# Use case 2: Predicting mental health

Download the Depression Dataset (kaggle.com) from
https://www.kaggle.com/datasets/anthonytherrien/depression-dataset/data.

## Question 1:

Use methods of your choice (e.g. exploratory data analysis, statistical methods, visualisations etc.) to extract useful insights from the data.

## Question 2:

Use suitable models to predict if an individual will suffer mental illness (variable titled "History of Mental Illness") using all or some of the other variables present in the data.

Your answer should also contain:

Narratives for including/excluding variables of choice; Narrative supporting the model/s of choice, An assessment of model performance, Assessment of potential model biases.

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Loading the dataset
df = pd.read_csv('depression_data.csv')

# Displaying the first few rows
df.head()
```

Out[2]:

| | Name | Age | Marital Status | Education Level | Number of Children | Smoking Status | Physical Activity Level | Employment Status | Income |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Christine Barker | 31 | Married | Bachelor's Degree | 2 | Non-smoker | Active | Unemployed | 26265.67 |
| **1** | Jacqueline Lewis | 55 | Married | High School | 1 | Non-smoker | Sedentary | Employed | 42710.36 |
| **2** | Shannon Church | 78 | Widowed | Master's Degree | 1 | Non-smoker | Sedentary | Employed | 125332.79 |
| **3** | Charles Jordan | 58 | Divorced | Master's Degree | 3 | Non-smoker | Moderate | Unemployed | 9992.78 |
| **4** | Michael Rich | 18 | Single | High School | 0 | Non-smoker | Sedentary | Unemployed | 8595.08 |

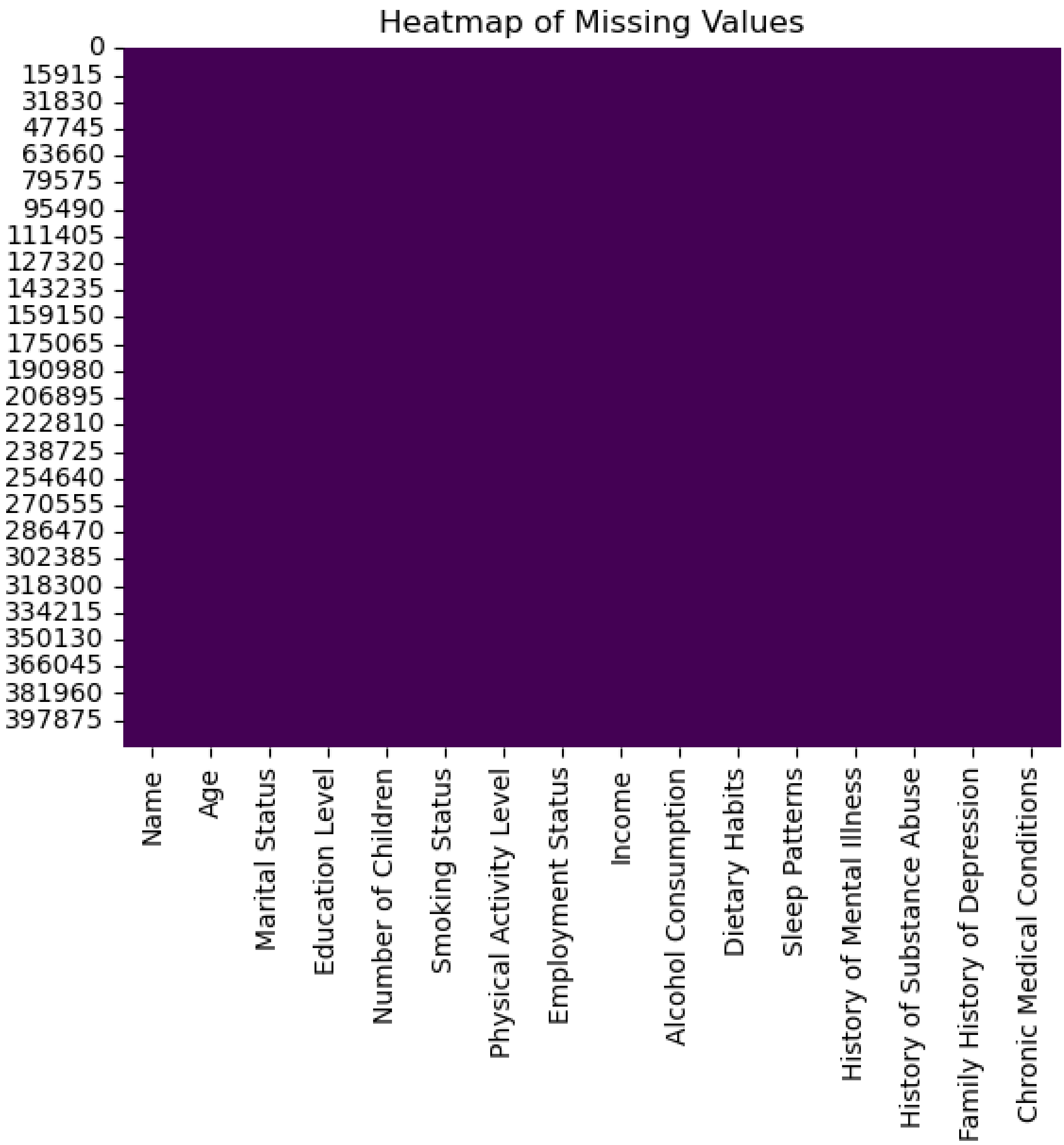
In [3]:
```
# Check data types, missing values, and summary
print(df.info());
df.describe(include='all');
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 413768 entries, 0 to 413767
Data columns (total 16 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   Name                          413768 non-null  object 
 1   Age                           413768 non-null  int64  
 2   Marital Status                413768 non-null  object 
 3   Education Level               413768 non-null  object 
 4   Number of Children            413768 non-null  int64  
 5   Smoking Status                413768 non-null  object 
 6   Physical Activity Level       413768 non-null  object 
 7   Employment Status             413768 non-null  object 
 8   Income                        413768 non-null  float64
 9   Alcohol Consumption           413768 non-null  object 
 10  Dietary Habits                413768 non-null  object 
 11  Sleep Patterns                413768 non-null  object 
 12  History of Mental Illness     413768 non-null  object 
 13  History of Substance Abuse    413768 non-null  object 
 14  Family History of Depression  413768 non-null  object 
 15  Chronic Medical Conditions    413768 non-null  object 
dtypes: float64(1), int64(2), object(13)
memory usage: 50.5+ MB
None
```

### This step ensures we understand the data types (categorical or numerical), detect missing values, and get basic statistical summaries.

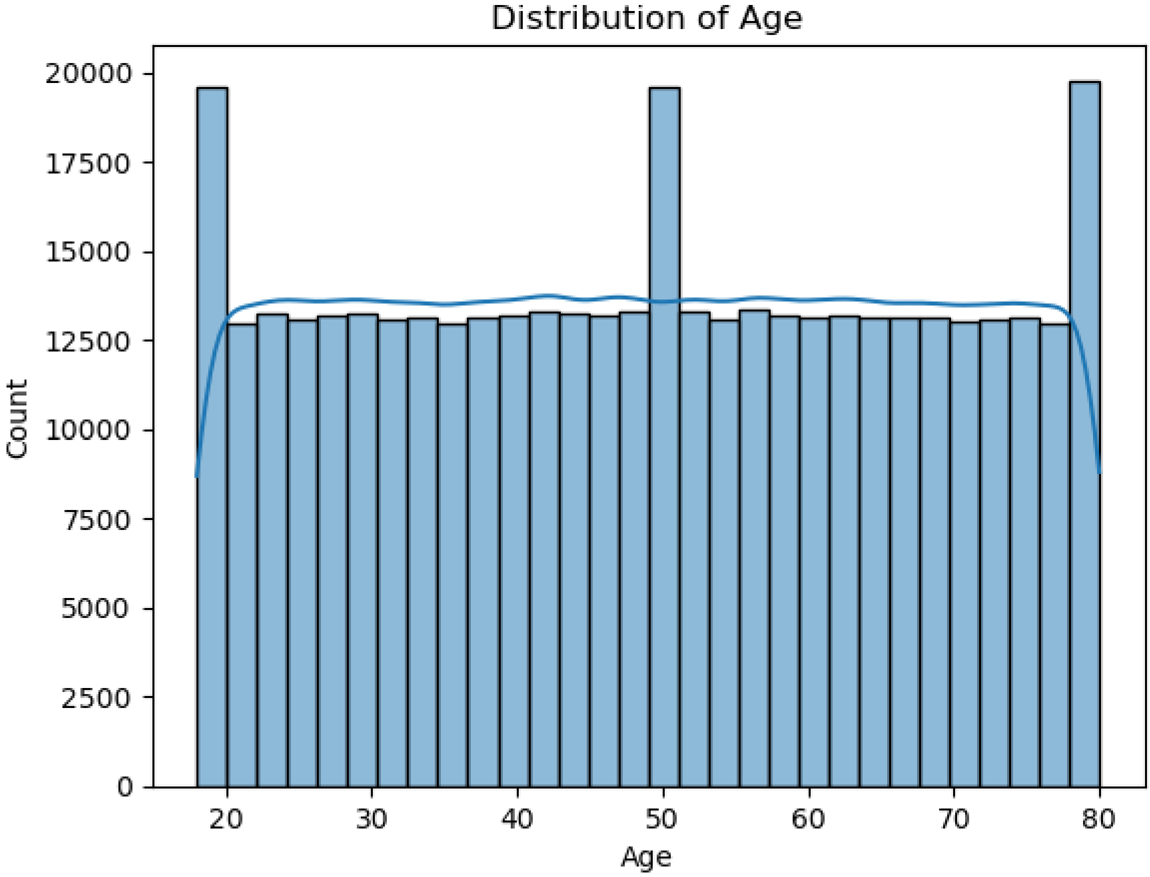
In [4]:
```python
# Check missing values
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)

# Visualize missing values
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Heatmap of Missing Values")
plt.show()

# Impute or drop missing values (example strategies)
df.fillna(method='ffill', inplace=True)
```

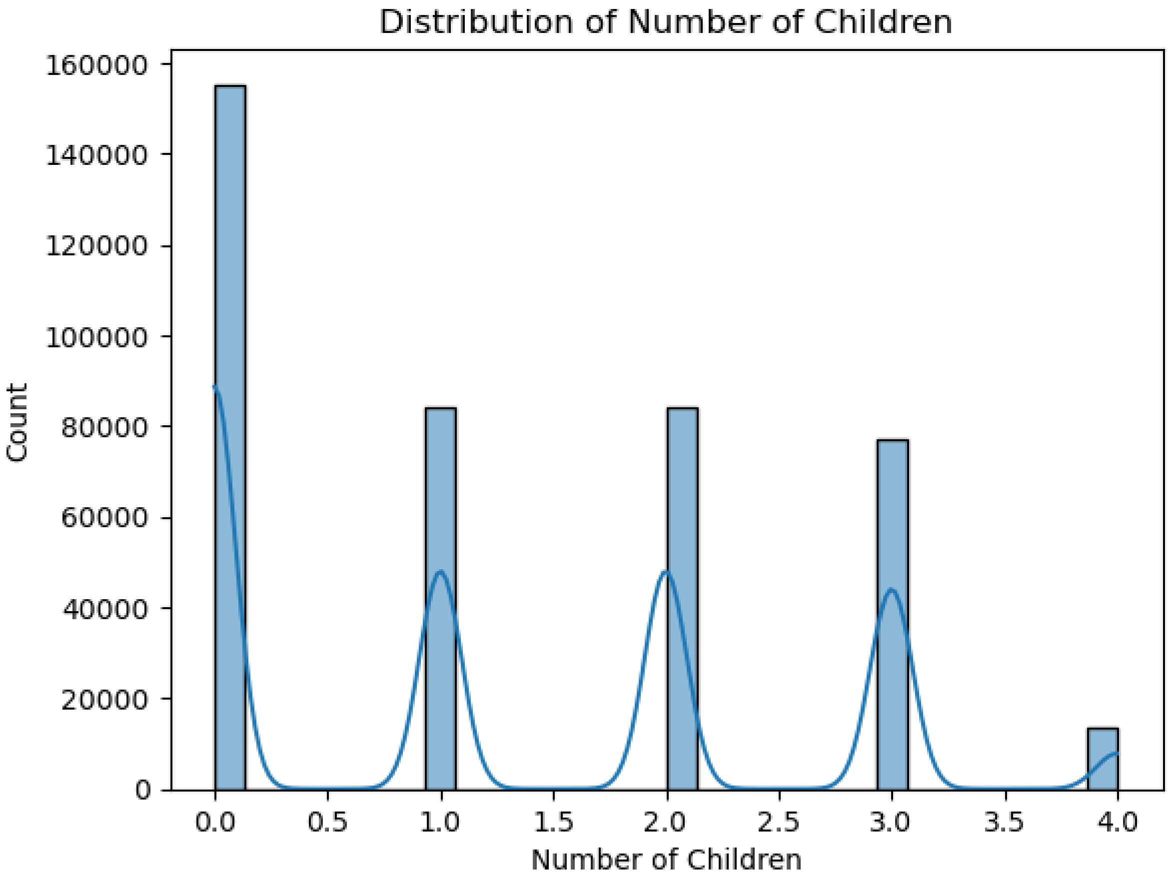
```
Missing Values:
 Name                            0
Age                             0
Marital Status                  0
Education Level                 0
Number of Children              0
Smoking Status                  0
Physical Activity Level         0
Employment Status               0
Income                          0
Alcohol Consumption             0
Dietary Habits                  0
Sleep Patterns                  0
History of Mental Illness       0
History of Substance Abuse      0
Family History of Depression    0
Chronic Medical Conditions      0
dtype: int64
```

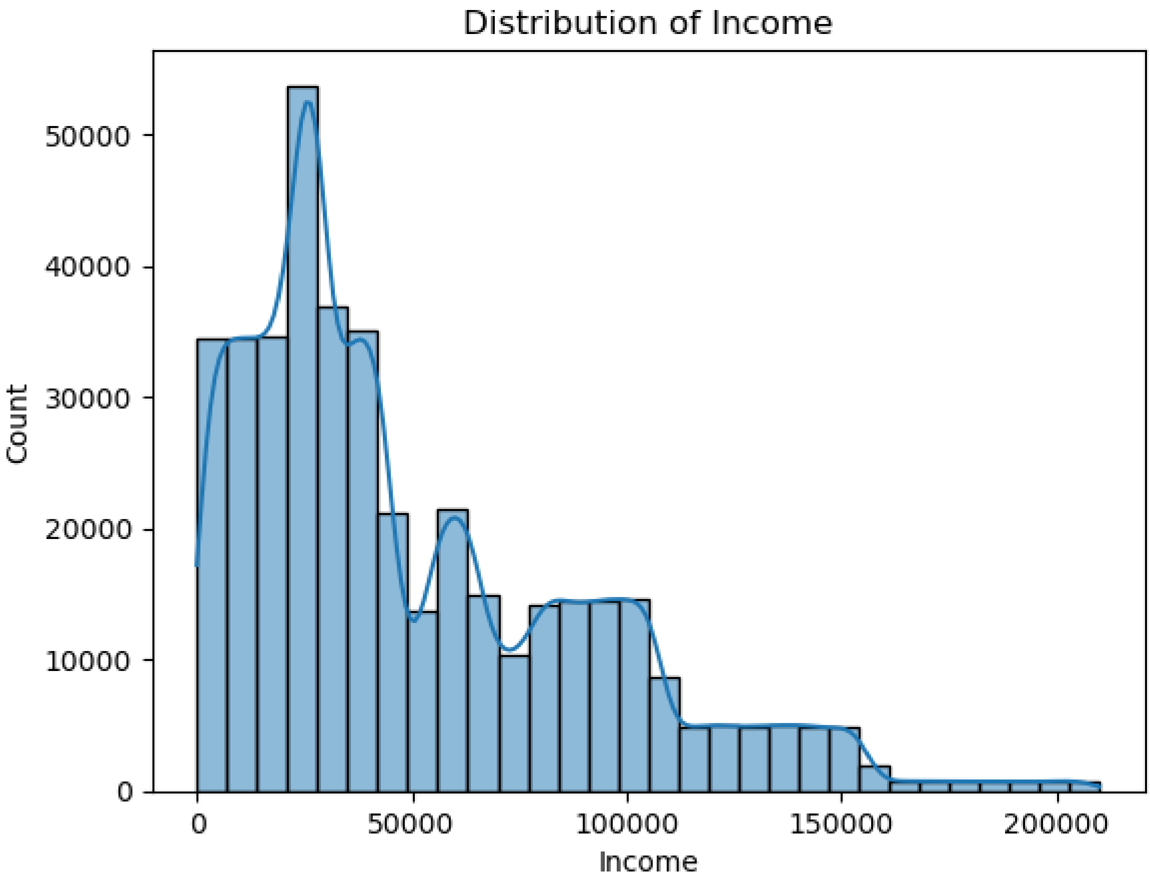



Missing data can significantly impact model performance. Imputing missing values ensures completeness without losing information. We didnt have to do anything as the dataset doenst contain any missing data. Probably its a synthetic dataset.

In [5]:
```
# Distribution of numerical variables
numerical_columns = ['Age', 'Number of Children', 'Income']
for col in numerical_columns:
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f"Distribution of {col}")
    plt.show()
```

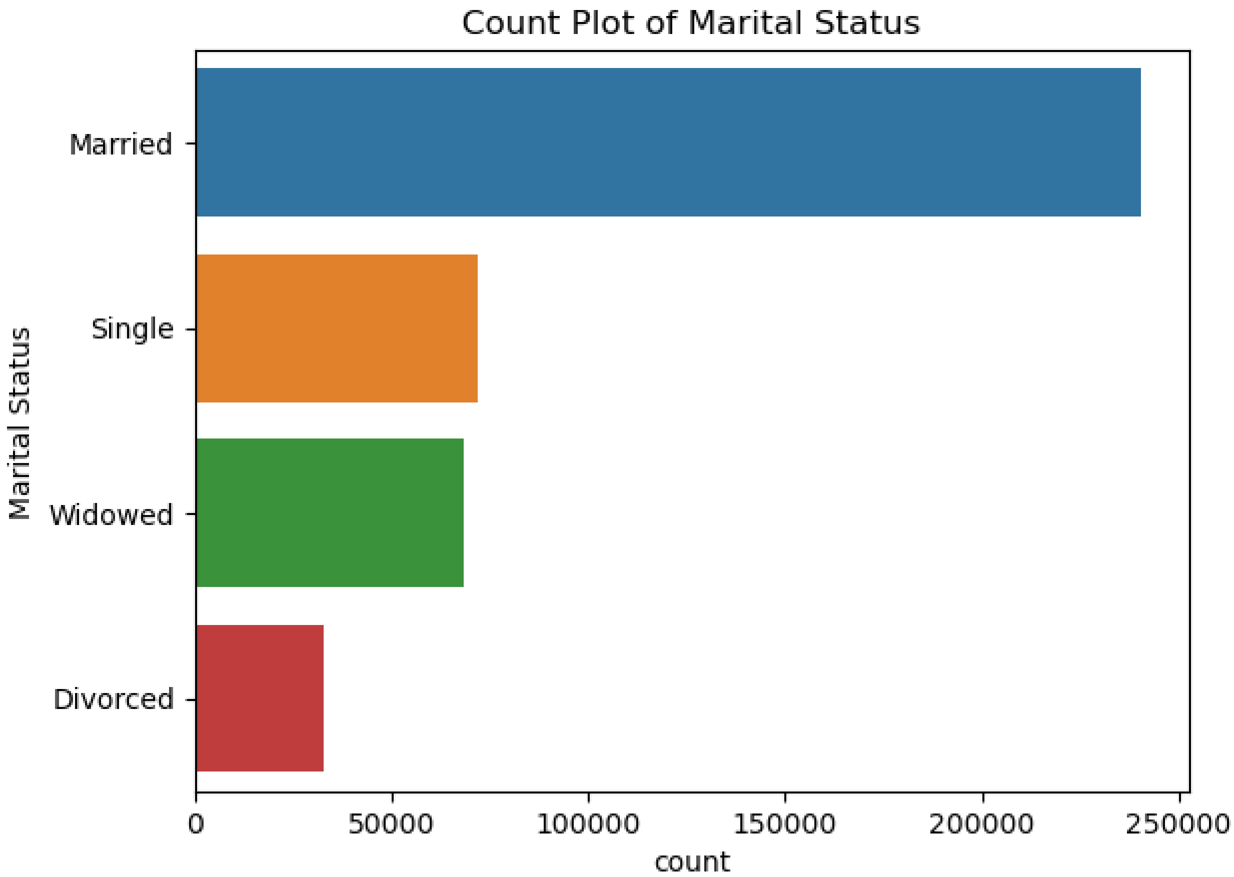

Distribution of Age
Count
Age
0
2500
5000
7500
10000
12500
15000
17500
20000
20
30
40
50
60
70
80


Distribution of Number of Children
Count
Number of Children
0
20000
40000
60000
80000
100000
120000
140000
160000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

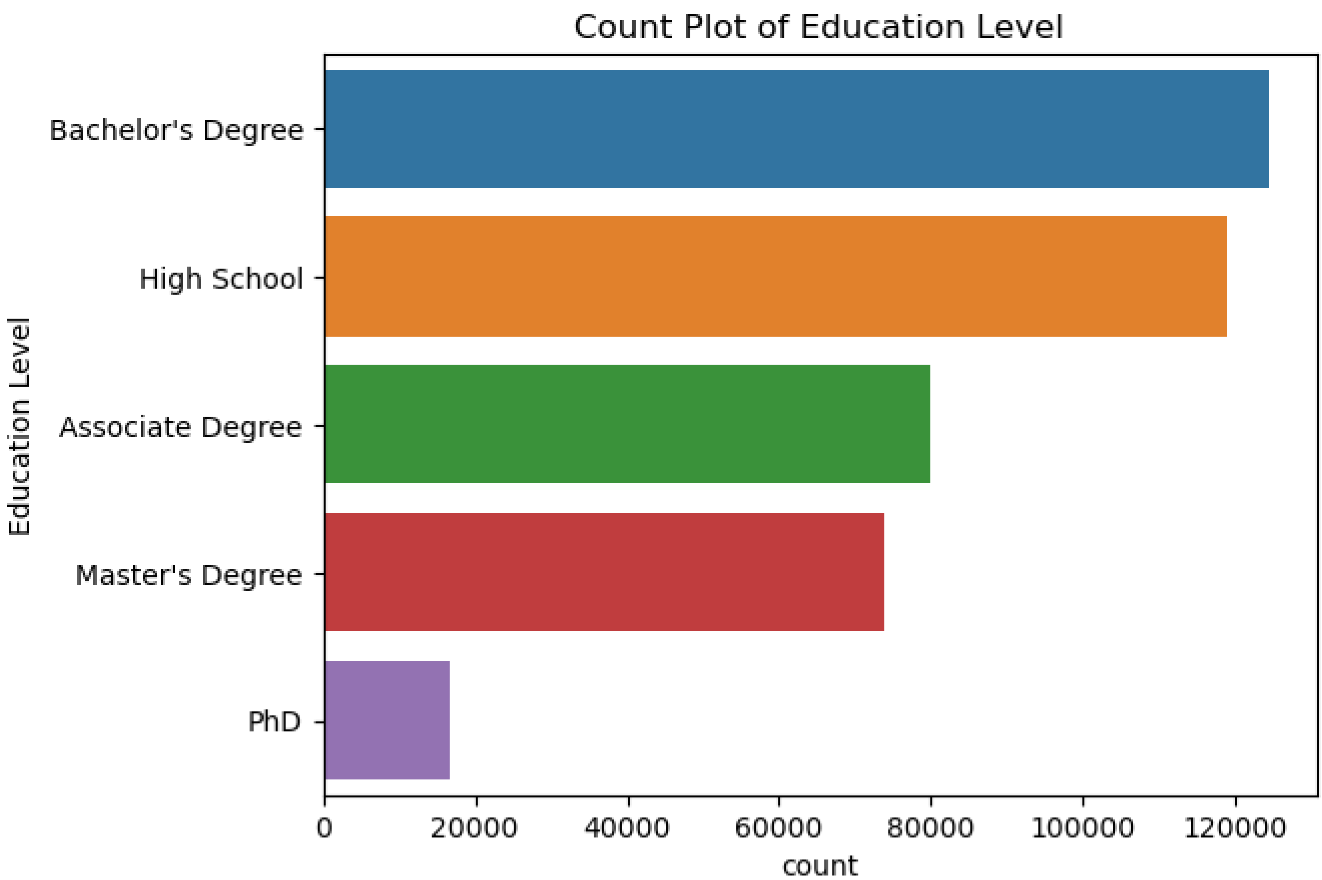



Identifying the spread and skewness of numerical variables helps in scaling or transformations.

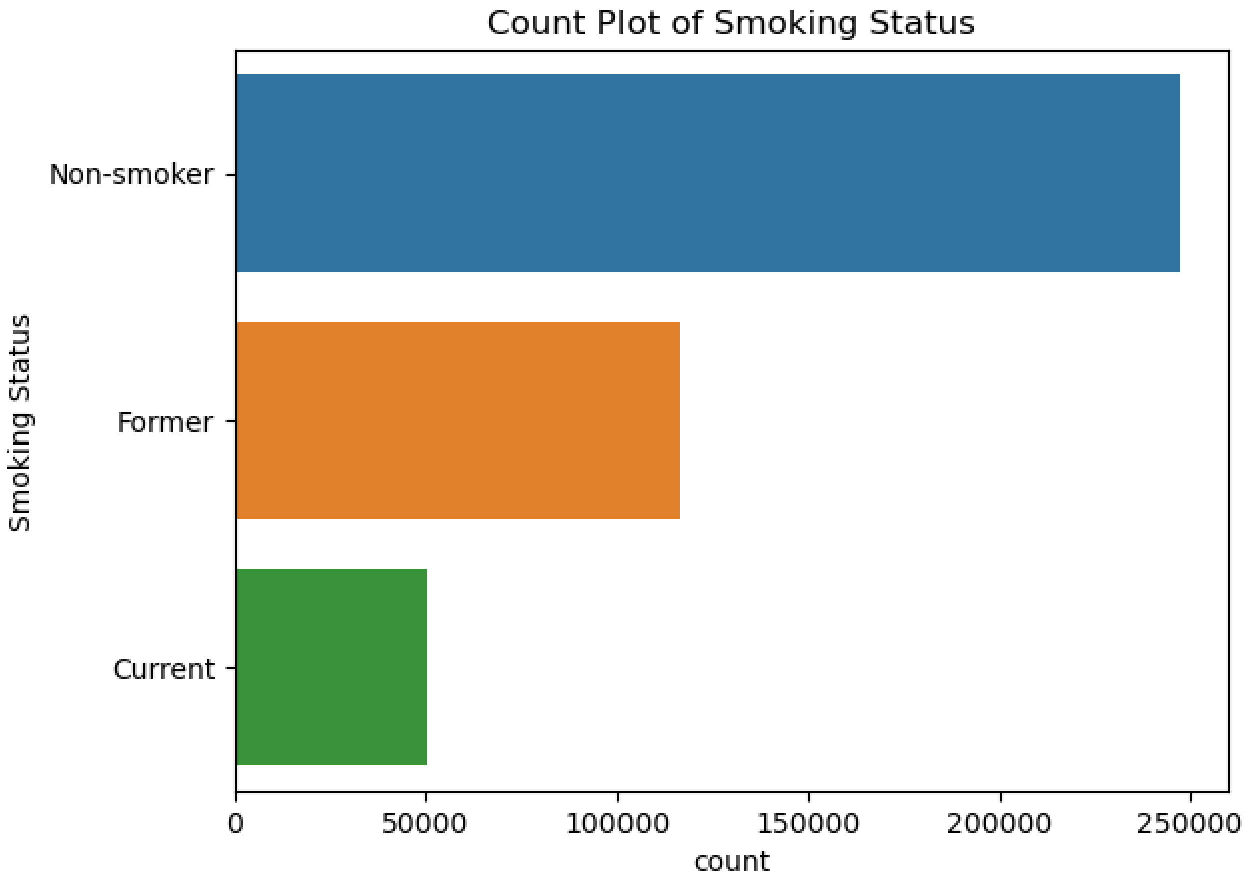
In [6]:
```
# Count plots for categorical variables
categorical_columns = ['Marital Status', 'Education Level', 'Smoking Status', 'Physic
                       'Employment Status', 'Alcohol Consumption', 'Dietary Habits',
                       'History of Mental Illness', 'History of Substance Abuse', 'Fa
                       'Chronic Medical Conditions']
for col in categorical_columns:
    sns.countplot(y=df[col], order=df[col].value_counts().index)
    plt.title(f"Count Plot of {col}")
    plt.show()
```

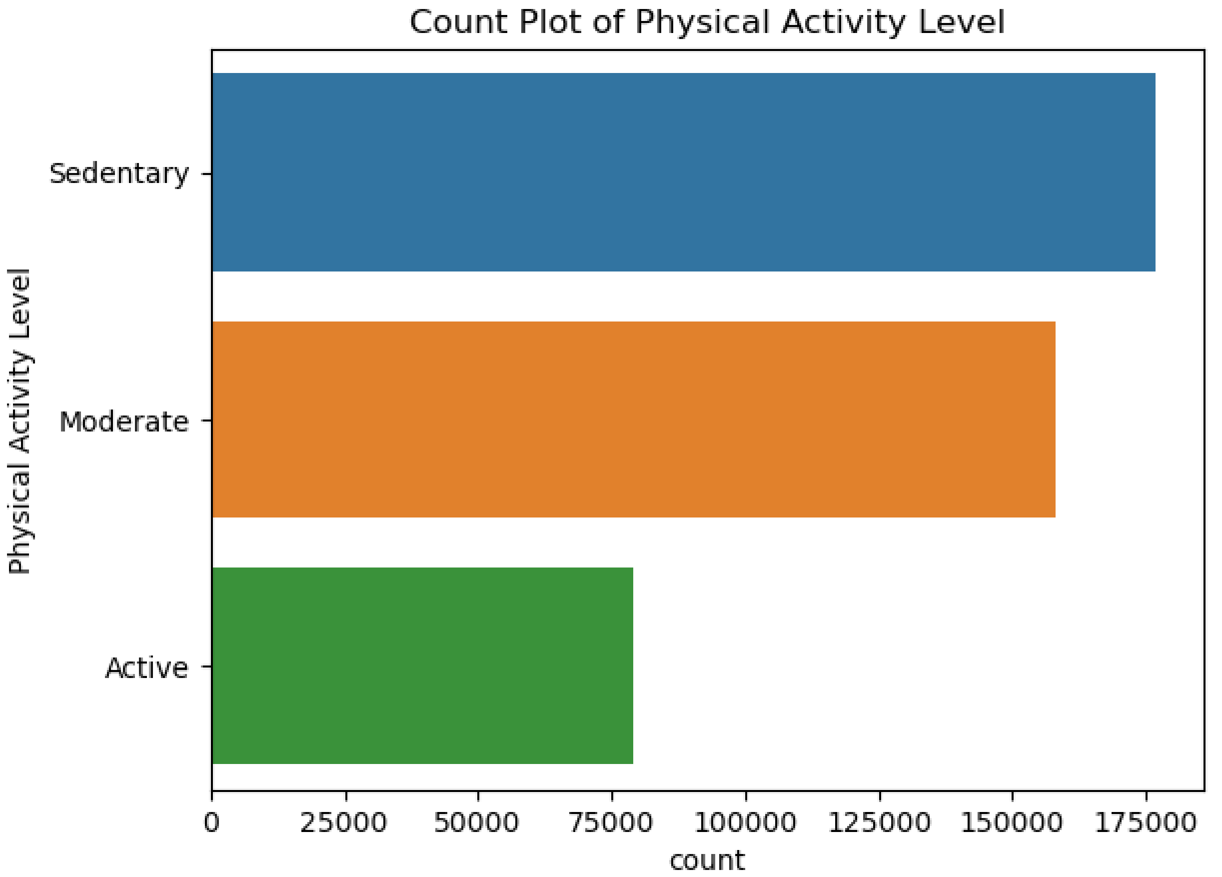


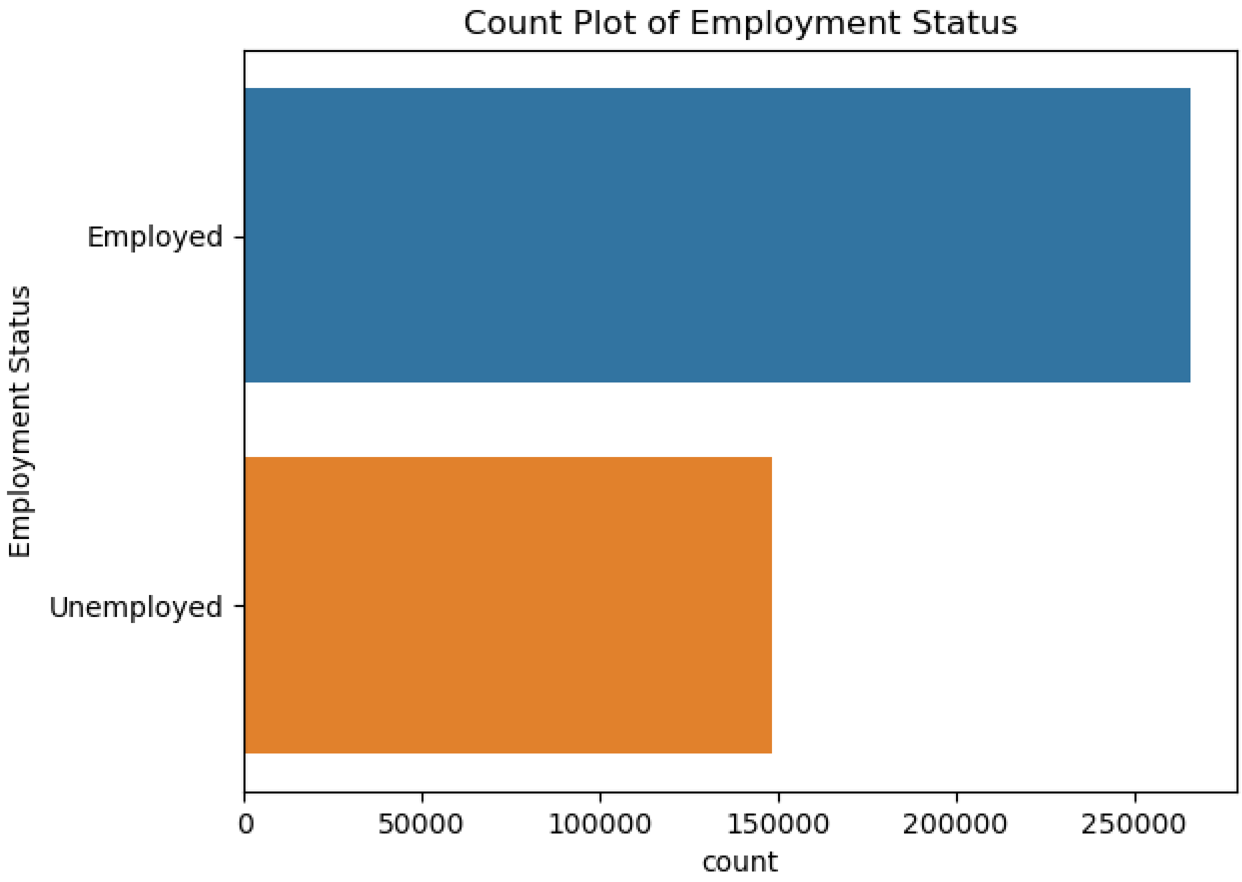
Count Plot of Marital Status
Marital Status
Married
Single
Widowed
Divorced
0
50000
100000
150000
200000
250000
count

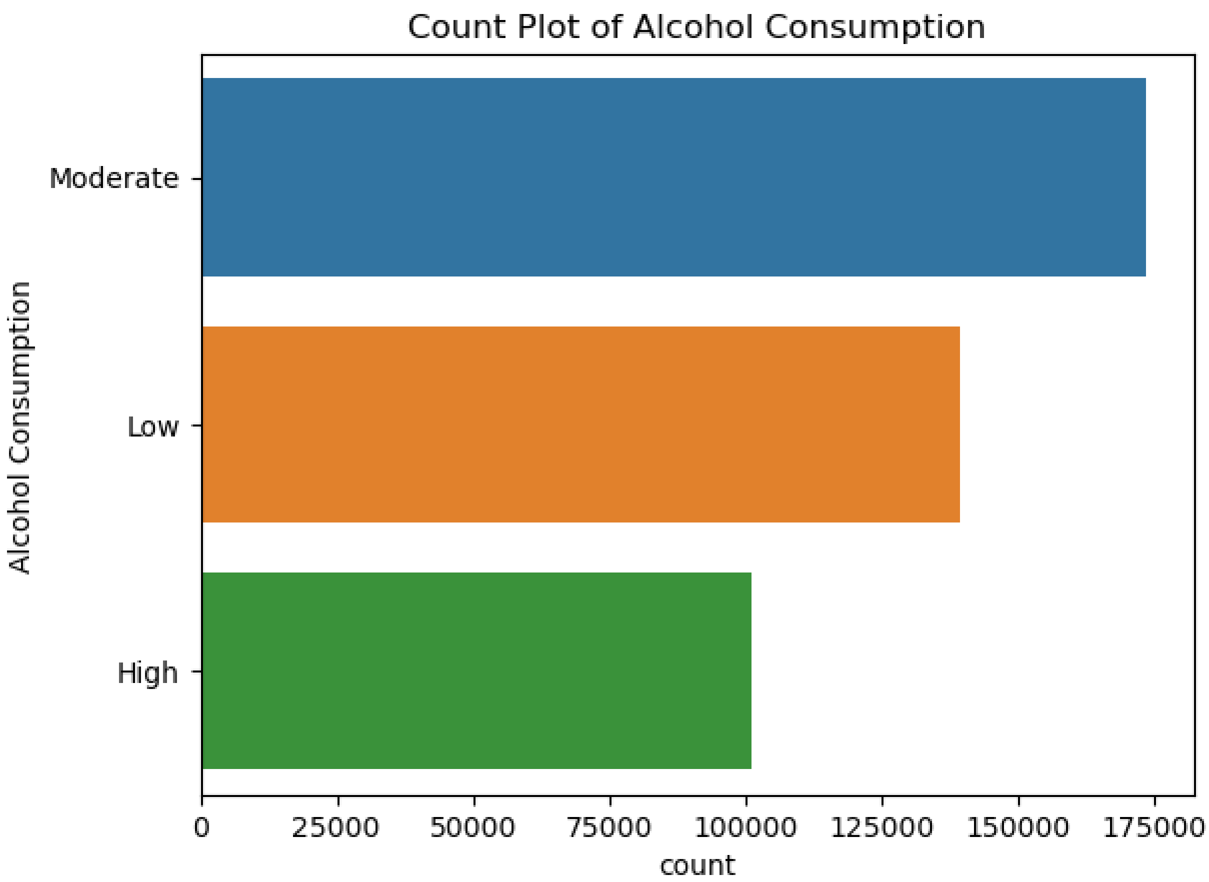



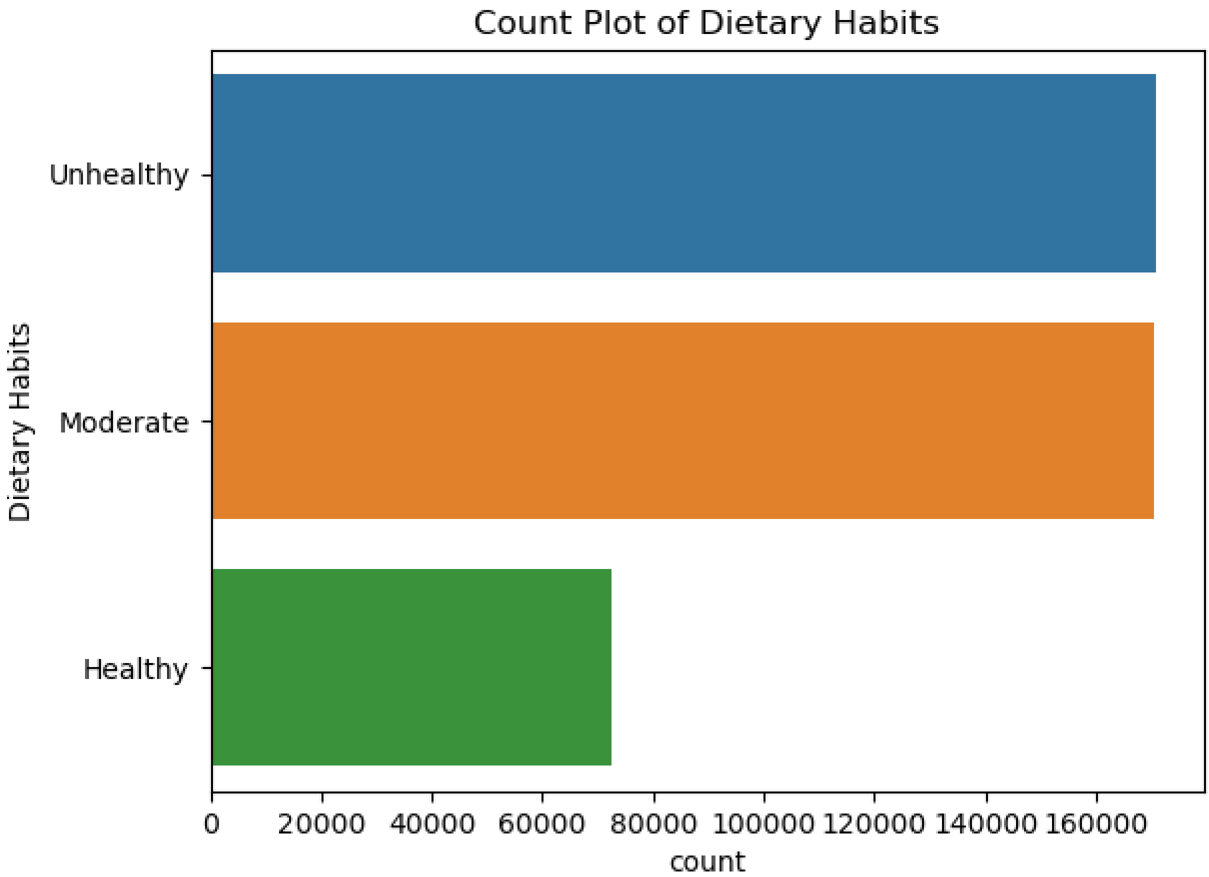
Count Plot of Education Level
Education Level
Bachelor's Degree
High School
Associate Degree
Master's Degree
PhD
0
20000
40000
60000
80000
100000
120000
count

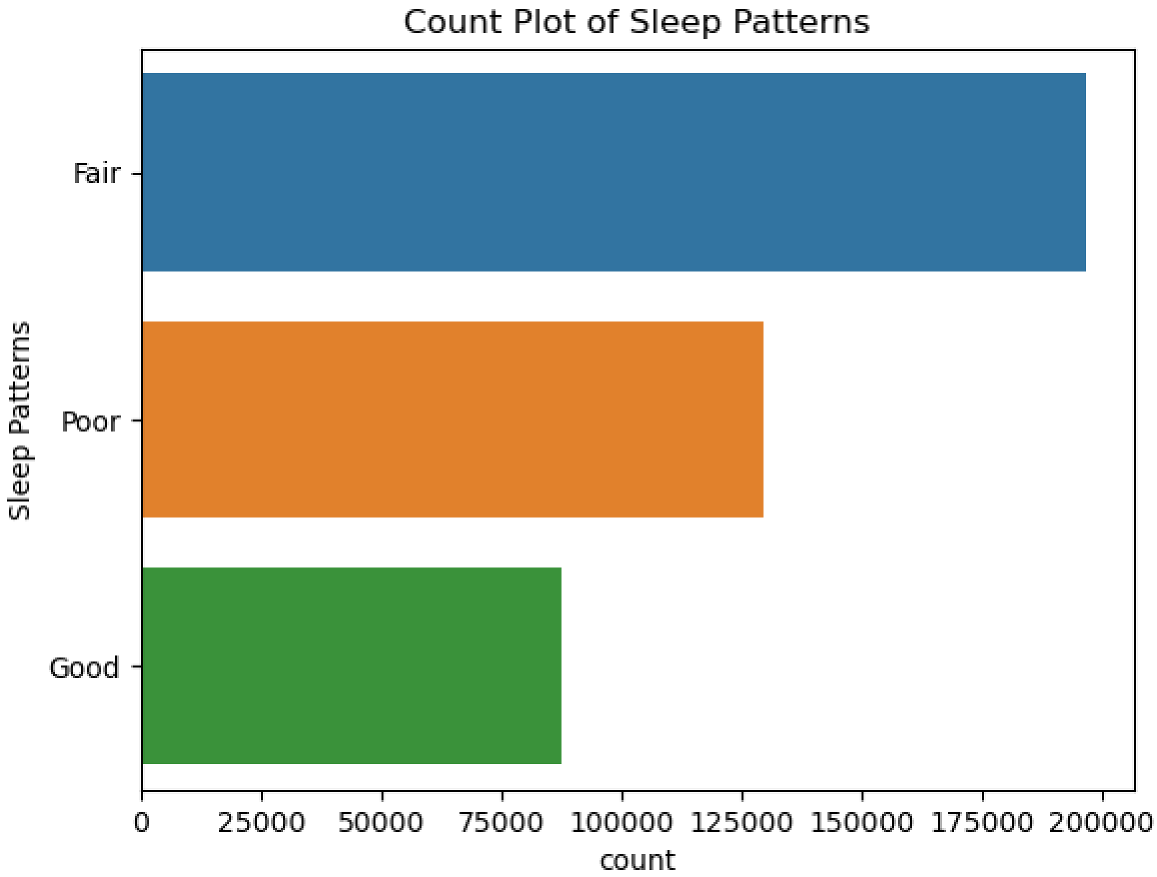

Count Plot of Smoking Status
Smoking Status
Non-smoker
Former
Current
0
50000
100000
150000
200000
250000
count

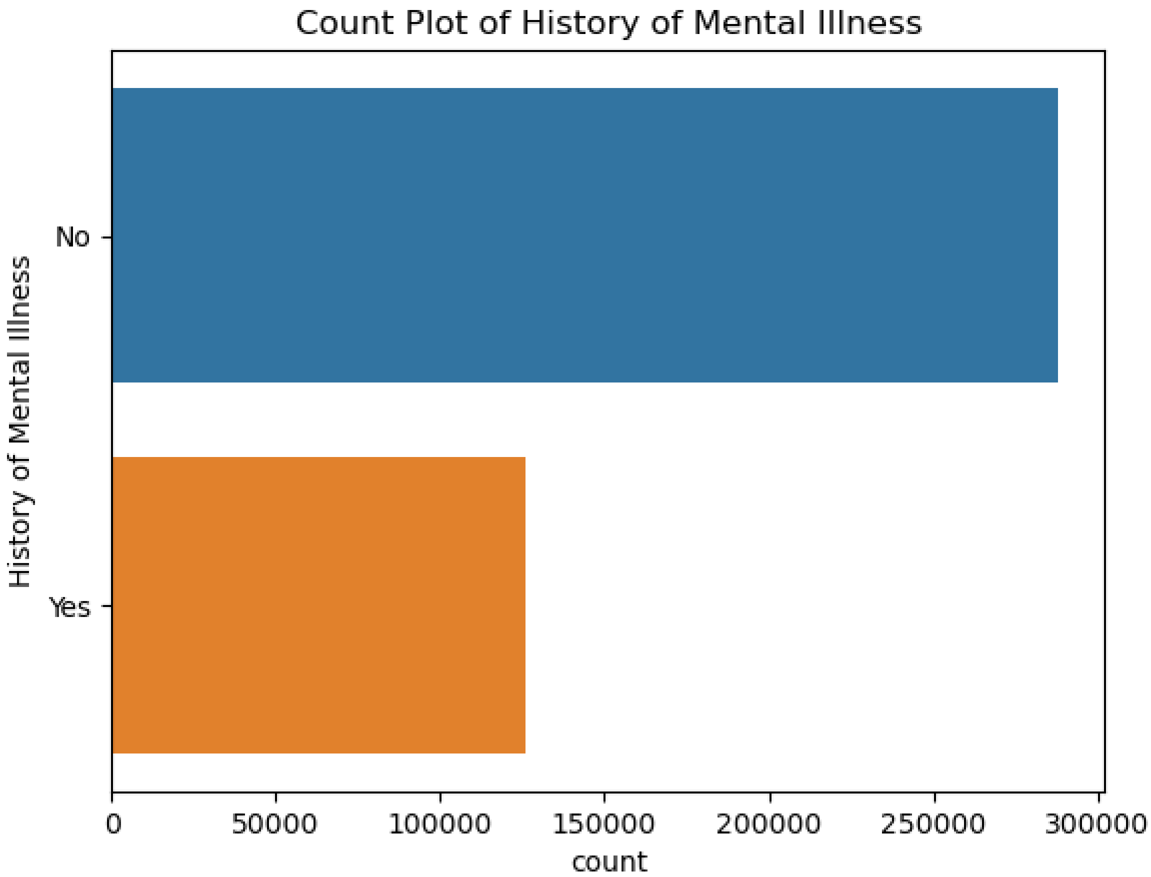

Count Plot of Physical Activity Level
Physical Activity Level
Sedentary
Moderate
Active
0
25000
50000
75000
100000
125000
150000
175000
count

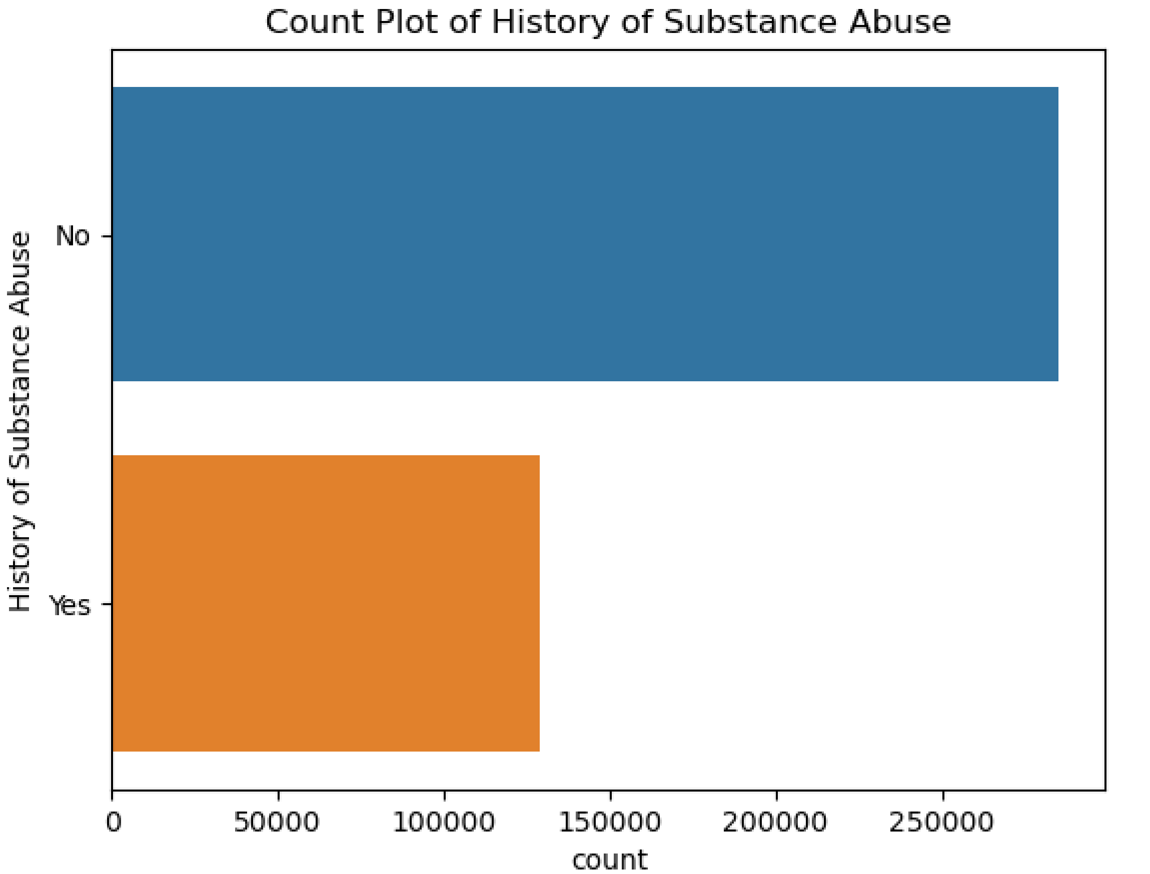

Count Plot of Employment Status
Employed
Unemployed
Employment Status
0
50000
100000
150000
200000
250000
count

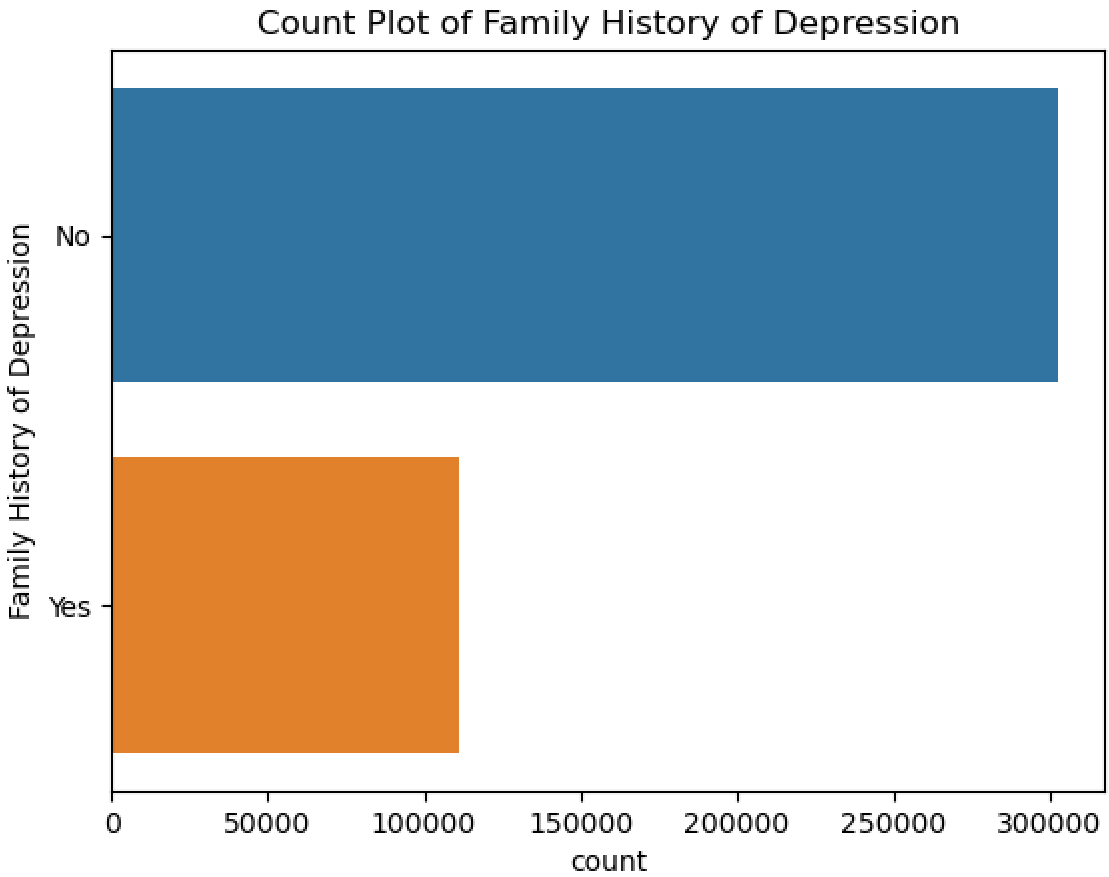

Count Plot of Alcohol Consumption
Moderate
Low
High
Alcohol Consumption
0
25000
50000
75000
100000
125000
150000
175000
count

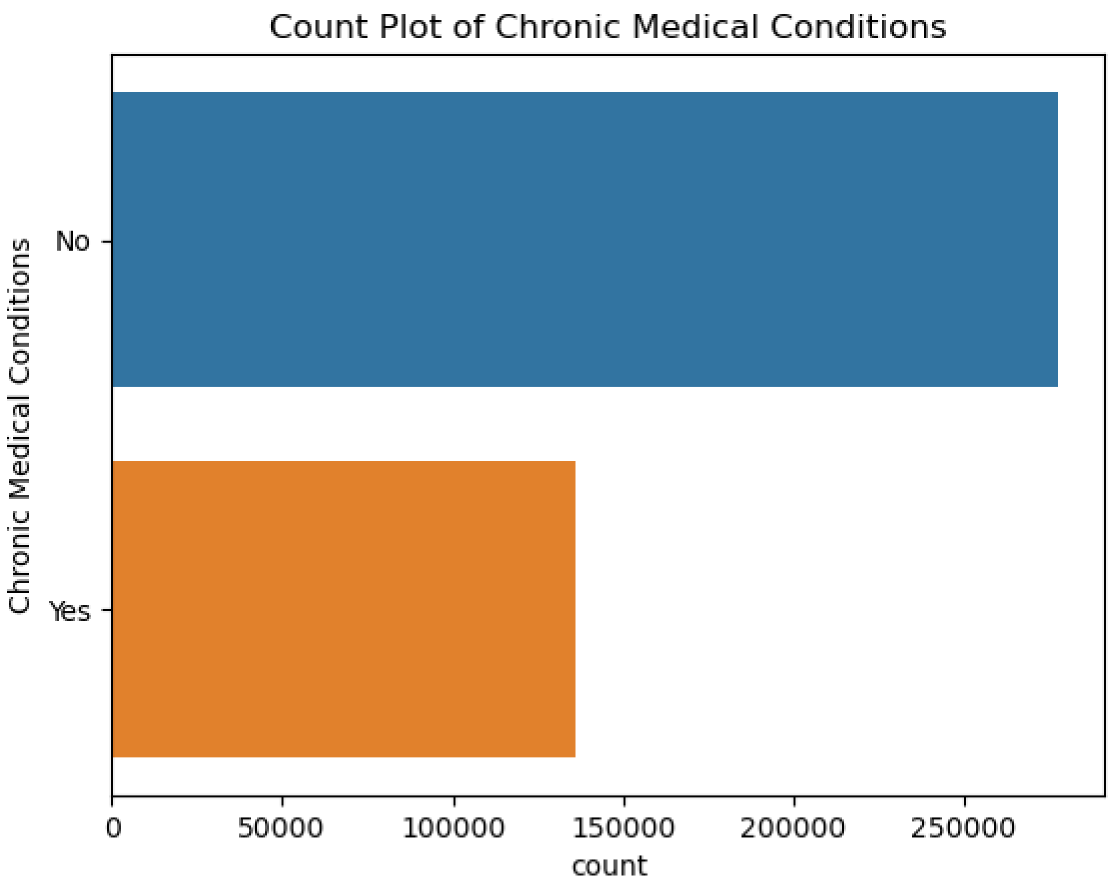

Count Plot of Dietary Habits
Unhealthy
Moderate
Healthy
Dietary Habits
0
20000
40000
60000
80000
100000
120000
140000
160000
count


Count Plot of Sleep Patterns
Fair
Poor
Good
Sleep Patterns
0
25000
50000
75000
100000
125000
150000
175000
200000
count


Count Plot of History of Mental Illness
History of Mental Illness
No
Yes
0
50000
100000
150000
200000
250000
300000
count

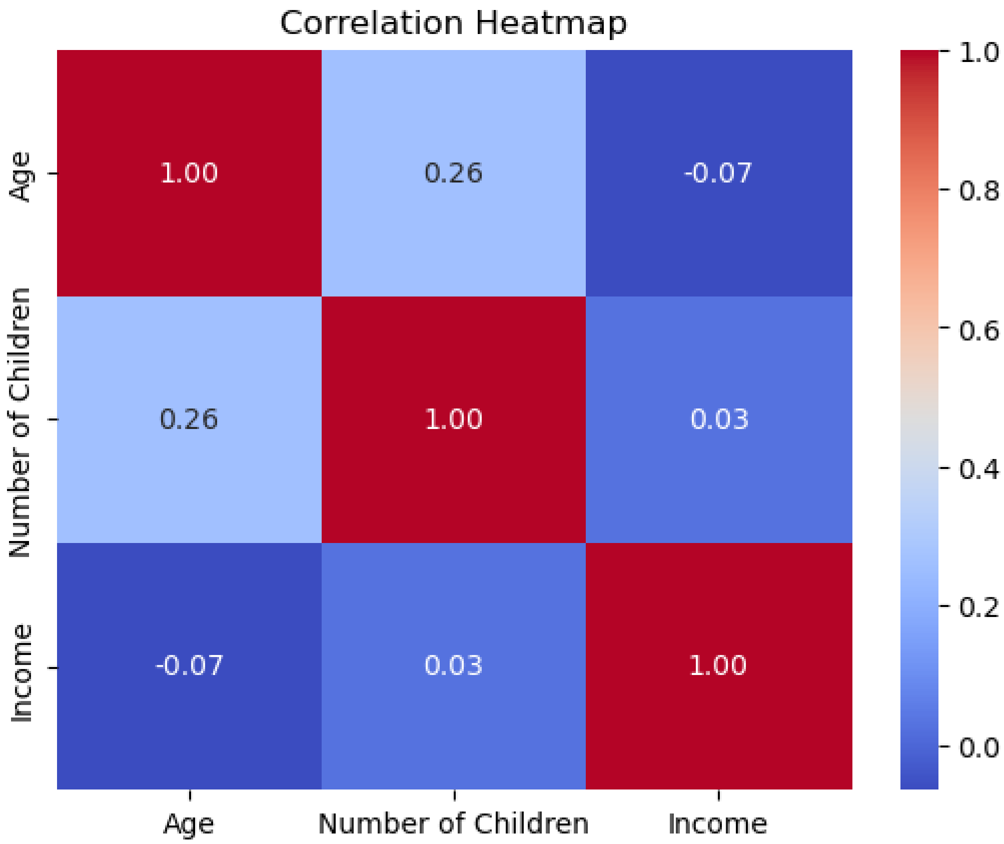

Count Plot of History of Substance Abuse
History of Substance Abuse
No
Yes
0
50000
100000
150000
200000
250000
count



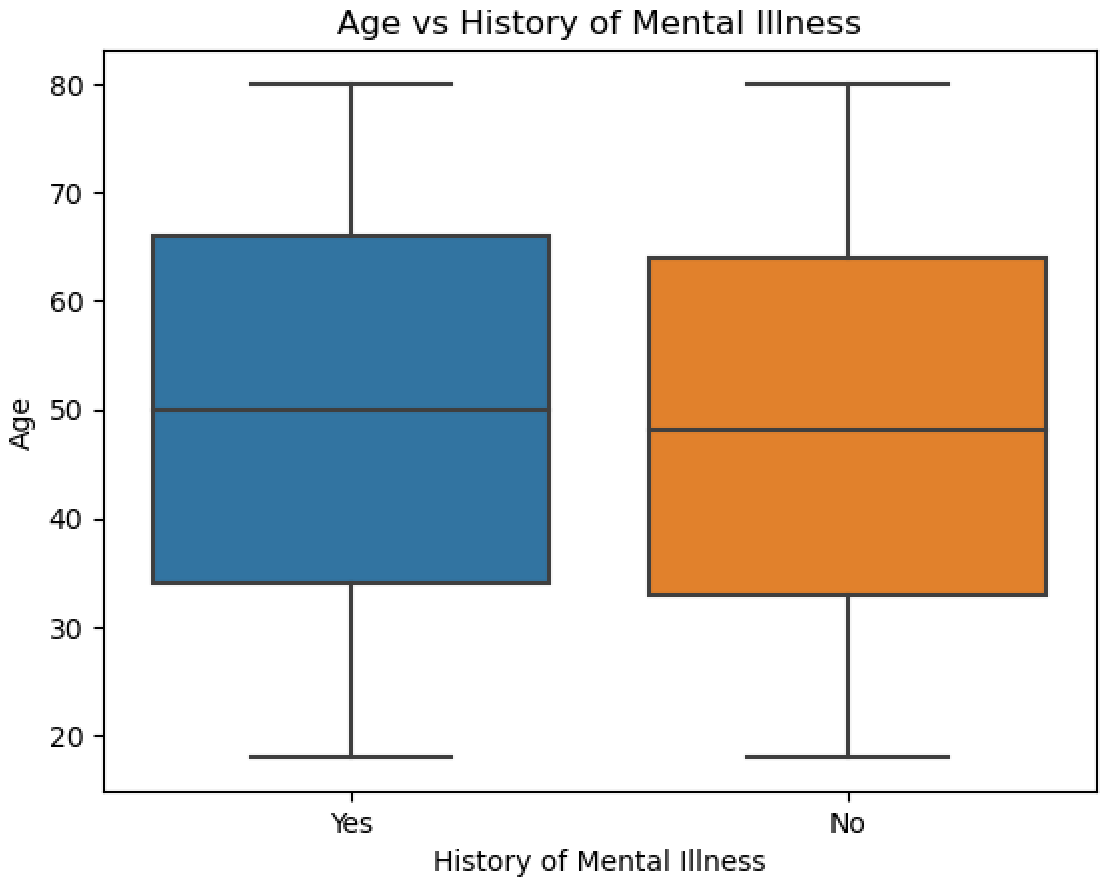



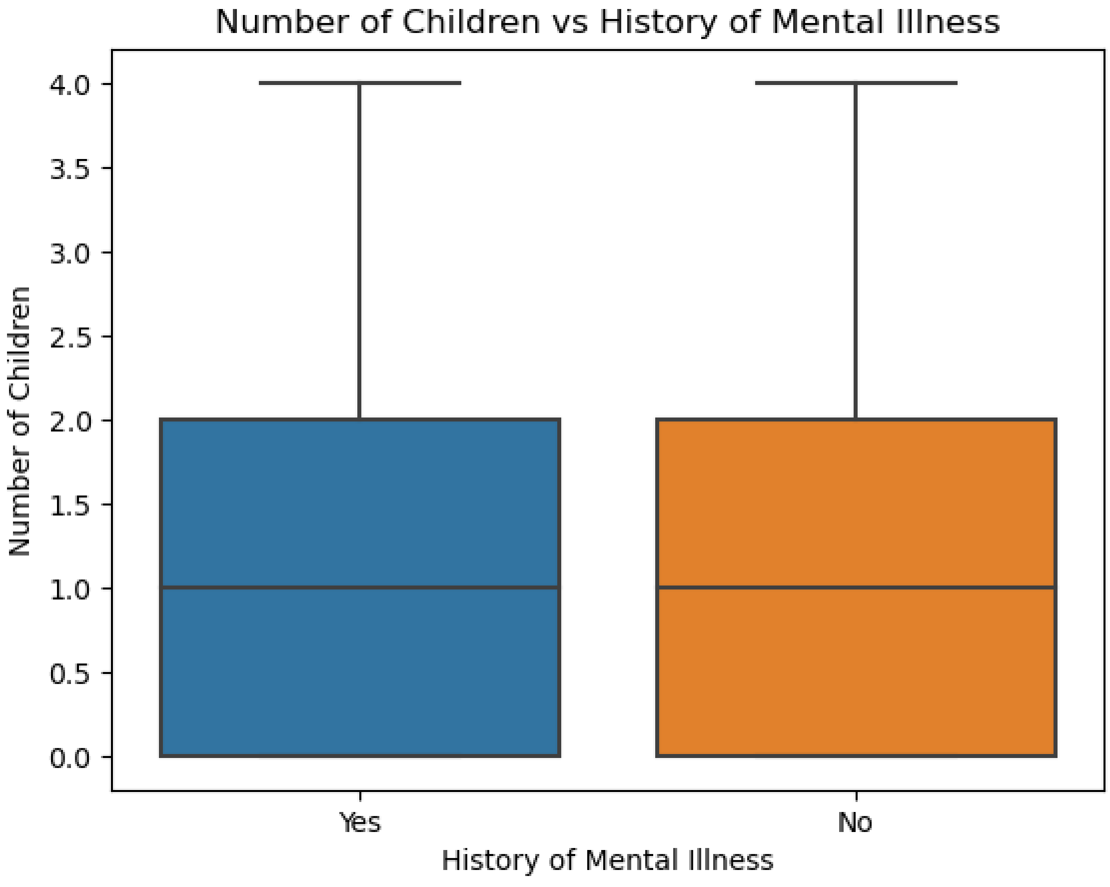

Correlation helps detect multicollinearity among numerical variables, which can affect model performance.

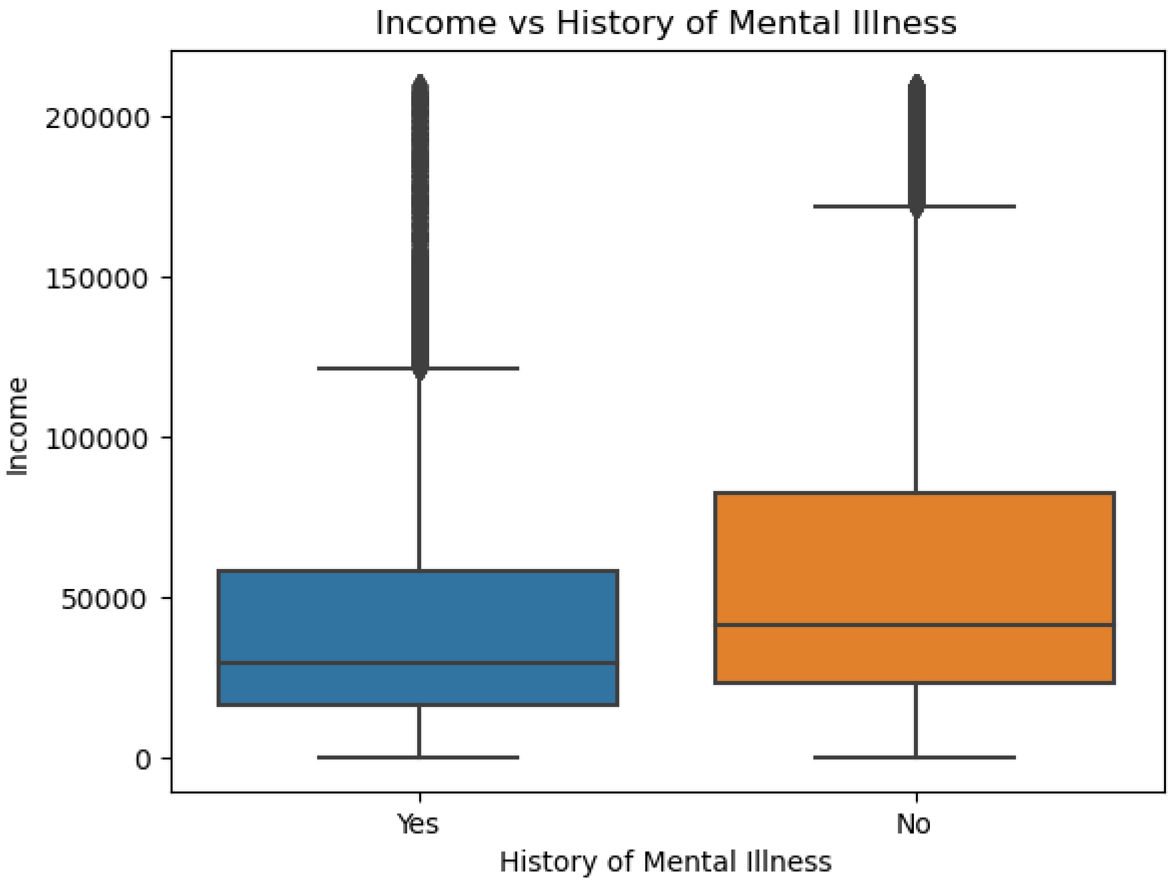
In [7]:
```
# Correlation heatmap
corr_matrix = df[numerical_columns].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



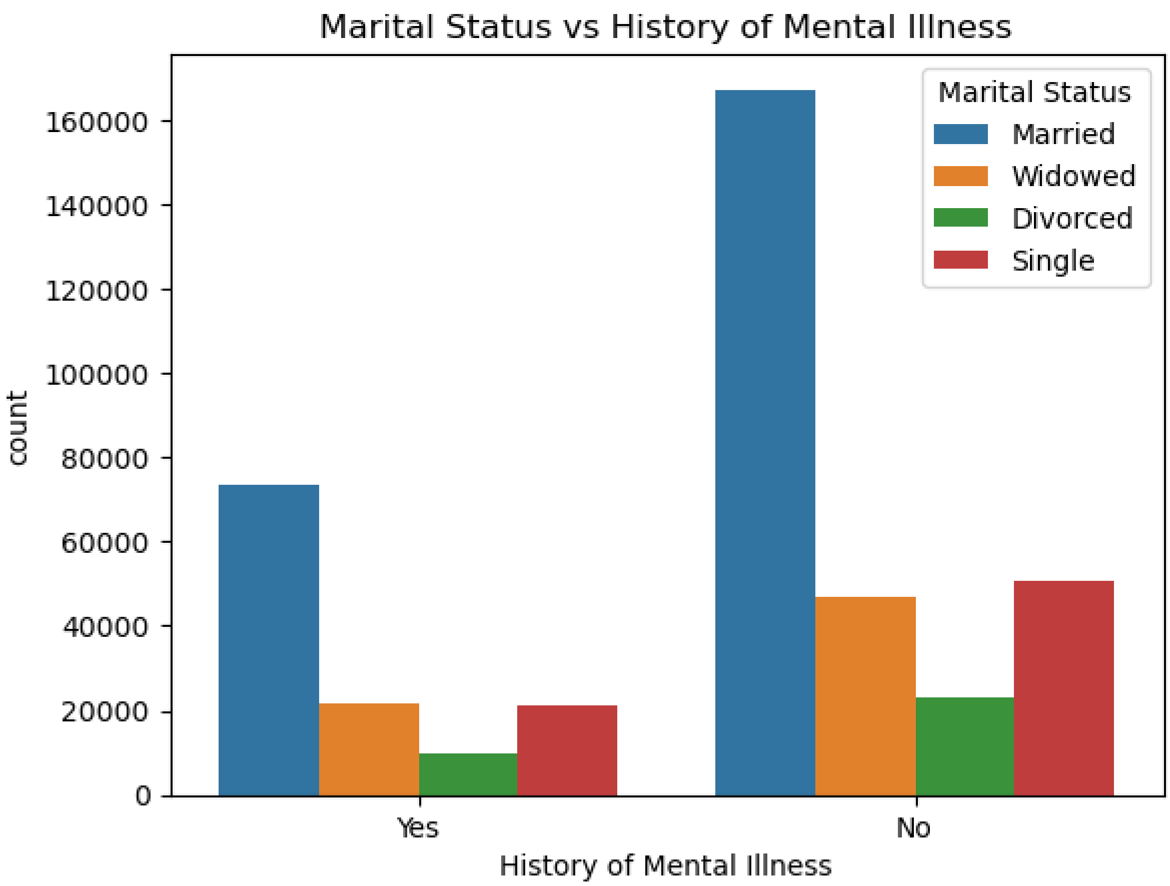

In [8]:
```
# Boxplots for numerical variables vs target
for col in numerical_columns:
    sns.boxplot(x='History of Mental Illness', y=col, data=df)
    plt.title(f"{col} vs History of Mental Illness")
    plt.show()

# Count plots for categorical variables vs target
for col in categorical_columns:
    if col != 'History of Mental Illness':  # Exclude target from predictors
        sns.countplot(x='History of Mental Illness', hue=col, data=df)
        plt.title(f"{col} vs History of Mental Illness")
        plt.show()
```


Age vs History of Mental Illness
80
70
60
50
40
30
20
Age
Yes
No
History of Mental Illness

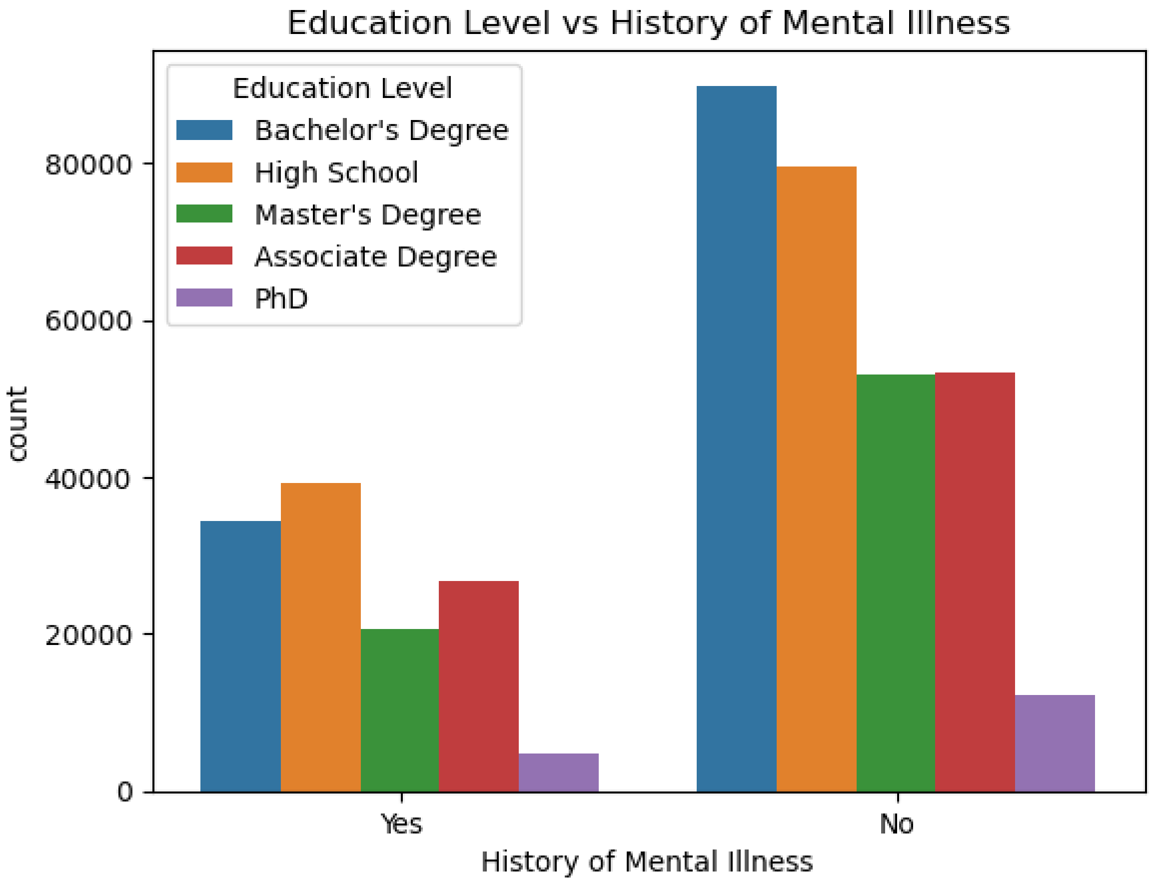

Number of Children vs History of Mental Illness
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Number of Children
Yes
No
History of Mental Illness

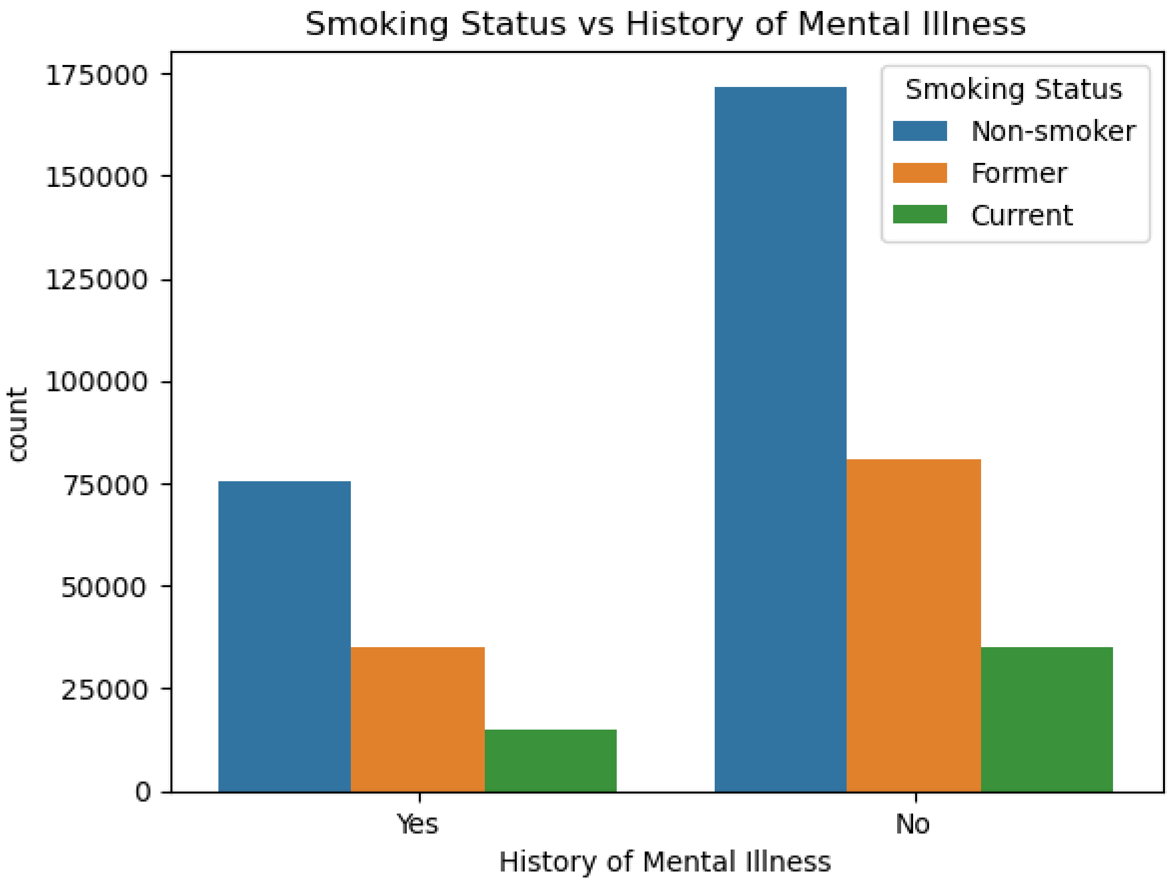


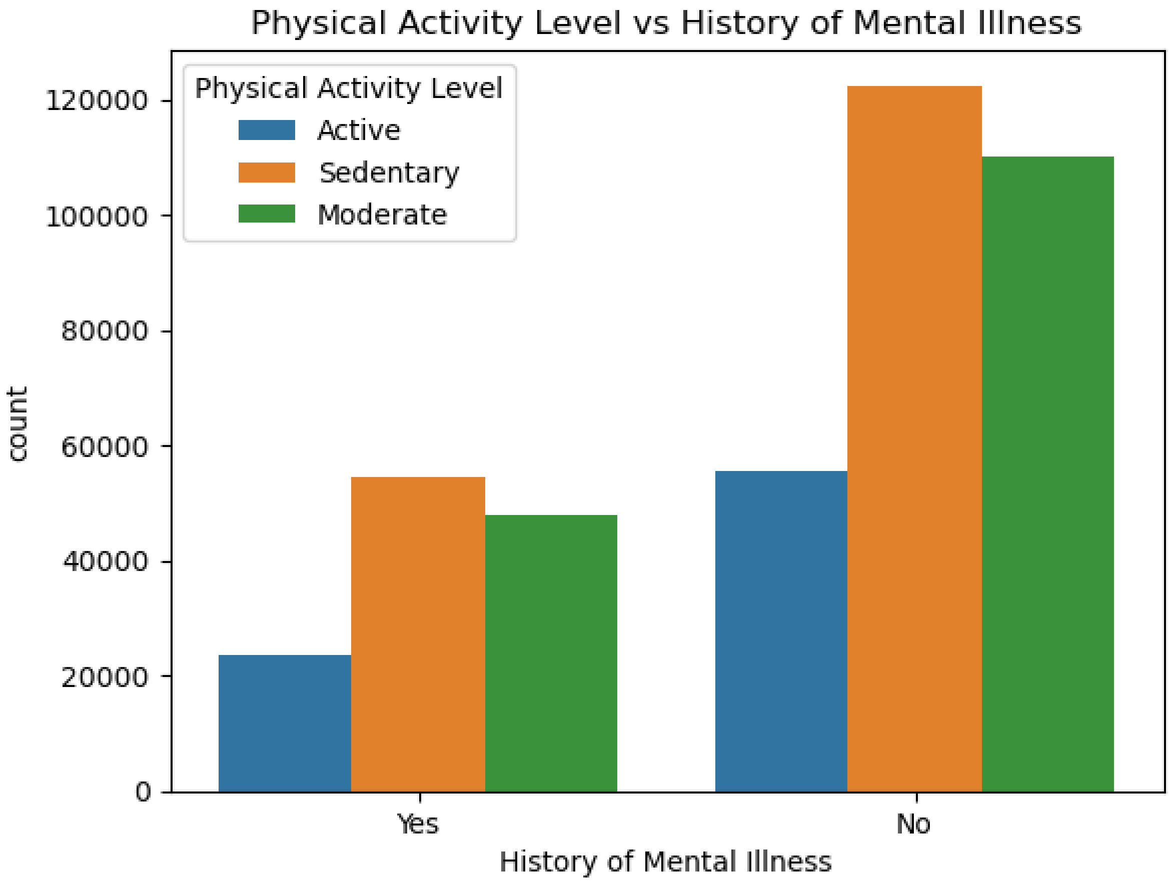
Income vs History of Mental Illness
200000
150000
100000
50000
0
Income
Yes
No
History of Mental Illness

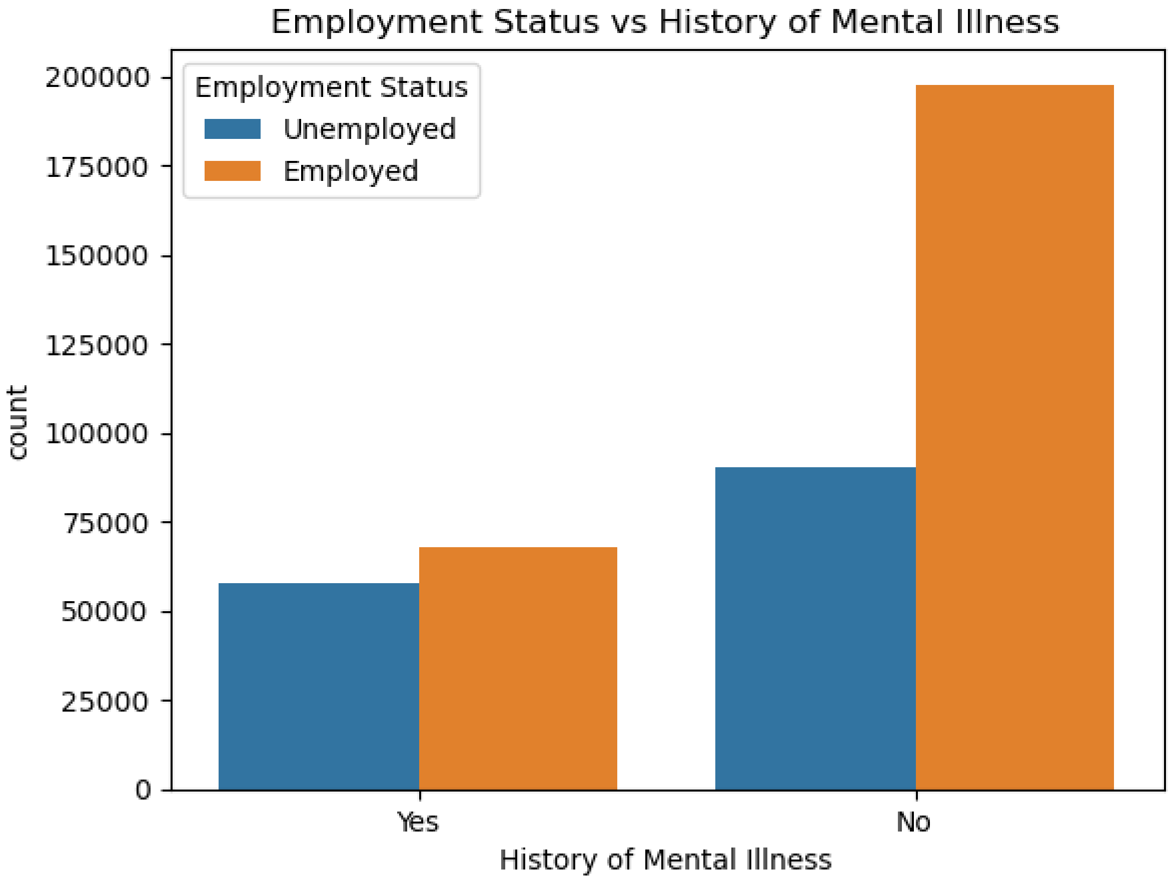



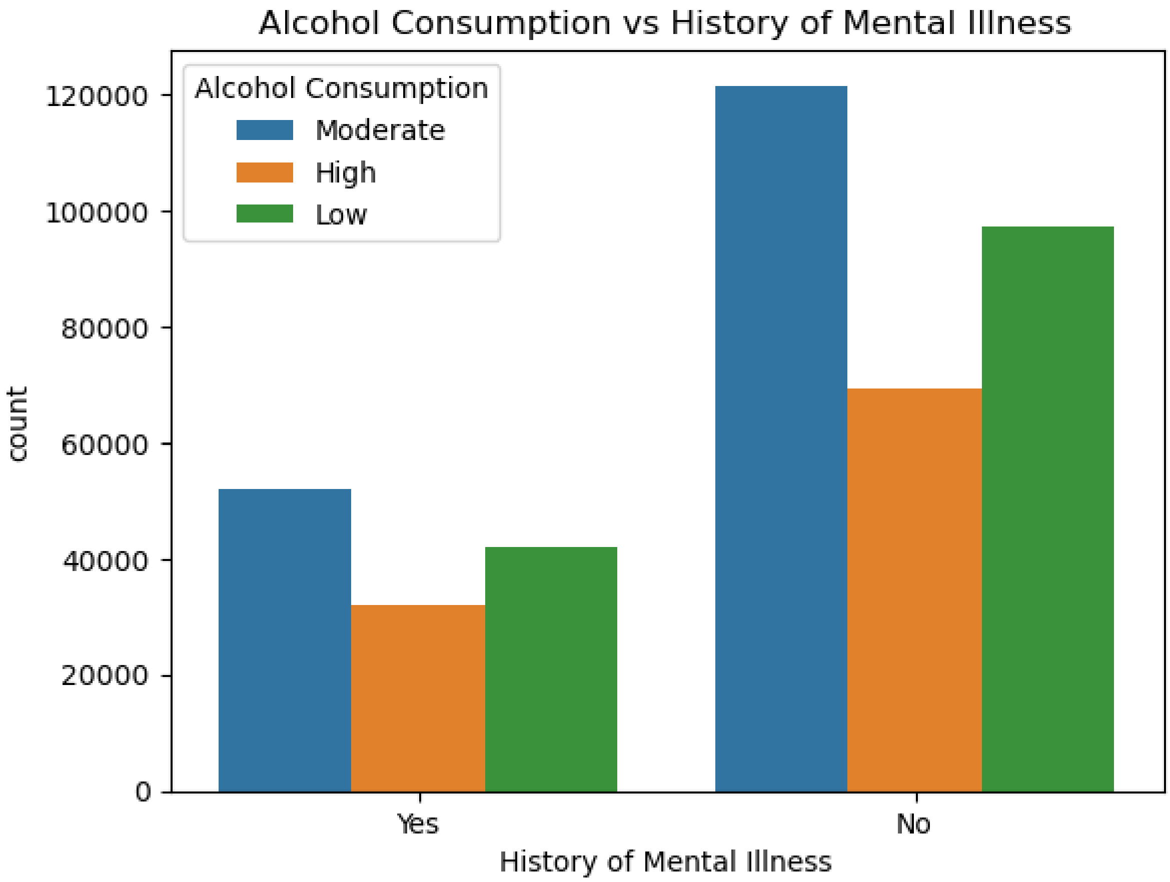
Marital Status vs History of Mental Illness
Marital Status
Married
Widowed
Divorced
Single
160000
140000
120000
100000
80000
60000
40000
20000
0
count
Yes
No
History of Mental Illness

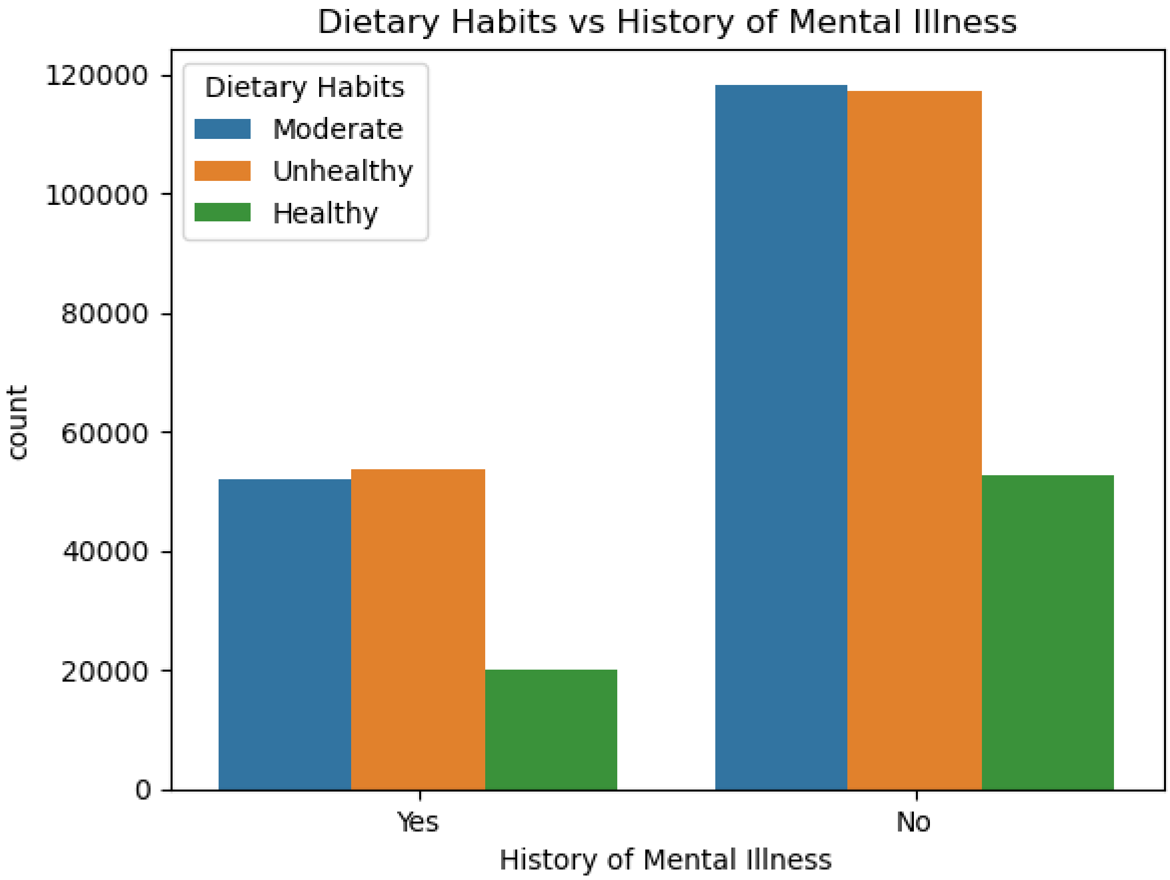

Education Level vs History of Mental Illness
Education Level
Bachelor's Degree
High School
Master's Degree
Associate Degree
PhD
80000
60000
40000
20000
0
count
Yes
No
History of Mental Illness

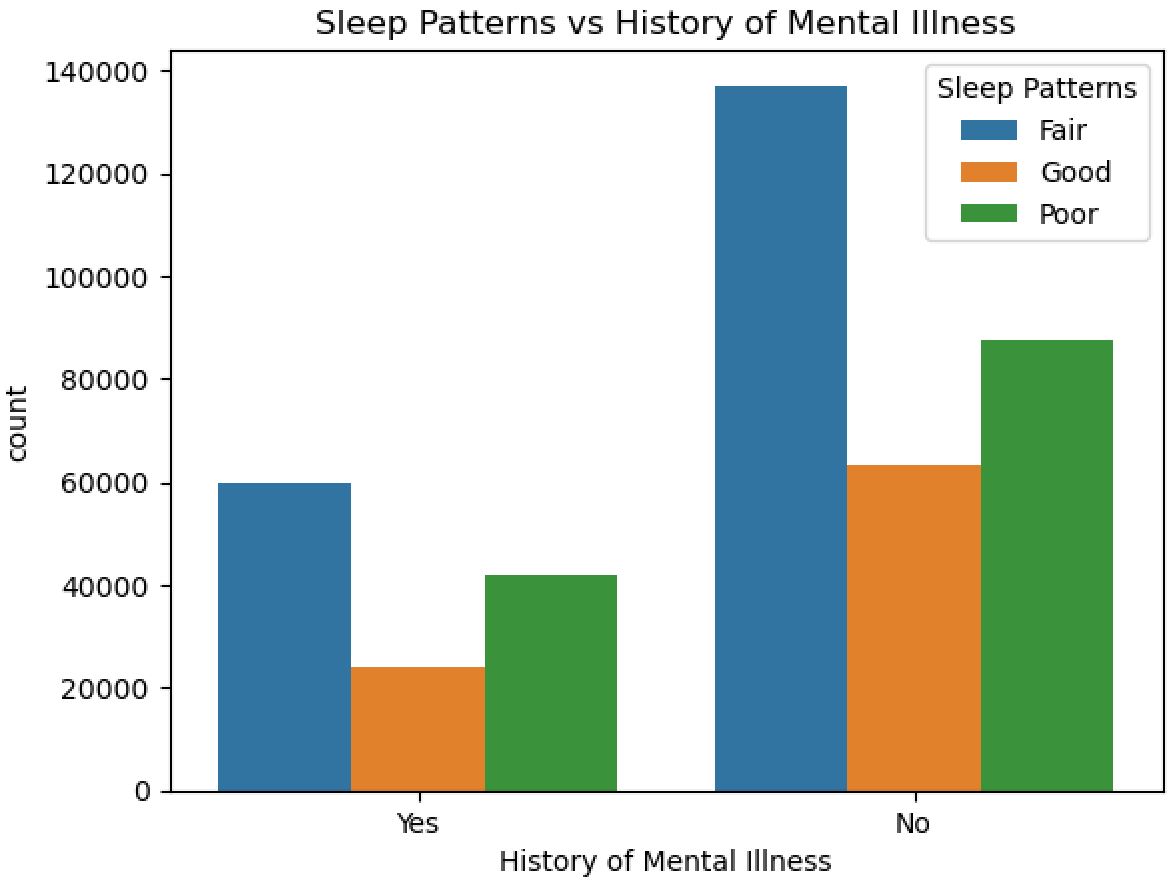

Smoking Status vs History of Mental Illness
Smoking Status
Non-smoker
Former
Current
175000
150000
125000
100000
75000
50000
25000
0
count
Yes
No
History of Mental Illness

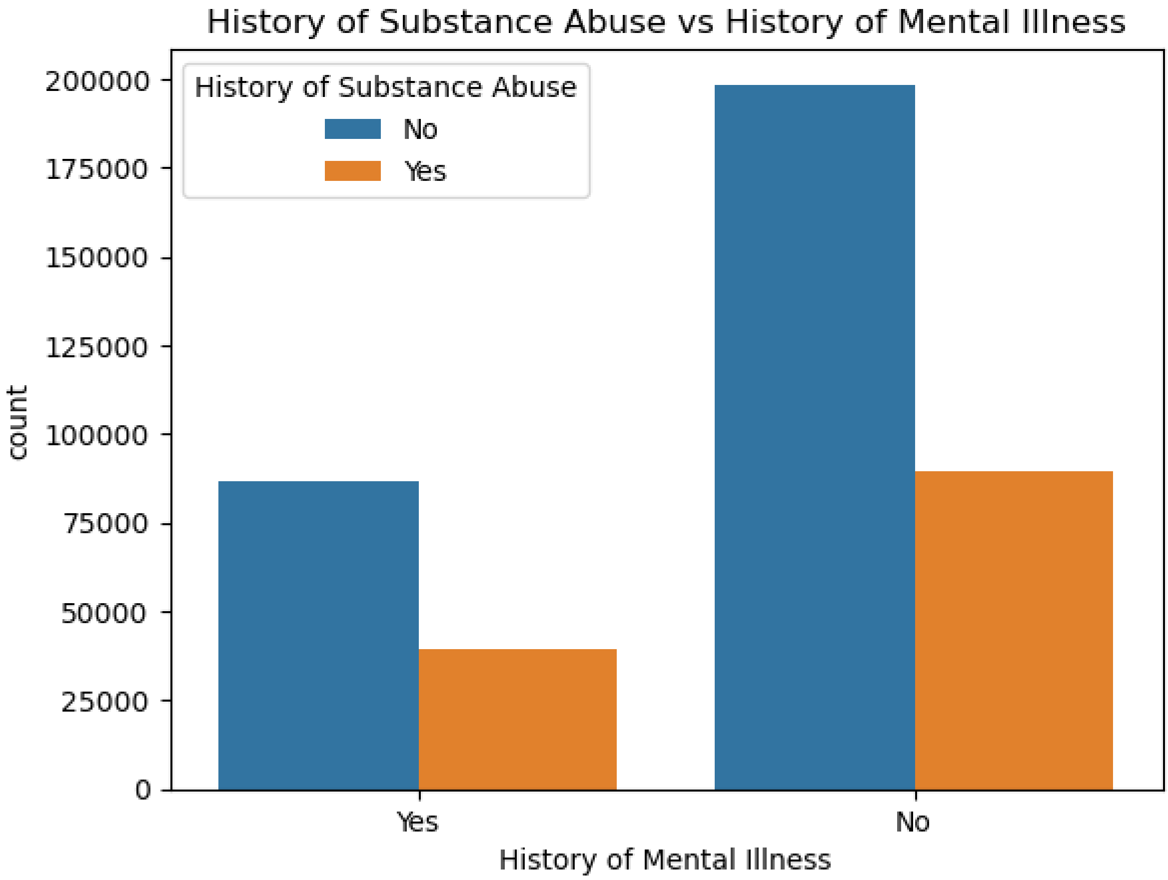

Physical Activity Level vs History of Mental Illness
Physical Activity Level
Active
Sedentary
Moderate
count
120000
100000
80000
60000
40000
20000
0
Yes
No
History of Mental Illness

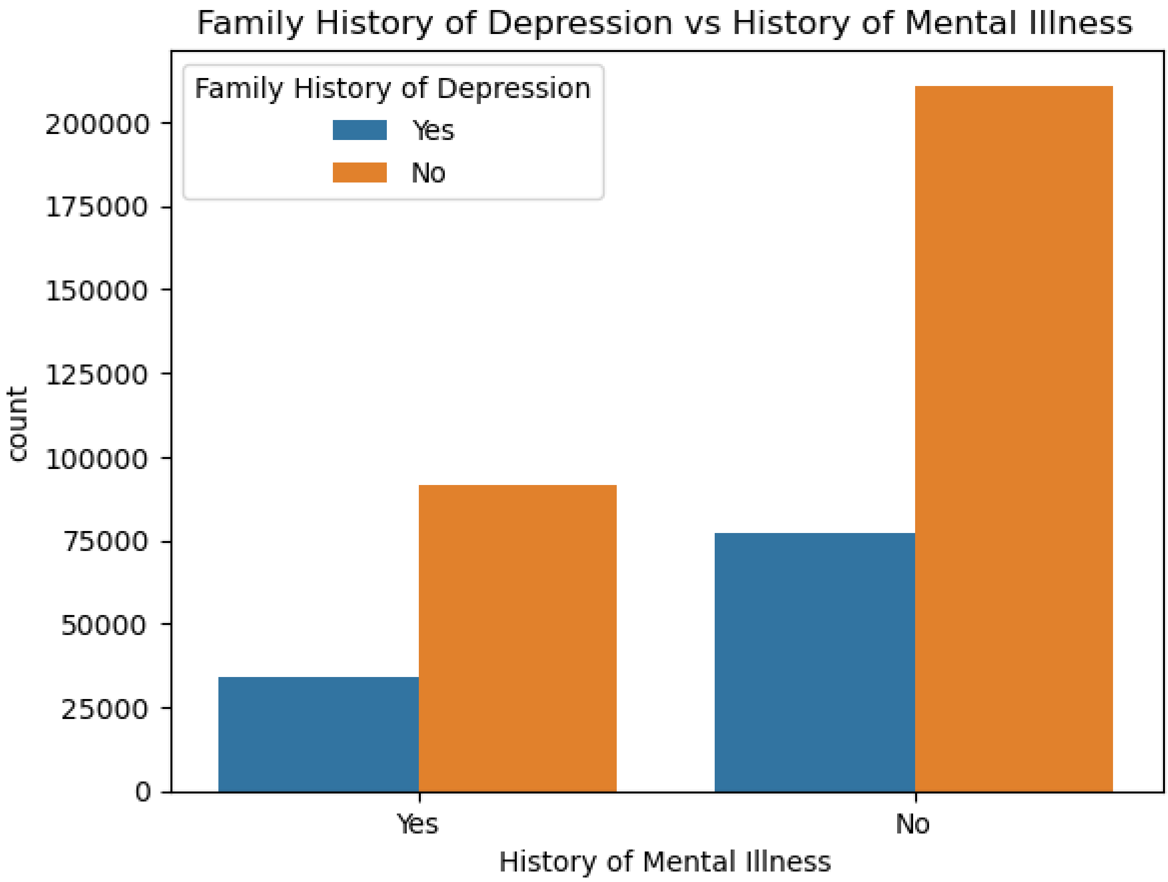

Employment Status vs History of Mental Illness
Employment Status
Unemployed
Employed
count
200000
175000
150000
125000
100000
75000
50000
25000
0
Yes
No
History of Mental Illness

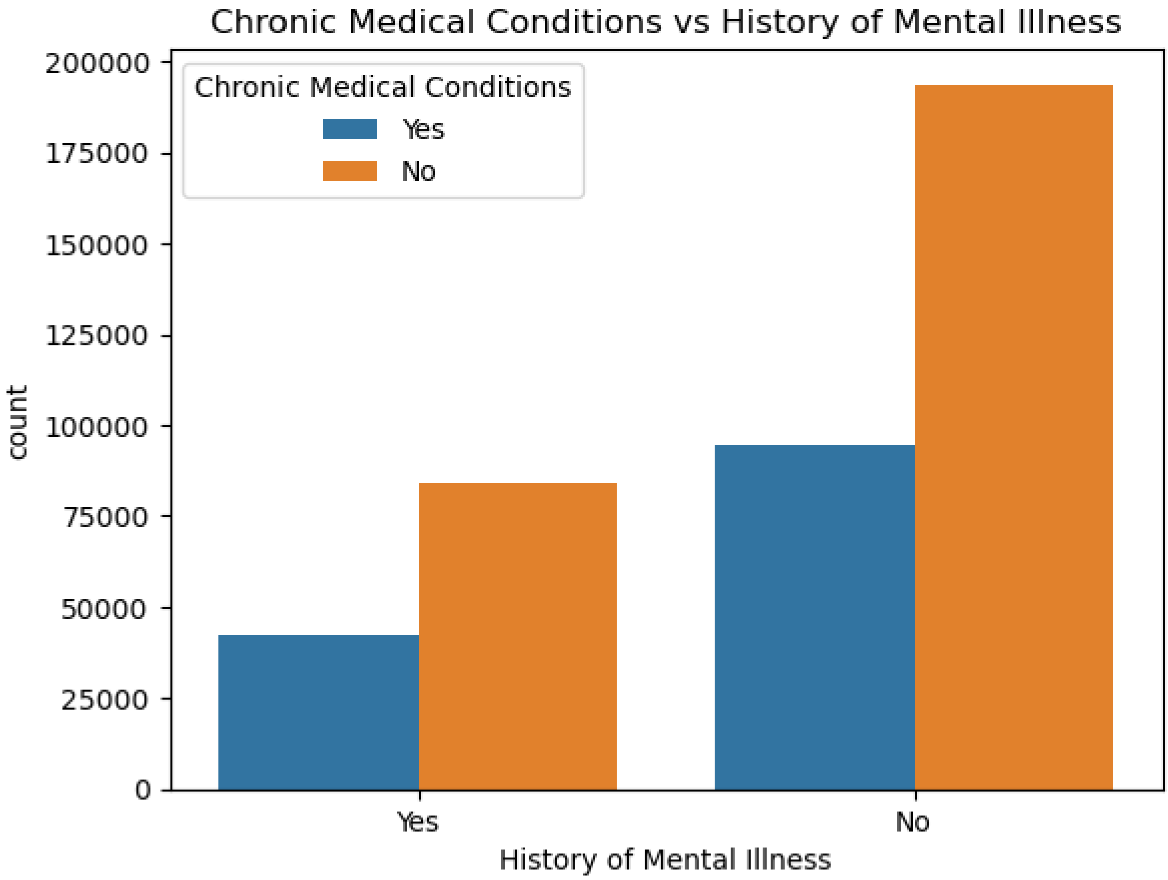

Alcohol Consumption vs History of Mental Illness
Alcohol Consumption
Moderate
High
Low
count
120000
100000
80000
60000
40000
20000
0
Yes
No
History of Mental Illness


Dietary Habits vs History of Mental Illness
Dietary Habits
Moderate
Unhealthy
Healthy
count
120000
100000
80000
60000
40000
20000
0
Yes
No
History of Mental Illness

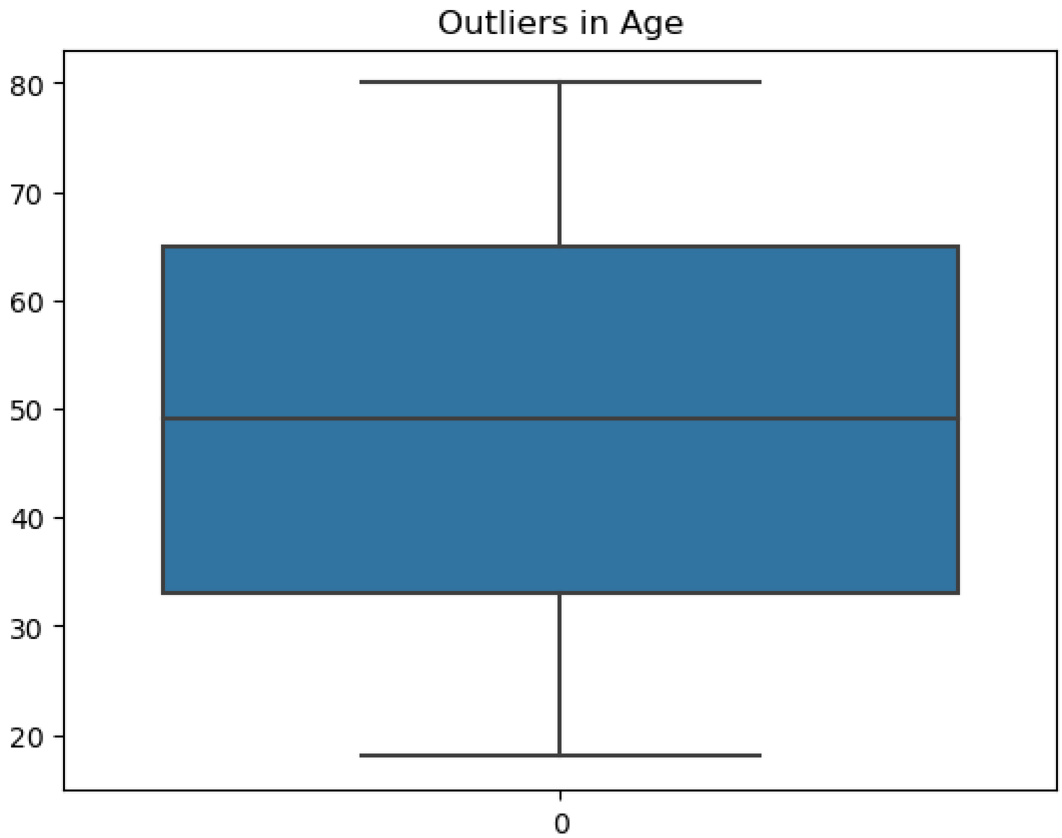

Sleep Patterns vs History of Mental Illness
Sleep Patterns
Fair
Good
Poor
count
140000
120000
100000
80000
60000
40000
20000
0
Yes
No
History of Mental Illness

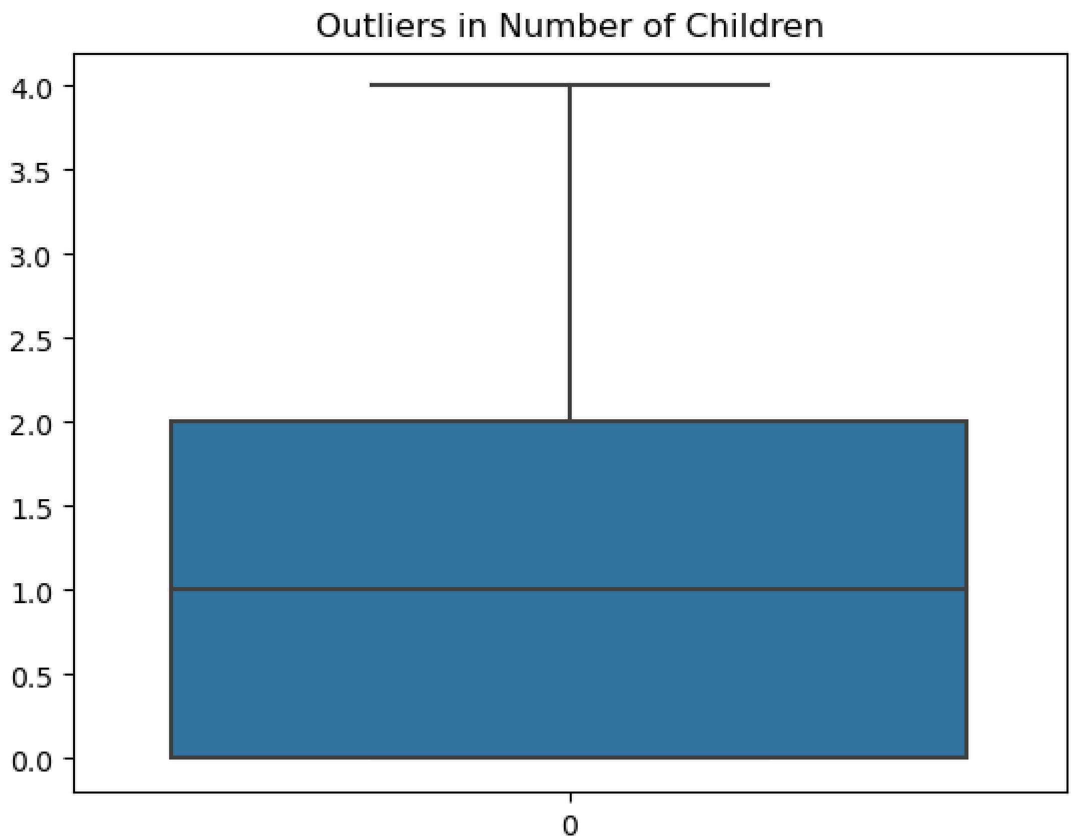

History of Substance Abuse vs History of Mental Illness
History of Substance Abuse
No
Yes
count
200000
175000
150000
125000
100000
75000
50000
25000
0
Yes
No
History of Mental Illness

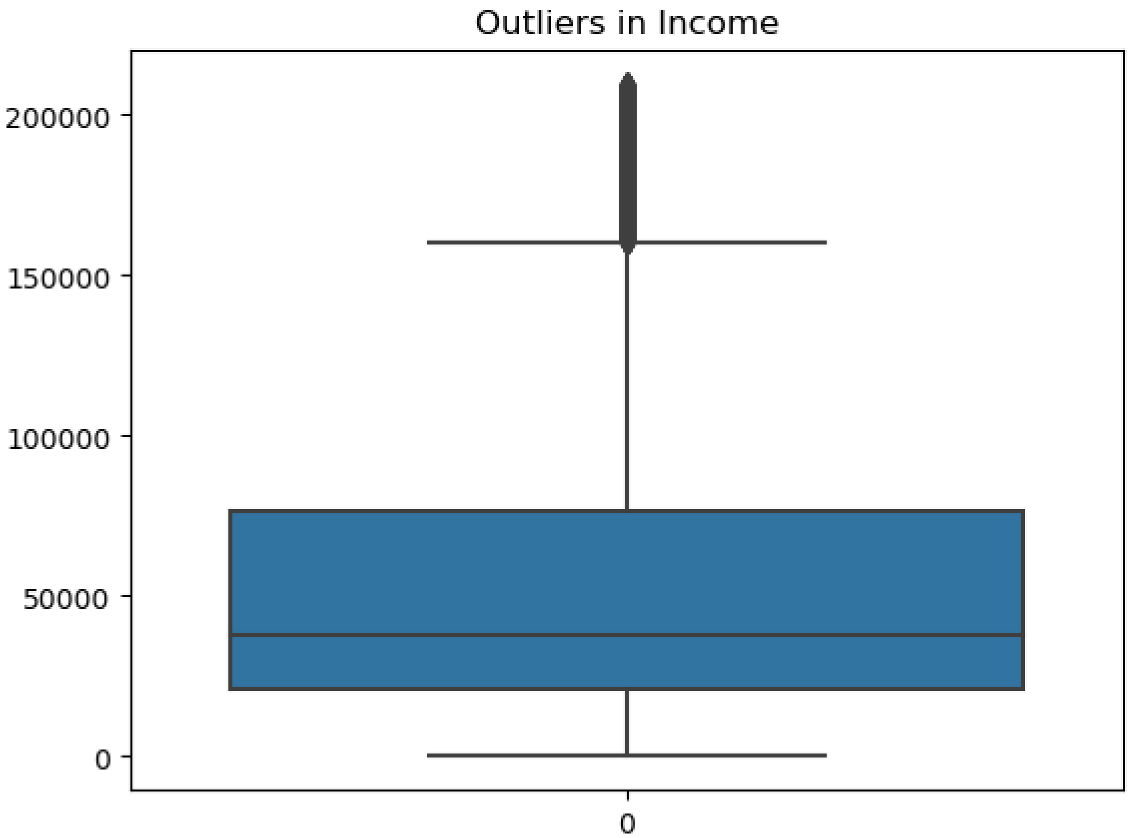



In [9]:
```
# Boxplot to identify outliers
for col in numerical_columns:
    sns.boxplot(df[col])
    plt.title(f"Outliers in {col}")
    plt.show()
```

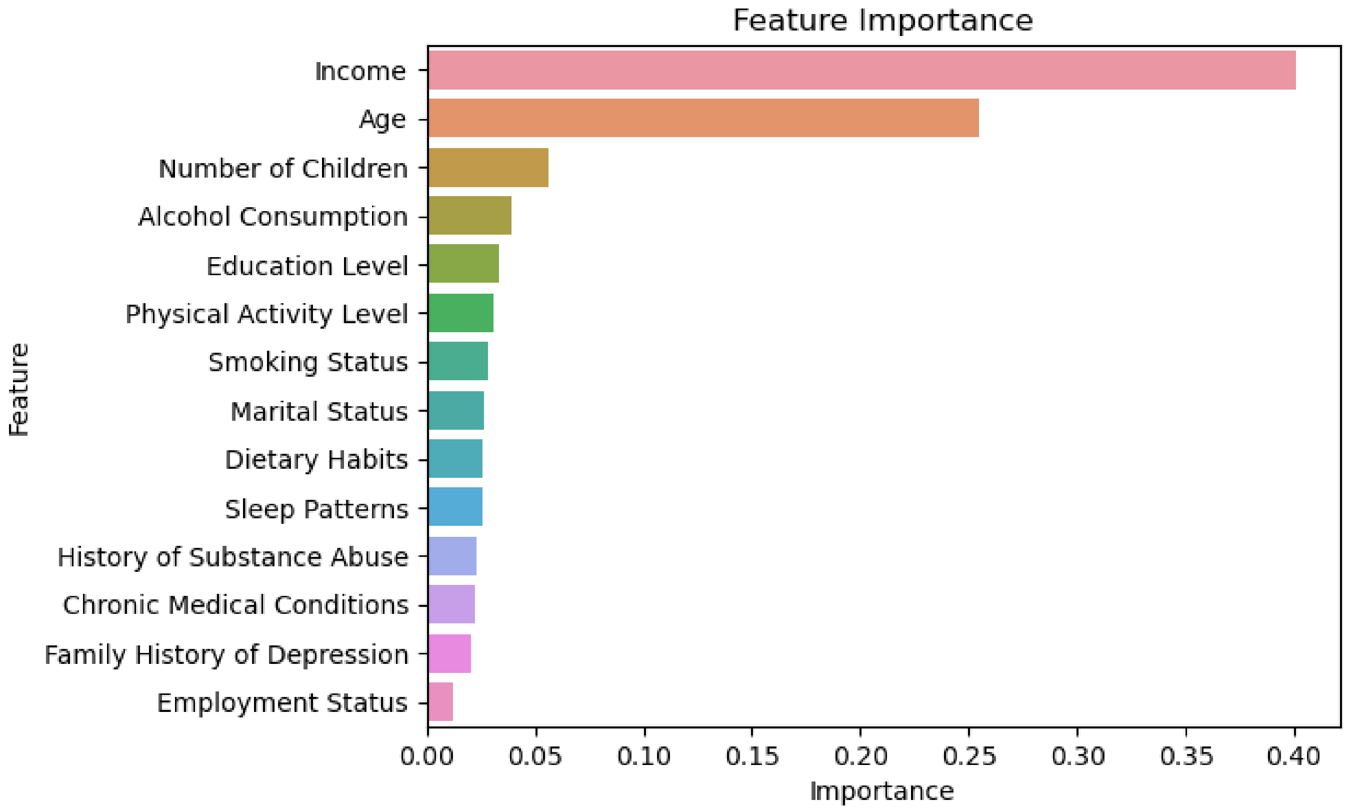

Outliers in Age
80
70
60
50
40
30
20
0


Outliers in Number of Children
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0




Outliers can distort the model, especially in sensitive algorithms like logistic regression. We might consider scaling or removing them if necessary.

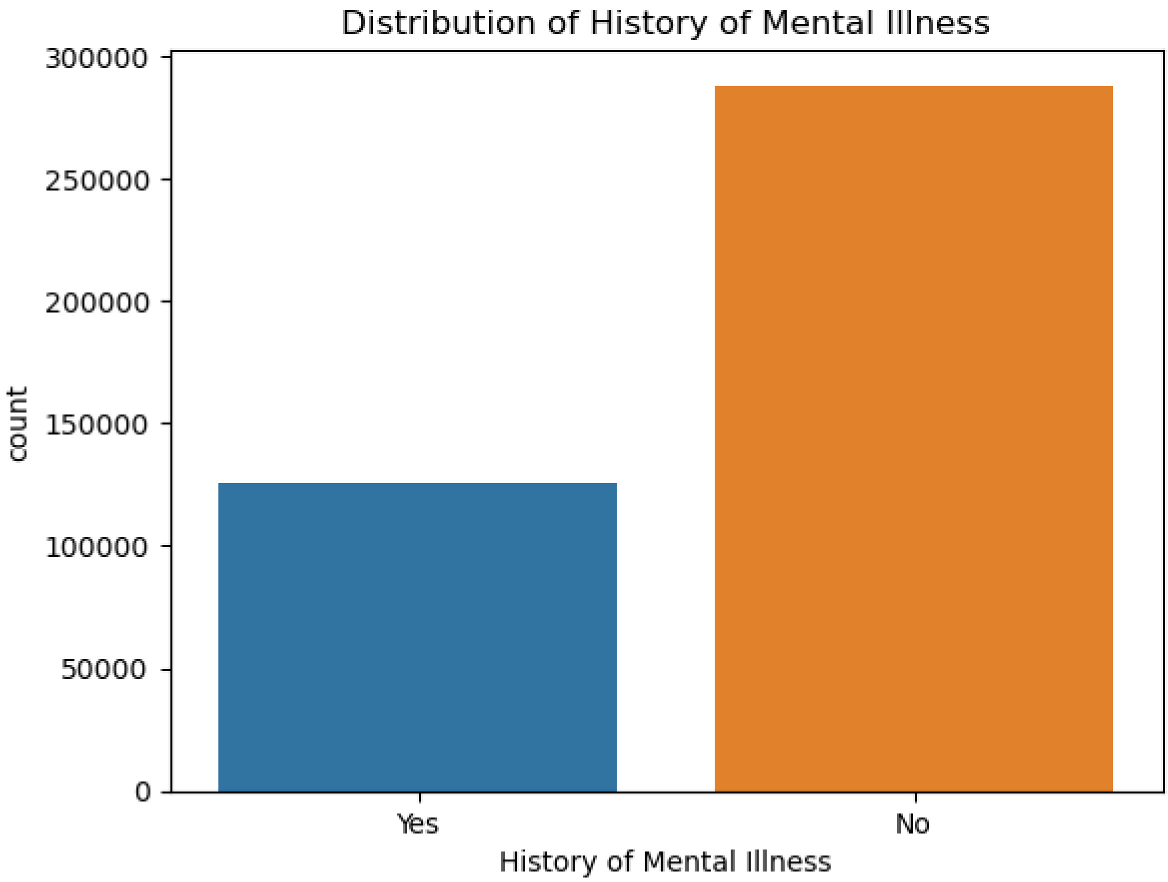
In [10]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# Encoding categorical variables for Random Forest
df_encoded = df.iloc[:, 1:].copy()

for col in categorical_columns:
    if df_encoded[col].dtype == 'object':
        df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col])

# Define X and y
X = df_encoded.drop('History of Mental Illness', axis=1)
y = LabelEncoder().fit_transform(df_encoded['History of Mental Illness'])

# Fit Random Forest to gauge feature importance
rf = RandomForestClassifier()
rf.fit(X, y)

# Plot feature importance
importance = pd.DataFrame({'Feature': X.columns,
                           'Importance': rf.feature_importances_}).sort_values(by='Im
                                                                                ascend
sns.barplot(x='Importance', y='Feature', data=importance)
plt.title("Feature Importance")
plt.show()
```



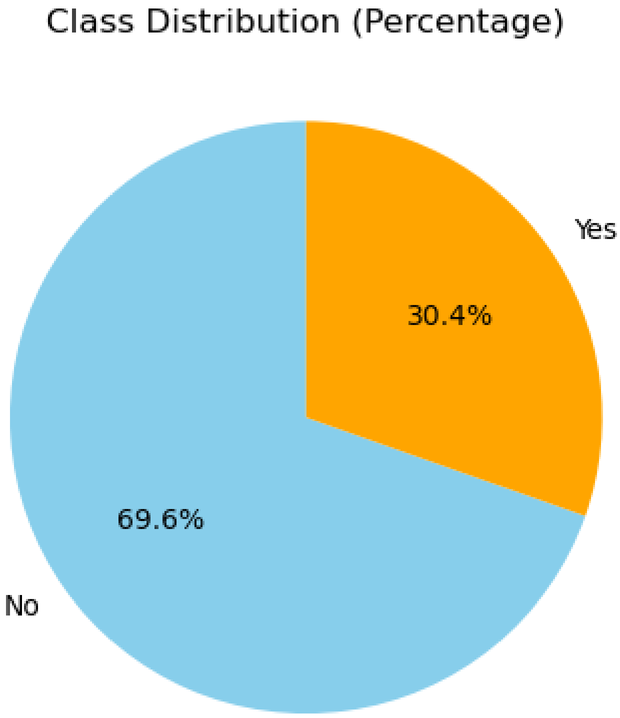

Preliminary feature importance provides insights into variables that may be predictive, guiding feature selection.

```
–Key Takeaways:

. Variables like Income, Age, Number of Children etc shows significant
relationships with mental illness.
```

In [11]:
```
# Check the distribution of the target variable
sns.countplot(x=df['History of Mental Illness'])
plt.title("Distribution of History of Mental Illness")
plt.show()

# Calculate class imbalance
class_counts = df['History of Mental Illness'].value_counts()
print("Class Imbalance:\n", class_counts)

# Visualize percentage imbalance
class_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90, colors=['skyblue', 'o
plt.title("Class Distribution (Percentage)")
plt.ylabel("")
plt.show()
```

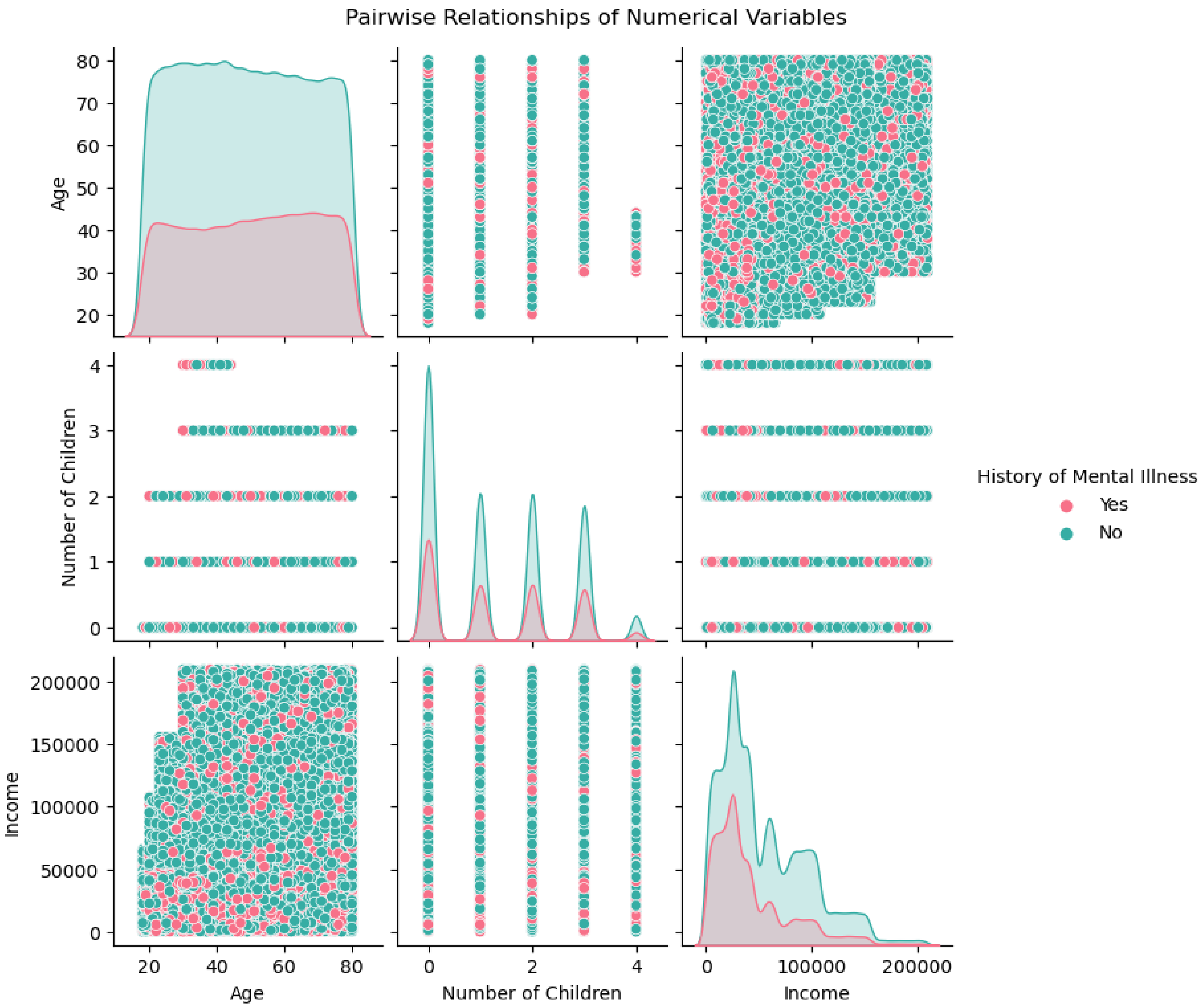



```
Class Imbalance:
 No      287943
Yes     125825
Name: History of Mental Illness, dtype: int64
```




Class imbalance in the target variable can significantly affect model performance, requiring techniques like oversampling or balanced algorithms. From this plot that the class imbalance isnt massive and we can move ahead without class rebalancing.

In [12]:
```
# Pairplot for numerical variables colored by the target variable
sns.pairplot(df, vars=numerical_columns, hue='History of Mental Illness', diag_kind='
plt.suptitle("Pairwise Relationships of Numerical Variables", y=1.02)
plt.show()
```



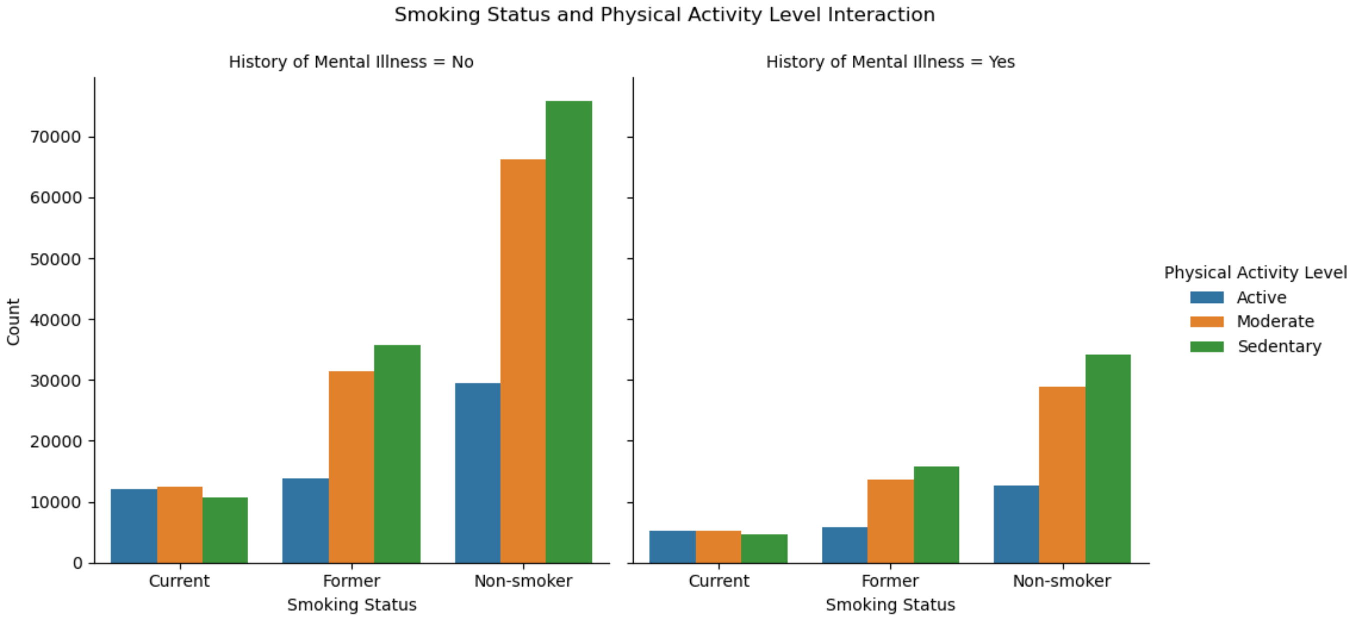

The pairplot is helpful for uncovering patterns and relationships between the variables, which can be useful when you move on to predictive modeling.

## Key insights:

- Age and Mental Illness: Younger individuals seem to have a slightly higher occurrence of mental illness.
- Number of Children and Mental Illness: There's no clear trend based on the number of children, but the number of children distribution for individuals with mental illness is quite sparse.
- Income and Mental Illness: People with a history of mental illness appear to be concentrated at the lower-income range, with fewer high-income individuals reporting mental health issues.

In [13]:

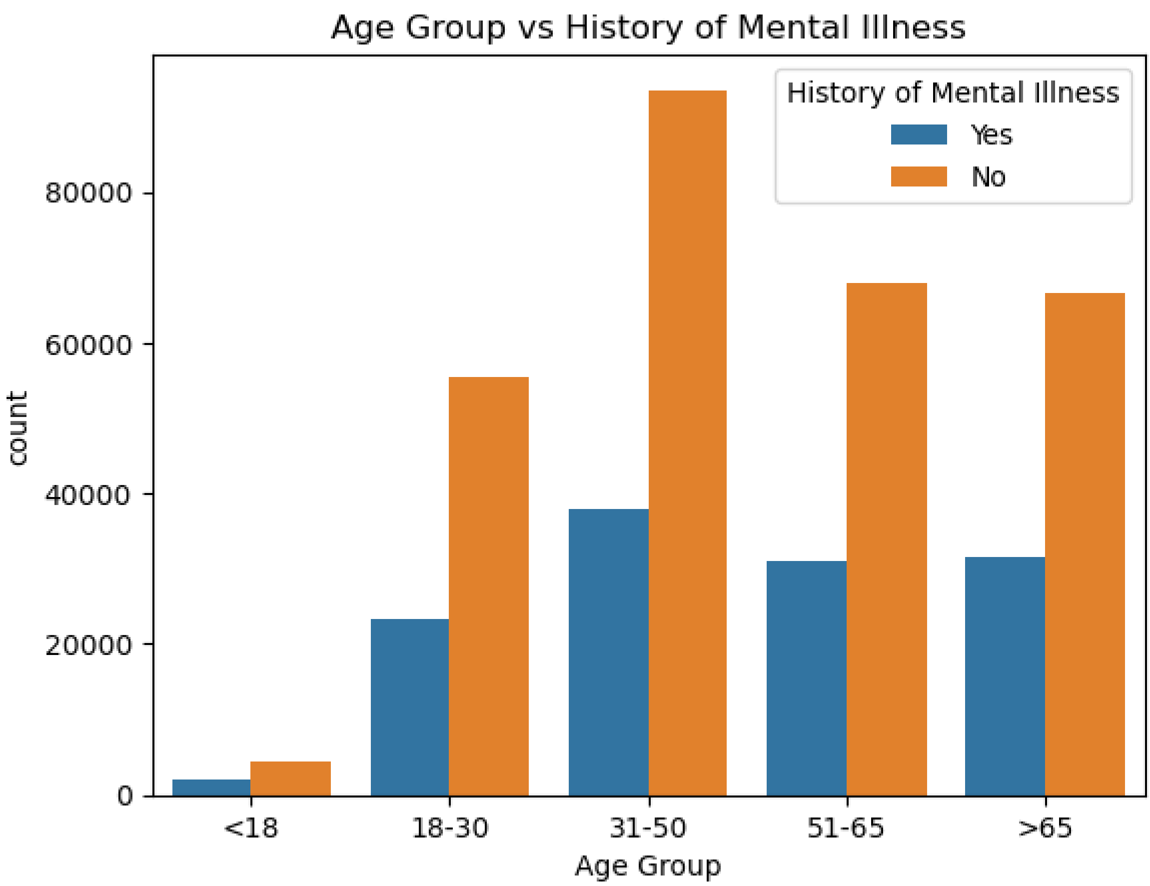
```
# Example: Smoking Status vs Physical Activity Level vs History of Mental Illness
interaction_df = df.groupby(['Smoking Status', 'Physical Activity Level', 'History of
sns.catplot(x='Smoking Status', y='Count', hue='Physical Activity Level', col='Histor
plt.suptitle("Smoking Status and Physical Activity Level Interaction", y=1.05)
plt.show()
```




Interaction effects between categorical variables can reveal deeper patterns and dependencies.

The plot suggests that people with no history of mental illness tend to have more active or moderate physical activity, especially among non-smokers. In contrast, those with a history of mental illness show a higher proportion of sedentary individuals, especially among non-smokers and former smokers, indicating a possible correlation between sedentary behavior and mental health.

In [13]:
```
# Cohort analysis by age
age_bins = [0, 18, 30, 50, 65, 100]
age_labels = ['<18', '18-30', '31-50', '51-65', '>65']
df['Age Group'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels)

# Distribution of History of Mental Illness by Age Group
sns.countplot(x='Age Group', hue='History of Mental Illness', data=df)
plt.title("Age Group vs History of Mental Illness")
plt.show()
```



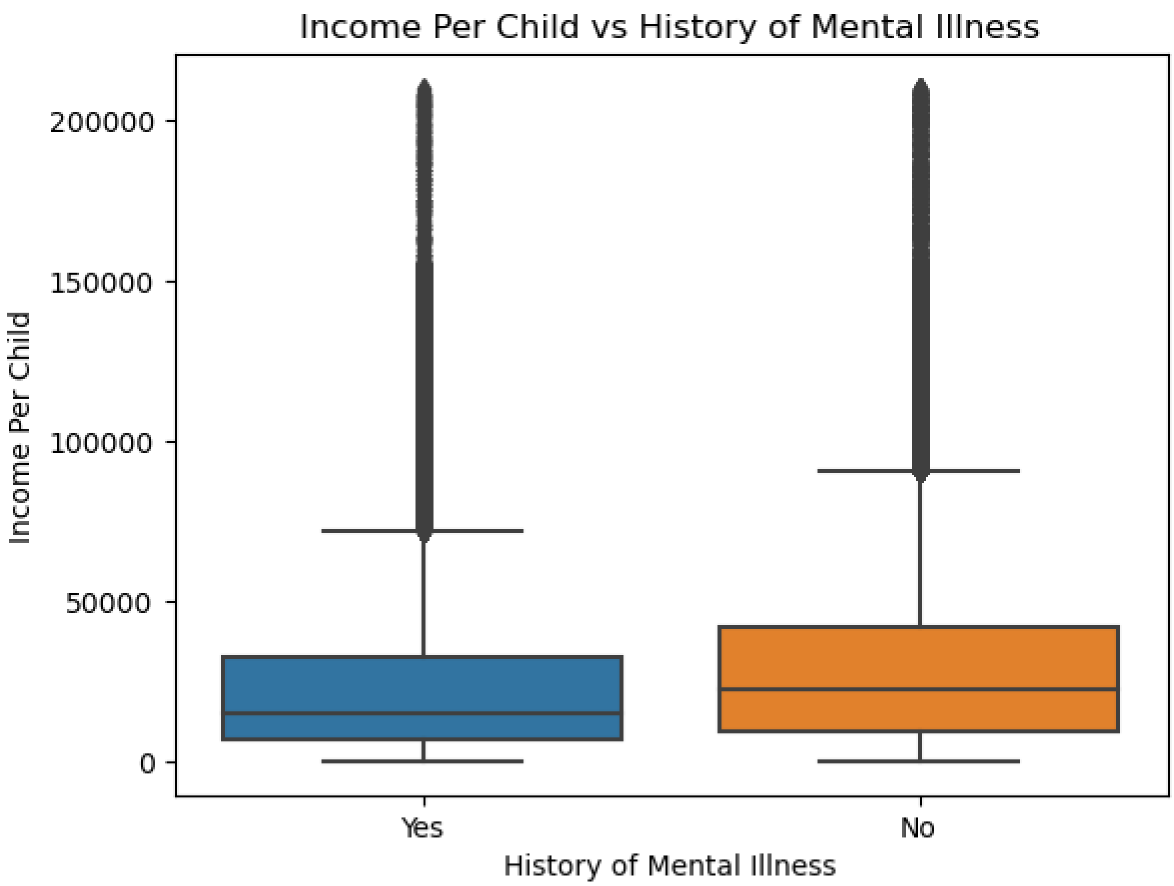
Temporal insights or age trends help us understand population-level risk factors for mental illness.

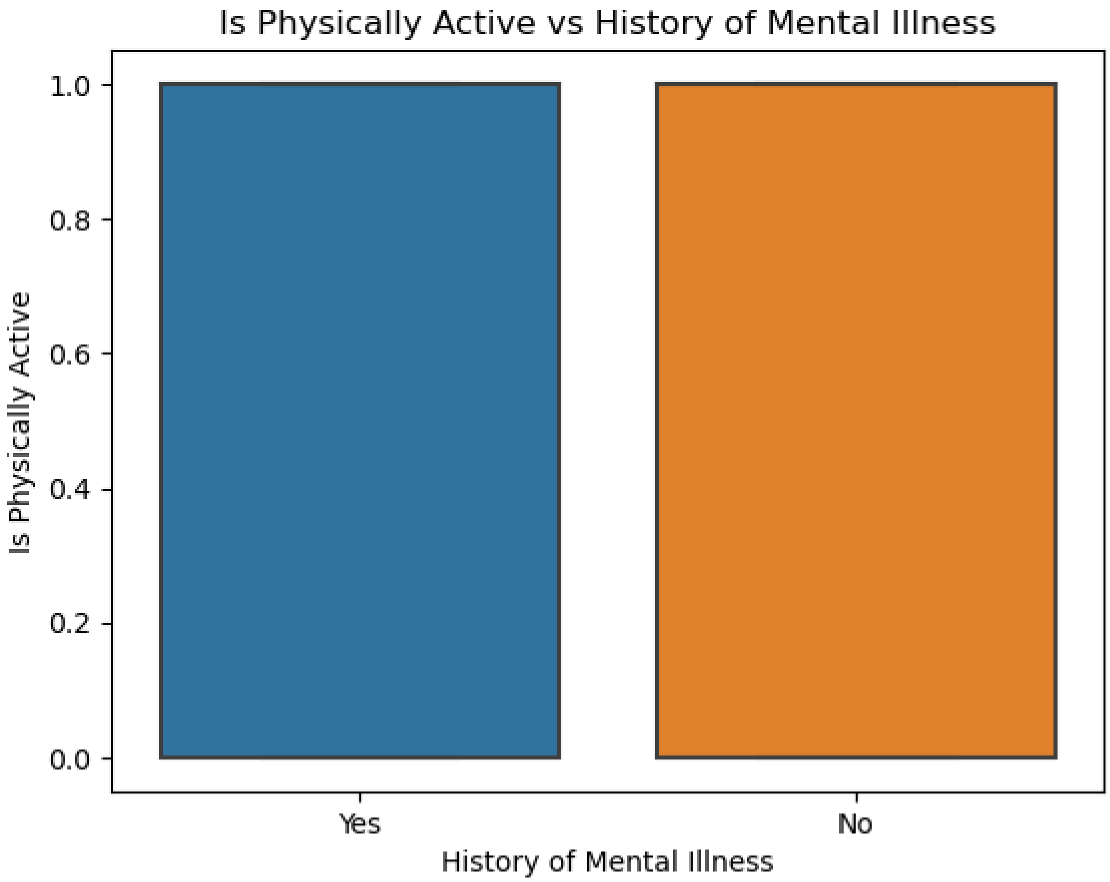
In [14]:
```
# VIF (Variance Inflation Factor) to detect multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Calculate VIF for numerical features
X = df[numerical_columns].dropna()  # Drop missing values for this calculation
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(len(X.column

print("Variance Inflation Factor (VIF):\n", vif_data)
```

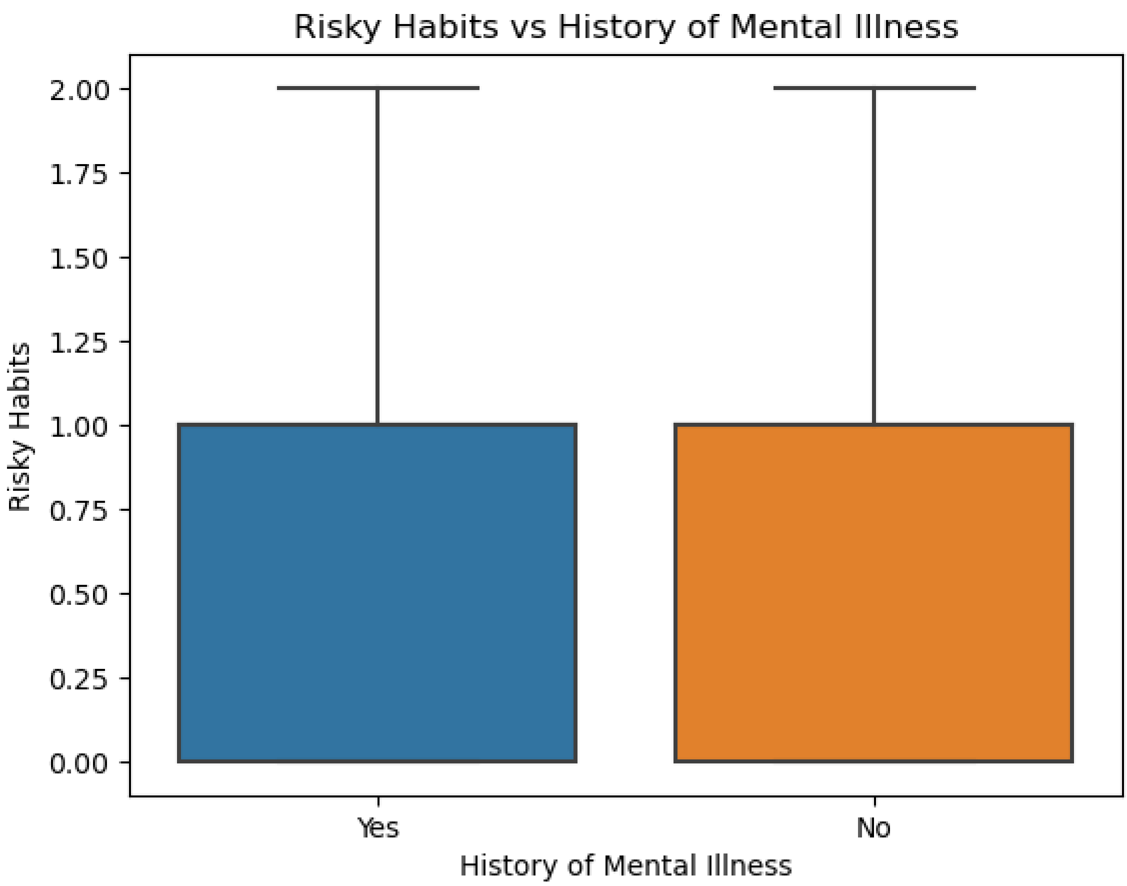
```
Variance Inflation Factor (VIF):
               Feature       VIF
0                 Age  3.076132
1  Number of Children  2.240282
2              Income  2.079540
```

Multicollinearity among numerical predictors can distort the interpretability and reliability of models like logistic regression.

In [15]:
```
# Create new features
df['Income Per Child'] = df['Income'] / (df['Number of Children'] + 1)  # Avoid divis
df['Is Physically Active'] = df['Physical Activity Level'].apply(lambda x: 1 if x in
df['Risky Habits'] = df['Smoking Status'].apply(lambda x: 1 if x != 'Non-smoker' else

# Check correlations of new features with target
new_features = ['Income Per Child', 'Is Physically Active', 'Risky Habits']
for col in new_features:
    sns.boxplot(x='History of Mental Illness', y=df[col], data=df)
    plt.title(f"{col} vs History of Mental Illness")
    plt.show()
```

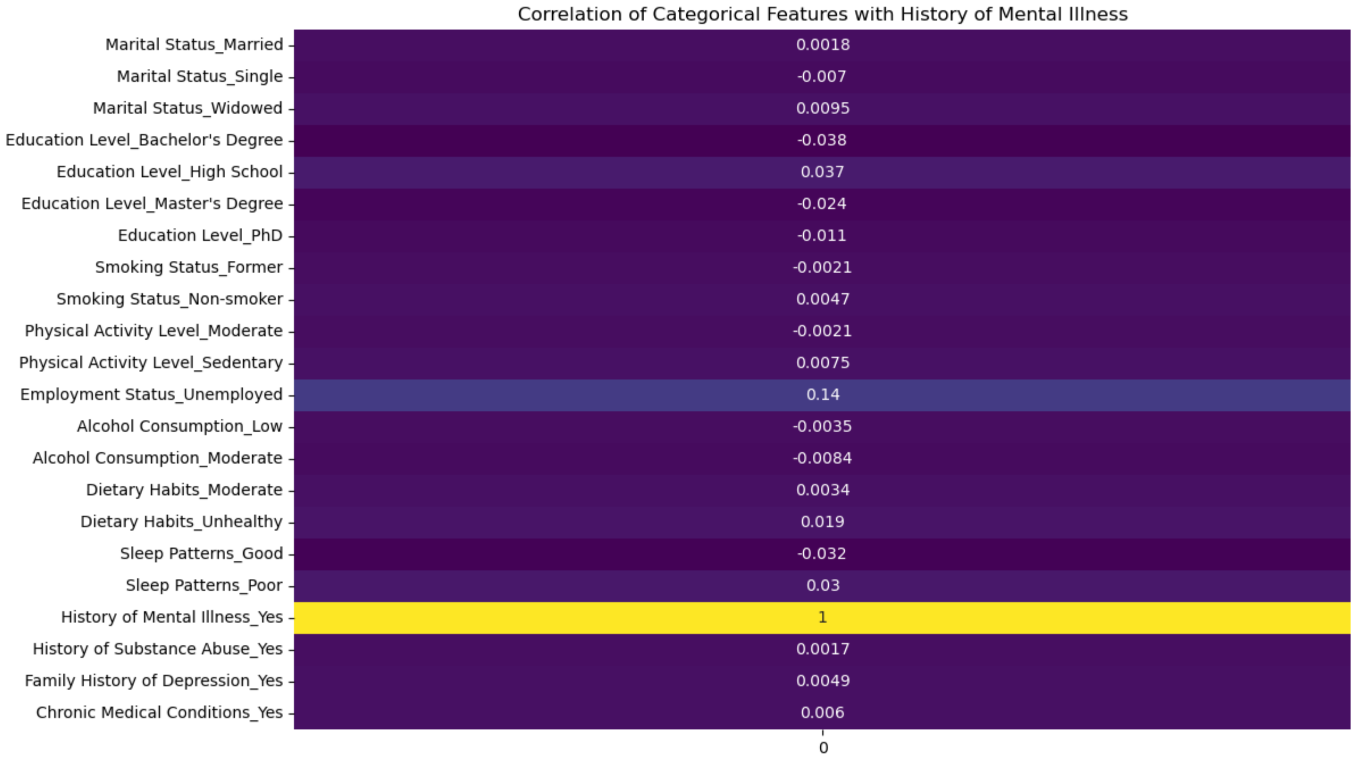





Derived features can enhance predictive power by capturing latent information or interactions.

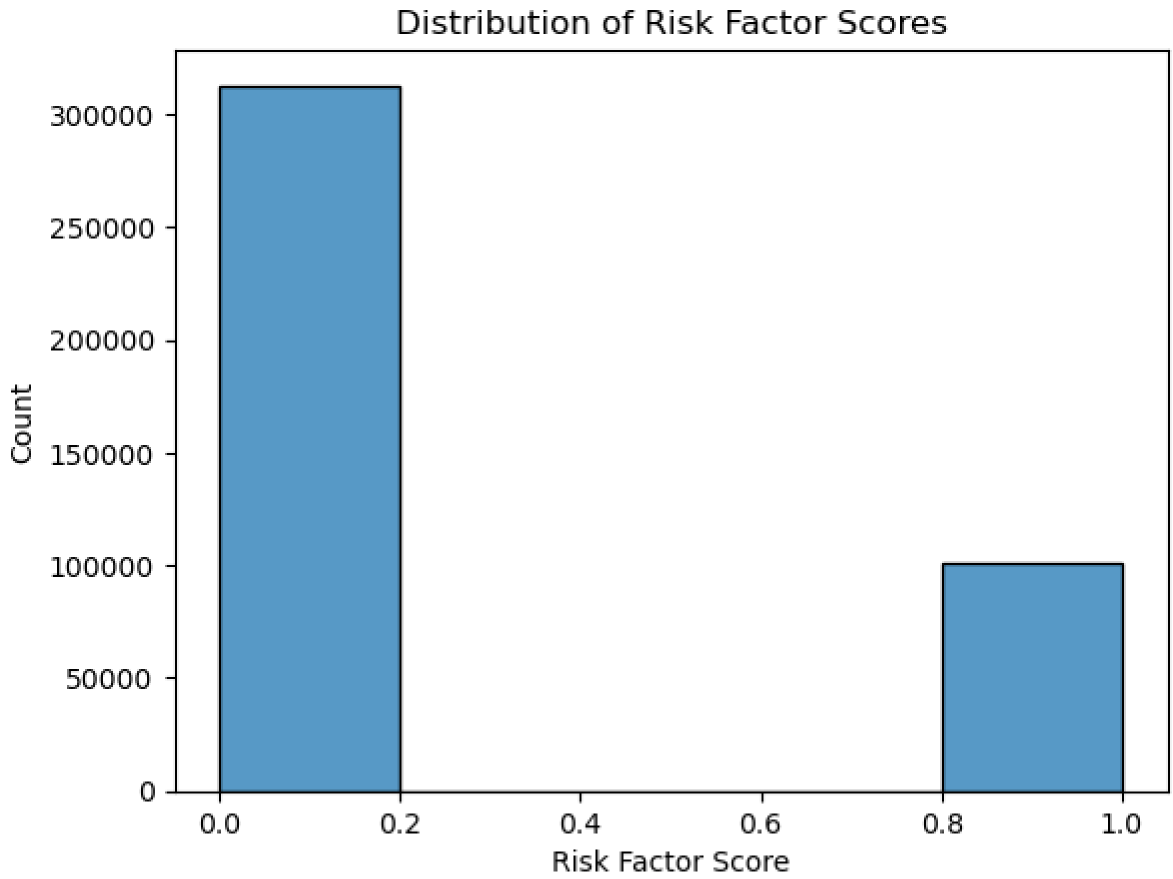
In [16]:
```
# Heatmap for categorical variables
categorical_encoded = pd.get_dummies(df[categorical_columns], drop_first=True)
correlation_categorical = categorical_encoded.corrwith(df['History of Mental Illness'

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_categorical.to_frame(), annot=True, cmap='viridis', cbar=Fals
plt.title("Correlation of Categorical Features with History of Mental Illness")
plt.show()
```

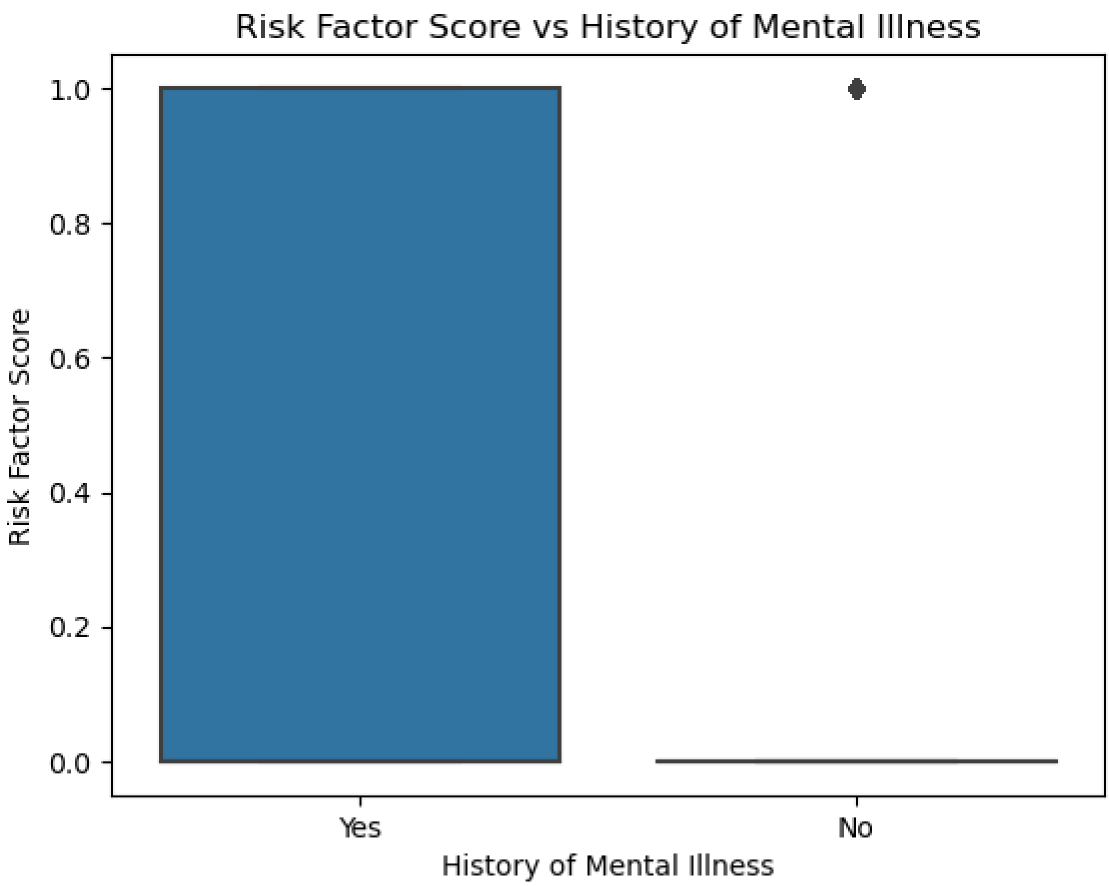



This identifies categorical predictors most strongly associated with the target variable.

Exploring spatial or regional differences can reveal environment-related mental health risks.

In [18]:
```
# Combining risky behaviors into a score
df['Risk Factor Score'] = df[['Smoking Status', 'Alcohol Consumption',
                              'Dietary Habits', 'Sleep Patterns']].apply(lambda x: (x

sns.histplot(df['Risk Factor Score'], kde=False, bins=5)
plt.title("Distribution of Risk Factor Scores")
plt.show()

# Risk factor score vs History of Mental Illness
sns.boxplot(x='History of Mental Illness', y='Risk Factor Score', data=df)
plt.title("Risk Factor Score vs History of Mental Illness")
plt.show()
```




Aggregating multiple behavioral features provides a composite risk metric, which might enhance model performance.

## Machine Learning

In [19]:
```
# Check and impute missing values
df.fillna(method='ffill', inplace=True)
```

In [20]:
```
from sklearn.preprocessing import LabelEncoder

categorical_columns = ['Marital Status', 'Education Level', 'Smoking Status', 'Physic
                       'Employment Status', 'Alcohol Consumption', 'Dietary Habits',
                       'History of Substance Abuse', 'Family History of Depression',

df_encoded = df.copy()
for col in categorical_columns:
    df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col])

# Encode the target variable
df_encoded['History of Mental Illness'] = df_encoded['History of Mental Illness'].app
```

Categorical variables are label-encoded for compatibility with machine learning models. Encoding the target variable as binary (0/1) makes it suitable for classification.

In [21]:
```
# Define features and target
X = df_encoded.drop(['Name', 'History of Mental Illness','Age Group'], axis=1)  # Exc
y = df_encoded['History of Mental Illness']
```

Variables like Name are excluded as they are irrelevant to prediction. Features like Age, Income, and Family History of Depression are included based on domain knowledge and EDA insights.

In [22]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
```

Stratified splitting ensures class distribution is maintained between train and test sets, mitigating class imbalance.

Model Selection We'll use the following models:

```
Logistic Regression: A baseline model for binary classification.
Random Forest: Handles non-linear relationships, feature importance,
and interactions well.
Gradient Boosting (XGBoost): Robust to outliers, effective for
imbalanced datasets.
Etc...
```

In [23]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, r

# Train logistic regression
log_reg = LogisticRegression(max_iter=500)
log_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_log = log_reg.predict(X_test)
print("Logistic Regression Report:\n", classification_report(y_test, y_pred_log))

# ROC-AUC Score
y_prob_log = log_reg.predict_proba(X_test)[:, 1]
print("Logistic Regression ROC-AUC:", roc_auc_score(y_test, y_prob_log))
```

```
Logistic Regression Report:
               precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.00      0.00      0.00     37748

    accuracy                           0.70    124131
   macro avg       0.35      0.50      0.41    124131
weighted avg       0.48      0.70      0.57    124131

Logistic Regression ROC-AUC: 0.5814749756779768
```

In [24]:

```
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict and evaluate
y_pred_rf = rf.predict(X_test)
print("Random Forest Report:\n", classification_report(y_test, y_pred_rf))

# Feature Importance
importances = pd.DataFrame({'Feature': X.columns, 'Importance': rf.feature_importance
print(importances)
```

```
Random Forest Report:
               precision    recall  f1-score   support

           0       0.70      0.92      0.79     86383
           1       0.36      0.11      0.17     37748

    accuracy                           0.67    124131
   macro avg       0.53      0.51      0.48    124131
weighted avg       0.60      0.67      0.60    124131

                         Feature  Importance
7                         Income    0.252429
14              Income Per Child    0.243333
0                            Age    0.181240
2                Education Level    0.033455
3             Number of Children    0.032876
9                Dietary Habits    0.029995
10               Sleep Patterns    0.029414
1                 Marital Status    0.028849
11  History of Substance Abuse    0.025368
13  Chronic Medical Conditions    0.024755
8            Alcohol Consumption    0.024650
12 Family History of Depression    0.022656
5         Physical Activity Level    0.021597
4                 Smoking Status    0.014498
16                  Risky Habits    0.012253
6              Employment Status    0.009738
15          Is Physically Active    0.008745
17             Risk Factor Score    0.004151
```

In [25]:

```
from xgboost import XGBClassifier

# Train XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb.fit(X_train, y_train)

# Predict and evaluate
y_pred_xgb = xgb.predict(X_test)
print("XGBoost Report:\n", classification_report(y_test, y_pred_xgb))
```

```
# ROC-AUC Score
y_prob_xgb = xgb.predict_proba(X_test)[:, 1]
print("XGBoost ROC-AUC:", roc_auc_score(y_test, y_prob_xgb))
```

```
XGBoost Report:
              precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.35      0.01      0.01     37748

    accuracy                           0.69    124131
   macro avg       0.52      0.50      0.42    124131
weighted avg       0.59      0.69      0.57    124131

XGBoost ROC-AUC: 0.5938558286037807
```

In [26]:

```
from imblearn.over_sampling import SMOTE

# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Train models on resampled data (example with Random Forest)
rf_smote = RandomForestClassifier(n_estimators=100, random_state=42)
rf_smote.fit(X_resampled, y_resampled)

# Predict and evaluate
y_pred_rf_smote = rf_smote.predict(X_test)
print("Random Forest with SMOTE Report:\n", classification_report(y_test, y_pred_rf_s
```

```
Random Forest with SMOTE Report:
              precision    recall  f1-score   support

           0       0.71      0.84      0.77     86383
           1       0.35      0.20      0.25     37748

    accuracy                           0.64    124131
   macro avg       0.53      0.52      0.51    124131
weighted avg       0.60      0.64      0.61    124131
```

In [27]:

```
from sklearn.neighbors import KNeighborsClassifier

# Train KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict and evaluate
y_pred_knn = knn.predict(X_test)
y_prob_knn = knn.predict_proba(X_test)[:, 1]

print("KNN Report:\n", classification_report(y_test, y_pred_knn))
print("KNN ROC-AUC:", roc_auc_score(y_test, y_prob_knn))
```

```
KNN Report:
              precision    recall  f1-score   support

           0       0.70      0.83      0.76     86383
           1       0.34      0.20      0.25     37748

    accuracy                           0.64    124131
   macro avg       0.52      0.52      0.51    124131
weighted avg       0.59      0.64      0.61    124131

KNN ROC-AUC: 0.5309372293869056
```

In [28]:

```
from sklearn.naive_bayes import GaussianNB
```

```python
# Train Naive Bayes
nb = GaussianNB()
nb.fit(X_train, y_train)

# Predict and evaluate
y_pred_nb = nb.predict(X_test)
y_prob_nb = nb.predict_proba(X_test)[:, 1]

print("Naive Bayes Report:\n", classification_report(y_test, y_pred_nb))
print("Naive Bayes ROC-AUC:", roc_auc_score(y_test, y_prob_nb))
```

```
Naive Bayes Report:
               precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.00      0.00      0.00     37748

    accuracy                           0.70    124131
   macro avg       0.35      0.50      0.41    124131
weighted avg       0.48      0.70      0.57    124131

Naive Bayes ROC-AUC: 0.5908258741500213
```

In [29]:
```python
from sklearn.neural_network import MLPClassifier

# Train Neural Network
mlp = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=500, random_state=42)
mlp.fit(X_train, y_train)

# Predict and evaluate
y_pred_mlp = mlp.predict(X_test)
y_prob_mlp = mlp.predict_proba(X_test)[:, 1]

print("MLP Report:\n", classification_report(y_test, y_pred_mlp))
print("MLP ROC-AUC:", roc_auc_score(y_test, y_prob_mlp))
```

```
MLP Report:
               precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.00      0.00      0.00     37748

    accuracy                           0.70    124131
   macro avg       0.35      0.50      0.41    124131
weighted avg       0.48      0.70      0.57    124131

MLP ROC-AUC: 0.5
```

In [30]:
```python
from sklearn.ensemble import AdaBoostClassifier

# Train AdaBoost
ada = AdaBoostClassifier(n_estimators=50, random_state=42)
ada.fit(X_train, y_train)

# Predict and evaluate
y_pred_ada = ada.predict(X_test)
y_prob_ada = ada.predict_proba(X_test)[:, 1]

print("AdaBoost Report:\n", classification_report(y_test, y_pred_ada))
print("AdaBoost ROC-AUC:", roc_auc_score(y_test, y_prob_ada))
```

```
AdaBoost Report:
               precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.00      0.00      0.00     37748

    accuracy                           0.70    124131
   macro avg       0.35      0.50      0.41    124131
weighted avg       0.48      0.70      0.57    124131

AdaBoost ROC-AUC: 0.5935171172701406
```

In [31]:
```python
from sklearn.ensemble import GradientBoostingClassifier

# Train GBM
gbm = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, ra
gbm.fit(X_train, y_train)

# Predict and evaluate
y_pred_gbm = gbm.predict(X_test)
y_prob_gbm = gbm.predict_proba(X_test)[:, 1]

print("GBM Report:\n", classification_report(y_test, y_pred_gbm))
print("GBM ROC-AUC:", roc_auc_score(y_test, y_prob_gbm))
```

```
GBM Report:
               precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.20      0.00      0.00     37748

    accuracy                           0.70    124131
   macro avg       0.45      0.50      0.41    124131
weighted avg       0.55      0.70      0.57    124131

GBM ROC-AUC: 0.5981765519598958
```

In [32]:
```python
from lightgbm import LGBMClassifier

# Train LightGBM
lgbm = LGBMClassifier(n_estimators=100, learning_rate=0.1, max_depth=-1, random_state
lgbm.fit(X_train, y_train)

# Predict and evaluate
y_pred_lgbm = lgbm.predict(X_test)
y_prob_lgbm = lgbm.predict_proba(X_test)[:, 1]

print("LightGBM Report:\n", classification_report(y_test, y_pred_lgbm))
print("LightGBM ROC-AUC:", roc_auc_score(y_test, y_prob_lgbm))
```

```
LightGBM Report:
               precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.33      0.00      0.00     37748

    accuracy                           0.70    124131
   macro avg       0.51      0.50      0.41    124131
weighted avg       0.59      0.70      0.57    124131

LightGBM ROC-AUC: 0.5965538831195312
```

In [33]:
```python
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

# Define base models
```

```python
estimators = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_stat
]

# Define Stacking Classifier
stacking = StackingClassifier(estimators=estimators, final_estimator=LogisticRegressi
stacking.fit(X_train, y_train)

# Predict and evaluate
y_pred_stack = stacking.predict(X_test)
y_prob_stack = stacking.predict_proba(X_test)[:, 1]

print("Stacking Report:\n", classification_report(y_test, y_pred_stack))
print("Stacking ROC-AUC:", roc_auc_score(y_test, y_prob_stack))
```

```
Stacking Report:
               precision    recall  f1-score   support

           0       0.70      1.00      0.82     86383
           1       0.38      0.00      0.01     37748

    accuracy                           0.70    124131
   macro avg       0.54      0.50      0.41    124131
weighted avg       0.60      0.70      0.57    124131

Stacking ROC-AUC: 0.5940969452009496
```

In [34]:

```python
from sklearn.ensemble import BaggingClassifier

# Train Bagging Classifier
bagging = BaggingClassifier(n_estimators=50, random_state=42)
bagging.fit(X_train, y_train)

# Predict and evaluate
y_pred_bagging = bagging.predict(X_test)
y_prob_bagging = bagging.predict_proba(X_test)[:, 1]

print("Bagging Report:\n", classification_report(y_test, y_pred_bagging))
print("Bagging ROC-AUC:", roc_auc_score(y_test, y_prob_bagging))
```

```
Bagging Report:
               precision    recall  f1-score   support

           0       0.70      0.91      0.79     86383
           1       0.36      0.11      0.17     37748

    accuracy                           0.67    124131
   macro avg       0.53      0.51      0.48    124131
weighted avg       0.60      0.67      0.60    124131

Bagging ROC-AUC: 0.5536048471626477
```

In [35]:

```python
# Collect metrics
models = ['GBM', 'LightGBM', 'Stacking', 'Bagging','Logistic Regression', 'XGBoost',
roc_auc_scores = [roc_auc_score(y_test, y_prob_gbm),
                  roc_auc_score(y_test, y_prob_lgbm),
                  roc_auc_score(y_test, y_prob_stack),
                  roc_auc_score(y_test, y_prob_bagging),
                roc_auc_score(y_test, y_prob_log),
                  roc_auc_score(y_test, y_prob_xgb),
                  roc_auc_score(y_test, y_prob_knn),
                  roc_auc_score(y_test, y_prob_nb),
                  roc_auc_score(y_test, y_prob_mlp),
                  roc_auc_score(y_test, y_prob_ada)]
```

```
results_df = pd.DataFrame({'Model': models, 'ROC-AUC': roc_auc_scores}).sort_values(b
results_df.reset_index(drop=True)
```

Out[35]:

| | Model | ROC-AUC |
|---|---|---|
| 0 | GBM | 0.598177 |
| 1 | LightGBM | 0.596554 |
| 2 | Stacking | 0.594097 |
| 3 | XGBoost | 0.593856 |
| 4 | AdaBoost | 0.593517 |
| 5 | Naive Bayes | 0.590826 |
| 6 | Logistic Regression | 0.581475 |
| 7 | Bagging | 0.553605 |
| 8 | KNN | 0.530937 |
| 9 | MLP | 0.500000 |

The model performance, as indicated by the ROC-AUC scores, reveals that none of the models demonstrate particularly strong predictive power. The top-performing model, **GBM**, achieves a modest ROC-AUC of 0.598, which is barely above random guessing (0.5), suggesting that it is not capturing the underlying patterns in the data effectively. Other gradient-boosting models like **LightGBM** and **XGBoost** also show similar performance, with scores around 0.59. **Logistic Regression** and **Naive Bayes** fare slightly worse, further indicating a lack of strong separation between the positive and negative classes. Models like **KNN**, **Bagging**, and **MLP** have even lower performance, with **MLP** performing no better than random chance. This suggests that the features used for prediction may not be providing enough information, or the data might require further preprocessing, feature engineering, or hyperparameter tuning to improve model performance. Additionally, considering the lackluster results across multiple models, it's possible that the data is synthetic or lacks sufficient real-world complexity, which could be contributing to the poor model performance. This would explain why the models are not able to find meaningful patterns or make accurate predictions.

Next Steps: I would like to check if the results improve if we generate some synthetic data so that the class imbalance can be mitigated.

## Using SMOTE for all algorithms:

In [36]:

```
from imblearn.over_sampling import SMOTE
from tqdm import tqdm

# Load dataset
df = pd.read_csv('depression_data.csv')

# Identify categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns

# Encode categorical variables using LabelEncoder
df_encoded = df.copy()
label_encoder = LabelEncoder()
for col in categorical_columns:
    df_encoded[col] = label_encoder.fit_transform(df_encoded[col])

# Split features and target
X = df_encoded.drop('History of Mental Illness', axis=1)  # Features
```

```
y = df_encoded['History of Mental Illness']  # Target variable

# Split dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Handle class imbalance with SMOTE
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# Check class distribution after SMOTE
print("Class distribution after SMOTE:")
print(pd.Series(y_train_balanced).value_counts())

# Define the models
models = [
    GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, rand
    LGBMClassifier(n_estimators=100, learning_rate=0.1, max_depth=-1, random_state=42
    StackingClassifier(
        estimators=[
            ('rf', GradientBoostingClassifier(n_estimators=50, random_state=42)),
            ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss', ran
        ],
        final_estimator=LogisticRegression(max_iter=1000, random_state=42)
    ),  # Stacking
    XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    AdaBoostClassifier(n_estimators=100, random_state=42),  # AdaBoost
    GaussianNB(),  # Naive Bayes
    LogisticRegression(max_iter=1000, random_state=42, class_weight='balanced'),  # L
    BaggingClassifier(n_estimators=50, random_state=42),  # Bagging
    KNeighborsClassifier(n_neighbors=5),  # KNN
    MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=500, random_state=42) # MLP
]

# Function to train and evaluate models
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test):
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Evaluate the model
    report = classification_report(y_test, y_pred, output_dict=True)
    auc_score = roc_auc_score(y_test, y_prob)
    return report, auc_score, y_prob

# Initialize a dictionary to store results
results = {}
y_probs = {}

# Train and evaluate each model using tqdm for progress tracking
for model in tqdm(models, desc="Training and Evaluating Models"):
    model_name = model.__class__.__name__
    report, auc_score, y_prob = train_and_evaluate_model(model, X_train_balanced, X_t
    results[model_name] = {"classification_report": report, "roc_auc": auc_score}
    y_probs[model_name] = y_prob

# Create a list of models and their corresponding ROC-AUC scores
models = ['GBM', 'LightGBM', 'Stacking', 'Bagging','Logistic Regression', 'XGBoost',
roc_auc_scores = [
    roc_auc_score(y_test, y_probs['GradientBoostingClassifier']),
    roc_auc_score(y_test, y_probs['LGBMClassifier']),
    roc_auc_score(y_test, y_probs['StackingClassifier']),
    roc_auc_score(y_test, y_probs['BaggingClassifier']),
```

```
    roc_auc_score(y_test, y_probs['LogisticRegression']),
    roc_auc_score(y_test, y_probs['XGBClassifier']),
    roc_auc_score(y_test, y_probs['KNeighborsClassifier']),
    roc_auc_score(y_test, y_probs['GaussianNB']),
    roc_auc_score(y_test, y_probs['MLPClassifier']),
    roc_auc_score(y_test, y_probs['AdaBoostClassifier'])
]

# Create a DataFrame to show ROC-AUC scores for each model
results_df = pd.DataFrame({'Model': models, 'ROC-AUC': roc_auc_scores}).sort_values(b
results_df.reset_index(drop=True, inplace=True)

# Print results
results_df
```

```
Class distribution after SMOTE:
0    230354
1    230354
Name: History of Mental Illness, dtype: int64
Training and Evaluating Models: 100%|██████████| 10/10 [10:41<00:00, 64.11s/it]
```

Out[36]:

| | Model | ROC-AUC |
|---|---|---|
| 0 | Logistic Regression | 0.586976 |
| 1 | Naive Bayes | 0.579277 |
| 2 | LightGBM | 0.568819 |
| 3 | XGBoost | 0.564051 |
| 4 | Stacking | 0.561375 |
| 5 | GBM | 0.554880 |
| 6 | Bagging | 0.544582 |
| 7 | AdaBoost | 0.539856 |
| 8 | KNN | 0.527026 |
| 9 | MLP | 0.500009 |

# The results are not much different.

In [ ]:

Next Steps: Now I would like to check if the results improve if we undersample the data and mitigate the class imbalance.

## Using undersampling for all algorithms:

In [38]:

```
from imblearn.under_sampling import RandomUnderSampler

# Load the dataset
df = pd.read_csv('depression_data.csv')

# Split features and target
X = df.drop('History of Mental Illness', axis=1)  # Features
y = df['History of Mental Illness']               # Target variable

# Encode categorical variables
X_encoded = X.copy()
categorical_columns = X.select_dtypes(include=['object']).columns
```

```python
for col in categorical_columns:
    X_encoded[col] = LabelEncoder().fit_transform(X_encoded[col])

# Encode the target variable (History of Mental Illness)
y_encoded = LabelEncoder().fit_transform(y)

# Split dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded, test_size=0

# Handle class imbalance with Undersampling
undersample = RandomUnderSampler(random_state=42)
X_train_undersampled, y_train_undersampled = undersample.fit_resample(X_train, y_trai

# Check class distribution after undersampling
print("Class distribution after Undersampling:")
print(pd.Series(y_train_undersampled).value_counts())

# Define the models
models = {
    'GradientBoostingClassifier': GradientBoostingClassifier(n_estimators=100, learni
    'LGBMClassifier': LGBMClassifier(n_estimators=100, learning_rate=0.1, max_depth=-
    'StackingClassifier': StackingClassifier(
        estimators = [
            ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
            ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss', ran
        ],
        final_estimator=LogisticRegression(max_iter=1000, random_state=42)
    ),
    'XGBClassifier': XGBClassifier(use_label_encoder=False, eval_metric='logloss', ra
    'AdaBoostClassifier': AdaBoostClassifier(n_estimators=100, random_state=42),
    'GaussianNB': GaussianNB(),
    'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42, class_we
    'BaggingClassifier': BaggingClassifier(n_estimators=50, random_state=42),
    'KNeighborsClassifier': KNeighborsClassifier(n_neighbors=5),
    'MLPClassifier': MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_st
}

# Initialize dictionary to store predicted probabilities
y_probs = {}

# Function to train, predict and evaluate models
def train_and_evaluate_model(model_name, model, X_train, X_test, y_train, y_test):
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]  # Get probabilities for ROC-AUC score

    # Store the probabilities
    y_probs[model_name] = y_prob

    # Evaluate the model
    report = classification_report(y_test, y_pred, output_dict=True)
    auc_score = roc_auc_score(y_test, y_prob)
    return auc_score

# Train and evaluate each model using tqdm for progress tracking
roc_auc_scores = []
for model_name, model in tqdm(models.items(), desc="Training and Evaluating Models"):
    auc_score = train_and_evaluate_model(model_name, model, X_train_undersampled, X_t
    roc_auc_scores.append(auc_score)

# Create a DataFrame to show ROC-AUC scores for each model
results_df = pd.DataFrame({'Model': list(models.keys()), 'ROC-AUC': roc_auc_scores}).
```

```
results_df.reset_index(drop=True, inplace=True)

# Print results
results_df
```

```
Class distribution after Undersampling:
0    100660
1    100660
dtype: int64
Training and Evaluating Models: 100%|██████████| 10/10 [05:59<00:00, 35.90s/it]
```

Out[38]:

| | Model | ROC-AUC |
|---|---|---|
| **0** | GradientBoostingClassifier | 0.597323 |
| **1** | LGBMClassifier | 0.596211 |
| **2** | StackingClassifier | 0.594238 |
| **3** | XGBClassifier | 0.593339 |
| **4** | AdaBoostClassifier | 0.592597 |
| **5** | GaussianNB | 0.591666 |
| **6** | LogisticRegression | 0.587053 |
| **7** | BaggingClassifier | 0.563216 |
| **8** | KNeighborsClassifier | 0.532159 |
| **9** | MLPClassifier | 0.520258 |

### Not much different either - so we have to settle with this performance unless we get some new data.

In [ ]:

In [ ]:

# Narratives for Including/Excluding Variables of Choice

1. **Included Variables:**

- **Age:** Mental health conditions can correlate with age groups, as younger and older populations may have different risk factors.
- **Marital Status:** Social support from relationships might influence mental health.
- **Education Level:** Higher education may correlate with awareness and access to mental health resources.
- **Employment Status:** Unemployment or work stress could contribute to mental health issues.
- **Income:** Financial stability plays a significant role in stress and mental health.
- **Dietary Habits, Physical Activity Level, and Sleep Patterns:** These are key lifestyle factors linked to mental well-being.
- **Family History of Depression:** A known risk factor for mental illness due to genetic predisposition.
- **Chronic Medical Conditions:** Physical health conditions can increase the likelihood of mental health challenges.

- **History of Substance Abuse:** Substance abuse often coexists with mental health disorders.

2. **Excluded Variables:**

    - **Name:** Irrelevant to the prediction and introduces potential biases.

---

## Narrative Supporting the Models of Choice

1. **Gradient Boosting, LightGBM, and XGBoost:**

    - These models handle complex interactions between features effectively.
    - Suitable for imbalanced datasets with their ability to focus on hard-to-classify samples.
    - Gradient Boosting generally provides robust performance but is computationally intensive.

2. **Stacking:**

    - Combines the strengths of multiple models to improve predictive performance.
    - Suitable when individual models show complementary strengths.

3. **AdaBoost:**

    - Focuses on misclassified samples, making it effective for datasets with imbalanced classes.

4. **Logistic Regression:**

    - A baseline model for binary classification with interpretability.
    - Useful for understanding feature importance.

5. **Naive Bayes:**

    - Effective for smaller datasets and categorical variables, though simplistic for complex interactions.

6. **Bagging:**

    - Combines weak learners to reduce variance and improve stability.

7. **KNN:**

    - Non-parametric and suitable for datasets with clear clusters, though sensitive to noise and high dimensionality.

8. **MLP (Multi-Layer Perceptron):**

    - Captures non-linear relationships and interactions between features, ideal for deeper insights in larger datasets.

---

## Assessment of Model Performance

- **Evaluation Metrics:**

    - **ROC-AUC Score:** Measures the ability of the model to distinguish between classes. Higher scores indicate better performance.
    - **Classification Report (Precision, Recall, F1-Score):**
        - Precision ensures reduced false positives.
        - Recall ensures reduced false negatives, critical in predicting mental health risks.
        - F1-Score balances precision and recall for a holistic measure.

- **Performance Insights:**

- Ensemble models (e.g., Gradient Boosting, LightGBM, XGBoost, Stacking) are expected to perform well due to their ability to handle non-linear relationships.
- Simpler models like Logistic Regression and Naive Bayes provide interpretability but might lack predictive power for complex patterns.

---

## Assessment of Potential Model Biases

1. **Class Imbalance:**
   - Addressed using **SMOTE**, which oversamples the minority class to ensure fair representation.
2. **Feature Bias:**
   - Variables like income and employment might indirectly reflect socioeconomic biases.
   - Family history might overemphasize genetic predispositions, potentially ignoring other risk factors.
3. **Model-Specific Biases:**
   - **Overfitting:** Complex models like Gradient Boosting and MLP may overfit the training data.
   - **Underfitting:** Simpler models like Logistic Regression and Naive Bayes may underfit if feature interactions are significant.
4. **Sampling Bias:**
   - If the dataset is not representative of diverse populations, the model might not generalize well across demographics.
5. **Interpretability vs. Complexity:**
   - Models like Gradient Boosting and XGBoost may lack interpretability, posing challenges for clinical or ethical justifications.

---

## Next Steps

- **Hyperparameter Tuning:** Optimize the models using GridSearchCV or RandomizedSearchCV.
- **Feature Engineering:** Investigate interaction terms, polynomial features, or domain-specific transformations.
- **Bias Mitigation:**
  - Use fairness metrics (e.g., demographic parity, equal opportunity) to ensure equitable predictions.
  - Regularly audit predictions across demographic groups.

By balancing interpretability, fairness, and predictive power, these models and evaluations can guide meaningful interventions for mental health awareness and prevention.

In [ ]:

```
```

In [ ]:

```
```