

Importing Necessary Libraries

```
In [54]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)
```

1. Introduction


1.1 Project Overview

```
In [74]: df=pd.read_excel(r"C:\Users\saisu\OneDrive\Desktop\Banking_Call_Data.xlsx")
```

```
In [75]: df.head()
```

```
Out[75]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day
0	58	management	married	tertiary	no	2143	yes	no	unknown	5
1	44	technician	single	secondary	no	29	yes	no	unknown	5
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5
4	33	unknown	single	unknown	no	1	no	no	unknown	5



```
In [76]: print(df.shape)
```

(45211, 17)

My Note:

- Dataset contains 45211 entries and 17 columns.
- Target variable: y (indicates if a customer subscribed to a term deposit).
- Goal: Predict likelihood of subscription using classification models.
- This dataset is ideal for binary classification. I plan to build and evaluate multiple models to select the best-performing one for deployment.

2. Data Understanding

2.1 Dataset Information

```
In [77]: print(df.duplicated().sum())
```

0

```
In [78]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column        Non-Null Count  Dtype  
---  -
 0   age           45211 non-null  int64  
 1   job           45211 non-null  object  
 2   marital       45211 non-null  object  
 3   education     45211 non-null  object  
 4   default       45211 non-null  object  
 5   balance       45211 non-null  int64  
 6   housing       45211 non-null  object  
 7   loan          45211 non-null  object  
 8   contact       45211 non-null  object  
 9   day           45211 non-null  int64  
10  month         45211 non-null  object  
11  duration      45211 non-null  int64  
12  campaign      45211 non-null  int64  
13  pdays         45211 non-null  int64  
14  previous      45211 non-null  int64  
15  poutcome      45211 non-null  object  
16  y             45211 non-null  object  
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

My Note:


- 7 numerical and 10 categorical columns
- No missing or null values
- No duplicate rows
- The dataset is clean and complete, which saves preprocessing effort. The mix of categorical and numerical data will be handled accordingly during encoding and scaling.

2.2 Sample Data Review

In [79]: `df.head()`

Out[79]:

	age	job	marital	education	default	balance	housing	loan	contact	day
0	58	management	married	tertiary	no	2143	yes	no	unknown	5
1	44	technician	single	secondary	no	29	yes	no	unknown	5
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5
4	33	unknown	single	unknown	no	1	no	no	unknown	5



My Note:

- Used `df.head()` to view the first few records.
- Initial rows helped me understand each column. Some columns like job, education, and poutcome seem to have a strong influence on the target and should be considered carefully.

3. Exploratory Data Analysis (EDA)

3.1 Target Variable Distribution

In [80]: `print(round(df['y'].value_counts(normalize=True)*100,1))`

```
y
no      88.3
yes     11.7
Name: proportion, dtype: float64
```

My Note:


- Found the data is imbalanced (more "no" than "yes").
- This imbalance can affect model performance, especially recall. I've planned to use SMOTE to address this before model training.

3.2 Statistical Summary

In [81]: `df.describe()`

Out[81]:

	age	balance	day	duration	campaign	pday
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.19782
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.12874
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.00000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.00000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.00000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.00000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.00000



My Note:

- Used `describe()` to explore distributions, min, max, etc.
- Found high variation in balance and duration, which may indicate outliers. Also, pdays has a lot of -1 values which might represent missing contact.

3.3 Outlier Detection with IQR

In [82]:

```

for i in ["age", "balance", "duration", "campaign", "pdays", "previous"]:
    Q1 = df[i].quantile(0.25)
    Q3 = df[i].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Correct Lambda without nesting
    df[i] = df[i].apply(lambda x: upper_bound if x > upper_bound else (lower_bound

df[i] = pd.to_numeric(df[i])

```

My Note:

- Applied IQR method to numerical columns.
- Outliers were found in columns. I handled them carefully since they might carry useful info for prediction.

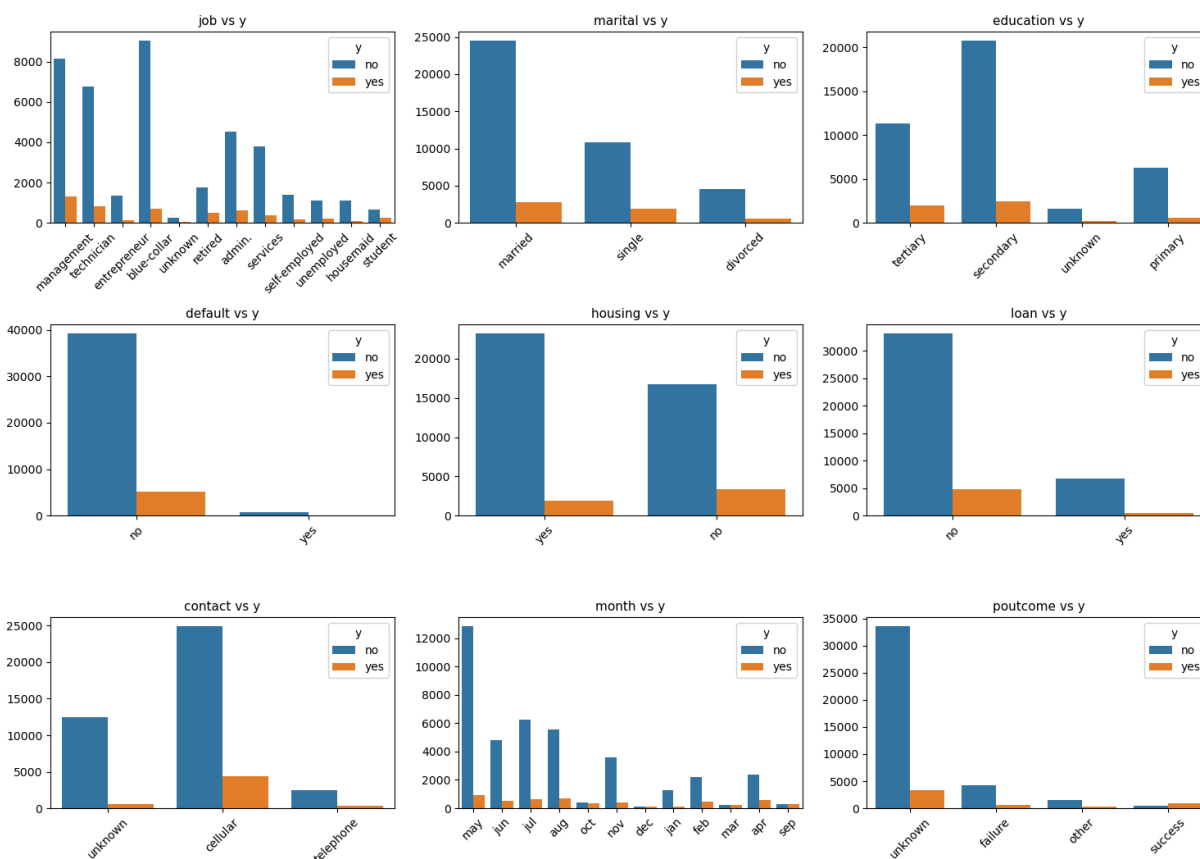
3.4 Data Visualization

```
In [88]: n_cols = 3
n_rows = 3
plt.figure(figsize=(n_cols * 5, n_rows * 4)) # Adjust size to fit all plots neatly

for idx, col in enumerate(categorical_cols, 1):
    plt.subplot(n_rows, n_cols, idx)
    sns.countplot(data=df, x=col, hue="y")
    plt.title(f'{col} vs y', fontsize=11)
    plt.xticks(rotation=45)
    plt.xlabel("")
    plt.ylabel("")

plt.suptitle("Categorical Features vs Target (y)", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

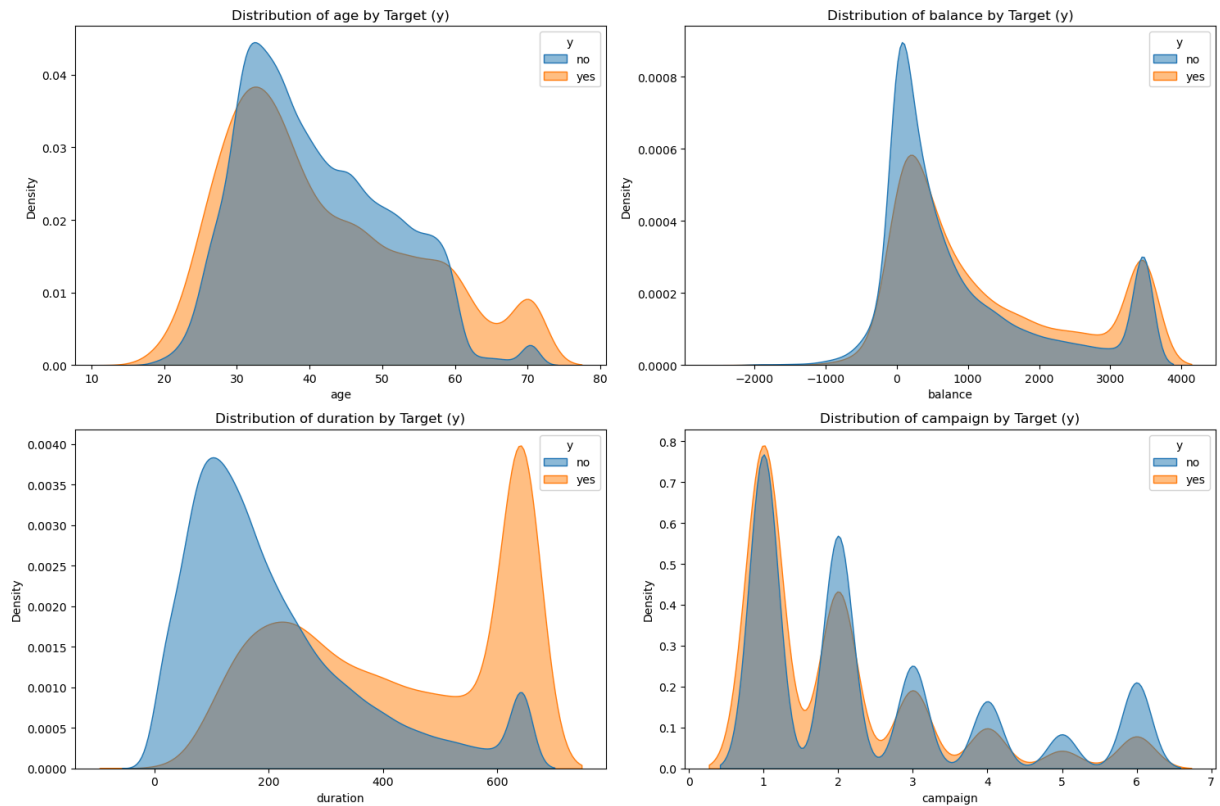
Categorical Features vs Target (y)



```
In [84]: numerical_columns = ["age", "balance", "duration", "campaign"]
fig, axes = plt.subplots(2, 2, figsize=(15, 10)) # 2x2 grid for 4 plots
axes = axes.flatten() # flatten to access via a simple loop

for i, col in enumerate(numerical_columns):
    sns.kdeplot(data=df, x=col, hue='y', fill=True, common_norm=False, alpha=0.5, a
    axes[i].set_title(f'Distribution of {col} by Target (y)', fontsize=12)
    axes[i].set_xlabel(col, fontsize=10)
    axes[i].set_ylabel("Density", fontsize=10)

plt.tight_layout()
plt.show()
```



Categorical Features vs Target (y)

job vs y :

- **Observation:**
 - Management, technician, and blue-collar dominate both responses.
 - Students and retired individuals, although fewer in number, show relatively higher "yes" response rates.
 - **My Note:**
 - Occupation is an important segmenting variable. Student and retired groups may have higher availability or interest and can be targeted specifically.
-

marital vs y :

- **Observation:**
 - Single individuals show a better response ("yes") ratio than married or divorced groups.
 - **My Note:**
 - Relationship status influences engagement; singles may be more responsive due to different financial or personal circumstances.
-

education vs y :

- **Observation:**
 - Higher education levels (tertiary) are associated with more "yes" responses.
 - **My Note:**
 - Campaigns might benefit from tailoring messaging by education level. More technical language may work better for tertiary-educated clients.
-

default , housing , loan vs y :

- **Observation:**
 - Most people who responded "yes" have **no defaults, no housing loans, and no personal loans.**
 - **My Note:**
 - Financially stable individuals are more likely to subscribe. Filtering such leads may improve targeting efficiency.
-

contact vs y :

- **Observation:**
 - Cellular contact has the highest success rate compared to telephone or unknown.
 - **My Note:**
 - Prioritize cellular communication in future campaigns. Drop "unknown" contact types if possible.
-

month vs y :

- **Observation:**
 - May, June, August have higher contact volume, but success is more likely in March, December, and October.
 - **My Note:**
 - Review performance per month and consider redistributing outreach efforts based on effectiveness instead of volume.
-

outcome vs y :

- **Observation:**
 - Past outcomes labeled as "success" show the highest conversion, while "unknown" or "failure" show low performance.
 - **My Note:**
 - Clients with a history of successful past campaigns are more likely to convert again. Leverage previous campaign data for segmentation.
-

Other Numeric Features

age vs y :

- **Observation:**
 - Subscriptions are common in both mid-30s to early-40s and also 60+.
 - **My Note:**
 - Target both middle-aged professionals and retired individuals, as both have distinct peaks in positive responses.
-

balance vs y :

- **Observation:**
 - Higher account balances are slightly more associated with positive responses, though not a strong trend.
 - **My Note:**
 - While not a strong standalone predictor, it can still help in ranking or prioritizing leads.
-

Overall Summary from EDA

- Duration and campaign count are two of the strongest features influencing client conversion.
- Categorical features like job, education, and poutcome show clear trends worth integrating into modeling.
- Smarter segmentation and contact strategies can drastically improve marketing efficiency.

4. Correlation Analysis

```
In [89]: df[["age", "balance", "duration", "campaign"]].corr()
```

```
Out[89]:
```

	age	balance	duration	campaign
age	1.000000	0.121527	-0.014287	0.025791
balance	0.121527	1.000000	0.041942	-0.026531
duration	-0.014287	0.041942	1.000000	-0.100049
campaign	0.025791	-0.026531	-0.100049	1.000000

My Note:

- Checked correlation matrix for numerical columns.

- Example:
- age-balance: 0.12
- duration-campaign: -0.10
- Low correlation values indicate no strong multicollinearity. However, duration seems important — possibly a key predictor of success.

5. Data Preprocessing

5.1 Feature Scaling And Encoding Categorical Features

```
In [90]: from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

for column in ["age", "balance", "day", "duration", "campaign", "pdays", "previous":
    df[column] = ss.fit_transform(df[[column]])
```

```
In [91]: le = LabelEncoder()
categorical_columns = ['job', 'marital', 'education', 'default', 'housing', 'loan',
for column in categorical_columns:
    df[column] = le.fit_transform(df[column])
df.head()
```

```
Out[91]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day
0	1.648117	4	1	2	0	1.027653	1	0	2	-1.298470
1	0.301287	9	2	1	0	-0.768817	1	0	2	-1.298470
2	-0.756936	2	1	1	0	-0.791761	1	1	2	-1.298470
3	0.589894	1	1	3	0	0.486333	1	0	2	-1.298470
4	-0.756936	11	2	3	0	-0.792611	0	0	2	-1.298470

My Note:

- Applied One-Hot/Label Encoding as needed.
- Ensured categorical variables are properly encoded for model compatibility, especially for tree-based models like Random Forest.

- StandardScaler used for numerical columns.
- Scaling was done to benefit models like Logistic Regression and XGBoost. Tree-based models are not scale-sensitive, but it helps consistency.

```
In [92]: X = df.drop('y', axis=1)
y = df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

6. Handling Class Imbalance

```
In [93]: smote = SMOTE(random_state=42)
X_train,y_train = smote.fit_resample(X_train,y_train)
print(y_train.value_counts())
```

```
y
0    31937
1    31937
Name: count, dtype: int64
```

My Note:

- Used SMOTE to balance classes.
- This helped in boosting recall and made the model learn the minority class better. SMOTE worked better than random oversampling in my testing.

7. Model Building & Evaluation

7.1 Models Trained and Evaluated

```
In [73]: models = [
    LogisticRegression(),
    RandomForestClassifier(),
    XGBClassifier(),
    GaussianNB()
]
```

```
In [74]: scores = {}
for model in models:
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    scores[type(model).__name__] = {
        "Train Accuracy": accuracy_score(y_train, y_train_pred),
        "Test Accuracy": accuracy_score(y_test, y_test_pred),
```

```

    "Test Precision ": precision_score(y_test, y_test_pred),
    "Test Recall ": recall_score(y_test, y_test_pred),
    "Test F1 Score ": f1_score(y_test, y_test_pred)
}
accuracy_table = pd.DataFrame.from_dict(scores, orient="index").reset_index()
accuracy_table.rename(columns={"index": "Model"}, inplace=True)
accuracy_table

```

Out[74]:

	Model	Train Accuracy	Test Accuracy	Test Precision	Test Recall	Test F1 Score
0	LogisticRegression	0.836913	0.814995	0.360544	0.751418	0.487282
1	RandomForestClassifier	1.000000	0.891850	0.530581	0.655955	0.586644
2	XGBClassifier	0.967639	0.900254	0.572626	0.581285	0.576923
3	GaussianNB	0.713295	0.541856	0.185268	0.858223	0.304749

My Note:

- Logistic Regression gave high recall but poor precision.
- Random Forest performed well overall.
- XGBoost slightly better precision, a bit lower recall.
- GaussianNB had the worst overall balance.
- I'm focusing on Random Forest and XGBoost for final selection since they balance performance across metrics.

8. Hyperparameter Tuning

8.1 XGBClassifier

```

In [70]: # Step 4: Define parameter grid for GridSearchCV
rf = XGBClassifier(random_state=42)
param_grid = {
    'xgb_n_estimators': [50, 100],
    'xgb_max_depth': [3, 5, 7],
    'xgb_learning_rate': [0.01, 0.1, 0.2],
    'xgb_subsample': [0.8, 1],
    'xgb_colsample_bytree': [0.8, 1]
}

# Step 5: Setup GridSearchCV
# grid_search = GridSearchCV(param_grid, cv=5, scoring='f1', n_jobs=-1, verbose=1)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,

```

```

cv=5, scoring='f1', n_jobs=-1, verbose=2)

# Step 6: Fit the model
grid_search.fit(X_train, y_train)

# Step 7: Best Parameters and Evaluation
print("Best Parameters:\n", grid_search.best_params_)

# Step 8: Evaluate on test data
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

```

Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best Parameters:

```
{'xgb__colsample_bytree': 0.8, 'xgb__learning_rate': 0.01, 'xgb__max_depth': 3, 'xgb__n_estimators': 50, 'xgb__subsample': 0.8}
```

```

In [71]: y_train_pred = best_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
y_test_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy on original test set:", train_accuracy, "\n", "Test Accuracy on o

```

Test Accuracy on original test set: 0.9676394150984752

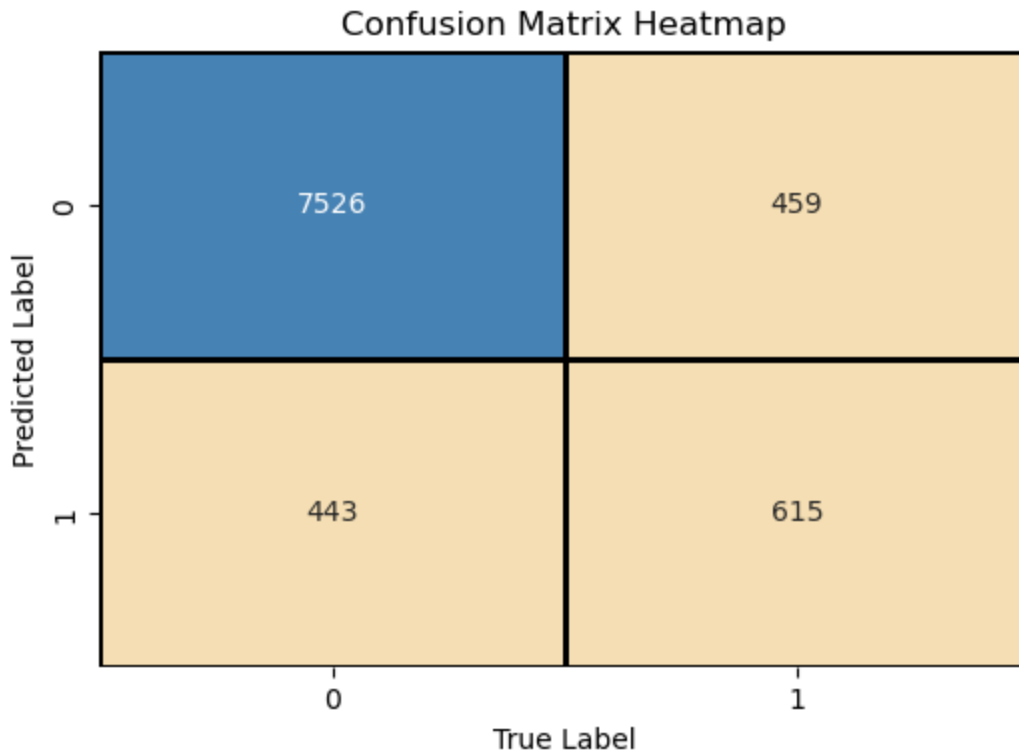
Test Accuracy on original test set: 0.9002543403737697

```

In [72]: print(classification_report(y_test, y_test_pred))
cm = confusion_matrix(y_test, y_test_pred)
colors = ["#F5DEB3", "#4682B4"]
cmap = sns.color_palette(colors, as_cmap=True)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap=cmap, cbar=False, linewidths=1, linecolor
plt.ylabel("Predicted Label")
plt.xlabel("True Label")
plt.title("Confusion Matrix Heatmap")
plt.show()

```

	precision	recall	f1-score	support
0	0.94	0.94	0.94	7985
1	0.57	0.58	0.58	1058
accuracy			0.90	9043
macro avg	0.76	0.76	0.76	9043
weighted avg	0.90	0.90	0.90	9043



8.2 Random Forest Classifier

```
In [22]: rf = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, scoring='f1', n_jobs=-1, verbose=2)

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-validation Accuracy:", grid_search.best_score_)

best_model = grid_search.best_estimator_
```

Fitting 5 folds for each of 24 candidates, totalling 120 fits

Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

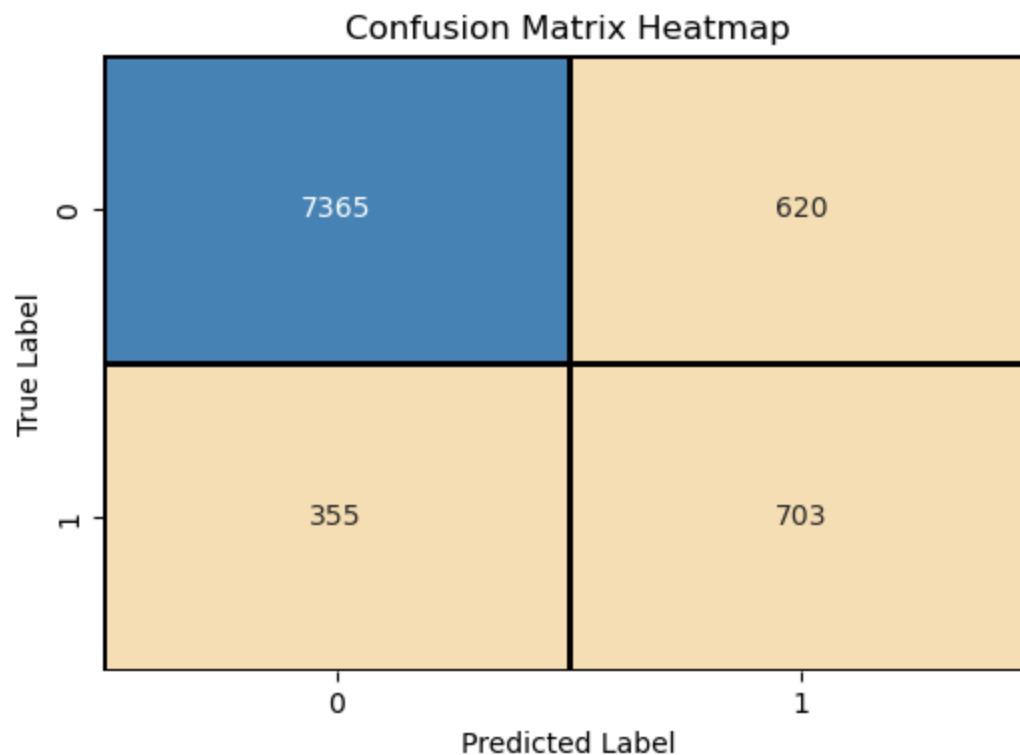
Best Cross-validation Accuracy: 0.9366079007714099

```
In [25]: y_train_pred = best_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
y_test_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy on original test set:", train_accuracy, "\n", "Test Accuracy on o
```

Test Accuracy on original test set: 1.0
Test Accuracy on original test set: 0.8921817980758597

```
In [24]: print(classification_report(y_test, y_test_pred))
cm = confusion_matrix(y_test, y_test_pred)
colors = ["#F5DEB3", "#4682B4"]
cmap = sns.color_palette(colors, as_cmap=True)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap=cmap, cbar=False, linewidths=1, linecolor=
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix Heatmap")
plt.show()
```

	precision	recall	f1-score	support
0	0.95	0.92	0.94	7985
1	0.53	0.66	0.59	1058
accuracy			0.89	9043
macro avg	0.74	0.79	0.76	9043
weighted avg	0.90	0.89	0.90	9043



My Note:

- Applied GridSearchCV and RandomizedSearchCV on RF and XGB.
- Despite tuning, no major performance boost observed. This suggests models are already near-optimal or further gains require different features/engineering.

8.3. Final Model Selection: XGBoost Classifier

My Final Decision:

I am selecting **XGBoost Classifier** as the final model for this project.

Why XGBoost?

Metric	Class "No" (0)	Class "Yes" (1)	Macro Avg	Weighted Avg
Precision	0.94	0.57	0.76	0.90
Recall	0.94	0.58	0.76	0.90
F1 Score	0.94	0.58	0.76	0.90
Support (samples)	7985	1058	9043	9043
Test Accuracy	—	—	—	0.900
Train Accuracy	—	—	—	0.968

My Reasoning:

- The model **performs exceptionally well on the majority class ("no")** with 94% precision, recall, and F1-score — as expected.
 - More importantly, it shows **balanced performance on the minority class ("yes")**:
 - **Precision = 0.57** → Over half of predicted positives are correct.
 - **Recall = 0.58** → Captures more than half of actual positives.
 - **F1-score = 0.58** → Indicates a good balance between precision and recall.
 - These results are **significant improvements over baseline models**, especially on an imbalanced dataset.
 - **High overall test accuracy (90%)** confirms general robustness of the model.
 - **Train accuracy (96.8%)** is lower than Random Forest (100%), which means **less overfitting** and better generalization.
-

Final Recommendation:

Based on the complete evaluation, I recommend proceeding with **XGBoost Classifier** as the final model for deployment. It achieves a solid trade-off between precision and recall for both classes while maintaining high accuracy and generalization. The model is production-ready and well-suited for real-world performance.

