Summary

Our project is based on the student mental health dataset, and our goal in this project was to determine whether or not there was a correlation between anxiety and another variable, and if we could possibly predict the likelihood of someone having anxiety.

We felt inclined to taking on this dataset because of how valuable it could be to have a way to possibly predict anxiety rates among students.

Data Exploration and Data Visualizations

For Data Exploration, I compared and analyzed all numerical/quantitative variables in the dataset, which were: Age, and GPA.

For Data Visualization, I will compare the numerical variables with anxiety to try to see if there is any pattern or correlation between anxiety and the two numerical variables.

I will also test for any correlation between anxiety and categorical variables, like gender, major, and class status.

Data Exploration:
Numerical Variables

```
df.info()
df.describe(include='all')
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 22811 entries, 0 to 24999
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                22811 non-null  int64
 1   gender            22811 non-null  object
 2   age               22811 non-null  int64
 3   major             22811 non-null  object
 4   gpa               22811 non-null  float64
 5   class_status      22811 non-null  object
 6   marital_status    22811 non-null  object
 7   have_depression   22811 non-null  object
 8   have_anxiety      22811 non-null  object
 9   have_panicattacks 22811 non-null  object
 10  seeked_treatment  22811 non-null  object
dtypes: float64(1), int64(2), object(8)
memory usage: 2.1+ MB
```

	id	gender	age	major	gpa	class_status	marital_status	have_depression	have_anxiety	have_panicattacks	seeked_treatment
count	22811.000000	22811	22811.000000	22811	22811.000000	22811	22811	22811	22811	22811	22811
unique	NaN	8	NaN	9	NaN	4	2	2	2	2	2
top	NaN	Male	NaN	Social Sciences	NaN	Freshman	No	No	No	No	No
freq	NaN	10402	NaN	4936	NaN	11316	20737	15211	15176	18230	19575
mean	12501.157161	NaN	19.994169	NaN	2.903879	NaN	NaN	NaN	NaN	NaN	NaN
std	7214.929155	NaN	1.998314	NaN	0.635372	NaN	NaN	NaN	NaN	NaN	NaN
min	1.000000	NaN	17.000000	NaN	1.800000	NaN	NaN	NaN	NaN	NaN	NaN
25%	6253.500000	NaN	18.000000	NaN	2.360000	NaN	NaN	NaN	NaN	NaN	NaN
50%	12474.000000	NaN	20.000000	NaN	2.910000	NaN	NaN	NaN	NaN	NaN	NaN
75%	18768.500000	NaN	22.000000	NaN	3.460000	NaN	NaN	NaN	NaN	NaN	NaN
max	25000.000000	NaN	23.000000	NaN	4.000000	NaN	NaN	NaN	NaN	NaN	NaN

Mean

Age: 19.994169 (20 years old)

GPA: 2.903879 (2.9 avg gpa)

Median

Age: 20 (20 years old)

GPA: 2.91 (2.91 median gpa)

Min

Age: 17 (17 years old)

GPA: 1.8 (1.8 min. gpa)

Max

Age: 23 (23 years old)

GPA: 4.0 (4.0 max. gpa)

Range

Age: 23-17= 6 year age range

GPA: 4.0-1.8= 2.2 gpa range

IQR

Age: 22-18= 4 year interquartile age range

GPA: 3.46-2.36= 1.10 Interquartile GPA Range

Standard Deviation

Age: 1.998314 (standard deviation of about 2 years)

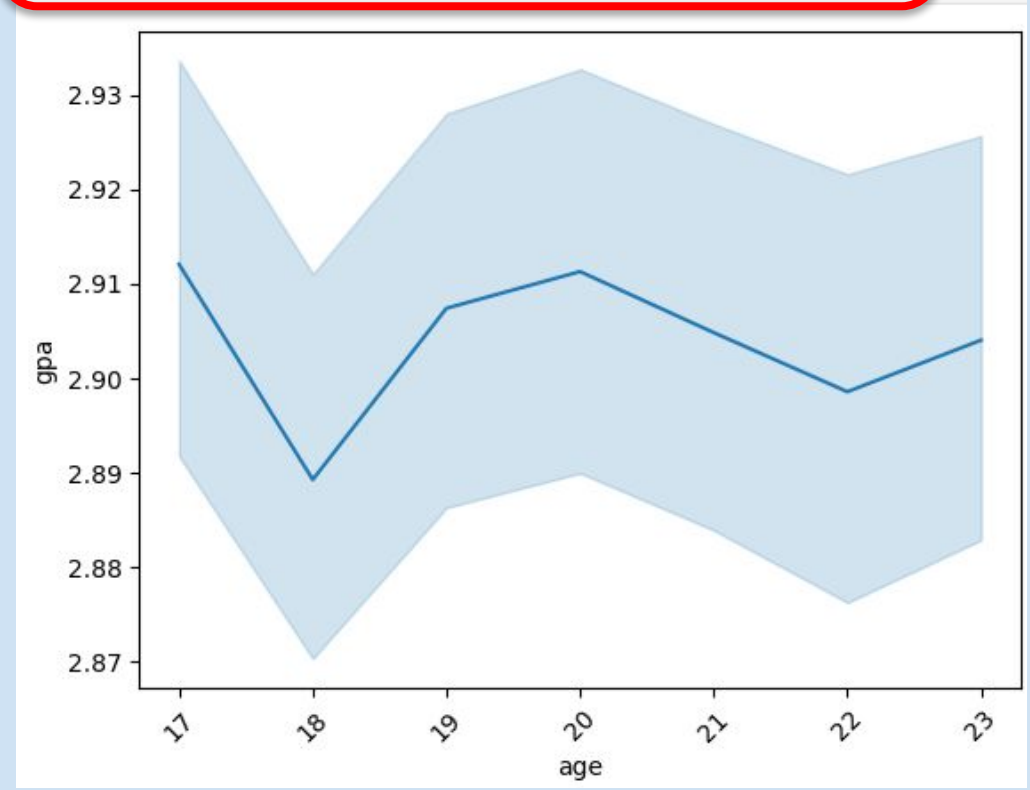
GPA: 0.635372 (standard deviation of about 0.64 points)

Data Visualization: Comparing numerical variables.

Line plot comparing age and gpa

Interpretation:
Students ages 17 and 21 are most likely to have a highest gpa's, while those ages 18 and 22, are shown to have the lowest gpa's.

```
sns.lineplot(data=data, x="age", y="gpa")
plt.xticks(rotation=45)
plt.show()
#the type of major does not significantly affect gpa given the graph
```

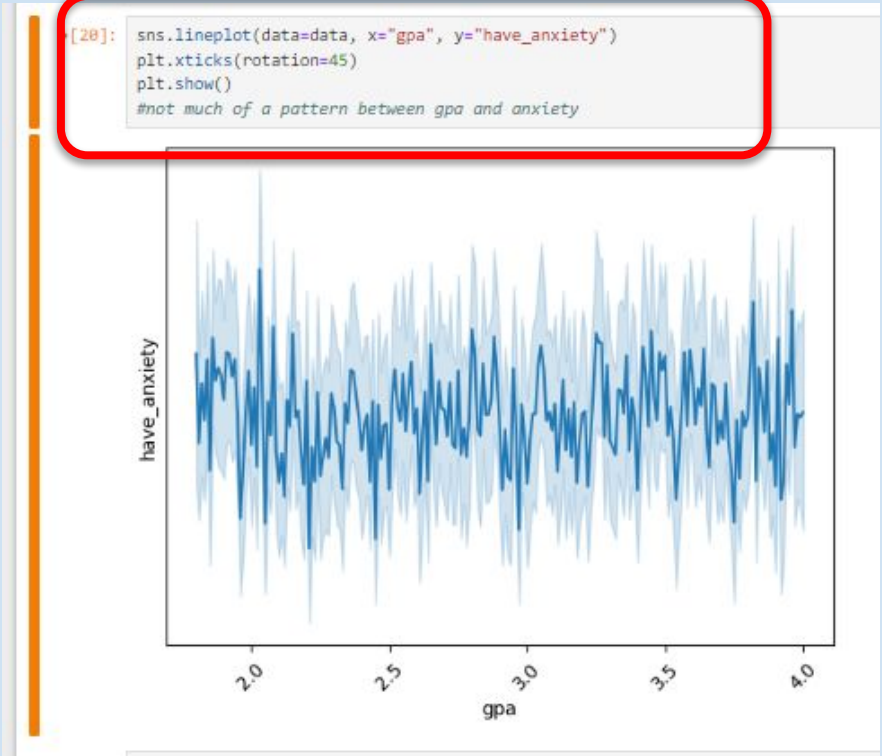


Anxiety vs. GPA

There doesn't seem to be a pattern or significant correlation between gpa and anxiety.

Interpretation:

There is no correlation or relationship between anxiety rates and gpa.

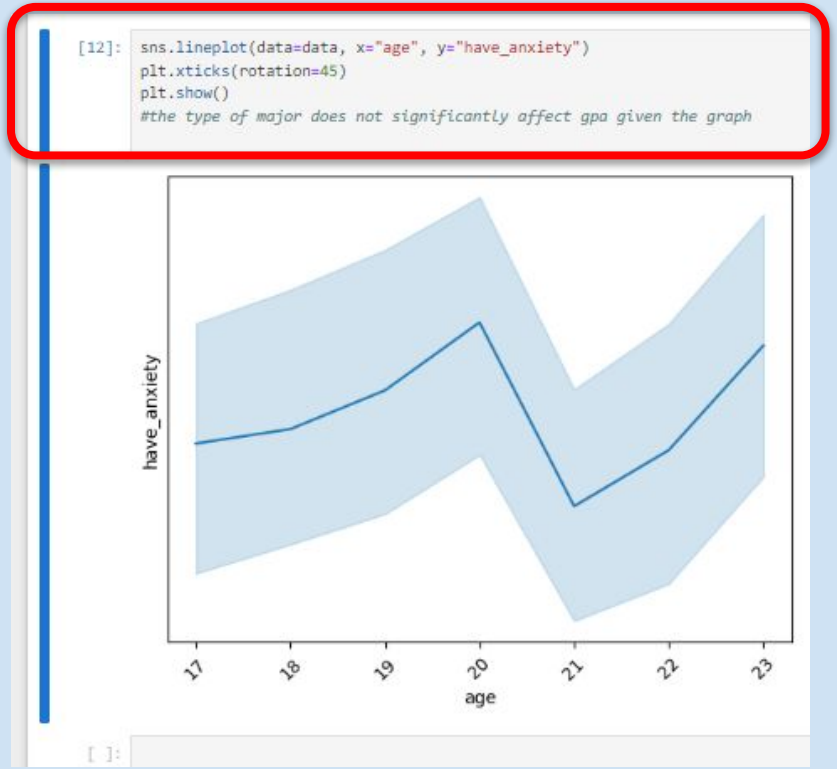


Anxiety vs. Age

There is a noticeable pattern with Age and Anxiety; Anxiety rates peak at age 20, followed by a massive dip in rates, reaching its lowest point at around age 21, with a gradual rise afterwards.

Interpretation:

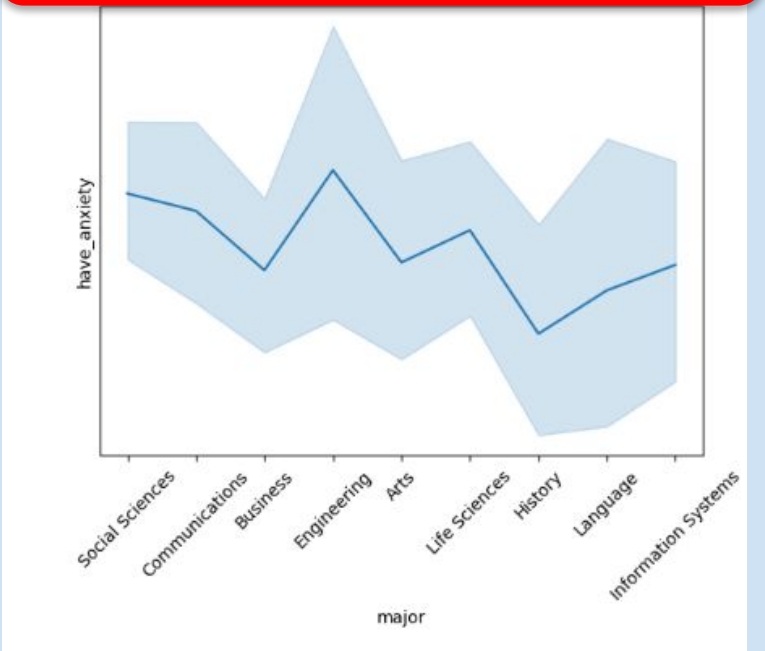
Among students ages ranging from 17-23, those ages 20, and 23 are most likely to have anxiety. Students who are 21, are shown to be least likely to have anxiety.



Anxiety vs. Majors

Students who major in engineering are shown to be most likely to have anxiety; students studying history are shown to be least likely to have anxiety.

```
[21]: sns.lineplot(data=data, x="major", y="have_anxiety")
      plt.xticks(rotation=45)
      plt.show()
      #the type of major does not significantly affect gpa given the graph
```

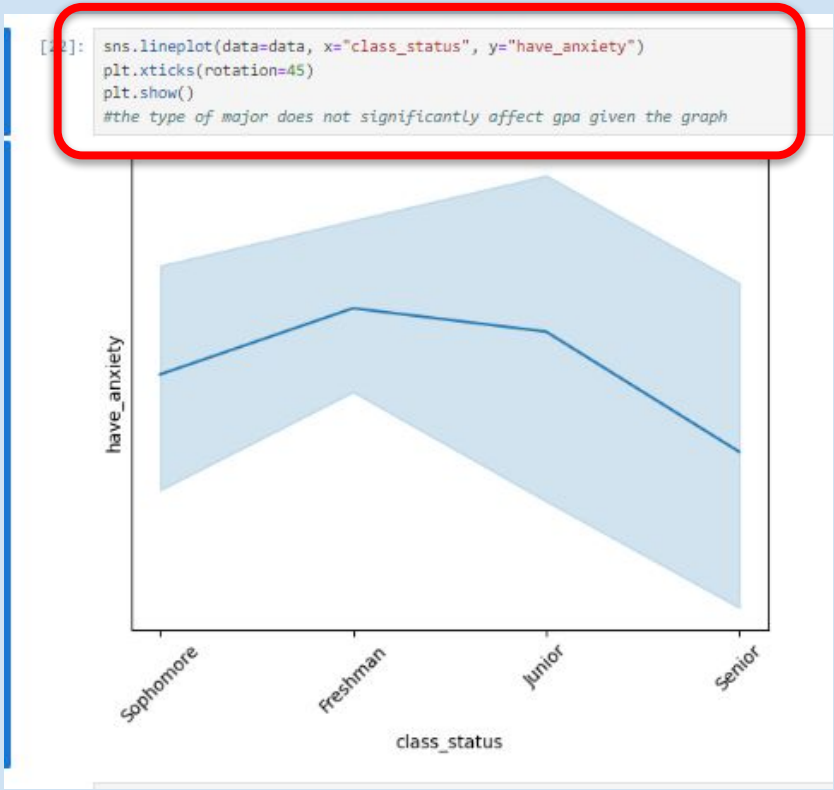


Vs. Class Status

Seniors are least likely to have anxiety

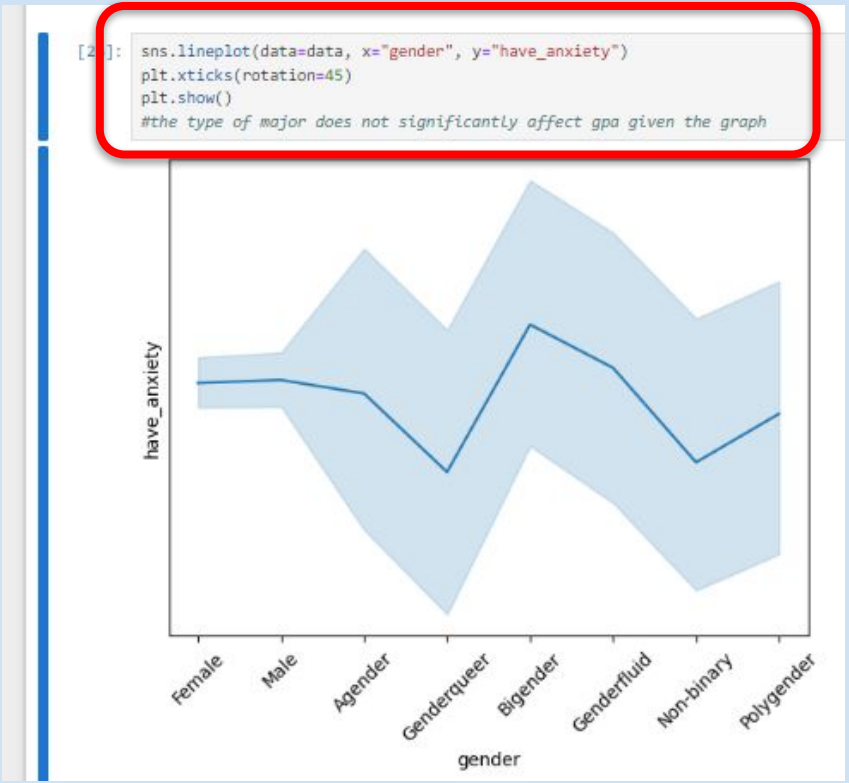
Freshmen are most likely to have anxiety, only marginally.

There is some noticeable correlation between class status and anxiety, but only really among seniors is the difference significant.



Vs. Gender

Students identifying as Bigender are more likely to have anxiety, while students who are genderqueer and/or non-binary, are shown to be least likely to have anxiety.



Data Curation

Data Curation

Missing values found in mental data are removed using dropna():

```
data = pd.read_csv('final_project_student_mental_health_dataset.csv', index_col = 'id')
data[data.isna().any(axis=1)]
```

id	gender	age	major	gpa	class_status	marital_status	have_depression	have_anxiety	have_panicattacks	seeked_treatment
3	Male	20	Business	1.96	Junior	NaN	No	No	Yes	Yes
37	Male	20	Social Sciences	2.51	Freshman	No	Yes	NaN	NaN	No
50	Female	23	Information Systems	3.99	Freshman	NaN	Yes	No	No	No
66	Male	17	Life Sciences	3.81	Sophomore	NaN	Yes	No	No	No
79	Female	22	Social Sciences	3.46	Freshman	NaN	No	No	No	No
...	...	...	...	...	...	...	...	...	...	...
24967	Male	17	Social Sciences	3.97	Freshman	NaN	No	No	No	No
24981	Female	18	Social Sciences	2.62	Freshman	NaN	No	Yes	No	No
24986	Male	21	Business	2.49	Sophomore	NaN	No	No	No	No
24992	Female	22	Communications	3.64	Freshman	No	NaN	No	No	NaN
24997	Male	20	Business	1.90	Freshman	No	NaN	Yes	No	NaN

```
data = data.dropna()
data
```

Dummy coding all categorical variables in mental health dataset:

```
data_dummied = pd.get_dummies(data)[['age', 'gpa', 'gender_Agender', 'gender_Bigender', 'gender_Female',
       'gender_Genderfluid', 'gender_Genderqueer', 'gender_Male',
       'gender_Non-binary', 'gender_Polygender', 'major_Arts',
       'major_Business', 'major_Communications', 'major_Engineering',
       'major_History', 'major_Information Systems', 'major_Language',
       'major_Life Sciences', 'major_Social Sciences', 'class_status_Freshman',
       'class_status_Junior', 'class_status_Senior', 'class_status_Sophomore', 'marital_status_Yes',
        'have_depression_Yes', 'have_anxiety_Yes', 'have_panicattacks_Yes', 'seeked_treatment_Yes']]
data_dummied.columns
```

```
Index(['age', 'gpa', 'gender_Agender', 'gender_Bigender', 'gender_Female',
       'gender_Genderfluid', 'gender_Genderqueer', 'gender_Male',
       'gender_Non-binary', 'gender_Polygender', 'major_Arts',
       'major_Business', 'major_Communications', 'major_Engineering',
       'major_History', 'major_Information Systems', 'major_Language',
       'major_Life Sciences', 'major_Social Sciences', 'class_status_Freshman',
       'class_status_Junior', 'class_status_Senior', 'class_status_Sophomore',
       'marital_status_Yes', 'have_depression_Yes', 'have_anxiety_Yes',
       'have_panicattacks_Yes', 'seeked_treatment_Yes'],
      dtype='object')
```

Feature Engineering

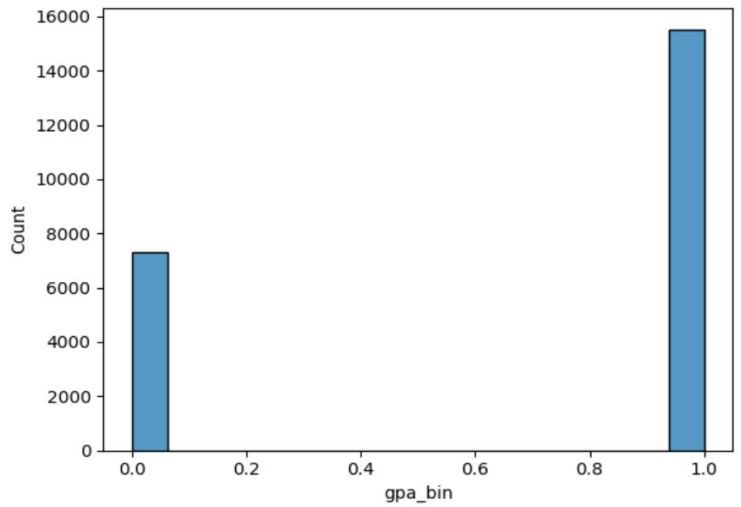
Making a binary transformation between people who have a gpa below 2.5 and people who have a gpa above 2.5:

The majority people have a gpa greater than 2.5

```
from sklearn.preprocessing import Binarizer

data2 = data_dummied.copy()

transformer = Binarizer(threshold = 2.5)
transformer.fit(data2[['gpa']])
transformer.transform(data2[['gpa']])

data2['gpa_bin'] = transformer.transform(data2[['gpa']])
data2
```

```
sns.histplot(x=data2['gpa_bin'])
```

```
<Axes: xlabel='gpa_bin', ylabel='Count'>
```



Machine Learning

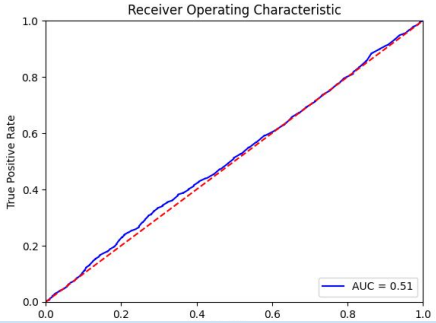
Machine Learning Model (Random Forest)

```
[15]: #Here, we are splitting our data into a training and testing dataset where we are testing if a person has anxeity or not
from sklearn.model_selection import train_test_split

X = df_dummies.drop(['have_anxiety_Yes'],axis=1)
y = df_dummies['have_anxiety_Yes']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[16]: #Here, we are actually creating a random forest and we are giving the learning model our training dataset
#After using the training dataset, the model will try to predict the outcome of a new dataset "X_test", based on what it learned...
#...from its training data.
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier()
forest.fit(X_train, y_train)
predictions = forest.predict(X_test)
```

Test Train Split on dataset, our target variable, y, is if the person has anxiety. Then we, do random forest classifier and graph results.

```
[21]: #Plotting the ROC using the data from the random tree classifier.
#Looking at the graph, the model isn't the best at predicting if a person has anxiety as the AUC, the blue line, nearly matches the ROC.
import sklearn.metrics as metrics

probs = forest.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



The Random Forest Model isn't the best as the blue and red lines nearly match.

Machine Learning Model (Random Forest)

```
•[66]: #This gives us a summary of what occurred with our random tree classifier and tells us that the learning model isn't the best at predicting true
from sklearn.metrics import classification_report,confusion_matrix

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

```
              precision    recall  f1-score   support

       False       0.67      0.78      0.72      3048
        True       0.34      0.23      0.28      1515

    accuracy                           0.60      4563
   macro avg       0.51      0.51      0.50      4563
weighted avg       0.56      0.60      0.57      4563

[[2376  672]
 [1163  352]]
```

Here is the classification report and we can see that the model only guessed correctly 57% of the time.

Machine Learning Model (KNeighborsClassifier)

[33]:
```
#Hyperparameter tuning for KNeighborsClassifer by using Grid Search
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

K_grid_param = {
    'n_neighbors': [2, 4, 6, 8, 9, 10],
    'weights': ['uniform', 'distance']
}

knn_model = KNeighborsClassifier()

grid_search = GridSearchCV(estimator=knn_model, param_grid=K_grid_param, cv=5)

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_

#printing best parameters
print("Best Hyperparameters:", best_params)
```

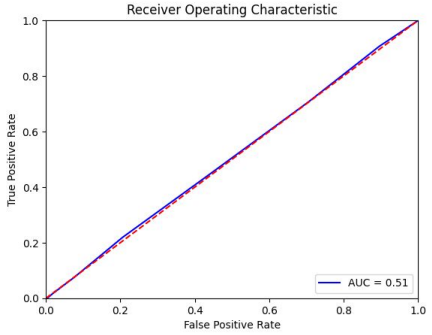
```
Best Hyperparameters: {'n_neighbors': 10, 'weights': 'uniform'}
```

[34]:
```
from sklearn.neighbors import KNeighborsClassifier

#knn model is being used, looks at nearest variables and we are using the best parameters we found.
knn_model = KNeighborsClassifier(n_neighbors = 10, weights = 'uniform')
knn_model.fit(X_train, y_train)

y_pred = knn_model.predict(X_test)
```

[70]:
```
from sklearn.metrics import classification_report,confusion_matrix

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

```
              precision    recall  f1-score   support

       False       0.67      0.92      0.78      3048
        True       0.33      0.07      0.12      1515

    accuracy                           0.64      4563
   macro avg       0.50      0.50      0.45      4563
weighted avg       0.56      0.64      0.56      4563

[[2818  230]
 [1402  113]]
```

[36]:
```
import sklearn.metrics as metrics

probs = knn_model.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Here, I did hyperparameter tuning using grid search. Then I performed a knn model using the best parameters.

The model only guessed correctly 56% of the time.

Machine Learning Model (XGBClassifier)

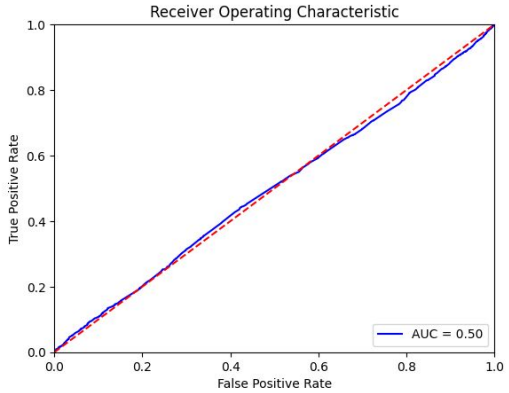
[37]:
```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

X_grid_param = {
    'learning_rate': [0.05, 0.1, 0.3],
    'n_estimators': [10, 50, 100, 200]
}

xgb_model = XGBClassifier()

#here we do a gridsearch to find the best parameters for the XGBRegressor
grid_search = GridSearchCV(estimator=xgb_model, param_grid=X_grid_param, cv=5)

#Train model w/ GridSearch CV
grid_search.fit(X_train, y_train)

#access the best model + params
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

#Print best parameters
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'learning_rate': 0.05, 'n_estimators': 10}
```

[38]:
```
XGB_model = XGBClassifier(learning_rate = 0.05, n_estimators = 10)
XGB_model.fit(X_train, y_train)
```

[38]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=10, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

[39]:
```
import sklearn.metrics as metrics

probs = XGB_model.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[75]:
```
#XGBoost Classification Report
from sklearn.metrics import classification_report,confusion_matrix

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

```
              precision    recall  f1-score   support

       False       0.67      1.00      0.80      3048
        True       0.00      0.00      0.00      1515

    accuracy                           0.67      4563
   macro avg       0.33      0.50      0.40      4563
weighted avg       0.45      0.67      0.54      4563

[[3048    0]
```

Preformed hyperparameter tuning. Then did XGBClassifier, and graphed results. This model didn't do the best either as both lines almost match up. Overall, the dataset might be the cause of the performance of the models as there isn't much positive results.

Thank You

Thank you for listening, and we hope we did well!