CMSC 176 MP Presentation

Acosta | Espique | Olaguera

The Dataset

- The dataset we were asked to use machine learning algorithms on is a Depression Dataset.
- There are 30 features in the dataset that may affect a person's mental health, and these variables would either make a person depressed or not.
- After those 30 features, there is a class variable called "Depressed", which has a value of 0 or 1, and it stands for not depressed or depressed respectively.

The Dataset

- Aside from that, the dataset has more than 600 vectors or rows for the different cases that respondents had.
- The objective is to determine if a person may or may not be depressed given the dataset.

Learning Algorithms to Use

- For the supervised learning algorithm, we opted to use Naïve Bayes and Random Forest.
- Reasons for using Naïve Bayes:
  - Simplicity
  - Speed and efficiency

Learning Algorithms to Use

- For the supervised learning algorithm, we opted to use Naïve Bayes and Random Forest.
- Reasons for using Random Forest:
  - Effective handling of categorical features
  - Non-linearity handling
  - Robustness to noise and outliers
  - Overfitting control

Learning Algorithms to Use

- For the unsupervised learning algorithm, we opted to use K-means clustering.
- This is a model wherein new variables are classified based on existing clusters.

Naïve Bayes Algorithm

- Naïve Bayes is a type of classification algorithm wherein probability is used to predict an object, in this case, depression is the one being predicted.
- It is called as such as the features are independent of each other, in this case, the features in the dataset are not based on each other.

Naïve Bayes Algorithm

- To be able to use this algorithm, libraries would have to be imported.
- This is to allow the machine problem to work using Python.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import numpy as np
```

✓ 0.0s Python

Naïve Bayes Algorithm

- The dataset is then loaded into the program.
- Pre-processing was done in the form of converting the columns into categorical data.

```python
# Load the depression dataset
dataset_path = 'c:/Users/Mark Lance Olaguera/Pictures/Depression Dataset.csv'
dataset = pd.read_csv(dataset_path)
```

✓ 0.0s Python

```python
# Convert all columns to categorical data type
for column in dataset.columns:
    dataset[column] = dataset[column].astype('category')
```

✓ 0.0s Python

Naïve Bayes Algorithm

- Aside from that, the dataset is checked for any missing values, in this case, there are none. Inconsistent entries are checked.

```python
# Check for missing values
if dataset.isnull().sum().sum() > 0:
    print("Warning: Missing values detected! Please handle them before proceeding.")
else:
    print("No missing values detected.")
```

✓ 0.0s Python

```
No missing values detected.
```

```python
# Check for inconsistent entries
for column in dataset.columns:
    unique_values = dataset[column].unique()
    print(f"{column} - Unique Values: {unique_values}")
```

✓ 0.0s Python

Naïve Bayes Algorithm

```
AGERNG - Unique Values: ['26-30', '21-25', '16-20', '31-35', '46-50', '41-45', '56-60', '36-40', '61+', '51-55']
Categories (10, object): ['16-20', '21-25', '26-30', '31-35', ..., '46-50', '51-55', '56-60', '61+']
GENDER - Unique Values: ['Female', 'Male']
Categories (2, object): ['Female', 'Male']
EDU - Unique Values: ['Post Graduate', 'HSC', 'Graduate', 'SSC']
Categories (4, object): ['Graduate', 'HSC', 'Post Graduate', 'SSC']
PROF - Unique Values: ['Unemployed', 'Service holder (Private)', 'Student', 'Service holder (Government)', 'Other', 'Businessman']
Categories (6, object): ['Businessman', 'Other', 'Service holder (Government)', 'Service holder (Private)', 'Student', 'Unemployed']
MARSTS - Unique Values: ['Unmarried', 'Married', 'Divorced']
Categories (3, object): ['Divorced', 'Married', 'Unmarried']
RESDPL - Unique Values: ['Town', 'City', 'Village']
Categories (3, object): ['City', 'Town', 'Village']
LIVWTH - Unique Values: ['With Family', 'Without Family']
Categories (2, object): ['With Family', 'Without Family']
ENVSAT - Unique Values: ['Yes', 'No']
Categories (2, object): ['No', 'Yes']
POSSAT - Unique Values: ['Yes', 'No']
Categories (2, object): ['No', 'Yes']
FINSTR - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
DEBT - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
PHYEX - Unique Values: ['Sometimes', 'Never', 'Regularly']
Categories (3, object): ['Never', 'Regularly', 'Sometimes']
SMOKE - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
DRINK - Unique Values: ['Yes', 'No']
Categories (2, object): ['No', 'Yes']
ILLNESS - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
```

Naïve Bayes Algorithm

```
PREMED - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
EATDIS - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
AVGSLP - Unique Values: ['More than 8 hours', '6 hours', '8 hours', '7 hours', '5 hours', 'Below 5 hours']
Categories (6, object): ['5 hours', '6 hours', '7 hours', '8 hours', 'Below 5 hours', 'More than 8 hours']
INSOM - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
TSSN - Unique Values: ['2-4 hours a day', '5-7 hours a day', '8-10 hours a day', 'More than 10 hours a day', 'Less than 2 hours']
Categories (5, object): ['2-4 hours a day', '5-7 hours a day', '8-10 hours a day', 'Less than 2 hours', 'More than 10 hours a day']
WRKPRE - Unique Values: ['No Pressure', 'Moderate', 'Mild', 'Severe']
Categories (4, object): ['Mild', 'Moderate', 'No Pressure', 'Severe']
ANXI - Unique Values: ['Yes', 'No']
Categories (2, object): ['No', 'Yes']
DEPRI - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
ABUSED - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
CHEAT - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
THREAT - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
SUICIDE - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
INFER - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
CONFLICT - Unique Values: ['Yes', 'No']
Categories (2, object): ['No', 'Yes']
LOST - Unique Values: ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
DEPRESSED - Unique Values: [0, 1]
Categories (2, int64): [0, 1]
```

Naïve Bayes Algorithm

- The features and target variable are separated.
- The categorical columns are converted into numerical data.
- The data is split into training and testing types.

Naïve Bayes Algorithm

```python
# Separate features and target variable
X = dataset.drop(columns=['DEPRESSED'])  # Assuming 'DEPRESSED' is the target column
y = dataset['DEPRESSED']
```

✓ 0.0s Python

```python
# Convert categorical columns to numerical using category codes
X = X.apply(lambda col: col.cat.codes)
y = y.cat.codes
```

✓ 0.0s Python

```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

✓ 0.0s Python

Naïve Bayes Algorithm

- The Naïve Bayes algorithm is then initialized and then trained to predict the test set.
- The model is then evaluated based on the accuracy, precision, recall, and f1 score.

Naïve Bayes Algorithm

```python
# Initialize the Categorical Naive Bayes classifier
nb_classifier = CategoricalNB()
```

✓ 0.0s Python

```python
# Train the classifier
nb_classifier.fit(X_train, y_train)
```

✓ 0.0s Python

CategoricalNB

```
CategoricalNB()
```

```python
# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)
```

✓ 0.0s Python

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

✓ 0.0s Python

Naïve Bayes Algorithm

- The results of the evaluation are then displayed.
- The algorithm has an accuracy of 88.43%, precision of 0.97, recall of 0.84, and f1 score of 0.90.
- A confusion matrix is also made as well to show the predicted and actual values obtained.

Naïve Bayes Algorithm

```python
# Display results
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

✓ 0.0s Python

```
Accuracy: 88.43%
Precision: 0.97
Recall: 0.84
F1-Score: 0.90
Confusion Matrix:
[[42  2]
 [12 65]]
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.95      0.86        44
           1       0.97      0.84      0.90        77

    accuracy                           0.88       121
   macro avg       0.87      0.90      0.88       121
weighted avg       0.90      0.88      0.89       121
```

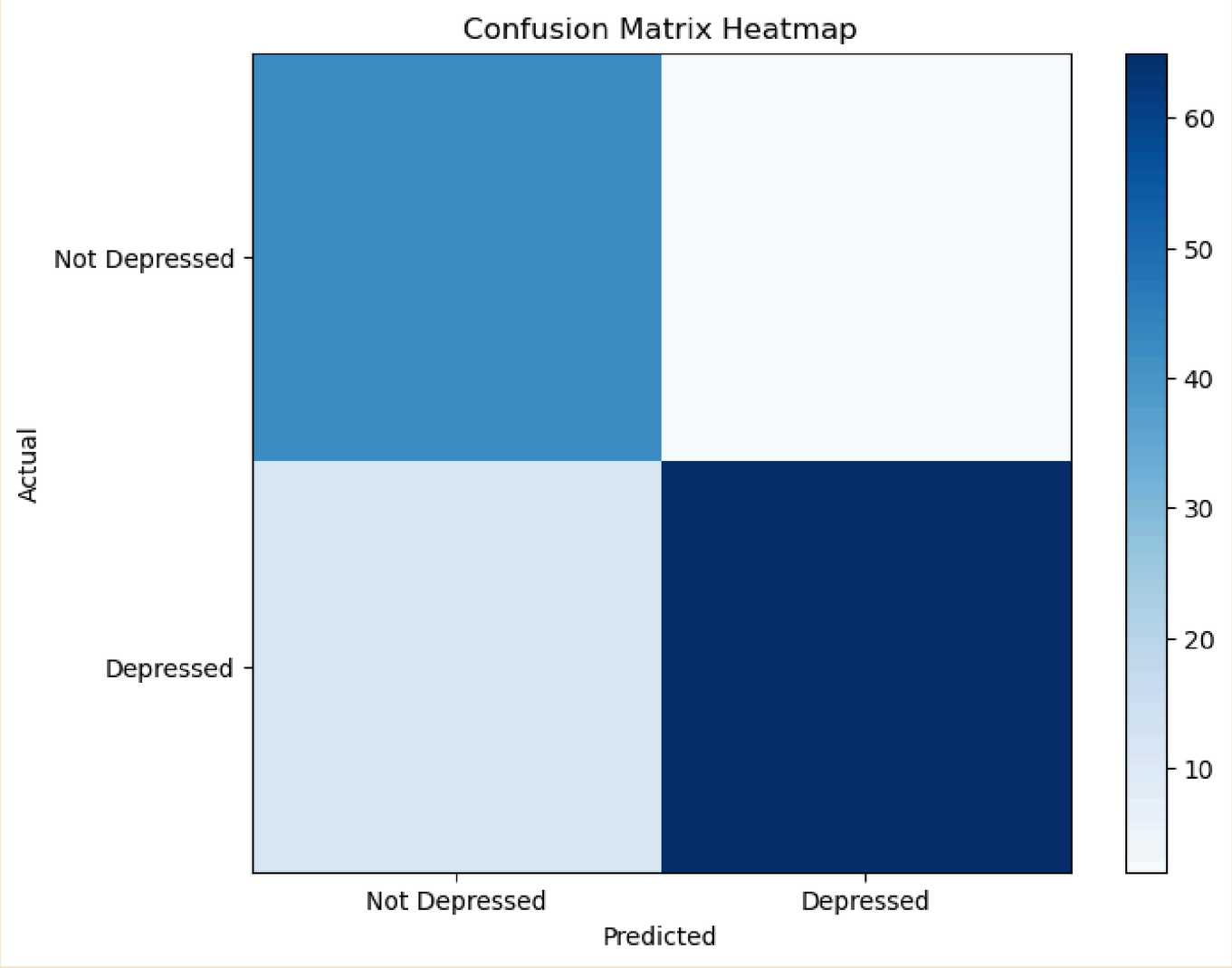
Naïve Bayes Algorithm

- A heat map is also made by using the confusion matrix that was generated earlier.

```python
# Visualize confusion matrix as a heatmap
plt.figure(figsize=(8,6))
plt.imshow(conf_matrix, interpolation='nearest', cmap='Blues')
plt.title('Confusion Matrix Heatmap')
plt.colorbar()
plt.xticks([0, 1], ['Not Depressed', 'Depressed'])
plt.yticks([0, 1], ['Not Depressed', 'Depressed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Python

Naïve Bayes Algorithm



Random Forest Algorithm

- Random Forest is a type of classification algorithm that uses multiple decision trees to predict an outcome. In this case, it is used to predict whether an individual is experiencing depression based on various features in the dataset.
- Multiple decision trees are created during training and their results are aggregated.

Random Forest Algorithm

Importing of Libraries

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.tree import plot_tree
```

Random Forest Algorithm

Definition of Columns for Encoding

```
# Define columns for encoding
binary_columns = ['ENVSAT', 'POSSAT', 'FINSTR', 'DEBT', 'SMOKE', 'DRINK', 'ILLNESS', 'PREMED',
                  'EATDIS', 'INSOM', 'ANXI', 'DEPRI', 'ABUSED', 'CHEAT', 'THREAT', 'SUICIDE',
                  'INFER', 'CONFLICT', 'LOST']
ordinal_columns = ['AVGSLP', 'TSSN', 'WRKPRE']
categorical_columns = ['AGERNG', 'GENDER', 'EDU', 'PROF', 'MARSTS', 'RESDPL', 'LIVWTH']
```

Random Forest Algorithm

Column Data Conversion

```
# Convert binary columns to numerical values
for col in binary_columns:
    dataset[col] = dataset[col].map({'Yes': 1, 'No': 0})

# Apply ordinal encoding for ordinal columns
ordinal_mappings = {
    'AVGSLP': {"Below 5 hours": 1, "5 hours": 2, "6 hours": 3, "7 hours": 4, "8 hours": 5, "More than 8 hours": 6},
    'TSSN': {"Less than 2 hours": 1, "2-4 hours a day": 2, "5-7 hours a day": 3, "8-10 hours a day": 4, "More than 10 hours": 5},
    'WRKPRE': {"No Pressure": 1, "Mild": 2, "Moderate": 3, "Severe": 4},
    'PHYEX': {"Never": 1, "Sometimes": 2, "Regularly": 3}
}

for col, mapping in ordinal_mappings.items():
    dataset[col] = dataset[col].map(mapping)

# Apply one-hot encoding for non-ordinal categorical columns
dataset = pd.get_dummies(dataset, columns=categorical_columns, drop_first=True)
```

Random Forest Algorithm

Data Splitting and Model Training

```
# Separate features and target variable
X = dataset.drop(columns=['DEPRESSED'])  # Assuming 'DEPRESSED' is the target column
y = dataset['DEPRESSED'].cat.codes

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
```

Random Forest Algorithm

Model Evaluation

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Display results
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

Random Forest Algorithm

- Model Performance:
  - Accuracy: 86.78%
  - Precision: 0.87
  - Recall: 0.94
  - F1 score: 0.90
  - Confusion Matrix

```
[[33,11]
 [5, 72]]
```

Random Forest Algorithm

- Model Performance:
  - Classification Report

Classification Report:

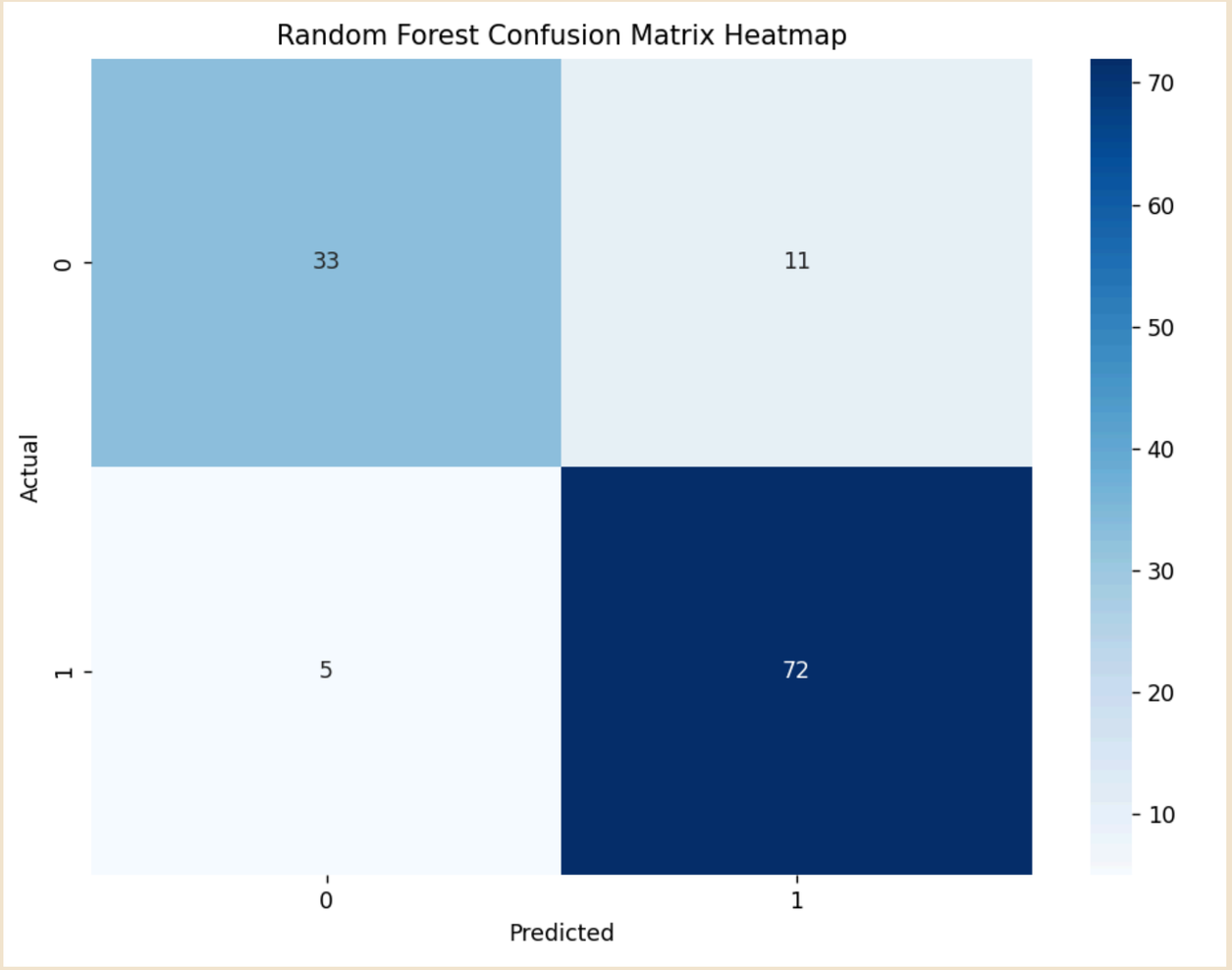
	precision	recall	f1-score	support
0	0.87	0.75	0.80	44
1	0.87	0.94	0.90	77
accuracy			0.87	121
macro avg	0.87	0.84	0.85	121
weighted avg	0.87	0.87	0.87	121

Random Forest Algorithm

Heatmap of Confusion Matrix

```
# Visualize confusion matrix as a heatmap
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')
plt.title('Random Forest Confusion Matrix Heatmap')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()
```

Random Forest Algorithm

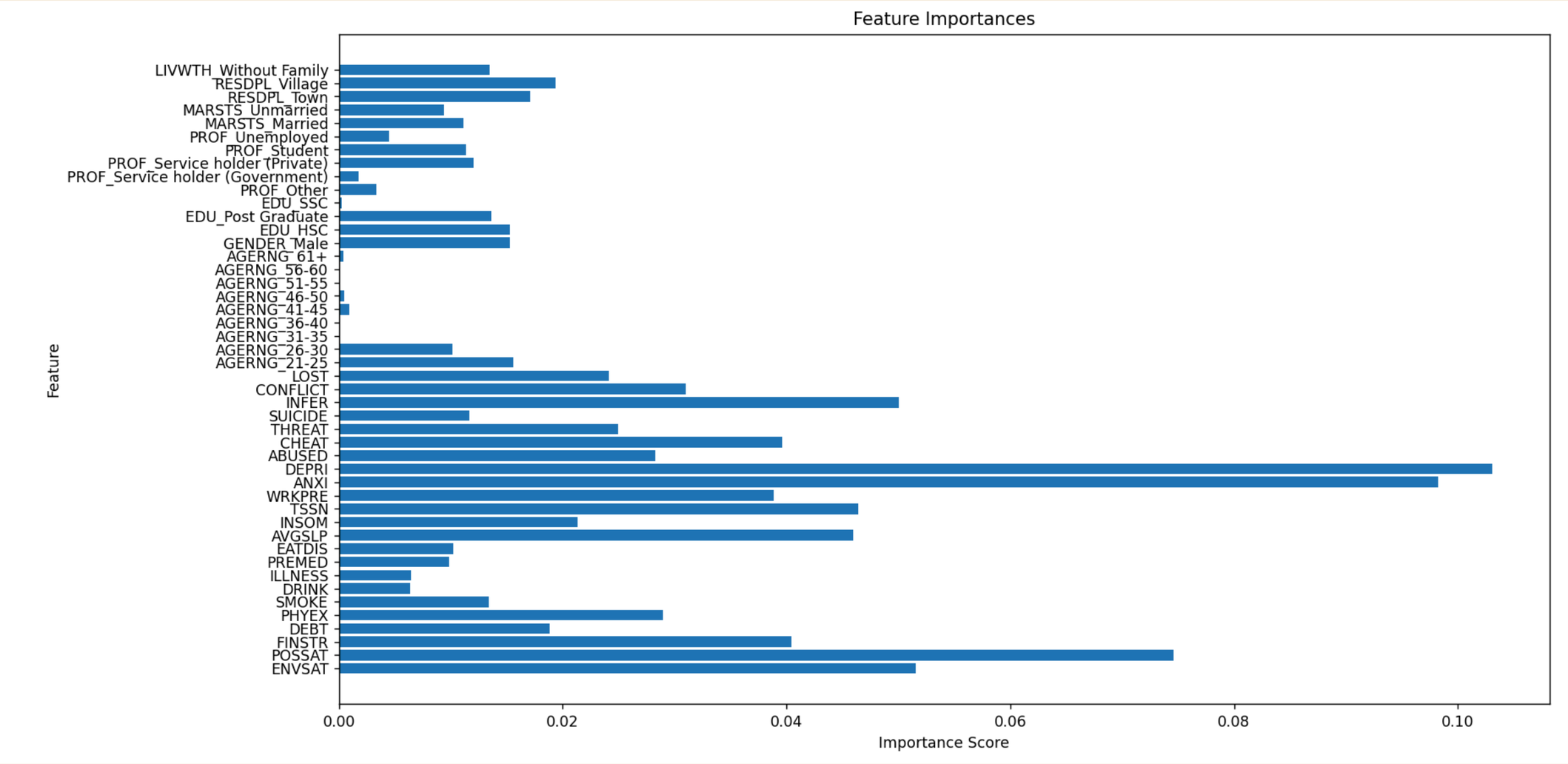


Random Forest Algorithm

Feature Importance Plot

```
# Feature Importance Plot
feature_importances = rf_classifier.feature_importances_
features = X.columns
plt.figure(figsize=(12,6))
plt.barh(features, feature_importances)
plt.title('Feature Importances')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

Random Forest Algorithm

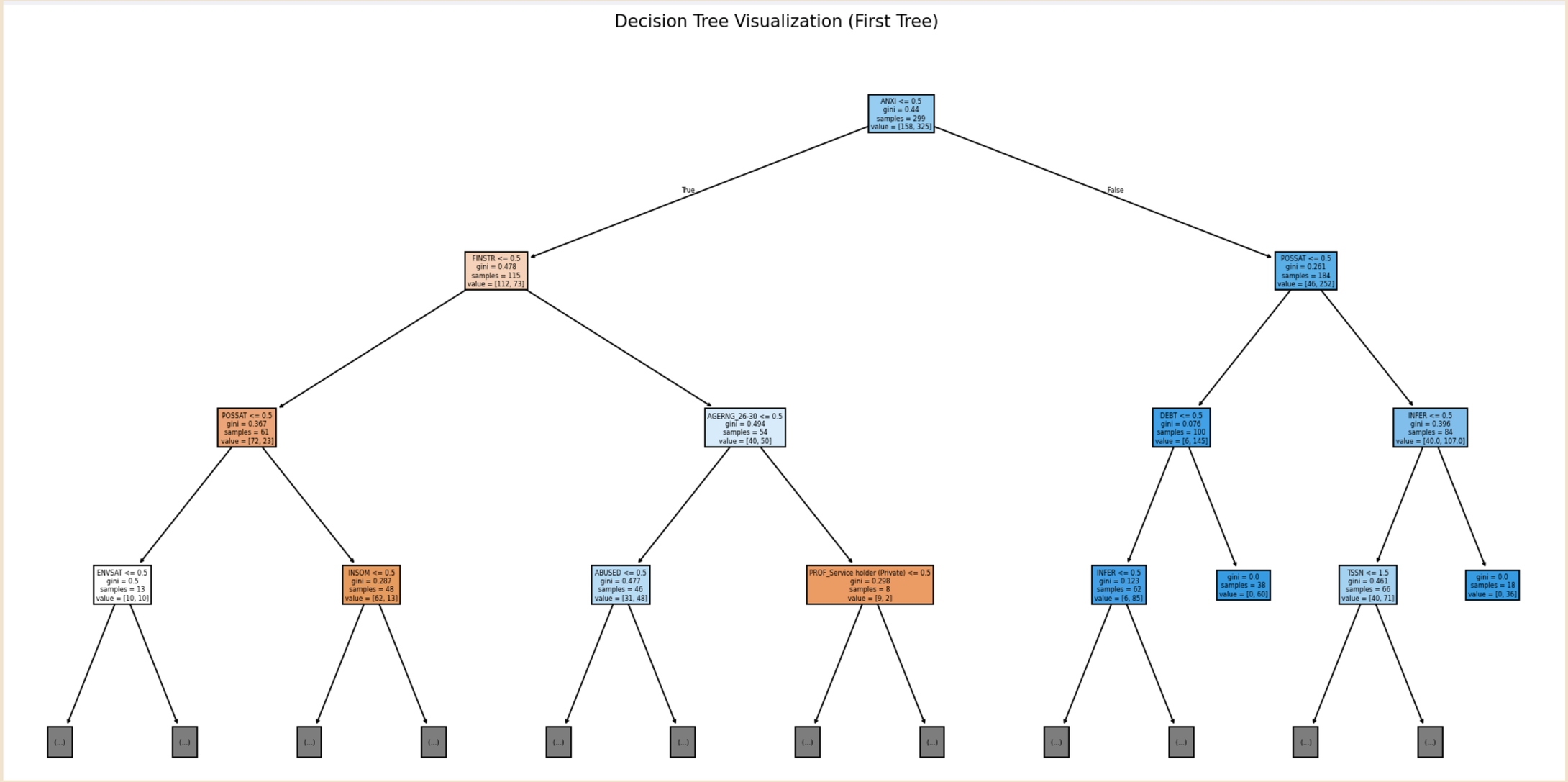


Random Forest Algorithm

Decision Tree Visualization

```
# Decision Tree Visualization (first tree in the forest)
plt.figure(figsize=(20,10))
plot_tree(rf_classifier.estimators_[0], filled=True, feature_names=X.columns, max_depth=3)
plt.title("Decision Tree Visualization (First Tree)")
plt.tight_layout()
plt.show()
```

Random Forest Algorithm



K-means Clustering

- K-means clustering is another classification algorithm in which objects are predicted based on n observations that are grouped into k clusters.
- Unlike Naïve Bayes and Random Forest, this falls under unsupervised learning thus the data is unlabeled.

K-means Clustering

- The following libraries were imported for k-means:

```python
# Import libraries
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
```

K-means Clustering

- The dataset is loaded, showing the shape of the table.
- The dataset has 604 rows and 31 columns, and they will be preprocessed for later.

K-means Clustering

```python
# Load the dataset
file_path = './Depression Dataset.csv'  # Replace with actual path
df = pd.read_csv(file_path)
print(f'Dataset Shape: {df.shape}')
df.head()
```

Python

```
Dataset Shape: (604, 31)
```

	AGERNG	GENDER	EDU	PROF	MARSTS	RESDPL	LIVWTH	ENVSAT	POSSAT	FINSTR	...	ANXI	DEPRI	ABUSED	CHEAT	THREAT	SUICIDE
0	26-30	Female	Post Graduate	Unemployed	Unmarried	Town	With Family	Yes	Yes	No	...	Yes	No	No	No	No	No
1	26-30	Male	Post Graduate	Service holder (Private)	Unmarried	City	With Family	Yes	No	Yes	...	Yes	Yes	Yes	No	No	No
2	21-25	Male	HSC	Student	Unmarried	City	With Family	Yes	Yes	No	...	Yes	Yes	No	No	No	No
3	16-20	Male	HSC	Student	Unmarried	City	With Family	No	Yes	No	...	Yes	Yes	No	Yes	No	No
4	21-25	Male	Graduate	Student	Unmarried	Town	With Family	No	Yes	Yes	...	Yes	Yes	No	No	No	No

5 rows × 31 columns

K-means Clustering

- The dataset is preprocessed by encoding the columns, scaling said columns, and determining the optimal number of clusters for the algorithm.

K-means Clustering

```python
# Preprocessing: Encode categorical columns and scale features
X = df.drop(columns=['DEPRESSED'])  # Exclude target column
for column in X.columns:
    if X[column].dtype == 'object':
        X[column] = LabelEncoder().fit_transform(X[column])
```

✓ 0.0s Python

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

✓ 0.0s Python

```python
# Determine optimal number of clusters using the elbow method
inertias = []
K = range(2, 11)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)
```

✓ 0.2s Python

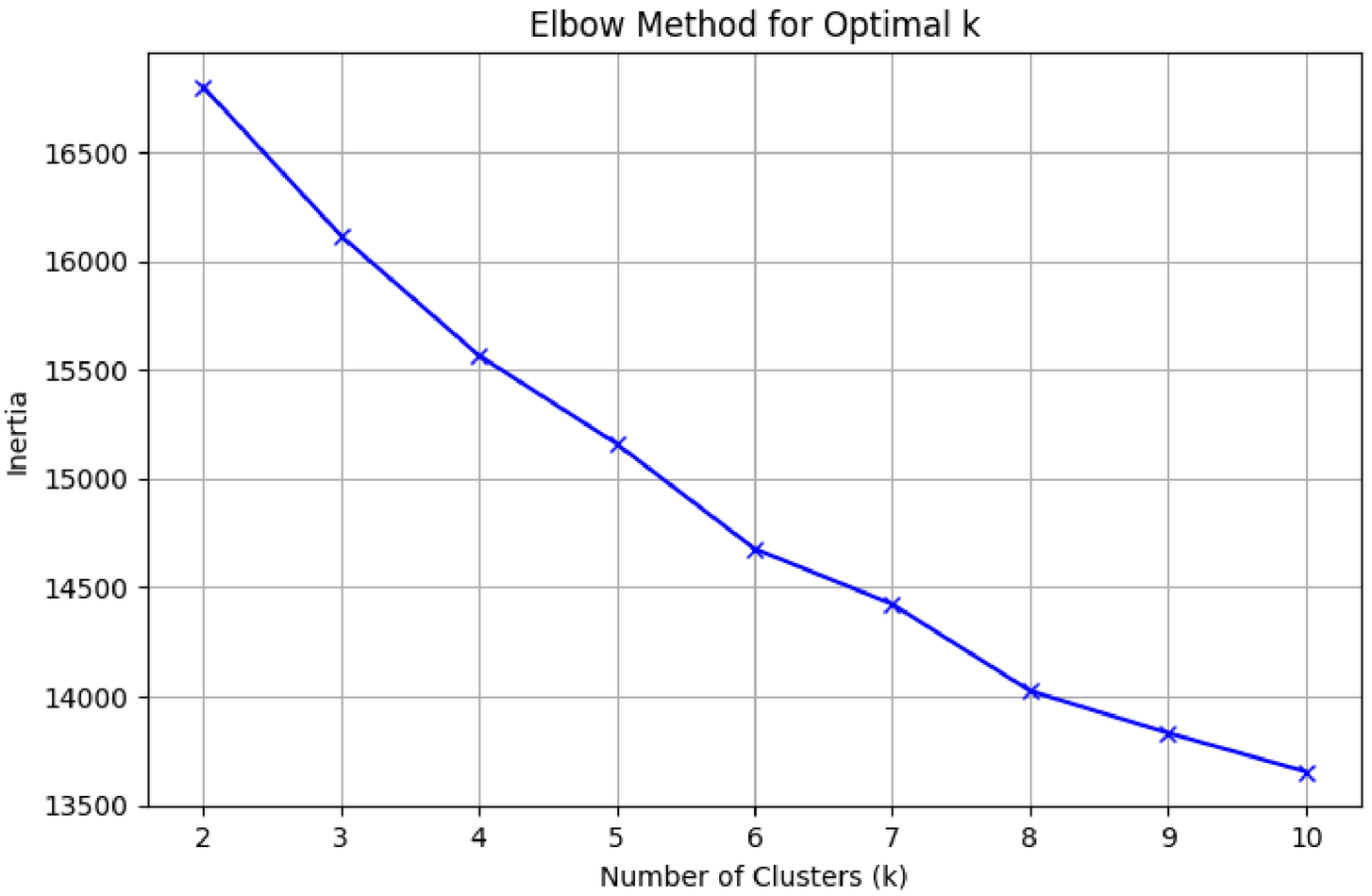
K-means Clustering

- The elbow curve is plotted using this cell. The elbow is barely visible as the graph is almost a straight line.

```python
# Plot elbow curve
plt.figure(figsize=(8, 5))
plt.plot(K, inertias, 'bx-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.grid()
plt.show()
```

✓ 0.1s Python

K-means Clustering



K-means Clustering

- The kneelocator is used to determine the optimal k, in this case, it is 8. The k-means is fitted with the optimal k.

```python
# Detect optimal k using KneeLocator
knee = KneeLocator(K, inertias, curve='convex', direction='decreasing')
optimal_k = knee.knee
print(f"Optimal k detected: {optimal_k}")
```

✓ 0.0s Python

```
Optimal k detected: 8
```

```python
# Fit K-Means with optimal k
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
df['Cluster'] = clusters
```

✓ 0.0s Python

K-means Clustering

- The silhouette score of the dataset with 8 clusters is determined to be 0.05

```
# Silhouette Score
silhouette_avg = silhouette_score(X_scaled, clusters)
print(f"\nSilhouette Score for k={optimal_k}: {silhouette_avg:.2f}")
```

✓ 0.0s Python

```
Silhouette Score for k=8: 0.05
```

K-means Clustering

- The cluster centers are analyzed by sorting out the highest and lowest values using this cell.

```python
# Analyze cluster centers
cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=X.columns)
print("\nDistinctive features for each cluster:")
for i in range(len(cluster_centers)):
    print(f"\nCluster {i}:")
    sorted_features = cluster_centers.iloc[i].sort_values()
    print("Highest values:")
    print(sorted_features[-5:])
    print("\nLowest values:")
    print(sorted_features[:5])
```

✓ 0.0s Python

K-means Clustering

```
Distinctive features for each cluster:

Cluster 0:
Highest values:
SMOKE      0.322049
ILLNESS    0.411212
LIVWTH     0.503760
EDU        0.539012
AGERNG     0.740804
Name: 0, dtype: float64

Lowest values:
PROF      -2.003193
MARSTS    -0.897953
RESDPL    -0.656814
CHEAT     -0.317243
EATDIS    -0.306719
Name: 0, dtype: float64

Cluster 1:
Highest values:
PROF       0.227427
MARSTS     0.299939
RESDPL     0.314275
ENVSAT     0.340639
GENDER     0.536006
Name: 1, dtype: float64

Lowest values:
DEPRI     -0.580617
ANXI      -0.565547
INFER     -0.457069
EATDIS    -0.433974
ABUSED    -0.356508
Name: 1, dtype: float64

Cluster 2:
Highest values:
CHEAT      0.370726
LOST       0.384493
GENDER     0.406557
SMOKE      1.504525
DRINK      3.567530
Name: 2, dtype: float64

Lowest values:
PROF       -0.296034
ILLNESS    -0.224692
ANXI       -0.175158
PHYEX      -0.144537
ENVSAT     -0.101122
Name: 2, dtype: float64

Cluster 3:
Highest values:
PROF       0.339853
ABUSED     0.355333
ANXI       0.469271
INSOM      0.472859
EATDIS     2.167513
Name: 3, dtype: float64

Lowest values:
SMOKE      -0.323097
DRINK      -0.280306
GENDER     -0.212364
SUICIDE    -0.210259
AGERNG     -0.173096
Name: 3, dtype: float64

Cluster 4:
Highest values:
PREMED     0.128551
AVGSLP     0.216465
ENVSAT     0.284763
POSSAT     0.313146
PROF       0.421433
Name: 4, dtype: float64

Lowest values:
POSSAT    -0.698967
ENVSAT    -0.582030
GENDER    -0.385899
DRINK     -0.280306
AGERNG    -0.272433
Name: 7, dtype: float64

Lowest values:
GENDER    -1.718118
SMOKE     -0.505685
MARSTS    -0.467749
FINSTR    -0.467069
CHEAT     -0.448352
Name: 4, dtype: float64

Cluster 5:
Highest values:
TSSN       0.637395
INSOM      0.797251
PREMED     1.500798
ILLNESS    1.860355
AGERNG     5.637096
Name: 5, dtype: float64

Lowest values:
PROF      -2.686473
MARSTS    -2.502284
PHYEX     -0.504597
RESDPL    -0.311440
CHEAT     -0.286443
Name: 5, dtype: float64

Cluster 6:
Highest values:
THREAT     0.447294
ANXI       0.480017
CHEAT      0.497978
FINSTR     0.551402
DEPRI      0.720292
Name: 6, dtype: float64

Lowest values:
EATDIS     -0.438498
ENVSAT     -0.384975
SUICIDE    -0.313340
DRINK      -0.280306
POSSAT     -0.255487
Name: 6, dtype: float64

Cluster 7:
Highest values:
THREAT       0.665091
CONFLICT     0.706540
INFER        0.878576
CHEAT        0.937996
SUICIDE      3.191424
Name: 7, dtype: float64
```

K-means Clustering

- The clusters are compared with the actual labels. To visualize the contingency, a heatmap is created as well.

K-means Clustering

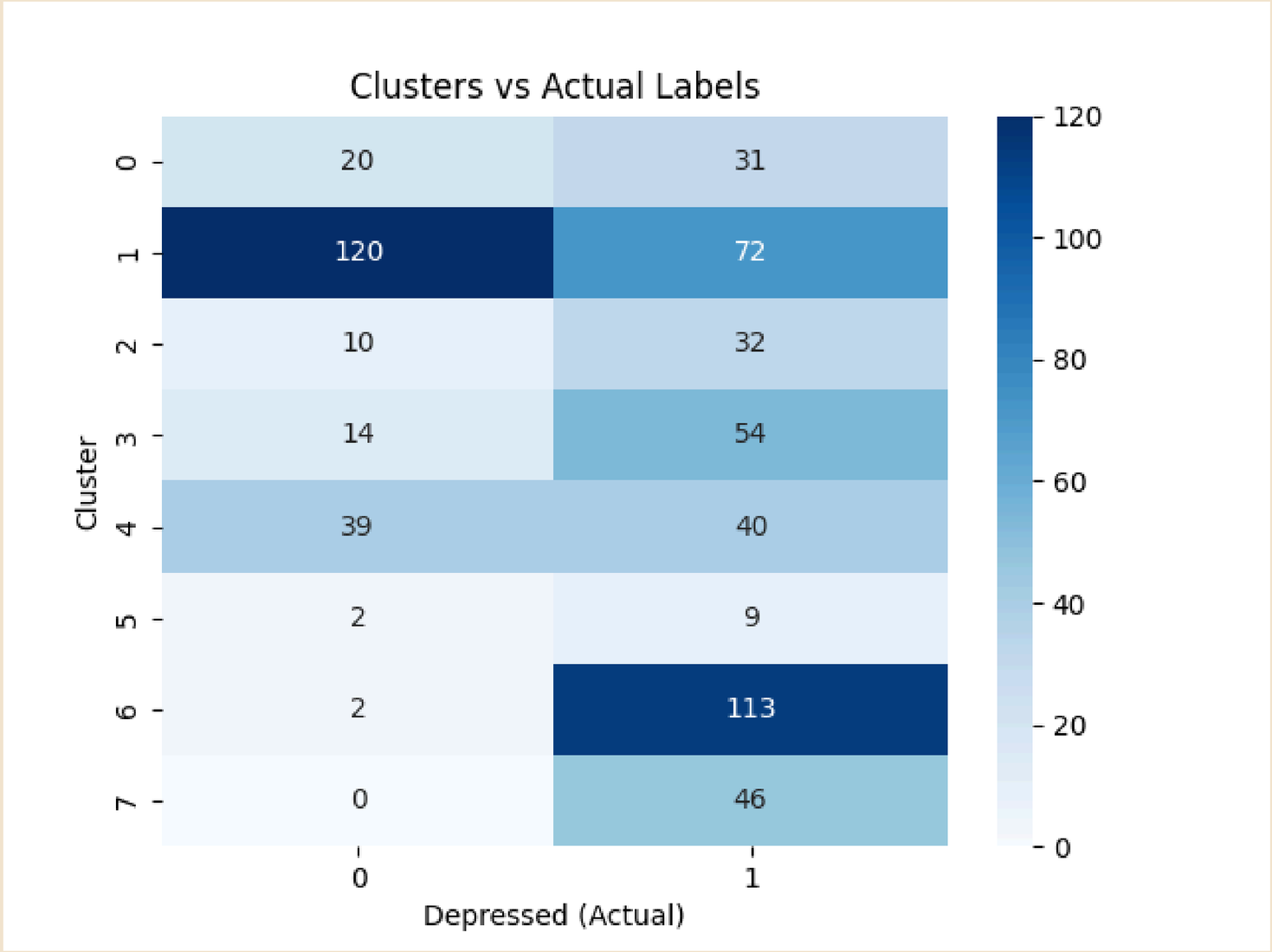
```python
# Compare clusters with actual target variable
if 'DEPRESSED' in df.columns:
    contingency_table = pd.crosstab(df['Cluster'], df['DEPRESSED'])
    print("\nContingency Table (Clusters vs Actual Labels):")
    print(contingency_table)

    # Visualize contingency table as heatmap
    sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
    plt.title('Clusters vs Actual Labels')
    plt.ylabel('Cluster')
    plt.xlabel('Depressed (Actual)')
    plt.show()
```

✓ 0.2s Python

```
Contingency Table (Clusters vs Actual Labels):
DEPRESSED    0    1
Cluster
0           20   31
1          120   72
2           10   32
3           14   54
4           39   40
5            2    9
6            2  113
7            0   46
```

K-means Clustering



K-means Clustering

- To visualize the clusters, Principal Component Analysis is used to depict the distinct clusters using this cell.

```python
# Visualize the clusters using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
plt.figure(figsize=(8, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis', edgecolor='k', alpha=0.7)
plt.colorbar(label='Cluster')
plt.title('K-Means Clustering Visualization (PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

✓ 0.1s Python

K-means Clustering