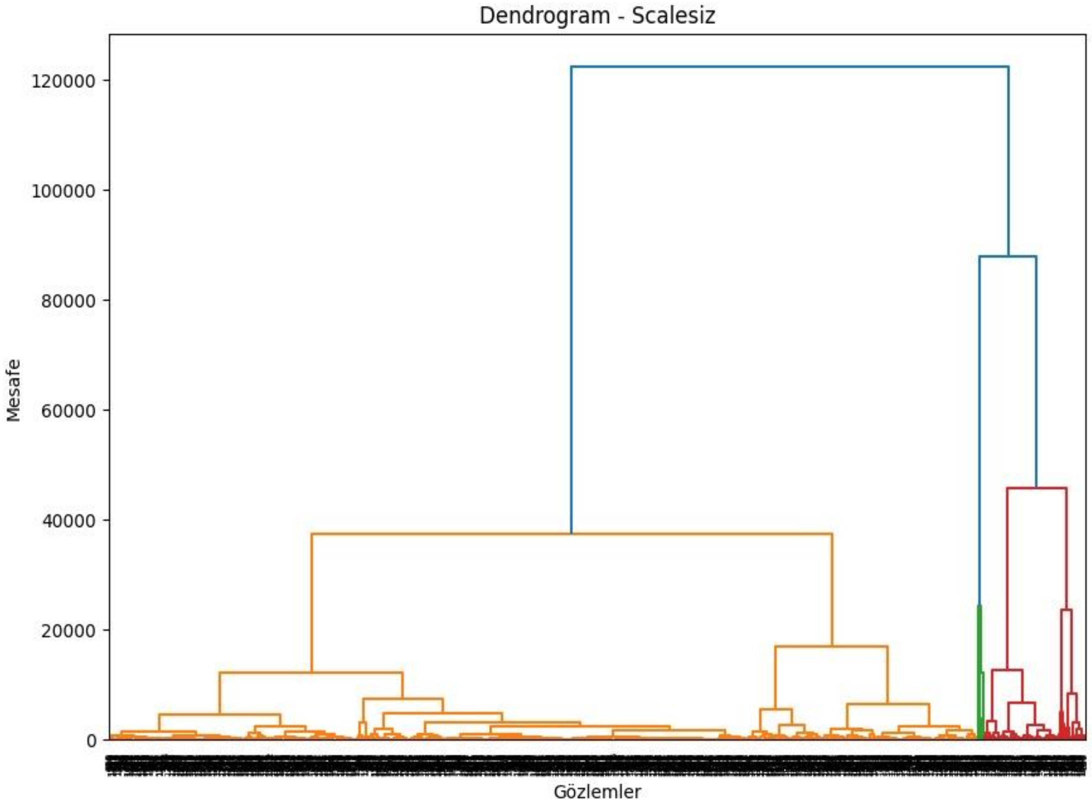
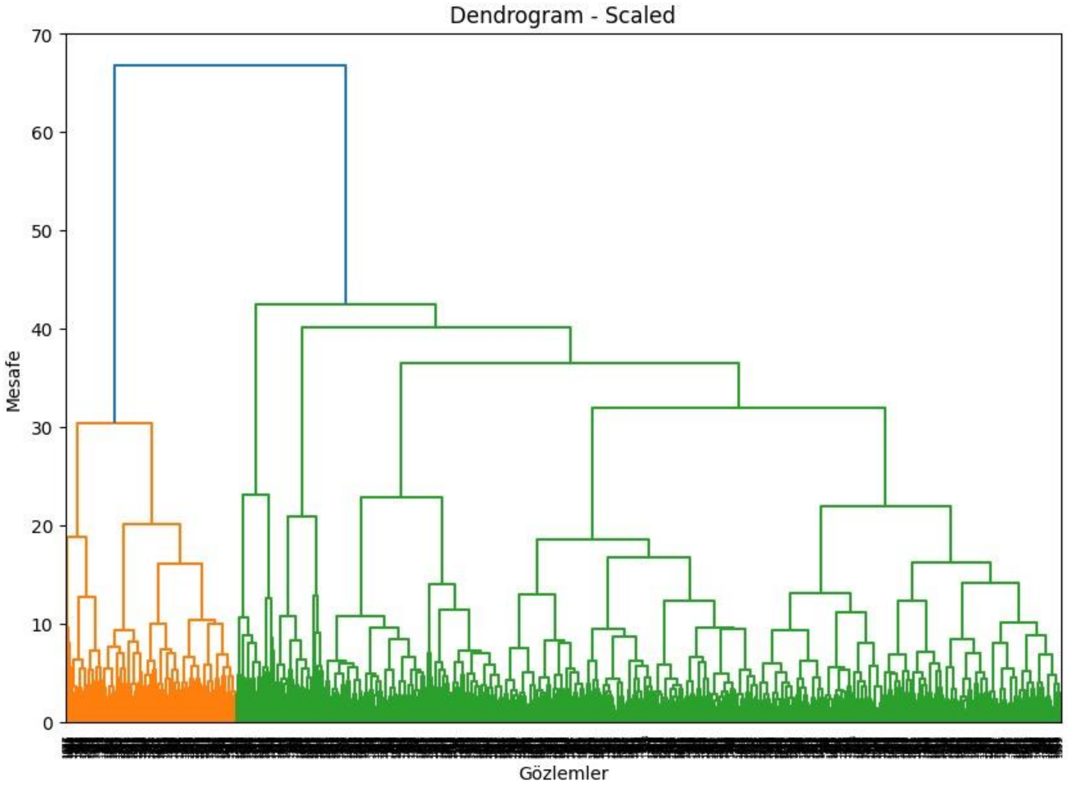
Efficient Telemarketing

Enes YILDIRIM 20211603001

Demir SARRAÇ 20231603050

Dendrograms





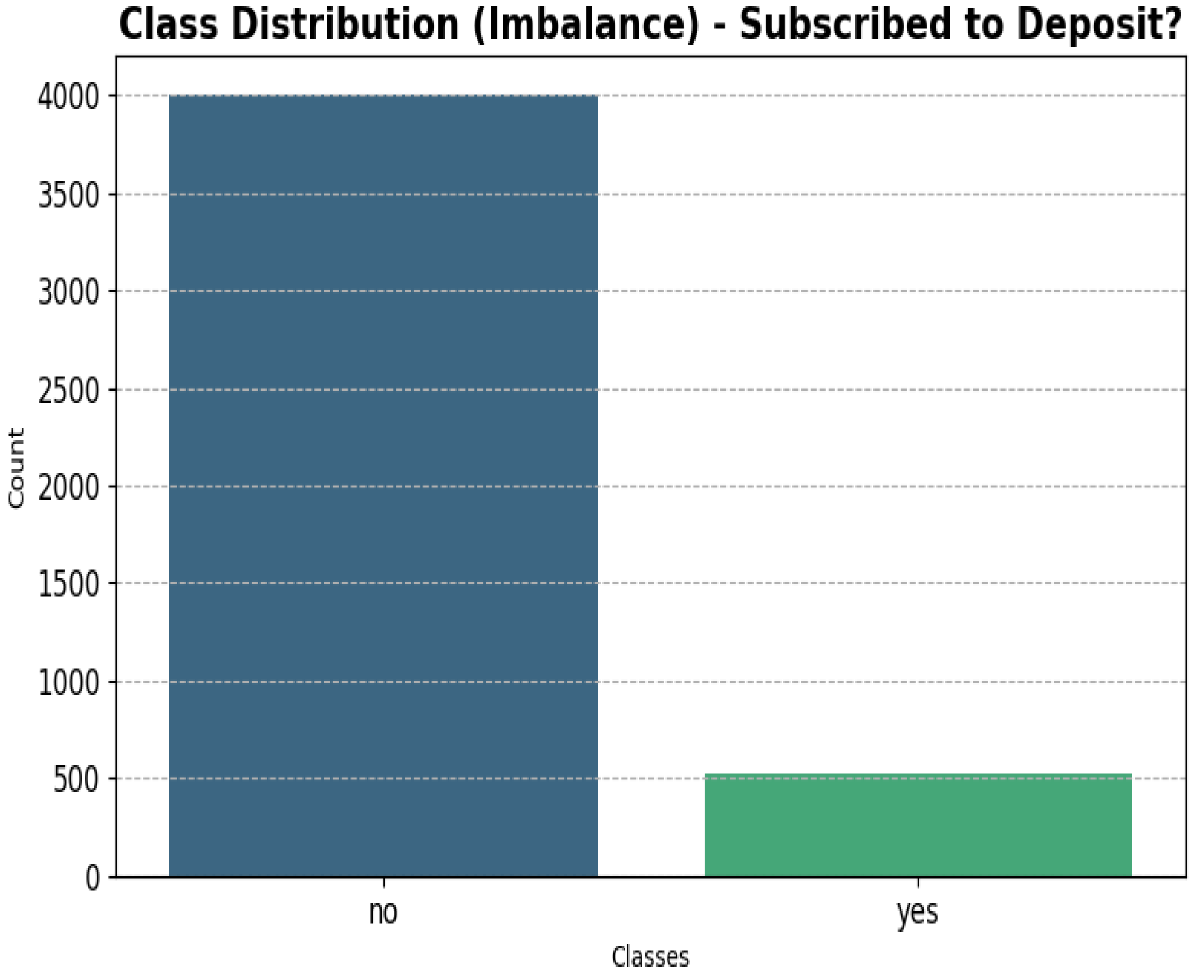
Is Data Set Balanced?

Unfortunately, our dataset contains 4000 "no" and 521 "yes" labels.

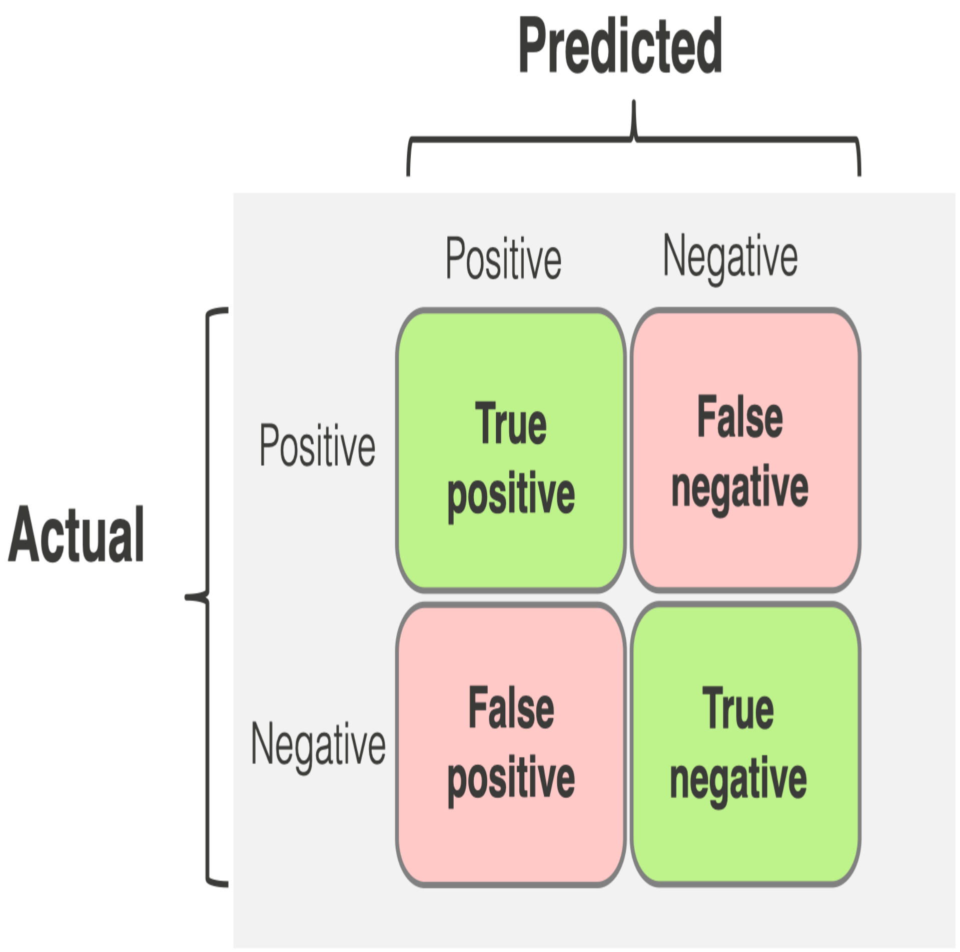
This means our dataset is imbalanced.

We will balance the dataset by downsampling the majority class.

Later, to compare, we will apply SMOTE.



- Recall that, using our dataset, we are trying to predict whether the client will subscribe to a deposit or not.
- Since we have **limited human resources**, we aim to predict those who will subscribe in advance to increase efficiency.
- To do this, we **prioritize** the **recall metric**, as it helps us minimize false negatives and ensures we don't miss potential subscribers.
- In this case, a **false negative** means **missing a potential customer** who would actually subscribe — leading to a significant loss in efficiency and potential revenue.



```python
no_df = df[df["y"] == "no"]
yes_df = df[df["y"] == "yes"]

downsampled_no = resample(no_df,
                          replace=False,
                          n_samples = len(yes_df),
                          random_state=57
)

balanced_df = pd.concat([yes_df, downsampled_no])
print(balanced_df.info())
print(balanced_df.shape)
print(balanced_df["y"].value_counts())
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 1042 entries, 13 to 979
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        1042 non-null   int64
 1   job        1042 non-null   object
 2   marital    1042 non-null   object
 3   education  1042 non-null   object
 4   default    1042 non-null   object
 5   balance    1042 non-null   int64
 6   housing    1042 non-null   object
 7   loan       1042 non-null   object
 8   contact    1042 non-null   object
 9   day        1042 non-null   int64
 10  month      1042 non-null   object
 11  duration   1042 non-null   int64
 12  campaign   1042 non-null   int64
 13  pdays      1042 non-null   int64
 14  previous   1042 non-null   int64
 15  poutcome   1042 non-null   object
 16  y          1042 non-null   object
dtypes: int64(7), object(10)
memory usage: 146.5+ KB
None
...
y
yes    521
no     521
Name: count, dtype: int64
```

Logistic Regression & Treshold

```
logistic_model = LogisticRegression(random_state=57)
logistic_model.fit(X_train_scaled, y_train)

y_pred  =logistic_model.predict(X_test_scaled)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

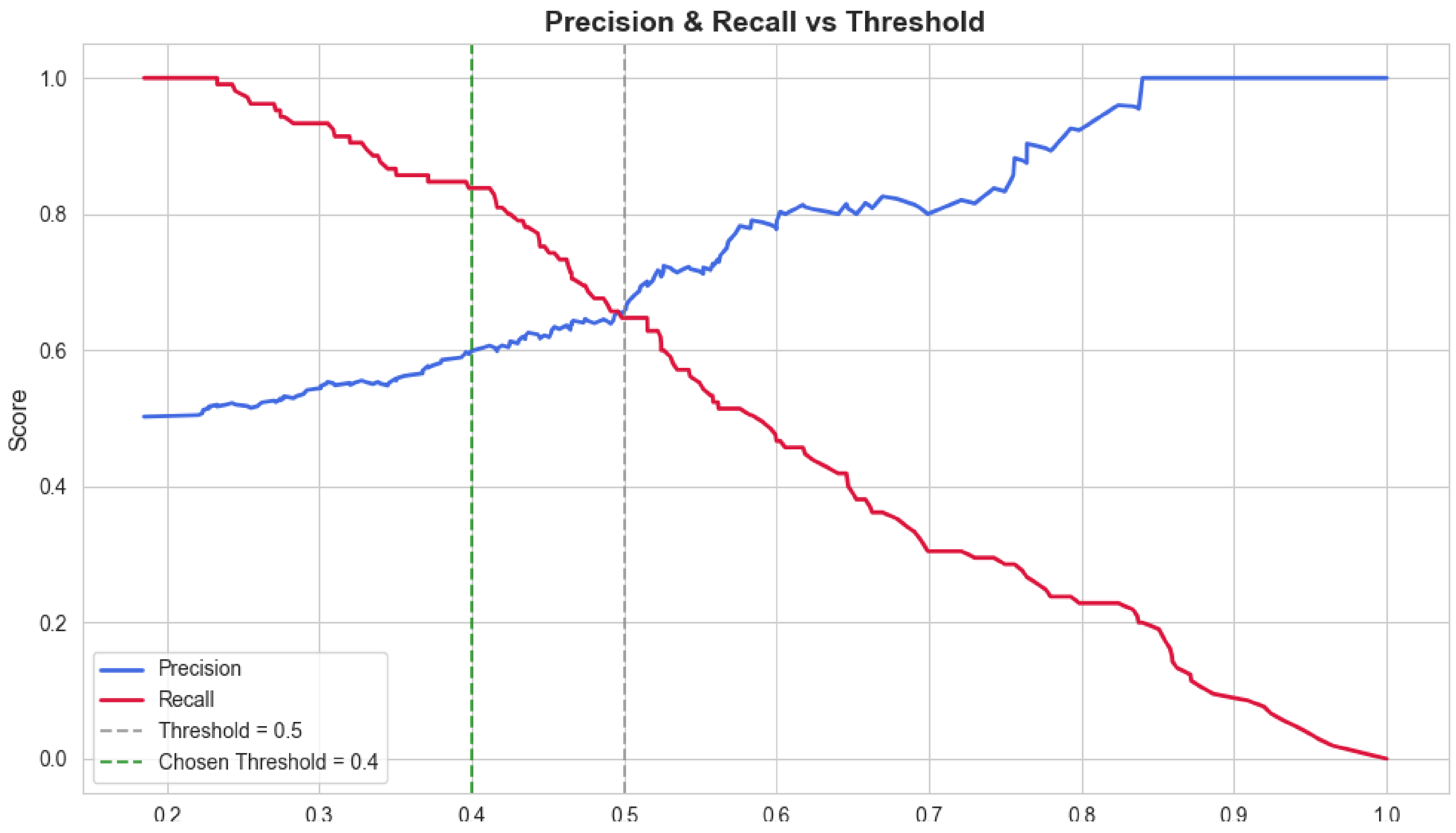
✓ 0.0s

```
[[69 35]
 [37 68]]
              precision    recall  f1-score   support

           0       0.65      0.66      0.66       104
           1       0.66      0.65      0.65       105

    accuracy                           0.66       209
   macro avg       0.66      0.66      0.66       209
weighted avg       0.66      0.66      0.66       209
```

```
y_probs = logistic_model.predict_proba(X_test_scaled)[:,1]
treshold = 0.4

y_pred_custom = (y_probs >= treshold).astype(int)

print(confusion_matrix(y_test, y_pred_custom))
print(classification_report(y_test, y_pred_custom))
```

✓ 0.0s

```
[[46 58]
 [17 88]]
              precision    recall  f1-score   support

           0       0.73      0.44      0.55       104
           1       0.60      0.84      0.70       105

    accuracy                           0.64       209
   macro avg       0.67      0.64      0.63       209
weighted avg       0.67      0.64      0.63       209
```



- To find the best trade-off point, we need to know the cost of each type of mistake.
- However, a detailed cost-sensitive analysis is beyond the scope of our current project

KNN & Neighbour Number

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
[[67 37]
 [42 63]]
              precision    recall  f1-score   support

           0       0.61      0.64      0.63       104
           1       0.63      0.60      0.61       105

    accuracy                           0.62       209
   macro avg       0.62      0.62      0.62       209
weighted avg       0.62      0.62      0.62       209
```

```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
[[69 35]
 [47 58]]
              precision    recall  f1-score   support

           0       0.59      0.66      0.63       104
           1       0.62      0.55      0.59       105

    accuracy                           0.61       209
   macro avg       0.61      0.61      0.61       209
weighted avg       0.61      0.61      0.61       209
```

Decision Tree Classifier & Random Forest Classifier

```
dt_model = DecisionTreeClassifier(random_state=57, max_depth=5)

dt_model.fit(X_train_scaled, y_train)

y_pred = dt_model.predict(X_test_scaled)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

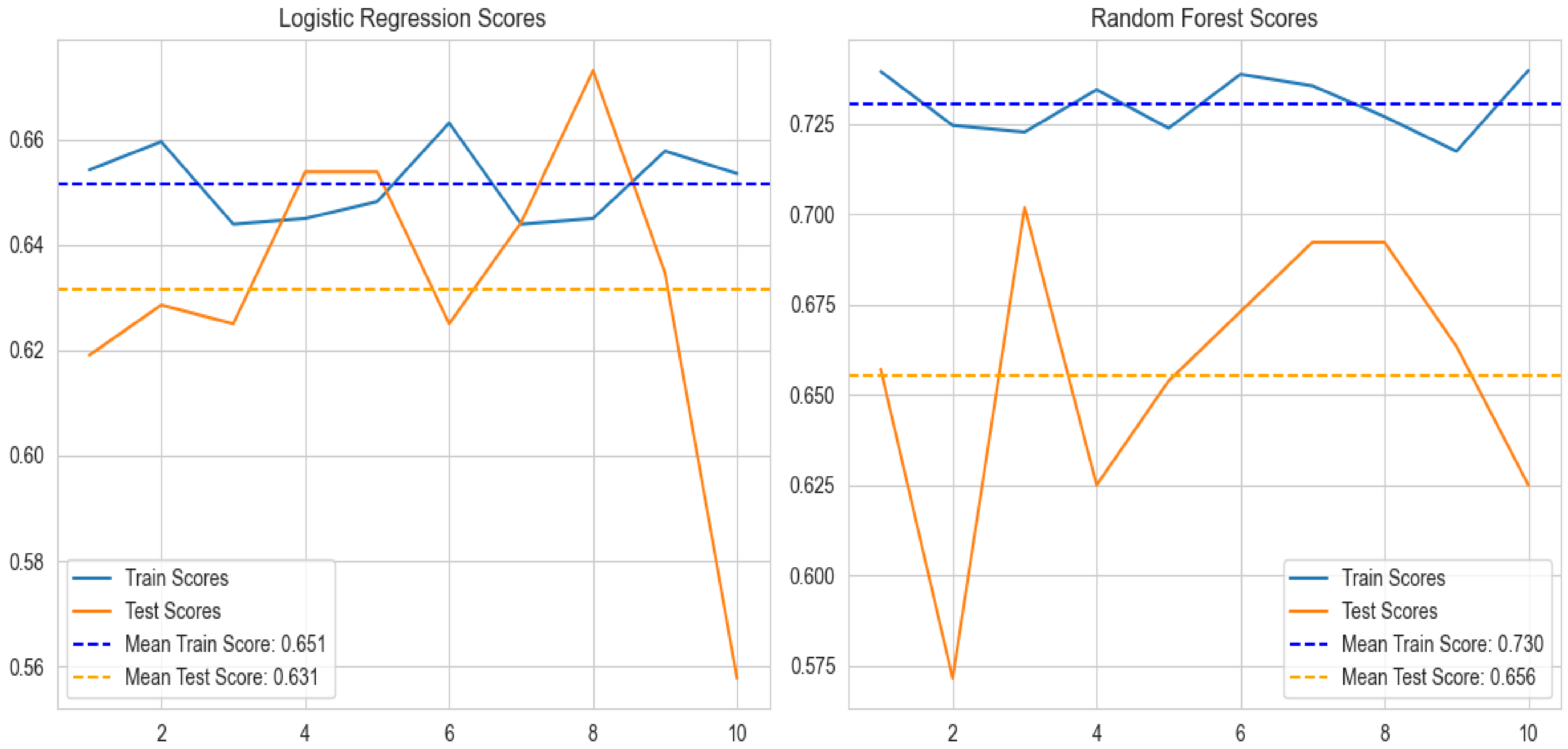
✓ 0.0s

```
[[33 71]
 [17 88]]
              precision    recall  f1-score   support

           0       0.66      0.32      0.43       104
           1       0.55      0.84      0.67       105

    accuracy                           0.58       209
   macro avg       0.61      0.58      0.55       209
weighted avg       0.61      0.58      0.55       209
```

```
rf_model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=57)
rf_model.fit(X_train_scaled, y_train)
y_pred = rf_model.predict(X_test_scaled)

print(rf_model.score(X_test_scaled, y_test))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
total = tn + fp + fn + tp
```

✓ 0.0s

```
0.6746411483253588
[[83 21]
 [47 58]]
              precision    recall  f1-score   support

           0       0.64      0.80      0.71       104
           1       0.73      0.55      0.63       105

    accuracy                           0.67       209
   macro avg       0.69      0.68      0.67       209
weighted avg       0.69      0.67      0.67       209
```




```
logistic regression scores  0.6514193781815121 0.631492673992674
random forest scores  0.7304332886565799 0.6555494505494506
```

Logistic Regression With SMOTE

```
logistic_model = LogisticRegression(random_state=57)
logistic_model.fit(X_train_res_scaled, y_train_res)

y_pred  =logistic_model.predict(X_test_scaled)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
[[745  56]
 [ 82  22]]
              precision    recall  f1-score   support

           0       0.90      0.93      0.92       801
           1       0.28      0.21      0.24       104

    accuracy                           0.85       905
   macro avg       0.59      0.57      0.58       905
weighted avg       0.83      0.85      0.84       905
```

```
y_probs = logistic_model.predict_proba(X_test_scaled)[:,1]
treshold = 0.4

y_pred_custom = (y_probs >= treshold).astype(int)

print(confusion_matrix(y_test, y_pred_custom))
print(classification_report(y_test, y_pred_custom))
```

✓ 0.0s

```
[[716  85]
 [ 70  34]]
              precision    recall  f1-score   support

           0       0.91      0.89      0.90       801
           1       0.29      0.33      0.30       104

    accuracy                           0.83       905
   macro avg       0.60      0.61      0.60       905
weighted avg       0.84      0.83      0.83       905
```

In the graphs above, SMOTE was applied and we used a balanced dataset consisting of 4000 "yes" and 4000 "no" labels. Interestingly, downsampling performed better than SMOTE in terms of recall. Additionally, we once again observed that lowering the threshold increases recall.