SEEKING TREATMENT FOR MENTAL ILLNESS

GOALS: build a predictive model to determine if someone would seek treatment for a mental illness or not. I geared this project towards everyday friends and family members who feel someone close to them may be struggling with a mental illness.

DATASET: Mental Health Dataset from Kaggle

The features are all clear and defined and the overall data looks clean. The data is in a CSV format. It consists of just under 300,000 rows and 17 columns. The data types are all objects and are mostly strings or booleans.

EDAs and Feature Selection:

- Dropped Timestamps column
- Kept only cases from USA, dropped Country column
- Dropped Null Values from the Self-Employed column
- Downsized dataset to balance Treatment column
  - 50,000 Yes values
  - 50,000 No values

Left with 100,000 rows

Dataset = entirely categorical values.

Created Chi-Squared test For Loop to determine which features correlate the most with my target variable.

Variable: Gender
Chi-Squared Statistic: 2399.6832634032644
P-value: 0.00000
Degrees of Freedom: 1
Decision: Reject the null hypothesis - There is a significant association between the variables.

Variable: Social_Weakness
Chi-Squared Statistic: 0.6468916752472398
P-value: 0.72365
Degrees of Freedom: 2
Decision: Fail to reject the null hypothesis - There is no significant association between the variables.

CONFUSION MATRIX COMPONENTS:

- **True Positives**: The number of individuals who were correctly predicted to seek treatment for a mental illness (i.e., the model predicted "Yes" for treatment, and the actual value was also "Yes").
- **True Negatives**: The number of individuals who were correctly predicted not to seek treatment for a mental illness (i.e., the model predicted "No" for treatment, and the actual value was also "No").
- **False Positives**: The number of individuals who were incorrectly predicted to seek treatment for a mental illness (i.e., the model predicted "Yes" for treatment, but the actual value was "No").
- **False Negatives**: The number of individuals who were incorrectly predicted not to seek treatment for a mental illness (i.e., the model predicted "No" for treatment, but the actual value was "Yes").

TARGET VARIABLE: Treatment Column (Mapped to Yes = 1 and No = 0)

METRIC: Accuracy and ROC (Balanced Dataset)

ONE HOT ENCODER: Used as only preprocessor. (18 columns including Treatment after OHE)

PIPELINE: Set up Basic Pipeline with Column Transformer, One Hot Encoder and empty Model slot.

Baseline Model – Logistic Regression

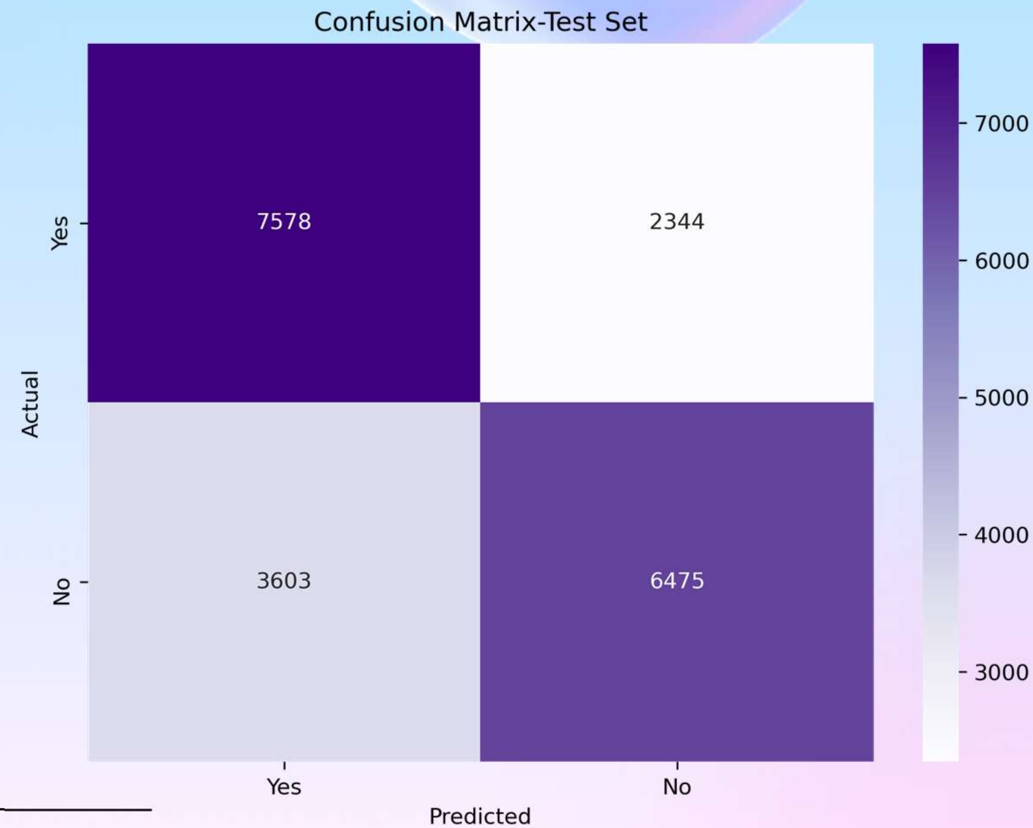
Accuracy Score: 70.26% on Test Set

AUC-ROC Score: 76.9%

The Accuracy score on the Test set is: 0.70265

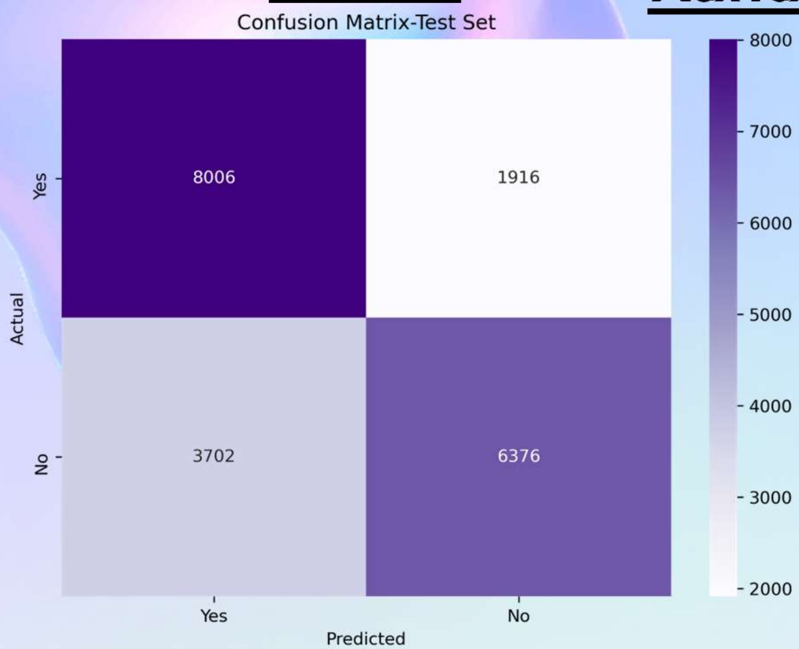
Classification Report on the Test set:

	precision	recall	f1-score	support
0	0.73	0.64	0.69	10078
1	0.68	0.76	0.72	9922
accuracy			0.70	20000
macro avg	0.71	0.70	0.70	20000
weighted avg	0.71	0.70	0.70	20000



Random Forest Classifier

ORIGINAL



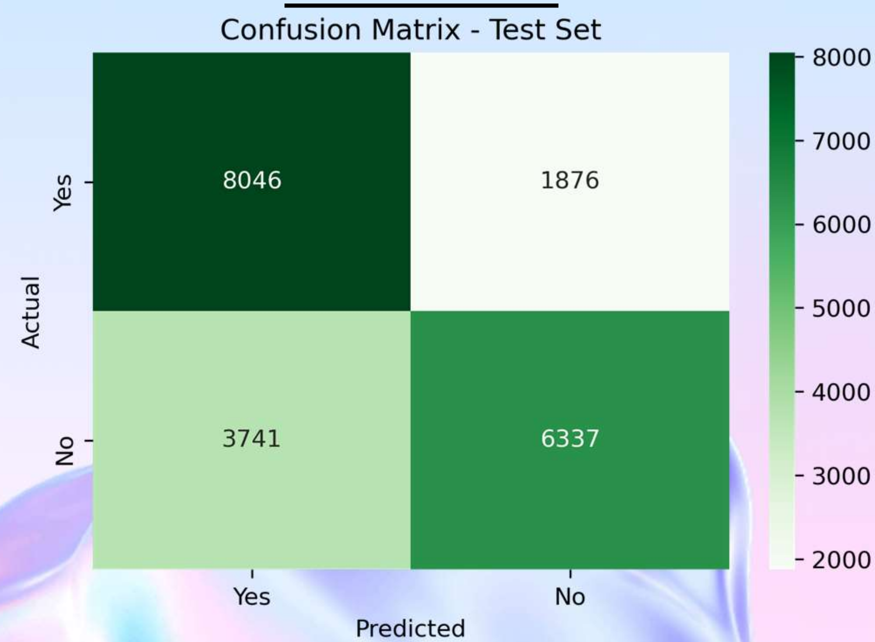
ACCURACY SCORES:

Original Train Set: 72.75% --- **HyperTuned Train Set:** 72.76%

Original Test Set: 71.91% --- **HyperTuned Test Set:** 71.915%

ROC Curve Original Test Set: 79.527%

HYPERTUNED



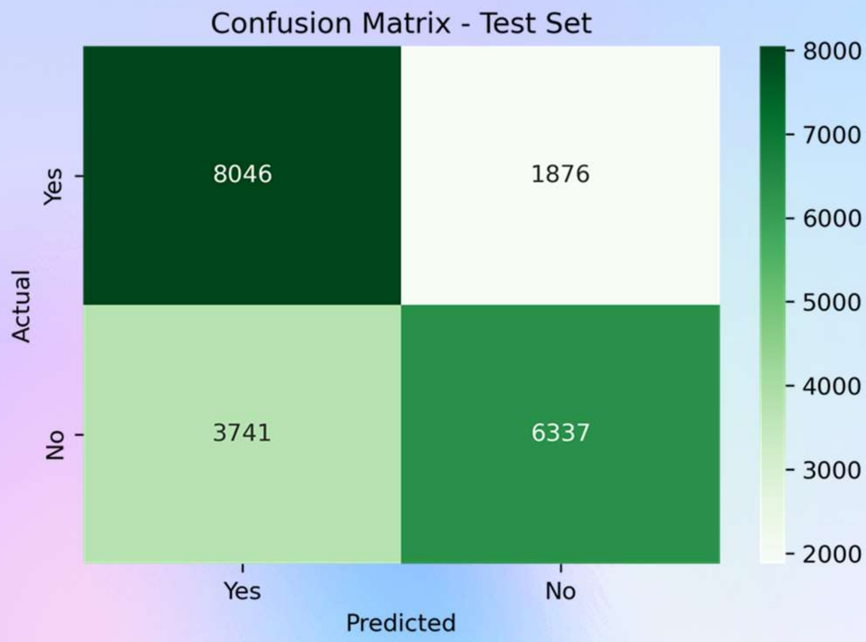
TP: Original Test Set: 8,006 ------ HyperTuned Test Set: 8,046

TN: Original Test Set: 6,376 ------ HyperTuned Test Set: 6,337

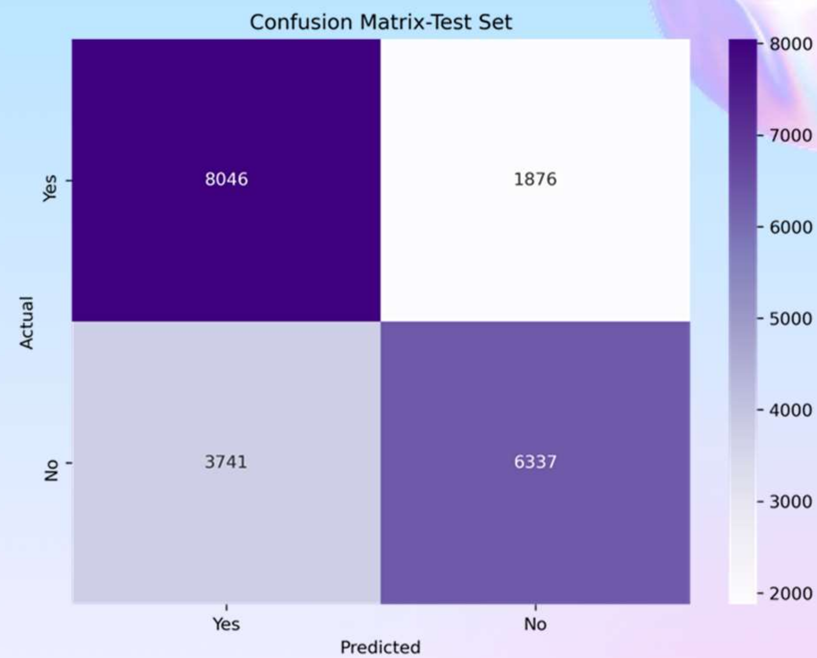
FP: Original Test Set: 3,702 ------ HyperTuned Test Set: 3,741

FN: Original Test Set: 1,916 ------ HyperTuned Test Set: 1,876

HYPERTUNED - RandomForest

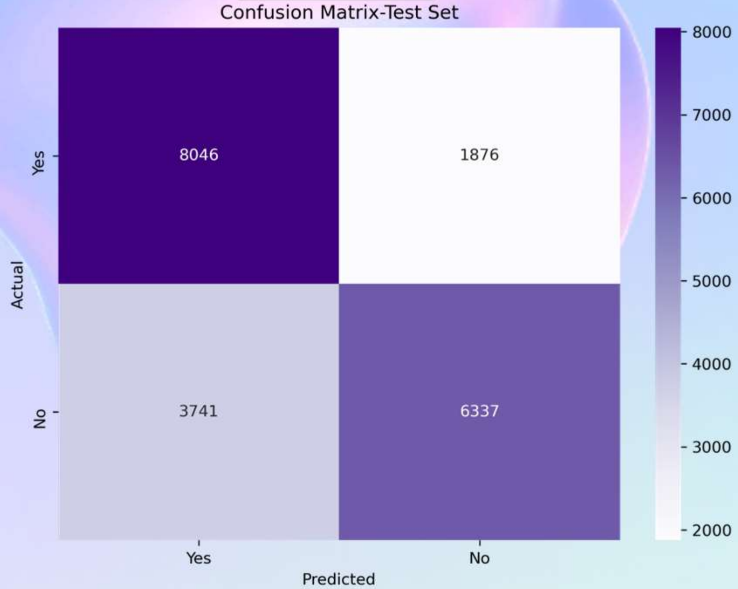


ORIGINAL - XGBoost



XGBoost Classifier

ORIGINAL

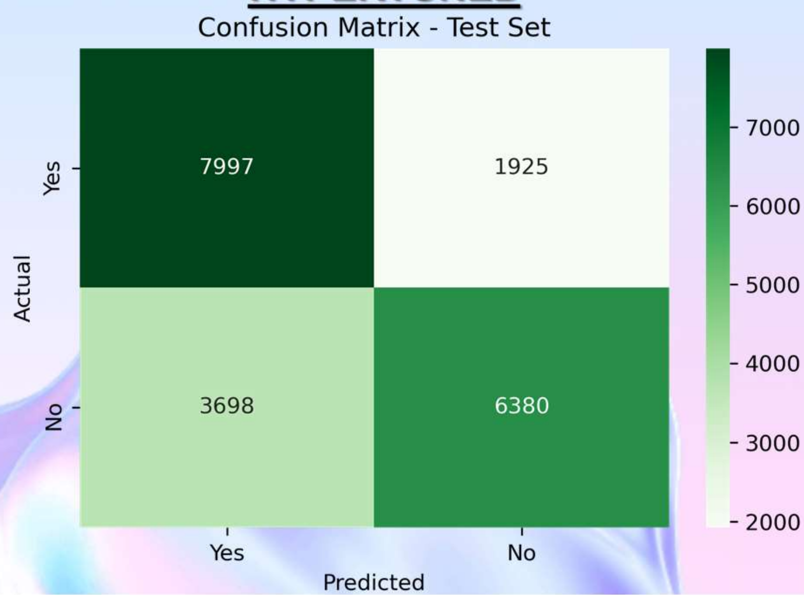


Original Train Set: 72.76% --- **HyperTuned Train Set:** 72.73%

Original Test Set: 71.915% --- **HyperTuned Test Set:** 71.885%

ROC Curve Original Test Set: 79.534%

HYPERTUNED



TP: Original Test Set: 8,046 ------ HyperTuned Test Set: 7,997

TN: Original Test Set: 6,337 ------ HyperTuned Test Set: 6,380

FP: Original Test Set: 3,741 ------ HyperTuned Test Set: 3,698

FN: Original Test Set: 1,876 ------ HyperTuned Test Set: 1,925

MODEL STACKING

STEPS TO MODEL STACKING:

1. Defined Base Models as the Logistic Regression model and the HyperTuned Random Forest and XGBoost model.

2. Created a Meta-Model as my final estimator.

3. Stacked the base models and final estimator using Stacking Classifier.

4. Established and Fit the Final Pipeline

5. Calculated the Accuracy and Classification Report.

6. Rinse and Repeat steps 2 through 5 to experiment with different models and parameters to use as the Meta-Model.

7. Optional: Create Confusion Matrix

```python
# Defined base models with the respective Best Model Parameters
base_models = [
    ('log_reg', LogisticRegression(max_iter=1000, random_state=42)),
    ('random_forest', RandomForestClassifier()),
    ('xgboost', XGBClassifier())
]
```

```python
# Created the Meta-model and experimented using several different model types such as RandomFores
meta_model = KNeighborsClassifier(n_neighbors=10)

# Created the stacking ensemble using Stacking Classifier
stacking_ensemble = StackingClassifier(estimators=base_models, final_estimator=meta_model, cv=5)

# Final pipeline including preprocessing and stacking ensemble
final_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('stacked_models', stacking_ensemble)
])

# Fited the pipeline on the Training Set
final_pipeline.fit(X_train, y_train)

# Made predictions on the Test Set using the final pipeline
y_pred_stack = final_pipeline.predict(X_test)
y_pred_proba = final_pipeline.predict_proba(X_test)[:, 1]

# Calculated the Accuracy and Classification Reports
accuracy_stackTest = accuracy_score(y_test, y_pred_stack)
classification_report_stackTest = classification_report(y_test, y_pred_stack)


print(f'Accuracy on Test Data: {accuracy_stackTest}')
print('Classification Report on Test Data:\n', classification_report_stackTest)
```

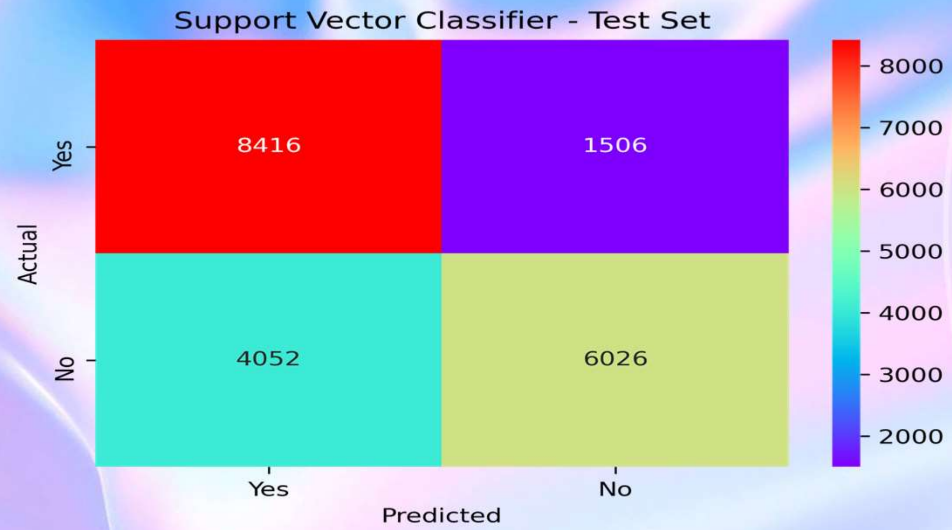
EXPERIMENTS AND FINDINGS:

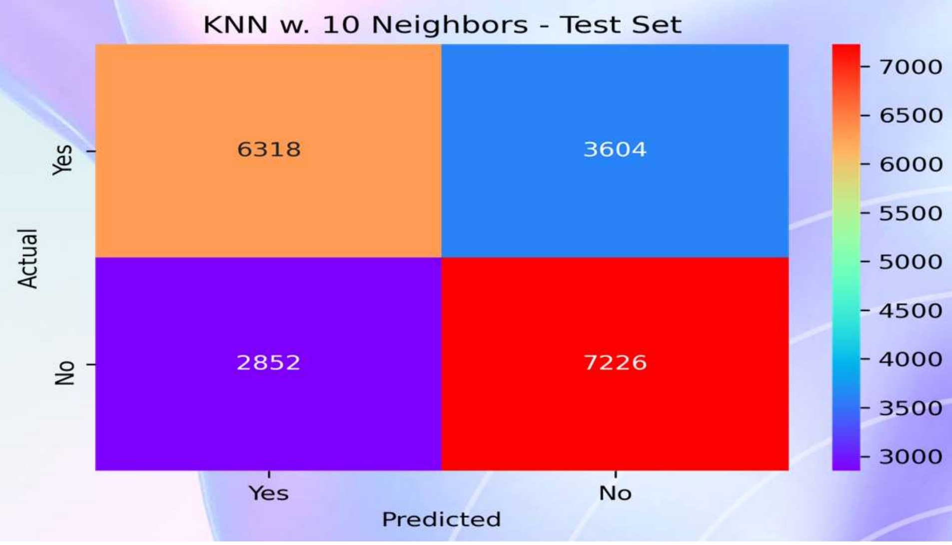
Support Vector Classifier:
Accuracy: = 72.21%
TP: = 8,416
TN: = 6,026
FP: = 4,052
FN: = 1,506

K-Nearest Neighbor:
Accuracy: = 67.72%
TP: = 6,318
TN: = 7,226
FP: = 2,852
FN: = 3,604

Random Forest:
Accuracy: = 72.06%
TP: = 8,177
TN: = 6,235
FP: = 3,843
FN: = 1,745

Logistic Regression:
Accuracy = 72.05%
TP: = 8,183
TN: = 6,227
FP: = 3,851
FN: = 1,739





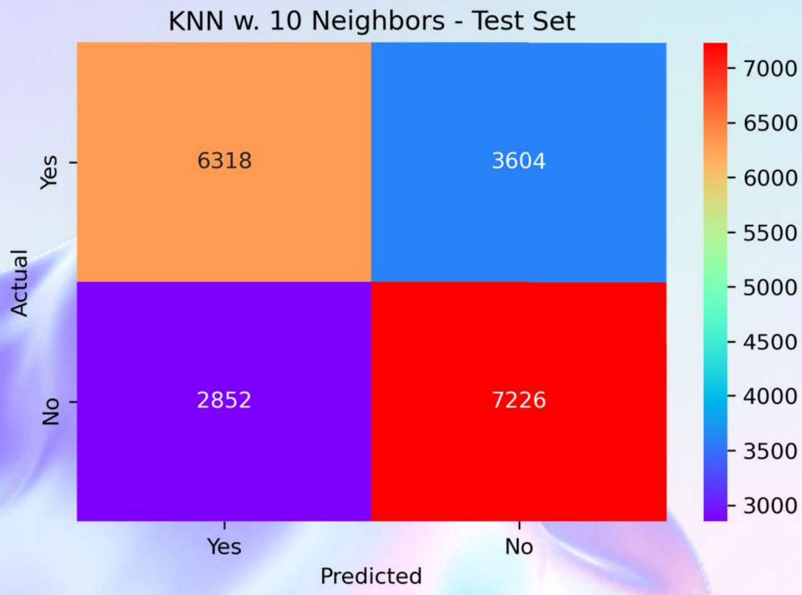
- **SVC had the highest Accuracy score of all the models. High False Positives.**
- **KNN w. 10 Neighbors had the lowest Accuracy Score. Lowest False Positives and Highest True Negatives. 76% ROC score.**

FINAL MODEL SELECTION

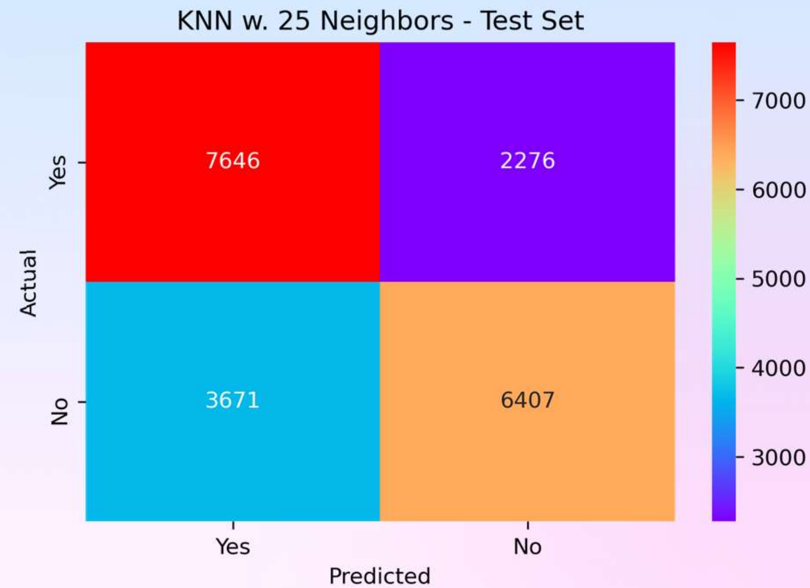
- Ran GridSearchCV to find the best Number of Neighbors for my KNN model
- Ran Experiments with various Number of Neighbors in the Stacked Model

- KNN w. 10 Neighbors was still the best model for the results I was looking for.

Accuracy: 67.72%
ROC Score: 76%



Accuracy: 70.265%
ROC Score: 78%



CONCLUSION:

- Was able to slightly raise overall Accuracy score.
- Chose KNN w. 10 Neighbors.
  - Most True Negatives
  - Least False Positives

NEXT STEPS:

1. Rerun my models and focus on Feature Importance to try to improve my overall scores
2. Add in an interactive Visualization that shows feature correlation
3. Deploy the model with appropriate warnings and resources available

KNN

THANK YOU

Lotus Baumgarner

LotusBaumgarner@gmail.com

https://www.linkedin.com/in/lotus-baumgarner/

https://github.com/Lotus-baumgarner/Will-They-Seek-Treatment