Informatics Institute of Technology

Data Mining – CMM 704

Course Work

MSc in Big Data Analytics

Submitted by:

S.Z. Raeesul Islam

20232953/ 2409649

Table of Contents

Clustering – Customer Personality Analysis

Introduction

Understanding customer personality through data analysis is crucial for businesses to succeed in today's competitive business landscape. Businesses can gain valuable insights that drive strategic decision-making and enhance customer engagement by analyzing demographic data, purchasing habits, and marketing responses. The Customer Personality Analysis dataset offers rich information, including demographic attributes, purchasing habits, and marketing campaign responses. This part analyzes the dataset to uncover hidden patterns, segment customers, and derive actionable insights. Advanced data mining techniques and statistical analyses help illuminate factors shaping customer behavior, identify opportunities for personalized marketing strategies, and foster stronger customer relationships.

Research Questions

- How do different ***demographic factors*** (such as age, education, marital status, and income) correlate with ***customer spending behavior***?
- Are there distinct ***customer segments*** based on their ***purchasing habits*** and ***responses*** to marketing campaigns?
- Is there a ***relationship*** between ***recency of purchase*** and customer loyalty or satisfaction?
- Are there any ***trends or patterns*** in ***website visits*** and their relationship with purchasing behavior?

Data Mining Problem and Objectives

Problem

The main problem is to ***segment customers*** effectively based on their behavior and demographics to tailor marketing strategies and product offerings accordingly.

Objectives

- Perform clustering to identify distinct customer segments.
- Analyze the characteristics and behaviors of each segment.
- Understand the factors influencing customer responses to marketing campaigns.
- Identify potential areas for improvement in customer satisfaction and loyalty.
- Provide actionable insights for marketing strategies and product development.

Formulation of the Problem

Data Mining Techniques

Clustering techniques such as K-means, hierarchical clustering, or density-based clustering will segment customers based on their attributes and behaviors. Additionally, association rule mining can identify patterns in customer behavior, such as relationships between different types of purchases or responses to marketing campaigns.

Problem Formulation

Given the Customer Personality Analysis dataset, the objective is to segment customers into ***homogenous groups*** based on their demographic information, purchasing behavior, responses to marketing campaigns, and other relevant attributes. This segmentation will allow us to gain insights into different customer personas and tailor marketing strategies and product offerings to meet their specific needs and preferences.

Approach

- **Data Preprocessing:** Clean the data, handle missing values, and transform categorical variables.
- **Exploratory Data Analysis (EDA):** Understand the distribution of variables, identify correlations, and visualize patterns in the data.
- **Clustering:** Apply clustering algorithms to segment customers into distinct groups based on their attributes and behaviors.
- **Interpretation:** Analyze the characteristics and behaviors of each segment, identify key differences and similarities, and interpret the results in the context of the research questions and business objectives.
- **Evaluation:** Evaluate the effectiveness of the clustering solution based on internal metrics (e.g., silhouette score) and external validation (e.g., domain expert feedback).
- **Insights & Recommendations:** Provide actionable insights and recommendations for marketing strategies, product development, and customer relationship management based on the findings from the analysis.

Implementation

Data Wrangling.

```
# load the customer personality dataset
customer_data = pd.read_csv('Data/Customer_personality.csv')
print(customer_data.shape)
customer_data.head()
```

```
(2240, 29)
```

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94

The customer personality dataset is a structured collection of information containing 2240 individual records. Each record represents a unique customer within the dataset. The dataset is organized into 29 features, providing various customer attributes or characteristics. These columns include customer demographic information such as birth year, Education, income level, and more, the amount spent on different products such as wines, fruits, and more, and other information like promotions.

```
# Information on features
customer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2240 non-null   int64
 1   Year_Birth           2240 non-null   int64
 2   Education            2240 non-null   object
 3   Marital_Status       2240 non-null   object
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64
 6   Teenhome             2240 non-null   int64
 7   Dt_Customer          2240 non-null   object
 8   Recency              2240 non-null   int64
 9   MntWines             2240 non-null   int64
 10  MntFruits            2240 non-null   int64
 11  MntMeatProducts      2240 non-null   int64
 12  MntFishProducts      2240 non-null   int64
 13  MntSweetProducts     2240 non-null   int64
```

```
# check the null counts
customer_data.isnull().sum()
```

```
ID                     0
Year_Birth             0
Education              0
Marital_Status         0
Income                24
Kidhome                0
Teenhome               0
Dt_Customer            0
Recency                0
MntWines               0
MntFruits              0
MntMeatProducts        0
MntFishProducts        0
MntSweetProducts       0
MntGoldProds           0
NumDealsPurchases      0
NumWebPurchases        0
NumCatalogPurchases    0
```

From the above output, several key observations can be made:

- **Missing Income Values:** The dataset contains missing values in the ***'Income'*** column, which need to be addressed to ensure data accuracy.

- **Dt_Customer Format:** The ***'Dt_Customer'*** column, representing customer join dates, isn't in DateTime format, hindering time-based analysis.
- **Categorical Features:** Some features are categorical ('datatype: object'), requiring encoding into numeric forms for analysis, like one-hot or label encoding.

Based on the data quality checks, we have to do data preprocessing and feature engineering for the customer personality dataset. So, let's start with data preprocessing and feature engineering.

Data Preprocessing and Feature Engineering

Handling missing values

The statement indicates that within the dataset, specifically in the "***Income***" variable, there are 24 instances where the value is missing. To handle these missing values, a common approach is to replace them with a meaningful estimate. In this case, the ***mean income*** value from the available data will be used as a substitute for the missing values.

```
# replace missing values of Income with mean of Income
customer_data['Income'].fillna(customer_data['Income'].mean(),
                               inplace = True)
```

Change the Dt_Customer datatype.

The ***"Dt_Customer"*** column in the dataset signifies the registration date for each customer with the firm. Initially, to facilitate proper data-based analysis, it's essential to convert the datatype of this column to datetime format. After converting the column to datetime, the next step involves examining the data to identify the newest and oldest recorded registration dates.

```
# change datatype of Dt_Customer
customer_data["Dt_Customer"] = pd.to_datetime(customer_data["Dt_Customer"],
                                              format="%d-%m-%Y")

dates = []
for i in customer_data["Dt_Customer"]:
    i = i.date()
    dates.append(i)

# Dates of the newest and oldest recorded customer
print("The newest customer's enrolment date in the records:",max(dates))
print("The oldest customer's enrolment date in the records:",min(dates))
```

```
The newest customer's enrolment date in the records: 2014-06-29
The oldest customer's enrolment date in the records: 2012-07-30
```

Based on the above output, 2014-06-29 is the newest customer registered date and 2012-07-30 is the oldest customer registered date.

Feature Creation and Extraction

I'll embark on feature extraction to enhance data insights in the upcoming phase. Initially, I'll derive customers' ***ages*** by subtracting their ***birth years*** from the year 2014.

```
# Extract the "Age" of a customer in 2014.
customer_data["Age"] = 2014 - customer_data['Year_Birth']
```

Following this, I'll calculate ***"Spent"*** representing total expenditure across various categories over two years.

```
# Get the total spent by the customer in various categories.
customer_data["Spent"] = (customer_data["MntWines"] + customer_data["MntFruits"] +
                          customer_data["MntMeatProducts"] + customer_data["MntFishProducts"] +
                          customer_data["MntSweetProducts"] + customer_data["MntGoldProds"])
```

Leveraging ***"Marital_Status"*** I'll construct ***"Living_With"*** to discern household dynamics, designating *"partner"* for those married or in a relationship, and *"alone"* otherwise.

```
# Deriving living situation by marital status.
customer_data["Living_With"] = customer_data["Marital_Status"].replace({"Married":"Partner",
                                                                        "Together":"Partner",
                                                                        "Absurd":"Alone",
                                                                        "Widow":"Alone",
                                                                        "YOLO":"Alone",
                                                                        "Divorced":"Alone",
                                                                        "Single":"Alone",})
```

The ***"Children"*** feature will quantify the total number of children, encompassing both ***kids*** and ***teenagers***, within a household.

```
# Extract the total children living in the household
customer_data["Children"] = customer_data["Kidhome"] + customer_data["Teenhome"]
```

Additionally, I'll introduce ***"Family_Size"*** to provide further granularity on household composition.

```
# Find the family size in the householde
customer_data["Family_Size"] = (customer_data["Living_With"].replace({"Alone": 1,
                                                                      "Partner":2})
                                + customer_data["Children"])
```

To delineate parenthood status, ***"Is_Parent"*** will be generated, assigning a value of 1 if children are present and 0 otherwise.

```
# Extracting parent or not
customer_data["Is_Parent"] = np.where(customer_data['Children'] > 0, 1, 0)
```

Then the ***"Total_Promos"*** feature will represent the total number of accepted promotions. Extracted using the summation of the ***AcceptedCmd*** features.

```
# Extract a sum of accepted promotions
customer_data["Total_Promos"] = (customer_data["AcceptedCmp1"] + customer_data["AcceptedCmp2"] +
                                 customer_data["AcceptedCmp3"] + customer_data["AcceptedCmp4"] +
                                 customer_data["AcceptedCmp5"])
```

Lastly, I'll simplify ***"Education"*** into three categories to streamline analysis.

```
# Segmenting education levels in 3 groups
customer_data["Education"] = customer_data["Education"].replace({"Basic":"Undergraduate",
                                                                 "2n Cycle":"Undergraduate",
                                                                 "Graduation":"Graduate",
                                                                 "Master":"Postgraduate",
                                                                 "PhD":"Postgraduate"})
```

For clarification, I will *rename* some column names.

```
# Rename some column names for clarity
customer_data = customer_data.rename(columns={"MntWines": "Wines",
                                              "MntFruits":"Fruits",
                                              "MntMeatProducts":"Meat",
                                              "MntFishProducts":"Fish",
                                              "MntSweetProducts":"Sweets",
                                              "MntGoldProds":"Gold"})
```

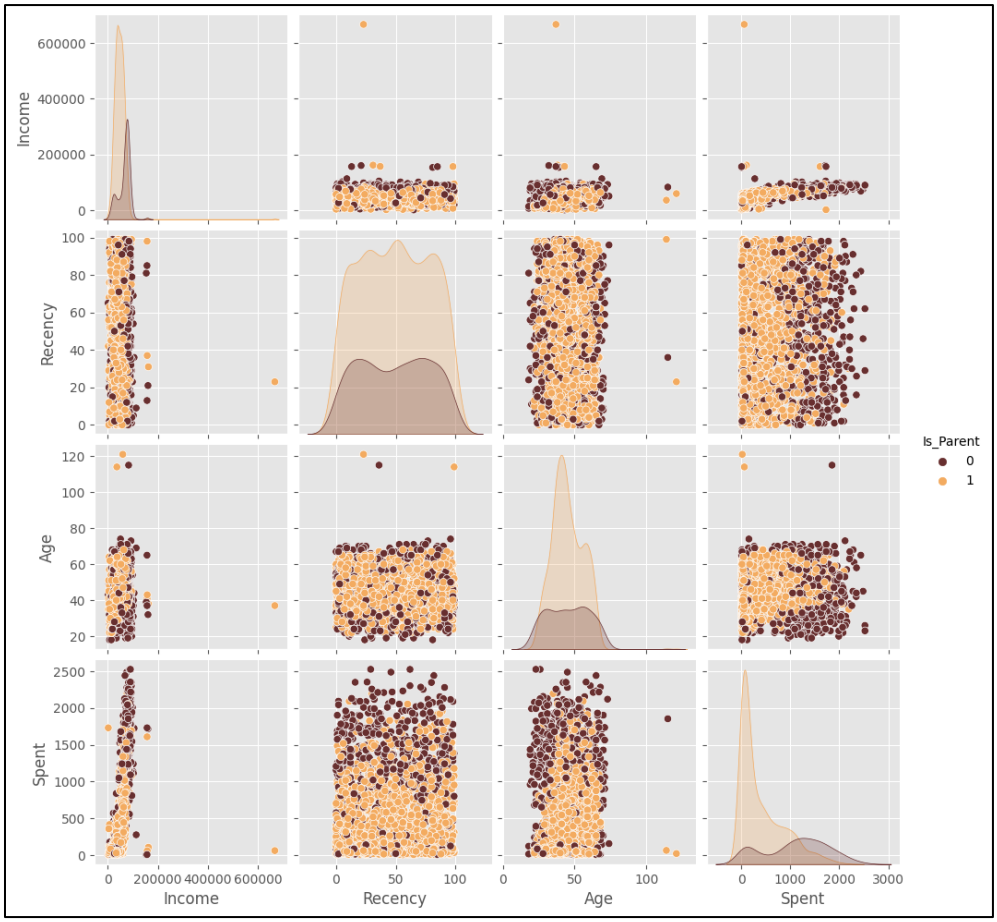
Redundant features will then be *eliminated* to refine the dataset, optimizing it for subsequent analysis and modeling endeavors.

```
# Drop redundant features
to_drop = ["Marital_Status", "Dt_Customer", "Z_CostContact", 'AcceptedCmp3',
           'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
           "Z_Revenue", "Year_Birth", "ID"]
customer_data = customer_data.drop(to_drop, axis=1)
customer_data.shape
```

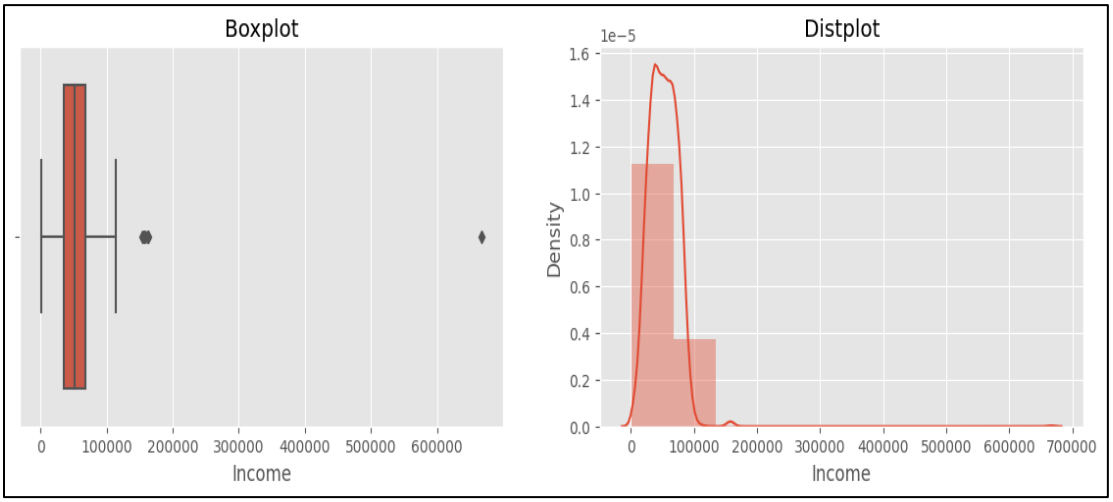
```
(2240, 25)
```

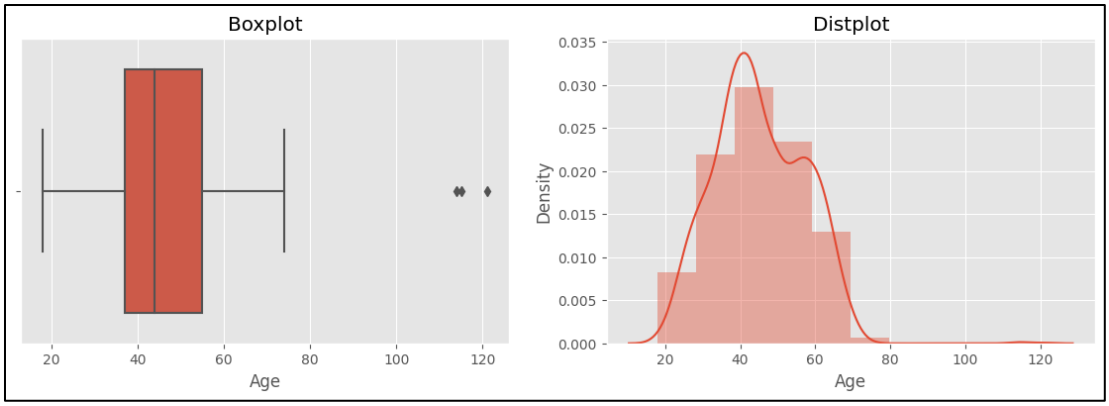
After the feature extraction and drop of redundant features, the customer personality dataset has 2240 records with 25 features.

Handling with Outliers



The pair plot reveals outliers in ***Income, Age, and Spent*** features. Boxplots will be used to verify these outliers, providing a clear visual representation of data distribution.





The pair plots and box plots have unveiled clear indications of outliers within the ***'Income', 'Age', and 'Spent'*** features. To ensure the robustness of our analysis, we'll systematically address these outliers individually.

1. Income Outliers

To start, we'll determine the upper and lower boundary limits for the 'Income' feature.

```
******* Boundary Limits of Income ********
IQR of Income :  32751.0
Upper Limit of Income :  117416.25
Lower Limit of Income :  -13587.75
Number of Outliers Records :  8
```

Upon examination, it appears that one data point significantly exceeds the upper limit. We'll exclude this outlier and address other outliers by substituting them with the upper boundary value of the 'Income' feature.

```
# remove max value records of Income
customer_data = customer_data[~(customer_data['Income'] > 500000)]
customer_data.shape
```

```
(2239, 29)
```

```
# Impute outliers with upper limit
customer_data.loc[customer_data['Income'] > 117416.25, 'Income'] = 117416.25
```

2. **Age Outliers**

First, we'll establish upper and lower boundary limits for the 'Age' feature.

```
******* Boundary Limits of Age ********
IQR of Age :  18.0
Upper Limit of Age :  82.0
Lower Limit of Age :  10.0
Number of Outliers Records :  3
```

Once determined, we'll identify three outliers within the 'Age' feature and proceed to remove them.

```
# remove outliers records of Age
customer_data = customer_data[~(customer_data['Age'] > 82.0)]
```

3. **Spent Outliers**

First, we'll determine the upper and lower boundary limits of the 'Spent' feature.

```
******* Boundary Limits of Spent ********
IQR of Spent :  976.75
Upper Limit of Spent :  2510.625
Lower Limit of Spent :  -1396.375
Number of Outliers Records :  3
```

Once identified, we'll focus on addressing the three outliers present in the 'Spent' feature. Our approach involves imputing these outlier values with the upper boundary value of the 'Spent' feature.

```
# Impute outliers with upper limit
customer_data.loc[customer_data['Spent'] > 2510.625, 'Spent'] = 2510.625
```

Having completed data preprocessing and feature engineering, we're now poised to conduct Exploratory Data Analysis (EDA). EDA involves comprehensively examining the dataset to extract meaningful insights, patterns, and relationships.

Exploratory Data Analysis (EDA)

Having completed the data cleansing and Feature Engineering phases, we're now ready to delve into the analysis of the customer personality data set.

Data Analysis

```
The number of values for feature Education :3 -- ['Graduate' 'Postgraduate' 'Undergraduate']
The number of values for feature Income :1965
The number of values for feature Kidhome :3 -- [0 1 2]
The number of values for feature Teenhome :3 -- [0 1 2]
The number of values for feature Recency :100
The number of values for feature Wines :775
The number of values for feature Fruits :158
The number of values for feature Meat :557
The number of values for feature Fish :182
The number of values for feature Sweets :177
The number of values for feature Gold :213
The number of values for feature NumDealsPurchases :15 -- [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 15]
The number of values for feature NumWebPurchases :15 -- [ 0  1  2  3  4  5  6  7  8  9 10 11 23 25 27]
The number of values for feature NumCatalogPurchases :14 -- [ 0  1  2  3  4  5  6  7  8  9 10 11 22 28]
The number of values for feature NumStorePurchases :14 -- [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13]
The number of values for feature NumWebVisitsMonth :16 -- [ 0  1  2  3  4  5  6  7  8  9 10 13 14 17 19 20]
The number of values for feature Complain :2 -- [0 1]
The number of values for feature Response :2 -- [0 1]
The number of values for feature Age :56
The number of values for feature Spent :1053
The number of values for feature Living_With :2 -- ['Alone' 'Partner']
The number of values for feature Children :4 -- [0 1 2 3]
The number of values for feature Family_Size :5 -- [1 2 3 4 5]
The number of values for feature Is_Parent :2 -- [0 1]
The number of values for feature Total_Promos :5 -- [0 1 2 3 4]
```

Based on the output, the dataset contains two categorical features, and the rest are numerical. We need to separate them into two groups for further analysis.

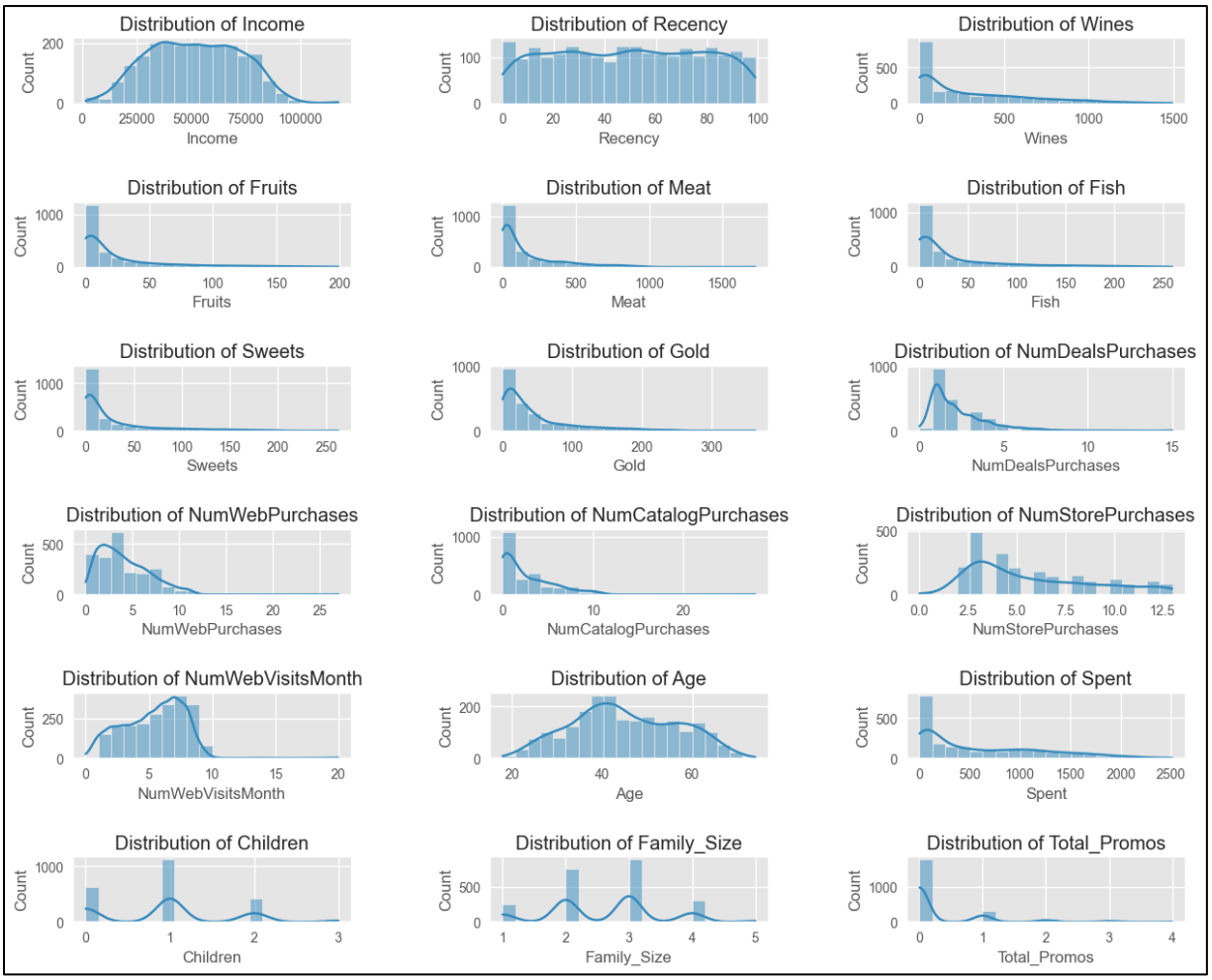
```
Number of Categorical Variables :  2
Categorical Variables :  Index(['Education', 'Living_With'], dtype='object')
Number of Numerical Variables :  23
Numerical Variables :  Index(['Income', 'Kidhome', 'Teenhome', 'Recency', 'Wines', 'Fruits', 'Meat',
       'Fish', 'Sweets', 'Gold', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'Complain', 'Response', 'Age', 'Spent', 'Children', 'Family_Size',
       'Is_Parent', 'Total_Promos'],
      dtype='object')
```

Following the separation of features into numerical and categorical groups, the next step is to delve into the categorical features. By exploring the unique values within these features, we gain insights into the diversity and distribution of categorical data.

```
****** Analysis of Education *****
Value Counts
Graduate         1126
Postgraduate      855
Undergraduate     255
Name: Education, dtype: int64

Value Counts %
Graduate         50.357782
Postgraduate     38.237925
Undergraduate    11.404293
Name: Education, dtype: float64
```

```
****** Analysis of Living_With *****
Value Counts
Partner    1442
Alone       794
Name: Living_With, dtype: int64

Value Counts %
Partner    64.490161
Alone      35.509839
Name: Living_With, dtype: float64
```

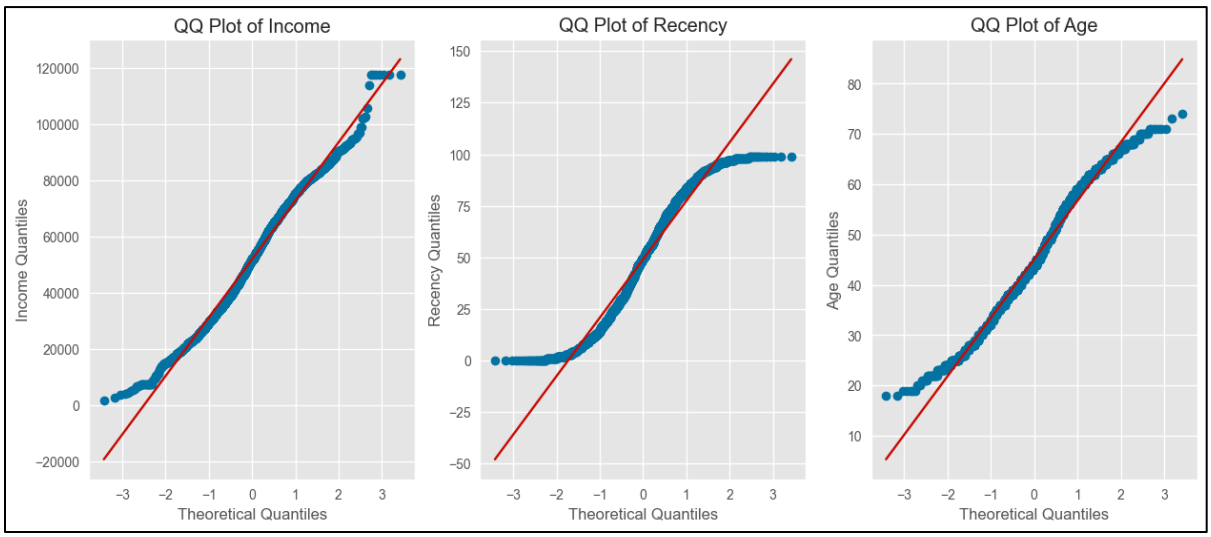
Histogram plots



Upon examining the Histogram Plots, a notable trend emerges: most features exhibit a right-skewed distribution. This indicates that the majority of data points cluster towards the lower end of the range, with a long tail extending towards higher values. However, exceptions to this pattern are observed in the ***'Income', 'Recency', 'Age', and 'Family_size'*** features, which demonstrate distributions that deviate from the right-skewed trend.

Q-Q Plots

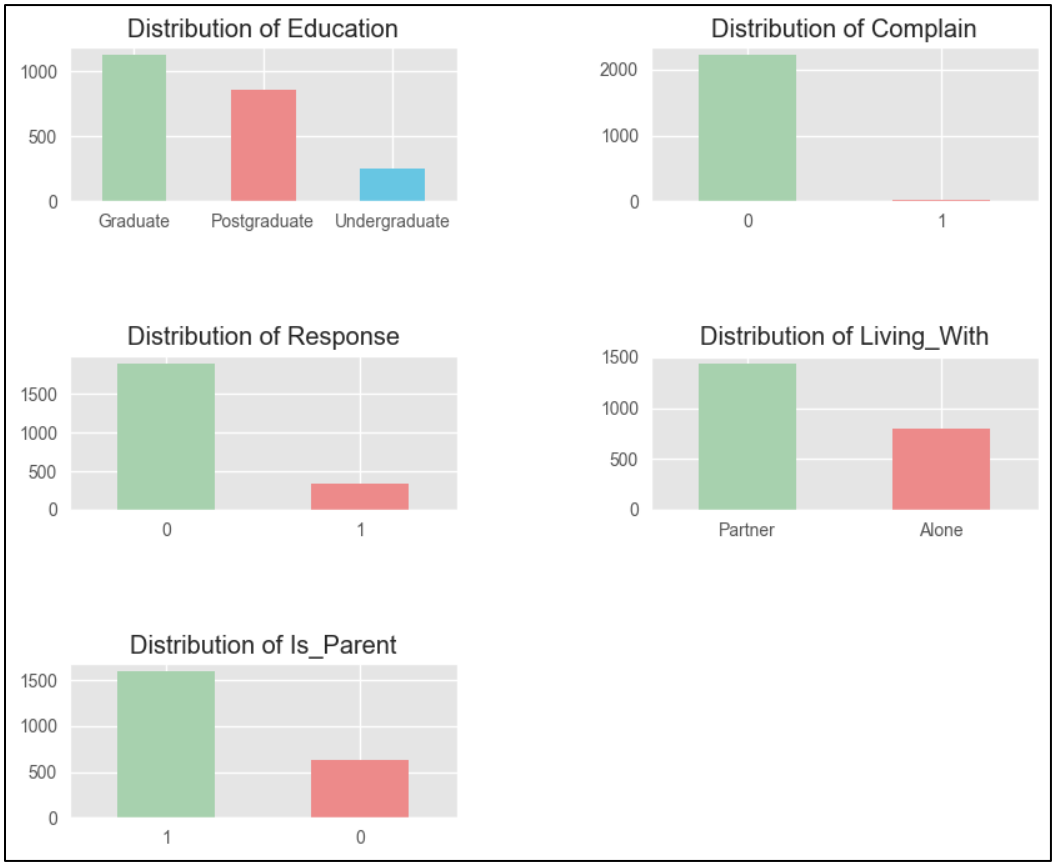
Let's confirm the distribution of ***'Income', 'Recency', and 'Age'*** features using Q-Q plots and the Shapiro-Wilk Test.



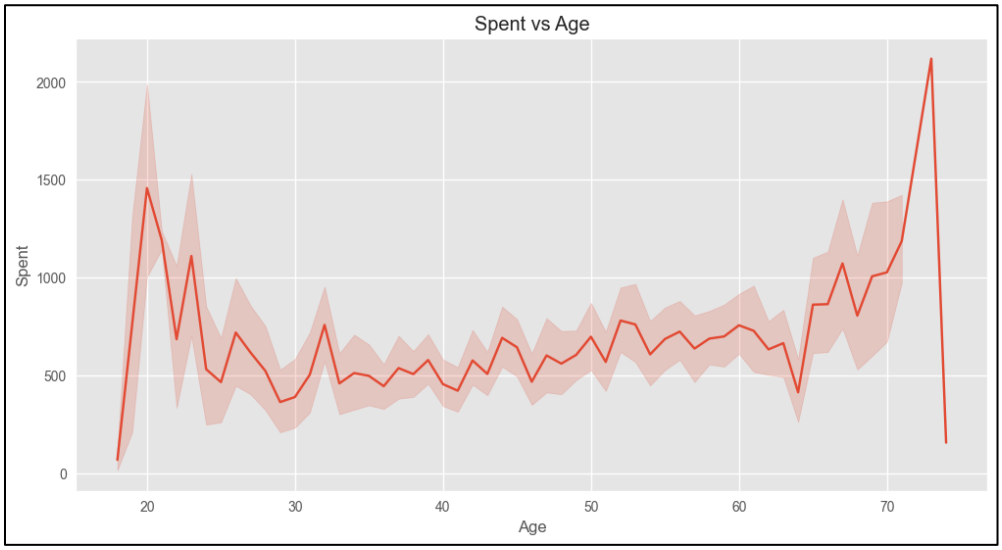

```
Shapiro-Wilk Test for Income
Shapiro-Wilk Test Statistic: 0.9875
p-value: 0.00
The data in Income might not be normally distributed.

Shapiro-Wilk Test for Recency
Shapiro-Wilk Test Statistic: 0.9540
p-value: 0.00
The data in Recency might not be normally distributed.

Shapiro-Wilk Test for Age
Shapiro-Wilk Test Statistic: 0.9833
p-value: 0.00
The data in Age might not be normally distributed.
```

Based on the Q-Q plots and the Shapiro-Wilk Test, we conclude ***'Income', 'Recency', and 'Age'*** features might not be normally distributed.

Bar Plots

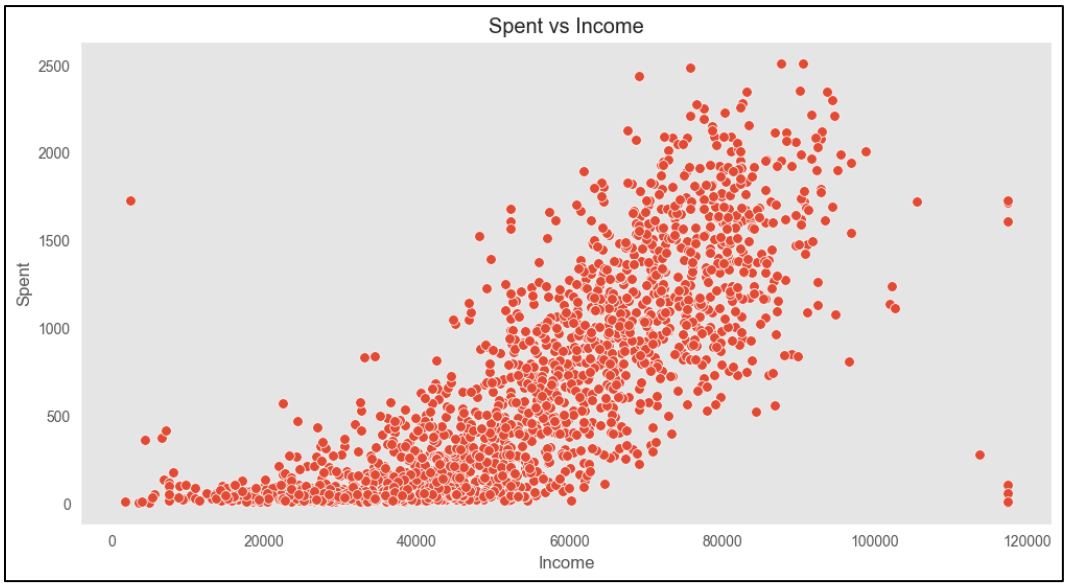


Relationship between Age and Spent



Analyzing the ***'Spent' vs 'Age'*** line plot reveals a positive trend: as the age of customers increases, their spending also tends to increase. This observation is further supported by the *correlation score of 0.114*, indicating a weak positive correlation between age and spending.

Relationship between Income and Spent



By examining the scatter plot depicting the relationship between ***'Spent' and 'Income'***, it's evident that as income levels rise, there is a corresponding increase in spending. This positive trend is visually apparent in the data points, indicating a strong positive correlation between the two variables. The *correlation coefficient of 0.807* quantifies this relationship, confirming a significant and positive linear association between income and spending patterns.

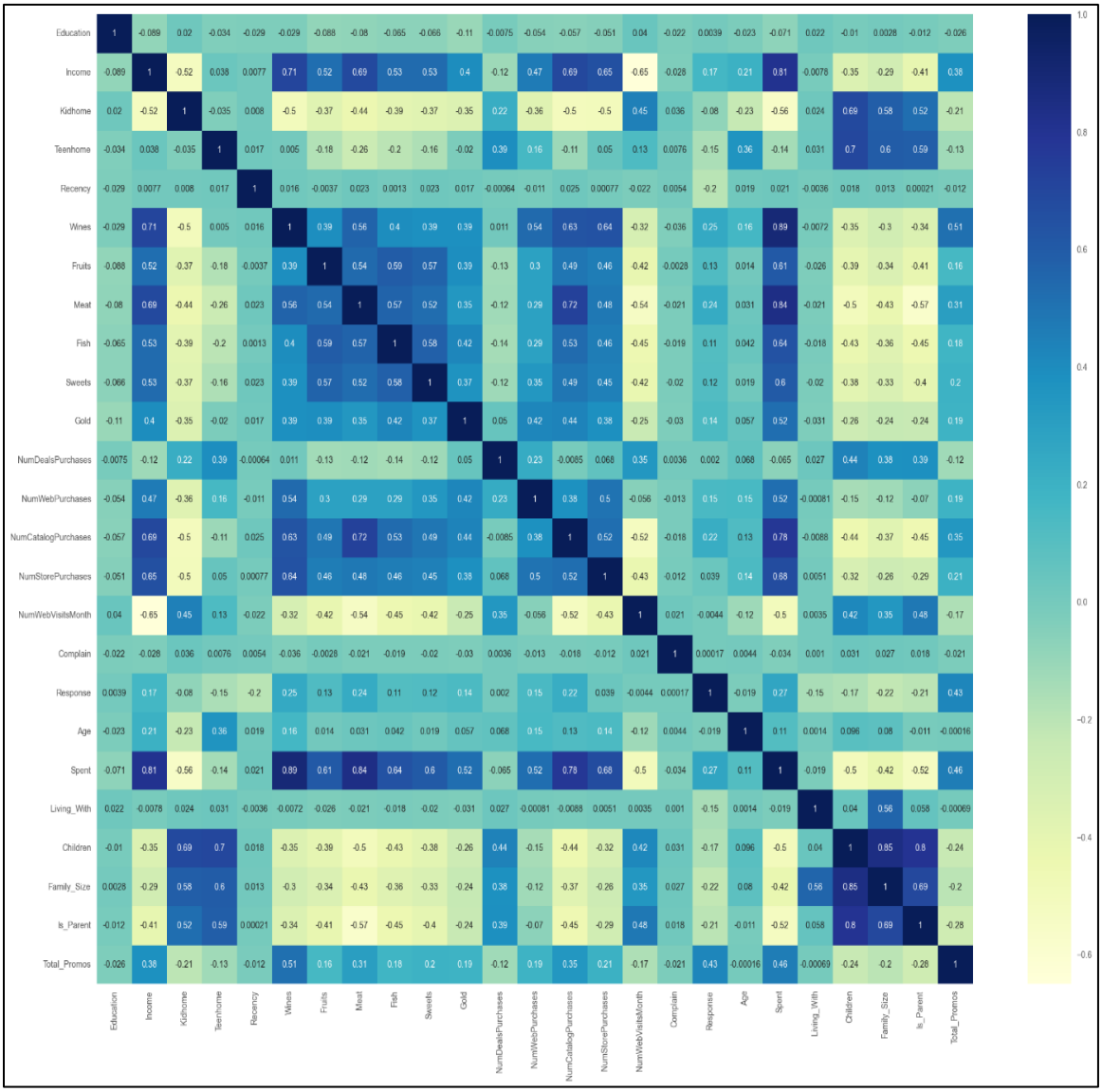
Handling Categorical Features

Upon inspection of the data analysis, we find two categorical variables, both of which are nominal. To facilitate analysis, we opt to convert these categorical variables into numerical form using the ***label encoding*** method.

```
# Convert to numerical using LabelEncoder
le = LabelEncoder()

for i in categorical:
    customer_encoded[i] = customer_encoded[[i]].apply(le.fit_transform)
```

Correlation Heatmap

We'll examine the correlation among the features to understand how they relate to each other. Correlation analysis helps identify patterns and dependencies within the dataset.



Next, we'll conduct a correlation analysis to identify features with a correlation coefficient above the 0.8 threshold.

```
# call the function with 0.8 threshold
cor_feature(customer_encoded, 0.8)
```

```
{'Family_Size', 'Spent'}
```

Notably, we've found two features, ***'Family_Size' and 'Spent'*** exhibiting a high correlation. We can remove both from the dataset, but we will keep those two columns.

Data Scaling

Moving forward, we'll perform data scaling as the next step. This involves standardizing the scale of all features within the dataset using the standard scaler technique.

```
# Scaling
scaler = StandardScaler()
scaler.fit(customer_encoded)
scaled_data = pd.DataFrame(scaler.transform(customer_encoded),
                           columns = customer_encoded.columns)
```

Perform PCA

The clustering process involves managing numerous features but managing them can be challenging due to correlations and redundancy. To address this, dimensionality reduction techniques are employed, condensing the original set into smaller principal variables.

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. Steps in this section:

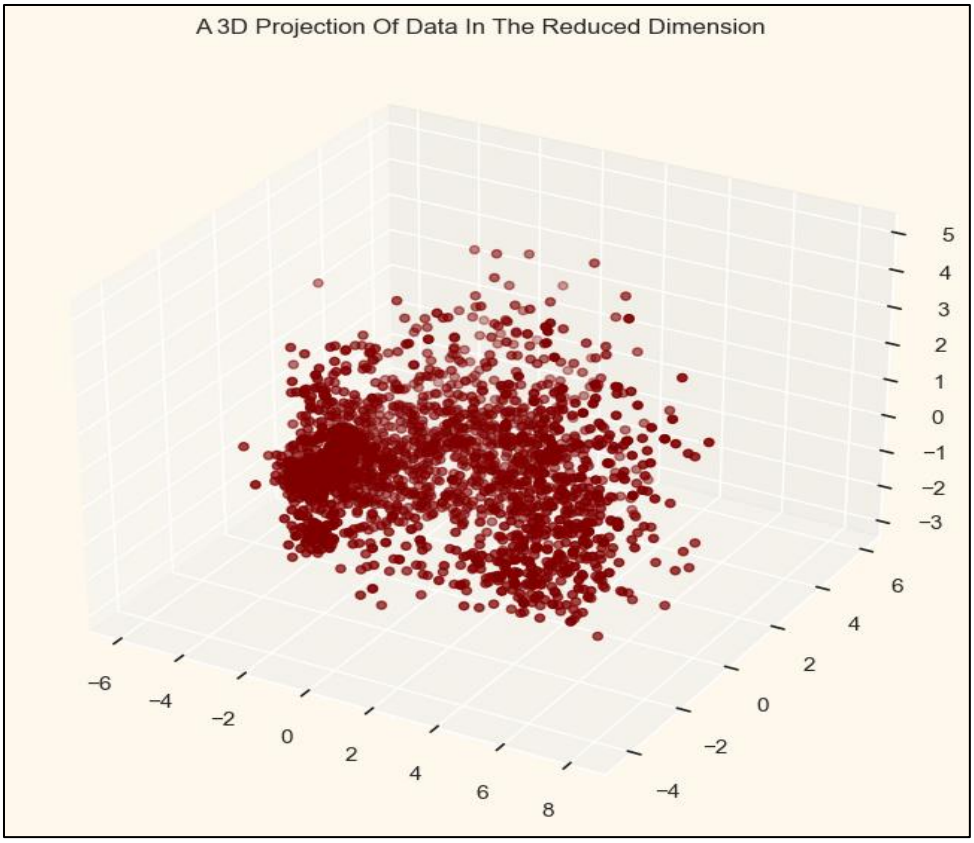
- Dimensionality reduction with PCA
- Plotting the reduced data frame.

In preparation for the clustering process, we will employ PCA to reduce the dimensionality of the dataset to 4.

```
# Applying PCA
pca = PCA(n_components = 4)
pca.fit(scaled_data)
explained_variance = pca.explained_variance_ratio_
explained_variance
```

```
array([0.33999569, 0.11582338, 0.06188597, 0.05238572])
```

We'll generate a 3D projection of the data in a reduced dimensionality space. This plot visualizes the dataset using three PCA columns: col1, col2, and col3.



With data preprocessing, feature engineering, EDA, and performing PCA completed, we're now ready to proceed with clustering the customer personality data. Clustering involves grouping similar data points based on certain criteria. We'll employ two popular clustering algorithms: agglomerative clustering and K-means clustering.

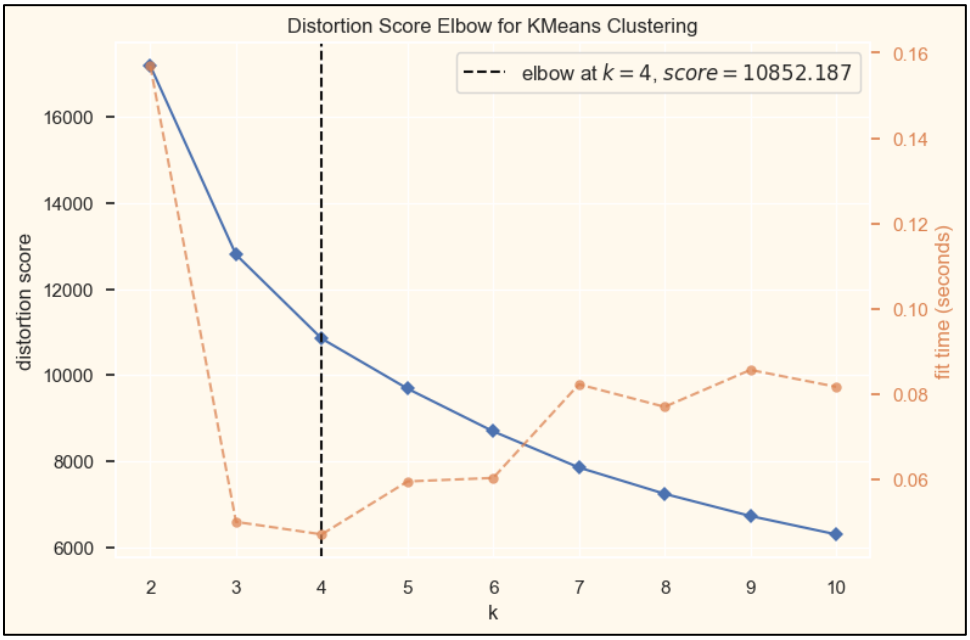
Agglomerative Clustering

Agglomerative clustering is a method that creates a hierarchical tree of clusters by merging similar clusters until all data points belong to a single cluster. This structure offers valuable insights into customer segment relationships. Unlike k-means, agglomerative clustering can accommodate various shapes and densities, reflecting the complexity of customer personality data. It produces a dendrogram, aiding analysts in understanding the hierarchical relationships between clusters.

Agglomerative clustering involves three main steps:

1. **Elbow Method:** Determine optimal cluster count.
2. **Agglomerative Clustering:** Perform hierarchical clustering.
3. **Cluster Examination:** Visualize clusters with scatter plots.

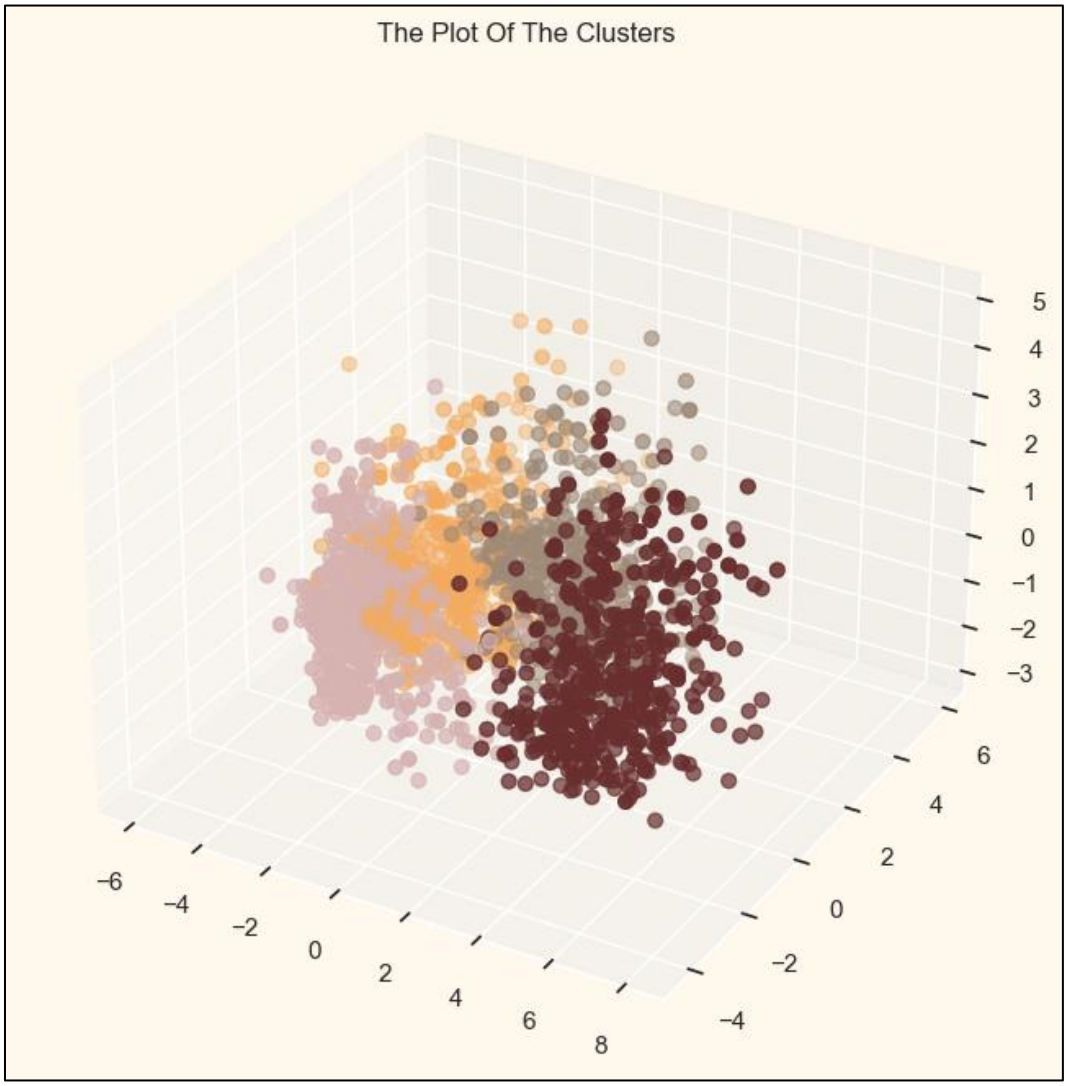
We'll begin by conducting a preliminary analysis using the elbow method. This technique plots the variance explained as a function of the number of clusters. By observing the plot, we can identify the "elbow point," which signifies the optimal number of clusters to form.



Based on the findings from the preceding analysis, it's determined that four clusters would be optimal for grouping the data effectively. To proceed, we'll employ the Agglomerative Clustering Model to finalize these clusters.

```
# Initiating the Agglomerative Clustering model
agg = AgglomerativeClustering(n_clusters=4)

# fit model and predict clusters
yhat_AC = agg.fit_predict(PCA_ds)
PCA_ds["Clusters_agg"] = yhat_AC

# Adding the Clusters feature to the orignal dataframe.
customer_data["Clusters_agg"]= yhat_AC
```

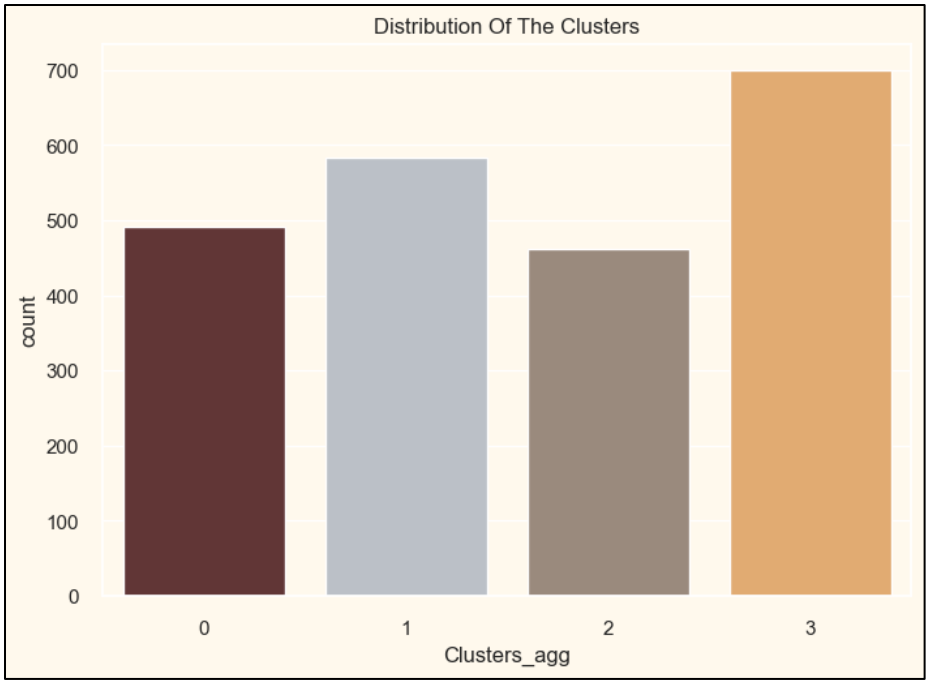
To gain insights into the clusters formed, we'll visualize their distribution in a three-dimensional (3-D) space. This approach enables us to observe the spatial arrangement and separation of clusters in a multidimensional feature space.



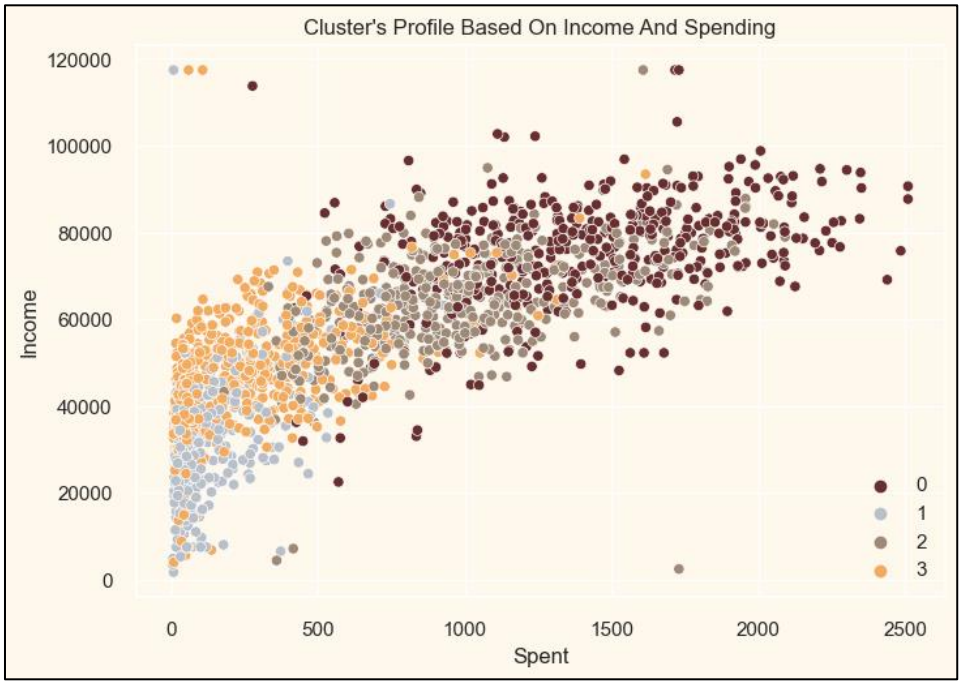
Model Evaluation

In the absence of tagged features for evaluation in this unsupervised clustering task, our focus shifts to understanding the patterns within the clusters formed. This entails exploring the data within the context of these clusters through EDA. By closely examining the characteristics and behaviors of data points within each cluster, we aim to uncover meaningful insights and discern the inherent nature of these cluster patterns.

To begin, we'll examine the distribution of groups within our clustering analysis. This involves understanding how data points are grouped or clustered together based on certain characteristics or features.



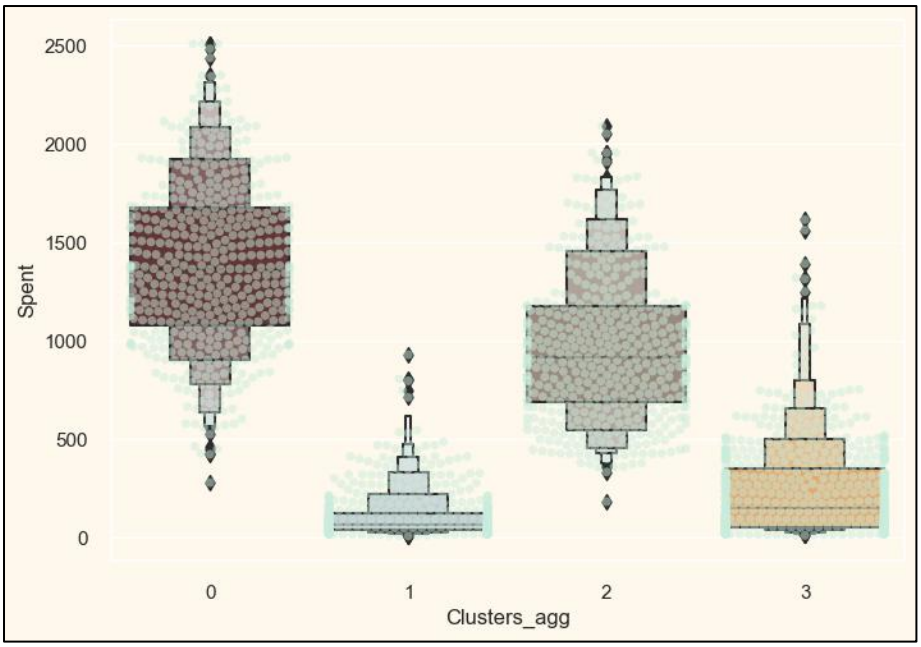
The clusters appear to be evenly dispersed across the dataset. This indicates that the data points within each cluster are well-distributed and balanced, without significant concentration in any particular region.



The income versus spending plot reveals distinct clusters within the data.

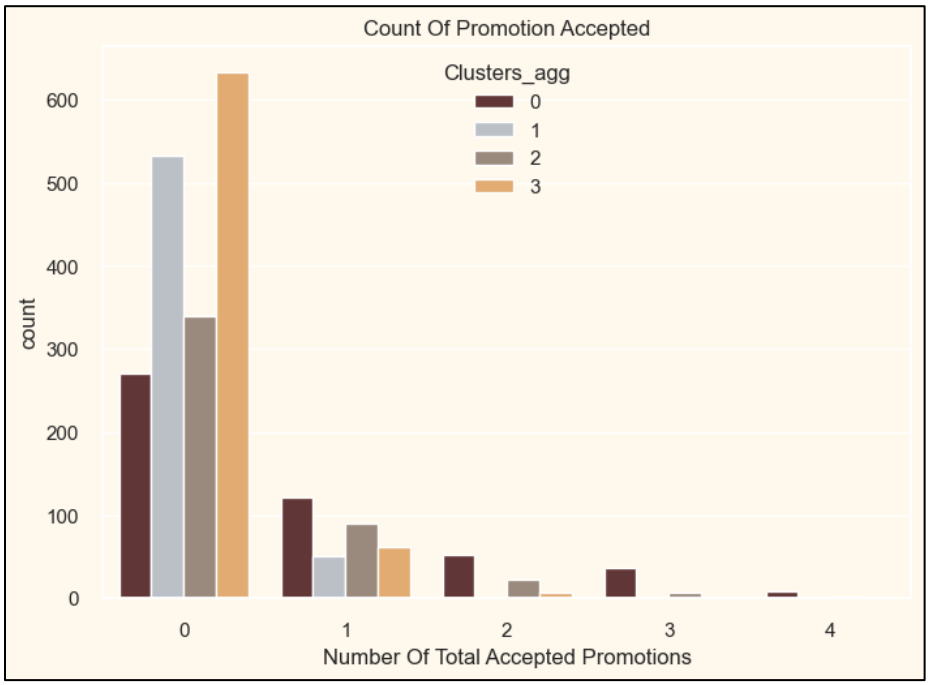
1. **Group 0:** Characterized by high spending and high income.
2. **Group 1:** Represents individuals with low spending and low income.
3. **Group 2:** Comprises individuals with average spending and average income.
4. **Group 3:** This group exhibits low spending and average income.

Next, I will explore cluster distribution across various product categories like Wines, Fruits, Meat, Fish, Sweets, and Gold, focusing on their distribution across the dataset.



The plot above provides a visual representation indicating that cluster 0 comprises the largest group of customers, closely followed by cluster 2.

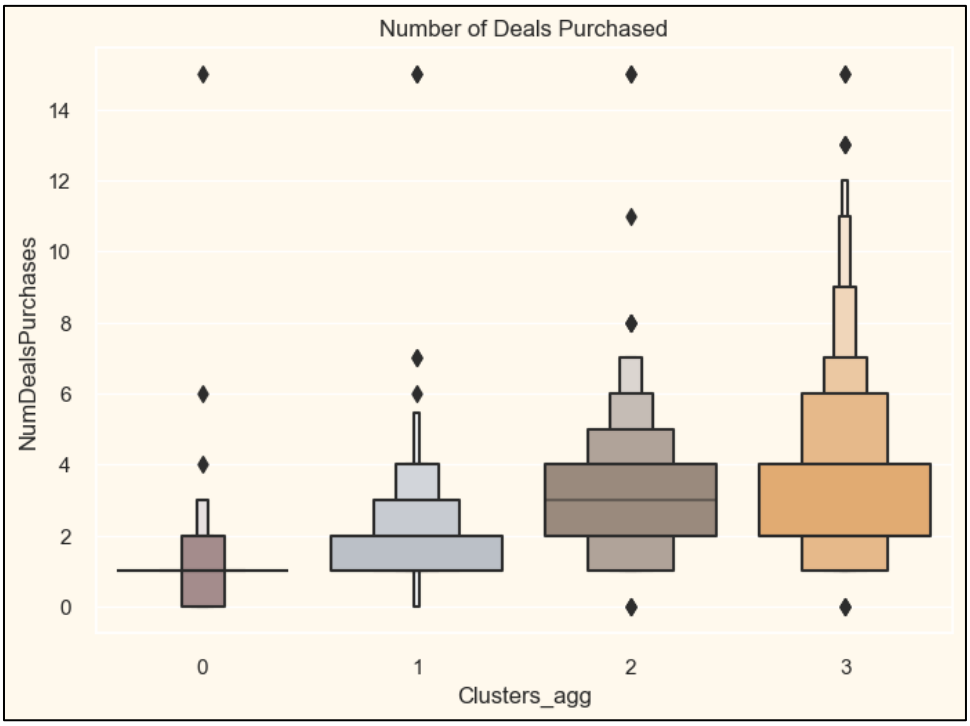
We will now delve into an exploration of our past campaigns to assess their performance and efficacy.



The campaigns have not yielded a significant response thus far, with only a limited number of participants overall. Notably, no participant has engaged with all five campaigns. This

suggests a need for more targeted and meticulously planned campaigns to stimulate sales.

Next, I will Plot the number of deals purchased.

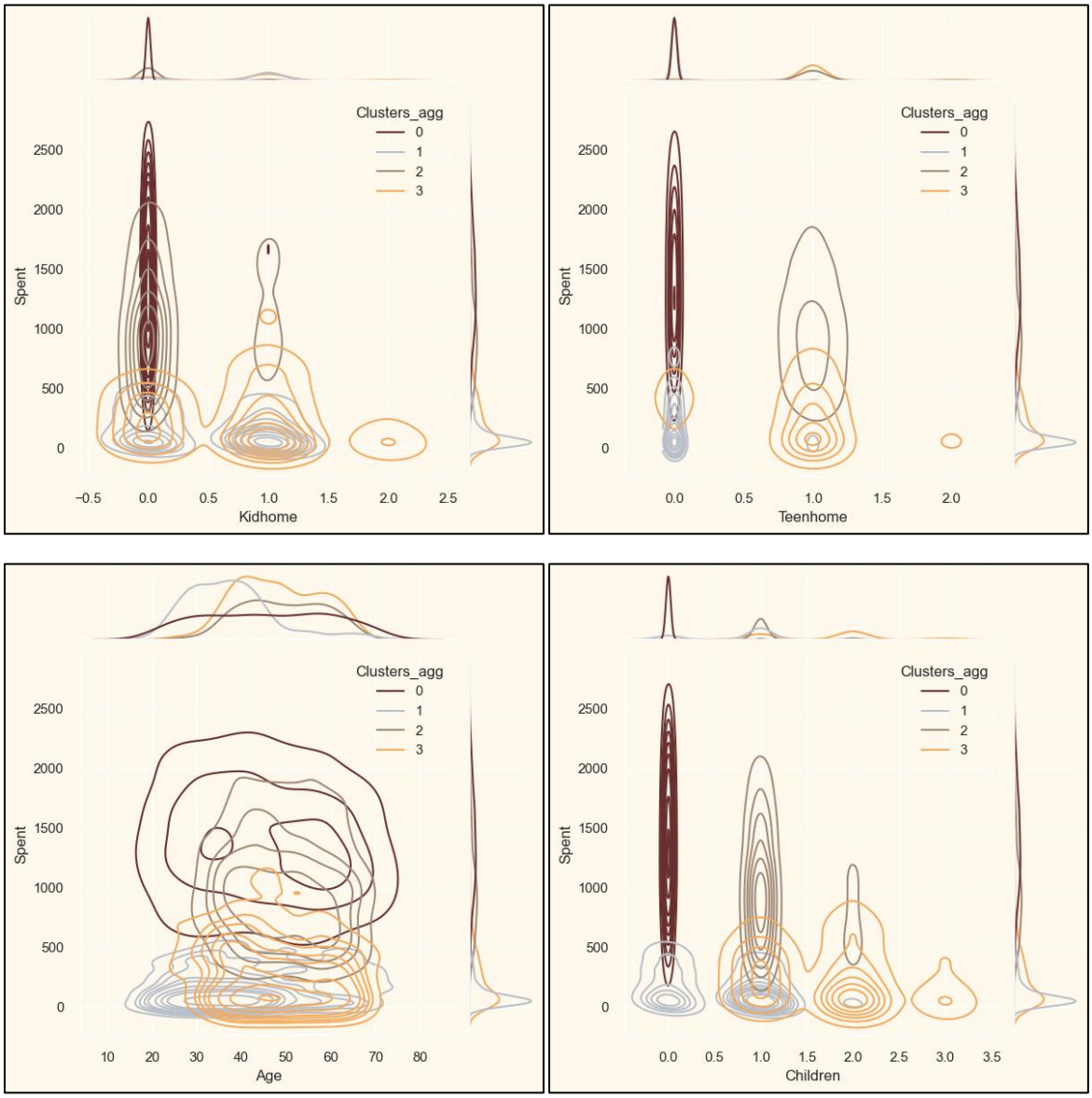


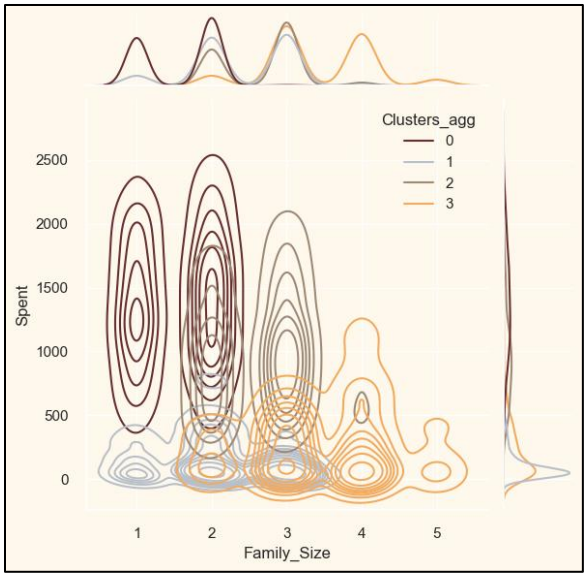
The deals have shown promising outcomes. It has the best outcome with Cluster 3 and Cluster 2. However, our star customers Cluster 0 are not much into the deals. Nothing seems to attract Cluster 1 overwhelmingly.

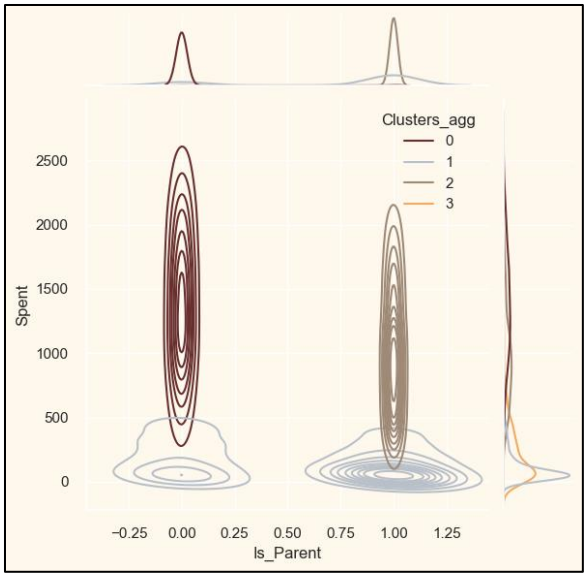
Profiling

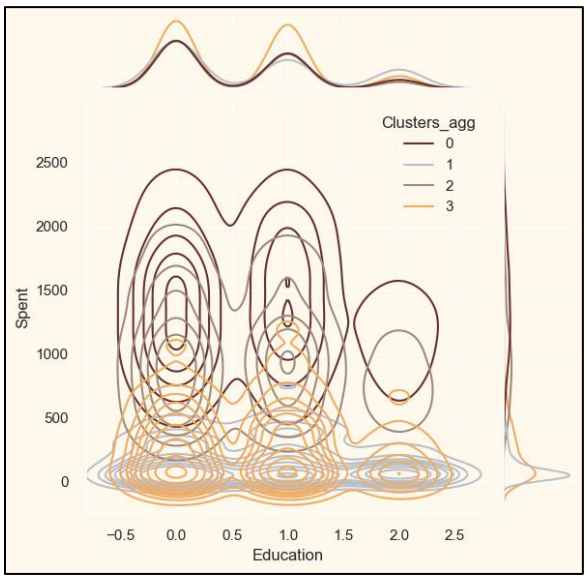
Now we've created the clusters and examined their purchase behavior. Let's check who's in these clusters. For that, we will profile the clusters formed and determine who is our star customer and who needs additional attention from the retail store's marketing team.

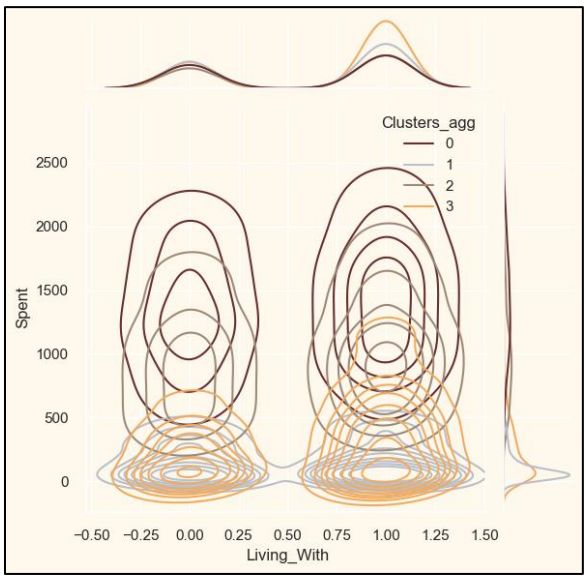
To decide, I will plot some of the features that are indicative of the customer's characteristics concerning the cluster they are in. I will draw inferences based on the outcomes.




Clusters_agg
0
1
2
3
Spent
2500
2000
1500
1000
500
0
1
2
3
4
5
Family_Size


Clusters_agg
0
1
2
3
Spent
2500
2000
1500
1000
500
0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
Is_Parent


Clusters_agg
0
1
2
3
Spent
2500
2000
1500
1000
500
0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Education


Clusters_agg
0
1
2
3
Spent
2500
2000
1500
1000
500
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
Living_With

Conclusion

Cluster 0:

- They are not parents, and they don't have children.
- Approximately 15 to 75 aged people.
- The family has 1 or 2 members.
- High income and high spending group.

Cluster 1:

- The majority of these people are parents.
- At the max have 3 members in the family and at least 1.
- Span all ages.
- Some have one kid, and some do not have any kids at home.
- The majority have only one child at home.
- Low-income and low-spending groups.

Cluster 2:

- They are parents.
- At the max have 4 members in the family and at least 2.
- Approximately 30 to 70 aged people.
- The majority have a child at home.
- Average income and average spending group.

Cluster 3:

- The majority of these people are parents.
- Approximately 25 to 70 aged people.
- At the max have 5 members in the family and at least 2. (5 family members fall into this Cluster).
- Some have one or two kids, and some do not have any kids at home.
- The majority have two children at home.
- Average income and low spending group.

K-Means Clustering

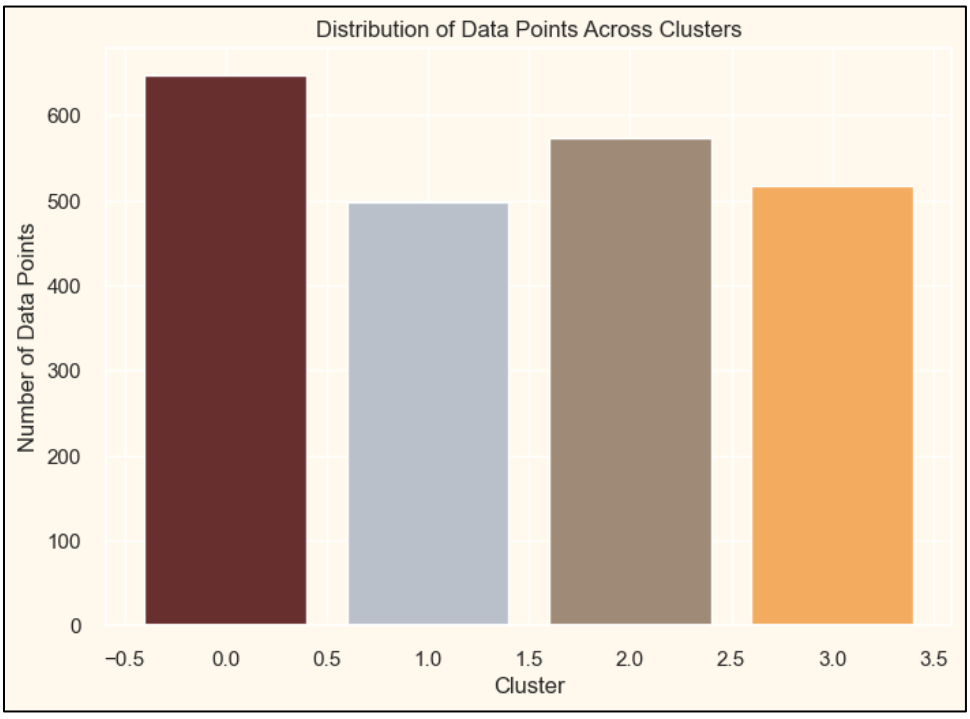
K-means clustering is a computationally efficient and scalable method suitable for analyzing customer spending behavior. It is suitable for large datasets and is easy to interpret, as each cluster is represented by its centroid. K-means also allows for the specification of the desired number of clusters, allowing analysts to explore different segmentation scenarios and choose the most meaningful solution based on domain knowledge and business objectives.

We have already determined that four clusters would be optimal for grouping the data effectively. To proceed, we'll employ the K-means Clustering Model to finalize these clusters.

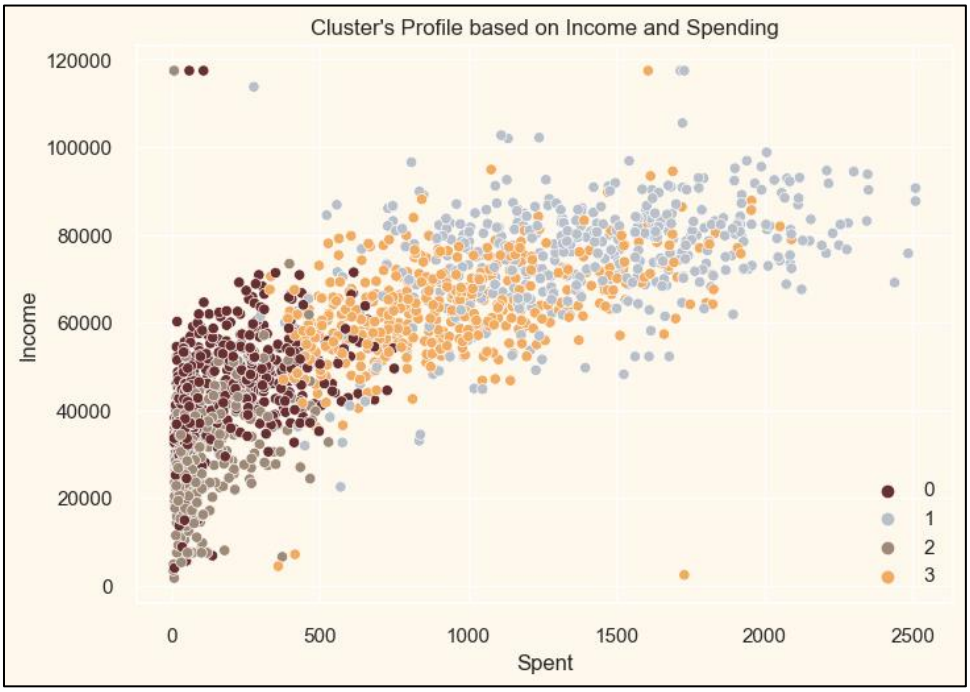
```
# apply kmeans with k=4
kmeans = KMeans(n_clusters=4, random_state=42)
cluster_labels = kmeans.fit_predict(PCA_ds)

customer_data['Cluster_km'] = cluster_labels
customer_encoded['Cluster_km'] = cluster_labels
```

Model Evaluation

To begin, we'll examine the distribution of groups within our clustering analysis. This involves understanding how data points are grouped or clustered together based on certain characteristics or features.



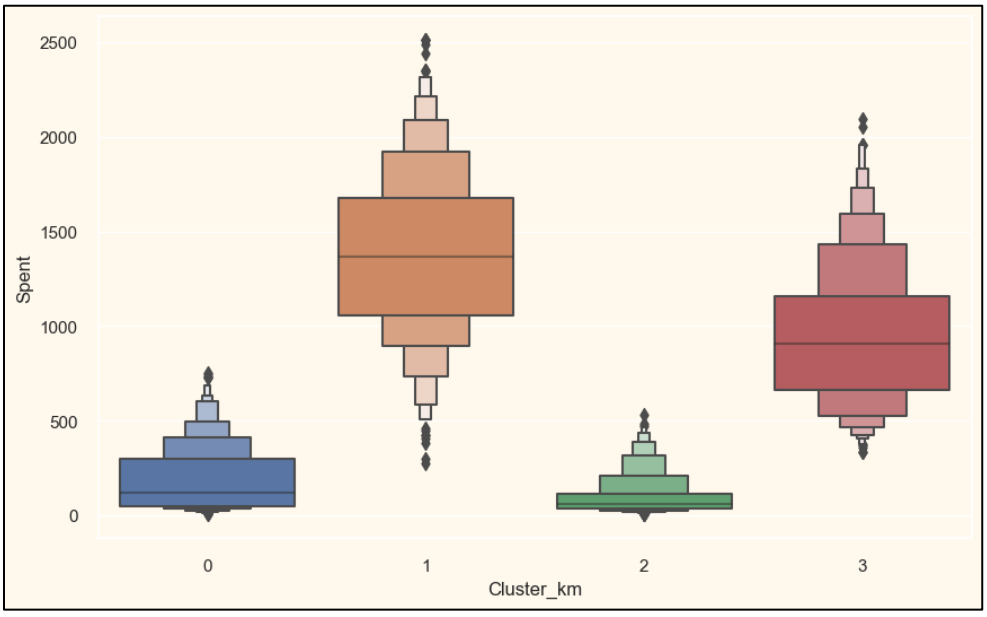
The clusters appear to be evenly dispersed across the dataset. This indicates that the data points within each cluster are well-distributed and balanced, without significant concentration in any particular region.



The income versus spending plot reveals distinct clusters within the data.

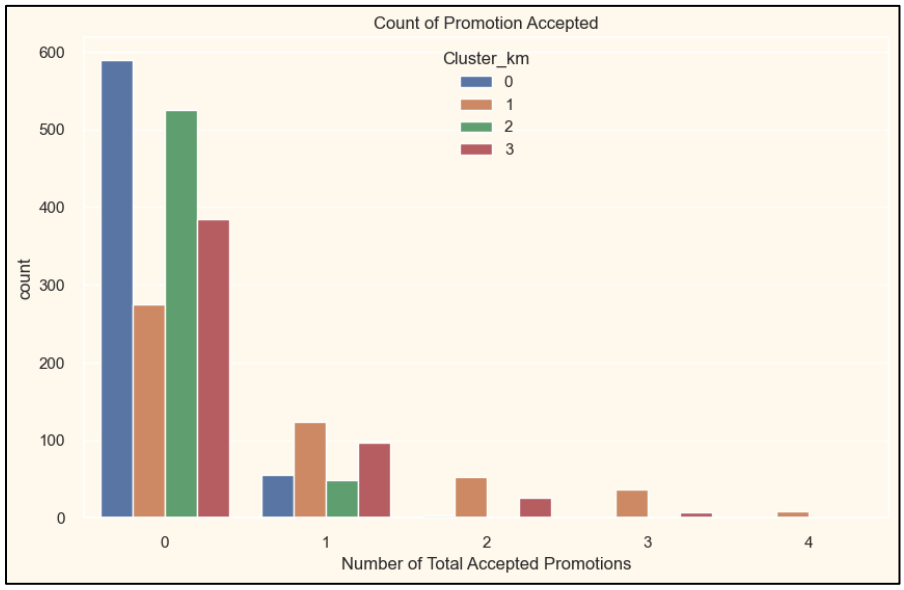
1. **Group 0:** Characterized by low spending and low income.
2. **Group 1:** Represents individuals with high spending and high income.
3. **Group 2:** Comprises individuals with low spending and low income.
4. **Group 3:** This group exhibits average spending and low income.

Next, I will explore the box plot for spent according to Cluster.



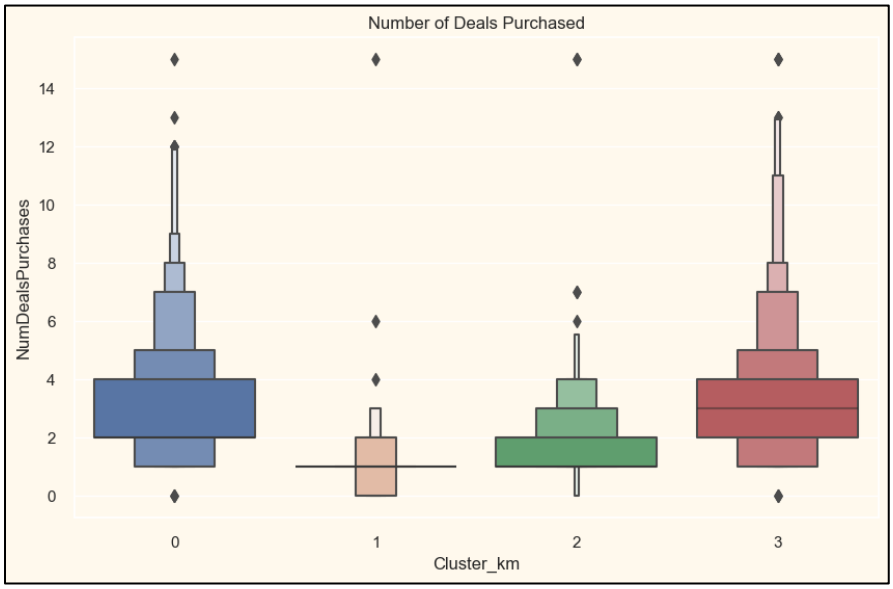
The plot above provides a visual representation indicating that clusters 1 and 2 spending.

We will now delve into an exploration of our past campaigns to assess their performance and efficacy.



The campaigns have not yielded a significant response thus far, with only a limited number of participants overall. Notably, no participant has engaged with all five campaigns. This suggests a need for more targeted and meticulously planned campaigns to stimulate sales.

Next, I will Plot the number of deals purchased.

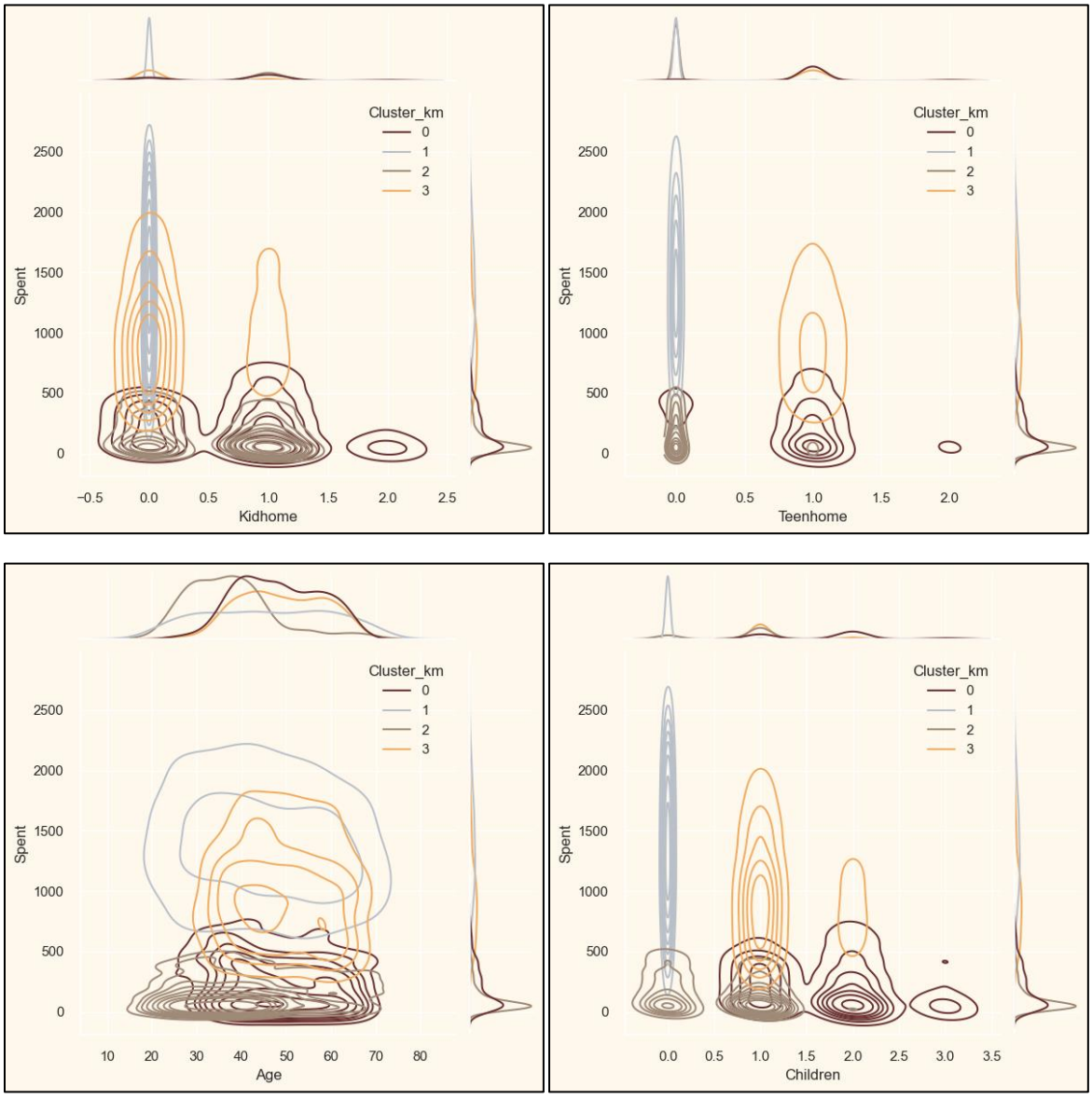


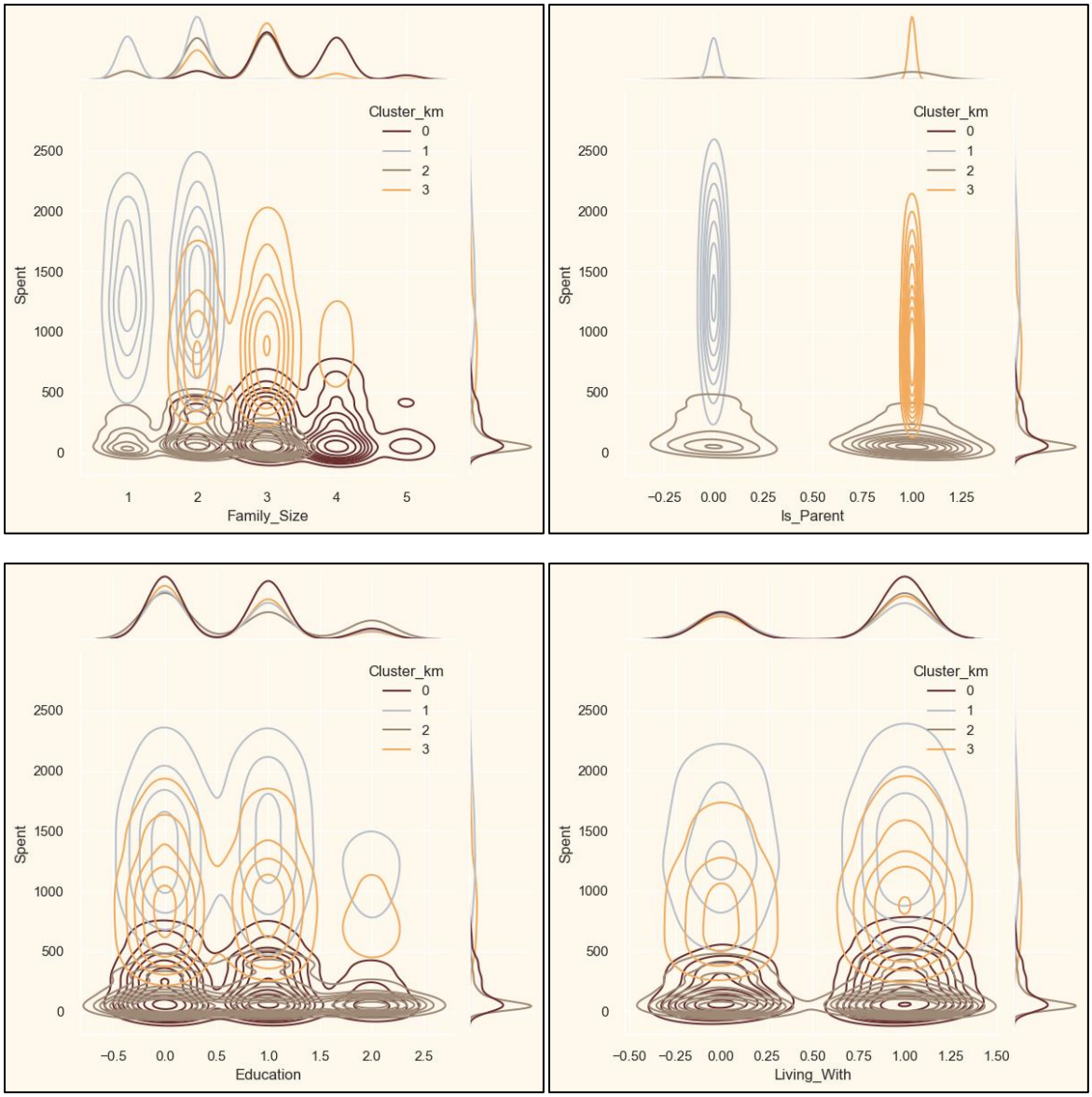
The deals have shown promising outcomes. It has the best outcome with Cluster 0 and Cluster 3. However, our star customers Cluster 1 are not much into the deals. Nothing seems to attract Cluster 2 overwhelmingly.

Profiling

Now we've created the clusters and examined their purchase behavior. Let's check who's in these clusters. For that, we will profile the clusters formed and determine who is our star customer and who needs additional attention from the retail store's marketing team.

To decide, I will plot some of the features that are indicative of the customer's characteristics concerning the cluster they are in. I will draw inferences based on the outcomes.




Cluster_km
0
1
2
3
Spent
2500
2000
1500
1000
500
0
1
2
3
4
5
Family_Size
Cluster_km
0
1
2
3
Spent
2500
2000
1500
1000
500
0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
Is_Parent
Cluster_km
0
1
2
3
Spent
2500
2000
1500
1000
500
0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Education
Cluster_km
0
1
2
3
Spent
2500
2000
1500
1000
500
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
Living_With

Conclusion

Cluster 0:

- Approximately 25 to 70 aged people.
- The majority have two children at home.
- At the max have 5 members in the family and at least 2. (5 family members fall into this Cluster).
- Some have one or two kids, and some do not have any kids at home.
- Low spending and low-income group.

Cluster 1:

- They are not parents, and they don't have children.
- The family has 1 or 2 members.
- Approximately 20 to 75 aged people.
- Some have one kid, and some do not have any kids at home.
- The majority have only one child at home.
- High spending and high-income group.

Cluster 2:

- The majority of these people are parents.
- At the max have 3 members in the family and at least 1.
- Span all ages.
- The majority have a child at home.
- Low spending and low-income group.

Cluster 3:

- They are parents.
- Approximately 30 to 70 aged people.
- At the max have 4 members in the family and at least 2.
- Some have one kid, and some do not have any kids at home.
- The majority have a child at home.
- Average spending and low-income group.

Model Comparison

In the final step, get the Silhouette Score for both k-means and agglomerative clustering models. The Silhouette Score is a metric used to evaluate the quality of clusters formed by clustering algorithms.

```
Silhouette Scores:
K-Means: 0.3615
Agglomerative : 0.3531
```

- ☆ The K-Means algorithm's Silhouette Score of 0.3615 indicates a moderately well-performing cluster, with a score closer to 1 indicating dense and well-separated clusters.
- ☆ The Silhouette Score for Agglomerative Clustering is 0.3531, slightly lower than K-Means. This score indicates well-defined clusters but may have less cohesion or separation compared to K-Means. A score closer to 1 indicates better clustering performance. Agglomerative Clustering performs similarly to K-Means

Relating Clusters to the Problem

- **Understanding Customer Preferences:** By analyzing the clusters, businesses can gain a deeper understanding of customer preferences and behaviors. For example, Cluster 1 might consist of affluent, middle-aged customers who prefer luxury products, while Cluster 2 could comprise young, budget-conscious individuals who prioritize discounts and promotions.
- **Tailoring Marketing Strategies:** Each cluster represents a different target audience with distinct preferences and needs. Businesses can tailor their marketing strategies and product offerings to cater to the specific preferences of each cluster. For instance, Cluster 3, consisting of married customers with children, might respond well to family-oriented promotions and bundled deals.
- **Enhancing Customer Engagement:** Understanding the characteristics of each cluster enables businesses to engage with customers more effectively. For instance, Cluster 4, comprising tech-savvy millennials, might prefer personalized online experiences and targeted digital marketing campaigns over traditional advertising channels.
- **Optimizing Product Development:** Clustering analysis can also provide insights into product development opportunities. By identifying clusters with unmet needs or preferences, businesses can innovate and develop new products or services tailored to specific customer segments, thereby driving customer satisfaction and loyalty.

Actionable Insights:

- ❖ Allocate marketing resources effectively by targeting high-value clusters with tailored campaigns.
- ❖ Develop personalized promotions and product recommendations to enhance customer engagement and loyalty.
- ❖ Optimize pricing and discount strategies based on the preferences of each cluster.
- ❖ Enhance customer service and support offerings to address the unique needs of different customer segments.
- ❖ Continuously monitor and analyze customer behavior to adapt strategies and offerings in response to changing preferences and market dynamics.

Regression – House Price Prediction

Introduction

The house price dataset presents an intriguing challenge, aiming to predict housing prices based on a variety of factors such as house area, number of bedrooms, furnishing status, proximity to main roads, presence of guestrooms and basements, and availability of air conditioning. Despite its small size, the dataset poses significant complexity due to strong multicollinearity among its variables. This complexity adds layers of difficulty to the task of building accurate predictive models.

Research Questions

- How do various features such as house area, number of bedrooms, and presence of amenities influence housing prices?
- What is the impact of multicollinearity on the accuracy of predictive models for housing prices?
- Can we identify the most influential factors in determining housing prices?
- How does the proximity to main roads affect property values?
- Are there any interactions among different features that significantly affect housing prices?

Data Mining Problem and Objectives

Problem

The data mining problem revolves around ***building regression models*** to predict housing prices accurately despite the presence of strong multicollinearity among the predictor variables.

Objectives

- Develop regression models that accurately predict housing prices based on the provided features.
- Identify and mitigate the impact of multicollinearity on model performance.
- Determine the most influential factors affecting housing prices.
- Investigate interactions among predictor variables to improve model accuracy.
- Assess the impact of proximity to main roads on property values.

Formulation of the Problem

Given a dataset containing information about various features of houses, as well as their corresponding prices, the task is to build regression models that can predict housing prices accurately. However, the presence of strong multicollinearity among predictor

variables presents a challenge in developing robust models. Therefore, data preprocessing techniques such as feature selection, dimensionality reduction, and regularization will be employed to mitigate multicollinearity and improve model performance. Additionally, interactions among predictor variables will be explored to uncover hidden patterns that influence housing prices. The ultimate goal is to develop regression models that provide accurate predictions while effectively handling the complexities inherent in the dataset.

Implementation

Data Wrangling.

```
# load the dataset
house_data = pd.read_csv('Data/Housing.csv')
print(house_data.shape)
house_data.head()
```

```
(545, 13)
```

	price	area	bedrooms	bathrooms	stories	mainroad
0	13300000	7420	4	2	3	yes
1	12250000	8960	4	4	4	yes
2	12250000	9960	3	2	2	yes
3	12215000	7500	4	2	2	yes
4	11410000	7420	4	1	2	yes

The House Price dataset comprises a total of 545 entries and encompasses 13 distinct features. Each entry represents a unique observation within the dataset, while the features provide various attributes or characteristics related to house prices.

```
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           540 non-null    float64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: float64(1), int64(5), object(7)
```

```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           5
prefarea          0
furnishingstatus  0
dtype: int64
```

From the observed outputs, it's evident that the dataset contains null values. Before model building, it's imperative to undertake data preprocessing and feature engineering tasks.

Data Preprocessing and Feature Engineering

Handling with missing values

Within the dataset, we identified the presence of five missing values. To facilitate data integrity and transparency, it's essential to display these specific records with missing information.

```
# display the null records
house_data[house_data.isnull().any(axis=1)]
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking
7	10150000	16200	5	3	2	yes	no	no	no	no	NaN
14	9240000	7800	3	2	2	yes	no	no	no	no	NaN
45	7560000	6000	3	2	3	yes	no	no	no	yes	NaN
49	7420000	7440	3	2	1	yes	yes	yes	no	yes	NaN
79	6650000	6000	3	2	3	yes	yes	no	no	yes	NaN

In our dataset comprising over 500 records, five records contain missing values. We have few amounts of records so; we can't ignore null records. We can use the imputation techniques to handle these null records.

Let's extract unique value counts of the parking feature.

```
house_data['parking'].value_counts()
```

```
0.0    294
1.0    126
2.0    108
3.0     12
Name: parking, dtype: int64
```

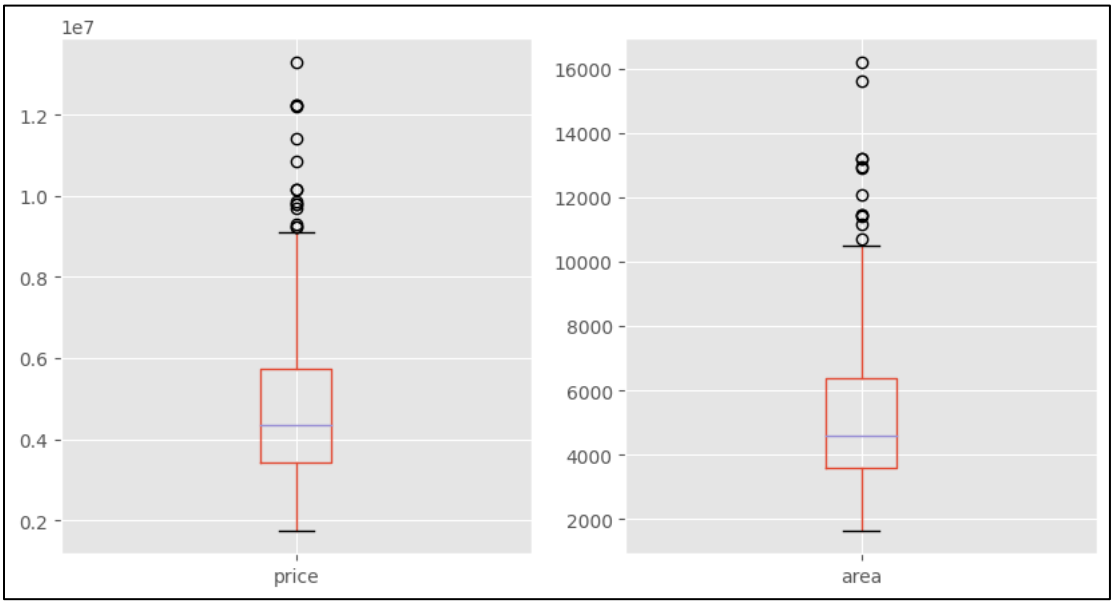
The mode of the parking feature is 0 (no parking). So, let's impute missing values with 0.

```
# impute with 0
house_data['parking'].fillna(0, inplace=True)
house_data['parking'].isnull().sum()
```

```
0
```

Handling with outliers

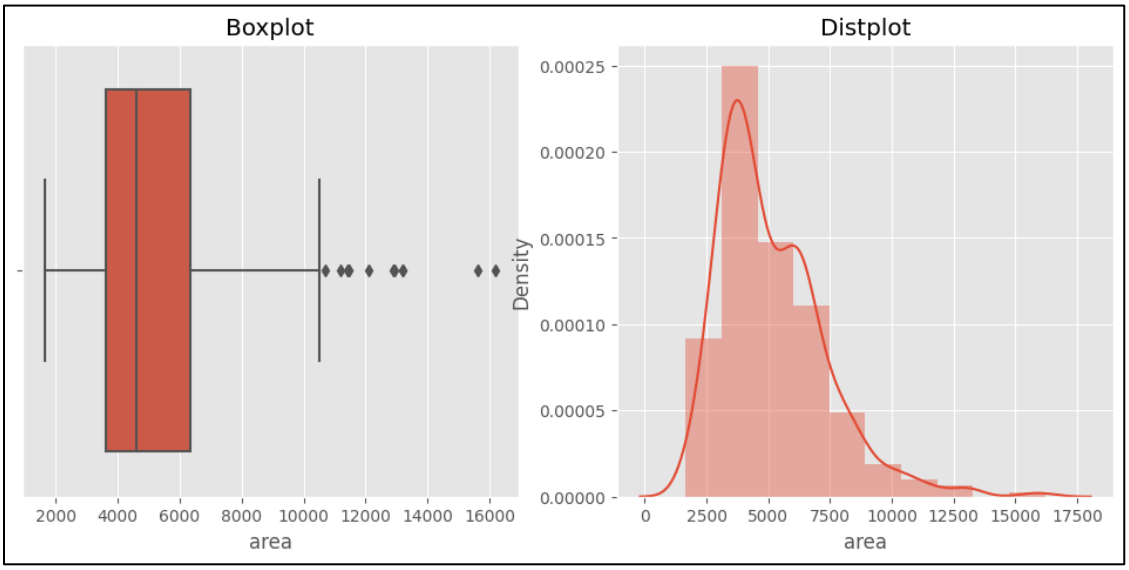
The next step is to address outliers. To begin, we'll generate box plots for key features including ***'price' and 'area'***. These visualizations will provide insights into the distribution of data and highlight any potential outliers present within these features.



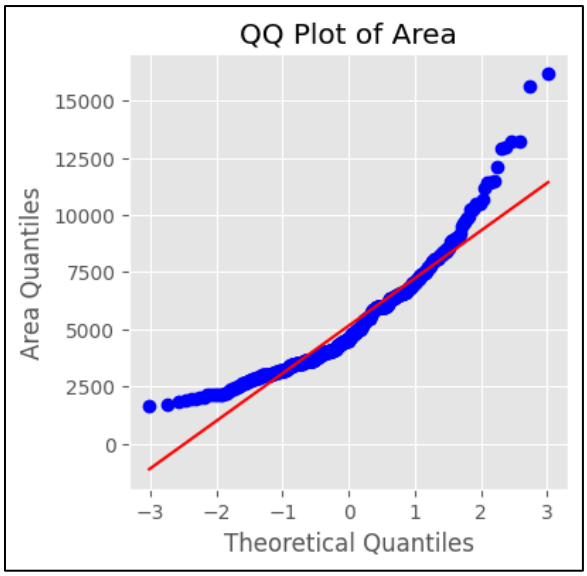
Upon reviewing the box plot output, it's evident that outliers exist within the 'price' and 'area' variables.

To address these outliers effectively, we first need to detect them and establish both the lower and upper boundaries of the area.

```
IQR of area :  2760.0
Upper Limit of area :  10500.0
Lower Limit of area :  -540.0
Number of Outliers Records :  12
```



Upon analysis, we've identified 12 records containing outliers, which deviate significantly from the rest of the dataset. Additionally, the distribution of data points for the 'area' variable exhibits a right-skewed distribution. Let's do the Q-Q plot and Shapiro-Wilk test for the area feature and confirm it.



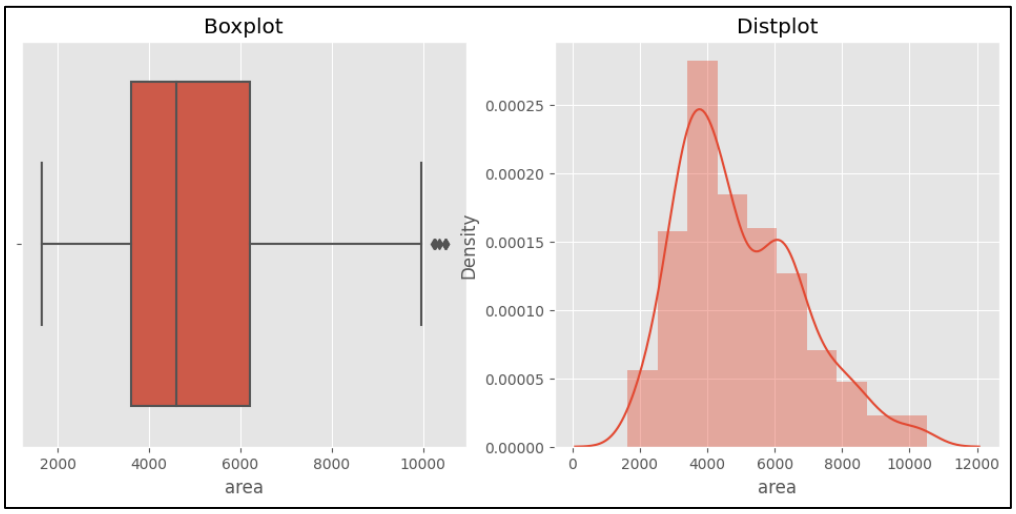

```
Shapiro-Wilk Test for Area
Shapiro-Wilk Test Statistic: 0.9113
p-value: 0.00
The data in area might not be normally distributed.
```

Based on the above results, we can say the area feature is not normally distributed.

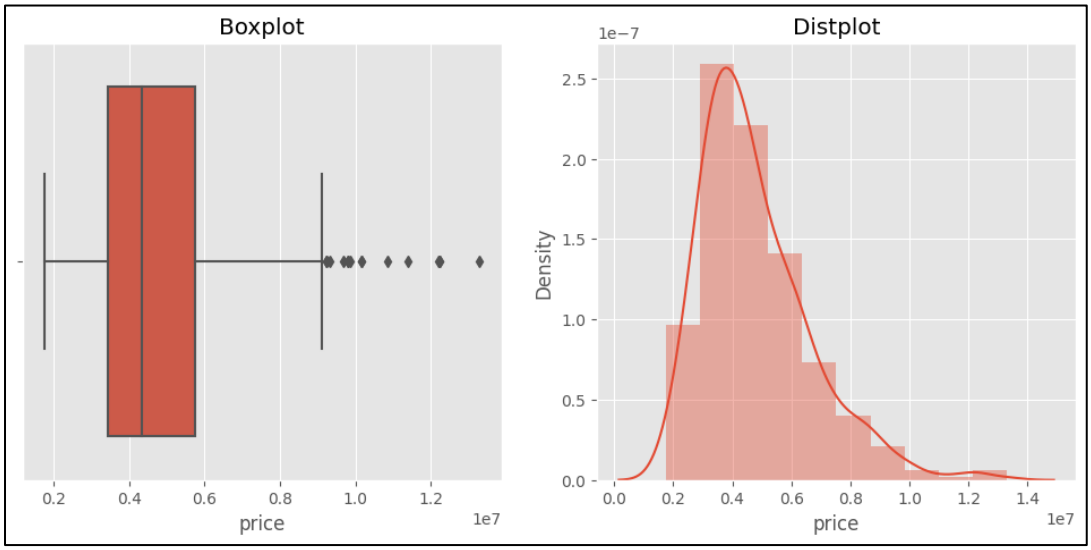
The median imputation is a robust technique for handling outliers of right-skewed data. Let's impute the area outliers with median value.

```
# Impute outliers with median
house_data.loc[house_data['area'] > 10500, 'area'] = house_data['area'].median()
```

After handling the outliers.



Next, we'll examine the outliers and data distribution of the target variable 'Price' feature.



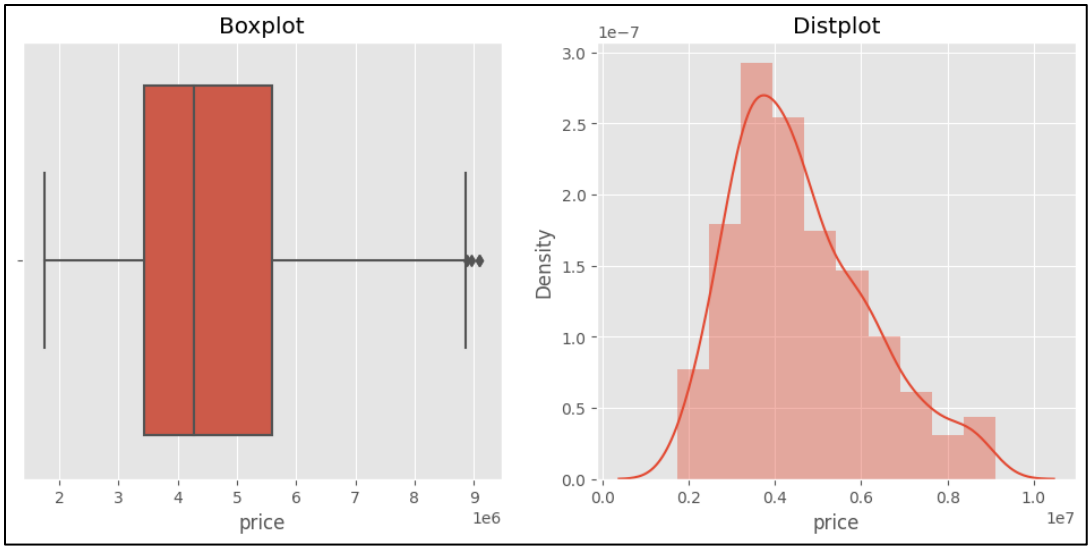
The plots reveal outliers in the 'Price' feature, with a right-skewed distribution. We'll detect outliers, establish upper and lower boundaries, and count outlier records for further analysis.

```
IQR of price :  2310000.0
Upper Limit of price :  9205000.0
Lower Limit of price :  -35000.0
Number of Outliers Records :  15
```

Among the 15 records in the dataset, outliers have been identified. Given that 'Price' serves as a target column for modeling, it's crucial to maintain its accuracy. Therefore, the most effective approach to handling these outliers is by removing the corresponding records from the original dataset.

```
# remove outliers
house_data = house_data[~(house_data['price'] > upper_price)]
house_data.shape
```

```
(530, 13)
```



Following the removal of outliers, we'll visualize the distribution of the 'price' variable using a boxplot. Despite the outlier removal process, a few outliers are still present within the data. Rather than further adjustments at this stage, we'll retain these outliers and proceed to the subsequent phase of analysis.

Exploratory Data Analysis (EDA)

Having completed the data cleansing and Feature Engineering phases, we're now ready to delve into the analysis of the customer personality data set.

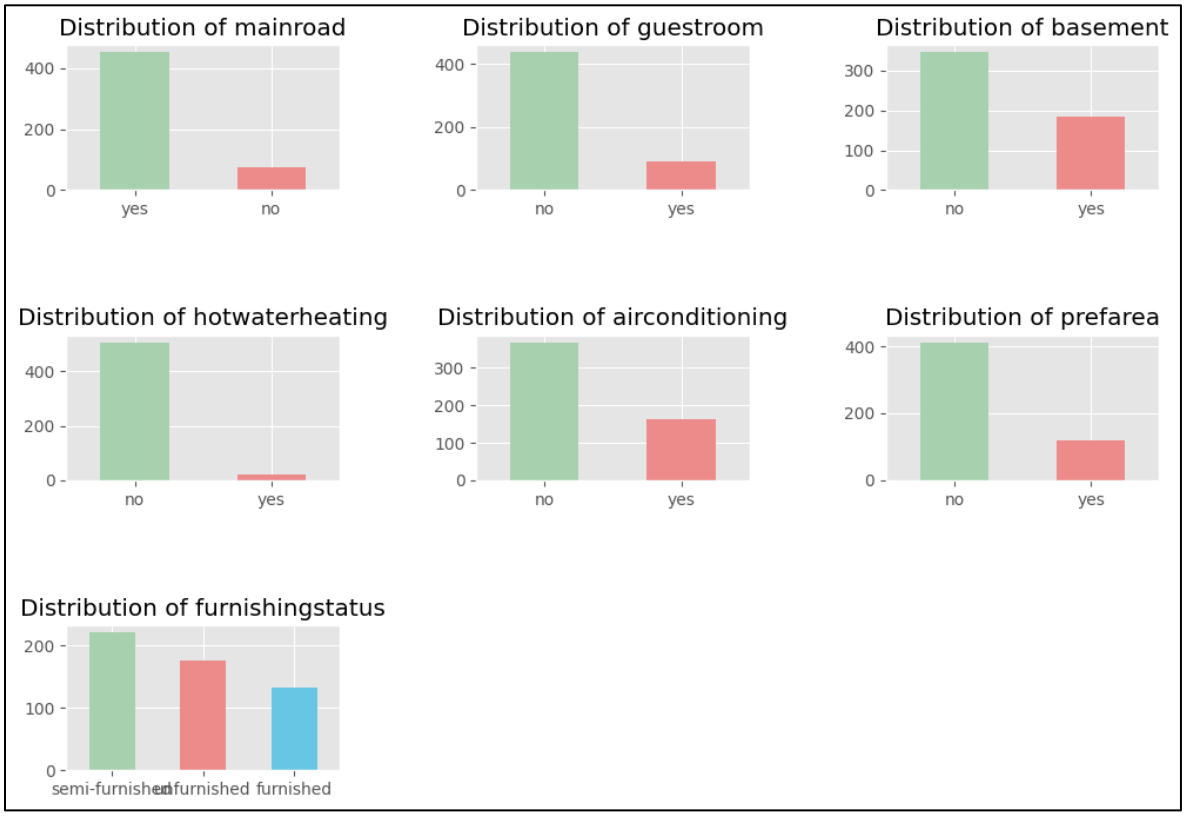
Data Analysis

```
The number of values for feature price :208
The number of values for feature area :268
The number of values for feature bedrooms :6 -- [1 2 3 4 5 6]
The number of values for feature bathrooms :3 -- [1 2 3]
The number of values for feature stories :4 -- [1 2 3 4]
The number of values for feature mainroad :2 -- ['no' 'yes']
The number of values for feature guestroom :2 -- ['no' 'yes']
The number of values for feature basement :2 -- ['no' 'yes']
The number of values for feature hotwaterheating :2 -- ['no' 'yes']
The number of values for feature airconditioning :2 -- ['no' 'yes']
The number of values for feature parking :4 -- [0. 1. 2. 3.]
The number of values for feature prefarea :2 -- ['no' 'yes']
The number of values for feature furnishingstatus :3 -- ['furnished' 'semi-furnished' 'unfurnished']
```

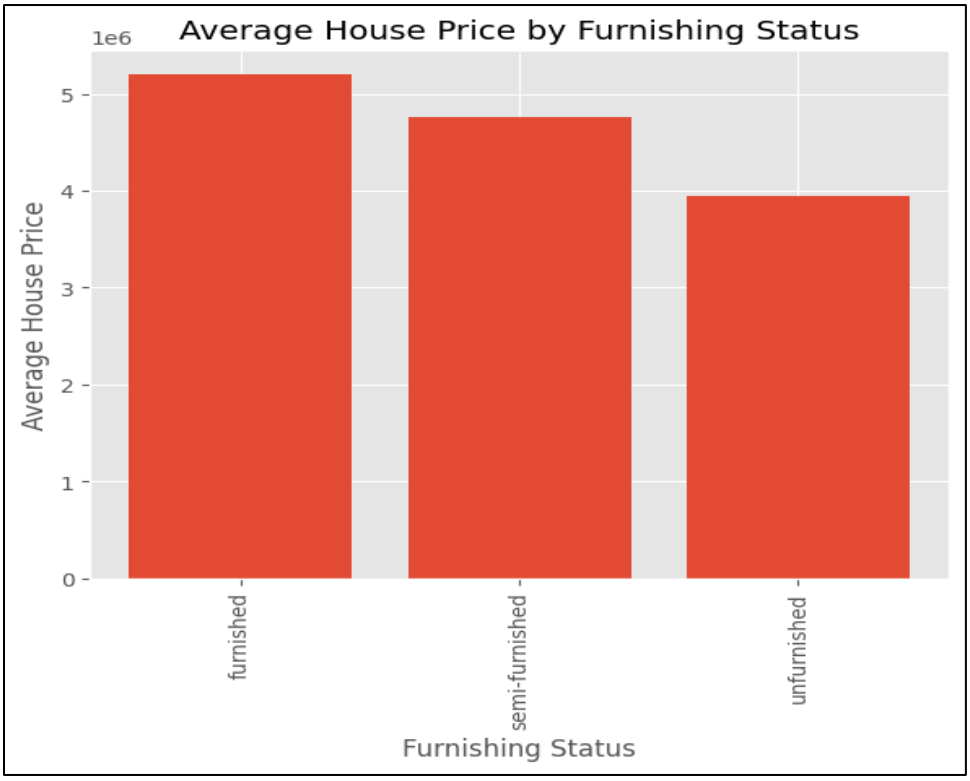
We need to separate them into two groups for further analysis.

```
Number of Categorical Variables :  7
Categorical Variables :  Index(['mainroad', 'guestroom', 'basement', 'hotwaterheating',
       'airconditioning', 'prefarea', 'furnishingstatus'],
      dtype='object')
Number of Numerical Variables :  6
Numerical Variables :  Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking'], dtype='object')
```

Distribution of Categorical Features

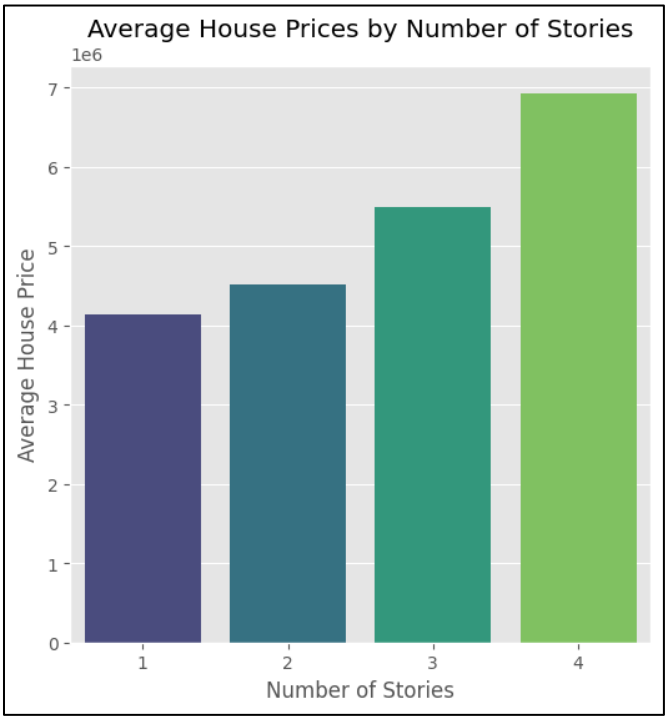


Average House Price vs Furnished Status



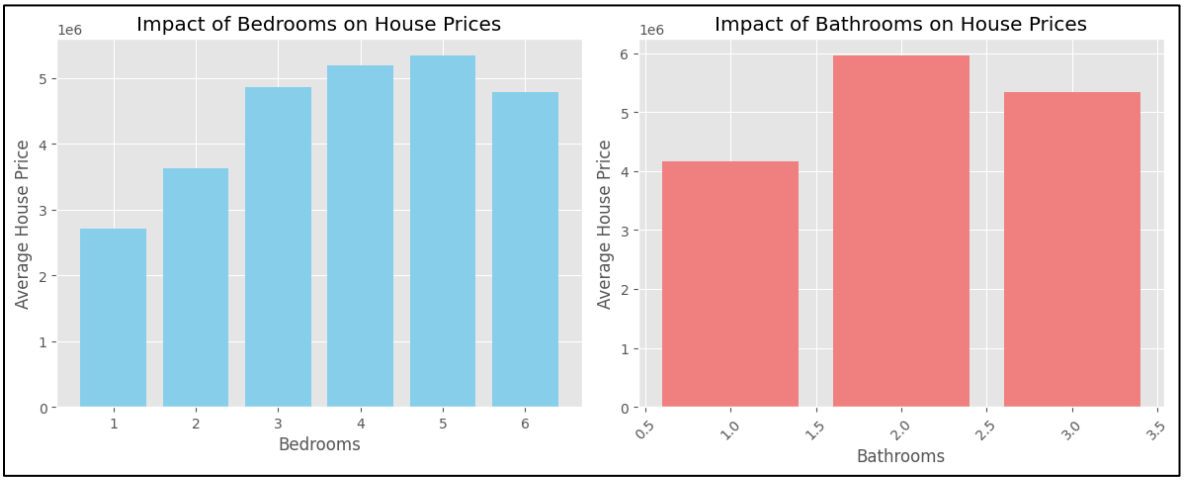
From the presented plot, it's evident that the Furnished house has a notably higher average house price compared to other furnished status.

House Stories Vs Average House Price.



Upon analysis, we notice a substantial price variation linked to the number of stories of the house, indicating its significant influence on house pricing. This observation suggests a positive correlation between the number of stories and house prices. Essentially, as the number of stories increases, there is a corresponding rise in house prices.

Bedrooms and Bathrooms Vs Average House Price



From the depicted bar plots, several insights emerge:

- There seems to be a positive correlation between the number of bedrooms and house prices. This suggests that houses with more bedrooms tend to be more expensive.
- Similar to the left graph, there appears to be a positive correlation between the number of bathrooms and house prices. Houses with more bathrooms seem to be more expensive on average.

Handling with Categorical variables

After completing data cleansing, feature engineering, outlier treatment, and Exploratory Data Analysis (EDA). Now, as we move forward, it's essential to address the categorical features present in the house price dataset.

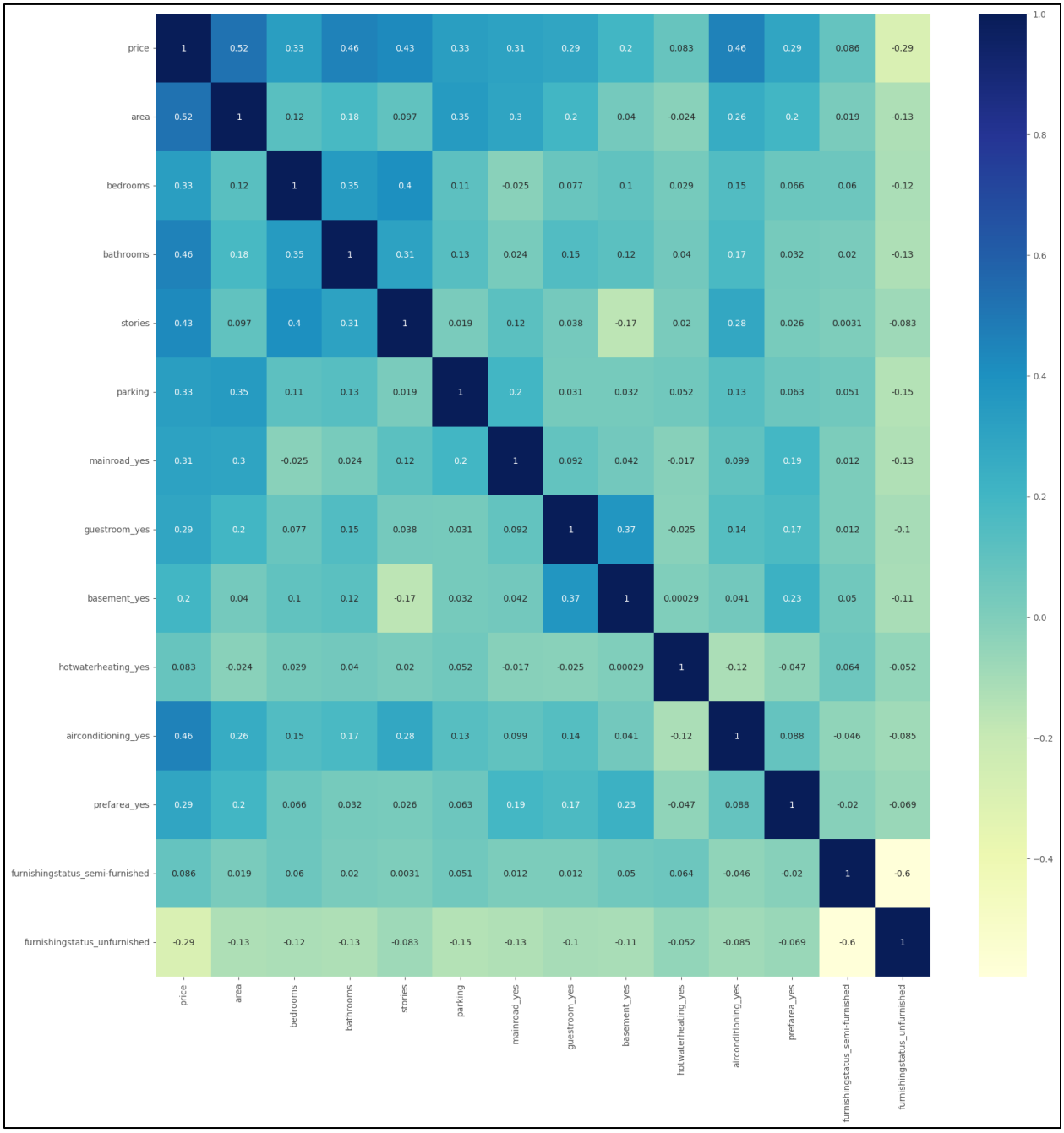
OneHot encoding.

The house price dataset has 6 categorical features, and all are nominal. So, let's do the label encoding using the ***OneHot encoding*** technique.

```
house_encoded = pd.get_dummies(house_data, columns=categorical,
                               drop_first=True)
```

Correlation Heatmap

We'll examine the correlation among the features to understand how they relate to each other. Correlation analysis helps identify patterns and dependencies within the dataset.



Based on the above heatmap we can conclude the following:

- All the features have a positive correlation with Price. The Area feature has a strong positive correlation with price.

Next, we'll conduct a correlation analysis to identify features with a correlation coefficient above the 0.8 threshold.

```
# call the function with 0.8 threshold
cor_feature(house_encoded, 0.8)
```

```
set()
```

Notably, we've not found any features with high correlation. So, all the features are important to house price prediction.

Independent and Dependent Features

The next step is to identify the independent (X) and dependent (y) features for model prediction. Here 'Price' is a dependent feature, and the rest are independent features.

```
# Independent features
X = house_encoded.drop(['price'], axis=1)

# Dependent feature
y = house_encoded['price']
```

Data Scaling

Moving forward, we'll perform data scaling as the next step. This involves standardizing the scale of all features within the dataset using the ***MinMax scaler*** technique.

```
sc = MinMaxScaler()
sc.fit(X)
X_scaled = pd.DataFrame(sc.transform(X),
                        columns = X.columns)
X_scaled.head()
```

	area	bedrooms	bathrooms	stories	parking	mainroad_yes
0	0.491525	0.6	0.0	0.333333	0.666667	1.0
1	0.559322	0.6	0.5	0.333333	0.333333	1.0
2	0.774011	0.4	0.5	1.000000	0.666667	1.0
3	0.333333	0.4	0.5	0.333333	0.666667	1.0
4	0.538983	0.4	0.5	0.333333	0.333333	1.0

Model Training and Prediction

For model training and prediction, first Initialize the ***KFold cross-validation***. After that through the folds looping split the data into training and testing. Then perform regression (Linear, Lasso, Ridge) for each fold and make a prediction.

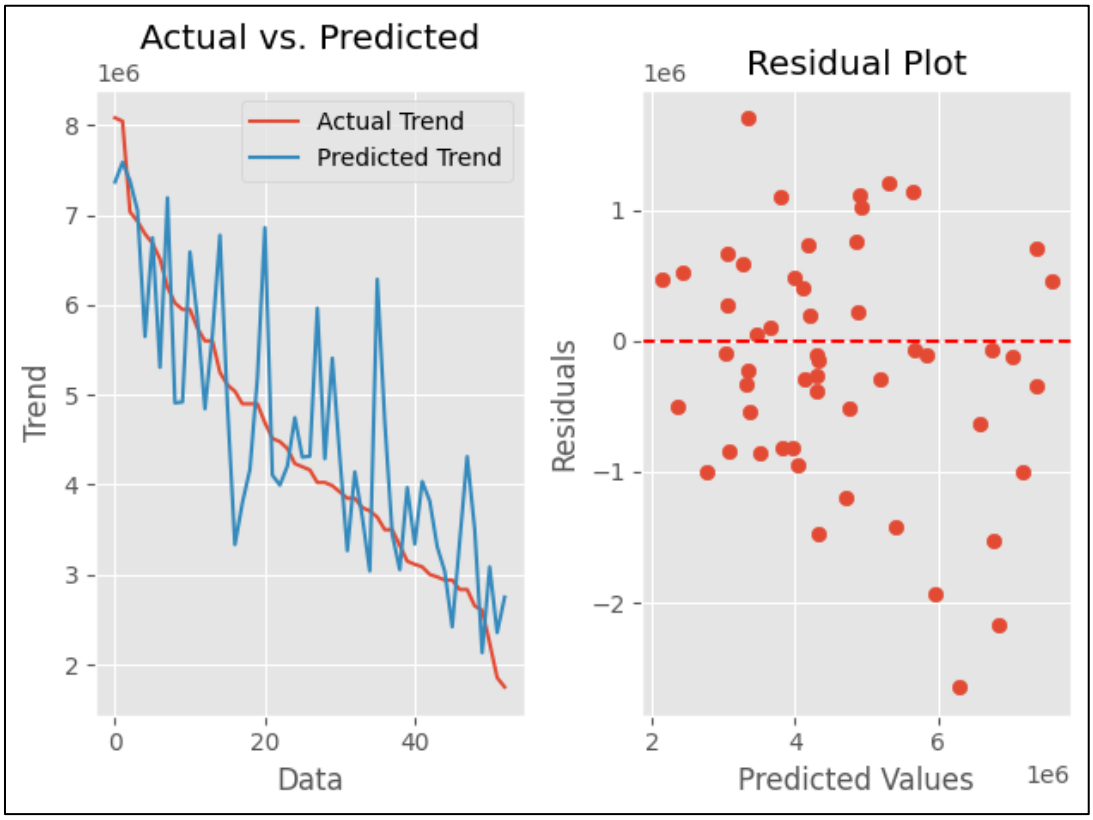
```
# KFold object for splitting data
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Iterate through each fold
for train_index, test_index in kf.split(X_scaled):
    # Split data into training and testing sets for this fold
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Linear Regression
    linear_model = LinearRegression()
    linear_model.fit(X_train, y_train)
    y_pred_linear = linear_model.predict(X_test)

    # Lasso Regression
    lasso_model = Lasso()
    lasso_model.fit(X_train, y_train)
    y_pred_lasso = lasso_model.predict(X_test)

    # Ridge Regression
    ridge_model = Ridge()
    ridge_model.fit(X_train, y_train)
    y_pred_ridge = ridge_model.predict(X_test)
```

Model Evaluation

We will conduct the model evaluation for linear, lasso, and ridge regression techniques using various performance metrics. These metrics include ***Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared score, and residual plots***.

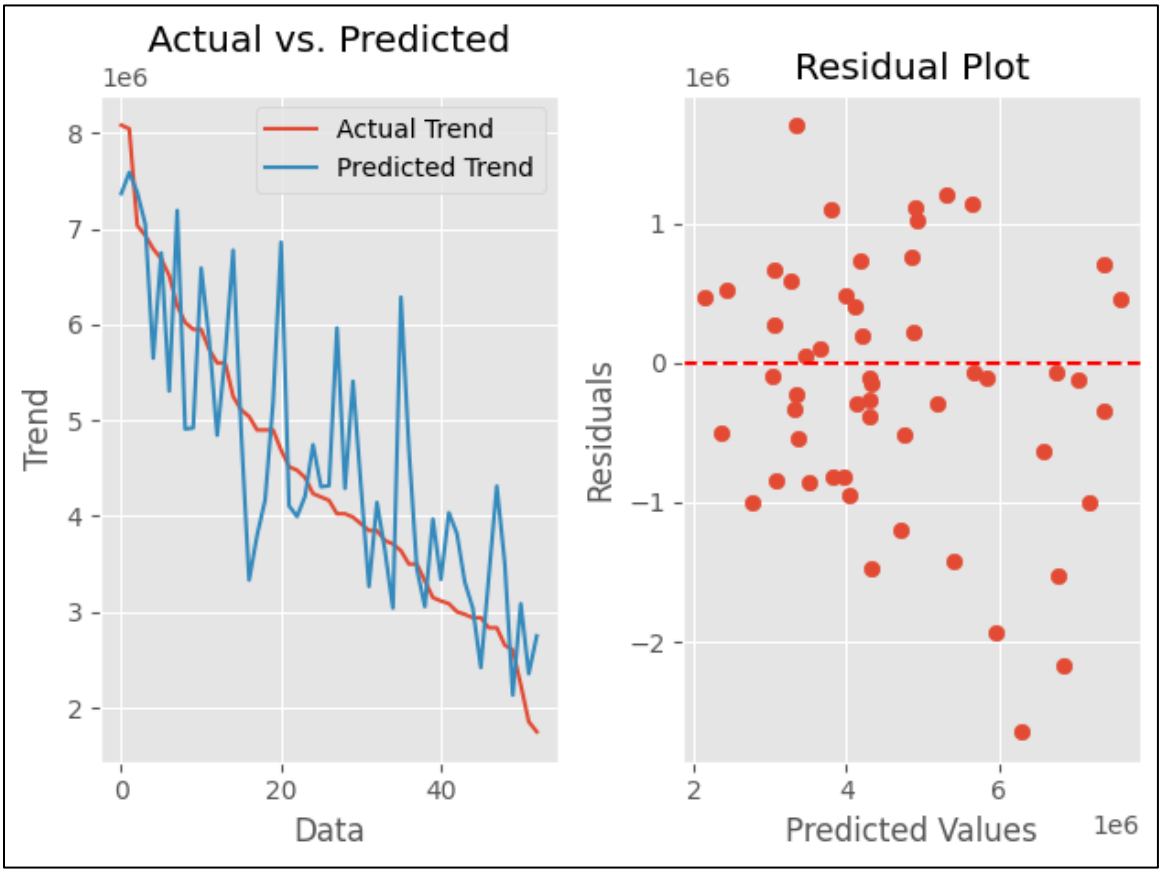
Linear Regression

```
Coefficients: [ 2.41515982e+02  9.12410830e+04  7.68531583e+05  4.43662320e+05
  2.04207857e+05  4.46402292e+05  3.24228636e+05  3.20139144e+05
  1.09424560e+06  8.62467848e+05  5.19841801e+05  6.66742251e+04
 -3.38580869e+05]
Intercept: 373538.2183275316
Mean Absolute Error :  709930.8012881123
Mean Squared Error :  828607800676.851
Root Mean Squared Error :  910278.9686007531
R^2 score :  0.6385420836094562
```



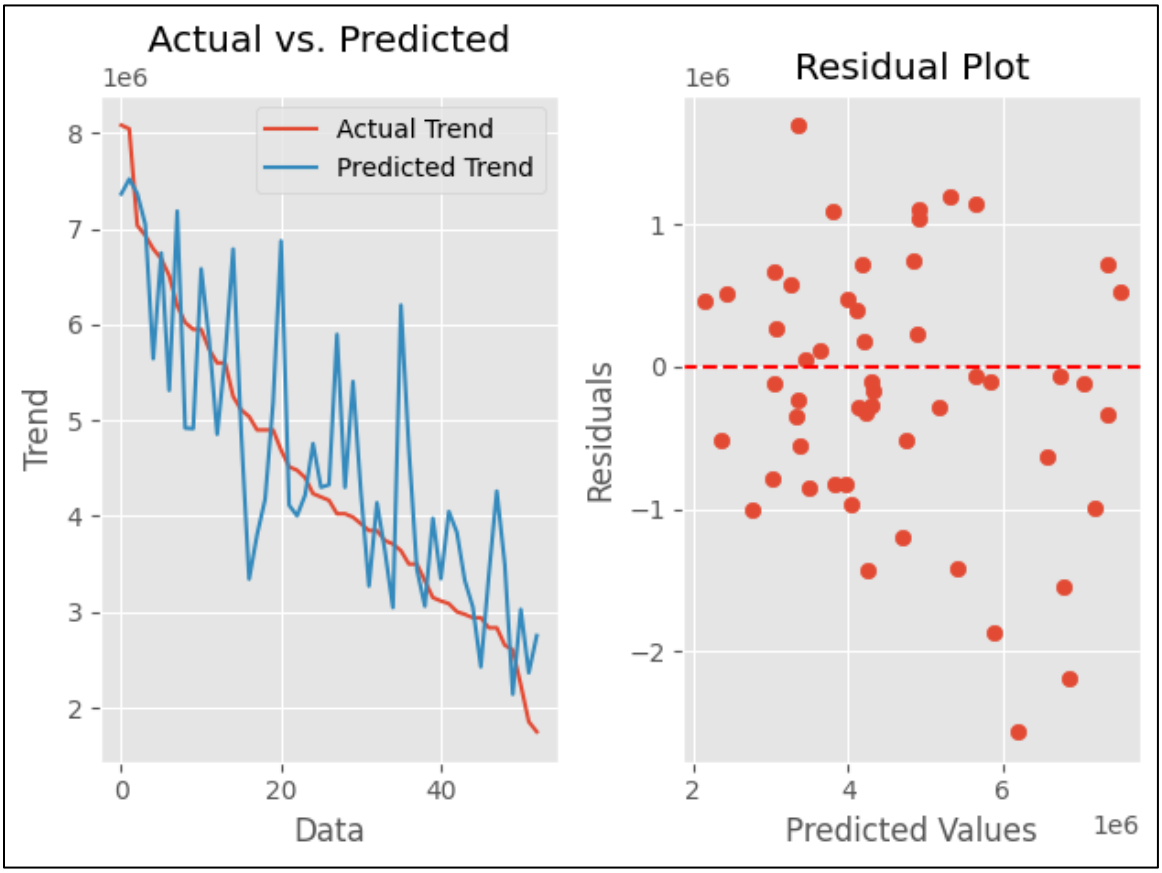
Lasso Regression

```
Coefficients: [ 2.41517557e+02  9.12405004e+04  7.68528500e+05  4.43662811e+05
  2.04207439e+05  4.46394285e+05  3.24223842e+05  3.20138879e+05
  1.09421504e+06  8.62461792e+05  5.19836516e+05  6.66707651e+04
 -3.38580794e+05]
Intercept: 373548.8302306719
Mean Absolute Error :  709928.6373105225
Mean Squared Error :  828599346499.2891
Root Mean Squared Error :  910274.3248599782
R^2 score :  0.6385457715175399
```



Ridge Regression

```
Coefficients: [ 2.43327992e+02  9.29950274e+04  7.61533662e+05  4.45204134e+05
  2.05130505e+05  4.39422781e+05  3.22551222e+05  3.22071352e+05
  1.02592992e+06  8.49897956e+05  5.11772087e+05  6.80286215e+04
 -3.37404310e+05]
Intercept: 377711.62233498786
Mean Absolute Error :  705082.872413793
Mean Squared Error :  811482389978.7563
Root Mean Squared Error :  900823.1735356037
R^2 score :  0.6460125844461719
```



Based on the model evaluation of Linear, Lasso, and Ridge Regression models, the ***Ridge regression*** model achieved slightly better accuracy than Linear and Lasso regression models. So, the Ridge regression model is a good model for house price predictions.

Regression Modeling Results

- ❖ **Model Performance:** We developed multiple regression models using techniques such as linear regression, ridge regression, and lasso regression to predict housing prices. Through rigorous evaluation using metrics like mean squared error (MSE), R-squared, and cross-validation, we identified the best-performing model.
- ❖ **Feature Importance:** Our analysis revealed the most influential factors affecting housing prices. Features such as house area, number of bedrooms, presence of amenities (e.g., air conditioning, basement), and proximity to main roads emerged as significant predictors of property values.
- ❖ **Handling Multicollinearity:** Given the presence of strong multicollinearity among predictor variables, we employed techniques like feature selection to mitigate its impact on model performance. These techniques helped improve the robustness and stability of our regression models.

Actionable Insights

- ☆ **Optimal Pricing Strategy:** By understanding the factors that drive housing prices, real estate professionals can adopt a more data-driven approach to pricing properties. Properties with larger areas, more bedrooms, and desirable amenities are likely to command higher prices in the market.
- ☆ **Investment Opportunities:** Investors can use regression modeling results to identify investment opportunities in the real estate market. Properties located near main roads or with potential for additional amenities (e.g., basement conversion, installation of air conditioning) may offer higher returns on investment.
- ☆ **Targeted Marketing:** Real estate agents can leverage insights from regression models to tailor their marketing strategies. For example, they can highlight specific features of a property (e.g., spacious layout, modern amenities) that are known to appeal to potential buyers and justify the asking price.

Classification – Customer Churn Prediction

Introduction

In today's highly competitive telecommunications industry, retaining customers is critical to long-term growth and profitability. Understanding customer churn, or the phenomena in which customers abandon services, is critical for telco firms. The Telco Customer attrition dataset provides a comprehensive set of customer-related data targeted at predicting attrition over a specified timeframe. This dataset allows us to investigate numerous client qualities and interactions with services to design customer retention tactics.

Research Questions

1. What factors contribute most to customer churn?
   - Identify which specific services or customer attributes have the highest correlation with churn.
2. Are there any patterns in customer demographics that are associated with higher churn rates?
   - Explore if gender, age range, or the presence of partners and dependents impact churn.
3. How does contract type influence customer retention?
   - Examine if customers on different contract types (month-to-month, one year, two years) exhibit varying churn rates.
4. Does the payment method affect customer loyalty?
   - Investigate whether customers using specific payment methods are more or less likely to churn.

Data Mining Problem and Objectives

Problem

The data mining task at hand is to ***create a prediction model*** that can identify customers who are likely to churn.

Objectives

The primary objective of using previous customer data is to predict which customers are likely to terminate their services shortly. This predictive capacity enables telcos to proactively intervene with targeted retention campaigns, reducing churn and sustaining revenue streams.

Problem Formulation

Given the Telco Customer Churn dataset, the problem can be reduced to a binary classification task: predict whether or not a customer will churn within a given timeframe. Each instance in the dataset represents a distinct client, and the goal variable, "Churn," shows whether the customer left within the previous month (1 for churn, 0 for kept). The goal is to create a classification model that can accurately forecast the possibility of churn based on customer qualities, service subscriptions, and demographic data.

Using supervised learning approaches such as logistic regression, decision trees, random forests, or gradient boosting, we hope to build a model that can generalize well to new data and successfully distinguish between churned and kept clients. Feature engineering, model selection, and hyperparameter tuning will be essential elements in the predictive modeling process to maximize performance metrics such as accuracy, precision, recall, and F1 score.

Approach

- **Input Features:** Services signed up for (phone, multiple lines, internet, etc.) Customer account information (tenure, contract type, payment method, etc.), Demographic information (gender, age range, partners, dependents)
- **Output:** Churn column indicating whether the customer churned (1) or not (0)
- **Data Preprocessing:** Handle missing values, encode categorical variables, and scale numerical features as necessary.
- **Split the dataset** into training and testing sets.
- **Model Selection:** Choose appropriate classification algorithms (e.g., logistic regression, decision trees, random forests, or gradient boosting) for prediction.
- **Model Evaluation:** Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1-score on the test set.

Implementation

Data Wrangling.

```
# load the customer dataset
customer_data = pd.read_csv('Data/Telco-Customer-Churn.csv')
print(customer_data.shape)
customer_data.head()
```

```
(7043, 21)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service
1	5575-GNVDE	Male	0	No	No	34	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service
4	9237-HQITU	Female	0	No	No	2	Yes	No

The Telco customer churn dataset consists of 7043 records, each representing a unique customer, and encompasses 21 columns, also known as variables or features. These columns contain various attributes related to customer demographics, Services that each customer has signed up for, Customer account information, and churn status.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

The output above indicates that there are no missing values in the Customer dataset, which is a positive sign for data integrity. However, it's noted that there is an inconsistency in the data type of the ***"TotalCharges"*** feature. While the feature should ideally be of float data type, it's incorrectly classified as an Object in the dataset. This discrepancy will be addressed during the Data Preprocessing phase.

Now, let's Investigate all the elements within each Feature of the telco customer dataset.

```
The number of values for feature customerID :7043
The number of values for feature gender :2 -- ['Female' 'Male']
The number of values for feature SeniorCitizen :2 -- [0 1]
The number of values for feature Partner :2 -- ['No' 'Yes']
The number of values for feature Dependents :2 -- ['No' 'Yes']
The number of values for feature tenure :73
The number of values for feature PhoneService :2 -- ['No' 'Yes']
The number of values for feature MultipleLines :3 -- ['No' 'No phone service' 'Yes']
The number of values for feature InternetService :3 -- ['DSL' 'Fiber optic' 'No']
The number of values for feature OnlineSecurity :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature OnlineBackup :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature DeviceProtection :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature TechSupport :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature StreamingTV :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature StreamingMovies :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature Contract :3 -- ['Month-to-month' 'One year' 'Two year']
The number of values for feature PaperlessBilling :2 -- ['No' 'Yes']
The number of values for feature PaymentMethod :4 -- ['Bank transfer (automatic)' 'Credit card (automatic)' 'Electronic check'
 'Mailed check']
The number of values for feature MonthlyCharges :1585
The number of values for feature TotalCharges :6531
The number of values for feature Churn :2 -- ['No' 'Yes']
```

The output above presents detailed information regarding the unique value counts for each feature in the dataset, along with the unique values themselves (displayed if the count is less than 20). Upon analysis, we observe that the ***'Customer ID'*** feature has unique value counts equivalent to the total number of records, indicating its distinctiveness for each customer and the ***'Senior Citizen'*** feature has binary values so, should change those values into 'Yes' or 'No' for analysis purposes. Features such as ***'Tenure', 'Monthly Charges', and 'Total Charges'*** are identified as numerical, signifying quantitative data. Conversely, the remaining features are categorized as categorical, representing qualitative attributes.

Data Preprocessing and Feature Engineering

With data quality checks completed, we're now ready to proceed with the data preprocessing and feature engineering of the telco customer dataset.

Data Cleansing

The analysis of data quality reveals that the ***'customerID'*** feature consists of 7043 unique values, which aligns with the total number of records in the dataset. As each entry in the ***'customerID'*** column is unique, it does not provide predictive power or contribute to pattern recognition. Consequently, it's advisable to remove the ***'customerID'*** column from the dataset to streamline analysis and enhance model performance.

```
# Remove Customer ID
customer_data.drop(['customerID'], axis=1, inplace=True)
```

The ***'TotalCharges'*** feature, ideally representing a numerical value, is currently stored as an object datatype within the dataset. To ensure proper analysis, we need to convert it into a numerical datatype. After this conversion, we'll examine the ***'TotalCharges'*** column to identify any null or missing records.

```
# 'TotalCharges' convert into numerical
customer_data['TotalCharges'] = pd.to_numeric(customer_data['TotalCharges'],
                                              errors='coerce')

# check any null values in 'TotalCharges'
customer_data['TotalCharges'].isna().sum()
```

```
11
```

Upon inspection, we identify 11 null records in this column. In the context of handling missing values, various strategies exist, including removing records with missing values, imputing mean, or median values, and more. Let's see the null records.

gender	SeniorCitizen	Partner	Dependents	tenure	TotalCharges	Churn
Female	0	Yes	Yes	0	NaN	No
Male	0	No	Yes	0	NaN	No
Female	0	Yes	Yes	0	NaN	No
Male	0	Yes	Yes	0	NaN	No
Female	0	Yes	Yes	0	NaN	No
Male	0	Yes	Yes	0	NaN	No
Male	0	Yes	Yes	0	NaN	No

All the '***TotalCharges'*** null records with no churn and 0 tenure. These are likely new customers who have just started using the service within the past month and have not been charged yet, hence there is no way to determine their churn status. Since there are only 11 rows, we will just impute the null values with 0.

```
# impute null records
customer_data['TotalCharges'].fillna(0, inplace=True)
```

The ***'Senior Citizen'*** is a categorical feature, and that feature contains binary values. The 'Senior Citizen' value 1 is represented 'Yes' and 0 is represented 'No'. Let's replace binary values with 'Yes' or 'No'.

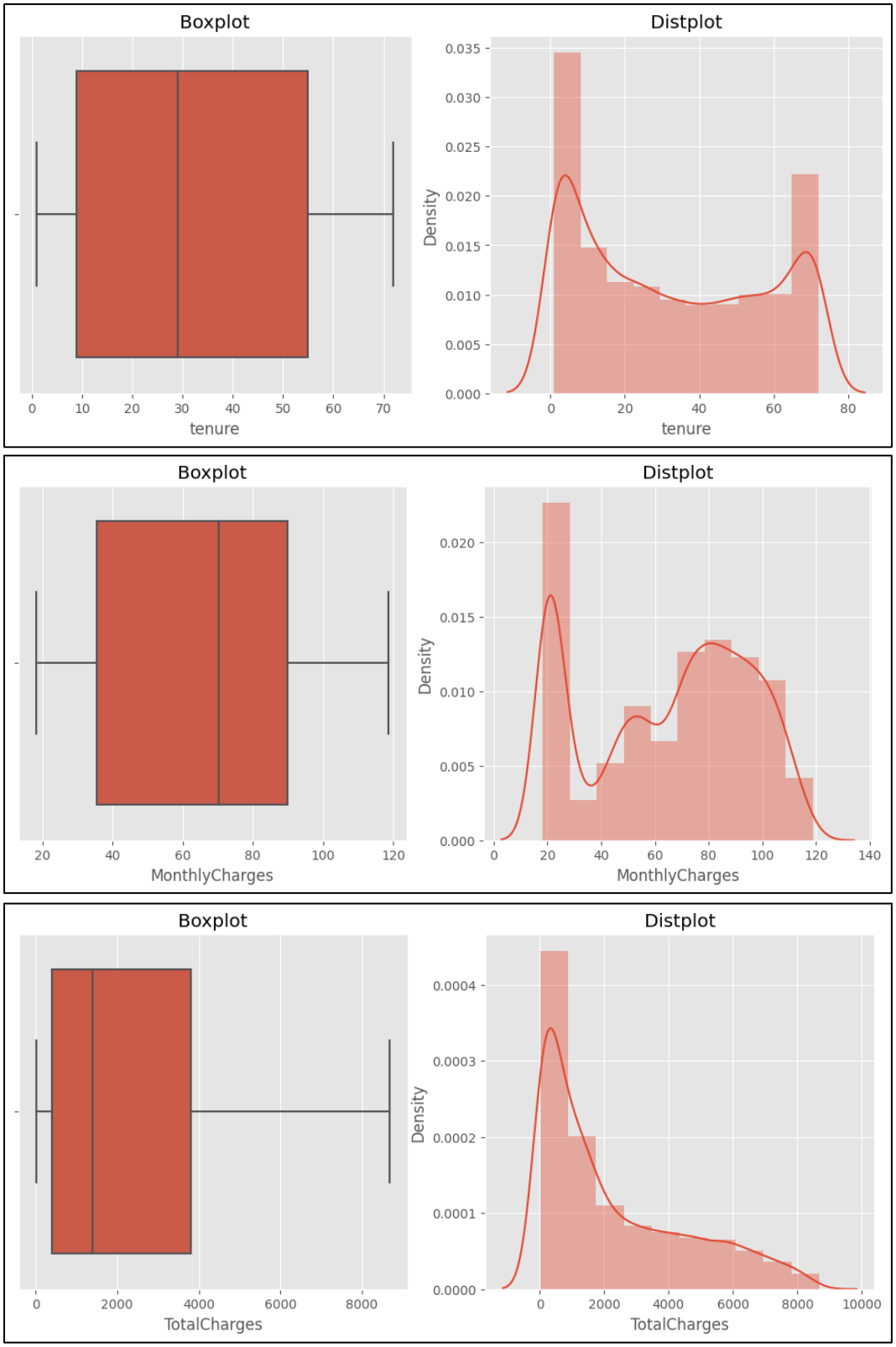
```
# Replace 'SeniorCitizen' values
customer_data['SeniorCitizen'] = customer_data['SeniorCitizen'].replace({0: 'No',
                                                                         1: 'Yes'})
```

Outlier Detection

Next, our focus will be on identifying and examining outliers within the numerical data.

	tenure	MonthlyCharges	TotalCharges
count	7032.000000	7032.000000	7032.000000
mean	32.421786	64.798208	2283.300441
std	24.545260	30.085974	2266.771362
min	1.000000	18.250000	18.800000
25%	9.000000	35.587500	401.450000
50%	29.000000	70.350000	1397.475000
75%	55.000000	89.862500	3794.737500
max	72.000000	118.750000	8684.800000

The customer dataset contains four numerical features. we'll focus our attention on plotting box plots for the ***'tenure,' 'MonthlyCharges', and 'TotalCharges'*** columns. By visually inspecting these plots, we aim to identify any potential outliers within these variables.


Boxplot
Distplot
tenure
Density
Boxplot
Distplot
MonthlyCharges
Density
Boxplot
Distplot
TotalCharges
Density

```
IQR of tenure :  46.0
Upper Limit of tenure :  124.0
Lower Limit of tenure :  -60.0
Number of Outliers Records :  0
```

```
IQR of MonthlyCharges :  54.275
Upper Limit of MonthlyCharges :  171.27499999999998
Lower Limit of MonthlyCharges :  -45.824999999999996
Number of Outliers Records :  0
```

```
IQR of TotalCharges :  3393.2875000000004
Upper Limit of TotalCharges :  8884.66875
Lower Limit of TotalCharges :  -4688.481250000001
Number of Outliers Records :  0
```

After analyzing the boxplot and calculating the upper and lower boundaries, we can confidently assert that the customer dataset doesn't contain any outliers.

Data Redundancy

In the next step, we will analyze the presence of data redundancy within the customer churn dataset.

```
The number of values for feature MultipleLines :3 -- ['No' 'No phone service' 'Yes']
The number of values for feature InternetService :3 -- ['DSL' 'Fiber optic' 'No']
The number of values for feature OnlineSecurity :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature OnlineBackup :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature DeviceProtection :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature TechSupport :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature StreamingTV :3 -- ['No' 'No internet service' 'Yes']
The number of values for feature StreamingMovies :3 -- ['No' 'No internet service' 'Yes']
```

In the output provided, certain columns contain redundant values such as 'No internet service' and 'No phone service'. These values essentially convey the same meaning as 'No'. To streamline the dataset and avoid redundancy, it's prudent to replace occurrences of 'No internet service' and 'No phone service' with simply 'No'.

```
# replace 'No internet service' and  'No phone service' into 'No'
customer_data.replace('No internet service','No',inplace=True)
customer_data.replace('No phone service','No',inplace=True)
```

```
customer_data['MultipleLines'].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

Feature Creation and Extraction

In the upcoming phase, we will use feature engineering for the telco customer churn data. Feature engineering involves the creation of existing features to enhance the predictive power of the dataset for the specific task at hand.

Initially, a new feature called ***"Total Services"*** will be generated. This feature aggregates the total number of services utilized by each customer. It encompasses various services, including internet service, phone service, online security, online backup, device protection, tech support, streaming TV, and streaming movies.

```
# Create Total Service Received
customer_data['Total_Services'] = (
    customer_data[['PhoneService', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
                   'DeviceProtection', 'TechSupport', 'StreamingTV',
                   'StreamingMovies']] != 'No').sum(axis=1)
customer_data.head()
```

Then create another new feature called ***"Avg Monthly Payments".*** Dividing the Total Charges by Tenure gives us a quotient that represents the average monthly payment made by the customer throughout their subscription. This calculation helps in understanding the average monthly expenditure of customers over their subscription period, providing insights into their payment behavior and the affordability of the service.

```
# create average monthly payments
customer_data["Avg_Monthly_Payments"] = customer_data["TotalCharges"] / customer_data["tenure"]
customer_data.head()
```

Finally, create a new feature named ***"Total Months".*** Dividing the Total Charges by the Monthly Charges gives us a quotient that represents the number of months the customer's total charges equate to base on their monthly charge rate. This calculation can help in understanding the customer's duration for which they have been subscribed to the service, based on the total charges they have accrued.

```
# Create total months
customer_data['Total_Months'] = customer_data['TotalCharges'] / customer_data['MonthlyCharges']
customer_data.head()
```

Exploratory Data Analysis - EDA

Having completed the data cleansing and Feature Engineering phases, we're now ready to delve into the analysis of the customer data set.

Data Analysis

Initially, identifying the numerical and categorical features present within the dataset.

```
Number of Categorical Variables :  17
Categorical Variables :  Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
       'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn'],
      dtype='object')
Number of Numerical Variables :  6
Numerical Variables :  Index(['tenure', 'MonthlyCharges', 'TotalCharges', 'Total_Services',
       'Avg_Monthly_Payments', 'Total_Months'],
      dtype='object')
```

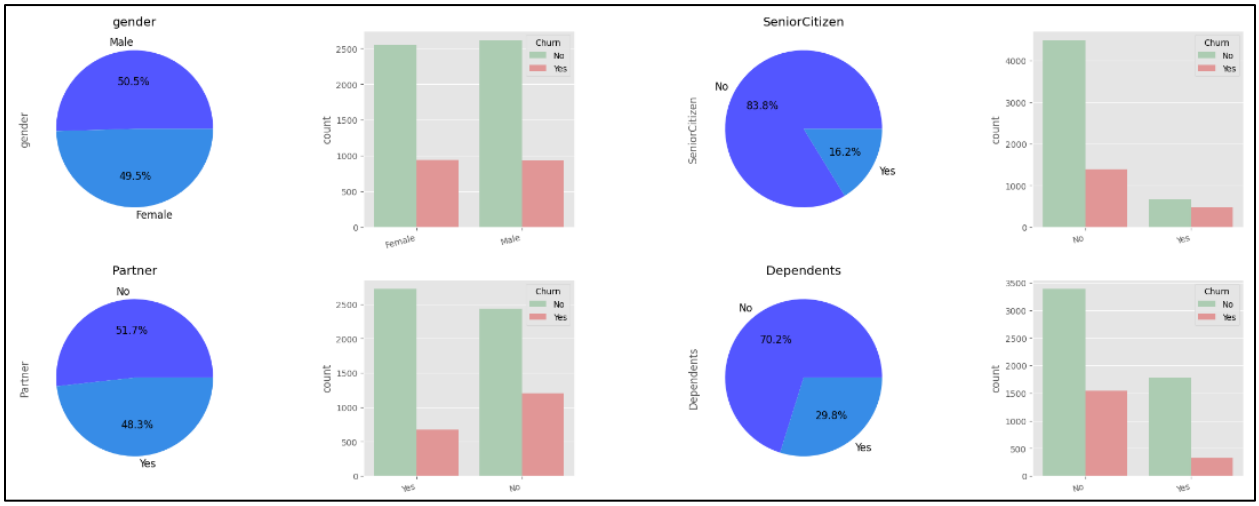
The customer dataset comprises 17 categorical features and 6 numerical features. After segregating the features into numerical and categorical groups, the next step is to analyze each categorical feature individually. This involves examining each categorical feature's value counts, value counts in percentage, relationship with the target feature, and relationship with the target feature in percentage.

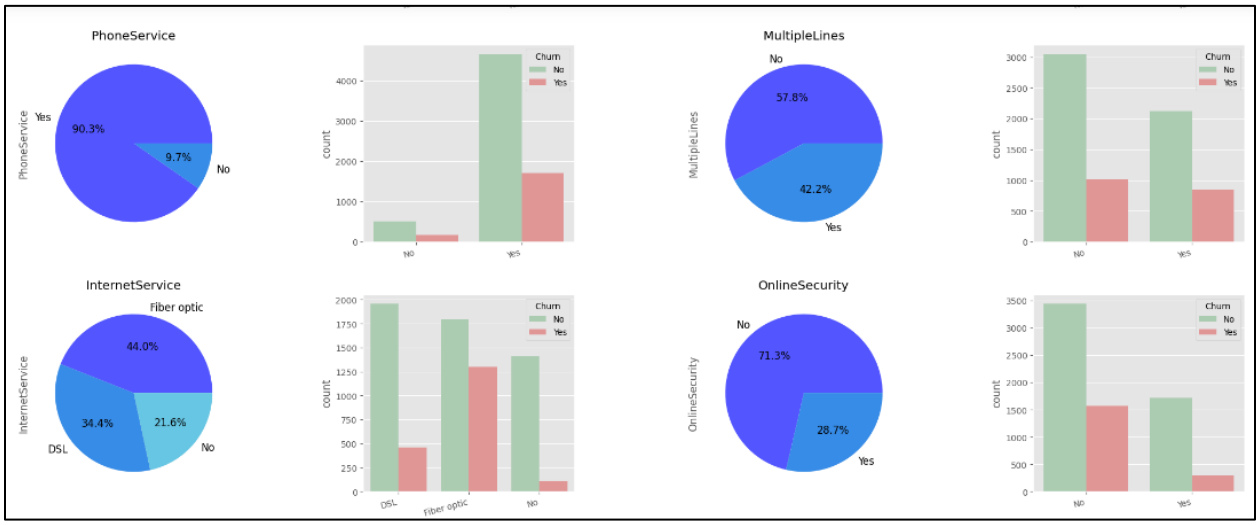
```
****** Analysis of gender *****
Value Counts
Male      3549
Female    3483
Name: gender, dtype: int64

Value Counts %
Male      50.469283
Female    49.530717
Name: gender, dtype: float64

Relationship with Target Variable
Churn  gender
No     Male      2619
       Female    2544
Yes    Female     939
       Male       930
Name: gender, dtype: int64

Relationship with Target Variable in %
Churn  gender
No     Male      37.244027
       Female    36.177474
Yes    Female    13.353242
       Male      13.225256
Name: gender, dtype: float64
```

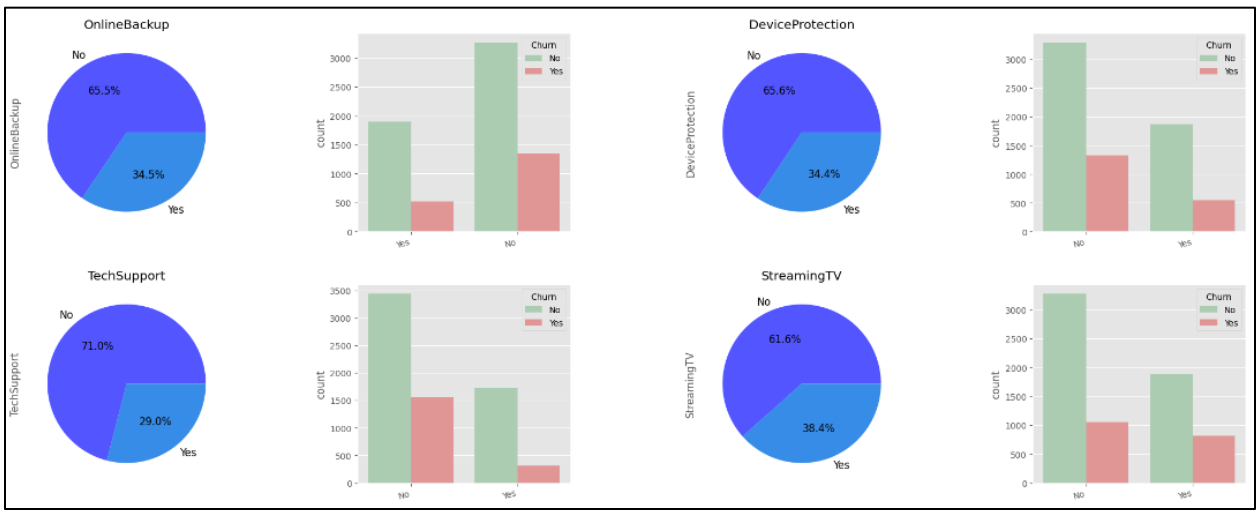
```
****** Analysis of Dependents *****
Value Counts
No     4933
Yes    2099
Name: Dependents, dtype: int64

Value Counts %
No     70.150739
Yes    29.849261
Name: Dependents, dtype: float64

Relationship with Target Variable
Churn  Dependents
No     No            3390
       Yes           1773
Yes    No            1543
       Yes            326
Name: Dependents, dtype: int64

Relationship with Target Variable in %
Churn  Dependents
No     No            48.208191
       Yes           25.213311
Yes    No            21.942548
       Yes            4.635950
Name: Dependents, dtype: float64
```

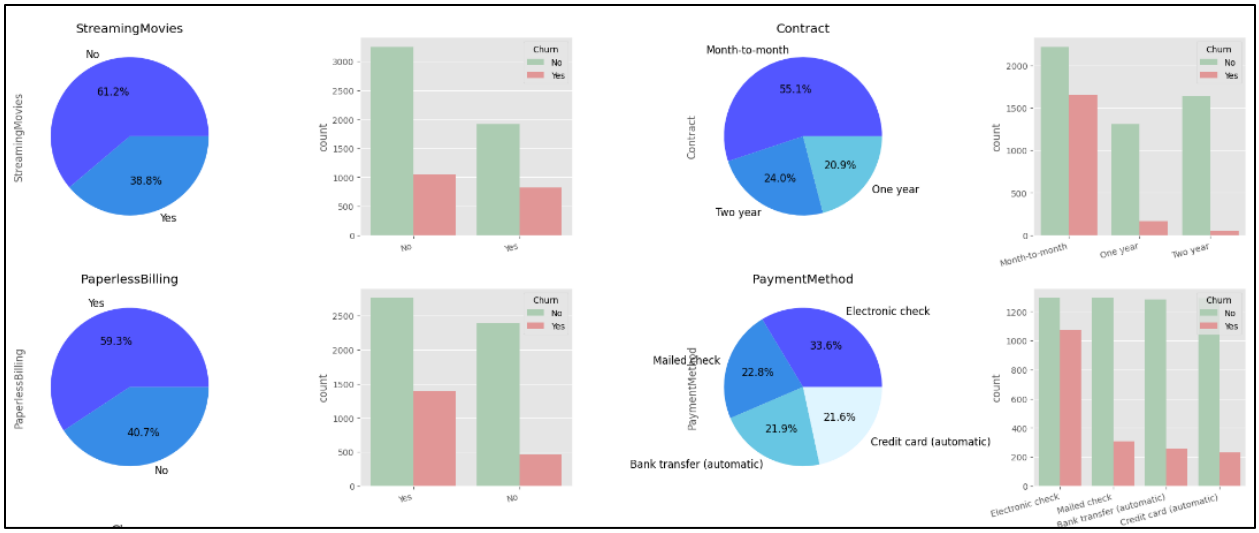
The analysis output provided above offers insights into the ***'gender'*** variable. Within the dataset, there are 3549 (50.47%) males and 3483 (49.53%) females. Regarding the relationship with the target variable 'churn,' 2619 (37.24%) males and 2544 (36.18%) females have churned with a status of 'No,' while 930 (13.35%) males and 939 (13.22%) females have churned with a status of 'Yes.' Similarly, this analysis can be extended to examine the relationship between other categorical variables and the target variable.

Distribution of Categorical Features





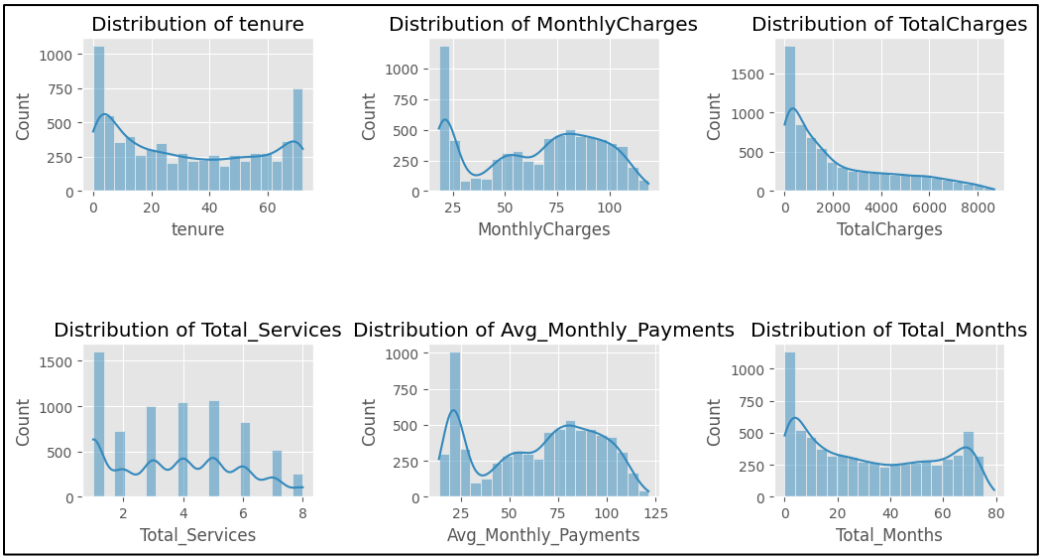




Based on the above outputs, we can conclude the following:

- The gender feature does not affect the client's decision.
- The older people are more likely to refuse services.
- Clients in relationships, as well as clients with children, are less likely to refuse services. Perhaps the company will present favorable family tariffs.
- Customers with fiber optic more often refuse services. Customers who do not use the Internet very rarely refuse.
- Clients who use protection systems, as well as those who use cloud storage, are more likely to refuse. Competitors also have favorable package offers with additional services.
- Customers who do not contact technical support are more likely to refuse.
- Logical, clients with a short-term contract leave more often.
- Customers who receive and pay bills conservatively are less likely to change service providers.

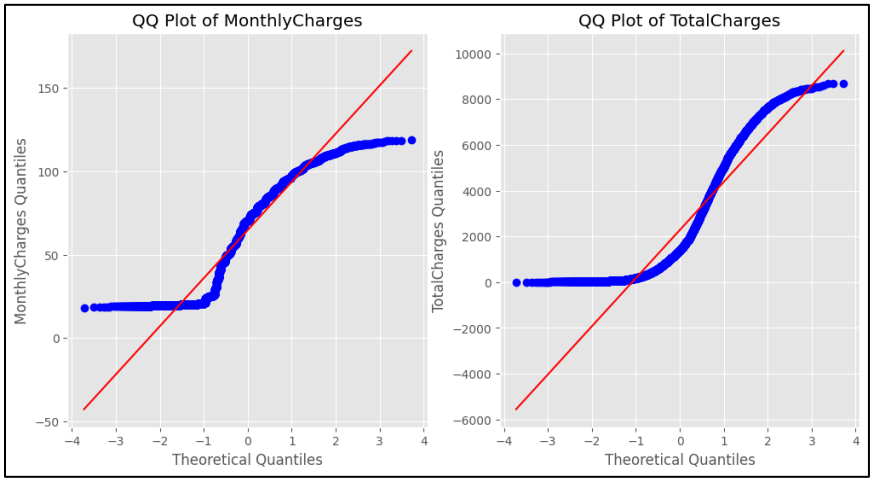
Distribution of Numerical Features



Upon examining the Histogram Plots, a notable trend emerges: Tenure, total services, and total months all have right-skewed distributions, suggesting that a larger portion of customers have lower values for these features. Monthly charges and total charges appear to have normal distributions, indicating that most customers have values centered around an average value.

Q-Q plots

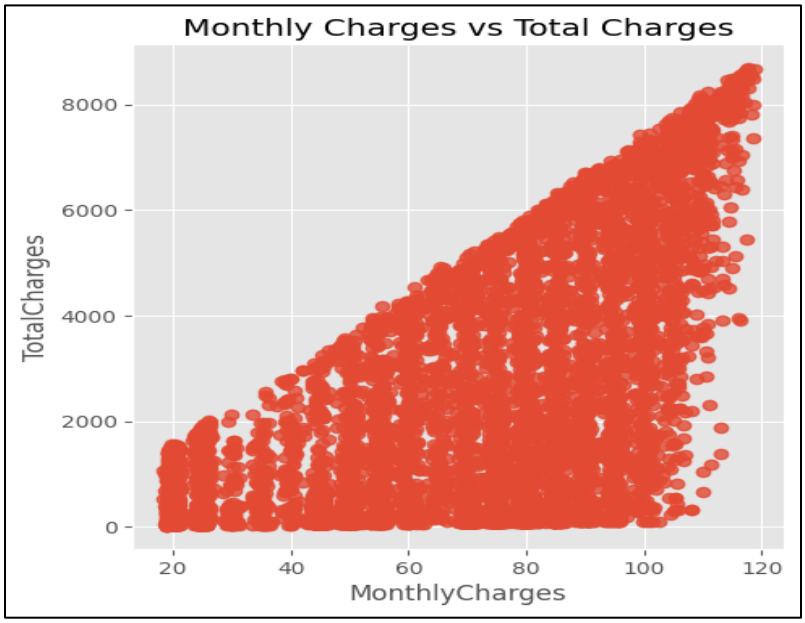
Let's confirm the distribution of Monthly charges and total charges features using Q-Q plots and the Shapiro-Wilk Test.




```
Shapiro-Wilk Test for MonthlyCharges
Shapiro-Wilk Test Statistic: 0.9209
p-value: 0.00
The data in MonthlyCharges might not be normally distributed.

Shapiro-Wilk Test for TotalCharges
Shapiro-Wilk Test Statistic: 0.8602
p-value: 0.00
The data in TotalCharges might not be normally distributed.
```

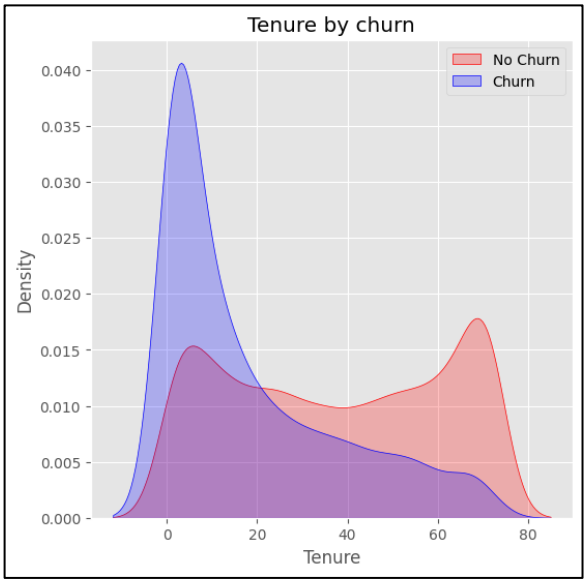
Based on the Q-Q plots and the Shapiro-Wilk Test, we conclude Monthly charges and total charges features might not be normally distributed.

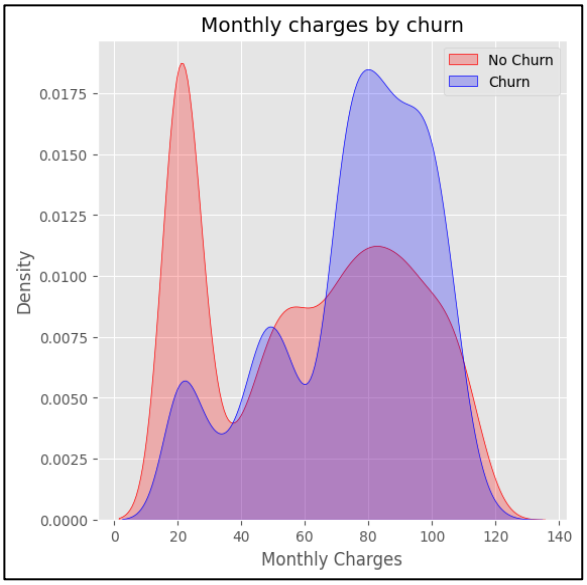
Monthly Charges vs Total Charges

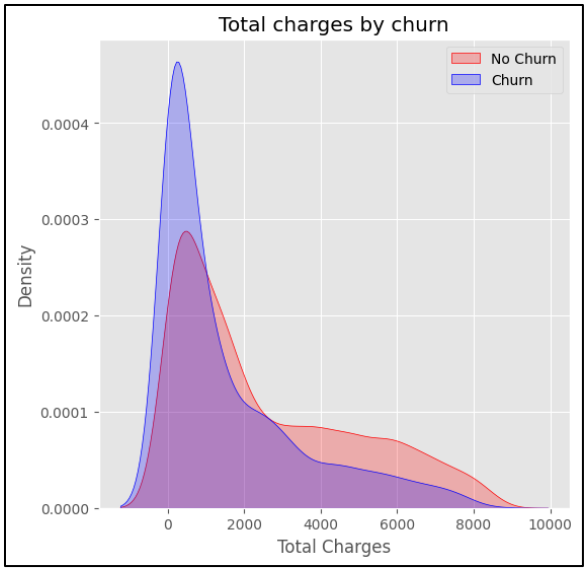


The conclusion drawn from the lmplot above suggests a positive correlation between Monthly Charges and Total Charges. In simpler terms, as the Monthly Charges increase, so do the Total Charges incurred.

Churn by Tenure, Monthly Charges, and Total Charges







- The density plot depicting the relationship between ***Tenure and Churn*** reveals insightful patterns. It suggests that churn rates are elevated among customers with shorter tenure periods, indicating a propensity for new customers to churn. Conversely, as tenure increases, the likelihood of churn diminishes, with a higher concentration of customers exhibiting no churn.
- The density plot analysis between ***Monthly Charges and Churn*** reveals an insightful trend. It suggests that churn rates tend to increase notably as Monthly Charges rise, indicating a correlation between higher charges and customer attrition. Conversely, the density of non-churn instances is higher when Monthly Charges are lower, suggesting a potential inverse relationship between charges and customer retention.
- The conclusion drawn from the density plot depicting the relationship between ***Total Charges and Churn*** is rather unexpected. It reveals that instances of both churn and no-churn are notably high when Total Charges are low.

Handling with Categorical Features

The customer dataset consists of 16 categorical features, which include the target feature, and all of these categorical features are nominal. The next step to prepare the data for analysis involves converting these categorical features into a numeric format.

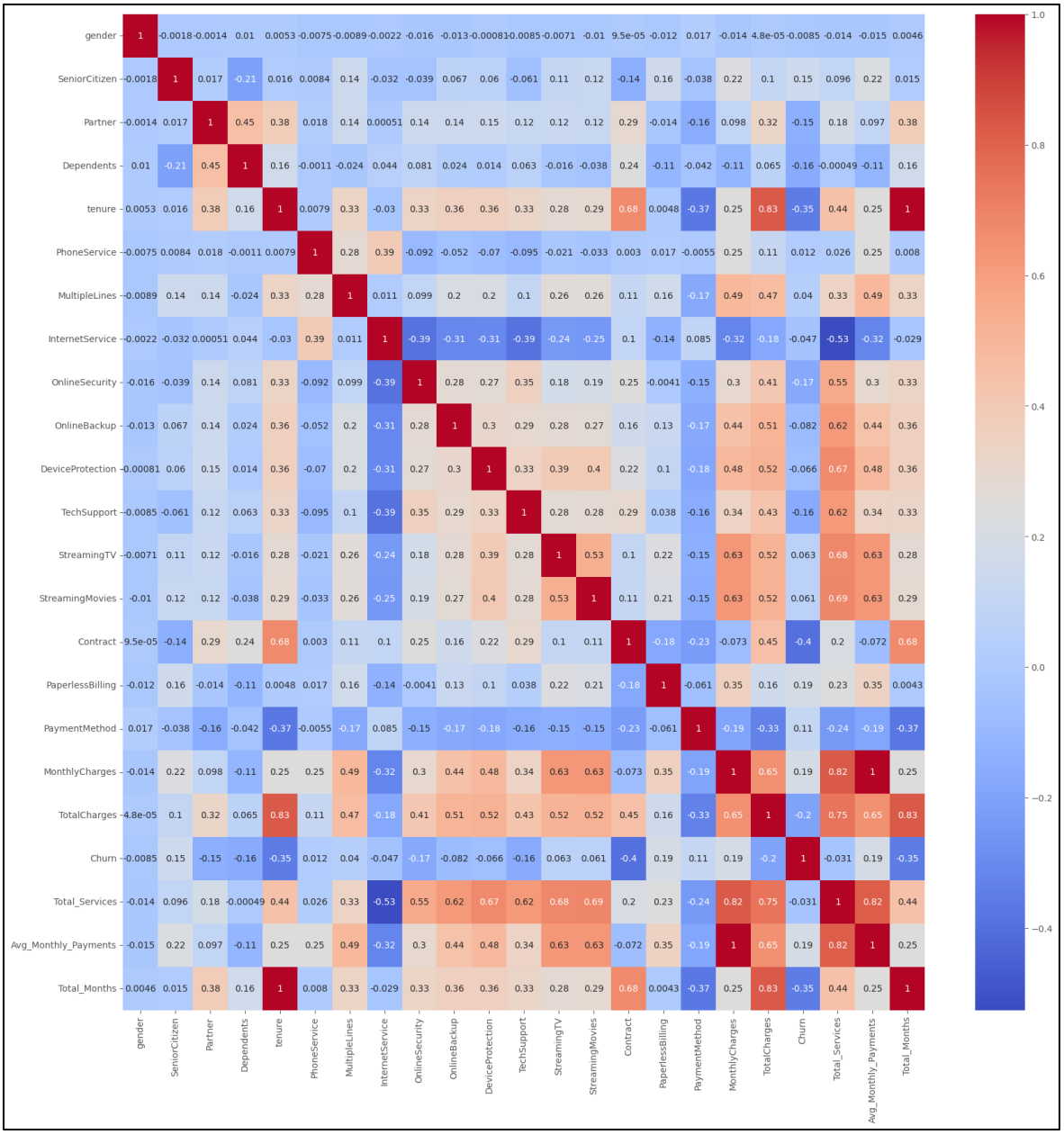
Label encoding.

We will perform label encoding on the categorical features.

```
# label encoding
le = LabelEncoder()

for col in categorical:
    customer_encoded[col] = le.fit_transform(customer_encoded[col])
```

Correlation Heatmap

We'll examine the correlation among the features to understand how they relate to each other. Correlation analysis helps identify patterns and dependencies within the dataset.



Based on the above heatmap we can conclude the following:

- The ***total month*** feature is 100% correlated with the ***tenure*** feature and the ***Average Monthly Payments*** feature is 100% correlated with ***Monthly Charges*** features. So, we can remove the Total Month and Average Monthly Payments features.
- Senior Citizens, Phone Service, Multiple Lines, Streaming TV, Streaming Movies, Paperless Billing, Payment Methods, Monthly Charges, and Average Monthly Payments have a positive correlation with Churn.

- Gender, Dependents, Partner, Internet Services, Online Security, Online Backup, Device Protection, Tech Support, Contract, Total Services, tenure, Total Months, and Total charges have a negative correlation with Churn.

Let extract features with more than 0.8 correlation.

```
# call the function with 0.9 threshold
cor_feature(customer_encoded, 0.9)
```

```
{'Avg_Monthly_Payments', 'Total_Months'}
```

In the above code, the ***Average Monthly Payments and Total Months*** features are highly correlated with other features. Let's remove those highly correlated features.

```
# drop highly correlated features
customer_encoded = customer_encoded.drop(['Avg_Monthly_Payments', 'Total_Months'],
                                         axis=1)
customer_encoded.shape
```

```
(7043, 21)
```

Scaling the data

In the process of scaling the data, we will apply scales to ***Tenure, Monthly Charges, and Total Charges*** features. Scaling ensures that all features are on a similar scale, preventing certain features from dominating others due to their larger magnitude.

```
scale_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
# Scaling the numerical data
sc = StandardScaler()

customer_encoded[scale_cols] = sc.fit_transform(customer_encoded[scale_cols])
```

Split the data into Train and Test

For model training, we need training and testing datasets so let's split the training dataset into training and testing datasets. Before splitting the data, we should identify and separate the Independent (X) and Dependents (y) variables.

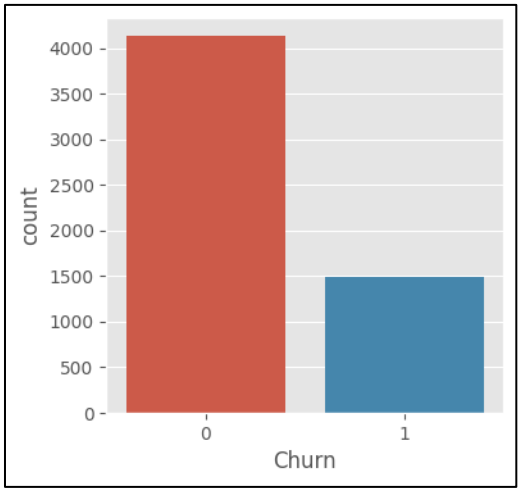
```
# Independent features
X = customer_encoded.drop(['Churn'], axis=1)

# Dependent feature
y = customer_encoded['Churn']
```

Once identified the X and y variables then split the dataset into training and testing. Here we are going to split 80% of the data into training and the rest are testing.

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.2,
                                                    random_state = 42)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(5634, 20) (5634,)
(1409, 20) (1409,)
```

Handling Imbalanced Data



Based on the above count plot, we can say there is imbalanced data. We can handle this imbalanced data using over-sampling or under-sampling techniques. Let's get the count of Churn classes.

```
The number of classes Counter({0: 4130, 1: 1495})
```

The minority class (Churn Yes) has 1495 counts, and the majority class (Churn No) has 4130 counts. We have two options to handle this imbalanced data: over-sampling (SMOTE) and under-sampling (Near Miss). Over-sampling aims to balance class distribution by randomly increasing minority class examples by replicating them, and under-sampling aims to balance class distribution by randomly eliminating majority class examples. In this case, we are going to use the ***SMOTE-ENN*** technique. SMOTE-ENN (combining SMOTE with the Edited Nearest Neighbors (ENN) technique).

```
smn = SMOTEENN()
X_train_ns, y_train_ns = smn.fit_resample(X_train, y_train)

print("The number of classes before fit {}".format(Counter(y_train)))
print("The number of classes after fit {}".format(Counter(y_train_ns)))
```

```
The number of classes before fit Counter({0: 4130, 1: 1495})
The number of classes after fit Counter({1: 2922, 0: 2272})
```

After SMOTE-ENN, the data is now balanced. So now we can go to the model selection and training phases.

Model Training and Prediction

Our aim for this project is to predict customer churn ('Yes' or 'No'), and this is a binary classification problem. There are several algorithms available to solve classification problems in machine learning.

- ☆ **Basic classification algorithms:** Logistic regression, Decision tree classifier, k-nearest neighbors, Support vector machine, Naïve Bayes.
- ☆ **Ensemble classification algorithms:** Random forest classifier, AdaBoost, Gradient boosting, XG Boost, and more.

For a time, we are going to select and train the following algorithms: Logistic regression, k-nearest neighbors, Decision Tree, Random Forest, and XG Boost.

We possess two distinct training datasets: one before addressing imbalanced data and the other after implementing techniques to handle imbalance. To evaluate their efficacy, we'll train the model using both datasets and subsequently compare their performance metrics.

Logistic Regression

Logistic Regression is a simple yet powerful technique for binary classification tasks such as churn prediction. It assigns probabilities to each class, making it easier to comprehend and understand. It also works well when the features have a linear relationship with the response's log-odds.

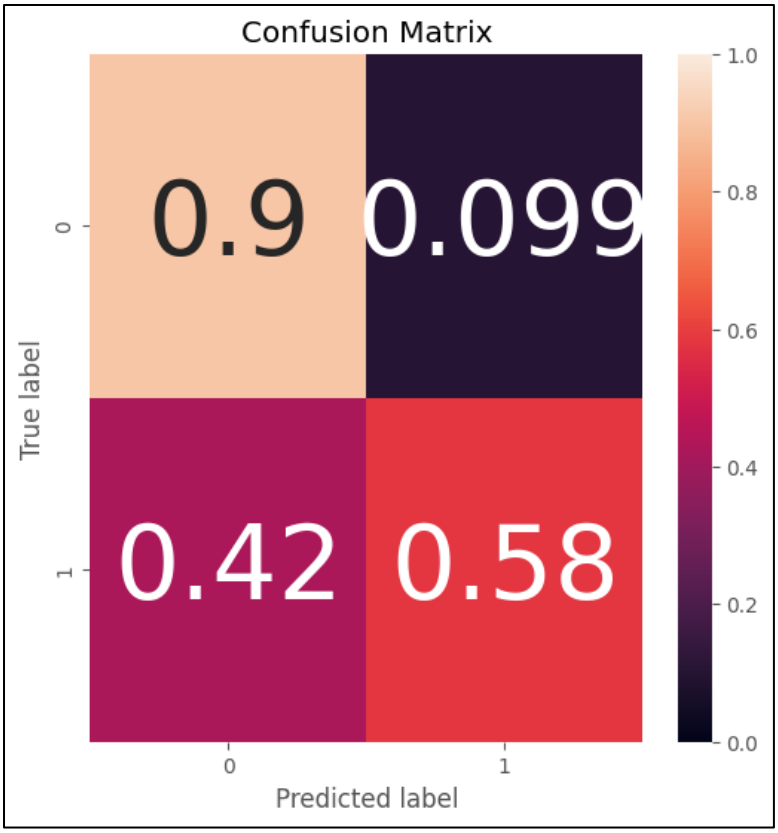
Initially, train the Logistic regression model with 2000 maximum iteration parameters and make predictions before handling imbalanced data.

```
# before handling imbalance
model_log = LogisticRegression(max_iter=2000)
model_log.fit(X_train, y_train)
```

```
# Predict the response for train dataset
tr_pred_log = model_log.predict(X_train_ns)
```

```
# prediction of test data
y_pred_log = model_log.predict(X_test)
```

After the model training and predictions, let’s do the model evaluation for the logistic model. Here we perform the following evaluation metrics: confusion matrix, training and testing accuracy, precision, recall, and f1 score.




```
Training Accuracy: 0.7749951085893172
Accuracy: 0.8161816891412349
Recall: 0.5817694369973191
Precision: 0.678125
F1 Score: 0.6262626262626263
```

Then train another Logistic regression model with 2000 maximum iteration parameters and make predictions after handling imbalanced data.

```
bal_log = LogisticRegression(max_iter=2000)
bal_log.fit(X_train_ns, y_train_ns)
```

Then get the accuracy of both logistic regression models.

```
***** Logistic model ******
Accuracy for without handle imbalance data:  0.8161816891412349
Accuracy for with handle imbalanced data:  0.7118523775727467
```

Based on the above output, we can observe before handling an imbalanced data model got better accuracy than after handling an imbalanced data model.

K-Nearest Neighbors

KNN is a basic algorithm that detects local patterns in data. It is nonparametric and adapts well to a variety of datasets. However, it may suffer from the curse of dimensionality and may not perform as well in high-dimensional environments.

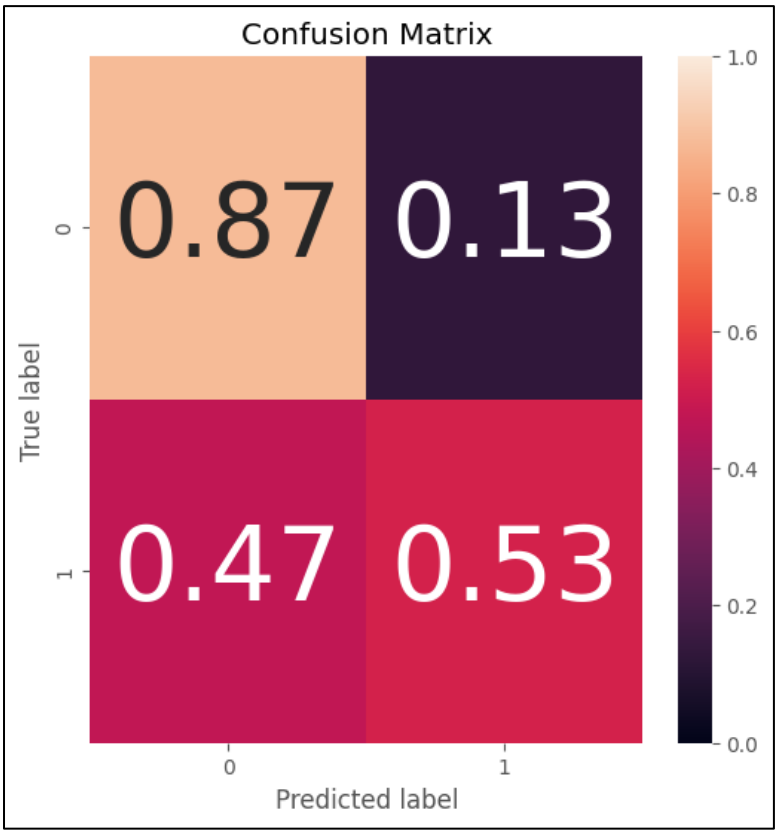
First train the KNN model with 7 number of neighbors, 'Minkowski' as a metric, and p is 2, and make predictions before handling imbalanced data.

```
model_knn = KNeighborsClassifier(n_neighbors=7,
                                 metric='minkowski',
                                 p=2)
model_knn.fit(X_train, y_train)
```

```
# Predict the response for training dataset
tr_pred_knn = model_knn.predict(X_train_ns)
```

```
# prediction of test data
y_pred_knn = model_knn.predict(X_test)
```

After the model training and predictions, let's do the model evaluation for the KNN model. Here we perform the following evaluation metrics: confusion matrix, training and testing accuracy, precision, recall, and f1 score.



```
Training Accuracy: 0.8583447466249267
Accuracy: 0.7799858055358411
Recall: 0.5254691689008043
Precision: 0.5254691689008043
F1 Score: 0.5584045584045585
```

Then train another KNN model with the same parameters and make predictions after handling imbalanced data.

```
bal_knn = KNeighborsClassifier(n_neighbors=7)
bal_knn.fit(X_train_ns, y_train_ns)
```

Then get the accuracy of both KNN models.

```
***** K-NN model ******
Accuracy for without handle imbalance data:  0.7799858055358411
Accuracy for with handle imbalanced data:  0.6792051100070973
```

Based on the above output, we can observe before handling an imbalanced data model got better accuracy than after handling an imbalanced data model.

Random Forest

Random Forests address the issue of overfitting in individual decision trees by aggregating predictions from several trees. They handle non-linearity well, emphasize feature relevance, and give overall robust performance. Random Forests are appropriate for datasets containing a large number of features.

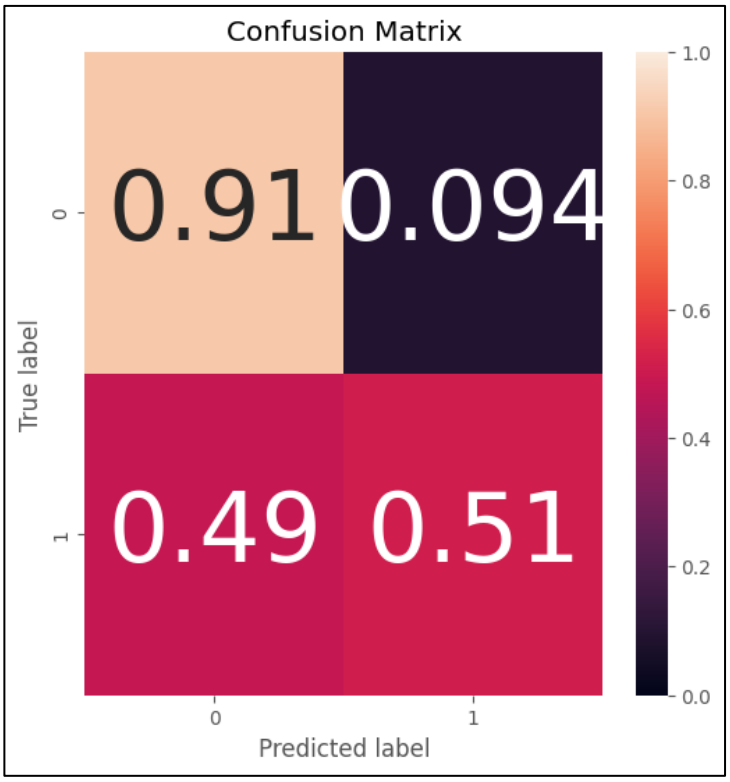
Initially, train the Random Forest model with the following parameters: number of estimators is 50, maximum depth is 10, number of jobs is -1, and random state is 42 and make predictions before handling imbalanced data.

```
model_rfc = RandomForestClassifier(n_estimators=50,
                                   max_depth=10,
                                   n_jobs=-1,
                                   random_state=42)
model_rfc.fit(X_train, y_train)
```

```
# Predict the response for training dataset
tr_pred_rfc = model_rfc.predict(X_train_ns)
```

```
# prediction of test data
y_pred_rfc = model_rfc.predict(X_test)
```

After the model training and predictions, let's do the model evaluation for the Random Forest model. Here we perform the following evaluation metrics: confusion matrix, training and testing accuracy, precision, recall, and f1 score.



```
Training Accuracy: 0.8039522598317355
Accuracy: 0.8026969481902059
Recall: 0.8026969481902059
Precision: 0.6643598615916955
F1 Score: 0.5800604229607251
```

Then train another Random Forest model with the same parameters and make predictions after handling imbalanced data.

```
bal_rfc = RandomForestClassifier(n_estimators=50,
                                 max_depth=10,
                                 n_jobs=-1,
                                 random_state=42)
bal_rfc.fit(X_train_ns, y_train_ns)
```

Then get the accuracy of both Random Forest models.

```
***** Random Forest model ******
Accuracy for without handle imbalance data:  0.8026969481902059
Accuracy for with handle imbalanced data:  0.7175301632363378
```

Based on the above output, we can observe before handling an imbalanced data model got better accuracy than after handling an imbalanced data model.

XG Boost

XGBoost excels at capturing complicated associations while maintaining high predicted accuracy. They sequentially create trees, providing greater weight to misclassified occurrences, resulting in enhanced performance. These algorithms frequently outperform other methods in predicting accuracy.

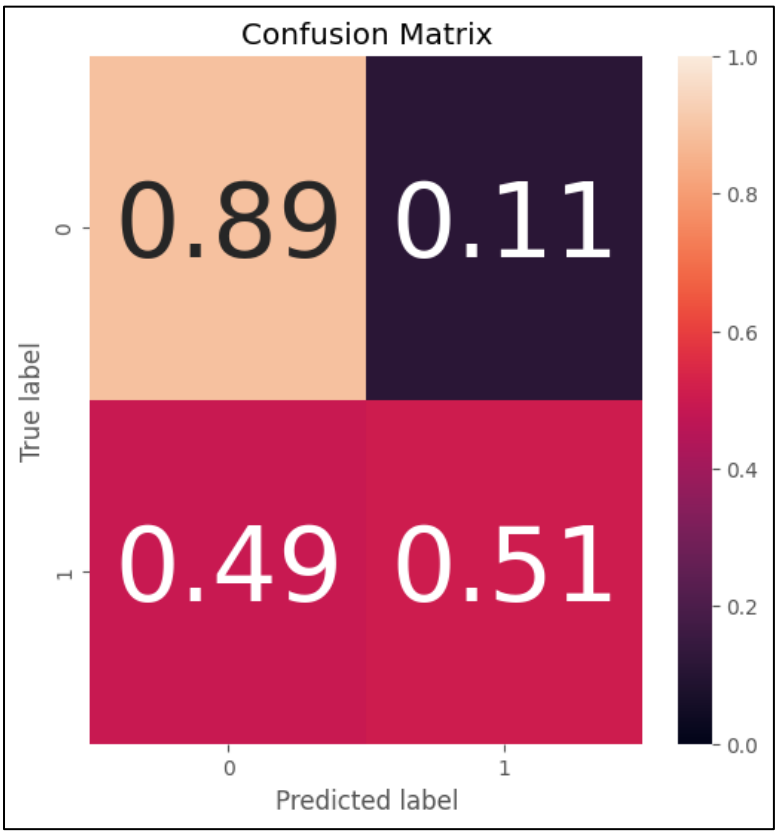
Initially, train the XG Boost model with binary logistic as objective, random state is 42, and enable the categorical, and make predictions before handling imbalanced data.

```
model_xgb = XGBClassifier(objective="binary:logistic",
                          random_state=42,
                          enable_categorical=True)
model_xgb.fit(X_train, y_train)
```

```
# Predict the response for training dataset
tr_pred_xgb = model_xgb.predict(X_train_ns)
```

```
# prediction of test data
y_pred_xgb = model_xgb.predict(X_test)
```

After the model training and predictions, let's do the model evaluation for the logistic model. Here we perform the following evaluation metrics: confusion matrix, training and testing accuracy, precision, recall, and f1 score.



```
Training Accuracy: 0.8644100958716494
Accuracy: 0.7877927608232789
Recall: 0.5067024128686327
Precision: 0.6217105263157895
F1 Score: 0.5583456425406205
```

Then train another XG Boost model with the same parameters and make predictions after handling imbalanced data.

```
bal_xgb = XGBClassifier(objective="binary:logistic",
                        random_state=42,
                        enable_categorical=True)
bal_xgb.fit(X_train_ns, y_train_ns)
```

Then get the accuracy of both XG Boost models.

```
***** XG Boost model ******
Accuracy for without handle imbalance data:  0.7877927608232789
Accuracy for with handle imbalanced data:  0.7239176721078779
```

Based on the above output, we can observe before handling an imbalanced data model got better accuracy than after handling an imbalanced data model.

Note:

Following model training and assessing performance both before and after addressing imbalanced data, it's evident that the model achieved higher accuracy before handling the imbalance. Therefore, we'll proceed with the model trained on the original dataset, as it exhibited superior performance.

Hyperparameter Tuning

After finishing model training and getting performance evaluations, we will focus on hyperparameter tuning to improve model performance. To enhance the performance of our models, we fine-tune their parameters. We'll use ***RandomizedSearchCV***, a technique that quickly examines a variety of hyperparameter combinations to get the best configuration. By carefully tweaking these parameters, we hope to improve the model's predictive accuracy and generalization capabilities.

During the hyperparameter tuning phase, we follow a systematic approach across all models. Firstly, we compile a list of hyperparameters for each model, setting the stage for fine-tuning. Next, we utilize ***RandomizedSearchCV*** to train and fit the model, optimizing hyperparameters to enhance performance. Subsequently, we extract the best parameters and refit the model accordingly. Following model fitting, we evaluate performance using metrics such as confusion matrix and accuracy. Finally, we conduct a comprehensive comparison between the model's pre-tuning and post-tuning states, analyzing performance metrics like accuracy, recall, precision, and f1 score to assess improvement.

Logistic Regression

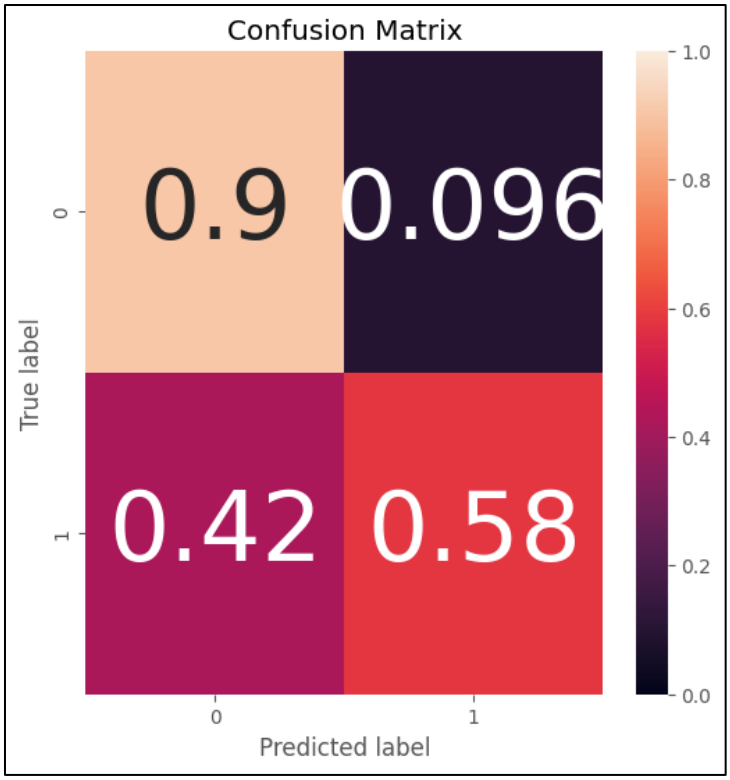
```
# list the hyperparameters for tuning
log_parameters = [{'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']},
                  {'penalty':['none',  'l2']},
                  {'C':[ 0.01, 0.1, 1, 10, 100]},
                  {'max_iter':[100, 2000, 3000, 4000, 5000, 7000, 10000, 20000, 50000]}]

# Tuned logistic model
tuned_log = RandomizedSearchCV(model_log,
                               log_parameters,
                               scoring = 'f1',
                               cv = 5,
                               verbose=3)

tuned_log.fit(X_train, y_train)
```

```
# getting the model with the best parameters
tuned_log.best_estimator_
```

```
LogisticRegression(max_iter=2000, penalty='none')
```

```
# Final logistic model
final_log = LogisticRegression(max_iter=2000,
                               penalty='none')
final_log.fit(X_train, y_train)
```

```
# use testing data to predict
y_pred_log2 = final_log.predict(X_test)
```




```
***** Logistic Model ******

Before Hyperparameter Tuning
Accuracy: 0.8161816891412349
Precision: 0.678125
Recall: 0.5817694369973191
F1 Score: 0.6262626262626263

After Hyperparameter Tuning
Accuracy: 0.8183108587650816
Precision: 0.6857142857142857
Recall: 0.579088471849866
F1 Score: 0.627906976744186
```

There is a slight improvement in the Logistic model after the hyperparameter tuning.

K-Nearest Neighbors

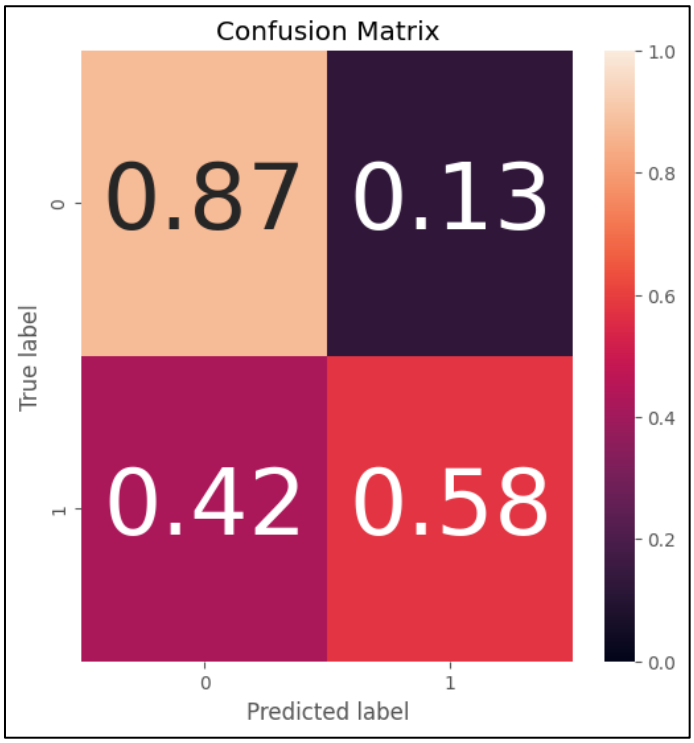
```
# list hyperparameter of knn
knn_params = {'n_neighbors' : range(2, 12),
              'weights' : ['uniform', 'distance'],
              'metric' : ['minkowski', 'euclidean', 'manhattan']}

# new logistic model
tuned_knn = RandomizedSearchCV(model_knn,
                               knn_params,
                               scoring = 'f1',
                               cv = 5,
                               verbose=3)

tuned_knn.fit(X_train, y_train)
```

```
# getting the model with the best parameters
tuned_knn.best_estimator_
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=11)
```

```
# Final KNN model
final_knn = KNeighborsClassifier(metric='euclidean',
                                 n_neighbors=11)
final_knn.fit(X_train, y_train)
```

```
# use testing data to predict
y_pred_knn2 = final_knn.predict(X_test)
```



```
***** K-NN Model ******

Before Hyperparameter Tuning
Accuracy: 0.7799858055358411
Precision: 0.5957446808510638
Recall: 0.5254691689008043
F1 Score: 0.5584045584045585

After Hyperparameter Tuning
Accuracy: 0.794889992902768
Precision: 0.6213872832369942
Recall: 0.5764075067024129
F1 Score: 0.5980528511821974
```

There is a little improvement in the KNN model after the hyperparameter tuning.

Random Forest

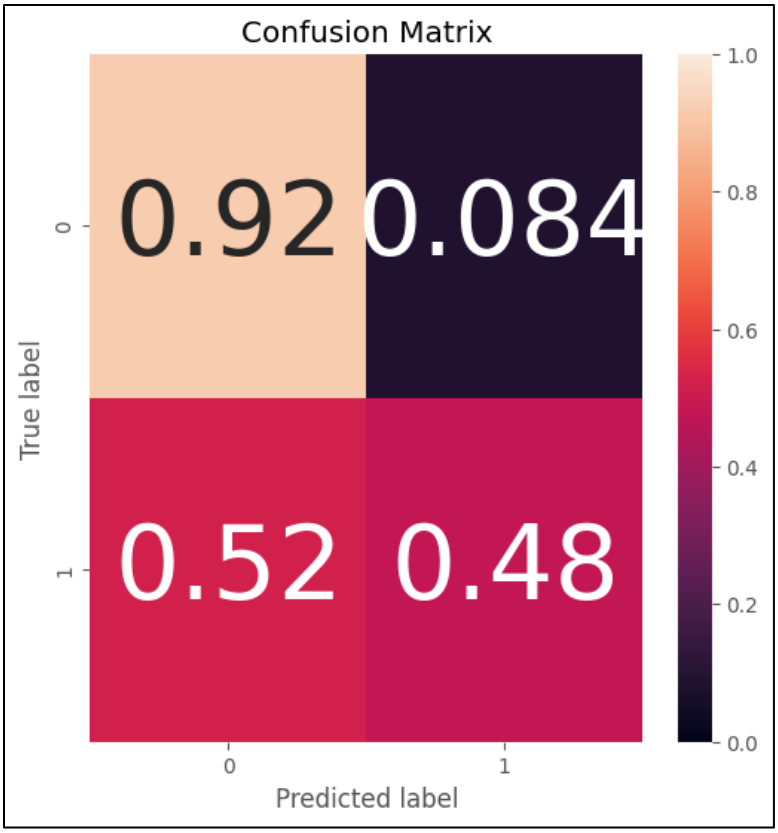
```
# set options for the hyperparameters
rf_param_grid = {
    'n_estimators': [25, 50, 60, 80, 100, 120, 150],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [5, 7, 10, 12, 15, 20, 25, None],
    'min_samples_leaf': [1, 2, 3, 4, 5],
    'min_samples_split': [2, 3, 4, 5, 6, 7]}

# new logistic model
tuned_rfc = RandomizedSearchCV(model_rfc,
                               rf_param_grid,
                               cv=5,
                               scoring='f1',
                               verbose=3)

tuned_rfc.fit(X_train, y_train)
```

```
# getting the model with the best parameters
tuned_rfc.best_estimator_
```

```
RandomForestClassifier(max_depth=20, min_samples_leaf=4, min_samples_split=7,
                       n_estimators=60, random_state=42)
```

```
# Final Random Forest model
final_rfc = RandomForestClassifier(max_depth=20, min_samples_leaf=4,
                                   min_samples_split=7,
                                   n_estimators=60, random_state=42)
final_rfc.fit(X_train, y_train)
```

```
# use testing data to predict
y_pred_rfc2 = final_rfc.predict(X_test)
```




```
***** Random Forest Model ******

Before Hyperparameter Tuning
Accuracy: 0.8026969481902059
Precision: 0.6643598615916955
Recall: 0.514745308310992
F1 Score: 0.5800604229607251

After Hyperparameter Tuning
Accuracy: 0.8005677785663591
Precision: 0.6729323308270677
Recall: 0.47989276139410186
F1 Score: 0.5602503912363067
```

There are no improvements in the Random Forest model after the hyperparameter tuning.

XG Boost

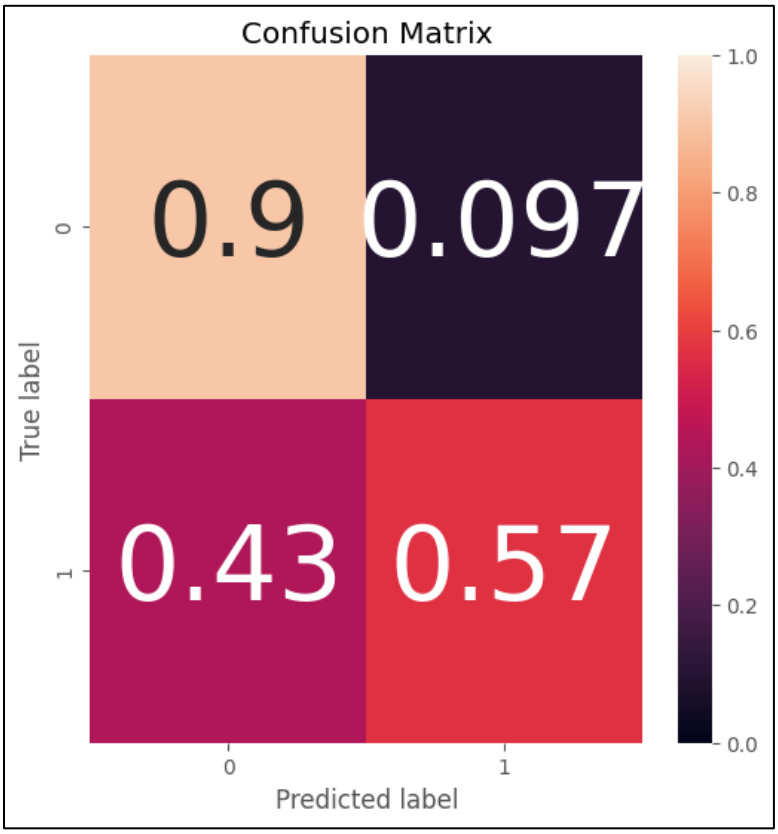
```
xgb_param_grid = {
    "colsample_bytree": [0.3, 0.4, 0.5, 0.7, 0.8],
    "gamma": [0.0, 0.1, 0.2, 0.3, 0.4, 0.5],
    'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2, 0.3],
    "max_depth": range(1,12),
    "n_estimators": [100, 200, 300, 400, 500],
    "min_child_weight": range(1, 7),
    'subsample': [0.2, 0.3, 0.4, 0.5, 0.6, 0.7]
}

tuned_xgb = RandomizedSearchCV(model_xgb,
                               xgb_param_grid,
                               cv=5,
                               scoring='f1',
                               verbose=3)

tuned_xgb.fit(X_train, y_train)
```

```
# getting the model with the best parameters
tuned_xgb.best_estimator_
```

```
XGBClassifier
        colsample_bylevel=None, colsample_bynode=None,
        colsample_bytree=0.3, device=None, early_stopping_rounds=None,
        enable_categorical=True, eval_metric=None, feature_types=None,
        gamma=0.5, grow_policy=None, importance_type=None,
        interaction_constraints=None, learning_rate=0.3, max_bin=None,
        max_cat_threshold=None, max_cat_to_onehot=None,
        max_delta_step=None, max_depth=1, max_leaves=None,
        min_child_weight=1, missing=nan, monotone_constraints=None,
        multi_strategy=None, n_estimators=200, n_jobs=None,
        num_parallel_tree=None, random_state=42, ...)
```

```
# Final Random Forest model
final_xgb = XGBClassifier(base_score=None, booster=None, callbacks=None,
        colsample_bylevel=None, colsample_bynode=None,
        colsample_bytree=0.3, device=None, early_stopping_rounds=None,
        enable_categorical=True, eval_metric=None, feature_types=None,
        gamma=0.5, grow_policy=None, importance_type=None,
        interaction_constraints=None, learning_rate=0.3, max_bin=None,
        max_cat_threshold=None, max_cat_to_onehot=None,
        max_delta_step=None, max_depth=1, max_leaves=None,
        min_child_weight=1, monotone_constraints=None,
        multi_strategy=None, n_estimators=200, n_jobs=None,
        num_parallel_tree=None, random_state=42)
final_xgb.fit(X_train, y_train)
```

```
# use testing data to predict
y_pred_xgb2 = final_xgb.predict(X_test)
```

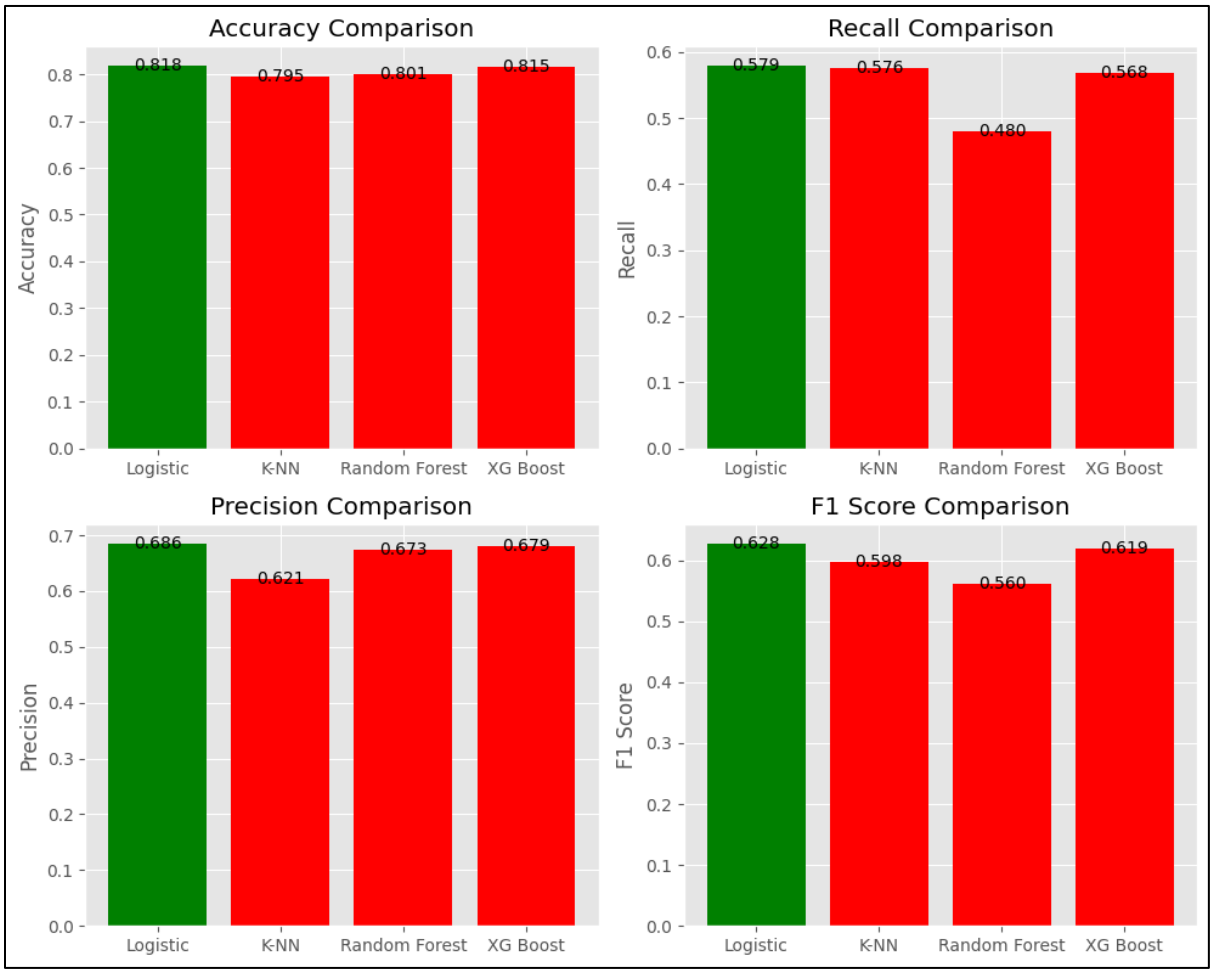



```
***** XG Boost Model ******

Before Hyperparameter Tuning
Accuracy: 0.7877927608232789
Precision: 0.6217105263157895
Recall: 0.5067024128686327
F1 Score: 0.5583456425406205

After Hyperparameter Tuning
Accuracy: 0.8147622427253371
Precision: 0.6794871794871795
Recall: 0.5683646112600537
F1 Score: 0.618978102189781
```

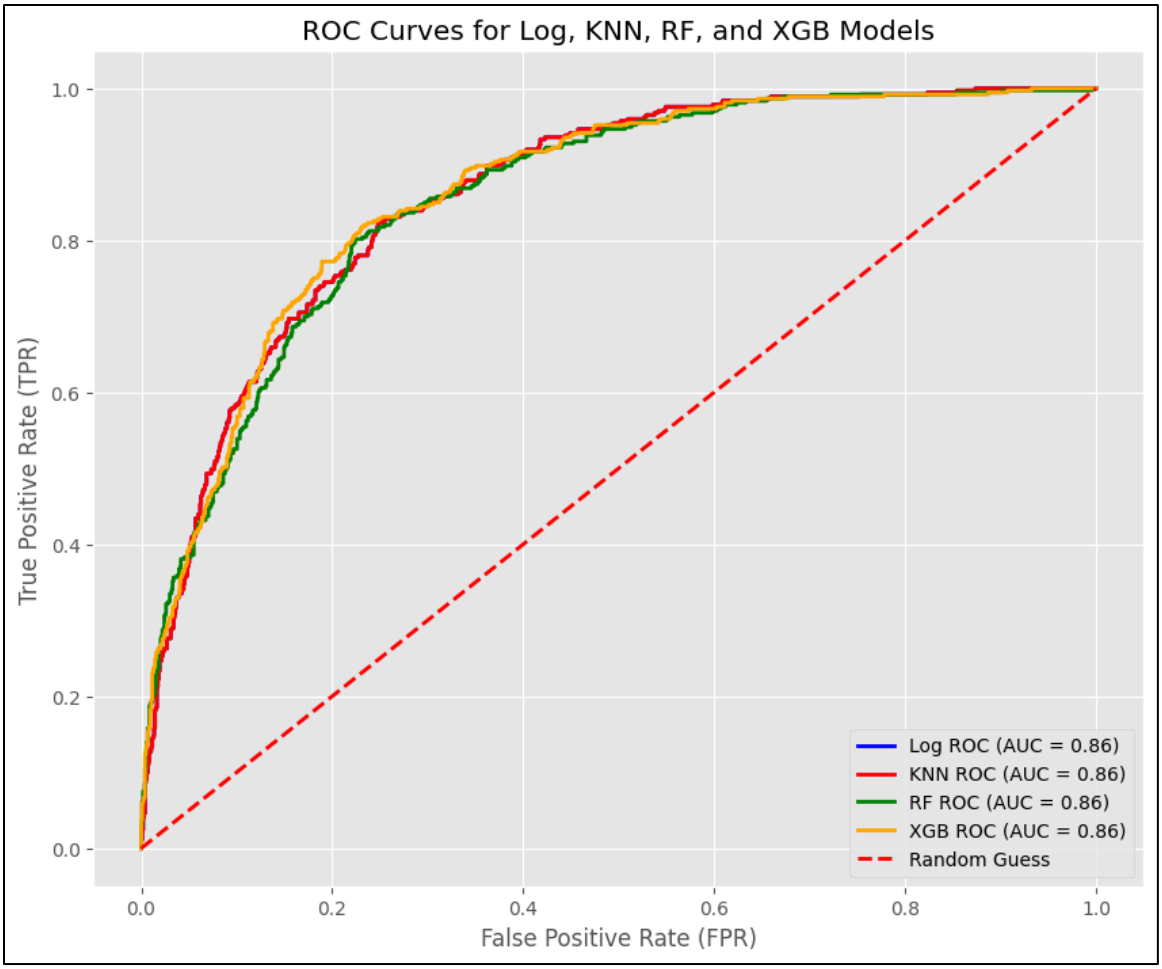
There is a good improvement in the XG Boost model after the hyperparameter tuning.

Model Comparison

After completing model training and hyperparameter tuning, our focus shifts to selecting the optimal model for predicting customer churn. To accomplish this, we'll compare various model performance evaluation metrics. By examining metrics such as accuracy, precision, recall, F1-score, and ROC-AUC, we gain insights into each model's predictive capability. This thorough evaluation process enables us to make an informed decision regarding the most suitable model for our specific churn prediction task.



- ☆ **Accuracy:** Logistic Regression has the highest accuracy (0.818) and KNN has the lowest accuracy (0.795)
- ☆ **Recall:** Logistic Regression has the highest recall (0.579) and Random Forest has the lowest recall (0.48)
- ☆ **Precision:** Logistic Regression has the highest precision (0.686) and KNN has the lowest precision (0.621)
- ☆ **F1 Score:** Logistic Regression has the highest F1 score (0.628) and Random Forest has the lowest F1 score (0.56)



☆ Based on the ROC-AUC curve, all four models appear to perform well at classifying customer churn prediction. There is little difference between the performance of Logistic Regression, Random Forest, and XG Boost, while KNN may have a slightly lower performance.

NOTE: So, the conclusion is the Logistic model is the best model for predicting customer churn.

New Data Prediction

We have done all the processes for the Customer Churn prediction model. Let's see how the model works for random input data.

```
# do some testing
final_log.predict(pd.DataFrame(columns=X_test.columns,
                               data=np.array([1, 0, 0, 0, 0.351370, 1, 1, 2, 0, 0,
                                              0, 0, 0, 0, 0, 1,
                                              0, -1.313208, -0.566163, -1.297286]).reshape(1,20)))
```

```
array([0])
```

Expecting output and model-provided output are the same. So, Our model works well for random input data also.

Perform Principal Components Analysis

We got testing accuracy for all models is less than 75%. So, let's do PCA with 90% components and see any improvements in built models.

```
# Applying PCA
pca = PCA(n_components=4)
xr_train_pca = pca.fit_transform(X_train)
xr_test_pca = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
explained_variance
```

```
array([0.55149121, 0.13087574, 0.07364673, 0.05189526])
```

Then let's fit the model for the new dataset and compare any improvements in the model after performing PCA.

```
# Logistic
pca_log = LogisticRegression(max_iter=2000,
                             penalty='none')
pca_log.fit(xr_train_pca, y_train)
```

```
# Knn
pca_knn = KNeighborsClassifier(metric='euclidean',
                               n_neighbors=11)
pca_knn.fit(xr_train_pca, y_train)
```

```
# Random Forest
pca_rfc = RandomForestClassifier(max_depth=20, min_samples_leaf=4,
                                 min_samples_split=7,
                                 n_estimators=60, random_state=42)
pca_rfc.fit(xr_train_pca, y_train)
```

```
# XG Boost
pca_xgb = XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.3, device=None, early_stopping_rounds=None,
              enable_categorical=True, eval_metric=None, feature_types=None,
              gamma=0.5, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.3, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=1, max_leaves=None,
              min_child_weight=1, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,
              num_parallel_tree=None, random_state=42)
pca_xgb.fit(xr_train_pca, y_train)
```

```
**************** Logistic Accuracy ****************
Accuracy before PCA:  0.8183108587650816
Accuracy after PCA:  0.8133427963094393

**************** K-NN Accuracy ****************
Accuracy before PCA:  0.794889992902768
Accuracy after PCA:  0.7970191625266146

**************** Random Forest Accuracy ****************
Accuracy before PCA:  0.8005677785663591
Accuracy after PCA:  0.8034066713981547

**************** XG Boost Accuracy ****************
Accuracy before PCA:  0.8147622427253371
Accuracy after PCA:  0.8048261178140526
```

The conclusion, there are not many improvements in the model accuracy after performing the PCA.

Actionable Insights

- **Customer segmentation:** Divide customers into groups based on their expected churn rates. Determine which high-risk segments require focused retention measures.
- **Service Analysis:** Determine the impact of various services on churn. Identify services with greater attrition rates and plan targeted promotions or enhancements.
- **Demographic Patterns:** Investigate demographic characteristics that affect turnover. Determine whether certain age groups or genders are more prone to churn and adjust marketing techniques accordingly.