Machine Learning Project

Baani Singh [544768]

Yana Holub [540807]

Edanur Karatas [548829]

D1.1) Dataset exploration

<u>Dataset Composition:</u> 9000 rows × 11 columns

- 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)

<u>Visual Insights on Feature Distributions</u>

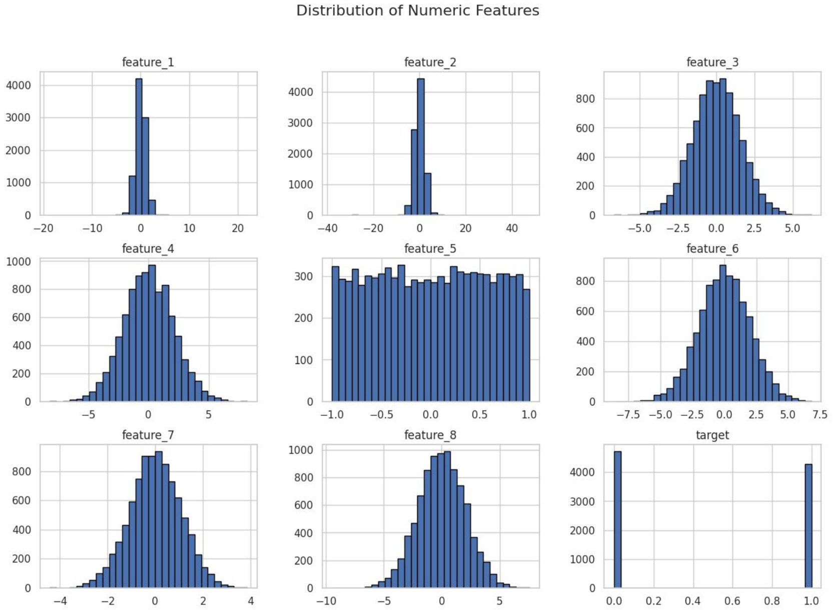
- *Feature_1 & feature_2:* Sharp central peaks and long tails, high kurtosis or outliers, requires clipping
- *Feature_3, 4, 6, 7, 8:* Approximately Gaussian (bell-shaped) distributions, suitable for models assuming normality
- *Feature_5:* Uniformly distributed, low individual predictive power
- *Target variable:*Reasonably balanced distribution, reduces risk of class imbalance in models

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

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	category_1	category_2	target
0	0.496714	1.146509	-0.648521	0.833005	0.784920	-2.209437	-1.300105	-2.242241	Above Average	Region C	1
1	-0.138264	-0.061846	NaN	0.403768	0.704674	-2.498565	-1.339227	-1.942298	Below Average	Region A	0
2	0.647689	1.395115	-0.764126	1.708266	-0.250029	1.956259	1.190238	1.503559	High	Region C	1
3	1.523030	2.657560	-2.461653	2.649051	0.882201	3.445638	2.120913	3.409035	High	Region B	1
4	-0.234153	-0.499391	0.576097	-0.441656	0.610601	0.211425	0.935759	-0.401463	Below Average	Region C	0
5	-0.234137	-0.699415	0.268972	-0.702775	0.702283	-0.332383	0.453958	-0.826721	Below Average	Region A	0
6	1.579213	3.117904	-2.885133	3.312708	0.864708	2.045283	1.531547	1.771851	High	Region A	1
7	0.767435	1.730870	-1.445877	1.411070	0.874003	0.674730	0.812931	1.489838	High	Region A	1
8	-0.469474	-0.877919	0.575087	-0.532917	-0.519870	NaN	-3.002925	-4.779960	Below Average	Region A	0
9	0.542560	1.314738	-0.403383	1.456165	-0.744625	1.987345	0.431966	3.309386	High	Region C	1

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
print(df.shape)
(9000, 11)
```



Categorical data distribution:

- *Category 1:* The distribution is relatively balanced, with "Low" and "High" categories dominating, and "Above Average" and "Below Average" making up the rest.
- *Category 2:* Region B and Region A have nearly equal representation (~40%), while Region C accounts for a smaller portion (~20%).

Strong Positive Correlation

- *feature_1 vs feature_2:* Forms a perfect diagonal line. They are functionally identical (r = 1.00)
- *feature_6 vs feature_7 vs feature_8:* Tight linear clustering, r > 0.89 → likely the same latent structure

Strong Positive Correlation

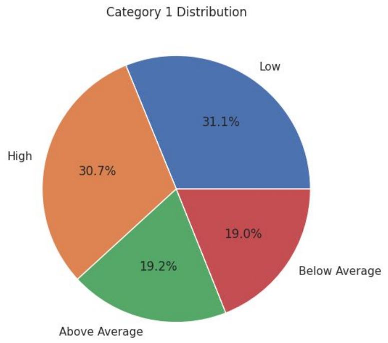
feature_3 vs feature_4: Very tight downward-sloping line, confirms r = –0.96, perfect for interaction terms or ratios

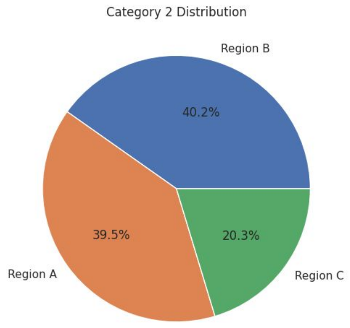
No Correlation / No Pattern:

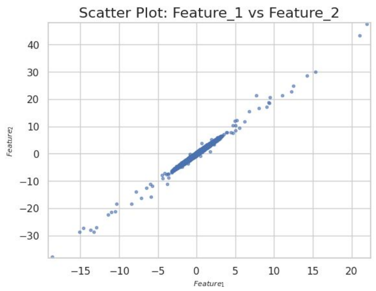
feature_5 with others: Appears as pure scatter. Evenly spread, no trend → matches its uniform distribution

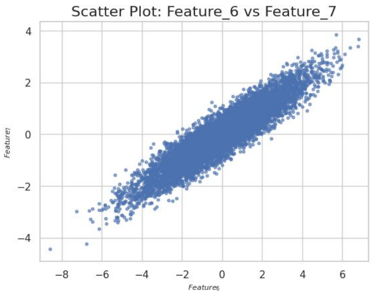
Target Relationships:

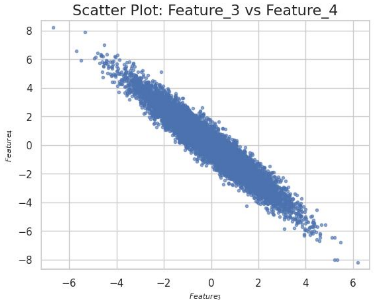
feature_1, 2, 3, 4 vs target: Show weak vertical clustering. Some class separation (0 vs 1)

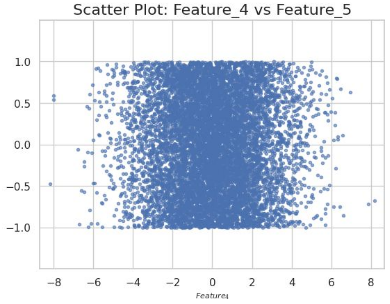












Feature Groupings

feature_6, feature_7, feature_8: r = 0.89–0.97

- Highly intercorrelated block → 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 = –0.83

Strong opposing trends → ideal for ratios or interaction terms

Interaction Strong Positive Correlations

feature_1 & feature_2: r = +1.00

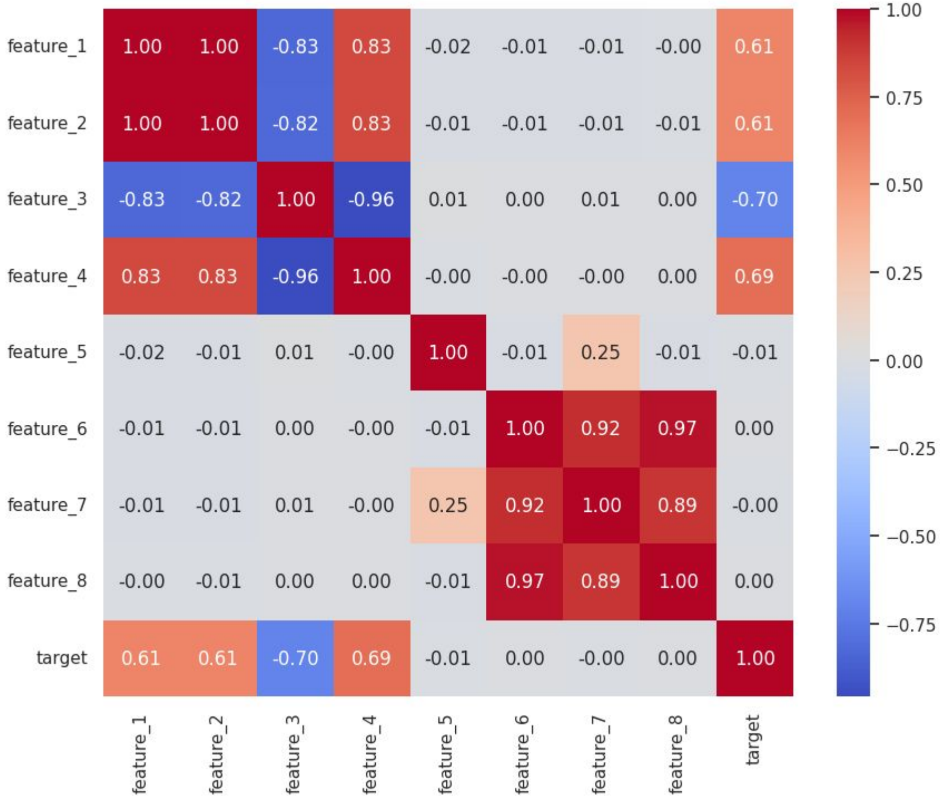
- Perfectly correlated
- feature_1/2 & feature_4: r = +0.83

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: These features have most predictive power

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



D2.1) Data Preprocessing

<u>Mean imputation for missing values:</u>

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 (vs 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%)

```
[146] missing_count = df.isnull().sum()
      missing_percent = (missing_count / len(df)) * 100

      # 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]

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

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

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

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

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

```
[149] 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
dtype: int64
```

Outlier Treatment with IQR Clipping

- Method for handling extreme outliers in numerical features.
- 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

However this does it does alter the original data distribution especially near the edges, where clipped values accumulate.We can observer this in the final data distribution graphs as the edges are raised due in comparison to the original data distribution.

Identify and Treat Outliers -> IQR Clipping

```
[150] # Define numeric features to clip
numerical_columns = ['feature_1', 'feature_2', '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

    # Count how many values would be clipped
    original = df[col]
    n_clipped = ((original < lower) | (original > upper)).sum()

    # Apply clipping
    df[col] = np.clip(original, lower, upper)

    print(f"{col}: {n_clipped} values clipped")
```

```
feature_1: 113 values clipped
feature_2: 106 values clipped
feature_3: 107 values clipped
feature_4: 71 values clipped
feature_5: 0 values clipped
feature_6: 119 values clipped
feature_7: 69 values clipped
feature_8: 61 values clipped
```

Encoding

Label Encoding – category_1

- Values: "Low", "Below Average", "Above Average", "High"
- Mapped to integers: 0, 1, 2 ,3
- Preserves ordinal structure
- Suitable for models which it is useful to have inherent order

One-Hot Encoding – category_2

- Values: "Region A", "Region B", "Region C"
- Transformed into separate binary columns
- Handles nominal categories (no inherent order)
- Avoids unnecessary order

Encoding -> One-Hot & Label Enc.

```
[151] # 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)

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

[153] # 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)

[154] # 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_R
```

	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

```
[155] print("Unique values in encoded columns:\n", df[['category_1_encoded']].nunique())

Unique values in encoded columns:
 category_1_encoded    4
dtype: int64
```

Scaling

Rescales features to have:

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

Distribution Improvements :

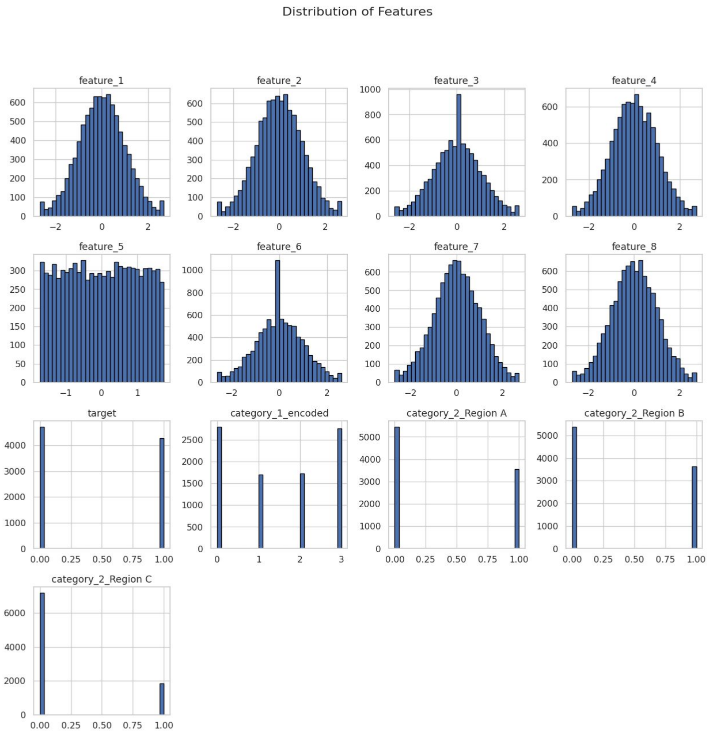
- Features like feature_1, feature_3, feature_4, and feature_6 now show: Centered, bell-shaped curves.
- Tighter, more compact tails
- Outlier-prone features now appear more symmetrical and controlled

Scaling -> StandardScaler

```
from sklearn.preprocessing import StandardScaler

numerical_columns = ['feature_1','feature_2','feature_3','feature_3', 'feature_4',
                     'feature_5', 'feature_6', 'feature_7', 'feature_8']
scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

```
df.hist(bins=30, figsize=(15,10), edgecolor='black')
plt.suptitle('Distribution of Features', fontsize=16)
plt.show()
```



D3.1) Exploratory Data Analysis

<u>Category 1 (Encoded) vs Target:</u> Strong separation observed:

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

<u>Category 2 (Regions A, B, C) vs Target:</u>

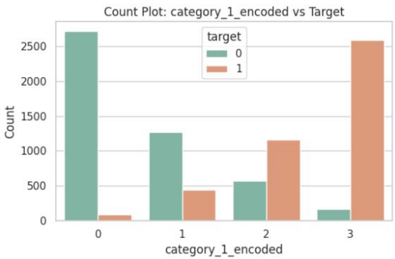
- More evenly distributed across target classes
- Minor skews present towards target 0, but overall weaker predictive value

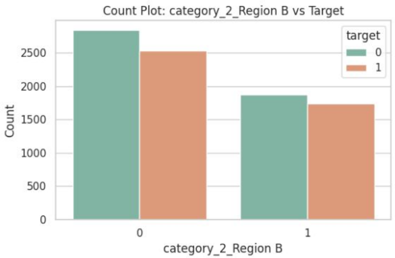
<u>Feature 3 vs Target:</u> Shows strong class separation

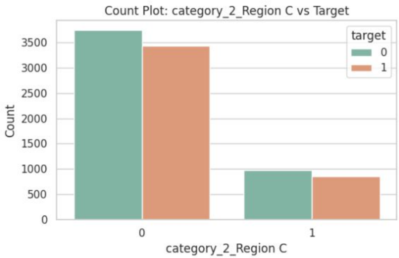
- Median for Target = 0 ≈ 0.75 vs Median for Target = 1 < –0.5
- Clear distributional difference → highly discriminative feature

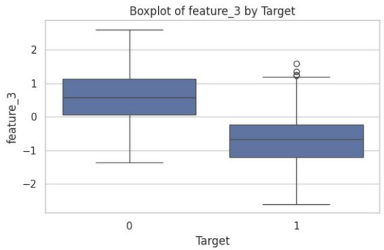
<u>Feature 5 vs Target:</u>

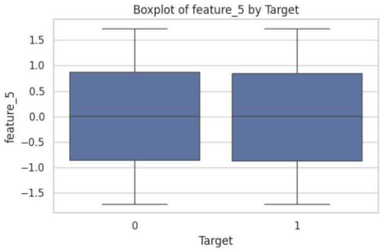
- Boxplots for both classes are nearly identical
- Heavy overlap in distributions → low predictive value











T-Tests for Feature Significance

To check which features are important, we compared the average values of each numeric feature between the two target groups (Class 0 and Class 1) using a T-Test. We used scipy.stats.ttest_ind on all numeric features except the 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
- **Not Significant** ($p > 0.05$):
  feature_5 to feature_8, one-hot category_2
  → Weak class separation

This confirm what we observed from the bar chart(category 2 is a wear predictor) and box plots(feature 3 is a strong predictor and feature 5 is a weak one) in the previous slide.

T-test

```
[165] 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
```

<u>Chi-Square Test of Independence:</u> 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.

<u>Observations</u>

- 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.

Chi-Square statistic: Measures how expectations compare to actual observed frequencies.

-χ^2 is small ($O \sim E$) → Observed data fits expected data = likely no relationship if

-χ^2 is large ($O \neq E$) → there is a big dif

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.

```
[260] # Chi-square test for categorical features
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
```

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:

- 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.

Feature 2:

feature_2 / feature_3: Division of feature_2 → Strong positive correlation with target. feature_3 → Strong negative correlation with target. Combines two predictive signals with opposing directions.

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.

```
# 1.Sum of Most Correlated Features
df['feature_1_5_mean'] = df[['feature_1', 'feature_2','feature_3', 'feature_4', 'feature_5']].

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

```
# 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 3: Interaction Term (feature_2 × feature_4)

Multiplication of two strongly predictive and positively correlated features. Feature_2 × feature_4 .Identified via correlation analysis and t-tests.

Potential Impact :

- Aligned with correlation structure: Since feature_2 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.
- 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.

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 :

- 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.

```
# 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())
```

```
# 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["featur

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

D5.1) Comparing model performance

Overview of Model Evaluation Process

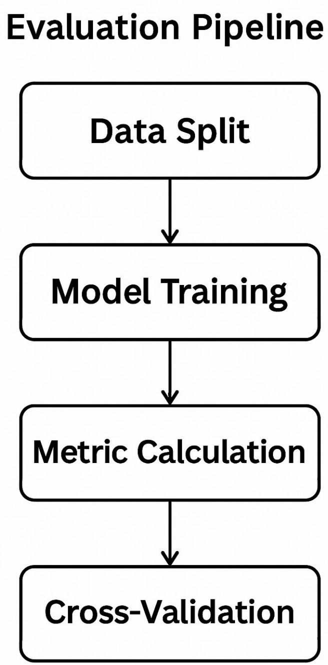
Model Evaluation Approach

Dataset split: 80% training, 20% testing
Models evaluated:

- Gradient Boosting
- Random Forest
- AdaBoost

Evaluation Metrics:

- Accuracy, Precision, Recall, F1-Score, ROC-AUC
- 5-Fold Cross-Validation: For more robust performance




```python
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, classification_report, roc_curve
)

def train_evaluate_plot(model, X_train, y_train, X_test, y_test, model_name='Model', plot_roc=True, plot_confusion=True):
    """
    Train the model, evaluate metrics, and plot confusion matrix and ROC curve.

    Args:
        model: scikit-learn estimator
        X_train, y_train: training data
        X_test, y_test: testing data
        model_name: str, label for plots
        plot_roc: bool, whether to plot ROC curve
    Returns:
        dict of metrics
    """

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, 'predict_proba') else None

    # Metrics calculation
    metrics = {
        '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),
        'Confusion Matrix': confusion_matrix(y_test, y_pred),
        # 'Classification Report': classification_report(y_test, y_pred, output_dict=True)
    }
    if y_proba is not None:
        try:
            metrics['ROC-AUC'] = roc_auc_score(y_test, y_proba)
        except:
            metrics['ROC-AUC'] = None
    else:
        metrics['ROC-AUC'] = None

    # Plot confusion matrix
    if plot_confusion:
        plt.figure(figsize=(6, 4))
        sns.heatmap(metrics['Confusion Matrix'], annot=True, fmt='d', cmap='Blues')
        plt.title(f'Confusion Matrix: {model_name}')
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.tight_layout()
        plt.show()

    # Print metrics
    print(f"\n{model_name} - Evaluation Summary:\n")
    for key, value in metrics.items():
        if key != 'Confusion Matrix':
            if isinstance(value, dict):
                print(f"\n{key}:\n")
                print(classification_report(y_test, y_pred))
            elif value is not None:
                print(f"{key:<12}: {value:.4f}")
            else:
                print(f"{key:<12}: N/A")

    return metrics
```

```python
results_dict = {}
for name, model in models.items():
    print(f"\nTraining and evaluating {name} with Engineered Features")
    metrics = train_evaluate_plot(model, X_train, y_train, X_test, y_test, model_name=name)
    results_dict[name] = metrics
```

Key Insights and Model Selection

Model: Gradient Boosting

Accuracy: 88.00%

Strong overall prediction performance across 1,800 instances.

Confusion Matrix Insights:

- **True Negatives (857)** – Correctly predicted class 0
- **True Positives (730)** – Correctly predicted class 1
- **False Positives (90)** – Predicted 1, actual was 0
- **False Negatives (123)** – Predicted 0, actual was 1

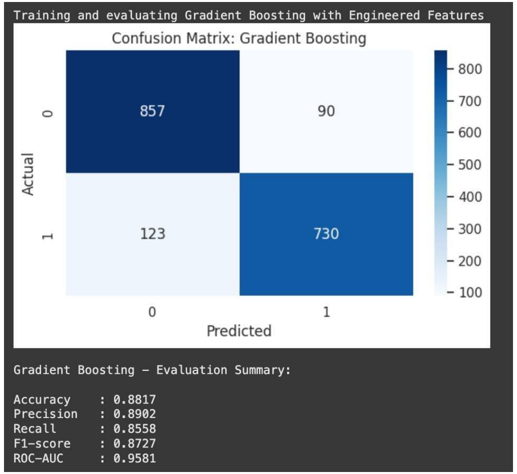
Balanced performance with slightly fewer false alarms and missed positives than Random Forest.

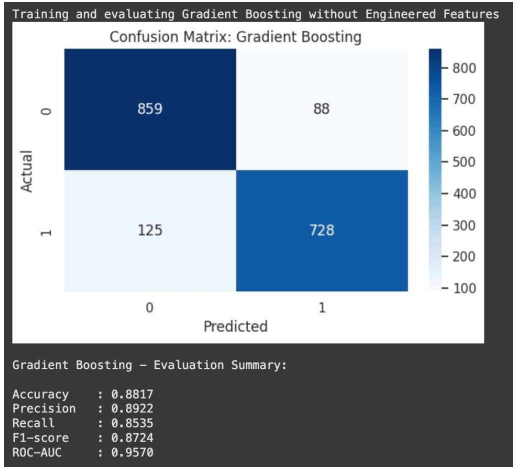
With Engineered Features:

- **Recall improves** to **0.8488**, enhancing detection of positives
- **ROC-AUC rises** to **0.9581**, showing improved class separation
- **F1-score improves slightly** (from 0.8724 → 0.8727)
- **Minor precision drop** (0.8922 → 0.8902), indicating a few more false positives
- Confusion matrix remains stable, confirming consistent prediction behavior

Conclusion:

Gradient Boosting offers **the highest accuracy and ROC-AUC**, with strong, stable performance. Feature engineering yields slight gains—useful for high-precision tasks.





Key Insights and Model Selection

Model: Random Forest

Accuracy: 87.67%

Indicates strong overall prediction performance across both classes.

Confusion Matrix Insights:

- **True Negatives (850):** Correctly identified class 0 instances
- **True Positives (728):** Correctly identified class 1 instances
- **False Positives (97):** Incorrectly predicted positive when actual was negative
- **False Negatives (125):** Missed actual positives

Slight bias toward class 0, but still balanced and reliable.

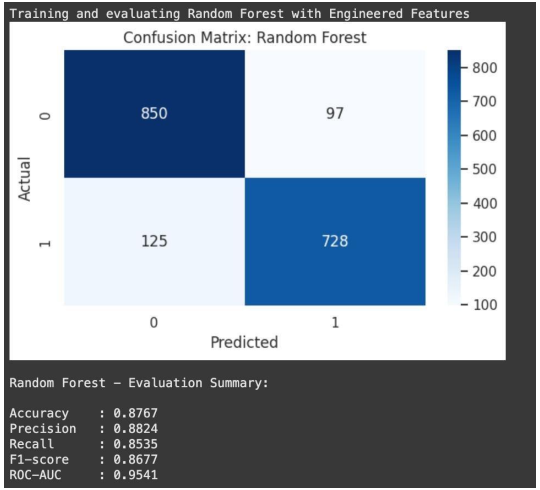
With Engineered Features:

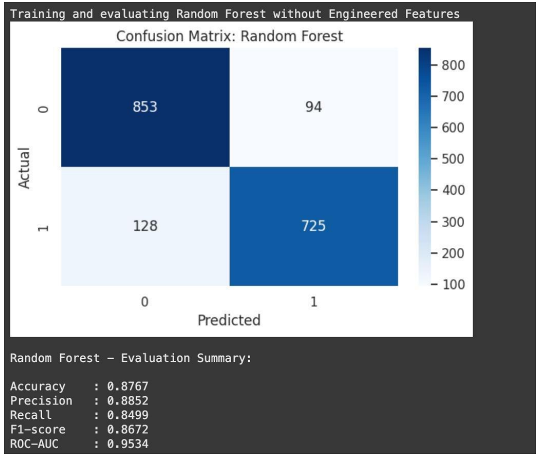
- **Recall improves** to **0.8535**, boosting the model's ability to detect true positives
- **ROC-AUC remains high** (>0.95), indicating excellent separation between classes
- Marginal trade-off in precision, but **F1-score remains strong**, reflecting a good balance

Conclusion:

Random Forest demonstrates **robust, consistent performance**.

Engineered features enhance sensitivity (recall) with minimal trade-offs, making it well-suited for real-world deployment where missing positives is critical.





Key Insights and Model Selection

Model: AdaBoost

Accuracy: 87.67%

Confusion Matrix Insights:

- **True Negatives (859)** – Correctly predicted class 0
- **True Positives (719)** – Correctly predicted class 1
- **False Positives (88)** – Predicted 1, actual was 0
- **False Negatives (134)** – Predicted 0, actual was 1

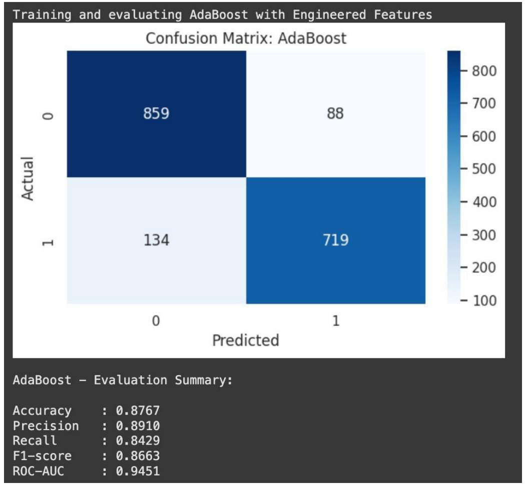
Slightly more missed positives, but fewer false alarms compared to other models.

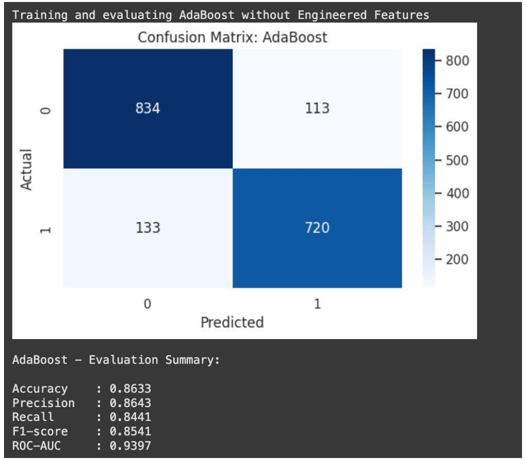
With Engineered Features:

- **Precision increases** to **0.8910**, reducing false positives
- **Recall improves** to **0.8429**, boosting positive detection
- **ROC-AUC improves** from 0.9397 → **0.9451**, enhancing class separation
- Confusion matrix shows **fewer false positives**, indicating cleaner predictions

Conclusion:

AdaBoost offers **reliable and balanced performance**. With engineered features, it **reduces false positives** and slightly improves recall, making it a solid choice for scenarios prioritizing **precision and interpretability**.





Model Performance Summary

Model Performance with Engineered Features
Model Performance without 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 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

Best Model Gradient Boosting:

Gradient Boosting ranks highest overall, with the best balance across metrics and strongest class separation (ROC-AUC).

AdaBoost shows highest precision, minimizing false positives.

Random Forest remains competitive with strong recall and consistent performance

```
[97] 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	0.883000	0.885600	0.866100	0.875600	0.959400
AdaBoost	2	0.882100	0.895900	0.851100	0.872800	0.945400
Random Forest	3	0.880300	0.887900	0.857000	0.871900	0.955600

D6) Hyperparameter tuning & Best-tuned models

Hyperparameter Tuning Strategy

Tuning Methods Used

GridSearchCV: Exhaustive, precise tuning

Applied to all 3 models

Evaluated with cross-validation

Final Tuning Takeaways

Why Gradient Boosting Wins

Among the tuned models,

Gradient Boosting achieved the highest accuracy (**88.00%**) and the highest ROC-AUC score (**0.957**). It slightly outperformed both **Random Forest** (accuracy: **87.33%**, ROC-AUC: **0.957**) and **AdaBoost** (accuracy: **87.67%**, ROC-AUC: **0.942**).

	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

D7.1) Model interpretation

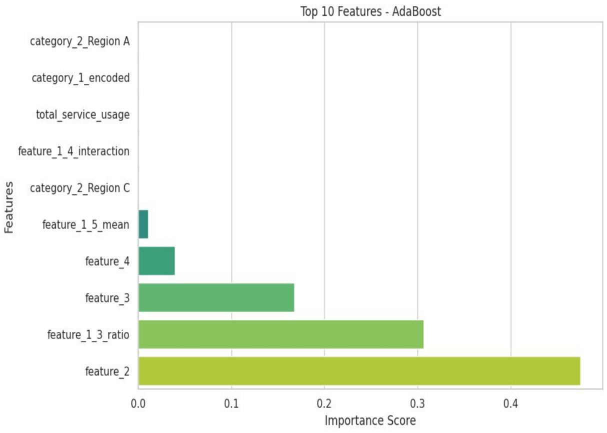
Global Interpretation

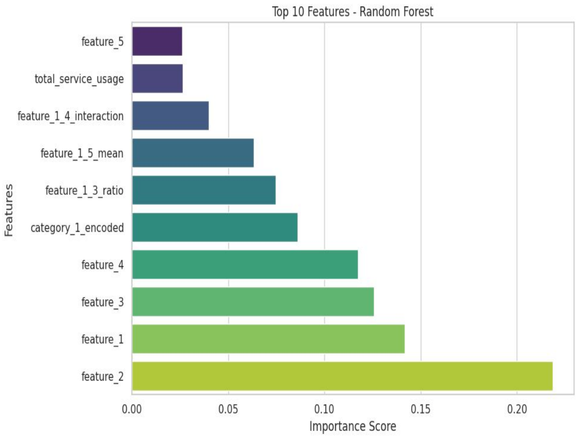
Feature Importance (SHAP)

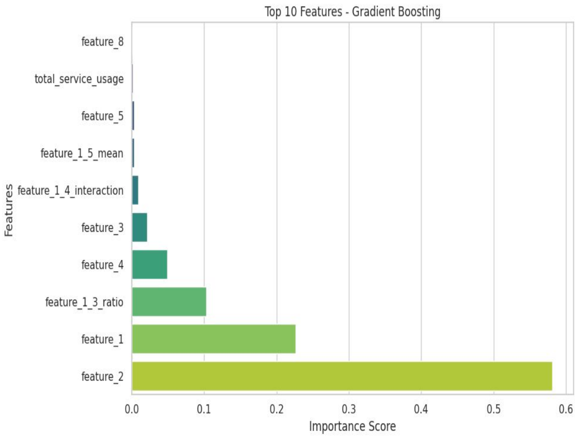
Top features: *feature_2*
Less important: *feature_1_5_mean*

Top features: *feature_2*
Less important: *feature_5, total_service_usage*

Top features: *feature_2*
Less important: *fearure_8, total_service_usage , fearure_5, feature_1_5_mean*







Local Interpretation (LIME)

Local Explanations with LIME

All models predicted Class 1, but used different reasoning:

- **Gradient Boosting** relied on strong features: feature_1_5_mean, feature_5, feature_2; confidence: 92%
- **Random Forest** used multiple moderate features like feature_1, feature_4, feature_5; highest confidence: 95%
- **AdaBoost** was least confident 59%, driven by extreme values in feature_6, feature_7, feature_8

feature_1_5_mean was a key driver across all models.