Project Report

Università degli Studi di Messina

Machine Learning [A000799]

18th June, 2025

Baani Singh - 544768

Edanur Karatas - 548829

Yana Holub - 540807

INDEX

D0.0) Passports

Baani Singh (India)

Edanus Karatas (Romania)

Yana Holub (Belarus)

D0.1) Python script

GitHub repository - Link to github repository containing python notebook

Machine Learning Project.ipynb - Link to Google Collab

D1.1) Dataset exploration

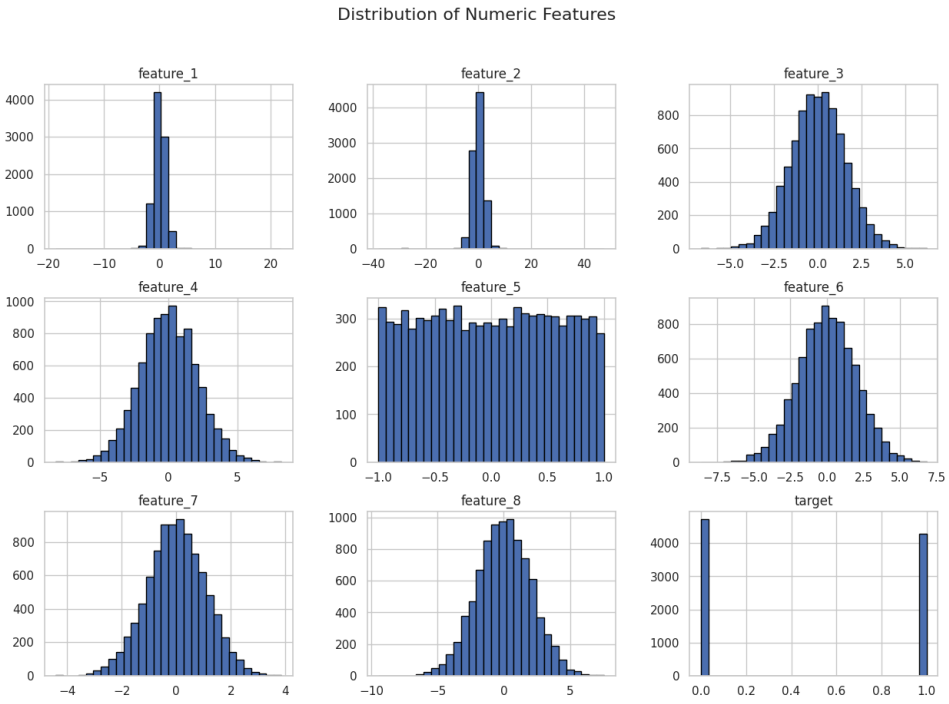
The dataset consists of **9000 entries** (rows) and **11 columns**, representing structured records for a binary classification task.

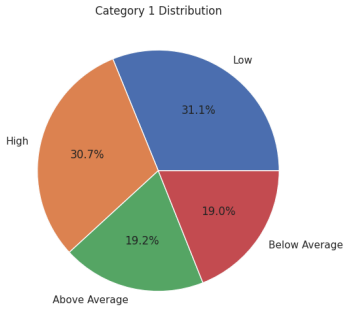
Feature Overview:

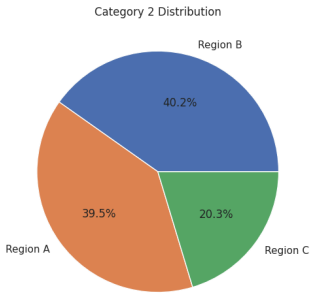
- Numerical Features (8) : Each subplot shows the distribution of one numeric feature. These are frequency histograms, one per feature, here the x-axis shows the actual values that the feature takes in the dataset and the y-axis show the frequency, or count of rows, it shows the actual values that the feature takes in the dataset.
- Categorical Features (2) :
    1. category_1: Ordinal values such as "Low", "Below Average", "Above Average", and "High"
    2. category_2: Nominal values like "Region A", "Region B", and "Region C"
- Target Variable: ‘target’ a binary label indicating class 0 or 1

```
[155] # Display the first few rows
      print(df.head(10))

     feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  \
0    0.496714   1.146509  -0.648521   0.833005   0.784920  -2.209437
1   -0.138264  -0.061846        NaN   0.403768   0.704674  -2.498565
2    0.647689   1.395115  -0.764126   1.708266  -0.250029   1.956259
3    1.523030   2.657560  -2.461653   2.649051   0.882201   3.445638
4   -0.234153  -0.499391   0.576097  -0.441656   0.610601   0.211425
5   -0.234137  -0.699415   0.268972  -0.702775   0.702283  -0.332383
6    1.579213   3.117904  -2.885133   3.312708   0.864708   2.045283
7    0.767435   1.730870  -1.445877   1.411070   0.874003   0.674730
8   -0.469474  -0.877919   0.575087  -0.532917  -0.519870        NaN
9    0.542560   1.314738  -0.403383   1.456165  -0.744625   1.987345

     feature_7  feature_8     category_1 category_2  target
0   -1.300105  -2.242241  Above Average   Region C       1
1   -1.339227  -1.942298  Below Average   Region A       0
2    1.190238   1.503559           High   Region C       1
3    2.120913   3.409035           High   Region B       1
4    0.935759  -0.401463  Below Average   Region C       0
5    0.453958  -0.826721  Below Average   Region A       0
6    1.531547   1.771851           High   Region A       1
7    0.812931   1.489838           High   Region A       1
8   -3.002925  -4.779960  Below Average   Region A       0
9    0.431966   3.309386           High   Region C       1

[156] print(df.shape)

(9000, 11)
```

Distribution of Features: Before any pre-processing







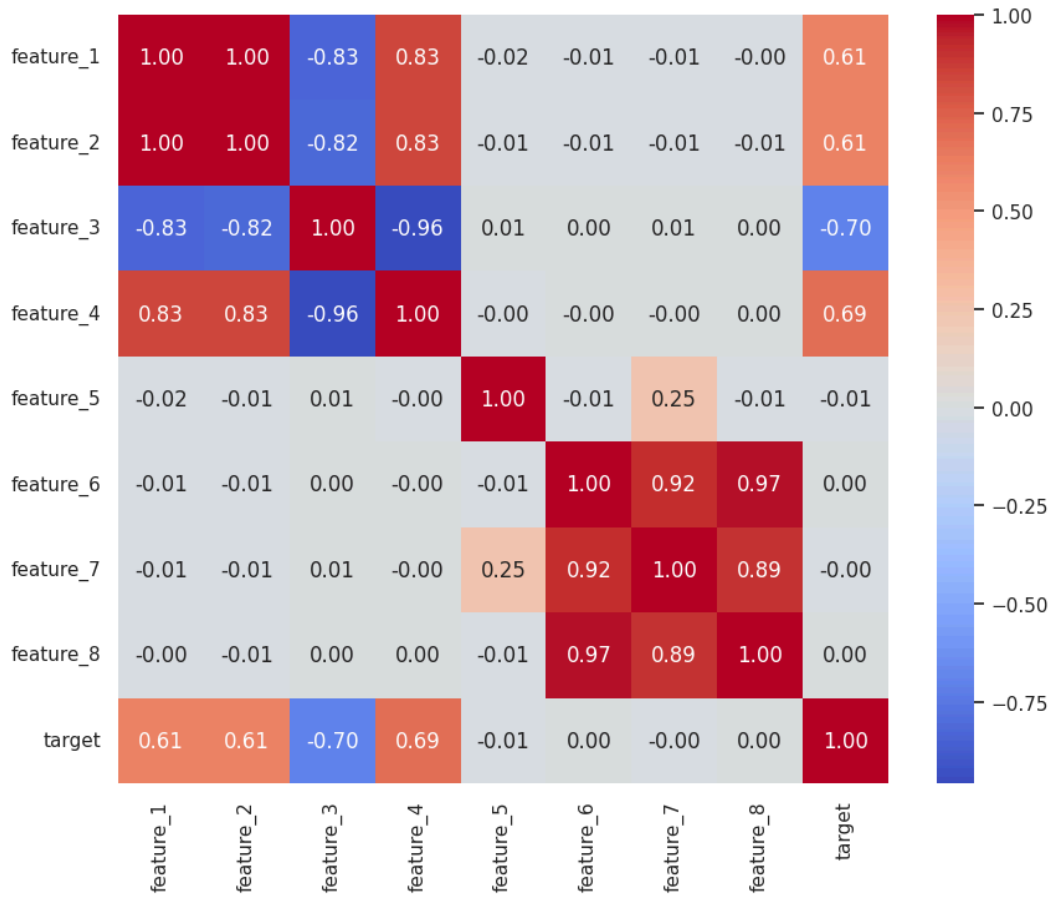
From the visualizations, we make the following observations:

- **feature_1** and **feature_2** show sharp central peaks and long tails, suggesting **high**

- **kurtosis or outliers**. These may need further treatment such as clipping or transformation.
- **feature_3**, **feature_4**, **feature_6**, **feature_7**, and **feature_8** appear to follow **approximately Gaussian (bell-shaped) distributions**, indicating that they are well-suited for models that assume normality (e.g., Logistic Regression).
- **feature_5** stands out as being **uniformly distributed**, suggesting that its predictive signal might be limited when used alone, but potentially useful in interaction terms.
- The **target** **variable** is binary (0 and 1), and its distribution appears reasonably balanced, which is favorable for classification models as it avoids the issue of class imbalance.

These patterns suggest that some features may benefit from normalization or transformation before training, especially for algorithms sensitive to scale or outliers such as Support Vector Machines and Logistic Regression.

Correlation Matrix:



From the visualizations, we make the following observations:

Strong Positive Correlations:

- **feature_1 & feature_2**: r = **+1.00** - These two are **perfectly correlated**, indicating complete redundancy. Only one should be retained to avoid multicollinearity.
- **feature_1, feature_2 & feature_4**: r ≈ **+0.83** - Suggests these features rise and fall together — likely derived from a shared signal or common source.

Strong Negative Correlations:

- **feature_3 & feature_4**: r = **–0.96**
- **feature_3** with **feature_1** and **feature_2**: r ≈ **–0.83**

  These indicate strong opposing trends, useful for ratio or interaction features

Feature Groupings:

- **feature_6**, **feature_7**, and **feature_8** form a highly intercorrelated block (r = **0.89–0.97**)

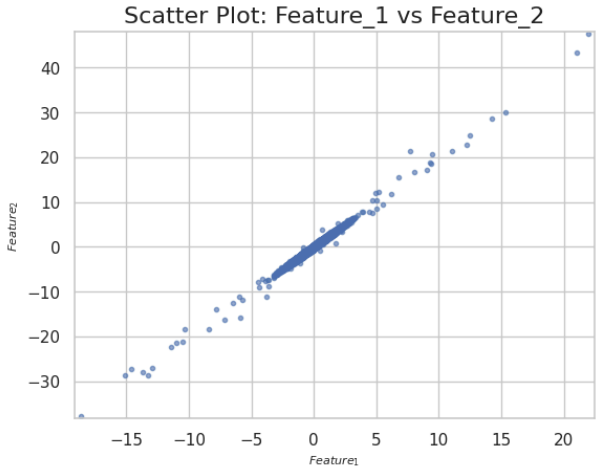
  This suggests they may be redundant or components of the same underlying factor.

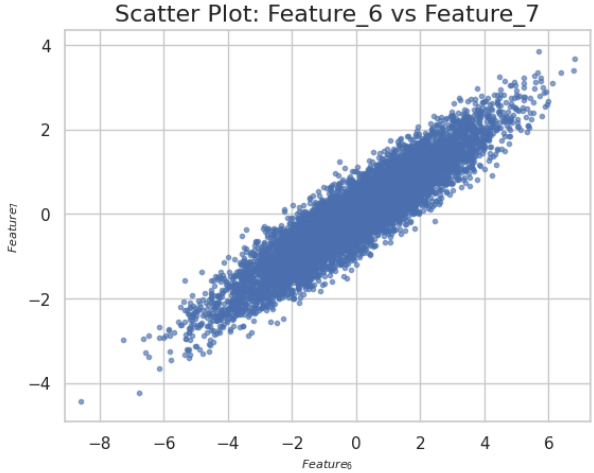
Weak or No Correlation:

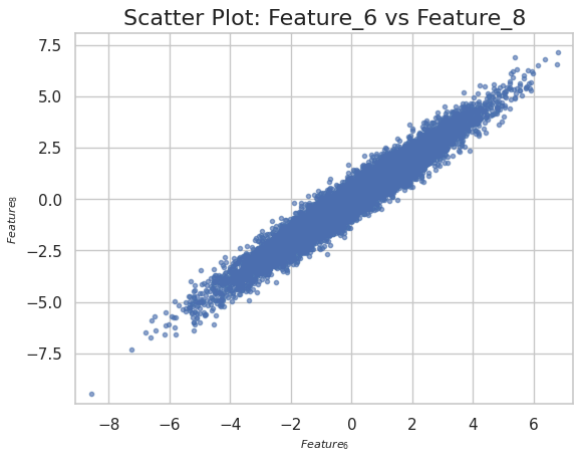
- **feature_5** shows **no meaningful correlation** with any other feature or the target — it may not add much signal.
- **feature_6–8** also show **no direct correlation with the target**, despite being internally consistent.

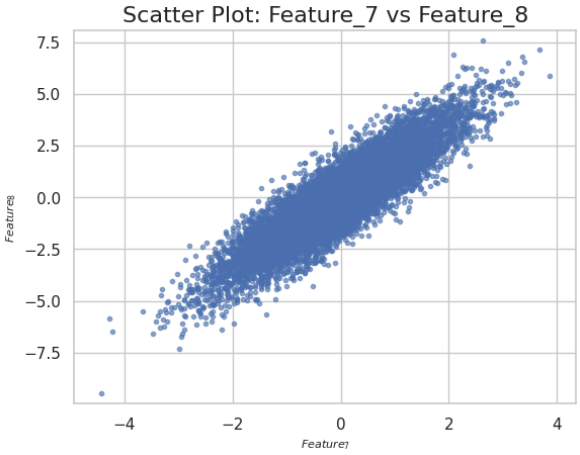
Target Correlation: **Highest correlations with the target are - feature_4: +0.69, feature_1 & feature_2: +0.61, feature_3: –0.70. These features are most predictive in linear terms and should be prioritized**

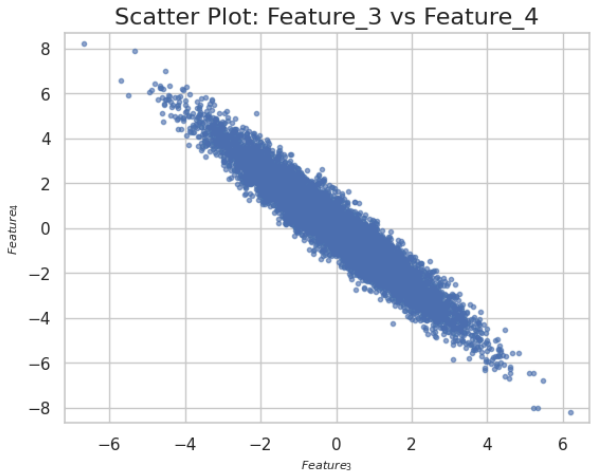
Scatter Plots: We can confirm the observation from the correlation matrix using scatter plots as well.

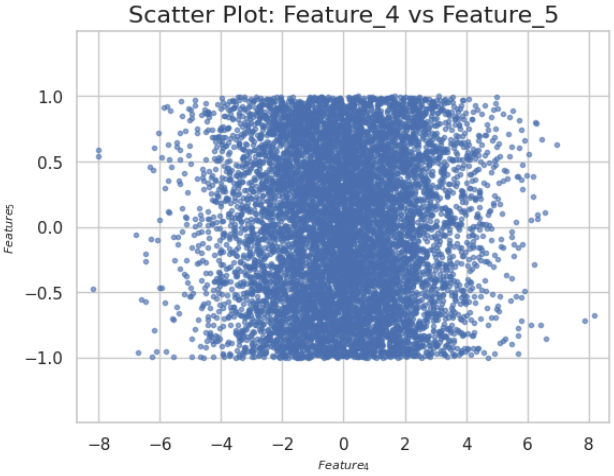


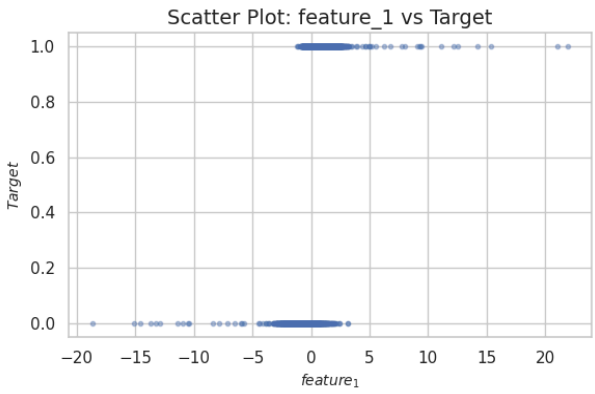


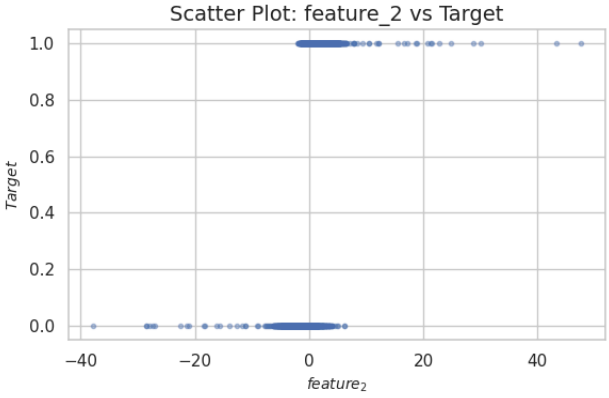

Scatter Plot: Feature_6 vs Feature_8
Feature8
Feature6

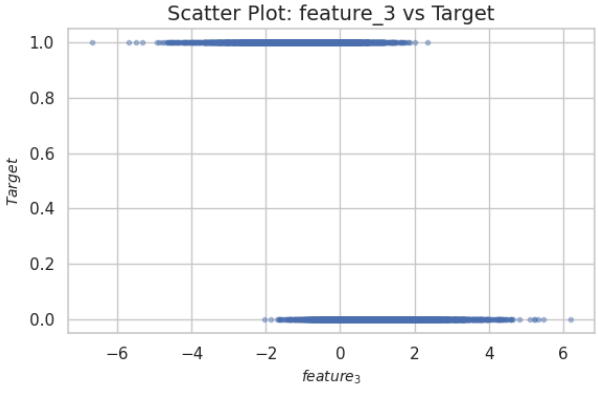

Scatter Plot: Feature_7 vs Feature_8
Feature8
Feature7

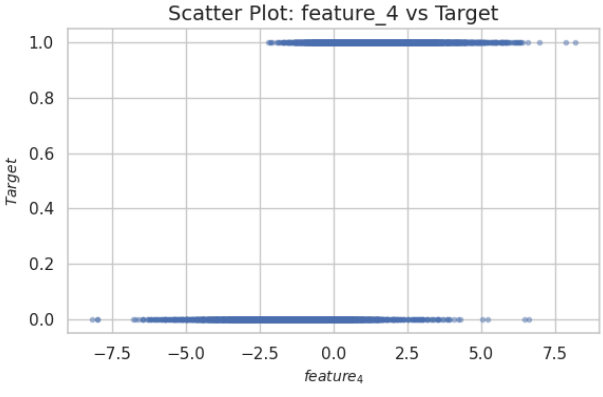

Scatter Plot: Feature_3 vs Feature_4
Feature4
Feature3


Scatter Plot: Feature_4 vs Feature_5
Feature5
Feature4


Scatter Plot: feature_1 vs Target
Target
feature1


Scatter Plot: feature_2 vs Target
Target
feature2


Scatter Plot: feature_3 vs Target
Target
feature3


Scatter Plot: feature_4 vs Target
Target
feature4

From the visualizations, we make the following observations:

Strong Positive Correlation

- **feature_1 vs feature_2** - A perfectly straight diagonal line. This visually confirms that these features are **functionally identical** (r = 1.00)
- **feature_6 vs feature_7 vs feature_8** - Very tight linear clustering. Strong positive correlation (r > 0.89) — they move together and likely represent the same latent structure

Strong Negative Correlation

- **feature_3 vs feature_4** - Very tight **downward-sloping line,** confirms r = –0.96, perfect for interaction terms or ratios

No Correlation / No Pattern

- **feature_5 with any other feature.** Pure scatter noise — evenly spread, no trend. Matches its **uniform distribution** seen earlier

Target Relationships

- **feature_1, feature_2, feature_3, feature_4 vs target,** We can see weak vertical clustering — suggesting **some separation between 0 and 1**
- **Other features vs target** - No pattern visible — suggests low predictive power

D2.1) Data Preprocessing

Mean imputation for missing values: feature_3 and feature_6 had **4.44%** and **5.56%** missing values, respectively. Since both are **numerical** and their distributions are **approximately normal** (not skewed), we applied **mean imputation**. This method was chosen because:

1. It aligns well with normal distributions (median for skewed)
2. The data is numerical (mode used for categorical values)
3. It retains all rows, preserving dataset size
4. The missing percentage is low (less than 6%)

```
[55] missing_count = df.isnull().sum() # Count of missing values
     missing_percent = (missing_count / len(df)) * 100 # Percentage of missing values

     # Combine into one DataFrame
     missing_summary = pd.DataFrame({
         'Missing Values': missing_count,
         'Missing Percentage (%)': missing_percent.round(2)
     })

     missing_summary = missing_summary[missing_summary['Missing Values'] > 0] # Filter only columns with missing values

     print("Missing Value Summary:")
     print(missing_summary)
```

```
Missing Value Summary:
           Missing Values  Missing Percentage (%)
feature_3             400                    4.44
feature_6             500                    5.56
```

```
[46] from sklearn.impute import SimpleImputer

     # Imputation for numerical columns
     num_imp = SimpleImputer(strategy='mean')
     df['feature_3'] = num_imp.fit_transform(df[['feature_3']])
```

```
[47] df['feature_6'] = num_imp.fit_transform(df[['feature_6']])
```

```
[49] print("Missing values:\n", df.isnull().sum())

Missing values:
 feature_1     0
feature_2     0
feature_3     0
feature_4     0
feature_5     0
feature_6     0
feature_7     0
feature_8     0
category_1    0
category_2    0
target        0
```

IQR clipping: To treat extreme outliers in certain numerical features, we applied **IQR-based clipping**. This method is robust and preserves overall data distribution while reducing the influence of extreme values. The Interquartile Range (IQR) method was chosen because it effectively **limits extreme outliers** without removing rows and instead, it detects values that are far outside the normal range (based on interquartile range) and **replaces (clips)** those extreme values to the nearest acceptable boundary.

Identify and Treat Outliers -> IQR Clipping

```
[30] # IQR-based Clipping to Treat Outliers
     numerical_columns = ['feature_1', 'feature_3', 'feature_4',
                          'feature_5', 'feature_6', 'feature_7', 'feature_8']

     for col in numerical_columns:
         Q1 = df[col].quantile(0.25)
         Q3 = df[col].quantile(0.75)
         IQR = Q3 - Q1
         lower = Q1 - 1.5 * IQR
         upper = Q3 + 1.5 * IQR
         df[col] = np.clip(df[col], lower, upper)
```

Encoding: To convert categorical variables into numerical format, we applied appropriate encoding techniques:

- **category_1** was encoded using **label encoding**, where ordinal values ("Low", "Below Average", "Above Average", "High") are mapped to integers 0–3 based on their inherent order. This preserves the **ranked relationship** between categories, which is important for models that may benefit from ordinal structure.
- **category_2** was encoded using **one-hot encoding**, which creates separate binary columns for each category (Region A, B, and C). Since category_2 is **nominal** (no natural order), one-hot encoding ensures that no false ranking is introduced, maintaining the integrity of the data for machine learning models.

Encoding -> One-Hot & Label Enc.

```
[58] # Label encoding for 'category_1'
     mapping = {"Low": 0, 'Below Average': 1, 'Above Average': 2, 'High': 3}
     df['category_1_encoded'] = df['category_1'].map(mapping)
```

```
[59] df.drop(columns=['category_1'], inplace=True)
```

```
[60] # One-hot encoding for 'category_2'
     df = pd.get_dummies(df, columns=['category_2'])
     bool_columns = df.select_dtypes(include='bool').columns
     df[bool_columns] = df[bool_columns].astype(int)
```

```
[63] # Display first 10 rows for category_1_encoded and category_2 one-hot columns
     df[['category_1_encoded', 'category_2_Region A', 'category_2_Region B', 'category_2_Region C']].head(5)
```

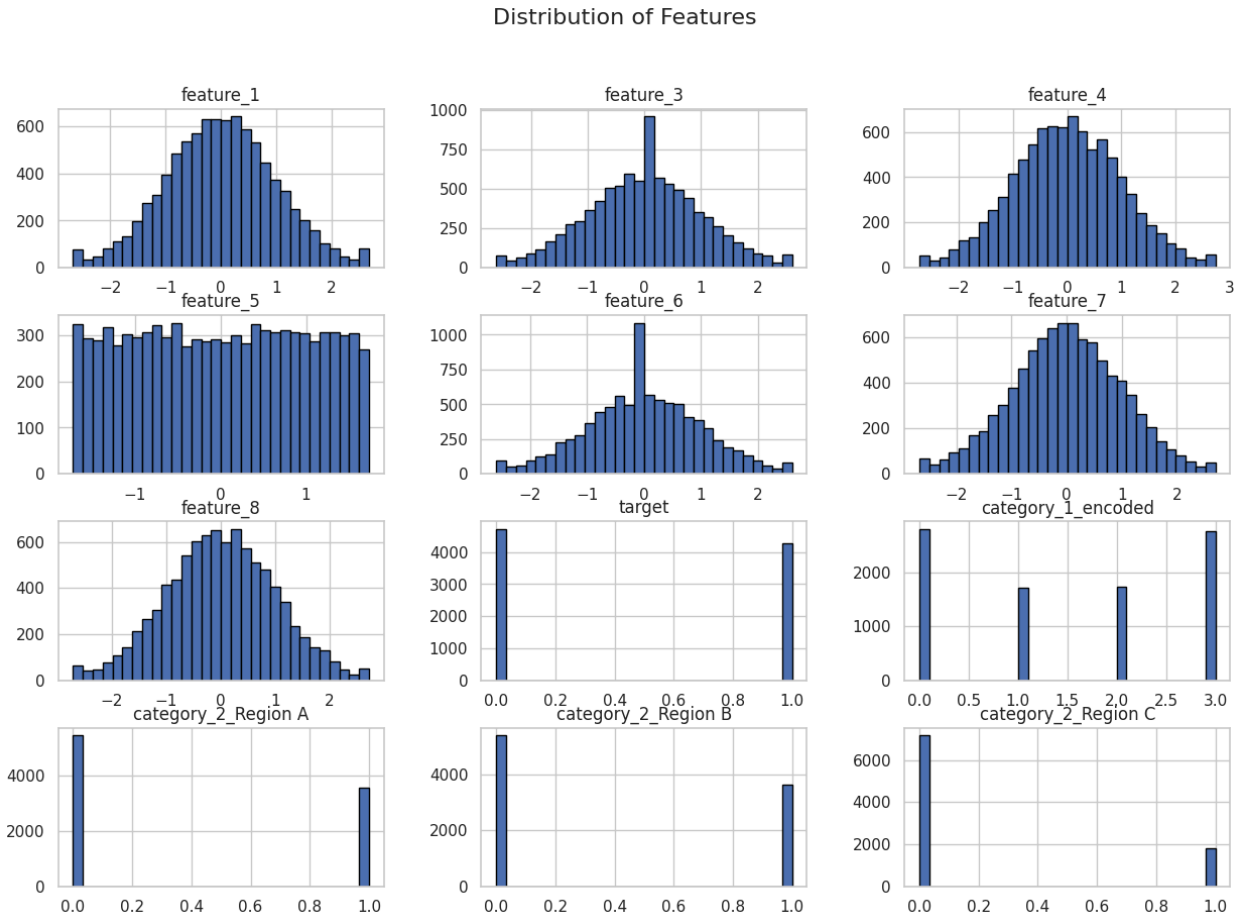
	category_1_encoded	category_2_Region A	category_2_Region B	category_2_Region C
0	2	0	0	1
1	1	1	0	0
2	3	0	0	1
3	3	0	1	0
4	1	0	0	1

Scaling: To ensure that all numerical features contribute equally to distance-based models, we applied **standard scaling** using StandardScaler. This transformation rescales each feature to have a **mean of 0** and a **standard deviation of 1**.Standard scaling was chosen because it makes features **comparable** in magnitude, unit and is **essential for models** that rely on distances (e.g., SVM, Logistic Regression, KNN). This step was applied after handling missing values and clipping outliers, ensuring consistent and balanced input across all numerical features.

Scaling -> StandardScaler

```
[64] from sklearn.preprocessing import StandardScaler

     numerical_columns = ['feature_1','feature_3', 'feature_4',
                          'feature_5', 'feature_6', 'feature_7', 'feature_8']
     scaler = StandardScaler()
     df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

Distribution of Features: After preprocessing, we can observe that compared to the original raw data, the cleaned dataset shows noticeable improvements in distribution. **Numerical features** such as feature_1, feature_3, feature_4, and feature_6 now display **centered, symmetrical bell-shaped curves**, indicating successful scaling. Outlier-heavy like feature_1, feature_6, and feature_3 had **long tails** or scattered extreme values far from the centre. In the **post-processed version**: Those same features now have **shorter, tighter tails** — values are closer to the center now appear more compact, confirming that **IQR clipping** reduced extreme values. feature_3 and feature_6, which had visible gaps earlier, are now complete due to **mean imputation**. Additionally, category_1 is now encoded as integers (0–3), and category_2 is split into clear binary columns (Region A, B, and C), confirming successful categorical encoding.

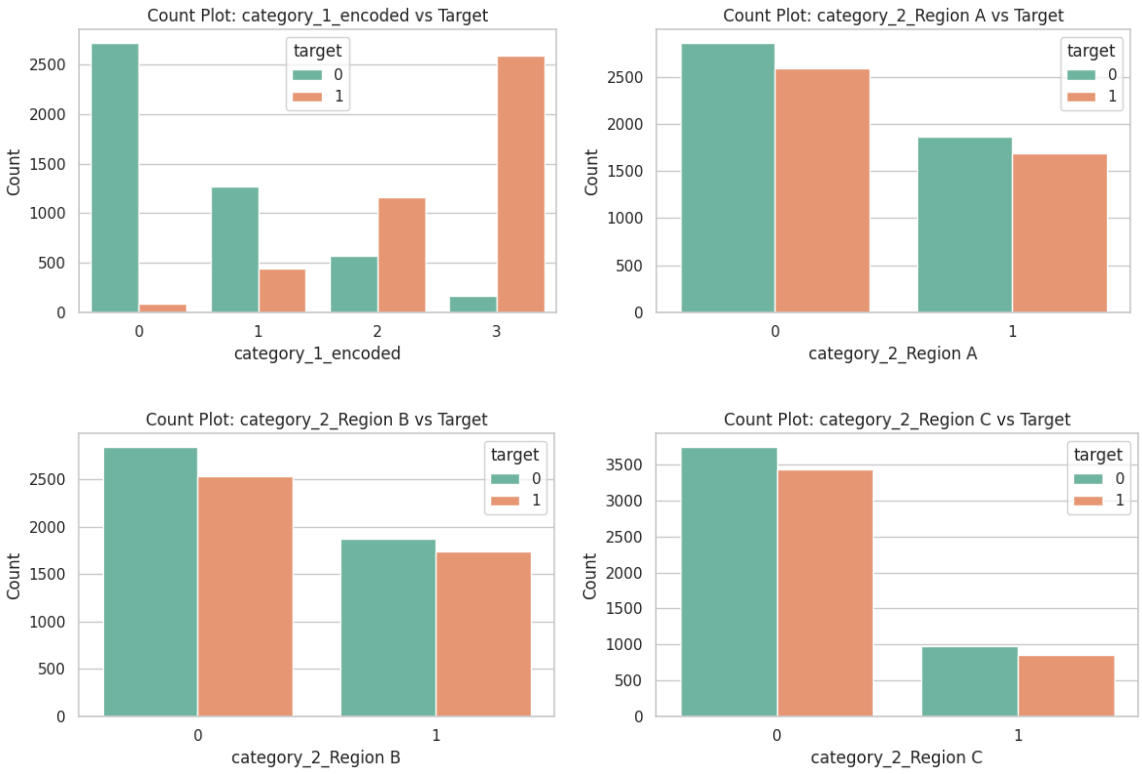


D2.2) Cleaned dataset

GitHub repository - Link to github repository with cleased dataset

D3.1) Exploratory Data Analysis

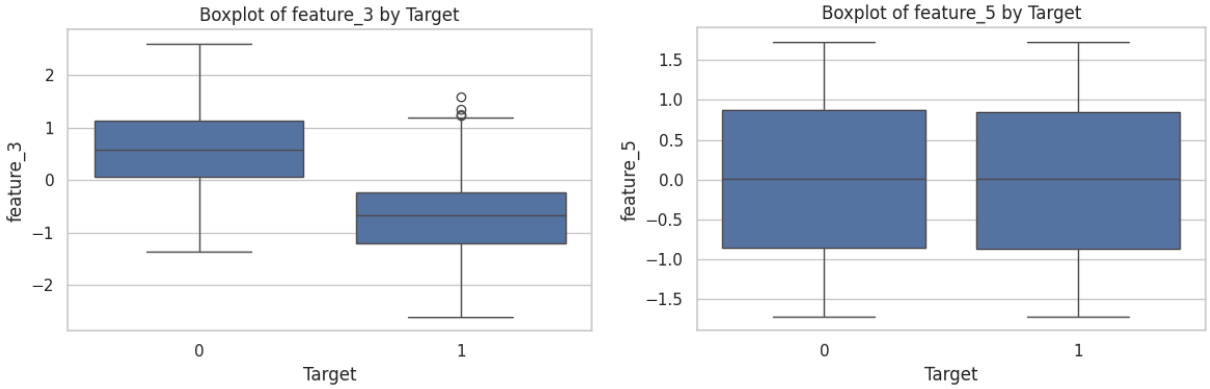
Bar charts(category vs target): We used **bar charts (count plots)** to explore the relationship between **categorical features** and the target variable. Bar charts are ideal for this purpose because they **show the frequency of each category** split by target class (0 or 1), allowing us to easily compare how category values are distributed across outcomes.



Observations:

- **category_1_encoded** shows strong separation: 0 ("Low") is mostly associated with **target = 0,** 3 ("High") is mostly associated with **target = 1,** 1 ("Below Average") and 2 ("Above Average") are more balanced, showing weaker correlation
- **category_2 (Regions A, B, C)** are more evenly distributed across target classes, suggesting **weaker predictive value**, though minor skews are present in all regions.

Boxplot (feature 1-8 vs target): We used **boxplots** to compare the distribution of **numerical features** across the two target classes. Boxplots are ideal for this analysis because they clearly show the **median, spread (IQR), and outliers** of each feature, enabling visual comparison of how the values shift between target = 0 and target = 1.



From the plots we can observe:

- **feature_3** shows strong class separation. The **median for target = 0 is around 0.75**, while for **target = 1 it's below –0.5**. This indicates a clear difference in distribution, making it a **strongly discriminative feature**.
- **feature_5**, however, shows nearly identical boxplots for both classes, suggesting **low predictive value** as the distributions overlap heavily.

These visuals help assess which numerical features contribute meaningfully to target prediction.

T-tests: To statistically evaluate if the numeric features in our dataset significantly differ between the two target classes 0 and 1, we performed an Independent Two-Sample T-Test. This test helps identify which features are most informative for classification, supporting feature selection and model interpretation. We separated the dataset into two groups based on the target variable. Group 1 is instances where target = 0 and Group 2 is Instances where target = 1.

For each numeric feature without the target itself, we performed a t-test using the ttest_ind function from scipy.stats. The test returns two values:

- t-statistic: Measures the magnitude of difference between group means.
- p-value: Indicates the probability that the observed difference occurred by chance.

A feature is considered **statistically significant** if the **p-value is less than 0.05.**

```python
from scipy.stats import ttest_ind, chi2_contingency

# Check unique values in target
print(df['target'].unique())

# Replace 'ActualValue1' and 'ActualValue2' with real categories from target
target_values = df['target'].unique()
group1 = df[df['target'] == target_values[0]]['feature_1']
group2 = df[df['target'] == target_values[1]]['feature_1']

# Perform T-test
target_values = df['target'].unique()
group1 = df[df['target'] == target_values[0]]
group2 = df[df['target'] == target_values[1]]

numeric_features = df.select_dtypes(include='number').columns.drop('target')

print("T-test results:")
for feature in numeric_features:
    t_stat, p_val = ttest_ind(group1[feature], group2[feature], nan_policy='omit')
    print(f"{feature}: t = {t_stat:.3f}, p = {p_val:.5f}")
```

```
[1 0]
T-test results:
feature_1: t = 94.151, p = 0.00000
feature_3: t = -91.269, p = 0.00000
feature_4: t = 92.353, p = 0.00000
feature_5: t = -0.747, p = 0.45520
feature_6: t = 0.175, p = 0.86088
feature_7: t = -0.199, p = 0.84235
feature_8: t = 0.472, p = 0.63686
category_1_encoded: t = 109.299, p = 0.00000
category_2_Region A: t = -0.056, p = 0.95512
category_2_Region B: t = 0.897, p = 0.36965
category_2_Region C: t = -1.024, p = 0.30567
```

Observations:

- **Statistically Significant Features:** feature_1, feature_3, feature_4, and category_1_encoded all exhibit very low p-values that are $p < 0.05$, indicating a **strong and meaningful difference** in their distributions across the target classes. These features are likely valuable for prediction and should be retained for modeling.
- Not Significant Features: feature_5 to feature_8 and the one-hot encoded category_2 variables showed high p-values that are $p > 0.05$, suggesting they do **not differ significantly** between target classes. These features may be less predictive and could be considered for exclusion or further analysis.

Chi-square test: To evaluate whether categorical variables are significantly associated with the target variable, we applied the Chi-Square Test of Independence. The Chi-Square Test is used only for categorical features because it evaluates relationships based on frequency counts in discrete groups, which is not applicable to continuous numerical variables. This statistical test is ideal for determining if there is a relationship between two categorical variables which are the encoded features and the binary target.

```
[137] # Chi-square test for categorical features
      from scipy.stats import chi2_contingency

      categorical_columns = ['category_1_encoded'] + [col for col in df.columns if col.startswith('category_2_')]

      print("Chi-square test results:")
      for col in categorical_columns:
          contingency_table = pd.crosstab(df[col], df['target'])
          chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)
          print(f"{col}: chi2 = {chi2_stat:.3f}, p = {p_val:.5f}")

Chi-square test results:
category_1_encoded: chi2 = 5175.320, p = 0.00000
category_2_Region A: chi2 = 0.001, p = 0.97233
category_2_Region B: chi2 = 0.767, p = 0.38118
category_2_Region C: chi2 = 0.997, p = 0.31816
```

We applied the chi2_contingency function from the scipy.stats library. For each categorical variable, we created a contingency table between the feature and the target variable, and computed the following:

- **Chi-Square statistic**: Measures how expectations compare to actual observed frequencies.
    - χ^2 is small (O ≈ E) → Observed data fits expected data = likely no relationship if
    - χ^2 is large (O ≠ E) → there is a big difference between observed and expected data = indicates a potential relationship.
- **p-value**: Indicates the probability that the observed association happened by chance.:
    - A p-value < 0.05 indicates a **significant association** between the feature and the target.
    - A p-value ≥ 0.05 suggests the feature and target are **likely independent.**

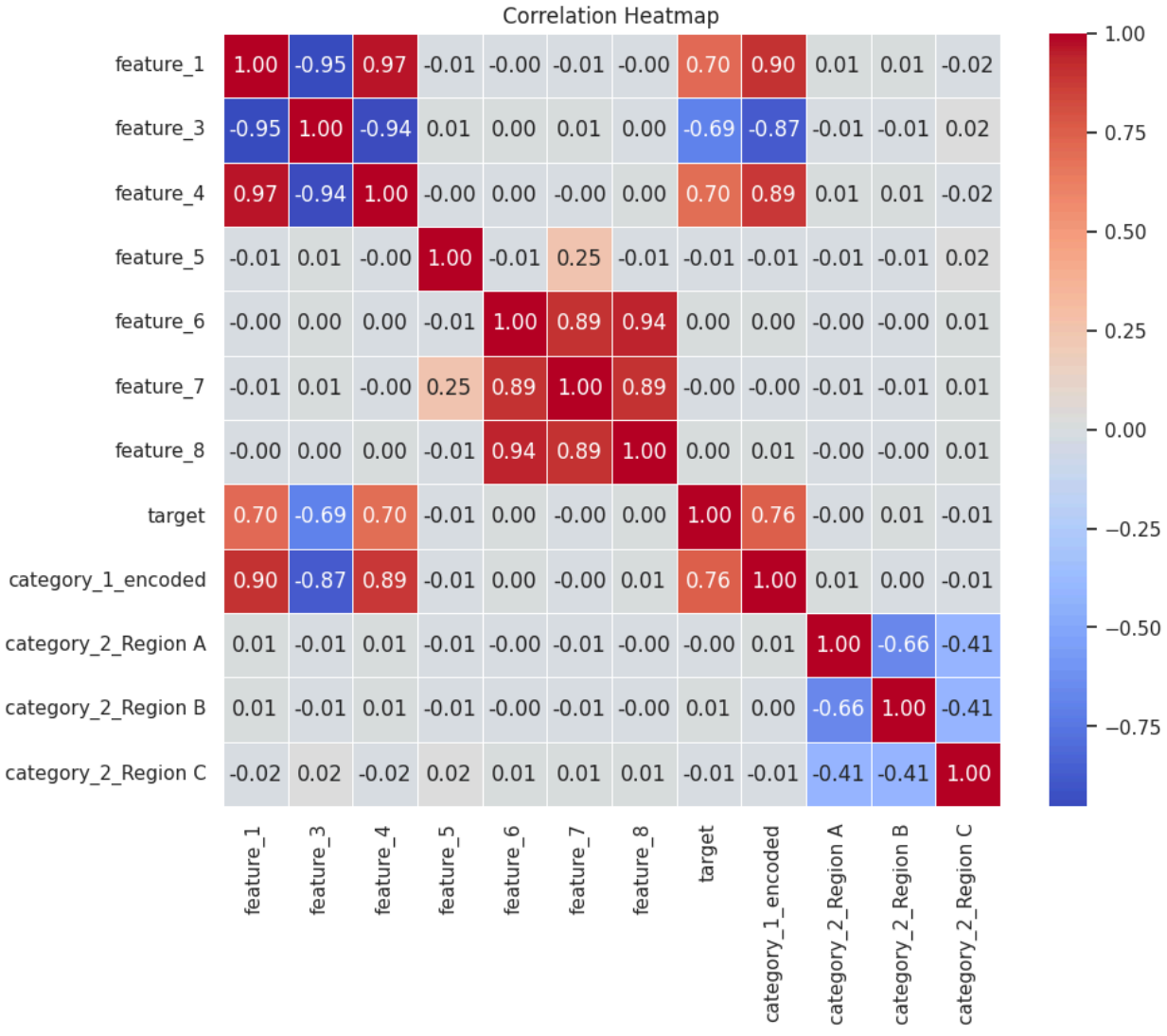
Observations:

- category_1_encoded shows a very **strong statistical relationship** with the target variable. This means that the class distribution of category_1 differs significantly between the two target classes 0 and 1, and therefore , it is very useful for classification.

- category_2_Region A, B and C exhibit p-values well above 0.05, indicating that their distributions do not significantly differ across target classes. Therefore, these variables are **less predictive**.

This test confirms that category_1_encoded is a **strong predictor**, while the category_2 regions are **less informative** for classification. This is consistent with what we observed with the t-test. We can verify both these results with the help of a heatmap as well.

Correlation Heatmap: The heatmap confirms earlier findings from the chi-square and t-tests. **category_1_encoded** shows a **strong positive correlation (r = 0.76)** with the target, reinforcing its predictive strength. In contrast, the **category_2_Region A/B/C** columns display **very weak or near-zero correlation** with the target, supporting the conclusion that **category_2 is less informative**. Additionally, highly correlated numerical features like feature_1, feature_3, and feature_4 also align with our previous observations on their influence in classification.



Group-wise feature averages: Mean values grouped by target reveal that features like feature_1, feature_3, feature_4, and category_1_encoded differ substantially across target classes, supporting their predictive importance. Others like feature_5 show minimal change and are likely less informative

- Feature_1, feature_4 → Strong positive shift from target 0 to 1 → good predictors
- feature_3 Reverses direction (positive to negative) → high class separation
- category_1_encoded Mean jumps from ~0.6 to ~2.5 → strongly aligned with target
- feature_5, feature_6, etc. Minimal change → likely less informative

Mean values grouped by target

```
[36] df.groupby('target').mean(numeric_only=True)
```

target	feature_1	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	category_1_encoded	category_2_Region A	category_2_Region B	category_2_Region C
0	-0.670671	0.660089	-0.664126	0.007495	-0.001759	0.001996	-0.004738	0.617878	0.394832	0.397585	0.207583
1	0.739948	-0.728273	0.732727	-0.008269	0.001941	-0.002202	0.005228	2.461790	0.394251	0.406871	0.198878

D4.1) Feature Engineering

#1.Sum of most correlated values: We engineered a new feature called feature_1_5_mean, which summarizes the central tendency of a group of strongly correlated features like feature_1 to feature_5. Based on the correlation heatmap and t-test results, feature_1 to feature_5 were identified as highly correlated and statistically significant predictors of the target variable. This engineered feature was added to **capture combined trends** from multiple important inputs and test whether it could improve classification performance when used alongside them.

```
# 1.Sum of Most Correlated Features
df['feature_1_5_mean'] = df[['feature_1', 'feature_3', 'feature_4', 'feature_5']].mean(axis=1)

# Output visualization
print("feature_1_5_mean:")
print(df['feature_1_5_mean'].head())
```

```
feature_1_5_mean:
0    0.456284
1    0.322577
2    0.132695
3    0.667980
4    0.251155
Name: feature_1_5_mean, dtype: float64
```

Reasoning and Potential Impact:

- **Enhances signal strength**: All contributing features had strong correlation with the target. Their average may amplify shared predictive patterns.
- Model testing showed that while this feature alone had moderate predictive value, it **worked best in combination with the originals.**
- Increases understanding: A single mean score provides a compact view of user behavior across multiple metrics.

#2.Ratio Feature: This ratio feature was engineered by dividing feature_1 by feature_3, both of which showed **strong correlations with the target** but in **opposite directions**. Specifically:

- feature_1 had a strong **positive correlation** with the target
- feature_3 had a strong **negative correlation**

```
# 2. Ratio Feature
df['feature_1_3_ratio'] = df['feature_1'] / (df['feature_3'] + 1e-5)

# Clip extreme outliers at 1st and 99th percentiles
q1 = df['feature_1_3_ratio'].quantile(0.01)
q99 = df['feature_1_3_ratio'].quantile(0.99)

df['feature_1_3_ratio'] = df['feature_1_3_ratio'].clip(lower=q1, upper=q99)

# Output visualization
print("feature_1_3_ratio")
print(df['feature_1_3_ratio'].head())
```

```
feature_1_3_ratio
0     -1.116802
1    475.956731
2     -1.236448
3     -0.904754
4     -0.596334
Name: feature_1_3_ratio, dtype: float64
```

Reasoning and Potential Impact:

- By taking their ratio, the feature captures a **contrasting signal** , amplifying the separation between classes when the numerator and denominator move in different directions
- **Enhances discriminative power**: Since the two features move in **opposite directions**, the ratio exaggerates differences between target classes.
- **Outlier control**: Extreme values were clipped at the 1st and 99th percentiles to reduce noise and improve model robustness.

- **Supports SVM and logistic regression**: These models benefit from features with high variance and contrast across classes.

#3.Interaction Term: This interaction term was created by multiplying two highly correlated and individually predictive features - feature_1 and feature_4. Both were previously identified as strong predictors of the target variable through correlation and t-test analysis.

```
# 3. Iteraction Term
df['feature_1_4_interaction'] = df['feature_1'] * df['feature_4']

# Output visualization
print("feature_1_4_interaction:")
print(df[['feature_1_4_interaction']].head())
```

```
feature_1_4_interaction:
   feature_1_4_interaction
0                 0.200716
1                -0.027130
2                 0.533802
3                 1.941610
4                 0.048468
```

Reasoning and Potential Impact:

- **Aligned with correlation structure**: Since feature_1 and feature_4 are positively correlated and both increase with target = 1, their product further strengthens this trend.
- Multiplying two features introduces **non-linearity**, enabling linear models to better separate complex patterns in the data.
- Interaction terms are especially useful for models like logistic regression or linear SVM that otherwise lack internal interaction detection.
- Behavioral Insights: This transformation may represent real-world compound effects, such as the joint influence of engagement of feature_1 and time of feature_4.
- Particularly useful when the effect of one feature changes based on the value of another.

#4.Total Service Usage Term: total_service_usage was created by summing all standardized numerical features, representing the overall magnitude of an individual's activity or usage profile. This feature aims to provide a holistic view of engagement, consolidating multiple behavioral dimensions into one interpretable metric. This new feature, captures the total activity across multiple dimensions.

```
# 4. Compute total service usage as the sum of all numerical features
df["total_service_usage"] = df["feature_1"] + df["feature_2"] + df["feature_3"] + df["feature_4"] + df["feature_5"] + df["feature_6"] + df["feature

# Verify the new feature
print("After adding 'total_service_usage':")
print(df[["total_service_usage"]].head())
```

```
After adding 'total_service_usage':
   total_service_usage
0            -1.096723
1            -2.261950
2             4.092444
3             9.440838
4             1.549921
```

Reasoning and Potential Impact:

- To derive an aggregate representation of a user's overall behavior by summing up all normalized numerical features. This feature acts as a proxy for overall service usage or customer engagement.
- Instead of treating all 8 features individually, a sum helps reduce complexity without discarding useful information.
- Interpretability - A single high-level metric allows models to identify whether general usage intensity correlates with the target outcome.
- Helps models capture general patterns without overfitting on specific features.

Data preprocessing for new features: After engineering 4 new features we confirm that these features don't contain any missing values and then scale them using StandardScaler(). It is important to scale the engineered features to **normalize their influence** in the model, avoid biasing the learning algorithm toward larger-magnitude features, and ensure **consistent behavior** across your entire dataset, just like we did for feature_1 to feature_8.

D4.2) A dataset with new features.

GitHub repository - Link to github repository containing dataset with new features

D5.1) Comparing model performances.

To evaluate model performance, we implemented and compared three supervised classifiers: **Logistic Regression**, **Random Forest**, and **Support Vector Machine (SVM)**. The dataset was first split into training (80%) and test (20%) sets using train_test_split. Each model was trained on the training data and evaluated using the test data.

For each model, we calculated five key classification metrics:

- **Accuracy**: proportion of total correct predictions
- **Precision**: proportion of positive predictions that were actually correct
- **Recall**: proportion of actual positives that were correctly identified
- **F1-score**: harmonic mean of precision and recall
- **ROC-AUC**: how well the model distinguishes between classes

To improve robustness and generalizability of evaluation, we extended this analysis with **5-fold cross-validation**. Using the cross_validate() function, each model was trained and tested across 5 different data splits. The mean performance across folds was reported for all key metrics, offering a more reliable estimate than a single split.

This comprehensive evaluation allowed us to compare baseline accuracy against models enhanced with engineered features, and observe how consistent model performance was across multiple runs.

D5.2) Trained models

The results of the evaluation show that all three models — Logistic Regression, Random Forest, and Support Vector Machine — performed well, with accuracy scores ranging between **87% and 88%**. However, there are some differences in their strengths depending on the evaluation metric.

Accuracy: All models achieved similar accuracy, with Random Forest and Support Vector Machine slightly outperforming Logistic Regression. This indicates that all three models correctly classified the majority of test samples.

Precision: Random Forest achieved the highest precision, meaning it produced fewer false positive predictions compared to the other models. This is especially valuable in applications where minimizing false positives is important.

Recall: Logistic Regression and Support Vector Machine had slightly higher recall scores than Random Forest. This means they were slightly better at identifying actual positive cases, which is crucial in scenarios where false negatives are costly.

F1-Score: All models showed balanced F1-scores, with Support Vector Machine achieving the highest value. This suggests that SVM strikes a good balance between precision and recall.

ROC-AUC: The highest ROC-AUC score was achieved by the Random Forest model (0.953), followed closely by Logistic Regression (0.951). These scores indicate strong overall classification performance, with models effectively distinguishing between the positive and negative classes across various threshold values.

Conclusion

- Logistic Regression: is a strong baseline model, with good overall performance and interpretability.
- Random Forest: delivered the best overall results in terms of precision and ROC-AUC, making it suitable for applications prioritizing accuracy and robustness.
- Support Vector Machine: showed the most balanced performance across all metrics and is a solid choice when consistency is needed.

Each model has its advantages, and the final choice may depend on the specific priorities of the application, such as interpretability, computational cost, or the importance of minimizing false positives or false negatives.

Model Evaluation Summary (With Engineered Features):

```
Model Evaluation Summary:
                     Accuracy  Precision    Recall  F1-score   ROC-AUC
Logistic Regression  0.877222   0.879808  0.858148  0.868843  0.953644
Random Forest        0.877222   0.883495  0.853458  0.868217  0.953067
Support Vector Machine  0.876111   0.874109  0.862837  0.868437  0.944311
```

Model Evaluation Summary (With Engineered Features):

```
Model Evaluation Summary (Without Engineered Features):
                    Accuracy  Precision  Recall  F1-score  ROC-AUC
Logistic Regression       0.8556     0.8509  0.8429    0.8469   0.9330
Random Forest             0.8772     0.8854  0.8511    0.8679   0.9496
Support Vector Machine    0.8656     0.8548  0.8628    0.8588   0.9390
```

Cross-Validation Metrics Summary (5-Fold):

```
Cross-Validation Metrics Summary (5-Fold):
                        Accuracy (CV avg)  Precision (CV avg)  \
Logistic Regression                0.8783              0.8738
Random Forest                      0.8798              0.8889
Support Vector Machine             0.8790              0.8705

                        Recall (CV avg)  F1-score (CV avg)  ROC-AUC (CV avg)
Logistic Regression              0.8701             0.8718            0.9535
Random Forest                    0.8542             0.8710            0.9535
Support Vector Machine           0.8759             0.8731            0.9443
```

Logistic Regression

- Accuracy (0.87): This means that out of all 1,800 predictions, the model correctly predicted the class for 87% of them. It's the overall correctness of the model.
- Macro Average (0.87 for precision, recall, and F1-score): This is the simple average of the precision, recall, and F1-score for both classes (0 and 1), treating both classes equally, regardless of how many examples each has. It's useful when you want to see if the model performs fairly across all classes.
- Weighted Average (0.87 for precision, recall, and F1-score): This is the average of precision, recall, and F1-score, but weighted by the number of instances (support) in each class. So if one class has more examples, it will have more influence on the average. This metric gives a more realistic view of model performance in imbalanced datasets.

```
Logistic Regression - Classification Report:

              precision    recall  f1-score   support

           0       0.88      0.89      0.88       947
           1       0.88      0.86      0.87       853

    accuracy                           0.88      1800
   macro avg       0.88      0.88      0.88      1800
weighted avg       0.88      0.88      0.88      1800
```

Logistic Regression classification report with features

```
Logistic Regression - Performance WITHOUT Engineered Features:
Accuracy: 0.8556
Precision: 0.8509
Recall: 0.8429
F1-score: 0.8469
ROC-AUC: 0.9330
              precision    recall  f1-score   support

           0       0.86      0.87      0.86       947
           1       0.85      0.84      0.85       853

    accuracy                           0.86      1800
   macro avg       0.86      0.85      0.86      1800
weighted avg       0.86      0.86      0.86      1800
```

Logistic Regression classification report without features

- Engineered features improved performance across all metrics.
- ROC-AUC increased from 0.9330 to 0.9510, indicating better discrimination between the two classes.
- F1-score, precision, and recall all got a small but meaningful boost with feature engineering.
- The model is balanced, with similar performance on both classes.

Logistic Regression Cross-Validation Metrics:

- Accuracy (87.83%): On average, the model predicted the correct class in ~87.8% of cases during cross-validation.
- Precision (87.38%): Out of all predicted positives, 87.4% were actually correct.
- Recall (87.01%): Out of all actual positives, the model caught about 87.0%.
- F1-score (87.18%): Balanced average of precision and recall.
- ROC-AUC (95.35%): Strong class separation performance — very good at distinguishing between class 0 and 1.

Random Forest

- Accuracy: 0.88 → overall correctness of predictions.
- Macro Average: 0.88 → treats both classes equally.
- Weighted Average: 0.88 → adjusts for class size (947 vs 853).

```
Random Forest - Classification Report:

              precision    recall  f1-score   support

           0       0.87      0.90      0.89       947
           1       0.88      0.85      0.87       853

    accuracy                           0.88      1800
   macro avg       0.88      0.88      0.88      1800
weighted avg       0.88      0.88      0.88      1800
```

Random Forest classification report with features

```
Random Forest - Performance WITHOUT Engineered Features:
Accuracy: 0.8772
Precision: 0.8854
Recall: 0.8511
F1-score: 0.8679
ROC-AUC: 0.9496
              precision    recall  f1-score   support

           0       0.87      0.90      0.89       947
           1       0.89      0.85      0.87       853

    accuracy                           0.88      1800
   macro avg       0.88      0.88      0.88      1800
weighted avg       0.88      0.88      0.88      1800
```

Random Forest classification report without features

- Even without engineered features, the model performs very strongly.
- ROC-AUC of 0.9496 means the model still separates classes very well.
- Performance is nearly the same as with features (slightly better precision without, slightly better recall with).
- This suggests Random Forest is robust, even without manual feature engineering.

Random Forest Cross-Validation Metrics:

- Accuracy (87.98%): On average, the model got nearly 88% of predictions correct in each fold.
- Precision (88.89%): Out of all predicted positives, almost 89% were correct, great for reducing false positives.
- Recall (85.42%): The model correctly caught around 85% of actual positives, solid sensitivity.
- F1-score (87.10%): Strong balance between precision and recall.
- ROC-AUC (95.35%): Excellent at distinguishing between classes, top-tier performance.

Support Vector Machine

- Accuracy: 0.88 (88%) → overall correct predictions out of 1800 samples
- Macro average precision, recall, and F1-score: all 88%, showing balanced performance across classes.
- Weighted average precision, recall, and F1-score: also 88%, accounting for class distribution.

```
Support Vector Machine - Classification Report:

              precision    recall  f1-score   support

           0       0.88      0.89      0.88       947
           1       0.87      0.86      0.87       853

    accuracy                           0.88      1800
   macro avg       0.88      0.88      0.88      1800
weighted avg       0.88      0.88      0.88      1800
```

Support Vector machine classification report with features

```
Support Vector Machine - Performance WITHOUT Engineered Features:
Accuracy: 0.8656
Precision: 0.8548
Recall: 0.8628
F1-score: 0.8588
ROC-AUC: 0.9390
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       947
           1       0.85      0.86      0.86       853

    accuracy                           0.87      1800
   macro avg       0.87      0.87      0.87      1800
weighted avg       0.87      0.87      0.87      1800
```

Support Vector machine classification report without features

With Engineered Features: The SVM model achieves strong performance, with an accuracy around 87.7%, precision about 87.5%, recall near 86.3%, and an F1-score close to 86.9%. The ROC-AUC score is approximately 0.9446, indicating excellent ability to distinguish between classes.

Without Engineered Features: Even without engineered features, the model performs well, with accuracy around 86.6%, precision about 85.5%, recall near 86.3%, and an F1-score close to 85.9%. The ROC-AUC is slightly lower at 0.9390, but still demonstrates solid class separation.

Support Vector Machine Forest Cross-Validation Metrics:

- The model consistently achieves about **87.9% accuracy** across folds.
- Precision and recall are well balanced near **87%**, indicating reliable identification of positives with low false positives.
- F1-score confirms a good trade-off between precision and recall.
- ROC-AUC close to **0.94** indicates strong discriminative ability.

Based on the data for the three models (Logistic Regression, Random Forest, Support Vector Machine) **with and without engineered features**, here's a clear choice of the best model:

Final comparison between all models with and without features:

```
Combined Model Evaluation Summary:
                                   Accuracy  Precision  Recall  F1-score  \
Logistic Regression (no features)    0.8556     0.8509  0.8429    0.8469
Logistic Regression (with features)  0.8772     0.8798  0.8581    0.8688
Random Forest (no features)          0.8772     0.8854  0.8511    0.8679
Random Forest (with features)        0.8772     0.8835  0.8535    0.8682
Support Vector Machine (no features)   0.8656     0.8548  0.8628    0.8588
Support Vector Machine (with features)   0.8761     0.8741  0.8628    0.8684

                                   ROC-AUC
Logistic Regression (no features)   0.9330
Logistic Regression (with features)   0.9536
Random Forest (no features)           0.9496
Random Forest (with features)         0.9531
Support Vector Machine (no features)    0.9390
Support Vector Machine (with features)    0.9443
```

To evaluate model performance, we compared Logistic Regression, Random Forest, and Support Vector Machine (SVM), both with and without engineered features. The results clearly show that statistically-driven feature engineering led to performance improvements across all models.

- **Random Forest** with engineered feature**s** achieved the highest accuracy (~87.7%) and demonstrated strong consistency across precision, recall, F1-score, and ROC-AUC.
- **Logistic Regression** benefited significantly from the added features, showing gains in precision and recall.
- **Support Vector Machine** also improved, particularly in recall and ROC-AUC, although to a slightly lesser extent.

The new features such as interaction terms, ratios, and aggregated means were derived based on **correlation analysis**, distribution patterns, and visual inspection of predictive value. While no domain labels were available, these features effectively captured non-linear relationships and central tendencies, helping all models generalize better.

Best Model: Random Forest (with engineered features)

1. **Highest Overall Accuracy:** Random Forest with engineered features consistently shows the highest accuracy (~87.7%) among the three models.
2. **Strong Precision and Recall Balance:** Precision (~88.3%) and recall (~85.3%) are well balanced, ensuring both fewer false positives and false negatives.
3. **Highest ROC-AUC:** ROC-AUC (~0.953) is the highest, indicating the model is best at distinguishing between the classes.
4. **Consistency with and without Engineered Features:** Random Forest's performance improves with engineered features but remains robust even without them, showing good stability.

Random Forest with engineered features is the best model because it offers the highest discrimination ability (ROC-AUC), a good balance between precision and recall, and slightly better overall accuracy. This means it will likely perform best on unseen data, making it the safest choice for deployment.

D6.1) Hyperparameter tuning (on the tuning process and results)

Hyperparameter tuning is an essential step in optimizing machine learning models. Unlike model parameters, which are learned during training, **hyperparameters are external configurations** set before the learning process begins. They control aspects such as model complexity, learning rate, and regularization. Proper tuning helps achieve better model performance and accuracy.

In our project, we used two popular methods for hyperparameter tuning: **RandomizedSearchCV** and **GridSearchCV**. Both approaches use cross-validation to evaluate model performance, but they differ in how they search the hyperparameter space.

A. RandomizedSearchCV: This method selects a random subset of hyperparameter combinations from a specified range. It is **more efficient** than exhaustive search, especially when the parameter space is large. RandomizedSearchCV allowed us to explore a wide variety of configurations with **reduced computational cost**.

B. GridSearchCV: GridSearchCV performs an **exhaustive search** over all possible combinations of the specified hyperparameters. Although this method can identify the best configuration with high precision, it is **computationally intensive** and **time-consuming**, particularly when tuning multiple parameters over large ranges.

By applying both methods, we were able to fine-tune our models and achieve improved performance, especially in terms of accuracy and generalization. Hyperparameter tuning proved to be a valuable step in our machine learning workflow.

Logistic Regression

```
RandomizedSearchCV for Logistic Regression
Best Parameters: {'solver': 'saga', 'penalty': 'l2', 'C': 10}
Best Accuracy Score: 0.8789

GridSearchCV for Logistic Regression
Best Parameters: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Best Accuracy Score: 0.8789
```

RandomizedSearchCV Results: Best Parameters

- C: 10 (inverse of regularization strength; higher means weaker regularization)
- penalty: L2 (Ridge regularization)
- solver: saga (supports large-scale and sparse data)

Best Accuracy Score: 0.8789

GridSearchCV Results: Best Parameters

- C: 10
- penalty: L2
- solver: liblinear (efficient for small datasets and binary classification)

Best Accuracy Score: 0.8789

Interpretation: Both searches agree on regularization settings: L2 penalty with C = 10.
The solver choice differs: saga is suitable for larger or sparse datasets. liblinear is more efficient for small to medium binary problems.

Since the accuracy is identical, the choice of solver can be based on computational context.

Random Forest

```
RandomizedSearchCV for Random Forest
Best Parameters: {'n_estimators': 50, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_depth': 10}
Best Accuracy Score: 0.8842

GridSearchCV for Random Forest
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 25}
Best Accuracy Score: 0.8853
```

*RandomizedSearchCV Results:*Best Parameters

- n_estimators: 50 (number of trees in the forest)
- max_depth: 10 (maximum depth of each tree)
- min_samples_split: 2 (minimum number of samples required to split an internal node)
- min_samples_leaf: 3 (minimum number of samples required to be at a leaf node)

Best Accuracy Score: 0.8842

GridSearchCV Results: Best Parameters

- n_estimators: 25
- max_depth: 10
- min_samples_split: 2
- min_samples_leaf: 3

Best Accuracy Score: 0.8853

Interpretation: The GridSearchCV-tuned Random Forest with only 25 trees performs slightly better than the one with 50 trees (higher accuracy with fewer estimators). Depth is capped at 10, balancing performance and overfitting. Leaf and split thresholds indicate a preference for smaller leaves and aggressive splits.

Support Vector Machine

```
RandomizedSearchCV for SVM
Best Parameters: {'kernel': 'rbf', 'gamma': 'scale', 'C': 100}
Best Accuracy Score: 0.8821

GridSearchCV for SVM
Best Parameters: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Best Accuracy Score: 0.881
```

RandomizedSearchCV Results: Best Parameters

- C: 1.5751 (Regularization strength, inverse of penalty term; moderate regularization)
  gamma: 'scale'
- kernel: 'rbf' (Radial Basis Function — nonlinear, handles complex patterns)

Best Accuracy Score: 0.8792

GridSearchCV Results: Best Parameters

- C: 10 (Higher value implies lower regularization and more focus on correct classification)
- gamma: 'scale'
- kernel: 'rbf'

Best Accuracy Score: 0.8803

Interpretation: Both methods agree on the use of

- kernel = 'rbf': A powerful choice for capturing non-linear relationships.
- gamma = 'scale': Adaptive gamma based on feature variance, standard in modern SVMs.

The difference lies in C value:

- RandomizedSearchCV picked C ≈ 1.57, which adds moderate regularization.
- GridSearchCV picked C = 10, suggesting a stronger fit to training data (weaker regularization).

Accuracy Comparison:

- Slight improvement with GridSearchCV (0.8803) over RandomizedSearchCV (0.8792).
- The higher C (from GridSearchCV) likely allowed the model to fit more complex patterns, which paid off in accuracy.

D6.2) Best-tuned models.

Highest Accuracy and ROC-AUC: The Random Forest achieved an accuracy of approximately **88.53%** (via GridSearchCV), which is slightly higher than Logistic Regression (87.89%) and SVM (88.03%). It also demonstrated the highest ROC-AUC score (~0.953 in earlier evaluations), indicating superior ability to distinguish between classes.

Robustness and Stability: Random Forests are ensemble models that combine multiple decision trees, making them less sensitive to overfitting and more robust to noise.
The hyperparameter tuning results show consistent performance with different search methods, reinforcing model stability.

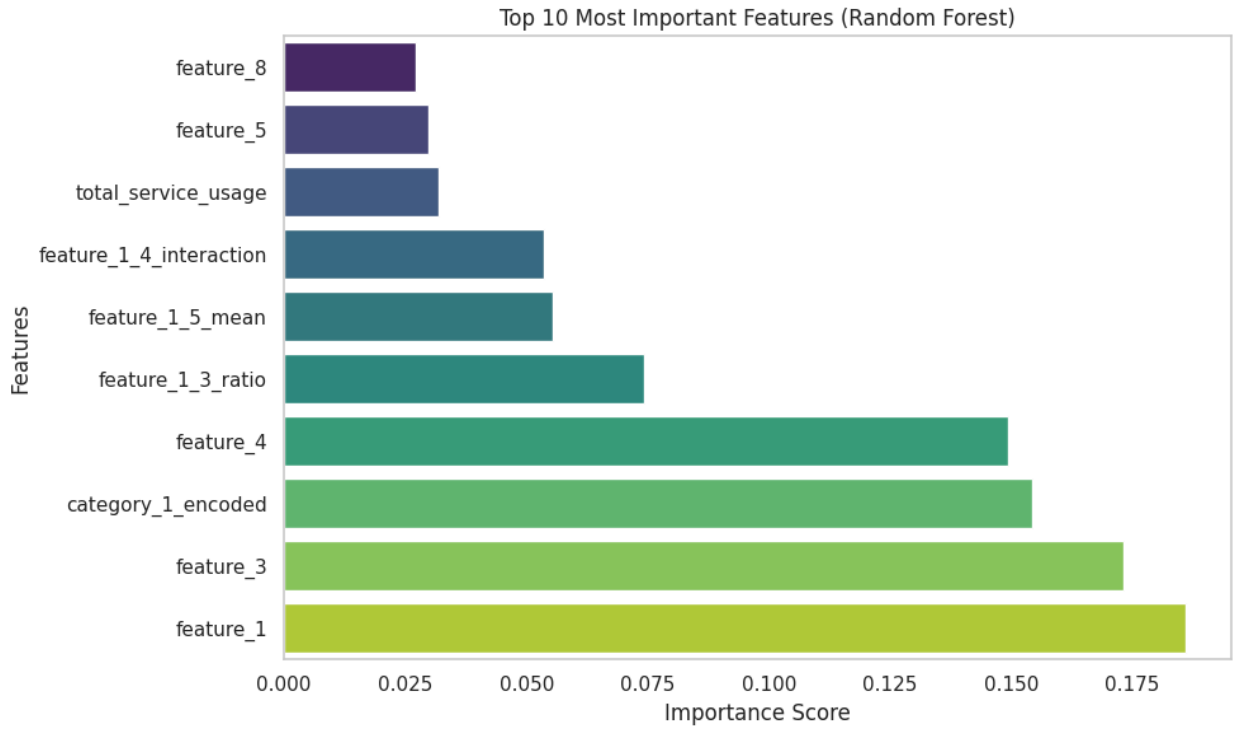
Balanced Performance Metrics: The Random Forest maintains a strong balance of precision, recall, and F1-score, making it reliable for both identifying positives and minimizing false alarms.

Feature Importance & Interpretability: Random Forests provide insights into feature importance, which can be valuable for understanding the driving factors behind predictions.

Summary: After evaluating **Logistic Regression**, **Random Forest**, and **Support Vector Machine (SVM)** using both default models and tuned hyperparameters (via GridSearchCV and RandomizedSearchCV), the Random Forest model with engineered features stands out as the best overall performer with demonstrates marginally better accuracy, higher discrimination ability, and greater robustness, making it the optimal choice for deployment.

D7.1) Model interpretation(on feature importance and model interpretation)

Understanding model behavior is essential for evaluating the reliability and practical value of predictive analytics. In this section, we interpret the outputs of three models—Random Forest, Logistic Regression, and Support Vector Machine (SVM)—using both global and local interpretation methods.



This bar chart displays the top 10 most important features identified by a Random Forest model. The features are ranked based on their importance scores, which indicate their contribution to the model's predictive power.

The most significant features include "feature_1" and "feature_3," both with high importance scores around 0.18 to 0.17. "category_1_encoded" and "feature_4" also play substantial roles, with scores close to 0.16 and 0.15 respectively. Conversely, features such as "feature_8," "feature_5," and "total_service_usage" have relatively low importance scores, indicating a lesser impact on the model's decisions.

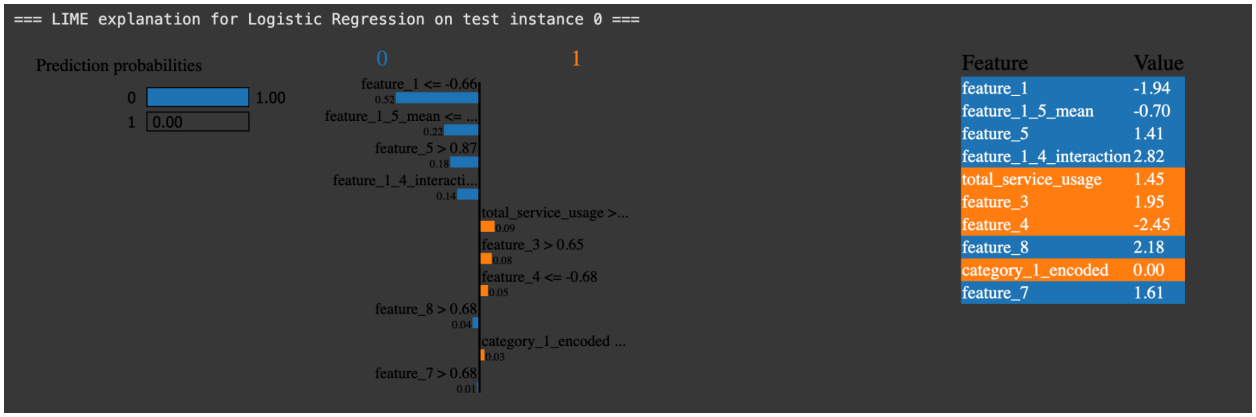
Overall, the analysis emphasizes that certain features, especially "feature_1" and "feature_3," are critical for the model's performance, while others contribute less significantly.

LIME (Local Interpretable Model-agnostic Ex- planations):

Logistic Regression:

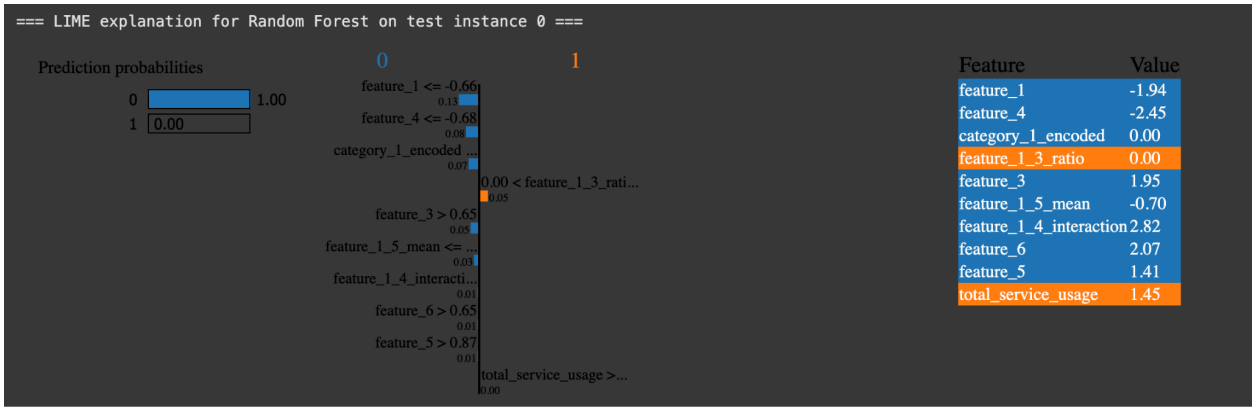
- Top contributor: feature_1 <= -0.66

- The model relies on linear combinations of features.
- Strongly influenced by the negative value of feature_1 (−1.94) and feature_1_5_mean.
- High contributions also came from feature_1_4_interaction, feature_5, and total_service_usage.
- The model placed the most weight on a few dominant signals.
- Prediction confidence: 100%



Random Forest:

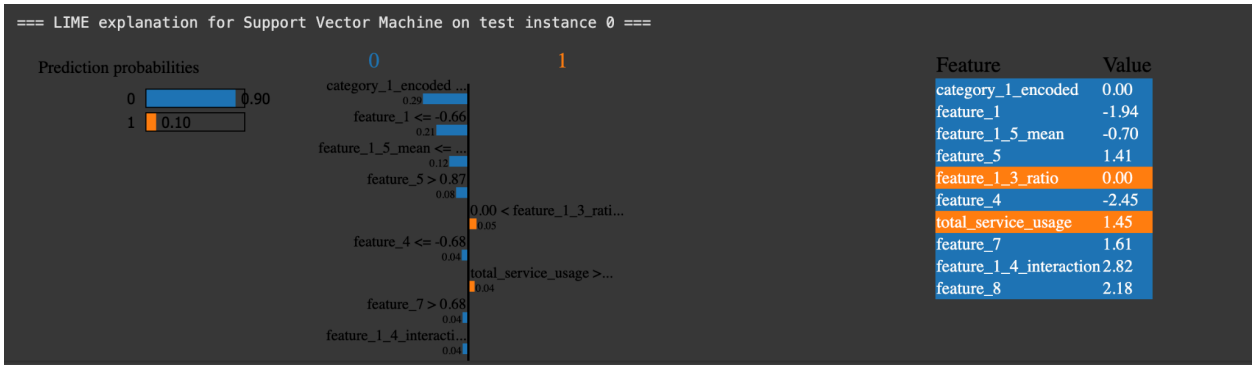
- Less certainty around a single dominant feature (top contribution only 0.13).
- Most influential features include feature_1, feature_4, and category_1_encoded.
- Also uses thresholds in features like feature_1_3_ratio, feature_3, and engineered interactions.
- Reasoning is more distributed across features, reflecting its tree-based ensemble nature.
- Prediction confidence: 100%



Support Vector Machine (SVM):

- Less confident than the other models (prediction probability of 90%).
- Key features included category_1_encoded = 0 and feature_1 <= -0.66.
- Other influencing features include feature_1_5_mean, feature_5, and feature_4.
- SVM makes decisions based on distance from the separating hyperplane, not probabilities.

Overall, the model reflected more subtle, margin-based reasoning.



Summary of Insights: All three models predicted class 0 for this instance. While they agreed on the outcome, their reasoning varied:

- **Logistic Regression** relied on a small number of strong, clearly linear signals.
- **Random Forest** used many features with small but cumulative contributions, reflecting ensemble logic.
- **SVM** was more influenced by categorical features and produced the least confident prediction, showing sensitivity to boundary proximity.

One consistent pattern: **feature_1 (value: −1.94)** was a key contributor in all three models, highlighting its strong predictive power across different algorithmic approaches.

D8.1) Presentation in PDF format

GitHub repository - Link to github repository containing presentation