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

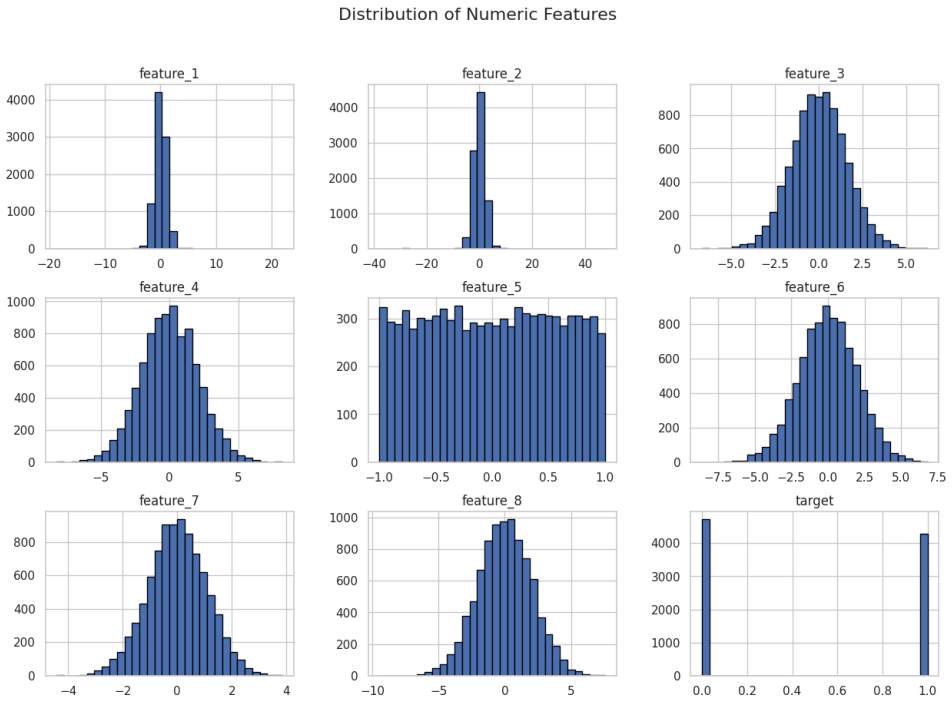
The dataset consists of **9000 entries** (rows) and **11 columns**, representing structured records for a binary classification task.

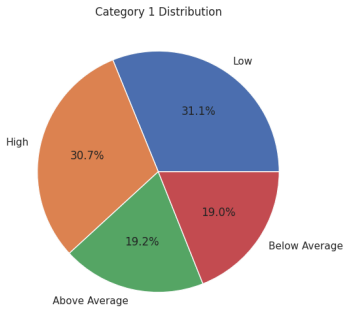
Feature Overview:

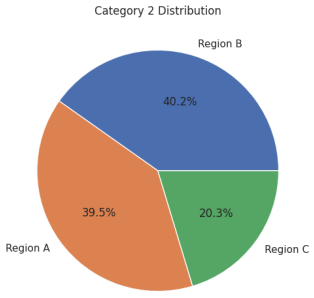
- Numerical Features (8) : There are 8 numerical feature
- Categorical Features (2) :
    1. category_1: Ordinal values such as "Low", "Below Average", "Above Average", and "High"
    2. category_2: Nominal values like "Region A", "Region B", and "Region C"
- Target Variable: ‘target’ a binary label indicating class 0 or 1 used for classification tasks

```
[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. 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.





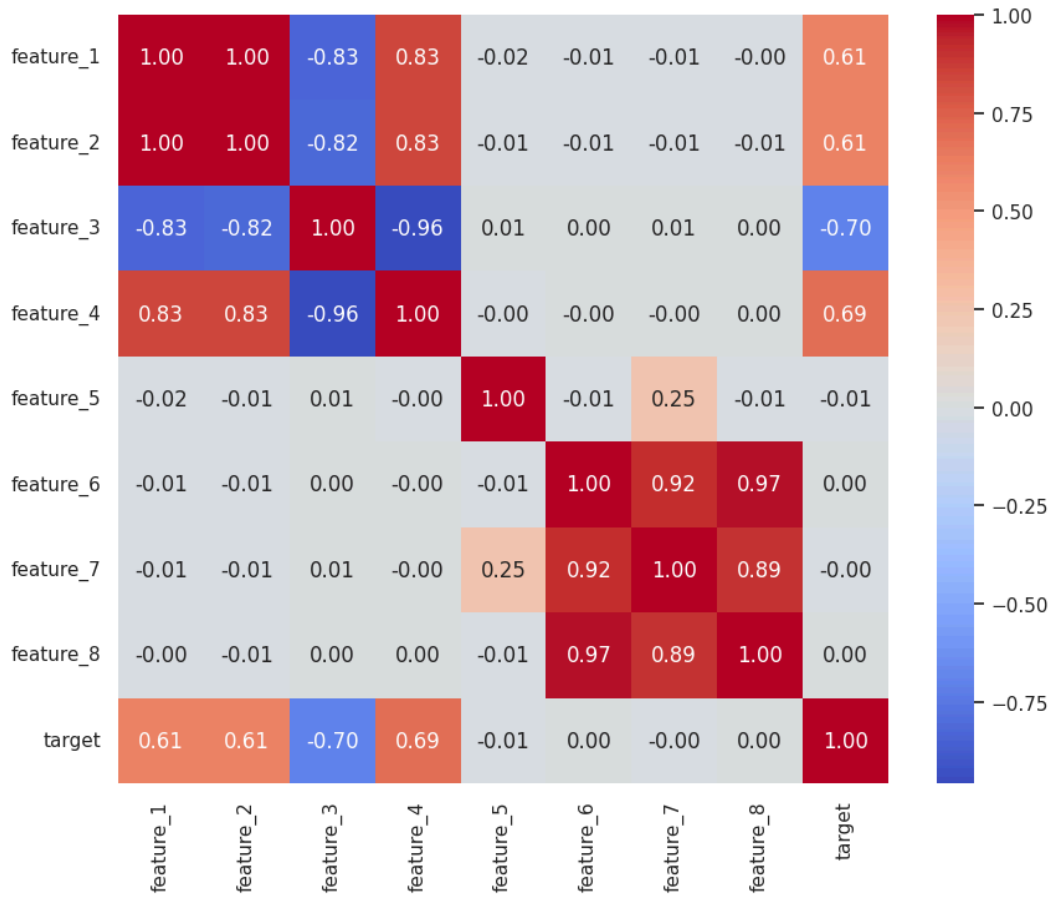


From the visualizations, we make the following observations:

- 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.
- 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.

Correlation Matrix:



From the visualizations, we make the following observations:

Strong Positive Correlations:

- feature_1 and feature_2: r = **+1.00** - These two are **perfectly correlated**. That means if feature one value increases then feature two increases as well.
- feature_1, feature_2 and 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 and 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:

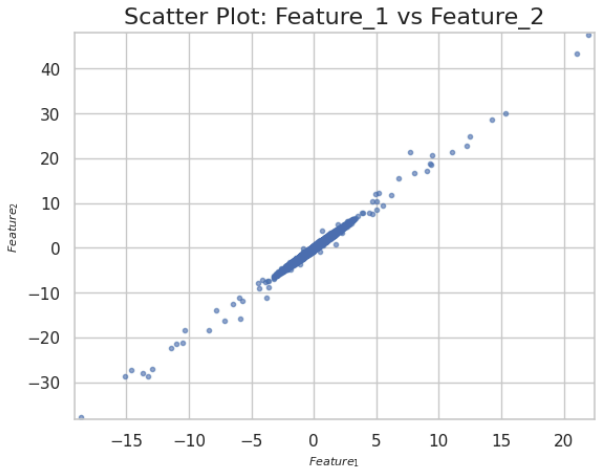
- feature_6, feature_7, and feature_8 form a highly intercorrelated block (r = **0.89–0.97**)

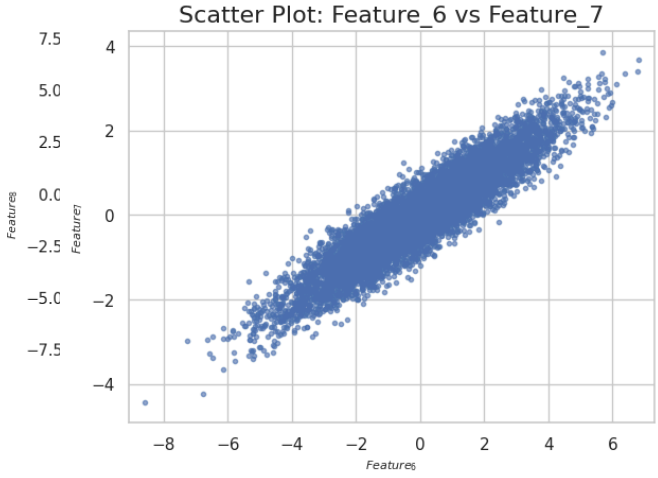
Weak or No Correlation:

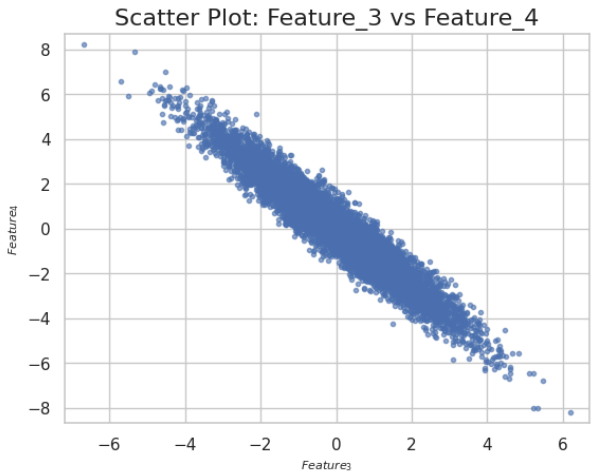
- feature_5 shows **no meaningful correlation** with any other feature or the target — it may not add much signal.
- feature_6–8 also show **no direct correlation with the target**, despite being internally consistent.

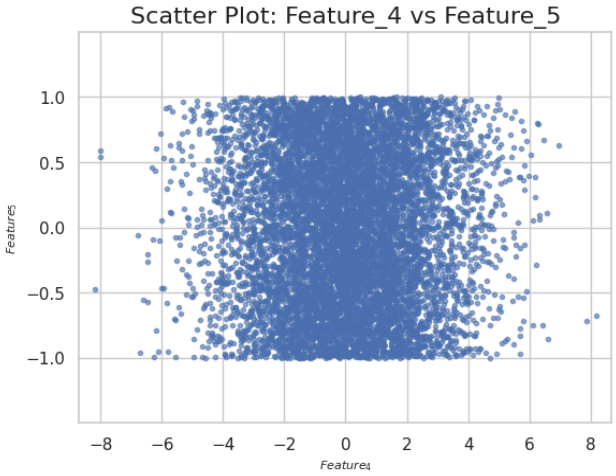
Target Correlation: **Highest correlations with the target are -** feature_4**: +0.69,** feature_1 **&** feature_2**: +0.61,** feature_3**: –0.70. These features are most predictive.**

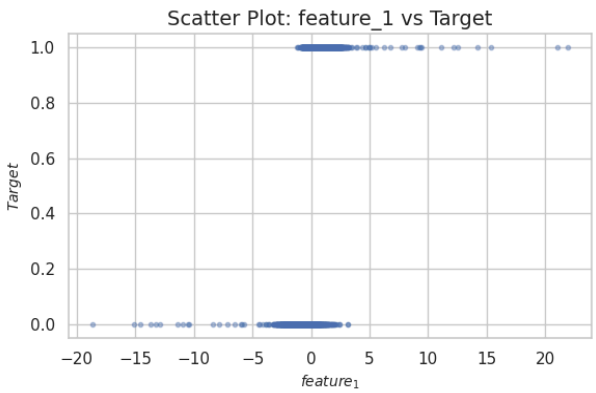
Scatter Plots: We can confirm the observation from the correlation matrix using scatter plots as well.

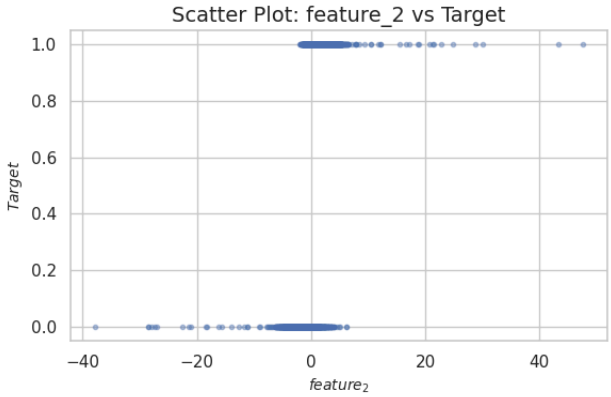


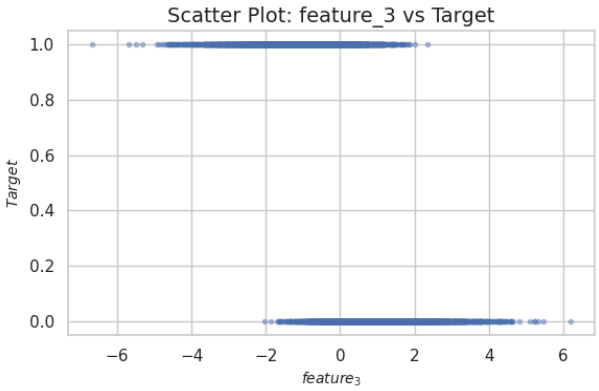


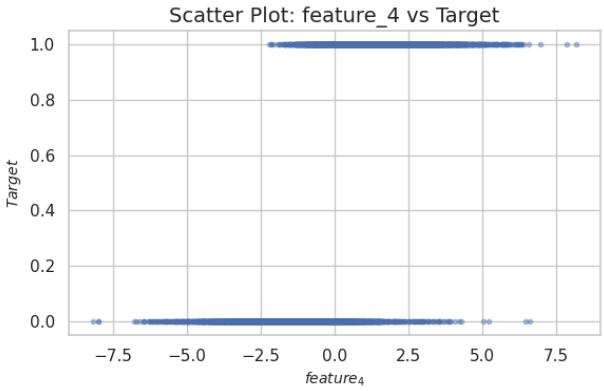












From the visualizations, we make the following observations:

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)
```

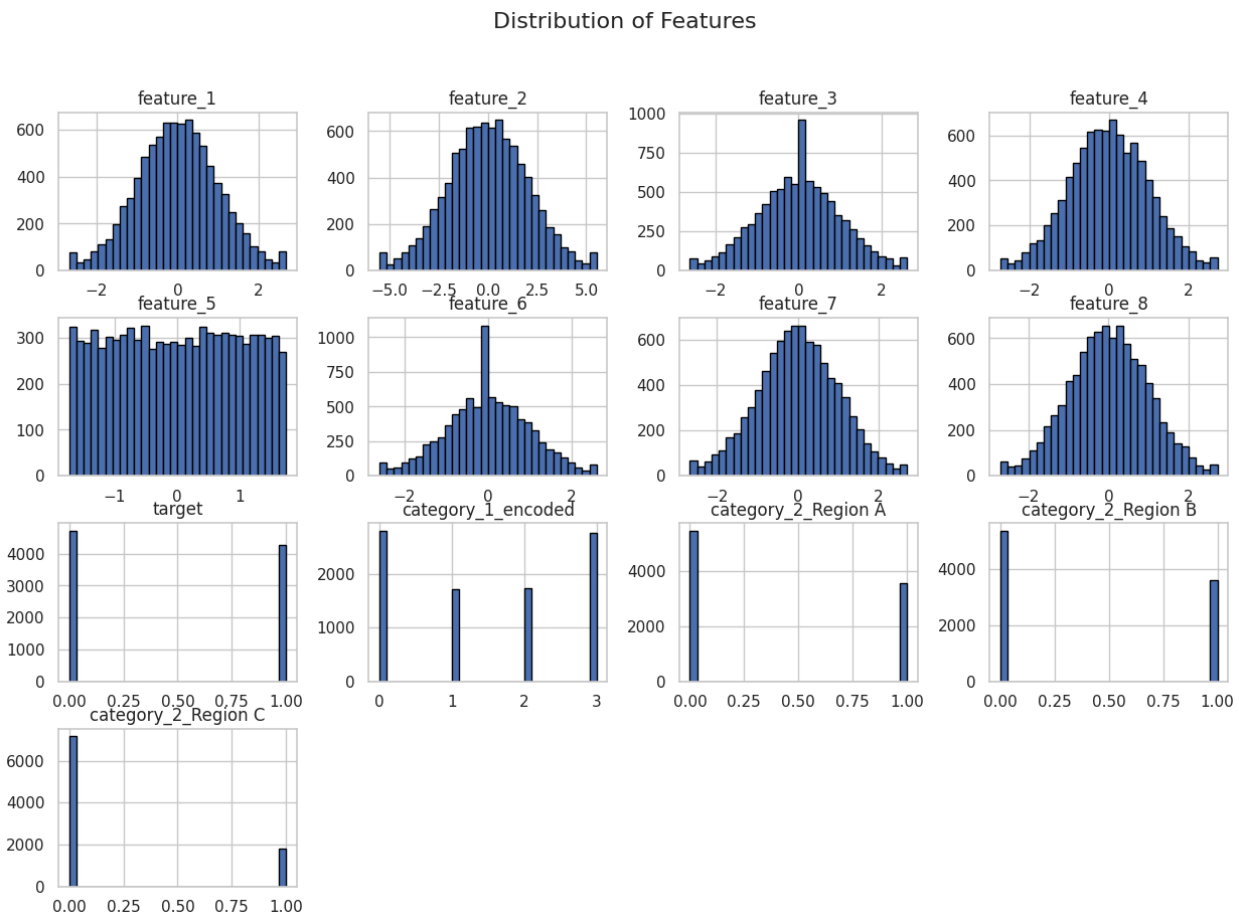
	category_1_encoded	category_2_Region A	category_2_Region B	category_2_Region C
0	2	0	0	1
1	1	1	0	0
2	3	0	0	1
3	3	0	1	0
4	1	0	0	1

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. 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_2, feature_3, feature_4, and feature_6 now display **centered, symmetrical bell-shaped curves**, indicating successful scaling. Outlier-heavy features 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 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.

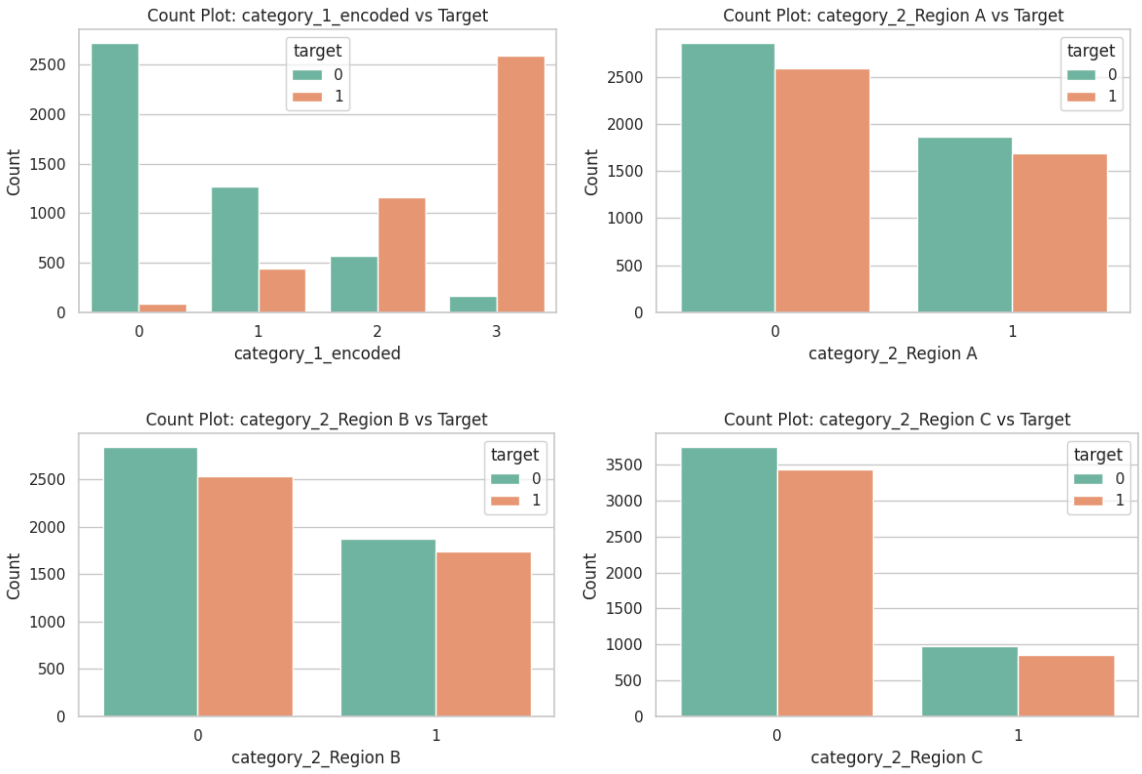


D2.2) Cleaned dataset

GitHub repository - Link to github repository with cleansed dataset

D3.1) Exploratory Data Analysis

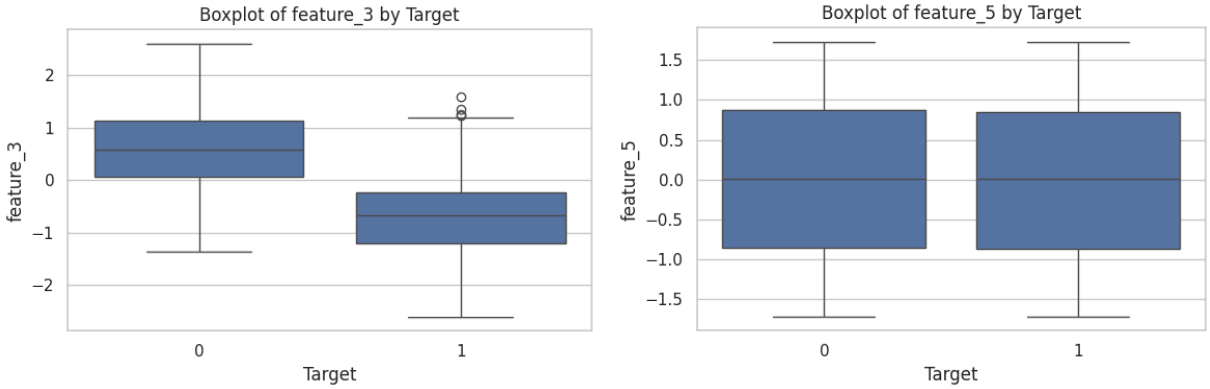
Bar charts(category vs target): We used **bar charts (count plots)** to explore the relationship between **categorical features** and the target variable. Bar charts are ideal for this purpose because they **show the frequency of each category** split by target class (0 or 1), allowing us to easily compare 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_2: t = 94.958, 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.
    - X^2 t is small (O ~ E) → Observed data fits expected data = likely no relationship if
    - X^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.**

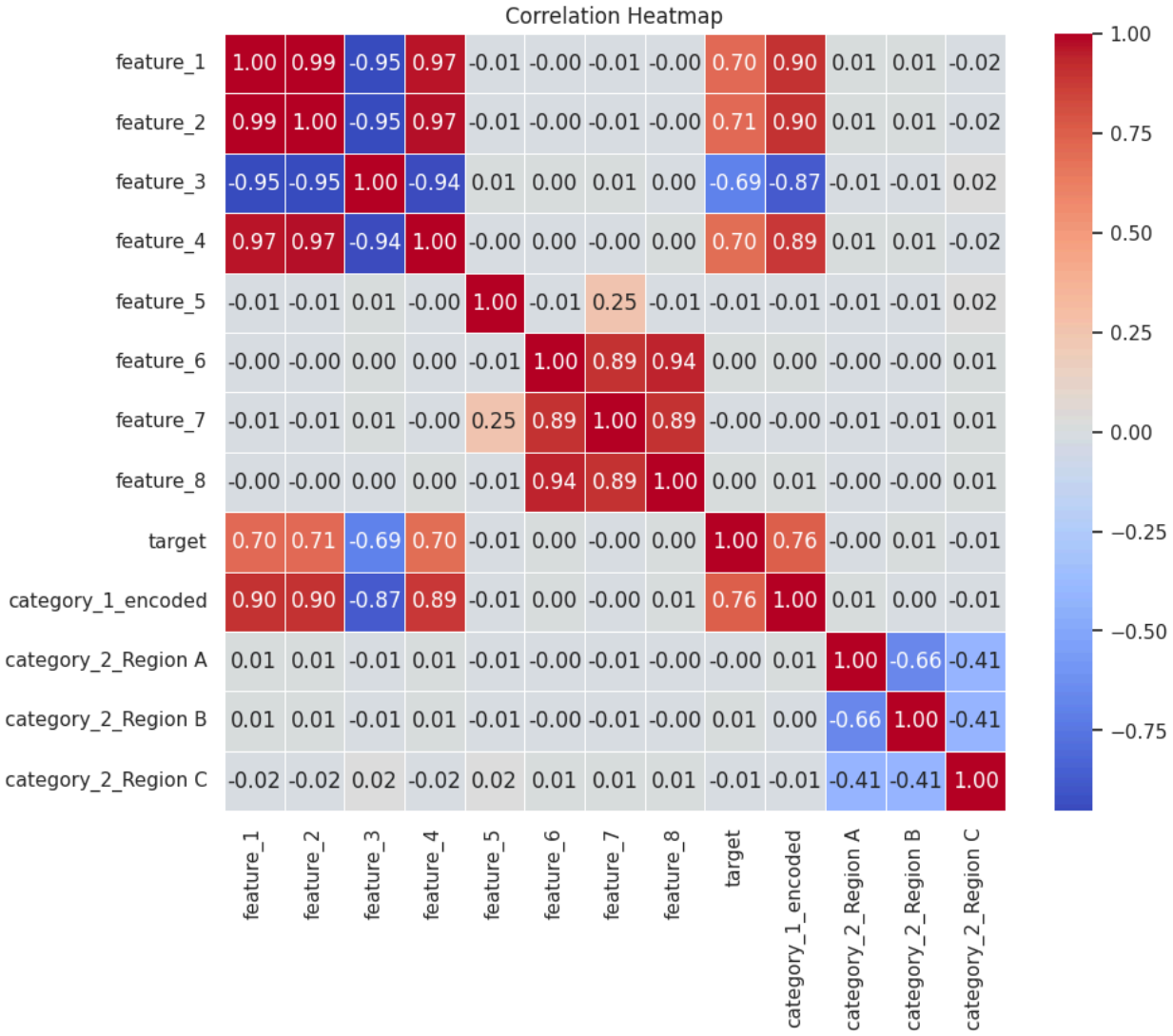
Observations:

- 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

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

target	feature_1	feature_2	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	-1.380133	0.660089	-0.664126	0.007495	-0.001759	0.001996	-0.004738	0.617878	0.394832	0.397585	0.207583
1	0.739948	1.522863	-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.

```
0s
# 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_2 by feature_3, both of which showed **strong correlations with the target** but in **opposite directions**. Specifically:

- feature_2 had a strong **positive correlation** with the target
- feature_3 had a strong **negative correlation**

```
[263] # 2. Ratio Feature
      df['feature_2_3_ratio'] = df['feature_2'] / (df['feature_3'] + 1e-5)

      # Output visualization
      print("feature_2_3_ratio")
      print(df['feature_2_3_ratio'].head())

      feature_2_3_ratio
      0     -1.278879
      1    106.673283
      2     -1.321890
      3     -0.784237
      4     -0.634791
```

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.

#3. Interaction Term: This interaction term was created by multiplying two highly correlated and individually predictive features feature_2 and feature_4. Both were previously identified as strong predictors of the target variable through correlation and t-test analysis.

```
[264] # 3. Iteraction Term
      df['feature_2_4_interaction'] = df['feature_2'] * df['feature_4']

      # Output visualization
      print("feature_2_4_interaction:")
      print(df[['feature_2_4_interaction']].head())

      feature_2_4_interaction:
         feature_2_4_interaction
      0                 0.229846
      1                -0.006081
      2                 0.570689
      3                 1.682980
      4                 0.051594
```

Reasoning and Potential Impact:

- **Aligned with correlation structure**: Since feature_2 and feature_4 are positively correlated and both increase with target = 1(as observed from the t-test score having extremely low p-values for both features and the box-plots which show **median values** of both features are higher for target = 1 than for target = 0) their product further strengthens this trend.
- Multiplying two features introduces **non-linearity**, enabling linear models to better separate complex patterns in the data.
- This transformation may represent real-world compound effects, such as the joint influence of engagement of feature_2 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 ensemble learning models: **Gradient Boosting**, **Random Forest**, and **AdaBoost**. 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_models() 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.

The results of the evaluation show that all three models Ada Boosting, Random Forest, and Gradient Boosting 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 three ensemble models achieved very similar accuracy scores, with Gradient Boosting slightly outperforming Random Forest and AdaBoost. This indicates that each model was able to correctly classify the majority of test instances with gradient boosting showing more overall accuracy.

Precision: AdaBoosting 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: Gradient Boosting and Random Forest had slightly higher recall scores than AdaBoosting. 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 Gradient Boosting achieving the highest value. This suggests that Gradient Boosting strikes a good balance between precision and recall.

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

Conclusion

- Gradient Boosting the best overall performance across all metrics, including accuracy, F1-score, and ROC-AUC, making it the most balanced model.
- Random Forest closely followed, showing strong recall and ROC-AUC, making it suitable for scenarios where identifying positives is crucial.
- AdaBoost stood out for its high precision, making it valuable in applications where minimizing false positives is important.

- 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 With Engineered Features:

Rank	Model	Accuracy	Precision	Recall	F1-score	Confusion Matrix	ROC-AUC
1	Gradient Boosting	0.881667	0.890244	0.855803	0.872684	[[857 90] [123 730]]	0.958062
2	Random Forest	0.876667	0.882424	0.853458	0.867700	[[850 97] [125 728]]	0.954131
3	AdaBoost	0.876667	0.890954	0.842907	0.866265	[[859 88] [134 719]]	0.945077

Model Evaluation Summary (With Engineered Features):

Model Evaluation Without Engineered Features:

Rank	Model	Accuracy	Precision	Recall	F1-score	Confusion Matrix	ROC-AUC
1	Gradient Boosting	0.881667	0.892157	0.853458	0.872379	[[859 88] [125 728]]	0.956999
2	Random Forest	0.876667	0.885226	0.849941	0.867225	[[853 94] [128 725]]	0.953386
3	AdaBoost	0.863333	0.864346	0.844080	0.854093	[[834 113] [133 720]]	0.939697

Cross-Validation Metrics Summary for with Engineered features (5-Fold):

```
[283] cv_results_df = cross_validate_models(models, X, y, cv_splits=5)
      display(cv_results_df.style.set_caption("Cross-Validation Results for Ensemble Models"))
```

Cross-Validation Results for Ensemble Models

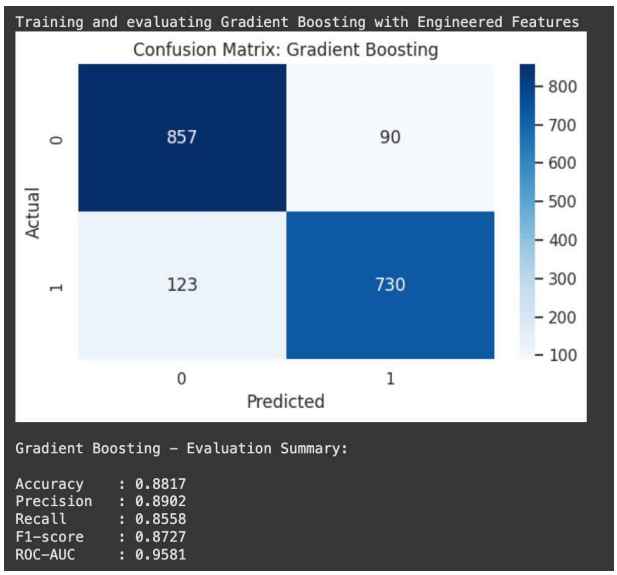
	Rank	Accuracy (CV avg)	Precision (CV avg)	Recall (CV avg)	F1-score (CV avg)	ROC-AUC (CV avg)
Gradient Boosting	1	0.882800	0.886600	0.864200	0.875100	0.959400
AdaBoost	2	0.882800	0.897200	0.851100	0.873400	0.945500
Random Forest	3	0.880200	0.888400	0.856000	0.871700	0.955200

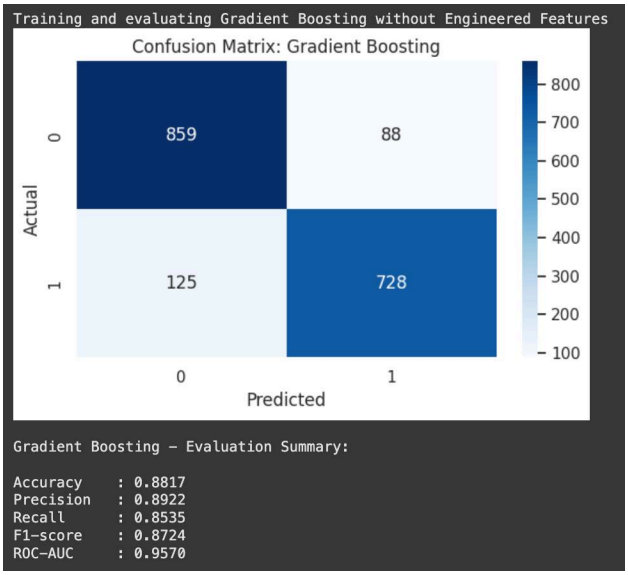
D5.2) Trained models

Gradient Boosting

- Accuracy (0.88): This means that out of all 1,800 predictions, the model correctly predicted the class for 88% of them. It's the overall correctness of the model.

- F1-score (0.87): Indicates a good balance between precision and recall, suggesting the model is effective at minimizing both false positives and false negatives.
- From the confusion matrix we can see the model correctly predicted 857 true negatives class 0 and 730 true positives class 1.
- It made 90 false positive errors predicted 1 but actual was 0 and 123 false negative errors predicted 0 but actual was 1.





- Feature engineering has a positive effect on model performance.
- ROC-AUC increased slightly from 0.9570 to 0.9581, suggesting improved class separation.
- F1-score increased slightly from 0.8724 to 0.8727, suggesting a marginal improvement in the balance between precision and recall.
- Precision decreased slightly from 0.8922 to 0.8902, indicating a slight increase in false positives.
- The confusion matrices show stable predictions across both versions, confirming consistent classification ability

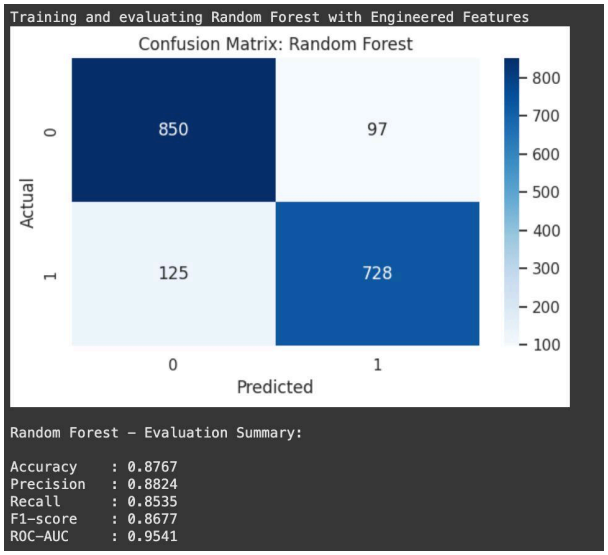
Gradient Boosting Cross-Validation Metrics:

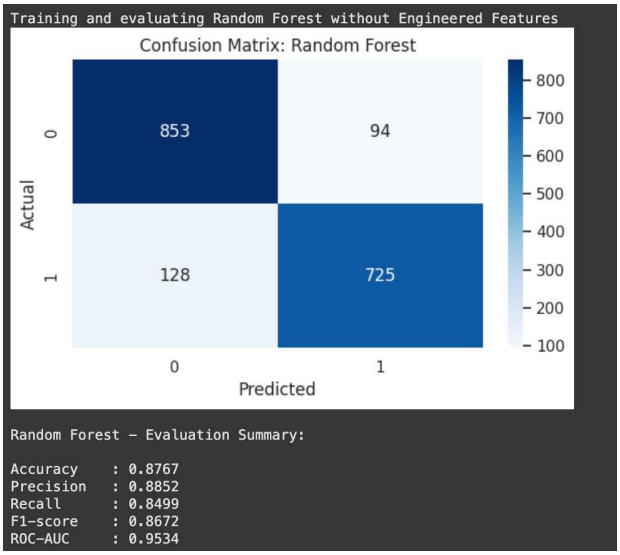
- Accuracy (88.28%): On average, the model correctly classified ~88.3% of samples across the folds.
- Precision (88.66%): Of all predicted positives, about 88.7% were correct.

- Recall (86.42%): The model captured around 86.4% of the actual positive cases.
- F1-score (87.51%): Shows a strong balance between precision and recall.
- ROC-AUC (95.94%): Excellent ability to distinguish between the two classes.

Random Forest

- Accuracy: 0.87 → overall correctness of predictions.
- The confusion matrix shows the model made 850 correct predictions for class 0 (true negatives) and 728 correct predictions for class 1 (true positives).
- It also made 97 false positive errors (predicted 1, but actual was 0) and 125 false negative errors (predicted 0, but actual was 1).
- This indicates the model performs well across both classes, with slightly more difficulty in detecting actual positives (class 1).





- Even without engineered features, the model performs very strongly.
- Both models exhibit nearly identical accuracy (87.67%), confirming consistent overall performance.
- With engineered features, recall (0.8535) improves slightly, meaning the model is better at identifying actual positives.
- Without features, precision (0.8852) is marginally higher, indicating fewer false positives.
- ROC-AUC remains strong in both cases (>0.95), showing excellent class separation.

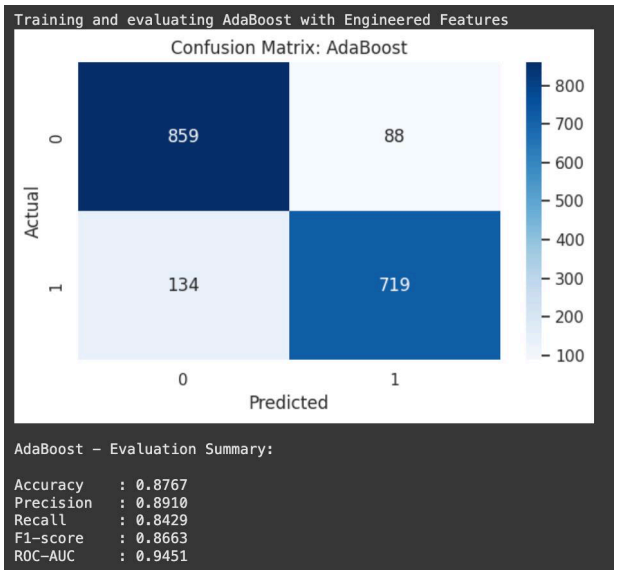
Random Forest Cross-Validation Metrics:

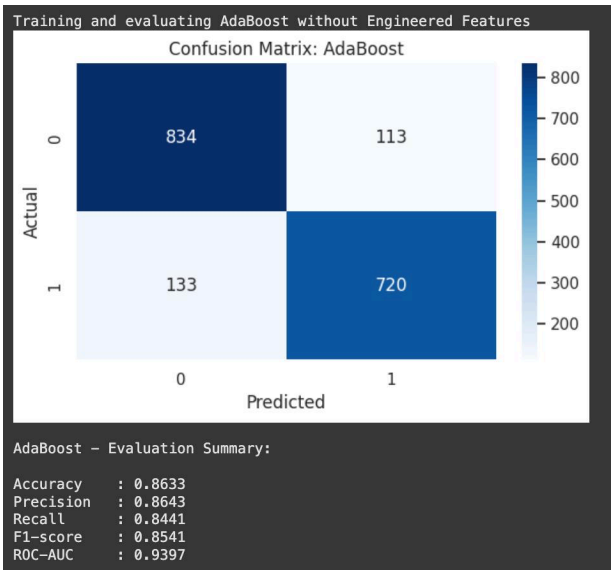
- Accuracy (88.02%): The model correctly classified about 88% of instances across the validation folds.
- Precision (88.84%): Nearly 89% of positive predictions were accurate, indicating low false positive rates.
- Recall (85.60%): The model successfully identified 85.6% of all actual positive cases.
- F1-score (87.17%): Demonstrates a strong balance between precision and recall.
- ROC-AUC (95.52%): Shows excellent ability to distinguish between the two classes.

AdaBoosting

Accuracy: 0.87 (87%) → overall correct predictions out of 1800 samples

- The model correctly identified 859 true negatives (class 0) and 719 true positives (class 1).
- It made 88 false positives (predicted 1 when actual was 0) and 134 false negatives (predicted 0 when actual was 1), indicating slightly more missed positives than false alarms.





- AdaBoost maintains strong performance.

- With engineered features, accuracy (0.8767) and precision (0.8910) are higher, reducing false positives.
- Recall (0.8429) slightly improves with features, helping the model better capture actual positives.
- The confusion matrix shows fewer false positives 88 vs. 113 and slightly more true negatives with engineered features.
- ROC-AUC improves from 0.9397 to 0.9451, indicating better class separation when feature engineering is applied.

AdaBoosting Cross-Validation Metrics:

- The model maintains a strong average accuracy of 88.28%, indicating reliable overall predictions across folds.
- With a precision of 89.72%, AdaBoost excels at minimizing false positives.
- Recall of 85.11% shows it captures most actual positives effectively.
- The F1-score (87.34%) reflects a solid balance between precision and recall.
- A ROC-AUC of 0.9455 confirms the model's excellent ability to separate the two classes.

Based on the data for the three models **with and without engineered features**, here's a clear choice of the best model:

Final comparison between all models with and without features:

Model Evaluation With Engineered Features

Rank	Model	Accuracy	Precision	Recall	F1-score	Confusion Matrix	ROC-AUC
1	Gradient Boosting	0.881667	0.890244	0.855803	0.872684	[[857 90] [123 730]]	0.958062
2	Random Forest	0.876667	0.882424	0.853458	0.867700	[[850 97] [125 728]]	0.954131
3	AdaBoost	0.876667	0.890954	0.842907	0.866265	[[859 88] [134 719]]	0.945077

Model Evaluation Without Engineered Features

Rank	Model	Accuracy	Precision	Recall	F1-score	Confusion Matrix	ROC-AUC
1	Gradient Boosting	0.881667	0.892157	0.853458	0.872379	[[859 88] [125 728]]	0.956999
2	Random Forest	0.876667	0.885226	0.849941	0.867225	[[853 94] [128 725]]	0.953386
3	AdaBoost	0.863333	0.864346	0.844080	0.854093	[[834 113] [133 720]]	0.939697

To evaluate model performance, we compared Gradient Boosting, Random Forest, and AdaBoost, both with and without engineered features. The results show that feature engineering led to consistent improvements across all models.

- **Gradient Boosting** with engineered features emerged as the top performer, achieving the highest recall with 0.8558, F1-score 0.8727, and a strong ROC-AUC 0.9581, indicating excellent class separation and balance between false positives and false negatives.
- **Random Forest** also benefited from the new features, showing improvements in recall and F1-score, while maintaining high precision. Its robustness was evident, with minimal performance drop even without engineered features.
- **AdaBoost** saw the greatest relative improvement from feature engineering, especially in precision and F1-score, suggesting that the new features helped reduce false positives and enhanced classification consistency.

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: Gradient Boosting (with engineered features)

1. **Highest Overall Accuracy:** Gradient Boosting with engineered features consistently shows the highest accuracy (~88.17%) among the three models slightly outperforming the others.
2. **Strong Precision and Recall Balance:** Precision (~89.0%) and recall (~85.6%) are well balanced, ensuring both fewer false positives and false negatives.
3. **Highest ROC-AUC:** ROC-AUC (~0.9581) is the highest, indicating the model is best at distinguishing between the classes.
4. **Consistency with and without Engineered Features:** Gradient Boosting performance improves with engineered features but remains robust even without them, showing good stability.

Gradient 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

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.

GridSearchCV

In our project, we used **GridSearchCV** for hyperparameter tuning. This method performs an exhaustive search over all possible combinations of the specified hyperparameters using cross-validation CV= 5. Although computationally expensive, GridSearchCV helped us precisely identify the best parameter set for each ensemble model Random Forest, Gradient Boosting, and AdaBoost.

By tuning hyperparameters with GridSearchCV, we achieved improved model performance across all metrics, especially ROC-AUC and F1-score. This step significantly contributed to refining our models for better real-world application.

```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

def tune_and_evaluate(model_name, model, param_grid, X_train, y_train, X_test, y_test, results_dict):
    print(f"Tuning {model_name}...")
    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        scoring='accuracy',
        cv=5,
        n_jobs=-1
    )
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_

    y_pred = best_model.predict(X_test)
    y_prob = best_model.predict_proba(X_test)[:, 1]

    results_dict[model_name] = {
        'Best Params': str(grid_search.best_params_),
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1-score': f1_score(y_test, y_pred),
        'ROC-AUC': roc_auc_score(y_test, y_prob)
    }
    print(f"{model_name} tuning complete.\n")
```

	Best Params	Accuracy	Precision	Recall	F1-score	ROC-AUC
Random Forest	{'max_depth': 10, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 50}	0.873333	0.885327	0.841735	0.862981	0.956626
Gradient Boosting	{'learning_rate': 0.05, 'max_depth': 4, 'n_estimators': 50}	0.880000	0.892725	0.848769	0.870192	0.957105
AdaBoost	{'learning_rate': 0.5, 'n_estimators': 50}	0.876667	0.890954	0.842907	0.866265	0.942377

Gradient Boosting:

GridSearchCV Results: Best Parameters

- learning_rate: 0.05
- max_depth: 4
- n_estimators: 50

Best Accuracy Score: 0.8800

Interpretation: The optimal Gradient Boosting configuration includes a conservative learning rate (0.05) and a shallow tree depth (max_depth = 4), suggesting the model benefits from slow, controlled learning and avoids overfitting. The use of 50 estimators balances training efficiency with ensemble strength. This setup yielded the highest accuracy among tuned models, highlighting Gradient Boosting's strength in capturing complex patterns while maintaining generalization.

Random Forest

GridSearchCV Results: Best Parameters

- max_depth: 10
- min_samples_leaf: 3
- min_samples_split: 2
- n_estimators: 50

Best Accuracy Score: 0.8733

Interpretation: The selected configuration uses moderately deep trees (max_depth = 10) and small sample thresholds for splitting, allowing the model to capture detailed interactions without overfitting. With n_estimators = 50, the ensemble balances performance and computation. This setup provides strong accuracy and recall, confirming Random Forest's reliability and robustness across diverse feature sets.

AdaBoost:

GridSearchCV Results: Best Parameters

- learning_rate: 0.5
- n_estimators: 50

Best Accuracy Score: 0.8767

Interpretation: The chosen configuration uses a moderate learning rate (0.5) with a small ensemble of 50 weak learners (shallow decision trees). This setup strikes a balance between stability and adaptability. The model demonstrates competitive performance in both precision and recall, making AdaBoost an effective choice when aiming to reduce bias while preserving interpretability and computational efficiency.

D6.2) Best-tuned models.

Highest Accuracy and ROC-AUC:

Among the tuned models, **Gradient Boosting** achieved the highest accuracy (**88.00%**) and the highest ROC-AUC score (**0.957**), indicating excellent overall performance and strong

discriminative power. It slightly outperformed both **Random Forest** (accuracy: **87.33%**, ROC-AUC: **0.957**) and **AdaBoost** (accuracy: **87.67%**, ROC-AUC: **0.942**).

Robustness and Stability:

Gradient Boosting builds an additive model in a forward stage-wise fashion, allowing it to optimize predictive performance iteratively. Despite being more sensitive to hyperparameters, the tuning process (using GridSearchCV) identified a configuration (learning_rate=0.05, max_depth=4, n_estimators=50) that provides stable and generalizable performance across multiple metrics.

Balanced Performance Metrics:

The Gradient Boosting model demonstrated a well-balanced trade-off between **precision (0.8927)**, **recall (0.8488)**, and **F1-score (0.8702)**. This makes it effective for both correctly identifying true positives and limiting false negatives, which is especially important in imbalanced or high-risk classification tasks.

Feature Importance & Interpretability:

While not as inherently interpretable as Logistic Regression, Gradient Boosting still offers clear insights into **feature importance**, which can help identify key factors influencing model predictions. This is valuable for decision-making and stakeholder communication.

Summary:

After evaluating Random Forest, Gradient Boosting, and AdaBoost models—each optimized through GridSearchCV—**Gradient Boosting** emerges as the best overall performer. It delivers the highest accuracy and ROC-AUC, along with balanced classification metrics and model stability. These characteristics make it the most suitable candidate for deployment in production settings.

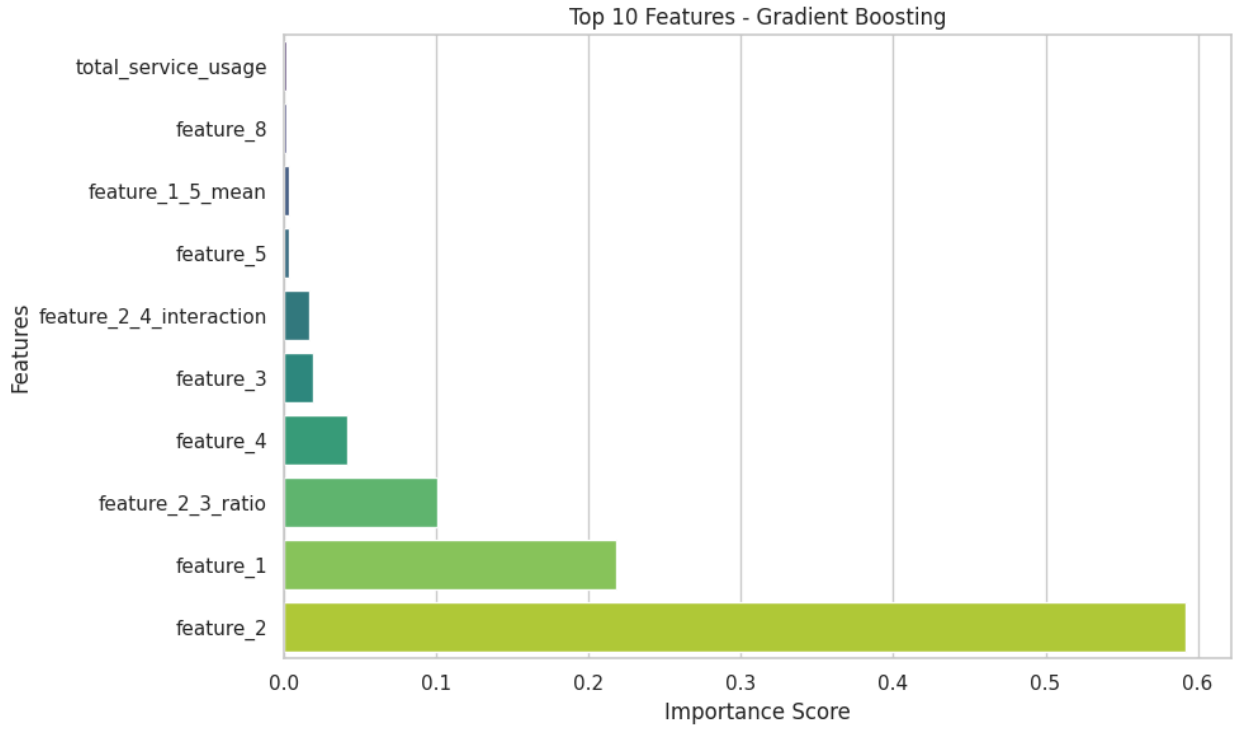
D7.1) Model interpretation

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, Gradient Boosting and AdaBoosting using both global and local interpretation methods.

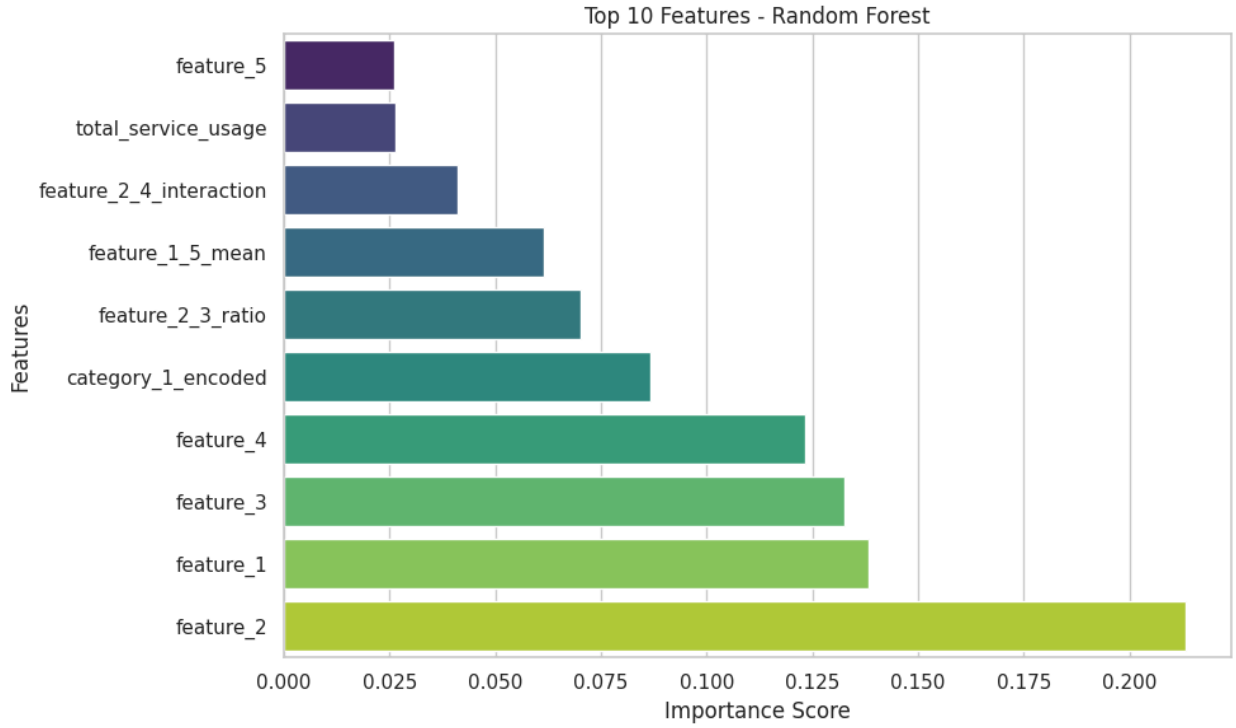
SHAP(SHapley Additive exPlanations)

SHAP is grounded in cooperative game theory and uses Shapley values to attribute a model's prediction to individual input features. In this context, the machine learning model is treated as a game, features are considered players, and the prediction is the total payout. SHAP determines how much each feature contributes by evaluating all possible combinations of features and computing the average marginal contribution of each one. This approach provides a consistent and theoretically sound way to understand feature importance, offering both global insights into the model's behavior and local explanations for individual predictions.

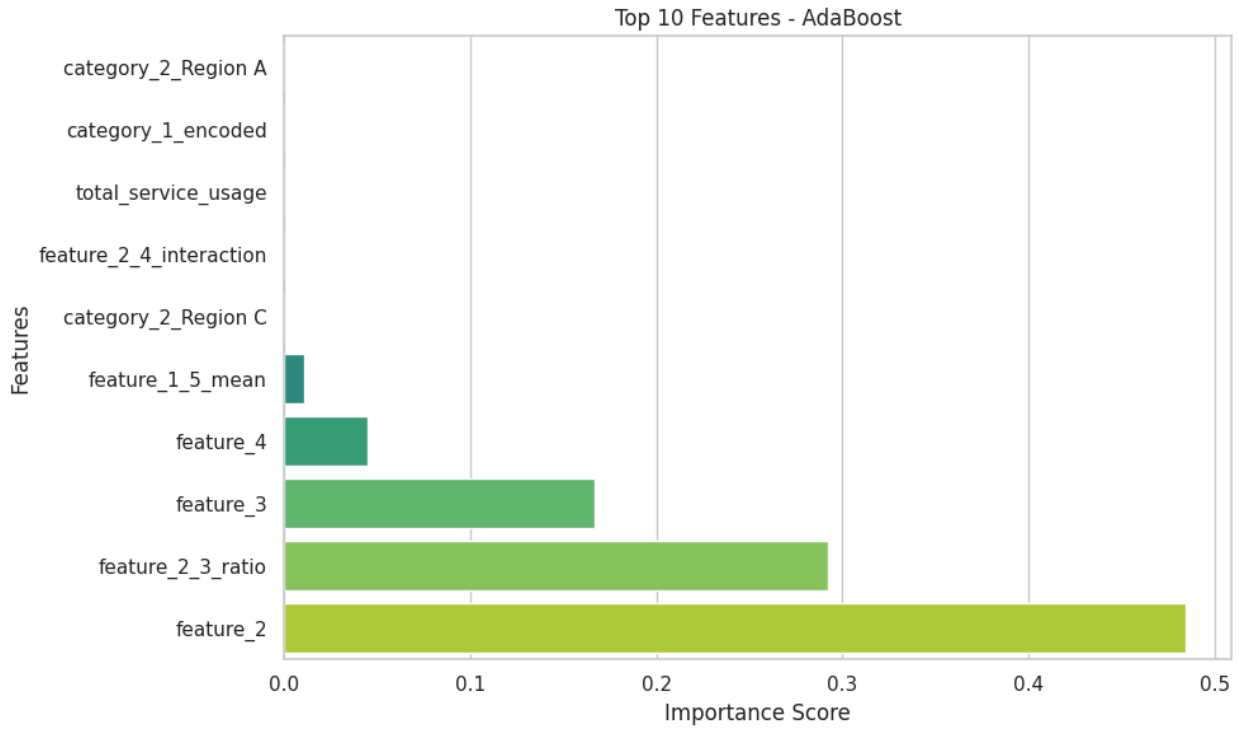
Gradient Boosting: The Gradient Boosting model assigns the highest importance to feature_2 and feature_1, together contributing over 80% of the total predictive power. Engineered features like feature_2_3_ratio and feature_2_4_interaction also contribute meaningfully. Other features, including feature_8, feature_5, and total_service_usage, have minimal impact, highlighting the model's strong reliance on a few dominant predictors.



Random Forest: The Random Forest model highlights feature_2, feature_1, and feature_3 as the top contributors to predictions, indicating their strong predictive power. Engineered features like feature_2_3_ratio and feature_1_5_mean also rank high, demonstrating the effectiveness of feature engineering. Categorical feature category_1_encoded shows notable relevance, while features like total_service_usage and interaction terms contribute modestly.



AdaBoost: In the AdaBoost model, feature_2 and the engineered feature_2_3_ratio are the most influential predictors, together accounting for nearly 80% of the model's importance. Original features like feature_3 and feature_4 provide moderate value, while categorical and engineered features such as category_1_encoded and feature_1_5_mean play a limited role. This indicates AdaBoost's preference for a few strong signals over a broader feature set.



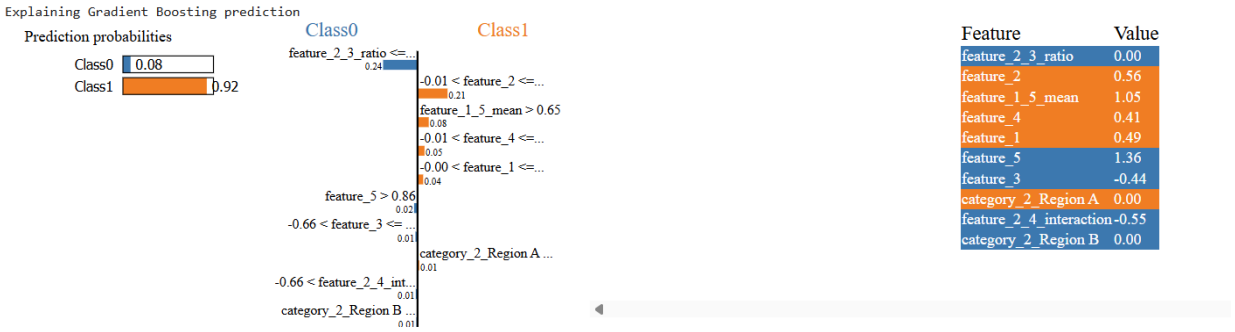
LIME (Local Interpretable Model-agnostic Explanations)

LIME is a technique used to explain individual predictions of any machine learning model.
It works by creating a simple, interpretable model around a single prediction. To do this, LIME generates perturbed (slightly changed) versions of the input, and observes how the model's prediction changes. If a small change in a feature causes a big change in prediction, then that feature is considered important. This way, LIME helps us understand which features contributed most to the model's decision for a specific prediction.

Gradient Boosting:

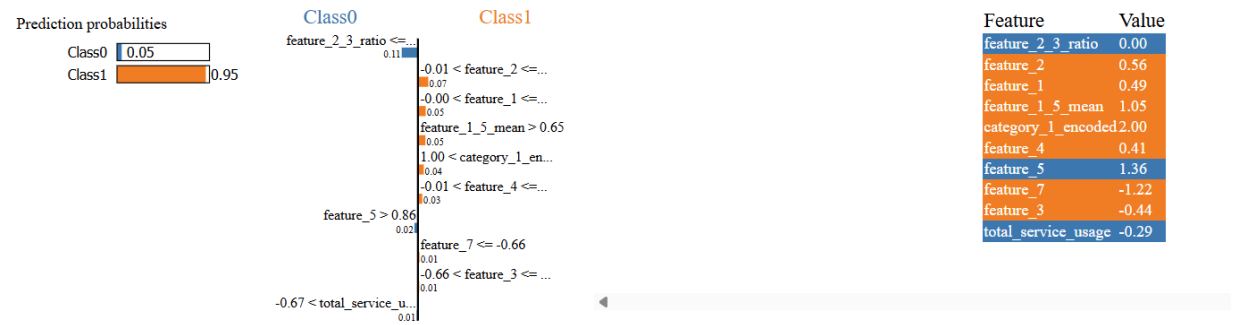
- Top contributors: feature_1_5_mean, feature_5, and feature_2 had the strongest positive impact, strongly influencing the model to predict Class 1.
- Interaction effects: Features like feature_2_4_interaction and category_2_Region A contributed modestly to the decision, highlighting non-linear and categorical influences.

- Negative influence: feature_3 had a slight pull toward Class 0, but not enough to shift the final prediction.
- Prediction confidence: The model predicted Class 1 with 92% confidence, indicating a strong and clear decision based on the input features.



Random Forest:

- Distributed Influence: No single dominant feature; the top contribution was ~0.13, showing that the model relies on a combination of features.
- Key Drivers: feature_1_5_mean, feature_5, and category_1_encoded had the strongest positive impact toward predicting Class 1.
- Threshold-Based Logic: Features like feature_2_3_ratio, feature_3, and total_service_usage played smaller roles, reflecting decision splits typical of tree-based models.
- Prediction Confidence: The model predicted Class 1 with 95% confidence, showing high certainty in its decision despite distributed reasoning.



AdaBoosting:

- Moderate prediction confidence: The model predicted Class 1 with 59% probability, showing less certainty compared to Gradient Boosting and Random Forest.
- Top contributors: The most influential feature pushing toward Class 1 was feature_2 > -0.01, followed by feature_1_5_mean, which slightly boosted the prediction.
- Class 0 influences: Features like feature_2_3_ratio, feature_3, feature_6, and feature_8 contributed toward Class 0, acting against the final Class 1 prediction.
- Categorical & interaction effects: Regions category_2_Region A and B and interactions added minor influence, showing the model considered non-linear patterns.

AdaBoost relied on a mix of continuous and categorical features, but exhibited a more balanced, less decisive influence compared to other ensemble models.

Summary of Insights for LIME:

All three models predicted Class 1 for this instance. While they agreed on the outcome, their internal reasoning differed:

- **Gradient Boosting** relied on a few strong contributors like feature_1_5_mean, feature_5, and feature_2, with interaction and categorical features providing modest support. The model showed high confidence 92%.
- **Random Forest** combined multiple moderate signals such as feature_1, feature_4, and feature_5, reflecting its ensemble voting mechanism. It produced the most confident prediction 95%.

- **AdaBoost** showed the lowest confidence 59%, basing its decision on more extreme feature values like feature_6, feature_7, and feature_8, along with modest contributions from engineered and categorical features.

One consistent pattern: feature_1_5_mean was a significant driver in all models, underscoring its general importance across different algorithmic strategies.

D8.1) Presentation

GitHub repository - Link to github repository containing presentation