

Import Libraries

```
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

Load Cleaned and Merged Data for predictions

```
df=pd.read_csv('data_for_predictions.csv')
df.head(5)
```



	Unnamed: 0	id	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_1
0	0	24011ae4ebbe3035111d65fa7c15bc57	0.000000	4.739944	0.000000	0.0000
1	1	d29c2c54acc38ff3c0614d0a653813dd	3.668479	0.000000	0.000000	2.2809
2	2	764c75f661154dac3a6c254cd082ea7d	2.736397	0.000000	0.000000	1.6898
3	3	bba03439a292a1e166f80264c16191cb	3.200029	0.000000	0.000000	2.3820
4	4	149d57cf92fc41cf94415803a877cb4b	3.646011	0.000000	2.721811	2.6500

5 rows × 64 columns



```
df.drop(['Unnamed: 0', 'id'],axis=1,inplace=True)
```

Split Data for Train & Test

```
(X_train,X_test,y_train,y_test)=train_test_split(df.drop('churn',axis=1),df['churn'],test_size=0.3,random_s
X_train.shape,X_test.shape
```



((10224, 61), (4382, 61))

Model selection and training

```
# Define the model
model = RandomForestClassifier(random_state=42)

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False]
}

# Set up GridSearchCV
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
```

```

cv=5,                # 5-fold cross-validation
scoring='accuracy',
n_jobs=2,
verbose=2
)

# Fit the model
grid_search.fit(X_train, y_train)
# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)

```

 [Show hidden output](#)


Prediction and Evaluation

```


best_model = grid_search.best_estimator_
y_pred=best_model.predict(X_test)

```

y_pred[:5]


 array([0, 0, 0, 0, 0])

y_test[:5]

 churn

4947	0
5868	0
6805	0
1323	0
11759	0

best_model.score(X_test,y_test)

 0.9046097672295755

T **B** *I* <>          


****Classification Report & Confusion Matrix****

Classification Report & Confusion Matrix

```

classification_report=sklearn.metrics.classification_report(y_test,y_pred)
print(classification_report)

```



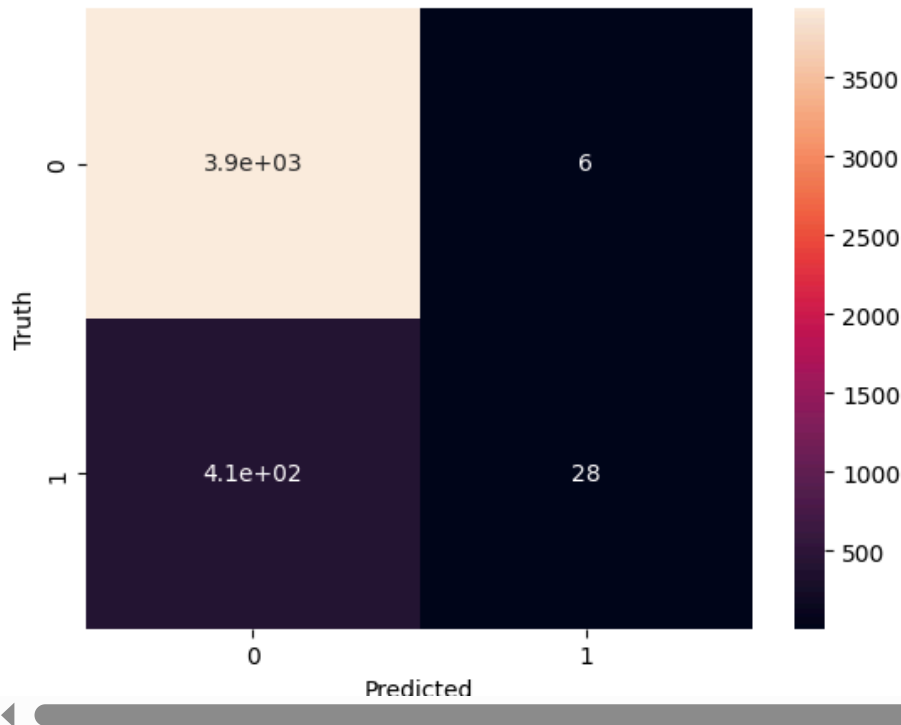
	precision	recall	f1-score	support
0	0.91	1.00	0.95	3942
1	0.82	0.06	0.12	440
accuracy			0.90	4382
macro avg	0.86	0.53	0.53	4382
weighted avg	0.90	0.90	0.87	4382

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[3936,    6],
       [ 412,   28]])
```

```
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(50.72222222222214, 0.5, 'Truth')
```



```
y_test.value_counts()
```

```
count
churn
0      3942
1       440
```

```
dtype: int64
```

Insights

1. Accuracy is high, but misleading:

The overall accuracy is ~90.5%, This suggests a high-performing model at first glance. However, accuracy can be misleading in cases of class imbalance

2. Class Imbalance is evident:

Class 0 (negative class) dominates the dataset with 3942 instances, while class 1 (positive class) only has 440 instances. This imbalance can cause the model to bias its predictions toward the majority class (class 0), leading to suboptimal performance for class 1.

3. The model struggles with predicting class 1:

False Negatives (FN): There are 416 instances where the model wrongly predicted class 0 instead of class 1. This means that class 1 instances are being missed in favor of predicting class 0.

False Positives (FP): Only 11 instances are wrongly predicted as class 1 when they are actually class 0. This shows that class 1 is rarely predicted incorrectly.

True Positives (TP): Only 24 instances of class 1 were correctly identified. This highlights a major issue in detecting the positive class, which could be critical depending on the application (e.g., fraud detection, rare disease diagnosis).

Precision and Recall for class 1 (positive class):

Precision (how many predicted class 1 are actually class 1): The model has a decent precision for class 1 (about 68.6%), meaning that when it predicts class 1, it's often correct, but it still has some false positives.

Recall (how many actual class 1 are detected):

Recall is very low for class 1 (5.45%), meaning the model is missing most of the actual class 1 instances. This is a major concern because it indicates that class 1 is being severely under-predicted.

Precision-Recall trade-off:

There's a significant precision-recall trade-off. While precision is relatively okay for class 1 (68.6%), the recall is very poor (5.45%). This means that the model has a high tendency to predict the majority class (class 0), and when it does predict class 1, it's often correct, but it's missing a lot of actual class 1 instances.

**** Possible Solutions to Improve the Model:****

Handle Class Imbalance:

Oversample the minority class (class 1) or undersample the majority class (class 0) to make the dataset more balanced.

Alternatively, use SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic samples for class 1.

Use Class Weights:

Assign higher class weights to the minority class (class 1) to penalize the model more for incorrect predictions on this class. This can encourage the model to focus on correctly classifying class 1.

Use Different Evaluation Metrics:

Accuracy alone is misleading. Focus on other metrics such as F1-score, Precision, and Recall:

F1-score for class 1: A balance between precision and recall, especially useful when you care equally about false positives and false negatives.

Adjust the Decision Threshold:

Consider adjusting the decision threshold to predict class 1 more often. The current threshold of 0.5 might not be ideal, especially when class 1 is underrepresented.

