

Machine Learning Project

Baani Singh [544768]

Yana Holub [540807]

Edanur Karatas [548829]

D1.1) Dataset exploration

```
[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)
```

- 9000 rows × 11 columns
- Includes numerical, categorical, and target features

Feature Types

- Numerical Features (8): feature_1 to feature_8
- Categorical Features (2):
 - category_1: Ordinal (Low, Below Avg, Above Avg, High)
 - category_2: Nominal (Region A, B, C)
- Target: Binary (0 or 1)

Visual Insights on Feature Distributions

feature_1 & feature_2

- Sharp central peaks and long tails
- Suggest high kurtosis or outliers
- May require clipping or transformation

feature_3, 4, 6, 7, 8

- Approximately Gaussian (bell-shaped) distributions
- Suitable for models assuming normality

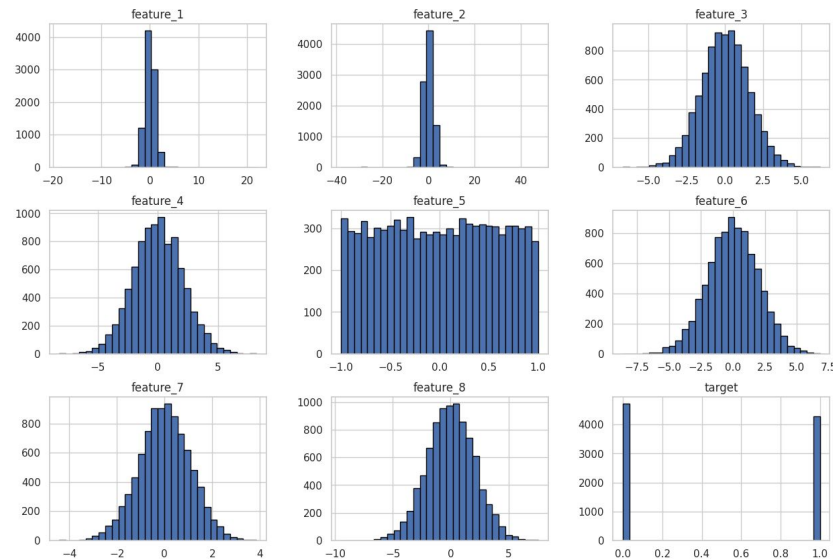
feature_5

- Uniformly distributed
- Low individual predictive power
- May help in interaction terms

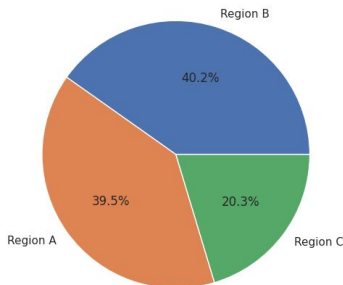
Target variable:

- Target variable Binary (0 and 1)
- Reasonably balanced distribution
- Reduces risk of class imbalance in models

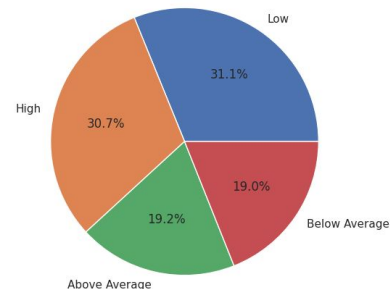
Distribution of Numeric Features



Category 2 Distribution



Category 1 Distribution



Feature Groupings

feature_6, feature_7, feature_8: $r = 0.89-0.97$

- Highly intercorrelated block → likely redundant or reflect a shared latent factor
- Weak or No Correlation

Strong Negative Correlations

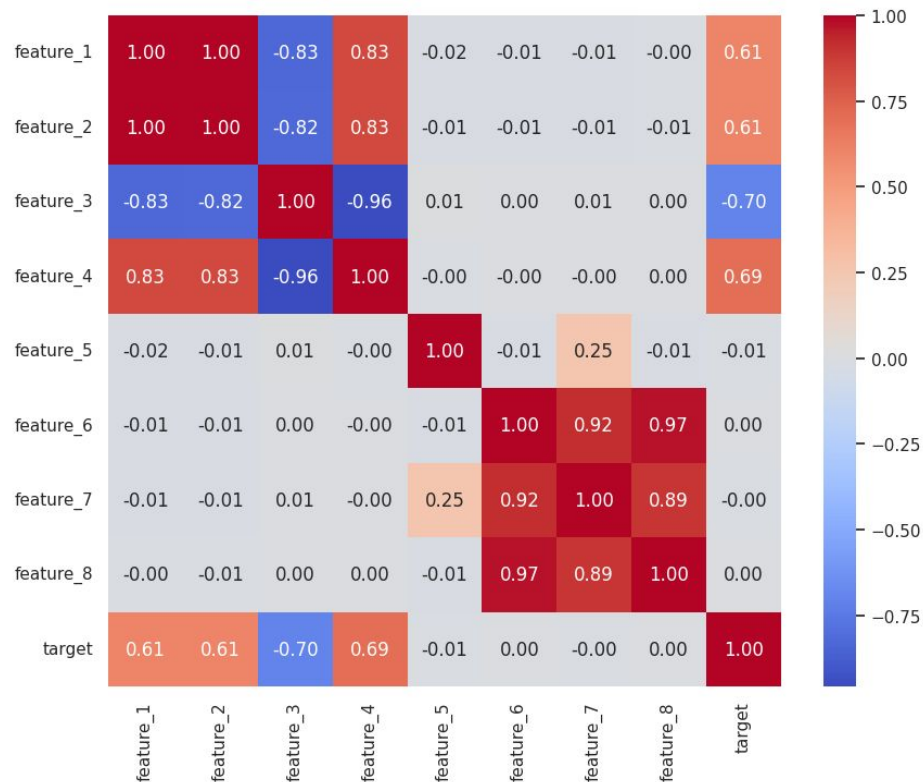
- feature_3 & feature_4: $r = -0.96$
- feature_3 with feature_1/2: $r \approx -0.83$

Strong opposing trends → ideal for ratios or interaction terms

Interaction Strong Positive Correlations

feature_1 & feature_2: $r = +1.00$

- Perfectly correlated thus complete redundancy
- One should be dropped to avoid multicollinearity
- feature_1/2 & feature_4: $r \approx +0.83$



Weak or No Correlation

feature_5

- No significant correlation with any feature or the target
- Limited individual predictive power

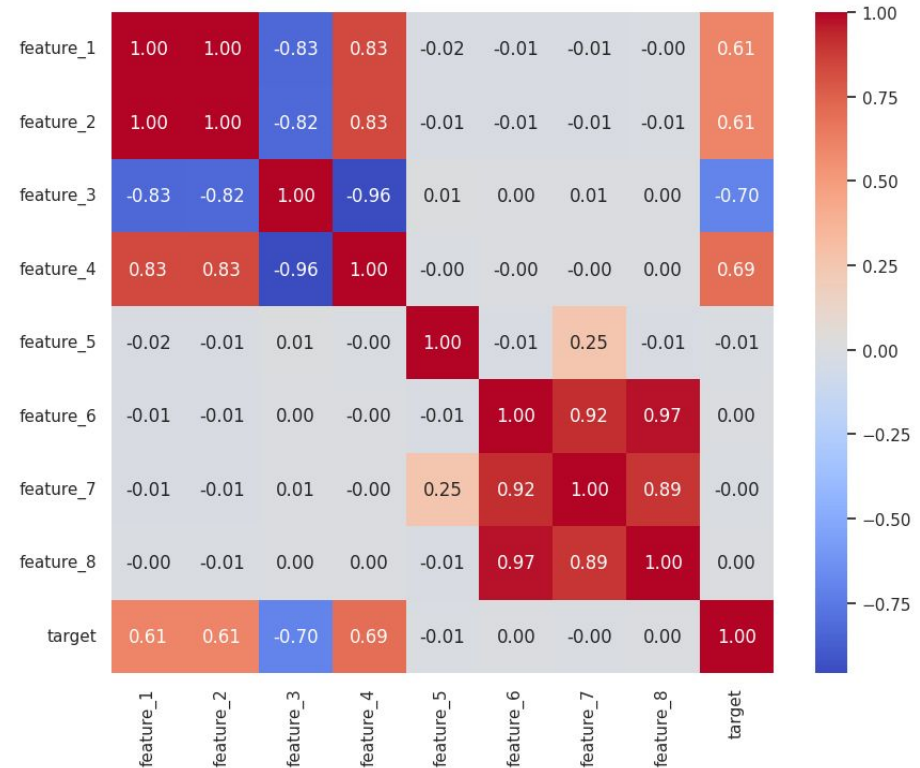
feature_6–8:

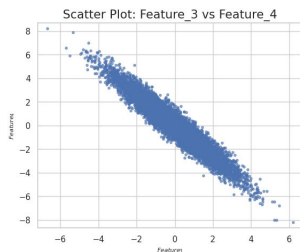
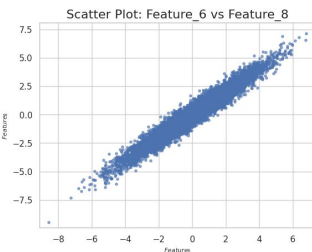
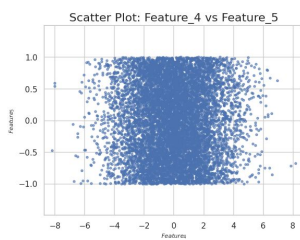
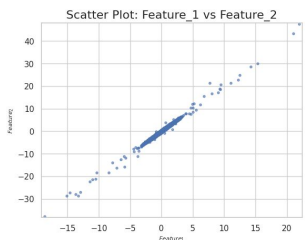
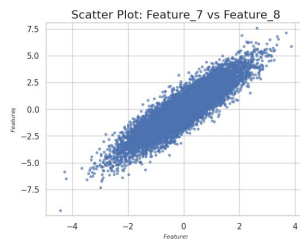
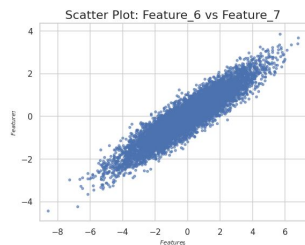
- No direct correlation with target despite being internally consistent

Correlation with Target

- *feature_4*: +0.69
- *feature_1* & *feature_2*: +0.61
- *feature_3*: -0.70

These features are most predictive and should be prioritized in modeling





Strong Positive Correlation

feature_1 vs feature_2

- Forms a perfect diagonal line
- Confirms they are functionally identical ($r = 1.00$)

feature_6 vs feature_7 vs feature_8

- Tight linear clustering
- $r > 0.89 \rightarrow$ likely the same latent structure

Strong Positive 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 others

- Appears as pure scatter
- Evenly spread, no trend \rightarrow matches its uniform distribution

Target Relationships:

feature_1, 2, 3, 4 vs target

- Show weak vertical clustering. Some class separation (0 vs 1)

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 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

Outlier Treatment with IQR Clipping

- Method for handling extreme outliers in numerical features. Preserves data distribution while minimizing distortion.
- Clips extreme values to the nearest valid boundary.
- Detects values far outside the normal range using IQR thresholds.
- No row deletion → All records are retained
- Reduces influence of extreme values on analysis/modeling
- Maintains integrity and overall shape of the dataset

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)
```

Label Encoding – category_1

- Values: "Low", "Below Average", "Above Average", "High"
- Mapped to integers: 0 → 3
- Preserves ordinal structure
- Suitable for models that leverage ranked relationships
- Compact and efficient representation

One-Hot Encoding – category_2

- Values: "Region A", "Region B", "Region C"
- Transformed into separate binary columns
- Handles nominal categories (no inherent order)
- Avoids false ranking of categories
- Increases interpretability for many 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)

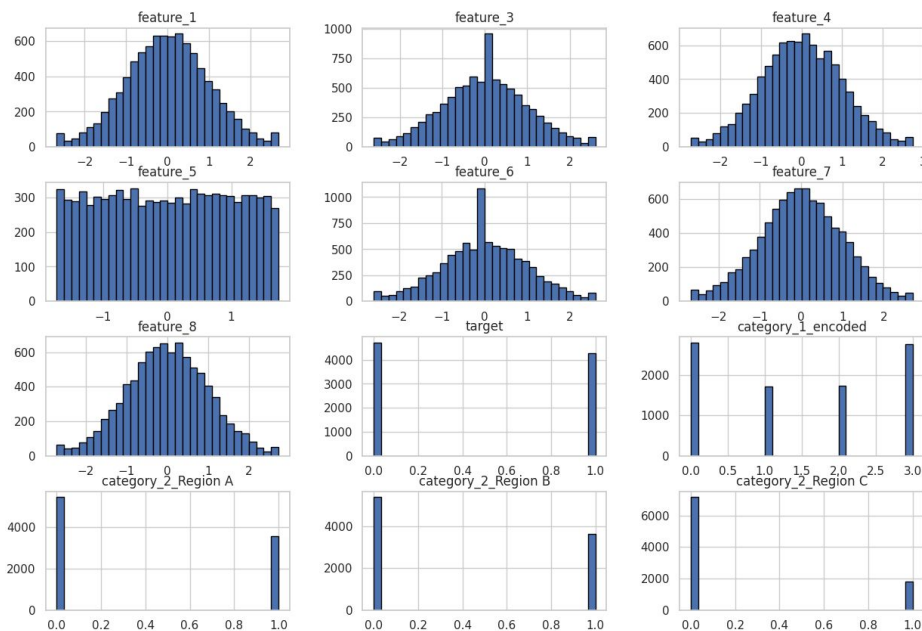
[63] # Display first 10 rows for category_1_encoded and category_2 one-hot columns
df[['category_1_encoded', 'category_2_Region A', 'category_2_Region B', 'category_2_Region C']].head(5)
```


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



Scaling

Rescales features to :

- Mean = 0. Standard Deviation = 1.
- Makes features comparable in: Magnitude and Unit

Distribution Improvements :

- Features like feature_1, feature_3, feature_4, and feature_6 now show: Centered, bell-shaped curves.
- Tighter, more compact tails
- Outlier-prone features now appear more symmetrical and controlled

D3.1) Exploratory Data Analysis

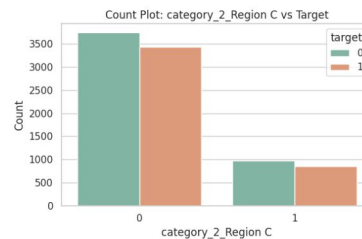
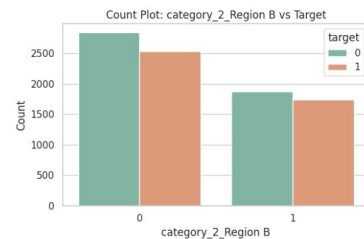
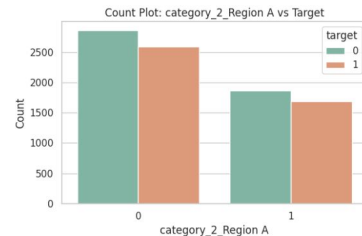
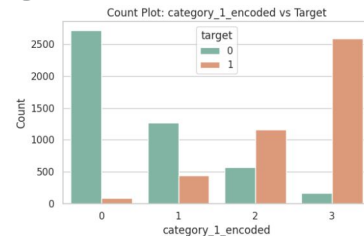
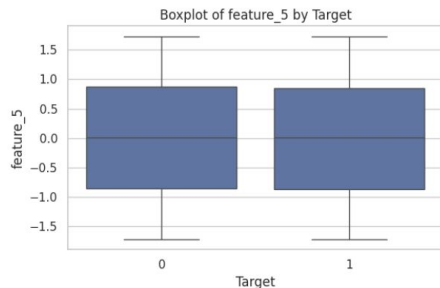
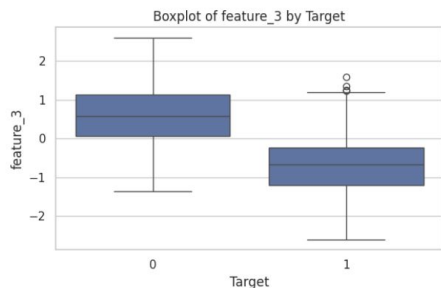
Category 1 (Encoded) vs Target:

Strong separation observed:

- 0 ("Low") → mostly Target = 0
- 3 ("High") → mostly Target = 1
- 1 ("Below Average") & 2 ("Above Average") → more balanced, weaker correlation

Category 2 (Regions A, B, C) vs Target:

- More evenly distributed across target classes
- Minor skews present, but overall weaker predictive value



Feature 3 vs Target:

- Shows strong class separation
- Median for Target = 0 ≈ 0.75
- Median for Target = 1 ≈ -0.5
- Clear distributional difference → highly discriminative feature

Feature 5 vs Target:

- Boxplots for both classes are nearly identical
- Heavy overlap in distributions → low predictive value

T-Tests for Feature Significance

To evaluate feature relevance, we performed Independent Two-Sample T-Tests comparing numeric feature distributions between target classes 0vs1. Used `scipy.stats.ttest_ind` on all numeric features excluding target.

Key Metrics: t-statistic: Measures group difference. p-value: Significance of difference threshold: $p < 0.05$

Results:

- **Significant Features** $p < 0.05$:
feature_1, feature_3, feature_4, category_1_encoded
→ Strongly differentiate between classes; retain for modeling.
- **Not Significant** ($p > 0.05$):
feature_5 to feature_8, one-hot category_2
→ Weak class separation; consider removal or further analysis.

```
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
```

Chi-Square Test of Independence

Designed for categorical features. Based on frequency counts in discrete groups. Not applicable to continuous numerical variables.

Use Case in Our Analysis: Applied to encode categorical features vs. binary target.

Observations

- `Category_1_encoded`: Shows a strong statistical relationship with the target.
- Class distribution varies significantly between target = 0 and 1. Useful predictor for classification is `category_2_Region A, B, C`: p-values > 0.05.
- No significant distribution differences across target classes, less predictive.

Used `chi2_contingency` from `scipy.stats`. Created contingency tables between: Each categorical feature The target variable.

```
# 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

Chi-Square Statistic :

- Small ($O \approx E$): Observed \approx Expected \rightarrow likely no relationship
- Large ($O \neq E$): Observed \neq Expected \rightarrow likely significant relationship

p-value:

- $p < 0.05 \rightarrow$ Significant association with target
- $p \geq 0.05 \rightarrow$ Likely independent of target

D4.1) Feature Engineering

Feature 1:

Feature_1_5_mean: Combines feature_1 to feature_5 into a single averaged feature. Captures the combined trends of a group of strongly correlated variables. Derived From: Correlation heatmap and T-test results.

Potential Impact:

- Enhances signal strength by combining aligned predictors
- Has moderate standalone predictive value
- Performs best when used alongside original features

```
# 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 2:

feature_1 / feature_3: Division of feature_1 → Strong positive correlation with target. feature_3 → Strong negative correlation with target. Combines two predictive signals with opposing directions.

Potential Impact:

- Improves model performance in: SVM, Logistic Regression, and other models benefiting from high contrast and variance.
- Outlier Control: Values clipped at the 1st and 99th percentiles for robustness
- Net Result: Highlights class differences more strongly.
- Contributes to better decision boundaries

```
# 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 3: Interaction Term (feature_1 × feature_4)

Multiplication of two strongly predictive and positively correlated features. Feature_1 × feature_4 .Identified via correlation analysis and t-tests.

Potential Impact :

- Boosts performance in models that don't detect interactions natively, such as: Logistic Regression, SVM.
- Enhances class separability.
- Contributes to richer feature space without model complexity

```
# 3. Interaction 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 4: total_service_usage

Sum of all standardized numerical features. Represents the overall magnitude of features. Captures total engagement across multiple dimensions.

Potential Impact :

- Helps models detect general behavioral patterns.
- Reduces risk of overfitting to individual features.
- Offers a single interpretable signal for usage strength.

```
# 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"]

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

D5.1) Comparing model performances.

Overview of Model Evaluation Process

Model Evaluation Approach

Dataset split: 80% training, 20% testing

Models evaluated:

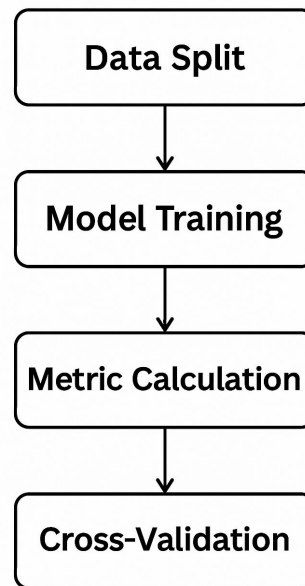
- Logistic Regression
- Random Forest
- Support Vector Machine (SVM)

Evaluation Metrics:

- Accuracy, Precision, Recall, F1-Score, ROC-AUC

5-Fold Cross-Validation: For more robust performance

Evaluation Pipeline



Model Performance Summary

Model Performance without Engineered Features

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	85.56%	85.09%	84.29%	84.69%	93.30%
Random Forest	87.72%	88.54%	85.11%	86.79%	94.96%
SVM	86.56%	85.48%	86.28%	85.88%	93.90%

Model Performance with Engineered Features

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	87.83%	87.38%	87.01%	87.18%	95.35%
Random Forest	88.53%	88.89%	85.42%	87.10%	95.35%
SVM	88.03%	87.5%	86.3%	86.9%	94.46%

Key Insights and Model Selection

Best Model: Random Forest

Best accuracy & ROC-AUC with engineered features

Balanced precision & recall

Robust with or without engineered features

Conclusion: Random Forest is optimal for deployment

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

Cross-Validation Metrics Summary

Best Model: Random Forest

- All three models show strong generalization with high ROC-AUC (>94%).
- Random Forest has the highest precision, making it ideal if false positives are costly.
- SVM offers balanced performance, slightly outperforming others in recall.

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

D6.1) Hyperparameter tuning

Hyperparameter Tuning Strategy

Tuning Methods Used

RandomizedSearchCV: Efficient, broad search

GridSearchCV: Exhaustive, precise tuning

Applied to all 3 models

Evaluated with cross-validation

```
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 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 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
```

Tuning Results Summary

Best Hyperparameters & Scores

Model	Method	Best Params (shortened)	Accuracy
Logistic Regression	GridSearchCV	C=10, penalty=L2, solver=liblinear	87.89%
Random Forest	GridSearchCV	n=25, depth=10, leaf=3	88.53%
SVM	GridSearchCV	C=10, kernel=rbf, gamma=scale	88.03%

D6.2) Best-tuned models.

Final Tuning Takeaways

Why Random Forest Wins

- Best accuracy (88.53%)
- ROC-AUC (~0.953)
- Consistent across both tuning methods
- Strong generalization and low variance
- Feature importance available for interpretation

```
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
```

D7.1) Model interpretation

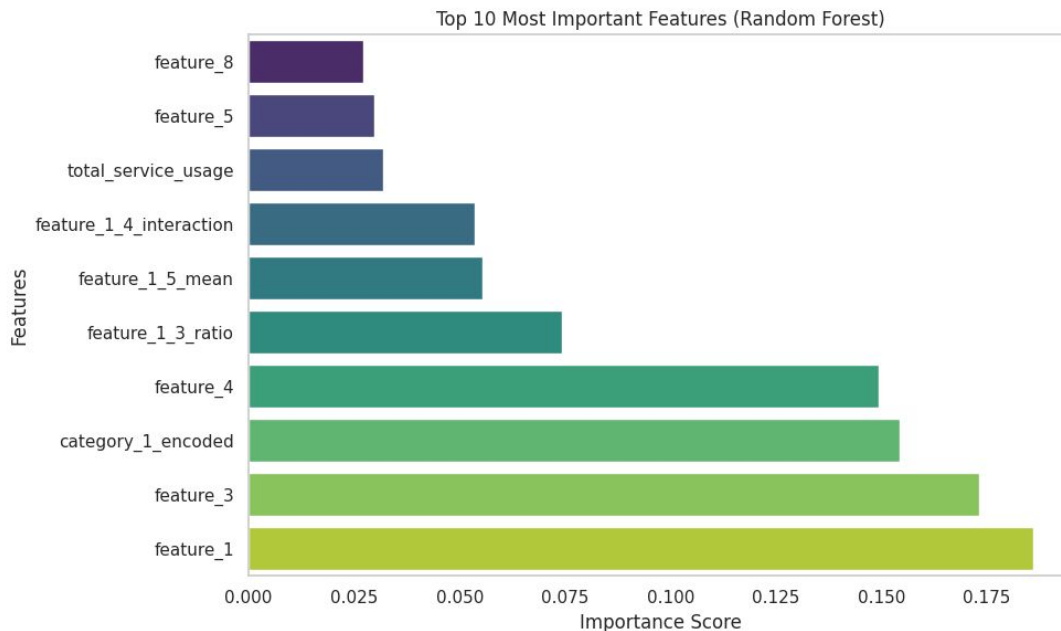
Global Interpretation – Random Forest

Feature Importance (Global View)

Top features: feature_1, feature_3,
category_1_encoded

Feature_1 = most predictive in all models

Less important: feature_8, feature_5



Local Interpretation (LIME)

Local Explanations with LIME

Logistic Regression: strong reliance on feature_1

Random Forest: distributed reasoning across features

SVM: margin-based logic, lower confidence

All models agreed on prediction for the sample case

