decided it best to combine all of those into one file. Redoing some of it to make it work correctly.

07.05.24

Thinking through how to gain most insight into the data. Figuring out how to best make use of the results from the apriori algorithm and trying to combine it with the original dataset as new features.

08.05.24

Creating a function for customer segmentation, which labels the customers as 'High Value', 'Medium Value' and 'Low Value'. This is based on RFM (Recency, Frequency, Monetary) analysis. Using pandas qcut to segment the customers into these categories.

09.05.24

Working on trying to combine rules from the MBA to the original dataset. Successfully managed to get the top 5 rules sorted by confidence into binary features in the original dataset.

10.05.24

I decided to include the top 10 rows of rules sorted by confidence (instead of 5). These are now binary features merged to the 'data' dataset. Saved the new dataset as 'data_featured.csv'.

11.05.24

Going through the code so far, removing unnecessary print statements, cleaning up the code a bit, refining comments. Started working on the machine learning part on the dataset.

12.05.24

More clearly defining the goals for machine learning. Regarding customer segmentation, in addition to the rfm analysis, I will **run unsupervised clustering algorithms** on the dataset. I first tried to run kMeans clustering without PCA/SVD. It seemed like the program froze. I then tried to reduce dimensionality using PCA first, got a warning about the sparse data, and a recommendation to use Truncated SVD instead. So, I did that. Same results, frozen terminal (or at least nothing happening for about 40 minutes, I manually aborted the operation.) Figured out that it was the measuring algorithm that was the bottleneck. I was trying to use the silhouette_score. Finally figured out that I could set the sample size to limit it. Set it to 100,000. Then I got the results in 1m38s. That's more to my liking.

For the sales strategy part the goal is to **find strong associations** in the inventory and purchase history, to gain insights into **cross-selling strategies**. My first approach was to use the rules mined in the apriori analysis in the dataset for ML. But the number of binary features became large. So instead, I will try to compare the results from the apriori with the ML part, and see if there are deeper patterns to find.

Reading about what models is best to use on sparse data: https://www.kdnuggets.com/2023/04/best-machine-learning-model-sparse-data.html

Looks like logistic regression could be a good choice to start with for the sales stragety part, and kMeans clustering for the customer segmentation part.

13.05.24

When running the kMeans successfully, I found out one of the clusters held:

'Manual'

'POST',

'DOT',(

'ADJUST',

'D'(for discount),

'CRUK'(Cruk commission),

Removing those in the data_analysis script as these entries does not contribute to reach my goals. Another thing I found when visually analyzing the results, is that there are some transactions that are negative, although they do not belong to the cancelled orders column. The cancelled order column was dropped earlier. This could be transactions that are returned, and the customer get the money back, as I find the same amount in positive numbers as well. I also found another small cluster, containing a few entries of manual adjustment in the database. I will also remove these in the data_analysis script. Started with 5 clusters, but reduced to 3 to maintain the granularity, as I have used 3 bins in the rfm analysis. Trying to display the 3 clusters and the 3 rfm bins on a heatmap.

14.05.24

I am satisfied with the results from the initial data_analysis.py and customer_segmentation.py scripts. Moving on to work on the customer_lifetime_value.py script. When thinking through this part, I realize I should have split the data as the data I have is from December 1. 2009 to December 9. 2011. To utilize machine learning for the regression purpose of predicting customer lifetime value (CLV) I need to generate the target label from the sum of one year of purchase history per customer.

The rules generated from the apriori algorithm and the rfm analysis and clustering done on the whole dataset is good. However, for the CLV regression I will need to split the data before apriori, rfm and clustering to avoid data leakage. Making some adjustments to the data_analysis.py script to handle both datasets.

15.05.24

Today was spend working on the customer_lifetime_value.py script. Implemented Linear Regression instead of logistic regression, Random Forest and Gradient Boosting models.

16.05.24

Struggling with getting my model training to work as intended. Got it working eventually. Fount out that Linear Regression and Gradient boosting predicts negative numbers. I have two options regarding this: 1. Clip the negative numbers to 0 or check the scaling of the data in the preprocessor. I will investigate the scaling anyway, so I'll start there. If things are as they should, it could be the nature of the regression models that it output these kinds of numbers.

17.05.24

Did some reading on CLV: https://www.netsuite.com/portal/resource/articles/ecommerce/customer-lifetime-value-clv.shtml

The models I have are trained on one year of data. So, the predictions will be one year of customer spending.

18.05.24

Rewriting some code in data_analysis.py, moving things around, writing better comments and docstrings.

Moving the code from customer_segmentation.py into a function in data_analysis.py. This to include all generated features before splitting the data to create the target label.

Dealt with outliers in the futurespend column as well. Struggled still to find out why, the predictions were unrealistic high.

19.05.24

Found out I had forgot to scale the y (target label), now predictions look more realistic. Running gridsearch CV again to find the best hyper-parameters. With the param grid I've set, it took forever, I let it run for 27hours before manually stopping it. Started working on the report.

20.05.24

Major setback/discovery today. Initially the plan was to create a future time window to compare the predictions up against. Today I found out that none of the customers from my first year of data was present in the second year. I've got 5942 unique customers in the complete dataset (after some data cleaning). So instead of going for the initial plan, I will have to rely on the test set in the model pipeline alone. I'll add this to my growing bag of experience. **This finding alone makes me think this online retail company has a major customer churn problem.**

21.05.24

Kept trying to get the predictions to be realistic. Instead of using the whole dataset I cut it into the two years, after the preprocessing. The plan now is to use the first part to train models, and the second part for validation.

22.05.24

Kept getting unrealistic predictions. Spent the whole day trying to solve it. Finally figured out I had forgotten to multiply quantity with price before aggregating the customer data. Now it looks promising.

23.05.24

Working on streamlining the process of model selection better. It now selects the best model base on the lowest mean squared error (MSE). Made some more adjusting of the target label. The target label is now called Actual_CLV and holds TotalSpend * 5. This to train the model for predicting clv values for a 5-year relationship.

24.05.24

Finishing the adjusting of outliers and model hyperparameter tuning. Keep working on the report.

25.05.24

Working on the report. Also found out I had made a mistake in how I checked if customers were present in both years of data. Turns out they did not lose all their customers from one year to the next. There were over 2700 customers in both original datasets.

26.05.24

Working on the report. Figuring out whether to try to remove outliers or not. Because limiting data too much with outlier removal, a decision was made to keep outliers. Two of the models used can handle this kind of data well. RandomForest and GradientBoosting. Linear regression is not the best for this kind. Of data. Next time I will do more research up front.

27.05.24

Working on the report.

28.05.24

Working on the report.

29.05.24

Working on the report.

Changing the filenames of my files to the format required.

aw38.ep.per.idar.rod.1.py

aw38.ep.per.idar.rod.2.py

aw38.ep.per.idar.rod.3.csv

aw38.ep.per.idar.rod.4.csv

aw38.ep.per.idar.rod.5.pdf

30.05.24

Working on the report.

31.05.24

Final adjustment of the report.

Reading through the code once more.