University of Messina

Bachelor's in Data Analysis

Academic year - 2024/2025

Machine Learning Project Report

Students:

Pham Gia Khiem - 551026 (Vietnam)

Mohammed Hassan-541140 (Bangladesh)

Fazlur Rahman- 541927 (Bangladesh)

Supervisor:

Prof. Giacomo Fiumara

List of Deliverables

0.0 Pictures of Passports

0.1 Google Collab Notebook

1.1. Dataset Exploration

2.1. Data Preprocessing

2.2. A Cleaned Dataset

3.1. Exploratory Data Analysis

4.1. Feature Engineering

4.2. A Dataset with new features

5.1. Comparison of different model performance

5.2. Trained Models

6.1. Hyperparameter tuning

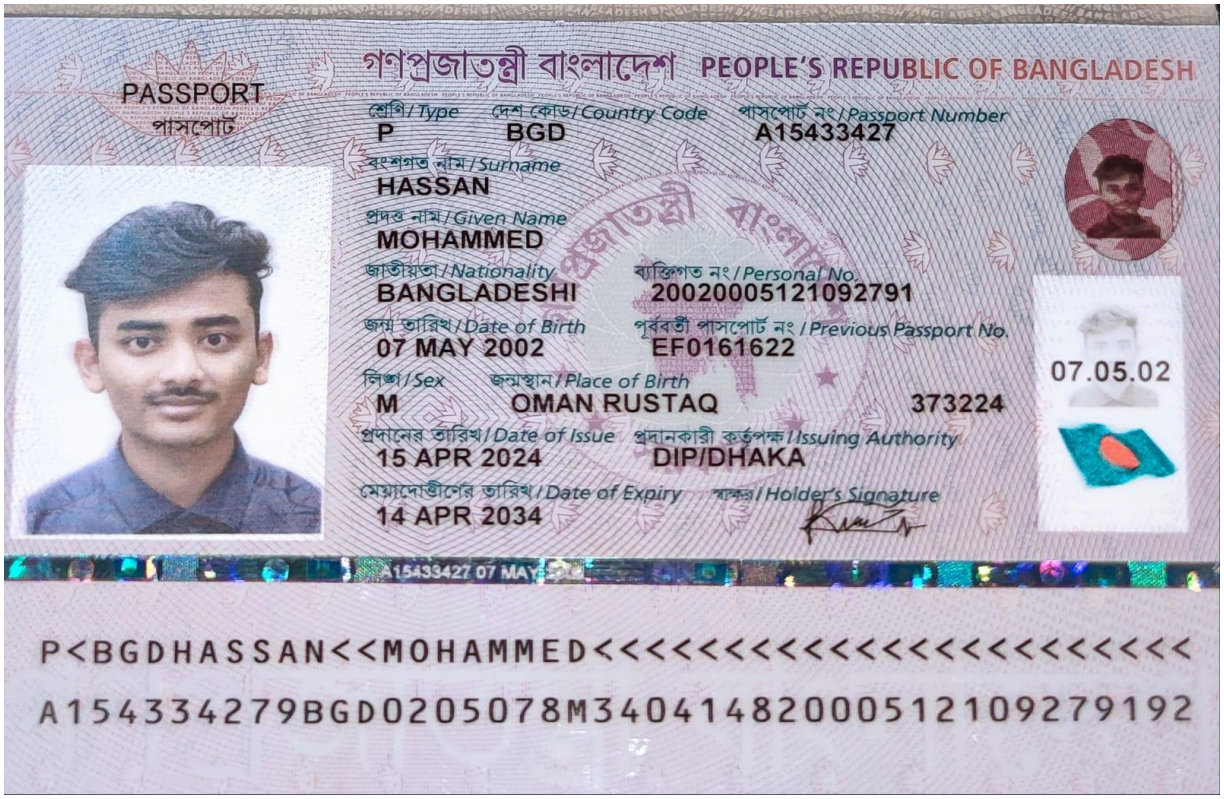
6.2 Best tuned models

7.1 Model Interpretation

8.0 Domain Knowledge and Business Logic

8.1 Team presentation in PDF format

0.0 Passports



PASSPORT
পাসপোর্ট

গণপ্রজাতন্ত্রী বাংলাদেশ PEOPLE'S REPUBLIC OF BANGLADESH

শ্রেণি / Type: P
দেশ কোড / Country Code: BGD
পাসপোর্ট নং / Passport Number: A15433427

বংশগত নাম / Surname
HASSAN

প্রদত্ত নাম / Given Name
MOHAMMED

জাতীয়তা / Nationality
BANGLADESHI

ব্যক্তিগত নং / Personal No.
20020005121092791

জন্ম তারিখ / Date of Birth
07 MAY 2002

পূর্ববর্তী পাসপোর্ট নং / Previous Passport No.
EF0161622

লিঙ্গ / Sex: M
জন্মস্থান / Place of Birth: OMAN RUSTAQ 373224

প্রদানের তারিখ / Date of Issue
15 APR 2024

প্রদানকারী কর্তৃপক্ষ / Issuing Authority
DIP/DHAKA

মেয়াদোত্তীর্ণের তারিখ / Date of Expiry
14 APR 2034

ধারক / Holder's Signature

A15433427 07 MAY

P<BGDHASSAN<<MOHAMMED<<<<<<<<<<<<<<<<<<<<<<<<

A154334279BGD0205078M3404148200051210927 9192

PERSONAL DATA AND EMERGENCY CONTACT

Name:	FAZLUR RAHMAN
Father's Name:	MUJIBUR RAHMAN
Mother's Name:	MUSRAFATUL ZANNAT
Spouse's Name:	
Permanent Address:	SOUTH BHURSHI, WARD NO.04, PATIYA, PATIYA - 4370, CHATTOGRAM
Emergency Contact:	
Name:	MUJIBUR RAHMAN
Relationship:	FATHER
Address:	37 HILLVIEW, R 3, PANCHLAISH, CHITTAGONG MEDICAL COLLEGE - 4203, CHATTOGRAM
Telephone No:	01819354917

A026402491

PASSPORT
পাসপোর্ট

গণপ্রজাতন্ত্রী বাংলাদেশ PEOPLE'S REPUBLIC OF BANGLADESH

শ্রেণি / Type	দেশ কোড / Country Code	পাসপোর্ট নং / Passport Number
P	BGD	A02640249

বংশগত নাম / Surname
RAHMAN

প্রদত্ত নাম / Given Name
FAZLUR

জাতীয়তা / Nationality	ব্যক্তিগত নং / Personal No.
BANGLADESHI	20021591615100659
জন্ম তারিখ / Date of Birth	পূর্ববর্তী পাসপোর্ট নং / Previous Passport No.
03 FEB 2002	

লিঙ্গ / Sex	জন্মস্থান / Place of Birth	
M	CHATTOGRAM	991775
প্রদানের তারিখ / Date of Issue	প্রদানকারী কর্তৃপক্ষ / Issuing Authority	
29 DEC 2021	DIP/DHAKA	
মেয়াদোত্তীর্ণের তারিখ / Date of Expiry	ধারক / Holder's Signature	
28 DEC 2031		

P<BGDRAHMAN<<FAZLUR<<<<<<<<<<<<<<<<<<<<<<<<

A026402491BGD0202033M3112283215916151006595 2

1.1 Dataset Exploration

The dataset used in this project consists of 9000 rows and 11 columns. The columns are as follows:

- **Feature_1 to Feature_8:**
  These are numerical features representing the primary attributes of the dataset.
- **Category_1 and Category_2**:
  These are categorical features that capture distinct classifications within the data.
- **Target**: This is the dependent variable which the model aims to predict

```
df = pd.read_csv('data/final_project_dataset_complete.csv', index_col = False)
df
```

	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
...	...	...	...	...	...	...	...	...	...	...	...
8995	0.101630	0.400250	NaN	-0.019412	-0.063150	0.077627	0.540975	-0.169030	Above Average	Region A	0
8996	1.167218	2.177774	-1.716067	1.994835	0.350043	-0.544915	0.089050	-0.944220	High	Region C	1
8997	1.588447	3.333945	-2.615488	3.476880	-0.933276	2.027658	0.239583	1.951491	High	Region C	1
8998	-0.684987	-1.599835	1.063341	-1.252109	-0.724001	2.228943	0.989794	1.936476	Low	Region B	0
8999	0.801182	1.808647	-1.383917	1.841776	0.702840	-0.339918	-0.027882	-0.462744	High	Region C	1

9000 rows × 11 columns

Basic Information and Statistical Details about the Dataset:

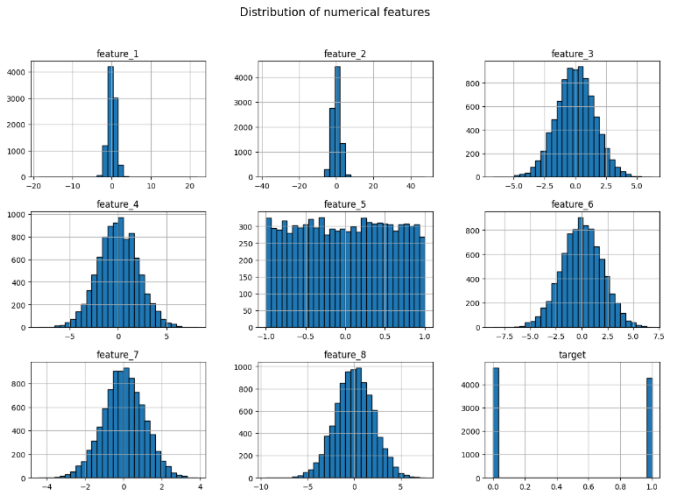
	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	target
count	9000.000000	9000.000000	8600.000000	9000.000000	9000.000000	8500.000000	9000.000000	9000.000000	9000.000000
mean	0.000427	0.003349	0.003235	-0.008481	-0.002177	-0.006447	0.000592	0.003348	0.475444
std	1.241318	2.508324	1.542901	2.061784	0.577415	1.981615	1.075064	2.043643	0.499424
min	-18.665400	-37.852816	-6.676680	-8.190124	-0.999791	-8.590782	-4.422265	-9.474989	0.000000
25%	-0.680062	-1.382610	-1.022085	-1.399928	-0.502614	-1.329040	-0.700078	-1.356620	0.000000
50%	-0.003938	-0.016698	0.005196	-0.019541	0.001695	-0.003137	-0.000097	-0.007584	0.000000
75%	0.680513	1.380228	1.038571	1.394151	0.497004	1.324897	0.731942	1.402024	1.000000
max	21.934496	47.603454	6.203055	8.189001	0.999914	6.803751	3.857219	7.572578	1.000000

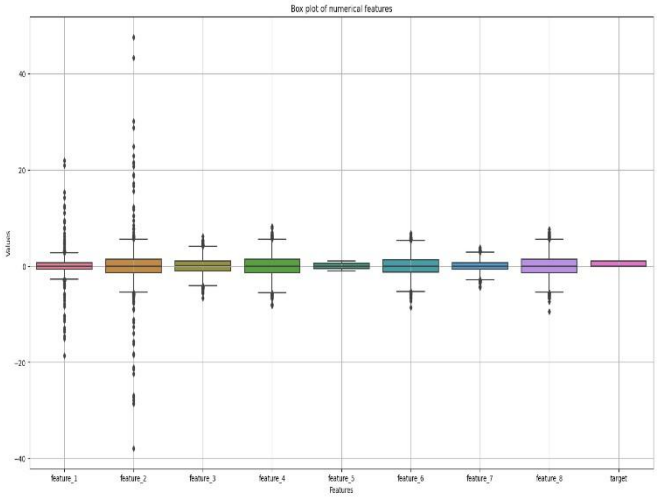
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9000 entries, 0 to 8999
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   feature_1   9000 non-null   float64
 1   feature_2   9000 non-null   float64
 2   feature_3   8600 non-null   float64
 3   feature_4   9000 non-null   float64
 4   feature_5   9000 non-null   float64
 5   feature_6   8500 non-null   float64
 6   feature_7   9000 non-null   float64
 7   feature_8   9000 non-null   float64
 8   category_1  9000 non-null   object 
 9   category_2  9000 non-null   object 
 10  target      9000 non-null   int64  
dtypes: float64(8), int64(1), object(2)
```

Here, we observed the statistical details of each column, which provided insights into the data types of the features. This understanding will assist in making necessary adjustments during the analysis.

Initial distribution of features

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	target
feature_1	1.000000	0.995360	-0.831705	0.831922	-0.015305	-0.010813	-0.010843	-0.004581	0.605940
feature_2	0.995360	1.000000	-0.823242	0.832002	-0.014715	-0.011519	-0.011393	-0.005394	0.606016
feature_3	-0.831705	-0.823242	1.000000	-0.957222	0.011309	0.003989	0.006042	0.001846	-0.704351
feature_4	0.831922	0.832002	-0.957222	1.000000	-0.003613	-0.000213	-0.001369	0.001202	0.694885
feature_5	-0.015305	-0.014715	0.011309	-0.003613	1.000000	-0.009365	0.252371	-0.012717	-0.007873
feature_6	-0.010813	-0.011519	0.003989	-0.000213	-0.009365	1.000000	0.921569	0.969998	0.001457
feature_7	-0.010843	-0.011393	0.006042	-0.001369	0.252371	0.921569	1.000000	0.894220	-0.002000
feature_8	-0.004581	-0.005394	0.001846	0.001202	-0.012717	0.969998	0.894220	1.000000	0.004658
target	0.605940	0.606016	-0.704351	0.694885	-0.007873	0.001457	-0.002000	0.004658	1.000000





These visualizations collectively provide a comprehensive understanding of the dataset's structure, relationships, and potential preprocessing needs. The **histograms** indicate that while some features exhibit a symmetric distribution, others follow a uniform or skewed pattern. The **correlation matrix** highlights that certain features are strongly correlated with the target variable, whereas others show little to no relationship. Later, we can visualize it using heatmap during EDA stage. Additionally, it reveals the correlation (both positive and negative) between different features. The **boxplot** provides insights into data congestion and helps identify potential outliers that may need to be addressed during preprocessing.

2.1 Data Preprocessing

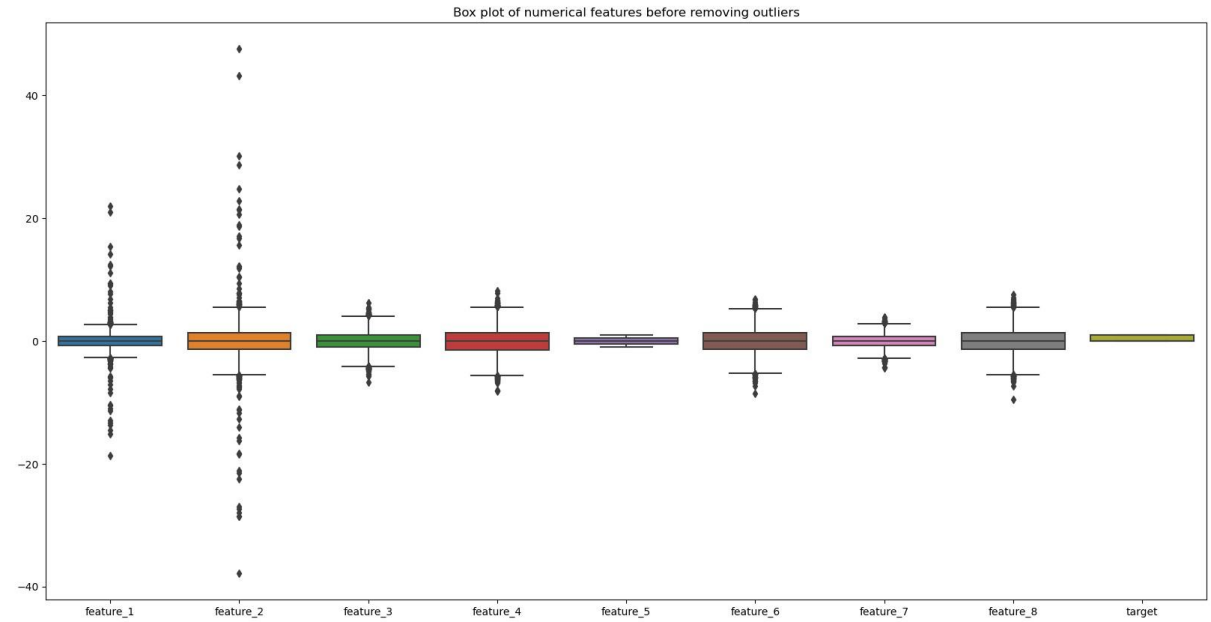
Handling missing values of Numerical Features:

```
[136] missing_values = df.isna().sum()
      missing_values = missing_values[missing_values > 0]
      print(missing_values)

      feature_3    400
      feature_6    500
      dtype: int64
```

Using (missing_values = df.isna().sum()), we identified approximately 400 and 500 missing values in **feature_3** and **feature_6**, respectively. To address this, we utilized the SimpleImputer function to replace the missing values with the median of their respective columns. Upon re-checking the dataset, we observed that there are no longer any missing values, allowing us to proceed with further analysis.

```
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
df['feature_3'] = imputer.fit_transform(df[['feature_3']])
df['feature_6'] = imputer.fit_transform(df[['feature_6']])
```

Handling outliers:



As shown in the **boxplot** above, there are many outliers in individual features. Therefore, we need to remove them to prevent skewing the results, ensuring

improved accuracy and reliability of the model. Therefore, we implement the IQR (interquartile range) method (as one of many ways of detecting outliers effectively)

```
def iqr_outliers(dataset, feature_name, multiplier= 1.5):
    Q1 = dataset[feature_name].quantile(0.25)
    Q3 = dataset[feature_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_limit = Q1 - multiplier * IQR
    upper_limit = Q3 + multiplier * IQR

    outliers = dataset[(dataset[feature_name] < lower_limit) | (dataset[feature_name] > upper_limit)]

    return outliers, {'lower_limit': lower_limit, 'upper_limit': upper_limit}

outliers_detected = {}
```

First, we calculate **Q1** (the 25th percentile), which represents the value below which 25% of the data points fall, and **Q3** (the 75th percentile), representing the value below which 75% of the data points fall. Next, we compute the **Interquartile Range (IQR)** using the formula: **IQR = Q3 – Q1**, which represents the range containing the middle 50% of the data.

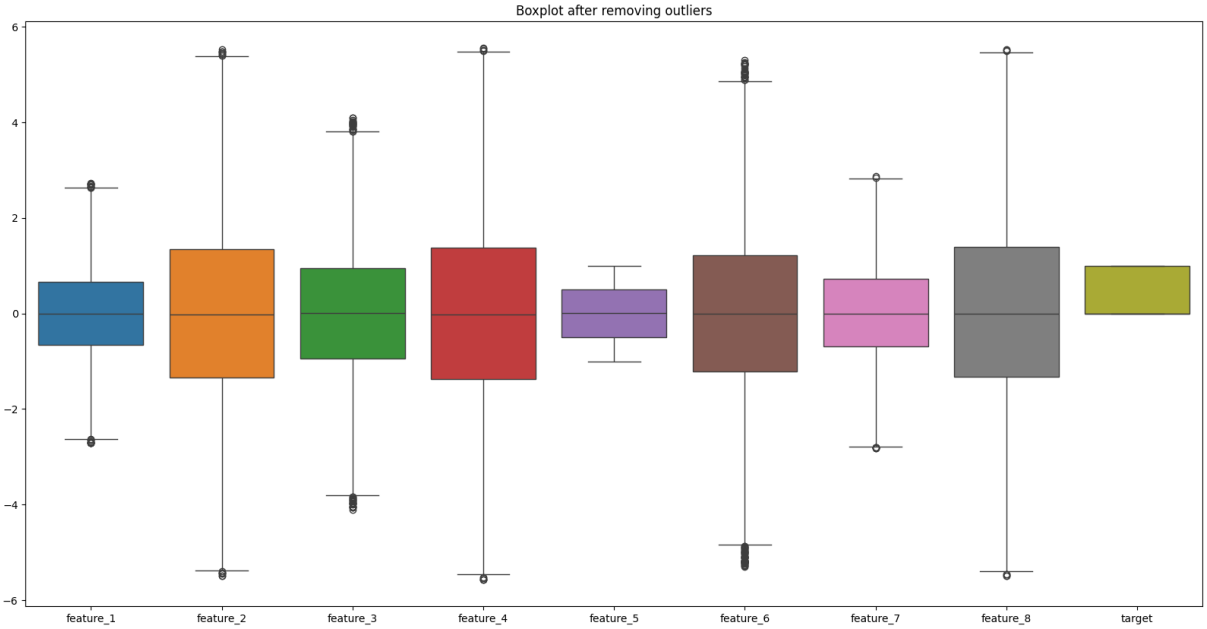
To determine the outlier limits, we calculate:

- **Lower Limit**: Q1 – (multiplier * IQR)
- **Upper Limit**: Q3 + (multiplier * IQR)

Any data points falling below the lower limit or above the upper limit are considered outliers.

After detecting outliers, treating them is necessary. In this case, we use a basic approach by replacing the outliers with the **median** value of their respective features.

```
df_1 = df.copy()
for feature, values in outliers_detected.items():
    outlier_indices = values['indices']
    median_value = df_1[feature].median()
    df_1.loc[outlier_indices, feature] = median_value
    print(f"Feature: {feature}")
    print(f"Replaced outliers in indices: {outlier_indices}")
    print('-' * 50)
```



After handling the outliers, we can visualize the boxplot once again to see if there are any outliers left. As we can see, the outliers have been greatly minimized.

The way of eliminating 100% outliers would not be practical in the real world since outliers sometimes would play a very important role in defining the meaning and pattern behind data. Removing outliers completely could lead to loss of information, reduction of computational power or creating bias decisions. Therefore, we can only minimize them as small as possible but not eliminate them completely.

Labelling the categorical features:

Since many models and algorithms are very sensitive to categorical features and require numerical features to work well with, we need to handle the categorical features using labeling methods.

```
categorical_features.head()
```

	category_1	category_2
0	Above Average	Region C
1	Below Average	Region A
2	High	Region C
3	High	Region B
4	Below Average	Region C

We perform **encoding** to convert categorical features into numerical form. In our case, we apply it to **category_1** and **category_2**.

```
df_2 = df_1.copy()
#for category_1
ordinal_mapping ={
    'Low': 0,
    'Below Average': 1,
    'Above Average': 2,
    'High': 3
}
df_2['category_1_encoded'] = df_2['category_1'].map(ordinal_mapping)

#for category_2
df_one_hot = pd.get_dummies(df_2['category_2'])
df_one_hot = df_one_hot.astype(int)
df_2 = pd.concat([df_2, df_one_hot], axis=1)

df_2.drop(columns=['category_1','category_2'], inplace =True)
df_2.head()
```

For feature_1, since it is ordinal data type (Below Average, Above Average, High and Low), we do it manually with a dictionary mapping. Since we know the order Low -> Below Average -> Above Average -> High so we apply it respectively 0, 1, 2 and 3. And then, we use map() method from pandas. It iterates through each

value in the column of category_1, for each value in the column, it looks up the keys in the ordinal_mapping.

For category_2, we basically apply One_Hot_Encoding and apply 0 and 1 for Region_A, Region_B and Region_C since they are all nominal data, orders do not matter here. This is the most appropriate and accurate for this type of data.

Finally, we drop the original **category_1** and **category_2** columns, as they are no longer needed after encoding.

2.2 A Cleaned Dataset

After replacing the missing values to prevent errors during model training, handling outliers to avoid skewed results, and labelling the categorical features, we obtained our cleaned dataset. This processed data was then exported as **'cleaned_dataset.csv'**.

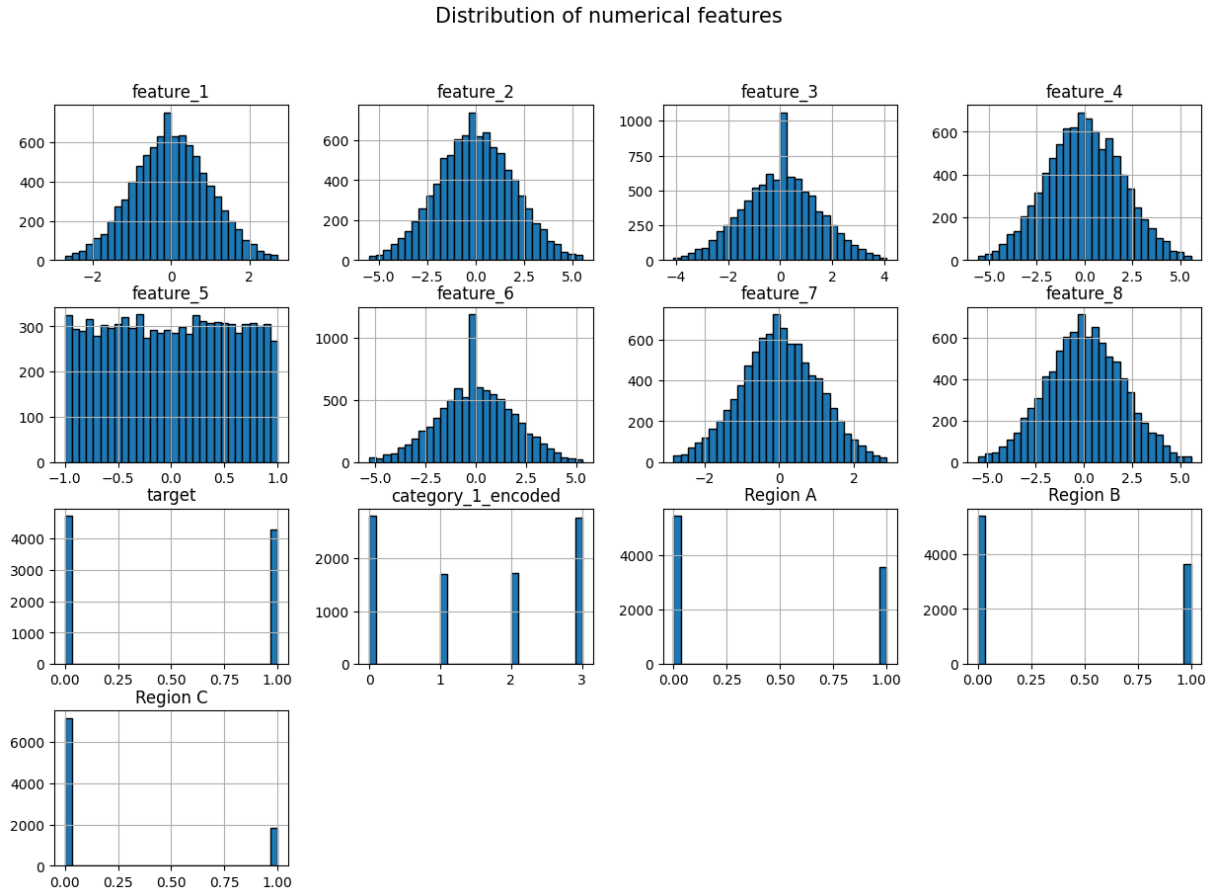
```
[148] # export cleaned dataset
      df_2.to_csv('C:\depression analysis\Machine Learning\final project\cleaned_dataset.csv')
```

3.1 Exploratory Data Analysis

In this phase, we conducted Exploratory Data Analysis (EDA) to gain a deeper understanding of the dataset. Through this process, we analyzed and visualized the data to uncover patterns, relationships, and potential issues. EDA allowed us to examine the distribution of features, identify correlations, detect outliers, and assess the overall structure of the dataset. These insights were instrumental in

guiding the subsequent steps of data preprocessing, feature engineering, and model development.

Here is the distribution of features:



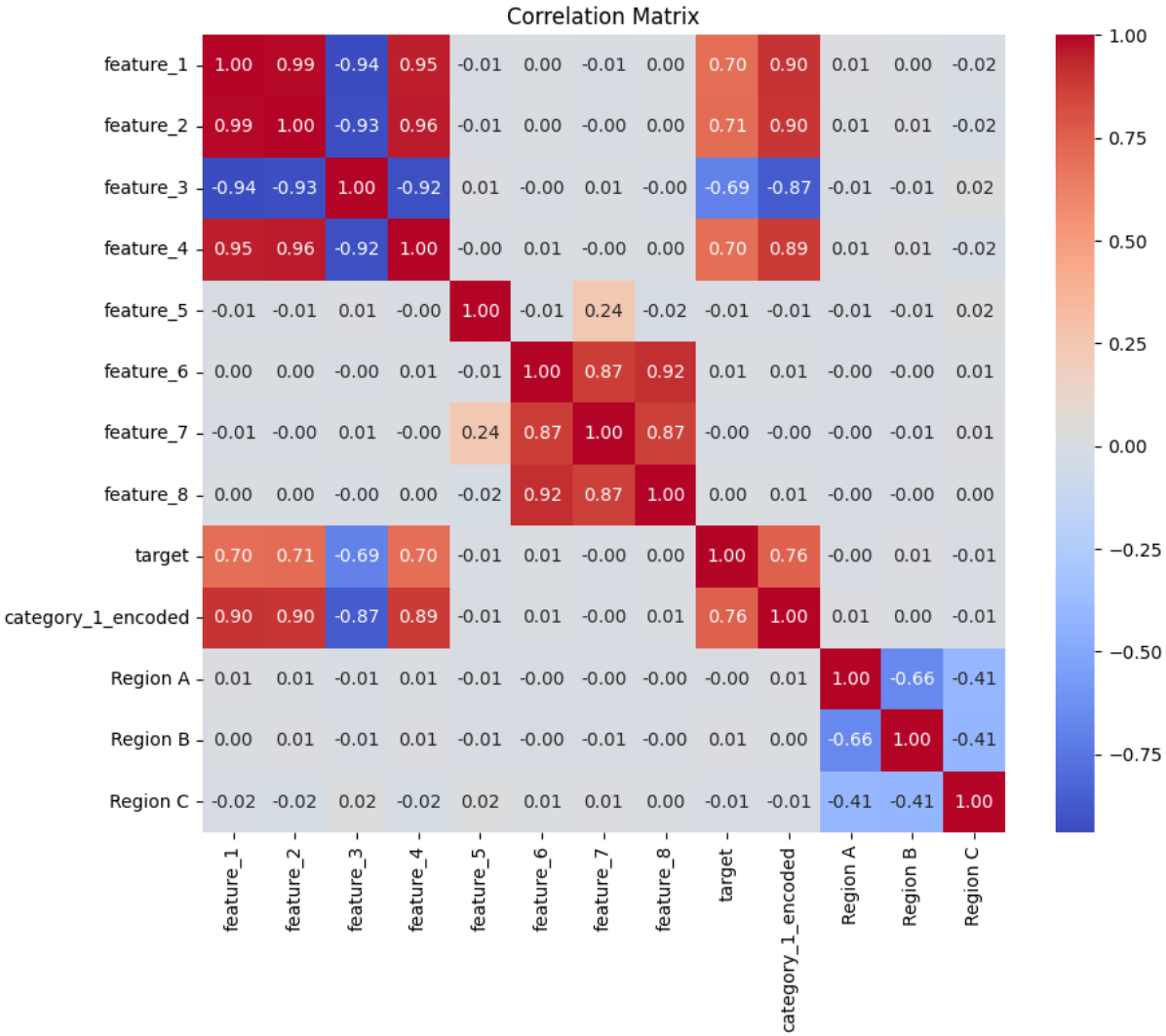
Feature_1, feature_2, feature_3, feature_4, feature_7 and feature_8 clearly show a bell-shaped curve, which is a characteristic of a normal (Gaussian) distribution. They are concentrated around the mean and become gradually less towards the tails.

Feature_5 strongly resembles a uniform distribution. The bars are roughly the same height across the entire range meaning that values between -1.0 and 1.0 occur with approximately equal frequency.

Feature_6 occurs to be skewed, it has a tail extending towards the positive side or right skewed to be more specific.

Features such as target, category_1_encoded, Region_A, Region_B, Region_C basically show their discrete value counts.

Implementing heatmap to visualize the correlation matrix:



Before feature engineering, the visualization of correlation matrix (heatmap above) provides a clear understanding of which features are mostly related to the target variable.

Feature_1, feature_2, feature_3, feature_4 and category_1_encoded strongly related to the target meanwhile other features do not show any correlation at all toward the target variable.

Additionally, as part of our Exploratory Data Analysis (EDA), we conducted chi-square tests for the categorical variables (**category_1** and **category_2**) and t-tests for the numerical features against the target variable.

- **Chi-Square Tests**: These tests were performed to determine if there is a statistically significant relationship between the categorical variables and the target variable. This helped us assess the relevance of these features in predicting the target.
- **T-Tests**: For numerical features, we evaluated whether the mean values differ significantly between the two target classes. This provided insight into which numerical features have distinguishing power for the target variable.

Result of T-Test and chi-square test:

Feature	t_statistic	p_value
feature_1	94.129918	0.000000
feature_2	95.081086	0.000000
feature_3	-90.742558	0.000000
feature_4	92.464130	0.000000
feature_5	-0.746804	0.455201
feature_6	0.499102	0.617720
feature_7	-0.210384	0.833373
feature_8	0.413642	0.679146
target	inf	0.000000
category_1_encoded	109.298957	0.000000
Region A	-0.056275	0.955124
Region B	0.897181	0.369646
Region C	-1.024417	0.305666

Feature	chi2_statistic	p_value
category_1_encoded	5175.320287	0.000000
region_a	0.001203	0.972329
region_b	0.766878	0.381185
region_c	0.996512	0.318156

Conclusion

- **Important Features**: Based on the results, **feature_1**, **feature_2**, **feature_3**, **feature_4**, and **category_1_encoded** are likely to be the most predictive features for the target variable.
- **Less Relevant Features**: Features like **feature_5**, **feature_6**, **feature_7**, **feature_8**, Region_A, Region_B and Region_C show little to no statistical relationship with the target and may be less impactful for the model.

4.1 FEATURE ENGINEERING

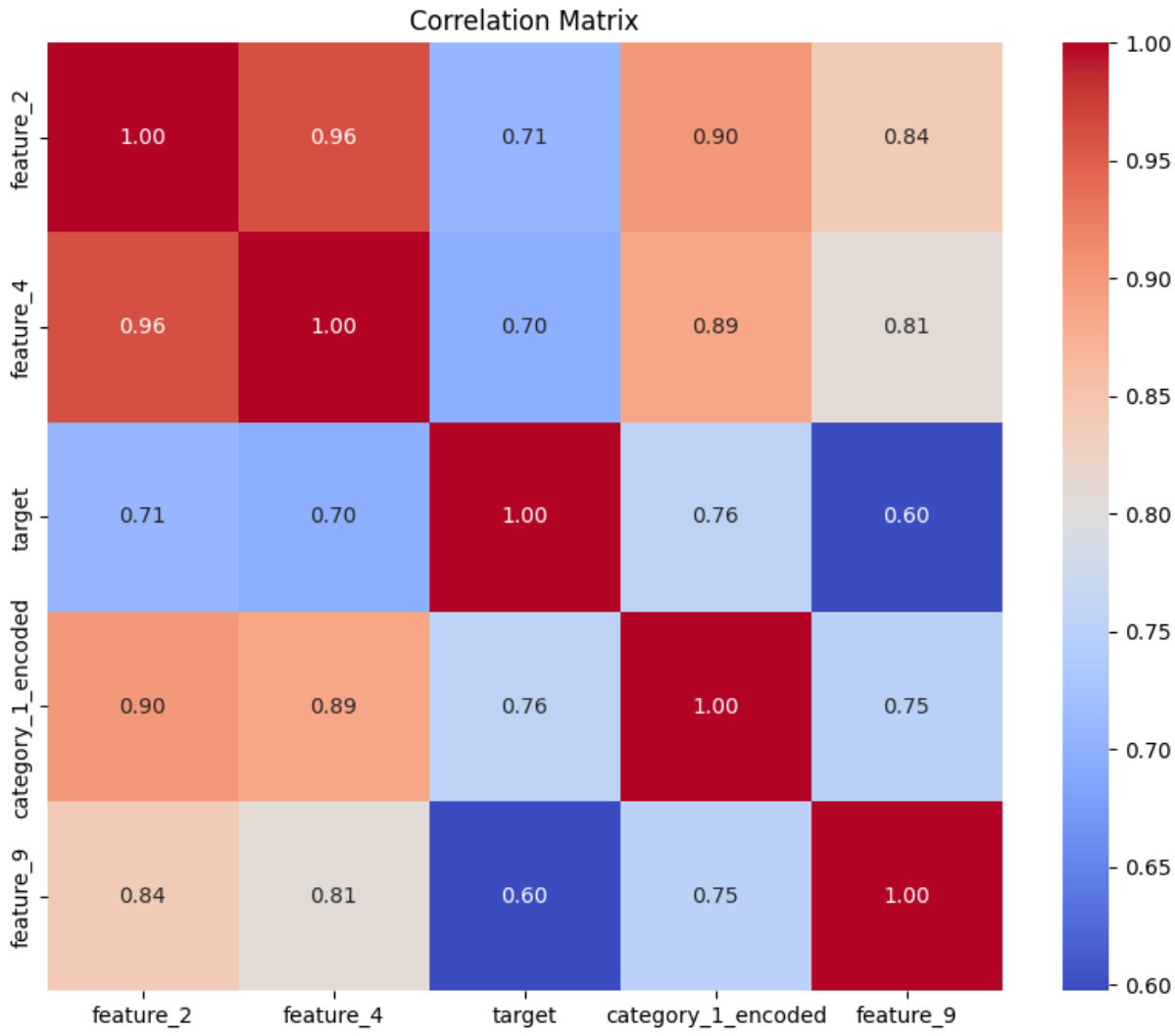
Raw data is often unordered, uninformative, or lacks context. Performing feature engineering carefully and insightfully can significantly improve model performance and interpretation. By creating new, relevant features and modifying existing ones, we simplify the model and reduce the risk of overfitting where the model memorizes the training data too closely, leading to poor performance on unseen data.

Moreover, we also remove irrelevant data according to the EDA stage above to avoid redundancy and thus reduce computational cost.

```
df_3 = df_2.copy()
# feature engineering
def feature_engineering(df_3):
    df_3['feature_9'] = df_3['feature_2'] * df_3['category_1_encoded']
    df_3 = df_3.drop(['feature_1', 'feature_3', 'feature_5', 'feature_6', 'feature_7', 'feature_8', 'Region A', 'Region B', 'Region C'
    return df_3

df_3 = feature_engineering(df_3)
df_3.head()
```

Based on our earlier observations, including insights from the correlation matrix, chi-square tests, and t-tests, we identified several features that contributed little or nothing to the target variable and were irrelevant to the model training process. Consequently, we decided to remove feature_3 **feature_5**, **feature_6**, **feature_7**, **feature_8**, and **Region A**, **Region B**, and **Region C** to streamline the dataset and enhance model efficiency. We also remove feature_1 since it and feature_2 are basically the same, it contributes the same amount of work for the models

Additionally, we introduced a new feature, **feature_9**, derived from **feature_2** and **category_1_encoded**. These two features showed a strong relationship in the correlation matrix, making it logical to combine them into a single feature. This new feature aims to better capture the combined effect of their interaction, thereby improving the model's predictive power.



Here is the heatmap to show the correlation between features (including the new one) toward the target variable.

Scaling the features:

After feature engineering, it's now good for us to scale the features. The reason why we didn't do it before is because feature engineering often changes scale, which means many feature engineering techniques directly change the scale of the features. That might lead to redundant and ineffective, the initial scaling step might be affected by the feature engineering step so it wouldn't be necessary to do it before hand.

```
#feature scaling
scaler = MinMaxScaler()
for feature in df_3:
    df_3[feature] = scaler.fit_transform(df_3[[feature]])

print(df_3.head())
```

We apply Min-Max Scaler to apply to feature scaling in this case, it would scale the features in the [0, 1]. Min-Max scaling does not change the shape of the original distribution. My normal distribution will remain the same and it is the same as skewed distribution. It just stretches or compresses it into the new range.

4.2 A Dataset with new features

After feature engineering the new dataset:

	feature_2	feature_4	target	category_1_encoded	feature_9
0	1.146509	0.833005	1	2	2.293017
1	-0.061846	0.403768	0	1	-0.061846
2	1.395115	1.708266	1	3	4.185345
3	2.657560	2.649051	1	3	7.972681
4	-0.499391	-0.441656	0	1	-0.499391

After performing feature engineering, we refined the original dataset by removing redundant features and introducing new ones to enhance model performance. The updated dataset now focuses on features that contribute significantly to predicting the target variable, as determined by insights from the correlation matrix, chi-square tests, and t-tests.

This new dataset is now better aligned with the goals of our analysis and serves as an optimized input for the subsequent modelling phase.

5.1 Comparison of different model performance

Model Training and Evaluation:

To evaluate model performance effectively, we first split the dataset into a training set and a testing set. The training set is used to train the models, while the testing set evaluates how well the trained models perform on unseen data, helping us avoid overfitting.

Model Comparison:

Model	Accuracy
Random Forest	0.842
Voting Classifier	0.861
Gradient Boosting	0.864

The table above compares the training and testing accuracies of the models without hyperparameter tuning. These results are based on default configurations, meaning no modifications were made to the parameters.

This is a bar chart comparing the performance metrics of three different models: **Random Forest**, **Gradient Boosting**, and **Voting Classifier.**

Overall Performance: All three models appear to perform quite well across all metrics, with scores generally above 0.8.

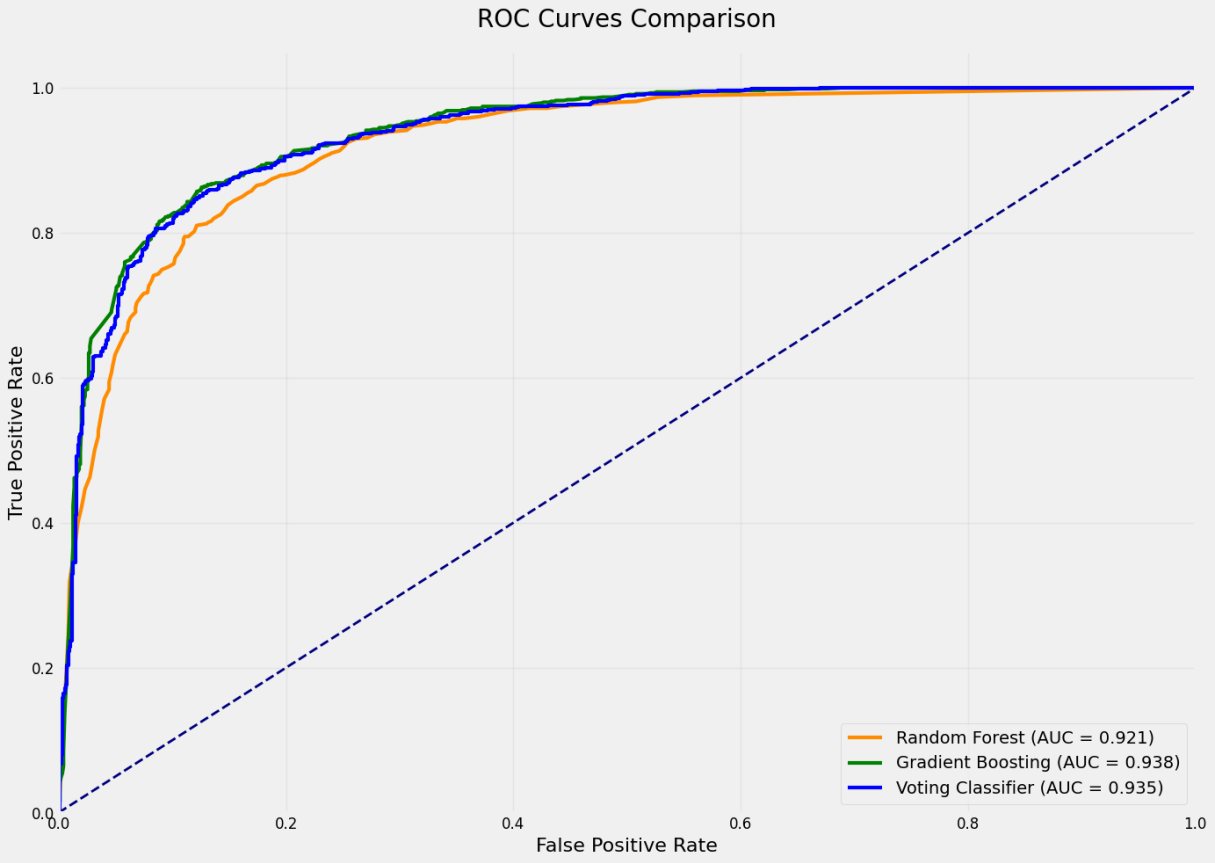
Consistency: The scores for Accuracy, Precision, Recall, and F1 Score are relatively close to each other for each model, indicating a good balance in their performance.

Comparison of Models:

Random Forest: Shows good performance, with all metrics hovering around 0.83-0.84.

Gradient Boosting: Seems to have slightly higher scores across all metrics compared to Random Forest, particularly for Precision and F1 Score, which are close to 0.88.

Voting Classifier: Appears to be the best-performing model among the three shown. It achieves the highest scores for Precision (close to 0.89) and F1 Score (around 0.86-0.87), and its Accuracy and Recall are also very strong, comparable to or slightly better than Gradient Boosting.



Model Performance vs. Random: All three models' ROC curves are significantly above the diagonal dashed line, indicating that they are performing much better than random guessing.

Closeness of Curves: The curves for all three models are very close to each other, especially at higher True Positive Rates, suggesting that they have similar predictive capabilities.

AUC Values Comparison:

- **Random Forest:** AUC = 0.921
- **Gradient Boosting:** AUC = 0.938
- **Voting Classifier:** AUC = 0.935

Best Performer (Based on AUC): Gradient Boosting has the highest AUC of 0.938, indicating that it is slightly better at distinguishing between positive and negative classes across various classification thresholds compared to the other two models.

Voting Classifier and Gradient Boosting: These two models have very similar AUC scores (0.935 vs 0.938), and their curves are almost indistinguishable for most of the range, especially at higher True Positive Rates.

Random Forest:

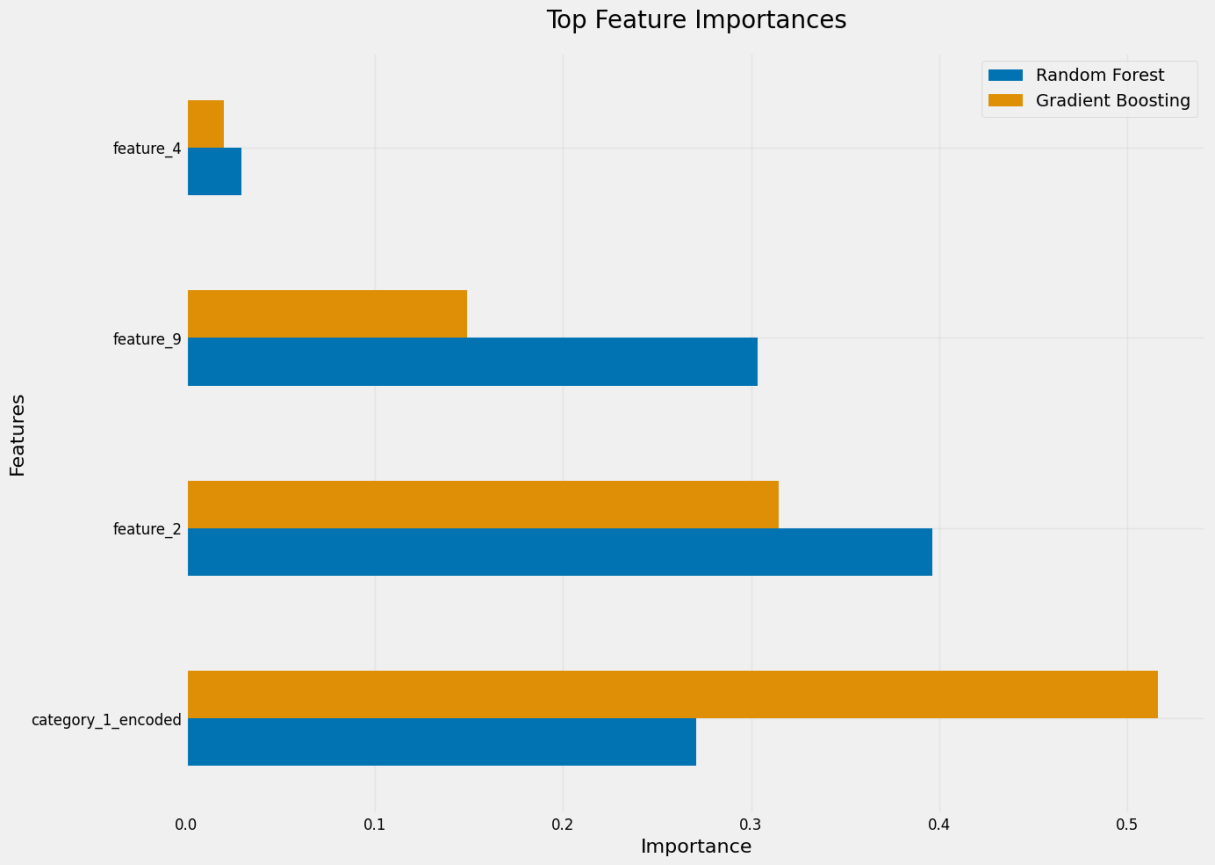
- The median score is around 0.84.
- The box is relatively narrow, suggesting consistency in performance across different folds.
- There are a couple of outlier points below and above the main cluster, indicating a few folds where performance was slightly lower or higher than usual.

Gradient Boosting:

- The median score appears to be slightly higher than Random Forest, around 0.86.
- The box is also narrow, indicating consistent performance.
- It has one outlier point, again suggesting a fold with slightly different performance. The whisker appears to reach a slightly higher score than Random Forest, and its lower whisker is higher too, indicating a generally better baseline.

Voting Classifier:

- The median score is very similar to Gradient Boosting, possibly slightly above 0.86.
- The box is very narrow, indicating excellent consistency in performance across cross-validation folds.
- It has one outlier point below the main cluster, similar to the other models.



"category_1_encoded" is the Most Important Feature: Both Random Forest and Gradient Boosting agree that "category_1_encoded" is the most important feature. Gradient Boosting assigns it a significantly higher importance (around 0.52) compared to Random Forest (around 0.28). This suggests that this categorical feature plays a crucial role in the predictions of both models, especially Gradient Boosting.

"feature_2" is Also Highly Important: "feature_2" is the second most important feature for both models. Random Forest assigns it the highest importance among all features (around 0.4), while Gradient Boosting also gives it high importance (around 0.33).

"feature_9" Shows Divergence: For "feature_9", Random Forest assigns a higher importance (around 0.29) than Gradient Boosting (around 0.15). This indicates that Random Forest leverages "feature_9" more heavily in its decision-making process than Gradient Boosting does.

"feature_4" has Low Importance: Both models assign very low importance to "feature_4", with Gradient Boosting giving it slightly more importance (around 0.02) than Random Forest (a very low positive value, almost negligible). This suggests "feature_4" is not very influential for either model's predictions.

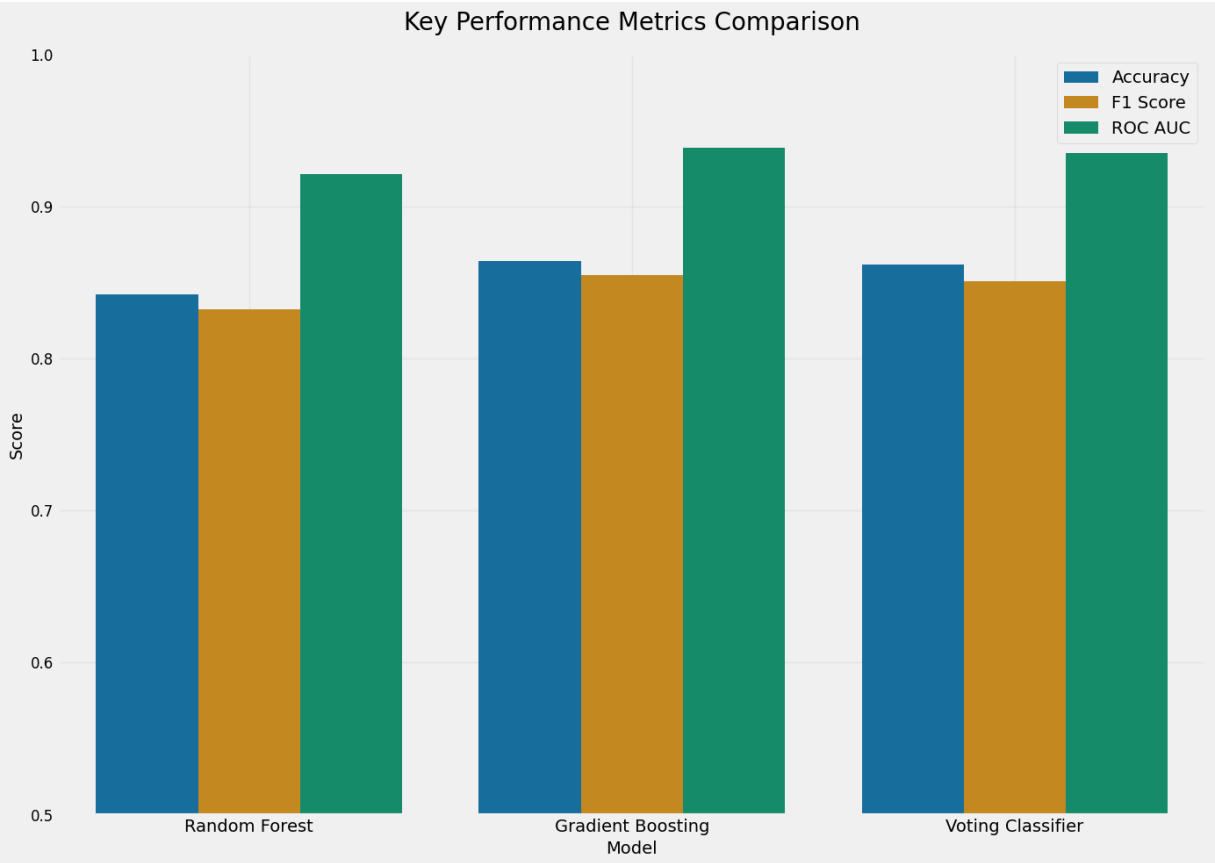
Model Performance Summary

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC	CV Mean	CV Std
Random Forest	0.842	0.839	0.825	0.832	0.921	0.836	0.010
Gradient Boosting	0.864	0.867	0.843	0.855	0.938	0.858	0.009
Voting Classifier	0.862	0.871	0.831	0.851	0.935	0.858	0.010

Random Forest: Shows solid overall performance. Its F1 Score is quite balanced. The ROC AUC is good, and its cross-validation mean is close to its overall accuracy, with a low standard deviation indicating stable performance.

Gradient Boosting: Consistently outperforms Random Forest across all metrics (Accuracy, Precision, Recall, F1 Score, and ROC AUC). Its ROC AUC is the highest among the three. Its CV Mean is also higher than Random Forest's, and its CV Std is the lowest, suggesting the most stable performance.

Voting Classifier: Very close in performance to Gradient Boosting. It has the highest Precision, and its F1 Score, Accuracy, and ROC AUC are very competitive with Gradient Boosting. Its CV Mean is identical to Gradient Boosting, and its CV Std is also very low, indicating excellent stability.



Random Forest: Shows solid performance across these key metrics. The ROC AUC is notably higher than Accuracy and F1 Score, which is common as AUC measures overall separability, while Accuracy and F1 are threshold-dependent.

Gradient Boosting: Consistently outperforms Random Forest in all three metrics. Its ROC AUC is the highest among the three models shown.

Voting Classifier: Shows performance very similar to Gradient Boosting across all three metrics. It seems to be performing slightly below Gradient Boosting in ROC AUC, but the differences are minimal.

5.2. Detailed Trained Models

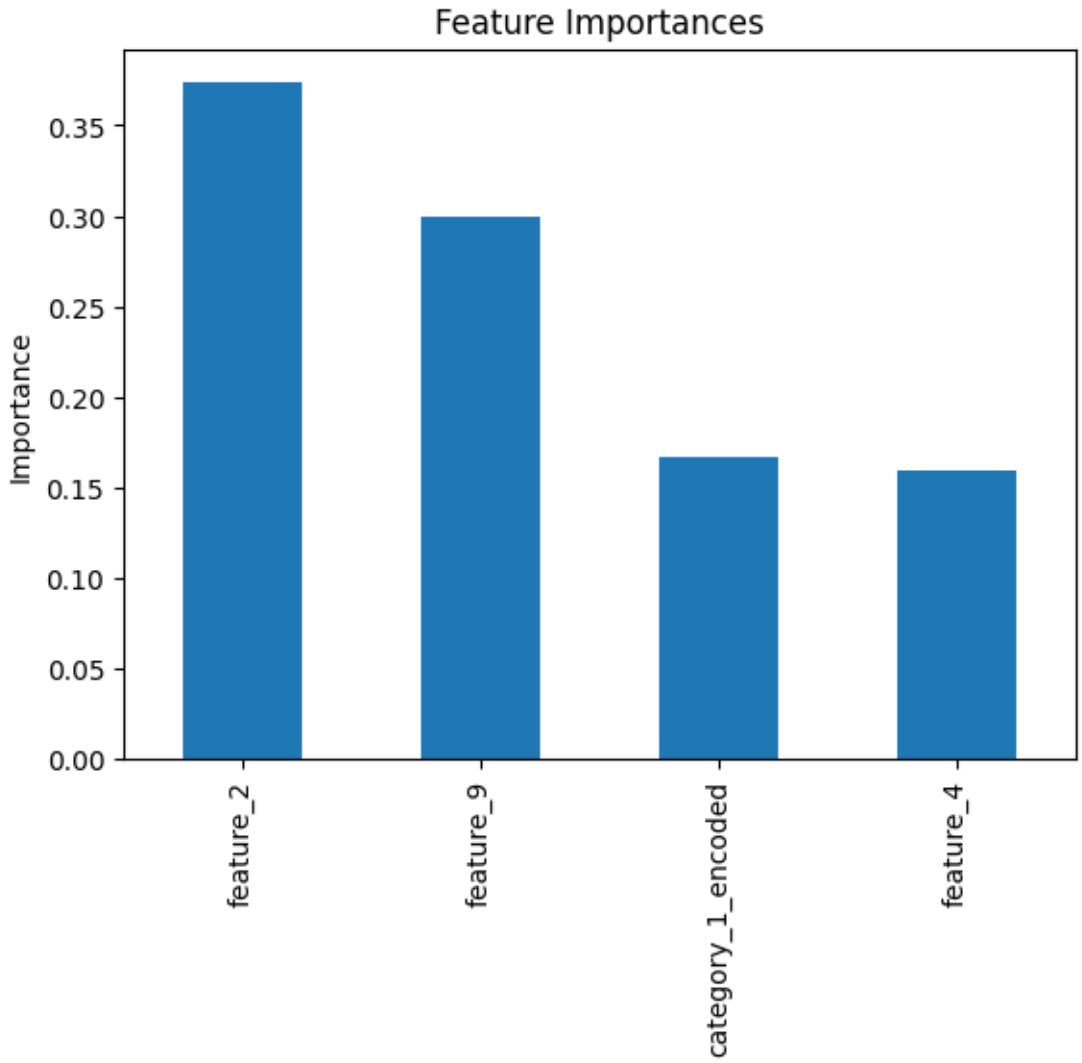
We used these models:

- Random Forest
- Gradient Boosting
- Voting Classifier

All 3 models, Random Forest, Voting Classifier and Gradient Boosting are ensemble learning methods, they all combine multiple models to create a stronger and more effective prediction.

- **Ensemble Methods:**
    - **Random Forest:** Creates a "forest" of decision trees, each trained on a different subset of the data, and combines their predictions.
    - **Voting:** Combines the predictions of multiple different models (which could include decision trees) by taking a vote.
    - **Gradient Boosting:** Builds models sequentially, where each new model tries to correct the errors of the previous ones.

RANDOM FOREST



Random Forest model has built-in feature importance mechanism.

The bar chart above clearly shows that feature_2 is the most important feature, feature_9 is the second most important. Category_1_encoded also shows a high importance meaning that the ordinal encoded feature is good. Feature_4 is the least important but still contributes to the model's performance.

	precision	recall	f1-score	support
0.0	0.84	0.86	0.85	947
1.0	0.84	0.83	0.83	853
accuracy			0.84	1800
macro avg	0.84	0.84	0.84	1800
weighted avg	0.84	0.84	0.84	1800

Here is the classification report which is a fundamental tool to show the evaluation of a classification model.

0.0: Represents the performance metrics for predicting the negative class (or class 0).

1.0: Represents the performance metrics for predicting the positive class (or class 1)

Precision:

For class 0.0: 0.84 means when the model predicted an instance belonged to class 0, it was correct 84% of the time.

For class 1.0: 0.84. This means when the model predicted an instance belonged to class 1, it was also correct 84% of the time.

Recall:

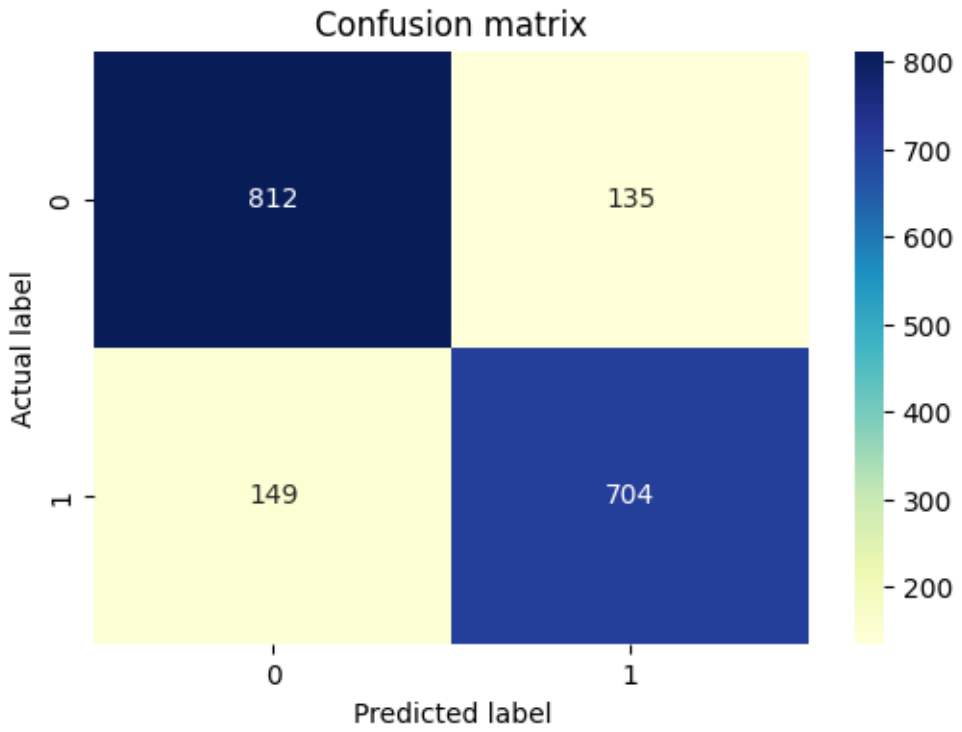
For class 0.0: 0.86. This means the model correctly identified 86% of all actual instances of class 0.

For class 1.0: 0.83. This means the model correctly identified 83% of all actual instances of class 1.

F1-score: it is a harmony between precision and recall. It provides a single score that balances both precision and recall, useful when you need a balance between them.

Support: this is the actual number of occurrences (instances) of each class in the true labels of the data.

class 0.0 got 947 instances and class 1.0 got 853 instances



Confusion matrix is another fundamental tool to understand deeper into the classification report

Top-Left (812): True Negatives (TN)

- The model correctly predicted **812** instances as class 0, and they were class 0. These are correct predictions for the negative class.

Top-Right (135): False Positives (FP)

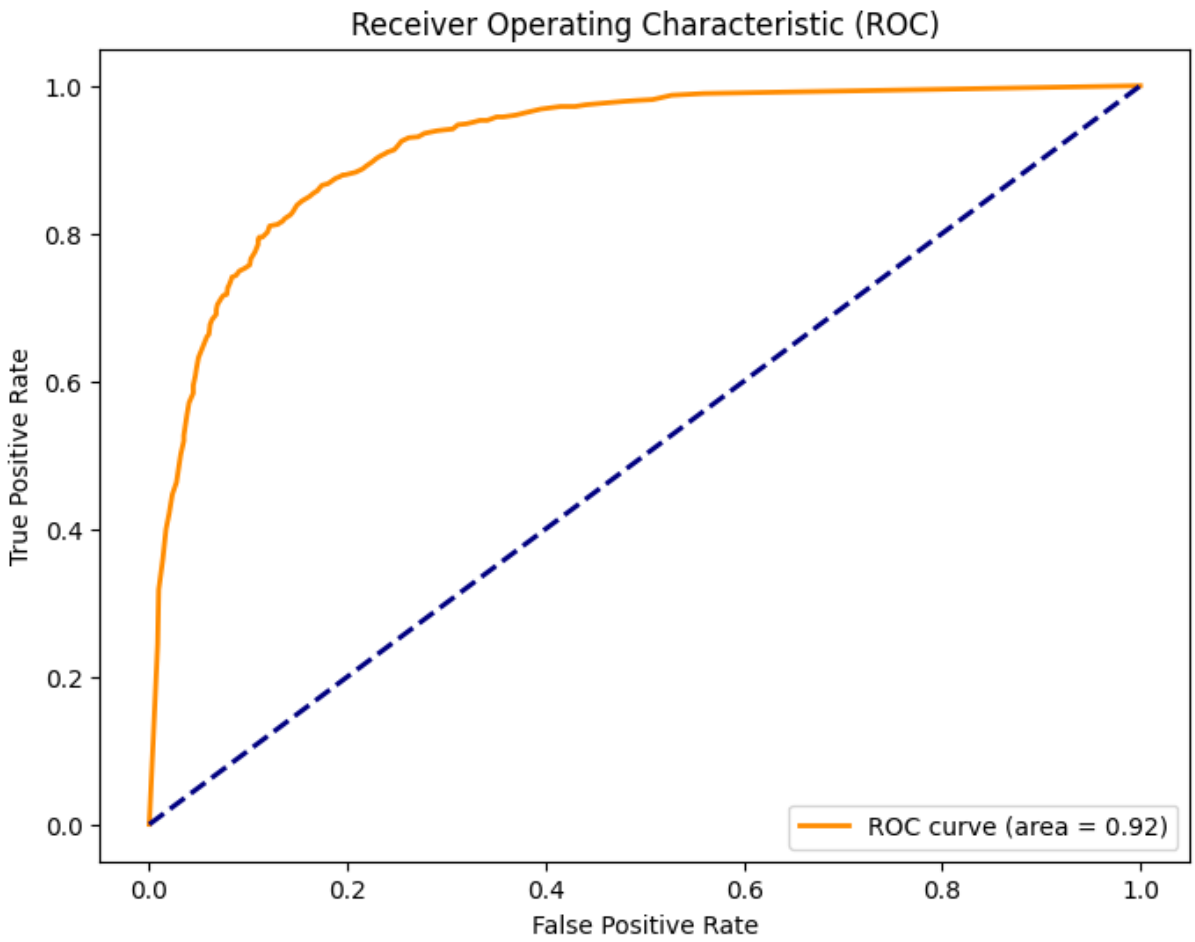
- The model incorrectly predicted **135** instances as class 1, but they were also class 0.

Bottom-Left (149): False Negatives (FN)

- The model incorrectly predicted **149** instances as class 0, but they were class 1.

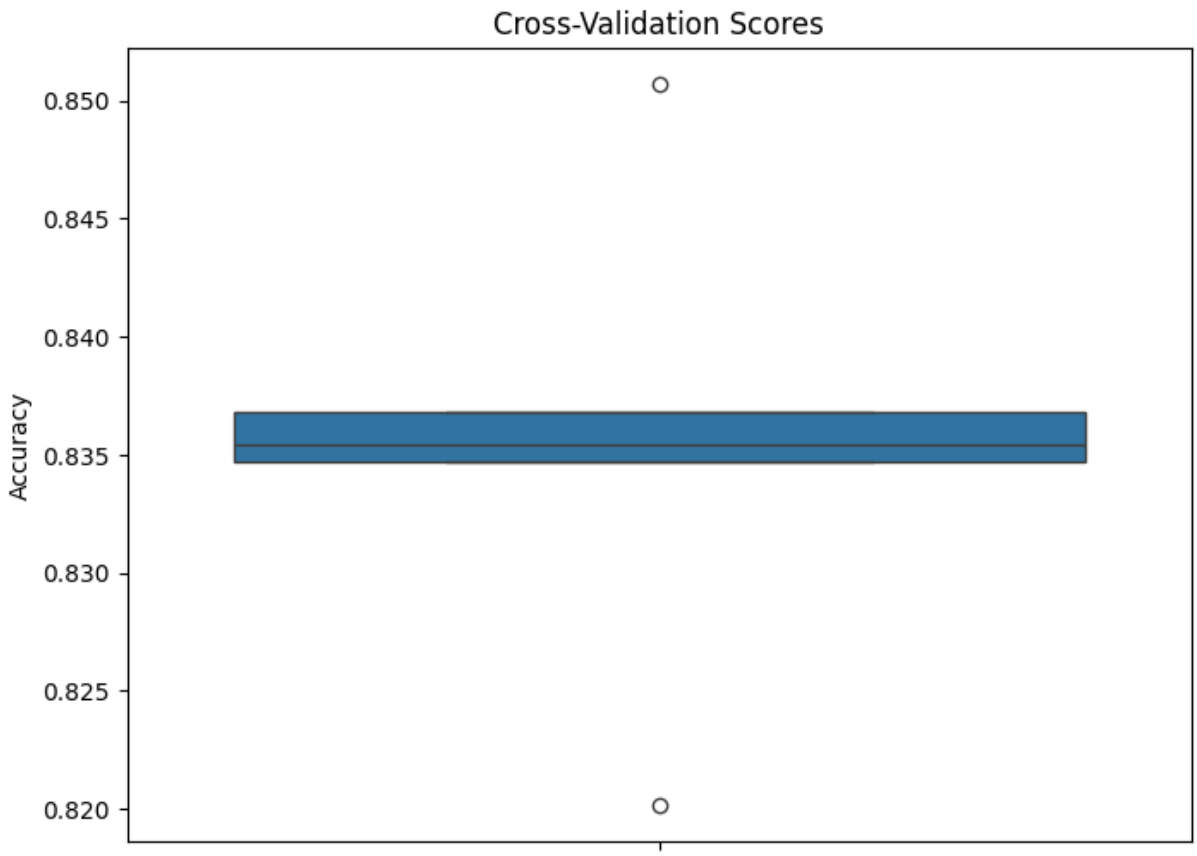
Bottom-Right (704): True Positives (TP)

- The model correctly predicted **704** instances as class 1, and they were 1. These are correct predictions for the positive class.



The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold

Random Forest AUC: 0.9213237087315902 is considered very good. This means that our random forest model has the ability to distinguish between the positive and negative classes. The orange curve is significantly far above the dashed blue diagonal line, confirming that our model performs much better than random guessing.



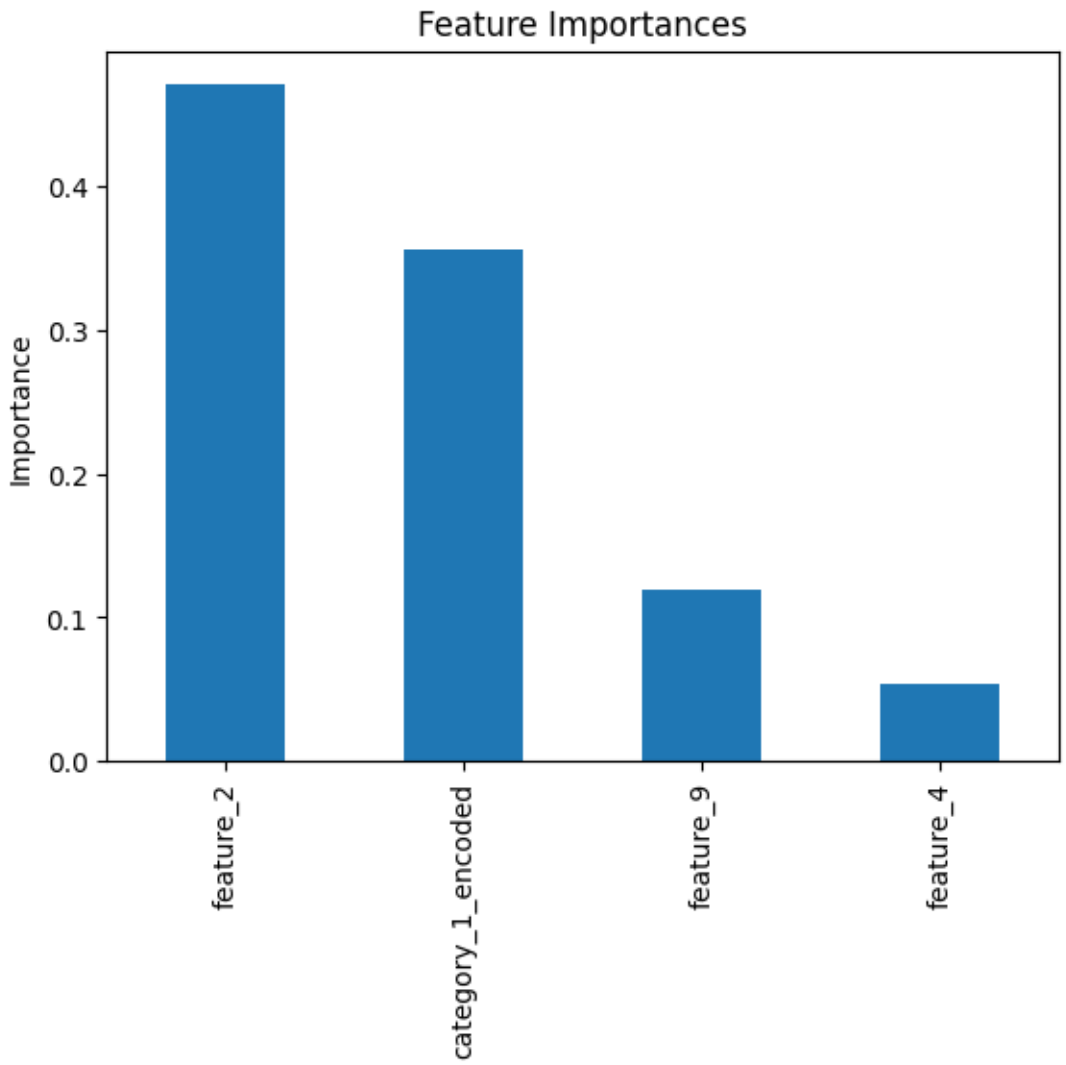
We apply cross validation to check for overfitting and to get a more reliable estimate of your model's performance on unseen data.

[0.82013889 0.83541667 0.83680556 0.85069444 0.83472222] is a list of individual scores obtained from 5-k folds cross validation since there are 5 scores.

This 0.8355555555555556 means the average of those 5 cross-validation scores

0.009686442096757054 is the standard deviation of the 5 cross-validation scores and the tight boxplot indicates that our model's performance is stable not overfit.

Gradient Boosting



Gradient Boosting also has built-in feature importance like Random Forest model.

Feature_2 remains the most important, category_1_encoded now becomes the second most important and then comes along feauture_9 and feature_4. Even though, feature_4 is the least important but it still constributes to the model's performance.

	precision	recall	f1-score	support
0.0	0.86	0.88	0.87	947
1.0	0.87	0.84	0.85	853
accuracy			0.86	1800
macro avg	0.86	0.86	0.86	1800
weighted avg	0.86	0.86	0.86	1800

Precision:

For class 0.0: 0.86 means when the model predicted an instance belonged to class 0, it was correct 86% of the time.

For class 1.0: 0.87. This means when the model predicted an instance belonged to class 1, it was also correct 87% of the time.

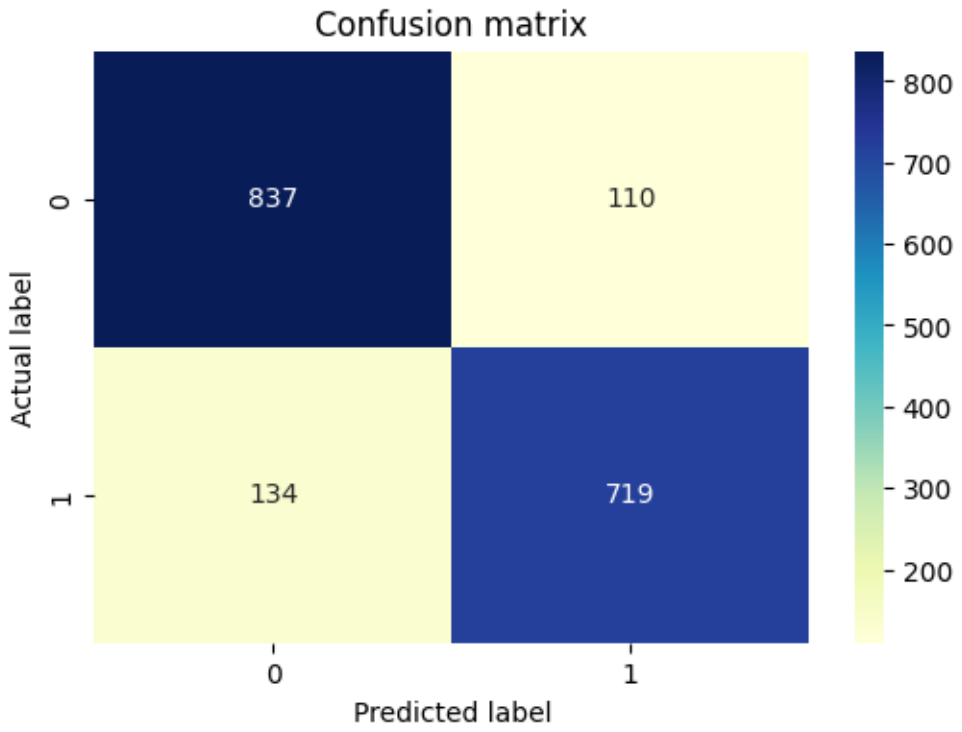
Recall:

For class 0.0: 0.88. This means the model correctly identified 88% of all actual instances of class 0.

For class 1.0: 0.84. This means the model correctly identified 84% of all actual instances of class 1.

F1-score: it is a harmony between precision and recall. It provides a single score that balances both precision and recall, useful when you need a balance between them.

Support: this is the actual number of occurrences (instances) of each class in the true labels of the data. **Class 0**.0 got 947 instances and class 1.0 got 853 instances



Top-Left (837): True Negatives (TN)

- The model correctly predicted **837** instances as class 0, and they were class 0. These are correct predictions for the negative class.

Top-Right (110): False Positives (FP)

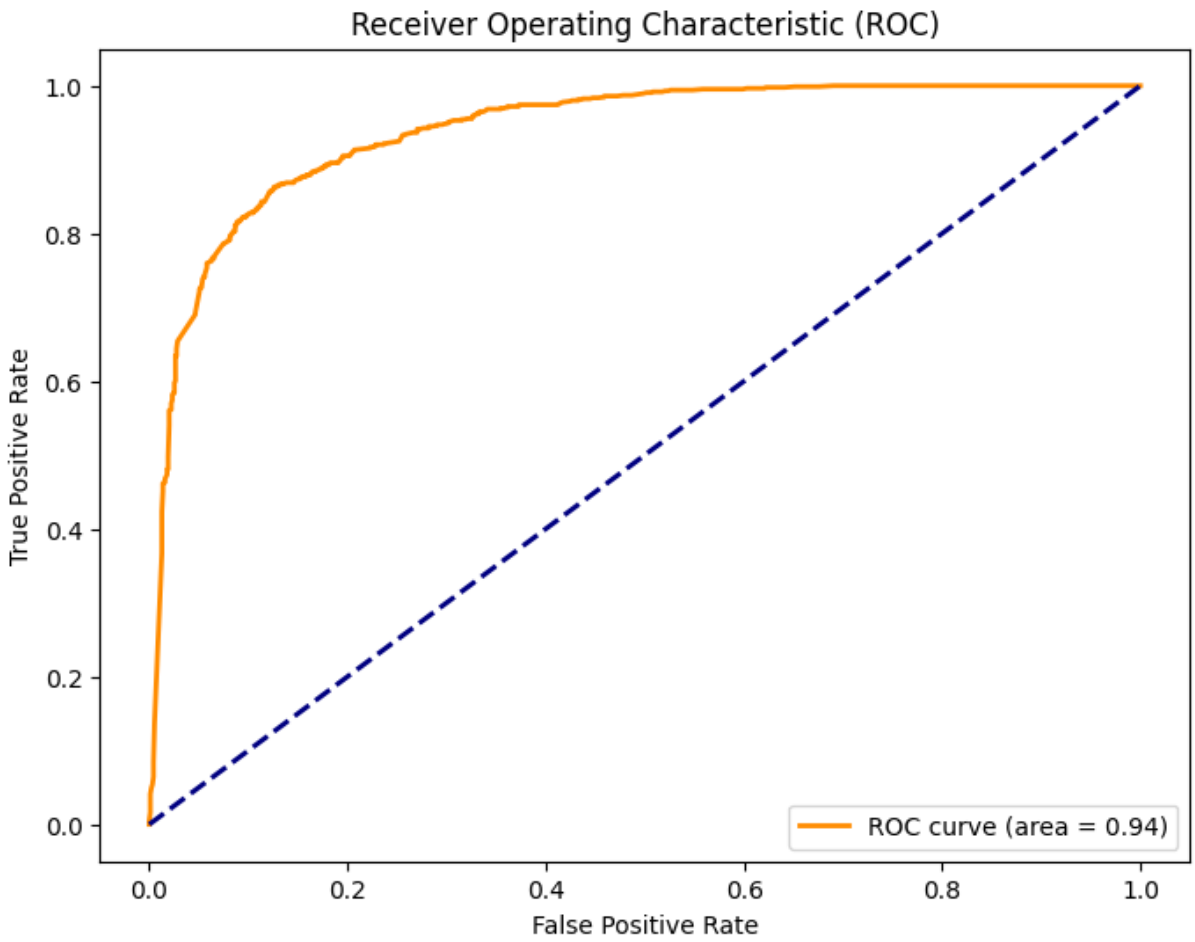
- The model incorrectly predicted **110** instances as class 1, but they were also class 0.

Bottom-Left (134): False Negatives (FN)

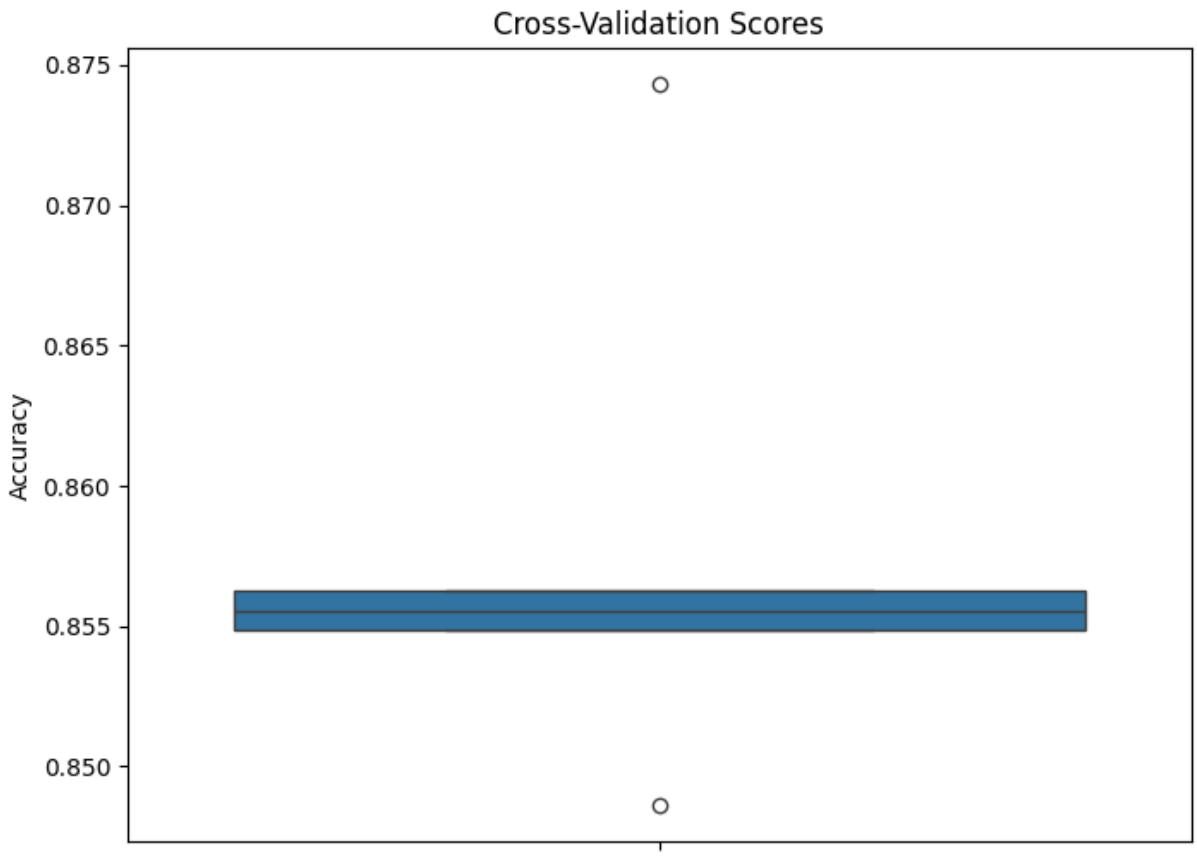
- The model incorrectly predicted **134** instances as class 0, but they were class 1.

Bottom-Right (719): True Positives (TP)

- The model correctly predicted **719** instances as class 1, and they were 1. These are correct predictions for the positive class.



Random Forest AUC: 0.9384847070591279 is also very good. This also means that our model can distinguish between the positive and negative classes. The orange curve is vastly far above the dashed blue diagonal line, confirming that our model performs much better than just randomly guessing.



We also apply cross-validation here to check for the risk of overfitting.

[0.84861111 0.85555556 0.85486111 0.87430556 0.85625] is a list of 5 k-folds cross-validation scores

0.8579166666666665 is the average of that list of cross-validation scores.

0.008635717531505464 is the standard deviation of the 5 cross-validation scores and the tight boxplot shows that our model's performance is stable and not overfit.

Voting Classifier

	precision	recall	f1-score	support
0.0	0.85	0.89	0.87	947
1.0	0.87	0.83	0.85	853
accuracy			0.86	1800
macro avg	0.86	0.86	0.86	1800
weighted avg	0.86	0.86	0.86	1800

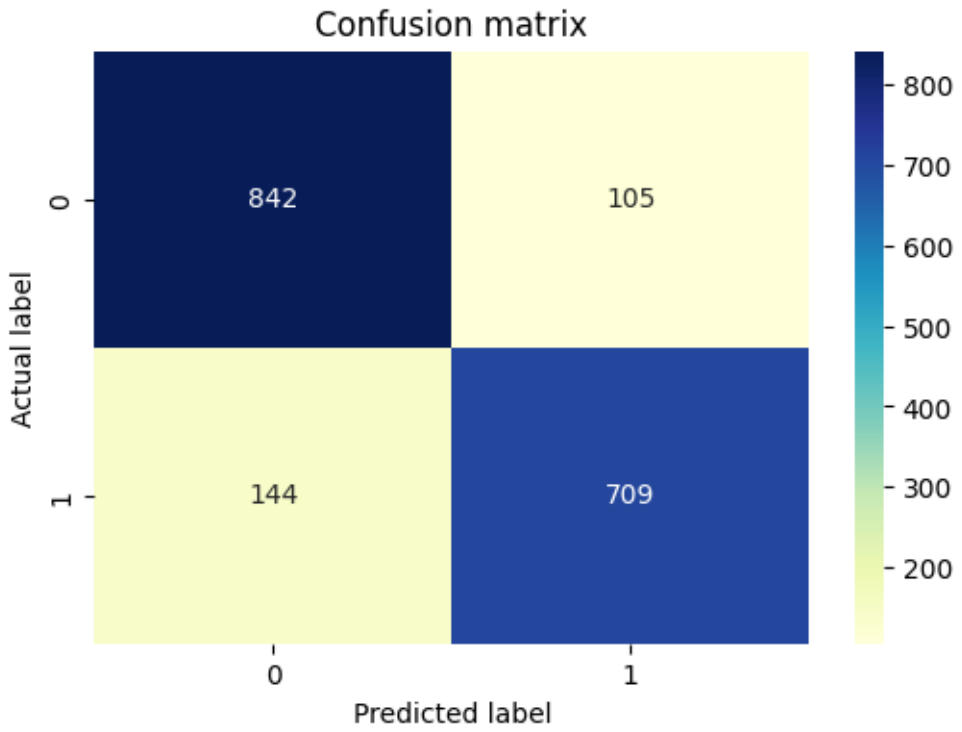
Recall:

For class 0.0: 0.85. This means the model correctly identified 85% of all actual instances of class 0.

For class 1.0: 0.87. This means the model correctly identified 87% of all actual instances of class 1.

F1-score: it is a harmony between precision and recall. It provides a single score that balances both precision and recall, useful when you need a balance between them.

Support: this is the actual number of occurrences (instances) of each class in the true labels of the data. **Class 0**.0 got 947 instances and class 1.0 got 853 instances



Top-Left (842): True Negatives (TN)

- The model correctly predicted **842** instances as class 0, and they were class 0. These are correct predictions for the negative class.

Top-Right (105): False Positives (FP)

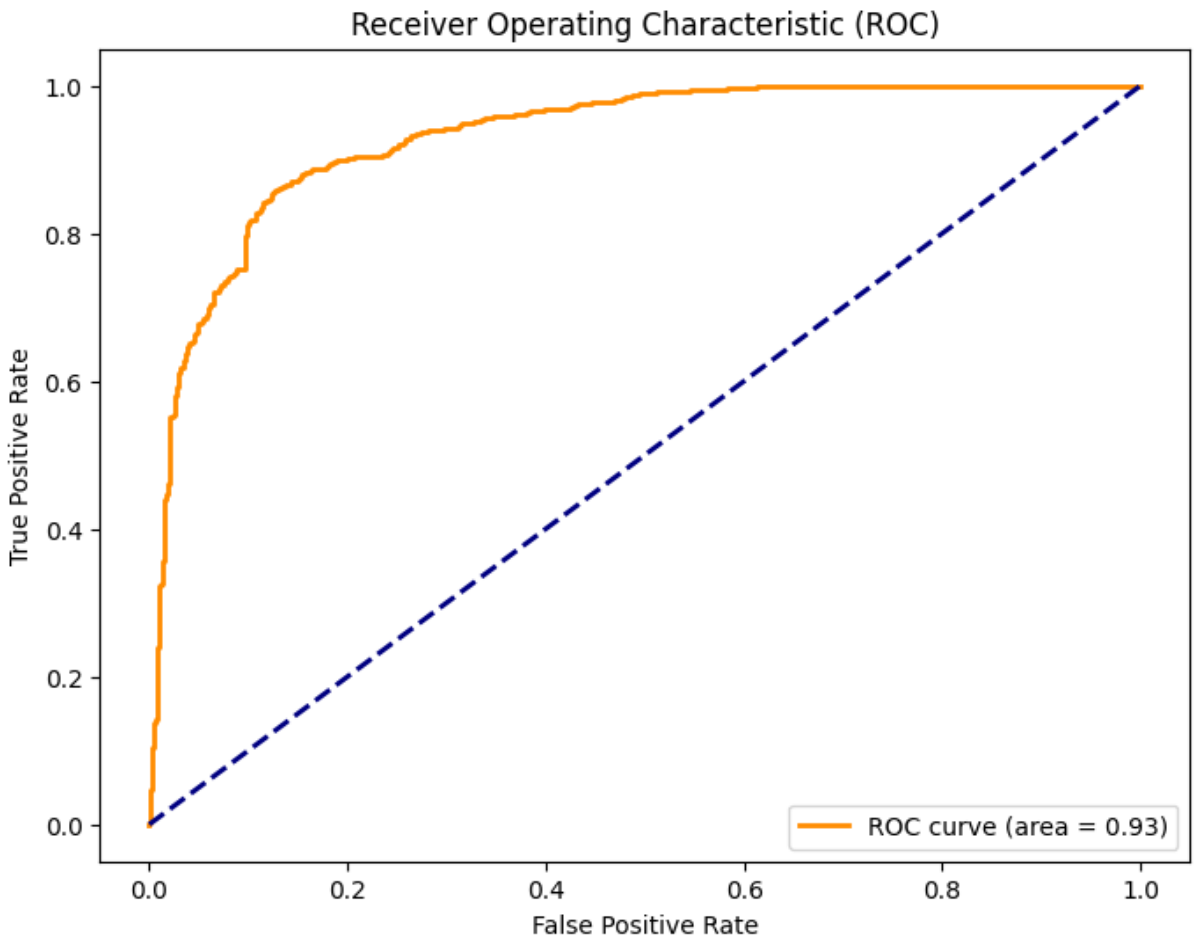
- The model incorrectly predicted **105** instances as class 1, but they were also class 0.

Bottom-Left (144): False Negatives (FN)

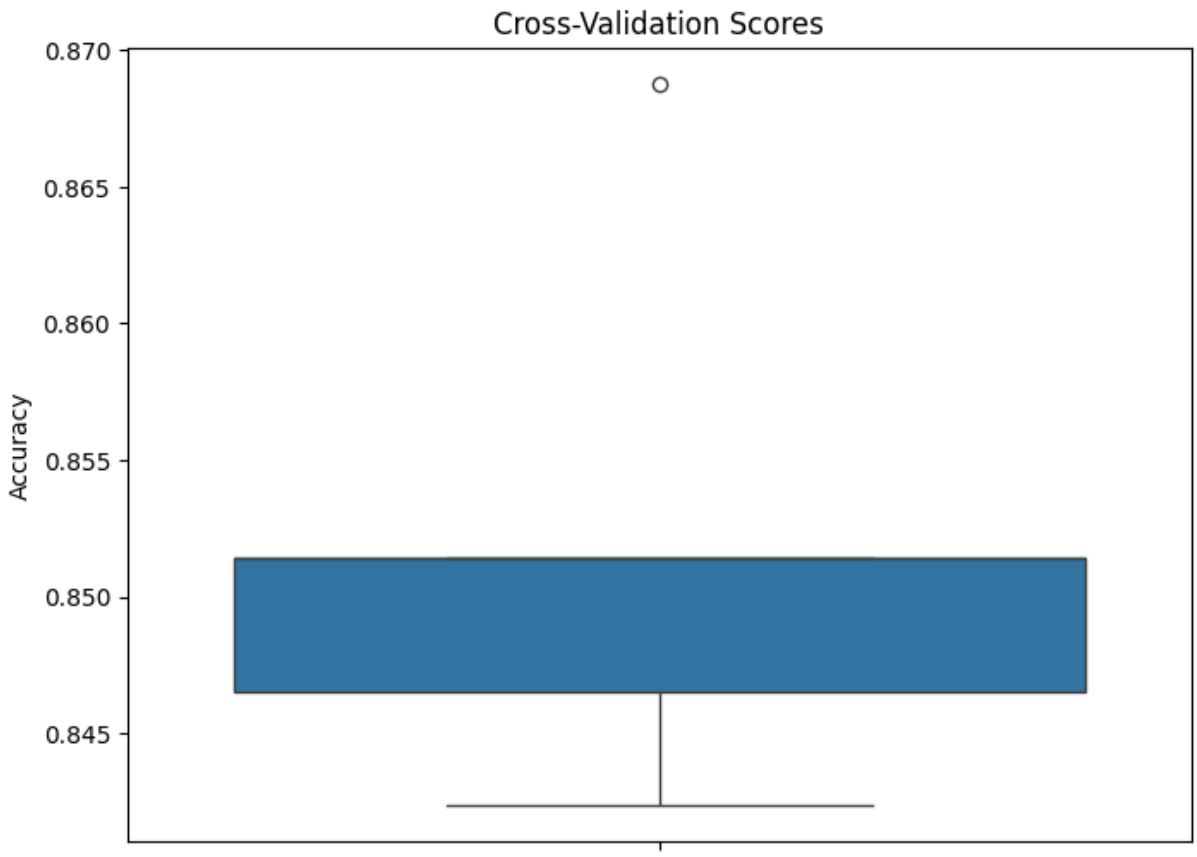
- The model incorrectly predicted **144** instances as class 0, but they were class 1.

Bottom-Right (709): True Positives (TP)

- The model correctly predicted **709** instances as class 1, and they were 1. These are correct predictions for the positive class.



Random Forest AUC: 0.9317187242739768 is also impressive. This also means that this model can distinguish between the positive and negative classes. The orange curve is vastly far above the dashed blue diagonal line, confirming that the model performs very good.



[0.84236111 0.84652778 0.85138889 0.86875 0.85138889] are the individual accuracy scores for each of the **5 folds** (or iterations) of the cross-validation.

0.8520833333333334 is the average (mean) of the 5 individual cross-validation scores.

0.008990306851490153 is the **standard deviation** of the 5 cross-validation scores indicating that model's performance is very consistent. There is no sign of overfitting, and it means that it generalizes well to unseen data

The box is quite narrow, and the spread of the points is relatively small. This visually reinforces the small standard deviation.

6.1 Hyperparameter tuning

Besides default models, we also have models with hyperparameter tuning. Those are basically customized pre-built models to make it perform better as we want.

We use **GridSearchCV** and **RandomizedSearchCV** for hyperparameter tuning. These techniques will find the best combination of hyperparameters for our model to optimize its performance.

Here is the comparison of the performance of the tuning models:

Model	Accuracy
Random Forest	0.863
Gradient Boosting	0.863
Voting Classifier	0.860

Random Forest

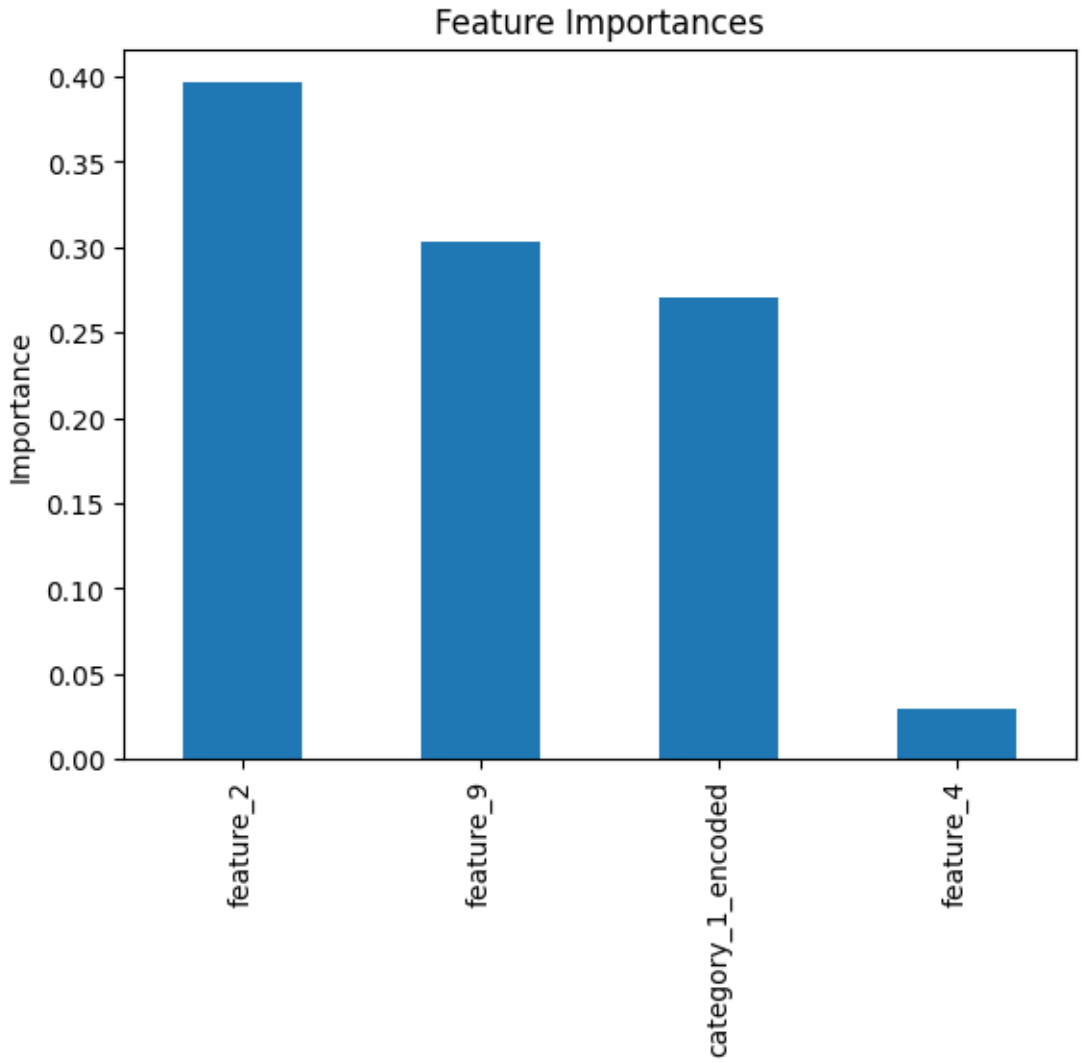
```
23m
#Random Forest Classifier Tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5,10,15],
    'min_samples_split': [2,5,10],
    'min_samples_leaf': [1,2,4],
    'max_leaf_nodes': [5,10,15]
}
```

```
rf_tuned = RandomForestClassifier(random_state = 42)
grid_search = GridSearchCV(estimator=rf_tuned, param_grid=param_grid,cv=5)
```

```
Best Parameters: {'max_depth': 10, 'max_leaf_nodes': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
```

In this model, we apply GridSearchCV, it systematically explores every possible combination of hyperparameter values defined in a grid. For each combination, it trains and evaluates the model using cross-validation.

It then shows the list of best parameters that our model can use to optimize performance.



This is also the feature importance that we have from this Random Forest tuning model.

Feature_2 remains the most important and feature_4 has been greatly reduced in its importance. Feature_9 and category_1_encoded remain very relevant to the model's performance.

	precision	recall	f1-score	support
0.0	0.86	0.89	0.87	947
1.0	0.87	0.84	0.85	853
accuracy			0.86	1800
macro avg	0.86	0.86	0.86	1800
weighted avg	0.86	0.86	0.86	1800

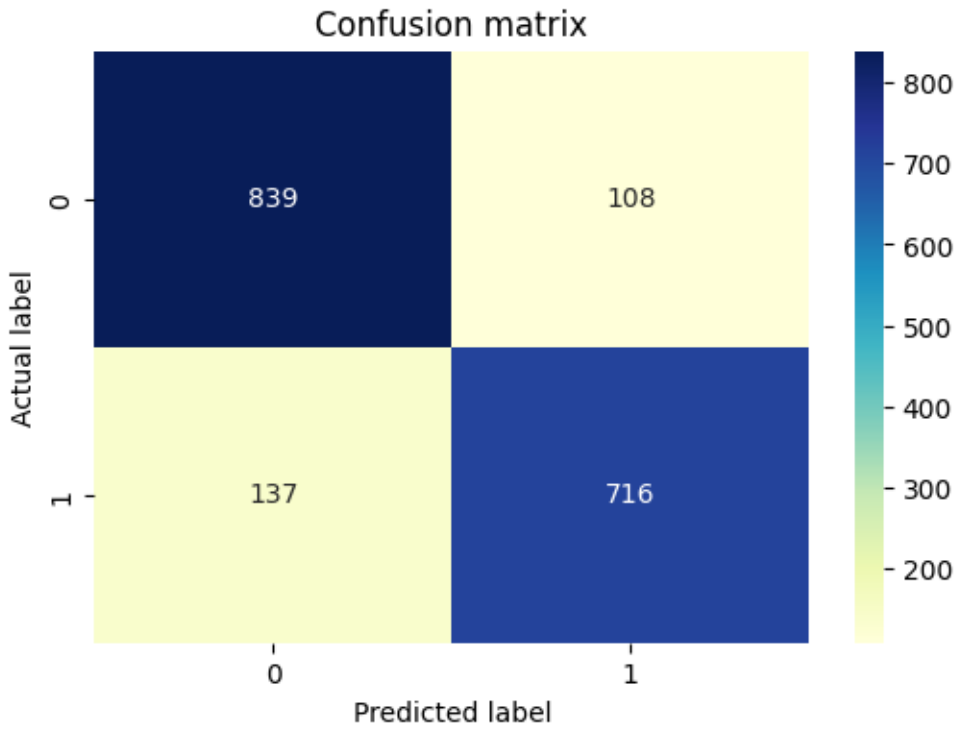
Recall:

For class 0.0: 0.86. This means the model correctly identified 86% of all actual instances of class 0.

For class 1.0: 0.87. This means the model correctly identified 87% of all actual instances of class 1.

F1-score: it is a harmony between precision and recall. It provides a single score that balances both precision and recall, useful when you need a balance between them.

Support: this is the actual number of occurrences (instances) of each class in the true labels of the data. **Class 0**.0 got 947 instances and class 1.0 got 853 instances



Top-Left (839): True Negatives (TN)

- The model correctly predicted **839** instances as class 0, and they were class 0. These are correct predictions for the negative class.

Top-Right (108): False Positives (FP)

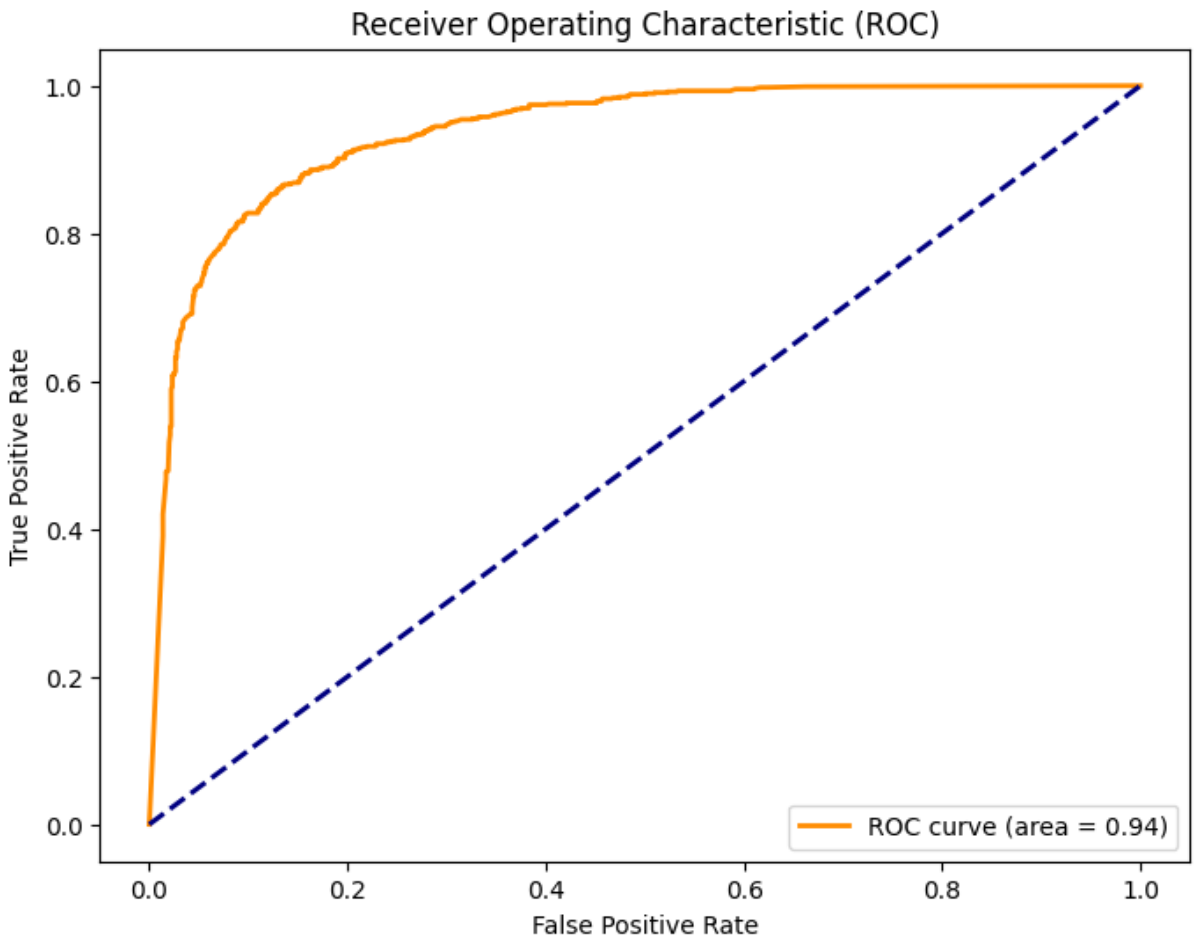
- The model incorrectly predicted **108** instances as class 1, but they were also class 0.

Bottom-Left (137): False Negatives (FN)

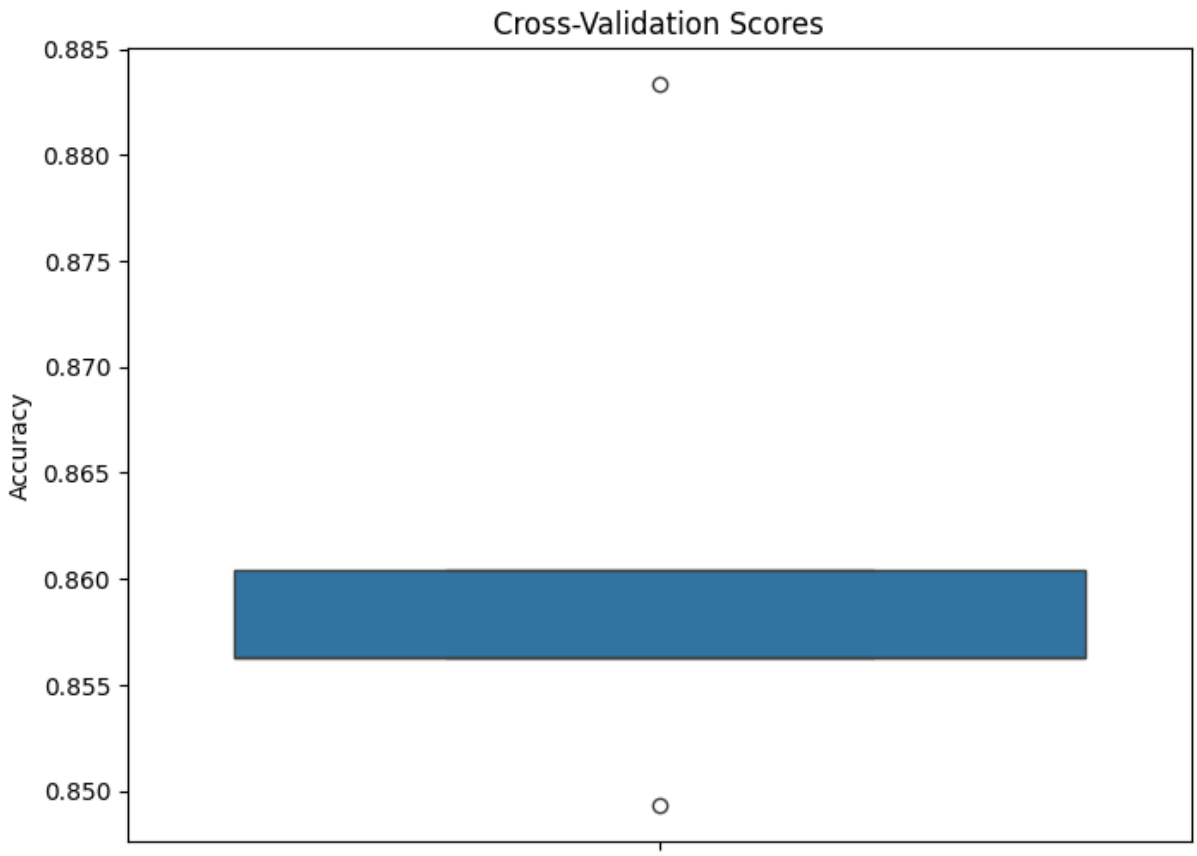
- The model incorrectly predicted **137** instances as class 0, but they were class 1.

Bottom-Right (716): True Positives (TP)

- The model correctly predicted **716** instances as class 1, and they were 1. These are correct predictions for the positive class.



Random Forest AUC: 0.9376552845971298 is very high, meaning that this model can distinguish between the positive and negative classes. The orange curve is far above the dashed blue diagonal line, confirming that the model performs very good.



[0.84930556 0.85625 0.85625 0.88333333 0.86041667] is a list of each 5-k folds cross-validation scores.

0.861111111111111 is the average of that list of scores

0.011669973076444445 is a very small standard deviation, which is a good sign of model stability.

The compact nature of the box (tight IQR) suggests that most scores are very close to the mean, confirming the model's overall stability.

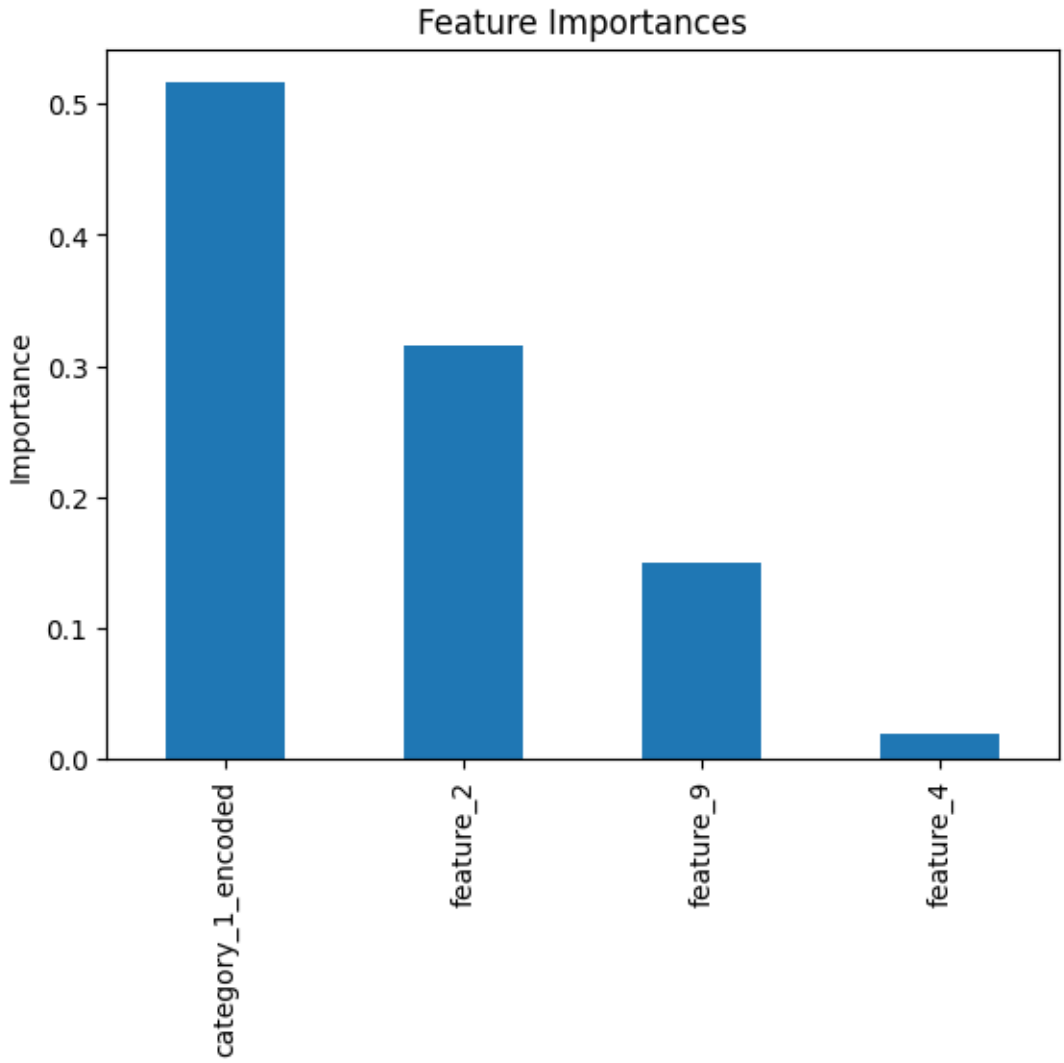
Gradient Boosting

```python
#Gradient Boosting Tuning
from sklearn.model_selection import RandomizedSearchCV
gb_param = {
    'n_estimators': [100,200,300],
    'learning_rate': [0.1, 0.01, 0.001],
    'min_samples_split':[2,5,10],
    'min_samples_leaf': [1,2,4],
    'max_depth':[5,10,15]
}
```

```python
gb_tuning = GradientBoostingClassifier()
gb_random_search = RandomizedSearchCV(estimator=gb_tuning, param_distributions=gb_param, cv=5, n_iter=30, random_state=42,
```

```
Best Parameters: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': 5, 'learning_rate': 0.01}
Best CV score: 0.8598611111111112
```

In this case, we apply RandomizedSearchCV technique to this hyperparameter tuning process because the GridSearchCV works very badly on this model (it took over 3 hours but still didn't provide anything). RandomizedSearchCV offers a more efficient alternative way by taking a random sampling approach to the hyperparameter space. It will randomly sample n_iter sets of parameters from the distribution we defined.



This is the feature importance of this model and we can see the huge change is that category_1_encoded is now the most important feature. Feature_2 is the second most important, meanwhile, feature_4 is the least important feature.

	precision	recall	f1-score	support
0.0	0.85	0.89	0.87	947
1.0	0.88	0.83	0.85	853
accuracy			0.86	1800
macro avg	0.86	0.86	0.86	1800
weighted avg	0.86	0.86	0.86	1800

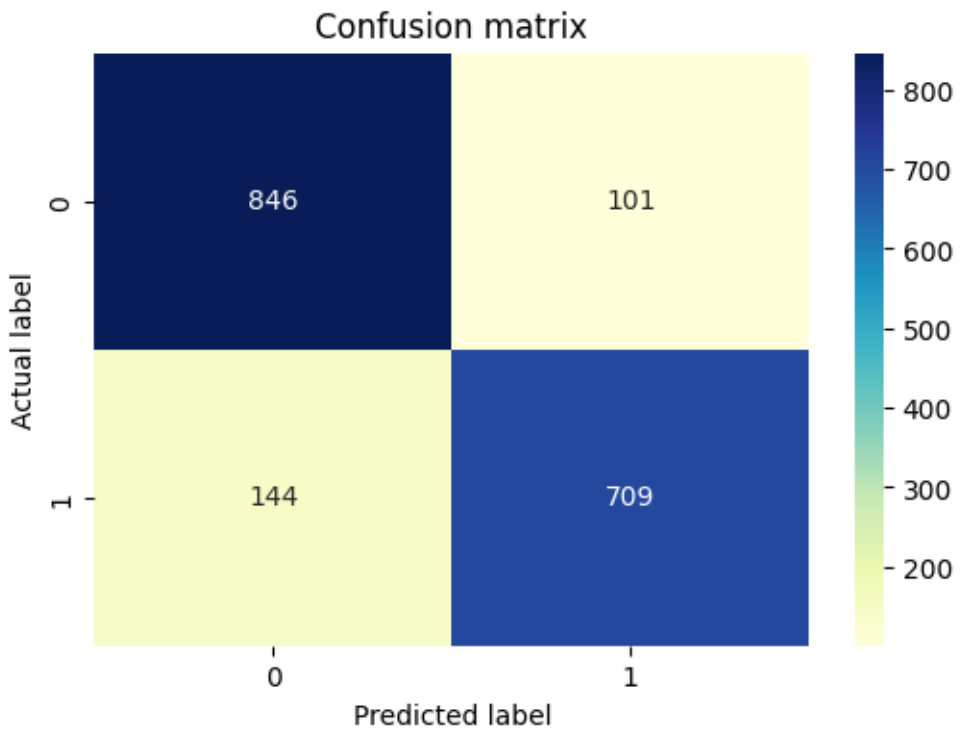
Recall:

For class 0.0: 0.85. This means the model correctly identified 85% of all actual instances of class 0.

For class 1.0: 0.88. This means the model correctly identified 88% of all actual instances of class 1.

F1-score: it is a harmony between precision and recall. It provides a single score that balances both precision and recall, useful when you need a balance between them.

Support: this is the actual number of occurrences (instances) of each class in the true labels of the data. **Class 0**.0 got 947 instances and class 1.0 got 853 instances



Top-Left (846): True Negatives (TN)

- The model correctly predicted **846** instances as class 0, and they were class 0. These are correct predictions for the negative class.

Top-Right (101): False Positives (FP)

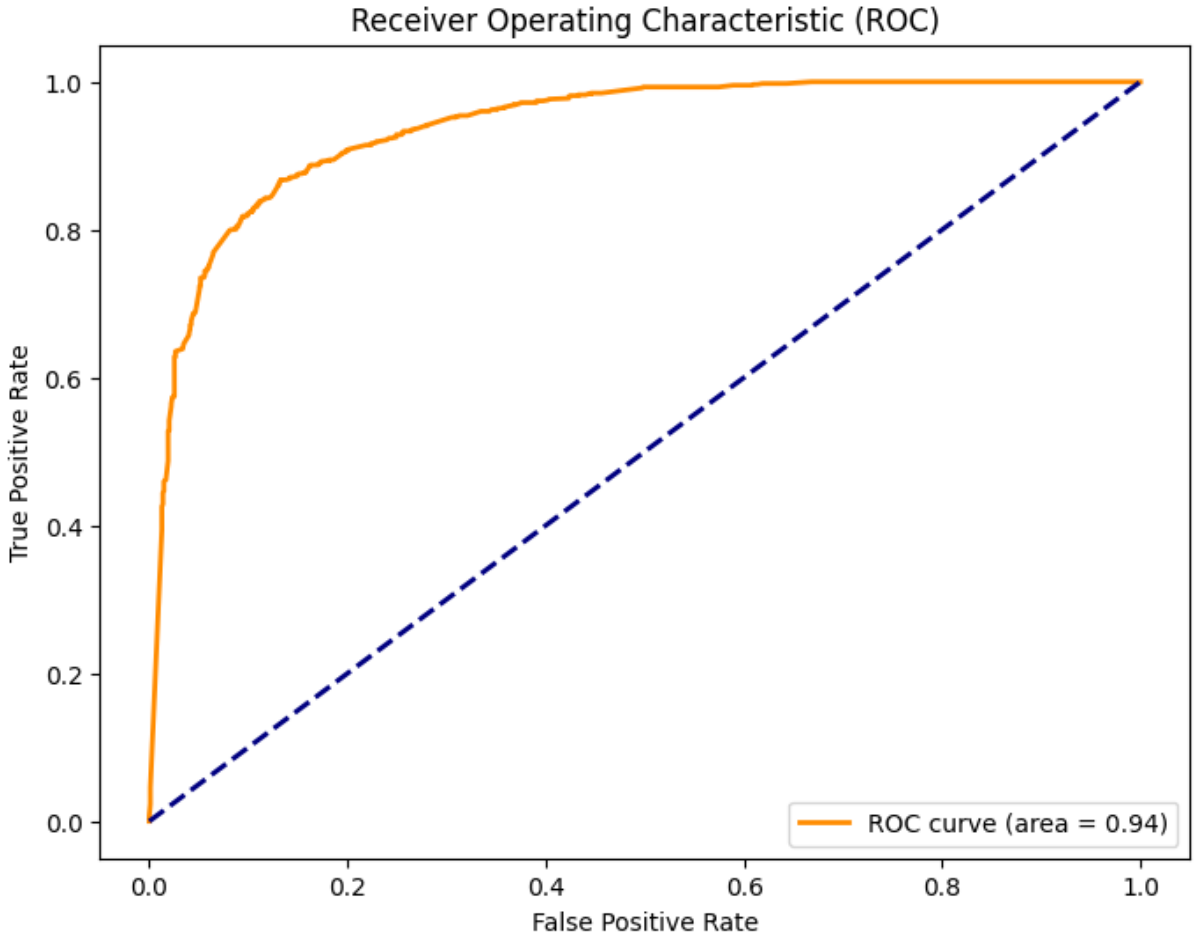
- The model incorrectly predicted **101** instances as class 1, but they were also class 0.

Bottom-Left (144): False Negatives (FN)

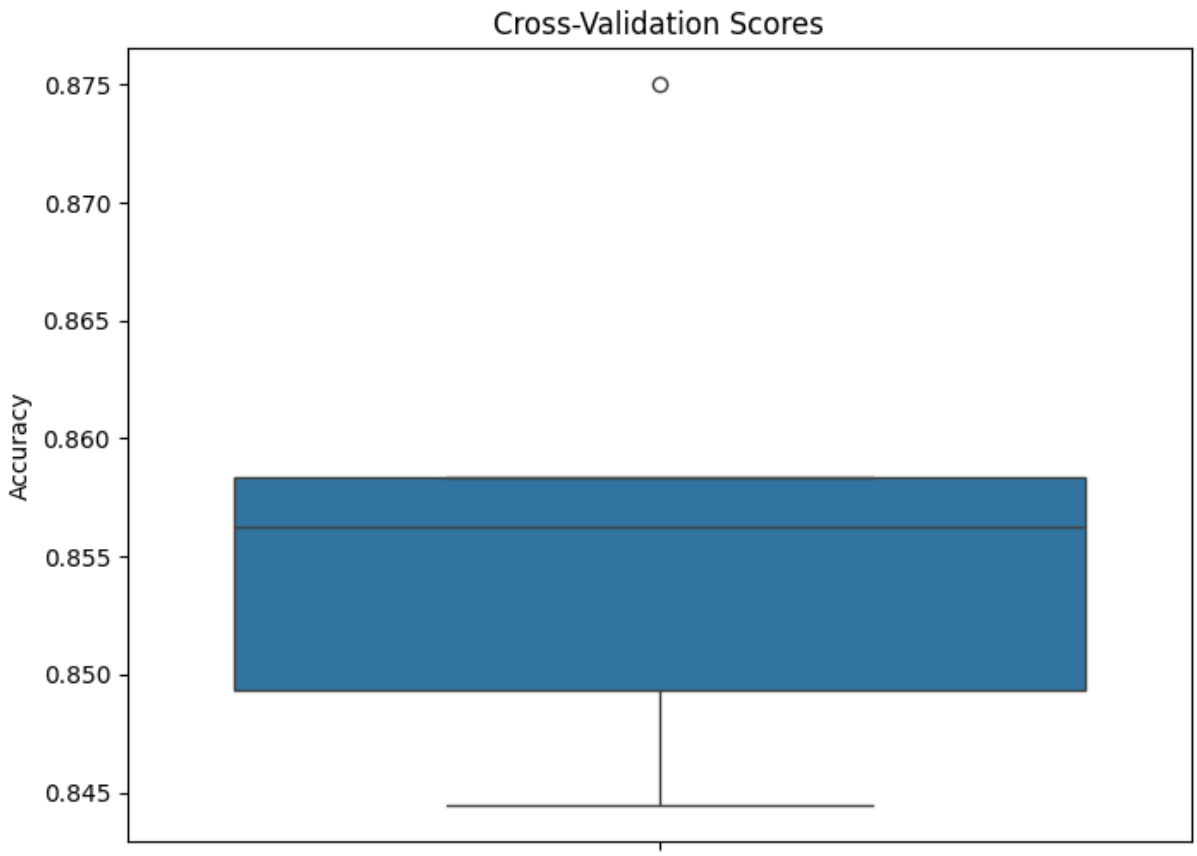
- The model incorrectly predicted **144** instances as class 0, but they were class 1.

Bottom-Right (709): True Positives (TP)

- The model correctly predicted **709** instances as class 1, and they were 1. These are correct predictions for the positive class.



Random Forest AUC: 0.9381597467661809 is quite high indicating that the model can differentiate between the positive and negative classes. The orange curve is far above the dashed blue diagonal line, confirming that the model performs very good.



[0.84444444 0.85625 0.84930556 0.875 0.85833333] is the 5 k-folds cross validation scores

0.8566666666666667 is the mean (average) of those 5 k-folds cross validation scores

0.01041759255144399 is the standard deviation, which is very small, meaning that the model's performance is robust and stable.

Voting Classifier

```
#Voting Classifier Tuning
vt_params = {
    'lr__C': [0.1, 1, 10],
    'svc__C': [0.1,1,10],
    'dt__max_depth':[5,10,15],
    'dt__min_samples_split':[2,5,10],
    'dt__min_samples_leaf': [1,2,4]

}
vt_tuning = VotingClassifier(estimators=[('lr',clf1),('svc',clf2),('dt',clf3)],voting='soft')
vt_random_search = RandomizedSearchCV(estimator=vt_tuning, param_distributions=vt_params)
```

```
Best Parameters: {'svc__C': 10, 'lr__C': 1, 'dt__min_samples_split': 10, 'dt__min_samples_leaf': 4, 'dt__max_depth': 10}
Best CV score: 0.8584722222222222
```

Voting Classifier always takes advantage of RandomizedSearchCV for its tuning. Those are the best parameters for this model to be as optimized as possible.

	precision	recall	f1-score	support
0.0	0.87	0.86	0.87	947
1.0	0.85	0.86	0.85	853
accuracy			0.86	1800
macro avg	0.86	0.86	0.86	1800
weighted avg	0.86	0.86	0.86	1800

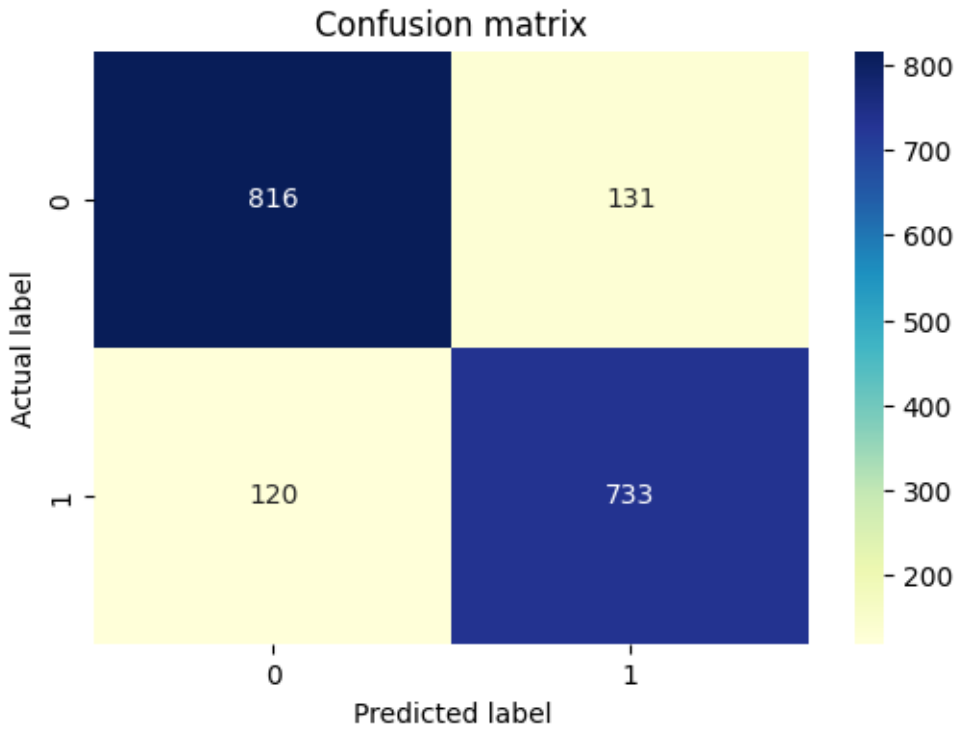
Recall:

For class 0.0: 0.87. This means the model correctly identified 87% of all actual instances of class 0.

For class 1.0: 0.85. This means the model correctly identified 85% of all actual instances of class 1.

F1-score: it is a harmony between precision and recall. It provides a single score that balances both precision and recall, useful when you need a balance between them.

Support: this is the actual number of occurrences (instances) of each class in the true labels of the data. **Class 0**.0 got 947 instances and class 1.0 got 853 instances



Top-Left (816): True Negatives (TN)

- The model correctly predicted **816** instances as class 0, and they were class 0. These are correct predictions for the negative class.

Top-Right (131): False Positives (FP)

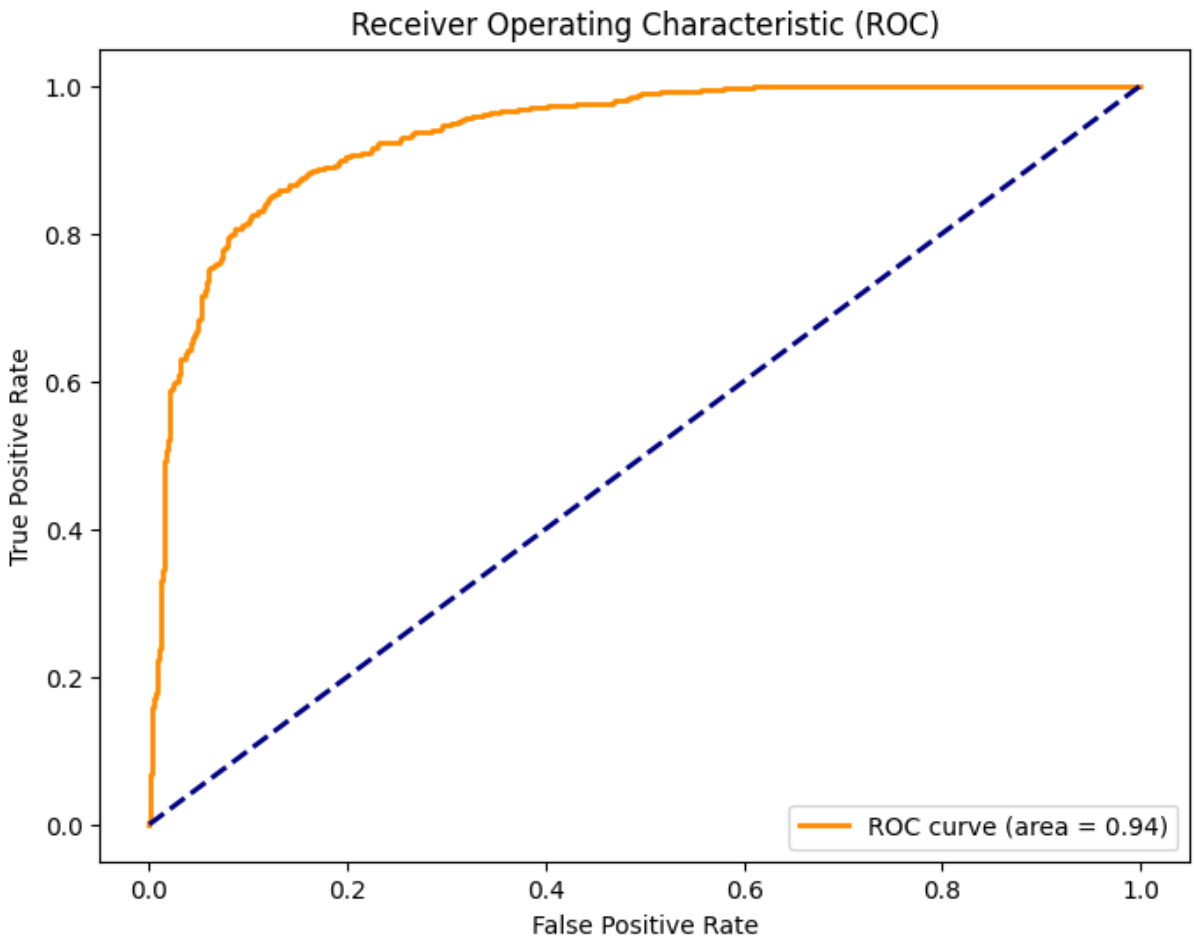
- The model incorrectly predicted **131** instances as class 1, but they were also class 0.

Bottom-Left (120): False Negatives (FN)

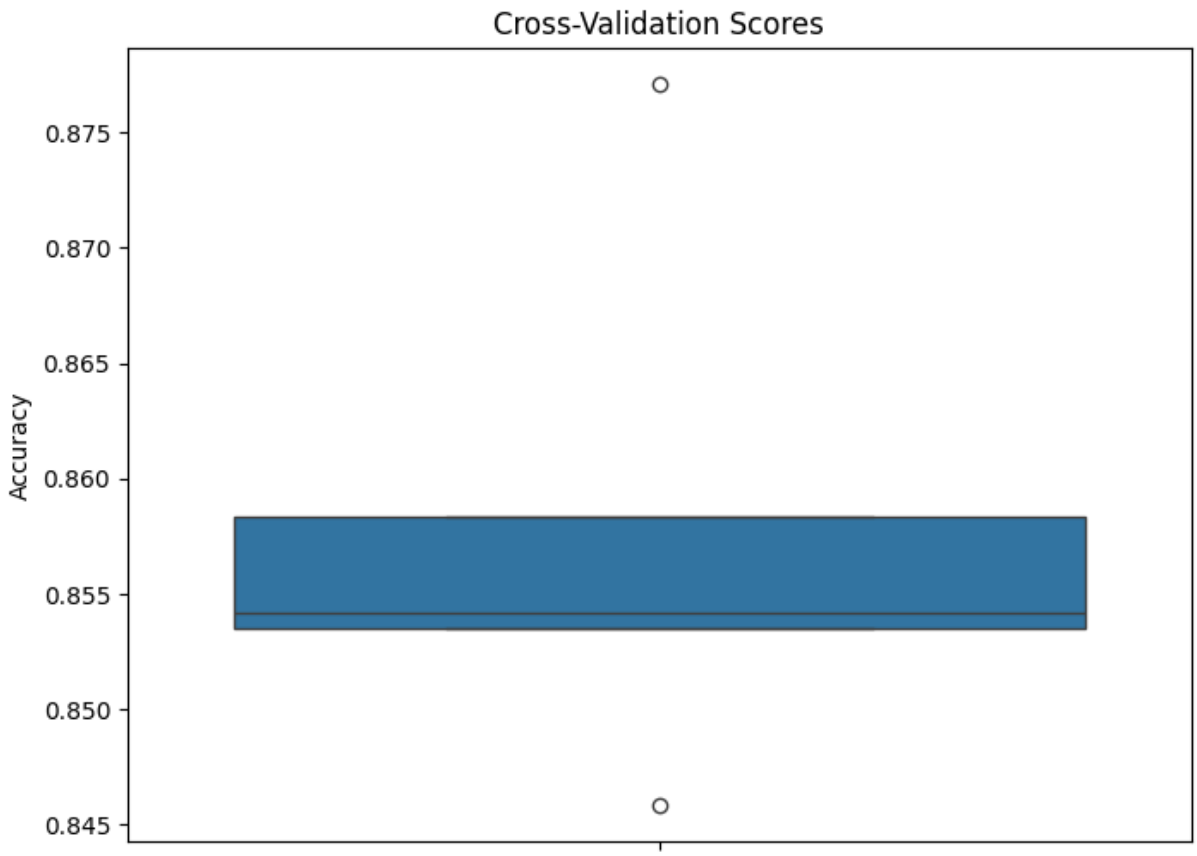
- The model incorrectly predicted **120** instances as class 0, but they were class 1.

Bottom-Right (733): True Positives (TP)

- The model correctly predicted **733** instances as class 1, and they were 1. These are correct predictions for the positive class.



Random Forest AUC: 0.9353483760031988 means that the model's performance is very good and effective. The orange curve is far above the dashed blue diagonal line, confirming that the model performs very impressive.



[0.84583333 0.85347222 0.85416667 0.87708333 0.85833333] is a list of 5 k-folds cross validation scores

0.8577777777777778 is the mean of its 5 k-folds scores

0.010461938658481383 means that the standard deviation is very small, indicating the stable and robust performance that the model achieves.

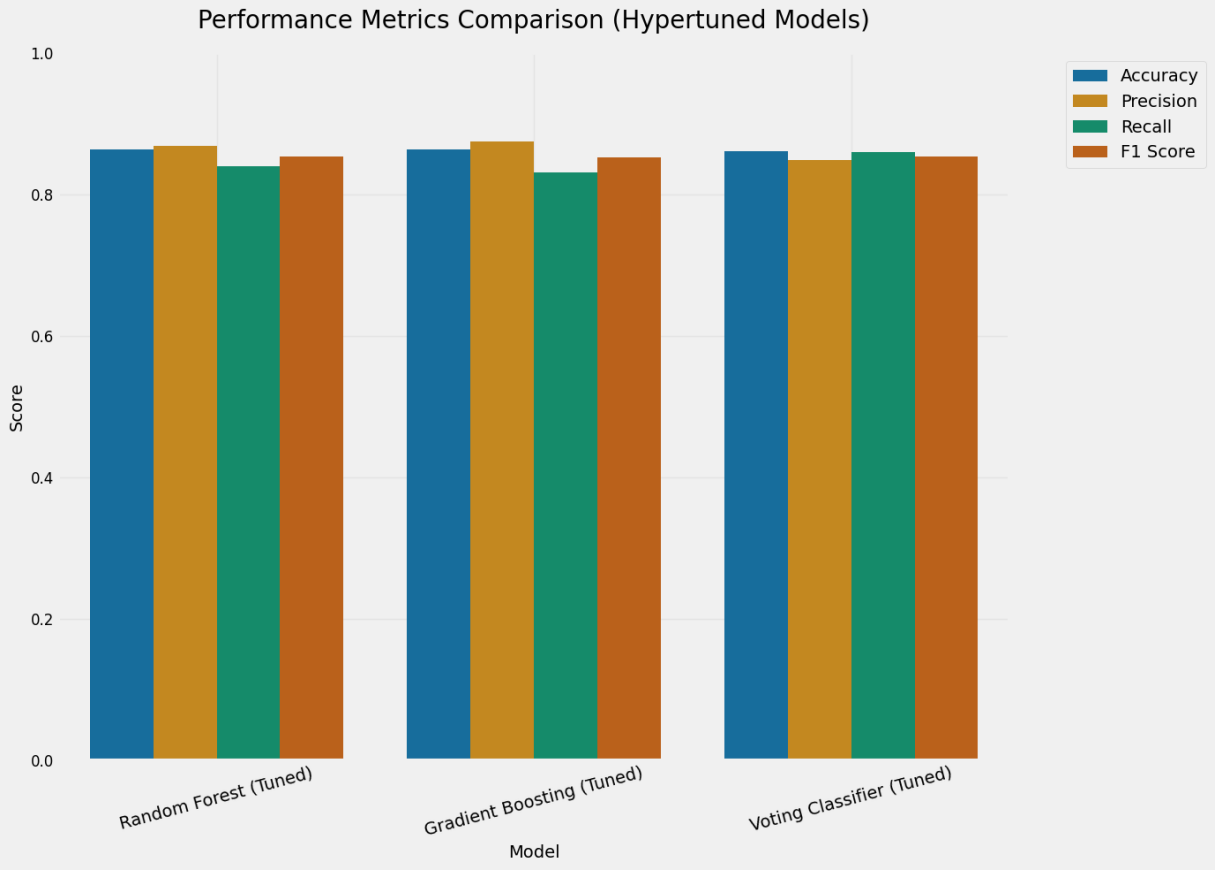
6.2 Best Tuned Models

Based on the result of the hyperparameter tuning process, Gradient Boosting model turns out to be the best tuned model out of all three models.

1. **Performance Metrics**:
    - Accuracy: 0.863 (tied with Random Forest)
    - AUC-ROC: 0.938 (highest among all models)
    - Balanced precision and recall across both classes (0.85-0.88)
2. **Feature Importance**:
    - The tuned model showed category_1_encoded as the most important feature, followed by feature_2
    - This suggests the model effectively learned to prioritize the most predictive features
3. **Stability**:
    - Cross-validation scores showed low standard deviation (0.0104)
    - Consistent performance across all folds (scores between 0.844-0.875)
4. **Efficient Tuning**:
    - Used RandomizedSearchCV which proved more efficient than GridSearchCV for this model

- Found optimal parameters in reasonable time (unlike GridSearchCV which failed to complete in over 3 hours)

Model	Accuracy	AUC ROC	Training Time
Random Forest	0.863	0.937	Moderate
Gradient Boosting	**0.863**	**0.938**	**Efficient**
Voting Classifier	0.860	0.935	Longest



Random Forest (Tuned):

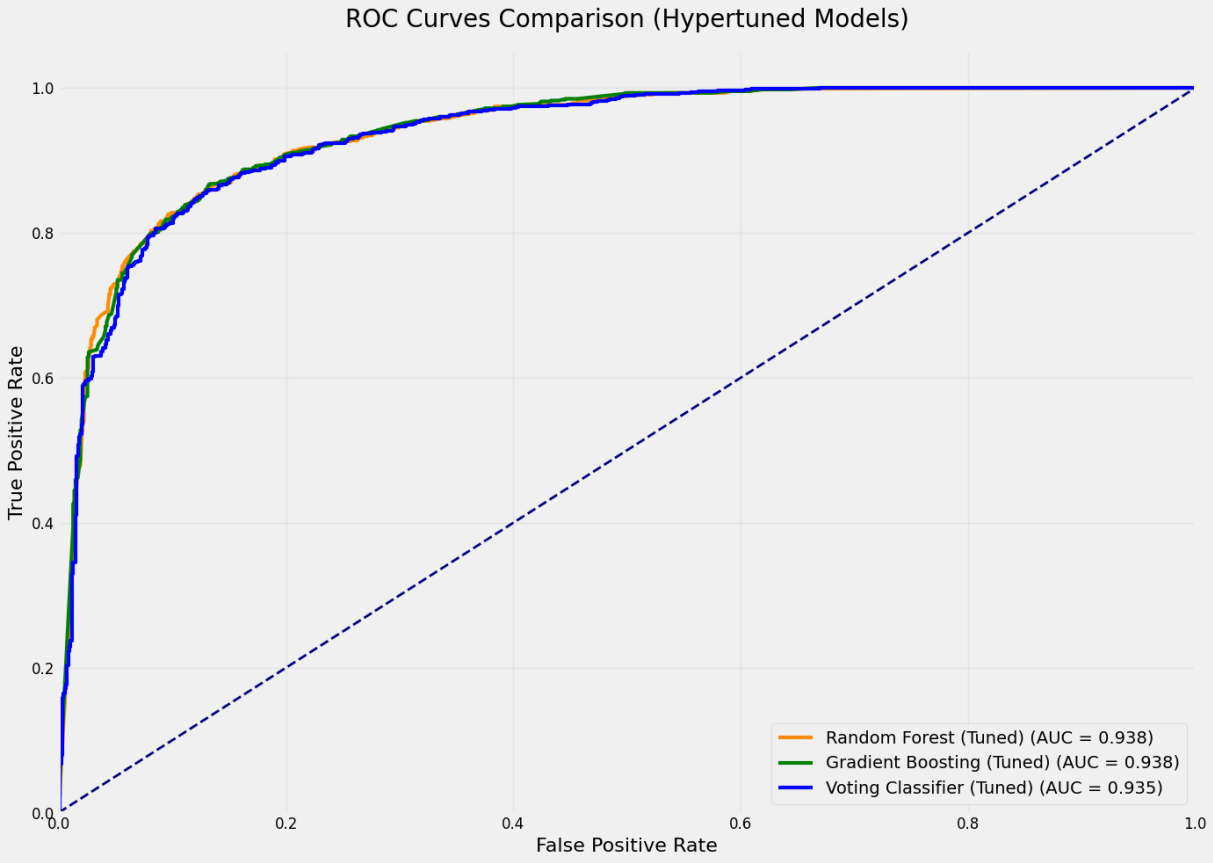
- Accuracy, Precision, Recall, and F1 Score are all very close, generally in the range of 0.85-0.87.
- Compared to the first "Performance Metrics Comparison" chart (which showed untuned models), it appears that Random Forest's performance has slightly improved across all metrics, with scores consistently staying above 0.85.

Gradient Boosting (Tuned):

- Accuracy, Precision, Recall, and F1 Score are all very high, with Precision being slightly above 0.87, and others around 0.86-0.87.
- Comparing to the untuned Gradient Boosting, the tuned version shows consistent high performance, perhaps maintaining or slightly improving its already strong scores. Precision appears to be particularly strong.

Voting Classifier (Tuned):

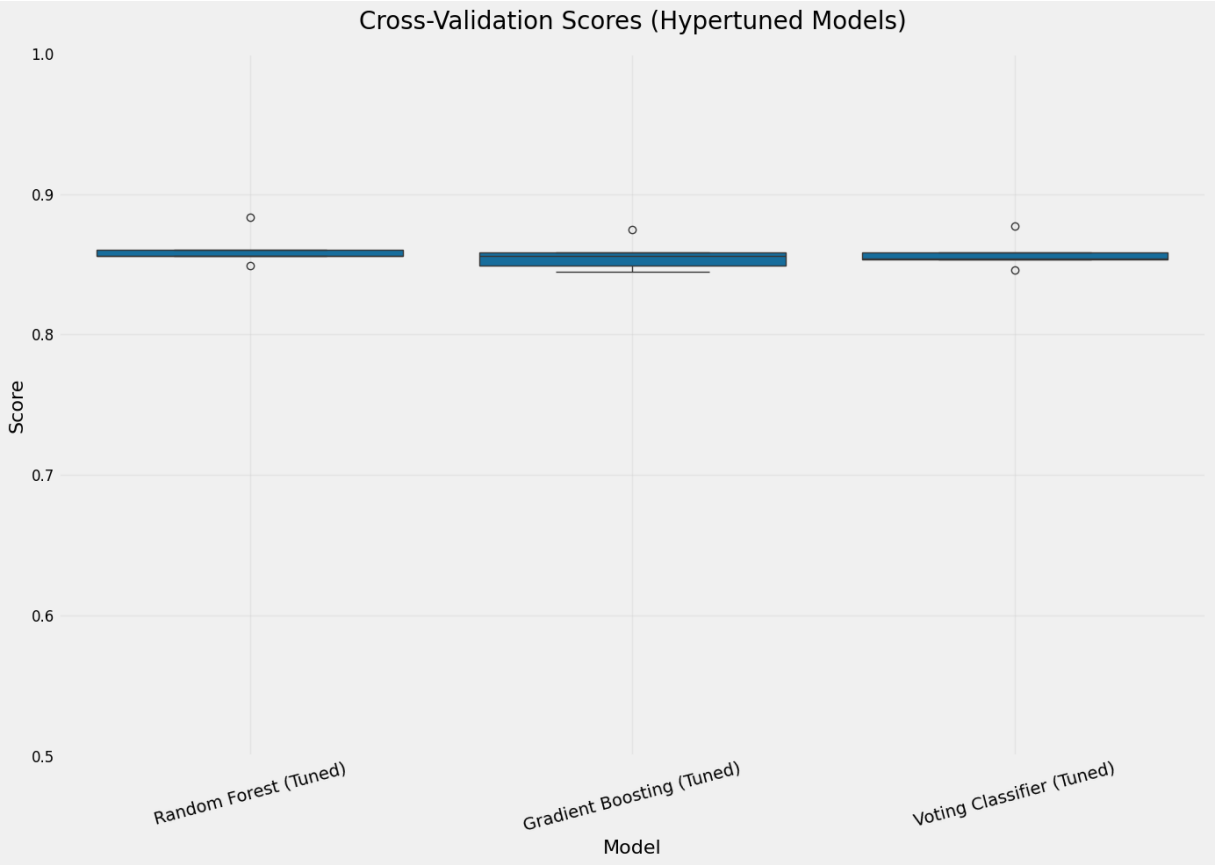
- Accuracy, Precision, Recall, and F1 Score are all very tight and high, generally around 0.86.
- Similar to Gradient Boosting, the tuned Voting Classifier maintains its very high performance. It seems to have very balanced metrics, with Precision, Recall, and F1 Score almost identical.



Significant Improvement for Random Forest: This is the most striking observation. The AUC for Random Forest has increased significantly from 0.921 to 0.938 after tuning. This indicates that hyperparameter optimization has greatly improved Random Forest's ability to discriminate between classes, bringing it up to the level of Gradient Boosting.

Gradient Boosting Maintains Excellent Performance: Gradient Boosting's AUC remains at 0.938, which was already very high. This suggests that its untuned performance was already close to optimal, or the tuning process simply confirmed its robust nature.

Voting Classifier Consistent: The Voting Classifier's AUC remains at 0.935. This indicates its robust performance, and while tuning might have refined its internal components, its overall discriminative power as measured by AUC stayed consistent.



Random Forest (Tuned):

- The median score appears to have shifted slightly upward, now clearly above 0.85 (closer to 0.86).
- The box remains very narrow, indicating excellent consistency.
- Still shows a couple of outliers, but their presence is normal in cross-validation.
- Improvement*:* The overall level of scores has slightly increased compared to its untuned version.

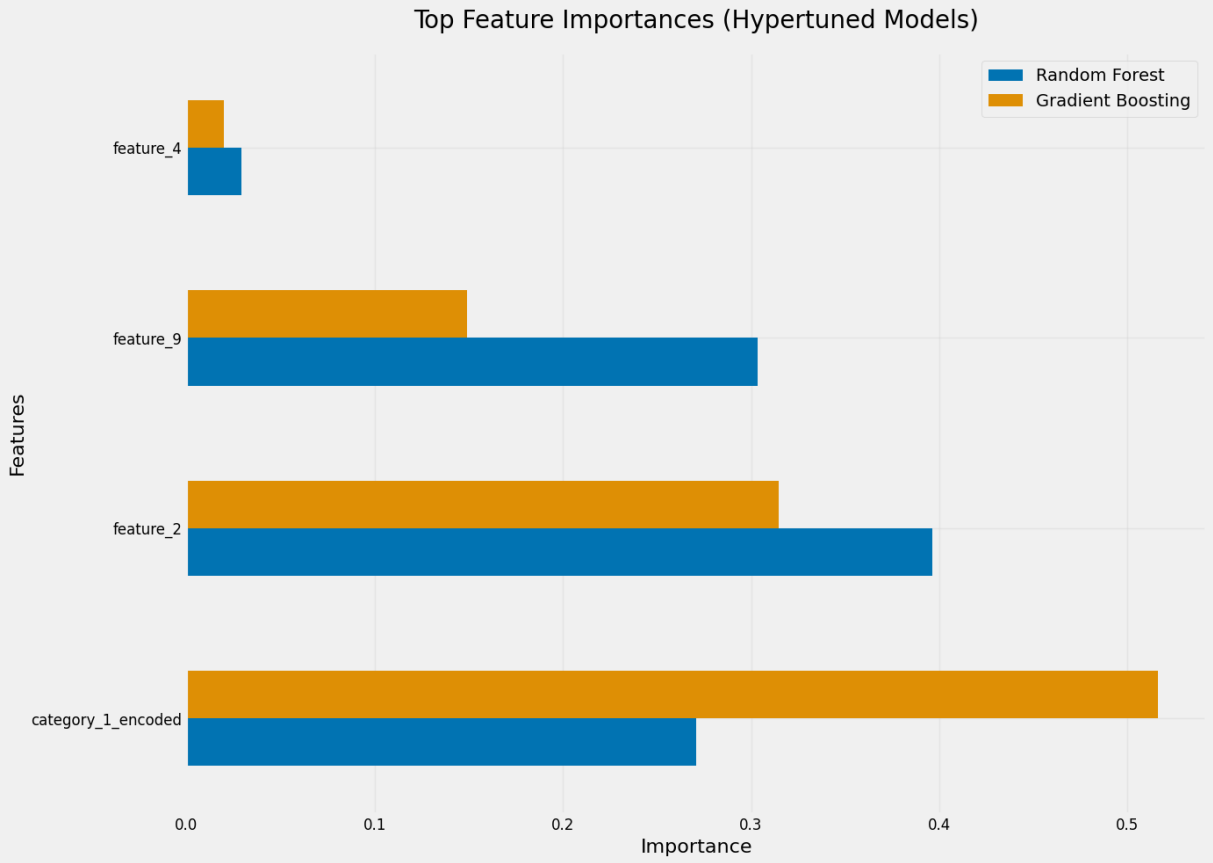
Gradient Boosting (Tuned):

- The median score remains very high, around 0.86, possibly even slightly higher than the untuned version's median.
- The box is extremely narrow, indicating outstanding consistency.

- Shows one outlier, similar to its untuned state.
- Consistency*:* Maintained its very high performance and consistency, perhaps tightening the distribution even further.

Voting Classifier (Tuned):

- The median score is also very high, around 0.86.
- The box is incredibly narrow, showcasing superb consistency.
- One outlier is visible.
- Consistency*:* Continues to exhibit extremely consistent and high performance, with potentially an even narrower score spread than its untuned counterpart.



"category_1_encoded":

- **Random Forest (Tuned):** Approximately 0.28.
- **Gradient Boosting (Tuned):** Approximately 0.52.
- *Interpretation:* The importance values for this feature appear virtually unchanged after tuning for both models. It remains the most important feature, especially for Gradient Boosting.

"feature_2":

- **Random Forest (Tuned):** Approximately 0.40.
- **Gradient Boosting (Tuned):** Approximately 0.33.

- Interpretation*:* The importance values for this feature also appear largely unchanged after tuning for both models. It remains the second most important feature.

"feature_9":

- **Random Forest (Tuned):** Approximately 0.29.
- **Gradient Boosting (Tuned):** Approximately 0.15.
- Interpretation*:* Again, the importance values for this feature seem consistent with the untuned versions. Random Forest still weights it significantly more than Gradient Boosting.

"feature_4":

- **Random Forest (Tuned):** Very low, close to 0.0.
- **Gradient Boosting (Tuned):** Approximately 0.02.
- Interpretation*:* This feature continues to have very low importance for both models.

Model Performance Summary (Hypertuned Models)

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC	CV Mean	CV Std
ndom Forest (Tun	0.864	0.869	0.839	0.854	0.938	0.861	0.012
dient Boosting (Tu	0.864	0.875	0.831	0.853	0.938	0.857	0.010
ting Classifier (Tur	0.861	0.848	0.859	0.854	0.935	0.858	0.010

Accuracy: Random Forest and Gradient Boosting have slightly higher accuracy (0.864) than the Voting Classifier (0.861).

Precision: Gradient Boosting shows the highest precision (0.875).

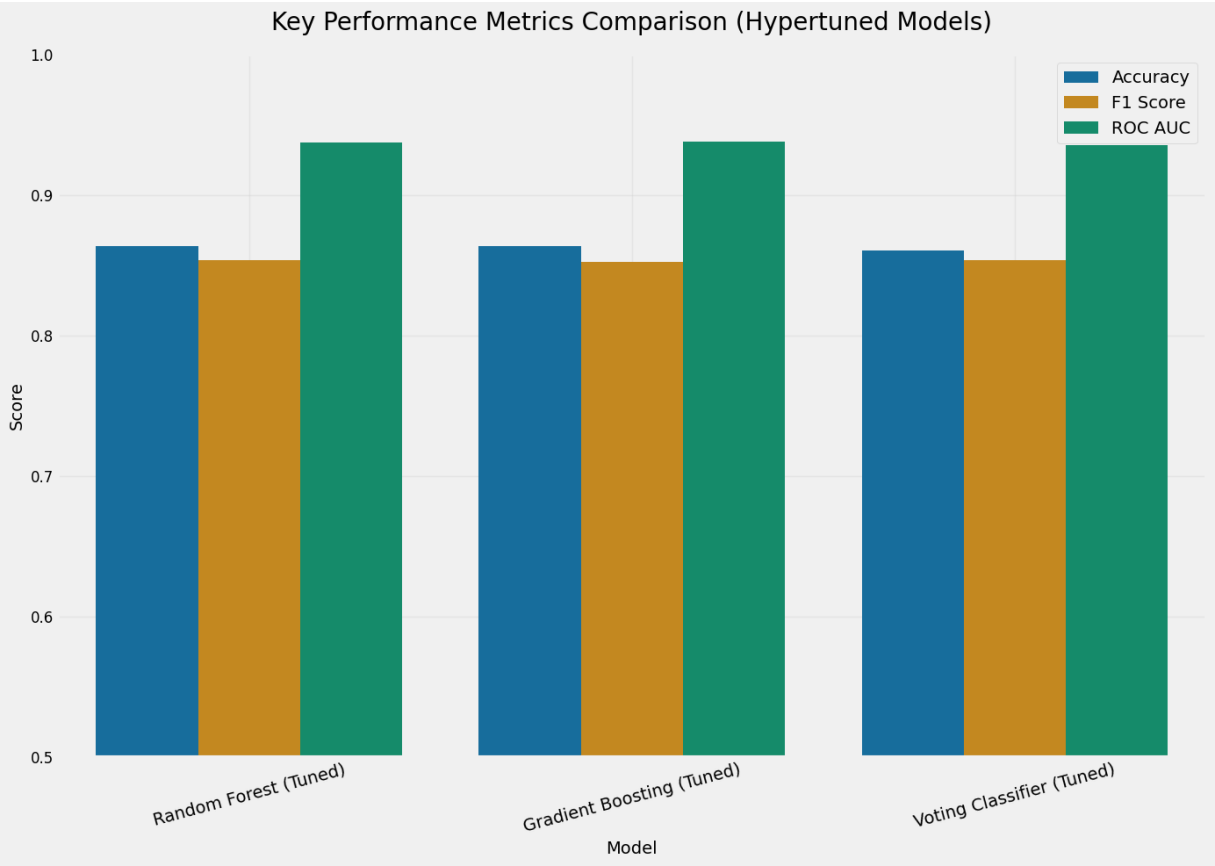
Recall: The Voting Classifier has the highest recall (0.859).

F1 Score: Random Forest and the Voting Classifier have a slightly higher F1 Score (0.854) than Gradient Boosting (0.853).

ROC AUC: Random Forest and Gradient Boosting have a slightly higher ROC AUC (0.938) than the Voting Classifier (0.935).

Cross-Validation:

- Random Forest has the highest CV Mean (0.861).
- Gradient Boosting and the Voting Classifier have a lower CV Standard Deviation (0.010) compared to Random Forest (0.012), indicating slightly more stable performance across folds for these two.

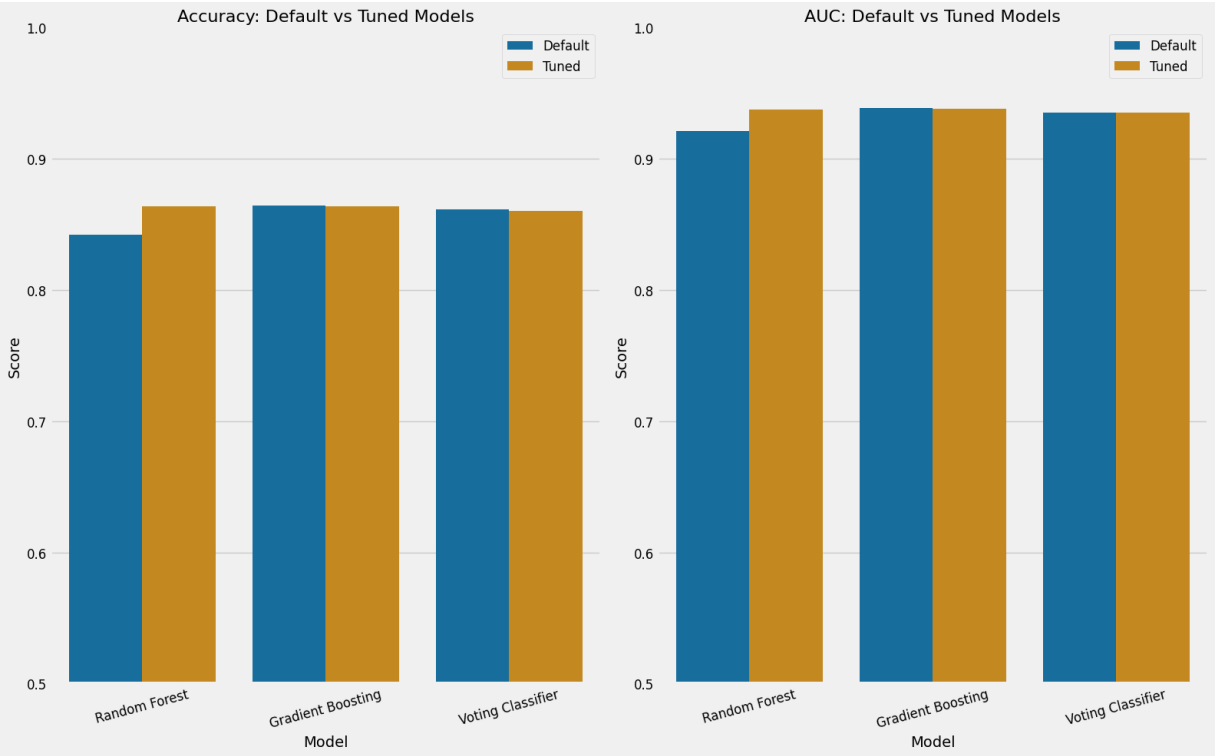


All three models perform well, with scores generally above 0.85 for Accuracy and F1 Score, and above 0.93 for ROC AUC.

ROC AUC is consistently the highest performing metric for all three models.

The **Random Forest** and **Gradient Boosting** models show very similar performance profiles across these three metrics.

The **Voting Classifier** shows slightly lower Accuracy and ROC AUC compared to the other two, but a comparable F1 Score.



Random Forest: The "Tuned" model shows a noticeable improvement in accuracy (around 0.87) compared to the "Default" model (around 0.84-0.85). It also shows a clear improvement in AUC (around 0.94) compared to the "Default" model (around 0.92).

Gradient Boosting: The "Tuned" model shows a very slight, almost negligible, improvement in accuracy over the "Default" model. Both are around 0.87. It shows a very slight improvement in AUC over the "Default" model. Both are very high, around 0.94.

Voting Classifier: Similar to Gradient Boosting, the "Tuned" model shows a very slight, almost negligible, improvement in accuracy over the "Default" model. Both are around 0.86-0.87. The "Tuned" model also shows a slight improvement in AUC over the "Default" model. Both are very high, around 0.93-0.94.

Performance Improvement After Hyperparameter Tuning

Model	Accuracy Improvement (%)	AUC Improvement (%)
Random Forest	2.57%	1.77%
Gradient Boosting	-0.06%	-0.03%
Voting Classifier	-0.13%	0.00%

Accuracy Improvement (%): This column shows the percentage change in the accuracy score of each model after hyperparameter tuning.

- **Random Forest:** Shows an accuracy improvement of **2.57%**. This indicates that tuning the hyperparameters led to a notable increase in the model's accuracy.
- **Gradient Boosting:** Shows an accuracy improvement of **-0.06%**. This negative percentage suggests a very slight decrease in accuracy after tuning. It's practically negligible and could be due to the stochastic nature of the model or the tuning process itself not finding a better set of parameters for accuracy than the default.
- **Voting Classifier:** Shows an accuracy improvement of **-0.13%**. Similar to Gradient Boosting, this indicates a very slight decrease in accuracy post-tuning, which is also likely negligible.

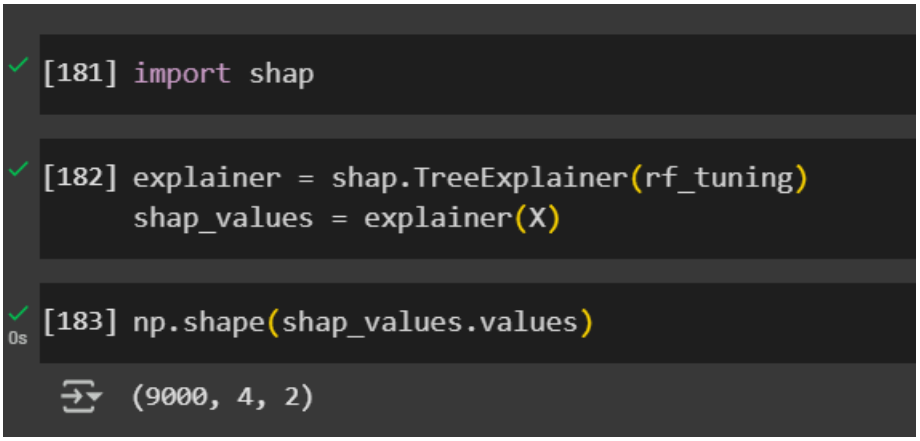
AUC Improvement (%): This column shows the percentage change in the AUC score of each model after hyperparameter tuning.

- **Random Forest:** Shows an AUC improvement of **1.77%**. This indicates a positive impact of tuning on the model's ability to distinguish between classes.
- **Gradient Boosting:** Shows an AUC improvement of **-0.03%**. This is a very marginal decrease, suggesting that tuning did not significantly improve (or slightly worsened) the AUC score compared to the default parameters.
- **Voting Classifier:** Shows an AUC improvement of **0.00%**. This indicates that hyperparameter tuning had no discernible impact on the AUC score for the Voting Classifier; the performance remained the same as with default parameters.

7.1 Model Interpretation

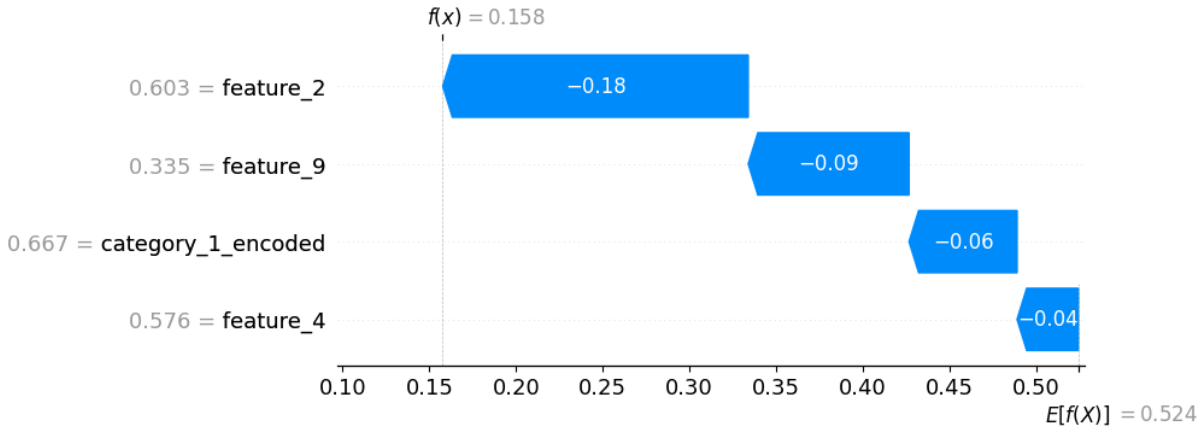
In this stage, we will dive mostly into information of feature importance and other model interpretations. It is good practice to apply SHAP to tuned models for more accurate explanations, ensuring robustness and providing both global and local interpretability.

Random Forest Interpretation



(9000, 4, 2) means that 9000 is the number of samples, 4 is the number of features and 2 is the number of classes.

For class 0:



Base Value (E[f(x)] = 0.524**)**: The average model output for class 0 across all samples is **0.524** (this is the starting point before feature contributions).

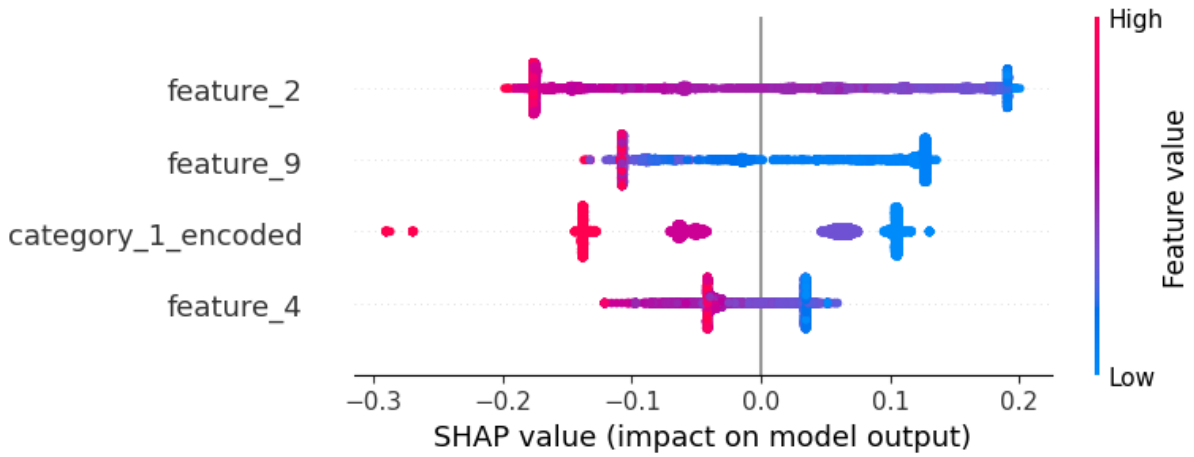
Prediction (f(x) = 0.158**)**: the model predicts **0.158** (closer to 0, meaning higher confidence in **class 0**).

Feature Contributions:

- feature_2 **(0.603)**: Decreases the prediction (negative SHAP value, pushing toward class 0).

- category_1_encoded **(0.667)**: Increases the prediction (positive SHAP value, pushing toward class 1).

- feature_9 **(0.335) and** feature_4 **(0.576)**: Have smaller but meaningful impacts.



Feature Ranking:

feature_2 → feature_9 → category_1_encoded -> feature_4 (left to right = most to least important).

feature_2:

Most dots are clustered around 0, suggesting this feature has minimal impact overall.

A few outliers with high (red) values push the prediction slightly negative (left of 0).

feature_9:

High values (red) tend to have positive SHAP values (increase the prediction).

Low values (blue) tend to have negative SHAP values (decrease the prediction).

category_1_encoded:

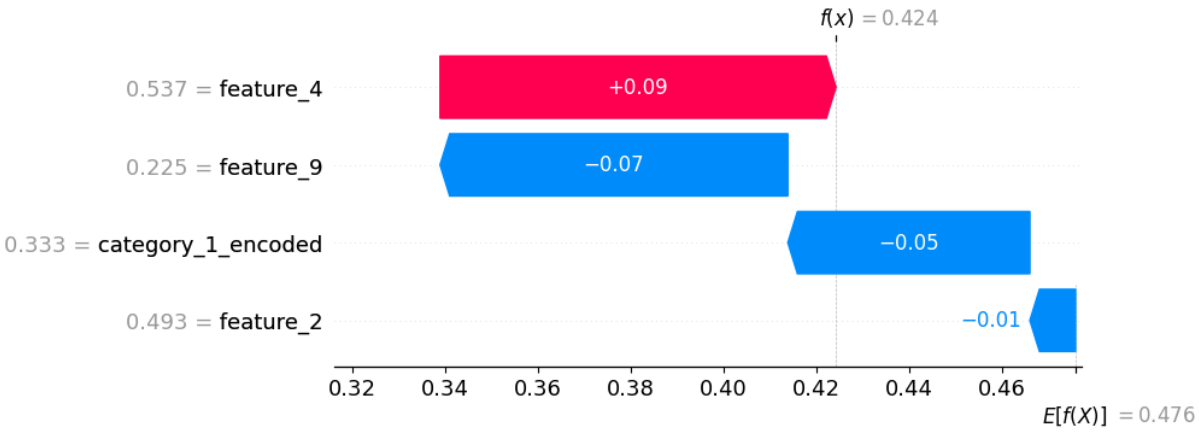
Shows a mix of impacts, with some high values (red) having strong positive or negative effects.

feature_4:

High values (red) are associated with negative SHAP values, reducing the prediction.

Low values (blue) are associated with positive SHAP values, increasing the prediction.

For class 1:



Base Value (E[f(X)] = 0.476**)**: The model's average prediction (across all samples) is **0.476**, slightly favoring **Class 0** (since it's <0.5).

Prediction (f(x) = 0.424**)**: For this specific instance, the output is **0.424** (still Class 0, but closer to the threshold).

Feature Contributions:

feature_4 = 0.537 contributed **+0.09**, increasing the prediction.

feature_9 = 0.225 contributed **-0.07**, decreasing the prediction.

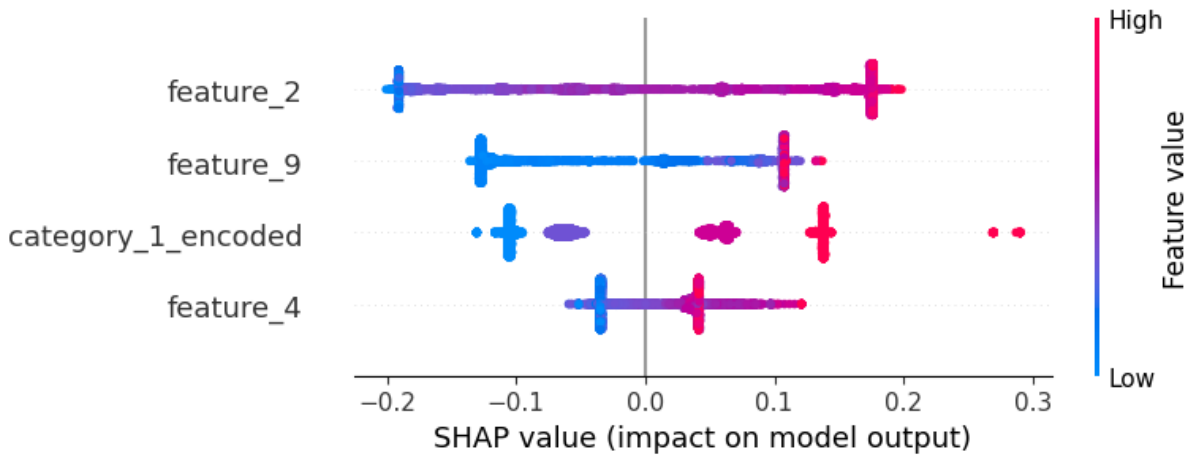
category_1_encoded = 0.333 contributed **-0.05**, decreasing the prediction.

feature_2 = 0.493 contributed **-0.01**, slightly decreasing the prediction.

The model started at the baseline prediction of 0.476.

feature_4 **had the largest positive impact (+0.09)**, but this was offset by negative contributions from other features: feature_9 (-0.07), category_1_encoded (-0.05), and feature_2 (-0.01).

The net effect was a reduction in the prediction to 0.424



Feature Ranking:
feature_2 > feature_9 > category_1_encoded > feature_4 (left to right = most to least important).

feature_2:

Most dots are clustered around 0.1 to 0.3 (positive impact).

High values (red) tend to push predictions higher.

Likely the most influential feature overall.

feature_9:

Mixed impact: Some high values (red) increase predictions, while others decrease them.

Wider spread suggests complex interactions with other features.

category_1_encoded:

Symmetric around 0, with both positive and negative impacts.

Categorical features often split predictions (e.g., one category increases output, another decreases it).

feature_4:

Low values (blue) dominate the negative SHAP values (left of 0).

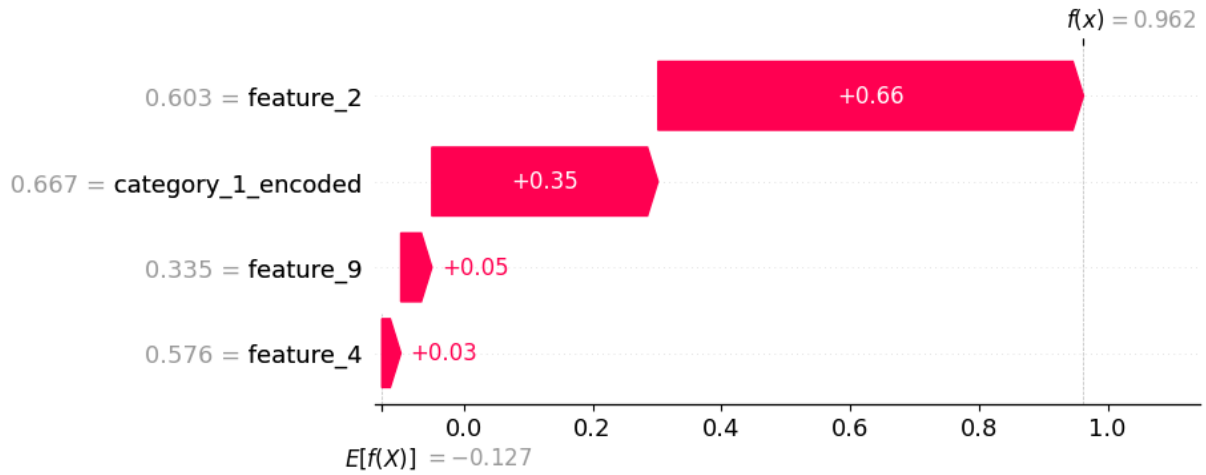
High values (red) have a minimal positive impact.

Gradient Boosting Interpretation

```
[186] explainer_gb = shap.TreeExplainer(gb_tuned)
      shap_values_gb = explainer_gb(X)

[187] shap_values_train = explainer.shap_values(X_train)
      shap_values_test = explainer.shap_values(X_test)

[188] np.shape(shap_values_gb.values)

      (9000, 4)
```

We also apply SHAP in this case for tuned gradient boosting model.

This is binary classification, so SHAP returns a single array of shapes which corresponds to the SHAP values for class 1 (positive class). (9000, 4) in which 9000 is the data points and 4 is still the number of features.



Baseline Value (E[f(X)] = -0.127)

This is the model's average prediction across all training data (i.e., what it would predict without any feature inputs for this instance).

Final Prediction (Top of the Plot, 0.8–1.0 Range)

The actual model output for this instance is not explicitly labeled but appears to land near 0.8–1.0 based on the arrows.

The combined feature contributions pushed the prediction significantly higher than the baseline.

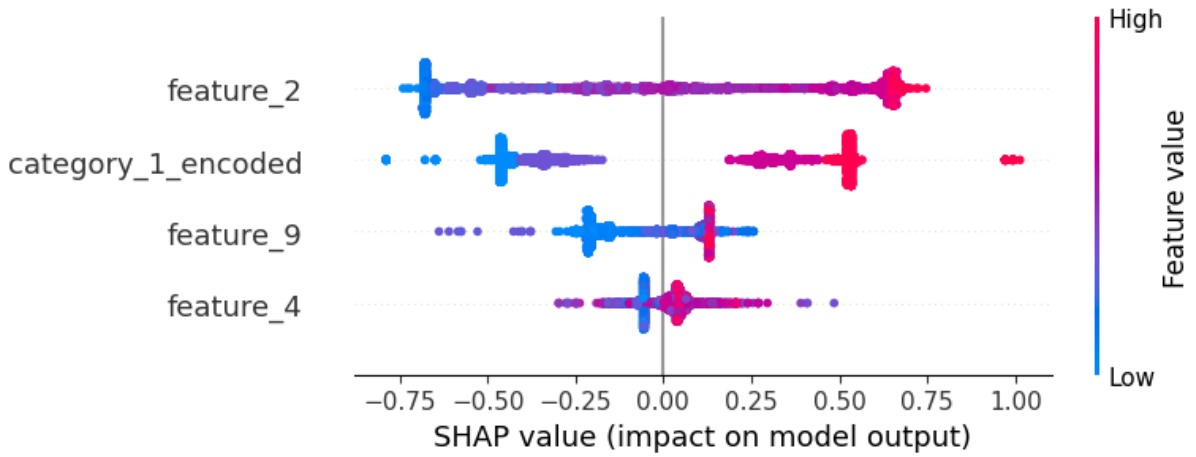
Feature Contributions (Arrows)
Each line represents a feature's impact:

feature_2 = 0.603: Contributed +0.66 (large positive push).

category_1_encoded = 0.667: Contributed +0.35 (positive push).

feature_9 = 0.335: Contributed +0.05 (small positive push).

feature_4 = 0.576: Contributed +0.03 (minimal positive push).



Feature Importance Ranking:
feature_2 > category_1_encoded > feature_9 > feature_4 (left to right = most to least impactful).

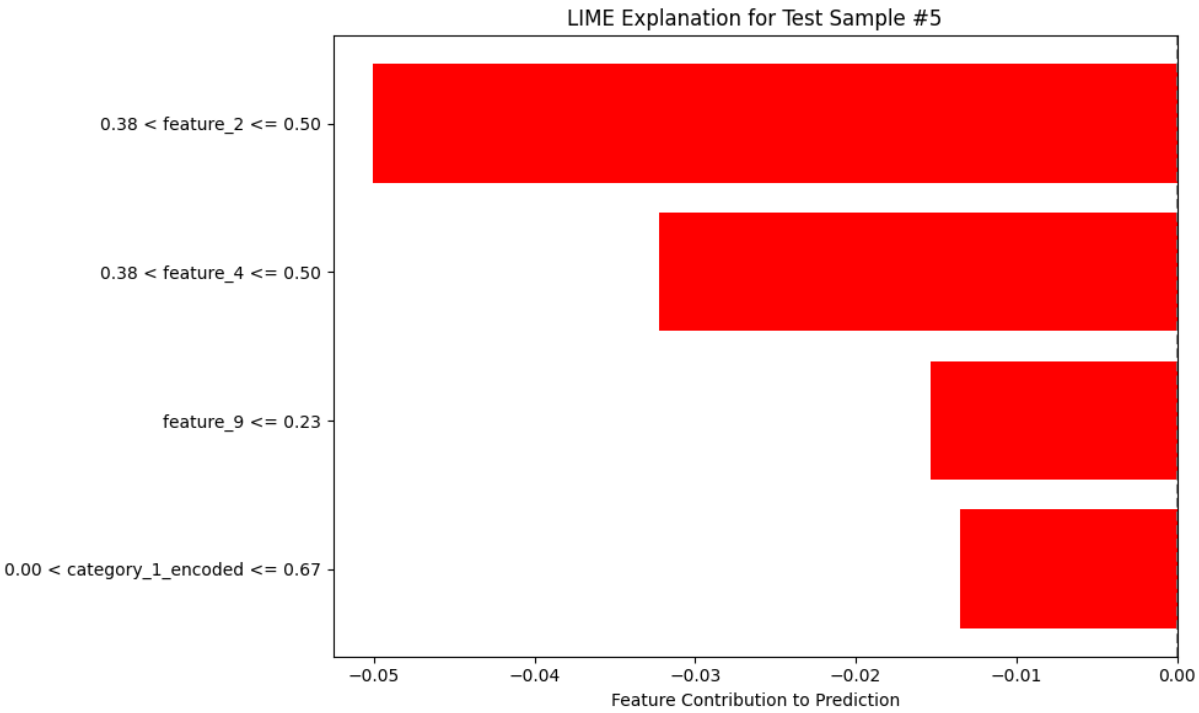
Directionality:

feature_2: High values (red) strongly push the model output toward Class 1 (positive SHAP).

category_1_encoded: High values moderately push toward Class 1.

feature_9 and feature_4: Both show mixed effects; the impact of their values on the model output is less consistent and generally weaker.

Lime Interpretation



LIME explains the prediction of **one data point** (in this case, sample #5) by showing **which features influenced the model's decision** — and whether they pushed the prediction **up or down**.

Top feature:

- 0.38 < feature_2 <= 0.50:
  This condition on feature_2 had the **biggest negative impact** on the prediction. It made the prediction lower more than any other feature.

Others:

- 0.38 < feature_4 <= 0.50: Also pushed the prediction lower.
- feature_9 <= 0.23: Same — contributed negatively.
- 0.00 < category_1_encoded <= 0.67: A categorical feature (probably one-hot or label encoded), also contributed a little to lowering the output.

6.2 Domain Knowledge and Business Logic

In order to derive actionable insights and align our data analysis with domain-specific business logic, we analyzed the **feature importance** using three different interpretability techniques: **SHAP**, **LIME**, and **model-based importance** from **Random Forest**, **Gradient Boosting**, and a **Voting Classifier**. To improve clarity and contextual relevance, the dataset's feature names were first renamed to be more interpretable for educational domain stakeholders.

The following table maps the original technical feature names to more descriptive names:

Original Column Name	New Descriptive Column Name
feature_2	Course Engagement Score (0–10)
feature_4	Average Time Spent Per Session (minutes)
feature_9	Number of Missed Deadlines
category_1_encoded	Student Type
target	Dropout Status

SHAP Analysis – Random Forest & Gradient Boosting Models

The SHAP (SHapley Additive exPlanations) technique was applied to the **Random Forest** and **Gradient Boosting** models to identify the most influential features contributing to dropout prediction.

Key Insights:

- **Most Impactful Feature**:
  - Course Engagement Score (feature_2)

Students with consistently low engagement are significantly more likely to drop out.

- **Second Most Important**:
  - Number of Missed Deadlines (feature_9)

Frequent deadline misses are indicative of poor time management or motivational issues.

- **Moderate Impact**:
  - Average Time Spent Per Session (feature_4)

Shorter session durations could reflect declining interest or difficulties in learning.

- **Low Impact**:
  - Student Type (category_1_encoded)

Slight pattern observed: part-time students show a marginally higher dropout tendency, possibly due to external commitments.

---

4. LIME Analysis – Voting Classifier Model

The **LIME (Local Interpretable Model-agnostic Explanations)** method was used to analyze the **Voting Classifier** model's decision-making on an individual test case.

Case Summary:

- **Prediction**: Not Dropping Out (Class 0)
- **Model Confidence**: 88%

Feature Contributions:

- **Course Engagement Score**

Despite the positive classification, a recent dip in engagement slightly pulled the prediction toward dropout. This signals early warning signs.

- **Other Features**

Had minimal impact on the model’s decision for this instance.

---

5. Business Implications & Domain Logic Integration

Based on the feature importance and explainability results, the following business logic and actions are recommended:

Feature	Interpretation	Business Action
Course Engagement Score	Strongest predictor — engaged students complete courses	Monitor engagement regularly and trigger alerts for sudden drops
Missed Deadlines	Signals poor time management or low motivation	Offer time-management workshops and personalized nudges
Session Duration	Short sessions may reflect loss of interest or comprehension challenges	Introduce interactive content or adaptive learning strategies
Student Type	Part-time learners are at slightly higher risk	Provide flexible support options and tailored communication

---

6. Conclusion

By combining **SHAP** and **LIME** explanations across different models, we developed a holistic understanding of the factors influencing student dropout. These insights enable institutions to design **targeted interventions**, improve **student retention**, and align predictive analytics with **business and academic priorities**.