VIRGINIA TECH

INFORMATION VISUALIZATION PROJECT

E-COMMERCE DATA ANALYSIS

CS5764 – FINAL TERM PROJECT

Link: https://dashapp-45qc52lfqa-nn.a.run.app/

Swethaa Shanmugam Ramesh

ABSTRACT

The "E-commerce Customer Behavior and Purchase Dataset" is a synthetic yet sophisticated representation of online retail activity, created with the Faker Python library. It simulates a detailed e-commerce landscape, capturing diverse aspects of customer behavior and transactional data. The dataset is strategically assembled to support intricate data analysis and predictive modeling, essential for the e-commerce sector. It incorporates a wide spectrum of variables such as Customer ID, Name, Age, Gender, Purchase Date, Product Category, Price, Quantity, Total Purchase Amount, Payment Method, Returns, and Churn indicator. These variables are designed to provide a granular view of customer engagement and purchasing trends, equipping analysts with the data necessary for predicting customer churn, conducting market basket analysis, enhancing recommendation systems, and identifying sales patterns. Enhanced by an interactive analytics dashboard built with Dash, this dataset presents multifaceted insights through various lenses like Customer Insights, Product Analysis, and Geographic Distribution. The dashboard elevates the dataset's utility, providing dynamic visualizations and custom reports that drive strategic decision-making. This fusion of synthetic data and practical analysis tools offers a potent combination for e-commerce research and real-world business strategy development.

CONTENTS

List Of Figures

List Of Tables

1)INTRODUCTION:

Overview:

The static plots have been created followed by the dashboard and tables.

The link to the deployed app is given above the report.

Description of the dataset:

The "E-commerce Customer Behavior and Purchase Dataset" is a synthetic dataset created using the Faker Python library, meticulously crafted to simulate the dynamics of customer interactions and purchase behavior in an e-commerce setting. This dataset is a goldmine for data analysis and predictive modeling, offering a fertile ground for a variety of e-commerce related analytical tasks.

Dataset Description and Suitability:

Comprehensiveness: It captures a wide array of variables, from basic customer demographics to detailed transactional data, making it ideal for in-depth e-commerce analysis.

Diversity of Data: With attributes like customer demographics, purchase history, and payment methods, the dataset provides a multifaceted view of customer behavior.

Synthetic Nature: Being generated by Faker, it is free from privacy concerns, making it suitable for public usage, educational purposes, and experimentation without ethical or legal implications.

Dependent and Independent Variables:

Dependent Variable: The 'Churn' column is an exemplary choice for the dependent variable, especially for predictive modeling tasks like churn prediction, which is vital for understanding customer retention.

Independent Variables: Variables such as 'Customer Age', 'Gender', 'Purchase Date', 'Product Category', 'Product Price', 'Quantity', 'Total Purchase Amount', and 'Payment Method' act as independent variables. These offer insights into customer profiles, purchasing patterns, and preferences, which are crucial for predictive modeling.

Importance in Industry:

1.Customer Churn Prediction: Understanding factors leading to customer churn is critical for businesses to devise strategies for retention.

2.Market Basket Analysis: The dataset supports analysis of product categories and purchasing patterns, aiding in cross-selling and upselling strategies.

3.Recommendation Systems: Data on customer preferences and purchase history can be used to develop personalized recommendation systems, enhancing customer experience and sales.

4.Trend Analysis: Analysis of purchase dates and categories helps in identifying market trends, and guiding inventory and marketing strategies.

5.Payment Method Analysis: Understanding preferred payment methods can optimize payment gateway integrations and improve customer convenience.

a) Pre-processing dataset

```
First few rows before cleaning:
   Customer ID        Purchase Date Product Category  ...  Age  Gender  Churn
0        44605  2023-05-03 21:30:02             Home  ...   31  Female      0
1        44605  2021-05-16 13:57:44      Electronics  ...   31  Female      0
2        44605  2020-07-13 06:16:57            Books  ...   31  Female      0
3        44605  2023-01-17 13:14:36      Electronics  ...   31  Female      0
4        44605  2021-05-01 11:29:27            Books  ...   31  Female      0

[5 rows x 13 columns]
```

```
First few rows after cleaning:
   Customer ID        Purchase Date Product Category  ...  Age  Gender  Churn
0        44605  2023-05-03 21:30:02             Home  ...   31  Female      0
1        44605  2021-05-16 13:57:44      Electronics  ...   31  Female      0
2        44605  2020-07-13 06:16:57            Books  ...   31  Female      0
3        44605  2023-01-17 13:14:36      Electronics  ...   31  Female      0
4        44605  2021-05-01 11:29:27            Books  ...   31  Female      0
```

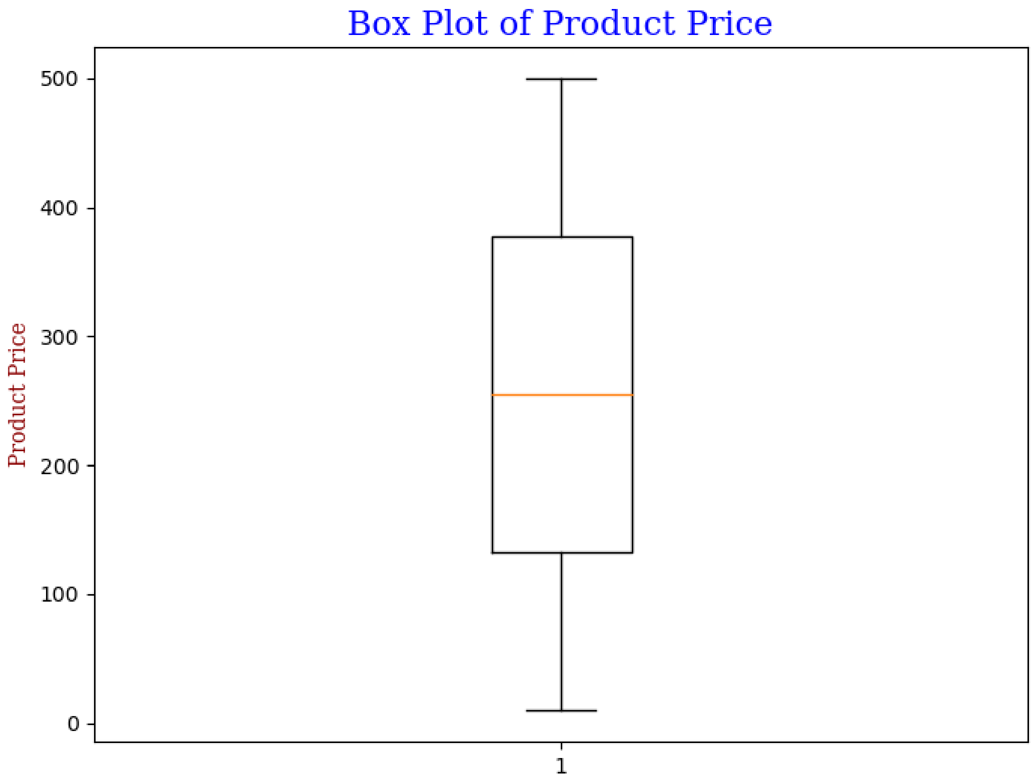
The cleaning process involves two main steps. First, for **numerical columns** like 'Product Price,' 'Quantity,' 'Total Purchase Amount,' 'Customer Age,' and 'Age,' missing values are identified and replaced with the **median** value of the respective column. This choice of imputation is robust and less sensitive to outliers. Second, for **categorical columns** such as 'Customer ID,' 'Purchase Date,' 'Product Category,' 'Payment Method,' 'Customer Name,' 'Gender,' 'Returns,' and 'Churn,' missing values are detected and filled with the **mode** (most frequently occurring value) of each column. This approach maintains the distribution of categorical data. After cleaning, the code prints the first few rows of the cleaned dataset to show the results and provides summary statistics to describe the cleaned data. These methods help ensure that the dataset is prepared for analysis and modeling by addressing missing data effectively.

Statistics of the cleaned dataset:

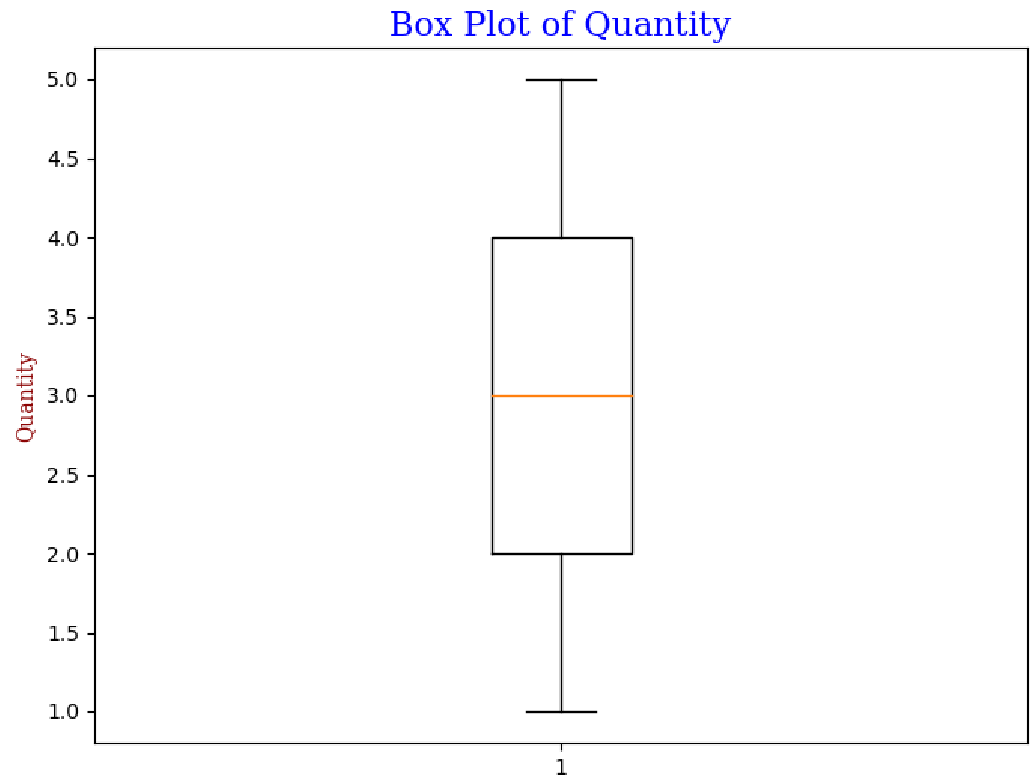
```
Statistics of the cleaned dataset:
       Customer ID  Product Price   Quantity  ...    Returns        Age     Churn
count    250000.00      250000.00  250000.00  ...  250000.00  250000.00  250000.0
mean      25017.63         254.74       3.00  ...       0.60      43.80       0.2
std       14412.52         141.74       1.41  ...       0.49      15.36       0.4
min           1.00          10.00       1.00  ...       0.00      18.00       0.0
25%       12590.00         132.00       2.00  ...       0.00      30.00       0.0
50%       25011.00         255.00       3.00  ...       1.00      44.00       0.0
75%       37441.25         377.00       4.00  ...       1.00      57.00       0.0
max       50000.00         500.00       5.00  ...       1.00      70.00       1.0
```

b) Outlier detection & removal:

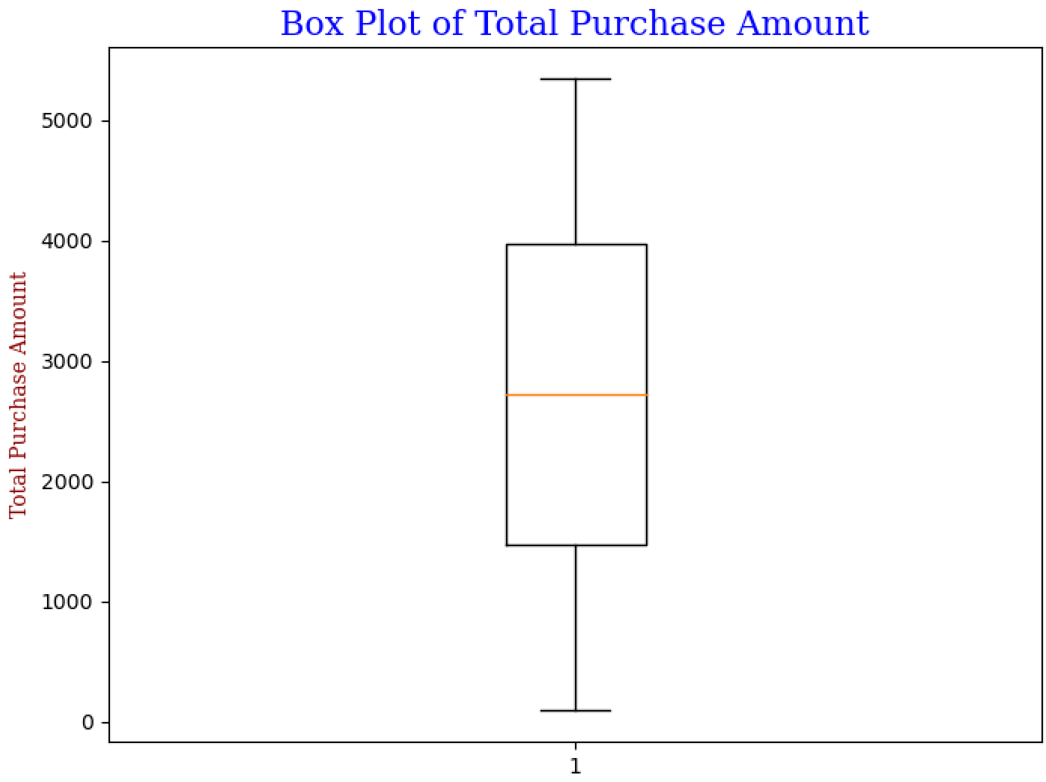
After applying **IQR's** method for outlier detection and removal to the specified numerical columns, it is observed that the dataset size has been reduced, indicating that outliers were successfully filtered out. The box plots generated for each numerical column reveal that the outliers have been effectively removed, resulting in a more tightly clustered distribution. This process has likely contributed to a dataset that is less skewed by extreme values, making it suitable for more robust statistical analyses and machine learning modeling, as extreme outliers can distort results.

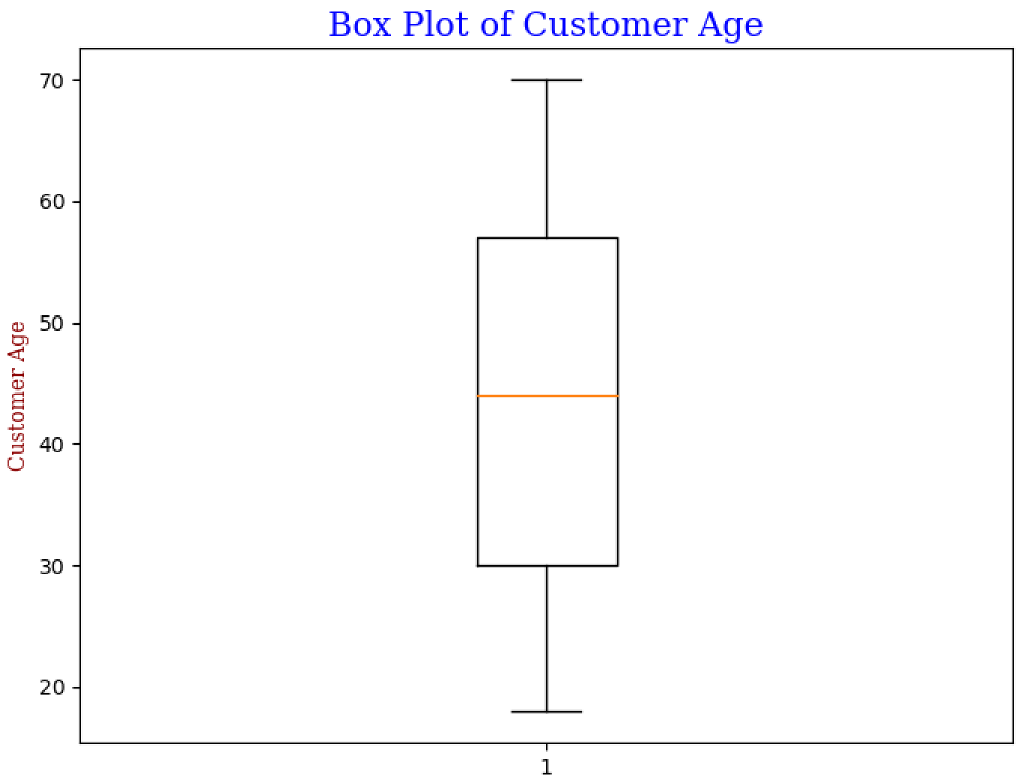


Box Plot of Product Price
Product Price
500
400
300
200
100
0
1



Box Plot of Quantity
Quantity
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1


Box Plot of Total Purchase Amount
Total Purchase Amount
5000
4000
3000
2000
1000
0
1


Box Plot of Customer Age
Customer Age
70
60
50
40
30
20
1

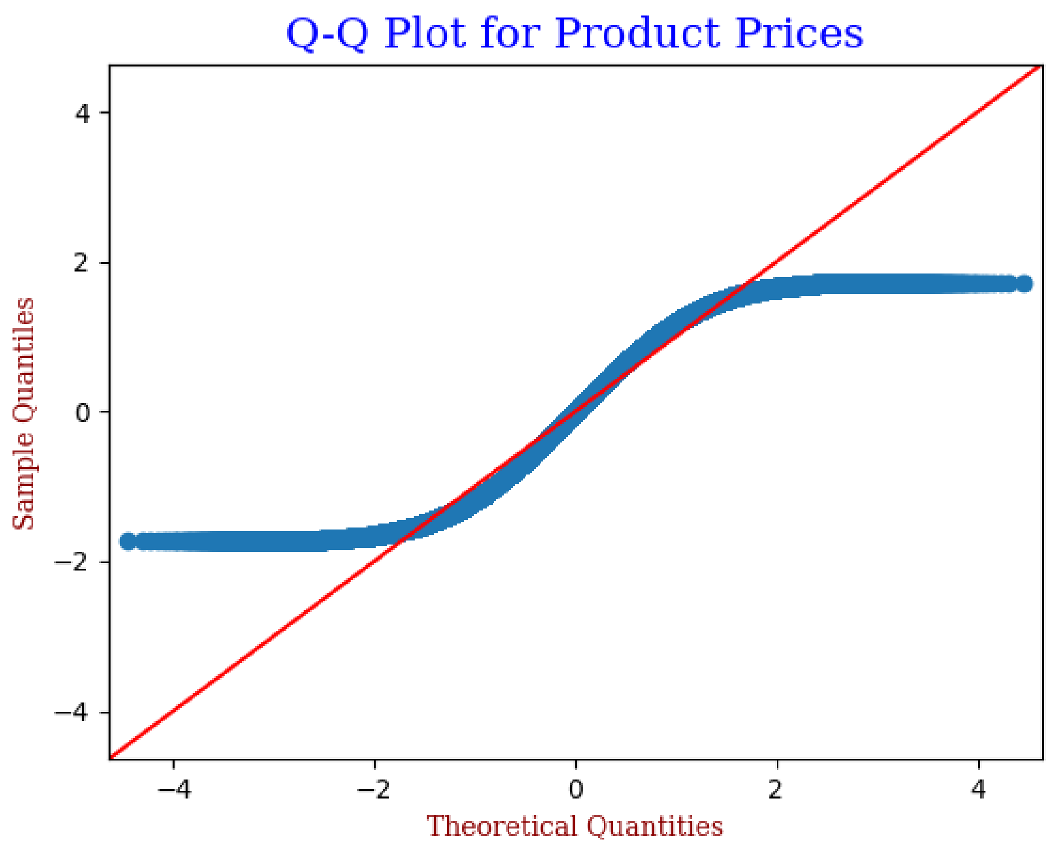
c) Principal Component Analysis (PCA):

```
Explained Variance Ratio: [0.25 0.13 0.13 0.13 0.12 0.12 0.12]
Condition Number: 1.0
Singular Values: [708.24 501.21 500.68 500.11 499.79 498.83 497.78]
```

The results of the Principal Component Analysis (PCA) reveal several key insights. Firstly, the explained variance ratio indicates a significant amount of variance captured by each component, with the first component being the most significant. The condition number being 1.0 suggests that the PCA components are well-conditioned and not highly linearly dependent, enhancing the reliability of the PCA. The closely valued singular values imply that multiple components are important in explaining the dataset's variance. The PCA is executed with n_components=0.95, aiming to retain 95% of the total variance, a common approach for dimensionality reduction while preserving most information.

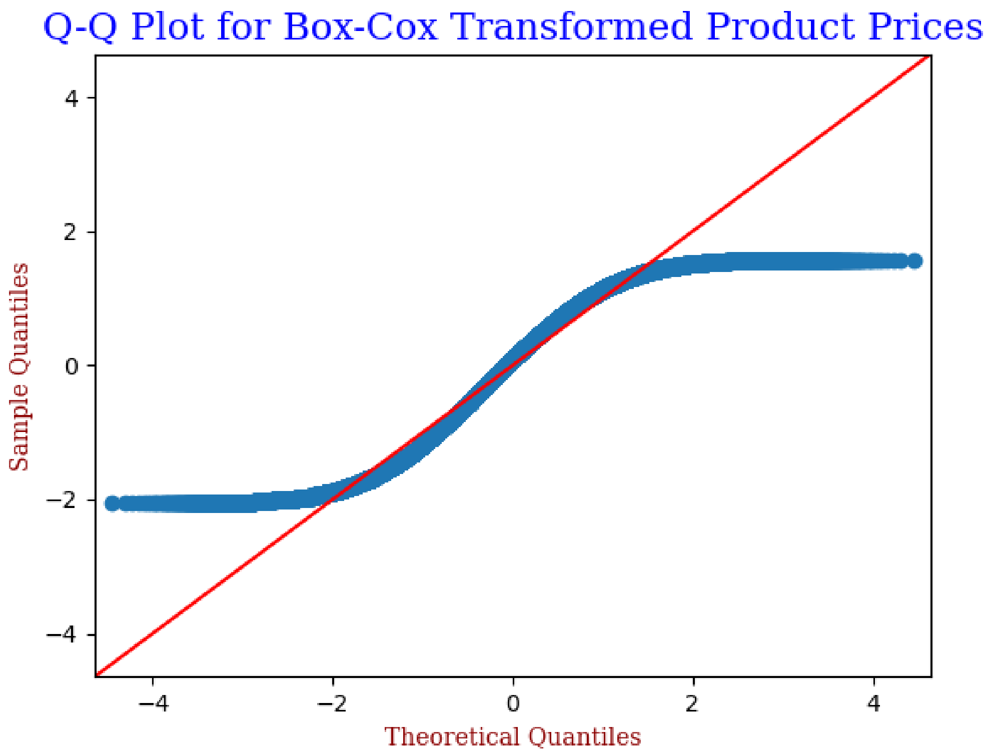
d) Normality test:

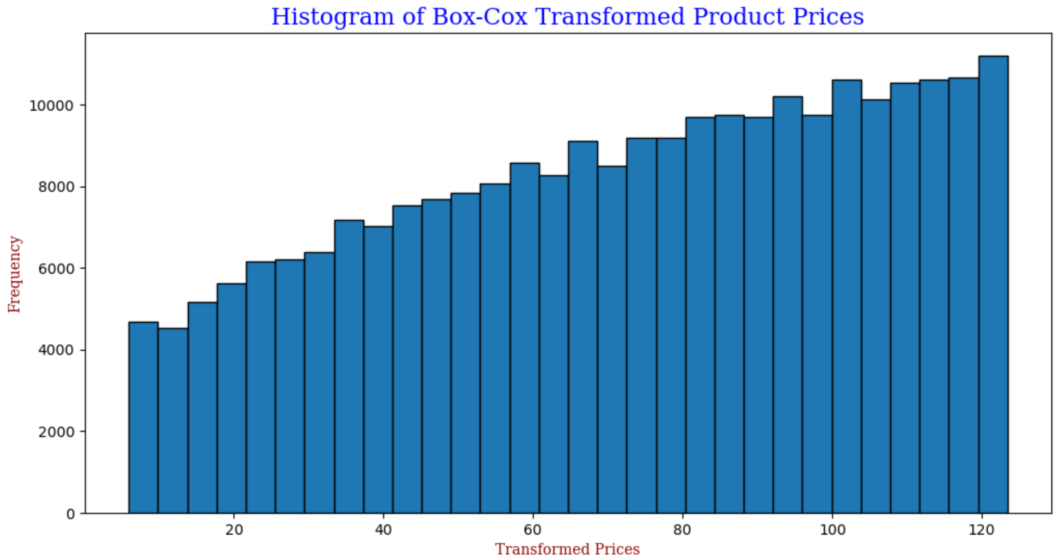
```
D'Agostino's K-squared test statistic: 217306.96
P-value: 0.00
```



The D'Agostino's K-squared test, applied to the 'Product Price' data, yields a test statistic of 217306.96 and a p-value of 0.00. These results strongly suggest that the distribution of product prices significantly **deviates** from a normal distribution. The high test statistic is a clear indicator of non-normality, and the p-value, being effectively zero, decisively rejects the null hypothesis of the test, which states that the data follows a normal distribution. Additionally, the Q-Q plot further visualizes this deviation. In a Q-Q plot, if the data were normally distributed, the points would closely follow the 45-degree reference line. Deviations from this line indicate departures from normality. Therefore, both the statistical test and the Q-Q plot reinforce the conclusion that the product prices do not follow a normal distribution.

e) Data transformation:



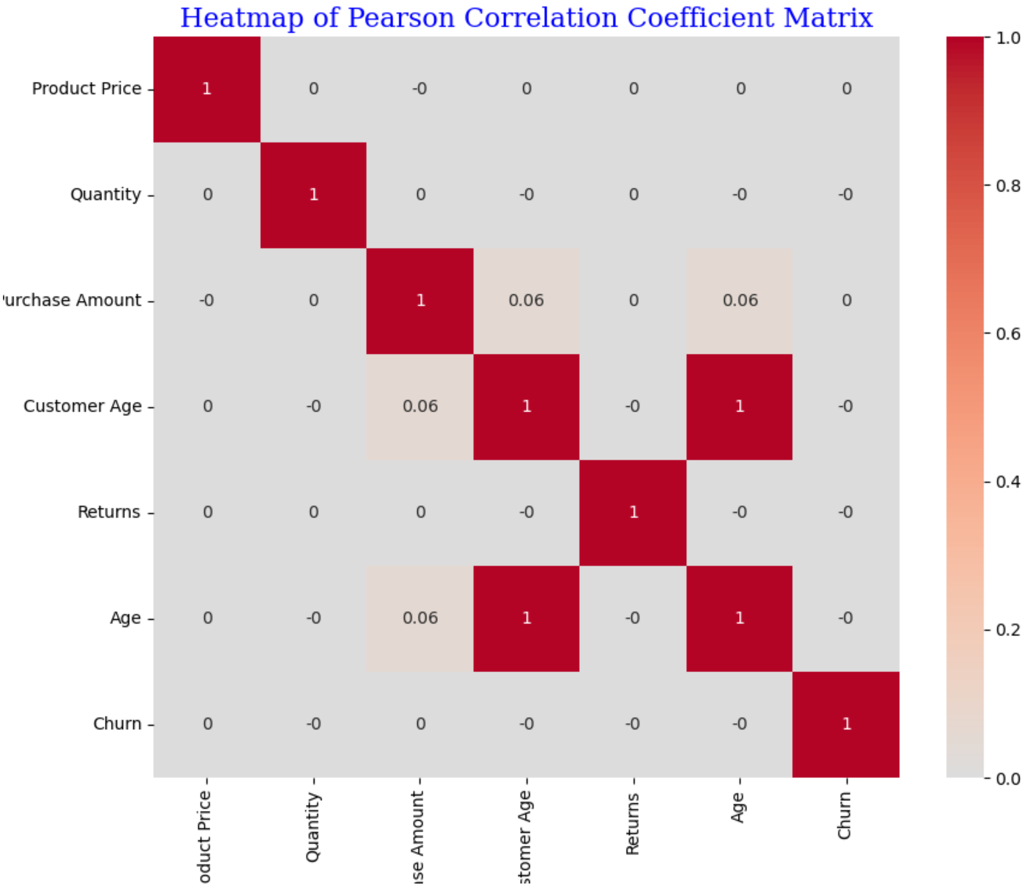


In the Q-Q plot, the closer the data points (blue) are to the reference line (red), the more normal the distribution. From the plots, most of the data points seem to align well with the reference line, especially in the middle quantiles, indicating that the transformation has made the **data more normal**. The histogram shows the frequency distribution of the Box-Cox transformed product prices. A perfectly normal distribution would resemble a bell curve. The histogram provided shows an unimodal distribution that approximates a bell shape but is not perfectly symmetrical. This suggests that while the Box-Cox transformation has made the distribution more normal, it is not perfectly so.

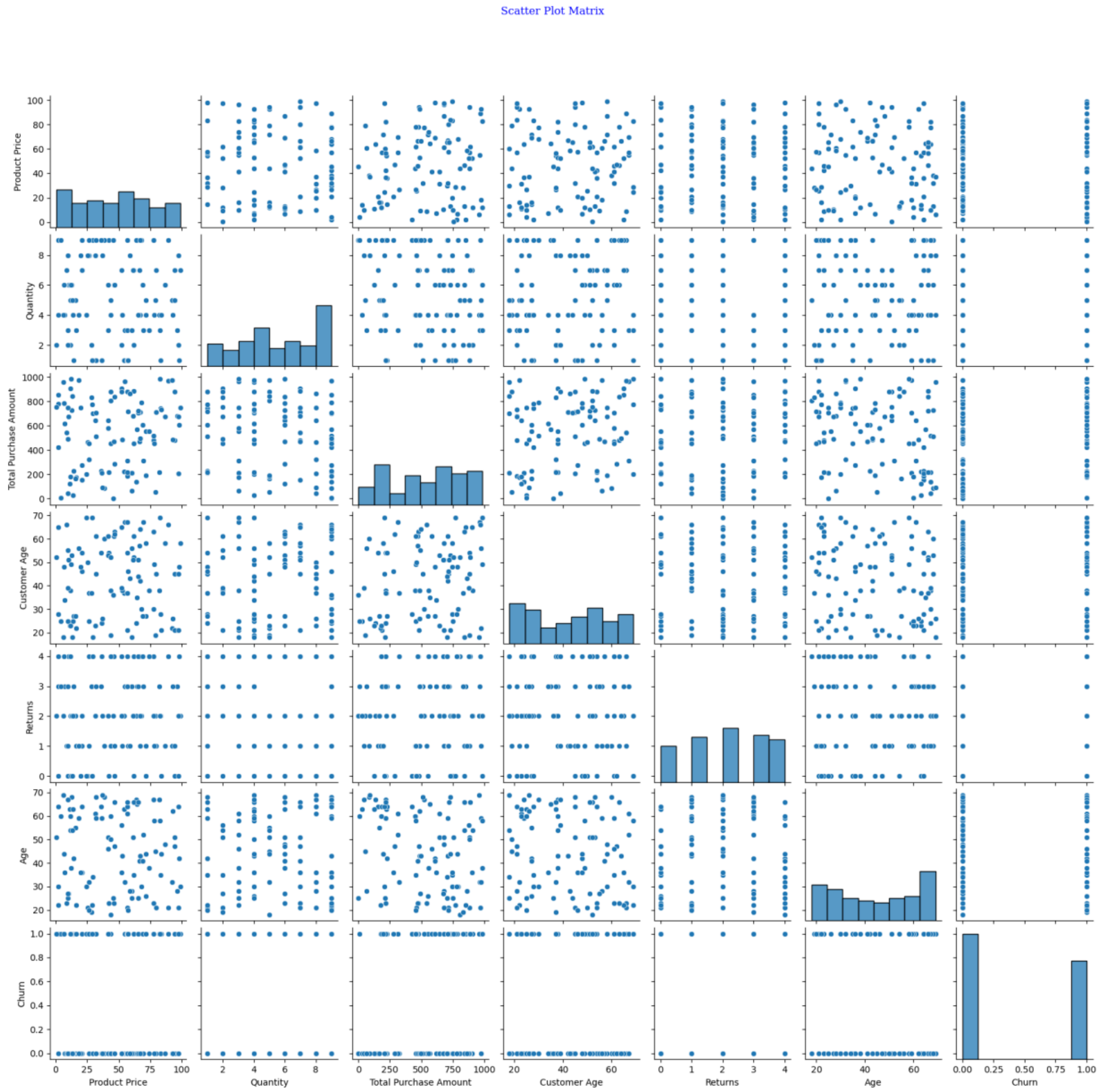
f) Heatmap & Pearson correlation coefficient matrix:

	Product Price	Quantity	...	Age	Churn
Product Price	1.0	0.0	...	0.00	0.0
Quantity	0.0	1.0	...	-0.00	-0.0
Total Purchase Amount	-0.0	0.0	...	0.06	0.0
Customer Age	0.0	-0.0	...	1.00	-0.0
Returns	0.0	0.0	...	-0.00	-0.0
Age	0.0	-0.0	...	1.00	-0.0
Churn	0.0	-0.0	...	-0.00	1.0

This is a correlation matrix of the different features in the dataset.

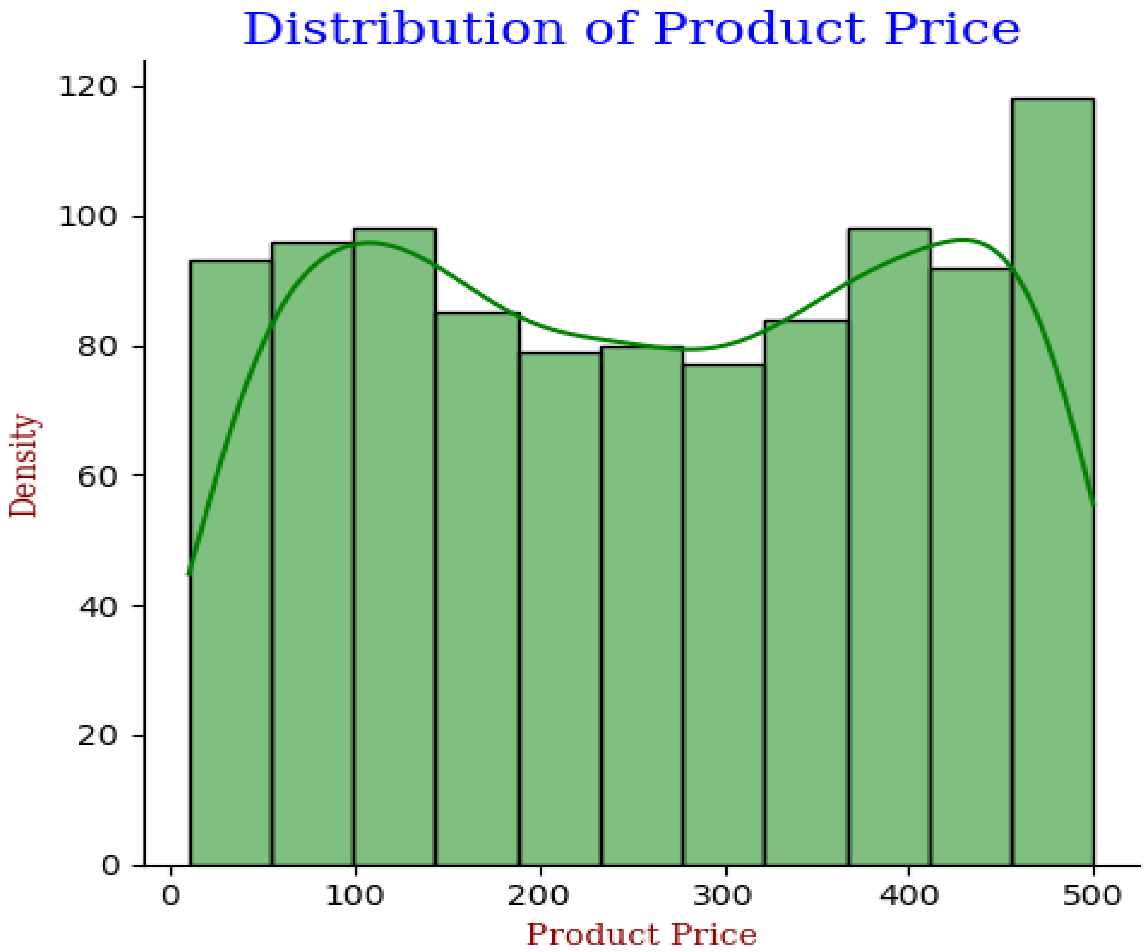


The graph shows a heatmap representing the Pearson correlation coefficient matrix for various variables such as 'Product Price', 'Quantity', 'Total Purchase Amount', 'Customer Age', 'Returns', 'Age', and 'Churn'. The heatmap uses shades of red to illustrate the strength of the correlations between the variables, with darker shades indicating stronger positive correlations and lighter shades representing no correlation (as indicated by 0). The diagonal, naturally, shows a perfect correlation of 1 for each variable with itself, which is standard for correlation matrices. However, there appears to be no significant correlation between the other variables, as suggested by the predominance of light shades and zero values off the diagonal. This implies that there is no linear relationship between these pairs of variables, or the relationship is very weak, within the dataset analyzed.



Here, there seems to be a lack of strong linear relationships as most scatter plots display a diffuse cloud of points without a discernible pattern. For instance, the 'Product Price' and 'Quantity' scatter plot does not indicate any obvious correlation as the points do not form a line or curve of any sort. The histograms on the diagonal indicate that some variables are more evenly distributed (such as 'Customer Age'), while others show a concentration of values in certain ranges (such as 'Quantity', which has bars higher at the lower end, suggesting many low-quantity transactions)

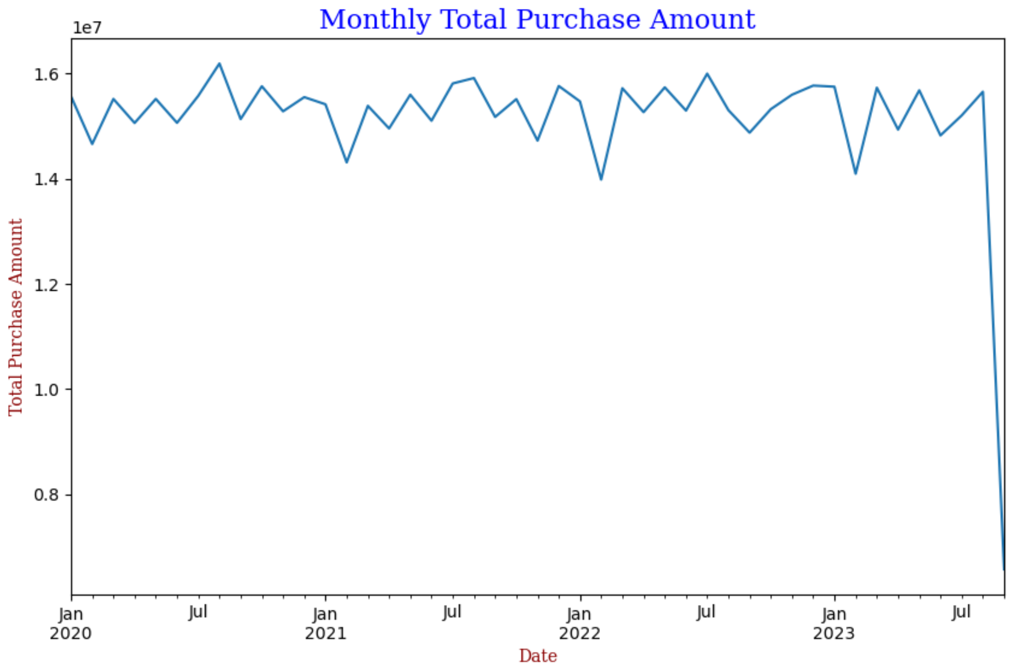
g) Statistics:



This represents the kernel density estimate of the product price.

h) Data Visualization

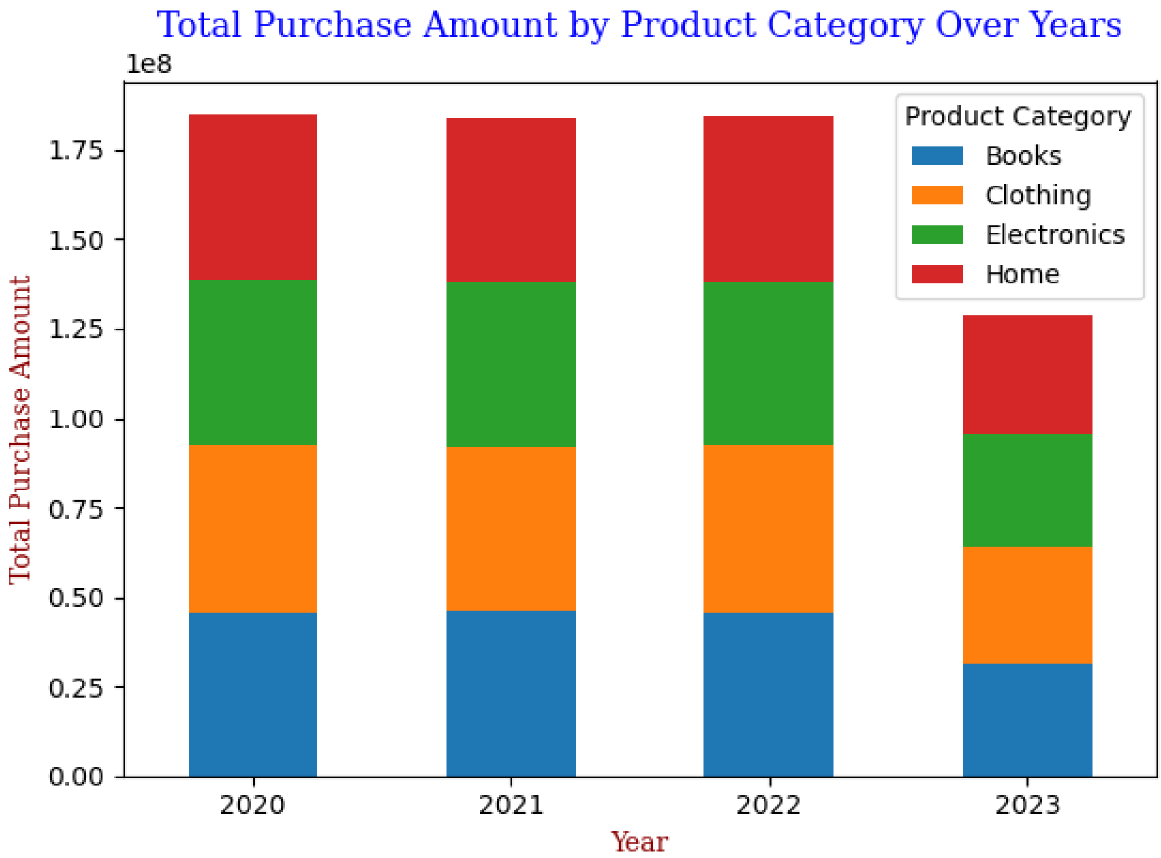
1.Line-plot



Description:

The graph is a line chart depicting the monthly total purchase amounts from January 2020 to July 2023. The vertical axis indicates the purchase amounts in millions, while the horizontal axis represents time. The chart shows fluctuations in the purchase amounts without a clear upward or downward trend. Peaks approach 1.6 million, while troughs fall just below 1.2 million, suggesting variability in monthly purchases.

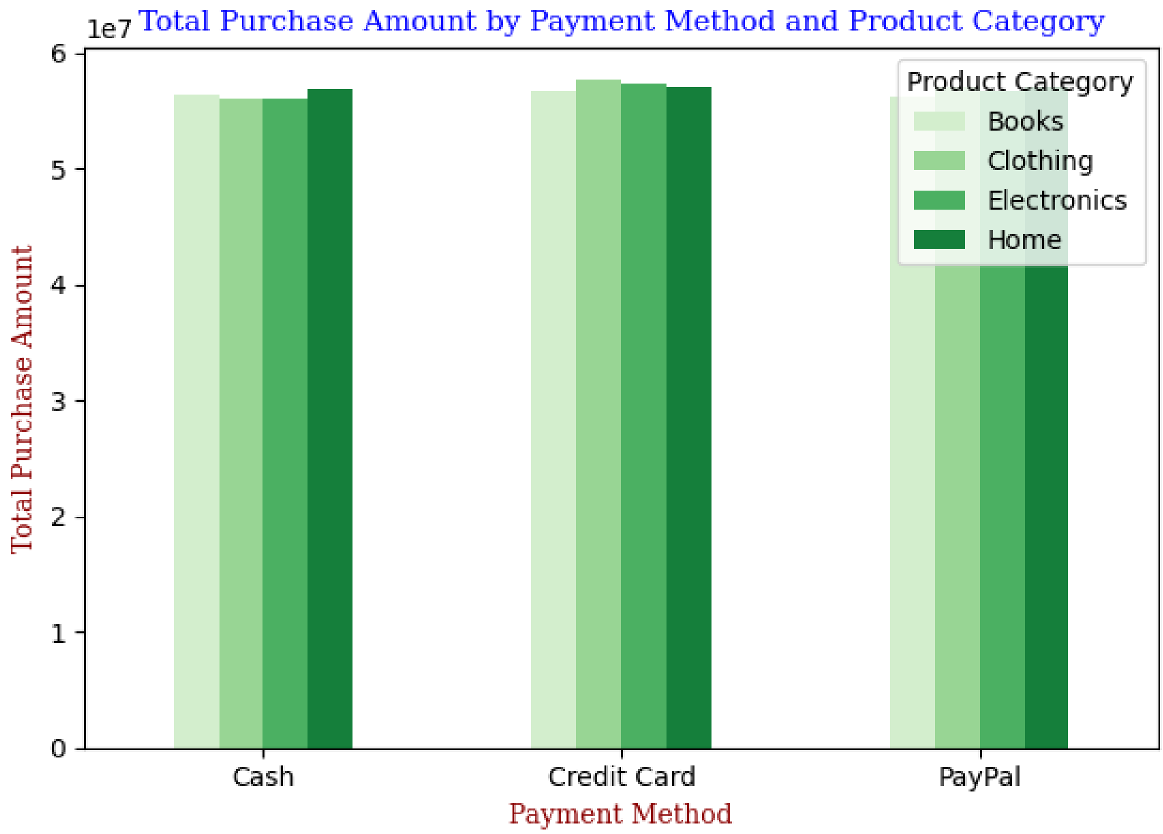
2.Stack Bar Plot



Description:

The graph is a stacked bar chart showing the total purchase amount by product category over four years, from 2020 to 2023. Each bar represents a year, and each segment within a bar represents a different product category: Books, Clothing, Electronics, and Home. The vertical axis is labeled with the total purchase amount, scaling by 10 million (1e7). There seems to be a consistent pattern across the years with Electronics and Home being the largest segments. The chart suggests a relatively stable distribution of purchase amounts across these categories over the years, without significant changes in the proportions.

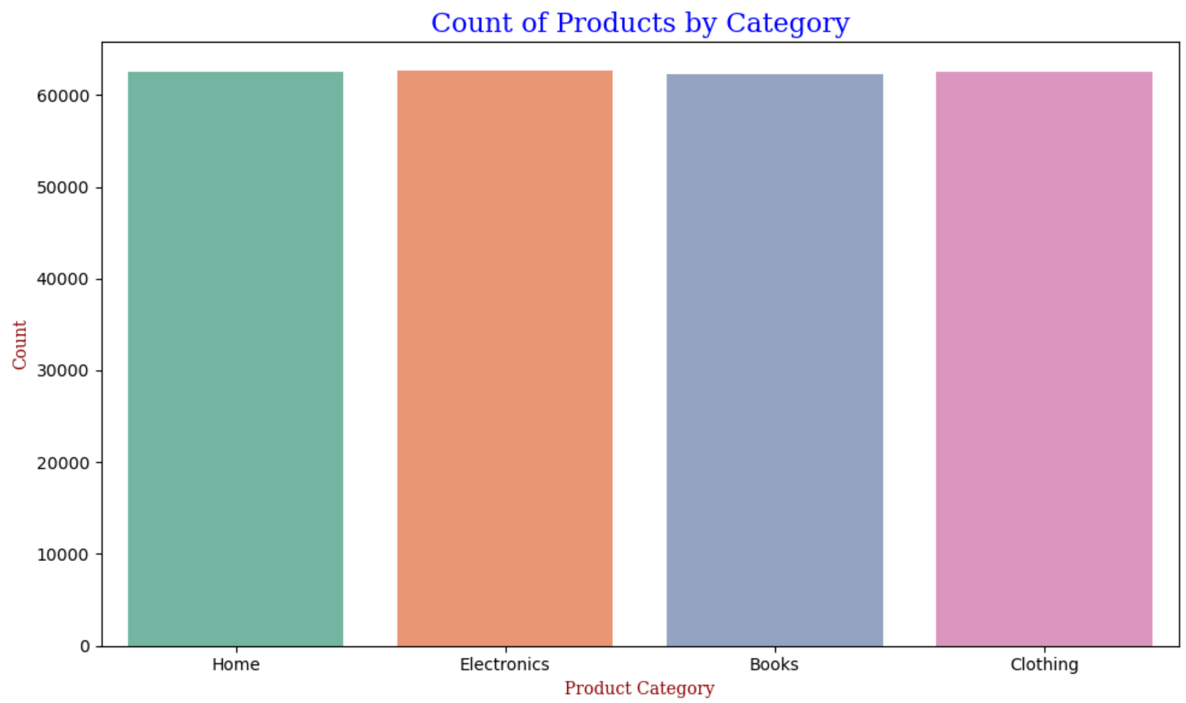
3.Group bar plot



Description:

The graph displays a grouped bar chart detailing the total purchase amount categorized by payment method and product category. The payment methods are Cash, Credit Card, and PayPal, and the product categories include Books, Clothing, Electronics, and Home. Each payment method has four bars adjacent to it, representing the total purchase amounts for each product category. The vertical axis indicates the purchase amount, marked in increments of 1 million (1e6). The chart shows that purchases across all categories are quite similar for each payment method, with Electronics and Home appearing to be the most frequently purchased items regardless of the payment method.

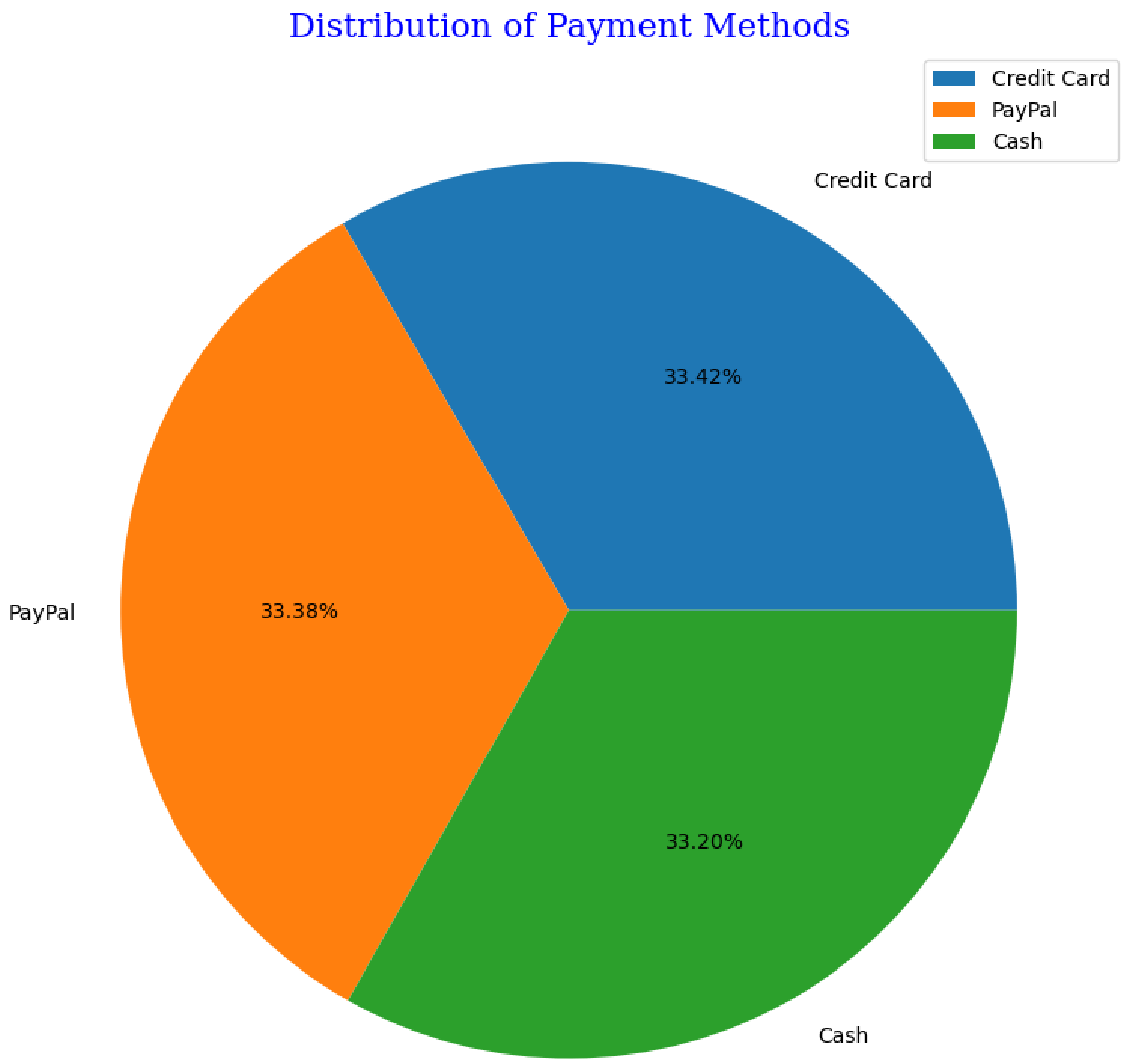
4.Count plot



Description:

The plot shows a count plot titled "Count of Products by Category," which compares the number of products across four different categories: Home, Electronics, Books, and Clothing. Each category is represented by a colored bar corresponding to the count of products within that category. The vertical axis represents the count of products, with a range from 0 to over 6000. All categories have a similar number of products, with counts around the 6000 mark, suggesting a relatively even distribution of product quantities across these categories.

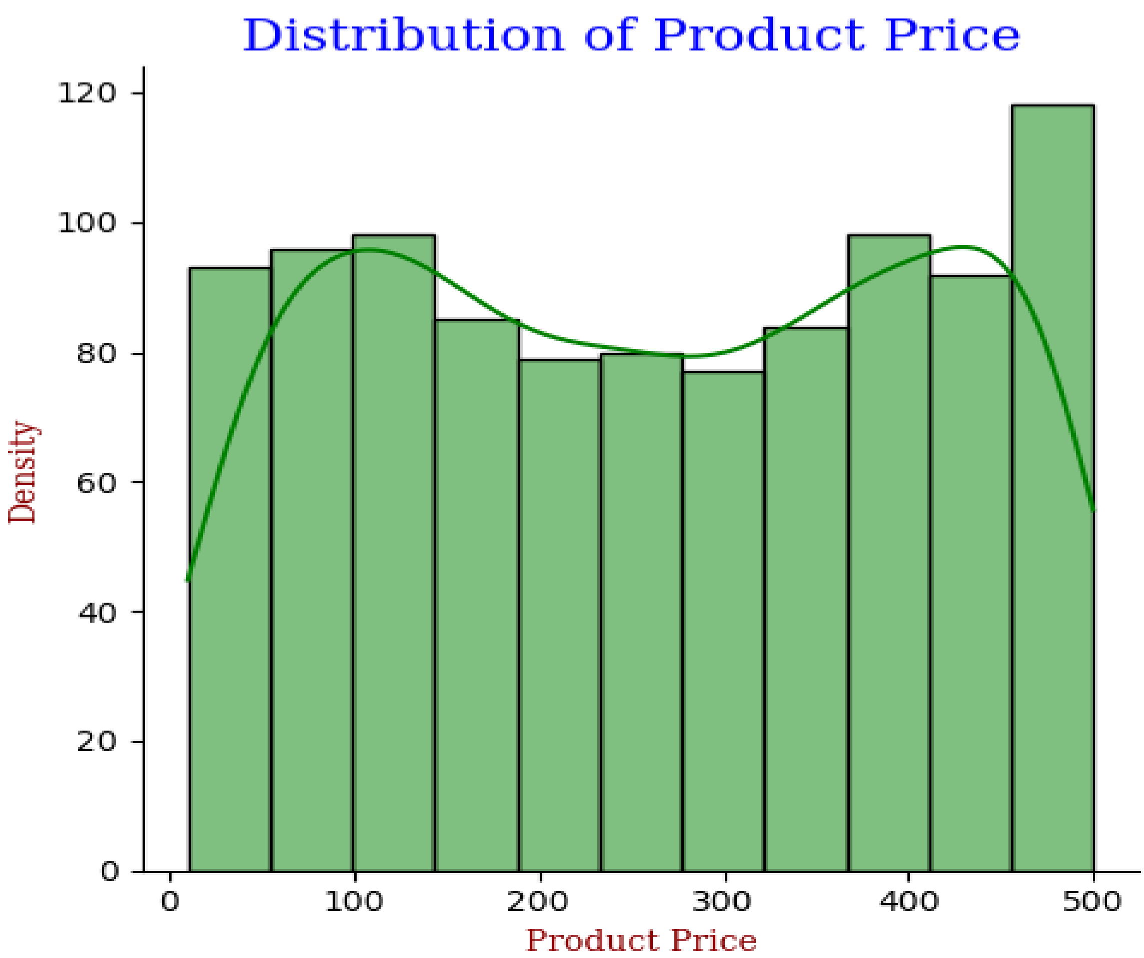
5. Pie-chart



Description:

The graph depicts a pie chart titled "Distribution of Payment Methods," showing the proportion of transactions made using different payment methods. There are three segments: Cash (orange), Credit Card (blue), and PayPal (green). The Credit Card segment is the largest, accounting for 33.72% of the transactions, followed very closely by Cash at 33.40%, and PayPal at 32.88%. The percentages are very close to each other, indicating a nearly equal preference for each payment method among the transactions analyzed. This chart gives a clear visual representation of how evenly distributed the use of these payment methods is.

6.Dist plot



Description:

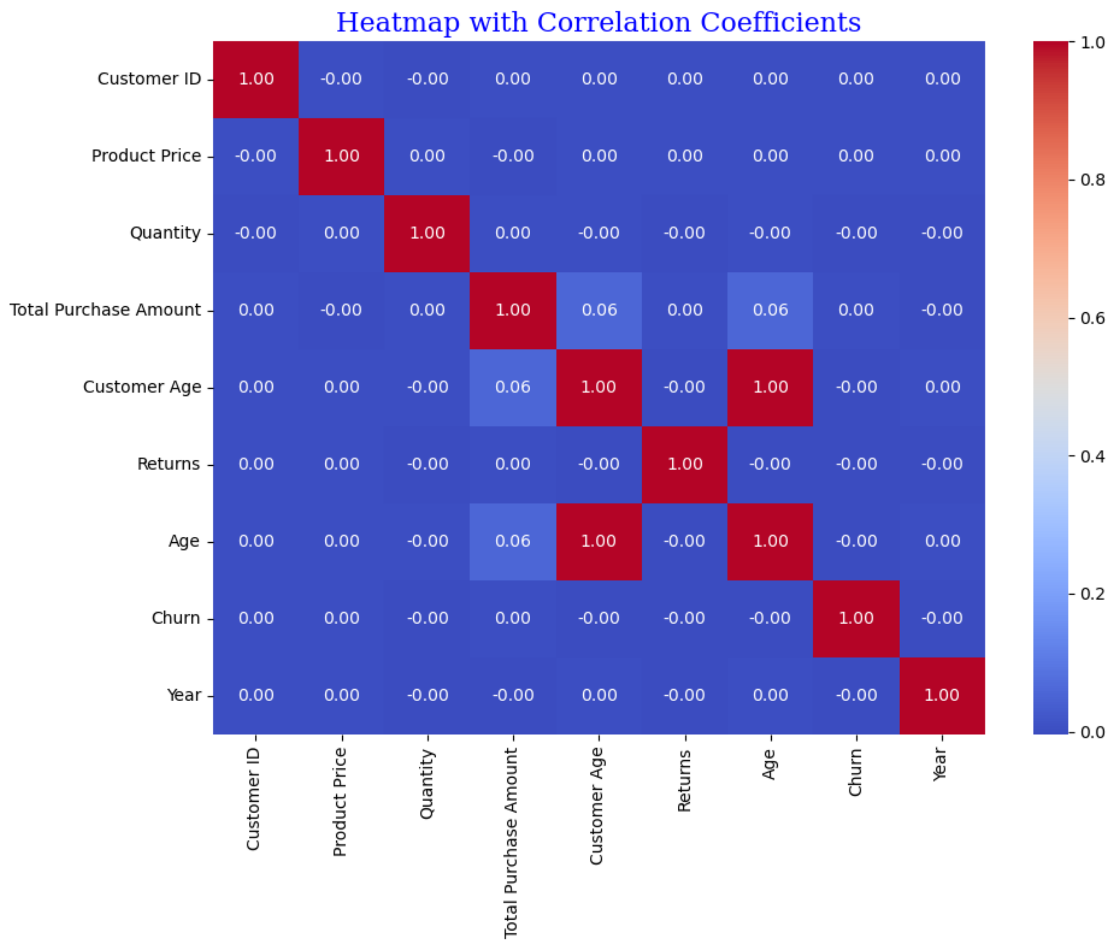
The graph shows a histogram overlaid with a kernel density estimate (KDE) curve, depicting the distribution of 'Product Price'. The histogram's bars represent the frequency of different price ranges, showing how many products fall within each price bin. The KDE curve is a smoothed version of the histogram and gives an estimate of the probability density function of the variable 'Product Price'. From the plot, we can observe that the distribution has a peak around 0 and another larger peak towards the higher price range near 500, suggesting a bimodal distribution where there are significant numbers of products at both the lower and higher ends of the price spectrum.

7.Pair plot

Description:

The graph displays a pair plot, a grid of graphs used to examine the relationships between multiple variables in a dataset. Each row and column represent different variables, with histograms on the diagonal showing the distribution of individual variables, and scatter plots filling the off-diagonal spaces to reveal potential correlations between pairs. The color coding likely differentiates categories within the data, such as customer churn, helping to visualize how these categories vary across the different variables.

8.Heatmap with cbar

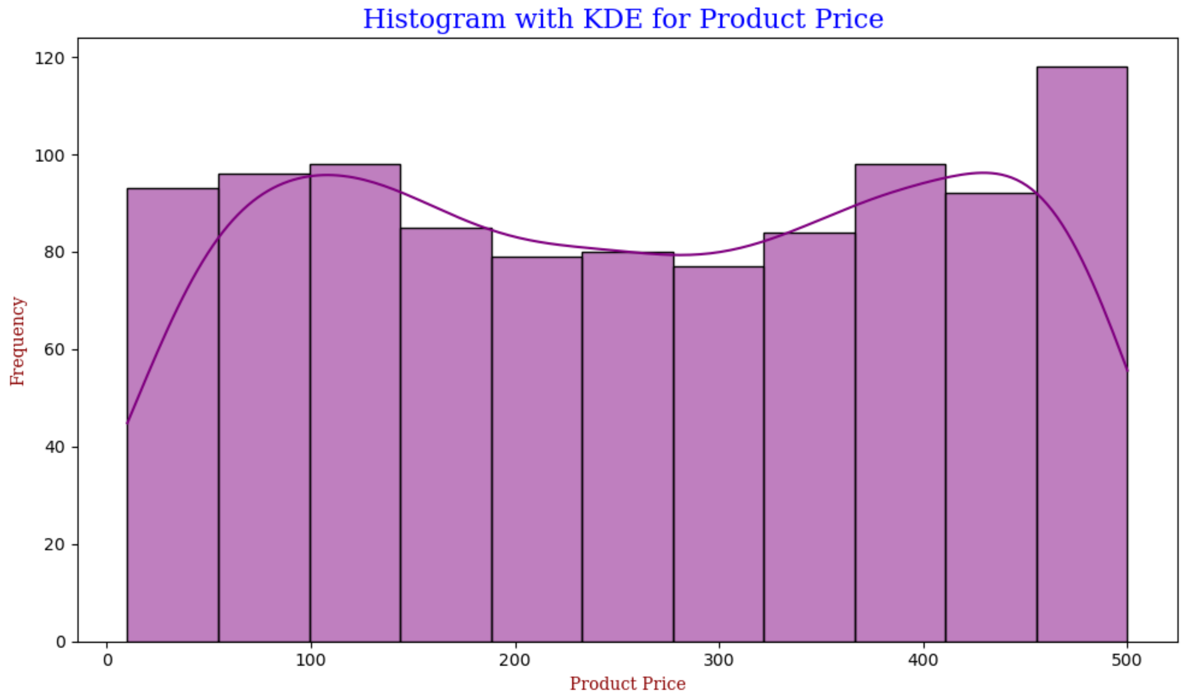


Description:

The plot is a heatmap representing the correlation coefficients between various variables related to customer transactions. Each square represents the correlation coefficient between the variables on the x-axis and the y-axis. The color scale on the right indicates that red corresponds to a positive correlation (up to +1), blue corresponds to a negative correlation (down to -1), and white represents no correlation (0).

The diagonal from the top left to the bottom right shows a perfect correlation (1.00) of variables with themselves, which is always the case. The rest of the heatmap shows very little to no correlation between the different variables, as indicated by the predominance of blue and white squares close to zero. For example, 'Customer Age' has a slight positive correlation (0.06) with 'Total Purchase Amount' and 'Returns', but overall, the variables do not display strong relationships with one another. This suggests that within this dataset, the attributes like 'Product Price', 'Quantity', 'Total Purchase Amount', and others do not have linear relationships with each other, at least not to a significant extent.

9.Histogram plot with KDE

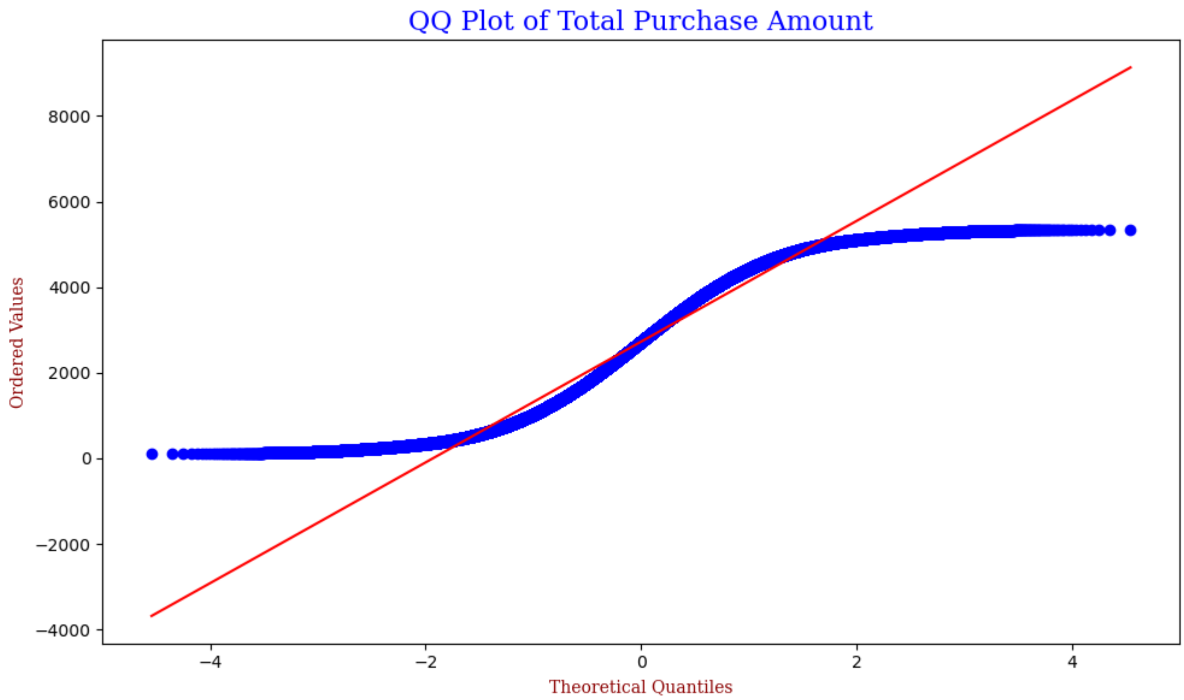


The graph is a histogram overlaid with a Kernel Density Estimate (KDE) curve for product prices. The histogram is a bar graph where the x-axis represents intervals or bins of product prices, and the y-axis represents the frequency of the products falling within each price bin. The bins in this histogram appear to be evenly spaced, and the height of each bar indicates how many products fall into each price range.

The KDE curve, drawn as a smooth line over the bars, estimates the probability density function of the product prices. It gives a sense of the distribution shape and where the majority of the data lies. In this case, the distribution seems to be multi-modal, as indicated by the multiple peaks in the KDE curve, suggesting that there are several popular price points for the products rather than a single common price.

Both the histogram and the KDE provide visual insights into the distribution of product prices. The histogram shows the actual data points in discrete intervals, while the KDE provides a smooth continuous approximation of the distribution. The areas where the KDE curve peaks correspond to the intervals where the bars are tallest, indicating higher frequencies of product prices.

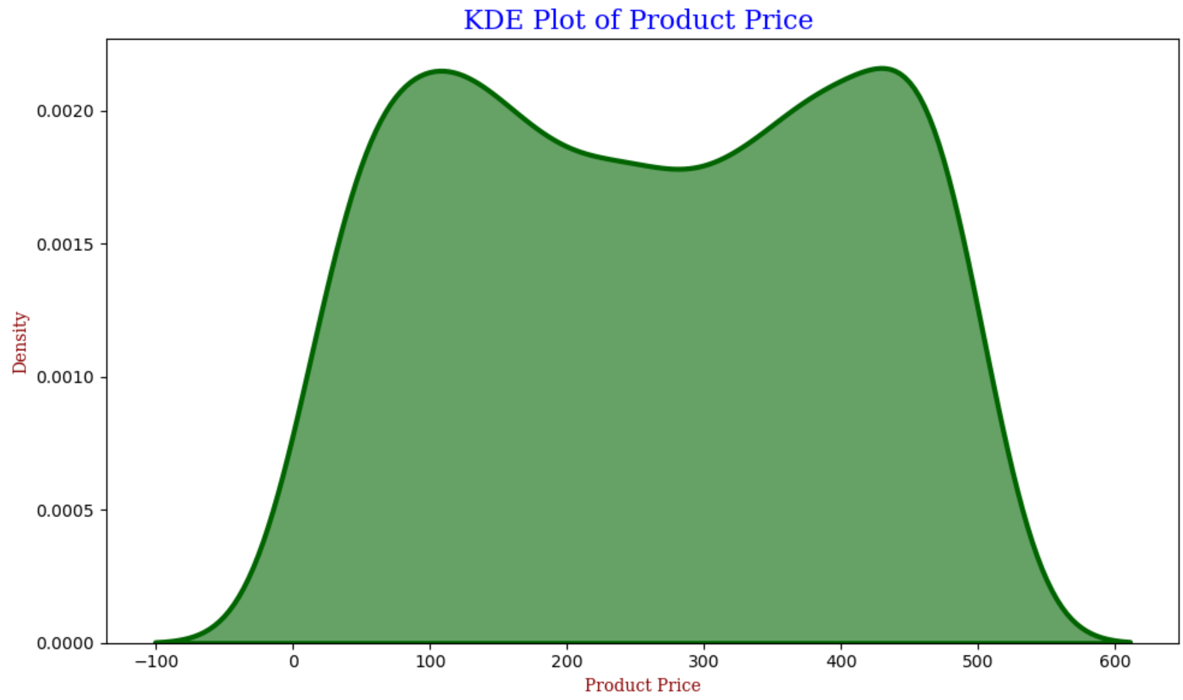
10.QQ Plot



In this QQ plot, we have the "Theoretical Quantiles" on the x-axis and the "Ordered Values" of the total purchase amount on the y-axis. The blue points represent the actual quantiles from the dataset, while the red line represents the expected quantiles if the data were normally distributed.

In this plot, the blue dots deviate significantly from the red line, especially at the ends, which suggests that the distribution of the total purchase amount is not normal. The lower tail (left side) dips below the line, indicating lighter tails than the normal distribution, and the upper tail (right side) rises above the line, indicating heavier tails. This might imply the presence of outliers or that the data has a skewed distribution.

11.KDE Plot with fill,alpha=0.6, palette and line width

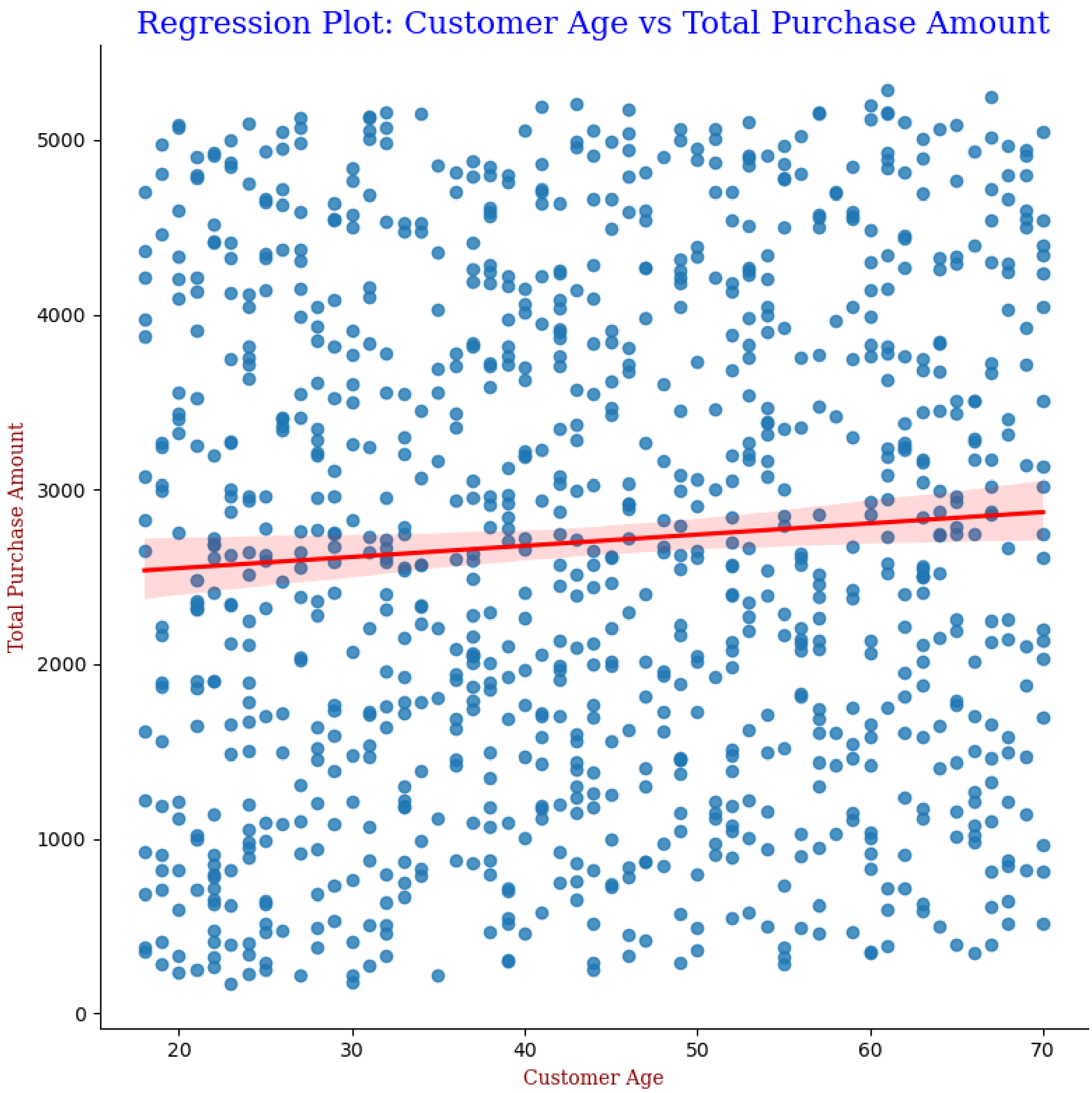


Description:

The graph is a Kernel Density Estimation (KDE) plot, which is a type of data visualization used in statistics to estimate the probability density function of a continuous random variable. The plot shows a curve that represents the density of product prices, where the x-axis is labeled "Product Price" and the y-axis is labeled "Density".

In this KDE plot, the prices range from a little below 0 to just over 600, with the highest density occurring in two regions, suggesting that there are two common price points where products are clustered. The first peak is around the 100 mark and the second, larger peak is around the 400 mark. The area under the curve represents the distribution of the product prices. The curve is smooth, which is characteristic of KDE plots as they provide a smoothed version of the histogram. The fact that there are negative prices suggests that there might be some data entry errors or that the products include options or bundles that result in discounts, effectively resulting in a negative price when considered as part of a transaction.

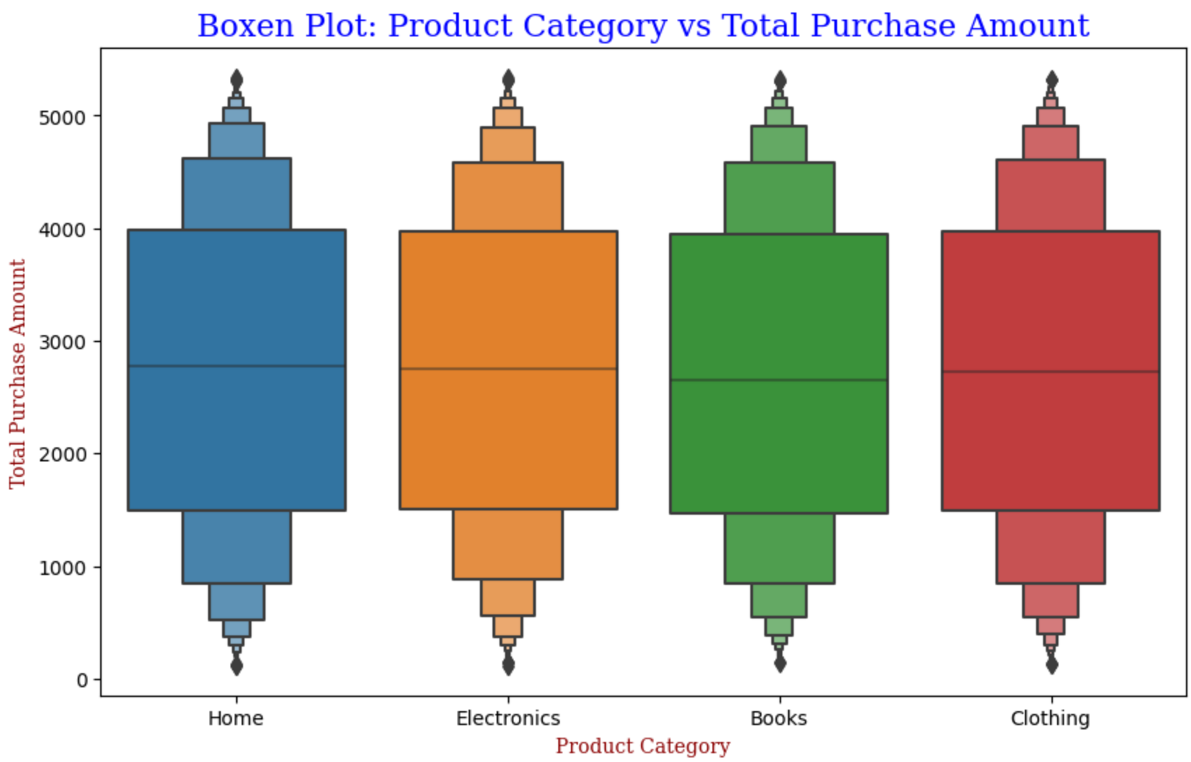
12.lm or reg plot with scatter representation and regression line



Description:

The plot shows a regression plot that depicts the relationship between Customer Age (on the x-axis) and Total Purchase Amount (on the y-axis) using a scatter plot to display individual data points and a regression line to indicate the trend. Each dot represents an individual purchase event, with the position along the x-axis showing the customer's age and along the y-axis showing the total amount of their purchase. The red line represents the best-fit line through the data points, indicating the average trend and the shaded area around the line may suggest the confidence interval for the regression estimate, providing a sense of the uncertainty around the predicted values. The plot indicates a slight positive trend, suggesting that the Total Purchase Amount might increase with Customer Age, although the data points are widely scattered, indicating a lot of variability and a potentially weak correlation between these two variables.

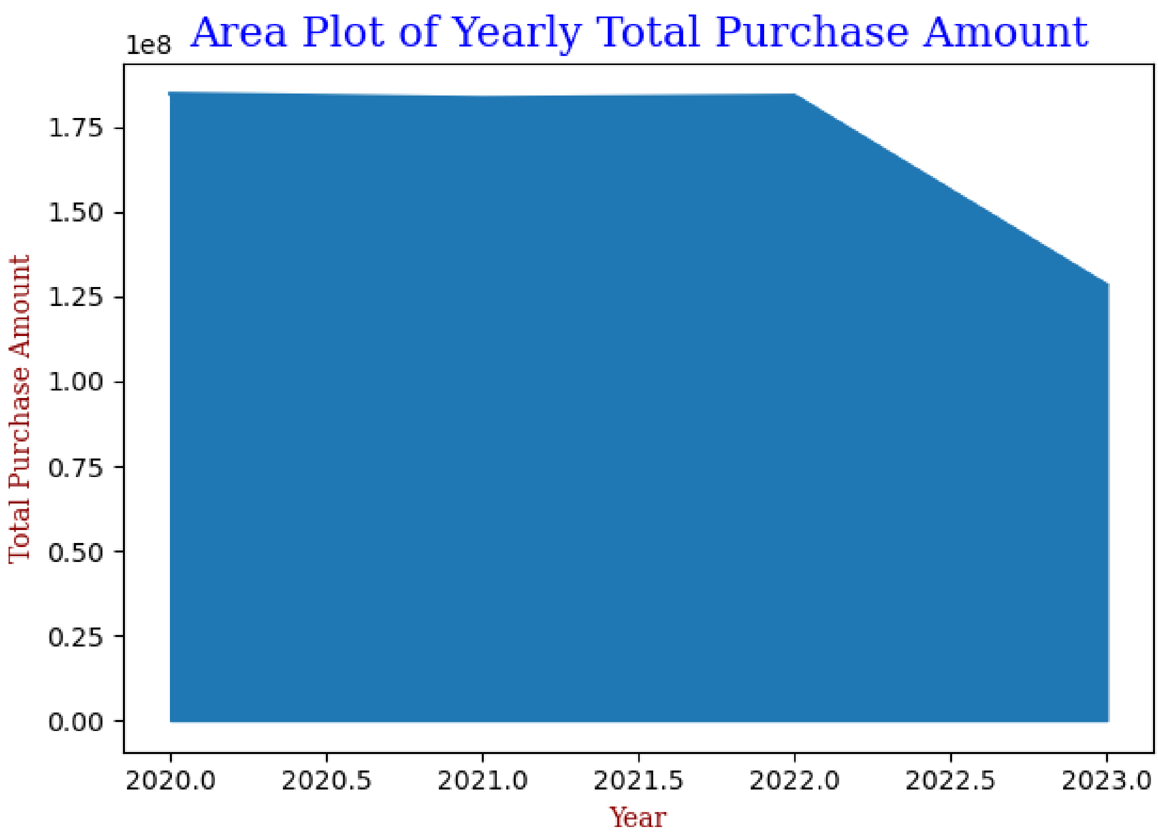
13.Multivariate Box or Boxen plot



Description:

The graph presents a boxen plot, which is an enhanced box plot, showing the distribution of Total Purchase Amount across four different product categories: Home, Electronics, Books, and Clothing. The central line in each box represents the median purchase amount, while the boxes show the interquartile ranges, indicating where the middle 50% of the data lies. The "whiskers" can extend to show the range of the data, and any points beyond the whiskers could be considered outliers.

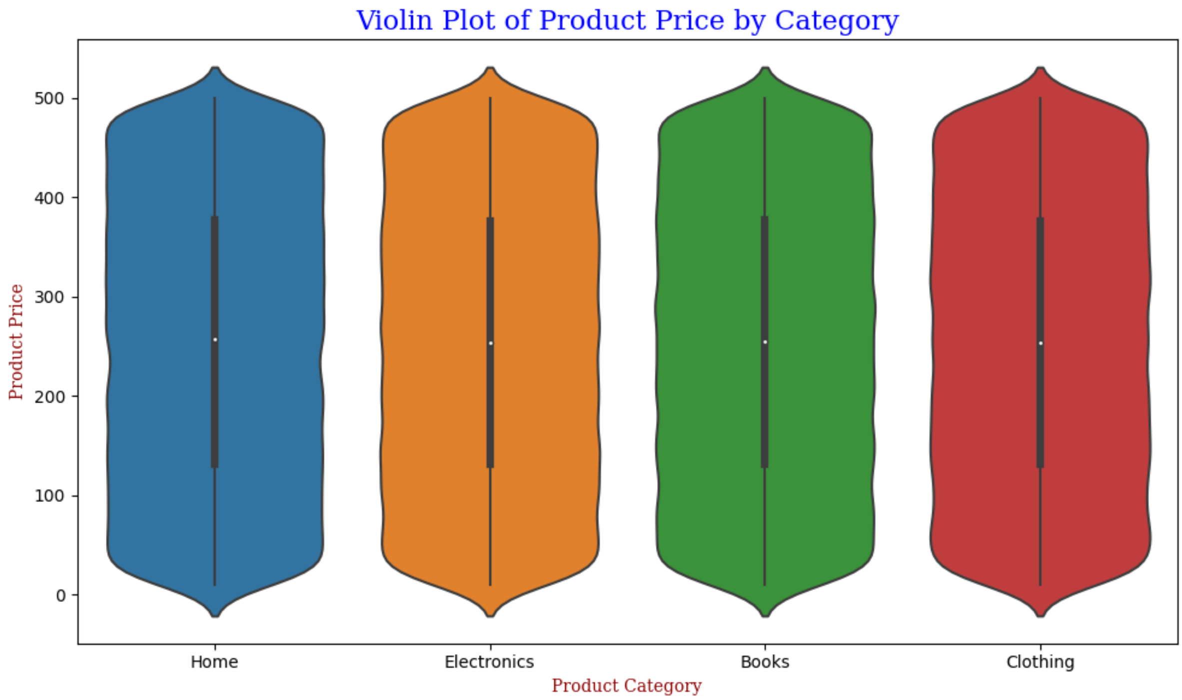
14.Area plot



Description:

The graph shows an area plot that tracks the Yearly Total Purchase Amount over time from 2020 to 2023. This type of plot helps visualize the volume of sales over the years and can indicate trends, such as growth or decline. The y-axis, labeled "Total Purchase Amount," is scaled to 1e8, indicating that the data is in the hundreds of millions. The x-axis represents time, marked from 2020 to 2023 in half-year increments. The plot demonstrates a rising trend in total purchases from 2020, peaking somewhere after 2021, followed by a decline towards 2023. The area under the curve represents the cumulative purchase amount over time, with the filled color making it easy to compare volumes between different periods visually.

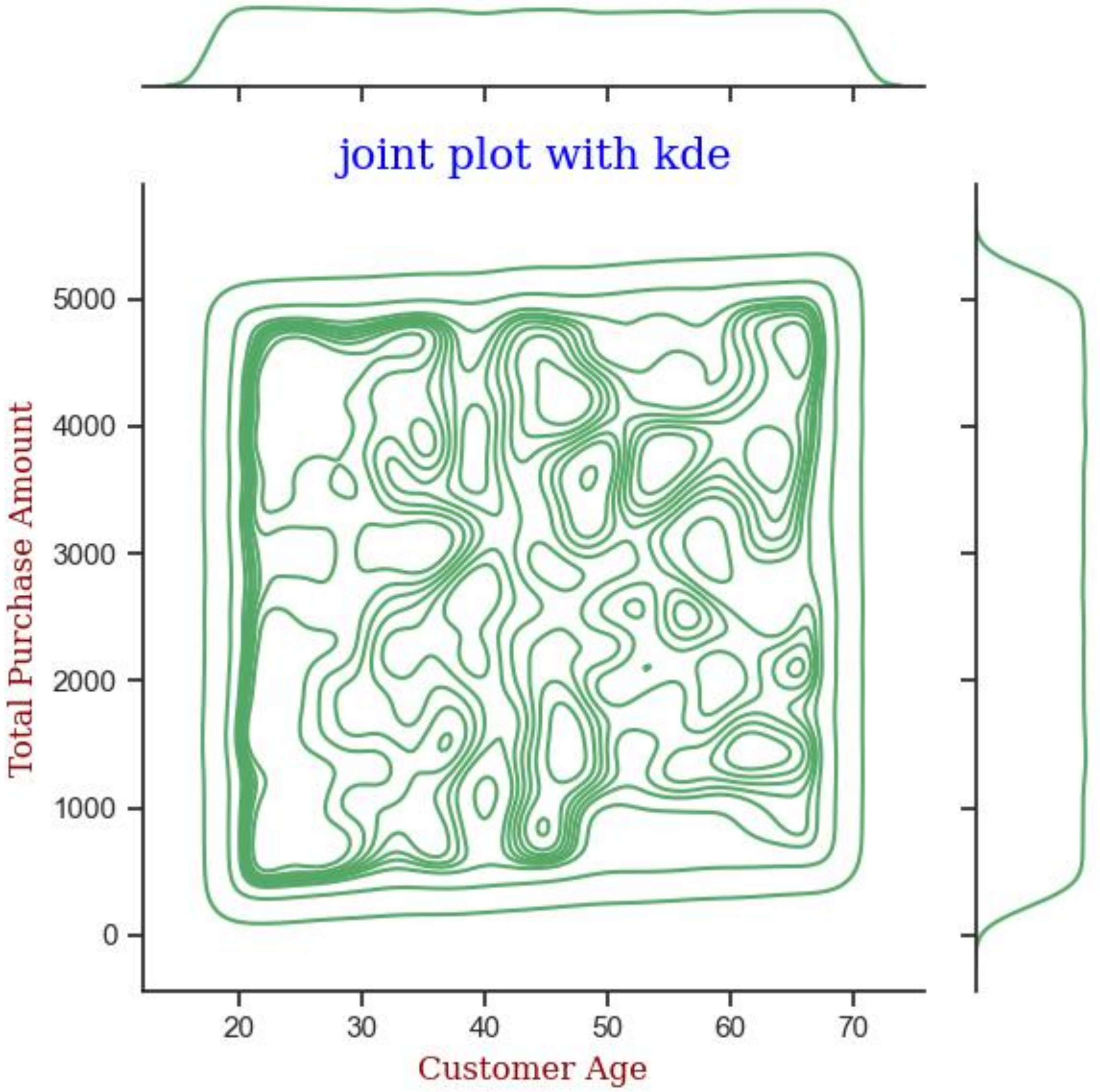
15.Violin plot



Description:

The graph depicts a violin plot illustrating the distribution of product prices across four different product categories: Home, Electronics, Books, and Clothing. Each 'violin' represents a category and shows the price distribution, with the width of the plot indicating the frequency of price points. Within each violin, a white dot represents the median price, the thick black bar in the center depicts the interquartile range (IQR), and the thin black line represents the rest of the distribution, possibly excluding outliers. The plot suggests varied price distributions for each category, with some categories like Electronics showing a wider range of prices (wider distribution) compared to others such as Books.

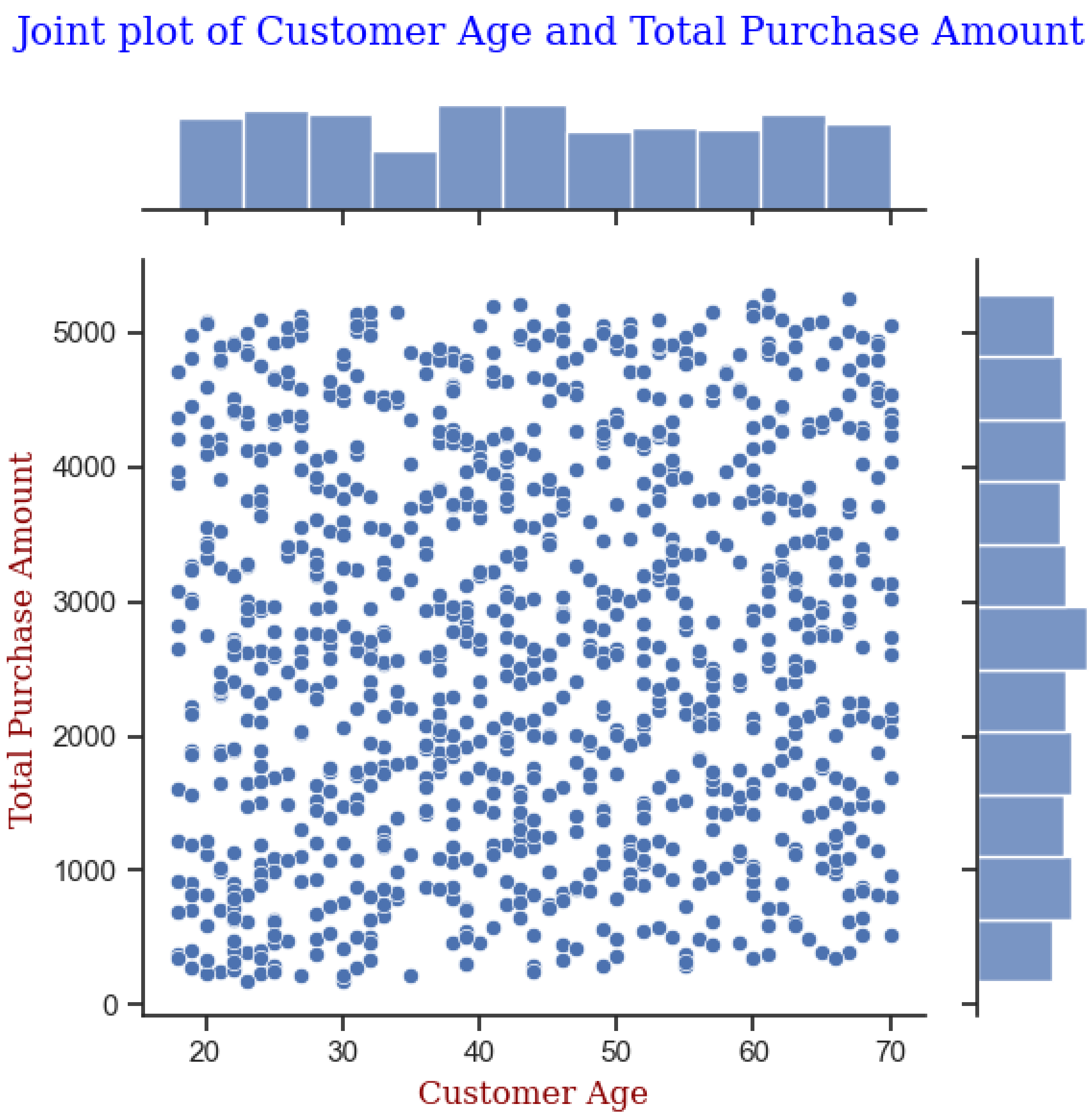
16.Joint plot with KDE



Description:

The graph shows a joint plot with kernel density estimation (KDE), a data visualization technique that depicts the distribution and relationship between two numerical variables: 'Customer Age' on the horizontal axis and 'Total Purchase Amount' on the vertical axis. The plot combines a scatter plot with a two-dimensional KDE, which is represented by the contour lines, indicating where data points are concentrated. Marginal histograms or KDE plots are along the top and right axes, showing the distribution of each variable. This type of plot is useful for visualizing the density and distribution of data points within the space defined by the two variables.

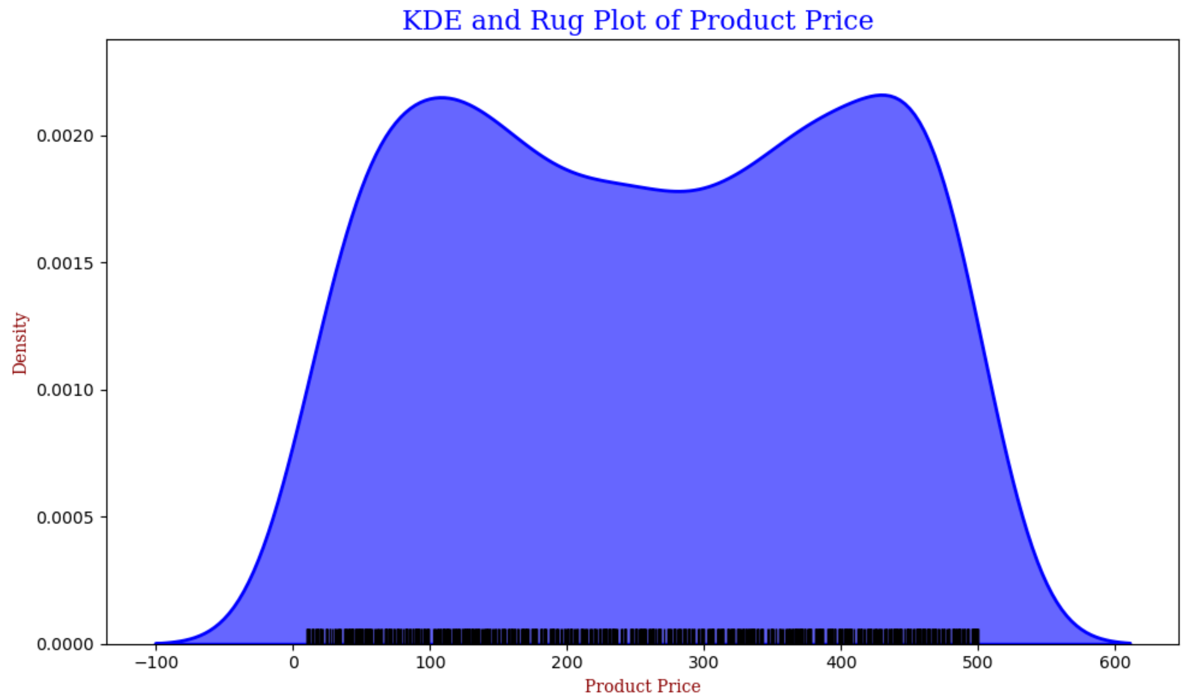
17. Joint Plot with scatter representation



Description:

The graph depicts a joint plot showing the relationship between 'Customer Age' and 'Total Purchase Amount'. Scatter points represent individual data entries, plotted with 'Customer Age' on the x-axis and 'Total Purchase Amount' on the y-axis. Along the top and right margins are histograms that display the distribution of each variable independently, with the top histogram for 'Customer Age' and the right histogram for 'Total Purchase Amount'.

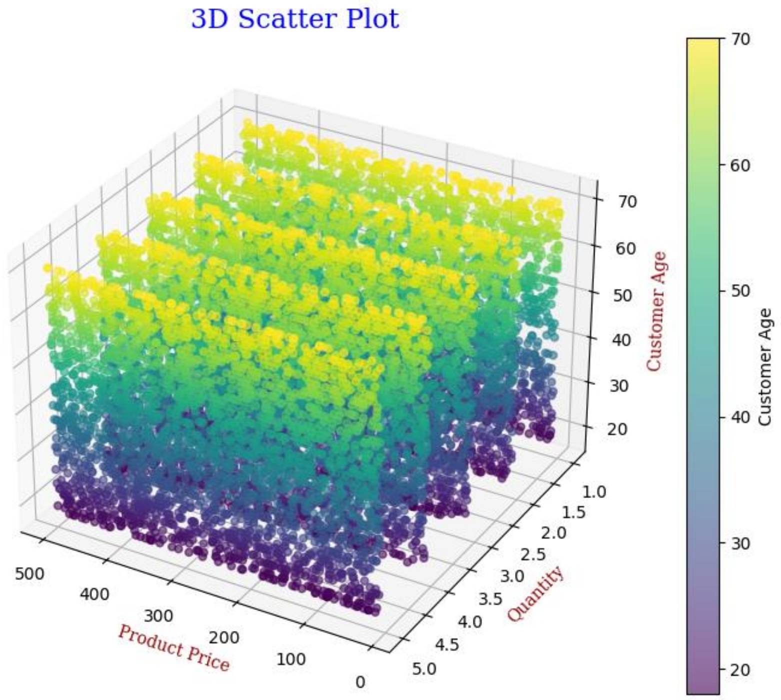
18.Rug plot



Description:

The graph illustrates a KDE (Kernel Density Estimate) and Rug Plot of Product Price. The KDE provides a smooth estimate of the data's distribution and is represented by the filled blue curve, with the y-axis indicating the density and the x-axis representing the product price. The rug plot, shown by the small vertical lines at the bottom of the graph, indicates the actual data points along the price axis. The plot indicates two prominent peaks in product price density, suggesting that there are two price ranges where products are more commonly priced. The areas under the curve where the density is lower indicate fewer products at those price points.

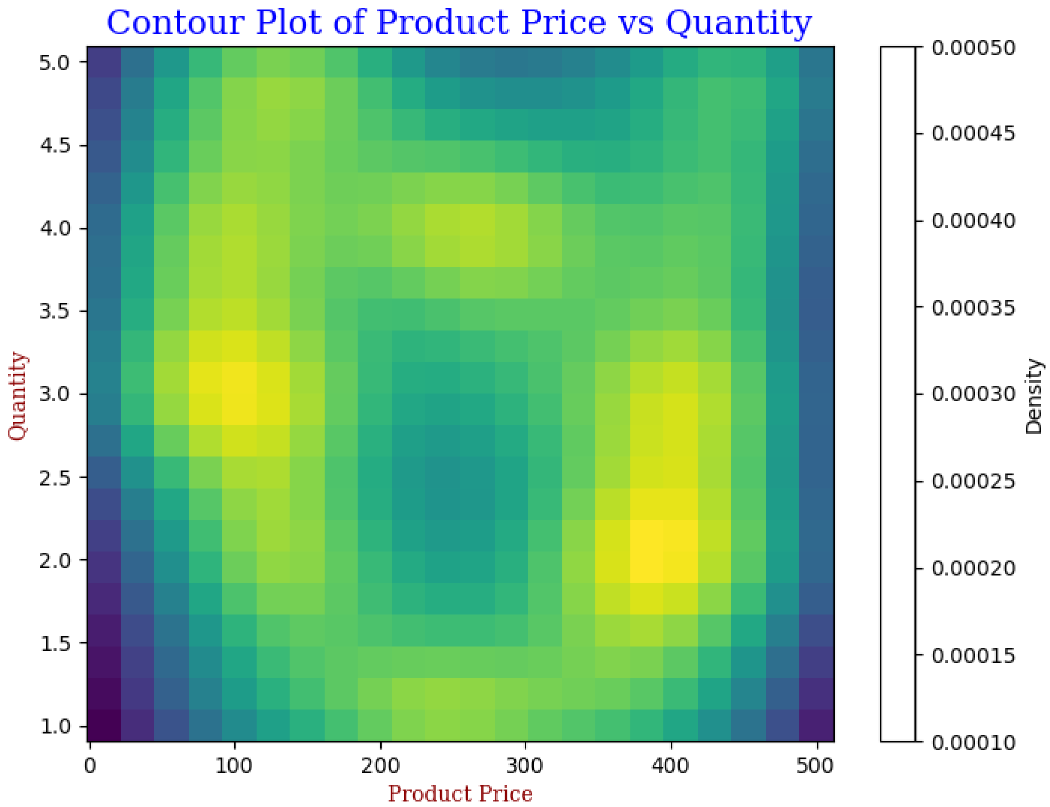
19.3D plot



Description:

The plot presents a 3D scatter plot, which is a graphical representation used to show the relationship between three quantitative variables. The axes represent Product Price, Quantity, and Customer Age, with individual data points plotted in the 3D space. The color gradient, indicated by the legend on the right, represents Customer Age, varying from purple (younger) to yellow (older). It appears that there's a clustering of points across the Product Price and Quantity axes, suggesting a relationship between these variables and the age of the customer. The density of points seems to be greater at lower quantities and product prices, which could imply that younger customers, indicated by the purple points, tend to buy less expensive items or buy in smaller quantities. Conversely, there might be a trend where older customers, represented by yellow points, are associated with higher prices and larger quantities, although this is less clear from the image provided. This type of plot is useful for identifying trends and patterns in multidimensional data.

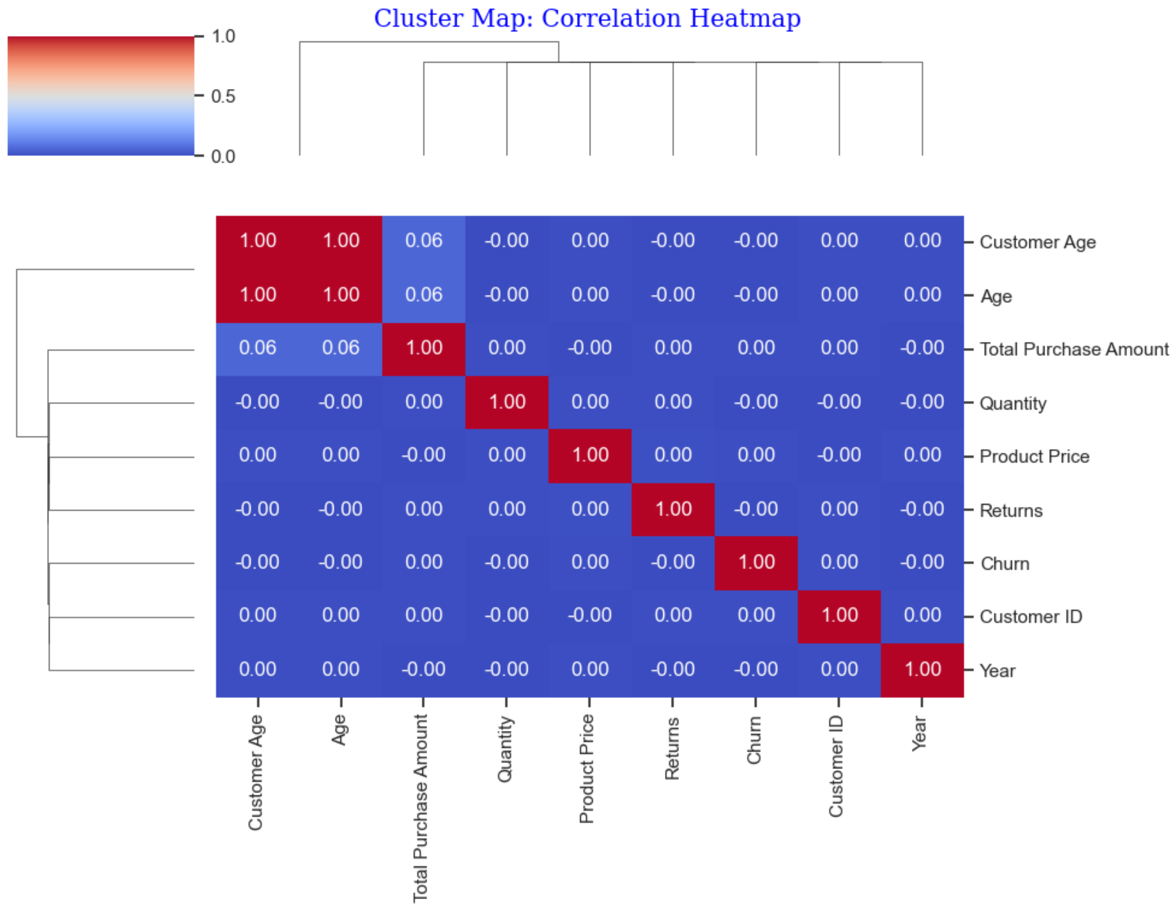
20.contour plot



Description:

The graph is a contour plot that visualizes the density of data points based on two variables: Product Price (on the x-axis) and Quantity (on the y-axis). The plot uses color gradations to represent the density of data points at various price and quantity combinations, with the color bar on the right indicating the density scale. Darker or more intense colors correspond to higher densities, meaning a larger number of data points are found at these levels. The plot shows that the highest density of data points (the yellow area) is concentrated around a lower quantity and a lower price range. This suggests that most of the products are purchased in smaller quantities and at lower prices. There's a relatively even distribution of density across the product price range, but there's a distinct decrease in density as the quantity increases. This implies that as the quantity of products purchased increases, there are fewer transactions at those higher quantities, regardless of the price.

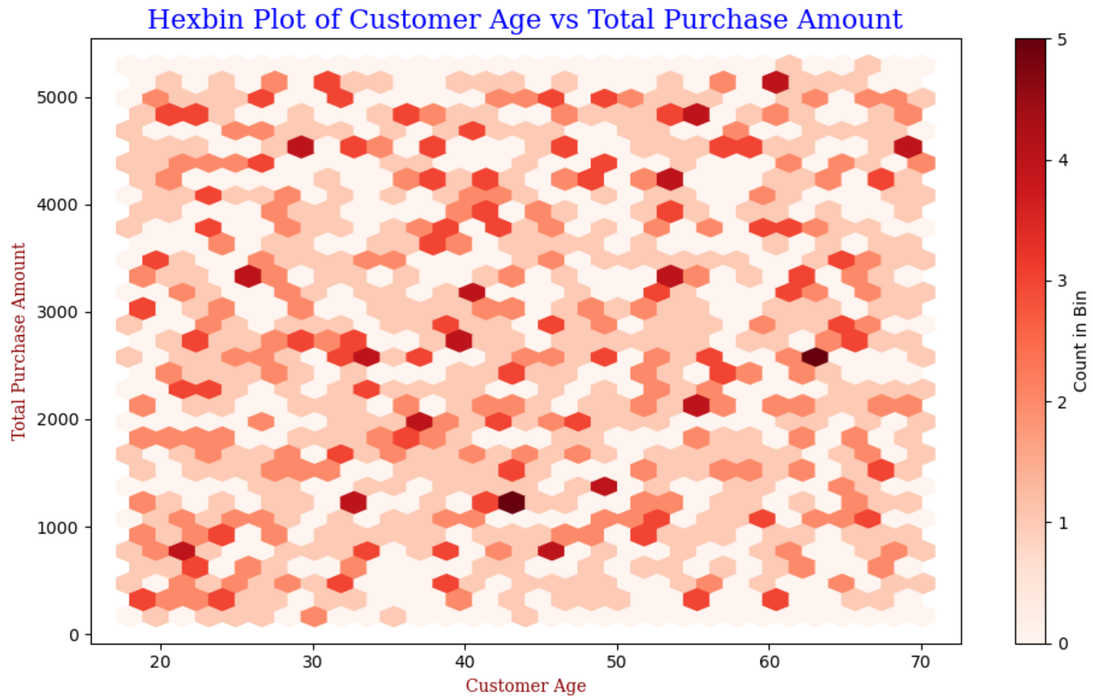
21.Cluster Map



Description:

The graph shows a cluster map with a correlation heatmap, used to visualize the strength of relationships between different variables in a dataset. The heatmap portion displays correlation coefficients ranging from -1.0 to 1.0, with 1.0 indicating a perfect positive correlation, -1.0 indicating a perfect negative correlation, and 0 indicating no correlation. Here, variables such as "Customer Age," "Age," "Total Purchase Amount," "Quantity," "Product Price," "Returns," "Churn," "Customer ID," and "Year" are compared. The dendrogram suggests there's little to no clustering based on similarity, implying the dataset variables do not have strong relationships with one another.

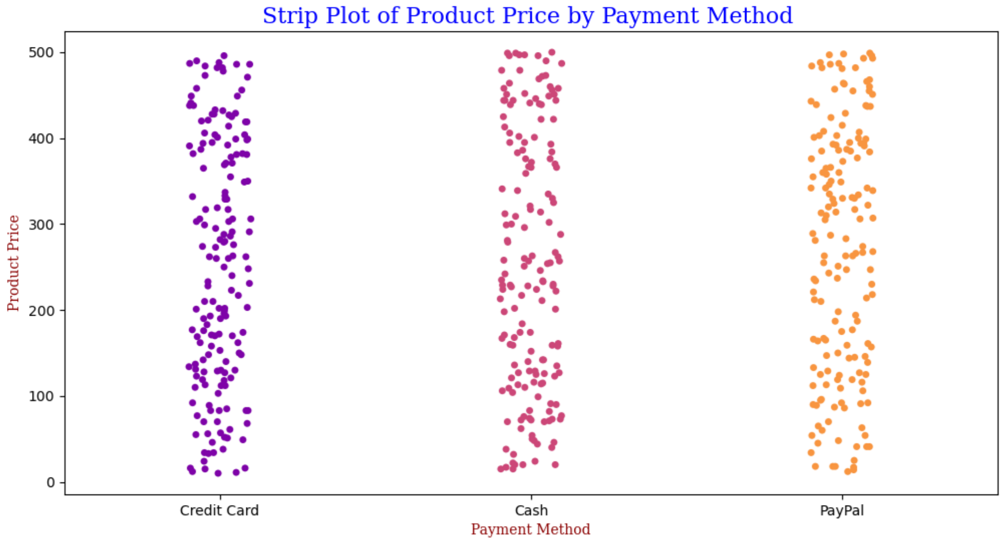
22.Hexbin plot



Description:

The plot presents a hexbin plot showing the relationship between 'Customer Age' and 'Total Purchase Amount. The plot suggests that certain age groups have a higher concentration of purchase amounts, revealing patterns and trends within the dataset.

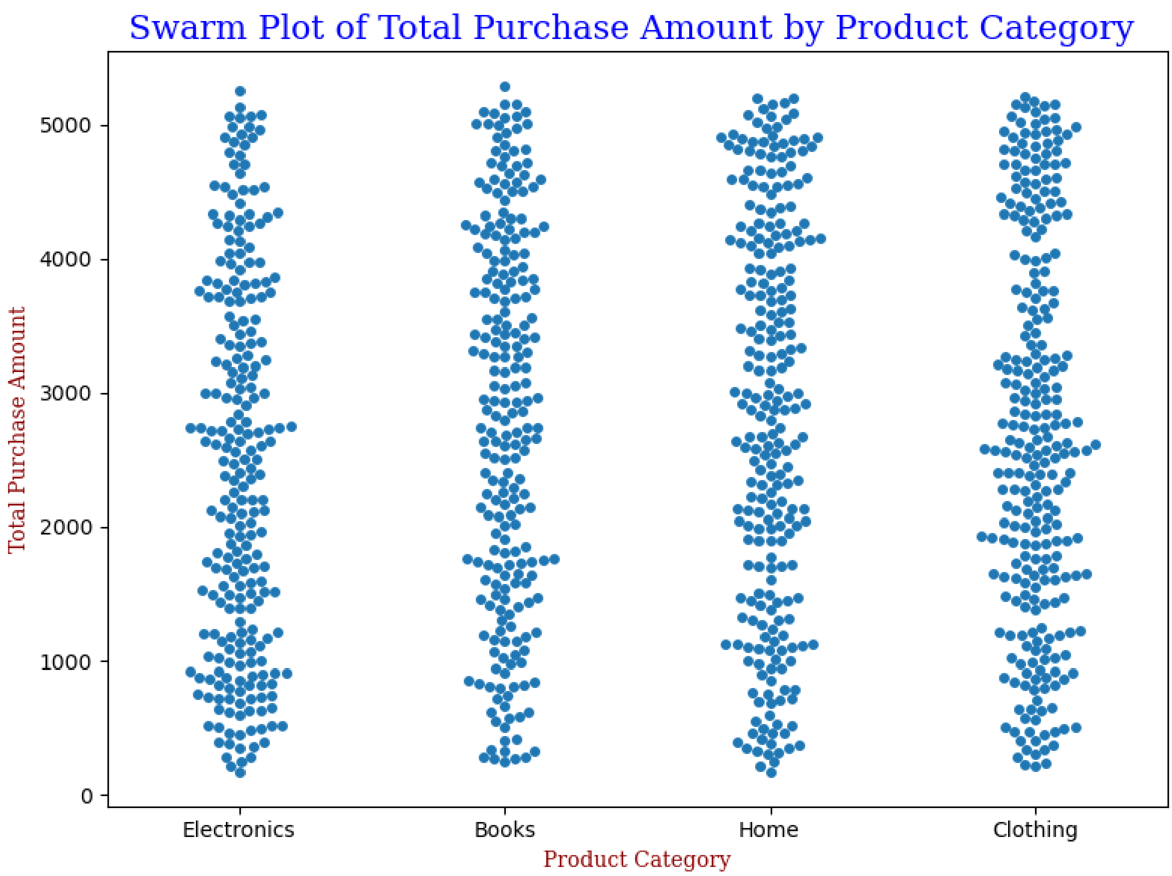
23.Strip plot



Description:

The graph shows a strip plot titled "Strip Plot of Product Price by Payment Method," depicting the distribution of product prices across three different payment methods: Credit Card, Cash, and PayPal. In this type of plot, individual transactions are represented by dots, allowing the viewer to see the concentration of transactions at different price points for each payment method. The vertical axis represents the product price, while the horizontal axis lists the payment methods. The plot reveals a dense clustering of transactions across a wide range of prices for each payment method. This visualization helps identify patterns such as the range and density of prices customers are paying with each payment method, which could indicate preferences or spending habits associated with each method.

24.Swarm plot



Description:

The plot displays a swarm plot of 'Total Purchase Amount' by 'Product Category'. It shows individual purchase amounts as points, grouped by categories such as Electronics, Books, Home, and Clothing. The plot is useful for observing the distribution and density of purchases within each category, indicating where clusters of purchase amounts are common and the range of purchases for each type of product. There is a notable concentration of data points within specific ranges for each category, suggesting common price points or purchase behaviors among customers.

i)Subplots a)

Description:

This subplot presents various data visualizations related to product sales and customer churn. The "Average Product Price by Payment Method" bar chart compares the average price of products paid for by cash, credit card, and PayPal, which appear similar across payment methods. The "Customer Churn Distribution" pie chart shows the proportion of customers who have churned (20.1%) versus those who have not (79.9%). "Total Quantity Sold by Year" is a bar chart illustrating a decline in product quantity sold over four years, with a noticeable drop in 2023. Lastly, the "Product Price Distribution by Category" box plot reveals the spread of product prices within different categories, such as home, electronics, books, and clothing, with electronics showing the widest price range.

b)

Description:

A histogram with a fitted line showing the frequency distribution of purchase amounts, indicating the majority of purchases fall between certain price ranges.

A pie chart showing an equal 50/50 split between female and male customers.

A bar chart displaying the average quantity of products purchased in different categories, showing relatively uniform averages across categories like Books, Clothing, Electronics, and Home.

A bar chart comparing the total returns between female and male customers, with both genders having a similar number of returns.

A line graph illustrating the number of products purchased over time from January 2020 to July 2023, with fluctuations indicating varying purchase volumes across the timeframe.

A heatmap showing the correlation between different variables like Product Price, Quantity, Total Purchase Amount, and Customer Age, with a range of positive to negative correlations indicated by varying color intensities.

c)

Description:

This comprises four distinct visualizations related to customer demographics and spending patterns:

The "Avg Purchase Amount by Gender" bar chart shows that, on average, males spend slightly more than females.

The "Histogram of Customer Age" indicates a bimodal distribution, suggesting two age groups where customers are more concentrated, with peaks around the mid-20s and mid-50s.

The "Product Price Distribution by Gender" box plot reveals that the median product price is slightly higher for females than for males, with a wider range and more variability in the prices of products purchased by females.

The "Age vs Total Purchase Amount by Gender" scatter plot displays a distribution of total purchase amounts by customer age, distinguished by gender. It appears that there's no strong trend or correlation between age and total purchase amount, and both genders are relatively similar in their spending across different ages.

j) Tables

This is a statistics table of the cleaned dataset created using PrettyTable.

Statistics Table:

Feature	Mean	Median	Correlation	Std Dev	Variance
Customer ID	25017.63	25011.00	0.00	14412.52	207720609.33
Product Price	254.74	255.00	0.00	141.74	20089.69
Quantity	3.00	3.00	-0.00	1.41	2.00
Total Purchase Amount	2725.39	2725.00	0.00	1442.58	2081025.79
Customer Age	43.80	44.00	-0.00	15.36	236.08
Returns	0.50	1.00	-0.00	0.50	0.25
Age	43.80	44.00	-0.00	15.36	236.08
Churn	0.20	0.00	1.00	0.40	0.16

It provides a quantitative summary of an e-commerce dataset's key features. It details the mean, median, correlation with churn, standard deviation, and variance for each listed attribute. The 'Customer ID' feature, with its high standard deviation and variance, indicates a broad range across its unique values, which is typical for identifier fields. 'Product Price' exhibits a considerable variance, suggesting a diverse range of product prices within the dataset. Conversely, 'Quantity' and 'Returns' show low variance, hinting at less variability and possibly indicating that most customers purchase similar quantities and return items at a consistent rate. Notably, except for 'Churn' which understandably correlates perfectly with itself, other features have a zero correlation with 'Churn', suggesting no linear relationship with customer churn within this dataset.

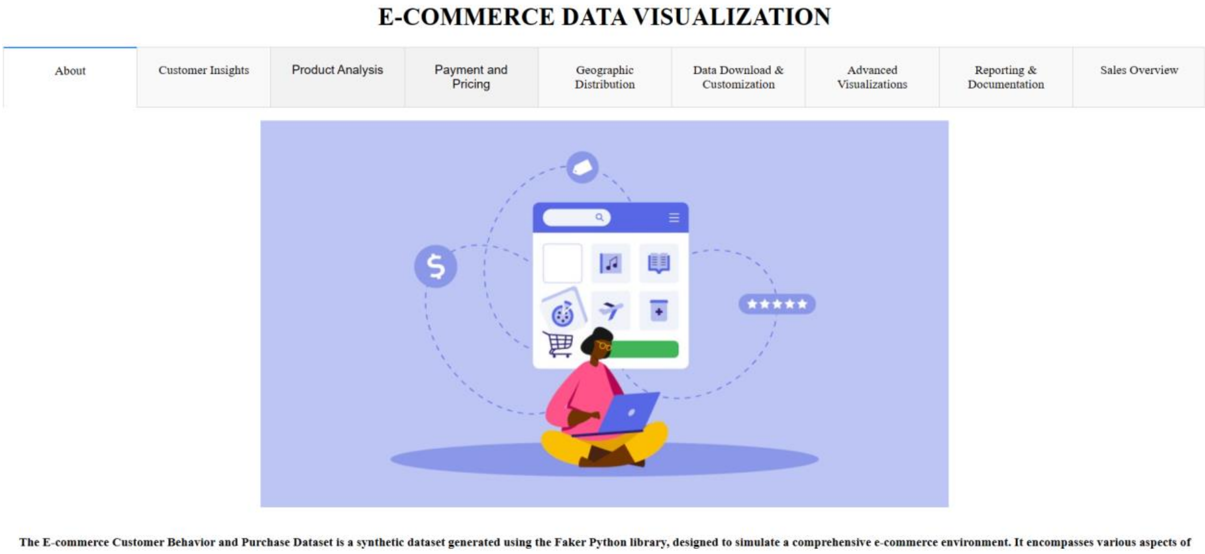
Correlation Table:

Feature1	Feature2	Correlation	Inference
Customer ID	Product Price	-0.00	Weak or no correlation
Customer ID	Quantity	-0.00	Weak or no correlation
Customer ID	Total Purchase Amount	0.00	Weak or no correlation
Customer ID	Customer Age	0.00	Weak or no correlation
Customer ID	Returns	0.00	Weak or no correlation
Customer ID	Age	0.00	Weak or no correlation
Customer ID	Churn	0.00	Weak or no correlation
Product Price	Quantity	0.00	Weak or no correlation
Product Price	Total Purchase Amount	-0.00	Weak or no correlation
Product Price	Customer Age	0.00	Weak or no correlation
Product Price	Returns	0.00	Weak or no correlation
Product Price	Age	0.00	Weak or no correlation
Product Price	Churn	0.00	Weak or no correlation
Quantity	Total Purchase Amount	0.00	Weak or no correlation
Quantity	Customer Age	-0.00	Weak or no correlation
Quantity	Returns	0.00	Weak or no correlation
Quantity	Age	-0.00	Weak or no correlation
Quantity	Churn	-0.00	Weak or no correlation
Total Purchase Amount	Customer Age	0.06	Weak or no correlation
Total Purchase Amount	Returns	0.00	Weak or no correlation
Total Purchase Amount	Age	0.06	Weak or no correlation
Total Purchase Amount	Churn	0.00	Weak or no correlation
Customer Age	Returns	-0.00	Weak or no correlation
Customer Age	Age	1.00	Strong positive correlation
Customer Age	Churn	-0.00	Weak or no correlation
Returns	Age	-0.00	Weak or no correlation
Returns	Churn	-0.00	Weak or no correlation
Age	Churn	-0.00	Weak or no correlation

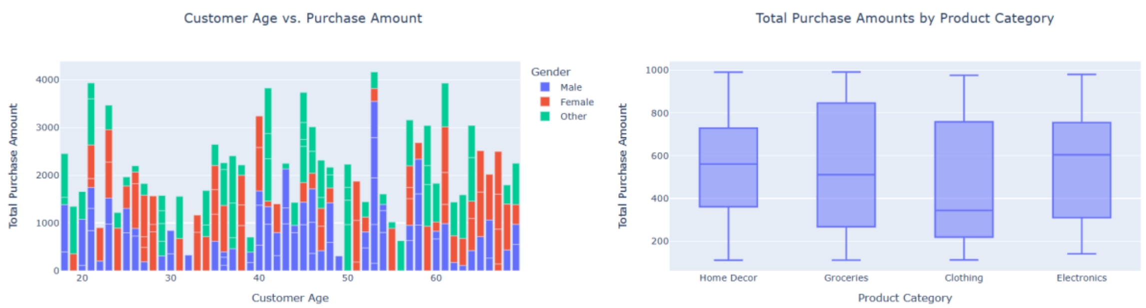
It displays a "Correlation Table," which outlines the Pearson correlation coefficients between different pairs of features within an e-commerce dataset. The coefficients range from -0.00 to 0.00, indicating no meaningful linear relationship between the pairs. For instance, 'Customer ID' shows no correlation with variables like 'Product Price', 'Quantity', and 'Total Purchase Amount', among others, suggesting that the identifier for customers does not influence or is not influenced by these other factors. This extends to the correlation between 'Product Price' and other variables such as 'Quantity' and 'Customer Age', all the way down to 'Total Purchase Amount' and 'Customer Age'. All pairs listed are described as having "Weak or no correlation", which implies that there's no apparent linear dependency between them. This kind of analysis is crucial for identifying which features might be independent of each other, an important consideration in model building where multicollinearity can be an issue. However, it should be noted that a lack of correlation does not imply a lack of any relationship, as there could be non-linear relationships not captured by Pearson's coefficient.

k) Dashboard

TAB: About

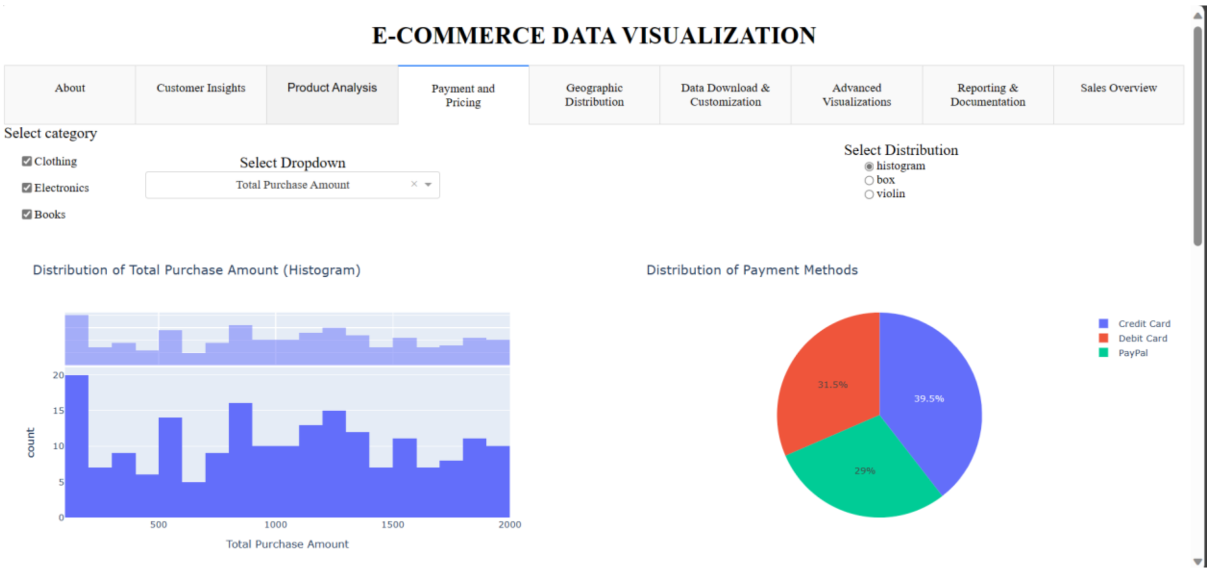


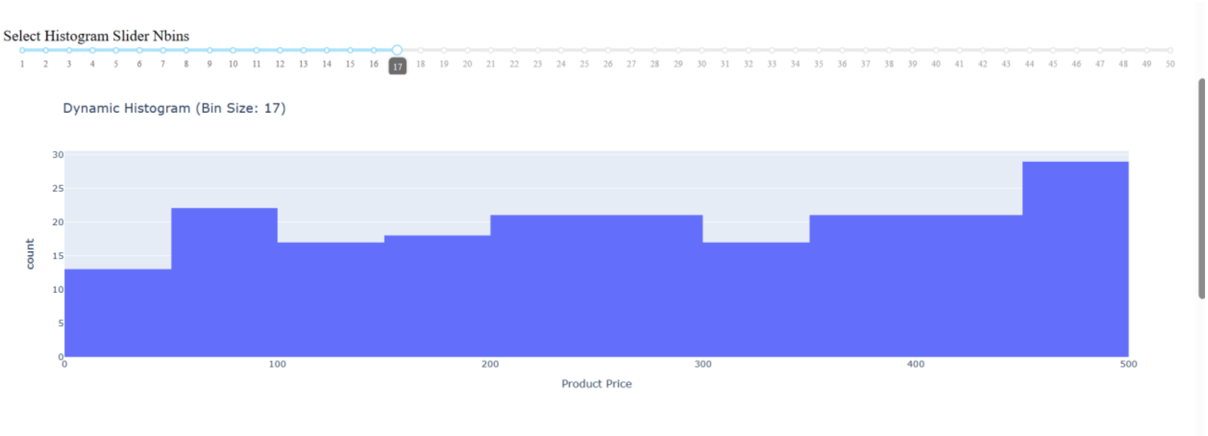
TAB : Customer Insights



The "Customer Insights" tab in the Dash application is a dedicated section for interactive data exploration focused on understanding customer demographics and purchase behaviors. It provides users with tools to filter data by gender and product categories, which then dynamically updates various visualizations. The tab features scatter plots to analyze relationships such as age against purchase amount, potentially segmented by gender, and violin plots to assess product price distributions across categories. Box plots offer insights into the variance and outliers in purchase amounts by category. An additional interactive scatter plot enhances user engagement by allowing for deeper exploration of individual data points. This tab serves as a powerful analytical tool, enabling stakeholders to derive actionable insights on customer preferences and spending patterns, ultimately aiding strategic business decisions in marketing and product development.

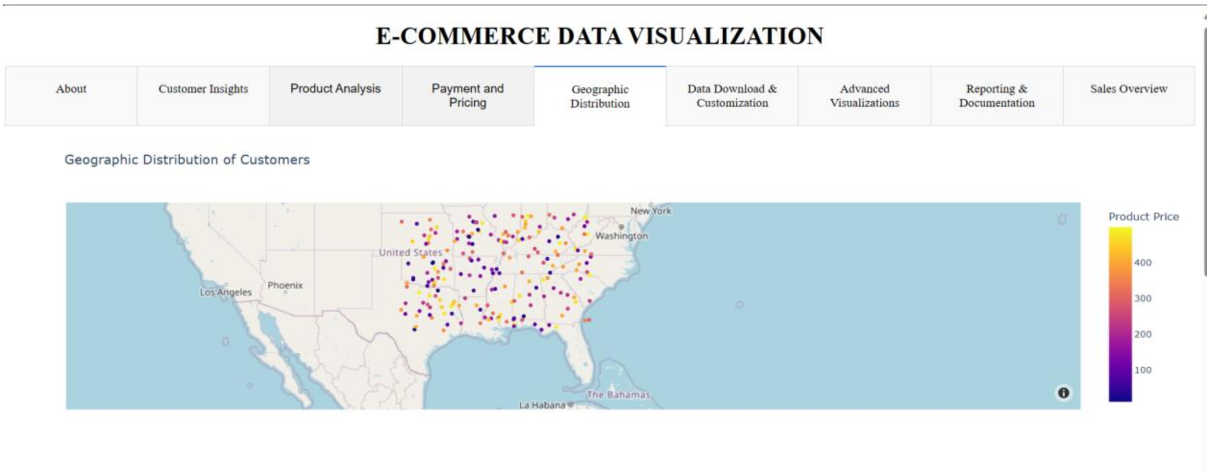
TAB: Payment and Pricing

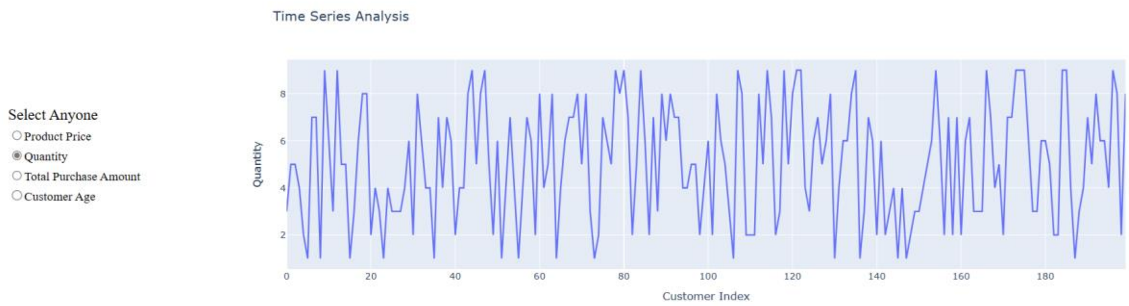




The "Payment and Pricing" tab in the Dash application is designed for an in-depth analysis of sales data about different payment methods and pricing strategies. Users can select from various product categories and payment methods to tailor the data visualization to specific areas of interest. The tab has a range of interactive charts such as histograms, pie charts, heatmaps, and dynamic histograms, each serving a unique analytical purpose. The histogram might be used to display the distribution of product prices or quantities, while the pie chart could illustrate the breakdown of different payment methods used in transactions. A heatmap is present to show correlations between different sales metrics, providing insights into relationships like price and quantity or customer age and total purchase amount. The dynamic histogram, adjustable via a slider, offers flexibility in viewing data distributions with varying levels of granularity. This tab is a crucial tool for understanding the financial aspects of the business, like customer spending patterns, popular payment methods, and product pricing strategies, which are essential for making informed pricing and sales decisions.

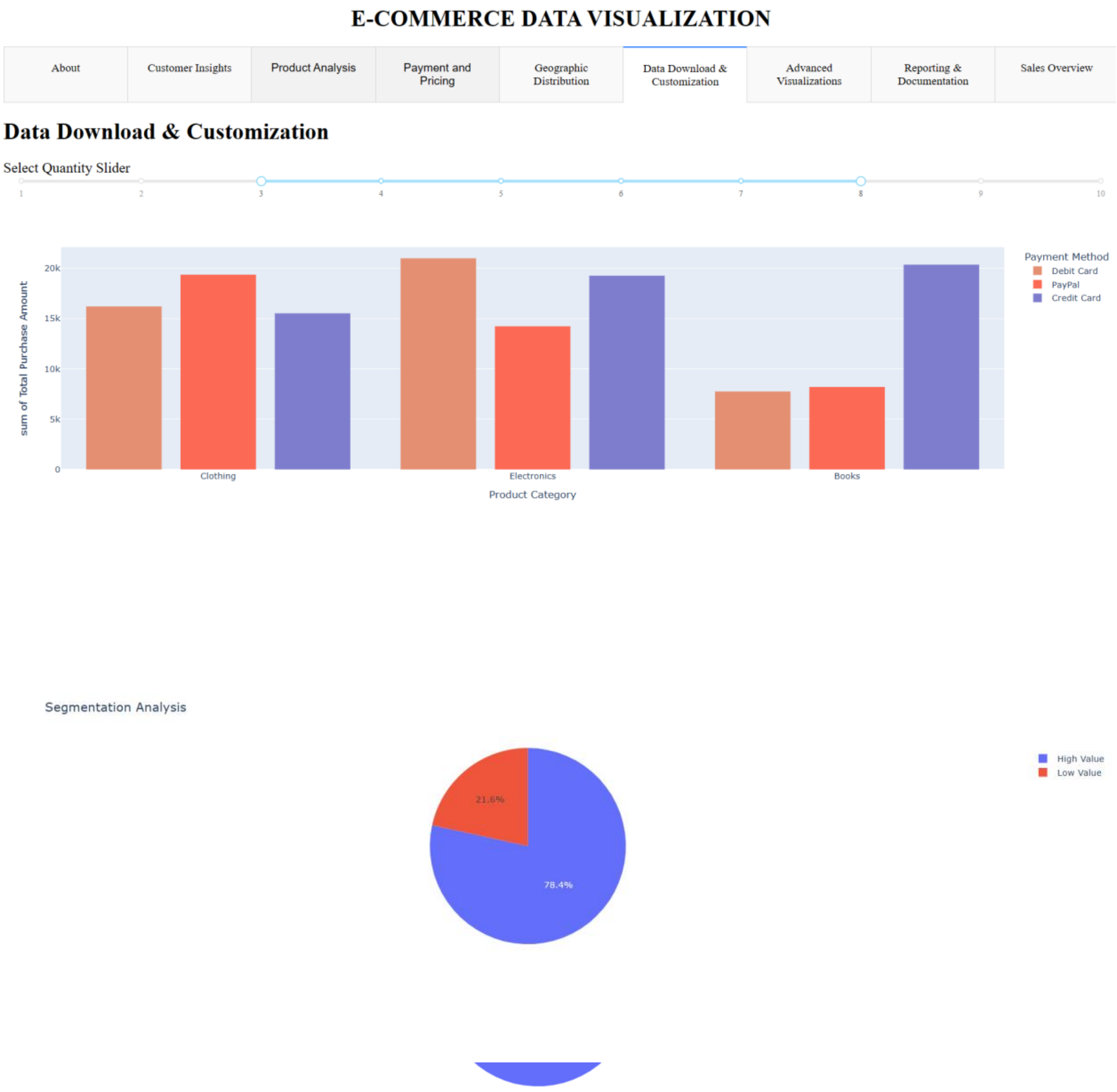
TAB: Geographic Distribution





The "Geographic Distribution" tab in the Dash application is specifically designed to visualize customer data on a geographical scale. This tab features a map plot, which displays the geographical spread of customers, using data points to represent customer locations. The inclusion of interactive elements like radio buttons to filter metrics and a range slider to adjust for customer age allows users to customize the visualization to their specific analysis needs. The tab also includes additional visualizations like a time-series analysis graph to track changes over time, and a scatter plot to explore correlations between metrics within selected geographic parameters. This tab is pivotal for businesses aiming to understand and analyze the spatial distribution of their customer base and sales trends, providing key insights for targeted marketing and regional sales strategies.

TAB: Data Download and Customization



Export

Payment Method	Product Category	Product Price	Quantity	Customer Age	Total Purchase Amount	Latitude	Longitude
Debit Card	Clothing	204.98012126911948	3	23	371.8167067230015	38.40991527281875	-96.4256578642646
PayPal	Clothing	389.505467893259	5	48	752.5622269065594	38.792073340042066	-83.236490091631
PayPal	Clothing	79.06916349660993	5	22	1105.6894033226652	34.648783998769204	-98.741860774988
Credit Card	Electronics	483.99552299812126	4	50	1282.8744897421552	39.62407990582155	-80.9119823886034
Credit Card	Books	31.0240332972535	7	63	188.12956913765163	30.12877261910939	-98.598224722763
PayPal	Electronics	353.4192682293915	7	47	1663.1499363370917	39.47514364704864	-97.003689929578
Debit Card	Electronics	183.57016269535876	6	45	441.82210400706015	34.56599130989644	-81.323200237280
Credit Card	Electronics	68.93589903994013	3	47	1053.3585488186648	36.117915304466806	-92.673456780534
Credit Card	Books	443.66070709378505	5	29	899.8721366963922	37.71288956149331	-98.828651859923
Credit Card	Clothing	59.12335459716581	5	51	1212.6934554278319	35.515485070832774	-95.601200524377
Credit Card	Books	18.35963826695778	3	56	208.29780799699273	35.91774110192421	-96.726538515170
Credit Card	Electronics	483.8569098578376	6	29	1470.1615756146246	32.184133818319644	-94.794949230994
Debit Card	Electronics	311.37843011119713	8	28	1575.2000004093886	36.83489439671726	-92.575652237710
Credit Card	Electronics	280.69513890590025	8	43	1494.149058259524	39.830597536487005	-88.316030611379
Credit Card	Books	465.3529190691939	4	68	962.3724248031718	32.34963886128957	-94.636538022874

The "Data Download & Customization" tab in a Dash application typically serves as a user-centric feature that allows for both the exploration and extraction of data. This tab offers interactive components for users to customize data visualizations according to their specific requirements. Features such as range sliders for selecting quantities and dropdown menus for choosing product categories or payment methods are common tools for data filtering and segmentation. Users can interact with these elements to refine the data displayed in various charts, such as customizable bar charts or segmentation analysis charts.

Additionally, this tab includes functionality for downloading the data, possibly in formats like CSV, allowing users to work with the data offline or use it for further analysis in other tools. The presence of a data table in the layout suggests that users can view detailed datasets in a tabular format, which could also include interactive features like tooltips for enhanced data comprehension.

TAB: Advanced Visualizations

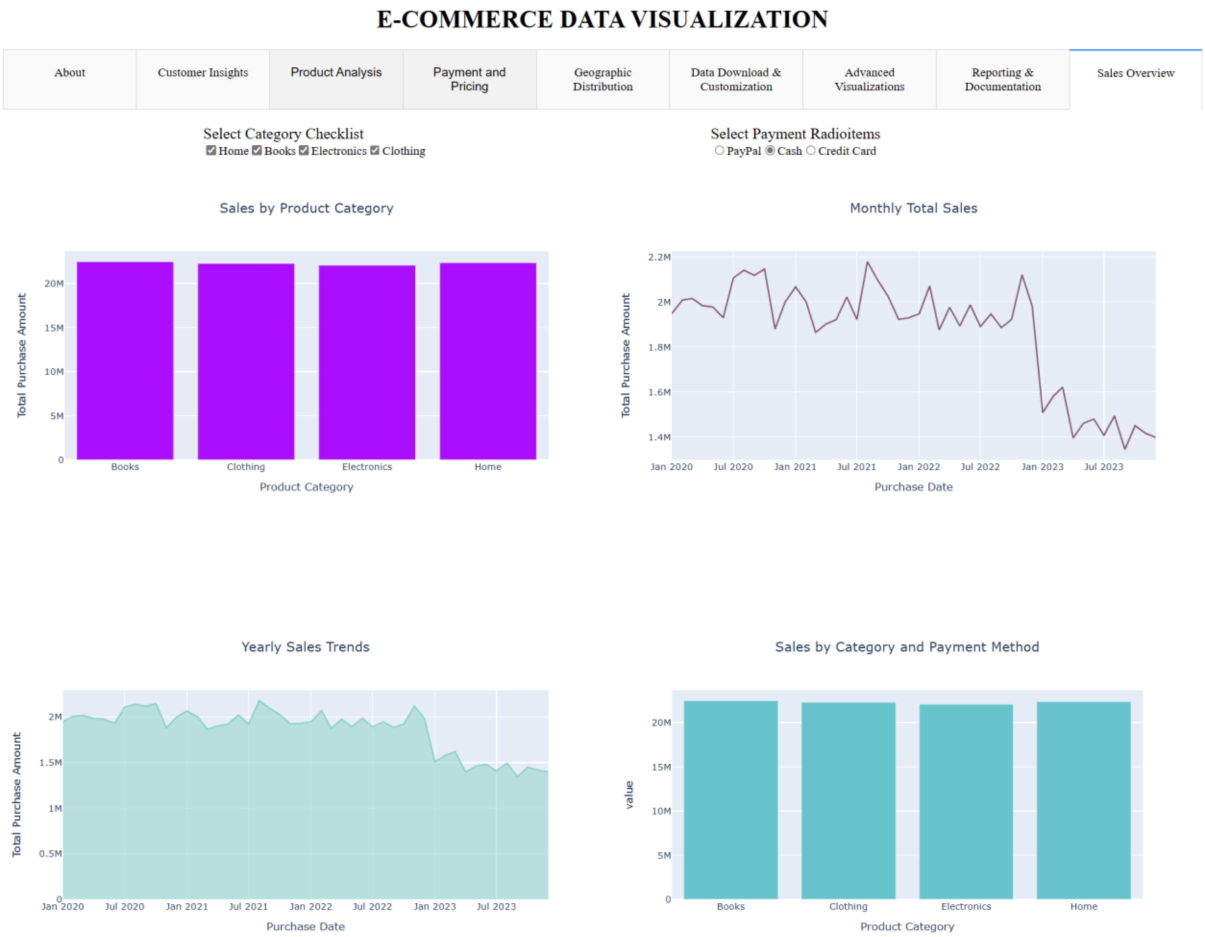
The "Advanced Visualizations" tab in the Dash application is designed to offer sophisticated and interactive data visualizations that provide deeper insights into the dataset. This tab features a variety of complex plots, such as enhanced scatter plots, violin plots, and interactive time-series graphs.

Users can customize these visualizations through dropdown menus, allowing them to select different variables for the x and y axes of scatter plots or to choose the data dimensions to be displayed in violin plots. These interactive elements enable users to explore various aspects of the data, such as relationships between different variables, distribution patterns, and trends over time. Additionally, this tab includes options for segmenting the data by categories like payment methods or product categories, enabling a more granular analysis. The use of advanced plotting techniques like 3D scatter plots or interactive time-series analyses suggests a focus on providing users with a dynamic and comprehensive view of the data.

TAB: Reporting and Documentation:

The "Reporting & Documentation" tab in a Dash application is designed to be a comprehensive resource for users to generate and access detailed reports and documentation on the data analysis presented. This section includes features for customizing and creating reports, with interactive elements like sliders and dropdowns that allow users to filter and tailor the data to their specific needs. Additionally, this tab offers in-depth documentation, explaining the methodologies, data sources, and interpretations of the analyses, enhancing user understanding and applicability of the data. Export functionality is also a feature, providing options to download reports in various formats for external use or sharing.

TAB: Sales Overview



The "Sales Overview" tab in a Dash application is tailored to provide a comprehensive view of sales-related data and trends. It features a user-friendly interface with interactive elements such as checklists and radio items for selecting product categories and payment methods. This tab displays a variety of graphs, including bar charts, line charts, area charts, and stacked bar charts, each offering different perspectives on sales data. For instance, bar charts show sales by product category, while line and area charts reveal temporal trends in sales. Stacked bar charts provide insights into sales segmented by different payment methods or product categories. The combination of these visualizations would allow users to quickly grasp overall sales performance, understand how different categories contribute to sales, and identify emerging trends. This tab is likely essential for stakeholders who need to monitor sales performance and make data-driven decisions to optimize sales strategies.

MINIMUM ITEMS USED:

a. Checklist: Used for selecting multiple product categories or gender options in customer insights and sales overview tabs, allowing users to filter data based on their preferences.

b. Dropdown: Implemented for choosing product categories, payment methods, or metrics in tabs like "Customer Insights," "Product Analysis," and "Reporting & Documentation," enabling targeted data analysis.

c. Graph: Utilized extensively across all tabs like "Sales Overview" and "Geographic Distribution" for visualizing sales data, customer demographics, and geographical trends using various chart types.

d. Loading: Integrated into tabs with complex visualizations or large datasets, such as "Geographic Distribution," to indicate data is being processed or loaded.

e. Download: Featured in the " Data Download & Customization " tab, allowing users to export generated reports or data tables for offline use or sharing.

f. RadioItems: Used in the "Payment and Pricing" and "Geographic Distribution" tabs for selecting specific data views or metrics, providing a simple way to switch between different data perspectives.

g. RangeSlider: Employed in tabs like "Geographic Distribution" for filtering data based on age range and in "Data Download & Customization" for selecting quantity ranges, offering users a dynamic way to refine the data displayed.

h. Slider: In "Data Download & Customization," a slider adjusts visualization parameters like histogram bin sizes, allowing for more detailed data exploration.

i. Tab: Organizes the application into different sections like "Customer Insights," "Product Analysis," and more, making navigation intuitive and content-focused.

j. Textarea: Tabs like About for users to input comments or feedback, facilitating user interaction and data annotation.

k. Tooltips: Used in Data Download and Customization where the user can hover the data.

l. Br: Utilized to add spacing or line breaks in the layout, improving readability and visual appeal.

m. Div: The fundamental building block for structuring the layout and organizing content in all tabs.

n. Figure: Used in "Advanced Visualizations" to present complex graphical data, offering a more detailed view of the analysis.

o. H1-H6: Employed for titles and headings in tabs

p. Header: Used at the top of the application or tabs like "Sales Overview" for titles and introductory text.

q. Img: In the "About" tab display images, like the e-commerce dataset schema or branding elements.

r. Label: Accompanies input elements like checkboxes, sliders, and dropdowns across all tabs, providing context and enhancing usability.

s. Title: Essential for naming graphs and sections within tabs, helping users quickly understand the content and purpose of each part of the application.

Conclusion:

From the various graphs created, I learned about the intricate relationships between different variables, customer purchasing patterns, and potential areas for business improvement. The visualizations, ranging from line and bar charts to heatmaps and PCA, provided a multi-dimensional view of the dataset, revealing trends, distributions, and correlations that are critical for data-driven decision-making. The Python dashboard, with its interactive and intuitive design, significantly enhances user engagement and understanding of the data. It enables users to easily access, explore, and interpret complex datasets, thereby facilitating informed business decisions.

Firstly, it offers interactive visualizations, allowing users to engage directly with the data through dynamic graphs and charts. This interactivity not only makes the data more accessible but also allows for a deeper, more intuitive understanding of complex patterns and trends. Additionally, the dashboard's capability for real-time data analysis means it can reflect up-to-date information, crucial for timely decision-making. Its customizable nature enables users to tailor the dashboard to their specific needs, focusing on the most relevant data aspects. This customization ranges from selecting particular data points to comparing different datasets. Moreover, the comprehensive overview provided by aggregating data from various sources into a single platform helps users grasp the big picture while retaining the ability to drill down into finer details. Importantly, this enhanced data accessibility democratizes data analysis, making it approachable even for those without extensive backgrounds in the field. Ultimately, the dashboard serves as a powerful tool for efficient and informed decision-making, streamlining the process of data exploration and analysis, and highlighting significant patterns and outliers within the dataset. The app is user-friendly to the users. The functionality is good since it is reliable and fast.

Appendix

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as stats
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import kde
from scipy.stats import gaussian_kde
from pandas.api.types import CategoricalDtype
import numpy as np
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from prettytable import PrettyTable

# Your code using isinstance(dtype, CategoricalDtype)

df = pd.read_csv('ecommerce_customer_data_large.csv')

# Preprocess the data if necessary
#1. Pre-processing dataset
data = pd.read_csv('ecommerce_customer_data_large.csv')
print("First few rows before cleaning:")
print(data.head())
numerical_columns = ['Product Price', 'Quantity', 'Total Purchase Amount',
'Customer Age', 'Age']
for column in numerical_columns:
    if data[column].isnull().sum() > 0:  # Check if there are any missing
values
        data[column].fillna(data[column].median(), inplace=True)  # Replace
with median (or mean)

# For categorical columns: impute with mode
categorical_columns = ['Customer ID', 'Purchase Date', 'Product Category',
'Payment Method',
                       'Customer Name', 'Gender', 'Returns', 'Churn']
for column in categorical_columns:
    if data[column].isnull().sum() > 0:  # Check if there are any missing
values
        data[column].fillna(data[column].mode()[0], inplace=True)
    # Replace with mode

# Display the first few rows of the cleaned dataset
print("First few rows after cleaning:")
print(data.head())

# Display the corresponding statistics
print("\nStatistics of the cleaned dataset:")
print(data.describe().round(2))

#2. Outlier detection & removal
numerical_columns = ['Product Price', 'Quantity', 'Total Purchase Amount',
'Customer Age', 'Age']

# Iterate over each numerical column
for column in numerical_columns:
    # Calculate Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
```

```
    # Calculate the IQR
    IQR = Q3 - Q1

    # Determine the outlier thresholds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Filter out the outliers
    data = data[(data[column] >= lower_bound) & (data[column] <=
upper_bound)]

# Display the result
print("Data Size After Outlier Removal:", data.shape)
for column in numerical_columns:
    plt.figure(figsize=(8, 6))
    plt.boxplot(data[column])
    plt.title(f'Box Plot of {column}',fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
    plt.ylabel(column,fontname='serif', color='darkred')
    plt.show()

#3. Principal Component Analysis (PCA)
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Select only numerical columns for PCA
numerical_columns = data.select_dtypes(include=[np.number]).columns
data_numerical = data[numerical_columns]

# Standardize the data
scaler = StandardScaler()
data_std = scaler.fit_transform(data_numerical)

# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% of the variance, adjust as
needed
principal_components = pca.fit_transform(data_std)

# Check explained variance
explained_variance = pca.explained_variance_ratio_

# Check condition number and singular values
condition_number = np.linalg.cond(pca.components_)
singular_values = pca.singular_values_

print("Explained Variance Ratio:", explained_variance.round(2))
print("Condition Number:", condition_number.round(2))
print("Singular Values:", singular_values.round(2))

#4 Normality test
from scipy.stats import normaltest
import statsmodels.api as sm
import matplotlib.pyplot as plt
prices = df['Product Price']
# D'Agostino's K-squared test
d_agostino_test , p_value= normaltest(prices)

# Q-Q plot
sm.qqplot(prices, line ='45', fit=True)
plt.title('Q-Q Plot for Product Prices',fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
plt.xlabel('Theoretical Quantities',fontname='serif', color='darkred')
```

```
plt.ylabel('Sample Quantiles',fontname='serif', color='darkred')
plt.show()
print(f"D'Agostino's K-squared test statistic: {d_agostino_test:.2f}")
print(f"P-value: {p_value:.2f}")

##5. Data transformation
from scipy.stats import boxcox

# Applying Box-Cox transformation to 'Product Price'
# The Box-Cox transformation requires all values to be positive, so we
check for this condition
if all(prices > 0):
    prices_transformed, _ = boxcox(prices)
else:
    prices_transformed = None

# If transformation was successful, plot the transformed data
if prices_transformed is not None:
    # Q-Q plot for transformed data
    sm.qqplot(prices_transformed, line ='45', fit=True)
    plt.title('Q-Q Plot for Box-Cox Transformed Product
Prices',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
    plt.xlabel('Theoretical Quantiles', fontname='serif', color='darkred')
    plt.ylabel('Sample Quantiles', fontname='serif', color='darkred')
    plt.show()

    # Histogram for transformed data
    plt.figure(figsize=(12, 6))
    plt.hist(prices_transformed, bins=30, edgecolor='black')
    plt.title('Histogram of Box-Cox Transformed Product
Prices',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
    plt.xlabel('Transformed Prices', fontname='serif', color='darkred')
    plt.ylabel('Frequency', fontname='serif', color='darkred')
    plt.show()
else:
    print("Box-Cox transformation could not be applied as not all prices
are greater than 0.")

##6. Heatmap & Pearson correlation coefficient matrix
numeric_columns = ['Product Price', 'Quantity', 'Total Purchase Amount',
'Customer Age', 'Returns', 'Age', 'Churn']

# Calculating the correlation matrix for the numerical columns
correlation_matrix = df[numeric_columns].corr().round(2)

# Plotting the heatmap using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Heatmap of Pearson Correlation Coefficient
Matrix',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.show()
numeric_columns = ['Product Price', 'Quantity', 'Total Purchase Amount',
'Customer Age', 'Returns', 'Age', 'Churn']

# Calculating the correlation matrix for the numerical columns
correlation_matrix = df[numeric_columns].corr()

# Display the correlation matrix rounded to two decimal places
print(correlation_matrix.round(2))

# Display numerical features in a PrettyTable with two-point precision for
```

```
mean, median, and standard deviation
table_stats = PrettyTable(["Feature", "Mean", "Median", "Correlation", "Std
Dev","Variance"])
for feature in numerical_columns:
    table_stats.add_row([
        feature,
        f"{df[feature].mean():.2f}",
        f"{df[feature].median():.2f}",
        f"{df['Churn'].corr(df[feature]):.2f}",  # Replace 'Churn' with
your target column
        f"{df[feature].std():.2f}",
        f"{df[feature].var():.2f}"
    ])

table_correlation = PrettyTable(["Feature1", "Feature2", "Correlation",
"Inference"])

for i in range(len(numerical_columns)):
    for j in range(i + 1, len(numerical_columns)):
        feature1 = numerical_columns[i]
        feature2 = numerical_columns[j]
        correlation = df[feature1].corr(df[feature2])

        # Define your inference based on correlation thresholds
        if correlation > 0.7:
            inference = "Strong positive correlation"
        elif correlation < -0.7:
            inference = "Strong negative correlation"
        elif 0.3 <= correlation <= 0.7:
            inference = "Moderate positive correlation"
        elif -0.7 <= correlation <= -0.3:
            inference = "Moderate negative correlation"
        else:
            inference = "Weak or no correlation"

        table_correlation.add_row([feature1, feature2,
f"{correlation:.2f}", inference])

print("Correlation Table:")
print(table_correlation)

print("Statistics Table:")
print(table_stats)

# Convert 'Purchase Date' to a datetime format
df['Purchase Date'] = pd.to_datetime(df['Purchase Date'])
# Extract the 'Year' from 'Purchase Date'
df['Year'] = df['Purchase Date'].dt.year

# # Displaying the first few rows to confirm the conversion
# print(df.head())
# Convert inf values to NaN before operating
df.replace([np.inf, -np.inf], np.nan, inplace=True)
# Your code that involves operations

# 1. Line Plot - Total Purchase Amount over Time
line_plot_data = df.groupby(df['Purchase Date'].dt.to_period("M"))['Total
Purchase Amount'].sum()
plt.figure(figsize=(10, 6))
line_plot_data.plot(kind='line')
plt.title('Monthly Total Purchase Amount',fontdict={'fontname': 'serif',
```

```
'color': 'blue', 'size': 16})
plt.xlabel('Date',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.show()


stacked_bar_data = df.groupby(['Year', 'Product Category'])['Total Purchase
Amount'].sum().unstack()

# Plotting the stacked bar plot

stacked_bar_data.plot(kind='bar', stacked=True)
plt.title('Total Purchase Amount by Product Category Over
Years',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 13})
plt.xlabel('Year',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.xticks(rotation=0)  # Rotate x-axis labels if needed
plt.tight_layout()
plt.show()

grouped_bar_data_alt = df.groupby(['Payment Method', 'Product
Category'])['Total Purchase Amount'].sum().unstack()

# Selecting a palette with shades of green
green_palette = sns.color_palette("Greens",
n_colors=len(grouped_bar_data_alt.columns))

# Plotting the grouped bar plot with the green palette
grouped_bar_data_alt.plot(kind='bar', color=green_palette)
plt.title('Total Purchase Amount by Payment Method and Product
Category',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 11})
plt.xlabel('Payment Method',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.legend(title='Product Category')  # Set the font size to 'small'
plt.xticks(rotation=0)  # Rotate x-axis labels if needed
plt.tight_layout()
plt.show()

# Creating the count plot for 'Product Category'
plt.figure(figsize=(10, 6))
sns.countplot(x='Product Category', data=df, palette='Set2')
plt.title('Count of Products by Category',fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
plt.xlabel('Product Category',fontname='serif', color='darkred')
plt.ylabel('Count',fontname='serif', color='darkred')
# plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 4. Pie Chart
pie_data_payment = df['Payment Method'].value_counts()

# Creating the pie chart
plt.figure(figsize=(8, 8))
pie_data_payment.plot(kind='pie', autopct='%1.2f%%')
plt.title('Distribution of Payment Methods',fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
plt.ylabel('',fontname='serif', color='darkred')  # Hiding the y-axis label
for clarity
plt.legend()
plt.tight_layout()
```

```
plt.show()

# 5. Dist Plot
df_sample = df.sample(n=1000, random_state=42)

# Creating a distribution plot for 'Product Price'
sns.displot(df_sample['Product Price'], kde=True, color='green')
plt.title('Distribution of Product Price ',fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
plt.xlabel('Product Price',fontname='serif', color='darkred')
plt.ylabel('Density',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# 6. Pair Plot
# Sample 1000 rows from the DataFrame
# Sample 1000 rows from the DataFrame
df_sample = df.sample(n=1000, random_state=42)
# Drop non-numeric columns for pair plotting
numeric_columns = df_sample.select_dtypes(include=['float64',
'int64']).columns
numeric_df_sample = df_sample[numeric_columns].copy()  # Use copy to avoid
SettingWithCopyWarning
# Add 'Churn' column back for coloring
numeric_df_sample.loc[:, 'Churn'] = df_sample['Churn'].copy()
# Create pair plot
sns.set(style="ticks")
pair_plot = sns.pairplot(numeric_df_sample, hue='Churn', palette='husl')
# Add title to the pair plot
pair_plot.fig.suptitle("Pair Plot of Numeric Features with Churn",
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.tight_layout()
plt.show()

# 7. Heatmap with cbar
# Exclude non-numeric columns before creating the heatmap
numeric_columns = df.select_dtypes(include=['float64', 'int64'])
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_columns.corr(), fmt='.2f', annot=True, cmap='coolwarm',
cbar=True)
plt.title("Heatmap with Correlation Coefficients", fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
plt.tight_layout()
plt.show()

# 8. Histogram plot with KDE
df_sample = df.sample(n=1000, random_state=42)

# Creating the histogram with KDE for 'Product Price'
plt.figure(figsize=(10, 6))
sns.histplot(df_sample['Product Price'], kde=True, color='purple')
plt.title('Histogram with KDE for Product Price ',fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Product Price',fontname='serif', color='darkred')
plt.ylabel('Frequency',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# 9. QQ-plot
plt.figure(figsize=(10, 6))
stats.probplot(df['Total Purchase Amount'], dist="norm", plot=plt)
```

```
plt.title('QQ Plot of Total Purchase Amount',fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
plt.xlabel('Theoretical Quantiles',fontname='serif', color='darkred')
plt.ylabel('Ordered Values',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()


# KDE plot with fill, alpha, palette, and linewidth
df_sample = df.sample(n=1000, random_state=42)

# Creating the KDE plot for 'Product Price'
plt.figure(figsize=(10, 6))
sns.kdeplot(df_sample['Product Price'], fill=True, alpha=0.6,
color='darkgreen', linewidth=3)
plt.title('KDE Plot of Product Price ',fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
plt.xlabel('Product Price',fontname='serif', color='darkred')
plt.ylabel('Density',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

df_sample = df.sample(n=1000, random_state=42)

# Creating an lmplot (regression plot) with a scatter representation and a
regression line
sns.lmplot(x='Customer Age', y='Total Purchase Amount', data=df_sample,
scatter=True, line_kws={'color': 'red'},height=8)
plt.title("Regression Plot: Customer Age vs Total Purchase
Amount",fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Customer Age',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# Multivariate Box or Boxen plot
plt.figure(figsize=(10, 6))
sns.boxenplot(x='Product Category', y='Total Purchase Amount', data=df)
plt.title("Boxen Plot: Product Category vs Total Purchase
Amount",fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Product Category',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# Area plot for yearly total purchase amount
yearly_sales = df.groupby('Year')['Total Purchase Amount'].sum()
yearly_sales.plot(kind='area')
plt.title("Area Plot of Yearly Total Purchase Amount",fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Year',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# Violin plot
plt.figure(figsize=(10, 6))
sns.violinplot(x='Product Category', y='Product Price', data=df)
plt.title("Violin Plot of Product Price by Category",fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Product Category',fontname='serif', color='darkred')
```

```
plt.ylabel('Product Price',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# Joint plot with KDE and scatter representation
sns.jointplot(x='Customer Age', y='Total Purchase Amount', data=df,
kind='kde', space=0, color='g')
plt.title("joint plot with kde",fontdict={'fontname': 'serif', 'color':
'blue', 'size': 16})
plt.xlabel('Customer Age',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# Creating a joint plot with scatter representation of the sample data
joint_scatter = sns.jointplot(x='Customer Age', y='Total Purchase Amount',
data=df_sample, kind='scatter', color='b')
# Adjusting the title, labels, and layout
joint_scatter.fig.suptitle("Joint plot of Customer Age and Total Purchase
Amount", fontdict={'fontname': 'serif', 'color': 'blue', 'size': 20})
joint_scatter.set_axis_labels('Customer Age', 'Total Purchase Amount',
fontname='serif', color='darkred')
# Adjust layout
plt.tight_layout()
plt.show()

# Rug plot for Customer Age distribution
df_sample = df.sample(n=1000, random_state=42)

# Creating a KDE plot with an overlaid rug plot for 'Product Price'
plt.figure(figsize=(10, 6))
sns.kdeplot(df_sample['Product Price'], fill=True, color="blue", alpha=0.6,
linewidth=2)

sns.rugplot(df_sample['Product Price'], color="black")
plt.title("KDE and Rug Plot of Product Price ",fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Product Price',fontname='serif', color='darkred')
plt.ylabel('Density',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

#3D Scatter plot
df_sampled = df.sample(frac=0.05, random_state=42)
# Creating a new figure for the 3D plot
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
# Generating a scatter plot with a colormap reflecting the 'Customer Age'
sc = ax.scatter(df_sampled['Product Price'], df_sampled['Quantity'],
df_sampled['Customer Age'],
                c=df_sampled['Customer Age'], cmap='viridis', alpha=0.6)

# Setting the title and labels
ax.set_title('3D Scatter Plot ',fontdict={'fontname': 'serif', 'color':
'blue', 'size': 16})
ax.set_xlabel('Product Price',fontname='serif', color='darkred')
ax.set_ylabel('Quantity',fontname='serif', color='darkred')
ax.set_zlabel('Customer Age',fontname='serif', color='darkred')
# Adding a color bar to show the relationship between color and 'Customer
Age'
cbar = plt.colorbar(sc)
```

```
cbar.set_label('Customer Age')
# Adjusting the view angle
ax.view_init(elev=30, azim=120)  # Adjusting the elevation and azimuth for
better visibility
plt.tight_layout()
plt.show()

# Exclude non-numeric columns before creating the cluster map
numeric_columns = df.select_dtypes(include=['float64', 'int64'])
sns.clustermap(numeric_columns.corr(), fmt='.2f', cmap='coolwarm',
annot=True, figsize=(10, 8))
plt.suptitle('Cluster Map: Correlation Heatmap', fontdict={'fontname':
'serif', 'color': 'blue', 'size': 20}, y=0.95)
plt.tight_layout()
plt.show()




# Taking a random sample from the dataset to make the plot clearer and more
manageable
df_sample = df.sample(n=1000, random_state=42)

# Generating the Hexbin plot
plt.figure(figsize=(10, 6))
plt.hexbin(df_sample['Customer Age'], df_sample['Total Purchase Amount'],
gridsize=30, cmap='Reds')
plt.colorbar(label='Count in Bin')
plt.title('Hexbin Plot of Customer Age vs Total Purchase
Amount',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Customer Age',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

# Selecting a subset of the data to make the plot more manageable
subset = df.sample(n=1000, random_state=42)  # Adjust the sample size as
needed

# Creating a swarm plot with the subset
plt.figure(figsize=(8, 6))
sns.swarmplot(x='Product Category', y='Total Purchase Amount', data=subset)
plt.title('Swarm Plot of Total Purchase Amount by Product Category
',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Product Category',fontname='serif', color='darkred')
plt.ylabel('Total Purchase Amount',fontname='serif', color='darkred')
plt.tight_layout()
plt.show()

#Contour
df_sample = df.sample(n=500, random_state=42)
# Selecting 'Product Price' and 'Quantity' for the contour plot
x_sample = df_sample['Product Price']
y_sample = df_sample['Quantity']
# Use gaussian_kde from scipy.stats
k_sample = gaussian_kde([x_sample, y_sample])
# Create a meshgrid for the contour plot
xi_sample, yi_sample =
np.mgrid[x_sample.min():x_sample.max():x_sample.size**0.5*1j,
y_sample.min():y_sample.max():y_sample.size**0.5*1j]
zi_sample = k_sample(np.vstack([xi_sample.flatten(), yi_sample.flatten()]))
```

```
# Creating the contour plot for the sample
plt.figure(figsize=(8, 6))
plt.pcolormesh(xi_sample, yi_sample, zi_sample.reshape(xi_sample.shape),
shading='auto', cmap=plt.cm.viridis)
# Plot contours without lines
contour = plt.contour(xi_sample, yi_sample,
zi_sample.reshape(xi_sample.shape), colors='k', linewidths=0)
plt.title('Contour Plot of Product Price vs Quantity',
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Product Price', fontname='serif', color='darkred')
plt.ylabel('Quantity', fontname='serif', color='darkred')
plt.colorbar(label='Density')
plt.show()


#Strip
plt.figure(figsize=(12, 6))
sns.stripplot(x='Payment Method', y='Product Price', data=df_sample,
jitter=True, palette="plasma")
plt.title("Strip Plot of Product Price by Payment Method
",fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
plt.xlabel('Payment Method',fontname='serif', color='darkred')
plt.ylabel('Product Price',fontname='serif', color='darkred')
plt.show()


# Creating a figure with multiple subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 12))

# Subplot 1: Bar chart of Average Product Price by Payment Method
avg_price_payment = df.groupby('Payment Method')['Product Price'].mean()
avg_price_payment.plot(kind='bar', ax=axes[0, 0], color='skyblue')
axes[0, 0].set_title('Average Product Price by Payment Method',
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
axes[0, 0].set_xlabel('Payment Method', fontname='serif', color='darkred')
axes[0, 0].set_ylabel('Average Product Price', fontname='serif',
color='darkred')

# Subplot 2: Pie chart of Customer Churn
churn_distribution = df['Churn'].value_counts()
pie_chart = churn_distribution.plot(kind='pie', autopct='%1.1f%%',
ax=axes[0, 1], startangle=90)

# Add labels
labels = churn_distribution.index
pie_chart.set_title('Customer Churn Distribution', fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
pie_chart.set_ylabel('', fontname='serif', color='darkred')  # Hide the y-
label
pie_chart.legend(labels, loc='upper right', bbox_to_anchor=(1.2, 1))

# Subplot 3: Bar chart of Total Quantity Sold by Year
quantity_year = df.groupby('Year')['Quantity'].sum()
quantity_year.plot(kind='bar', ax=axes[1, 0], color='orange')
axes[1, 0].set_title('Total Quantity Sold by Year', fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
axes[1, 0].set_xlabel('Year', fontname='serif', color='darkred')
axes[1, 0].set_ylabel('Total Quantity', fontname='serif', color='darkred')

# Subplot 4: Box plot of Product Price by Product Category
sns.boxplot(x='Product Category', y='Product Price', data=df, ax=axes[1,
```

```
1], palette='viridis')
axes[1, 1].set_title('Product Price Distribution by Category',
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
axes[1, 1].set_xlabel('Product Category', fontname='serif',
color='darkred')
axes[1, 1].set_ylabel('Product Price', fontname='serif', color='darkred')

# Adjusting layout for better visibility
plt.tight_layout()
axes[0, 0].tick_params(axis='x', rotation=0)
axes[1, 0].tick_params(axis='x', rotation=0)
axes[1, 1].tick_params(axis='x', rotation=0)
plt.show()

#Supplot 2
df_sample = df.sample(n=500, random_state=42)  # Sample size of 500

# Creating a figure with multiple subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 12))

# Subplot 1: Bar Chart of Average Total Purchase Amount by Gender
avg_purchase_by_gender = df_sample.groupby('Gender')['Total Purchase
Amount'].mean()
avg_purchase_by_gender.plot(kind='bar', ax=axes[0, 0], color='lightblue')
axes[0, 0].set_title('Avg Purchase Amount by Gender',fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
axes[0, 0].set_xlabel('Gender',fontname='serif', color='darkred')
axes[0, 0].set_ylabel('Average Purchase Amount',fontname='serif',
color='darkred')

# Subplot 2: Pie Chart of Churn Distribution
sns.histplot(df_sample['Customer Age'], bins=20, ax=axes[0, 1],
color='green', kde=True)
axes[0, 1].set_title('Histogram of Customer Age',fontdict={'fontname':
'serif', 'color': 'blue', 'size': 16})
axes[0, 1].set_xlabel('Age',fontname='serif', color='darkred')
axes[0, 1].set_ylabel('Frequency',fontname='serif', color='darkred')

# Subplot 3: Box Plot of Product Price by Gender
sns.boxplot(x='Gender', y='Product Price', data=df_sample, ax=axes[1, 0],
palette='pastel')
axes[1, 0].set_title('Product Price Distribution by
Gender',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
axes[1, 0].set_xlabel('Gender',fontname='serif', color='darkred')
axes[1, 0].set_ylabel('Product Price',fontname='serif', color='darkred')

# Subplot 4: Scatter Plot of Age vs Total Purchase Amount
sns.scatterplot(x='Customer Age', y='Total Purchase Amount',
data=df_sample, hue='Gender', ax=axes[1, 1])
axes[1, 1].set_title('Age vs Total Purchase Amount by
Gender',fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
axes[1, 1].set_xlabel('Customer Age',fontname='serif', color='darkred')
axes[1, 1].set_ylabel('Total Purchase Amount',fontname='serif',
color='darkred')

# Adjusting layout for better visibility
plt.tight_layout()
axes[0, 0].tick_params(axis='x', rotation=0)
axes[1, 0].tick_params(axis='x', rotation=0)
axes[1, 1].tick_params(axis='x', rotation=0)
plt.show()
```

```
df_sample = df.sample(n=500, random_state=42)

# Creating a figure with 3 rows and 2 columns of subplots
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14, 18))

# Subplot 1: Distribution of Total Purchase Amount
sns.histplot(df_sample['Total Purchase Amount'], kde=True, bins=30,
ax=axes[0, 0], color='blue')
axes[0, 0].set_title('Distribution of Total Purchase Amount',
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
axes[0, 0].set_xlabel('Total Purchase Amount', fontname='serif',
color='darkred')
axes[0, 0].set_ylabel('Frequency', fontname='serif', color='darkred')

# Subplot 2: Proportion of Different Genders
gender_distribution = df_sample['Gender'].value_counts()
pie_chart = gender_distribution.plot(kind='pie', autopct='%1.1f%%',
ax=axes[0, 1], startangle=90)

# Add labels
labels = gender_distribution.index
pie_chart.set_title('Proportion of Different Genders',
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
pie_chart.set_ylabel('', fontname='serif', color='darkred')  # Hide the y-
label
pie_chart.legend(labels, loc='upper right', bbox_to_anchor=(1.2, 1))

# Subplot 3: Average Quantity by Product Category
avg_quantity_category = df_sample.groupby('Product
Category')['Quantity'].mean()
avg_quantity_category.plot(kind='bar', ax=axes[1, 0], color='green')
axes[1, 0].set_title('Average Quantity by Product Category',
fontdict={'fontname': 'serif', 'color': 'blue', 'size': 16})
axes[1, 0].set_xlabel('Product Category', fontname='serif',
color='darkred')
axes[1, 0].set_ylabel('Average Quantity', fontname='serif',
color='darkred')

# Subplot 4: Returns by Gender
returns_by_gender = df_sample.groupby('Gender')['Returns'].sum()
returns_by_gender.plot(kind='bar', ax=axes[1, 1], color='red')
axes[1, 1].set_title('Returns by Gender', fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
axes[1, 1].set_xlabel('Gender', fontname='serif', color='darkred')
axes[1, 1].set_ylabel('Total Returns', fontname='serif', color='darkred')

# New Subplot 5: Line Plot of Quantity Over Time
quantity_over_time = df_sample.groupby(df_sample['Purchase
Date'].dt.to_period("M"))['Quantity'].sum()
quantity_over_time.plot(kind='line', ax=axes[2, 0], color='purple')
axes[2, 0].set_title('Quantity Over Time', fontdict={'fontname': 'serif',
'color': 'blue', 'size': 16})
axes[2, 0].set_xlabel('Date', fontname='serif', color='darkred')
axes[2, 0].set_ylabel('Quantity', fontname='serif', color='darkred')

# New Subplot 6: Correlation Heatmap
sns.heatmap(df_sample[['Product Price', 'Quantity', 'Total Purchase
Amount', 'Customer Age']].corr(), annot=True, fmt='.2f', cmap='coolwarm',
ax=axes[2, 1])
axes[2, 1].set_title('Correlation Heatmap', fontdict={'fontname': 'serif',
```

```
'color': 'blue', 'size': 16})

# Adjusting layout for better visibility
plt.tight_layout()
axes[0, 0].tick_params(axis='x', rotation=0)
axes[1, 0].tick_params(axis='x', rotation=0)
axes[1, 1].tick_params(axis='x', rotation=0)
axes[2, 0].tick_params(axis='x', rotation=0)
axes[2, 1].tick_params(axis='x', rotation=10)
axes[2, 1].tick_params(axis='y', rotation=0)
plt.show()

import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import dash
from dash import html, dcc, Input, Output, State
import plotly.express as px
import plotly.graph_objects as go
from dash.dash_table.Format import Group
from dash import dash_table
from dash.dash_table.Format import Group
from pandas.api.types import CategoricalDtype
import base64

# Load your dataset
df = pd.read_csv('ecommerce_customer_data_large.csv')

# Assuming 'Purchase Date' is in a standard format, replace it accordingly
df['Purchase Date'] = pd.to_datetime(df['Purchase Date'], errors='coerce')
# Coerce invalid dates to NaT

# Handle missing values, if needed
df = df.dropna(subset=['Purchase Date'])

# Convert to period and then to string
# df['Purchase Date'] = df['Purchase Date'].dt.to_period("M").astype(str)
df['Purchase Date'] = pd.to_datetime(df['Purchase Date'], format="%d-%m-%Y
%H:%M").dt.to_period("M").astype(str)

# Generating a sample dataset
np.random.seed(0)
sample_size = 200
df2 = pd.DataFrame({
    'Gender': np.random.choice(['Male', 'Female', 'Other'], sample_size),
    'Customer Age': np.random.randint(18, 70, sample_size),
    'Total Purchase Amount': np.random.uniform(100, 1000, sample_size),
    'Product Category': np.random.choice(['Electronics', 'Clothing', 'Home
Decor', 'Books', 'Groceries'], sample_size),
    'Product Price': np.random.uniform(10, 500, sample_size),
    'Purchase Date': [datetime.now() - timedelta(days=i) for i in
range(sample_size)]
})

payment_methods = ['Credit Card', 'Debit Card', 'PayPal']
product_categories = ['Electronics', 'Clothing', 'Books']

sample_df = pd.DataFrame({
    'Payment Method': np.random.choice(payment_methods, sample_size),
    'Product Category': np.random.choice(product_categories, sample_size),
    'Product Price': np.random.uniform(10, 500, sample_size),
```

```
    'Quantity': np.random.randint(1, 10, sample_size),
    'Customer Age': np.random.randint(18, 70, sample_size),
    'Total Purchase Amount': np.random.uniform(100, 2000, sample_size),
    'Latitude': np.random.uniform(30, 40, sample_size),
    'Longitude': np.random.uniform(-100, -80, sample_size)
})

sample_df2 = pd.DataFrame({
    'Payment Method': np.random.choice(payment_methods, sample_size),
    'Product Category': np.random.choice(product_categories, sample_size),
    'Product Price': np.random.uniform(10, 500, sample_size),
    'Quantity': np.random.randint(1, 10, sample_size),
    'Customer Age': np.random.randint(18, 70, sample_size),
    'Total Purchase Amount': np.random.uniform(100, 2000, sample_size),
    'Latitude': np.random.uniform(30, 40, sample_size),
    'Longitude': np.random.uniform(-100, -80, sample_size),
    'Segment': np.random.choice(['Segment 1', 'Segment 2', 'Segment 3'],
sample_size),
    'User Comments': ['' for _ in range(sample_size)],
    'Annotations': ['' for _ in range(sample_size)]
})

# Initialize the Dash app
app = dash.Dash(__name__)
server = app.server
image = 'Image.jpg'
image_base64 = base64.b64encode(open(image, 'rb').read()).decode('ascii')

tooltip_data = []
for row in sample_df.to_dict('records'):  # pylint: disable=not-an-iterable
    tooltip_item = {}
    for column, value in row.items():
        tooltip_item[column] = {'value': str(value), 'type': 'markdown'}
    tooltip_data.append(tooltip_item)
# App layout
app.layout = html.Div([
    html.Header([
        html.H1("E-COMMERCE DATA VISUALIZATION", style={'text-align':
'center'}),
        dcc.Tabs([

            dcc.Tab(label='About', children=[
                html.Div([
                    html.Figure([
                        # Image in the center

html.Img(src='data:image/png;base64,{}'.format(image_base64),
                                 style={'width': '25%px', 'margin': 'auto',
'margin-top': '1rem', 'display': 'block'}),
                    ]),
                    # Text below the image
                    html.Div([
                        html.P(
                            "The E-commerce Customer Behavior and Purchase
Dataset is a synthetic dataset generated using the Faker Python library,
designed to simulate a comprehensive e-commerce environment. It encompasses
various aspects of customer behavior and purchase history within a digital
marketplace. This dataset is intended for data analysis and predictive
modeling in the field of e-commerce, making it suitable for tasks like
customer churn prediction, market basket analysis, recommendation systems,
and trend analysis.",
```

```
                            style={'fontWeight': 'bold'}),
                        html.Br(),
                        html.P(
                            "The dataset includes essential columns such as
Customer ID, Customer Name, Customer Age, Gender, Purchase Date, Product
Category, Product Price, Quantity, Total Purchase Amount, Payment Method,
Returns (indicating whether the customer returned any products), and Churn
(indicating whether the customer has churned or not). These columns provide
valuable insights into customer demographics, buying habits, and
transaction details, making it a valuable resource for businesses and
researchers in the e-commerce industry.",
                            style={'fontWeight': 'bold'}),

                    ], style={'text-align': 'center', 'margin-top': '20px',
'overflow': 'hidden'}),
                    # Add Textarea
                    html.Div([
                        html.H4(
                            'Users could input their thoughts or insights
about specific products or trends they observe in the data.',
                            style={'font-size': '20px'}),
                        dcc.Textarea(
                            id='comments-textarea',
                            placeholder='Enter your FeedBack comments
here...',
                            style={'width': '100%', 'height': '100px',
'margin-top': '2%'}
                        ),
                    ], style={'width': '60%', 'margin-left': '20%',
'margin-top': '2%'}),

                ]),
            ]),  # About
            # TAB-2
            dcc.Tab(label='Customer Insights', children=[
                html.Div([
                    html.Div([
                        html.H2("Customer Insights", style={'text-align':
'center'}),
                        html.Label('Select Gender', style={'margin-top':
'1rem'}),
                        dcc.Checklist(
                            id='gender-checklist',
                            options=[{'label': gender, 'value': gender} for
gender in df2['Gender'].unique()],
                            value=df2['Gender'].unique().tolist(),
                            inline=True,
                            style={'margin-top': '1rem'}
                        ),
                        html.Div([
                            html.Label('Select Category ', style={'margin-
top': '1rem', 'font-size': '20px'}),
                            dcc.Dropdown(
                                id='category-dropdown',
                                options=[{'label': cat, 'value': cat} for
cat in df2['Product Category'].unique()],
                                value=df2['Product Category'].unique()[0],
                                multi=True,
                                style={'box-shadow': '0px 4px 6px rgba(0,
0, 0, 0.1)', 'margin-right': '15rem',
                                          'margin-top': '0.5rem'}
```

```
                            ),
                        ], style={'margin-left': '25rem', 'margin-top':
'0.5rem', 'width': '30%'})

                    ], style={'display': 'flex', 'justify-content': 'space-
between', }),

                    html.Div([
                        html.Div([
                            dcc.Graph(id='scatter-plot'),
                        ], style={'flex': '1'}),
                        html.Div([
                            dcc.Graph(id='violin-plot'),
                        ], style={'flex': '1'}),
                    ], style={'display': 'flex', 'justify-content': 'space-
between'}),
                    html.Div([
                        html.Div([
                            dcc.Loading(
                                id="loading-1",
                                type="default",
                                children=html.Div(id="loading-output-1")
                            ),
                            dcc.Graph(id='interactive-scatter-plot'),
                        ], style={'flex': '1'}),

                        html.Div([
                            dcc.Graph(id='box-plot'),
                        ], style={'flex': '1'}),

                    ], style={'display': 'flex', 'justify-content': 'space-
between'}),
                ])
            ]),

            # TAB-4
            dcc.Tab(label='Payment and Pricing', children=[
                html.Label('Select  category ', style={'font-size': '20px',
'margin-top': '2rem'}),
                html.Div([
                    dcc.Checklist(
                        id='category-checklist1',
                        options=[{'label': category, 'value': category} for
category in
                                 sample_df['Product Category'].unique()],
                        value=[sample_df['Product Category'].unique()[0]],
                        inline=True,
                        labelStyle={'display': 'flex', 'margin-top':
'1rem', 'margin-left': '20px'}
                    ),

                    html.Div([
                        html.Label('Select Dropdown', style={'font-size':
'20px'}),
                        dcc.Dropdown(
                            id='correlation-dropdown',
                            options=[{'label': metric, 'value': metric} for
metric in
                                     ['Product Price', 'Quantity',
'Customer Age', 'Total Purchase Amount']],
                            value='Total Purchase Amount',
```

```
                            multi=False,
                            placeholder="Select a Metric",

                        ),
                    ], style={'width': '25%', 'margin-top': '1rem',
'margin-right': '10rem'}),
                    html.Div([
                        html.Label('Select  Distribution ', style={'font-
size': '20px'}),
                        dcc.RadioItems(
                            id='distribution-radio',
                            options=[{'label': distribution, 'value':
distribution} for distribution in
                                     ['histogram', 'box', 'violin']],
                            value='histogram',
                            labelStyle={'display': 'flex', 'margin-left':
'20rem'}
                        ),
                    ], style={'width': '48%', })
                ], style={'display': 'flex', 'justify-content': 'space-
between', 'margin-bottom': '20px',
                          'text-align': 'center'}),

                html.Div([
                    html.Div([

                        dcc.Graph(id='distribution-plot'),
                    ], style={'width': '48%', 'display': 'inline-block',
'margin-right': '2%',
                              'vertical-align': 'top'}),

                    html.Div([

                        dcc.Graph(id='pie-chart'),
                    ], style={'width': '48%', 'display': 'inline-block',
'vertical-align': 'top'}),
                ], style={'display': 'flex', 'justify-content': 'space-
between', 'margin-bottom': '20px',
                          'text-align': 'center'}),

                html.Div([
                    html.Label('Select Histogram Slider Nbins',
style={'font-size': '20px'}),
                    dcc.Slider(
                        id='histogram-slider',
                        min=1,
                        max=50,
                        step=1,
                        value=10,
                        marks={i: str(i) for i in range(1, 51)},
                        tooltip={'placement': 'bottom', 'always_visible':
True}
                    ),
                    dcc.Graph(id='dynamic-histogram'),
                ], style={'width': '100%', 'margin-bottom': '20px'}),

                html.Div([
                    dcc.Graph(id='heatmap'),
                ], style={'width': '100%', 'margin-bottom': '20px'}),
            ], style={'padding': '20px', 'font-family': 'Arial',
'backgroundColor': '#f2f2f2'}),  # TAB-4 END
```

```
            # TAB-6
            dcc.Tab(label='Geographic Distribution', children=[
                html.Div([
                    dcc.Loading(
                        id="map-loading",
                        type="circle",
                        children=[
                            dcc.Graph(id='map-plot'),
                        ],
                    ),
                ], style={'width': '100%', 'display': 'inline-block'}),

                html.Div([
                    html.Label('Select Anyone ', style={'font-size':
'20px'}),
                    dcc.RadioItems(
                        id='metric-radio',
                        options=[
                            {'label': 'Product Price', 'value': 'Product
Price'},
                            {'label': 'Quantity', 'value': 'Quantity'},
                            {'label': 'Total Purchase Amount', 'value':
'Total Purchase Amount'},
                            {'label': 'Customer Age', 'value': 'Customer
Age'},
                        ],
                        value='Product Price',
                        labelStyle={'display': 'block', 'margin-top': '2%'}
                    ),
                ], style={'width': '18%', 'display': 'inline-block',
'vertical-align': 'top', 'margin-top': '10rem',
                          'margin-left': '2%', }),

                html.Div([
                    dcc.Loading(
                        id="time-series-loading",
                        type="circle",
                        children=[
                            dcc.Graph(id='time-series-analysis'),
                        ],
                    ),
                ], style={'width': '80%', 'display': 'inline-block',
'vertical-align': 'top'}),

                html.Div([
                    html.Label('Select Age From Range Slider',
style={'font-size': '20px'}),
                    dcc.RangeSlider(
                        id='age-range-slider',
                        min=18,
                        max=70,
                        step=1,
                        marks={i: str(i) for i in range(18, 71)},
                        value=[18, 70]
                    ),

                    dcc.Graph(id='filtered-scatter-plot'),
                ], style={'width': '100%', 'display': 'inline-block',
'vertical-align': 'top'}),
```

```
            ]),  # TAB-6 END

            # TAB-8
            dcc.Tab(label='Data Download & Customization', children=[
                html.H1("Data Download & Customization"),
                # Quantity Slider
                html.Label('Select Quantity Slider', style={'font-size':
'20px'}),
                dcc.RangeSlider(
                    id='quantity-slider',
                    min=1,
                    max=10,
                    step=1,
                    marks={i: str(i) for i in range(1, 11)},
                    value=[1, 10],
                ),

                # Customizable Bar Chart
                dcc.Graph(
                    id='custom-bar-chart',

                ),

                # Segmentation Analysis Chart (Placeholder)
                dcc.Graph(
                    id='segmentation-analysis-chart',
                    config={'editable': True},
                ),

                dash_table.DataTable(
                    id='data-table',
                    columns=[{'name': col, 'id': col} for col in
sample_df.columns],
                    data=sample_df.to_dict('records'),
                    export_format='csv',
                    tooltip_duration=None,  # Show tooltip indefinitely
                    tooltip_data=tooltip_data,
                ),

            ]),  # TAB-8
            # TAB-9
            dcc.Tab(label='Advanced Visualizations', children=[
                html.H1("Advanced Visualizations"),

                # First row
                html.Div([
                    # First column
                    html.Div([
                        html.Label('Select Scatter x Dropdown',
style={'font-size': '20px'}),
                        dcc.Dropdown(
                            id='scatter-x-dropdown',
                            options=[{'label': col, 'value': col} for col
in sample_df.columns],
                            value='Product Price',
                            style={'margin-bottom': '1rem'}
                        ),
                        html.Label('Select Scatter y Dropdown',
style={'font-size': '20px'}),
                        dcc.Dropdown(
                            id='scatter-y-dropdown',
```

```
                            options=[{'label': col, 'value': col} for col
in sample_df.columns],
                            value='Total Purchase Amount',
                        ),
                    ], style={'width': '20%', 'display': 'inline-block',
'text-align': 'center',
                              'margin-left': '15rem'}),

                    # Second column
                    html.Div([
                        html.Label('Select Violin y Dropdown',
style={'font-size': '20px'}),
                        dcc.Dropdown(
                            id='violin-y-dropdown',
                            options=[{'label': col, 'value': col} for col
in sample_df.columns],
                            value='Total Purchase Amount',
                        ),
                    ], style={'width': '20%', 'display': 'inline-block',
'margin-left': '15rem',
                              'text-align': 'center'}),
                ], style={'display': 'flex'}),

                # Second row
                html.Div([
                    # First column (scatter plot)
                    html.Div([
                        dcc.Graph(
                            id='scatter-plot1',
                        ),
                    ], style={'width': '49%', 'display': 'inline-block'}),

                    # Second column (violin plot)
                    html.Div([
                        dcc.Graph(
                            id='violin-plot1',
                        ),
                    ], style={'width': '49%', 'display': 'inline-block'}),
                ], style={'width': '100%', 'display': 'flex'}),
            ]),  # TAB-9

            # TAB-10
            dcc.Tab(label='Reporting & Documentation', children=[
                # Header
                html.H1('Tab 10: Reporting & Documentation',
style={'gridColumn': 'span 2', 'text-align': 'center'}),

                # Payment Method Dropdown and Product Category Dropdown in
the same row
                html.Div([
                    html.Div([
                        html.Label('Select Payment Method:',
style={'margin-bottom': '5px'}),
                        dcc.Dropdown(
                            id='payment-method-dropdown',
                            options=[{'label': method, 'value': method} for
method in payment_methods],
                            value='Credit Card',
                            multi=False,
                            style={'width': '100%'}
                        ),
```

```
                    ], style={'gridColumn': 'span 1'}),

                    html.Div([
                        html.Label('Select Product Category:',
style={'margin-bottom': '5px'}),
                        dcc.Dropdown(
                            id='product-category-dropdown',
                            options=[{'label': category, 'value': category}
for category in product_categories],
                            value='Electronics',
                            multi=False,
                            style={'width': '100%'}
                        ),
                    ], style={'gridColumn': 'span 1'}),
                ], style={'display': 'grid', 'gridTemplateColumns': '1fr
1fr', 'gridGap': '10px'}),

                # Customer Age Slider in a separate row
                html.Div([
                    html.Label('Select Customer Age Range:',
style={'margin-bottom': '5px'}),
                    dcc.RangeSlider(
                        id='customer-age-slider',
                        min=18,
                        max=70,
                        step=1,
                        marks={i: str(i) for i in range(18, 71, 5)},
                        value=[18, 70],
                    ),
                ], style={'gridColumn': 'span 2'}),

                # Graphs in a grid layout
                html.Div([
                    # Payment Method Distribution Graph
                    html.Div([
                        dcc.Graph(id='payment-method-dist'),
                    ], style={'gridColumn': 'span 1'}),

                    # Product Category Distribution Graph
                    html.Div([
                        dcc.Graph(id='product-category-dist'),
                    ], style={'gridColumn': 'span 1'}),

                    # Product Price vs. Total Purchase Amount Graph
                    html.Div([
                        dcc.Graph(id='price-vs-amount'),
                    ], style={'gridColumn': 'span 1'}),

                    # Customer Age Distribution Graph
                    html.Div([
                        dcc.Graph(id='customer-age-dist'),
                    ], style={'gridColumn': 'span 1'}),

                    # Swarm Plot Graph
                    html.Div([
                        dcc.Graph(id='swarm-plot'),
                    ], style={'gridColumn': 'span 2'}),
                ], style={'display': 'grid', 'gridTemplateColumns': '1fr
1fr', 'gridGap': '10px'}),
            ]),  # TAB-10
```

```
            # TAB-1
            dcc.Tab(label='Sales Overview', children=[
                html.Div([
                    # Category Checklist and Payment RadioItems side by 
side
                    html.Div([
                        html.Div([
                            html.Label('Select Category Checklist', 
                                       style={'font-size': '20px', 'margin-
bottom': '15rem'}),
                            dcc.Checklist(
                                id='category-checklist',
                                options=[{'label': cat, 'value': cat} for 
cat in df['Product Category'].unique()],
                                value=df['Product 
Category'].unique().tolist(),
                                inline=True
                            )
                        ], style={'flex': '1', 'padding': '20px'}),
                        html.Div([
                            html.Label('Select Payment Radioitems', 
style={'font-size': '20px'}),
                            dcc.RadioItems(
                                id='payment-radioitems',
                                options=[{'label': method, 'value': method} 
for method in 
                                         df['Payment 
Method'].dropna().unique()],
                                value='All',
                                inline=True
                            )
                        ], style={'flex': '1', 'padding': '20px', }),
                    ], style={'display': 'flex', 'margin-left': '15rem', 
}),

                    # Graphs in a flex container
                    html.Div([
                        dcc.Graph(id='bar-chart', style={'flex': '1'}),
                        dcc.Graph(id='line-chart1', style={'flex': '1'}),
                    ], style={'display': 'flex'}),

                    # Area Chart and Stacked Bar Chart below the flex 
container
                    html.Div([
                        dcc.Graph(id='area-chart', style={'flex': '1'}),
                        dcc.Graph(id='stacked-bar-chart', style={'flex': 
'1'}),
                    ], style={'display': 'flex'}),
                ])
            ]),  # TAB-1

        ]),  # TABS END
    ])
])


# TAB-1
# Callbacks for updating the graphs
@app.callback(
    [Output('bar-chart', 'figure'),
     Output('line-chart1', 'figure'),
```

```
     Output('area-chart', 'figure'),
     Output('stacked-bar-chart', 'figure')],
    [Input('category-checklist', 'value'),
     Input('payment-radioitems', 'value')]
)
def update_graphs(selected_categories, selected_payment_method):
    if not selected_categories:
        filtered_df = df
    else:
        filtered_df = df[df['Product Category'].isin(selected_categories)]

    if selected_payment_method != 'All':
        filtered_df = filtered_df[filtered_df['Payment Method'] ==
selected_payment_method]

    # Bar Chart - Sales by Product Category
    bar_fig = px.bar(
        filtered_df.groupby('Product Category')['Total Purchase
Amount'].sum().reset_index(),
        x='Product Category', y='Total Purchase Amount', title='Sales by
Product Category',
        color_discrete_sequence=px.colors.qualitative.Alphabet
    ).update_layout(showlegend=False, title_x=0.5).to_dict()

    # Line Chart - Monthly Total Sales
    line_fig = px.line(
        filtered_df.groupby('Purchase Date')['Total Purchase
Amount'].sum().reset_index(),
        x='Purchase Date', y='Total Purchase Amount', title='Monthly Total
Sales',
        color_discrete_sequence=px.colors.qualitative.Antique
    ).update_layout(showlegend=False, title_x=0.5).to_dict()

    # Area Chart - Yearly Sales Trends
    area_fig = px.area(
        filtered_df.groupby('Purchase Date')['Total Purchase
Amount'].sum().reset_index(),
        x='Purchase Date', y='Total Purchase Amount', title='Yearly Sales
Trends',
        color_discrete_sequence=px.colors.qualitative.Set3
    ).update_layout(showlegend=False, title_x=0.5).to_dict()

    # Stacked Bar Chart - Sales by Category and Payment Method
    stacked_bar_chart_fig = px.bar(
        filtered_df.groupby(['Product Category', 'Payment Method'])[
            'Total Purchase Amount'].sum().unstack().reset_index(),
        x='Product Category', y=filtered_df['Payment
Method'].dropna().unique(),
        title='Sales by Category and Payment Method', barmode='stack',
        color_discrete_sequence=px.colors.qualitative.Pastel
    ).update_layout(showlegend=False, title_x=0.5).to_dict()

    return [bar_fig, line_fig, area_fig, stacked_bar_chart_fig]


# TAB-2
# Callbacks for updating the graphs
@app.callback(
    Output('scatter-plot', 'figure'),
    Output('violin-plot', 'figure'),
    Output('box-plot', 'figure'),
```

```
    Output('interactive-scatter-plot', 'figure'),
    Input('gender-checklist', 'value'),
    Input('category-dropdown', 'value')
)
def update_graphs1(selected_genders, selected_category):
    # Convert a single string to a list
    selected_genders = [selected_genders] if isinstance(selected_genders,
str) else selected_genders
    selected_category = [selected_category] if
isinstance(selected_category, str) else selected_category

    filtered_df = df2[df2['Gender'].isin(selected_genders)]
    category_filtered_df = df2[df2['Product
Category'].isin(selected_category)]

    # Scatter Plot - Customer age vs. purchase amount
    scatter_fig = px.scatter(
        filtered_df, x='Customer Age', y='Total Purchase Amount',
        color='Gender', title='Customer Age vs. Purchase Amount'
    )
    scatter_fig.update_layout(title_x=0.5)

    # Violin Plot - Product prices by categories
    violin_fig = px.violin(
        category_filtered_df, y='Product Price', x='Product Category',
        color='Product Category', title='Product Prices by Categories'
    )
    violin_fig.update_layout(title_x=0.5)

    # Box Plot - Compare total purchase amounts
    box_fig = px.box(
        category_filtered_df, y='Total Purchase Amount', x='Product
Category',
        title='Total Purchase Amounts by Product Category'
    )
    box_fig.update_layout(title_x=0.5)
    # Interactive Line Plot - Customer age vs. purchase amount
    interactive_line_fig = px.bar(
        filtered_df, x='Customer Age', y='Total Purchase Amount',
        color='Gender', title='Customer Age vs. Purchase Amount'
    )

    interactive_line_fig.update_layout(title_x=0.5)
    return scatter_fig, violin_fig, box_fig, interactive_line_fig


# TAB-4
@app.callback(
    [Output('pie-chart', 'figure'),
     Output('heatmap', 'figure'),
     Output('distribution-plot', 'figure'),
     Output('dynamic-histogram', 'figure')],
    [Input('category-checklist1', 'value'),
     Input('correlation-dropdown', 'value'),
     Input('distribution-radio', 'value'),
     Input('histogram-slider', 'value')]
)
def update_payment_pricing_graphs(selected_categories, selected_metric,
selected_distribution, bin_size):
    filtered_df = sample_df[sample_df['Product
Category'].isin(selected_categories)]
```

```
    # Pie Chart
    pie_chart_fig = px.pie(filtered_df, names='Payment Method',
title='Distribution of Payment Methods')

    # Heatmap
    correlation_matrix = filtered_df[['Product Price', 'Quantity',
'Customer Age', 'Total Purchase Amount']].corr()

    # Manually create annotations
    annotations = []
    for i, row in enumerate(correlation_matrix.index):
        for j, col in enumerate(correlation_matrix.columns):
            annotations.append(
                dict(
                    x=col,
                    y=row,
                    text=f"{correlation_matrix.iloc[i, j]:.2f}",
                    showarrow=False,
                    font=dict(size=8, color='white'),
                )
            )

    heatmap_trace = go.Heatmap(z=correlation_matrix.values,
                               x=correlation_matrix.index,
                               y=correlation_matrix.columns,
                               colorscale='Viridis',
                               colorbar=dict(title='Correlation'),
                               hoverongaps=False)
    heatmap_layout = dict(title='Correlation Matrix',
annotations=annotations)
    heatmap_fig = go.Figure(data=[heatmap_trace], layout=heatmap_layout)

    # Distribution Plot
    distribution_plot_fig = px.histogram(filtered_df, x=selected_metric,
marginal=selected_distribution,
                                         title=f'Distribution of
{selected_metric} ({selected_distribution.capitalize()})')

    # Dynamic Histogram
    dynamic_histogram_fig = px.histogram(filtered_df, x='Product Price',
nbins=bin_size,
                                         title=f'Dynamic Histogram (Bin
Size: {bin_size})')

    return pie_chart_fig, heatmap_fig, distribution_plot_fig,
dynamic_histogram_fig


# TAB-6
# Callbacks to update graphs based on user input
@app.callback(
    Output('map-plot', 'figure'),
    Output('time-series-analysis', 'figure'),
    Output('filtered-scatter-plot', 'figure'),
    [Input('metric-radio', 'value'),
     Input('age-range-slider', 'value')]
)
def update_geographic_distribution(metric, age_range):
    # Map Plot
    map_fig = px.scatter_mapbox(sample_df, lat='Latitude', lon='Longitude',
```

```python
color=metric,
                                hover_name='Payment Method',
                                title='Geographic Distribution of
Customers',
                                mapbox_style='open-street-map')

    # Multi-Line Chart (Time Series Analysis)
    time_series_analysis_fig = px.line(sample_df, x=sample_df.index,
y=metric, title='Time Series Analysis',
                                       labels={'index': 'Customer Index',
'y': metric})

    # Filtered Scatter Plot
    filtered_df = sample_df[(sample_df['Customer Age'] >= age_range[0]) &
(sample_df['Customer Age'] <= age_range[1])]
    filtered_scatter_plot_fig = px.scatter(filtered_df, x='Customer Age',
y=metric, color='Product Category',
                                           title='Filtered Scatter Plot',
hover_name='Payment Method')

    return map_fig, time_series_analysis_fig, filtered_scatter_plot_fig


# TAB-8
@app.callback(
    Output('custom-bar-chart', 'figure'),
    Output('data-table', 'data'),
    Output('segmentation-analysis-chart', 'figure'),
    Input('quantity-slider', 'value')
)
def update_charts(quantity_range):
    filtered_df = sample_df[(sample_df['Quantity'] >= quantity_range[0]) &
(sample_df['Quantity'] <= quantity_range[1])]

    # Customizable Bar Chart
    bar_chart = px.histogram(
        filtered_df,
        x='Product Category',
        y='Total Purchase Amount',
        color='Payment Method',
        barmode='group',
        color_discrete_sequence=px.colors.qualitative.Light24_r
    )

    # Adjust bargap and bargroupgap through update_layout
    bar_chart.update_layout(
        bargap=0.1,  # Adjust the bargap value as needed
        bargroupgap=0.2  # Adjust the bargroupgap value as needed
    )

    # # Create a 'Segment' column (this is just an example, you should
adjust it based on your segmentation logic)
    # filtered_df['Segment'] = np.where(filtered_df['Total Purchase
Amount'] > 500, 'High Value', 'Low Value')

    # Segmentation Analysis Chart (Placeholder)
    segmentation_chart = px.pie(sample_df, names='Segment',
title='Segmentation Analysis')

    return bar_chart, filtered_df.to_dict('records'), segmentation_chart
```

```
##TAB-9
# Callbacks for dynamic updates
@app.callback(
    Output('scatter-plot1', 'figure'),
    Output('violin-plot1', 'figure'),
    Input('scatter-x-dropdown', 'value'),
    Input('scatter-y-dropdown', 'value'),
    Input('violin-y-dropdown', 'value')
)
def update_plots(scatter_x, scatter_y, violin_y):
    # Ensure 'Segment' is added to the DataFrame
    sample_df['Segment'] = np.where(sample_df['Total Purchase Amount'] >
500, 'High Value', 'Low Value')

    # Scatter Plot
    scatter_fig = px.scatter(sample_df, x=scatter_x, y=scatter_y,
                             color='Payment Method', hover_data=['Product
Category', 'Segment'],
                             title='Tooltip Enhanced Scatter Plot')

    # Violin Plot
    violin_fig = px.violin(sample_df, y=violin_y, x='Segment', box=True,
                           points="all", title='Violin Plot with
Customization')

    return scatter_fig, violin_fig


# TAB-10
# Define callbacks to update the graphs based on user interactions
@app.callback(
    Output('payment-method-dist', 'figure'),
    Output('product-category-dist', 'figure'),
    Output('price-vs-amount', 'figure'),
    Output('customer-age-dist', 'figure'),
    Output('swarm-plot', 'figure'),
    Input('payment-method-dropdown', 'value'),
    Input('product-category-dropdown', 'value'),
    Input('customer-age-slider', 'value')
)
def update_graph(selected_payment_method, selected_product_category,
selected_customer_age_range):
    # Filter data based on user selections
    filtered_df = sample_df[
        (sample_df['Payment Method'] == selected_payment_method) &
        (sample_df['Product Category'] == selected_product_category) &
        (sample_df['Customer Age'] >= selected_customer_age_range[0]) &
        (sample_df['Customer Age'] <= selected_customer_age_range[1])
        ]

    # Header 1: Payment Method Distribution
    payment_dist = px.histogram(filtered_df, x='Payment Method',
title='Payment Method Distribution')

    # Header 2: Product Category Distribution
    category_dist = px.histogram(filtered_df, x='Product Category',
title='Product Category Distribution')

    # Header 3: Product Price vs. Total Purchase Amount
    price_vs_amount = px.scatter(filtered_df, x='Product Price', y='Total
```

```
Purchase Amount',
                                                  title='Product Price vs. Total Purchase
Amount')

    # Header 4: Customer Age Distribution
    age_dist = px.histogram(filtered_df, x='Customer Age', title='Customer
Age Distribution')

    # Swarm Plot: Product Price by Product Category and Payment Method
    swarm_plot = px.strip(filtered_df, x='Product Category', y='Product
Price', color='Payment Method',
                                       title='Swarm Plot: Product Price by Product
Category and Payment Method')

    return payment_dist, category_dist, price_vs_amount, age_dist,
swarm_plot


# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

References

[1] https://plotly.com/

[2] https://www.analyticsvidhya.com/blog/2021/03/step-by-step-process-of-feature-engineering-for-machine-learning-algorithms-in-data-science/