# Trip Pricing with Taxi Mobility Analytics



The data used can be found in here:
https://www.kaggle.com/datasets/arashnic/taxi-pricing-with-mobility-analytics

# M1: Task exploration

An overview of the key skills and techniques in M1:

### Data Preprocessing

- **Handling Missing Data**: Identifying and handling missing values in specific columns like "Var1", "Type_of_cab", and "Customer_Since_Months". This involves assessing the amount of missing data and visualizing it using heatmaps and bar charts.
- **Data Cleaning**: Ensuring the data is free from inconsistencies and addressing missing values.

### Data Visualization

- **Bar Charts**: Used for visualizing the distribution of categorical variables (e.g., "Life_Style_Index", "Type_of_cab").
- **Heatmaps**: Visualizing missing values and correlations between variables. For example, heatmaps for missing data and pairwise correlations.
- **Pairplots**: Visualizing relationships and distributions of continuous variables to identify trends, skewness, and potential correlations.
- **Boxplots**: Used to understand the distribution, spread, and outliers in continuous variables (e.g., "Customer_Rating" by "Surge_Pricing_Type").
- **Mosaic Plots**: Used for categorical variable analysis, particularly to visualize relationships between two categorical variables (e.g., "Type_of_cab" and "Confidence_Life_Style_Index").
- **Histograms**: Visualizing the distribution of continuous variables (e.g., "Customer_Rating").

### Statistical Analysis

- **Descriptive Statistics**: Calculation of basic statistics (mean, median, standard deviation) for continuous variables.
- **Correlation Analysis**: Pairwise correlation calculations for continuous variables, identifying relationships between features (e.g., "Trip_Distance" and "Life_Style_Index").
- **Chi-Squared Tests**: For testing the dependency or independence between categorical variables (e.g., "Type_of_cab" and "Confidence_Life_Style_Index").
- **ANOVA (Analysis of Variance)**: Used to determine if there are statistically significant differences in continuous variables between different categories of a categorical variable (e.g., "Type_of_cab" and "Trip_Distance").

### Data Interpretation

- **Skewness and Distribution**: Identifying skewness in continuous variables (e.g., "Trip_Distance" being right-skewed, "Customer_Since_Months" being uniform).
- **Outlier Detection**: Identifying outliers in boxplots and histograms that could indicate anomalies or important trends.
- **Significant Relationships**: Identifying significant statistical relationships and dependencies using tests like ANOVA and Chi-squared, helping to understand how different features are interrelated.

### Exploratory Data Analysis (EDA)

- **Univariate Analysis**: Analyzing each variable individually to understand its distribution and central tendency (e.g., using histograms and descriptive statistics).
- **Bivariate Analysis**: Examining relationships between pairs of variables, both continuous-continuous and categorical-categorical, through pairplots, boxplots, and ANOVA.
- **Multivariate Analysis**: Assessing interactions between multiple variables using pairwise correlation and mosaic plots.

### Statistical Significance Testing

- **p-values**: Interpreting p-values from statistical tests (e.g., ANOVA, Chi-squared) to assess the significance of relationships and differences between variables.
- **Hypothesis Testing**: Using tests like Chi-squared and ANOVA to determine if observed relationships are statistically significant.

### Advanced Techniques

- **Correlation Heatmaps**: Understanding correlations between multiple continuous variables, highlighting strong or weak relationships.
- **Pairwise Comparisons**: Identifying interactions between continuous and categorical variables using statistical tests like ANOVA and visualizing them with boxplots and pairplots.

## Description of the task:

The task involves analysing historical data from Sigma Cabs, an Indian cab aggregator service, to predict the surge pricing type associated with different taxi trips. Surge pricing is a dynamic pricing strategy that businesses, such as taxi and delivery services, use to adjust prices in real-time based on demand. In the context of Sigma Cabs, surge pricing types are determined by various factors, including trip distance, type of cab, customer behavior, and other characteristics captured in the dataset. The predictive model aims to identify the surge pricing type for each trip request, enabling Sigma Cabs to match the right cabs with the right customers quickly and efficiently. This not only helps to optimise the cab allocation process but also to enhance customer satisfaction by providing more reliable and cost-effective service options. The target variable for this task (surge_pricing_type) indicates the type of surge pricing applied to a particular trip. This task can be marked as a multi-class classification problem. In multi-class classification, the goal is to categorize each instance into one of several classes, which, in our case, are the different types of surge pricing.

## Background of the dataset:

The dataset originates from Sigma Cabs, an Indian cab aggregator service. Sigma Cabs operates by providing a platform where customers can book cabs through their smartphone app. The service aggregates cabs from various providers and offers the best available option to their clients based on the requested trip details. Sigma Cabs has been operational for just under a year and during this period, they have been collecting data on surge pricing types from the service providers.

The motivation for solving this task is multifaceted, with benefits accruing to both the service provider and its customers:

- Operational Efficiency: Predicting surge pricing types accurately allows Sigma Cabs to match cabs with customers more effectively, optimising their dispatch system to ensure that cabs are available when and where they are needed most, reducing waiting times.
- Customer Satisfaction: By proactively managing surge pricing, Sigma Cabs can offer more competitive rates, enhancing customer satisfaction and loyalty. This is particularly important in a such competitive market, as India, where customers have multiple service options.
- Strategic Planning: Insights from the predictive model can inform Sigma Cabs' strategic decisions. Such as where to allocate resources more efficiently (analysing patterns in trip requests, can help predict demand across different times and locations), how to adjust pricing strategies, and how to tailor marketing efforts to target customer segments more effectively.
- Market Competitiveness: In the competitive landscape of cab aggregation services, being able to predict and manage surge pricing effectively gives Sigma Cabs a competitive edge. It allows them to offer better prices and service availability compared to competitors, potentially capturing a larger market share. Additionally, it can help with the implementation of the “price skimming” strategy.

Overall, the motivation behind analyzing this dataset lies in leveraging data-driven insights to enhance business strategies, improve customer satisfaction, increase operational efficiency, and drive higher profitability.

## Description of the dataset:

Features:

- Trip_ID: A unique identifier for each trip.
- Trip_Distance: The distance of the trip requested by the customer, in kilometers. This is a continuous variable that could influence the surge pricing type, as longer trips may have different pricing dynamics.
- Type_of_Cab: The category of the cab requested by the customer (e.g., Sedan, SUV), which may affect the pricing.
- Customer_Since_Months: The number of months the customer has been using Sigma Cabs' services, with 0 indicating the current month.
- Life_Style_Index: A proprietary index developed by Sigma Cabs indicating the lifestyle of the customer based on their behavior and interaction with the service.
- Confidence_Life_Style_Index: A categorical variable indicating the confidence Sigma Cabs has in the Life_Style_Index assigned to a customer.
- Destination_Type: Sigma Cabs categorizes each destination into one of 14 categories.
- Customer_Rating: The average lifetime rating of the customer by drivers.
- Cancellation_Last_1Month: The number of trips the customer has canceled in the last month.
- Var1, Var2, Var3: These are continuous variables that have been masked by Sigma Cabs. While their specific meanings are masked, they can be used for modeling purposes.
- Gender: The gender of the customer.

Target Variable:

- Surge_Pricing_Type: This is the target variable for the predictive model, indicating the type of surge pricing applied to a trip. It is a categorical variable with three distinct categories. These categories represent different levels or strategies of surge pricing, which the model aims to predict based on the features described above.

The Sigma Cabs dataset (sigma_cabs) is structured as follows:

- Number of Samples: There are 131,662 trip records in the dataset. Where 131662 are rows, with 14 being columns.
- Target Variable: Surge_Pricing_Type, with the distribution across three categories as follows: Type 2 (43.09%), Type 3 (36.24%), and Type 1 (20.67%). We will change the

type of our target values to the category, otherwise it will be treated as numerical continuous value.

- Dataset has a number of features, every one of which is stated and described above. We will drop Trip_ID, because we will not use it further and the shape will be (131662, 13).

The summary is shown in the picture 1.

```
RangeIndex: 131662 entries, 0 to 131661
Data columns (total 13 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Trip_Distance                131662 non-null  float64
 1   Type_of_Cab                  111452 non-null  object
 2   Customer_Since_Months        125742 non-null  float64
 3   Life_Style_Index             111469 non-null  float64
 4   Confidence_Life_Style_Index  111469 non-null  object
 5   Destination_Type             131662 non-null  object
 6   Customer_Rating              131662 non-null  float64
 7   Cancellation_Last_1Month     131662 non-null  int64
 8   Var1                         60632 non-null   float64
 9   Var2                         131662 non-null  int64
 10  Var3                         131662 non-null  int64
 11  Gender                       131662 non-null  object
 12  Surge_Pricing_Type           131662 non-null  category
dtypes: category(1), float64(5), int64(3), object(4)
```

Picture 1

Additional Data can be any data not already included in the dataset but which could be relevant and useful for enhancing analysis, models, or understanding of customer behavior and operational performance in the context of taxi mobility analytics. This supplementary data can serve various purposes, such as improving the accuracy of predictive models, providing deeper insights into customer behavior, or identifying operational efficiencies. Examples:

- Weather Data: Information about weather conditions (e.g., temperature, precipitation, weather events) can significantly influence travel behavior and cab demand (i.e. if it is raining customers are more likely to use the taxi services).
- Traffic Conditions: Real-time or historical traffic data can affect trip times, customer satisfaction, and pricing models.
- Public Transportation Schedules: Data on public transportation availability and schedules can provide context for demand patterns, especially in urban areas where people might choose between cabs and public transit based on convenience, cost, and timing.
- Events and Holidays: Information on local events, public holidays, and school vacations could be used to anticipate surges in demand or specific travel patterns.
- Economic Indicators: Various macro-economic data.
- Competitor Data: Information about competitors' pricing, promotions, and service offerings can provide a comparative perspective and help identify competitive

advantages or areas for improvement (usually such type of data is private, but several companies can conclude a contract to be able to access it).

Customer Feedback and Reviews: Textual data from customer feedback, social media mentions, and reviews can offer qualitative insights into customer satisfaction, service quality, and areas needing attention.

In this task we obtained two data sets: test and sigma_cabs. First one being the test set, and second one is the training set. The test set is used solely for assessing the performance of a fully-trained model. The key is that this dataset was not used in any way to make decisions about the model during the training phase. It provides an unbiased evaluation of the final model fit on the training dataset. The goal of having a validation set is to tune the parameters of your model, select features, and make decisions about which models to use without overfitting to your test set. Its primary role is to provide an unbiased evaluation of a model fit on the training dataset while tuning model parameters (e.g., hyperparameters). Since we have no separate validation set, there are several strategies to effectively validate models before testing. Such as splitting the training set (there are some common ratios to do that) or using cross-validation technique.

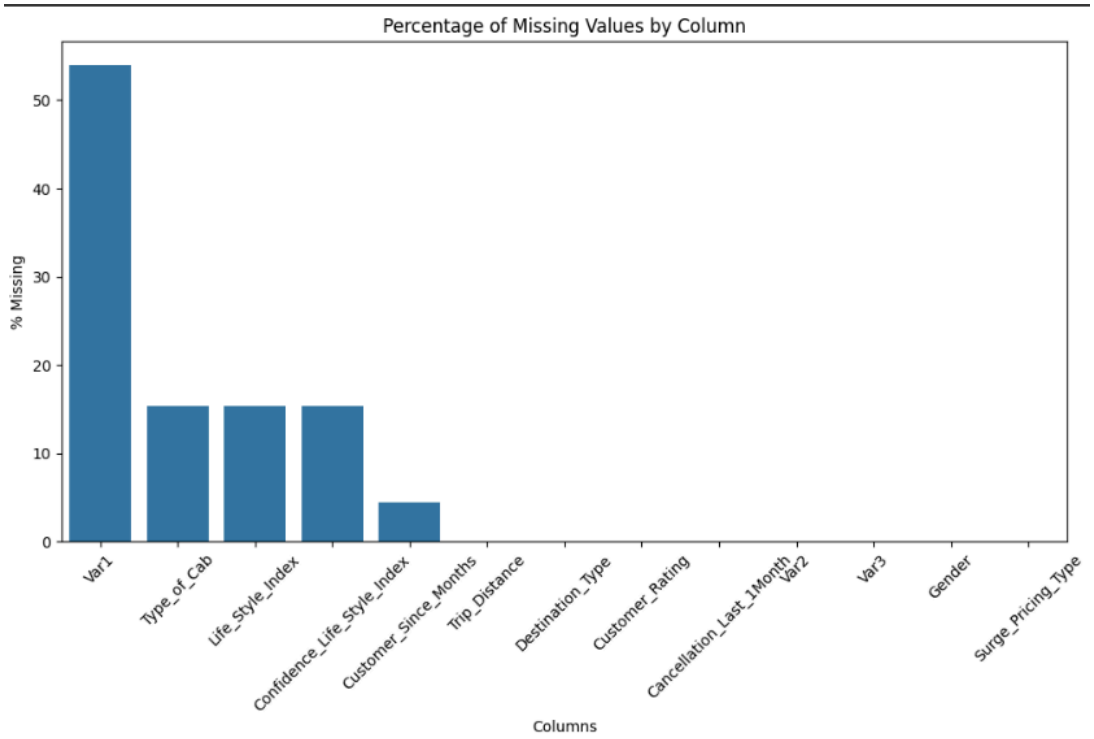
## Assess the quality of the data

There are missing values in ”Var1”, ”Type_of_cab”, ”Life_Style_Index”, ”Confidence_Life_Style_Index”, ”Customer_Since_Months”

| | Number of Missing Values | % Missing |
|---|---|---|
| Var1 | 71030 | 53.948748 |
| Type_of_Cab | 20210 | 15.349911 |
| Life_Style_Index | 20193 | 15.336999 |
| Confidence_Life_Style_Index | 20193 | 15.336999 |
| Customer_Since_Months | 5920 | 4.496362 |
| Trip_Distance | 0 | 0.000000 |
| Destination_Type | 0 | 0.000000 |
| Customer_Rating | 0 | 0.000000 |
| Cancellation_Last_1Month | 0 | 0.000000 |
| Var2 | 0 | 0.000000 |
| Var3 | 0 | 0.000000 |
| Gender | 0 | 0.000000 |
| Surge_Pricing_Type | 0 | 0.000000 |

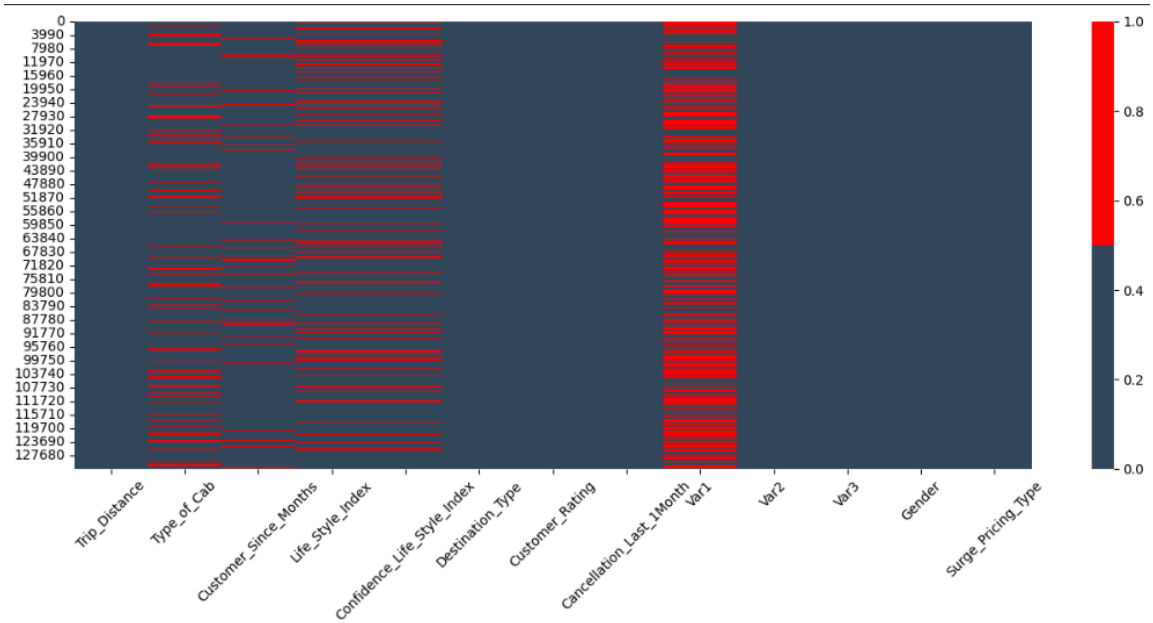
Picture 2

On the picture 3 is the bar charts for visualization of Picture 2 :




Picture 3

Heatmap for missing values is shown on the picture 4.




Picture 4

So we can see from the picture 2 and 3 that the column ”var1 has the biggest amount of missing values - around 50%, while others, particulary ”Type_of_cab”, ”Life_Style_Index”,

”Confidence_Life_Style_Index”, ”Customer_Since_Months” have around 15% of missing values.

## Perform preliminary data analysis

Basic statistics for every column is shown on the picture 5.

| | Trip_Distance | Type_of_Cab | Customer_Since_Months | Life_Style_Index | Confidence_Life_Style_Index | Destination_Type | Customer_Rating | Cancellation_Last_1Month | Var1 | Var2 | Var3 | Gender | Surge_Pricing_Type |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 131662.000000 | 111452 | 125742.000000 | 111469.000000 | 111469 | 131662 | 131662.000000 | 131662.000000 | 60632.000000 | 131662.000000 | 131662.000000 | 131662 | 131662.0 |
| unique | NaN | 5 | NaN | NaN | 3 | 14 | NaN | NaN | NaN | NaN | NaN | 2 | 3.0 |
| top | NaN | B | NaN | NaN | B | A | NaN | NaN | NaN | NaN | NaN | Male | 2.0 |
| freq | NaN | 31136 | NaN | NaN | 40355 | 77597 | NaN | NaN | NaN | NaN | NaN | 93900 | 56728.0 |
| mean | 44.200909 | NaN | 6.016661 | 2.802064 | NaN | NaN | 2.849458 | 0.782838 | 64.202698 | 51.202800 | 75.099019 | NaN | NaN |
| std | 25.522882 | NaN | 3.626887 | 0.225796 | NaN | NaN | 0.980675 | 1.037559 | 21.820447 | 4.986142 | 11.578278 | NaN | NaN |
| min | 0.310000 | NaN | 0.000000 | 1.596380 | NaN | NaN | 0.001250 | 0.000000 | 30.000000 | 40.000000 | 52.000000 | NaN | NaN |
| 25% | 24.580000 | NaN | 3.000000 | 2.654730 | NaN | NaN | 2.152500 | 0.000000 | 46.000000 | 48.000000 | 67.000000 | NaN | NaN |
| 50% | 38.200000 | NaN | 6.000000 | 2.798050 | NaN | NaN | 2.895000 | 0.000000 | 61.000000 | 50.000000 | 74.000000 | NaN | NaN |
| 75% | 60.730000 | NaN | 10.000000 | 2.946780 | NaN | NaN | 3.582500 | 1.000000 | 80.000000 | 54.000000 | 82.000000 | NaN | NaN |
| max | 109.230000 | NaN | 10.000000 | 4.875110 | NaN | NaN | 5.000000 | 8.000000 | 210.000000 | 124.000000 | 206.000000 | NaN | NaN |

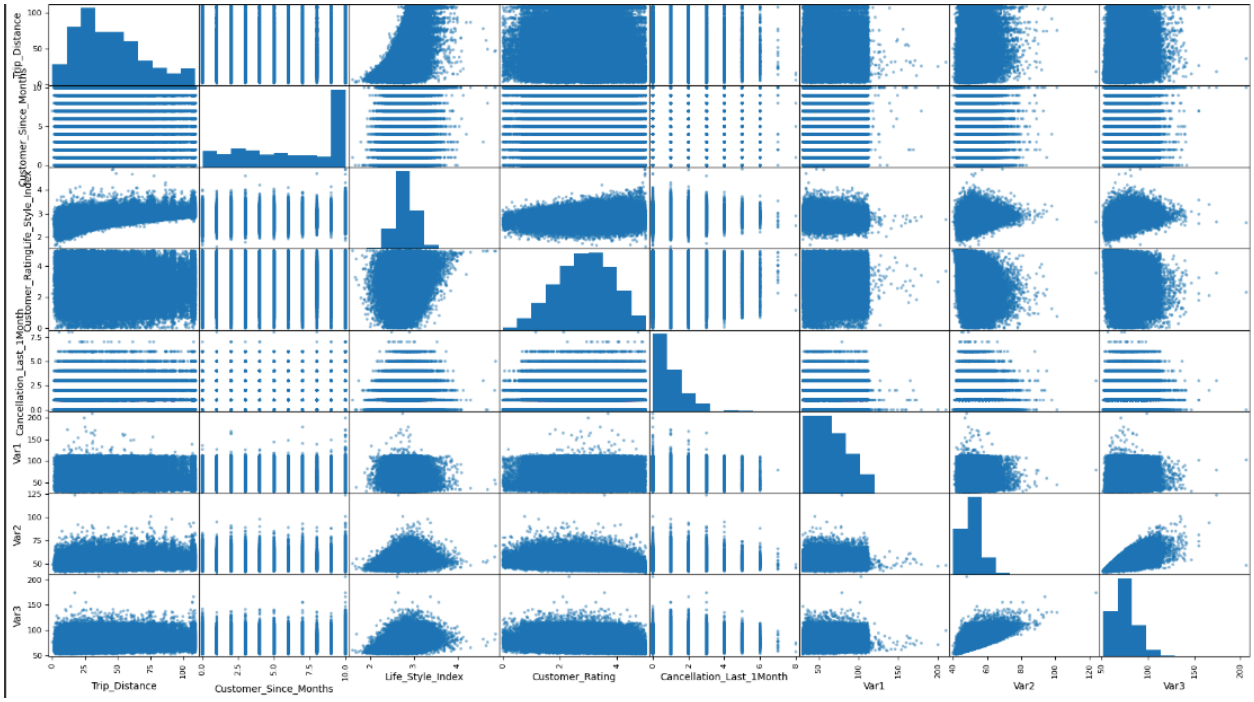
Picture 5

The medians, that mean that the half of data below this values the other half above, for continuous values is shown on picture 6.

```
Medians:
Trip_Distance                38.20000
Customer_Since_Months         6.00000
Life_Style_Index              2.79862
Customer_Rating               2.89500
Cancellation_Last_1Month      0.00000
Var1                         61.00000
Var2                         50.00000
Var3                         74.00000
dtype: float64
```

Picture 6

Distribution of numerical continuous features and their pairplots is shown on picture 7:




Picture 7

Quite brief description of distributions shown on picture 7:

Trip Distance: The distribution appears right-skewed, indicating that most trips are shorter, with a few longer trips with mean 44.2(that is shown on picture 5 as well).

Customer_Since_Months: This distribution looks uniform, indicating that customers are almost fairly evenly distributed across different lengths of service use except for being for 10 months, and with most customers using taxi around 10 months.

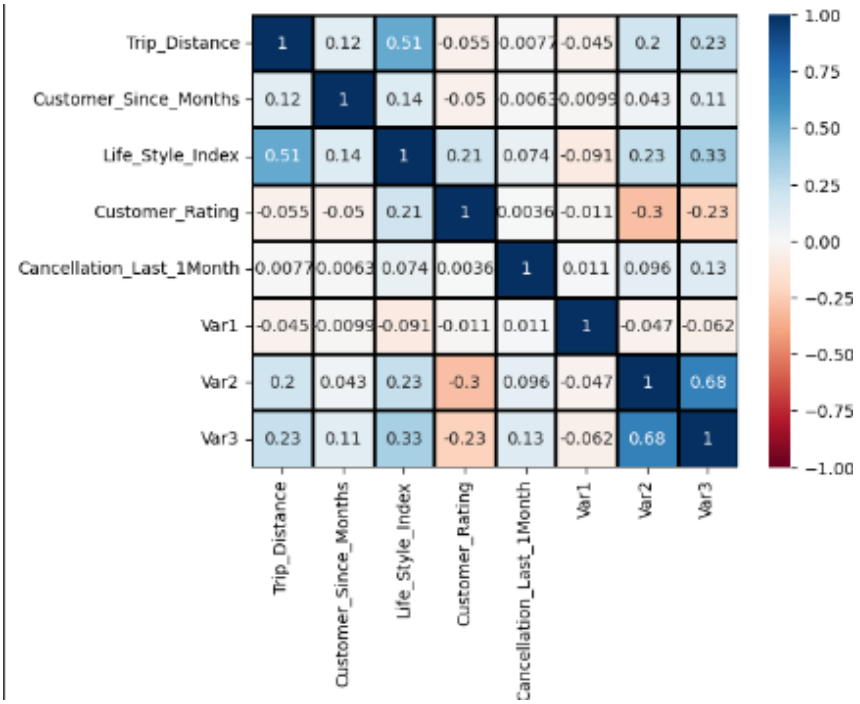
Life_Style_Index: Roughly speaking, appears to be normally distributed, indicating that most customers have a lifestyle index around a central value with fewer at the low and high ends with mean around 2.8.

Customer_Rating: This is roughly normally distributed but seems to be slightly left-skewed, indicating that there are more high ratings, with mean approximately 2.85(mean from picture 5)

Cancellation_Last_1 Month: Highly right-skewed, with most customers having no cancellations, and with quite significant number of customers having few cancellations, and a very few having a higher number of cancellations.

Var1, Var2, Var3: These are masked variables. Var1 appears right-skewed with mean 64.2; Var2 and Var3 are not uniform distributions with a slight peaks, having 51.2 and 75.1 means, respectively.

<u>Pairwise correlations between continuous and continuous values is shown on the picture 8:</u>




Picture 8

Quite short important description of correlations shown on picture 8:
From the correlation heatmap, it's clear that Var2 and Var3 have a strong positive correlation, meaning as one increases, so does the other.This positive relationship can be visible from the pairplot very well(picture 7).

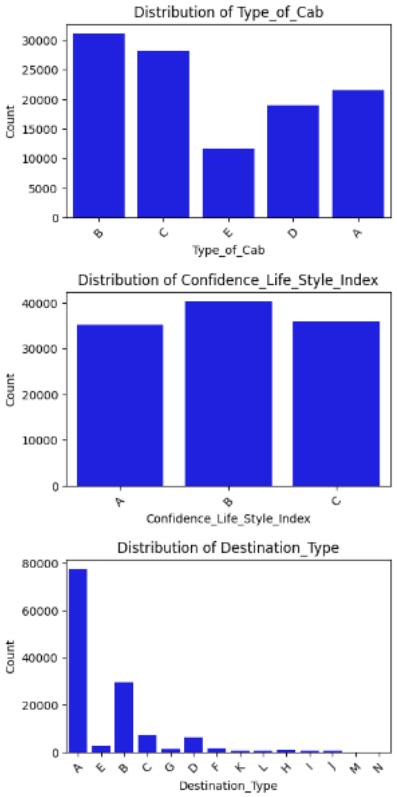
Trip_Distance and Life_Style_Index also show a moderate positive correlation, suggesting that longer trips might be associated with a higher lifestyle index. This positive relationship can be visible from the pairplot as well,and it seems that this relationship not linear(it can be seen on the picture 7).

Life_Style_Index and var3 have a slight positive relationship.With increasing one, the other is increasing also.

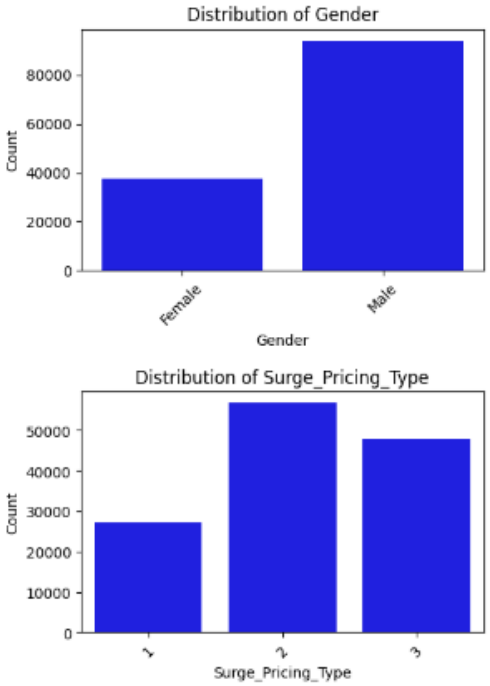
Customer rating and var2, Customer rating and var 3 have a slight negative relationship, meaning that the higher rating have the customer, the less values of var2 and var3.(it can be quite hardly seen on the picture 7)

For the rest of the pairs of features, the correlations are quite weak or negligible, and we can notice it even from pairplot, that there is no clear and strong relationships.

Distributions of categorical features and target are shown on the picture 9 and 10.


Distribution of Type_of_Cab
Count
Type_of_Cab
B
C
E
D
A
Distribution of Confidence_Life_Style_Index
Count
Confidence_Life_Style_Index
A
B
C
Distribution of Destination_Type
Count
Destination_Type
A
E
B
C
G
D
F
K
L
H
I
J
M
N

Picture 9


Distribution of Gender
Count
Female
Male
Gender
Distribution of Surge_Pricing_Type
Count
Surge_Pricing_Type

Picture 10

Quite brief description of distributions shown on picture 9 and 10:

Type_of_Cab: The distribution is not uniform, with certain categories occurring more frequently, particularly the 'B' category which has the highest frequency.

Confidence_Life_Style_Index: There are three categories, and the bar chart suggests that the distribution among these categories is not even. Categories 'A' and 'C' seem to be less common than 'B', with 'A' and 'C' having roughly equal counts, which are smaller than 'B'.

Destination_Type: This categorical variable shows a highly skewed distribution. One category ('A') is overwhelmingly more common than the others, with a steep drop-off in frequency for the remaining categories.

Gender: The distribution between the two categories 'Female' and 'Male' is imbalanced, with 'Male' being much more frequent than 'Female'.

Surge_Pricing_Type: There are three categories in this variable. The distribution is not even among them, with Type 2 being the most common, followed by Type 1, and Type 3 being the least common.

Pairwise correlations for categorical-categorical is written here:

Chi-Squared test for Type_of_Cab and Confidence_Life_Style_Index: The variables are dependent (p-value = 0.0).
Chi-Squared test for Type_of_Cab and Destination_Type: The variables are dependent (p-value = 0.0).
Chi-Squared test for Type_of_Cab and Gender: The variables are independent (p-value = 0.9747628372147825).
Chi-Squared test for Type_of_Cab and Surge_Pricing_Type: The variables are dependent (p-value = 0.0).
Chi-Squared test for Confidence_Life_Style_Index and Destination_Type: The variables are dependent (p-value = 6.475603121764031e-194).
Chi-Squared test for Confidence_Life_Style_Index and Gender: The variables are independent (p-value = 0.15914532609992274).
Chi-Squared test for Confidence_Life_Style_Index and Surge_Pricing_Type: The variables are dependent (p-value = 0.0).
Chi-Squared test for Destination_Type and Gender: The variables are independent (p-value = 0.6774149804023014).
Chi-Squared test for Destination_Type and Surge_Pricing_Type: The variables are dependent (p-value = 0.0).
Chi-Squared test for Gender and Surge_Pricing_Type: The variables are independent (p-value = 0.26289162562835167)

*Conclusion:* from these mini-outputs visible that, gender and other categorical features are

independent, and also target and gender is independent, with being dependent for features between each other, and features with the target.

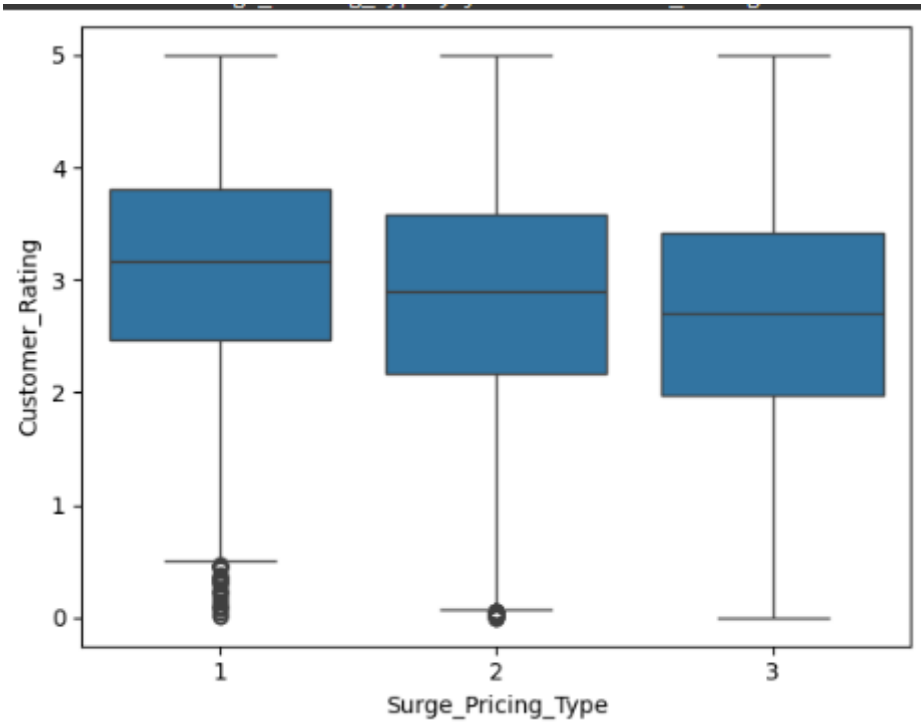
Pairwise correlations for numerical-categorical is written here:

ANOVA for Type_of_Cab and Trip_Distance: The variables are dependent (p-value = 1.658723349454301e-225).
ANOVA for Type_of_Cab and Customer_Since_Months: The variables are dependent (p-value = 0.0017099284219558196).
ANOVA for Type_of_Cab and Life_Style_Index: The variables are dependent (p-value = 8.000964478568974e-134).
ANOVA for Type_of_Cab and Customer_Rating: The variables are dependent (p-value = 2.0344770264238557e-282).
ANOVA for Type_of_Cab and Cancellation_Last_1Month: The variables are dependent (p-value = 7.869825728975572e-285).
ANOVA for Type_of_Cab and Var1: The variables are dependent (p-value = 7.13040131755329e-05).
ANOVA for Type_of_Cab and Var2: The variables are dependent (p-value = 9.850702957158185e-05).
ANOVA for Type_of_Cab and Var3: The variables are dependent (p-value = 1.6265732296697085e-28).
ANOVA for Confidence_Life_Style_Index and Trip_Distance: The variables are dependent (p-value = 0.0).
ANOVA for Confidence_Life_Style_Index and Customer_Since_Months: The variables are dependent (p-value = 1.130479637013696e-159).
ANOVA for Confidence_Life_Style_Index and Life_Style_Index: The variables are dependent (p-value = 0.0).
ANOVA for Confidence_Life_Style_Index and Customer_Rating: The variables are dependent (p-value = 0.0).
ANOVA for Confidence_Life_Style_Index and Cancellation_Last_1Month: The variables are dependent (p-value = 7.318954674416448e-123).
ANOVA for Confidence_Life_Style_Index and Var1: The variables are dependent (p-value = 1.2052418541189341e-05).
ANOVA for Confidence_Life_Style_Index and Var2: The variables are dependent (p-value = 9.015177047661916e-131).
ANOVA for Confidence_Life_Style_Index and Var3: The variables are dependent (p-value = 4.576322071607576e-264).
ANOVA for Destination_Type and Trip_Distance: The variables are dependent (p-value = 0.0).
ANOVA for Destination_Type and Customer_Since_Months: The variables are dependent (p-value = 6.436385906238674e-178).
ANOVA for Destination_Type and Life_Style_Index: The variables are dependent (p-value = 5.200342564123786e-251).
ANOVA for Destination_Type and Customer_Rating: The variables are dependent (p-value = 0.0).

ANOVA for Destination_Type and Cancellation_Last_1Month: The variables are dependent (p-value = 1.0834391766151342e-178).
ANOVA for Destination_Type and Var1: The variables are dependent (p-value = 0.0022695343894809784).
ANOVA for Destination_Type and Var2: The variables are dependent (p-value = 0.0).
ANOVA for Destination_Type and Var3: The variables are dependent (p-value = 7.765598846115361e-253).
ANOVA for Gender and Trip_Distance: The variables are independent (p-value = 0.43033149884544997).
ANOVA for Gender and Customer_Since_Months: The variables are independent (p-value = 0.6302399305725399).
ANOVA for Gender and Life_Style_Index: The variables are independent (p-value = 0.8128996215899846).
ANOVA for Gender and Customer_Rating: The variables are independent (p-value = 0.8650185543383901).
ANOVA for Gender and Cancellation_Last_1Month: The variables are independent (p-value = 0.40257422726401404).
ANOVA for Gender and Var1: The variables are independent (p-value = 0.8977350296662778).
ANOVA for Gender and Var2: The variables are independent (p-value = 0.24808158809366473).
ANOVA for Gender and Var3: The variables are independent (p-value = 0.20146783030254894).
ANOVA for Surge_Pricing_Type and Trip_Distance: The variables are dependent (p-value = 0.0).
ANOVA for Surge_Pricing_Type and Customer_Since_Months: The variables are dependent (p-value = 3.2463973308403438e-24).
ANOVA for Surge_Pricing_Type and Life_Style_Index: The variables are dependent (p-value = 2.52130302646828e-258).
ANOVA for Surge_Pricing_Type and Customer_Rating: The variables are dependent (p-value = 0.0).
ANOVA for Surge_Pricing_Type and Cancellation_Last_1Month: The variables are dependent (p-value = 0.0).
ANOVA for Surge_Pricing_Type and Var1: The variables are dependent (p-value = 2.487193795600867e-23).
ANOVA for Surge_Pricing_Type and Var2: The variables are dependent (p-value = 1.6302359318055932e-17).
ANOVA for Surge_Pricing_Type and Var3: The variables are dependent (p-value = 7.390507707458072e-75)

*Conclusion:* from these mini-outputs visible that, gender and all numerical continuous features are independent ,while the rest of categorical features (Type_of_Cab, Confidence_Life_Style_Index, Destination_Type, Surge_Pricing_Type) are dependent with all numerical continuous features

Surge_pricing_type – Customer_raiting box plot is shown on the picture 11.



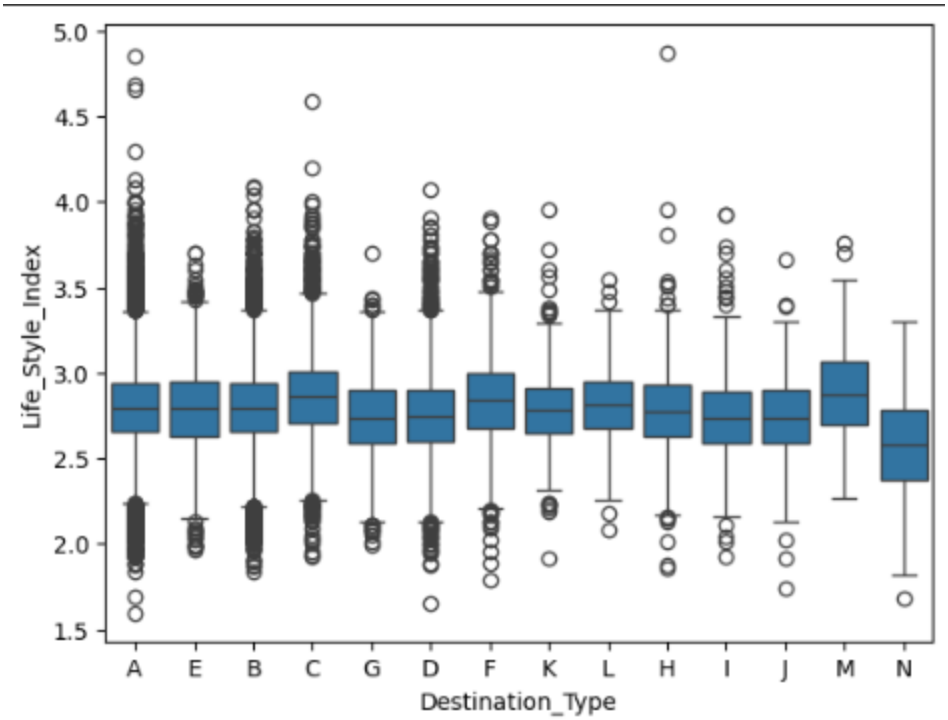

Picture 11

The box plot on the picture 11 displays the spread of 'Customer Rating' across different 'Surge Pricing Type' categories.
The median ratings for each surge pricing type are around the overall median of 2.895,and,approximately, indicating a central tendency around a similar value.

Despite the some similarities in medians, the ANOVA result with a p-value of 0.0 indicates that there are statistically significant differences in customer ratings among the different surge pricing types. While the medians are similar, there could be differences in the overall distribution of ratings across different surge pricing types that are not captured by the median alone.

There are outliers present for Surge Pricing Types 1 and 2, which are visibly below the main cluster of data. These outliers indicate that there are some customers with ratings significantly lower than the median rating for these surge pricing types. The histogram for Customer Rating that we provided before is right-skewed(picture 7), support our point as well, indicating that while most ratings are high, the outliers in the boxplot represent those few customers who have given much lower ratings.

Destination_type – Lifestyle_index box plot  is shown on the picture 12.



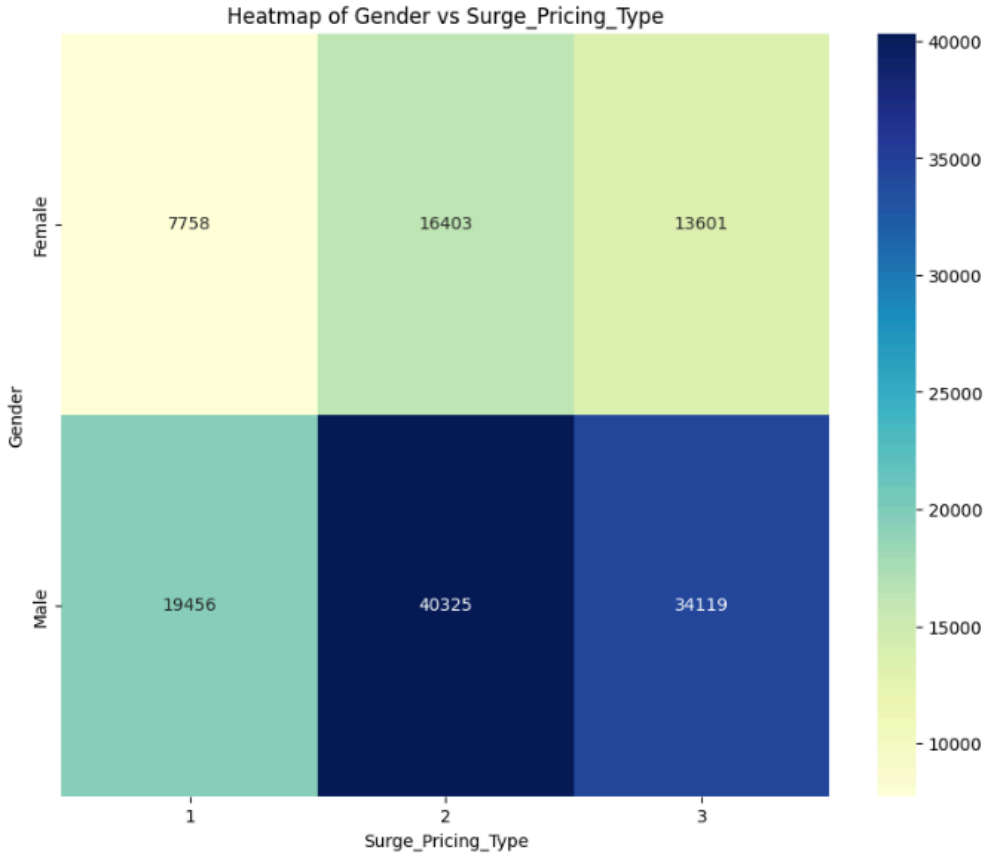

Picture 12

In this boxplot, there are noticeable variations in median Life Style Index values across different destination types, but many of them are near the overall median (2.79862). Some destination types have higher or lower medians, they don't appear to be drastically different.

The ANOVA results tell us that Destination Type and Life Style Index are statistically significantly dependent (p-value approximately 0), which means the variations in Life Style Index across destination types are not random; rather, the destination type has an influence on the Life Style Index.

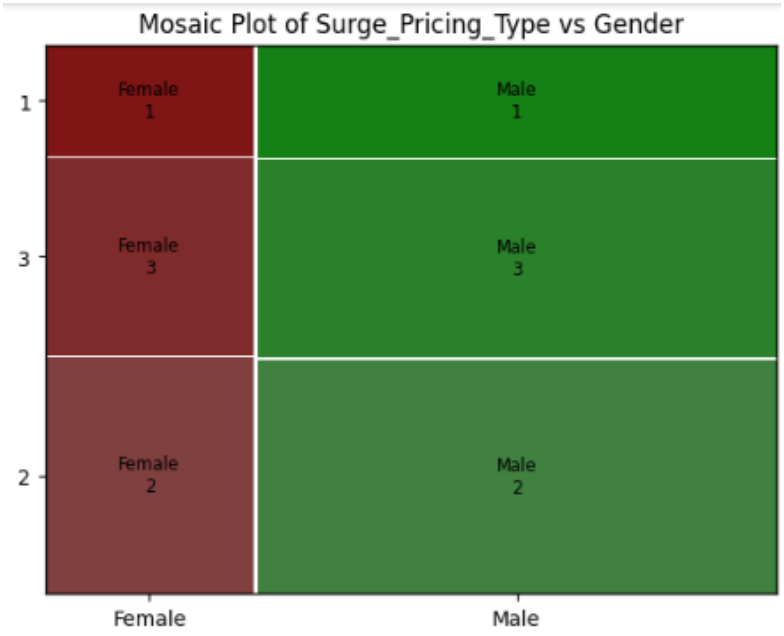
Outliers are scattered across various destination types. These outliers are both above and below the boxes, suggesting that while the majority of Life Style Index scores for each destination type are around the median (as shown by the box), there are customers with scores that are significantly different. This is in line with the distribution shown before for LifeStyle Index(which is on the picture 7), which appears to be more approximately normally distributed but still contains some outlier values.

Heatmap for gender-surge_pricing_type is shown on the picture 13.




Picture 13

Mosaic plot gender-surge_pricing_type  on the picture 14.

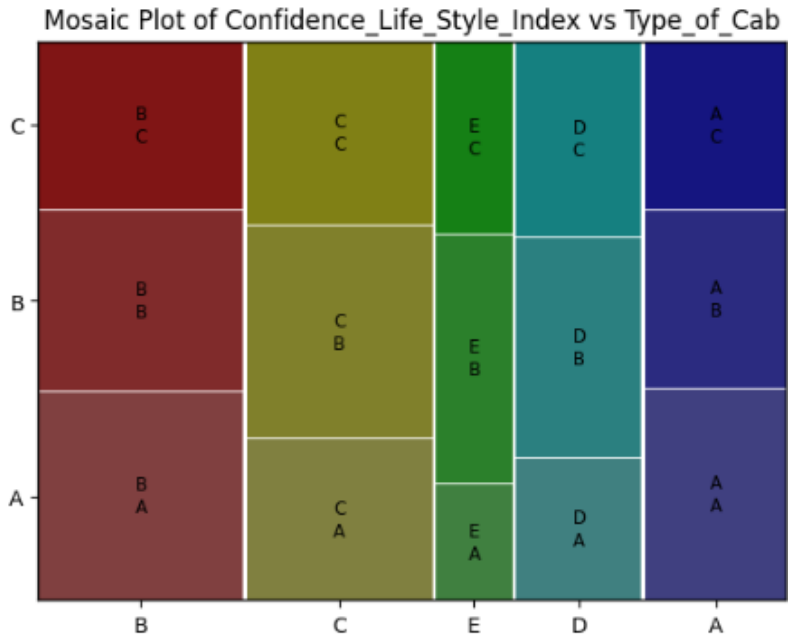


Picture 14

For both males and females, Surge_Pricing_Type 2 is the most common, followed by Surge_Pricing_Type 3, and then Surge_Pricing_Type 1,which it is visible from both heatmap and mosaic plots(pictures 13 and 14) and this trend is similar to the overall distribution of surge_pricing type(which is shown on the picture 10),

The counts for males are higher than those for females across all surge pricing types, which suggests there are simply more males in the dataset which is proven by distribution across the gender provided before(which is shown on the picture 10), and it is visible from heatmap(picture 13).

The Chi-squared test that we provided before suggests that there is no statistically significant association between Gender and Surge_Pricing_Type. Despite the raw counts being different due to the larger number of males in the dataset, the pattern across the three Surge Pricing Types appears relatively consistent between genders,. This consistency in pattern suggests that, roughly speaking,proportionally, males and females are similarly distributed across the different Surge Pricing Types, which is proven in our mosaic plot(picture 14).
Mosaic plot Type_of_cab- Confidence_Life_Style_Index is shown on the picture 15.

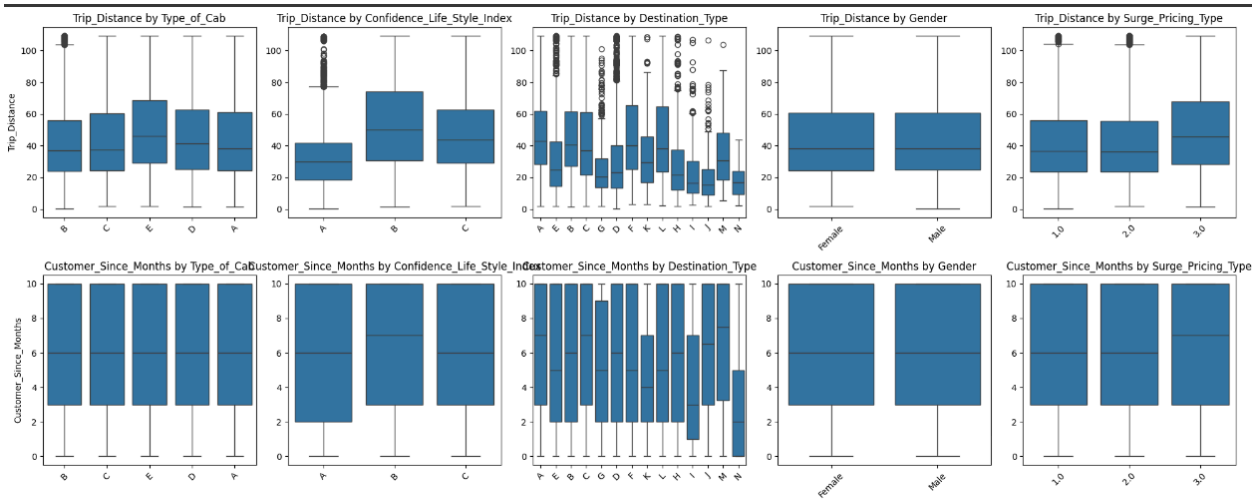



Picture 15

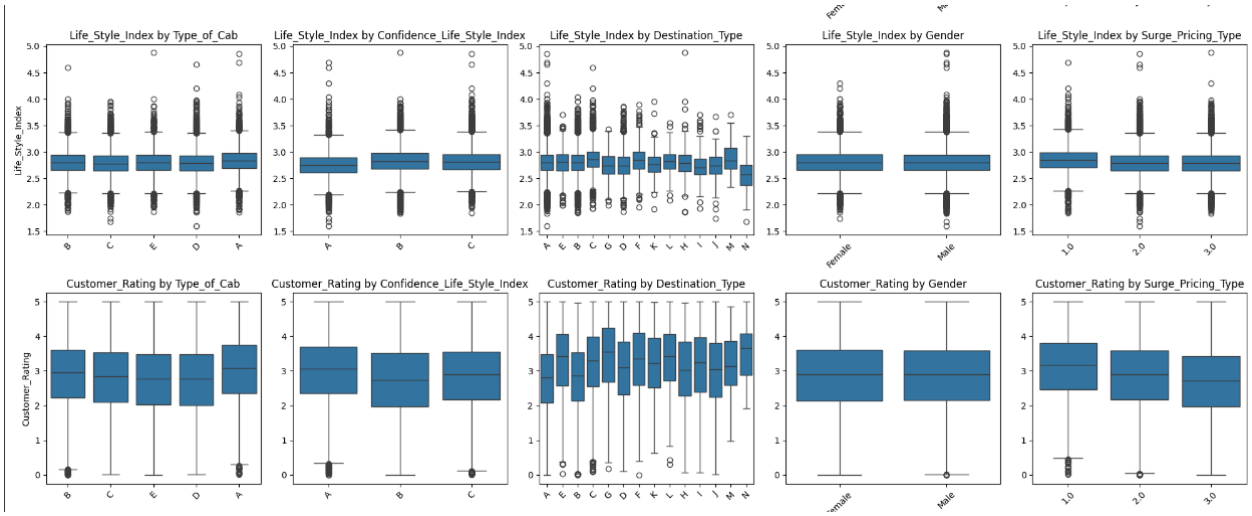
From the mosaic plot(picture 15), we can observe the following:

There's a variation in the size of rectangles within each type of cab category, suggesting that the distribution of Confidence_Life_Style_Index varies depending on the Type_of_Cab.
Some combinations, such as Type_of_Cab - B with Confidence_Life_Style_Index - A and Type_of_Cab - C with Confidence_Life_Style_Index - B appear more frequently, as indicated by larger rectangles.
Conversely, Type_of_Cab - E with Confidence_Life_Style_Index - A is less common, as shown by a smaller rectangle.

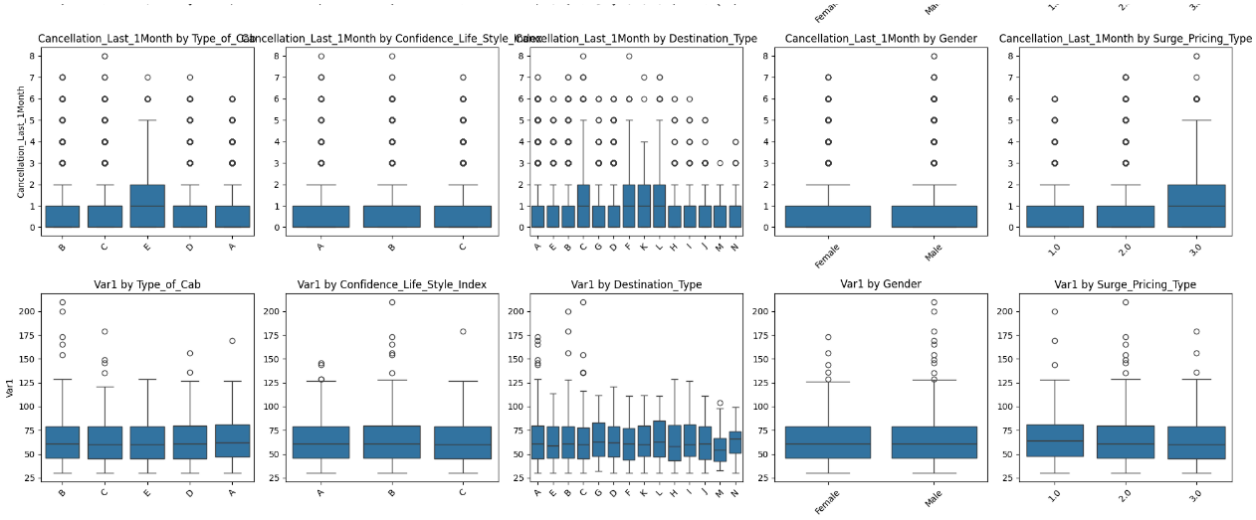
Here is shown all boxplots(continuous-categorical pairs).(picture 16,17,18,19)



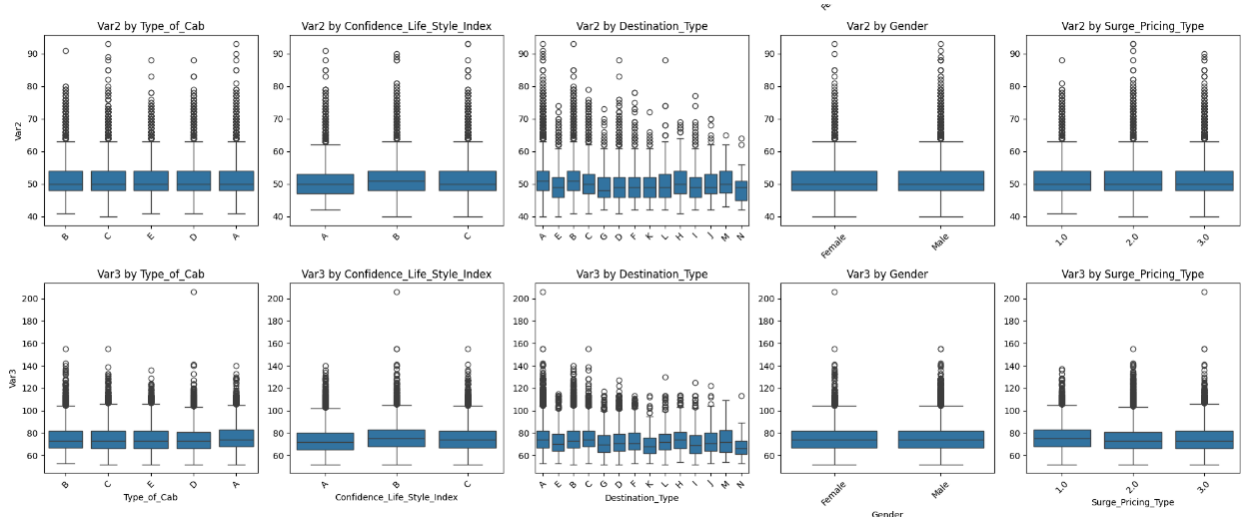

Picture 16




Picture 17




Picture 18




Picture 19

Let's interpret some noticeable box plots:

1.Trip Distance by Confidence Life Style Index:

The boxplot shows some outliers for longer trips across most categories of the Confidence Life Style Index. This aligns with the right skew, as these outliers represent the long trip distances(which is shown on the picture 7).

ANOVA indicates that the trip distances are dependent on the Confidence Life Style Index, suggesting that the variability in trip distance across the different levels of the Confidence Life Style Index is not random. The boxplot supports this, showing different medians and IQRs across categories.

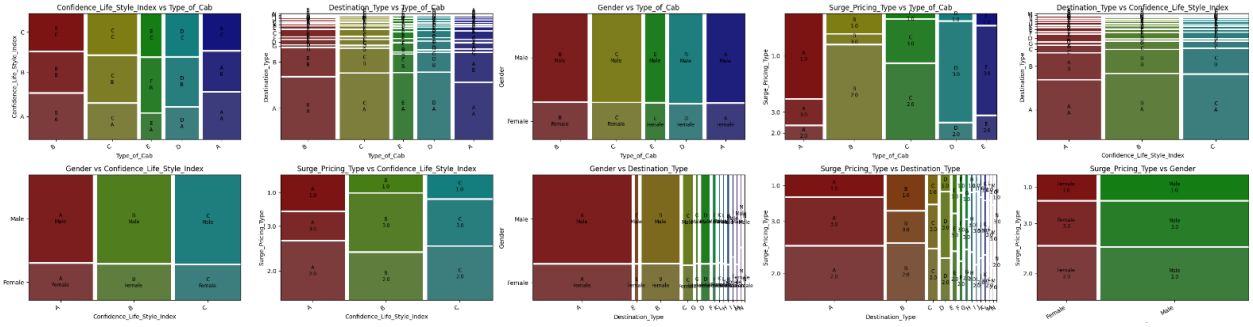
2.Trip Distance by Destination Type:

The trip distance varies considerably across different destination types, with some types showing a greater range and variability of trip distances than others.

The medians vary, indicating that the typical trip distance is different for each destination type. This could be related to the nature or location of each destination category. And the ANOVA result suggests a significant relationship between the type of destination and the trip distance.
There are many outliers, especially for certain destination types, suggesting that while most trips are short, there are significant numbers of unexpectedly long trips for those types.

3.Customer Since Months by Destination Type:
The boxplot showed some variation in customer loyalty across destination types, and the N destination is where new customers or those with a shorter duration of service use are more common.
And ANOVA suggests that the duration of customer engagement with the service (in months) is dependent on the destination type.

Here is shown all mosaic plots(categorical-categorical pairs).(picture 20)




Picture 20

Given the significance of the Chi-Squared test and the visual evidence from the mosaic plot, it is fair to say that the relationship between 'Type_of_Cab' and 'Surge_Pricing_Type' is outstanding in terms of showing a noticeable dependency or association between the two categorical variables.
And based on the mosaic plot, we can conclude that the combination of 'Type_of_Cab' category 'B' and 'Surge_Pricing_Type' 3.0 is notably more common compared to other combinations within the dataset. The size of the rectangle corresponding to this combination is substantially larger than others.

## Summary:

After defining the main objective of the project, first part of the research gave an insight of some problems of the stated task, as well as positive sides, such as properly described feature packs and two distinct data sets: training and test sets. The main parameters of data structure are shown in this paper.

Overall, the dataset exhibits relatively few missing values across most features, with the notable exception of the "Var1" feature, which shows approximately 50% missing values. In terms of correlations among continuous variables, quite noticeable correlations were observed between pairs such as Var2 and Var3, as well as between the Life Style Index and Trip

Distance. The Gender variable appears to be independent of all numerical continuous features. Conversely, other categorical features—specifically Type_of_Cab, Confidence_Life_Style_Index, Destination_Type, and Surge_Pricing_Type—show dependencies with the all numerical continuous features.In addition, Trip Distance by Destination Type is the most outstanding among other box plots. Regarding the interrelationships among all categorical variables, there is a general trend of dependency between each categorical values, except for Gender, which does not exhibit any dependency with other categorical variables.And the relationship between 'Type_of_Cab' and 'Surge_Pricing_Type' is outstanding.

By identifying key relationships between features and addressing data quality issues, such as missing values, it is now possible to process with development of a model that accurately predicts surge pricing types. This endeavor not only promises to enhance operational efficiency for Sigma Cabs but also stands to significantly improve customer satisfaction by offering more reliable and cost-effective services. The insights gained from analysis underline the importance of strategic data handling and feature understanding in tackling multi-class classification problems within the dynamic and competitive landscape of cab aggregator services.

# M2: Data preparation

An overview of the key skills and techniques in M2:

**Data Preparation and Cleaning:**

- **Handling Missing Data**: identifying and handle missing values by filling them with the mode or median (for categorical and continuous variables, respectively), using the `fillna` method in pandas.
- **Data Type Conversion**: The conversion of the target column to categorical type ensures it's treated appropriately by algorithms that expect categorical variables.

**Feature Engineering:**

- **Creating Hand-Crafted Features**: Based on domain knowledge, creating meaningful features like `Customer_Loyalty`, `Customer_Experience_Score`, `Trip_Efficiency_Score`, and `Var_Mean`. This requires a good understanding of the data and its context.
- **One-Hot Encoding**: Categorical data is transformed into binary vectors, ensuring the model can process the data correctly without introducing biases related to ordinal relationships.

**Data Splitting:**

- **Stratified Sampling**: The dataset is split into training and testing sets while preserving the target variable distribution, using `train_test_split` with the `stratify` parameter. This is crucial for ensuring that the training and test sets are representative of the overall class distribution, avoiding bias.

**Statistical Analysis and Feature Selection:**

- **ANOVA (Analysis of Variance)**: Performing ANOVA (`scipy.stats.f_oneway`) to test whether there are significant differences in the means of the new handcrafted features across the target classes. This is essential for understanding the relationship between features and the target variable.
- **Feature Selection**: Using `SelectKBest` along with `mutual_info_classif` to select the most informative features, improving the model's performance and reducing overfitting. Mutual information helps detect both linear and nonlinear relationships between features and the target variable.

**Visualization:**

- **Box Plots**: Using `seaborn.boxplot`, visualizing the distribution of handcrafted features across target categories. This helps in understanding the spread, median, and potential outliers in the data.
- **Correlation Analysis**: Analyzing correlations between features, especially in relation to the target variable, to identify which features might have strong relationships with the target.
- **Mosaic Plot**: While not explicitly mentioned, a mosaic plot could be used to visualize categorical feature dependencies, which is a useful visualization tool for categorical data analysis.

**Scaling and Normalization:**

- **Feature Scaling**: Applying standard scaling (`StandardScaler`) to ensure all features contribute equally to the model. This is especially important for algorithms that rely on distance metrics (like SVM, KNN) or gradient-based optimization methods (like logistic regression, neural networks).

**Machine Learning Foundations:**

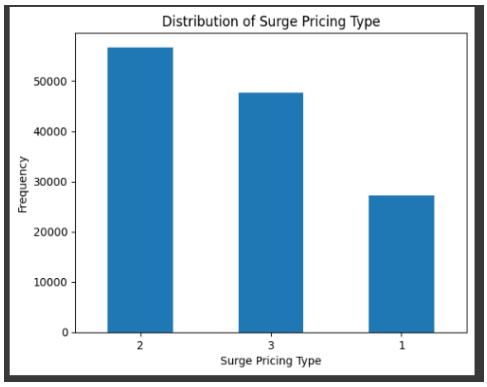
- **Data Transformation**: Converting data into appropriate formats (like converting dataframes to numpy arrays) ensures compatibility with machine learning algorithms.
- **Modeling Preparation**: While specific modeling is not covered in this section, the data preparation steps outlined are essential for building robust models, especially for algorithms that require numerical data with appropriate scaling and feature selection.

**Reproducibility:**

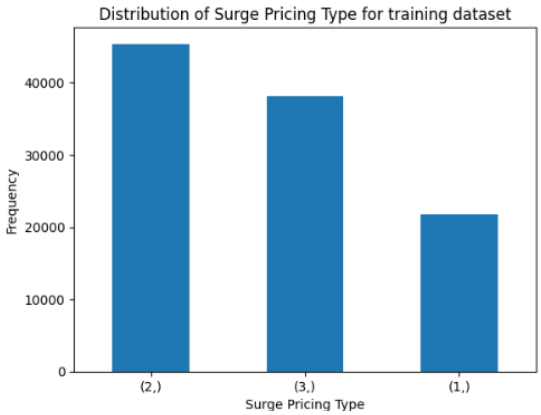
- **Random State**: Setting the random state in various steps (like splitting the data and scaling) ensures that the process is reproducible, which is crucial for experiments and comparisons.

So, at first we will drop Trip_Id and Var1 columns because Trip_Id does not have any valuable information for us and Var 1 because of the large amount of missing values(around 50%)( pic. 3).Then I will convert our target column to the "category" type, otherwise it will be treated as continuous. Then I set a seed for reproducibility.
During next step, we will divide our dataset on X, y dataframes where X will contain all features and y only target variable.Then we will convert X and y to numpy arrays. Then we will split our data to get next sets with proportion 80% for training and 20% for testing while preserving the target distribution using stratifying(A common split ratio for training and test sets is 80/20.This

split is considered a good balance because it provides enough data for learning the model while reserving a sufficient portion for an unbiased evaluation of its performance.): X_train, X_test, y_train, y_test using train_test_split.
*sklearn.model_selection.train_test_split*(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None) - Split arrays or matrices into random train and test subsets(but in our case it is not very random because we want to preserve target distribution after splitting)(and I fixed random state for reproducibility)
(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html documentation )
By stratifying, we maintain the class distribution, which helps in creating a more generalizable model and prevents introducing bias during training.




Picture 21




Picture 22

From pictures 21 and 22 it is visible that indeed the distribution of target variables is quite the same.

We need to fill missing values to allow algorithms that require complete data sets to operate correctly. Since in remained columns with missing values the amount it is not so big(pic 3,4), we will substitute them using DataFrame.fillna method that fill NA/NaN values using the specified method
(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html documentation). For Type_of_Cab and Confidence_Life_Style_Index we will substitute all

missing values with the mode since they are categorical. And we will substitute missing values in the 'Life_Style_Index' and 'Customer_Since_Months' columns with the mode and median, respectively.

Here is information about x_training set:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105329 entries, 0 to 105328
Data columns (total 11 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Trip_Distance                105329 non-null  float64
 1   Type_of_Cab                  105329 non-null  object
 2   Customer_Since_Months        105329 non-null  float64
 3   Life_Style_Index             105329 non-null  float64
 4   Confidence_Life_Style_Index  105329 non-null  object
 5   Destination_Type             105329 non-null  object
 6   Customer_Rating              105329 non-null  float64
 7   Cancellation_Last_1Month     105329 non-null  int64
 8   Var2                         105329 non-null  int64
 9   Var3                         105329 non-null  int64
 10  Gender                       105329 non-null  object
dtypes: float64(4), int64(3), object(4)
memory usage: 8.8+ MB
```

Picture 23

Here is information about x_test set:

```
RangeIndex: 26333 entries, 0 to 26332
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Trip_Distance                26333 non-null  float64
 1   Type_of_Cab                  26333 non-null  object
 2   Customer_Since_Months        26333 non-null  float64
 3   Life_Style_Index             26333 non-null  float64
 4   Confidence_Life_Style_Index  26333 non-null  object
 5   Destination_Type             26333 non-null  object
 6   Customer_Rating              26333 non-null  float64
 7   Cancellation_Last_1Month     26333 non-null  int64
 8   Var2                         26333 non-null  int64
 9   Var3                         26333 non-null  int64
 10  Gender                       26333 non-null  object
dtypes: float64(4), int64(3), object(4)
memory usage: 2.2+ MB
```

Picture 24

Then we need to *encode* our categorical data since many machine learning algorithms require numerical inputs. Encoding transforms categorical data into numerical representations, allowing algorithms to process them effectively.

For this, we will use One-Hot Encoding.

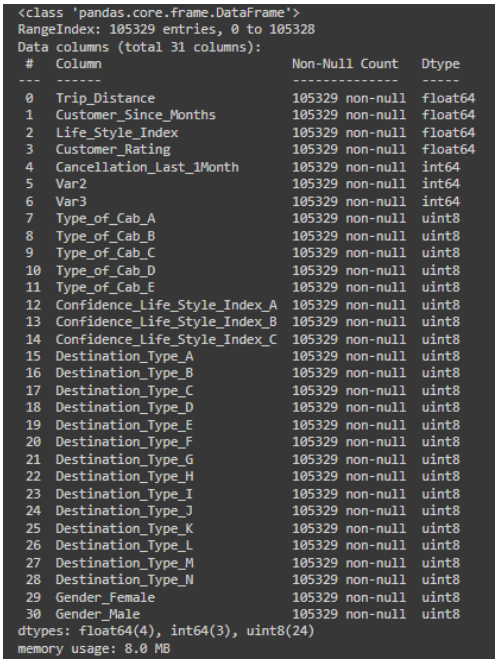
*One-Hot Encoding* is a popular technique where each category is represented as a binary vector. For each category, a new binary column is created, and only one of these columns has a value of 1 corresponding to the category, while the others are 0. This ensures that categorical variables with multiple categories can be represented as numerical features without introducing any ordinality between the categories. It also ensures that each category is represented independently, which can prevent biases in the model. Additionally, one-hot encoding allows algorithms to interpret categorical variables without making assumptions about the relationships between categories.

*pandas.get_dummies*(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False, dtype=None)

Convert categorical variable into dummy/indicator variables.

Each variable is converted in as many 0/1 variables as there are different values. Columns in the output are each named after a value; if the input is a Data Frame, the name of the original variable is prepended to the value.

(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html documentation )
Here is the result for x_training data frame:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105329 entries, 0 to 105328
Data columns (total 31 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   Trip_Distance                   105329 non-null  float64
 1   Customer_Since_Months           105329 non-null  float64
 2   Life_Style_Index                105329 non-null  float64
 3   Customer_Rating                 105329 non-null  float64
 4   Cancellation_Last_1Month        105329 non-null  int64
 5   Var2                            105329 non-null  int64
 6   Var3                            105329 non-null  int64
 7   Type_of_Cab_A                   105329 non-null  uint8
 8   Type_of_Cab_B                   105329 non-null  uint8
 9   Type_of_Cab_C                   105329 non-null  uint8
 10  Type_of_Cab_D                   105329 non-null  uint8
 11  Type_of_Cab_E                   105329 non-null  uint8
 12  Confidence_Life_Style_Index_A   105329 non-null  uint8
 13  Confidence_Life_Style_Index_B   105329 non-null  uint8
 14  Confidence_Life_Style_Index_C   105329 non-null  uint8
 15  Destination_Type_A              105329 non-null  uint8
 16  Destination_Type_B              105329 non-null  uint8
 17  Destination_Type_C              105329 non-null  uint8
 18  Destination_Type_D              105329 non-null  uint8
 19  Destination_Type_E              105329 non-null  uint8
 20  Destination_Type_F              105329 non-null  uint8
 21  Destination_Type_G              105329 non-null  uint8
 22  Destination_Type_H              105329 non-null  uint8
 23  Destination_Type_I              105329 non-null  uint8
 24  Destination_Type_J              105329 non-null  uint8
 25  Destination_Type_K              105329 non-null  uint8
 26  Destination_Type_L              105329 non-null  uint8
 27  Destination_Type_M              105329 non-null  uint8
 28  Destination_Type_N              105329 non-null  uint8
 29  Gender_Female                   105329 non-null  uint8
 30  Gender_Male                     105329 non-null  uint8
dtypes: float64(4), int64(3), uint8(24)
memory usage: 8.0 MB
```

Picture 25

We can see from picture 25 that dimensionality our data increased.
Then we will create new hand-crafted features.*Hand-crafted features*, also known as engineered features, are new features created from existing data based on domain knowledge, and they are intended to capture specific patterns, relationships, or characteristics present in the data.
In our case we will create 4 hand-crafted features:
1.Customer_Loyalty: This feature will be derived by combining Customer_Since_Months and Customer_Rating. These two columns are multiplied, under the assumption that a customer who has been using the service for a longer time and has a high rating is more loyal. This could affect the surge pricing type due to potentially different behavior in booking cabs.
2.Customer_Experience_Score: The Customer_Experience_Score handcrafted feature is designed by combining Customer_Rating with Cancellation_Last_1Month , using the formula: Customer_Experience_Score = Customer_Rating / (1 + cancellation_Last_1Month ). It provides a view of customer experience by balancing positive feedback (rating) against potentially negative actions (cancellations). The feature implicitly segments customers into those who rate highly but seldom cancel (indicating strong loyalty and satisfaction) versus those who may still rate positively but have higher cancellation rates, potentially signaling service or expectation mismatches.
3.Trip_Efficiency_Score: The Trip_Efficiency_Score is a handcrafted feature created from the product of Trip_Distance and Life_Style_Index. This feature merges two distinct aspects: the distance of the trip and the lifestyle index, presumably a measure of customer lifestyle preferences or qualities. It suggests a notion of value or efficiency derived from how far customers are willing to travel and their lifestyle attributes.Longer distances combined with

certain lifestyle indices could indicate higher engagement or dependence on the service.By understanding trip efficiency in the context of lifestyle, services can tailor experiences, offers, or recommendations more effectively, potentially increasing customer satisfaction and loyalty.(And from picture 8 is visible the correlation)
4.Var_Mean: Since Var2, and Var3 are masked continuous variables they can be used for modeling purposes, creating a feature that represents the mean of these variables for each customer that could provide a generalized indicator of whatever these variables represent. This assumes that the variables are related or measure aspects of behavior that could collectively influence the surge pricing type.
Here is some values for these new hand crafted features:

| | Customer_Loyalty | Trip_Efficiency_Score | Customer_Experience_Score | Var_Mean |
|---|---|---|---|---|
| 0 | 22.46625 | 90.350712 | 2.496250 | 55.0 |
| 1 | 0.00000 | 158.711534 | 2.116250 | 54.5 |
| 2 | 28.36250 | 126.673240 | 1.418125 | 58.0 |
| 3 | 7.55000 | 57.239664 | 0.503333 | 62.0 |
| 4 | 18.71500 | 37.779729 | 2.339375 | 58.0 |

Picture 26

Then I performed ANOVA using *scipy.stats.f_oneway(*samples, axis=0)* (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html documentation) to know whether there are dependencies between new handcrafted variables and target. scipy.stats.f_oneway(*samples, axis=0) performs one-way ANOVA.One-way ANOVA is used when there is one independent variable (factor) and one dependent variable, and we want to compare the means of the dependent variable across different levels of the independent variable. The purpose is to see if there's a significant effect of the independent variable on the dependent variable.And the one-way ANOVA tests the null hypothesis that two or more groups have the same population mean.

Here is the result:

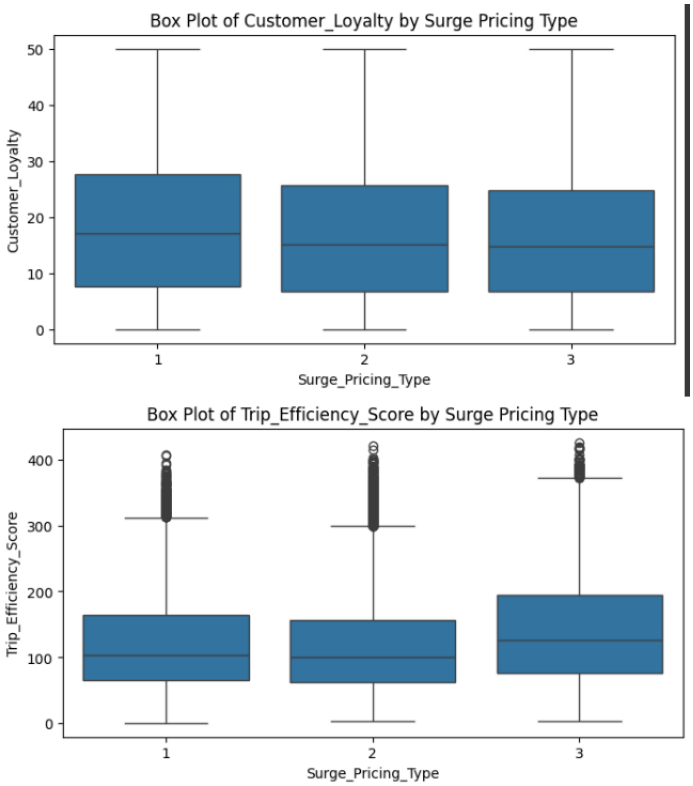
```
Customer_Loyalty is dependent on the target variable (F-statistic=175.51349821679926, p-value=7.9835128000899e-77)
Trip_Efficiency_Score is dependent on the target variable (F-statistic=1097.7408927044594, p-value=0.0)
Customer_Experience_Score is dependent on the target variable (F-statistic=3366.8389504411216, p-value=0.0)
Var_Mean is dependent on the target variable (F-statistic=99.49445704676549, p-value=6.77442588727005e-44)
```

Picture 27

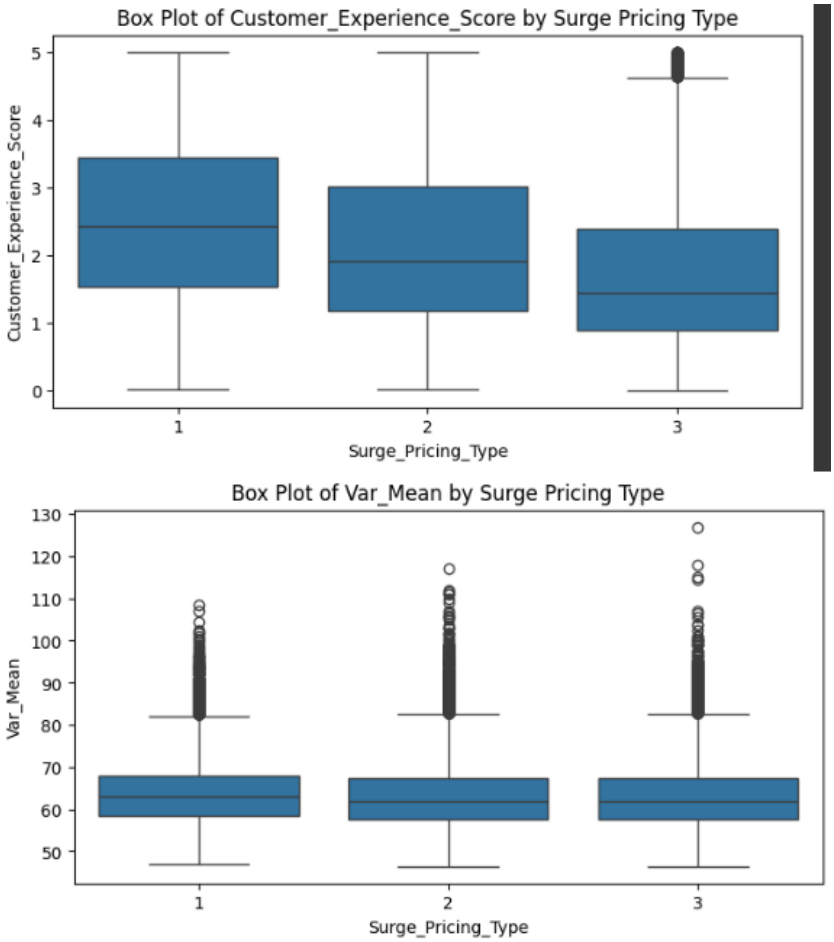
So target and all new handcrafted variables are dependent.(pic. 27)

Then the boxplots of the new handcrafted features and target are built using *seaborn.boxplot*(A box plot (or box-and-whisker plot) that show the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.

Here are the results:


Box Plot of Customer_Loyalty by Surge Pricing Type
Customer_Loyalty
Surge_Pricing_Type
0
10
20
30
40
50
1
2
3
Box Plot of Trip_Efficiency_Score by Surge Pricing Type
Trip_Efficiency_Score
Surge_Pricing_Type
0
100
200
300
400
1
2
3

Picture 28




Picture 29

So from pic. 28 and 29 we can say next:

-Customer_Loyalty:

Box Plot Insights: The box plot for Customer_Loyalty shows relatively similar medians across the surge pricing types, with type 1 having a slightly higher median and more variability as indicated by the longer box and whiskers. This may suggests that customers with higher loyalty scores may have 1st surge pricing type.There are no visible outliers, indicating that all Customer Loyalty scores fall within a reasonable range of values.

-Trip_Efficiency_Score:

Box Plot Insights: The Trip_Efficiency_Score shows a higher median for surge pricing type 3 and the most outliers for all types, especially type 2. The large number of outliers suggests that there are customers with very high efficiency scores, which could indicate infrequent but long trips or high lifestyle index values. The higher median for type 3 could mean that customers with higher efficiency can have this surge pricing type..

-Customer_Experience_Score:

Box Plot Insights: For Customer_Experience_Score, the median is noticeably higher for surge pricing type 1. This might suggest that customers who experience lower surge pricing have higher average ratings and fewer cancellations, which correlates with a better customer

experience.And there are outliers only for 3rd type of surge pricing type which mean that customer experience score can be very high for this type.

-Var_Mean:
Box Plot Insights: The box plot for Var_Mean shows less distinction between the medians of the surge pricing types. And there are many outliers for all types, indicating that there are a number of trips with unusually high Var Mean scores compared to the bulk of the data.

Then we will see dependency between target and features for further aims using sklearn.feature_selection.SelectKBest(https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html documentation).
*SelectKBest* is a feature selection method provided by the scikit-learn library in Python. It selects features according to the k highest scores. Essentially, it allows to set k to a fixed number of top features to select, and it will retain only those number of features in the dataset. The selection is based on statistical tests used to determine the relationship between each feature and the target variable.And these k best features used for better results of prediction.

Muutual_info_classif(https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html documentation) is a parameter within SelectKBest that specifies the statistical test to use for selecting the features. When score_func to mutual_info_classif is set,mutual information is chosen for using for a categorical target variable to score the features(I fixed random state in here by using lambda function for reproducibility)

Mutual information (MI) measures the dependency between variables. It is equal to zero if two random variables are independent and increases as the dependency between the variables increases. In the context of feature selection, mutual information is a nonparametric method that can capture any kind of statistical dependency, including nonlinear relationships, between a feature and the target variable.

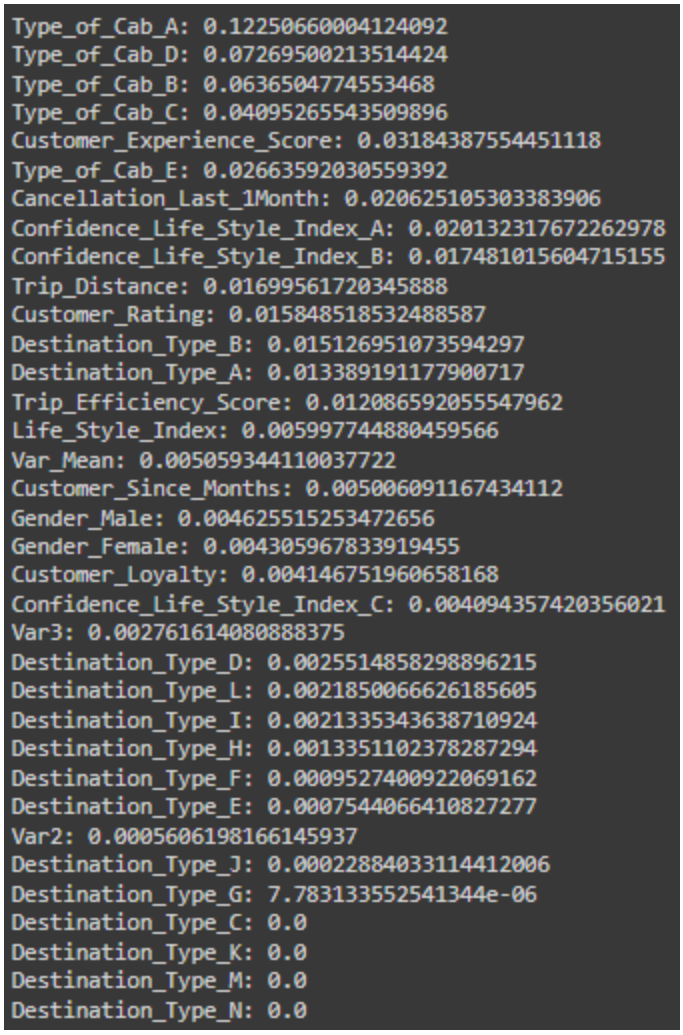
And here are some benefits of mutual_info_classif:
Detects Any Kind of Relationship: Mutual information can capture both linear and nonlinear relationships between features and the target variable.

Versatile: It makes no assumptions about the data distribution, making it applicable to a wide range of data types.

Applicable to Classification: It's tailored for discrete targets, making it suitable for classification tasks, including multiclass scenarios.

Robustness: Helps in reducing overfitting by selecting the most informative features.

Here are the results:




Picture 30

SelectKbest results.(Pic. 30)

Here is the information about our x_training set::

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105329 entries, 0 to 105328
Data columns (total 35 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   Trip_Distance                   105329 non-null  float64
 1   Customer_Since_Months           105329 non-null  float64
 2   Life_Style_Index                105329 non-null  float64
 3   Customer_Rating                 105329 non-null  float64
 4   Cancellation_Last_1Month        105329 non-null  float64
 5   Var2                            105329 non-null  float64
 6   Var3                            105329 non-null  float64
 7   Type_of_Cab_A                   105329 non-null  uint8
 8   Type_of_Cab_B                   105329 non-null  uint8
 9   Type_of_Cab_C                   105329 non-null  uint8
 10  Type_of_Cab_D                   105329 non-null  uint8
 11  Type_of_Cab_E                   105329 non-null  uint8
 12  Confidence_Life_Style_Index_A   105329 non-null  uint8
 13  Confidence_Life_Style_Index_B   105329 non-null  uint8
 14  Confidence_Life_Style_Index_C   105329 non-null  uint8
 15  Destination_Type_A              105329 non-null  uint8
 16  Destination_Type_B              105329 non-null  uint8
 17  Destination_Type_C              105329 non-null  uint8
 18  Destination_Type_D              105329 non-null  uint8
 19  Destination_Type_E              105329 non-null  uint8
 20  Destination_Type_F              105329 non-null  uint8
 21  Destination_Type_G              105329 non-null  uint8
 22  Destination_Type_H              105329 non-null  uint8
 23  Destination_Type_I              105329 non-null  uint8
 24  Destination_Type_J              105329 non-null  uint8
 25  Destination_Type_K              105329 non-null  uint8
 26  Destination_Type_L              105329 non-null  uint8
 27  Destination_Type_M              105329 non-null  uint8
 28  Destination_Type_N              105329 non-null  uint8
 29  Gender_Female                   105329 non-null  uint8
 30  Gender_Male                     105329 non-null  uint8
 31  Customer_Loyalty                105329 non-null  float64
 32  Trip_Efficiency_Score           105329 non-null  float64
 33  Customer_Experience_Score       105329 non-null  float64
 34  Var_Mean                        105329 non-null  float64
dtypes: float64(11), uint8(24)
memory usage: 11.3 MB
```

Picture 31

Then scaling on numerical continuous data will be performed.Scaling - is adjusting the range of feature values in a dataset to a common scale.It helps ensure that each feature contributes equally to the analysis, improves the convergence speed of algorithms, and helps avoid issues with numerical stability and precision. Scaling makes the training process more efficient and can lead to better performance of the model.

We will use sklearn.preprocessing.StandardScaler (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn-preprocessing-standardscaler documentation) - standardize features by removing the mean and scaling to unit variance.(It transforms the data such that each feature has a mean of 0 and a standard deviation of 1):

The standard score of a sample x is calculated as:

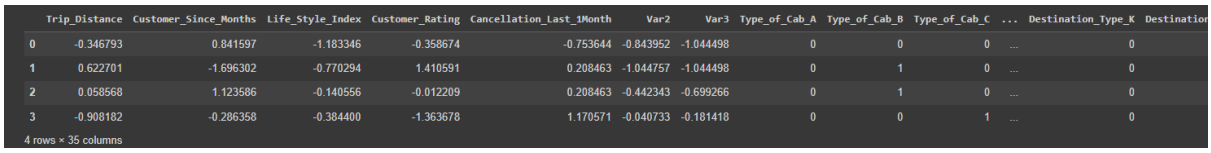
$z = (x - u) / s$

where:
z is the scaled value.
x is the original value of the feature.
μ is the mean of the feature.
s is the standard deviation of the feature.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform.

Choosing Standard Scaling is informed by its ability to normalize feature scales, facilitating model training and improving algorithm performance, particularly for models sensitive to feature magnitude. It assumes a Gaussian distribution, making it suitable for data approximating this pattern.
And standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data.

Here is result of scaling:



| | Trip_Distance | Customer_Since_Months | Life_Style_Index | Customer_Rating | Cancellation_Last_1Month | Var2 | Var3 | Type_of_Cab_A | Type_of_Cab_B | Type_of_Cab_C | ... | Destination_Type_K | Destinatior |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.346793 | 0.841597 | -1.183346 | -0.358674 | -0.753644 | -0.843952 | -1.044498 | 0 | 0 | 0 | ... | 0 | |
| 1 | 0.622701 | -1.696302 | -0.770294 | 1.410591 | 0.208463 | -1.044757 | -1.044498 | 0 | 1 | 0 | ... | 0 | |
| 2 | 0.058568 | 1.123586 | -0.140556 | -0.012209 | 0.208463 | -0.442343 | -0.699266 | 0 | 1 | 0 | ... | 0 | |
| 3 | -0.908182 | -0.286358 | -0.384400 | -1.363678 | 1.170571 | -0.040733 | -0.181418 | 0 | 0 | 1 | ... | 0 | |

4 rows × 35 columns

Picture 32

Then we will convert our final preprocessed data to numpy array for further using since they were dataframes.

# M3: Modeling

An overview of the key skills and techniques in M3:

**Machine Learning**:

- **Modeling and Baselines**: Using a simple baseline model (DummyClassifier) to establish a reference for evaluating more complex models. This approach helps to gauge whether advanced models are truly improving upon simpler ones.
- **Classification Models**:
    - **DummyClassifier**: Used as a baseline for comparison with more sophisticated models.
    - **Logistic Regression**: Applying One-vs-Rest (OvR) strategy for multi-class classification.
    - **Decision Trees**: Using decision rules to classify data based on feature splits.
    - **K-Nearest Neighbors (KNN)**: An instance-based learning algorithm where classification decisions are made based on the proximity of data points.
- **Model Evaluation**: Using techniques like confusion matrices to visualize and understand misclassifications, and classification reports to evaluate the overall performance of models.
- **Cross-validation**: Ensuring the models' robustness by evaluating their performance across multiple splits of the dataset to prevent overfitting.

**Hyperparameter Tuning**:

- **Grid Search**: Using `GridSearchCV` to perform an exhaustive search over a specified hyperparameter grid. This is crucial for optimizing the performance of models by fine-tuning hyperparameters such as regularization strength (`C`), tree depth, and number of neighbors (KNN).
- **Regularization**: Adjusting parameters like the penalty (`l2` for logistic regression) and maximum iterations to control model complexity and prevent overfitting.

**Performance Metrics**:

- **Macro and Weighted Averages**: Understanding the impact of class imbalances on evaluation scores and using macro and weighted averages to provide a more comprehensive assessment.
- **F1-Score, Precision, and Recall**: Balancing the trade-offs between precision and recall, particularly in imbalanced datasets.
- **Confusion Matrix**: Using confusion matrices to identify misclassifications and gain insights into model behavior across different classes.

**Optimization and Model Comparison**:

- **Model Comparison**: Training and comparing multiple models (e.g., One-vs-Rest Logistic Regression, Decision Trees, KNN) with optimized hyperparameters to choose the best-performing model.
- **Fitting Time Analysis**: Measuring the time taken to train models to understand computational efficiency, with KNN being the fastest in this case.

A baseline is a simple model or heuristic that provides a basis for comparison with more complex models.Baselines serve as benchmarks for comparing the performance of more

complex models, ensuring that improvements are meaningful. They provide a starting point for model development, help in debugging and validating the ML pipeline, aid in resource allocation, and guide incremental improvements. Essentially, baselines establish a minimum level of performance that needs to be surpassed by advanced models, ensuring that efforts are focused on solutions that provide tangible benefits.
For baseline I will use Dummy classifier with “most frequent strategy”(documentation https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html).
Time taken for fitting baseline model: 0.006128549575805664 seconds
Next we need to see metrics.
A metric is a measure used to evaluate the performance of a model on a dataset. Metrics provide quantitative assessments of how well a model is performing in terms of various aspects such as accuracy, precision, recall, F1 score, etc. These metrics help us understand how effective the model is in solving the given task and guide us in making decisions regarding model selection, tuning, and improvement.As regards for metrics in the project, we will use classification_report(documentation
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)
which includes the next metrics:

-Precision: Precision measures the accuracy of positive predictions. It is the ratio of true positive predictions to the total number of positive predictions made by the model. Precision indicates the proportion of correctly predicted positive cases among all predicted positive cases. Mathematically, it is defined as(pic.33):

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Picture 33

-Recall (Sensitivity): Recall measures the ability of the model to correctly identify positive instances from the total actual positive instances. It is the ratio of true positive predictions to the total number of actual positive instances in the data. Recall is also known as sensitivity or true positive rate (TPR). Mathematically, it is defined as(pic.34):

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Picture 34

-F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is particularly useful when the class distribution is imbalanced. F1-score is the harmonic mean of precision and recall, and it is calculated as(pic.35):

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Picture 35

-Support: Support is the number of actual occurrences of each class in the specified dataset. It represents the number of true instances for each class.

-Accuracy: accuracy is a commonly used metric to evaluate classification models. It measures the proportion of correctly predicted instances (both true positives and true negatives) among all instances.
And:
Macro Average:
The macro average calculates the unweighted mean of metrics (such as precision, recall, and F1-score) across all classes.
It treats each class equally, regardless of class imbalance, by computing the average without considering class support.

Weighted Average:
The weighted average calculates the mean of metrics (such as precision, recall, and
F1-score) across all classes, taking into account class imbalance. It assigns weights to each class based on their support (the number of instances) and computes a weighted mean, where larger classes contribute more to the average.
Here is the result of classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.00 | 0.00 | 0.00 | 5443 |
| 2 | 0.43 | 1.00 | 0.60 | 11346 |
| 3 | 0.00 | 0.00 | 0.00 | 9544 |
| accuracy | | | 0.43 | 26333 |
| macro avg | 0.14 | 0.33 | 0.20 | 26333 |
| weighted avg | 0.19 | 0.43 | 0.26 | 26333 |

Picture 36

So, from pic 36 is visible that:
It is clear that the classifier only makes predictions for class 2 and none for classes 1 and 3, hence the precision for classes 1 and 3 is 0. The precision for class 2 is 0.43, meaning that when it predicts class 2, it is correct 43% of the time.
The classifier has a recall of 1.00 for class 2, indicating that it correctly identifies all class 2 instances. However, the recall for classes 1 and 3 is 0, meaning it never correctly identifies instances of these classes.
The F1 score for class 2 is 0.60, which is somewhat moderate, but for classes 1 and 3, it's 0 because the precision and recall are both 0 for these classes.
Support: The support is the number of actual occurrences of the class in the dataset. Class 1 has 5443 instances, class 2 has 11346 instances, and class 3 has 9544 instances.

Accuracy: The accuracy of the classifier is 0.43, meaning that overall, it correctly predicts the class 43% of the time across all predictions.

Macro Avg: This is the average precision, recall, and F1 score between classes. The macro average does not take label imbalance into account, so despite the good performance on class 2, the poor performance on classes 1 and 3 brings down the macro average significantly (precision: 0.14, recall: 0.33, F1-score: 0.20).

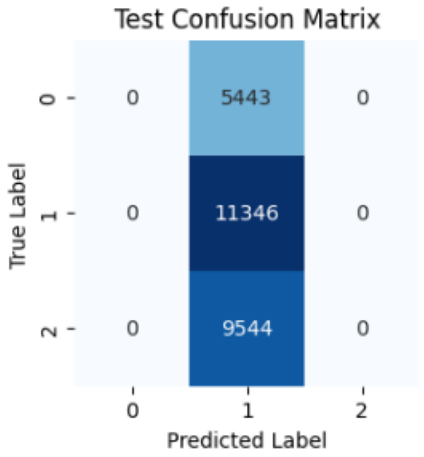
Weighted Avg: This is like the macro average, but takes label imbalance into account. The weighted average precision, recall, and F1 score are calculated by weighting the score of each class by the number of true instances in each class. The weighted averages are higher than the macro averages (precision: 0.19, recall: 0.43, F1-score: 0.26) because class 2 has more instances and pulls the average upwards.

Next, we will see the confusion matrix.
A confusion matrix is a table used to evaluate the performance of a classification model. It allows us to visualize the performance of a model by comparing predicted labels with actual labels across different classes.
Confusion matrices help identify patterns of misclassifications, understand which classes are difficult to predict, and assess the trade-offs between different evaluation metrics. They are essential tools for evaluating and fine-tuning classification models, providing valuable insights for model improvement.

Diagonal Elements (True Positives and True Negatives): These elements represent correct predictions. The higher the values along the diagonal, the better the model's performance.

Off-Diagonal Elements (False Positives and False Negatives): Off-diagonal elements represent incorrect predictions. False positives occur when the model predicts a class incorrectly, and false negatives occur when the model fails to predict a class correctly.

Here is confusion matrix:




Picture 37

From pic.37 is visible, that indeed  class 2 has been predicted for every instance.There are no instances where the true label of class 0 (row 0) or class 2 (row 2) were correctly predicted, since the corresponding diagonal cells are 0.This confusion matrix validates that the model is indeed a poor classifier as it is not predicting classes 0 or 2 at all, resulting in a high recall for class 1 but zero precision for classes 0 and 2. Essentially, this model is acting as if it is guessing or defaulting to a single class for all predictions, which is a clear indication that it is not learning from the features provided.

Next, we will train 3 machine learning algorithms. In particular, OneVsRestClassifier with LogisticRegression,  DecisionTreeClassifier, KNeighborsClassifier.

*OneVsRestClassifier with LogisticRegression*(https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html documentation)
At first, we will know what is logistic regression.
*Logistic Regression* is a statistical method for predicting binary outcomes. It's used in this context to estimate the probability that a given instance belongs to a class. Logistic regression can be extended to multi-class classification using OvR, where a separate logistic regression model is trained for each class to predict the probability that a given instance belongs to that class.I will use(OvR)
The One-vs-Rest (OvR) strategy, also known as One-vs-All (OvA), is a method for handling multi-class classification problems. Instead of fitting a single classifier, OvR fits one classifier per class. For each classifier, the class it represents is treated as the positive class, and all other classes are treated as the negative class. During prediction, the classifier with the highest confidence score determines the class label.

*DecisionTreeClassifier*(https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html documentation)
A Decision Tree is a non-parametric supervised learning method used for classification and regression. The model predicts the value of a target variable by learning simple decision rules inferred from the data features. It's called a decision tree because it starts with a single node (the "root"), which branches off into a number of solutions, just like the branches of a tree.
The key idea is to use a "tree" structure, where each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. The paths from root to leaf represent classification rules.

*KNeighborsClassifier*(https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html documentation)
The k-nearest neighbors (KNN) algorithm is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until classification. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.
KNN works by finding the distances between a query and all the examples in the data, selecting the specified number of examples (k) closest to the query, then votes for the most frequent label

(in the case of classification) or averages the labels (in the case of regression).

Next,I will perform hyperparameter-tuning.
Hyperparameter tuning refers to the process of finding the optimal hyperparameters for a machine learning model. Hyperparameters are configuration settings that are not directly learned from the data during training but are set prior to training and control aspects of the learning process.
Hyperparameter tuning helps improving model performance and avoiding overfitting or underfitting.
I will do that using GridSearchCV.
GridSearchCV is a technique for tuning hyperparameters in machine learning models, which is part of the Scikit-learn Python library. The "CV" in GridSearchCV stands for cross-validation, which is a method used to evaluate the performance of a model in a robust way. Here's a breakdown of its components and benefits:

Purpose of GridSearchCV:
Hyperparameter Tuning:

Hyperparameters are the configuration settings used to structure a machine learning model. They are not learned from the data but set prior to the training process and have a significant impact on the performance of a model.
GridSearchCV systematically works through multiple combinations of hyperparameter values, cross-validating as it goes to determine which set gives the best performance.
Cross-Validation:

During grid search, for each combination of hyperparameters, the model's performance is evaluated using cross-validation.
Cross-validation involves splitting the training data into several subsets, training the model on some of these subsets (training folds)(k-1 folds), and validating it on the remaining subsets (validation folds). This process repeats several times, each time with a different set of validation and training folds.

In GridSearchCV, the main hyperparameters that I specify are:

estimator: This parameter specifies the machine learning model that you want to tune.

param_grid: This parameter is a dictionary or a list of dictionaries where each dictionary contains hyperparameters and the corresponding values to be explored. Each key in the dictionary represents a hyperparameter, and the corresponding value is a list of values to try for that hyperparameter.

cv: This parameter specifies the cross-validation strategy to use. It can be an integer (to specify the number of folds in a K-fold cross-validation), a cross-validation splitter object, or an iterable yielding train/validation splits.

n_jobs: This parameter specifies the number of jobs to run in parallel during cross-validation. Setting it to -1 uses all available processors.

verbose: This parameter controls the verbosity of the output during the grid search. Higher values provide more verbose output.

Here are my hyperparameter values for OneVsRestClassifier with LogisticRegression:

```
# Hyperparameter values for OneVsRestClassifier with LogisticRegression
param_dist_lr = {
    'estimator__C': [0.001, 0.01, 0.1, 1, 10],  # Extending range of regularization strengths
    'estimator__penalty': ['l2'],
    'estimator__max_iter': [100, 200, 400]  # Number of iterations
}
```

Picture 38

From pic. 38:
estimator__C: This hyperparameter controls the inverse of the regularization strength. Regularization is a technique used to prevent overfitting by penalizing large coefficients. A smaller value of C means a stronger regularization, where the logistic regression model is discouraged from fitting the training data too closely. Larger values of C allow the coefficients to grow larger to fit the training data as well as possible.

estimator__penalty: This specifies the norm used in the penalization (regularization). The 'l2' penalty is the standard used in logistic regression and penalizes the square magnitude of the coefficients. The 'none' option means that no regularization is applied.

estimator__max_iter: The maximum number of iterations taken for the optimization algorithm to converge to a solution. If the algorithm hasn't converged after this many iterations, it will stop, possibly before finding the optimal coefficients.

Here are my hyperparameter values for DecisionTreeClassifier:

```
param_dist_dt = {
    'max_depth': [5, 10, 20, None],  # Added more granularity
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy'],  # Splitting criteria
    'min_impurity_decrease': [0.0, 0.01, 0.1]  # Minimum impurity decrease required for split
}
```

Picture 39

From pic. 39:
max_depth: The maximum depth of the tree. A tree with a greater depth can model more complex relationships by making more splits, but it can also lead to overfitting. A None value means that the tree will continue to expand until all leaves are pure or until all leaves contain less than the min_samples_split samples.

min_samples_split: The minimum number of samples required to split an internal node. Higher values prevent creating nodes that represent too few instances and can lead to a more generalized tree, but if this value is too high, the tree may be underfitted.

min_samples_leaf: The minimum number of samples required to be at a leaf node. This parameter smooths the model, especially in regression.

criterion: The function used to evaluate the quality of a split. 'gini' refers to Gini impurity, which is a measure of how often a randomly chosen element from the set would be incorrectly labeled(as a criterion, it aims to minimize the probability of misclassification.
). 'entropy' is a measure of information gain that is used for the same purpose.

min_impurity_decrease: A node will split if this split induces a decrease of the impurity greater than this value. Essentially, this parameter is used as a threshold to control the growth of the tree and to prevent overfitting. Splitting only continues if it significantly decreases impurity.
This hyperparameter is used to control the growth of the tree and prevent overfitting. When considering a split at a node, the decision tree algorithm will only choose to make the split if it decreases the overall impurity of the classes more than the min_impurity_decrease threshold.Impurity refers to the degree of disorganization or disorder in the data. A group of data points belonging to the same class is considered pure.
Impurity Decrease is the amount by which a proposed split would reduce that measure of impurity compared to the parent node.
.

Here are my hyperparameter values forKNeighborsClassifier:

```
param_dist_knn = {
    'n_neighbors': [1, 5, 10, 20, 25],  # Number of neighbors
    'weights': ['uniform', 'distance'],  # Weight function used in prediction
}
```

Picture 40

From pic. 40:
n_neighbors: This is the number of neighbors to use for kneighbors queries. In other words, when deciding the class of a given data point, the algorithm looks at the n closest points and bases its decision on their classes.

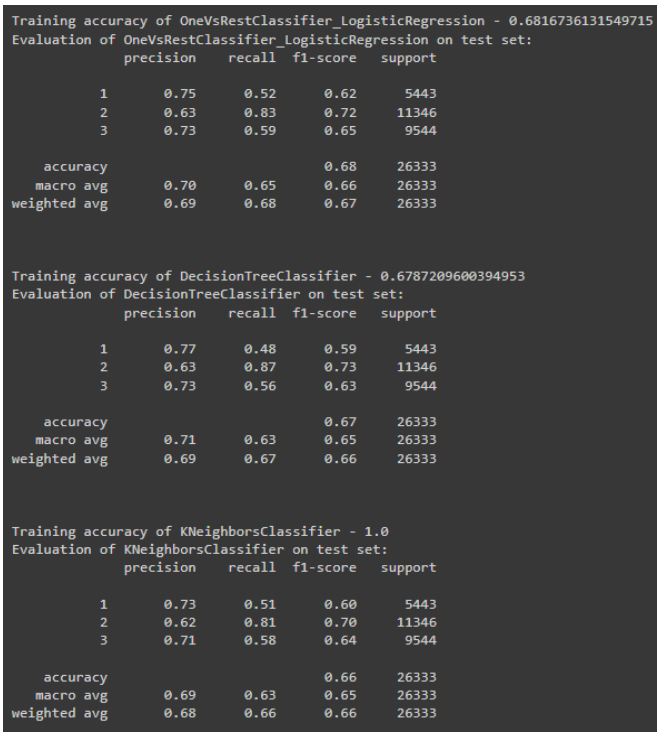
weights: Determines how the prediction is made from the k-neighbors. If uniform, all points are weighted equally. If distance, points are weighted by the inverse of their distance, so closer neighbors of a query point will have a greater influence than neighbors that are further away.

So, as a result, I will have these best parameters for OneVsRestClassifierLogisticRegression:
{'estimator__C': 0.001, 'estimator__max_iter': 100, 'estimator__penalty': 'l2'}

So, as a result, we will have these best parameters for DecisionTreeClassifier: {'criterion': 'entropy', 'max_depth': 5, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2}
So, as a result, we will have these best parameters for KNeighborsClassifier: {'n_neighbors': 25, 'weights': 'distance'}

Here is the results of using these methods with best parameters:
Fitting time for OneVsRestClassifier_LogisticRegression with best parameters: 1.0284671783447266 seconds
Fitting time for DecisionTreeClassifier with best parameters:0.631331205368042 seconds
Fitting time for KNeighborsClassifier with best parameters: 0.024256229400634766 seconds

Although all machine learning algorithms executions with the best parameters were quite quick, the fastest one is KNN.



```
Training accuracy of OneVsRestClassifier_LogisticRegression - 0.6816736131549715
Evaluation of OneVsRestClassifier_LogisticRegression on test set:
              precision    recall  f1-score   support

           1       0.75      0.52      0.62      5443
           2       0.63      0.83      0.72     11346
           3       0.73      0.59      0.65      9544

    accuracy                           0.68     26333
   macro avg       0.70      0.65      0.66     26333
weighted avg       0.69      0.68      0.67     26333



Training accuracy of DecisionTreeClassifier - 0.6787209600394953
Evaluation of DecisionTreeClassifier on test set:
              precision    recall  f1-score   support

           1       0.77      0.48      0.59      5443
           2       0.63      0.87      0.73     11346
           3       0.73      0.56      0.63      9544

    accuracy                           0.67     26333
   macro avg       0.71      0.63      0.65     26333
weighted avg       0.69      0.67      0.66     26333



Training accuracy of KNeighborsClassifier - 1.0
Evaluation of KNeighborsClassifier on test set:
              precision    recall  f1-score   support

           1       0.73      0.51      0.60      5443
           2       0.62      0.81      0.70     11346
           3       0.71      0.58      0.64      9544

    accuracy                           0.66     26333
   macro avg       0.69      0.63      0.65     26333
weighted avg       0.68      0.66      0.66     26333
```

Picture 41

OneVsRestClassifier with LogisticRegression training accuracy is relatively close to the test accuracy of 0.68, indicating that the model generalizes reasonably well to unseen data and there is no significant overfitting.

DecisionTreeClassifier training accuracy is quite close to the training accuracy. This suggests that the Decision Tree also generalizes fairly well, though the performance of the model is moderate.

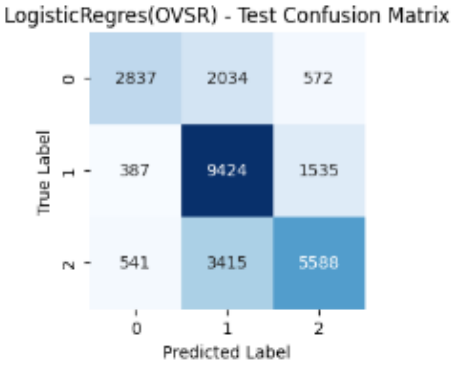
KNeighborsClassifiertraining accuracy is 1.0 (100%) but the test accuracy is 0.67 (67%), this disparity suggests that the KNN model is overfitting the training data.

Further analysis from pic.41:

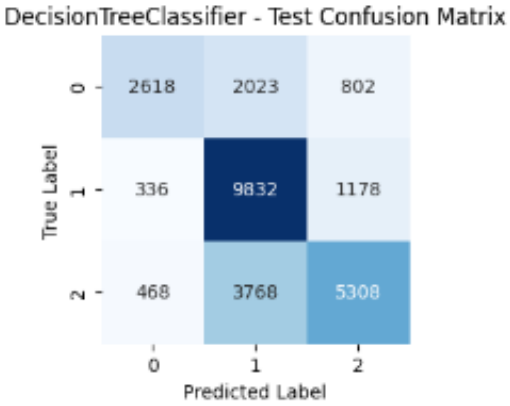
Class 2 is the most frequently correctly predicted class across all classifiers, likely because it has the most support (instances in the test set).

Class 1 and 3 are more challenging for the models, with varying degrees of precision and recall across the classifiers.

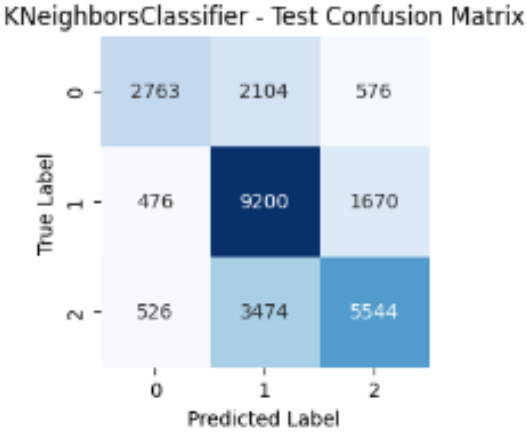
The OneVsRestClassifier with LogisticRegression appears to be the most balanced in terms of precision and recall across classes and has the highest overall accuracy, but the difference is not so big though.




Picture 42




Picture 43




Picture 44

From confusion matrices we can say that(pic. 42-44):

While Logistic Regression(OvR) is the best at correctly identifying Class 1,the KNN classifier appears to be the best at correctly identifying Class 3 instances and the Decision tree classifier is the best at correctly identifying Class 2.

Misclassifications between Classes 1 and 2 are common across all models, with KNN having the most misclassifications in that area, indicating potential overlaps or similarities between features of these classes that the models are finding difficult to separate.

While,roughly speaking, the confusion matrices for the Logistic Regression (OneVsRest), Decision Tree Classifier, and K-Nearest Neighbors Classifier are similar to one another, Logistic Regression(OvR) shows a more balanced performance and results from picture 41 proves it..

We can see that all 3 models are better that the baseline models if we will look at confusion matrices(picture 37,42-44) and overall performances.(picture 36 and 41).Especially we are interested in comparison of our best model with the baseline:

Now we will compare our best OneVsRestClassifier with LogisticRegression mode with the baseline model.(pic. 36 and 41).

Although executions time of these models were quite small, sure that the baseline model is much faster.

Time taken for fitting baseline model: 0.006128549575805664 seconds

Fitting time for OneVsRestClassifier_LogisticRegression with best parameters: 1.0284671783447266 seconds

The OneVsRestClassifier with LogisticRegression is significantly better than the baseline model in almost all metrics and it is also visible from confusion matrices pic. 37 and 42.

The best model is able to recognize instances of all classes to varying degrees of precision and recall, whereas the baseline model effectively only recognizes instances of class 2.

The accuracy of the best model is 0.68, which is substantially higher than the baseline's 0.43, indicating a much better overall performance

The weighted and macro averages are much higher for the best model, showing that its performance is more uniform across classes, rather than being skewed towards one class as with the baseline.

So overall the OneVsRestClassifier with LogisticRegression is the best model in the project.

# M4: Improvements

An overview of the key skills and techniques in M4:

**Machine Learning Techniques Development:**

- **Voting Classifier**: Applying the **VotingClassifier** from `sklearn.ensemble`, combining multiple classifiers (Logistic Regression, Decision Tree, and KNN) to improve accuracy and robustness. This involves both **hard** and **soft voting** strategies.
- **Hyperparameter Tuning**: Using **RandomizedSearchCV** for efficient hyperparameter search, which optimizes model performance by sampling from specified hyperparameter distributions.
- **Logistic Regression (One-vs-Rest)**: This model is used with **One-vs-Rest** strategy for multi-class classification.
- **Decision Tree Classifier**: A model that splits data into subsets based on feature values to classify instances.
- **K-Nearest Neighbors (KNN)**: A classification algorithm that predicts the class of a data point based on the majority vote of its neighbors.

**Neural Network Development:**

- **Deep Learning Architecture**: Designing a **neural network** using the **Sequential model** in TensorFlow/Keras, which consists of an input layer, hidden layers, and an output layer.
- **Batch Normalization**: Used to stabilize training by normalizing layer inputs and helping the model generalize better.
- **Activation Functions**:
    - **ReLU** for hidden layers to introduce non-linearity and improve training efficiency.
    - **Softmax** for the output layer to convert output into a probability distribution.
- **Optimizer and Loss Function**:
    - **SGD** (Stochastic Gradient Descent) and **Adam** for optimization, controlling the step size during training.
    - **Categorical Cross-Entropy** as the loss function for multi-class classification.
- **Performance Metrics**: Tracking both **accuracy** (proportion of correct predictions) and **loss** (discrepancy between predicted and actual values) during training and evaluation.

**Model Evaluation and Performance Metrics:**

- **Training, Validation, and Test Accuracy**: Assessing model performance across multiple datasets, noting that accuracy slightly improves through the different stages of model refinement.
- **Comparison of Models**: Comparing the **VotingClassifier** and **Neural Network** models, observing that the neural network achieves slightly better performance after fine-tuning hyperparameters.

**Hyperparameter Optimization:**

- **Optimizer**: Fine-tuning the optimizer (Adam vs. SGD) and **learning rate** to enhance model performance.
- **Batch Size**: Adjusting batch size to optimize the speed and efficiency of the training process.

**Model Deployment and Iterative Improvement:**

- Iteratively refining models, improving accuracy incrementally with each step. This iterative approach reflects a focus on continuous model enhancement through both algorithmic improvements and hyperparameter tuning.

So, if we round the accuracy of our best model so far to four decimal places, it is 0.6778.
For now we will improve our results  in 2 directions:

1.Using VotingClassifier.
Motivation:
Enhanced Accuracy and Robustness: A Voting Classifier combines the predictions of multiple models, typically leading to higher accuracy and robustness than individual models due to reduced variance and overfitting.
Flexibility in Model Combination: It allows the integration of diverse algorithms, utilizing their complementary strengths which can be crucial for capturing complex patterns within the data.
Practical Application: Ideal for datasets where no single model provides satisfactory performance, a Voting Classifier can leverage strengths across models (e.g., decision trees for non-linear relationships and logistic regression for linear patterns, for example).

2.Building a Neural Network..
Motivation:
Capability with High-Dimensional Data: Neural networks are particularly effective in managing high-dimensional data and excel in complex tasks like image and speech recognition due to their ability to learn feature hierarchies.
Customization and State-of-the-Art Results: They offer extensive customization options in architecture which can be tailored to specific tasks, often achieving state-of-the-art results in many domains.
Adaptation for Complex Features: Their deep learning capabilities make them suitable for tasks requiring the extraction of intricate patterns from large datasets, far surpassing traditional models in performance.

Let's start with VotingClassidier - sklearn.ensemble.VotingClassifier(documentation https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html).
This classifier allows to combine the predictions of multiple individual classifiers (e.g., support vector machines, random forests, logistic regression,decision trees and etc.) and aggregate them to make a final prediction. It supports both hard and soft voting mechanisms, where hard voting takes the majority vote from the ensemble of classifiers, and soft voting considers the average probabilities for each class across all classifiers. This ensemble technique often leads to more robust and accurate predictions compared to individual classifiers, especially when the individual models have different strengths and weaknesses.In our case we will combine 3 machine learning methods from the M3
OneVsRestClassifier with LogisticRegression, DecisionTreeClassifier,KNeighborsClassifier.

We will use RandomizedSearchCV (documentation https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html ) for searching for  the hyperparameter combination that yields the best performance. And we use it since it is more computationally efficient compared to Grid Search CV. n_iter in

this case equals to 50(it  is a parameter that specifies the number of parameter settings that are sampled from the specified parameter distributions)
Here is the parameter grid for RandomSearchCV:

```
params = {
    'lr__estimator__C': [0.01, 0.1, 1, 10],
    'lr__estimator__max_iter': [300, 600, 900],
    'dt__max_depth': [5, 10, 20, None],
    'dt__min_samples_split': [2, 5, 10],
    'dt__min_samples_leaf': [1, 2, 4],
    'dt__criterion': ['gini', 'entropy'],
    'dt__min_impurity_decrease': [0.0, 0.01, 0.1],
    'knn__n_neighbors': [1, 5, 10, 20, 25],
    'knn__weights': ['uniform', 'distance']
}
```

Picture 45

Best Parameters for VotingClassifier:

- Voting Strategy: Soft
- Logistic Regression (OneVsRestClassifier):
    - Maximum Iterations: 600
    - C (Regularization Strength): 0.01
- Decision Tree Classifier:
    - Minimum Samples Split: 10
    - Minimum Samples Leaf: 2
    - Min Impurity Decrease: 0.0
    - Maximum Depth: 10
    - Criterion: gini
- KNeighbors Classifier:
    - Weights: Distance
    - Number of Neighbors: 20

Performance:
Training Score: 0.6839
Test Set Accuracy: 0.6794

Generally, the performance improvement, although modest, is notable compared to previous models in M3.

Building upon this success, we are now advancing to the construction of a neural network.
At first, we will split our data again,by using train_test_split which was introduced before, since we need validation set.It splits the filtered training data (X_train_filtered) and corresponding labels (y_train) into training (X_train_filtered, y_train) and validation (X_val, y_val) sets(validation data is for monitoring generalization). The split is performed using a 5:1 ratio,

where 1/6 of the data is allocated to the validation set. Additionally, it ensures that the class distribution is preserved using stratified sampling.

Let's consider a Neural Network with the following architecture:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                1152

 batch_normalization (Batch  (None, 32)                128
 Normalization)

 activation (Activation)     (None, 32)                0

 dense_1 (Dense)             (None, 16)                528

 batch_normalization_1 (Bat  (None, 16)                64
 chNormalization)

 activation_1 (Activation)   (None, 16)                0

 dense_2 (Dense)             (None, 8)                 136

 batch_normalization_2 (Bat  (None, 8)                 32
 chNormalization)

 activation_2 (Activation)   (None, 8)                 0

 dense_3 (Dense)             (None, 3)                 27

 activation_3 (Activation)   (None, 3)                 0

=================================================================
Total params: 2067 (8.07 KB)
Trainable params: 1955 (7.64 KB)
Non-trainable params: 112 (448.00 Byte)
_________________________________________________________________
```

Picture 46

We use the sequential model which is a linear stack of layers:
The first layer is the input layer.
Hidden layers are the layers between the input and output layers where computations are performed. Each hidden layer consists of multiple neurons, and the number of hidden layers and neurons can vary depending on the complexity of the problem.
The last layer is output layer and should represent the number of classes.

There is Batch Normalization our architecture for the next reasons:
Stabilizes training: Batch normalization normalizes the activations of each layer, reducing internal covariate shift. This stabilizes training, allowing for higher learning rates and faster convergence.
Regularizes the model: Batch normalization acts as a form of regularization by adding noise to hidden layers, reducing overfitting and making the model more robust.
Reduces dependency on initialization: Batch normalization reduces the model's sensitivity to weight initialization, making it easier to train deep networks.

As for activation function, we use ReLU Activation:

Benefits:
Non-linearity: ReLU introduces non-linearity to the network, enabling it to learn complex relationships in data. Its simple thresholding operation accelerates training by avoiding the vanishing gradient problem.
Sparsity: ReLU sets negative values to zero, introducing sparsity and encouraging sparse representations. This can improve computational efficiency and generalization.
Efficiency: ReLU is computationally efficient compared to other activation functions like sigmoid or tanh, leading to faster training times.

One worthy highlight is the `softmax` activation function in the last layer, which can be used to convert any vector with k elements to a probability distribution of k possible outcomes

Before training a model, we need to configure the learning process, which is done via the compile method

```
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(learning_rate=0.05),
              metrics=['accuracy'])
```

Picture 47

It has three arguments:
loss function: the objective that the model will try to minimize, it can be the string identifier of an existing loss function or a function.Here we used the categorical_cross entropy, because the task is a multi class classification.
optimizer: this could be the string identifier of an existing optimizer or an instance of the Optimizer class.In our case SGD optimizer with learning rate 0.05.(The learning rate is a hyperparameter that controls the size of the step taken during the optimization process)
list of metrics: a metric could be the string identifier of an existing metric or a custom metric function.In our case it is an accuracy.

The model expects the labels in one-hot encoded format, that is why we will encode our y_train, y_val, y_test by using the to_categorical function.(documentation https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical)The to_categorical() function is used to convert a numerical label vector to a binary class matrix.

The number of epochs=30(the process of processing all training samples (ie. all batches) once) and batch_size=256(the model weights are tuned based on subsets of examples also called batches in one training step) during training in our NN.

As regards for  results, in NN we will consider 2 metrics: loss and accuracy. Accuracy is the proportion of correctly classified samples out of the total number of samples in the dataset(we want to maximize it. Loss measures the discrepancy between the predicted class probabilities and the true class labels for a given set of input samples.(we want to minimize it).The loss function quantifies how well the model's predictions align with the actual targets during training.

Here are the results:
Training accuracy: 0.6876
Training loss: 0.7342
Validation accuracy: 0.6887
Validation loss: 0.7335
Test accuracy: 0.6836
Test loss: 0.7462

The results indicate that the neural network is performing fairly consistently across training, validation, and test datasets.Notably, its performance surpasses that of the voting classifier.

Now, we will search for the optimal hyperparameter combinations in our neural network. We'll fine-tune hyperparameters such as the optimizer, learning rate, and batch size to enhance performance.

Here is the hyperparameter list:

```
params = {
    'optimizer': ['sgd', 'adam', 'adagrad'],
    'learning_rate': [0.01, 0.05, 0.1],
    'batch_size': [128, 256, 512]
}
```

Picture 48

Ultimately, we identified the optimal hyperparameters as follows:

- Optimizer: Adam
- Learning Rate: 0.05
- Batch Size: 128

And, as regards for the training and validation accuracy for the best model, the results are:
Training Accuracy: 0.6857
Validation Score: 0.6918

And test:
Test Loss: 0.7456
Test Accuracy: 0.6835

Overall, the VotingClassifier initially showed improved results over previous models in M3, achieving an accuracy of 67.94%. Subsequently, the neural network surpassed all earlier models with a modest increase, reaching an accuracy of 68.32%. After fine-tuning the hyperparameters, we achieved a further slight improvement, bringing the accuracy to 68.35%. This result is the best compared to all previous models. In conclusion, we enhanced our prediction accuracy by nearly 1% compared to the best model in M3. Although the improvement is modest, it signifies a more effective model.