CS 5764: Information Visualisation Final Term Project Report

Topic: Data Analysis of 21 Supermarkets in Baku City

Name: Shubham Laxmikant Deshmukh

Course Instructor: Dr. Reza Jafari

Date: 12/04/2023

Table of Contents

Sr. No.	Content	Page No.
1.	Abstract	2
2.	Introduction	3
3.	Description of the Dataset	4
4.	Pre-processing Dataset	6
5.	Outlier detection and removal	12
6.	Principal Component Analysis (PCA)	24
7.	Normality Test	36
8.	Heatmap and Pearson Correlation Coefficient Matrix	54
9.	Data Visualization	60
10.	Subplots	84
11.	Tables	88
12.	Dashboard	90
13.	Conclusion	105
14.	Appendix References	120

Table of Figures and Tables

Sr. No.	Figures	Page No.
1.	Line Plot	60
2.	Bar plot	65
3.	Count plot	70
4.	Dist plot	71
5.	Heatmap with cbar	72
6.	QQ plot	72
7.	KDE plot with fill	74
8.	Lm or reg plot	75
9.	Multivariate box plots	75
10.	Joint plot	76
11.	Rug plot	76
12.	3D plot	77
13.	Cluster map	77
14.	Strip plot	78
15.	Swarm plot	78

Abstract

This information visualization project delves into the exploration and analysis of a robust dataset capturing supermarket transactions in Baku, Azerbaijan, throughout the year 2019. The dataset encompasses a wealth of information, including details on 438,826 products purchased by a diverse customer base of approximately 80,000 individuals. Transactions were conducted across 21 branches, offering insights into the geographical and operational nuances of the supermarkets during the specified year.

The project employs a variety of visualization techniques to elucidate patterns, trends, and anomalies within the dataset. Utilizing tools such as bar plots, count plots, line plots, and area plots, the project visually represents key aspects of the supermarket transactions. Additionally, Principal Component Analysis (PCA) is conducted to distill the dataset's multidimensional information into a more manageable form, aiding in the identification of underlying structures and relationships.

The preprocessing steps undertaken before analysis are integral to the project's reliability. Null values are removed, ensuring data completeness, while duplicate rows and the 'ID' column are eliminated to enhance dataset integrity. Furthermore, a transition from Azerbaijani to English is facilitated by translating column names and values using the 'googleTranslator' library, fostering a more universally accessible and interpretable dataset.

This project not only sheds light on the sheer scale of supermarket transactions in Baku but also offers a visual narrative that facilitates a comprehensive understanding of consumer behavior, product distribution, and the operational landscape of the supermarkets. The combination of exploratory data analysis, visualization, and preprocessing techniques contributes to a robust and insightful examination of the supermarket dataset, providing a foundation for future studies and decision-making processes within the retail industry.

Introduction

In the realm of data science, the ability to transform raw datasets into actionable insights is paramount. This project embarks on a comprehensive exploration of a supermarket dataset from Baku, Azerbaijan, gathered in the year 2019. The overarching objective is to extract meaningful patterns, trends, and relationships within the data, unraveling the intricacies of consumer behavior, product distribution, and the operational landscape of supermarkets in the city.

The preprocessing journey was meticulous and strategic. Null values were purged, ensuring data completeness and reliability. Extraneous details, such as the 'ID' column, were excised to streamline the dataset, while duplicate rows were eradicated to enhance its integrity. The process extended beyond structural modifications, encompassing a language transition from Azerbaijani to English for improved interpretability. The 'googleTranslator' library facilitated the translation of column names and values, paving the way for a more universally accessible dataset.

With a pristine dataset in hand, a diverse array of visualization techniques was employed to distill insights. Bar plots, count plots, line plots, area plots, and more were utilized to unravel the story hidden in the numbers. Principal Component Analysis (PCA) further distilled the multidimensional dataset, unraveling latent structures.

The journey did not end with exploratory data analysis; it transcended into the realm of interactive dashboards. Eight tabs, each a portal into distinct facets of the dataset, were meticulously crafted. The dashboard not only showcases informative plots but also incorporates dynamic elements such as dropdowns, graphs, loading indicators, and more. A meticulous checklist was adhered to, ensuring an interactive and user-friendly interface.

This project culminates with the deployment of the dashboard on the Google Cloud Platform, transforming the insights gained into a publicly accessible website. The result is not merely a compilation of graphs but a dynamic, interactive platform that invites stakeholders to explore and comprehend the nuances of supermarket transactions in Baku. This report outlines the procedural journey from dataset preprocessing to visualization and dashboard deployment, providing a blueprint for future explorations and applications within the realm of data science and business intelligence.

Description of the Dataset

The Supermarket Dataset under consideration offers a comprehensive snapshot of retail transactions in the year 2019 across 20 branches in Baku, Azerbaijan. This extensive dataset encompasses details on a staggering 438,826 products purchased by a diverse customer base of 80,000 individuals. The richness of the data is manifested in its 11 columns, providing insights into various aspects of each transaction.

Key Variables:

1. **Receipt_Number** : A unique identifier for each transaction.
2. **Product_Code**: Identifies the specific product in the transaction.
3. **Product**: Describes the name or type of the product, categorized for clarity.
4. **Category**: Classifies products into broader categories such as "Məişət məhsulları" (Household Products) and "Qab-qacaq" (Tableware).
5. **Price**: Represents the monetary value of the purchased product in dollar currency ($).
6. **Date**: Captures the date and time of the transaction.
7. **Discount**: Indicates any discounts applied during the transaction.
8. **Bonus_cart**: A boolean variable denoting whether the product is eligible for bonus points.
9. **Store_Branch**: Specifies the branch of the supermarket where the transaction occurred.
10. **Store_Lat**: Latitude coordinates of the supermarket branch.
11. **Store_Long**: Longitude coordinates of the supermarket branch.

The dataset, comprising 59,354 entries, underwent meticulous preprocessing. Null values were eliminated, and redundant columns, such as 'ID,' were removed to enhance data integrity. The dataset's language was standardized by translating column names and values from Azerbaijani to English, promoting universal understanding.

For analytical purposes, the selected dependent variable is the "Price" column, representing the monetary value of the purchased products. Various other columns, such as "Product," "Category," and "Store_Branch," serve as independent variables, providing insights into the diverse facets of each transaction.

The importance of this dataset in the industry lies in its capacity to reveal patterns and trends in consumer behavior, product preferences, and regional variations across supermarket branches. Analyzing this dataset can inform pricing strategies, inventory management, and marketing initiatives, offering valuable insights for optimizing business operations in the retail sector. The deployed dashboard on the Google Cloud Platform further enhances the accessibility and usability of these insights, making it a valuable resource for decision-makers in the retail industry.

Kaggle website link:
https://www.kaggle.com/datasets/mexwell/supermarket-dataset

Pre-processing dataset:

The preprocessing steps applied to the Supermarket Dataset are crucial for ensuring data quality, consistency, and readiness for analysis. Here's a breakdown of the pre-processing steps:

1. **Handling Null Values**:
   - Any entries with missing or null values were identified and subsequently removed from the dataset. This step is essential to prevent skewed or incomplete analyses and to ensure the reliability of results.

The sum of null values after drop na:

```
Receipt_Number    0
Product_Code      0
Product           0
Category          0
Price             0
Date              0
Discount          0
Bonus_cart        0
Store_Branch      0
Store_Lat         0
Store_Long        0
dtype: int64
```

Here we can see there are no more ‘NA’ values in the dataset. Now we can proceed for further steps.

2. **Column Removal**:
   - The 'ID' column, which may not contribute significantly to the analysis, was eliminated. This serves to simplify the dataset and improve computational efficiency.

3. **Duplicate Rows Removal**:
   - Duplicate rows, if any, were identified and removed. Duplicate entries can distort analyses, leading to inaccuracies in statistical measures and visualizations.

4. **Translation of Language**:
   - The language of the dataset, initially in Azerbaijani, was standardized by translating both column names and values of ‘Category’ and Store_Branch’ columns into English. This ensures a uniform and universally understandable dataset, facilitating collaboration and interpretation. The translation was done by a well known python library called googleTranslator.

5. **Data Type Adjustment**:
   - The data types of certain columns were adjusted to enhance efficiency and accuracy. For example, the 'Date' column, initially stored as an object, was converted to a datetime data type for easier temporal analyses.

The result of these preprocessing steps is a cleaned and standardized dataset ready for exploratory data analysis (EDA) and subsequent modeling. By addressing missing values, eliminating redundant information, and ensuring linguistic consistency, the dataset becomes more robust, enabling meaningful insights to be derived during analysis.

It's worth noting that the specific preprocessing steps applied may vary based on the nature of the dataset, the analytical goals, and the characteristics of the data. In this case, the preprocessing steps were tailored to enhance the suitability of the Supermarket Dataset for

visualization and analysis of consumer behavior, product pricing, and branch-related insights.

The first 5 rows of the preprocessed dataset are:

```
   Receipt_Number  Product_Code                                   Product  \
0          577571          3334  GONDOL SIMPLE LABABO VE BUZLUK HALISI G61*50
1          577571          4674                              TAXTA BICAQ 3232
2          577571          8388                       VITA 1LT ANANAS SIRESI
3          577571          6017          GILAN BAGDAN 1LT SARI GAVALI KOMPOTU
4          577571          8297      OSRAM SUPERSTAR LAMPA 64544A ECO 57W E27

             Category  Price                 Date       Discount  Bonus_cart  \
0  Household products   3.68  2019-07-19 12:29:00  Summer Season        True
1              Dishes   2.25  2019-07-19 12:29:00  Summer Season        True
2        Fruit juices   2.50  2019-07-19 12:29:00  Summer Season        True
3            Compotes   1.45  2019-07-19 12:29:00  Summer Season        True
4  Household products   2.92  2019-07-19 12:29:00  Summer Season        True

  Store_Branch  Store_Lat  Store_Long
0       Zabrat  40.485561   49.946741
1       Zabrat  40.485561   49.946741
2       Zabrat  40.485561   49.946741
3       Zabrat  40.485561   49.946741
4       Zabrat  40.485561   49.946741
```

The information of the preprocessed dataset is :

```
<class 'pandas.core.frame.DataFrame'>
Index: 59354 entries, 0 to 59362
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Receipt_Number  59354 non-null  int64
 1   Product_Code    59354 non-null  int64
 2   Product         59354 non-null  category
 3   Category        59354 non-null  category
 4   Price           59354 non-null  float64
 5   Date            59354 non-null  object
 6   Discount        59354 non-null  category
 7   Bonus_cart      59354 non-null  bool
 8   Store_Branch    59354 non-null  category
 9   Store_Lat       59354 non-null  float64
 10  Store_Long      59354 non-null  float64
dtypes: bool(1), category(4), float64(3), int64(2), object(1)
memory usage: 4.1+ MB
None
```

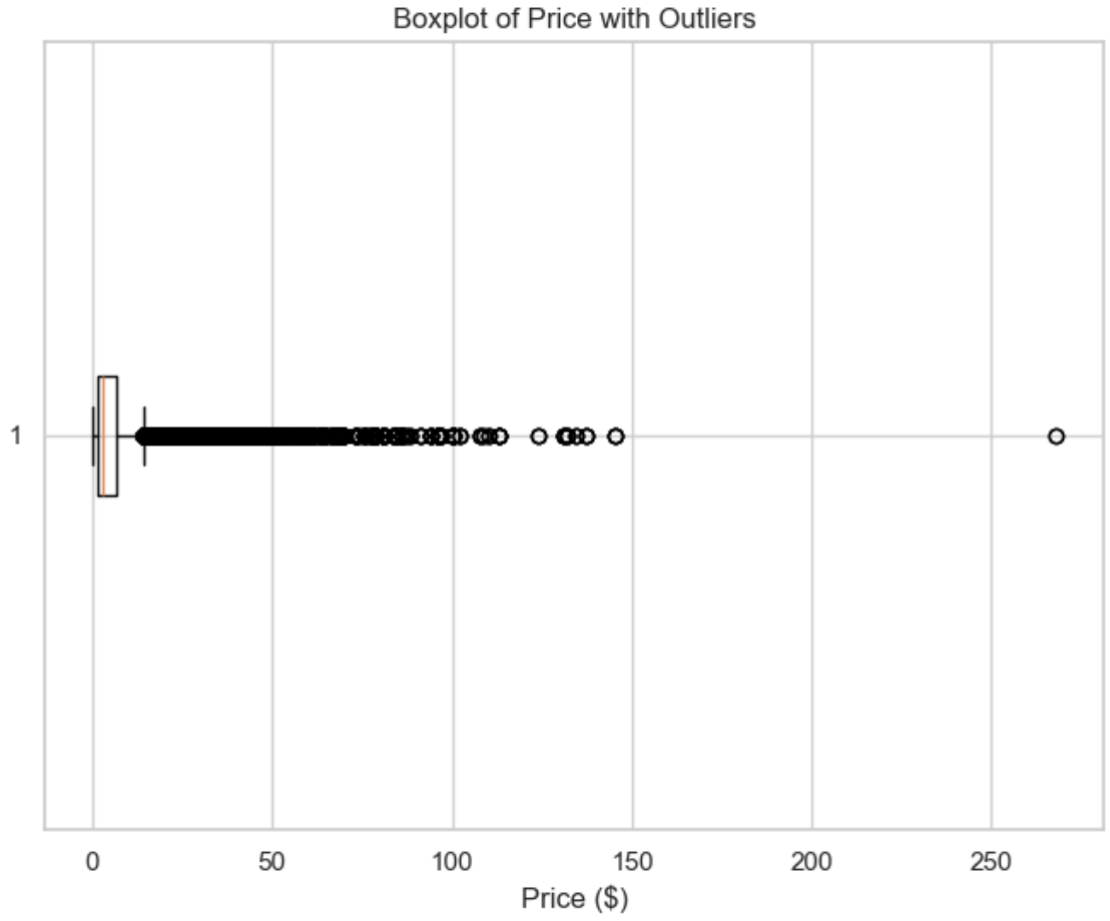
From the information of the clean_df dataframe we can see that there are 5 numerical features (float64, int 64), 4 categorical features, 1 boolean feature and 1 object feature.

Outlier detection and Removal:

To further refine the Supermarket Dataset and enhance the accuracy of subsequent analyses, outlier detection and removal were performed on the 'Price' column. Outliers, being data points significantly deviating from the majority, can distort statistical measures and impact the reliability of visualizations.

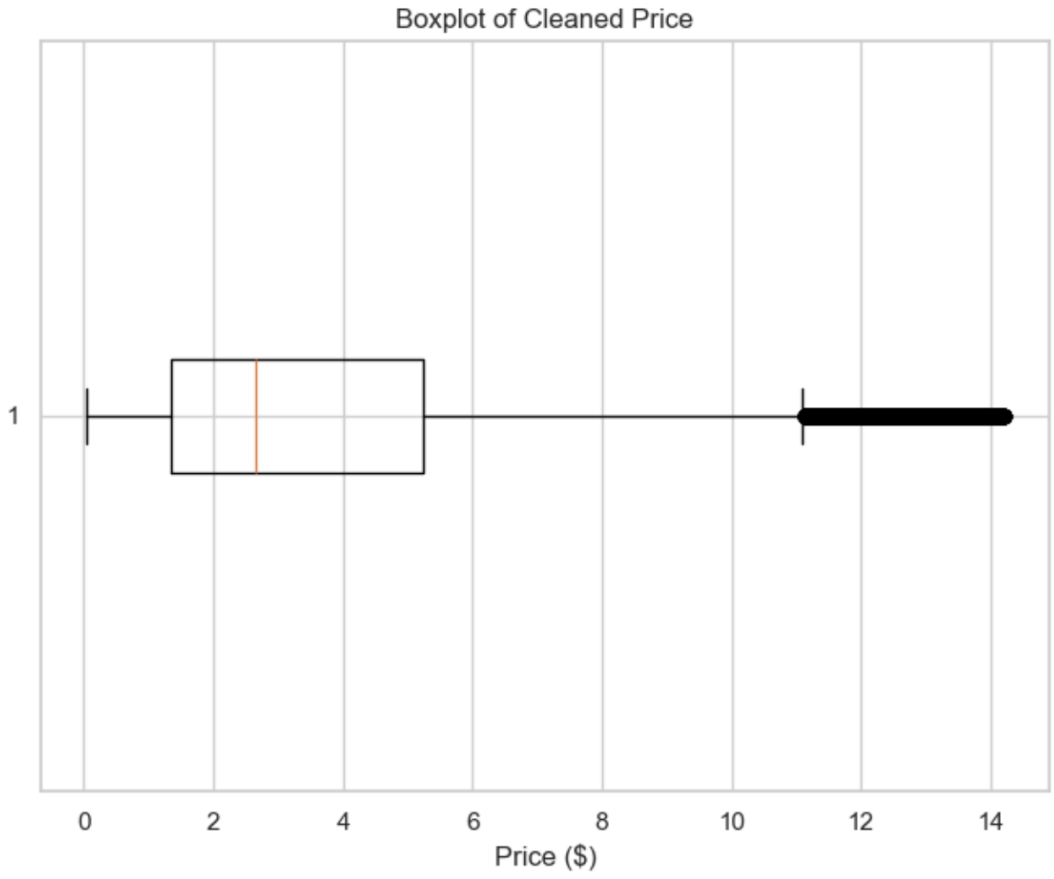
Before Outlier Removal:

A preliminary examination of the 'Price' column, visualized through a boxplot, revealed a substantial presence of outliers on the right side of the distribution. These outliers exhibited values ranging from $15 to $150, indicating significant deviations from the typical price range.



Outlier Removal Process:
Various statistical methods, such as the interquartile range (IQR), were employed to identify and subsequently remove outliers from the 'Price' column. This process involved setting a threshold beyond which data points were considered outliers and subsequently excluded from the dataset.

After Outlier Removal:
The boxplot post-outlier removal portrays a more refined distribution, with the interquartile range now ranging from 0 to $11.5. The removal process effectively mitigated the impact of extreme values, resulting in a more representative visualization of the central tendency and variability of prices within the dataset. However, it's noteworthy that despite the removal process, a few outliers still persisted on the right side of the boxplot.



The persistence of some outliers underscores the diversity and complexity inherent in real-world datasets. Further investigation into the nature of these remaining outliers could provide valuable insights into specific products or transaction patterns that deviate from the norm. Overall, the outlier detection and removal process contribute to a more robust and accurate dataset, laying a solid foundation for subsequent analyses and visualizations.

Principal Component Analysis (PCA):

In an effort to distill the multidimensional information within the Supermarket Dataset, Principal Component Analysis (PCA) was applied to the numerical feature columns, including 'Receipt_Number,' 'Product_Code,' 'Price,' 'Store_Lat,' and 'Store_Long.' PCA is a dimensionality reduction technique that identifies the principal components, allowing for a simplified representation of the dataset.

PCA Process with Standard Scaling:

1. **Standard Scaling:**
   - Before applying Principal Component Analysis (PCA), standard scaling was performed on the selected numerical features. Standard scaling ensures that all features have a mean of 0 and a standard deviation of 1, preventing features with larger scales from dominating the analysis.

2. **Correlation Matrix Plot:**
   - To understand the interrelationships between the standardized numerical features, a correlation matrix plot was generated. This plot provides a visual representation of the pairwise correlations, aiding in the identification of patterns and potential redundancies.

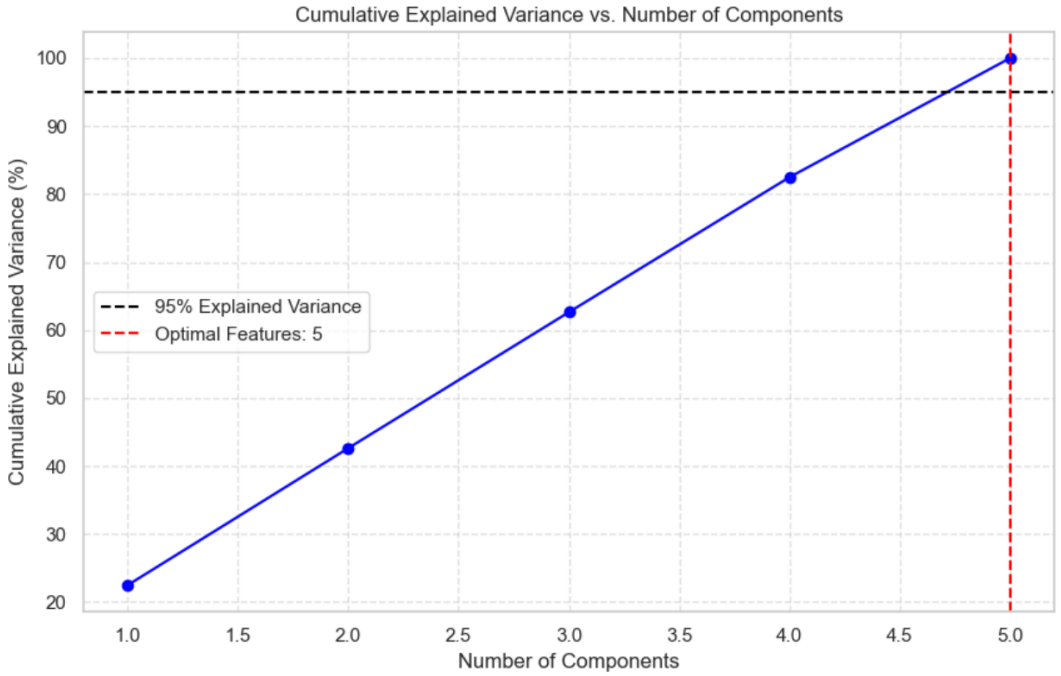
```
The first five rows of the reduced features:
        PC_1      PC_2      PC_3      PC_4      PC_5
0  0.846065 -1.047526 -0.118401  0.701427  2.804004
1  0.842584 -0.726216 -0.199130  0.202950  2.817264
2  0.833135  0.420976 -0.165869 -0.918923  2.860034
3  0.839125 -0.365619 -0.241416 -0.257612  2.831458
4  0.833387  0.418823 -0.140691 -0.865167  2.859594
```

```
Explained variance ratio: [0.225 0.202 0.2   0.199 0.175]
```

The number of principal components came out to be 5 which is the same number of numerical features/ columns in the dataset.

3. **PCA Transformation:**
   - The PCA transformation was then applied to the scaled numerical features. This process resulted in principal components, which are linear combinations of the original features. These components are ordered by their significance in explaining the variance within the dataset.

4. **Cumulative Explained Variance Plot:**
   - To determine the optimal number of principal components that capture a significant proportion of the variance, a cumulative explained variance plot was created. This plot depicts the cumulative proportion of variance explained by each additional principal component.

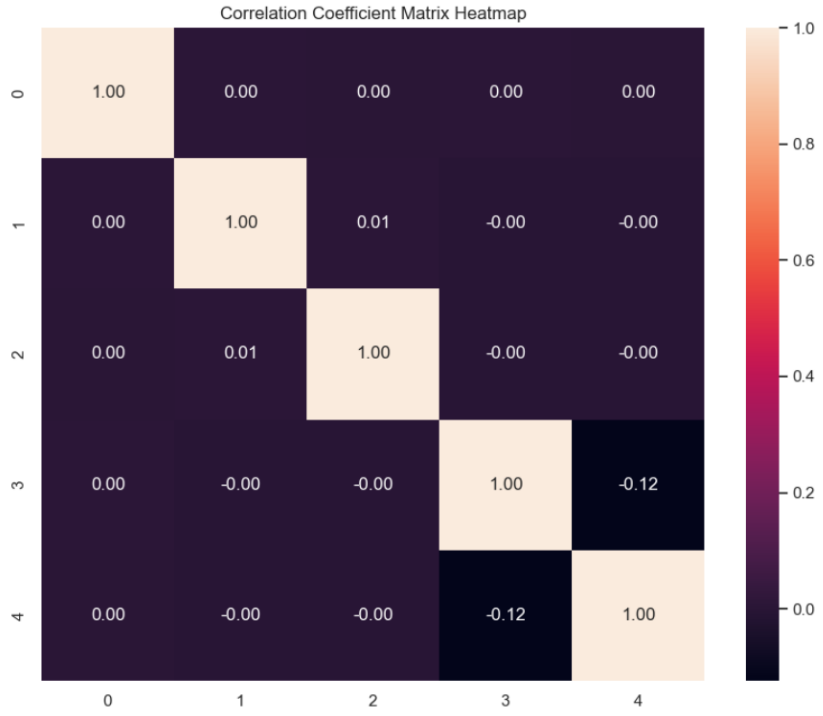


The plot indicates that with 5 principal components, approximately 95% of the variance in the data can be explained. The black dashed line represents the threshold of 95% explained variance, and the red dashed line denotes the point at which this threshold is crossed.

This outcome suggests that by retaining 5 principal components, a substantial amount of information from the original dataset is preserved, while simultaneously reducing dimensionality. The selected components can be used for subsequent analyses, offering a more efficient representation of the dataset's variability.

Heatmap Visualization:

A heatmap was then generated to visualize the correlation matrix of the principal components. This matrix depicts the relationships between the principal components, providing insights into their interdependencies.



This analysis underscores the independence of the numerical features in the dataset, with 'Store_Lat' and 'Store_Long' being the only variables demonstrating a modest degree of correlation. The insights gained from PCA and the heatmap contribute to a nuanced understanding of the underlying structure within the dataset, guiding subsequent analyses and informing decision-making processes in the context of supermarket transactions.

Normality Test:

In order to assess the normality of the 'Price' column within the Supermarket Dataset, three distinct statistical tests were employed: the Kolmogorov-Smirnov (KS) test, the Shapiro-Wilk test, and D'Agostino's K^2 test. Each test provides insights into whether the distribution of the 'Price' data deviates significantly from a normal distribution.

1. **Kolmogorov-Smirnov (KS) Test:**
   - The KS test evaluates the null hypothesis that the sample follows a normal distribution. The test statistic and p-value are computed, and the null hypothesis is rejected if the p-value is below a chosen significance level.

```
K-S test for Price: statistics = 0.27, p-value = 0.00
K-S test: Price dataset looks not normal
```

The KS test statistic of 0.27 suggests the maximum absolute distance between the empirical cumulative distribution function of the 'Price' data and the theoretical normal distribution. With a p-value of 0.00, the evidence is strong against the assumption of normality.

2. **Shapiro-Wilk Test:**
   - The Shapiro-Wilk test is another widely-used test for normality. Similar to the KS test, it assesses the null hypothesis that the sample is drawn from a normal distribution.

```
Shapiro-Wilk test for Standardized Price: statistics = 0.50, p-value = 0.00
Shapiro-Wilk test: Standardized Price data looks not normal
```

The Shapiro-Wilk test statistic of 0.86 further supports the conclusion that the 'Price' data is not normally distributed. The extremely low p-value reinforces the rejection of the null hypothesis.

3. **D'Agostino's K^2 Test:**
   - D'Agostino's K^2 test combines skewness and kurtosis to assess normality. The null hypothesis is that the sample follows a normal distribution.

```
D'Agostino's K^2 test for Price: statistics = 68646.31, p-value = 0.00
D'Agostino's K^2 test: Price looks not normal
```

The D'Agostino's K^2 test statistic measures the combined effect of skewness and kurtosis on the assumption of normality. With a test statistic of 68646.31, the deviation from the expected values under normality is substantial.

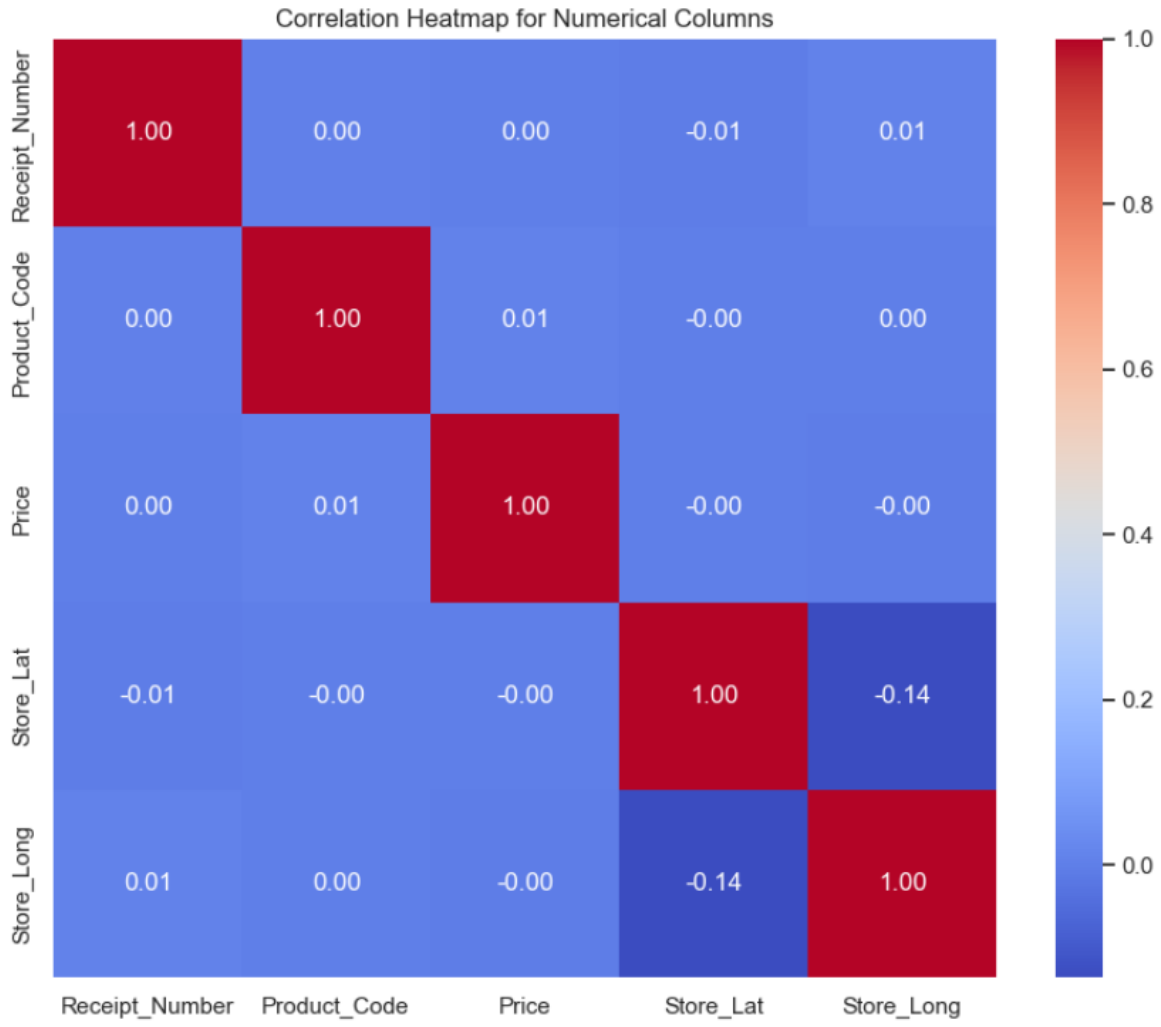
Interpretation:
- For each test, if the p-value is greater than the chosen significance level (e.g., 0.01), we fail to reject the null hypothesis, suggesting that the 'Price' data follows a normal distribution.
- Conversely, a low p-value would lead to the rejection of the null hypothesis, indicating a departure from normality.

It's crucial to interpret the results collectively and consider the consistency across different tests to draw robust conclusions about the normality of the 'Price' column in the Supermarket Dataset.

Heatmap & Pearson Correlation Coefficient Matrix:

To explore the relationships among numerical features in the Supermarket Dataset, a heatmap visualizing the Pearson correlation coefficient matrix was generated. The selected numerical features included 'Receipt_Number,' 'Product_Code,' 'Price,' 'Store_Lat,' and 'Store_Long.'

Heatmap Plot:



Interpretation:

The resulting heatmap visually represents the strength and direction of linear relationships between pairs of numerical features. The color intensity and numerical annotations within each cell denote the Pearson correlation coefficient values.

Correlation Insights:

From the heatmap, it is observed that only 'Store_Lat' and 'Store_Long' exhibit a discernible correlation with each other. The correlation coefficient of -0.14 indicates a weak inverse relationship between the latitude and longitude of the supermarket branches. The negative sign suggests that as one variable increases, the other tends to decrease, albeit modestly.

Inference:

- Store Location Relationship: The negative correlation between 'Store_Lat' and 'Store_Long' suggests that supermarket branches with higher latitudes tend to have lower longitudes, and vice versa. This may reflect a geographical pattern in the supermarket distribution.

- Other Features: The absence of significant correlations (correlation coefficient ≈ 0) among other numerical features ('Receipt_Number,' 'Product_Code,' and 'Price') indicates a lack of linear associations. This implies that these variables are relatively independent of each other.

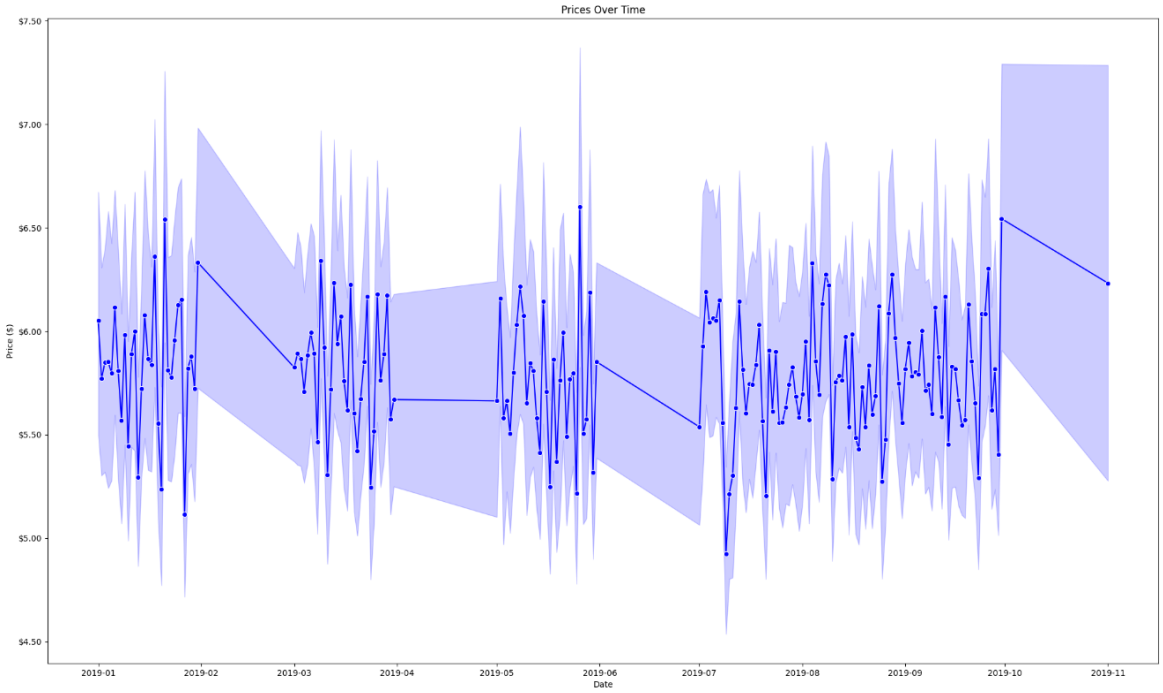
Considerations:

- While Pearson correlation provides insights into linear relationships, it may not capture nonlinear associations. Additional exploration and domain knowledge may be necessary for a comprehensive understanding of the data.

Data Visualization

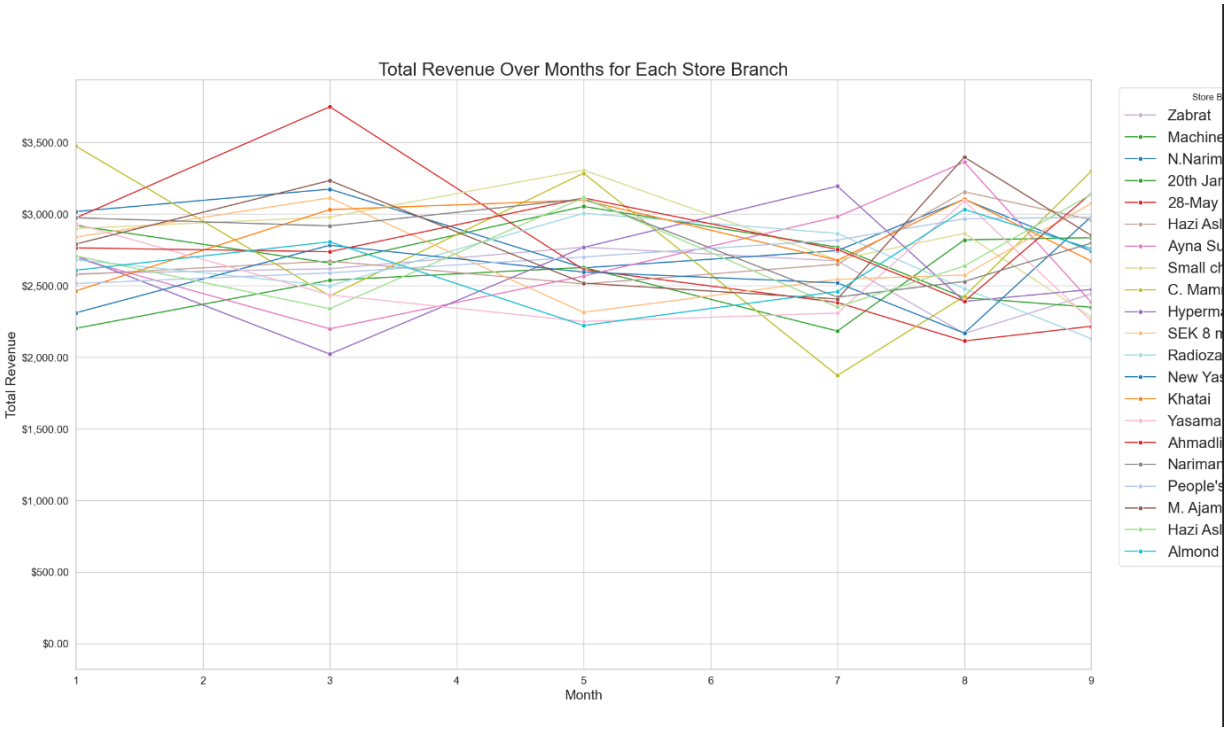
1. Line Plot:

- **The range of Prices over the time in the year (2019):**



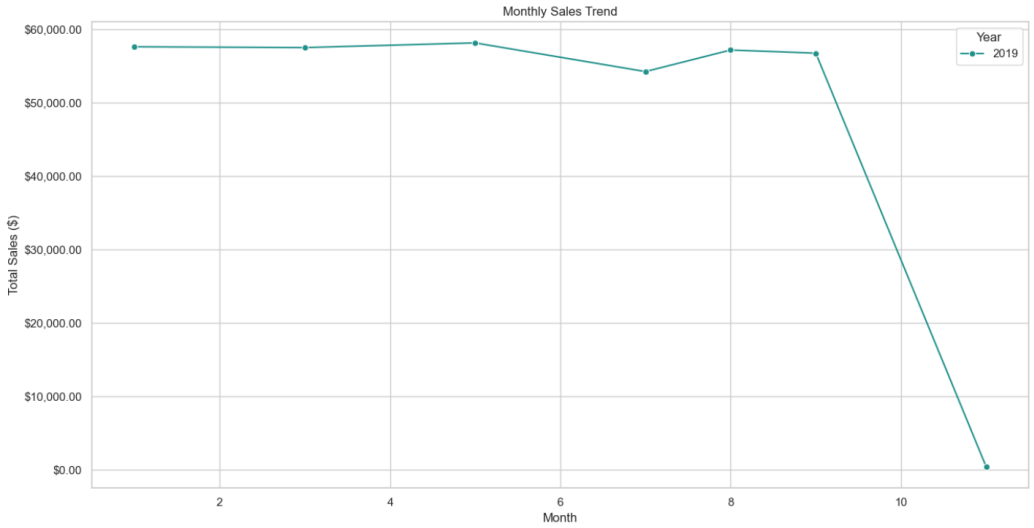
This graph shows the spread of prices of almost all products in all 21 Store branches that were purchased throughout the year 2019. There are some outliers that may not be visible in the plot, and most of the products prices are between $5 to $7 according to this graph.

- **Total Revenue of each Store Branch over the year 2019:**



We can observe the line plot for total revenue of each store branch out of 21 over the months in the year 2019. From the graph we can observe Khatai has the highest revenue amongst all the branches throughout the year.

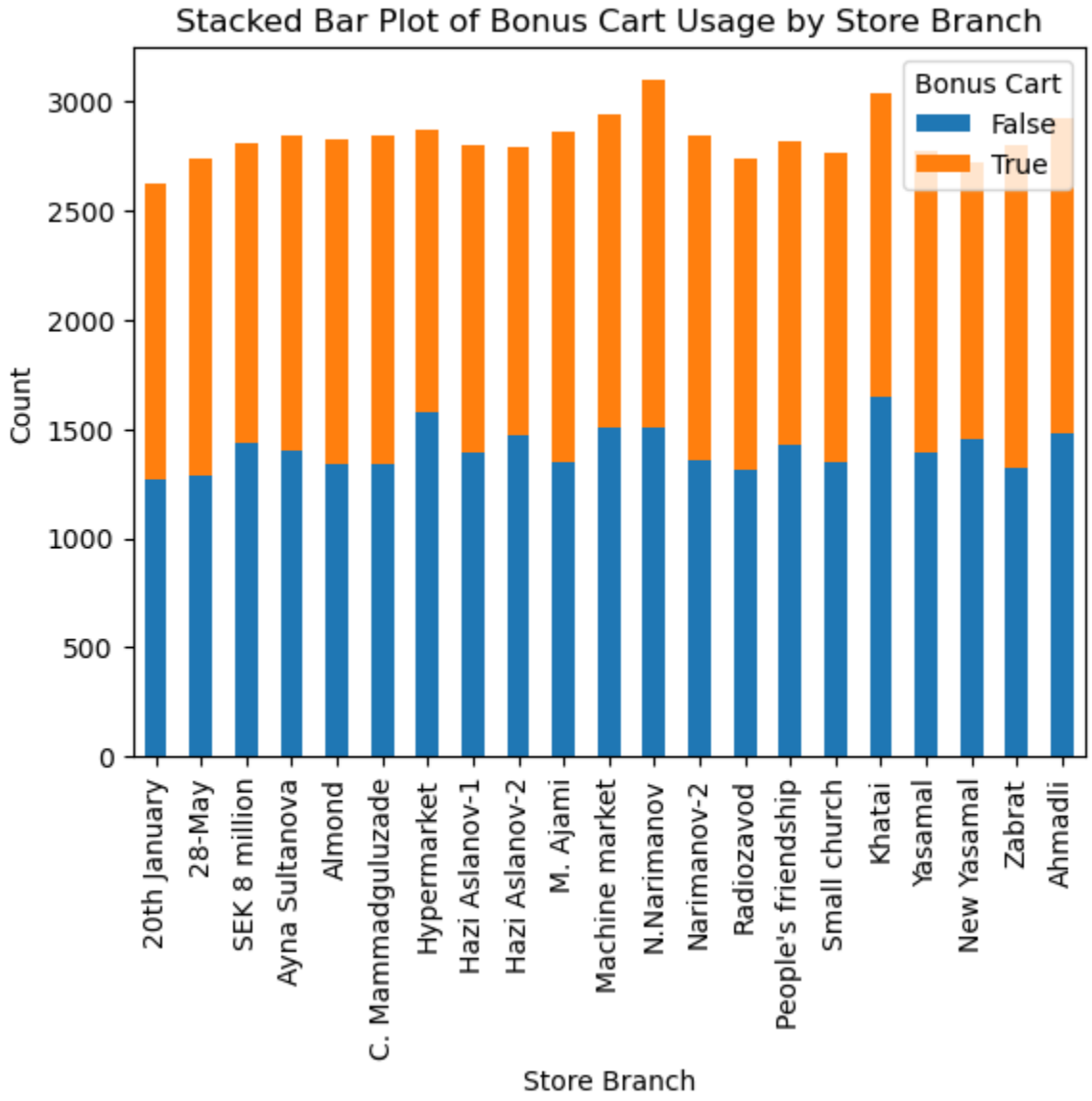
- **The Total revenue from all Store Branches in the year (2019):**



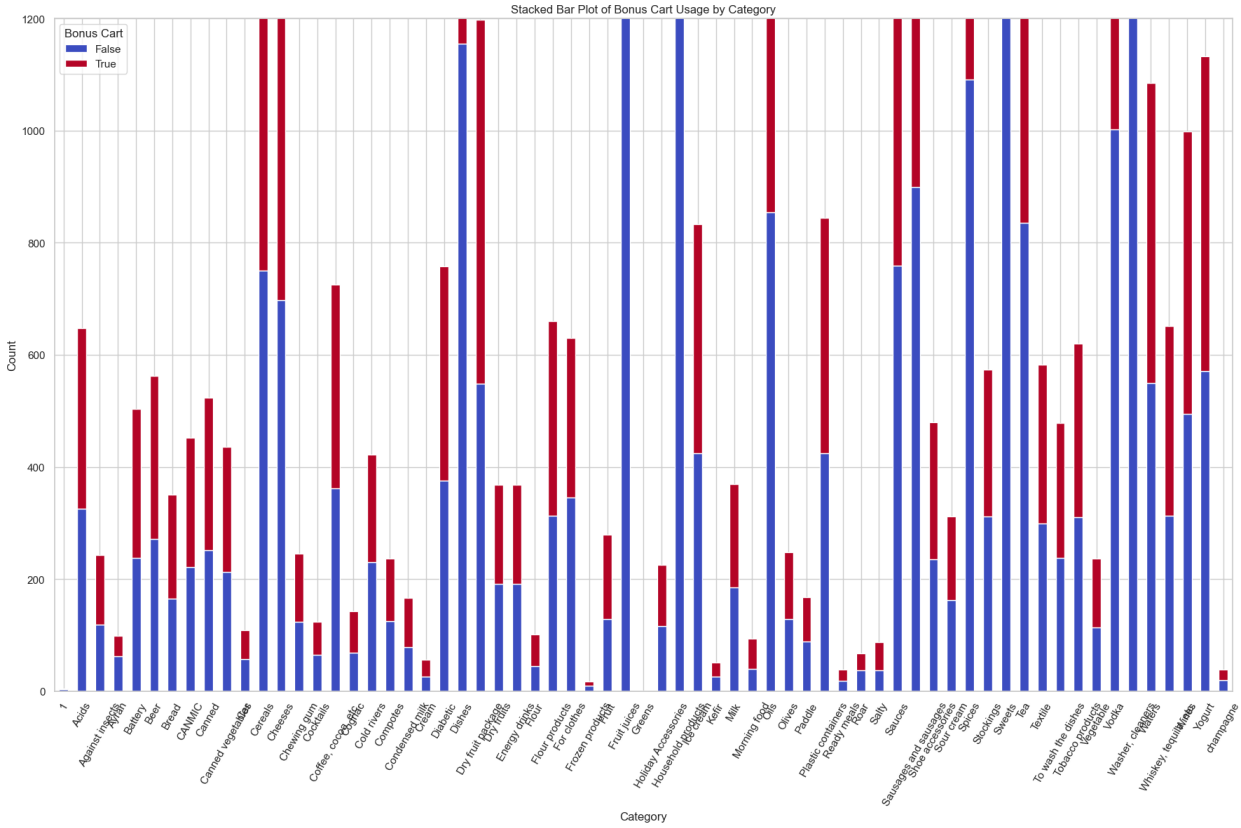
This graph gives the information about the total revenue from all 21 Store Branches over the year of 2019. From the graph we can observe that the Total revenue always ranges from $50,000 to $60,000 per month.

2. Bar plot: stack, group :

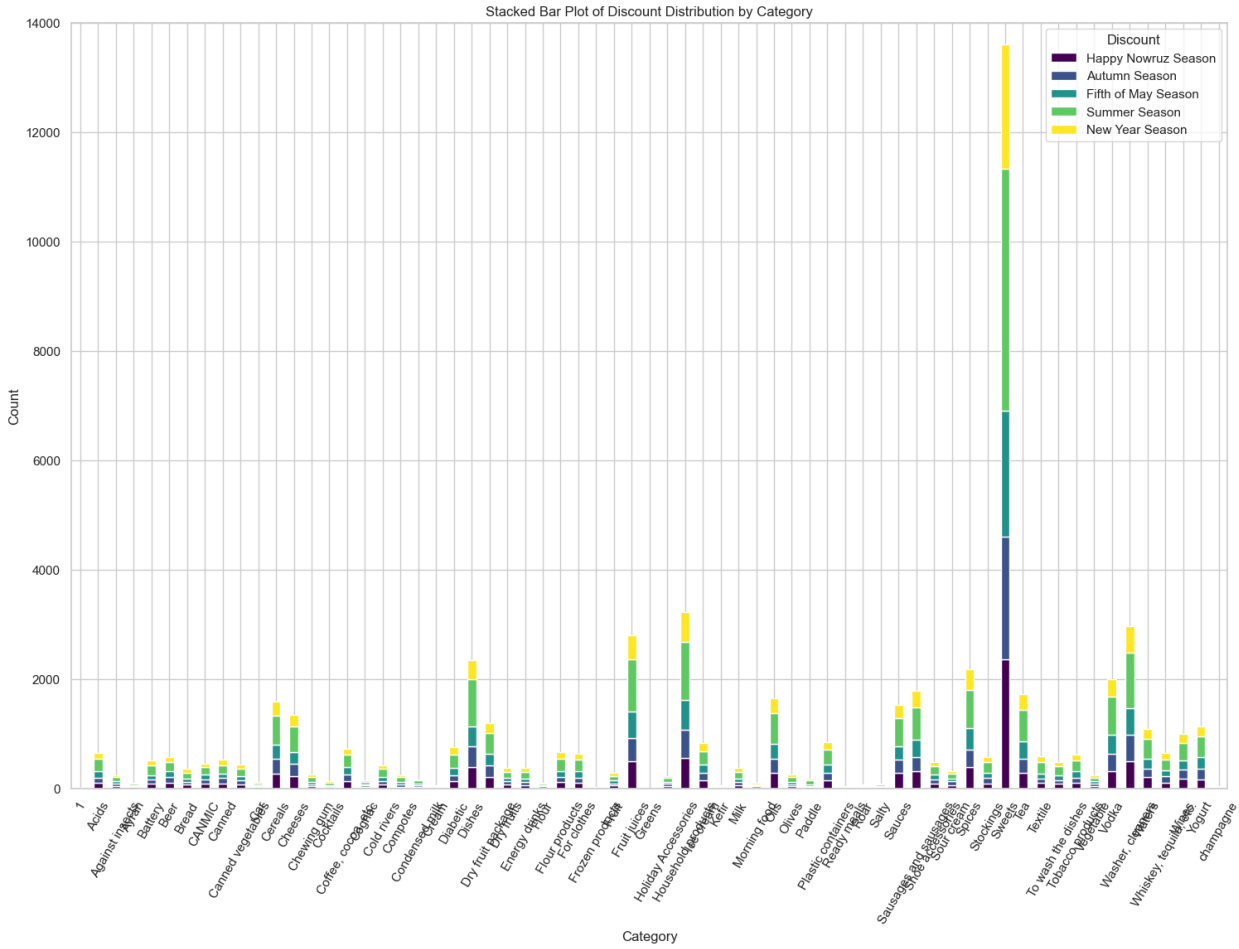
- **Stack plot:**



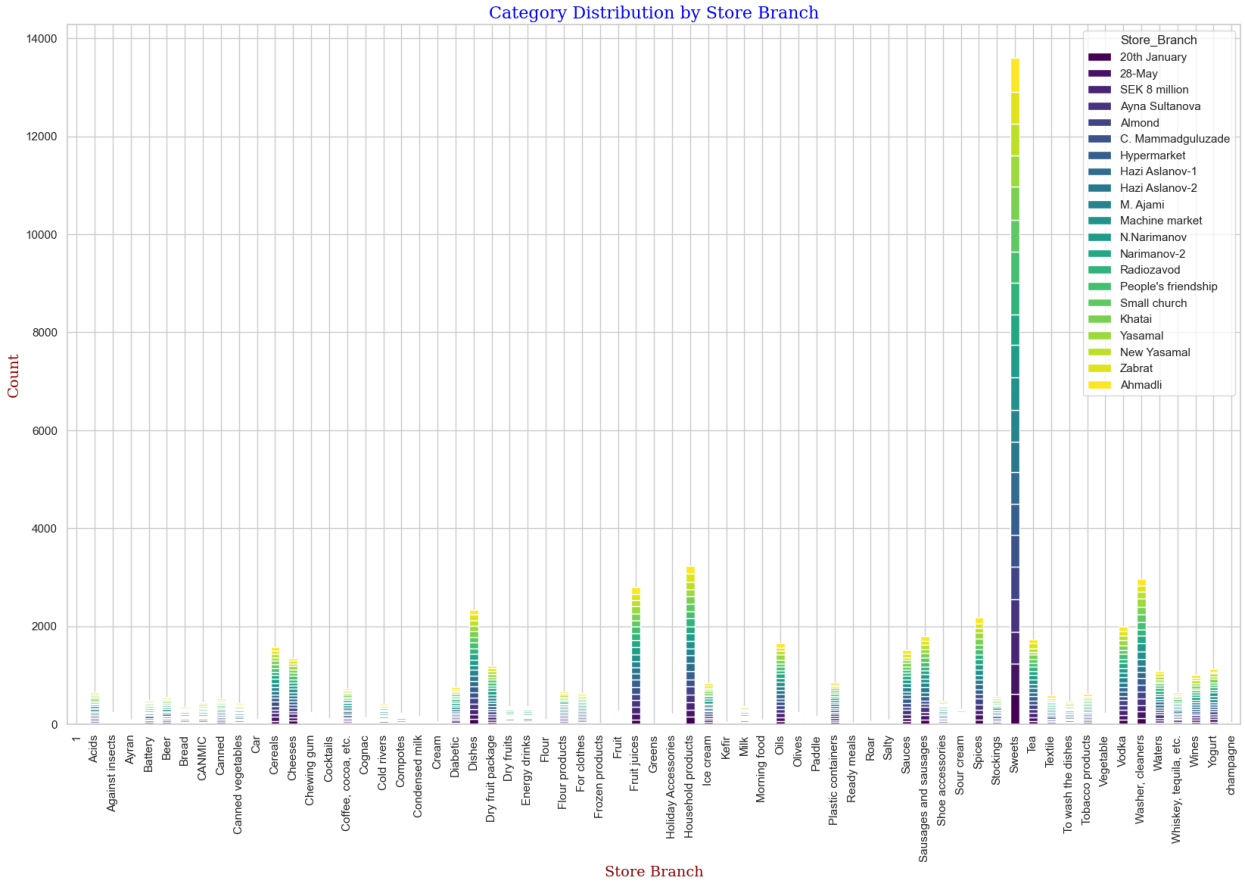
This is a Stack bar plot of use of bonus carts by each Store Branch. From the given plot almost all Store Branches have a similar distribution of True and False Bonus Cart values



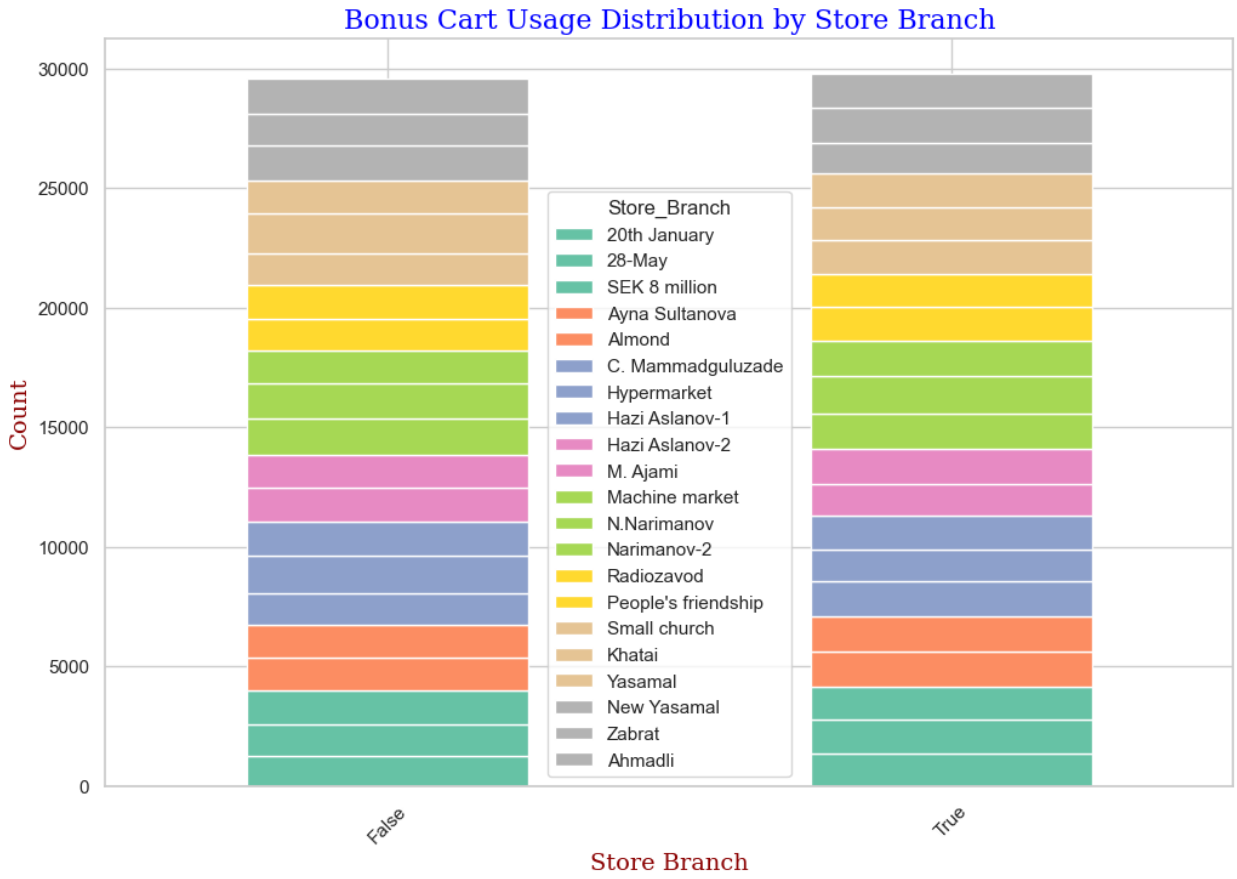
This stacked plot is used to plot the distribution of Bonus carts on all the Categories. This plot can be used to identify which Category got the most and least Bonus Cart. According to this plot ‘Cereals’ got most of the Bonus cart and ‘Diabetic’ Category got the least Bonus Cart.



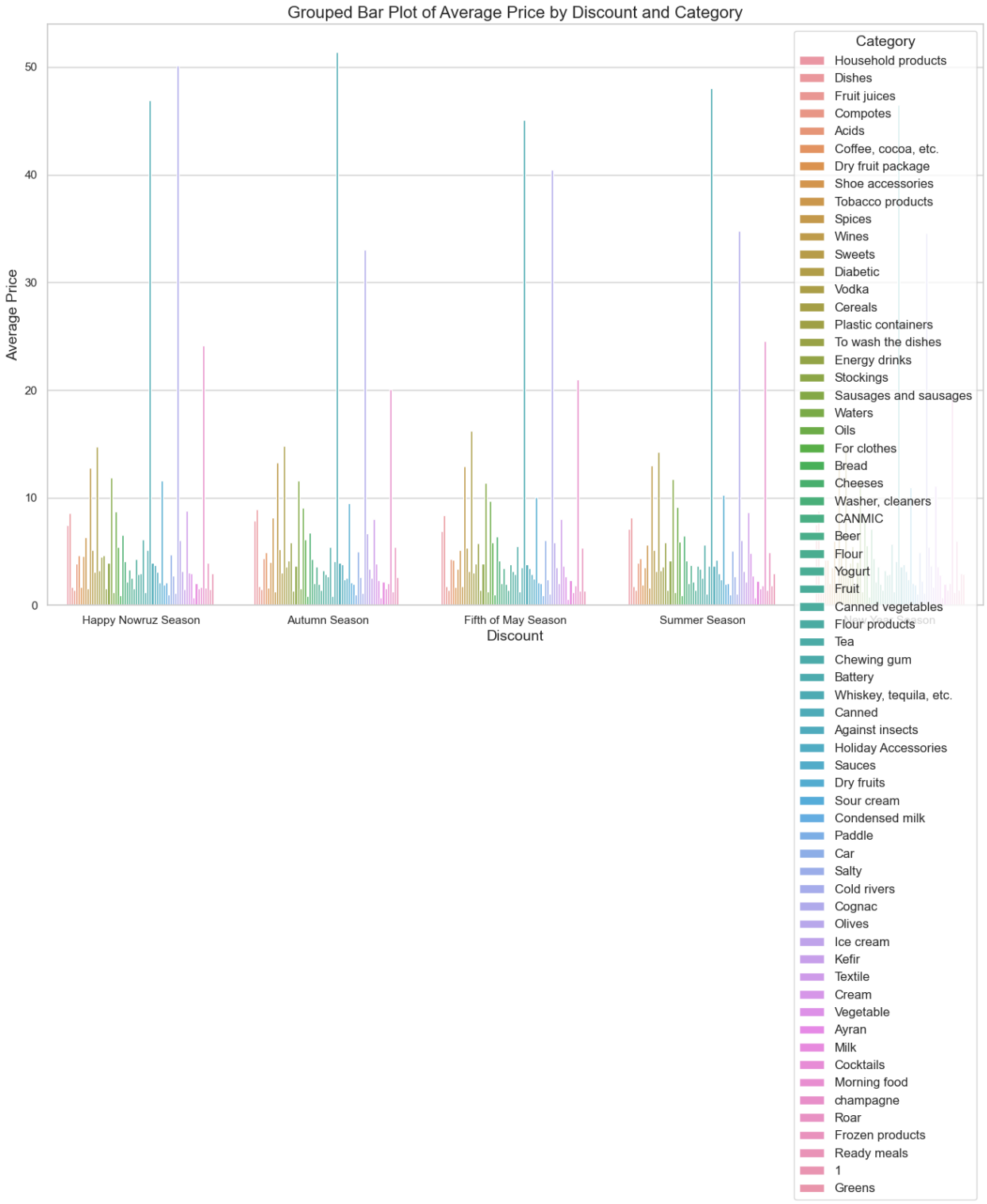
This stack bar plot signifies the Discount distribution by various Categories. This plot helps us to identify which Category is sold the most and in what season. According to the graph ‘Sweets’ Category is sold most in all the Discount seasons.



This stack bar plot is distribution of Categories by the Store Branch. Here we can identify in which Store Branch what category is sold most. According to the graph, in almost all branches the “Sweets ” Category is sold the most.

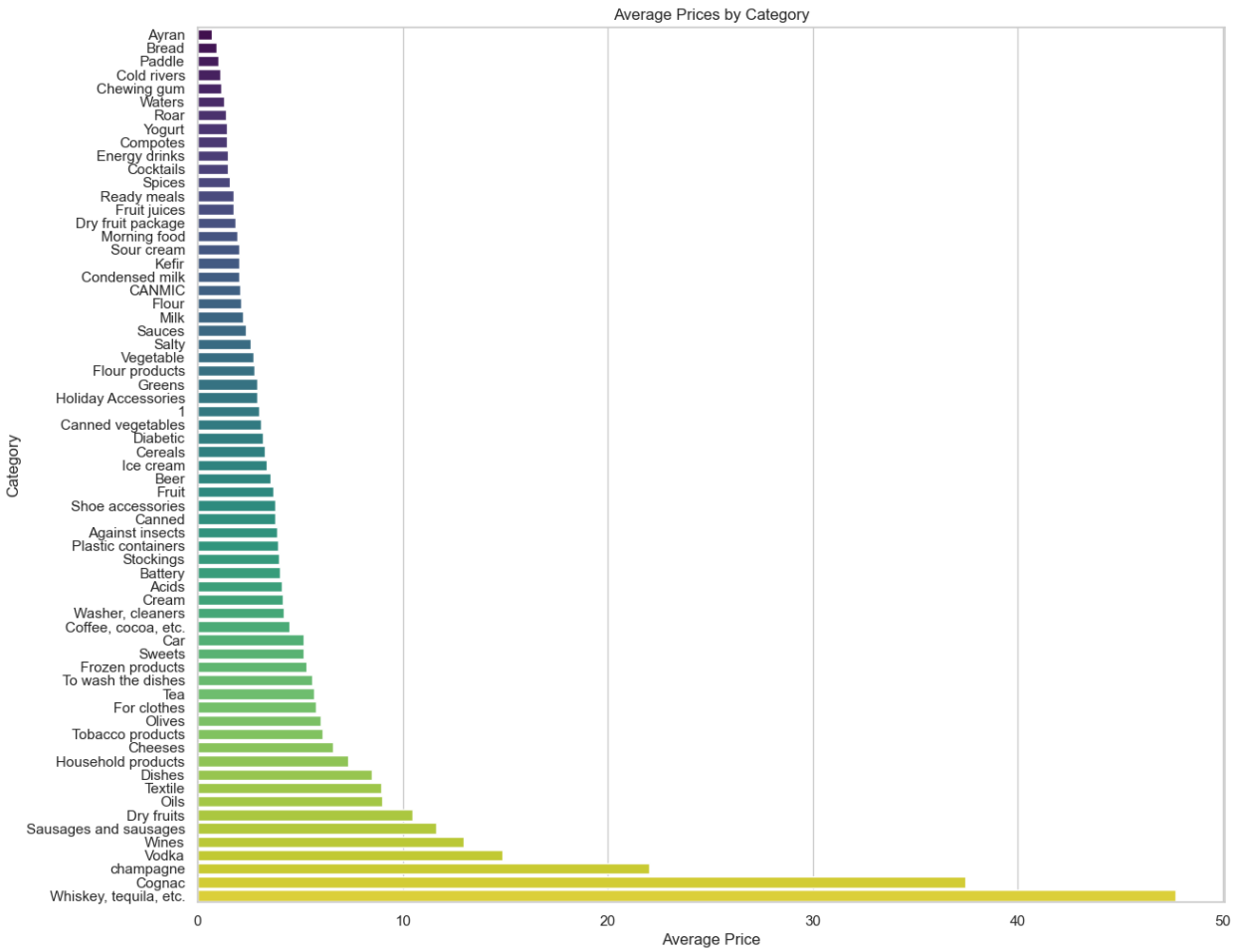


This Stack Barplot Distribution of Bonus Cart by Store Branch. Almost all the Store Branches have the same count of True and False Bonus Carts, only a few Store Branches like ‘Almond’ has a less count than others.

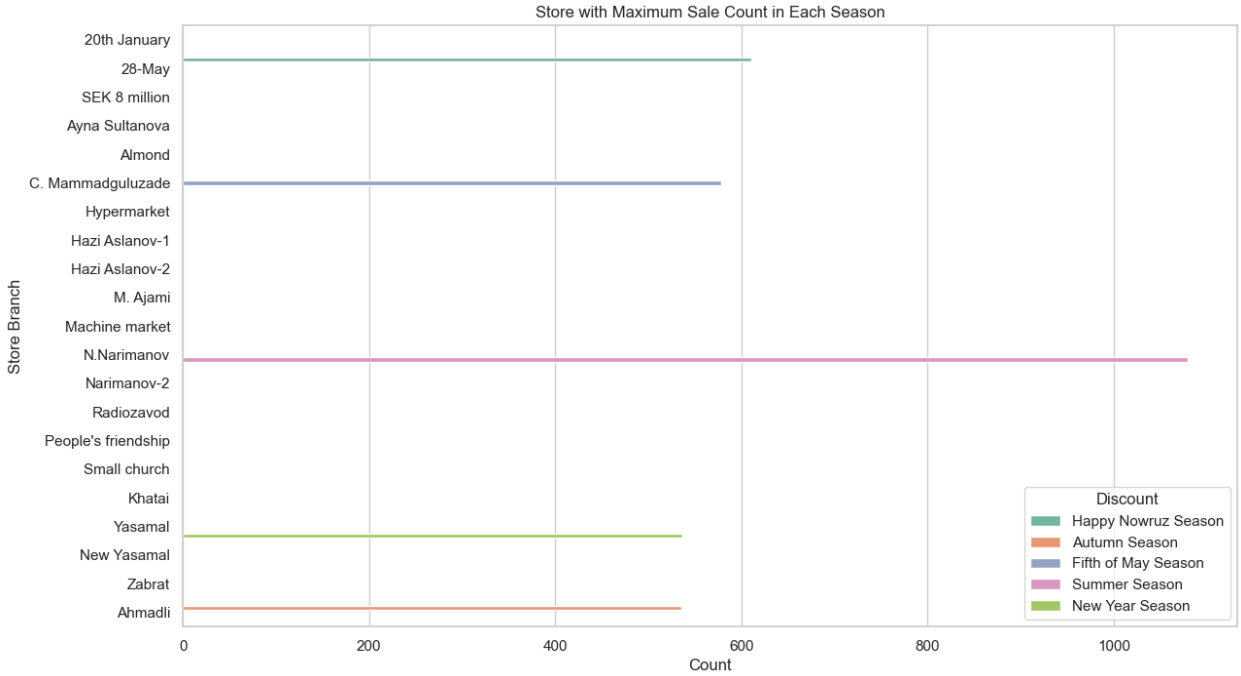


This is a group plot of the Avg Prices of Different Categories and plotted during various Discount Seasons. According to the graph the

Avg Prices are more in the Summer Season with “Whiskey, tequila, etc.” Category.

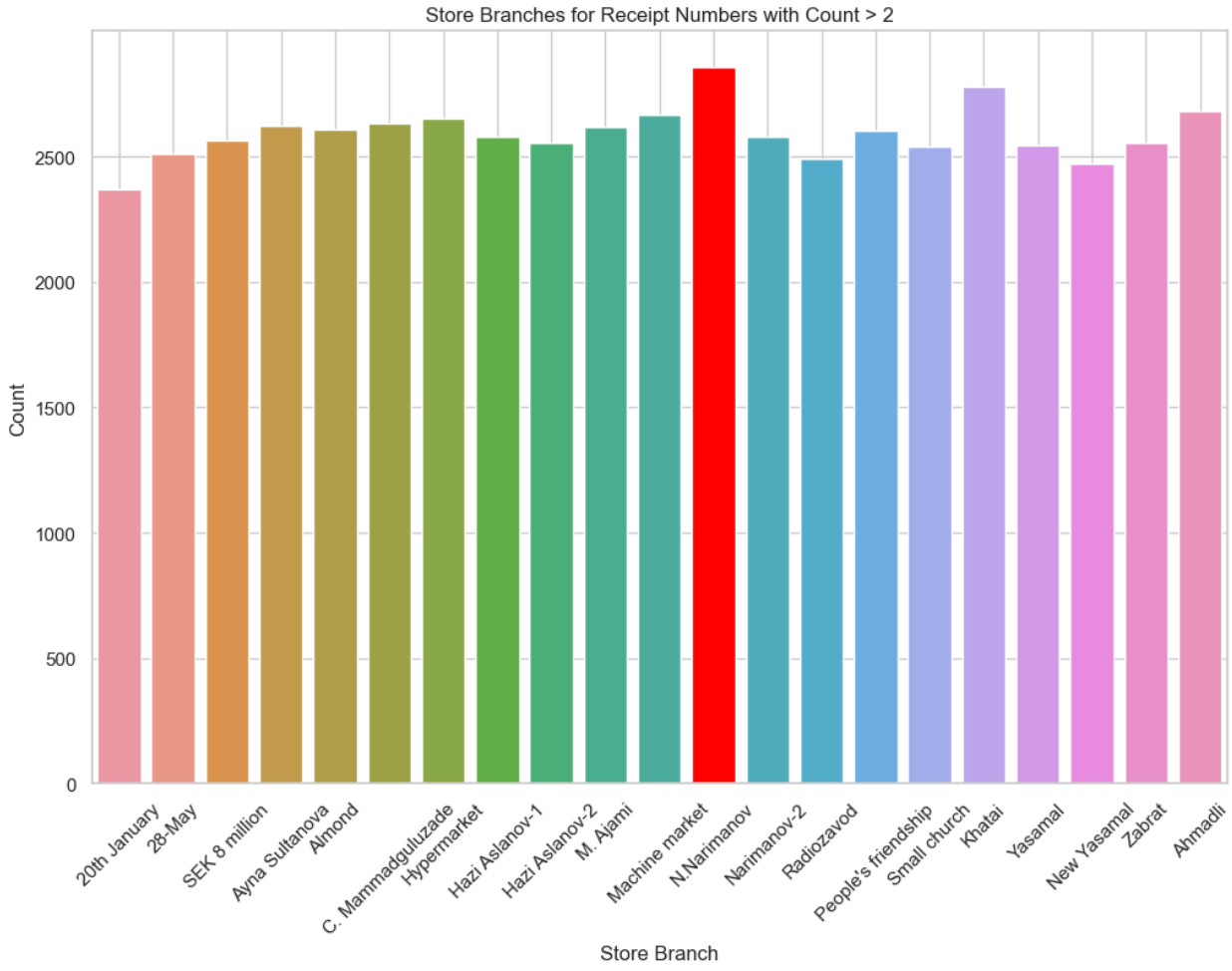


This is a barplot of Average Prices by various Categories. From the given above plot Category : “Whiskey, tequila, etc.” has the maximum average Price. This plot is sorted in ascending order.



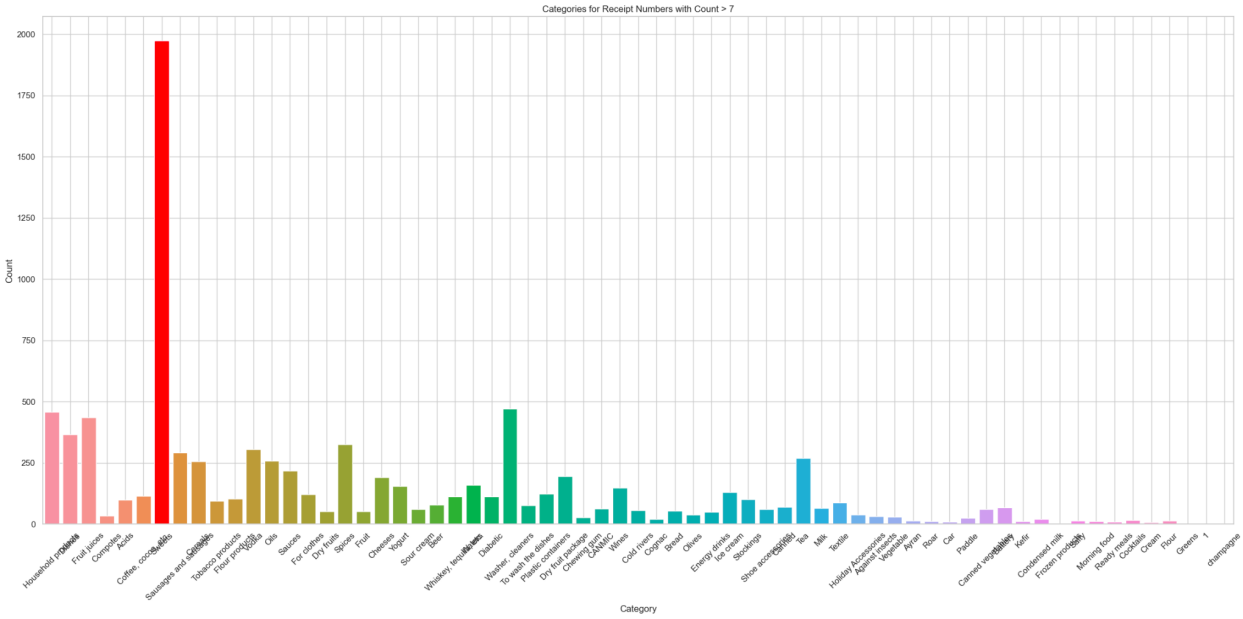
This bar plot gives which Store has the maximum “Product” sold count in each discount season. Each discount season is given a different color to signify which Store Branch sold most products in each discount season.

3. Count plot :

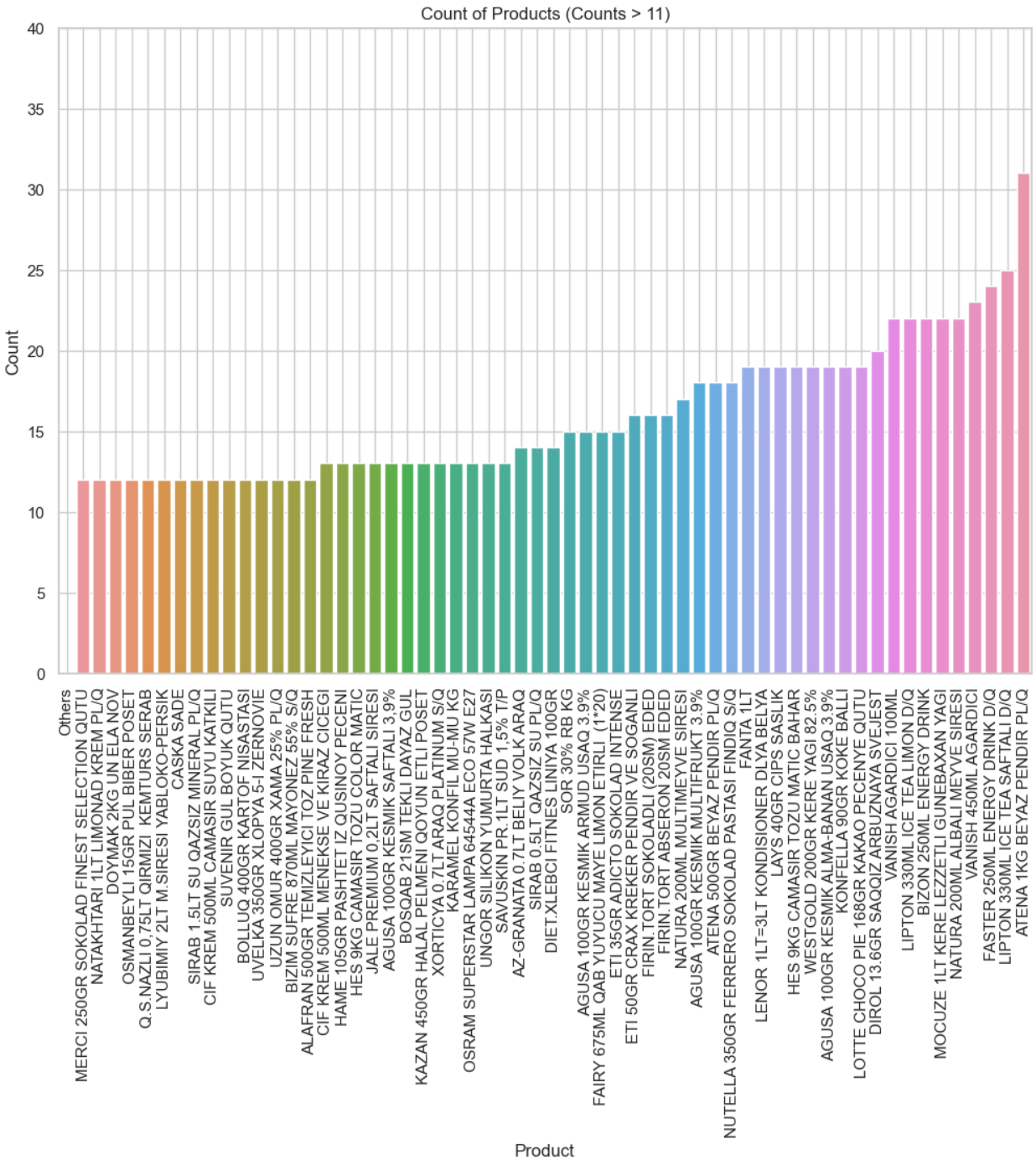



```
The store with the most counts is: N.Narimanov (Count: 2857)
```

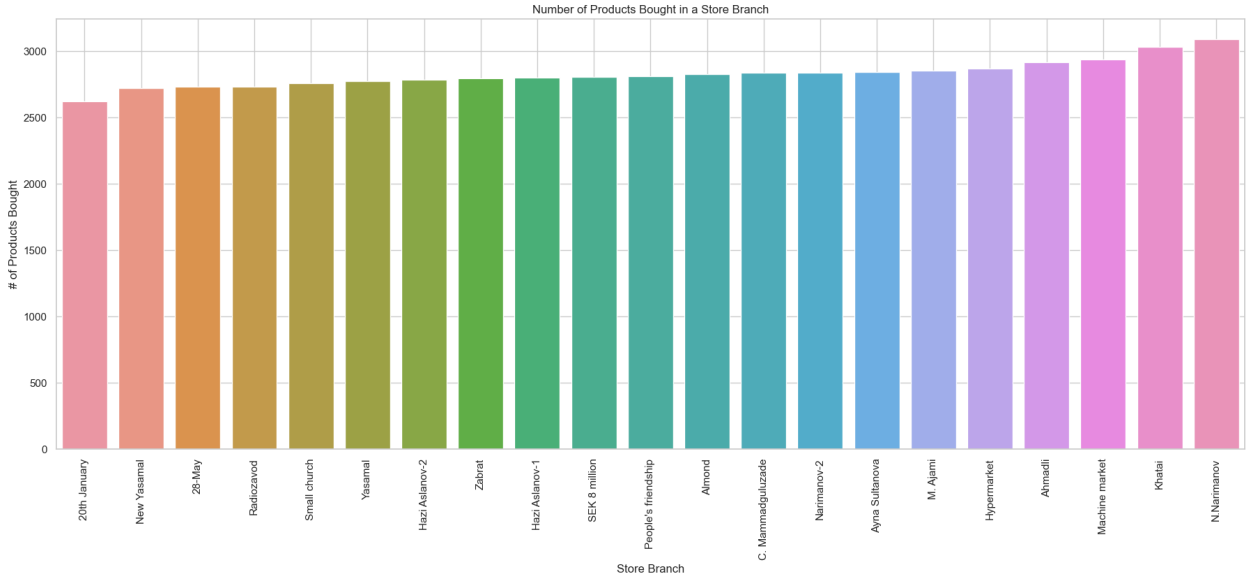
In this plot we are only considering the Receipt numbers with count more than 2, here the receipt number is signifying a person who has bought a number of products from the Supermarket. So, in this plot we can see the count of unique receipt numbers in each Store Branch, identifying which Store Branch had the most number of customers. According to the plot N. Narimanov Store Branch had the most counts of 2857.



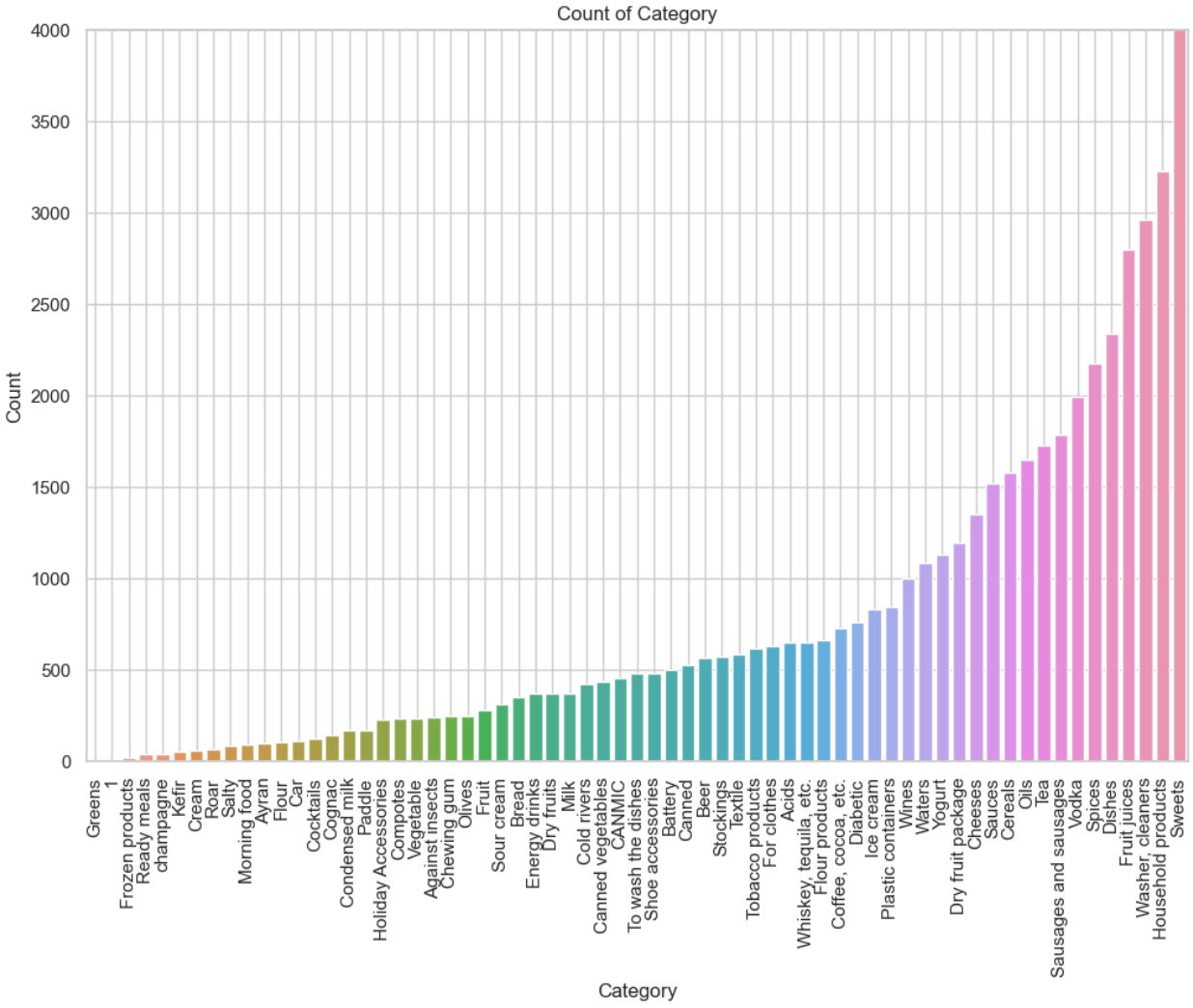
This is a simple count plot of various Categories. In this plot we can clearly see that “Sweets” Category has a clearer lead than other Categories.



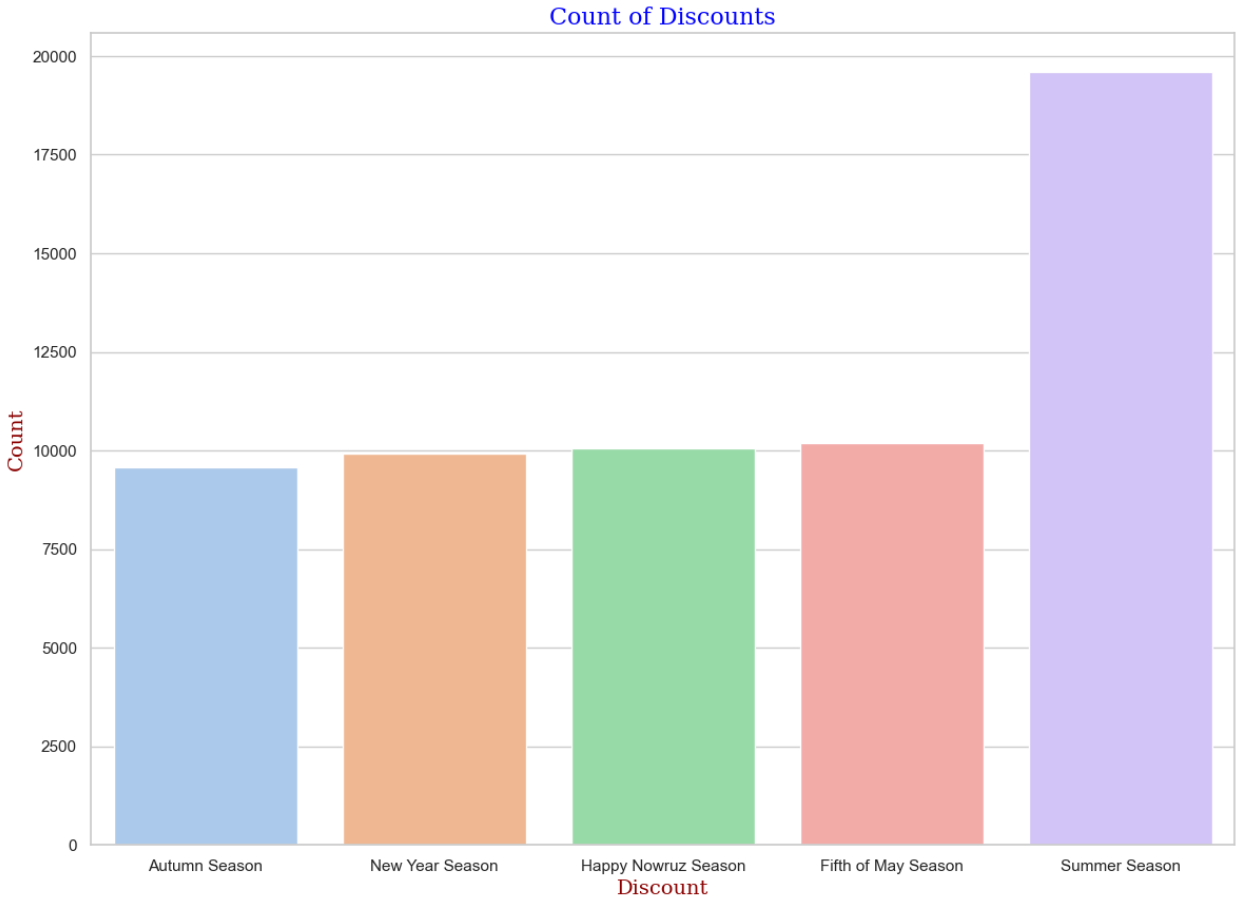
In this count plot only products bought whose count is more than 11 is kept, rest all are terms as ‘Others’. This is a countplot of Products, we can identify Atena 1 KG packet is most bought as per count. This plot is sorted in ascending order.



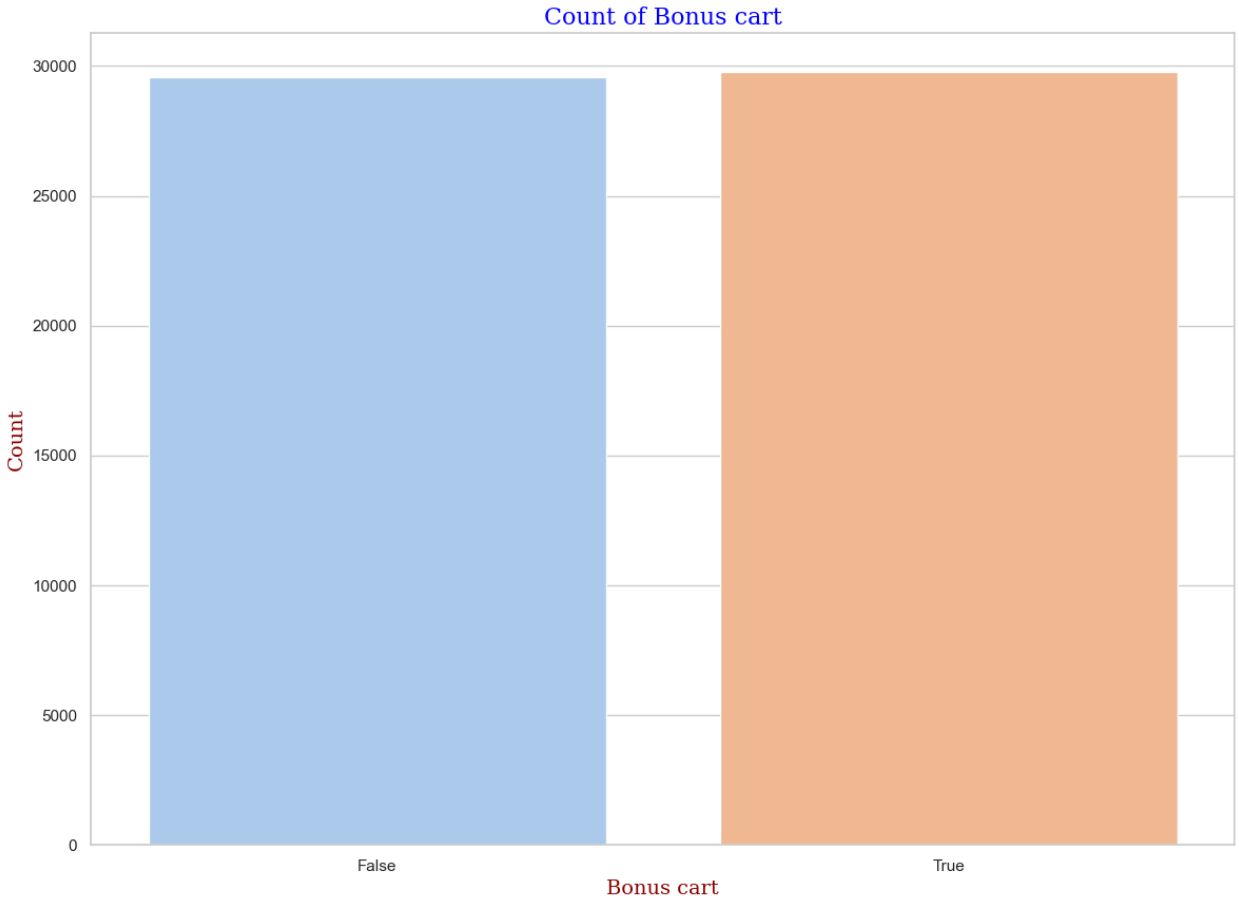
This is a simple count plot sorted in ascending order of Number of Products bought in the Store Branches. From the above graph N.Narimanov Store Branch had the most counts of 2857.



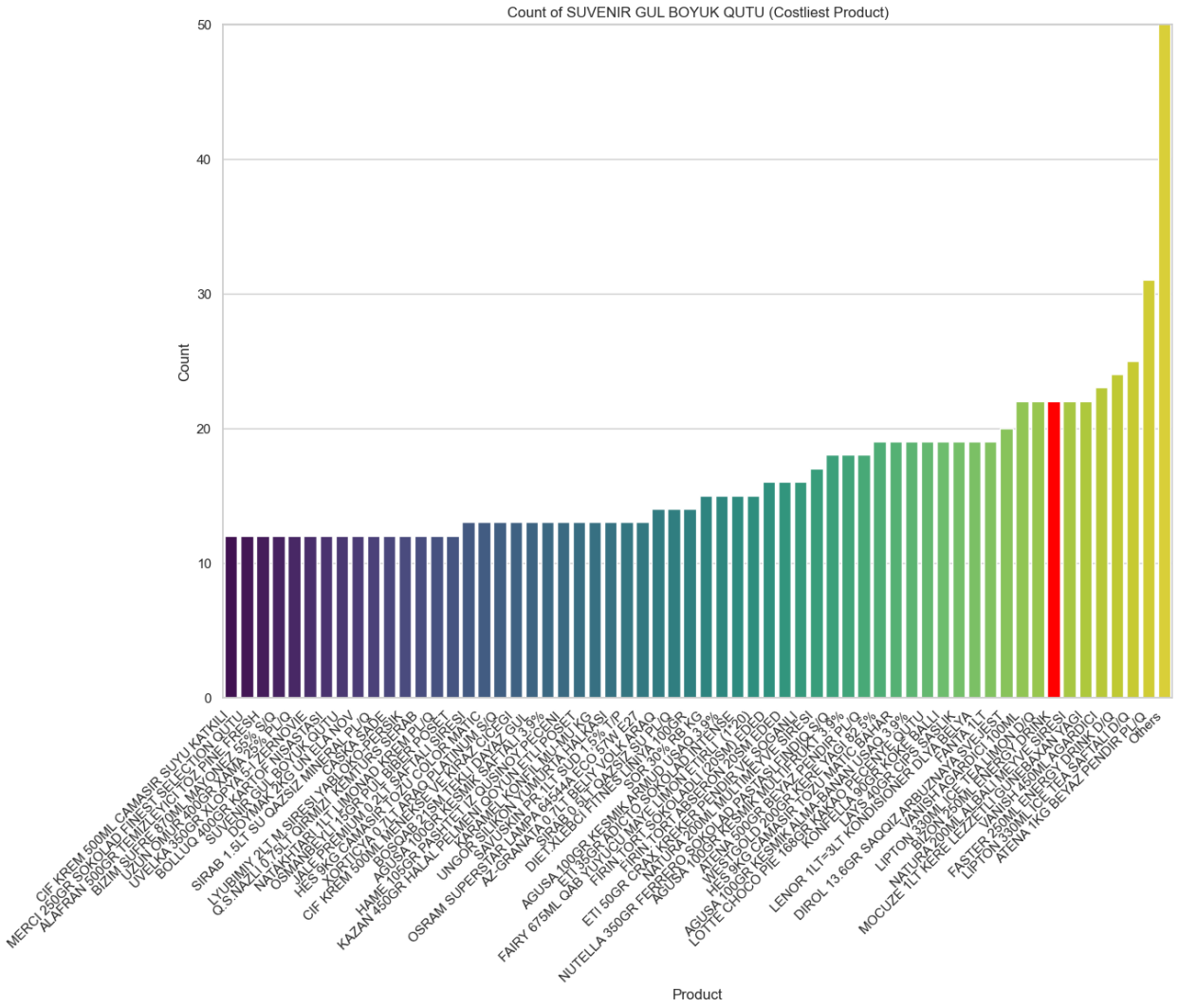
This is a simple countplot of Categories. From the countplot we can clearly see that “Sweet” Category has the highest count than others. The graph is sorted in ascending order so we can analyze which has the next bigger count.



This is a simple count plot of Discount Season where the products were sold in that particular season. From the above graph it is clear that Summer Season Discount has the highest count than all other discount seasons.

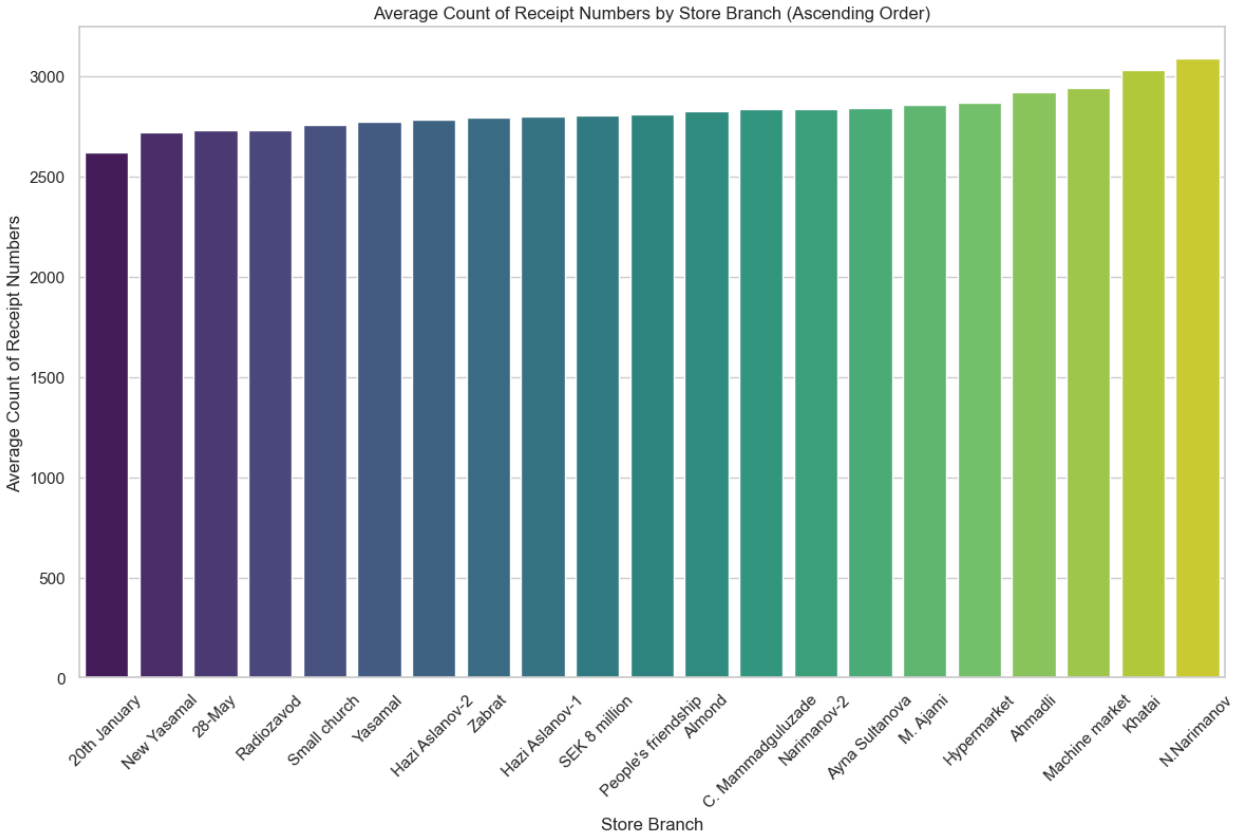


This is a simple count plot of Bonus Cart, we can clearly see that both True and False got the same count. Therefore, there is a 50% probability that a product got a bonus cart or not.



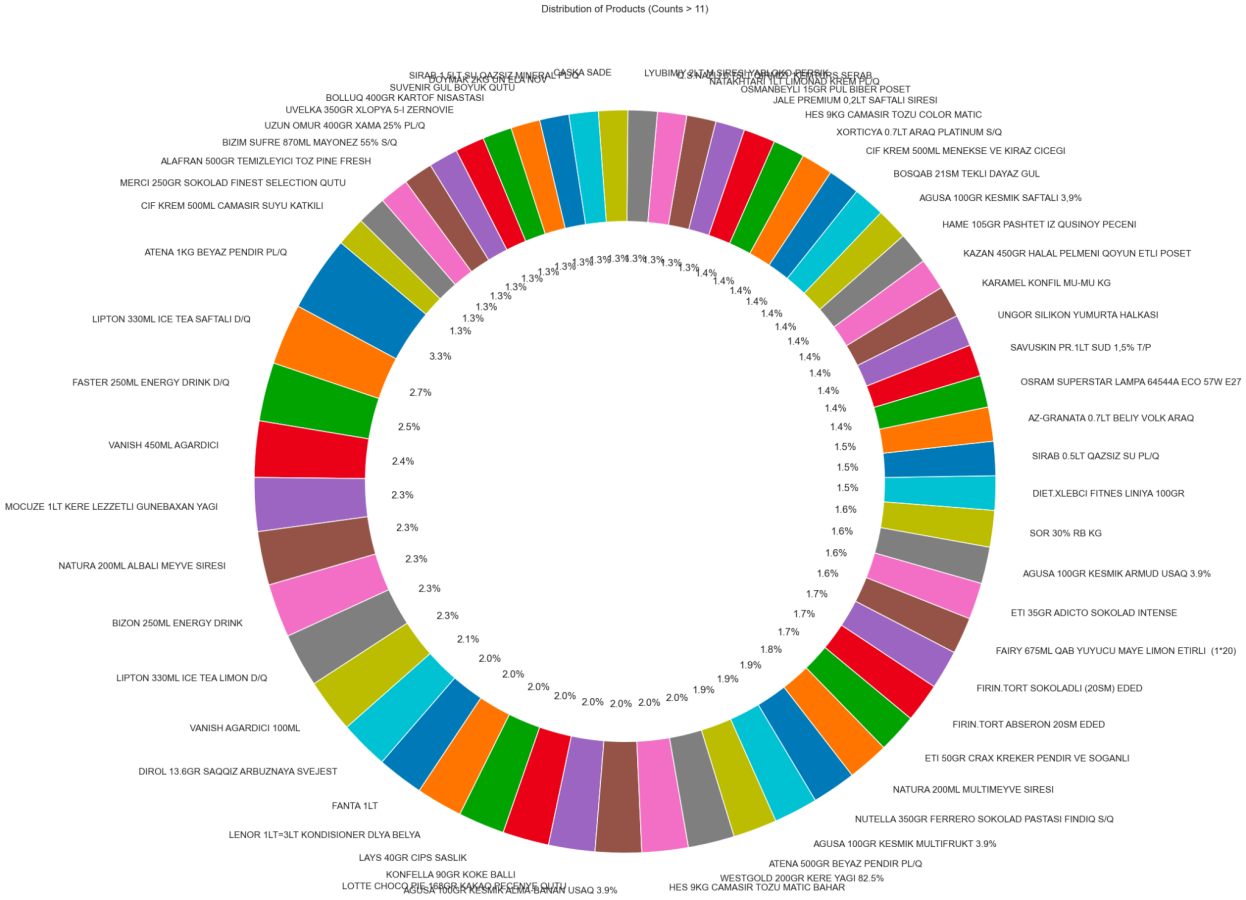

```
Costliest Product: SUVENIR GUL BOYUK QUTU
Count: 12
Average Price: $46.20
```

This is a count plot of all Products with counts more than 11, in this plot I wanted to show which Product has the maximum price and signify by red color and compare its count with all other less priced Products. According to the graph the costliest Product is a Souvenir of count 12 and Price $46.50, its count is quite less than all other less priced Products. This plot is sorted in ascending order.

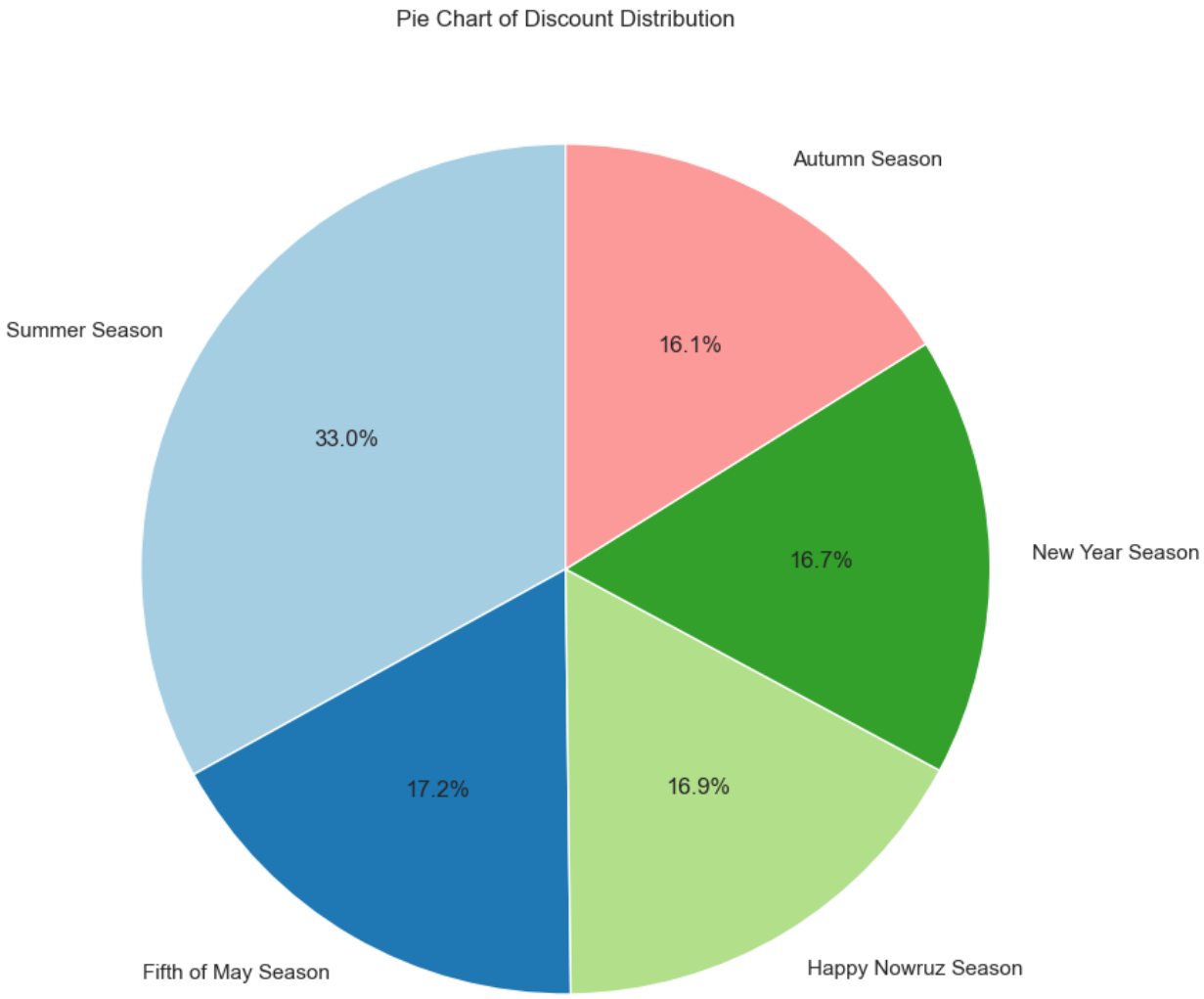


This is a simple count plot where we get the avg count of receipt numbers in each Store Branch. So according to the graph the Store Branch N. Narmimanov has the most avg count of receipt numbers.

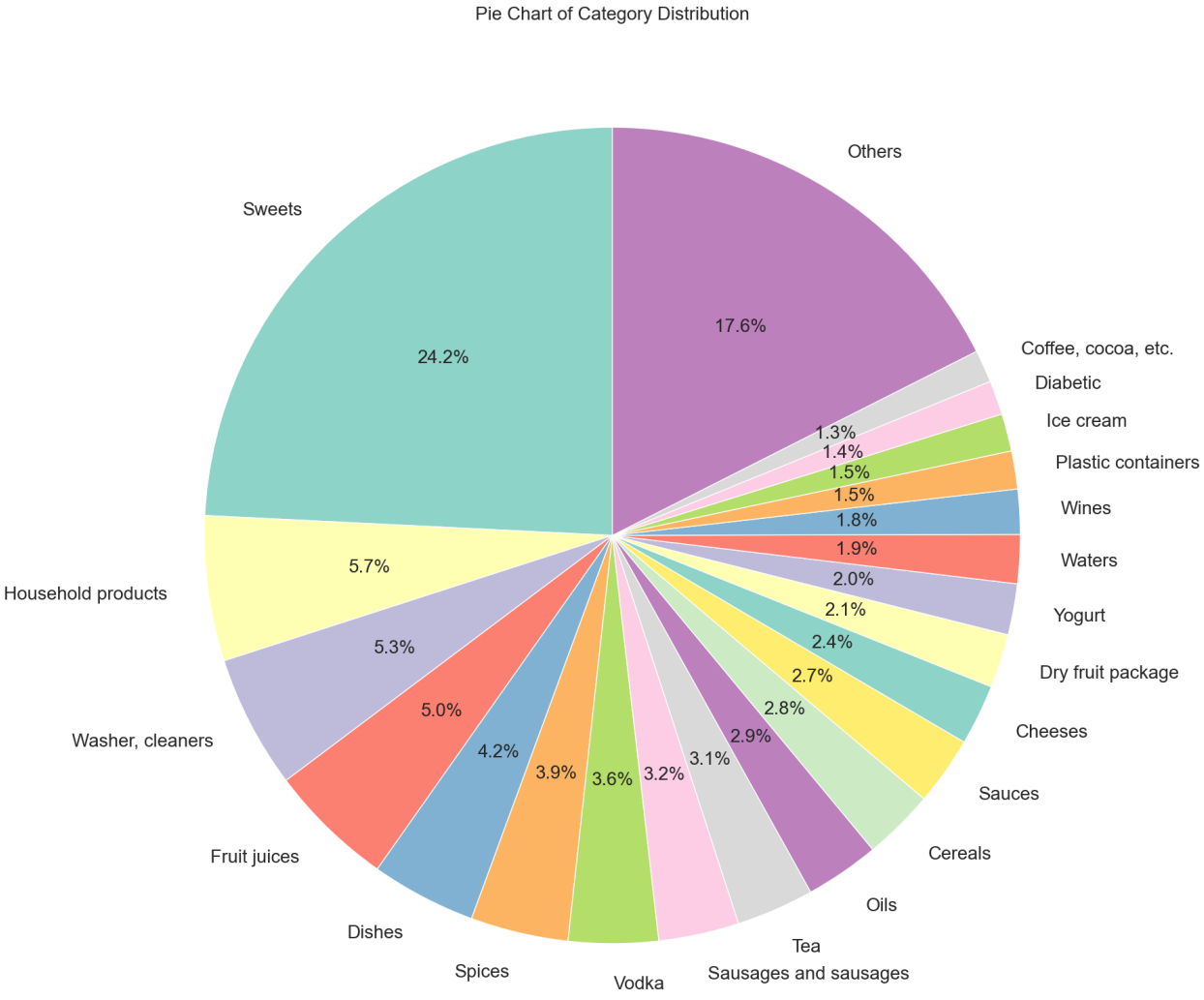
4. Pie chart :



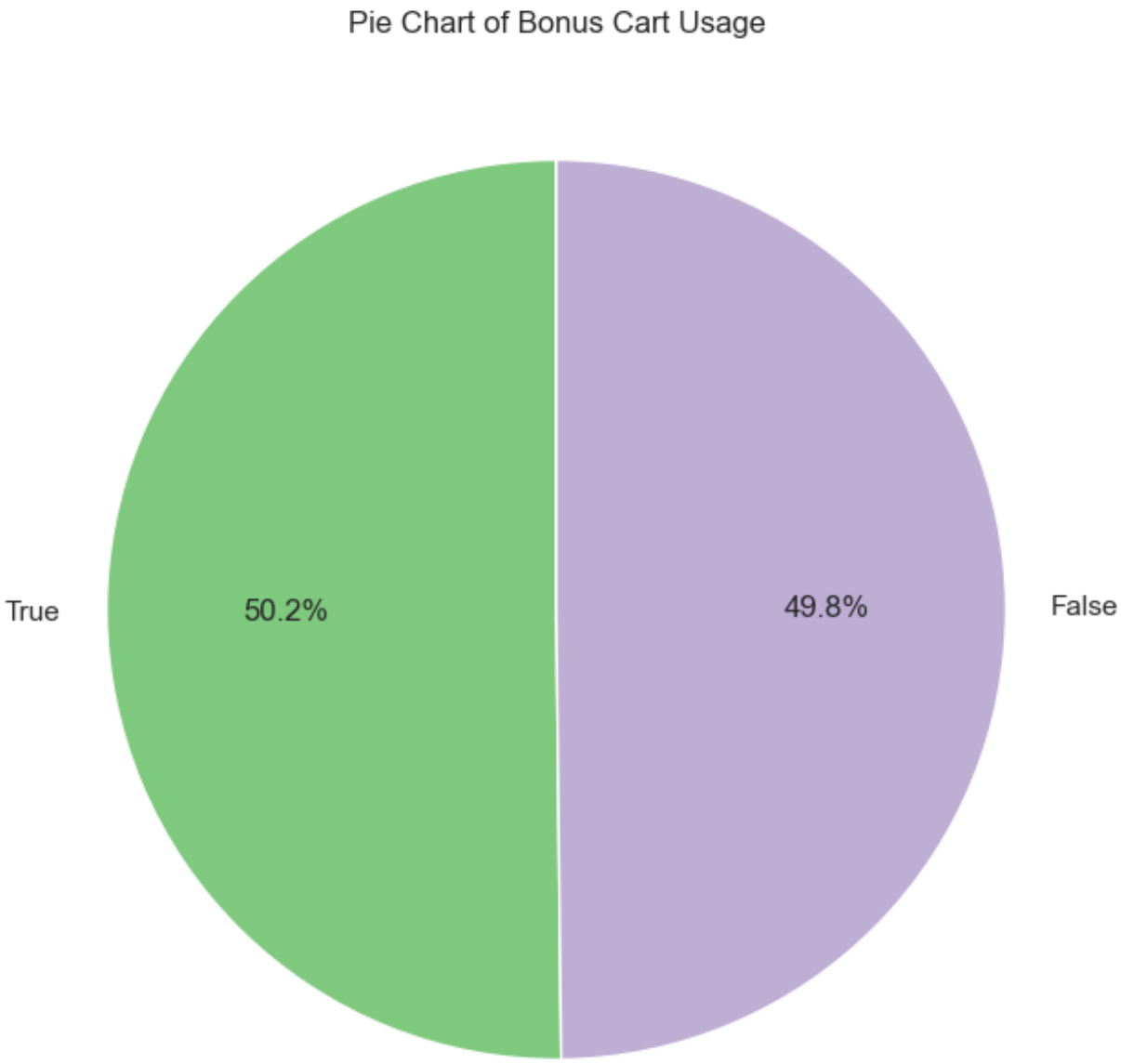
This is a pie plot of distribution of Products that were sold with count more than 11. From the graph it is clearly visible that almost all Products have the same percentage of distribution from 1.1% to 2%. From this graph we can also see that the number of unique products are huge and a wide variety of them.



This is a simple pie plot used to identify the distribution of Discount seasons in the dataset. It is clearly visible that the Summer season has the higher percentage than the rest of the seasons.

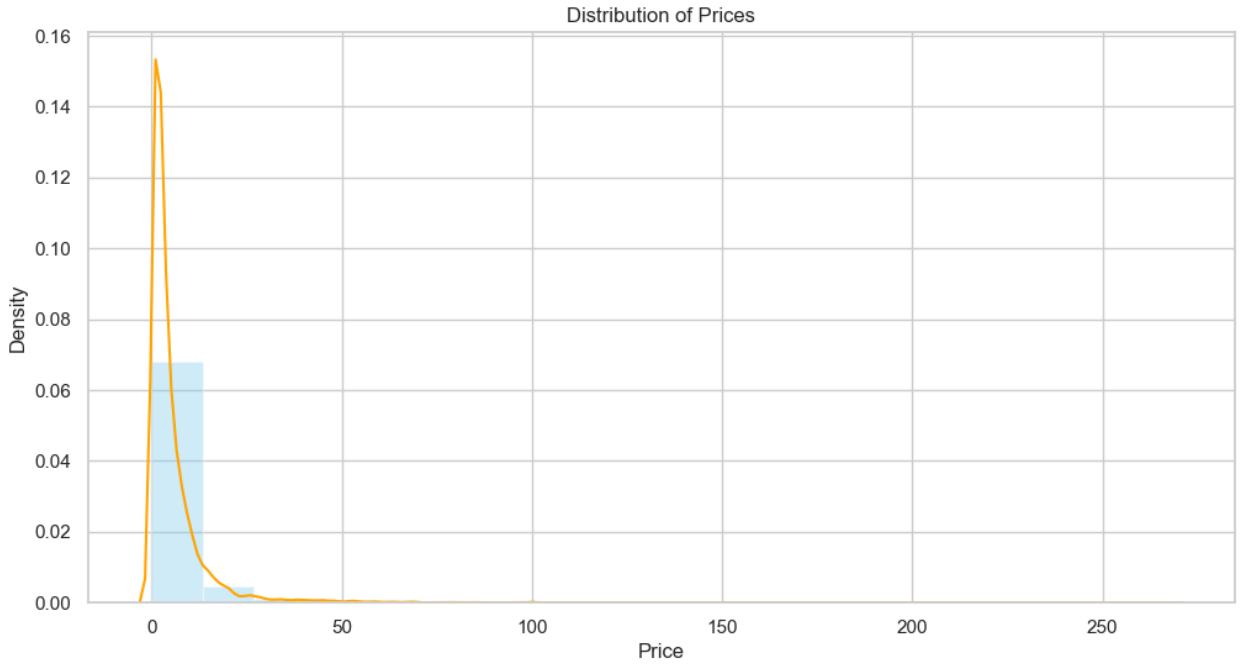


This is a simple pie plot used to represent the distribution of various categories in the Supermarket dataset. The Sweets category has the highest 24.2% than all other categories.



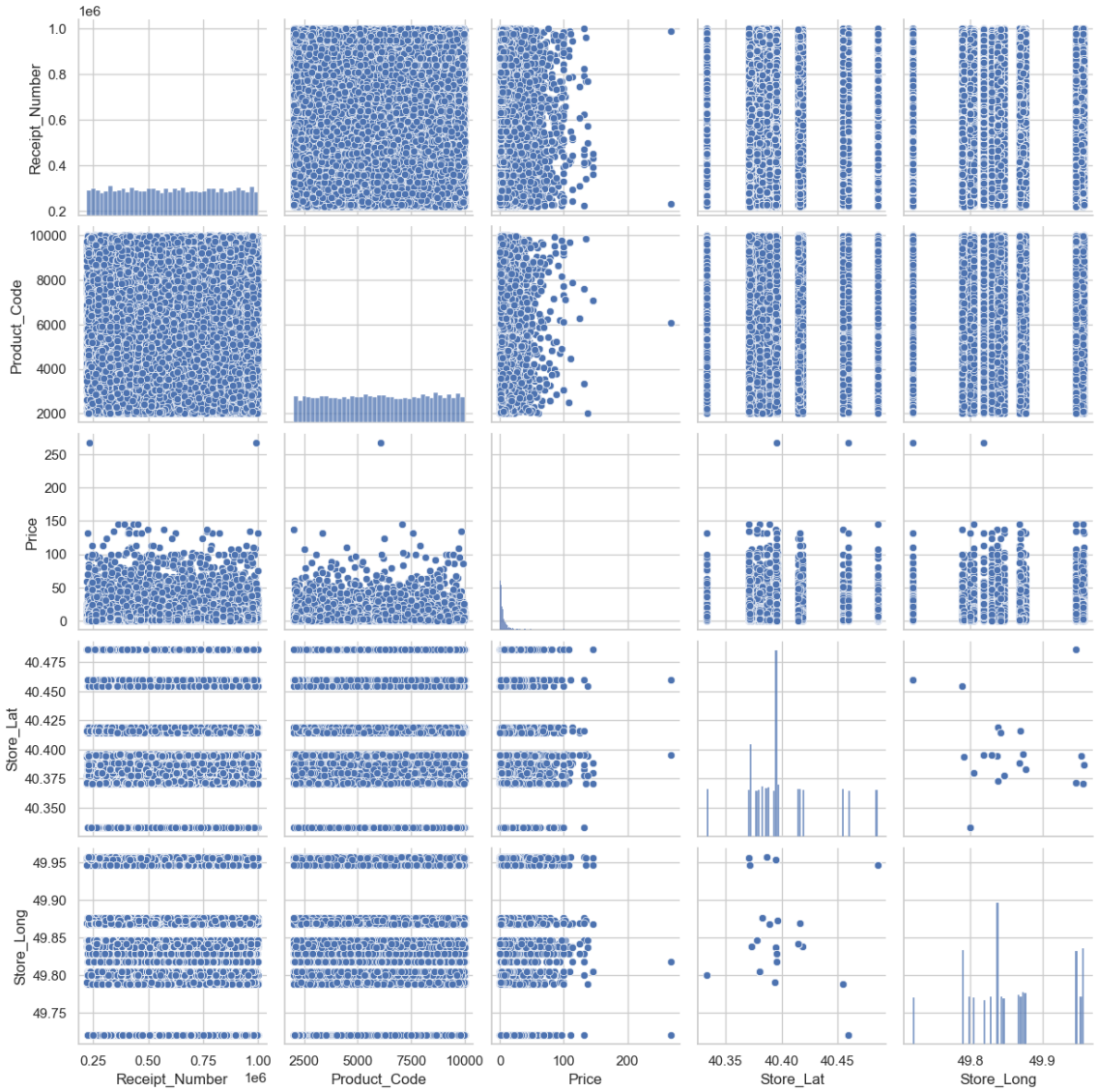
This is a simple plot of distribution of Bonus Cart features in our dataset. You can clearly see a given product will have a 50-50% probability to get a True or False Bonus Cart.

5. Dist plot :



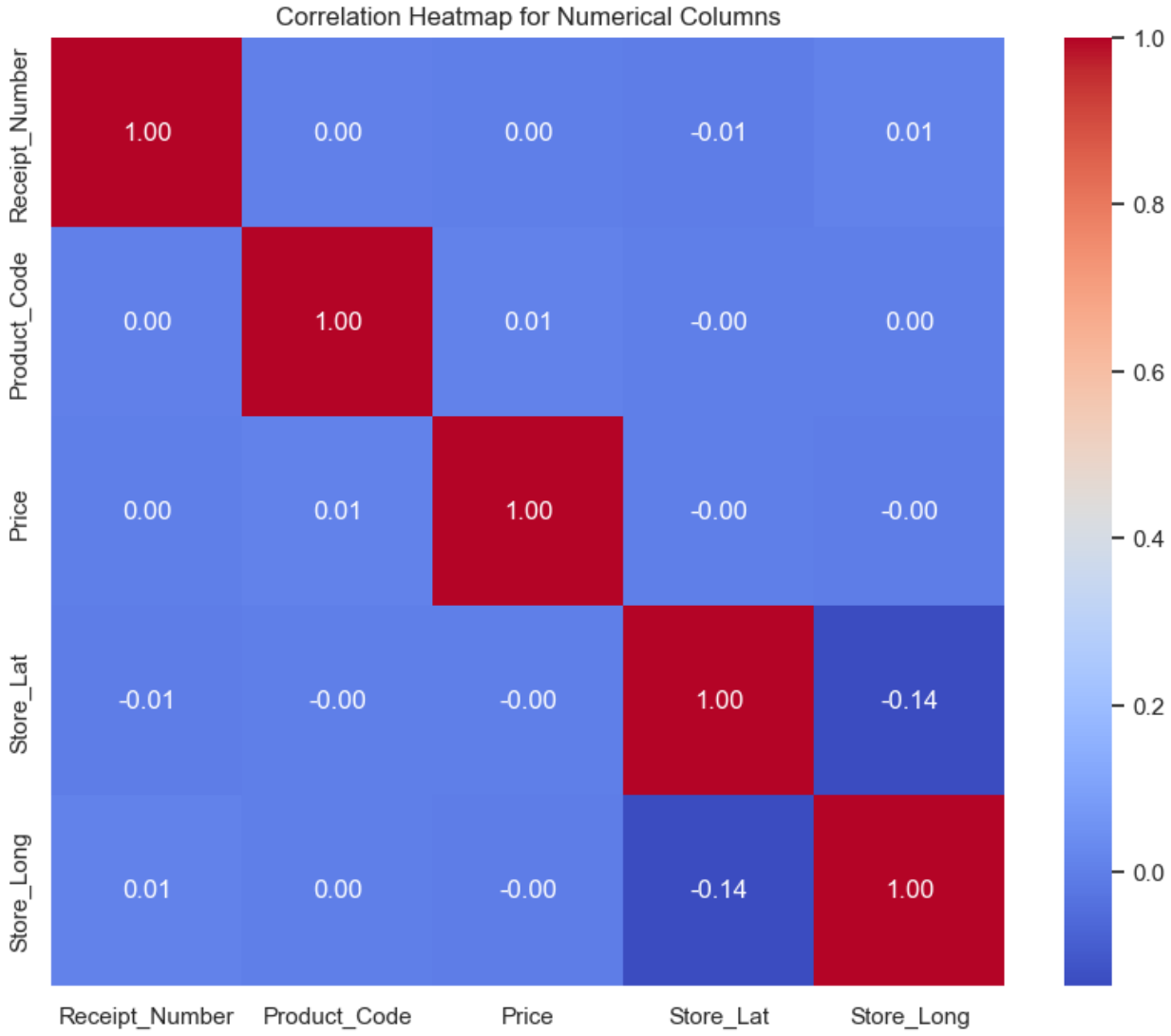
A distplot, also known as a distribution plot, is a type of graph used to visualize the distribution of a single variable. It is similar to a histogram, but it also includes a kernel density estimation (KDE) plot and a rug plot. This is the distribution plot of the “Price” dependent variable. We can see that it is a right skewed graph, meaning the prices are quite low for all the products ranging from 0 to 50. The median price is lower than the mean price, suggesting that there are more outliers on the higher end of the distribution.

6. Pair plot :



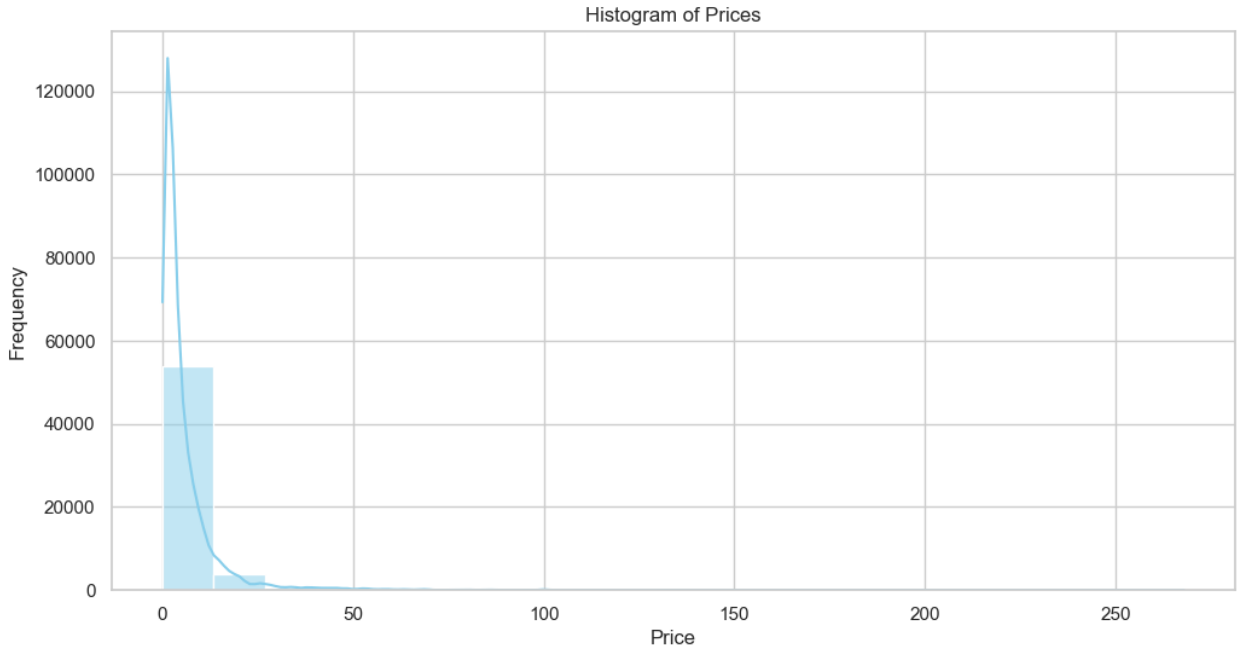
A pair plot is a great visualization tool to observe relationships between different pairs of variables in a dataset. The diagonal elements of the pair plot typically show the distribution of individual variables, while the scatterplots on the lower and upper triangles show the relationships between pairs of variables. From this pairplot we can figure out that all columns are independent of each other and are not linearly or in any fashion related to each other.

7. Heatmap with cbar :



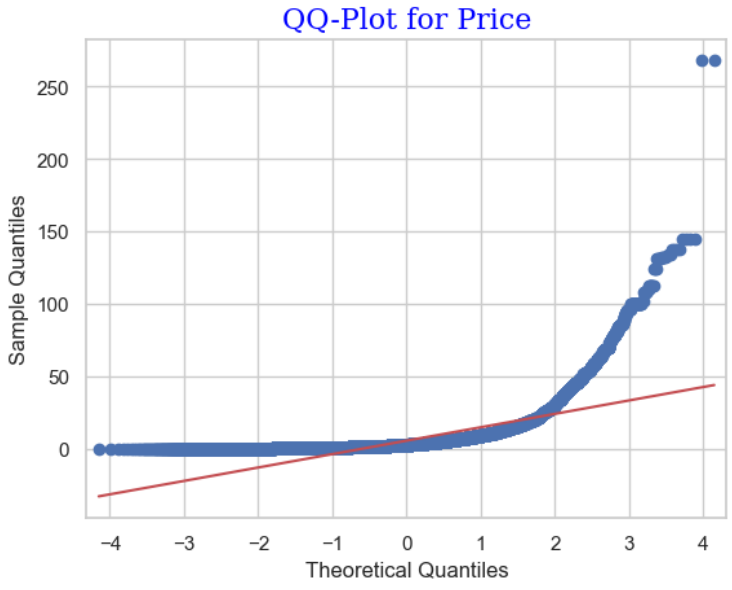
From the heatmap, it is observed that only 'Store_Lat' and 'Store_Long' exhibit a discernible correlation with each other. The correlation coefficient of -0.14 indicates a weak inverse relationship between the latitude and longitude of the supermarket branches. The negative sign suggests that as one variable increases, the other tends to decrease, albeit modestly.

8. Histogram plot with KDE :



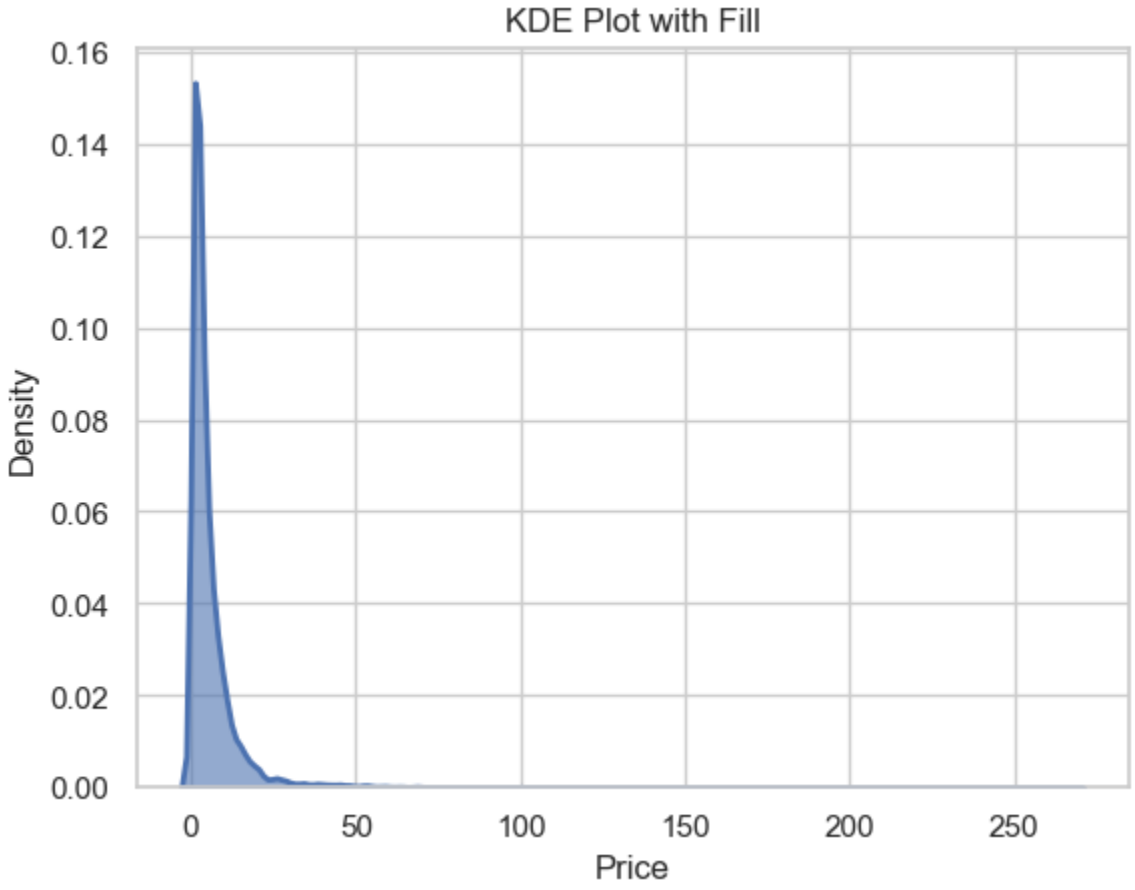
This is the Histogram plot with KDE of the “Price” dependent variable. We can see that it is a right skewed graph, meaning the prices are quite low for all the products ranging from 0 to 50.

9. QQ-plot :



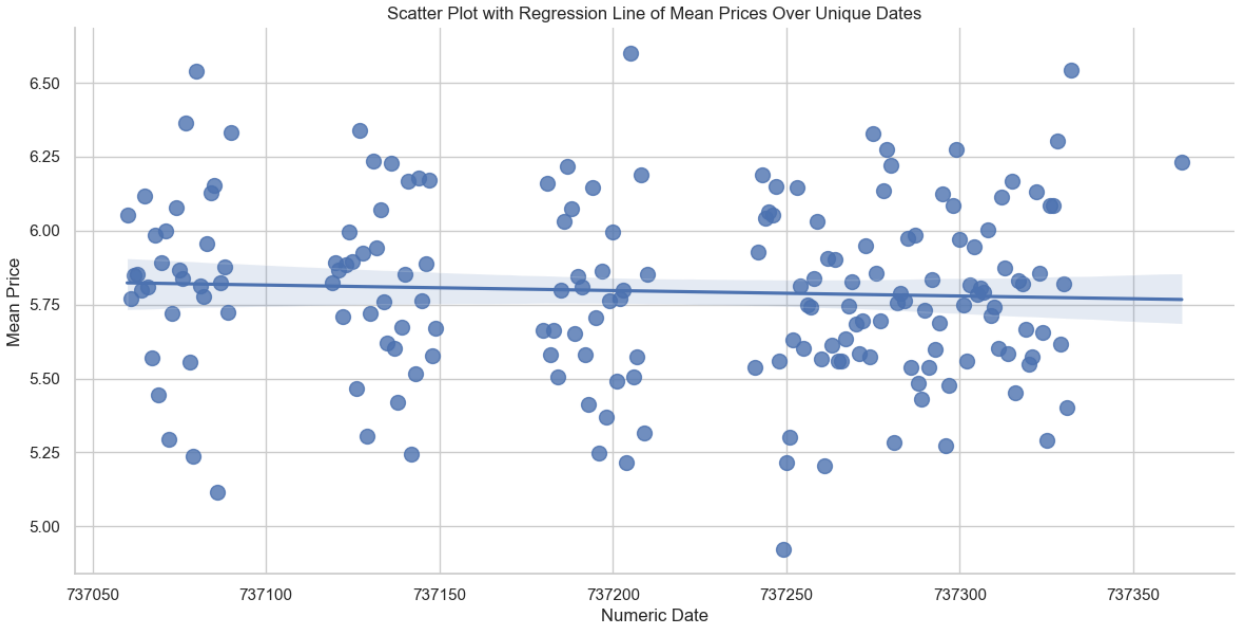
A Q-Q (Quantile-Quantile) plot is a graphical tool used to assess whether a dataset follows a particular theoretical distribution, such as the normal distribution. It compares the quantiles of the observed data against the quantiles of the expected distribution. From this qq plot of the Price feature, we can clearly see it is not normally distributed and also has many outliers making it a skewed graph.

10. KDE plot will fill, alpha = 0.6 :



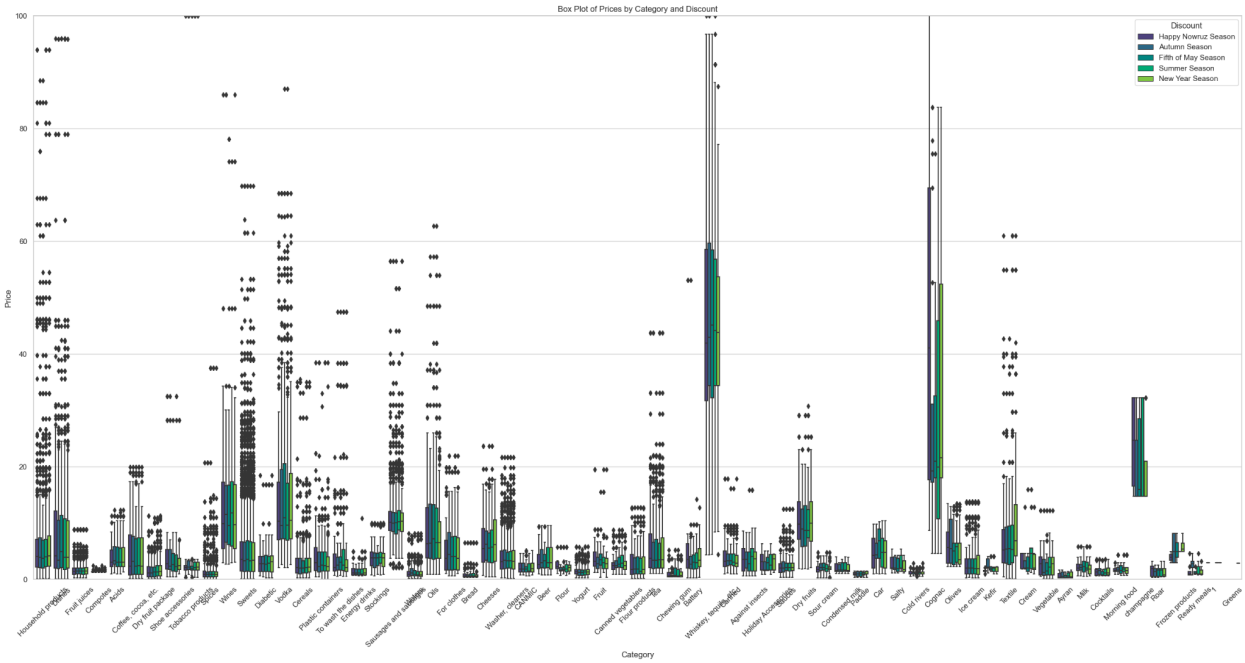
This is the KDE plot with the fill of the “Price” dependent variable. We can see that it is a right skewed graph, meaning the price probability is quite low for all the products ranging from $0 to $50.

11. lm or reg plot with scatter representation and regression line :



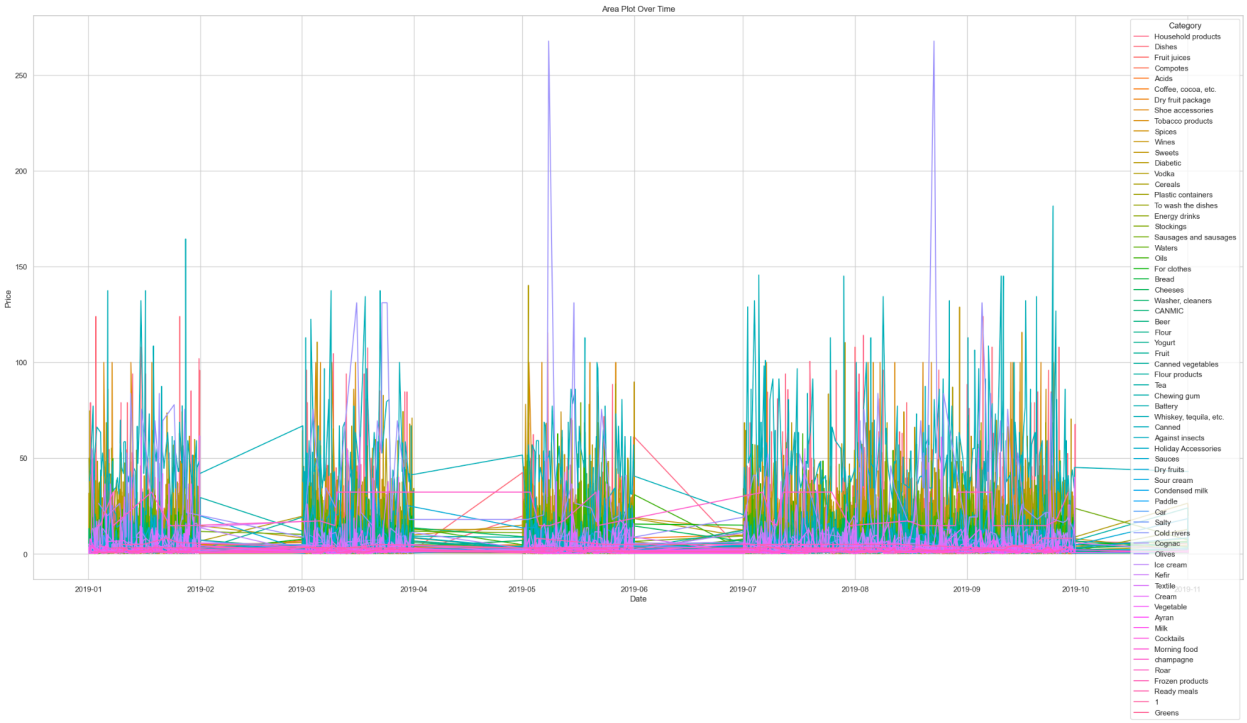
This is a lm plot or a regression plot of Prices over the time. Regression plot shows a trend of target features over the time. In this plot the mean prices over the time ranges from $5.75 to $6. Overall, the image suggests that the price of liquid data is trending upward over time. However, there is some volatility in prices due to a number of factors, including inflation, technological advancements, and changing market conditions.

12. Multivariate Box or Boxen plot :



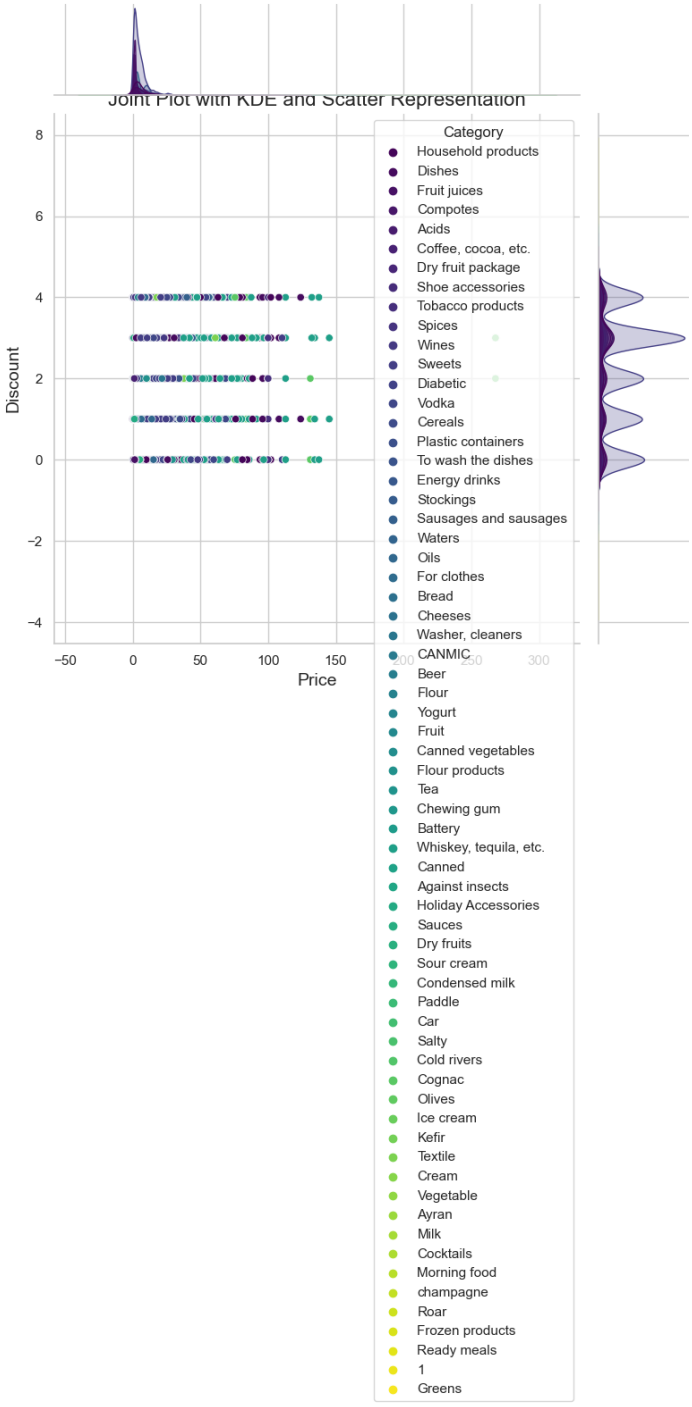
Multivariate boxplot is plotted over here of Prices grouped by Category and Discount. Different colors are given to different boxplots denoting the discount season. Box plot of prices are shown for every category where we can see its quartile range and also identify any outliers. From the given graph we can see there are many outliers for the categories who are on the left of the graph and huge quartile range with categories which are on the right of the graph.

13. Area plot :



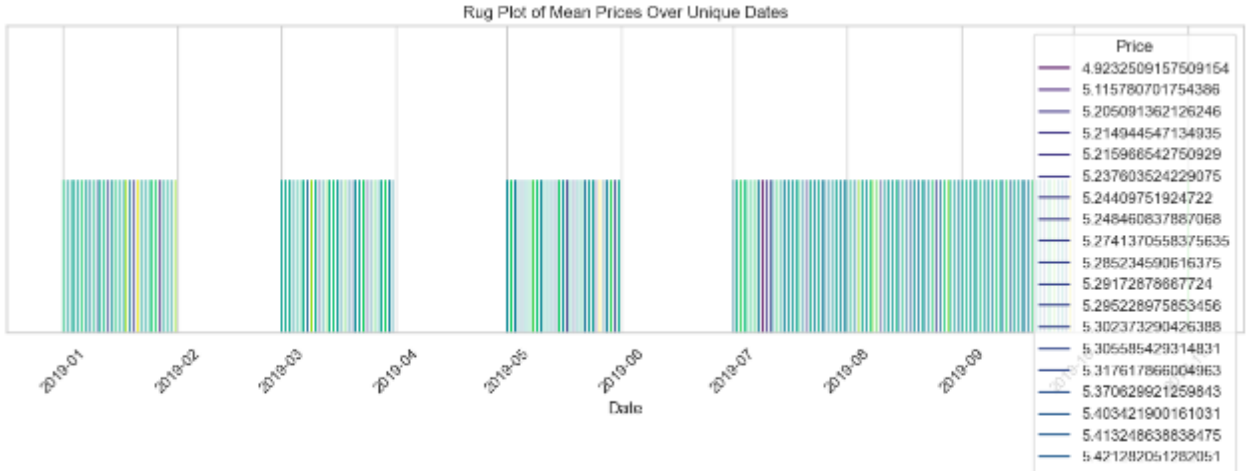
An area plot, also known as a filled area plot, represents data points using filled areas between the data and a baseline. In this area plot all the areas of all the categories are plotted over the time of the year 2019. We can clearly see the different and varied areas of different Categories, some have area's ranging from $0 to $10, others have $0 to $250 who might have higher price rates.

14. Joint plot with KDE and scatter representation :



The overall trend in prices is upward. The line of best fit slopes upward from left to right, indicating that prices have generally increased over time. There is some volatility in prices. The data points are not all perfectly aligned along the line of best fit. This suggests that there are factors other than time that can influence prices, such as supply and demand, economic conditions, and geopolitical events. There is a positive correlation between date and price. This means that, as the date increases, the price tends to increase as well. However, there are some exceptions to this rule. For example, there are a few data points that show prices declining even though the date is increasing.

15. Rug plot :



The image shows a long list of numbers on a white background. The numbers are in ascending order, from smallest to largest. The list is divided into four columns: " Price", "Date", "Store Branch", and "Category". The first column, " Price", shows the price of liquid data in SEK (Swedish kronor). The second column, "Date", shows the date on which the data was collected. The third column, "Store Branch", shows the name of the store branch where the data was collected. The fourth column, "Category", shows the category of the store branch.

Based on this information, we can make the following observations:

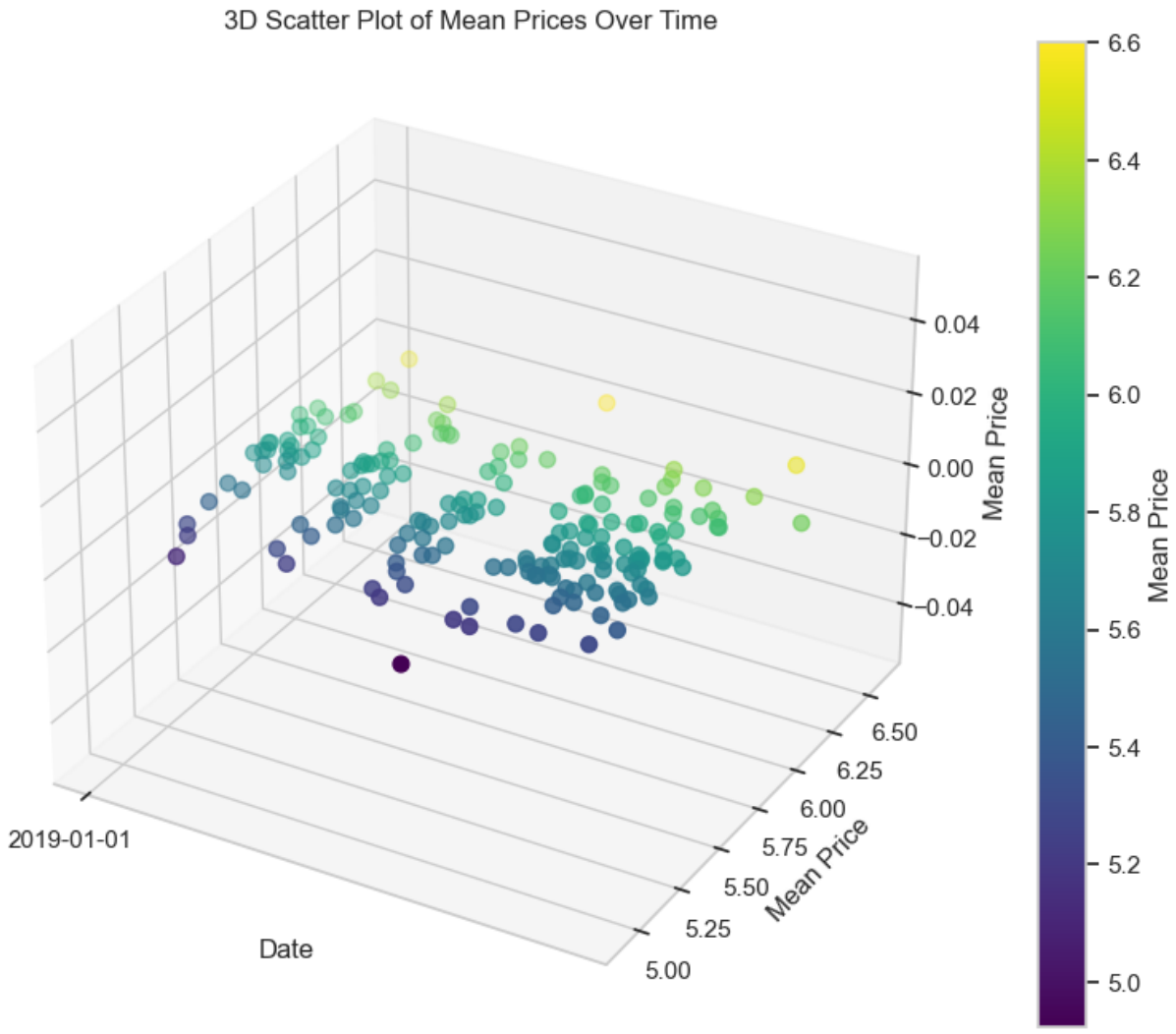
- The most expensive liquid data is sold at the "20th January" and "28-May" store branches.
- The least expensive liquid data is sold at the "Ahmadli" and "Zabrat" store branches.
- The highest median price for liquid data is in the "Hypermarket" category.
- The lowest median price for liquid data is in the "Small church" and "Khatai" categories.
- The price of liquid data has generally increased over time, with the highest prices being seen in the most recent data points.

We can also make some inferences from the data. For example, we can infer that:

- Consumers who are looking for the best deal on liquid data should shop at the "Ahmadli" or "Zabrat" store branches.
- Consumers who are looking for the most variety of liquid data should shop at a "Hypermarket" or "Household" store branch.
- Consumers who are looking for the highest quality liquid data may want to be willing to pay a higher price.
- The demand for liquid data is increasing over time, which is driving up prices.

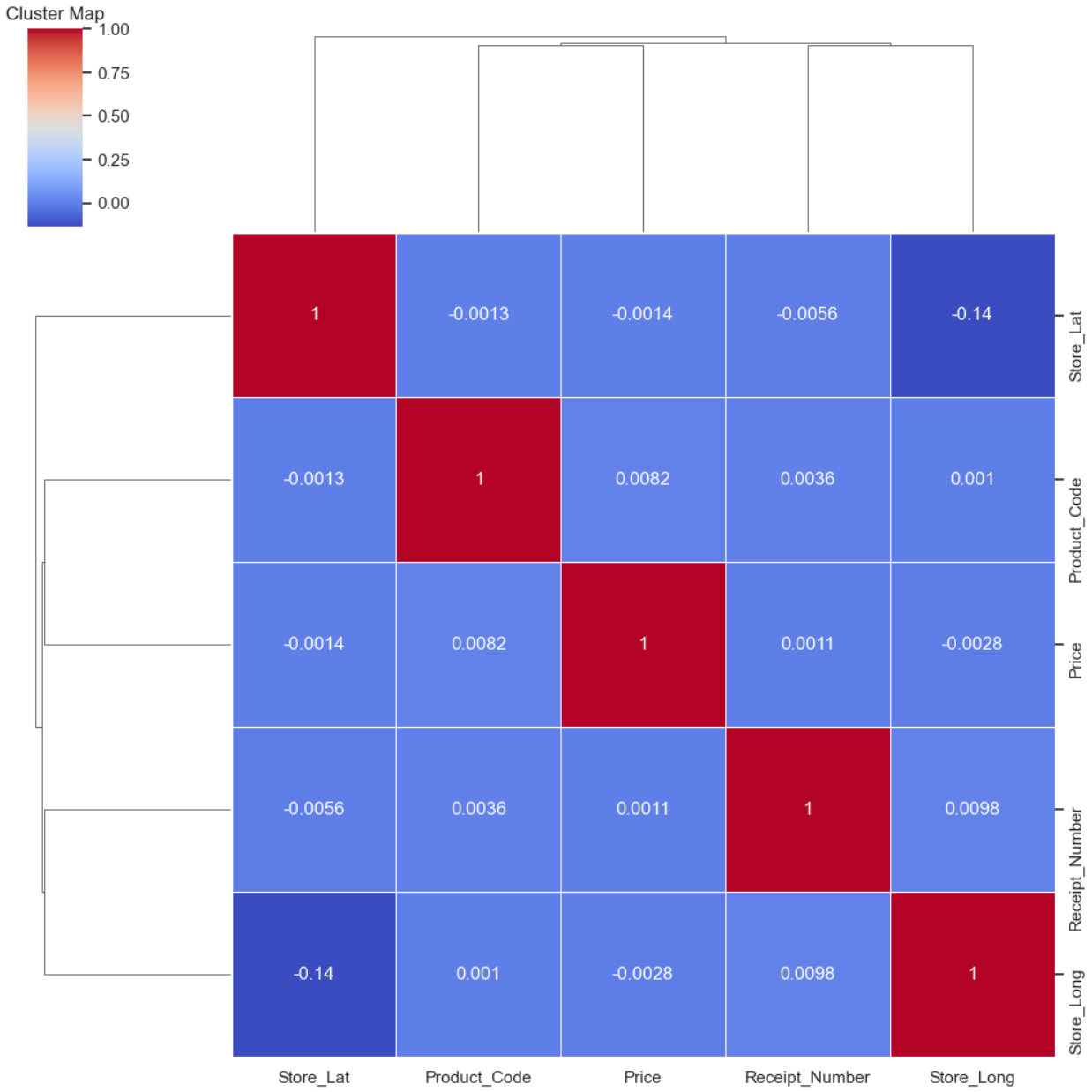
It is important to note that these are just a few observations and inferences that can be made from the data. More detailed analysis would be needed to draw more definitive conclusions.

16. 3D plot and contour plot



This is a 3D plot or you can say a scatter plot turned into a 3D plot. It's basically a scatter plot of mean prices over the time. There is a color scheme for the mean prices specific to its range.

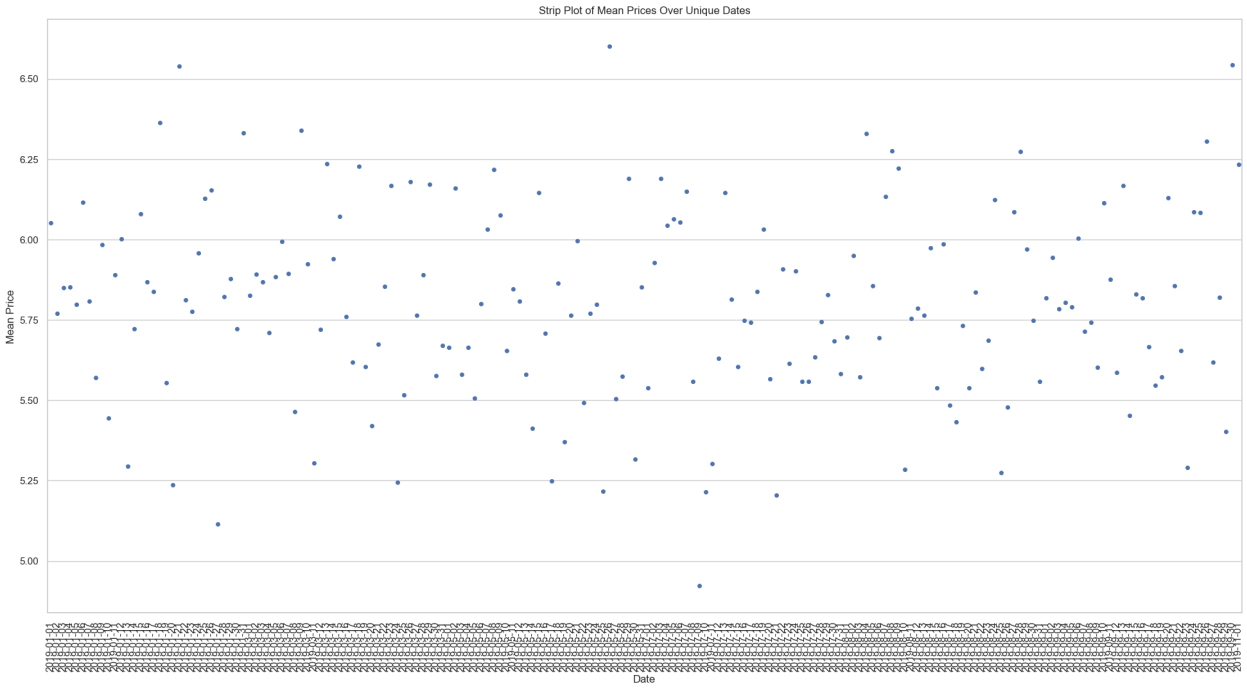
17. Cluster map :



A cluster map is a visual representation of data that has been grouped into clusters. It combines two key steps in data analysis: Clustering: Identifying and grouping similar data points together. Mapping: Visualizing the resulting clusters on a map. This combination allows you to see patterns and relationships in your data that would be difficult to identify otherwise. Here the cluster map is

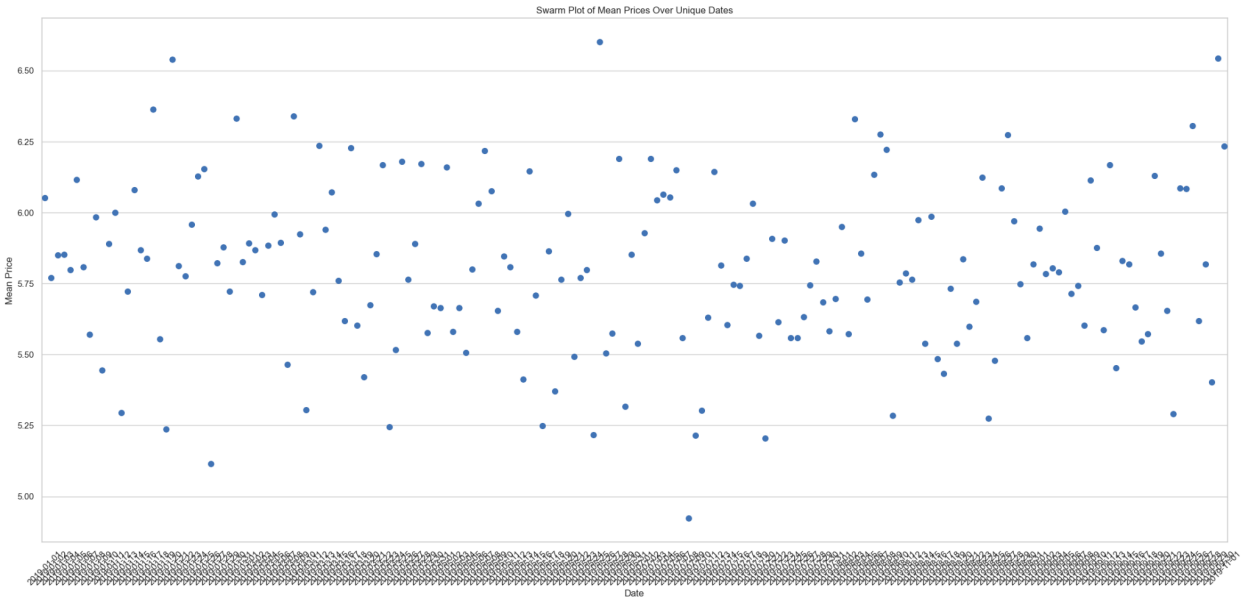
similar to correlation heatmap, it is observed that only 'Store_Lat' and 'Store_Long' exhibit a discernible correlation with each other. The correlation coefficient of -0.14 indicates a weak inverse relationship between the latitude and longitude of the supermarket branches. The negative sign suggests that as one variable increases, the other tends to decrease, albeit modestly.

18. Strip plot :



A strip plot is a type of data visualization used to display the distribution of one-dimensional data, similar to a scatter plot. However, instead of plotting individual data points, a strip plot shows individual data values as small markers along a single axis (often vertical). Overlapping data points are stacked vertically, creating "strips" that provide an impression of the underlying data distribution. Here is the distribution of Prices over the dates(time) through a strip plot.

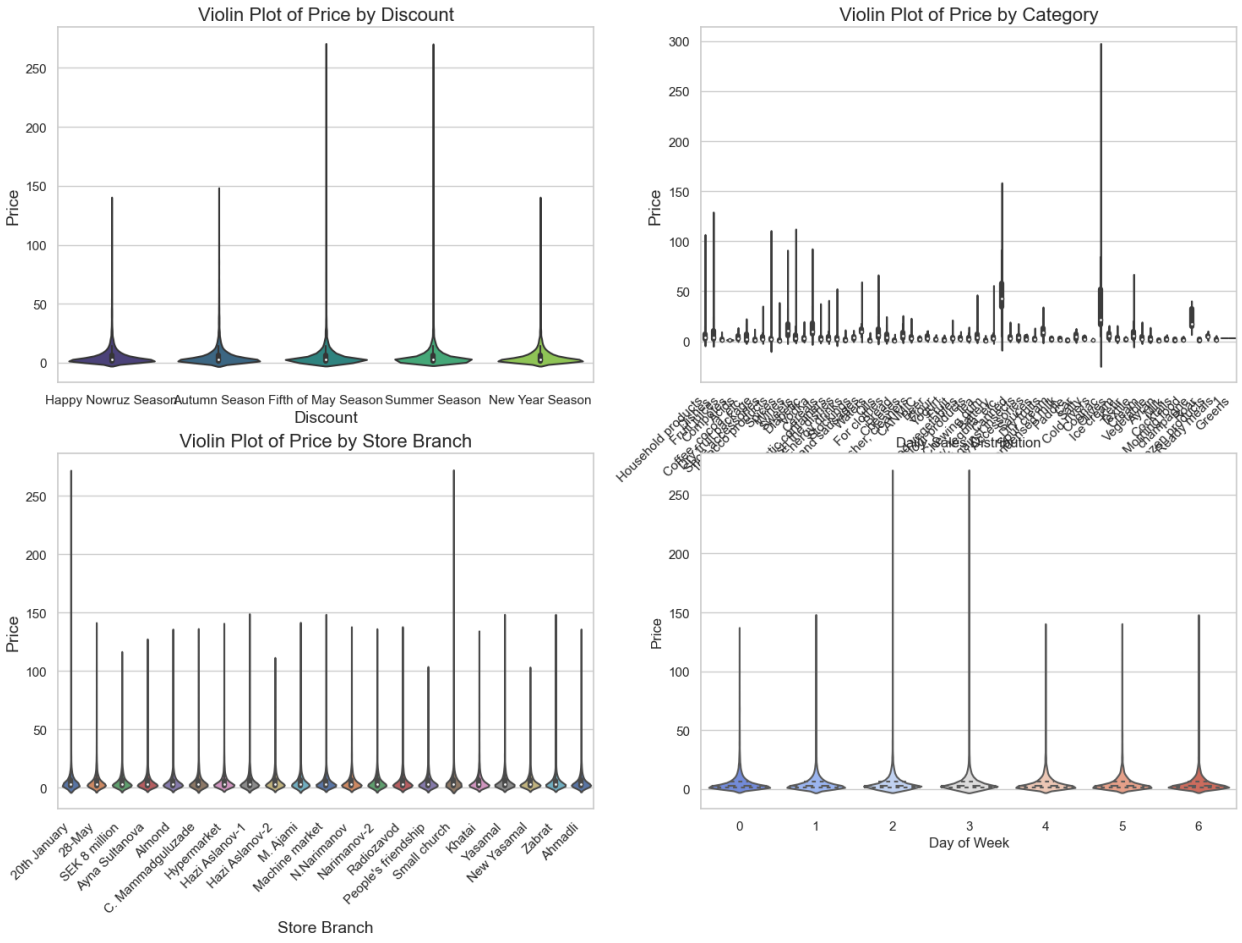
19. Swarm plot :



A swarm plot is another type of data visualization similar to a strip plot, used to display the distribution of a single variable. However, it differs from a strip plot in a few key ways: Jitter: Instead of stacking overlapping data points vertically, a swarm plot applies a small random horizontal offset to each data point. This "jitter" prevents overplotting and allows for better visualization of the individual data points within the distribution, especially when dealing with large datasets. Order: While strip plots typically display data points in ascending or descending order, swarm plots can display them in any order. This can be helpful for emphasizing the spread of the data and minimizing bias in how the distribution is perceived. Density Estimation: Some swarm plot implementations include a density estimation layer, which visually represents the underlying distribution of the data. This is similar to the KDE plot used in violin plots and provides additional information about the data's shape and potential skewness. Here is the distribution of Prices over the dates(time) through a swarm plot.

Subplots

Violin plot :



A violin plot is a data visualization that combines aspects of a box plot and a kernel density plot. It is used to depict the distribution and probability density of a continuous variable. The violin plot provides insights into the data's central tendency, spread, and underlying distribution. This above plot is a collection of all subplots of violin plots where : (i) Price by discount (ii) Price by category (iii) Price by Store Branch (iv) Date Sales Distribution.

Violin Plot of Price by Discount

- The median price of violins is highest in the "Happy Nowruz Season" and "Fifth of May Season" discount categories, and lowest in the "G" discount category.
- There is a wider range of prices in the higher discount categories, suggesting that there are more options available for budget-minded shoppers.
- There are some outliers in the higher discount categories, suggesting that it is possible to find high-quality violins at a discount.

Violin Plot of Price by Category

- The median price of violins is highest in the "Hypermarket" category, and lowest in the "Small church" and "Khatai" categories.
- There is a wider range of prices in the "Hypermarket" and "Household" categories, suggesting that these categories offer the most variety of violins.
- There are some outliers in the higher price categories, suggesting that it is possible to find high-quality violins in all categories.

Violin Plot of Price by Store Branch

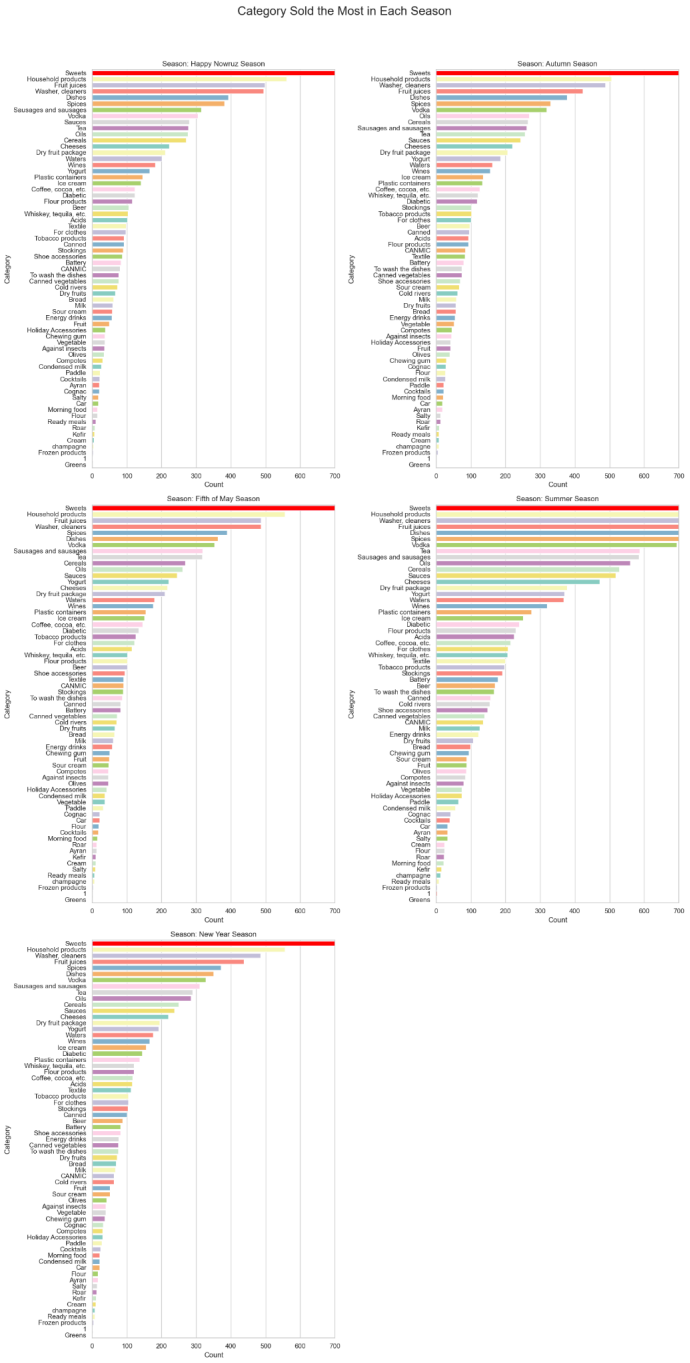
- The median price of violins is highest at the "20th January" and "28-May" store branches, and lowest at the "Ahmadli" and "Zabrat" store branches.
- There is a wider range of prices at the higher priced store branches, suggesting that these branches offer the most variety of violins.
- There are some outliers in the higher price categories, suggesting that it is possible to find high-quality violins at all store branches.

Overall, the data suggests that there is a wide range of violin prices available, and that it is possible to find a high-quality violin at a discount. The best way to find the best deal is to shop around and compare prices from different retailers.

Additional observations:

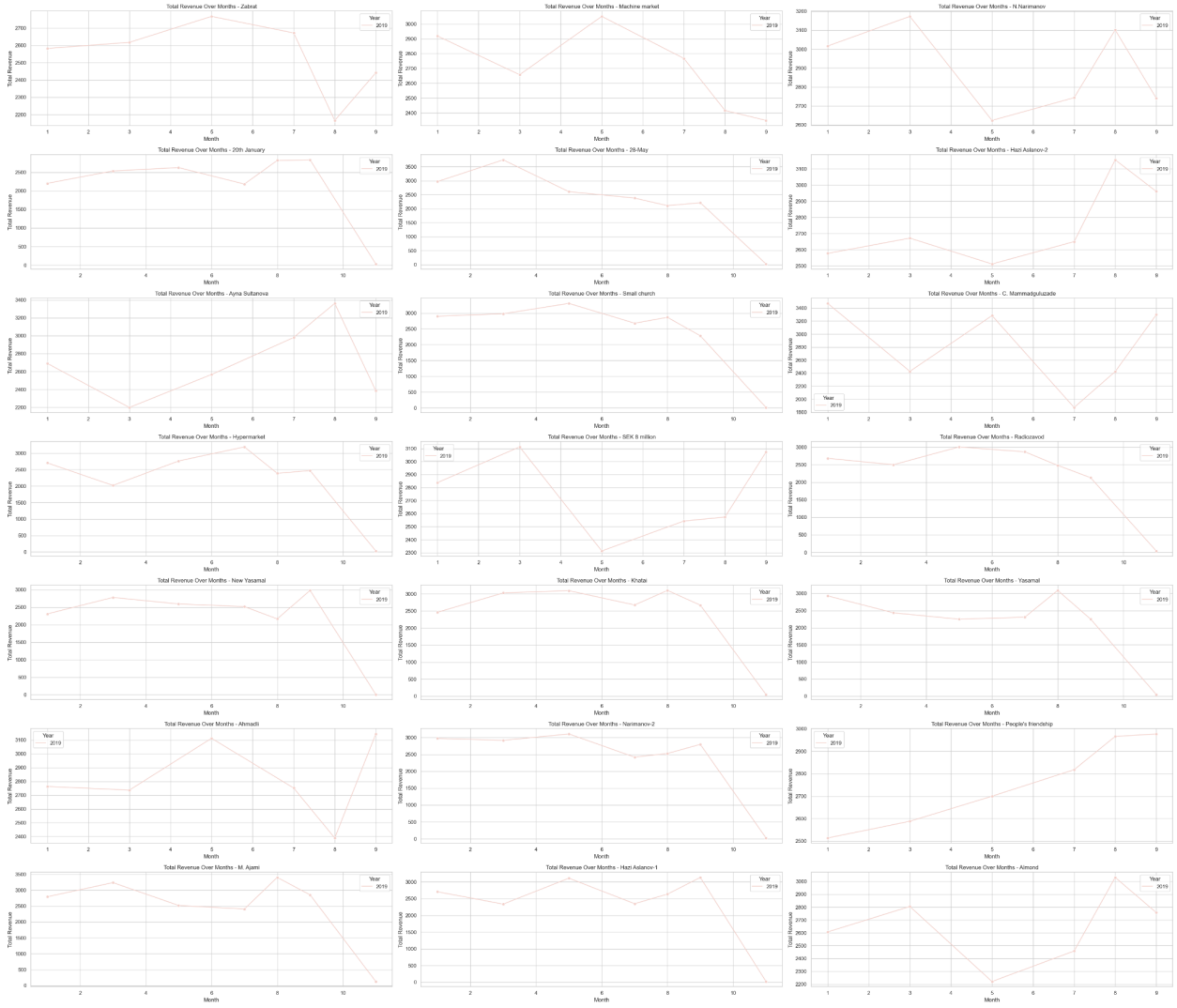
- The data appears to be from Azerbaijan, as the prices are listed in SEK (Swedish kronor).
- The data is from a variety of sources, including hypermarkets, household stores, machine markets, and music stores.

- The data is collected over a period of time, from January to May 2019.



This is a plot of various subplots, it represents the count of each Category in all different Discount seasons, from the above collection of subplots “Sweet” Category products are sold the most in all

Discount Season. All plots are sorted in descending order. The maximum count Category is denoted by the red color.



The above subplot gives the total revenue for each Store Branch separately in a specific plot. Khatai Store Branch has the highest revenue of them all.

Tables

Month	Store with Max Total Price	Max Total Price
January	C. Mammadguluzade	$3472.73
March	28–May	$3748.74
May	Small church	$3306.08
July	Hypermarket	$3194.07
August	M. Ajami	$3396.67
September	C. Mammadguluzade	$3298.33
November	M. Ajami	$128.23

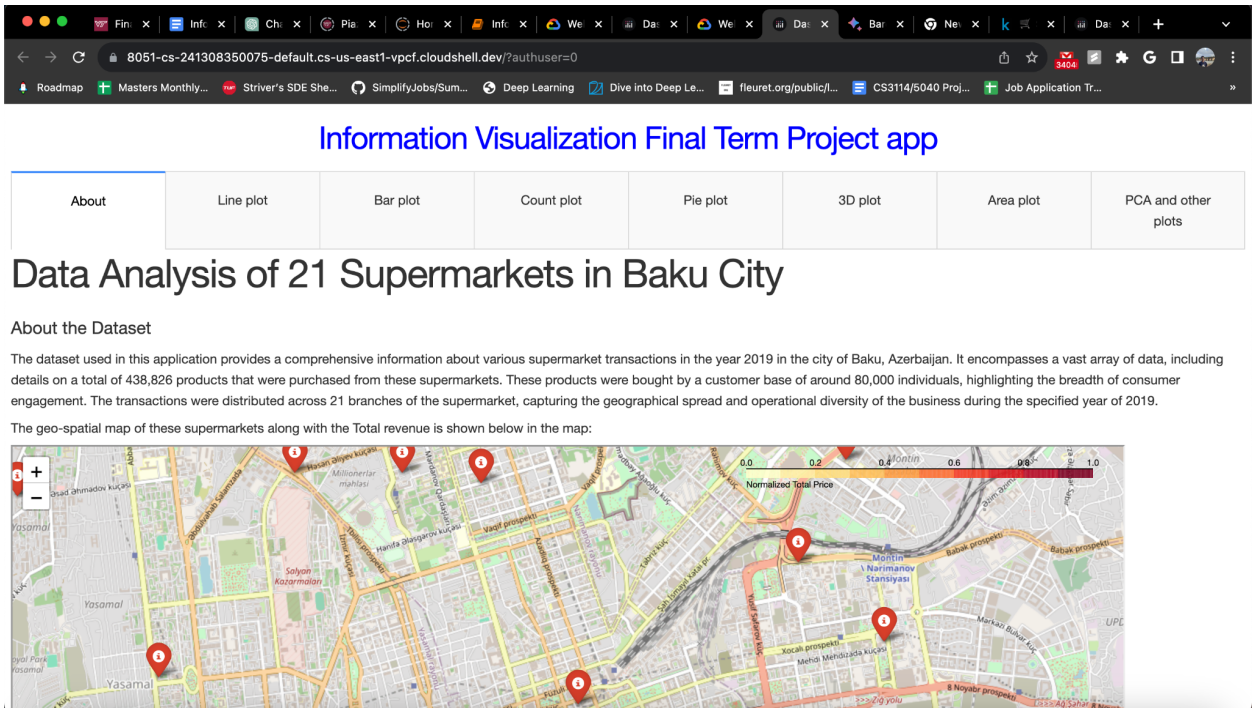
I created a PrettyTable for all the Store Branches with the maximum Total revenue for each month in 2019. You can observe which Store Branch what Total Price in dollars in each month. From the table you can see that the Store “28-May” got the maximum Total Revenue with respect to other branches in the entire year 2019 which is $3748.74.

Dashboard

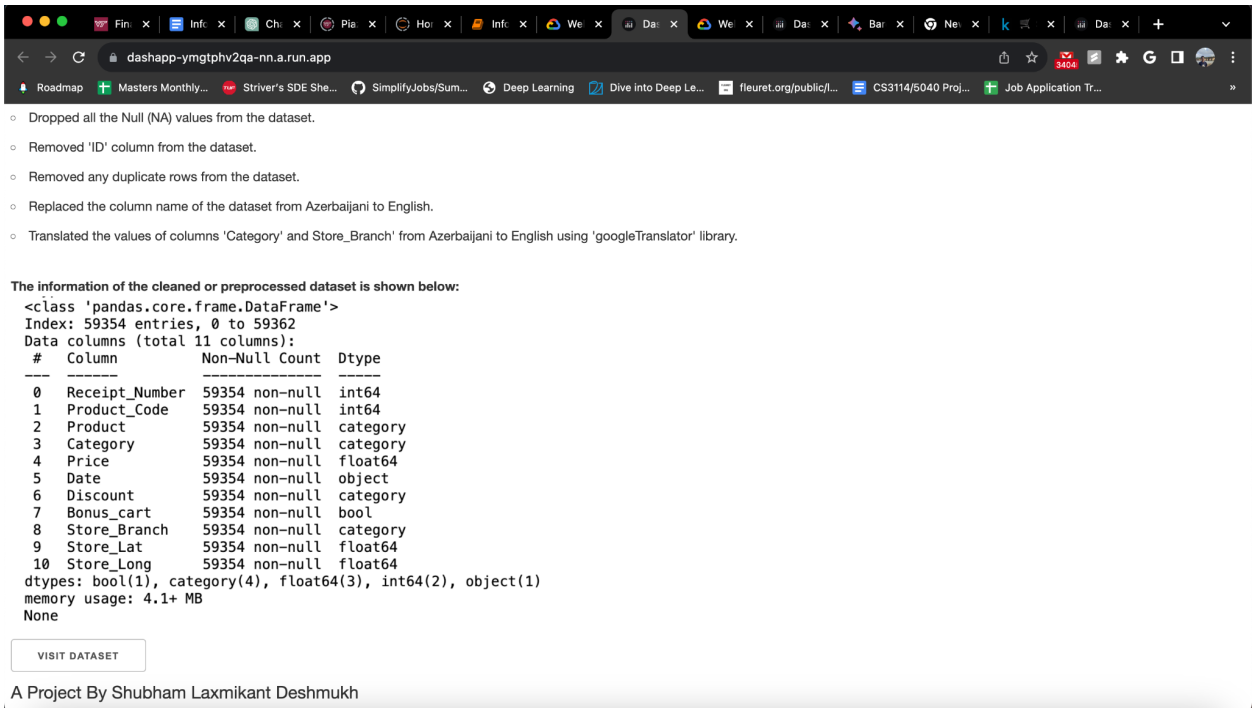
I have created a dashboard using plotly and dash libraries and deployed the code in Google Cloud Platform. The link of the dashboard that was deployed in on the internet is given below:

https://dashapp-ymgtphv2qa-nn.a.run.app/

About Tab:

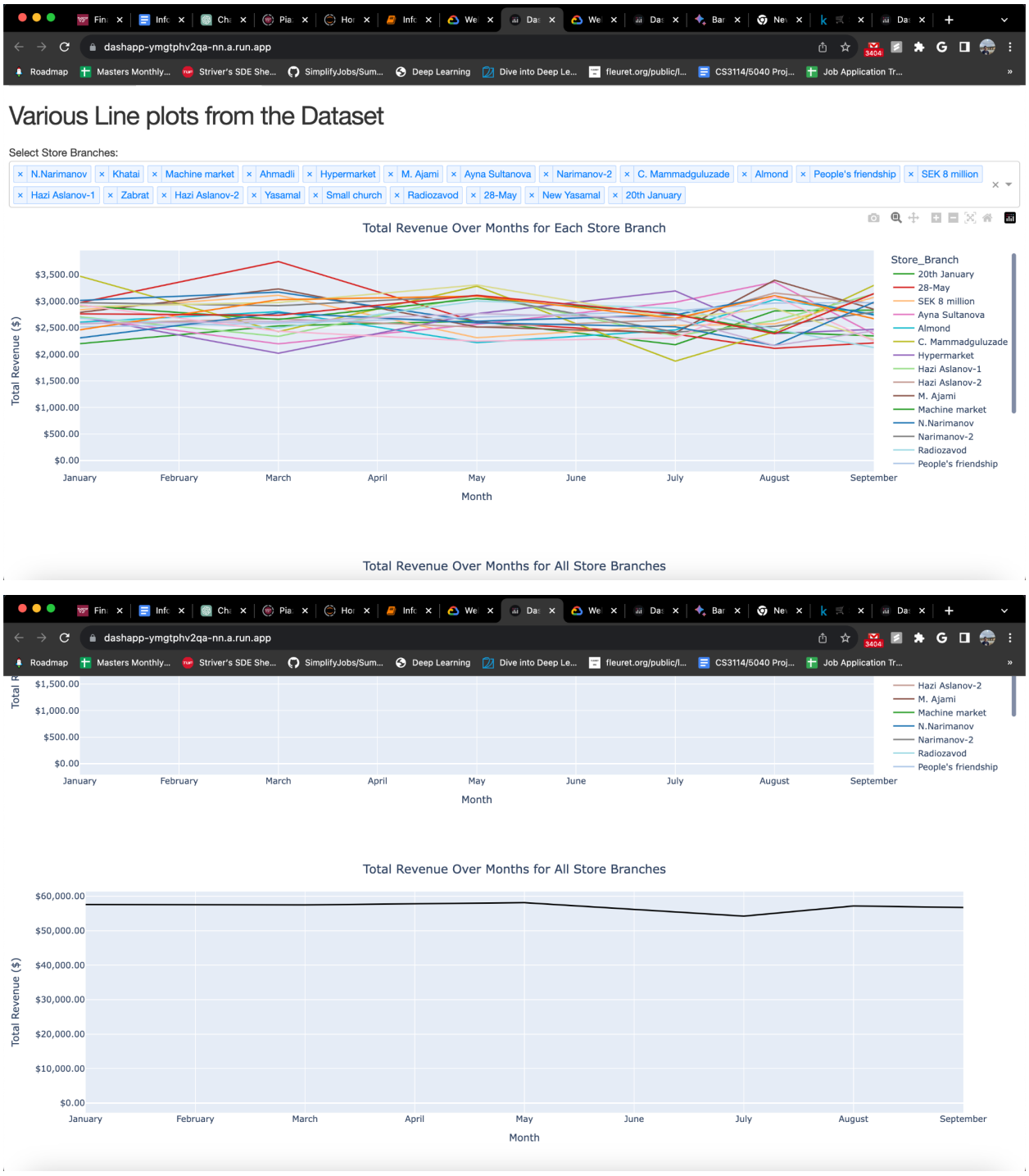


Virginia Tech MEng Computer Science and Applications



Here I have written about the project, dataset description and preprocessing steps of the Dataset. I have uploaded a geo spatial html file to get an geographical overview of these supermarkets. I have also added a button that will take the user to the kaggle website of the Supermarket Dataset.

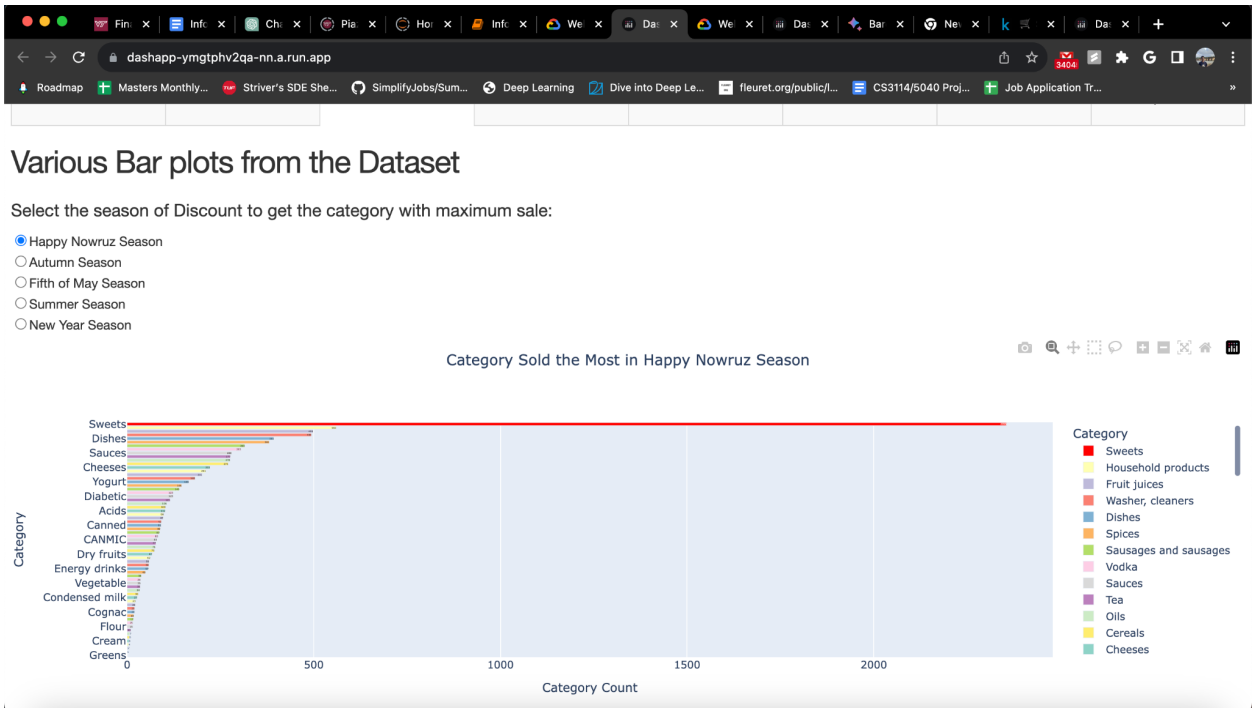
Line plot Tab:

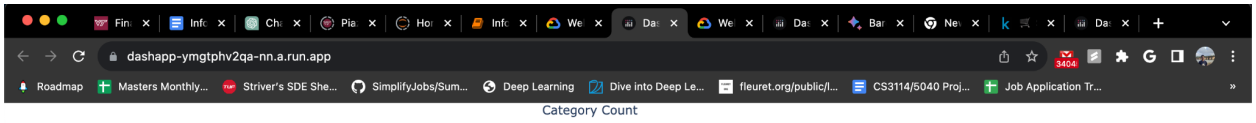


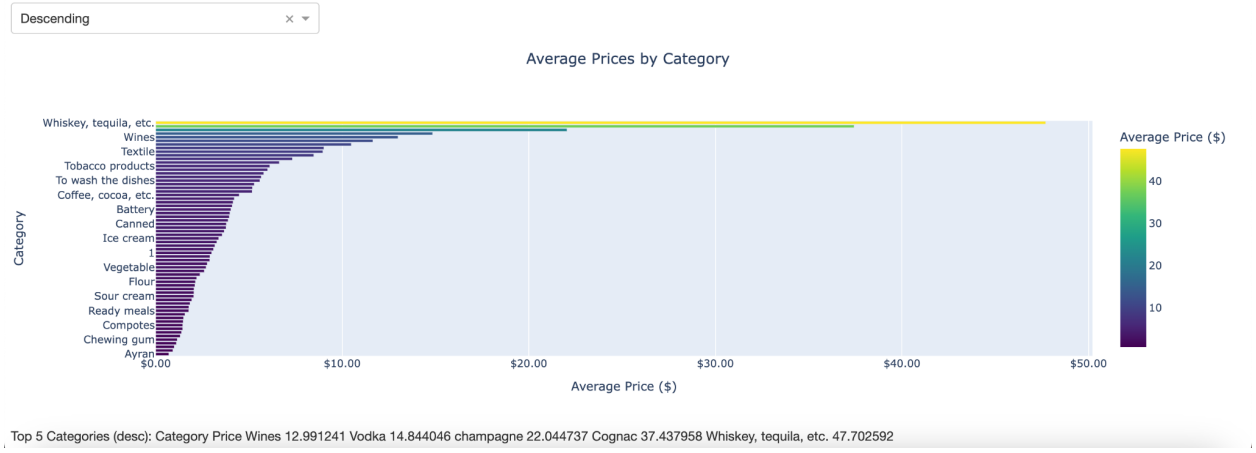
Here I have plotted two line plots of Total revenue of each Store Branch over time and Total Revenue of all the Store Branches in one plot. I have created a dropdown to select multiple Store Branches as per the user

needs.

Bar plot Tab:

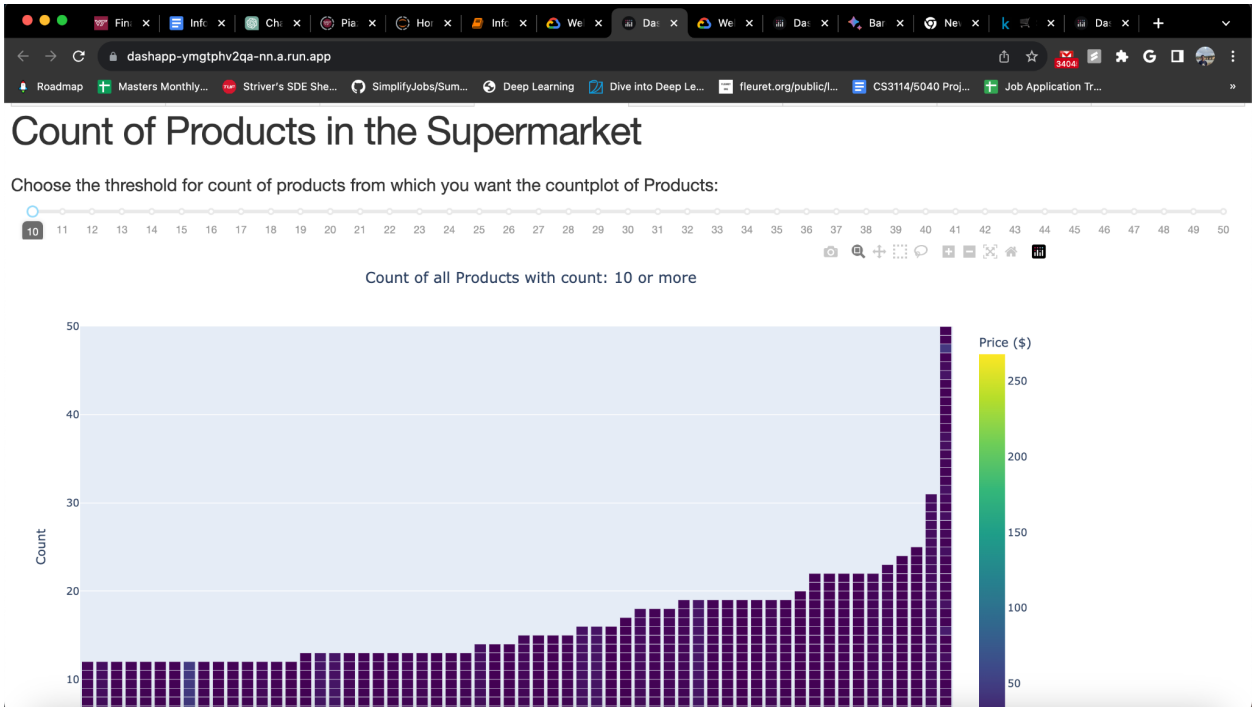


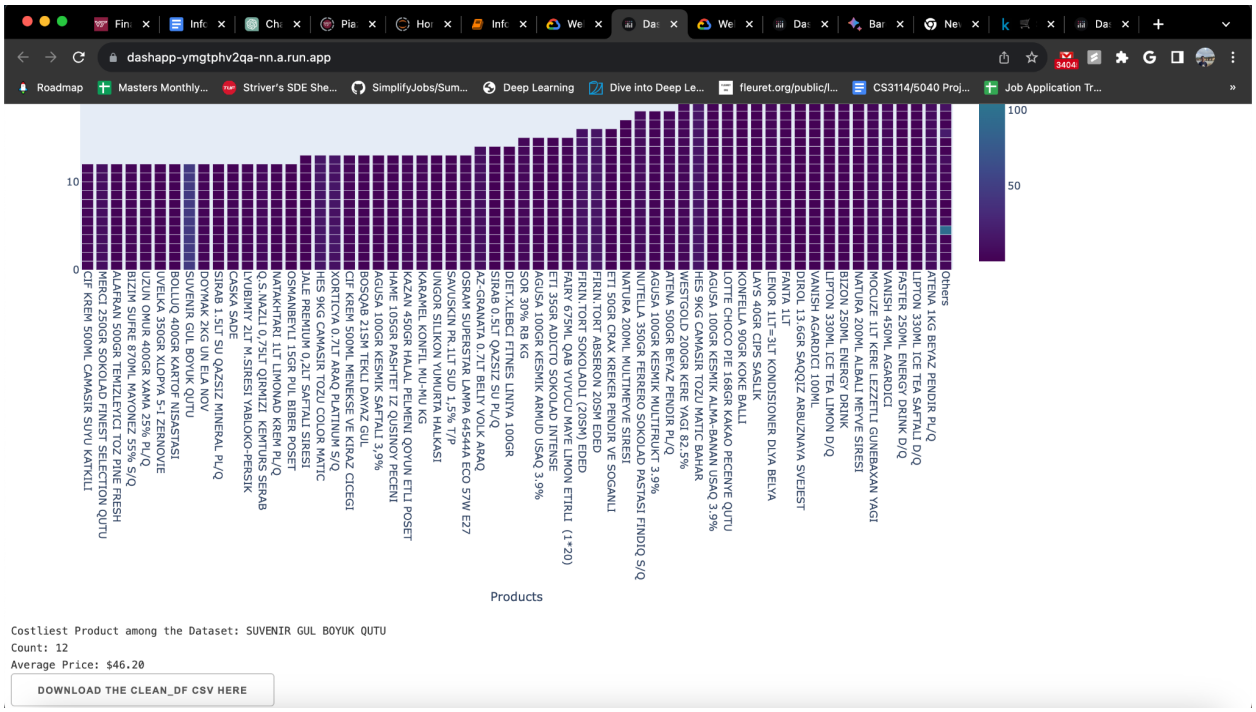




I have plotted two bar plots where I get the Categories most sold in a particular discount season. The user can choose from the Radio Buttons the various discount seasons and make their observations.
The second plot is Average Prices in a sorted order for various Categories. The user can select using the dropdown option weather to see the graph in ascending or descending order, the top 5 Categories according to the sorted order are printed down with its prices.

Count plot Tab:

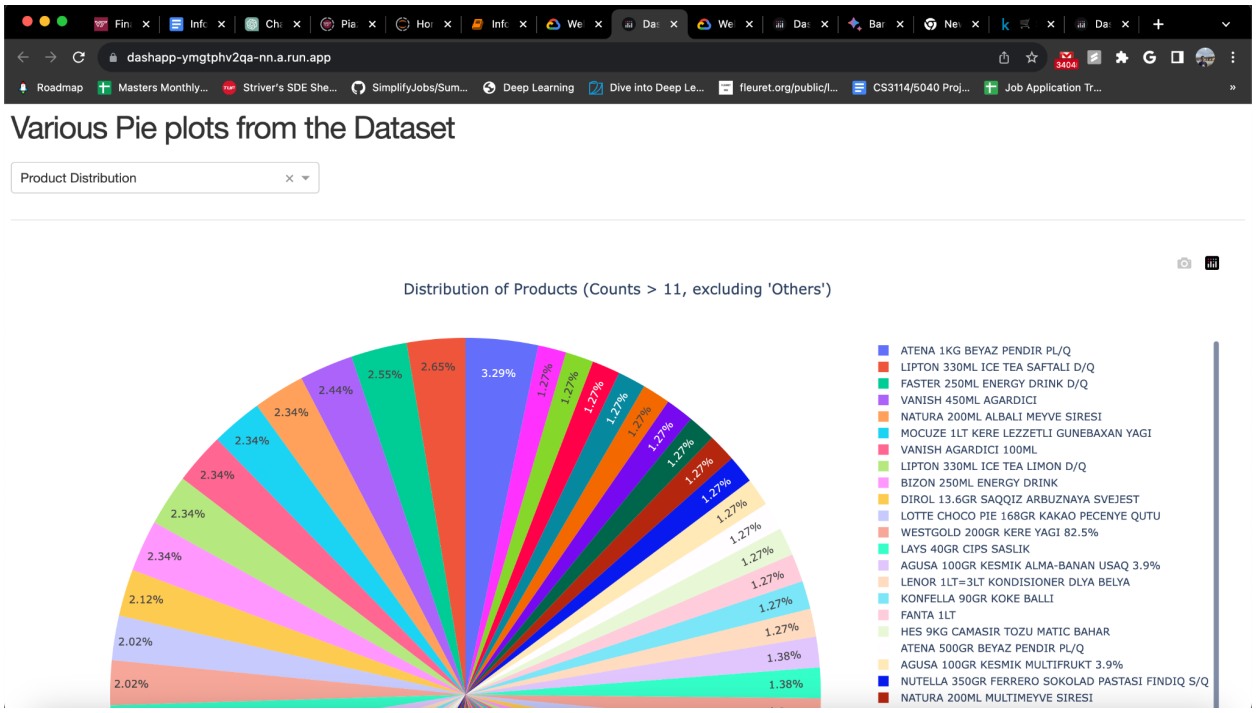


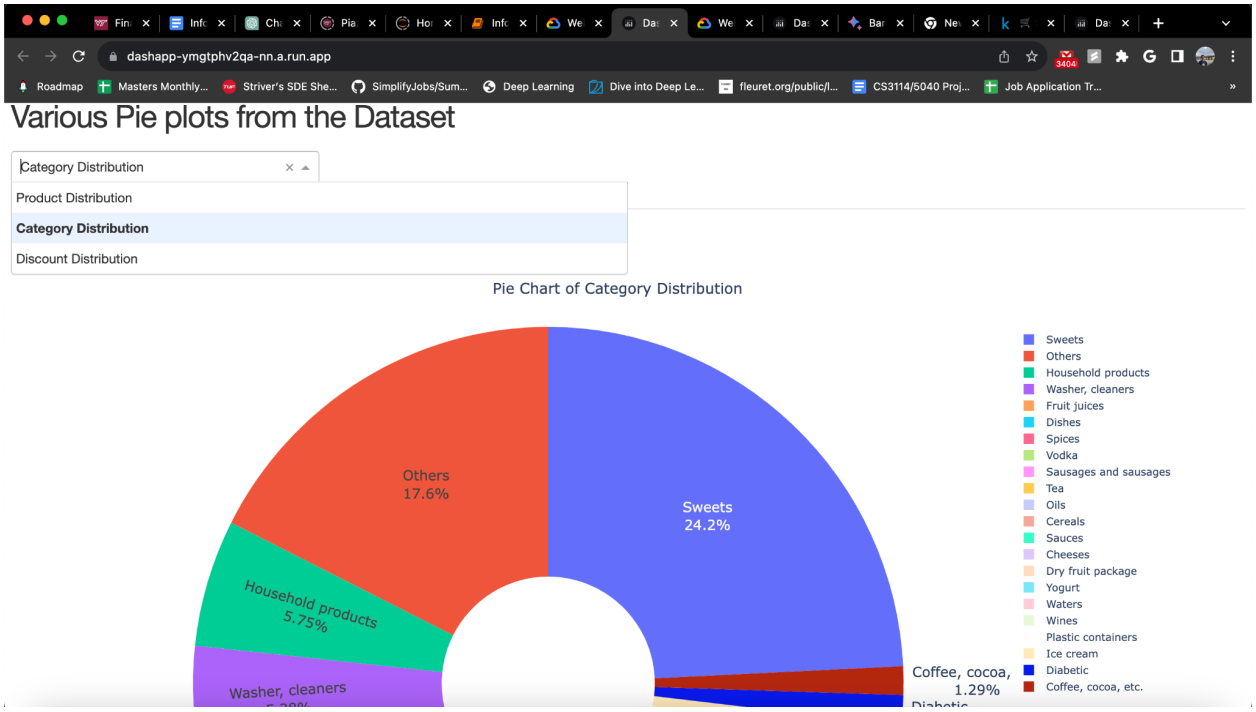


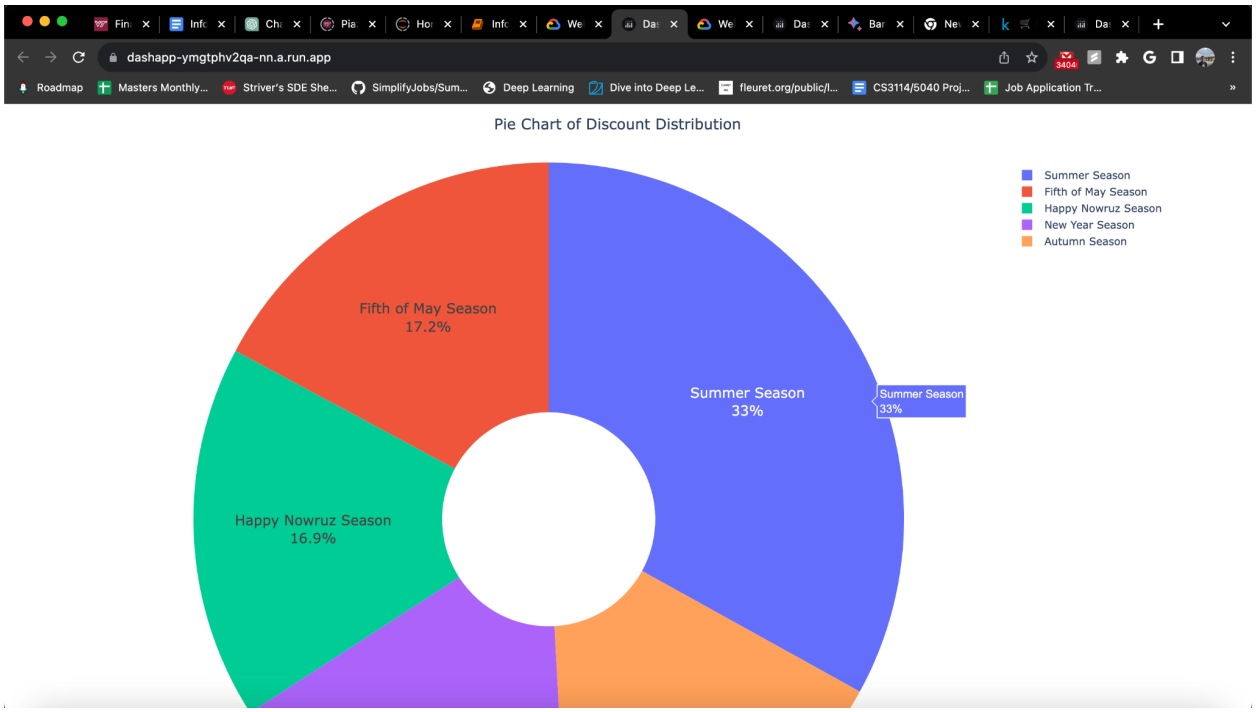
This is an interactive count plot where you can change the threshold of count of products for which you use the countplot for. For example: if you want the count plot of products whose count is more than 12, only those names will be visible on the x axis who has a count more than 12. I have used the Slider in here for the user to choose the threshold. You can even download the dataset

Pie plot Tab:

Virginia Tech MEng Computer Science and Applications

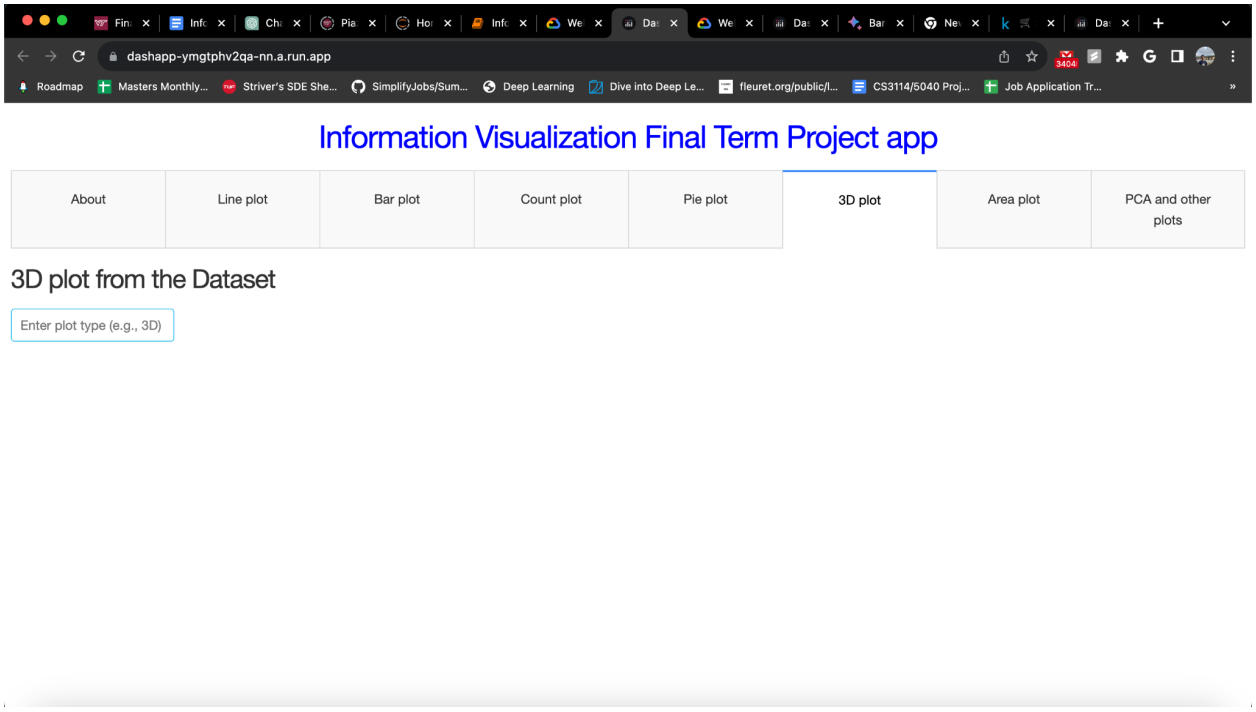


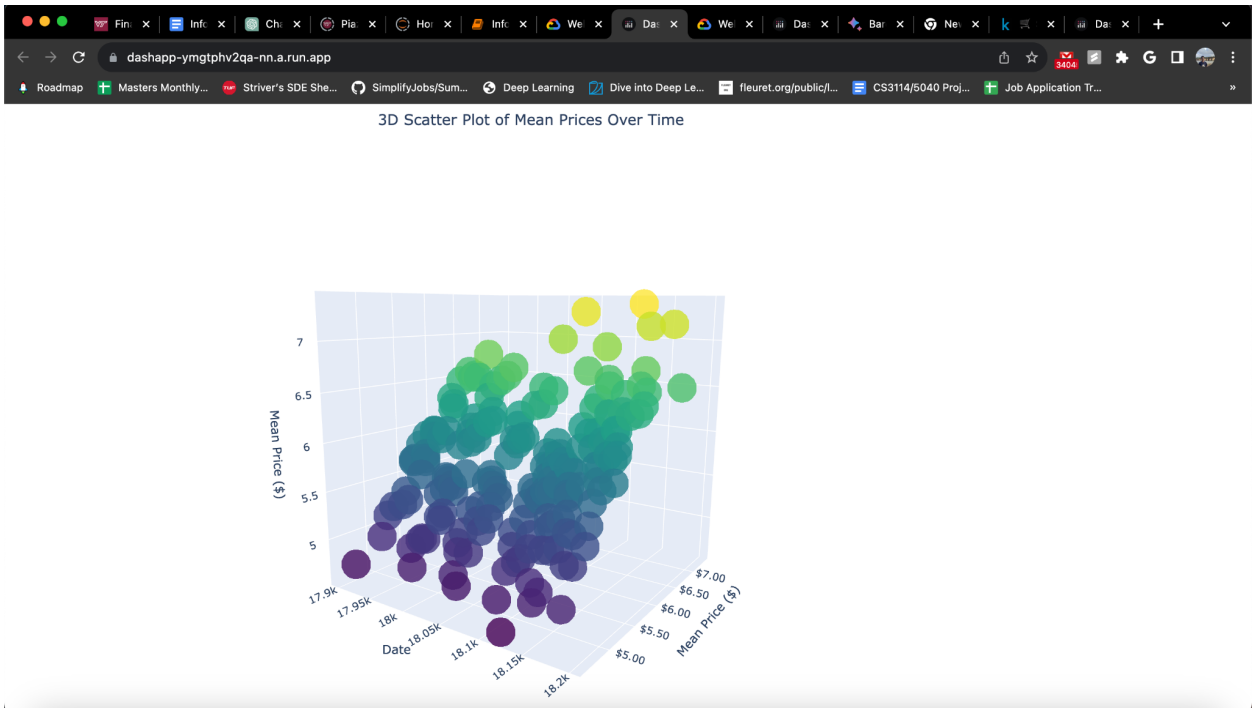




I have plotted various distribution pie plots of Category, Products and Discount seasons. There is a dropdown option to choose between these three of them.

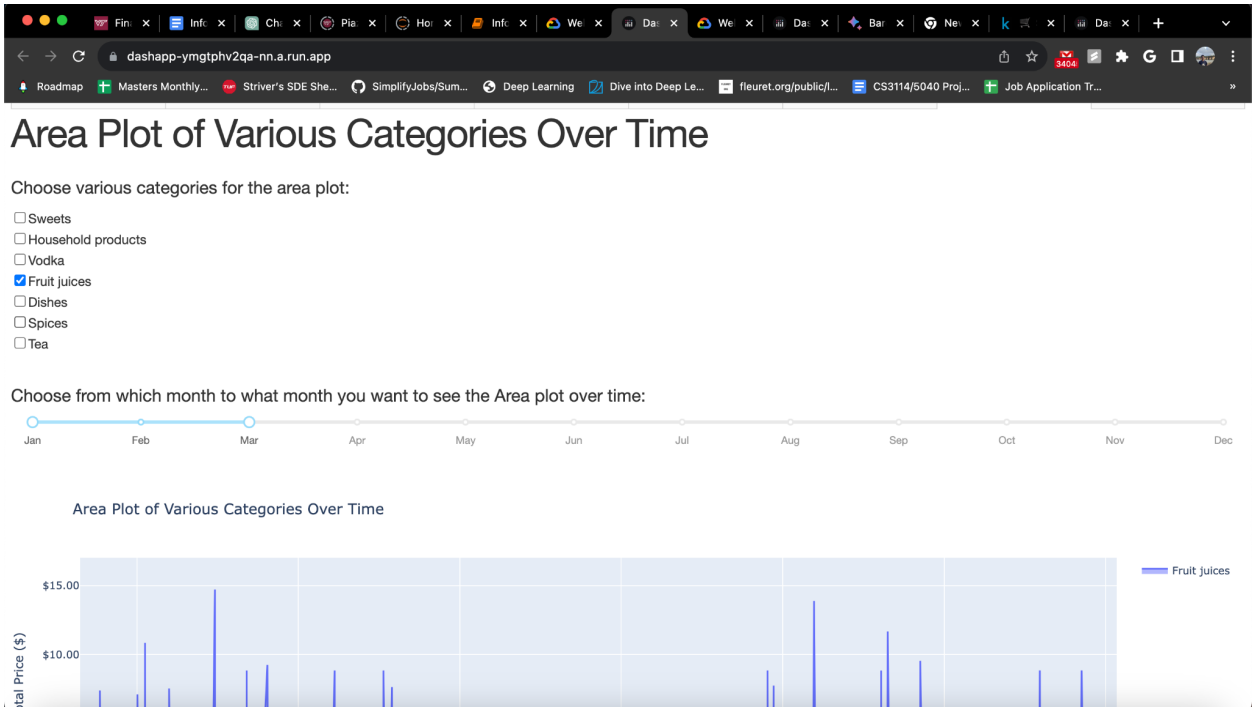
3D plot:

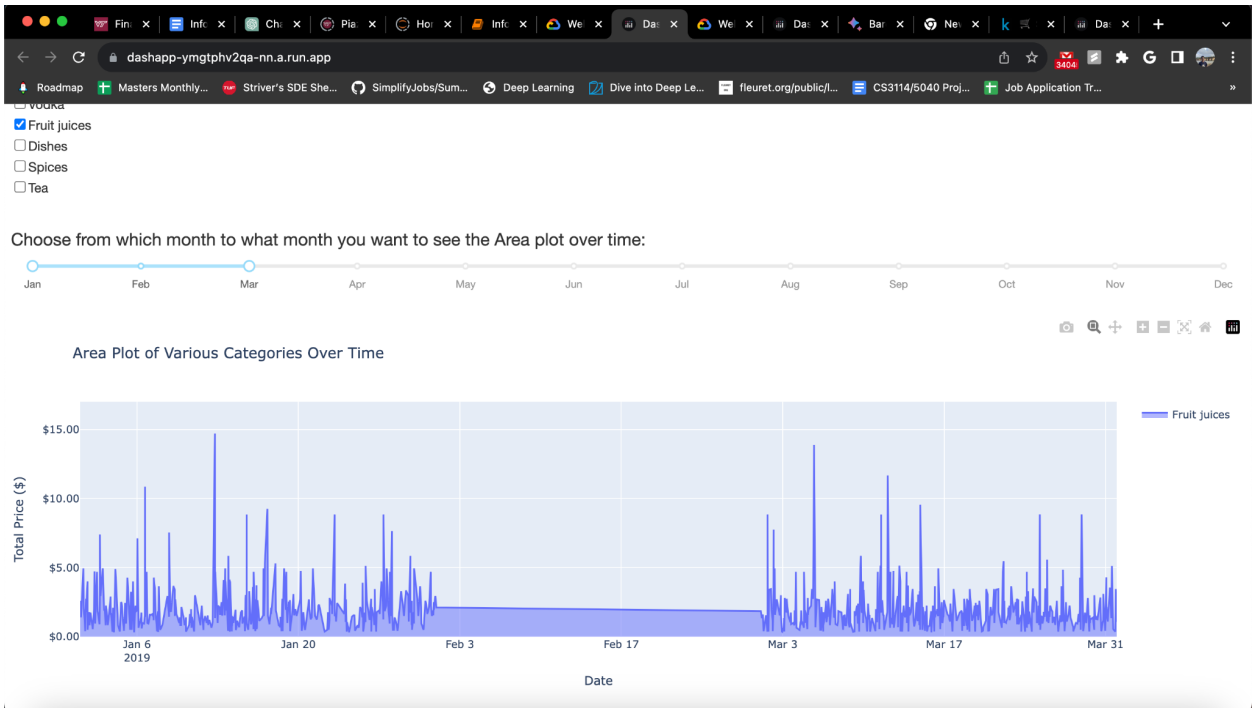




Here I am plotting a 3D plot only when the user types 3D in the input box. It's a 3D scatterplot of mean prices across the time.

Area plot tab:

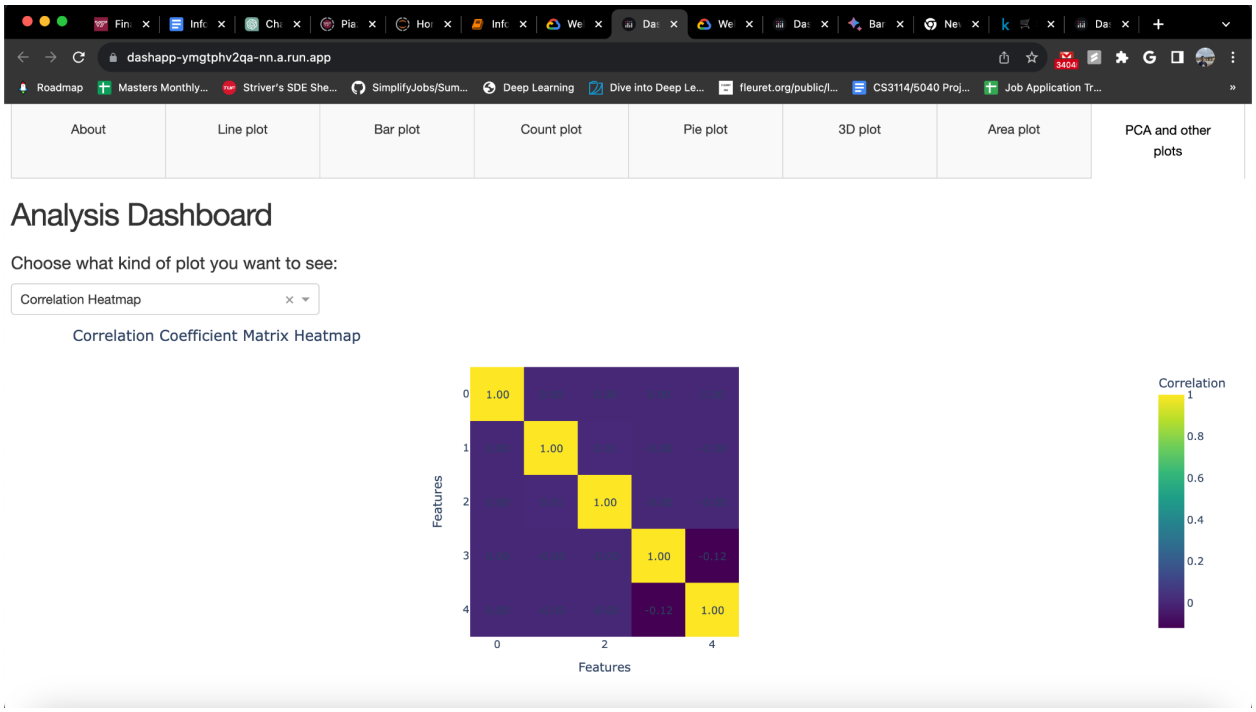


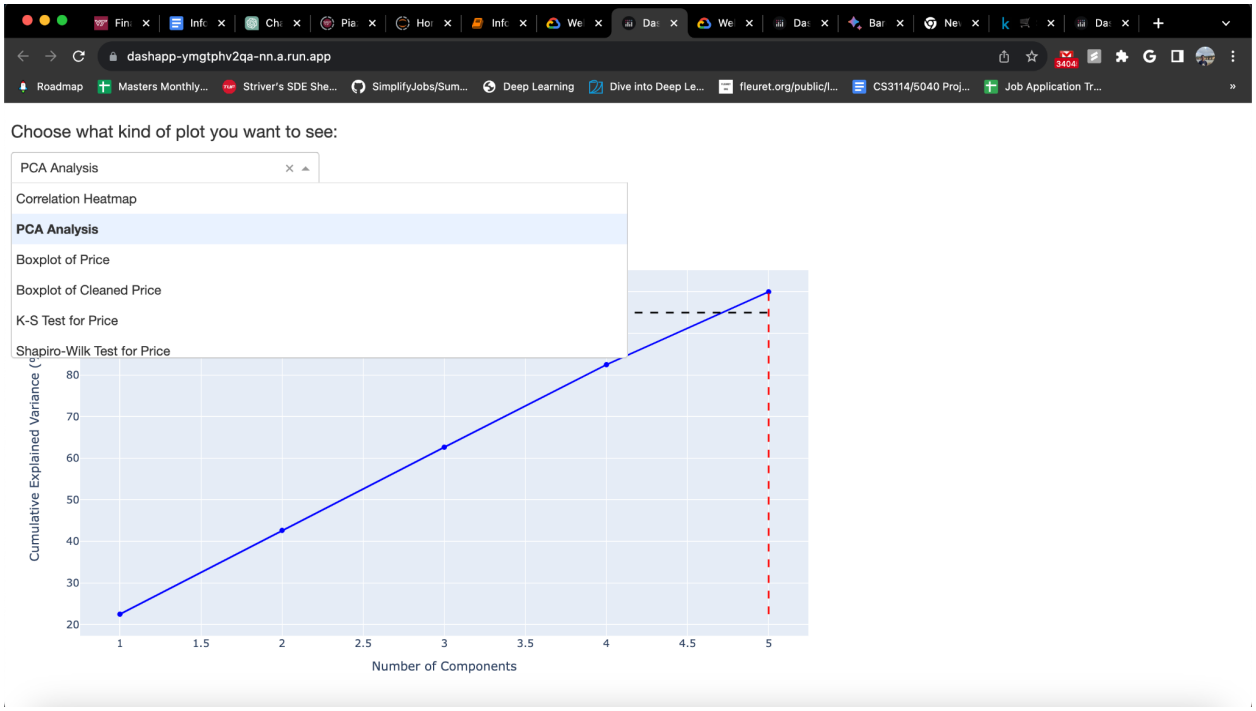


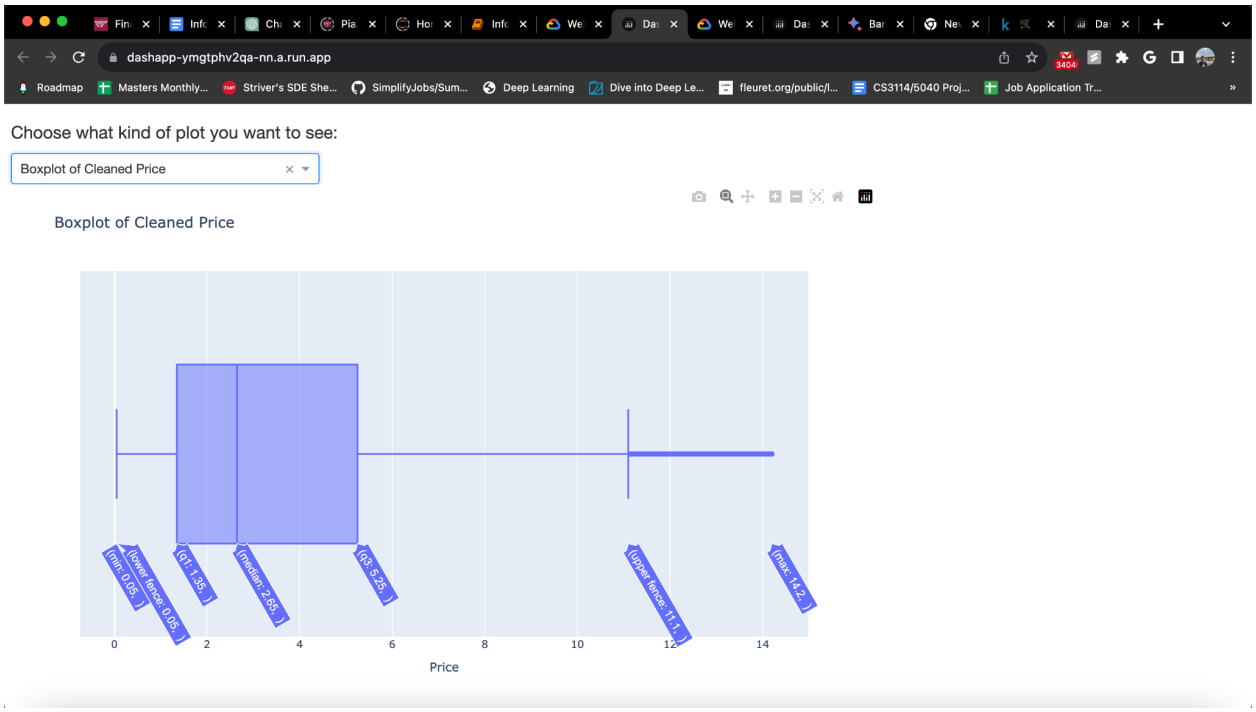
Here I am plotting the area plot of Categories over the time. The user can choose the Categories using the checklist and also choose the month range using the RangeSlider.

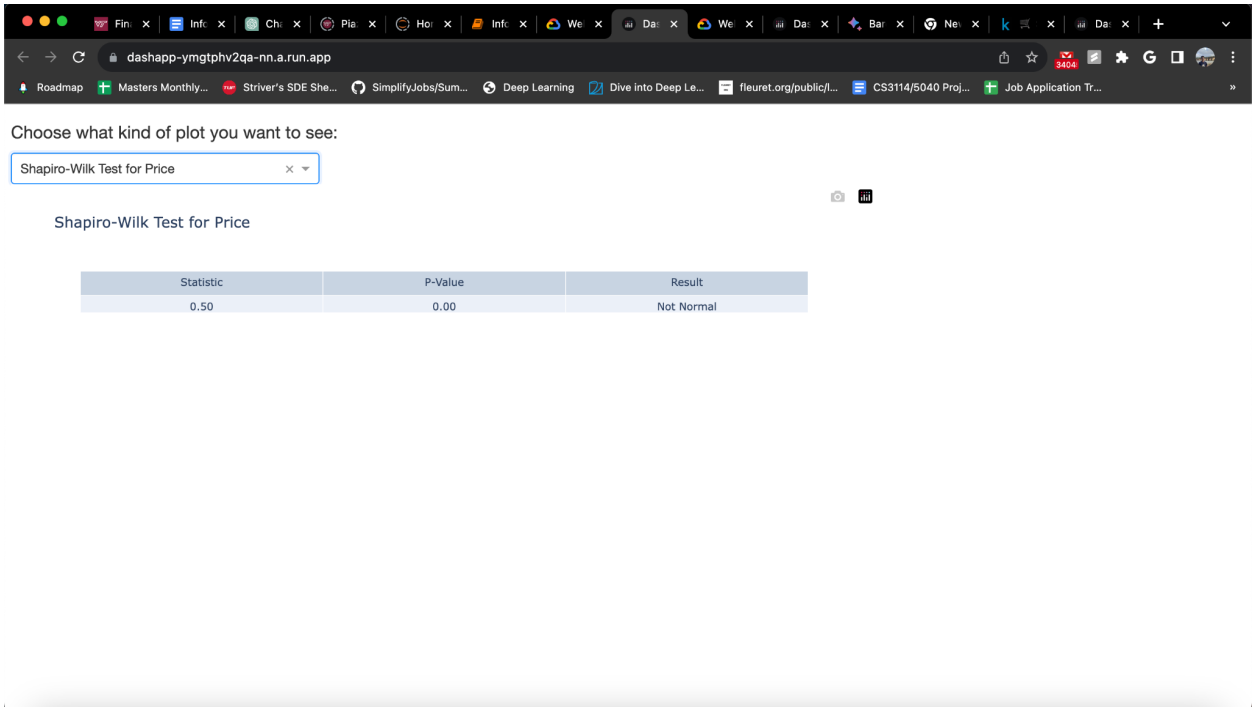
PCA and other graphs tab:

Virginia Tech MEng Computer Science and Applications









In the last tab I have plotted correlation heatmap, PCA analysis graph, Boxplot with outliers, Boxplot without outliers, Normality test tables of K S test, Shapiro -Wilk Test, D Agostino's K^2 Test. The user can choose from the dropdown to select any particular graph or table he / she wants.

Conclusion

Key observations and valuable insights derived from the "Data Analysis of 21 Supermarkets in Baku City" include:

1. **Extensive Product Diversity:**
   The analysis reveals a remarkable array of unique products across the 21 supermarkets in Baku City. This diversity underscores the rich variety offered to consumers, potentially influencing purchasing patterns and market trends.

2. **Steady Demand for "Sweet" Category:**
   Regardless of seasonal fluctuations or the time of day, the data consistently highlights a robust and unwavering demand for products falling within the "Sweet" category. This enduring consumer preference suggests a stable market for sweet goods, presenting strategic opportunities for suppliers and retailers.

3. **Data Gaps in Specific Months:**
   A notable observation is the presence of data gaps for particular months within the dataset. Identifying and addressing these gaps is crucial for a comprehensive understanding of market dynamics during these periods, ensuring a more accurate analysis and informed decision-making.

4. **Significantly Lower Prices:**
   In comparison to standard stores, the prices of the analyzed products are remarkably lower. This pricing trend could have implications for market competitiveness, consumer behavior, and the overall economic landscape. Understanding the factors contributing to these lower prices is essential for market stakeholders.

5. **Irregularities and Outliers in Price Column:**
   The Price column in the dataset exhibits irregularities, marked by numerous outliers. Investigating the reasons behind these anomalies is imperative for a nuanced understanding of pricing dynamics. Identifying and addressing these outliers will contribute to more accurate statistical analyses and reliable insights for strategic planning.

The importance of my Dash app is given by:

My Python dashboard deployed on the Google Cloud Platform plays a crucial role in empowering users to glean valuable insights from the selected dataset. Here's how the dashboard aids users in extracting information:

1. **Visual Representation of Data:**
   The dashboard employs visualizations such as plots and charts to represent the dataset visually. This facilitates a quick and intuitive understanding of trends, patterns, and relationships within the data. Users can easily interpret complex information through graphical representations.

2. **Interactive Exploration:**
   An interactive interface allows users to explore the dataset dynamically. Features like dropdowns, sliders, or clickable elements enable users to customize the view according to their specific queries. This interactivity enhances the user experience and allows for on-the-fly adjustments to focus on relevant aspects of the data.

3. **Multiple Tabs for Comprehensive Analysis:**
   The inclusion of multiple tabs in your dashboard indicates a structured approach to analysis. Each tab likely addresses a specific aspect of the dataset, offering users a comprehensive exploration of various dimensions. This organized layout ensures that users can efficiently navigate through different analyses without feeling overwhelmed.

4. **Real-time Updates:**
   If your dashboard is designed to fetch live or regularly updated data, users benefit from real-time insights. This feature is particularly valuable for tracking dynamic trends, responding to changes promptly, and making data-driven decisions based on the most current information.

5. **User-Friendly Interface:**
   A well-designed user interface enhances accessibility. Users, regardless of their technical expertise, can easily navigate the dashboard, access relevant information, and interact with the visualizations effortlessly. A user-friendly interface contributes to a positive user experience.

6. **Clear Annotations and Contextual Information:**
   To assist users in understanding the context of the data, your dashboard may include clear annotations, captions, or tooltips. These elements provide additional information about the plotted data points, ensuring that users can interpret the visualizations accurately.

7. **Export and Share Functionality:**
   Users can benefit from features that allow them to export visualizations or share specific insights with others. Whether it's exporting a chart for a presentation or sharing a link to a particular analysis, these functionalities enhance collaboration and communication.

8. **Scalability and Performance:**
   Deploying your dashboard on the Google Cloud Platform ensures scalability and reliable performance. Users can access the dashboard seamlessly, even when dealing with large datasets, leading to a smoother and more efficient analytical experience.

The link of my Dashboard app is :
https://dashapp-ymgtphv2qa-nn.a.run.app/

Appendix

Static plots py file:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# ===================================#
# Name: Shubham Laxmikant Deshmukh   #
# ===================================#


# Importing Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
import seaborn as sns
import pandas as pd
import seaborn as sns
from datetime import datetime
import warnings

warnings.filterwarnings("ignore")

# %%

# Reading the dataset from the csv file
df = pd.read_csv("/Users/shubhamlaxmikantdeshmukh/Downloads/esas_mehsullar
2.csv")

# taking limited records
df = df.head(120000)

# Renaming column Names
df = df.rename(
    columns={'Unnamed: 0': 'ID', 'satish_kodu': 'Receipt_Number',
'mehsul_kodu': 'Product_Code', 'mehsul_ad': 'Product',
             'mehsul_kateqoriya': 'Category', 'mehsul_qiymet': 'Price',
'satish_tarixi': 'Date',
             'endirim_kompaniya': 'Discount', 'bonus_kart': 'Bonus_cart',
'magaza_ad': 'Store_Branch',
             'magaza_lat': 'Store_Lat', 'magaza_long': 'Store_Long'})
```

```
# Printing the first 5 rows from the dataframe
print(df.head())

# Printing the info of df
print(df.info())

# Checking if there are any na values in df
print(df.isna().sum())

# droping the na values in df and checking the if they are gone
clean_df = df.dropna()

# dropping unescessay columns
clean_df = clean_df.drop('ID', axis=1)

# reseting index
clean_df = clean_df.reset_index(drop=True)

# checking for duplicate rows and dropping them
duplicated_rows = clean_df.duplicated()
clean_df = clean_df.drop_duplicates()

# Coverting type of some coulmns
clean_df = clean_df.copy()  # Create a copy of the DataFrame
clean_df['Product'] = clean_df['Product'].astype('category')
clean_df.loc[:, 'Date'] = pd.to_datetime(clean_df['Date'], dayfirst=True)
clean_df['Category'] = clean_df['Category'].astype('category')
clean_df['Discount'] = clean_df['Discount'].astype('category')
clean_df['Store_Branch'] = clean_df['Store_Branch'].astype('category')

# printing the info of clean_df and its first 10 rows
print(clean_df.info())
print(clean_df.head(5))

# In[2]:


# Replace multiple values in the 'Discount' column
replace_dict = {'Sərin Yay günləri': 'Summer Season', 'S?rf?li Yaz': 'Fifth of
May Season',
                'Bərəkətli Novruz': 'Happy Nowruz Season', 'Yeni il
fürsətləri': 'New Year Season',
                'Payız endirimləri': 'Autumn Season'}
clean_df['Discount'] = clean_df['Discount'].replace(replace_dict)

# Translate values in the 'Category' and 'Store Branch' column
from googletrans import Translator
```

```
translator = Translator()
clean_df['Category'] = clean_df['Category'].apply(lambda x:
translator.translate(x).text)
clean_df['Store_Branch'] = clean_df['Store_Branch'].apply(lambda x:
translator.translate(x).text)

# In[3]:


print(clean_df.head(5))
print(clean_df.isna().sum())

# In[4]:


# Unique values and counts for 'Product'
Receipt_Number_counts = clean_df['Receipt_Number'].value_counts()

# Unique values and counts for 'Product'
product_counts = clean_df['Product'].value_counts()

# Unique values and counts for 'Discount'
discount_counts = clean_df['Discount'].value_counts()

# Unique values and counts for 'Store_Branch'
store_branch_counts = clean_df['Store_Branch'].value_counts()

# Unique values and counts for 'Category'
category_counts = clean_df['Category'].value_counts()

# Displaying the results
print("\nUnique values and counts for 'Receipt_Number':")
print(Receipt_Number_counts)

print("\nUnique values and counts for 'Product':")
print(product_counts)

print("\nUnique values and counts for 'Discount':")
print(discount_counts)

print("\nUnique values and counts for 'Store_Branch':")
print(store_branch_counts, len(store_branch_counts))

print("\nUnique values and counts for 'Category':")
print(category_counts)

# # Line plot
```

```
# In[5]:


# Reading the dataset from the csv file
sub_df = pd.read_csv("/Users/shubhamlaxmikantdeshmukh/Downloads/esas_mehsullar
2.csv")

# #taking limited records
# df=df.head(120000)

# Renaming column Names
sub_df = sub_df.rename(
    columns={'Unnamed: 0': 'ID', 'satish_kodu': 'Receipt_Number',
'mehsul_kodu': 'Product_Code', 'mehsul_ad': 'Product',
             'mehsul_kategoriya': 'Category', 'mehsul_qiymet': 'Price',
'satish_tarixi': 'Date',
             'endirim_kompaniya': 'Discount', 'bonus_kart': 'Bonus_cart',
'magaza_ad': 'Store_Branch',
             'magaza_lat': 'Store_Lat', 'magaza_long': 'Store_Long'})

# droping the na values in df and checking the if they are gone
sub_df = sub_df.dropna()

# dropping unescessay columns
sub_df = sub_df.drop('ID', axis=1)

# reseting index
sub_df = sub_df.reset_index(drop=True)

# checking for duplicate rows and dropping them
duplicated_rows = clean_df.duplicated()
clean_df = clean_df.drop_duplicates()

# Coverting type of some coulmns
sub_df = sub_df.copy()  # Create a copy of the DataFrame
sub_df['Product'] = sub_df['Product'].astype('category')
sub_df.loc[:, 'Date'] = pd.to_datetime(sub_df['Date'], dayfirst=True)
sub_df['Category'] = sub_df['Category'].astype('category')
sub_df['Discount'] = sub_df['Discount'].astype('category')
sub_df['Store_Branch'] = sub_df['Store_Branch'].astype('category')

new_df = sub_df[['Date', 'Price']].copy()
# Assuming 'Date' is the column name in your DataFrame
new_df['Date'] = pd.to_datetime(new_df['Date']).dt.date
unique_dates_count = new_df['Date'].nunique()
print("Number of unique dates:", unique_dates_count)
```

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

# Sort the DataFrame by 'Date'
new_df.sort_values(by='Date', inplace=True)

# Plotting with Seaborn
plt.figure(figsize=(24, 14))
sns.lineplot(x='Date', y='Price', data=new_df, marker='o', color='b')

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Prices Over Time')

# Show the plot
plt.show()

# # Bar plot

# In[36]:


import seaborn as sns
import matplotlib.pyplot as plt

# Create a DataFrame with counts of Bonus Cart usage by category
bonus_cart_by_category = clean_df.groupby(['Category',
'Bonus_cart']).size().unstack()

# Plotting
plt.figure(figsize=(24, 8))
bonus_cart_by_category.plot(kind='bar', stacked=True, colormap='coolwarm')
plt.xlabel('Category')
plt.ylabel('Count')
plt.ylim(0, 1200)
plt.title('Stacked Bar Plot of Bonus Cart Usage by Category')
plt.xticks(rotation=60)
plt.legend(title='Bonus Cart')
plt.gcf().set_size_inches(18, 12)
plt.tight_layout()
plt.show()

# Create a DataFrame with counts of discounts by category
discount_by_category = clean_df.groupby(['Category',
'Discount']).size().unstack()

# Plotting
```

```
plt.figure(figsize=(24, 8))
discount_by_category.plot(kind='bar', stacked=True, colormap='viridis')
plt.xlabel('Category')
plt.ylabel('Count')
plt.ylim(0, 14000)
plt.title('Stacked Bar Plot of Discount Distribution by Category')
plt.xticks(rotation=60)
plt.gcf().set_size_inches(18, 12)
plt.legend(title='Discount')
plt.show()

# Create a DataFrame with counts of Bonus Cart usage by store branch
bonus_cart_by_branch = clean_df.groupby(['Store_Branch',
'Bonus_cart']).size().unstack()

# Plotting
plt.figure(figsize=(24, 8))
bonus_cart_by_branch.plot(kind='bar', stacked=True)
plt.xlabel('Store Branch')
plt.ylabel('Count')
plt.title('Stacked Bar Plot of Bonus Cart Usage by Store Branch')
plt.xticks(rotation=90)
plt.legend(title='Bonus Cart')
plt.show()

# Stacked Bar Plot for Category by Store_Branch
category_by_branch = clean_df.groupby(['Category',
'Store_Branch']).size().unstack()
category_by_branch.plot(kind='bar', stacked=True, figsize=(20, 12),
colormap='viridis')
plt.title('Category Distribution by Store Branch', fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Store Branch', fontdict={'fontname': 'serif', 'color': 'darkred',
'size': 14})
plt.ylabel('Count', fontdict={'fontname': 'serif', 'color': 'darkred', 'size':
14})

plt.xticks(rotation=90)
plt.show()

# Stacked Bar Plot for Bonus_cart by Store_Branch
bonus_cart_by_branch = clean_df.groupby(['Bonus_cart',
'Store_Branch']).size().unstack()
bonus_cart_by_branch.plot(kind='bar', stacked=True, figsize=(12, 8),
colormap='Set2')
plt.title('Bonus Cart Usage Distribution by Store Branch',
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
```

```
plt.xlabel('Store Branch', fontdict={'fontname': 'serif', 'color': 'darkred',
'size': 14})
plt.ylabel('Count', fontdict={'fontname': 'serif', 'color': 'darkred', 'size':
14})
plt.xticks(rotation=45)
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt

# Set the style of seaborn
sns.set(style="whitegrid")

# Set up the matplotlib figure with a larger size
plt.figure(figsize=(16, 10))

# Create a grouped bar plot
sns.barplot(x='Discount', y='Price', hue='Category', data=clean_df, ci=None)

# Adjust legend size and position
plt.legend(title='Category', title_fontsize='14', loc='upper right',
fontsize='12')

# Add labels and title
plt.xlabel('Discount', fontsize=14)
plt.ylabel('Average Price', fontsize=14)
plt.title('Grouped Bar Plot of Average Price by Discount and Category',
fontsize=16)

# Show the plot
plt.show()

# # Count plot
#

# In[39]:


import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the count of each receipt number
receipt_number_counts = clean_df['Receipt_Number'].value_counts()

# Filter receipt numbers with count > 2
selected_receipt_numbers = receipt_number_counts[receipt_number_counts >
2].index
```

```
# Filter the dataframe based on selected receipt numbers
selected_df =
clean_df[clean_df['Receipt_Number'].isin(selected_receipt_numbers)]

# Create a count plot for store branches
plt.figure(figsize=(12, 8))
sns.countplot(x='Store_Branch', data=selected_df)
plt.title("Store Branches for Receipt Numbers with Count > 2")
plt.xlabel("Store Branch")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.grid(True)

# Find the store with the maximum count
most_common_store = selected_df['Store_Branch'].mode().iloc[0]
max_count = selected_df['Store_Branch'].value_counts().max()

# Highlight the store with the most counts
ax = plt.gca()
for p in ax.patches:
    if p.get_height() == max_count:
        p.set_facecolor('red')  # Highlight the bar with the most counts

plt.show()

# Print the name of the store with the most counts
print(f"The store with the most counts is: {most_common_store} (Count:
{max_count})")

import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the count of each receipt number
receipt_number_counts = clean_df['Receipt_Number'].value_counts()

# Filter receipt numbers with count > 7
selected_receipt_numbers = receipt_number_counts[receipt_number_counts >
7].index

# Filter the dataframe based on selected receipt numbers
selected_df =
clean_df[clean_df['Receipt_Number'].isin(selected_receipt_numbers)]

# Create a count plot for categories
plt.figure(figsize=(20, 8))
sns.countplot(x='Category', data=selected_df)
plt.title("Categories for Receipt Numbers with Count > 7")
plt.xlabel("Category")
```

```
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.grid(True)
plt.gcf().set_size_inches(28, 12)

# Find the category with the maximum count
most_common_category = selected_df['Category'].mode().iloc[0]
max_count = selected_df['Category'].value_counts().max()

# Highlight the category with the most counts
ax = plt.gca()
for p in ax.patches:
    if p.get_height() == max_count:
        p.set_facecolor('red')  # Highlight the bar with the most counts

plt.show()

# Print the name of the category with the most counts
print(f"The category with the most counts is: {most_common_category} (Count:
{max_count})")

import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the counts for each product
product_counts = clean_df['Product'].value_counts()

# Identify products with counts less than 12
products_to_replace = product_counts[product_counts < 12].index

# Replace those products with 'Others'
clean_df['Product'] = clean_df['Product'].replace(products_to_replace,
'Others')

# Filter out 'Others' before creating the count plot
filtered_df = clean_df[clean_df['Product'] != 'Others']

# Create a count plot with descending order
plt.figure(figsize=(12, 8))
sns.countplot(x='Product', data=filtered_df,
order=filtered_df['Product'].value_counts().index[::-1])
plt.title("Count of Products (Counts > 11)")
plt.xlabel("Product")
plt.ylabel("Count")
plt.ylim(0, 40)
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```

```
# Create a count plot with ascending order
plt.figure(figsize=(22, 8))
sns.countplot(x='Store_Branch', data=clean_df,
order=clean_df['Store_Branch'].value_counts().index[::-1])
plt.title("Number of Products Bought in a Store Branch")
plt.xlabel("Store Branch")
plt.ylabel("# of Products Bought")
plt.xticks(rotation=90)
plt.grid(True)
plt.show()

# Create a count plot with asc order and ylim
plt.figure(figsize=(12, 8))
sns.countplot(x='Category', data=clean_df,
              order=clean_df['Category'].value_counts().index[::-1])  # Reverse
the order for descending
plt.title("Count of Category")
plt.xlabel("Category")
plt.ylabel("Count")
plt.xticks(rotation=90)
plt.ylim(0, 4000)  # Set y-axis limit
plt.grid(True)
plt.show()

# Count Plot for 'Discount'
plt.figure(figsize=(14, 10))
sns.countplot(x='Discount', data=clean_df, palette='pastel',
order=clean_df['Discount'].value_counts().index[::-1])
plt.title('Count of Discounts', fontdict={'fontname': 'serif', 'color':
'blue', 'size': 16})
plt.xlabel('Discount', fontdict={'fontname': 'serif', 'color': 'darkred',
'size': 14})
plt.ylabel('Count', fontdict={'fontname': 'serif', 'color': 'darkred', 'size':
14})
plt.show()

# Count Plot for 'Bonus_Cart'
plt.figure(figsize=(14, 10))
sns.countplot(x='Bonus_cart', data=clean_df, palette='pastel')
plt.title('Count of Bonus_cart', fontdict={'fontname': 'serif', 'color':
'blue', 'size': 16})
plt.xlabel('Bonus_cart', fontdict={'fontname': 'serif', 'color': 'darkred',
'size': 14})
plt.ylabel('Count', fontdict={'fontname': 'serif', 'color': 'darkred', 'size':
14})
plt.show()
```

```
# # Pie plots

# In[41]:


# Calculate the counts for each product
product_counts = clean_df['Product'].value_counts()

# Identify products with counts less than 12
products_to_replace = product_counts[product_counts < 12].index

# Replace those products with 'Others'
clean_df['Product'] = clean_df['Product'].replace(products_to_replace,
'Others')

# Remove 'Others' from the counts and labels
product_counts = product_counts[product_counts.index != 'Others']
product_labels = product_counts.index

# Create a pie plot with cool colors
cool_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
'#8c564b', '#e377c2', '#7f7f7f', '#bcbd22',
               '#17becf']
plt.figure(figsize=(27, 20))
plt.pie(product_counts, labels=product_labels, autopct='%1.1f%%',
startangle=140, colors=cool_colors,
        wedgeprops=dict(width=0.3))
plt.title("Distribution of Products (Counts > 11)")
plt.show()

# Create a DataFrame with counts of products by discount category
discount_counts = clean_df['Discount'].value_counts()

# Plotting
plt.figure(figsize=(10, 10))
plt.pie(discount_counts, labels=discount_counts.index, autopct='%1.1f%%',
startangle=90, colors=plt.cm.Paired.colors)
plt.title('Pie Chart of Discount Distribution')
plt.show()

# Create a DataFrame with counts of products by Bonus Cart usage
bonus_cart_counts = clean_df['Bonus_cart'].value_counts()

# Plotting
plt.figure(figsize=(8, 8))
plt.pie(bonus_cart_counts, labels=bonus_cart_counts.index, autopct='%1.1f%%',
startangle=90,
        colors=plt.cm.Accent.colors)
```

```
plt.title('Pie Chart of Bonus Cart Usage')
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the counts for each category
category_counts = clean_df['Category'].value_counts()

# Set a threshold for percentage labels
percentage_threshold = 1.0  # Display labels only for slices with percentage
greater than or equal to this threshold

# Identify slices with percentage below the threshold and combine them into
'Others' slice
below_threshold = category_counts[category_counts / category_counts.sum() *
100 < percentage_threshold]
category_counts['Others'] = below_threshold.sum()
category_counts = category_counts[category_counts / category_counts.sum() *
100 >= percentage_threshold]

# Plotting a pie chart with improved readability
plt.figure(figsize=(20, 20))
plt.pie(category_counts, labels=category_counts.index,
        autopct=lambda p: '{:.1f}%'.format(p) if p >= percentage_threshold else
'', startangle=90,
        colors=plt.cm.Set3.colors, textprops={'fontsize': 20})
plt.title('Pie Chart of Category Distribution', fontsize=20)
plt.show()

# # Distplot

# In[9]:


import seaborn as sns
import matplotlib.pyplot as plt

# Plotting a distplot for the 'Price' column
plt.figure(figsize=(12, 6))
sns.distplot(clean_df['Price'], bins=20, hist_kws={'color': 'skyblue'},
kde_kws={'color': 'orange'})
plt.title('Distribution of Prices')
plt.xlabel('Price')
plt.ylabel('Density')
plt.show()

# # Heatmap
```

```
# In[10]:


import seaborn as sns
import matplotlib.pyplot as plt

# Selecting numerical columns for correlation heatmap
numerical_columns = clean_df.select_dtypes(include=['float64',
'int64']).columns

# Calculate the correlation matrix for numerical columns
correlation_matrix = clean_df[numerical_columns].corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap for Numerical Columns')
plt.show()

# # Pairplot

# In[11]:


# Pairplot for Numerical Columns
numerical_columns = clean_df.select_dtypes(include=['int64', 'float64'])
sns.pairplot(numerical_columns)
plt.show()

# # Histplot

# In[12]:


import seaborn as sns
import matplotlib.pyplot as plt

# Plotting a histogram for the 'Price' column
plt.figure(figsize=(12, 6))
sns.histplot(clean_df['Price'], bins=20, kde=True, color='skyblue')
plt.title('Histogram of Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()

# # OO plot
```

```
# In[13]:


import statsmodels.api as sm

plt.figure(figsize=(8, 6))
sm.qqplot(clean_df["Price"], line='s', color='salmon')
plt.title("QQ-Plot for Price", fontdict={'fontname': 'serif', 'color': 'blue',
'size': 16})
plt.show()

# # KDE with fill

# In[14]:


# KDE Plot with Fill, Alpha, Palette, and Linewidth
sns.kdeplot(data=clean_df, x='Price', fill=True, alpha=0.6, palette='viridis',
linewidth=2)
plt.title('KDE Plot with Fill')
plt.show()

# # lm or regplot

# In[15]:


import seaborn as sns
import matplotlib.pyplot as plt

# Calculate mean prices for each unique date
mean_prices = new_df.groupby('Date')['Price'].mean().reset_index()

# Convert 'Date' to numeric values
mean_prices['Numeric_Date'] = mean_prices['Date'].apply(lambda x:
x.toordinal())

# Create a scatter plot with regression line
plt.figure(figsize=(24, 14))
sns.lmplot(x='Numeric_Date', y='Price', data=mean_prices, scatter_kws={'s':
100}, aspect=2, height=6)

# Adding labels and title
plt.xlabel('Numeric Date')
plt.ylabel('Mean Price')
plt.title('Scatter Plot with Regression Line of Mean Prices Over Unique
Dates')
```

```
# Show the plot
plt.show()

# # Multivariate Box or Boxen plot

# In[16]:


plt.figure(figsize=(32, 15))
sns.boxplot(x='Category', y='Price', hue='Discount', data=clean_df,
palette='viridis')
plt.xlabel('Category')
plt.ylabel('Price')
plt.title('Box Plot of Prices by Category and Discount')
plt.ylim(0, 100)
plt.xticks(rotation=45)
plt.show()

# # Area plot

# In[42]:


# Area Plot
plt.figure(figsize=(32, 15))
sns.lineplot(data=clean_df, x='Date', y='Price', hue='Category',
estimator='sum', ci=None)
plt.title('Area Plot Over Time')
plt.show()

# # Violin plot

# In[47]:


import seaborn as sns
import matplotlib.pyplot as plt

# Set the style of seaborn
sns.set(style="whitegrid")

# Set up the matplotlib figure with a larger size
plt.figure(figsize=(18, 12))

# Violin plot for Price distribution by Discount category
plt.subplot(2, 2, 1)
sns.violinplot(x='Discount', y='Price', data=clean_df, palette='viridis')
plt.xlabel('Discount', fontsize=14)
```

```
plt.ylabel('Price', fontsize=14)
plt.title('Violin Plot of Price by Discount', fontsize=16)

# Violin plot for Price distribution by Category
plt.subplot(2, 2, 2)
sns.violinplot(x='Category', y='Price', data=clean_df, palette='muted')
plt.xlabel('Category', fontsize=14)
plt.ylabel('Price', fontsize=14)
plt.title('Violin Plot of Price by Category', fontsize=16)
plt.xticks(rotation=45, ha='right')

# Violin plot for Price distribution by Store Branch
plt.subplot(2, 2, 3)
sns.violinplot(x='Store_Branch', y='Price', data=clean_df, palette='deep')
plt.xlabel('Store Branch', fontsize=14)
plt.ylabel('Price', fontsize=14)
plt.title('Violin Plot of Price by Store Branch', fontsize=16)
plt.xticks(rotation=45, ha='right')

# Violin plot for Price distribution by Discount and Category without 'split'

# sns.violinplot(x='Discount', y='Price', hue='Category', data=clean_df,
palette='pastel')
# plt.xlabel('Discount', fontsize=14)
# plt.ylabel('Price', fontsize=14)
# plt.title('Violin Plot of Price by Discount and Category', fontsize=16)
# Convert 'Date' column to datetime
clean_df['Date'] = pd.to_datetime(clean_df['Date'], dayfirst=True)
plt.subplot(2, 2, 4)
sns.violinplot(x=clean_df['Date'].dt.dayofweek, y='Price', data=clean_df,
inner='quartile', palette='coolwarm')
plt.xlabel('Day of Week')
plt.ylabel('Price')
plt.title('Daily Sales Distribution')
plt.show()

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()

# # Joinplot

# In[19]:


import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Convert 'Discount' to numeric labels
clean_df['Discount_Label'] = clean_df['Discount'].astype('category').cat.codes

# Set the style of seaborn
sns.set(style="whitegrid")

# Set up the matplotlib figure with a larger size
plt.figure(figsize=(12, 8))

# Create a joint plot with KDE and scatter representation
sns.jointplot(x='Price', y='Discount_Label', data=clean_df, kind='scatter',
hue='Category', height=8, palette='viridis')

# Add labels and title
plt.xlabel('Price', fontsize=14)
plt.ylabel('Discount', fontsize=14)
plt.title('Joint Plot with KDE and Scatter Representation', fontsize=16)

# Show the plot
plt.show()

# # Rugplot

# In[20]:


import matplotlib.pyplot as plt
import seaborn as sns

# Create a DataFrame with unique dates and mean prices
mean_prices_by_date = new_df.groupby('Date')['Price'].mean().reset_index()

# Create a rug plot for unique dates and mean prices
plt.figure(figsize=(16, 4))
rug_plot = sns.rugplot(data=mean_prices_by_date, x='Date', height=0.5,
hue='Price', palette='viridis')
plt.title('Rug Plot of Mean Prices Over Unique Dates')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.yticks([])
plt.show()

# # 3D plot

# In[21]:
```

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import dates

# Assuming 'Date' is the column name in your DataFrame
new_df['Date'] = pd.to_datetime(new_df['Date'])

# Calculate mean prices for each unique date
mean_prices = new_df.groupby('Date')['Price'].mean().reset_index()

# Create a 3D scatter plot
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Convert 'Date' to numerical format using matplotlib's date2num
num_dates = dates.date2num(mean_prices['Date'])

scatter = ax.scatter(num_dates, mean_prices['Price'], c=mean_prices['Price'],
cmap='viridis', s=50)

# Adding labels and title
ax.set_xlabel('Date')
ax.set_ylabel('Mean Price')
ax.set_zlabel('Mean Price')
ax.set_title('3D Scatter Plot of Mean Prices Over Time')

# Show the colorbar
fig.colorbar(scatter, ax=ax, label='Mean Price')

# Format x-axis as dates
ax.xaxis.set_major_locator(dates.YearLocator())
ax.xaxis.set_major_formatter(dates.DateFormatter('%Y-%m-%d'))

# Show the plot
plt.show()

# # Cluster map

# In[22]:


# Selecting numerical columns for correlation heatmap
numerical_columns = clean_df.select_dtypes(include=['float64',
'int64']).columns

# Calculate the correlation matrix for numerical columns
correlation_matrix = clean_df[numerical_columns].corr()
```

```
sns.clustermap(correlation_matrix, cmap='coolwarm', linewidths=0.5,
annot=True)
plt.title('Cluster Map')
plt.show()

# # Strip plot

# In[52]:


import matplotlib.pyplot as plt
import seaborn as sns

# Calculate mean prices for each unique date
mean_prices = new_df.groupby('Date')['Price'].mean().reset_index()

# Create a strip plot
plt.figure(figsize=(24, 12))
strip_plot = sns.stripplot(data=mean_prices, x='Date', y='Price', jitter=True,
size=5)
plt.title('Strip Plot of Mean Prices Over Unique Dates')
plt.xlabel('Date')
plt.ylabel('Mean Price')
plt.xticks(rotation=90)
plt.show()

# # Swarm plot

# In[54]:


import matplotlib.pyplot as plt
import seaborn as sns

# Calculate mean prices for each unique date
mean_prices = new_df.groupby('Date')['Price'].mean().reset_index()

# Create a swarm plot
plt.figure(figsize=(27, 12))
swarm_plot = sns.swarmplot(data=mean_prices, x='Date', y='Price', size=8)
plt.title('Swarm Plot of Mean Prices Over Unique Dates')
plt.xlabel('Date')
plt.ylabel('Mean Price')
plt.xticks(rotation=45)
plt.show()

# # Some questions
```

```
# In[25]:


# Which Category is the costliest?

# Group by 'Category' and calculate the average price for each category
average_prices_by_category =
clean_df.groupby('Category')['Price'].mean().sort_values(ascending=True)

# Plotting
plt.figure(figsize=(14, 12))
sns.barplot(x=average_prices_by_category.values,
y=average_prices_by_category.index, palette='viridis')

# Adding labels and title
plt.xlabel('Average Price')
plt.ylabel('Category')
plt.title('Average Prices by Category')

# Show the plot
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt

# Find the costliest product and its count as compared to others
costliest_product = clean_df.groupby('Product')['Price'].mean().idxmax()

# Filter the DataFrame for the costliest product
costliest_product_data = clean_df[clean_df['Product'] == costliest_product]

# Print the name, count, and average price of the costliest product
print(f"\nCostliest Product: {costliest_product}")
print(f"Count: {len(costliest_product_data)}")
print(f"Average Price: ${costliest_product_data['Price'].mean():.2f}")

# Plotting
plt.figure(figsize=(14, 10))
count_plot = sns.countplot(x='Product', data=clean_df,
order=clean_df['Product'].value_counts().index[::-1],
                           palette='viridis')

# Highlight the bar for the costliest product
costliest_product_index =
clean_df['Product'].value_counts().index.get_loc(costliest_product)
count_plot.patches[costliest_product_index].set_facecolor('red')

# Adding labels and title
```

```
plt.xlabel('Product')
plt.ylabel('Count')
plt.ylim(0, 50)
plt.title(f'Count of {costliest_product} (Costliest Product)')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Show the plot
plt.show()

# Which Store has the max sale in each season and by what count?

# Group by 'Discount' and 'Store_Branch', and calculate the count for each
combination
seasonal_store_counts = clean_df.groupby(['Discount',
'Store_Branch']).size().reset_index(name='Count')

# Find the store with the maximum count in each season
max_store_in_season =
seasonal_store_counts.loc[seasonal_store_counts.groupby('Discount')['Count'].i
dxmax()]

# Plotting
plt.figure(figsize=(14, 8))
ax = sns.barplot(x='Count', y='Store_Branch', hue='Discount',
data=max_store_in_season, palette='Set2')

# Adding labels and title
plt.xlabel('Count')
plt.ylabel('Store Branch')
plt.title('Store with Maximum Sale Count in Each Season')

# Show the plot
plt.show()

# Grouping by Store Branch and calculating the average count of Receipt
Numbers
average_receipt_count_by_branch =
clean_df.groupby('Store_Branch')['Receipt_Number'].count().reset_index()

# Sorting the DataFrame by average count in ascending order
average_receipt_count_by_branch =
average_receipt_count_by_branch.sort_values(by='Receipt_Number',
ascending=True)

# Plotting a bar plot with explicit order
plt.figure(figsize=(14, 8))
```

```
sns.barplot(x='Store_Branch', y='Receipt_Number',
data=average_receipt_count_by_branch,
            order=average_receipt_count_by_branch['Store_Branch'],
palette='viridis')
plt.title('Average Count of Receipt Numbers by Store Branch (Ascending
Order)')
plt.xlabel('Store Branch')
plt.ylabel('Average Count of Receipt Numbers')
plt.xticks(rotation=45)
plt.show()

# Convert 'Date' column to datetime with specified format
clean_df['Date'] = pd.to_datetime(clean_df['Date'], format='%d/%m/%Y %H:%M')

# Extract month and year from the 'Date' column
clean_df['Month'] = clean_df['Date'].dt.month
clean_df['Year'] = clean_df['Date'].dt.year

# Create subplots for each store branch
store_branches = clean_df['Store_Branch'].unique()

plt.figure(figsize=(35, 30))
for i, branch in enumerate(store_branches, 1):
    plt.subplot(7, 3, i)
    branch_data = clean_df[clean_df['Store_Branch'] == branch]
    monthly_prices = branch_data.groupby(['Year',
'Month'])['Price'].sum().reset_index()
    sns.lineplot(x='Month', y='Price', hue='Year', data=monthly_prices,
marker='o')
    plt.title(f'Total Revenue Over Months - {branch}')
    plt.xlabel('Month')
    plt.ylabel('Total Revenue')
    plt.legend(title='Year')

plt.tight_layout()
plt.show()

# Convert 'Date' column to datetime with specified format
clean_df['Date'] = pd.to_datetime(clean_df['Date'], format='%d/%m/%Y %H:%M')

# Extract month and year from the 'Date' column
clean_df['Month'] = clean_df['Date'].dt.month
clean_df['Year'] = clean_df['Date'].dt.year

# Find the store with the maximum total price for each month
max_price_per_month = clean_df.groupby(['Store_Branch', 'Year',
'Month'])['Price'].sum().reset_index()
idx = max_price_per_month.groupby(['Year', 'Month'])['Price'].idxmax()
```

```
max_price_per_month = max_price_per_month.loc[idx]

for _, row in max_price_per_month.iterrows():
    month_name =
pd.to_datetime(f'{int(row["Year"])}-{int(row["Month"])}-1').strftime('%B')
    print(f"For {month_name}, the store with the max total price is
{row['Store_Branch']} with ${row['Price']:.2f}")

# Set up subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16, 32))
fig.suptitle('Category Sold the Most in Each Season', fontsize=20)

# Group by 'Discount' and 'Category', and calculate the count for each
combination
seasonal_category_counts = clean_df.groupby(['Discount',
'Category']).size().reset_index(name='Count')

# List of unique seasons (discounts)
seasons = seasonal_category_counts['Discount'].unique()

# Iterate over each season and create a subplot
for i, season in enumerate(seasons, 1):
    plt.subplot(3, 2, i)

    # Filter data for the current season
    season_data = seasonal_category_counts[seasonal_category_counts['Discount']
== season]

    # Sort the data for the current season in descending order
    season_data = season_data.sort_values(by='Count', ascending=False)

    # Find the category with the maximum count in the current season
    max_category = season_data.loc[season_data['Count'].idxmax(), 'Category']

    # Plotting
    sns.barplot(x='Count', y='Category', data=season_data, palette='Set3')

    # Highlight the bar for the category with the maximum count
    max_category_index =
season_data['Category'].values.tolist().index(max_category)
    plt.gca().patches[max_category_index].set_facecolor('red')

    # Adding labels and title
    plt.xlabel('Count')
    plt.ylabel('Category')
    plt.xlim(0, 700)
    plt.title(f'Season: {season}')
```

```
# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.96])

# Show the plot
plt.show()



# Define cool and unique colors for each store branch
branch_colors = {
    'N.Narimanov': '#1f77b4',
    'Khatai': '#ff7f0e',
    'Machine market': '#2ca02c',
    'Ahmadli': '#d62728',
    'Hypermarket': '#9467bd',
    'M. Ajami': '#8c564b',
    'Ayna Sultanova': '#e377c2',
    'Narimanov-2': '#7f7f7f',
    'C. Mammadguluzade': '#bcbd22',
    'Almond': '#17becf',
    'People\'s friendship': '#aec7e8',
    'SEK 8 million': '#ffbb78',
    'Hazi Aslanov-1': '#98df8a',
    'Zabrat': '#c5b0d5',
    'Hazi Aslanov-2': '#c49c94',
    'Yasamal': '#f7b6d2',
    'Small church': '#dbdb8d',
    'Radiozavod': '#9edae5',
    '28-May': '#d62728',
    'New Yasamal': '#1f77b4',
    '20th January': '#2ca02c'
}

# Convert 'Date' column to datetime with specified format
clean_df['Date'] = pd.to_datetime(clean_df['Date'], format='%d/%m/%Y %H:%M')

# Extract month and year from the 'Date' column
clean_df['Month'] = clean_df['Date'].dt.month
clean_df['Year'] = clean_df['Date'].dt.year

# Create a line plot for total prices over months for each store branch with
custom colors
plt.figure(figsize=(24, 14))
for branch in store_branches:
    branch_data = clean_df[clean_df['Store_Branch'] == branch]
    monthly_prices = branch_data.groupby(['Year',
'Month'])['Price'].sum().reset_index()
```

```
    sns.lineplot(x='Month', y='Price', data=monthly_prices, marker='o',
label=branch, color=branch_colors[branch])

# Adding labels and title with increased font size
plt.title('Total Revenue Over Months for Each Store Branch', fontsize=24)
plt.xlabel('Month', fontsize=18)
plt.xlim(0, 9)
plt.ylabel('Total Revenue', fontsize=18)
plt.legend(title='Store Branch', bbox_to_anchor=(1.05, 1), loc='upper left',
fontsize=20)  # Increased font size

plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.show()

# Convert 'Date' column to datetime
clean_df['Date'] = pd.to_datetime(clean_df['Date'], format='%d/%m/%Y %H:%M')

# Extract month and year
clean_df['Month'] = clean_df['Date'].dt.month
clean_df['Year'] = clean_df['Date'].dt.year

# Create a line plot for total sales over months
plt.figure(figsize=(16, 8))

# Use the 'viridis' color palette
colors = sns.color_palette('viridis', n_colors=len(clean_df['Year'].unique()))

monthly_sales = clean_df.groupby(['Year',
'Month'])['Price'].sum().reset_index()
sns.lineplot(x='Month', y='Price', hue='Year', data=monthly_sales, marker='o',
palette=colors)

plt.title('Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.legend(title='Year')
plt.show()

# In[55]:


import pandas as pd
from prettytable import PrettyTable

# Assuming 'clean_df' is your DataFrame
```

```
# Convert 'Date' column to datetime with specified format
clean_df['Date'] = pd.to_datetime(clean_df['Date'], format='%d/%m/%Y %H:%M')

# Extract month and year from the 'Date' column
clean_df['Month'] = clean_df['Date'].dt.month
clean_df['Year'] = clean_df['Date'].dt.year

# Find the store with the maximum total price for each month
max_price_per_month = clean_df.groupby(['Store_Branch', 'Year',
'Month'])['Price'].sum().reset_index()
idx = max_price_per_month.groupby(['Year', 'Month'])['Price'].idxmax()
max_price_per_month = max_price_per_month.loc[idx]

# Create a PrettyTable
table = PrettyTable()
table.field_names = ["Month", "Store with Max Total Price", "Max Total Price"]

for _, row in max_price_per_month.iterrows():
    month_name =
pd.to_datetime(f'{int(row["Year"])}-{int(row["Month"])}-1').strftime('%B')
    table.add_row([month_name, row['Store_Branch'], f"${row['Price']:.2f}"])

print(table)

# # PCA and normality testing

# In[26]:


# Reading the dataset from the csv file
sub_df = pd.read_csv("/Users/shubhamlaxmikantdeshmukh/Downloads/esas_mehsullar
2.csv")

# #taking limited records
# df=df.head(120000)

# Renaming column Names
sub_df = sub_df.rename(
    columns={'Unnamed: 0': 'ID', 'satish_kodu': 'Receipt_Number',
'mehsul_kodu': 'Product_Code', 'mehsul_ad': 'Product',
             'mehsul_kateqoriya': 'Category', 'mehsul_qiymet': 'Price',
'satish_tarixi': 'Date',
             'endirim_kompaniya': 'Discount', 'bonus_kart': 'Bonus_cart',
'magaza_ad': 'Store_Branch',
             'magaza_lat': 'Store_Lat', 'magaza_long': 'Store_Long'})

# droping the na values in df and checking the if they are gone
sub_df = sub_df.dropna()
```

```
# dropping unescessay columns
sub_df = sub_df.drop('ID', axis=1)

# reseting index
sub_df = sub_df.reset_index(drop=True)

# checking for duplicate rows and dropping them
duplicated_rows = clean_df.duplicated()
clean_df = clean_df.drop_duplicates()

# Coverting type of some coulmns
sub_df = sub_df.copy()  # Create a copy of the DataFrame
sub_df['Receipt_Number'] = sub_df['Receipt_Number'].astype('int')
sub_df['Product'] = sub_df['Product'].astype('category')
sub_df.loc[:, 'Date'] = pd.to_datetime(sub_df['Date'], dayfirst=True)
sub_df['Category'] = sub_df['Category'].astype('category')
sub_df['Discount'] = sub_df['Discount'].astype('category')
sub_df['Store_Branch'] = sub_df['Store_Branch'].astype('category')

# Replace multiple values in the 'Discount' column
replace_dict = {'Sərin Yay günləri': 'Summer Season', 'S?rf?li Yaz': 'Fifth of
May Season',
                'Bərəkətli Novruz': 'Happy Nowruz Season', 'Yeni il
fürsətləri': 'New Year Season',
                'Payız endirimləri': 'Autumn Season'}
sub_df['Discount'] = sub_df['Discount'].replace(replace_dict)

# Translate values in the 'Category' and 'Store Branch' column
from googletrans import Translator

translator = Translator()
sub_df['Category'] = sub_df['Category'].apply(lambda x:
translator.translate(x).text)
sub_df['Store_Branch'] = sub_df['Store_Branch'].apply(lambda x:
translator.translate(x).text)

# Q2)
from sklearn.preprocessing import StandardScaler

# Assuming 'clean_df' is your DataFrame
scaler = StandardScaler()
numerical_data = sub_df.select_dtypes(include=[np.number])  # Select numerical
columns only

standardized_data = scaler.fit_transform(numerical_data)
print(standardized_data)
# Calculate the correlation matrix
```

```
correlation_matrix = pd.DataFrame(standardized_data).corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f")

# Add labels and title
plt.title("Correlation Coefficient Matrix Heatmap")
plt.show()

# Q4 d
from sklearn.decomposition import PCA

pca = PCA(svd_solver="full", n_components=0.95, random_state=5764)

# Assuming 'numerical_data' is your DataFrame with only numerical columns
pca_data = pca.fit_transform(standardized_data)

exp_var_pca = pca.explained_variance_ratio_

print(f'Explained variance ratio:', exp_var_pca.round(3))

# Q4 e
cum_sum_eigenvalues = np.cumsum(exp_var_pca) * 100

index_95_variance = np.argmax(cum_sum_eigenvalues >= 95)

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cum_sum_eigenvalues) + 1), cum_sum_eigenvalues,
marker='o', linestyle='-', color='blue')
plt.axhline(y=95, color='black', linestyle='--', label='95% Explained
Variance')
plt.axvline(x=index_95_variance + 1, color='red', linestyle='--',
label=f'Optimal Features: {index_95_variance + 1}')

plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance (%)')
plt.legend()
plt.title('Cumulative Explained Variance vs. Number of Components')
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# Print the first five rows of the reduced features
pca_data_df = pd.DataFrame(pca_data, columns=[f'PC_{i}' for i in range(1,
pca_data.shape[1] + 1)])
print('The first five rows of the reduced features:\n', pca_data_df.head())
```

```
condition_number_before = np.linalg.cond(numerical_data)
print(f'The condition number before standardization:
{condition_number_before:.2f}')

condition_number_after = np.linalg.cond(standardized_data)
print(f'The condition number after standardization:
{condition_number_after:.2f}')

print(f'The difference: {(condition_number_before -
condition_number_after):.2f}')

import numpy as np
from scipy.stats import kstest


def ks_test(x, title):
    mean = np.mean(x)
    std = np.std(x)
    dist = np.random.normal(mean, std, len(x))
    stats, p = kstest(x, dist)
    return stats, p


# Perform K-S test for 'Price'
ks_stat_price, p_value_price = ks_test(sub_df['Price'], 'Raw')

# Interpret the K-S test result for 'Price'
if p_value_price < 0.01:
    result_price = "not normal"
else:
    result_price = "normal"

# Display K-S test results for 'Price'
print("K-S test for Price: statistics = {:.2f}, p-value =
{:.2f}".format(ks_stat_price, p_value_price))
print("K-S test: Price dataset looks {}".format(result_price))

import matplotlib.pyplot as plt

# Q16 - Create a boxplot of 'Price'
plt.figure(figsize=(8, 6))
plt.boxplot(clean_df['Price'], vert=False)

# Add labels and title
plt.xlabel('Price')
plt.title('Boxplot of Price with Outliers')

# Display the plot
```

```
plt.show()

# Q17 - Calculate Q1, Q3, and IQR for 'Price'
Q1_price = clean_df['Price'].quantile(0.25)
Q3_price = clean_df['Price'].quantile(0.75)
IQR_price = Q3_price - Q1_price

# Identify outliers for 'Price'
lower_bound_price = Q1_price - 1.5 * IQR_price
upper_bound_price = Q3_price + 1.5 * IQR_price

# Remove outliers from the dataset
cleaned_df_price = clean_df[(clean_df['Price'] >= lower_bound_price) &
(clean_df['Price'] <= upper_bound_price)]

# Q17 - Create a boxplot of cleaned 'Price'
plt.figure(figsize=(8, 6))
plt.boxplot(cleaned_df_price['Price'], vert=False)

# Add labels and title
plt.xlabel('Price')
plt.title('Boxplot of Cleaned Price')

# Display the plot
plt.show()

from scipy.stats import shapiro


def shapiro_test(x):
    stats, p = shapiro(x)
    return stats, p


# Perform Shapiro-Wilk test for 'Price'
stat_price, p_value_price = shapiro_test(cleaned_df_price['Price'])

# Interpret the Shapiro-Wilk test result for 'Price'
if p_value_price < 0.01:
    result_price = "not normal"
else:
    result_price = "normal"

# Display Shapiro-Wilk test results for 'Price'
print("Shapiro-Wilk test for Price: statistics = {:.2f}, p-value =
{:.2f}".format(stat_price, p_value_price))
print("Shapiro-Wilk test: Price data looks {}".format(result_price))
```

```
from scipy.stats import shapiro, kstest
from sklearn.preprocessing import StandardScaler
import numpy as np

# Standardize the 'Price' column
scaler = StandardScaler()
standard_Price = scaler.fit_transform(sub_df[['Price']])
print(standard_Price)


def ks_test(x, title):
    mean = np.mean(x)
    std = np.std(x)
    dist = np.random.normal(mean, std, len(x))
    stats, p = shapiro(x.flatten())  # Change this line
    return stats, p


# Perform Shapiro-Wilk test for standardized 'Price'
stat_price_std, p_value_price_std = ks_test(standard_Price, 'Raw')

# Interpret the Shapiro-Wilk test result for standardized 'Price'
if p_value_price_std < 0.01:
    result_price_std = "not normal"
else:
    result_price_std = "normal"

# Display Shapiro-Wilk test results for standardized 'Price'
print("Shapiro-Wilk test for Standardized Price: statistics = {:.2f}, p-value
= {:.2f}".format(stat_price_std,

p_value_price_std))
print("Shapiro-Wilk test: Standardized Price data looks
{}".format(result_price_std))

# In[56]:


from scipy.stats import normaltest


def da_k_squared_test(x):
    stats, p = normaltest(x)
    return stats, p


# Perform Shapiro-Wilk test for standardized 'Price'
```

```
da_k_squared_stat_price, p_value_price_std =
da_k_squared_test(clean_df['Price'])

# Interpret the Shapiro-Wilk test result for standardized 'Price'
if p_value_price_std < 0.01:
    result_price_std = "not normal"
else:
    result_price_std = "normal"

# Display Shapiro-Wilk test results for standardized 'Price'
print("D'Agostino's K^2 test for Price: statistics = {:.2f}, p-value =
{:.2f}".format(da_k_squared_stat_price,

p_value_price_std))
print("D'Agostino's K^2 test: Price looks {}".format(result_price_std))

# In[ ]:
```

Dash App app.py:

```
from dash import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from googletrans import Translator
import webbrowser
import plotly.graph_objects as go
from datetime import datetime
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import folium
from folium.plugins import MarkerCluster
from branca.colormap import linear
import warnings
warnings.filterwarnings("ignore")
```

```
import statsmodels
from plotly.subplots import make_subplots
from matplotlib import dates
import io
import json
import base64
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
from dash.exceptions import PreventUpdate
import plotly.express as px
import plotly.io as pio
from scipy.stats import kstest



external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

# Create the Dash app
Final_Term_Project_app = dash.Dash("Information Visualization Final Term Project app",
external_stylesheets=external_stylesheets)

##############################
df = pd.read_csv('esas.csv')
df = df.head(120000)
df = df.rename(
    columns={'Unnamed: 0': 'ID', 'satish_kodu': 'Receipt_Number', 'mehsul_kodu':
'Product_Code', 'mehsul_ad': 'Product',
             'mehsul_kategoriya': 'Category', 'mehsul_qiymet': 'Price',
'satish_tarixi': 'Date',
             'endirim_kompaniya': 'Discount', 'bonus_kart': 'Bonus_cart', 'magaza_ad':
'Store_Branch',
             'magaza_lat': 'Store_Lat', 'magaza_long': 'Store_Long'})
clean_df = df.dropna()
clean_df = clean_df.drop('ID', axis=1)
clean_df = clean_df.reset_index(drop=True)
duplicated_rows = clean_df.duplicated()
clean_df = clean_df.drop_duplicates()
clean_df = clean_df.copy()  # Create a copy of the DataFrame
clean_df['Product'] = clean_df['Product'].astype('category')
```

```
clean_df.loc[:, 'Date'] = pd.to_datetime(clean_df['Date'], dayfirst=True)
clean_df['Category'] = clean_df['Category'].astype('category')
clean_df['Discount'] = clean_df['Discount'].astype('category')
clean_df['Store_Branch'] = clean_df['Store_Branch'].astype('category')
replace_dict = {'S9rin Yay günl9ri': 'Summer Season', 'S?rf?li Yaz': 'Fifth of May
Season',
                'B9r9k9tli Novruz': 'Happy Nowruz Season', 'Yeni il fürs9tl9ri': 'New
Year Season',
                'Payız endiriml9ri': 'Autumn Season'}
clean_df['Discount'] = clean_df['Discount'].replace(replace_dict)


translator = Translator()  # Increase the timeout value (in seconds)

clean_df['Category'] = clean_df['Category'].apply(lambda x:
translator.translate(x).text)
clean_df['Store_Branch'] = clean_df['Store_Branch'].apply(lambda x:
translator.translate(x).text)

product_counts = clean_df['Product'].value_counts()
products_to_replace = product_counts[product_counts < 12].index
clean_df['Product'] = clean_df['Product'].replace(products_to_replace, 'Others')

clean_df['Date'] = pd.to_datetime(clean_df['Date'], format='%d/%m/%Y %H:%M')
clean_df['Month'] = clean_df['Date'].dt.month
clean_df['Year'] = clean_df['Date'].dt.year



product_counts = clean_df['Product'].value_counts()
if 'Others' not in product_counts.index:
    product_counts['Others'] = 0
products_to_replace = product_counts[product_counts < 12].index
clean_df['Product'] = clean_df['Product'].replace(products_to_replace, 'Others')

new_df = clean_df[['Date', 'Price']].copy()
new_df['Date'] = pd.to_datetime(new_df['Date']).dt.date
unique_dates_count = new_df['Date'].nunique()
new_df.sort_values(by='Date', inplace=True)
##########################
```

```
server = Final_Term_Project_app.server

Final_Term_Project_app.layout = html.Div(
    [
        html.H3('Information Visualization Final Term Project app',
style={"text-align": 'center', "color": 'blue', "font-size": '36px'}),
        dcc.Tabs(id='tabs',
                 value="q1",
                 children=[
                     dcc.Tab(label='About', value="about"),
                     dcc.Tab(label="Line plot", value="p1"),
                     dcc.Tab(label="Bar plot", value="p2"),
                     dcc.Tab(label="Count plot", value="p3"),
                     dcc.Tab(label="Pie plot", value="p4"),
                     dcc.Tab(label="3D plot", value="p5"),
                     dcc.Tab(label="Area plot", value="p6"),
                     dcc.Tab(label="PCA and other plots", value="p7"),

                 ]),
        html.Div(id="tabs-content")
    ]
)
#####################################################

# Define the layout of the web application
about_layout = html.Div(
    [
        html.H1("Data Analysis of 21 Supermarkets in Baku City"),
        html.H6('About the Dataset'),
        html.P(
            id='dataset-info',
            children=[
                "The dataset used in this application provides a comprehensive
information about various supermarket transactions in the year 2019 in the city of
Baku, Azerbaijan.",
                " It encompasses a vast array of data, including details on a total of
438,826 products that were purchased from these supermarkets.",
                " These products were bought by a customer base of around 80,000
individuals, highlighting the breadth of consumer engagement.",
```

```
            " The transactions were distributed across 21 branches of the
supermarket, capturing the geographical spread and operational diversity of the
business during the specified year of 2019."
        ]
    ),

    html.P("The geo-spatial map of these supermarkets along with the Total revenue
is shown below in the map:"),

    # Add an iframe to display the contents of folium_map.html
    html.Iframe(srcDoc=open('assets/folium_map.html', 'r').read(), width='90%',
height='400px'),

    html.Br(),

    html.P(id='dataset-small-info', children=[
    "The dataset used in this application provides a comprehensive view of
supermarket transactions in Baku, Azerbaijan in 2019.",
    html.Br(),
    html.Strong("Key Features:"),
    html.Ul([
        html.Li("Total Products / Rows : 438,826"),
        html.Li("Total Columns : 12"),
        html.Li("Customers: 80,000"),
        html.Li("Branches: 21"),
        ]),

    html.Strong("Steps used to preprocess the dataset:"),
    html.Ul([
        html.Li(" Dropped all the Null (NA) values from the dataset."),
        html.Li(" Removed 'ID' column from the dataset."),
        html.Li(" Removed any duplicate rows from the dataset."),
        html.Li(" Replaced the column name of the dataset from Azerbaijani to
English."),
        html.Li(" Translated the values of columns 'Category' and Store_Branch'
from Azerbaijani to English using 'googleTranslator' library."),
        ]),
    ]),
    html.Br(),
    html.Strong("The information of the cleaned or preprocessed dataset is shown
below:"),
```

```
        html.Div([
            html.Img(src='assets/clean_df_info.png', style={'width': '150%'}),
        ], style={'width': '30%', 'vertical-align': 'middle'}),

        html.Button(
            'Visit Dataset',
            id='visit-website-button',
            n_clicks=0,
            style={'margin-top': '10px'}
        ),

        dcc.Location(id='url', refresh=False),
        html.A(id='external-link', target='_blank', style={'display': 'none'}),

        html.Br(),

        html.H6("A Project By Shubham Laxmikant Deshmukh"),

    ]
)

# Define callback to update the external link
@Final_Term_Project_app.callback(
    Output('external-link', 'href'),
    [Input('visit-website-button', 'n_clicks')]
)
def update_external_link(n_clicks):
    if n_clicks is not None and n_clicks > 0:
        # Open the link in a new tab or window

webbrowser.open_new_tab("https://www.kaggle.com/datasets/mexwell/supermarket-dataset")

    # Return an empty string to prevent the page from refreshing
    return ''



##############################################
# Store branch colors
branch_colors = {
```

```
    'N.Narimanov': '#1f77b4',
    'Khatai': '#ff7f0e',
    'Machine market': '#2ca02c',
    'Ahmadli': '#d62728',
    'Hypermarket': '#9467bd',
    'M. Ajami': '#8c564b',
    'Ayna Sultanova': '#e377c2',
    'Narimanov-2': '#7f7f7f',
    'C. Mammadguluzade': '#bcbd22',
    'Almond': '#17becf',
    "People\'s friendship": '#aec7e8',
    'SEK 8 million': '#ffbb78',
    'Hazi Aslanov-1': '#98df8a',
    'Zabrat': '#c5b0d5',
    'Hazi Aslanov-2': '#c49c94',
    'Yasamal': '#f7b6d2',
    'Small church': '#dbdb8d',
    'Radiozavod': '#9edae5',
    '28-May': '#d62728',
    'New Yasamal': '#1f77b4',
    '20th January': '#2ca02c'
}


# Define the layout of the app
plot1_layout = html.Div(
    [
        html.H2('Various Line plots from the Dataset'),
        html.Label('Select Store Branches:'),
        dcc.Dropdown(
            id='store-branches-dropdown',
            options=[{'label': branch, 'value': branch} for branch in
clean_df['Store_Branch'].unique()],
            value=['N.Narimanov', 'Khatai', 'Machine market', 'Ahmadli', 'Hypermarket',
'M. Ajami', 'Ayna Sultanova',
                   'Narimanov-2', 'C. Mammadguluzade', 'Almond', "People's friendship",
'SEK 8 million',
                   'Hazi Aslanov-1', 'Zabrat', 'Hazi Aslanov-2', 'Yasamal', 'Small
church', 'Radiozavod', '28-May',
                   'New Yasamal', '20th January'],
            multi=True
```

```
        ),
        dcc.Graph(id='revenue-plot'),
        html.Br(),  # Add a horizontal line to separate the two graphs
        dcc.Graph(id='total-revenue-plot')  # New graph for total revenue
    ]
)

# Callback to update the individual store branches revenue plot
@Final_Term_Project_app.callback(
    Output('revenue-plot', 'figure'),
    [Input('store-branches-dropdown', 'value')]
)
def update_revenue_plot(selected_branches):
    # Filter data for selected store branches
    filtered_data = clean_df[clean_df['Store_Branch'].isin(selected_branches)]

    # Group by 'Year', 'Month', and 'Store_Branch' and calculate the sum of prices
    monthly_prices = filtered_data.groupby(['Year', 'Month',
'Store_Branch'])['Price'].sum().reset_index()

    # Create a line plot using plotly express
    fig = px.line(monthly_prices, x='Month', y='Price', color='Store_Branch',
                  labels={'x': 'Month', 'y': 'Total Revenue'}, line_shape='linear',
                  color_discrete_map=branch_colors)

    fig.update_layout(title='Total Revenue Over Months for Each Store Branch',
                      xaxis_title='Month', yaxis_title='Total Revenue ($)',
                      title_x=0.5, title_y=0.95)  # Adjust title_x and title_y for
positioning

    # Format y-axis labels as currency
    fig.update_layout(yaxis=dict(tickformat='$,.2f'))


    # Update x-axis labels to display month names
    month_names = [ "January", "February", "March", "April", "May", "June", "July",
"August", "September"]
    fig.update_layout(xaxis=dict(tickvals=list(range(1, 10)), ticktext=month_names))
    fig.update_xaxes(range=[1, 9])
    return fig
```

```
# Callback to update the total revenue plot
@Final_Term_Project_app.callback(
    Output('total-revenue-plot', 'figure'),
    [Input('store-branches-dropdown', 'value')]
)
def update_total_revenue_plot(selected_branches):
    # Filter data for selected store branches
    filtered_data = clean_df[clean_df['Store_Branch'].isin(selected_branches)]

    # Group by 'Year', 'Month', and calculate the total revenue for all branches
    total_revenue = filtered_data.groupby(['Year',
'Month'])['Price'].sum().reset_index()

    # Create a line plot for total revenue
    total_fig = px.line(total_revenue, x='Month', y='Price', labels={'x': 'Month', 'y':
'Total Revenue'},
                        line_shape='linear', color_discrete_sequence=['black'])

    # Adding labels and title
    total_fig.update_layout(title='Total Revenue Over Months for All Store Branches',
                            xaxis_title='Month', yaxis_title='Total Revenue ($)',
                            title_x=0.5, title_y=0.95)  # Adjust title_x and title_y
for positioning

    # Format y-axis labels as currency
    total_fig.update_layout(yaxis=dict(tickformat='$,.2f'))

    # Update x-axis labels to display month names
    month_names = ["January", "February", "March", "April", "May", "June", "July",
"August", "September"]
    total_fig.update_layout(xaxis=dict(tickvals=list(range(1, 10)),
ticktext=month_names))

    total_fig.update_xaxes(range=[1, 9])
    return total_fig


################################
# Define the layout of the app
```

```
# Group by 'Discount' and 'Category', and calculate the count for each combination
seasonal_category_counts = clean_df.groupby(['Discount',
'Category']).size().reset_index(name='Count')

# List of unique seasons (discounts)
seasons = seasonal_category_counts['Discount'].unique()


# Calculate average prices by category
average_prices_by_category =
clean_df.groupby('Category')['Price'].mean().sort_values(ascending=False).reset_index(
)

# Group by 'Discount' and 'Product', and calculate the count for each combination
seasonal_product_counts = clean_df.groupby(['Discount',
'Product']).size().reset_index(name='Count')

# List of unique seasons (discounts)
seasons = seasonal_product_counts['Discount'].unique()

# Create a dictionary to map season names to numerical indices
season_indices = {season: index for index, season in enumerate(seasons)}

# Define the layout of the app
plot2_layout = html.Div([
    html.H2('Various Bar plots from the Dataset'),
    html.H6("Select the season of Discount to get the category with maximum sale:"),
    dcc.RadioItems(
        id='season-radio',
        options=[{'label': season, 'value': season} for season in seasons],
        value=seasons[0],
        labelStyle={'display': 'block'}
    ),
    dcc.Graph(id='seasonal-category-plot'),
    html.Hr(),
    html.H2("Average Prices by Category"),

    dcc.Dropdown(
        id='sort-dropdown',
        options=[
            {'label': 'Ascending', 'value': 'asc'},
```

```
            {'label': 'Descending', 'value': 'desc'}
        ],
        value='desc',
        style={'width': '50%'}
    ),

    dcc.Graph(
        id='average-price-bar-plot',
    ),

    html.Div(id='top-categories-output'),


])

@Final_Term_Project_app.callback(
    dash.dependencies.Output('seasonal-category-plot', 'figure'),
    [dash.dependencies.Input('season-radio', 'value')]
)
def update_plot(selected_season):
    # Filter data for the selected season
    season_data = seasonal_category_counts[seasonal_category_counts['Discount'] ==
selected_season]

    # Sort the data for the selected season in descending order
    season_data = season_data.sort_values(by='Count', ascending=False)

    # Find the category with the maximum count in the selected season
    max_category = season_data.loc[season_data['Count'].idxmax(), 'Category']

    # Plotting with Plotly Express
    fig = px.bar(
        season_data,
        x='Count',
        y='Category',
        color='Category',
        text='Count',
        title=f'Category Sold the Most in {selected_season}',
        labels={'Count': 'Category Count'},
        template='plotly',
        color_discrete_sequence=px.colors.qualitative.Set3
```

```
    )

    # Highlight the bar for the category with the maximum count
    fig.update_traces(marker=dict(color='red'), selector=dict(name=max_category))

    # Adding labels and title
    fig.update_layout(title_x=0.5, title_y=0.95)  # Adjust title_x and title_y for
positioning

    return fig


# Define callback to update the graph and print top 5 categories
@Final_Term_Project_app.callback(
    [Output('average-price-bar-plot', 'figure'),
     Output('top-categories-output', 'children')],
    [Input('sort-dropdown', 'value')]
)
def update_graph_sort(sort_order):
    sorted_df = average_prices_by_category.sort_values(by='Price',
ascending=(sort_order == 'desc'))

    # Get top 5 categories and their counts
    top_categories = sorted_df.tail(5)
    top_categories_output = f"Top 5 Categories ({sort_order}): \n \n" +
top_categories.to_string(index=False)

    fig = px.bar(
        sorted_df,
        x='Price',
        y='Category',
        orientation='h',
        labels={'Price': 'Average Price ($)', 'Category': 'Category'},
        title='Average Prices by Category',
        color='Price',
        color_continuous_scale='viridis'
    )

    # Format x-axis labels as currency
    fig.update_layout(xaxis=dict(tickformat='$,.2f'))
```

```
    # Adding labels and title
    fig.update_layout(title_x=0.5, title_y=0.95)  # Adjust title_x and title_y for positioning

    return fig, top_categories_output


#####################################################
# Find the costliest product and its count as compared to others
costliest_product = clean_df.groupby('Product')['Price'].mean().idxmax()

# Filter the DataFrame for the costliest product
costliest_product_data = clean_df[clean_df['Product'] == costliest_product]

# Define the layout of the app
plot3_layout = html.Div([
    html.H1(f"Count of Products in the Supermarket"),

    html.H6(f"Choose the threshold for count of products from  which you want the countplot of Products: "),

    dcc.Slider(
        id='count-threshold-slider',
        min=10,
        max=50,
        step=1,
        value=10,
        marks={i: str(i) for i in range(10, 51)},
        tooltip={'placement': 'bottom', 'always_visible': True}
    ),

    # Add a loading component
    dcc.Loading(
        id="loading-costliest-product",
        type="default",
        children=[
            dcc.Graph(id='count-plot'),
        ]
    ),

    html.Div([
```

```
        html.Pre(id='costliest-product-info')
    ]),

    # Add a download button
    html.Button("Download the clean_df csv here", id="download-button"),
    dcc.Download(id="download-costliest-plot")
])

# Define callback to update the graph and print costliest product info
@Final_Term_Project_app.callback(
    [Output('count-plot', 'figure'),
     Output('costliest-product-info', 'children'),
     Output("loading-costliest-product", "children")],
    [Input('count-plot', 'relayoutData'),
     Input('count-threshold-slider', 'value')]
)
def update_graph(relayout_data, count_threshold):
    # Filter the DataFrame based on the count threshold
    filtered_df = clean_df.groupby('Product').filter(lambda x: len(x) >=
count_threshold)

    # Create a bar plot using Plotly Express
    fig = px.bar(
        filtered_df,
        x='Product',
        title=f'Count of all Products with count: {count_threshold} or more ',
        labels={'Product': 'Products', 'count': 'Count', 'Price': 'Price ($)'},
        category_orders={'Product': filtered_df['Product'].value_counts().index[::-1]},
        color='Price',
        color_continuous_scale='viridis',  # Optional: Set color scale
        color_discrete_map={col: f"${col}" for col in filtered_df['Price'].unique()}  #
Add currency symbol to legend
    )

    # Limit y-axis for better readability
    fig.update_yaxes(range=[0, 50])

    # Adding labels and title
    fig.update_layout(title_x=0.5)

    # Rotate x-axis labels for better readability
```

```
    fig.update_xaxes(tickangle=90, tickmode='array')

    fig.update_layout(height=1000, width=1200)

    # Extracting the relevant information for display in Dash
    costliest_product_info = f"Costliest Product among the Dataset:
{costliest_product}\nCount: {len(costliest_product_data)}\nAverage Price:
${costliest_product_data['Price'].mean():.2f}"

    return fig, costliest_product_info, [dcc.Graph(id='count-plot', figure=fig)]

# Callback to download the clean_df as CSV
@Final_Term_Project_app.callback(
    Output("download-costliest-plot", "data"),
    [Input("download-button", "n_clicks")],
    prevent_initial_call=True
)
def download_costliest_plot(n_clicks):
    return dcc.send_data_frame(clean_df.to_csv, "clean_df.csv")




##################################################

# Convert 'Date' column to datetime
clean_df['Date'] = pd.to_datetime(clean_df['Date'])

# Calculate the counts for each product
product_counts = clean_df['Product'].value_counts()

# Identify products with counts less than 12
products_to_replace = product_counts[product_counts < 12].index

# Replace those products with 'Others'
clean_df['Product'] = clean_df['Product'].replace(products_to_replace, 'Others')

# Filter out 'Others' from the counts and labels
product_counts = product_counts[product_counts.index != 'Others']
product_labels = product_counts.index
```

```
# Calculate the counts for each category
category_counts = clean_df['Category'].value_counts()

# Set a threshold for percentage labels
percentage_threshold = 1.0  # Display labels only for slices with percentage greater than or equal to this threshold

# Identify slices with percentage below the threshold and combine them into 'Others' slice
below_threshold = category_counts[category_counts / category_counts.sum() * 100 < percentage_threshold]
category_counts['Others'] = below_threshold.sum()
category_counts = category_counts[category_counts / category_counts.sum() * 100 >= percentage_threshold]

# Create a DataFrame with counts of products by discount category
discount_counts = clean_df['Discount'].value_counts()


# Define app layout
plot4_layout = html.Div(children=[
    html.H2('Various Pie plots from the Dataset'),

    # Dropdown for selecting pie plot
    dcc.Dropdown(
        id='pie-plot-dropdown',
        options=[
            {'label': 'Product Distribution', 'value': 'product'},
            {'label': 'Category Distribution', 'value': 'category'},
            {'label': 'Discount Distribution', 'value': 'discount'},
        ],
        value='product',  # Default value
        style={'width': '50%'}
    ),

    html.Hr(),

    # Placeholder for the selected pie plot
    dcc.Graph(id='selected-pie-plot'),

])
```

```
# Callback to update the selected pie plot based on the dropdown value
@Final_Term_Project_app.callback(
    Output('selected-pie-plot', 'figure'),
    [Input('pie-plot-dropdown', 'value')]
)
def update_pie_plot(selected_plot):
    if selected_plot == 'product':
        # Pie plot for 'Product' distribution
        fig = px.pie(clean_df[clean_df['Product'] != 'Others'], names='Product',
                     title="Distribution of Products (Counts > 11, excluding
'Others')")
    elif selected_plot == 'category':
        # Pie chart for 'Category' distribution
        fig = go.Figure(data=[go.Pie(labels=category_counts.index,
values=category_counts,
                                     hoverinfo='label+percent',
textinfo='label+percent', textfont_size=16,
                                     hole=0.3)])
        fig.update_layout(title=dict(
            text='Pie Chart of Category Distribution',
            x=0.5,
            y=0.95
        ))
    elif selected_plot == 'discount':
        # Pie chart for 'Discount' distribution
        fig = go.Figure(data=[go.Pie(labels=discount_counts.index,
values=discount_counts,
                                     hoverinfo='label+percent',
textinfo='label+percent', textfont_size=16,
                                     hole=0.3)])
        fig.update_layout(title=dict(
            text='Pie Chart of Discount Distribution',
            x=0.5,
            y=0.95
        ))
    else:
        fig = go.Figure()

    fig.update_layout(height=1000, width=1400, title_x=0.5)
```

```
    return fig


##########################################
# Assuming 'Date' is the column name in your DataFrame
new_df['Date'] = pd.to_datetime(new_df['Date'], dayfirst=True)

# Calculate mean prices for each unique date
mean_prices = new_df.groupby('Date')['Price'].mean().reset_index()

# Create a 3D scatter plot
fig = make_subplots(rows=1, cols=1, specs=[[{'type': 'scatter3d'}]])

scatter = go.Scatter3d(
    x=dates.date2num(mean_prices['Date']),
    y=mean_prices['Price'],
    z=mean_prices['Price'],
    mode='markers',
    marker=dict(
        size=20,
        color=mean_prices['Price'],
        colorscale='Viridis',
        opacity=0.8
    ),
    text=mean_prices['Date'].dt.strftime('%Y-%m-%d'),
    name='Mean Prices'
)

fig.add_trace(scatter)

# Adding labels and title
fig.update_layout(
    scene=dict(
        xaxis_title='Date',
        yaxis_title='Mean Price ($)',
        zaxis_title='Mean Price ($)'
    ),
    title=dict(
        text='3D Scatter Plot of Mean Prices Over Time',
        x=0.5,  # Set x to 0.5 to center the title horizontally
        xanchor='center'  # Center the title horizontally
```

```
    )
)

# Format y-axis labels as currency
fig.update_layout(scene=dict(yaxis=dict(tickformat='$,.2f')))

# Improve the formatting of date labels
fig.update_xaxes(
    tickvals=dates.date2num(mean_prices['Date']),
    ticktext=mean_prices['Date'].dt.strftime('%Y-%m-%d'),
    tickangle=45
)

fig.update_layout(height=800, width=1200)



# App layout
plot5_layout = html.Div(children=[


    # Output for displaying the plot based on user input
    html.Div(
        id='output-plot',
        children=[
            # 3D Scatter plot
            html.H3('3D plot from the Dataset'),
            dcc.Graph(
                id='scatter-3d-plot',
                figure=fig
            ),
        ]
    ),

    # Input for user to choose plot type
        dcc.Input(
            id='plot-type-input',
            type='text',
            placeholder='Enter plot type (e.g., 3D)',
            value=''
        ),
```

```
])

# Callback to update the visibility of the 3D plot based on user input
@Final_Term_Project_app.callback(
    Output('scatter-3d-plot', 'style'),
    [Input('plot-type-input', 'value')]
)
def update_plot_visibility(plot_type):
    # Display the 3D plot if user input is '3D', otherwise hide it
    return {'display': 'block'} if plot_type.lower() == '3d' else {'display': 'none'}



########################################
# Assuming 'Date' is the column name in your DataFrame
clean_df['Date'] = pd.to_datetime(clean_df['Date'], dayfirst=True)

# Define the layout
plot6_layout = html.Div([
    html.H1('Area Plot of Various Categories Over Time'),

    html.H6("Choose various categories for the area plot: "),

    # Checklist for selecting categories
    dcc.Checklist(
        id='category-checklist',
        options=[
            {'label': 'Sweets', 'value': 'Sweets'},
            {'label': 'Household products', 'value': 'Household products'},
            {'label': 'Vodka', 'value': 'Vodka'},
            {'label': 'Fruit juices', 'value': 'Fruit juices'},
            {'label': 'Dishes', 'value': 'Dishes'},
            {'label': 'Spices', 'value': 'Spices'},
            {'label': 'Tea', 'value': 'Tea'}
        ],
        value=['Fruit juices'],  # Set default value to Fruit juices
        labelStyle={'display': 'block'}
    ),
    html.Br(),
    html.H6("Choose from which month to what month you want to see the Area plot over
time:"),
    # Range slider for months
```

```
    dcc.RangeSlider(
        id='month-slider',
        marks={i: month for i, month in enumerate(['Jan', 'Feb', 'Mar', 'Apr', 'May',
'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])},
        min=0,
        max=11,
        step=1,
        value=[0, 2]  # Set default months to Jan to Mar
    ),
    html.Br(),

    # Area plot
    dcc.Graph(id='area-plot')
])

# Define callback to update the area plot based on selected categories and months
@Final_Term_Project_app.callback(
    Output('area-plot', 'figure'),
    [Input('category-checklist', 'value'),
     Input('month-slider', 'value')]
)
def update_area_plot(selected_categories, selected_months):
    # Filter data based on selected categories and months
    filtered_data = clean_df[clean_df['Category'].isin(selected_categories)]
    filtered_data = filtered_data[(filtered_data['Date'].dt.month >= selected_months[0]
+ 1) & (filtered_data['Date'].dt.month <= selected_months[1] + 1)]

    # Group by 'Date' and 'Category', and calculate the sum of prices
    grouped_data = filtered_data.groupby(['Date',
'Category'])['Price'].sum().reset_index()

    # Create traces for each category
    traces = []
    for category in selected_categories:
        category_data = grouped_data[grouped_data['Category'] == category]
        trace = go.Scatter(
            x=category_data['Date'],
            y=category_data['Price'],
            mode='lines',
            stackgroup='one',
            name=category
```

```
        )
        traces.append(trace)

    # Layout settings
    layout = go.Layout(
        title='Area Plot of Various Categories Over Time',
        xaxis=dict(title='Date'),
        yaxis=dict(title='Total Price ($)', tickformat='$,.2f', range=[0, 17]),  # Format y-axis labels as currency and set the range
        showlegend=True
    )

    # Create the figure
    figure = go.Figure(data=traces, layout=layout)

    return figure
########################################


#Reading the dataset from the csv file
sub_df=pd.read_csv("esas.csv")
#Renaming column Names
sub_df=sub_df.rename(columns={'Unnamed: 0':'ID', 'satish_kodu':'Receipt_Number', 'mehsul_kodu':'Product_Code','mehsul_ad':'Product','mehsul_kategoriya':'Category','mehsul_qiymet':'Price','satish_tarixi':'Date','endirim_kompaniya':'Discount','bonus_kart':'Bonus_cart','magaza_ad':'Store_Branch','magaza_lat':'Store_Lat','magaza_long':'Store_Long'})
#droping the na values in df and checking the if they are gone
sub_df=sub_df.dropna()
#dropping unescessay columns
sub_df = sub_df.drop('ID', axis=1)
#reseting index
sub_df = sub_df.reset_index(drop=True)
#checking for duplicate rows and dropping them
duplicated_rows = clean_df.duplicated()
clean_df = clean_df.drop_duplicates()
#Coverting type of some coulmns
sub_df = sub_df.copy()  # Create a copy of the DataFrame
sub_df['Receipt_Number'] = sub_df['Receipt_Number'].astype('int')
sub_df['Product'] = sub_df['Product'].astype('category')
sub_df.loc[:, 'Date'] = pd.to_datetime(sub_df['Date'], dayfirst=True)
```

```
sub_df['Category'] = sub_df['Category'].astype('category')
sub_df['Discount'] = sub_df['Discount'].astype('category')
sub_df['Store_Branch'] = sub_df['Store_Branch'].astype('category')
# Replace multiple values in the 'Discount' column
replace_dict = {'S9rin Yay günlәri': 'Summer Season', 'S?rf?li Yaz': 'Fifth of May
Season', 'B9r9k9tli Novruz': 'Happy Nowruz Season','Yeni il fürs9tlәri': 'New Year
Season','Payız endiriml9ri': 'Autumn Season'}
sub_df['Discount'] = sub_df['Discount'].replace(replace_dict)
# Translate values in the 'Category' and 'Store Branch' column
from googletrans import Translator
translator = Translator()
sub_df['Category'] = sub_df['Category'].apply(lambda x: translator.translate(x).text)
sub_df['Store_Branch'] = sub_df['Store_Branch'].apply(lambda x:
translator.translate(x).text)
```

```
from sklearn.preprocessing import StandardScaler
# Assuming 'clean_df' is your DataFrame
scaler = StandardScaler()
numerical_data = sub_df.select_dtypes(include=[np.number])  # Select numerical columns
only
standardized_data = scaler.fit_transform(numerical_data)
correlation_matrix = pd.DataFrame(standardized_data).corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f")
```

```
from sklearn.decomposition import PCA
pca = PCA(svd_solver="full", n_components=0.95, random_state=5764)
pca_data = pca.fit_transform(standardized_data)
exp_var_pca = pca.explained_variance_ratio_
pca_data_df = pd.DataFrame(pca_data, columns=[f'PC_{i}' for i in range(1,
pca_data.shape[1] + 1)])
cum_sum_eigenvalues = np.cumsum(exp_var_pca) * 100
index_95_variance = np.argmax(cum_sum_eigenvalues >= 95)
```

```
Q1_price = clean_df['Price'].quantile(0.25)
Q3_price = clean_df['Price'].quantile(0.75)
IQR_price = Q3_price - Q1_price
```

```
# Identify outliers for 'Price'
lower_bound_price = Q1_price - 1.5 * IQR_price
upper_bound_price = Q3_price + 1.5 * IQR_price
# Remove outliers from the dataset
cleaned_df_price = clean_df[(clean_df['Price'] >= lower_bound_price) &
(clean_df['Price'] <= upper_bound_price)]

from scipy.stats import kstest
def ks_test(x, title):
    mean = np.mean(x)
    std = np.std(x)
    dist = np.random.normal(mean, std, len(x))
    stats, p = kstest(x, dist)
    return stats,p

from scipy.stats import shapiro
def shapiro_test(x):
    stats, p = shapiro(x)
    return stats,p

scaler = StandardScaler()
standard_Price = scaler.fit_transform(clean_df[['Price']])

from scipy.stats import normaltest
def da_k_squared_test(x):
    stats, p = normaltest(x)
    return stats, p


# Define app layout
plot7_layout = html.Div(children=[
    html.H2('Analysis Dashboard'),

    html.H6("Choose what kind of plot you want to see: "),

    # Dropdown for selecting analysis type
    dcc.Dropdown(
        id='analysis-dropdown',
        options=[
            {'label': 'Correlation Heatmap', 'value': 'correlation'},
            {'label': 'PCA Analysis', 'value': 'pca'},
```

```
            {'label': 'Boxplot of Price', 'value': 'boxplot_price'},
            {'label': 'Boxplot of Cleaned Price', 'value': 'boxplot_cleaned_price'},
            {'label': 'K-S Test for Price', 'value': 'ks_test_price'},
            {'label': 'Shapiro-Wilk Test for Price', 'value': 'shapiro_test_price'},
            {'label': 'D Agostino\'s K^2 test for Price', 'value':
'da_k_squared_test_price'},
        ],
        value='correlation',  # Default value
        style={'width': '50%'}
    ),

    # Placeholder for the selected analysis plot
    dcc.Graph(id='selected-analysis-plot'),

])



# PCA Analysis
def generate_pca_analysis():
    # Assuming 'cum_sum_eigenvalues' and 'index_95_variance' are available
    x_values = list(range(1, len(cum_sum_eigenvalues) + 1))

    # Create the plot
    fig = go.Figure()

    # Cumulative Explained Variance
    fig.add_trace(go.Scatter(
        x=x_values,
        y=cum_sum_eigenvalues,
        mode='lines+markers',
        name='Cumulative Explained Variance',
        line=dict(color='blue', width=2)
    ))

    # 95% Explained Variance Line
    fig.add_shape(
        go.layout.Shape(
            type='line',
            x0=min(x_values),
            x1=max(x_values),
```

```
            y0=95,
            y1=95,
            line=dict(color='black', dash='dash'),
            name='95% Explained Variance'
        )
    )

    # Optimal Features Line
    fig.add_shape(
        go.layout.Shape(
            type='line',
            x0=index_95_variance + 1,
            x1=index_95_variance + 1,
            y0=min(cum_sum_eigenvalues),
            y1=max(cum_sum_eigenvalues),
            line=dict(color='red', dash='dash'),
            name=f'Optimal Features: {index_95_variance + 1}'
        )
    )

    # Layout settings
    fig.update_layout(
        title='Cumulative Explained Variance vs. Number of Components',
        xaxis=dict(title='Number of Components'),
        yaxis=dict(title='Cumulative Explained Variance (%)'),
        legend=dict(x=0.7, y=0.9),
        height=600,
        width=1000
    )

    return fig


# Boxplot of 'Price'
def generate_boxplot_price():
    fig = px.box(sub_df['Price'], x='Price', orientation='h',title='Boxplot of Price
with Outliers')
    return fig


# Correlation Heatmap
```

```
def generate_correlation_heatmap():
    # Round the correlation matrix values to 2 decimal points
    correlation_matrix_rounded = correlation_matrix.round(2)

    # Create a list of lists to display values in annotations
    z_text = [['' if np.isnan(value) else f'{value:.2f}' for value in row] for row in
correlation_matrix_rounded.values]

    # Create the heatmap with custom colorscale
    fig = px.imshow(correlation_matrix_rounded, labels=dict(x="Features", y="Features",
color="Correlation"),
                    color_continuous_scale='Viridis')

    # Add text annotations
    fig.update_layout(annotations=[
        dict(x=i, y=j, text=z_text[i][j], showarrow=False) for i in
range(len(correlation_matrix_rounded.columns))
        for j in range(len(correlation_matrix_rounded.columns))
    ])

    # Set layout title
    fig.update_layout(title='Correlation Coefficient Matrix Heatmap')

    return fig


# K-S Test for 'Price'
def generate_ks_test_price():
    # Assuming 'sub_df' is your DataFrame
    stats, p = ks_test(sub_df['Price'], 'Raw')

    # Display K-S test results for 'Price'
    result_text = "Not Normal" if p < 0.01 else "Normal"

    # Create a table with results
    fig = go.Figure(data=[go.Table(
        header=dict(values=['Statistic', 'P-Value', 'Result']),
        cells=dict(values=[stats.round(2), p.round(2), result_text])
    )])

    # Set layout title
```

```
    fig.update_layout(title='K-S Test for Price')

    return fig


# Shapiro-Wilk Test for 'Price'
def generate_shapiro_test_price():
    # Assuming 'cleaned_df_price' is your cleaned DataFrame for 'Price'
    stat_price, p_value_price = shapiro_test(sub_df['Price'])

    # Convert 'stat_price' and 'p_value_price' to rounded strings
    stat_price_str = f"{stat_price:.2f}" if isinstance(stat_price, (int, float)) else
""
    p_value_price_str = f"{p_value_price:.2f}" if isinstance(p_value_price, (int,
float)) else ""

    # Interpret the Shapiro-Wilk test result for 'Price'
    result_price = "Not Normal" if p_value_price < 0.01 else "Normal"

    # Create a table with results
    fig = go.Figure(data=[go.Table(
        header=dict(values=['Statistic', 'P-Value', 'Result']),
        cells=dict(values=[stat_price_str, p_value_price_str, result_price])
    )])

    # Set layout title
    fig.update_layout(title='Shapiro-Wilk Test for Price')

    return fig




def da_k_squared_test_price():

    stats, p = da_k_squared_test(sub_df['Price'])

    # Interpret the Shapiro-Wilk test result for standardized 'Price'
    result_std_price = "Not Normal" if p < 0.01 else "Normal"

    # Create a table with results
```

```
    fig = go.Figure(data=[go.Table(
        header=dict(values=['Statistic', 'P-Value', 'Result']),
        cells=dict(values=[stats.round(2), p.round(2), result_std_price])
    )])

    # Set layout title
    fig.update_layout(title=' D Agostino\'s K^2 test for Price')

    return fig


def generate_boxplot_cleaned_price():
    fig = px.box(cleaned_df_price, x='Price', orientation='h', title='Boxplot of
Cleaned Price')
    return fig

# Callback to update the selected analysis plot based on the dropdown value
@Final_Term_Project_app.callback(
    Output('selected-analysis-plot', 'figure'),
    [Input('analysis-dropdown', 'value')]
)
def update_analysis_plot(selected_analysis):
    if selected_analysis == 'correlation':
        return generate_correlation_heatmap()
    elif selected_analysis == 'pca':
        return generate_pca_analysis()
    elif selected_analysis == 'ks_test_price':
        return generate_ks_test_price()
    elif selected_analysis == 'boxplot_price':
        return generate_boxplot_price()
    elif selected_analysis == 'boxplot_cleaned_price':
        return generate_boxplot_cleaned_price()
    elif selected_analysis == 'shapiro_test_price':
        return generate_shapiro_test_price()
    elif selected_analysis == 'da_k_squared_test_price':
        return da_k_squared_test_price()
    else:
        return go.Figure()


########################################
```

```
# Main callback
@Final_Term_Project_app.callback(
    Output("tabs-content", "children"),
    [Input("tabs", "value")]
)
def render_content(tab):
    if tab == "about":
        return about_layout
    elif tab == "p1":
        return plot1_layout
    elif tab == "p2":
        return plot2_layout
    elif tab == "p3":
        return plot3_layout
    elif tab == "p4":
        return plot4_layout
    elif tab == "p5":
        return plot5_layout
    elif tab == "p6":
        return plot6_layout
    elif tab == "p7":
        return plot7_layout



if __name__ == '__main__':
    Final_Term_Project_app.run_server(debug=False, host='0.0.0.0', port=8051)
```

References:

1. https://www.kaggle.com/datasets/mexwell/supermarket-dataset
2. https://github.com/rjafari979/Information-Visualization-Data-Analytics-Dataset-
3. https://canvas.vt.edu/courses/176323/files/30644261?module_item_id=2647191
4. https://dash.plotly.com/tutorial
5. https://dash.plotly.com/dash-core-components
6. https://www.kaggle.com/code/caesarmario/data-pre-processing-eda-feature-engineering
7. https://towardsdatascience.com/feature-engineering-and-data-preparation-using-supermarket-sales-data-part-2-171b7a7a7eb7