COSC 5610
Final Project: Panic Disorder Diagnosis
Submitted by: Mohammed Junaid (00627726)

Objective: In this project, you will work with a Panic Disorder Detection dataset that contains patient demographic information, symptoms, medical history, lifestyle factors, and mental health assessment results. The goal is to analyze the dataset, preprocess the data, resolve class imbalance, extract meaningful features, and evaluate machine learning models for predicting panic disorder.

Step 1: Data Exploration & Understanding

1. **Dataset shape and values:** Given data set has 120000 entries with a total of 17 columns. It has three numerical columns (include target column), three ordinal columns and remaining are nominal columns.

```
panic.info()
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120000 entries, 0 to 119999
Data columns (total 17 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Participant ID           120000 non-null  int64
 1   Age                      120000 non-null  int64
 2   Gender                   120000 non-null  object
 3   Family History           120000 non-null  object
 4   Personal History         120000 non-null  object
 5   Current Stressors        120000 non-null  object
 6   Symptoms                 120000 non-null  object
 7   Severity                 120000 non-null  object
 8   Impact on Life           120000 non-null  object
 9   Demographics             120000 non-null  object
 10  Medical History          89776 non-null   object
 11  Psychiatric History      90104 non-null   object
 12  Substance Use            79922 non-null   object
 13  Coping Mechanisms        120000 non-null  object
 14  Social Support           120000 non-null  object
 15  Lifestyle Factors        120000 non-null  object
 16  Panic Disorder Diagnosis 120000 non-null  int64
dtypes: int64(3), object(14)
memory usage: 15.6+ MB
```

2. **Missing values**: The Medical History column has 30224 empty values, Psychiatric History column has 29896 empty values, Substance Use column has 40078 empty values

```
from collections import defaultdict

target_column = 'Panic Disorder Diagnosis'

panic_null = panic.isnull()

null_counts = panic.isnull().sum()

#defining a dict to store empty values
panic_null_dict = defaultdict(list)

for index in range(len(panic_null.columns)):

    #Skipping the target column
    if(index == target_column):
        continue

    null_index = panic_null.loc[panic_null[panic_null.columns[index]]==True]

    if(len(null_index.index) == 0):
        continue
    else:
        #Adding these empty value indexes to a dictionary
        panic_null_dict[panic_null.columns[index]] = null_index.index
        print(f'The {null_index.columns[index]} column has {len(null_index.index)} empty values\n')

✓ 0.0s
The Medical History column has 30224 empty values

The Psychiatric History column has 29896 empty values

The Substance Use column has 40078 empty values
```

COSC 5610

Final Project: Panic Disorder Diagnosis

Submitted by: Mohammed Junaid (00627726)

3. **Duplicate values**: The dataset has total number of 19601 duplicate values**.**

```
panic_dupes = panic.duplicated()

target_column = 'Panic Disorder Diagnosis'
dupe_mask = panic.duplicated(subset=panic.columns.difference([target_column]))
num_dupes = dupe_mask.sum()

if(dupe_mask.any()):
    print(f'Yes, the dataset has total number of {num_dupes} duplicate values')
else:
    print('No dupes found')
```

✓ 0.1s

```
Yes, the dataset has total number of 19601 duplicate values
```

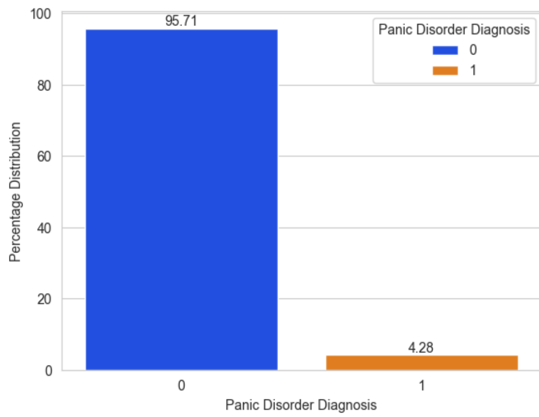
4. **Initial analysis**:

COSC 5610

Final Project: Panic Disorder Diagnosis

Submitted by: Mohammed Junaid (00627726)

5. **Class distribution:** There is an imbalance ‘0’ is majority class ‘1’ is the minority



Step 2: Data Cleaning & Transformation

1. **Reason for imputation by mean**: For the Medical History column 25.19% entries, Psychiatric History column 24.91% entries, Substance Use column, 33.4% entries are missing/non-null. This is why it would be a bad option to drop the rows, rather imputation by mean (most frequent occurrence) is the better option…

```
for col in missing_vars:
    mode = panic[col].mode()[0]
    panic.fillna({col: mode}, inplace= True )

panic.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120000 entries, 0 to 119999
Data columns (total 17 columns):
 #   Column                  Non-Null Count   Dtype 
---  ------                  --------------   ----- 
 0   Participant ID          120000 non-null  int64 
 1   Age                     120000 non-null  int64 
 2   Gender                  120000 non-null  object
 3   Family History          120000 non-null  object
 4   Personal History        120000 non-null  object
 5   Current Stressors       120000 non-null  object
 6   Symptoms                120000 non-null  object
 7   Severity                120000 non-null  object
 8   Impact on Life          120000 non-null  object
 9   Demographics            120000 non-null  object
 10  Medical History         120000 non-null  object
 11  Psychiatric History     120000 non-null  object
 12  Substance Use           120000 non-null  object
 13  Coping Mechanisms       120000 non-null  object
 14  Social Support          120000 non-null  object
 15  Lifestyle Factors       120000 non-null  object
 16  Panic Disorder Diagnosis  120000 non-null  int64
```

2. **Reasons for dropping duplicates:** No significant change in no significant change in the distribution of the class variables was noticed after dropping the duplicates, so it is a viable option.

COSC 5610

Final Project: Panic Disorder Diagnosis

Submitted by: Mohammed Junaid (00627726)

Before removing dupes:

	count	Percentage Distribution
Panic Disorder Diagnosis		
0	114858	95.71
1	5142	4.28

After removing dupes:

	count	Percentage Distribution
Panic Disorder Diagnosis		
0	96087	95.71
1	4312	4.29

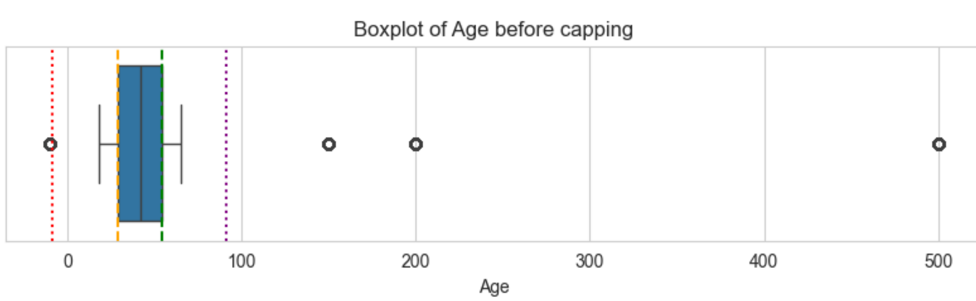
3. **Standardizing categorical entries:**

```
cat_cols = panic_cleaned.select_dtypes(include= object).columns
cat_cols

for col in cat_cols:

    #Converting to lower case
    panic_cleaned.loc[:, col] = (panic_cleaned[col].astype(str).str.lower().str.strip().str.replace(r'\s+', ' ', regex=True))

    print(f'The unique values for {col} columns are: \n')
    print(panic_cleaned[col].unique())
    print('*' + '-'*50 + '*' +'\n')
```

Symptoms and medical history columns had similar redundancies, so we remove them.

```
#Cleaning up symptoms and medical history column

#df.method({col: value}, inplace=True)
panic_cleaned[cat_cols[4]].replace(to_replace=['dizzines','panic attacks'], value=['dizziness', 'panic attack'], inplace=True)
panic_cleaned[cat_cols[8]].replace(to_replace=['diabetic'], value=['diabetes'], inplace=True)

for col in (cat_cols):
    print(f'The unique values for {col} columns are: \n')
    print(panic_cleaned[col].unique())
    print('*' + '-'*50 + '*' +'\n')
✓ 0.0s
```

4. **Date-fields**: None were found.
5. **Handling outliers in age column**: As it is the only numerical column we are able to find outliers in it.

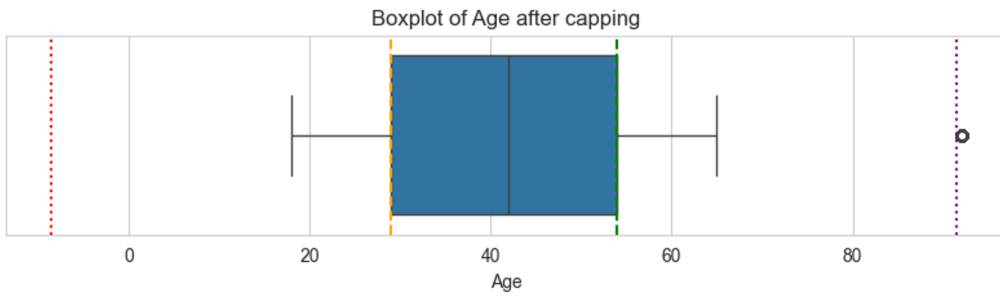


COSC 5610
Final Project: Panic Disorder Diagnosis
Submitted by: Mohammed Junaid (00627726)

Capping the age column to max and min values of the age column…

```python
#Cleaning the age column to remove all extreme outliers by using lower and upper bounds as clipping
min_age_actual = panic_cleaned['Age'][(panic_cleaned['Age'] > 0) & (panic_cleaned['Age'] <= 18) ]
print(f'The actual possible minimum age is : {min_age_actual[37]} \n')

panic_cleaned['Age'] = np.clip(panic_cleaned['Age'], a_min=min_age_actual[37], a_max=round(upper))
```
✓ 0.0s

New distribution of age column-



Step 3: Handling class imbalances

1. **Imbalanced class variables**: was highly imbalanced (95.7% negative, 4.3% positive)

```
This is the class distribution before SMOTE:
```

	count	Percentage Distribution
Panic Disorder Diagnosis		
0	96087	95.71
1	4312	4.29

2. **SMOTE for balancing: I**ncreased minority class samples to achieve a 1:1 ratio

```python
smote = SMOTE(random_state=45)

y= panic_cleaned['Panic Disorder Diagnosis']

resampled_X, resampled_y = smote.fit_resample(X_labelled, y)

panic_smote = pd.DataFrame(data= resampled_X)
panic_smote['Panic Disorder Diagnosis']= resampled_y
```
✓ 0.4s

```python
classes_smote = panic_smote['Panic Disorder Diagnosis'].value_counts().sort_index()
total = classes_smote.sum()
class_indexes = classes_smote.index
percentages = []

for index in class_indexes:
    percentage = round((classes_smote[index]/total)*100, 2)
    percentages.append(percentage)

#Creating a dataframe that shows us class distribution statistics
class_distribution_smote = pd.DataFrame(data=classes_smote)
class_distribution_smote['Percentage Distribution'] = percentages
print('The class distribution after SMOTE is: \n')
class_distribution_smote
```
✓ 0.0s

```
The class distribution after SMOTE is:
```

	count	Percentage Distribution
Panic Disorder Diagnosis		
0	96087	50.0
1	96087	50.0

COSC 5610

Final Project: Panic Disorder Diagnosis

Submitted by: Mohammed Junaid (00627726)

3. **Random undersampling of majority for balancing:**

```
rusampler = RandomUnderSampler(random_state= 40)

X_rus, y_rus = rusampler.fit_resample(X_labelled, y)

panic_rus = pd.DataFrame(data= X_rus)
panic_rus['Panic Disorder Diagnosis']= y_rus
✓ 0.0s
```

```
classes_rus = panic_rus['Panic Disorder Diagnosis'].value_counts().sort_index()
total = classes_rus.sum()
class_indexes = classes_rus.index
percentages_rus = []

for index in class_indexes:
    percentage = round((classes_rus[index]/total)*100, 2)
    percentages_rus.append(percentage)

#Creating a dataframe that shows us class distribution statistics
class_distribution_rus = pd.DataFrame(data=classes_smote)
class_distribution_rus['Percentage Distribution'] = percentages_rus
print('The class distribution after random undersampling of majority class is: \n')
class_distribution_rus
✓ 0.0s
```

```
The class distribution after random undersampling of majority class is:
```

	count	Percentage Distribution
Panic Disorder Diagnosis		
0	96087	50.0
1	96087	50.0

Step 4: Feature engineering

Feature engineering for both balanced and unbalanced dataframe was performed

1. **Label Encoding**: Performed for 'Severity', 'Impact on Life','Social Support' columns as they are ordinal in nature.

```
ordinal_cols = ['Severity', 'Impact on Life','Social Support' ]
ordinal_ordered=[['mild', 'moderate', 'severe'], ['mild', 'moderate', 'significant'], ['low' ,'moderate' ,'high']]

#Initialize a encoder
encoder = oe(categories=ordinal_ordered)

#Fitting values in specified order into the ordinal value columns
panic_cleaned[ordinal_cols] = encoder.fit_transform(panic_cleaned[ordinal_cols])

✓ 0.0s
```

```
C:\Users\ther3\AppData\Local\Temp\ipykernel_33268\3981579715.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-v
  panic_cleaned[ordinal_cols] = encoder.fit_transform(panic_cleaned[ordinal_cols])
```

For unbalanced dataframe

```
#Fitting values in specified order into the ordinal value columns
panic_cleaned_unbalanced[ordinal_cols] = encoder.fit_transform(panic_cleaned_unbalanced[ordinal_cols])
✓ 0.0s
```

2. One-hot encoding: For remaining categorical variables as they don't follow an inherent ranking order and can be converted into integer values as needed

```python
from sklearn.preprocessing import OneHotEncoder
```
✓ 0.0s

```python
#Defining nominal columns
nominal_cols = list(filter(lambda column: column not in ordinal_cols ,non_int_columns))
#nominal_cols

ohe = OneHotEncoder(sparse_output=False)

#Encoding only the nominal columns
ohe_encoded_nominal_cols = ohe.fit_transform(panic_cleaned[nominal_cols])

#Creating df of ohe nominal columns
ohencoded_df = pd.DataFrame(data=ohe_encoded_nominal_cols, columns= ohe.get_feature_names_out(nominal_cols), index=panic_cleaned.index)

#Drop original columns and concate the new one hot encoded values
panic_cleaned=panic_cleaned.drop(columns=nominal_cols)
panic_cleaned = pd.concat([panic_cleaned, ohencoded_df], axis=1)
```
✓ 0.3s

3. Min-max scaling: Performed on Age column to get scaled age column as we have already clipped the values to a max and min value range. This eliminates the varying numerical values and fits them in the [0,1] range measuring how far apart they are from the median.

```python
from sklearn.preprocessing import MinMaxScaler
```
✓ 0.0s

Generate

```python
scaler = MinMaxScaler()
panic_cleaned['Age_scaled'] = scaler.fit_transform(panic_cleaned[['Age']]).round(1)
```
✓ 0.0s

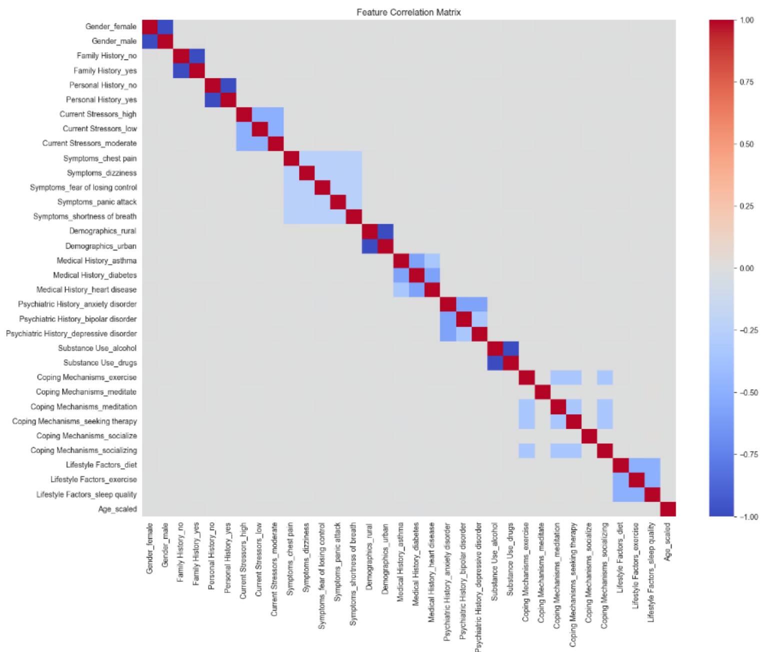
4. Dropped old columns: Dropped older columns which where transformed to new features

```python
features_to_drop = [
    'Participant ID',  'Age', 'Severity', 'Impact on Life','Social Support'
]

# Print columns before dropping for verification
print("Columns before dropping:")
print(panic_cleaned.columns.tolist())
# Drop these columns if they exist in the unbalanced dataset
panic_cleaned= panic_cleaned.drop(['Participant ID',  'Age', 'Severity', 'Impact on Life','Social Support'], axis=1)
print("Columns after dropping:")
print(panic_cleaned.columns.tolist())

# Print columns before dropping for verification
print("Columns before dropping UB:")
print(panic_cleaned_unbalanced.columns.tolist())
# Drop these columns if they exist in the unbalanced dataset
panic_cleaned_unbalanced= panic_cleaned_unbalanced.drop(['Participant ID',  'Age', 'Severity', 'Impact on Life','Social Support'], axis=1)
print("Columns after dropping UB:")
print(panic_cleaned_unbalanced.columns.tolist())
```
✓ 0.0s

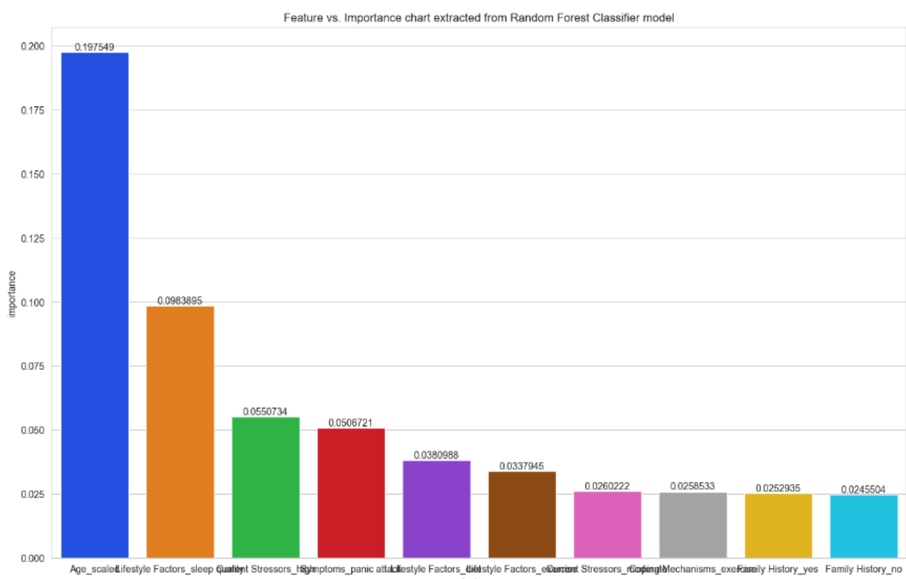
COSC 5610
Final Project: Panic Disorder Diagnosis
Submitted by: Mohammed Junaid (00627726)

Step 5: Feature importance analysis

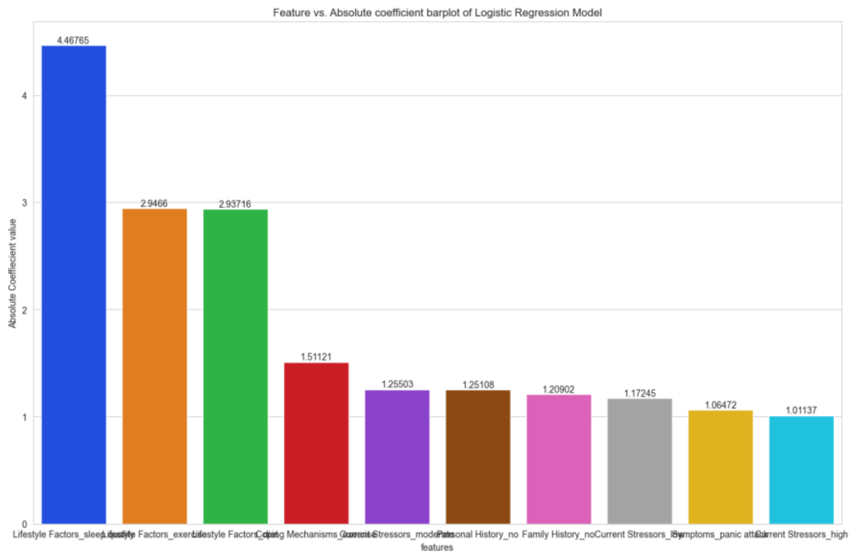
1. The only true numerical feature was Age. So performed correlation on remaining features only.



2. For Random Forest classifier, generated feature vs. importance graph for top 10 features. Age_scaled, Lifestyle Factors_sleep quality, and Current Stressors_high came out as top predictors.



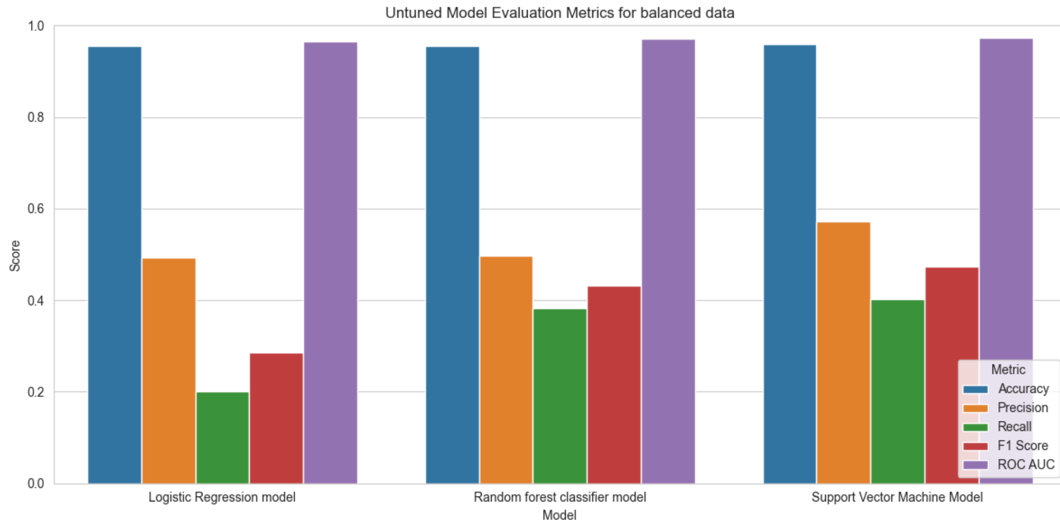
3. For log regression classifier model, coefficient analysis revealed similar results.

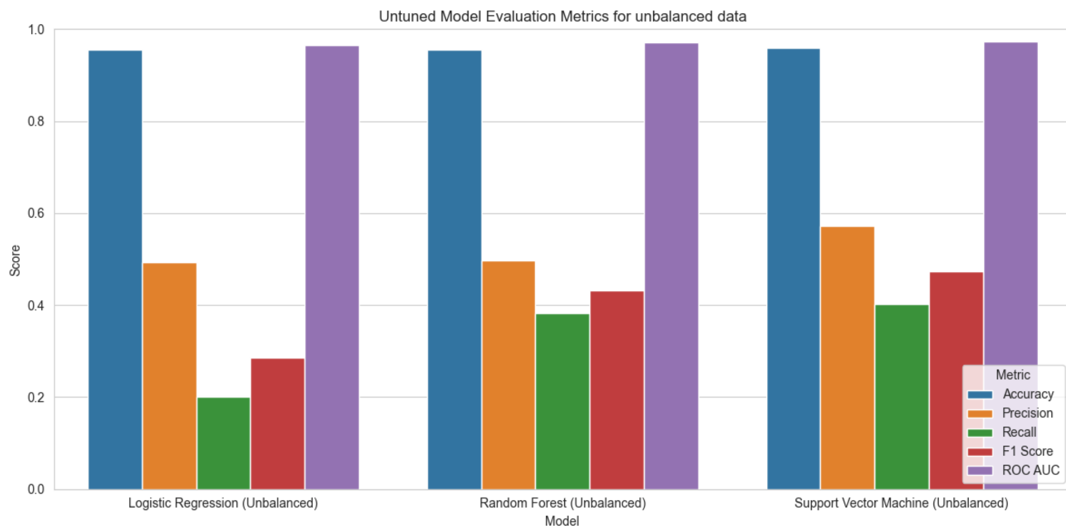


Step 7: Model training and evaluation

1. Split both balanced and unbalanced data in 70:30 test-train split
2. Trained three models: Logistic Regression, Random Forest, and Support Vector Machine (SVM).
3. Generated evaluation metrics and compared all three models on balanced and unbalanced data.

	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression model	0.954947	0.492727	0.200890	0.285413	0.965031
Random forest classifier model	0.955013	0.497110	0.382506	0.432342	0.971912
Support Vector Machine Model	0.959728	0.571429	0.403262	0.472838	0.972516
Logistic Regression (Unbalanced)	0.954947	0.492727	0.200890	0.285413	0.965031
Random Forest (Unbalanced)	0.955013	0.497110	0.382506	0.432342	0.971912
Support Vector Machine (Unbalanced)	0.959728	0.571429	0.403262	0.472838	0.972516





All models improved in recall and F1-score after class balancing and tuning, with Random Forest achieving the highest ROC AUC.
SVM required careful feature selection and scaling to avoid predicting only the majority class.

Step 8: Ethical considerations

AI models for mental health prediction can inherit and amplify biases present in the training data, such as underrepresentation of certain genders, ethnicities, or age groups. If a model is trained on data that does not reflect the diversity of the population, its predictions may be less accurate or fair for minority groups. This can result in unequal access to care, misdiagnosis, or overdiagnosis for some individuals. Ensuring fairness requires careful dataset curation, regular bias audits, and transparent reporting of model performance across subgroups.

COSC 5610
Final Project: Panic Disorder Diagnosis
Submitted by: Mohammed Junaid (00627726)

Using AI for mental health assessment raises significant ethical concerns. Patient privacy and data security must be strictly maintained, as mental health data is highly sensitive. There is also a risk of overreliance on automated decisions, which may lack the nuance and empathy of human clinicians. Inaccurate predictions could lead to stigma or inappropriate treatment. Therefore, AI tools should support, not replace, clinical judgment, and their use must be guided by ethical principles, transparency, and ongoing oversight.

Github link: https://github.com/junaid9248/COSC-5610_Data-Mining/tree/main/Final%20project