AMS 595 Group Project: Identifying Risk Factors for Major Depressive Disorder

Abby Bindelglass, Jane Condon, Nicholas Tardugno, Sydney Walters-Diaz

Data Preparation

In [3]:
```
pip install catboost
```

```
Requirement already satisfied: catboost in c:\users\janec\anaconda3\lib\site-package
s (1.2.8)
Requirement already satisfied: graphviz in c:\users\janec\anaconda3\lib\site-package
s (from catboost) (0.21)
Requirement already satisfied: matplotlib in c:\users\janec\anaconda3\lib\site-packa
ges (from catboost) (3.9.2)
Requirement already satisfied: numpy<3.0,>=1.16.0 in c:\users\janec\anaconda3\lib\si
te-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in c:\users\janec\anaconda3\lib\site-pac
kages (from catboost) (2.2.2)
Requirement already satisfied: scipy in c:\users\janec\anaconda3\lib\site-packages
(from catboost) (1.13.1)
Requirement already satisfied: plotly in c:\users\janec\anaconda3\lib\site-packages
(from catboost) (5.24.1)
Requirement already satisfied: six in c:\users\janec\anaconda3\lib\site-packages (fr
om catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\janec\anaconda3\li
b\site-packages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\janec\anaconda3\lib\site-pac
kages (from pandas>=0.24->catboost) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\janec\anaconda3\lib\site-p
ackages (from pandas>=0.24->catboost) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\janec\anaconda3\lib\site
-packages (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\janec\anaconda3\lib\site-pac
kages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\janec\anaconda3\lib\sit
e-packages (from matplotlib->catboost) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\janec\anaconda3\lib\sit
e-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\janec\anaconda3\lib\site-
packages (from matplotlib->catboost) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\janec\anaconda3\lib\site-packag
es (from matplotlib->catboost) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\janec\anaconda3\lib\site
-packages (from matplotlib->catboost) (3.1.2)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\janec\anaconda3\lib\site-
packages (from plotly->catboost) (8.2.3)
Note: you may need to restart the kernel to use updated packages.
```

Importing Necessary Libraries

In [5]:
```
!pip install --upgrade statsmodels
```

```
Requirement already satisfied: statsmodels in c:\users\janec\anaconda3\lib\site-pack
ages (0.14.6)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\janec\anaconda3\lib\site
-packages (from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\janec\anaconda3\lib\si
te-packages (from statsmodels) (1.13.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\janec\anaconda3\lib\s
ite-packages (from statsmodels) (2.2.2)
Requirement already satisfied: patsy>=0.5.6 in c:\users\janec\anaconda3\lib\site-pac
kages (from statsmodels) (0.5.6)
Requirement already satisfied: packaging>=21.3 in c:\users\janec\anaconda3\lib\site-
packages (from statsmodels) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\janec\anaconda3\li
b\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\janec\anaconda3\lib\site-pac
kages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\janec\anaconda3\lib\site-p
ackages (from pandas!=2.1.0,>=1.4->statsmodels) (2023.3)
Requirement already satisfied: six in c:\users\janec\anaconda3\lib\site-packages (fr
om patsy>=0.5.6->statsmodels) (1.16.0)
```

In [6]:
```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\janec\anaconda3\lib\site-package
s (4.6.0)
Requirement already satisfied: numpy>=1.17.0 in c:\users\janec\anaconda3\lib\site-pa
ckages (from lightgbm) (1.26.4)
Requirement already satisfied: scipy in c:\users\janec\anaconda3\lib\site-packages
(from lightgbm) (1.13.1)
```

In [7]:
```
# Insert libraries here
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import joblib
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, roc_curve, confusion_matrix, ConfusionMatrixDisplay,
    classification_report, brier_score_loss
)
from sklearn.calibration import calibration_curve
import lightgbm as lgb
from catboost import CatBoostClassifier
from scipy.stats import chi2_contingency
```

```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Importing the Data

In [9]:
```
# Reading the dataset into python using pandas
df = pd.read_csv("mhcld_puf_2023.csv", dtype={'year':int}, low_memory=False)
```

In [10]:
```
# Looking at the first few rows to ensure that the data has been imported correctly
df.head()
```

Out[10]:

	YEAR	AGE	EDUC	ETHNIC	RACE	SEX	SPHSERVICE	CMPSERVICE	OPISERVICE	RTCS
0	2023	4	-9	3	5	2	2	1	1	
1	2023	5	4	4	3	2	1	2	2	
2	2023	8	4	4	5	2	1	1	2	
3	2023	11	-9	3	5	2	2	1	2	
4	2023	3	3	4	5	2	1	1	2	

5 rows × 40 columns

Data Preprocessing

Mapping For Easier Readability

As shown above, the readability of the data is very poor, as is often seen with large survey data. To make the data easier to interpret and understand, we will map our variables to the values/explanations provided in the codebook. We will start with the target variable, "DEPRESSFLG".

In [14]:
```
# Target variable: depressive disorder variable map

depressflg_map = {
    1: "Depressive disorder reported",
    0: "Not reported"
}
```

We can also map the predictor variables into something that is easier to understand. We start with the co-occurring mental health disorder variables. Since the values have the same meaning for each of the variables, (i.e., 0 = disorder reported, 1 = disorder not reported), we can combine them into a single map.

In [16]:
```
# Predictor map: co-occurring mental health disorders map

disorder_flags = ["ANXIETYFLG", "ADHDFLG", "CONDUCTFLG", "DELIRDEMFLG", "BIPOLARFLG
```

```
        "ODDFLG", "PDDFLG", "PERSONFLG", "SCHIZOFLG", "OTHERDISFLG",
        "TRAUSTREFLG", "ALCSUBFLG"]

binary_flag_map = {1: "Reported", 0: "Not reported"}
```

For our next predictor variable, substance use disorder, we have two variables that we may use:

- SAP (binary): Gives a value of 1 if substance use disorder present, 2 if substance use disorder not present, and 0 if the response to this survey question is missing.
- SUB (numeric): Gives a value of 1-13 indicating a client's substance use diagnosis during the reporting period (e.g., 9 = alcohol abuse, 10 = cocaine abuse), or a value of -9 for a missing/invalid diagnosis.

Since the category codes of these variables are different from those of the co-occurring mental health disorder variables and are NOT binary, we must create a separate map. We can use the SAP variable to construct a simple yes/no variable indicating whether or not a survey respondent has been diagnosed with any type of substance use disorder. If we want to look into this further, the SUB variable indicates WHICH substance use disorder an individual has been diagnosed with.

In [18]:
```
# Predictor map: substance use map (SUB)

sub_use_map = {
    1: "Trauma/stressor disorder",
    2: "Anxiety disorder",
    3: "Attention deficit/hyperactivity disorder (ADHD)",
    4: "Conduct disorder",
    5: "Delirium/dementia disorder",
    6: "Bipolar disorder",
    7: "Depressive disorder",
    8: "Oppositional defiant disorder",
    9: "Pervasive developmental disorder",
    10: "Personality disorder",
    11: "Schizophrenia/psychotic disorder",
    12: "Alcohol or Substance Use Disorder",
    13: "Other disorder",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [19]:
```
# Predictor map: substance use map (SAP)
sap_map = {
    1: "Substance use problem reported",
    0: "No substance use problem reported",
    -9: "Missing",
    -8: "Not applicable"
}

# Applying to SAP column:
df["SAP_LABEL"] = df["SAP"].map(sap_map)
```

```
# Creating into a simple yes/no binary variable
df["HAS_SAP"] = df["SAP"].isin([1]).astype(int)
```

Some other useful predictor variables that we can map are age, sex, education status, marital status, residential status, veteran status, employment status, race/ethnicity, and geographic region (in the U.S.).

In [21]:
```
# Predictor map: age

age_map = {
    1: "0–11",
    2: "12–14",
    3: "15–17",
    4: "18–20",
    5: "21–24",
    6: "25–29",
    7: "30–34",
    8: "35–39",
    9: "40–44",
    10: "45–49",
    11: "50–54",
    12: "55–59",
    13: "60–64",
    14: "65+",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [22]:
```
# Predictor map: sex
sex_map = {
    1: "Male",
    2: "Female",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [23]:
```
# Predictor map: education level
educ_map = {
    1: "Special education",
    2: "8 years or less",
    3: "9–11 years",
    4: "12 years or GED",
    5: "13 years or more",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [24]:
```
# Predictor map: marital status
marstat_map = {
    1: "Never married",
    2: "Now married",
    3: "Separated",
    4: "Divorced or widowed",
    -9: "Missing",
```

```
    -8: "Not applicable"
}
```

In [25]:
```
# Predictor map: residential status
livarag_map = {
    1: "Experiencing homelessness",
    2: "Private residence",
    3: "Other",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [26]:
```
# Predictor map: veteran status
veteran_map = {
    1: "Veteran",
    2: "Not a veteran",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [27]:
```
# Predictor map: employment status
employ_map = {
    1: "Full-time",
    2: "Part-time",
    3: "Employed (not differentiated)",
    4: "Unemployed",
    5: "Not in labor force",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [28]:
```
# Predictor map: ethnicity
ethnic_map = {
    1: "Hispanic or Latino",
    2: "Not Hispanic or Latino",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [29]:
```
# Predictor map: race
race_map = {
    1: "White",
    2: "Black or African American",
    3: "American Indian or Alaska Native",
    4: "Asian",
    5: "Native Hawaiian or Pacific Islander",
    6: "Multiracial",
    -9: "Missing",
    -8: "Not applicable"
}
```

In [30]:
```
region_map = {
    1: "Northeast",
    2: "Midwest",
```

```
    3: "South",
    4: "West"
}
```

Creating a Pre-Processing Function

Next, we can create a pre-processing function that:

- Applies every mapping dictionary to the dataframe
- Creates binary flags (simple yes/no variable for appropriate variables)
- Handles missing values (using median/mode imputation)
- Returns a numeric dataframe for modeling, as well as a labeled dataframe that is easier to read/interpret

In [33]:
```
def preprocess_data(df):

    # Dealing with missing SAMHSA codes
    missing_codes = [-9, -8, -7, -6]
    df = df.replace(missing_codes, np.nan)

    # Preserving disorder flags as numeric
    for f in disorder_flags:
        if f in df.columns:
            df[f] = df[f].fillna(0).astype(int)

    # Preserving SUB and SAP as numeric predictors
    if "SUB" in df.columns:
        df["SUB"] = df["SUB"].fillna(0).astype(int)
    if "SAP" in df.columns:
        df["SAP"] = df["SAP"].fillna(0).astype(int)

    # Dropping unnecessary columns
    id_cols = ["CASEID", "STATEFIP"]
    df = df.drop(columns=[col for col in id_cols if col in df.columns])

    predictor_vars = [
        "AGE", "SEX", "EDUC", "MARSTAT", "LIVARAG",
        "VETERAN", "EMPLOY", "ETHNIC", "RACE",
        "SUB", "SAP", "REGION"
    ] + disorder_flags

    # Debugging: checking for missing predictor vars
    missing = [col for col in predictor_vars if col not in df.columns]
    if missing:
        print("Missing predictor columns:", missing)

    # Applying predictor selection
    df = df[[col for col in df.columns if col in predictor_vars or col == "DEPRESSF

    # Applying mappings to the dataframe
    df["AGE_LABEL"] = df["AGE"].map(age_map)
    df["SEX_LABEL"]  = df["SEX"].map(sex_map)
    df["EDUC_LABEL"] = df["EDUC"].map(educ_map)
```

```
    df["MARSTAT_LABEL"] = df["MARSTAT"].map(marstat_map)
    df["LIVARAG_LABEL"] = df["LIVARAG"].map(livarag_map)
    df["VETERAN_LABEL"] = df["VETERAN"].map(veteran_map)
    df["EMPLOY_LABEL"] = df["EMPLOY"].map(employ_map)
    df["ETHNIC_LABEL"] = df["ETHNIC"].map(ethnic_map)
    df["RACE_LABEL"] = df["RACE"].map(race_map)

    df["REGION_LABEL"] = df["REGION"].map(region_map)

    df["SUB_LABEL"] = df["SUB"].map(sub_use_map)
    df["SAP_LABEL"] = df["SAP"].map(sap_map)
    df["DEPRESS_LABEL"] = df["DEPRESSFLG"].map(depressflg_map)

    for f in disorder_flags:
        df[f + "_LABEL"] = df[f].map(binary_flag_map)

    # Creating binary modeling flags (simple yes/no binary variables)
    df["HAS_SUBSTANCE_USE"] = (df["SUB"] == 12).astype(int)
    df["HAS_SAP"] = (df["SAP"] == 1).astype(int)
    df["MDD"] = (df["DEPRESSFLG"] == 1).astype(int)
    df["ANY_OTHER_MH_DISORDER"] = df[disorder_flags].fillna(0).max(axis=1)
    df["IS_VETERAN"] = (df["VETERAN"] == 1).astype(int)
    df["IS_HOMELESS"] = (df["LIVARAG"] == 1).astype(int)
    df["IS_MARRIED"] = (df["MARSTAT"] == 2).astype(int)

    # Using one-hot encoding for categorical variables
    categ_cols = [
        "AGE_LABEL", "SEX_LABEL", "EDUC_LABEL", "MARSTAT_LABEL",
        "LIVARAG_LABEL", "VETERAN_LABEL", "EMPLOY_LABEL",
        "ETHNIC_LABEL", "RACE_LABEL",
        "SUB_LABEL", "SAP_LABEL", "REGION_LABEL"
    ]

    model_df = pd.get_dummies(df.copy(), columns=categ_cols, drop_first=True)

    # Removing all columns containing "_LABEL" (including dummy-expanded ones)
    label_cols = [c for c in model_df.columns if "_LABEL" in c]
    model_df = model_df.drop(columns=label_cols, errors="ignore")

    # Handling missing values using imputation
    num_cols = model_df.select_dtypes(include=["float64", "int64"]).columns
    cat_cols = model_df.select_dtypes(include=["object", "category"]).columns

    # For numerical variables: median imputation
    num_imputer = SimpleImputer(strategy="median")
    model_df[num_cols] = num_imputer.fit_transform(model_df[num_cols])

    # For categorical variables: mode imputation
    if len(cat_cols) > 0:
        cat_imputer = SimpleImputer(strategy="most_frequent")
        model_df[cat_cols] = cat_imputer.fit_transform(model_df[cat_cols])

    return df, model_df
```

In [34]:
```
# Calling the function
clean_df, model_df = preprocess_data(df)
```

```
model_df.isna().sum().sum() # Ensuring that missing values have been dealt with
```

Out[34]: 0

Constructing Target Vector and Predictor Matrix

In [36]:
```
# Target variable
y = model_df["MDD"]
```

In [37]:
```
# Predictors matrix
X = model_df.drop(columns=["MDD", "DEPRESSFLG"], errors="ignore")
```

In [38]:
```
## drop these to avoid redundancy with binary indicators
raw_vars_to_drop = [
    "VETERAN",
    "SAP",
    "SUB",
    "MARSTAT",
    "LIVARAG"
]
raw_vars_to_drop = [v for v in raw_vars_to_drop if v in X.columns]
X = X.drop(columns=raw_vars_to_drop)
print("Final predictors used in models:")
print(X.columns.tolist())
```

```
Final predictors used in models:
['AGE', 'EDUC', 'ETHNIC', 'RACE', 'SEX', 'EMPLOY', 'TRAUSTREFLG', 'ANXIETYFLG', 'ADH
DFLG', 'CONDUCTFLG', 'DELIRDEMFLG', 'BIPOLARFLG', 'ODDFLG', 'PDDFLG', 'PERSONFLG',
'SCHIZOFLG', 'ALCSUBFLG', 'OTHERDISFLG', 'REGION', 'HAS_SUBSTANCE_USE', 'HAS_SAP',
'ANY_OTHER_MH_DISORDER', 'IS_VETERAN', 'IS_HOMELESS', 'IS_MARRIED']
```

Splitting Data into Training, Validation and Test Set

In [40]:
```
model_df = model_df.reset_index(drop=True)
```

In [41]:
```
# Splitting into training and test set
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)
```

In [42]:
```
# Splitting training set into training and validation set
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.20, random_state=42, stratify=y_train_f
)
```

Exploratory Data Analysis

Overview of data shape and missingness

In [45]:
```
print("Shape of cleaned dataset:", clean_df.shape)
```

```
summary_df = pd.DataFrame({
    "Data Type": clean_df.dtypes,
    "Missing Values": clean_df.isna().sum(),
    "Non-Missing Values": clean_df.notna().sum()
})

summary_df
```

```
Shape of cleaned dataset: (7035641, 57)
```

Out[45]:

	Data Type	Missing Values	Non-Missing Values
AGE	float64	5113	7030528
EDUC	float64	3581010	3454631
ETHNIC	float64	980266	6055375
RACE	float64	850616	6185025
SEX	float64	13411	7022230
SUB	int32	0	7035641
MARSTAT	float64	2835619	4200022
SAP	int32	0	7035641
EMPLOY	float64	4299722	2735919
VETERAN	float64	3943173	3092468
LIVARAG	float64	2595659	4439982
TRAUSTREFLG	int32	0	7035641
ANXIETYFLG	int32	0	7035641
ADHDFLG	int32	0	7035641
CONDUCTFLG	int32	0	7035641
DELIRDEMFLG	int32	0	7035641
BIPOLARFLG	int32	0	7035641
DEPRESSFLG	int64	0	7035641
ODDFLG	int32	0	7035641
PDDFLG	int32	0	7035641
PERSONFLG	int32	0	7035641
SCHIZOFLG	int32	0	7035641
ALCSUBFLG	int32	0	7035641
OTHERDISFLG	int32	0	7035641
REGION	int64	0	7035641
AGE_LABEL	object	5113	7030528
SEX_LABEL	object	13411	7022230
EDUC_LABEL	object	3581010	3454631
MARSTAT_LABEL	object	2835619	4200022
LIVARAG_LABEL	object	2595659	4439982

	Data Type	Missing Values	Non-Missing Values
VETERAN_LABEL	object	3943173	3092468
EMPLOY_LABEL	object	4299722	2735919
ETHNIC_LABEL	object	6946424	89217
RACE_LABEL	object	850616	6185025
REGION_LABEL	object	4316	7031325
SUB_LABEL	object	6073595	962046
SAP_LABEL	object	3878170	3157471
DEPRESS_LABEL	object	0	7035641
ANXIETYFLG_LABEL	object	0	7035641
ADHDFLG_LABEL	object	0	7035641
CONDUCTFLG_LABEL	object	0	7035641
DELIRDEMFLG_LABEL	object	0	7035641
BIPOLARFLG_LABEL	object	0	7035641
ODDFLG_LABEL	object	0	7035641
PDDFLG_LABEL	object	0	7035641
PERSONFLG_LABEL	object	0	7035641
SCHIZOFLG_LABEL	object	0	7035641
OTHERDISFLG_LABEL	object	0	7035641
TRAUSTREFLG_LABEL	object	0	7035641
ALCSUBFLG_LABEL	object	0	7035641
HAS_SUBSTANCE_USE	int32	0	7035641
HAS_SAP	int32	0	7035641
MDD	int32	0	7035641
ANY_OTHER_MH_DISORDER	int32	0	7035641
IS_VETERAN	int32	0	7035641
IS_HOMELESS	int32	0	7035641
IS_MARRIED	int32	0	7035641

Summary of numeric and binary predictors

In [47]:
```
# Automatically detect binary variables
binary_vars = [col for col in clean_df.columns
               if clean_df[col].dropna().isin([0,1]).all()]  # must be strictly 0/1
```

```
# Detect real numeric variables (excluding IDs, if any)
numeric_vars = [
    col for col in clean_df.columns
    if clean_df[col].dtype in ["float64", "int64"] and col not in binary_vars
]

print("Numeric variables (not binary):")
print(numeric_vars)

print("\nBinary variables:")
print(binary_vars)
```

```
Numeric variables (not binary):
['AGE', 'EDUC', 'ETHNIC', 'RACE', 'SEX', 'MARSTAT', 'EMPLOY', 'VETERAN', 'LIVARAG',
'REGION']

Binary variables:
['TRAUSTREFLG', 'ANXIETYFLG', 'ADHDFLG', 'CONDUCTFLG', 'DELIRDEMFLG', 'BIPOLARFLG',
'DEPRESSFLG', 'ODDFLG', 'PDDFLG', 'PERSONFLG', 'SCHIZOFLG', 'ALCSUBFLG', 'OTHERDISFL
G', 'HAS_SUBSTANCE_USE', 'HAS_SAP', 'MDD', 'ANY_OTHER_MH_DISORDER', 'IS_VETERAN', 'I
S_HOMELESS', 'IS_MARRIED']
```

In [48]:
```
clean_df[numeric_vars].describe()
```

Out[48]:

	AGE	EDUC	ETHNIC	RACE	SEX	MARSTA
count	7.030528e+06	3.454631e+06	6.055375e+06	6.185025e+06	7.022230e+06	4.200022e+0
mean	6.927860e+00	3.475133e+00	3.750475e+00	4.574773e+00	1.533202e+00	1.492767e+0
std	4.034996e+00	1.086498e+00	5.016044e-01	1.120863e+00	4.988965e-01	9.800408e-0
min	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+0
25%	3.000000e+00	2.000000e+00	4.000000e+00	5.000000e+00	1.000000e+00	1.000000e+0
50%	7.000000e+00	4.000000e+00	4.000000e+00	5.000000e+00	2.000000e+00	1.000000e+0
75%	1.000000e+01	4.000000e+00	4.000000e+00	5.000000e+00	2.000000e+00	1.000000e+0
max	1.400000e+01	5.000000e+00	4.000000e+00	6.000000e+00	2.000000e+00	4.000000e+0

In [49]:
```
binary_summary = clean_df[binary_vars].mean().to_frame("Proportion (value=1)")
binary_summary.sort_values(by="Proportion (value=1)", ascending=False)
```

Out[49]:

	Proportion (value=1)
ANY_OTHER_MH_DISORDER	0.757386
HAS_SAP	0.361003
MDD	0.268916
DEPRESSFLG	0.268916
ANXIETYFLG	0.258544
TRAUSTREFLG	0.206445
OTHERDISFLG	0.128787
SCHIZOFLG	0.109068
ADHDFLG	0.104156
BIPOLARFLG	0.099487
IS_MARRIED	0.060510
ALCSUBFLG	0.053722
IS_HOMELESS	0.028676
PERSONFLG	0.022940
ODDFLG	0.021499
PDDFLG	0.020412
IS_VETERAN	0.014570
CONDUCTFLG	0.013888
DELIRDEMFLG	0.003347
HAS_SUBSTANCE_USE	0.002676

In [50]:
```
disorder_flags = ["ANXIETYFLG", "ADHDFLG", "CONDUCTFLG", "DELIRDEMFLG", "BIPOLARFLG
         "ODDFLG", "PDDFLG", "PERSONFLG", "SCHIZOFLG", "OTHERDISFLG",
         "TRAUSTREFLG", "ALCSUBFLG"]

for flag in disorder_flags:
    print("\n============================")
    print(f"MDD Prevalence by {flag}")
    print(clean_df.groupby(flag)['MDD'].mean())
```

```
============================
MDD Prevalence by ANXIETYFLG
ANXIETYFLG
0    0.239386
1    0.353603
Name: MDD, dtype: float64

============================
MDD Prevalence by ADHDFLG
ADHDFLG
0    0.283234
1    0.145766
Name: MDD, dtype: float64

============================
MDD Prevalence by CONDUCTFLG
CONDUCTFLG
0    0.271636
1    0.075782
Name: MDD, dtype: float64

============================
MDD Prevalence by DELIRDEMFLG
DELIRDEMFLG
0    0.269240
1    0.172393
Name: MDD, dtype: float64

============================
MDD Prevalence by BIPOLARFLG
BIPOLARFLG
0    0.290432
1    0.074162
Name: MDD, dtype: float64

============================
MDD Prevalence by ODDFLG
ODDFLG
0    0.272923
1    0.086552
Name: MDD, dtype: float64

============================
MDD Prevalence by PDDFLG
PDDFLG
0    0.272721
1    0.086343
Name: MDD, dtype: float64

============================
MDD Prevalence by PERSONFLG
PERSONFLG
0    0.268454
1    0.288587
Name: MDD, dtype: float64
```

```
============================
MDD Prevalence by SCHIZOFLG
SCHIZOFLG
0    0.293825
1    0.065444
Name: MDD, dtype: float64

============================
MDD Prevalence by OTHERDISFLG
OTHERDISFLG
0    0.287367
1    0.144102
Name: MDD, dtype: float64

============================
MDD Prevalence by TRAUSTREFLG
TRAUSTREFLG
0    0.286918
1    0.199719
Name: MDD, dtype: float64

============================
MDD Prevalence by ALCSUBFLG
ALCSUBFLG
0    0.271802
1    0.218091
Name: MDD, dtype: float64
```

Frequency distributions for categorical codes

In [52]:
```
categ_cols = [
        "AGE_LABEL", "SEX_LABEL", "EDUC_LABEL", "MARSTAT_LABEL",
        "LIVARAG_LABEL", "VETERAN_LABEL", "EMPLOY_LABEL",
        "ETHNIC_LABEL", "RACE_LABEL",
        "SUB_LABEL", "SAP_LABEL", "REGION_LABEL"
    ]

freq_results = {}

for col in categ_cols:
    freq_table = clean_df[col].value_counts(dropna=False)
    percent_table = clean_df[col].value_counts(normalize=True, dropna=False) * 100

    result = pd.DataFrame({
        'Count': freq_table,
        'Percent': percent_table.round(2)
    })

    freq_results[col] = result
```

In [53]:
```
for col in categ_cols:
    print(freq_results[col])
```

AGE_LABEL	Count	Percent
0–11	833690	11.85
30–34	659153	9.37
25–29	598836	8.51
35–39	577313	8.21
15–17	543032	7.72
12–14	511537	7.27
40–44	507480	7.21
65+	441242	6.27
21–24	421321	5.99
50–54	412237	5.86
55–59	407541	5.79
45–49	404937	5.76
60–64	357827	5.09
18–20	354382	5.04
NaN	5113	0.07

SEX_LABEL	Count	Percent
Female	3744266	53.22
Male	3277964	46.59
NaN	13411	0.19

EDUC_LABEL	Count	Percent
NaN	3581010	50.90
12 years or GED	1290887	18.35
8 years or less	860675	12.23
13 years or more	641105	9.11
9–11 years	626458	8.90
Special education	35506	0.50

MARSTAT_LABEL	Count	Percent
Never married	3175733	45.14
NaN	2835619	40.30
Divorced or widowed	446780	6.35
Now married	425724	6.05
Separated	151785	2.16

LIVARAG_LABEL	Count	Percent
Private residence	3842390	54.61
NaN	2595659	36.89
Other	395836	5.63
Experiencing homelessness	201756	2.87

VETERAN_LABEL	Count	Percent
NaN	3943173	56.05
Not a veteran	2989958	42.50
Veteran	102510	1.46

EMPLOY_LABEL	Count	Percent
NaN	4299722	61.11
Not in labor force	1189134	16.90
Unemployed	781444	11.11
Full-time	420560	5.98
Part-time	213948	3.04
Employed (not differentiated)	130833	1.86

```
                          Count  Percent
ETHNIC_LABEL
NaN                     6946424    98.73
Hispanic or Latino        52799     0.75
Not Hispanic or Latino    36418     0.52
                                     Count  Percent
RACE_LABEL
Native Hawaiian or Pacific Islander  3971313    56.45
American Indian or Alaska Native     1216014    17.28
NaN                                   850616    12.09
Multiracial                           723559    10.28
White                                 145776     2.07
Black or African American             105053     1.49
Asian                                  23310     0.33
                                                   Count  Percent
SUB_LABEL
NaN                                               6073595    86.33
Conduct disorder                                   197572     2.81
Oppositional defiant disorder                      187347     2.66
Depressive disorder                                141946     2.02
Bipolar disorder                                    95335     1.36
Schizophrenia/psychotic disorder                    90059     1.28
Pervasive developmental disorder                    70664     1.00
Other disorder                                      57563     0.82
Attention deficit/hyperactivity disorder (ADHD)     39500     0.56
Delirium/dementia disorder                          26560     0.38
Alcohol or Substance Use Disorder                   18827     0.27
Anxiety disorder                                    12615     0.18
Personality disorder                                12268     0.17
Trauma/stressor disorder                            11790     0.17
                                         Count  Percent
SAP_LABEL
NaN                                    3878170    55.12
Substance use problem reported         2539885    36.10
No substance use problem reported       617586     8.78
                Count  Percent
REGION_LABEL
South         2257956    32.09
West          2105781    29.93
Midwest       1465215    20.83
Northeast     1202373    17.09
NaN              4316     0.06
```

Chi-square tests

In [55]:

```
chi_results = {}

for col in categ_cols:
    print(f"\n=== Chi-Square Test: {col} vs MDD ===")

    # Build contingency table
    contingency_table = pd.crosstab(clean_df[col], clean_df["MDD"])

    # Perform test
    chi2, p_value, dof, expected = chi2_contingency(contingency_table)
```

```
    # Store result
    chi_results[col] = {
        "Chi-Square": chi2,
        "p-Value": p_value,
        "Degrees of Freedom": dof
    }

    print(f"Chi-Square = {chi2:.4f}, p-Value = {p_value:.6f}, df = {dof}")
```

```
=== Chi-Square Test: AGE_LABEL vs MDD ===
Chi-Square = 269678.1406, p-Value = 0.000000, df = 13

=== Chi-Square Test: SEX_LABEL vs MDD ===
Chi-Square = 106233.2910, p-Value = 0.000000, df = 1

=== Chi-Square Test: EDUC_LABEL vs MDD ===
Chi-Square = 76010.0540, p-Value = 0.000000, df = 4

=== Chi-Square Test: MARSTAT_LABEL vs MDD ===
Chi-Square = 59423.9977, p-Value = 0.000000, df = 3

=== Chi-Square Test: LIVARAG_LABEL vs MDD ===
Chi-Square = 16434.5418, p-Value = 0.000000, df = 2

=== Chi-Square Test: VETERAN_LABEL vs MDD ===
Chi-Square = 516.3726, p-Value = 0.000000, df = 1

=== Chi-Square Test: EMPLOY_LABEL vs MDD ===
Chi-Square = 3039.0346, p-Value = 0.000000, df = 4

=== Chi-Square Test: ETHNIC_LABEL vs MDD ===
Chi-Square = 230.7621, p-Value = 0.000000, df = 1

=== Chi-Square Test: RACE_LABEL vs MDD ===
Chi-Square = 6243.5240, p-Value = 0.000000, df = 5

=== Chi-Square Test: SUB_LABEL vs MDD ===
Chi-Square = 5313.0387, p-Value = 0.000000, df = 12

=== Chi-Square Test: SAP_LABEL vs MDD ===
Chi-Square = 24050.9200, p-Value = 0.000000, df = 1

=== Chi-Square Test: REGION_LABEL vs MDD ===
Chi-Square = 47637.0532, p-Value = 0.000000, df = 3
```

Row-wise percentage distributions

In [57]:
```
for col in categ_cols:
    print(col)
    display(pd.crosstab(clean_df[col], clean_df["MDD"], normalize='index') * 100)
```

```
AGE_LABEL
```

MDD	0	1
AGE_LABEL		
0–11	95.374300	4.625700
12–14	78.274885	21.725115
15–17	69.181374	30.818626
18–20	67.485369	32.514631
21–24	68.500739	31.499261
25–29	70.537342	29.462658
30–34	71.664849	28.335151
35–39	72.193593	27.806407
40–44	71.497005	28.502995
45–49	69.666640	30.333360
50–54	67.633667	32.366333
55–59	66.212970	33.787030
60–64	66.076903	33.923097
65+	67.566777	32.433223

SEX_LABEL

MDD	0	1
SEX_LABEL		
Female	67.986356	32.013644
Male	78.920086	21.079914

EDUC_LABEL

MDD	0	1
EDUC_LABEL		
12 years or GED	64.975013	35.024987
13 years or more	63.412390	36.587610
8 years or less	80.250850	19.749150
9–11 years	66.064445	33.935555
Special education	83.729511	16.270489

MARSTAT_LABEL

MDD	0	1
MARSTAT_LABEL		
Divorced or widowed	64.384037	35.615963
Never married	77.035066	22.964934
Now married	65.161231	34.838769
Separated	65.758804	34.241196

LIVARAG_LABEL

MDD	0	1
LIVARAG_LABEL		
Experiencing homelessness	70.968397	29.031603
Other	79.383634	20.616366
Private residence	69.635539	30.364461

VETERAN_LABEL

MDD	0	1
VETERAN_LABEL		
Not a veteran	72.262319	27.737681
Veteran	69.026436	30.973564

EMPLOY_LABEL

MDD	0	1
EMPLOY_LABEL		
Employed (not differentiated)	63.740799	36.259201
Full-time	61.208389	38.791611
Not in labor force	64.912281	35.087719
Part-time	62.778806	37.221194
Unemployed	65.897364	34.102636

ETHNIC_LABEL

MDD	0	1
ETHNIC_LABEL		
Hispanic or Latino	68.874411	31.125589
Not Hispanic or Latino	73.581745	26.418255

RACE_LABEL

MDD	0	1
RACE_LABEL		
American Indian or Alaska Native	75.058758	24.941242
Asian	75.984556	24.015444
Black or African American	71.607665	28.392335
Multiracial	73.572715	26.427285
Native Hawaiian or Pacific Islander	71.601433	28.398567
White	71.768329	28.231671

SUB_LABEL

MDD	0	1
SUB_LABEL		
Alcohol or Substance Use Disorder	69.607479	30.392521
Anxiety disorder	73.182719	26.817281
Attention deficit/hyperactivity disorder (ADHD)	74.288608	25.711392
Bipolar disorder	67.262810	32.737190
Conduct disorder	62.906687	37.093313
Delirium/dementia disorder	68.795181	31.204819
Depressive disorder	71.449002	28.550998
Oppositional defiant disorder	69.167908	30.832092
Other disorder	72.892309	27.107691
Personality disorder	70.060320	29.939680
Pervasive developmental disorder	66.240235	33.759765
Schizophrenia/psychotic disorder	64.951865	35.048135
Trauma/stressor disorder	66.598813	33.401187

SAP_LABEL

MDD	0	1
SAP_LABEL		
No substance use problem reported	81.067576	18.932424
Substance use problem reported	71.325592	28.674408

REGION_LABEL

MDD	0	1
REGION_LABEL		
Midwest	67.237914	32.762086
Northeast	77.703674	22.296326
South	71.972660	28.027340
West	75.765856	24.234144

Visualize effect size using Cramer's V

In [59]:
```
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    r, k = confusion_matrix.shape
    return np.sqrt(chi2 / (n * (min(r - 1, k - 1))))

for col in categ_cols:
    ct = pd.crosstab(clean_df[col], clean_df["MDD"])
    cv = cramers_v(ct)
    print(f"{col}: Cramer’s V = {cv:.4f}")
```

```
AGE_LABEL: Cramer’s V = 0.1959
SEX_LABEL: Cramer’s V = 0.1230
EDUC_LABEL: Cramer’s V = 0.1483
MARSTAT_LABEL: Cramer’s V = 0.1189
LIVARAG_LABEL: Cramer’s V = 0.0608
VETERAN_LABEL: Cramer’s V = 0.0129
EMPLOY_LABEL: Cramer’s V = 0.0333
ETHNIC_LABEL: Cramer’s V = 0.0509
RACE_LABEL: Cramer’s V = 0.0318
SUB_LABEL: Cramer’s V = 0.0743
SAP_LABEL: Cramer’s V = 0.0873
REGION_LABEL: Cramer’s V = 0.0823
```

Data Visualization

In [61]:
```
model_df.head()
model_df.keys()
```

Out[61]:
```
Index(['AGE', 'EDUC', 'ETHNIC', 'RACE', 'SEX', 'SUB', 'MARSTAT', 'SAP',
       'EMPLOY', 'VETERAN', 'LIVARAG', 'TRAUSTREFLG', 'ANXIETYFLG', 'ADHDFLG',
       'CONDUCTFLG', 'DELIRDEMFLG', 'BIPOLARFLG', 'DEPRESSFLG', 'ODDFLG',
       'PDDFLG', 'PERSONFLG', 'SCHIZOFLG', 'ALCSUBFLG', 'OTHERDISFLG',
       'REGION', 'HAS_SUBSTANCE_USE', 'HAS_SAP', 'MDD',
       'ANY_OTHER_MH_DISORDER', 'IS_VETERAN', 'IS_HOMELESS', 'IS_MARRIED'],
      dtype='object')
```

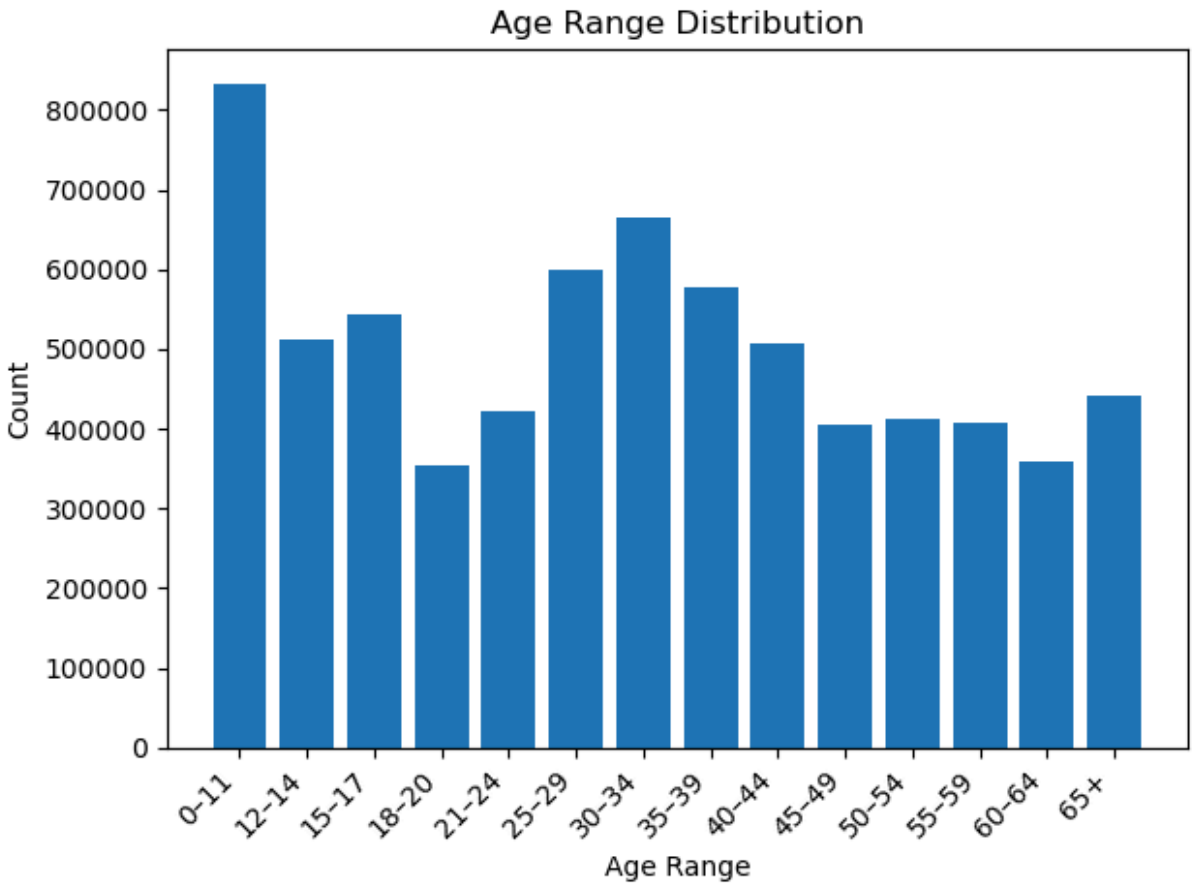
In [62]:
```
cols = ['AGE', 'EDUC', 'ETHNIC', 'RACE', 'SEX', 'SUB', 'MARSTAT', 'SAP',
       'EMPLOY', 'VETERAN', 'LIVARAG', 'TRAUSTREFLG', 'ANXIETYFLG', 'ADHDFLG',
       'CONDUCTFLG', 'DELIRDEMFLG', 'BIPOLARFLG', 'DEPRESSFLG', 'ODDFLG',
```

```
       'PDDFLG', 'PERSONFLG', 'SCHIZOFLG', 'ALCSUBFLG', 'OTHERDISFLG',
       'REGION', 'HAS_SUBSTANCE_USE', 'HAS_SAP', 'MDD',
       'ANY_OTHER_MH_DISORDER', 'IS_VETERAN', 'IS_HOMELESS', 'IS_MARRIED']

[print(c, set(model_df[c].to_list())) for c in cols]
```

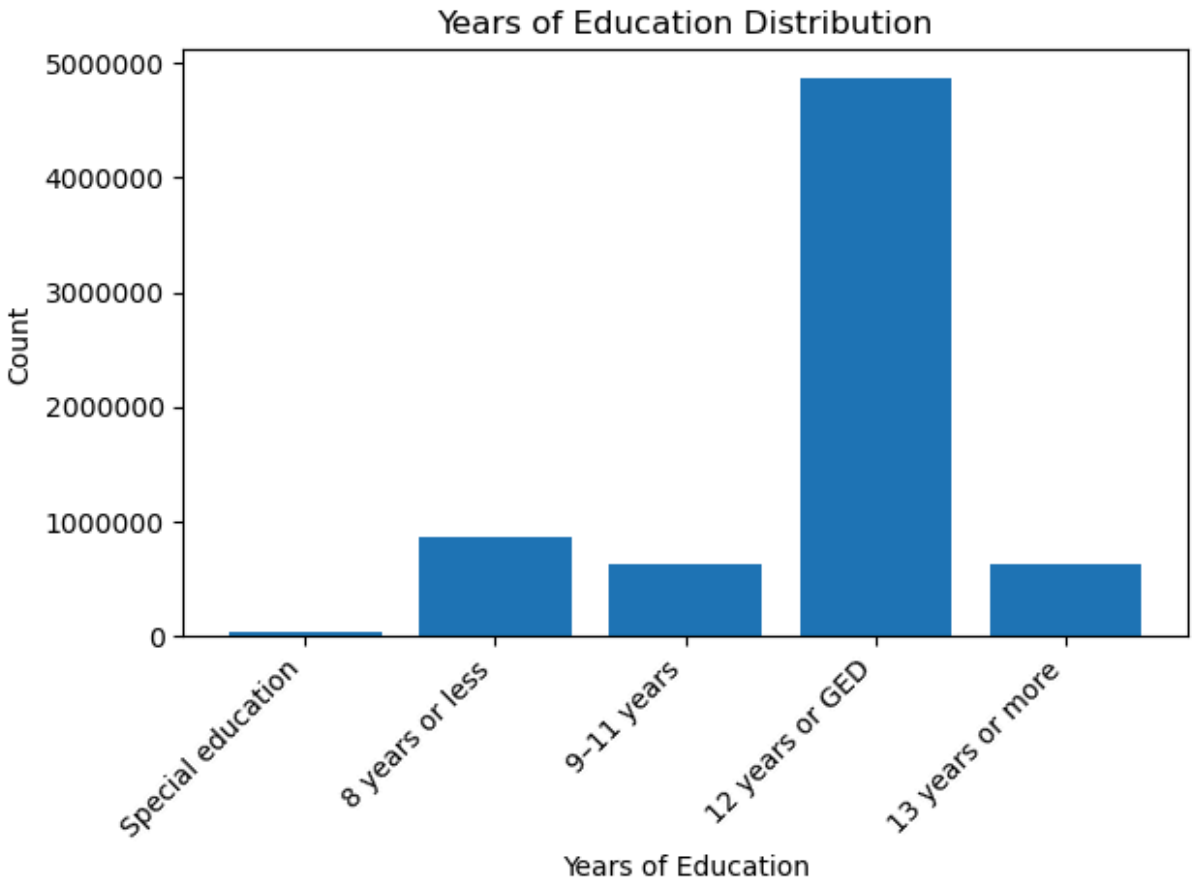
```
AGE {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0}
EDUC {1.0, 2.0, 3.0, 4.0, 5.0}
ETHNIC {1.0, 2.0, 3.0, 4.0}
RACE {1.0, 2.0, 3.0, 4.0, 5.0, 6.0}
SEX {1.0, 2.0}
SUB {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}
MARSTAT {1.0, 2.0, 3.0, 4.0}
SAP {0, 1, 2}
EMPLOY {1.0, 2.0, 3.0, 4.0, 5.0}
VETERAN {1.0, 2.0}
LIVARAG {1.0, 2.0, 3.0}
TRAUSTREFLG {0, 1}
ANXIETYFLG {0, 1}
ADHDFLG {0, 1}
CONDUCTFLG {0, 1}
DELIRDEMFLG {0, 1}
BIPOLARFLG {0, 1}
DEPRESSFLG {0.0, 1.0}
ODDFLG {0, 1}
PDDFLG {0, 1}
PERSONFLG {0, 1}
SCHIZOFLG {0, 1}
ALCSUBFLG {0, 1}
OTHERDISFLG {0, 1}
REGION {0.0, 1.0, 2.0, 3.0, 4.0}
HAS_SUBSTANCE_USE {0, 1}
HAS_SAP {0, 1}
MDD {0, 1}
ANY_OTHER_MH_DISORDER {0, 1}
IS_VETERAN {0, 1}
IS_HOMELESS {0, 1}
IS_MARRIED {0, 1}
```

Out[62]:
```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None]
```

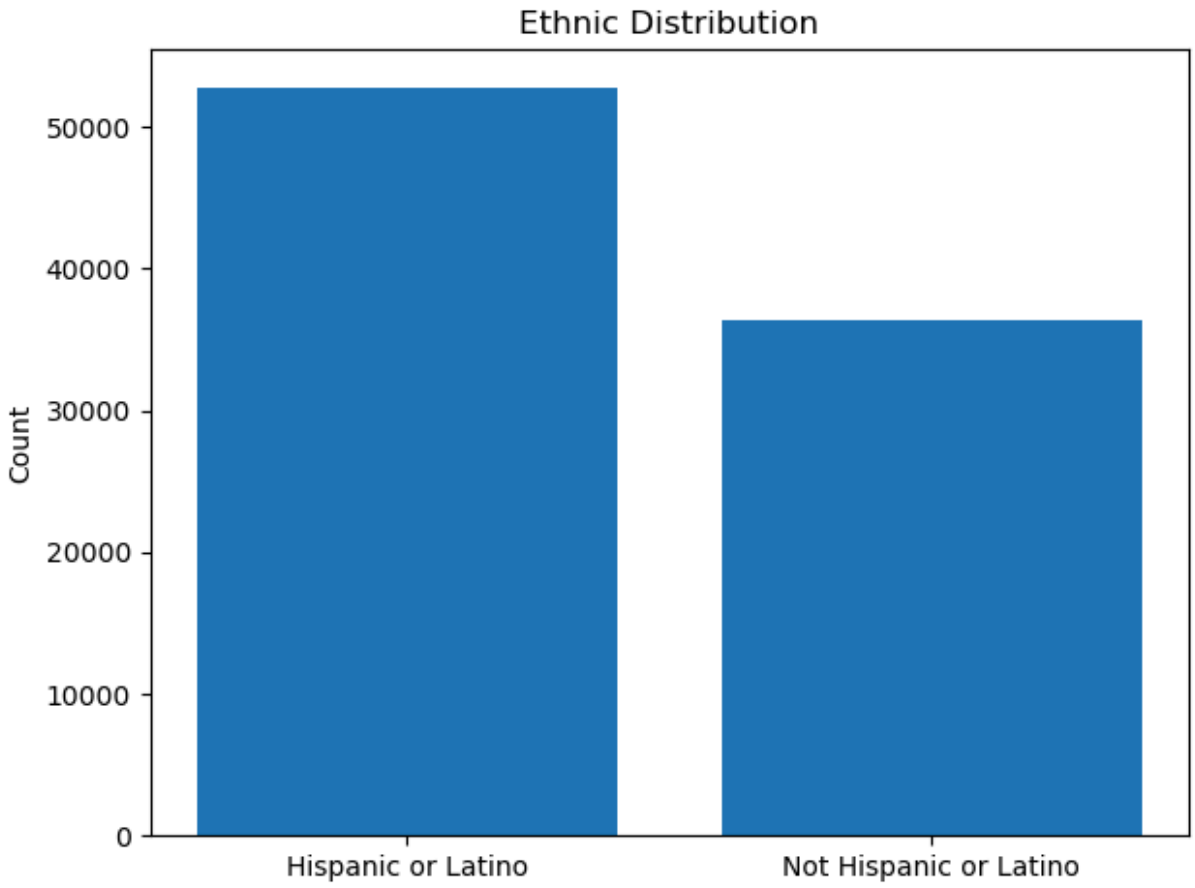
In [63]:
```
# Histogram of AGE
mapped = model_df["AGE"].map(age_map)

counts = mapped.value_counts().reindex(age_map.values())[:14]

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Age Range")
plt.ylabel("Count")
plt.title("Age Range Distribution")
plt.tight_layout()
plt.show()
```



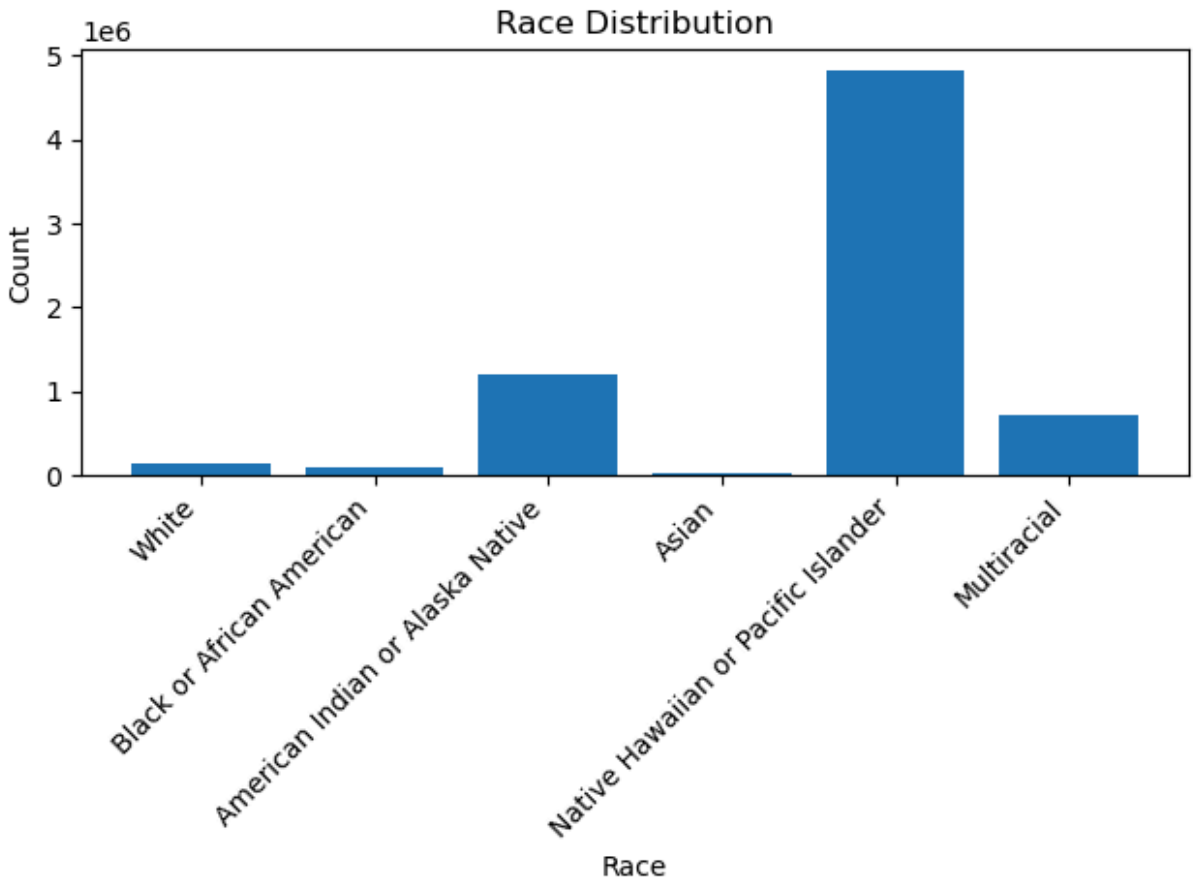
In [64]:
```
# Histogram of EDUC
mapped = model_df["EDUC"].map(educ_map)

counts = mapped.value_counts().reindex(educ_map.values())

plt.bar(counts.index, counts.values)
plt.ticklabel_format(style='plain', axis="y")
plt.xticks(rotation=45, ha="right")
plt.xlabel("Years of Education")
plt.ylabel("Count")
plt.title("Years of Education Distribution")
plt.tight_layout()
plt.show()
```



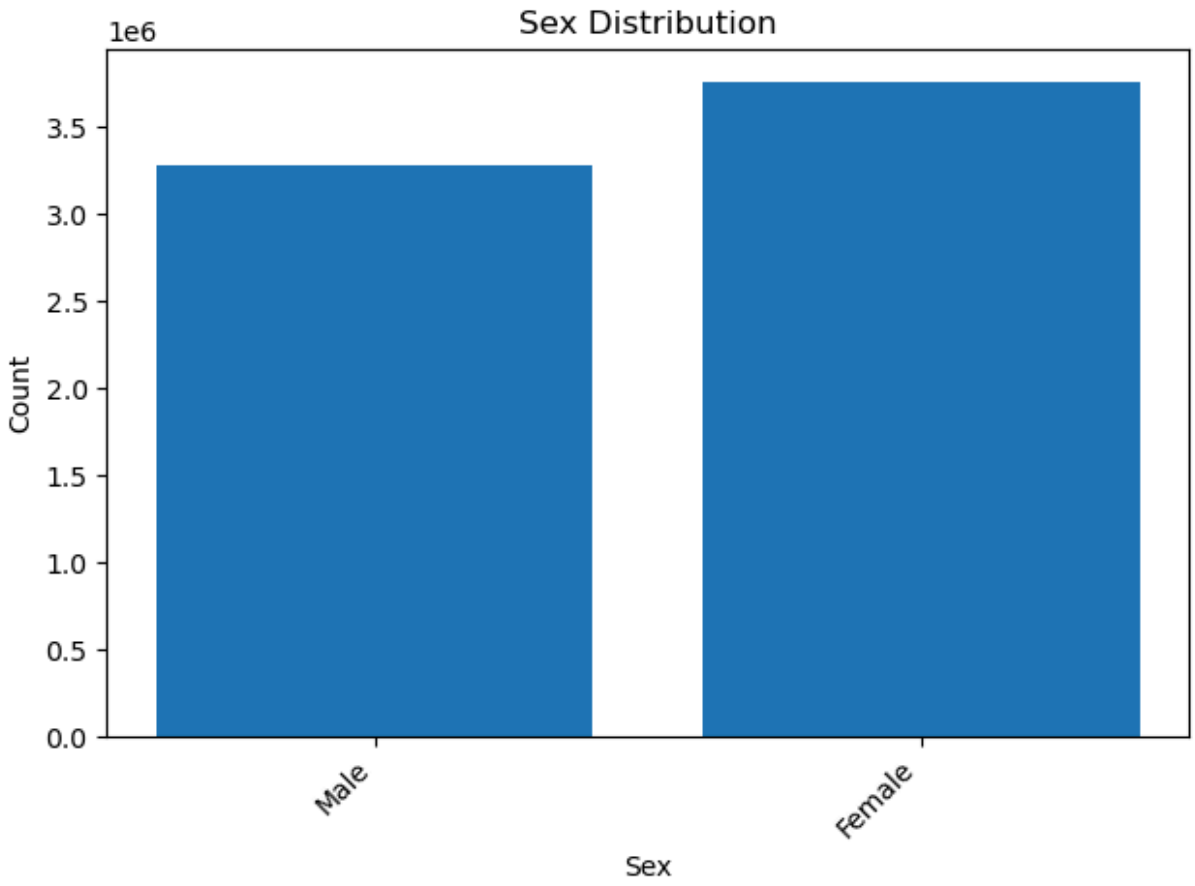
In [65]:
```
# Histogram of ETHNIC
mapped = model_df["ETHNIC"].map(ethnic_map)

counts = mapped.value_counts().reindex(ethnic_map.values())

plt.bar(counts.index, counts.values)
plt.ylabel("Count")
plt.title("Ethnic Distribution")
plt.tight_layout()
plt.show()
```



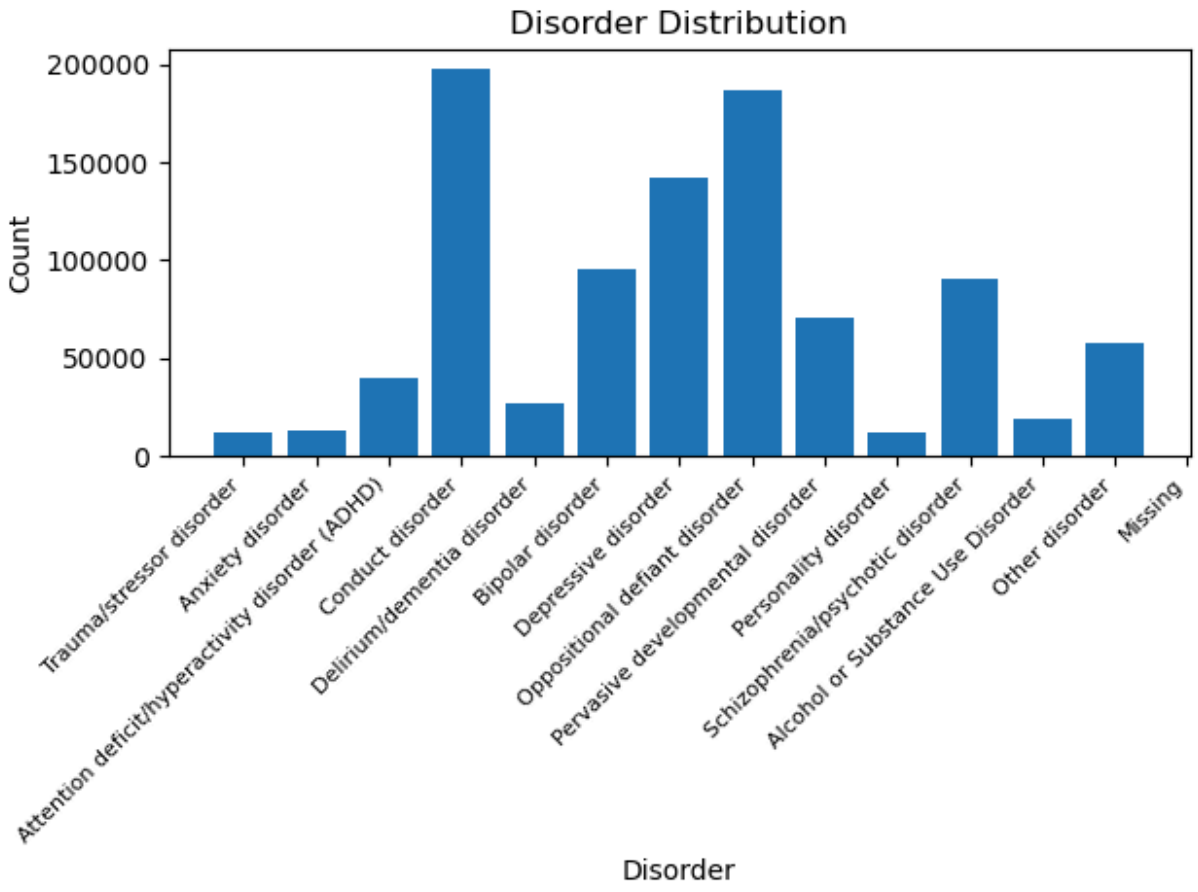
In [66]:
```
# Histogram of RACE
mapped = model_df["RACE"].map(race_map)

counts = mapped.value_counts().reindex(race_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Race")
plt.ylabel("Count")
plt.title("Race Distribution")
plt.tight_layout()
plt.show()
```



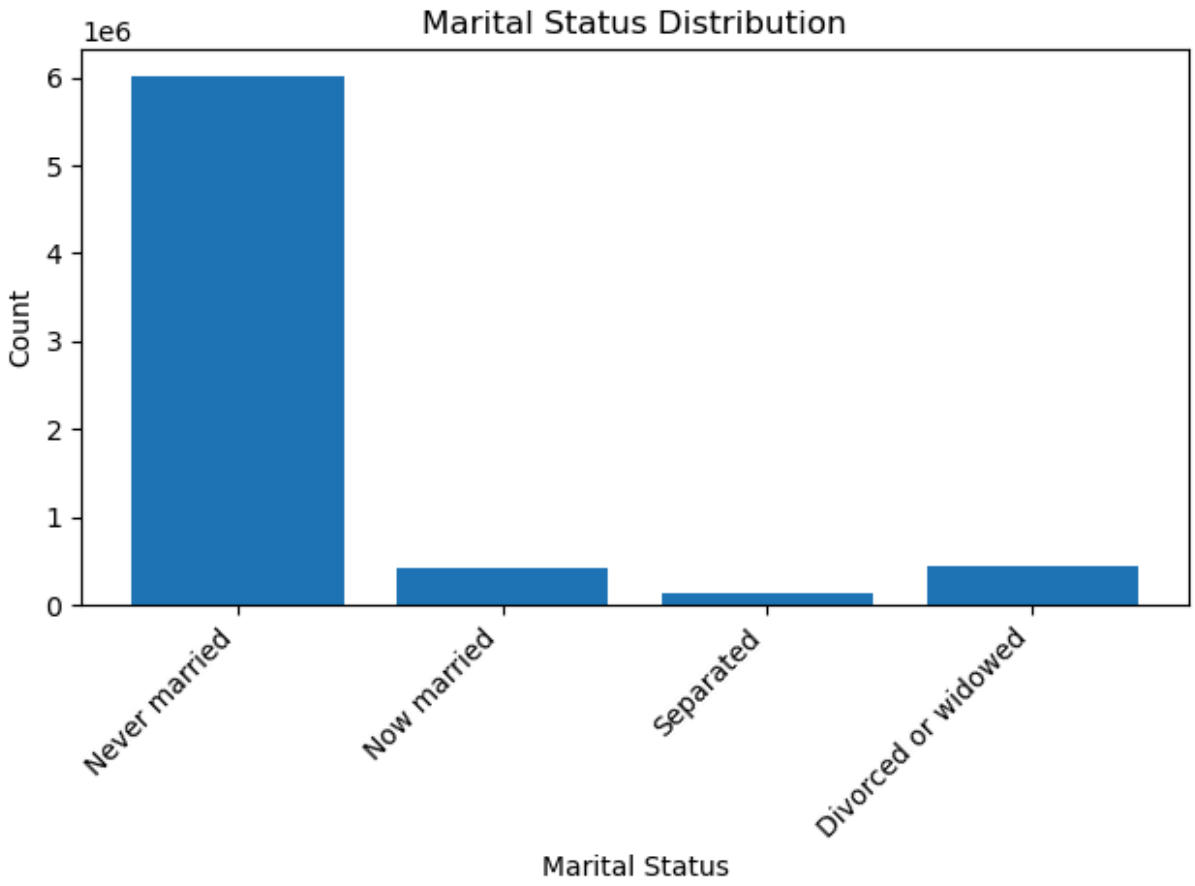
In [67]:
```
# Histogram of SEX
mapped = model_df["SEX"].map(sex_map)

counts = mapped.value_counts().reindex(sex_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Sex")
plt.ylabel("Count")
plt.title("Sex Distribution")
plt.tight_layout()
plt.show()
```



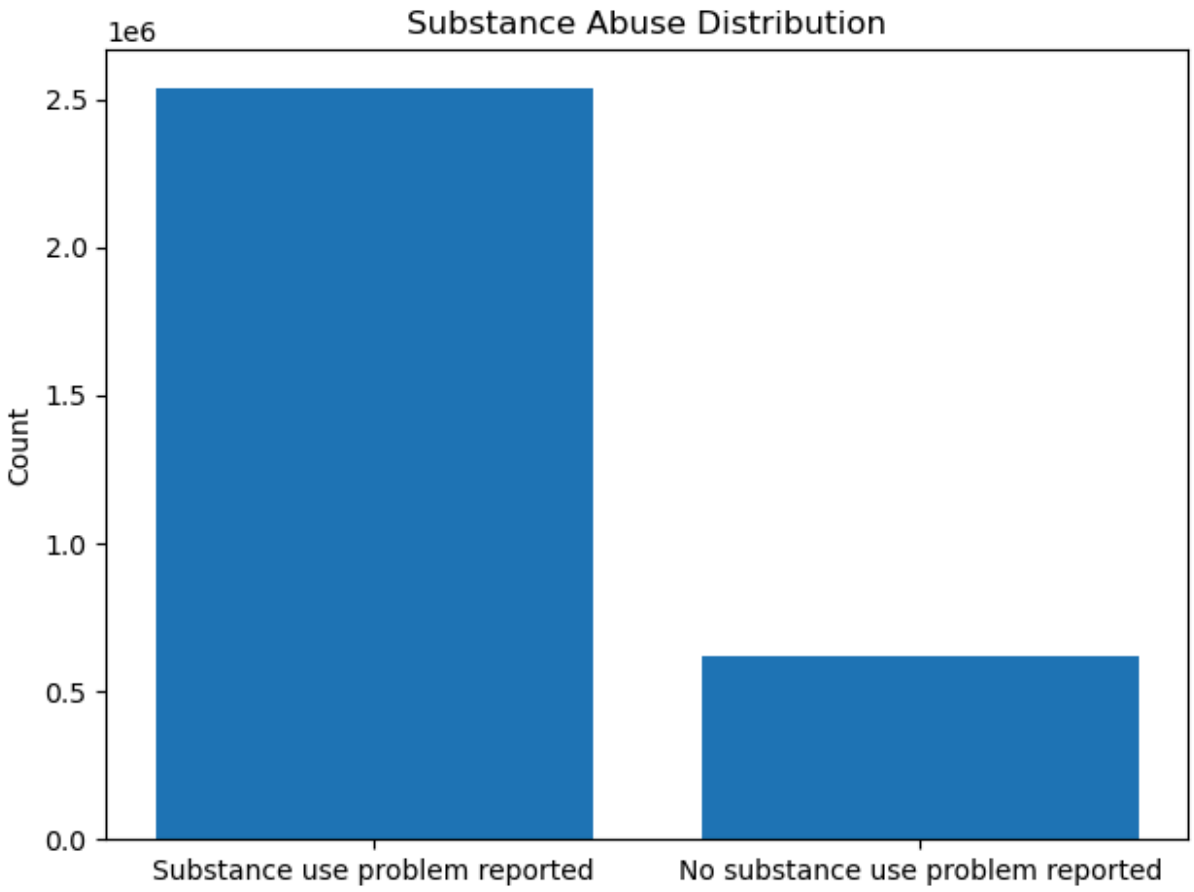
In [68]:
```
# Histogram of SUB

mapped = model_df["SUB"].map(sub_use_map)

counts = mapped.value_counts().reindex(sub_use_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right", fontsize=8)
plt.xlabel("Disorder")
plt.ylabel("Count")
plt.title("Disorder Distribution")
plt.tight_layout()
plt.show()
```



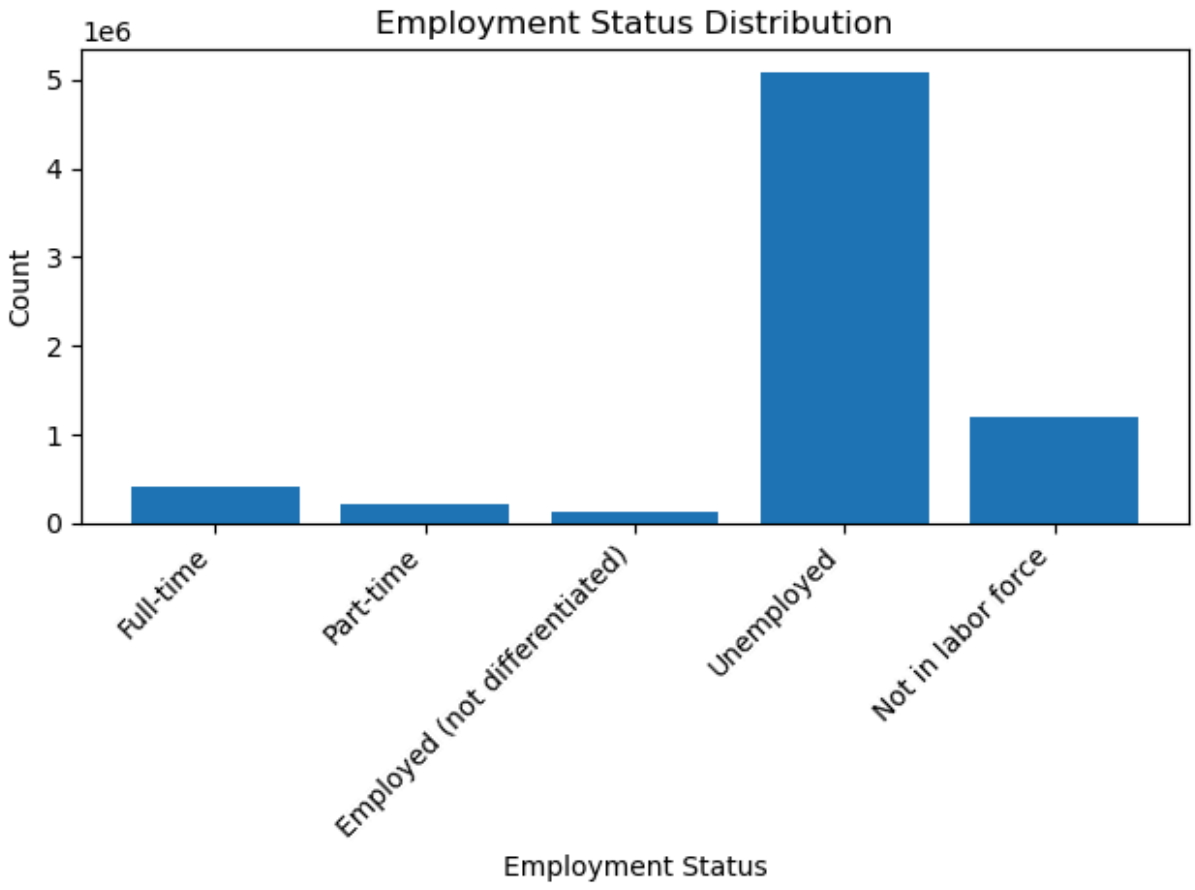
In [69]:
```
# Histogram of MARSTAT

mapped = model_df["MARSTAT"].map(marstat_map)

counts = mapped.value_counts().reindex(marstat_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Marital Status")
plt.ylabel("Count")
plt.title("Marital Status Distribution")
plt.tight_layout()
plt.show()
```



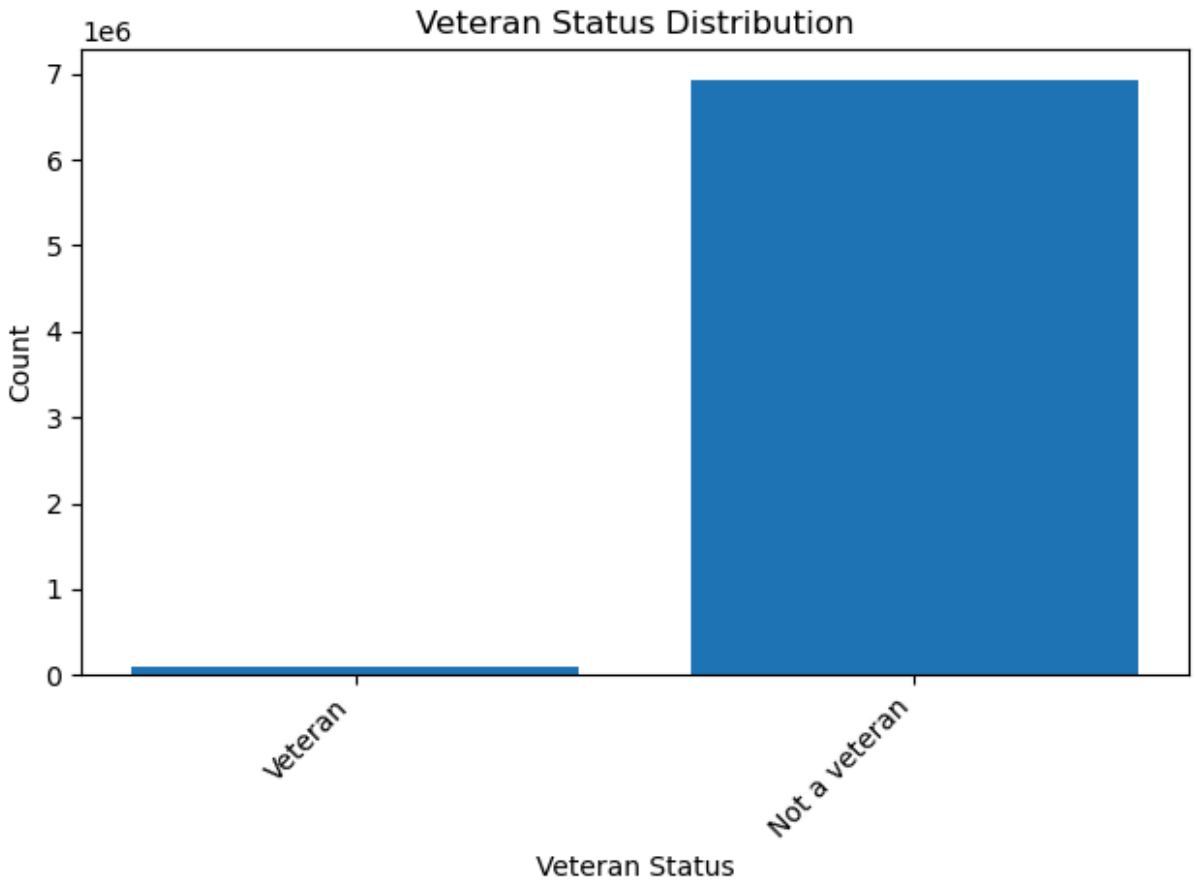
In [70]:
```
# Histogram of SAP
mapped = model_df["SAP"].map(sap_map)

counts = mapped.value_counts().reindex(sap_map.values())

plt.bar(counts.index, counts.values)
plt.xlabel("")
plt.ylabel("Count")
plt.title("Substance Abuse Distribution")
plt.tight_layout()
plt.show()
```



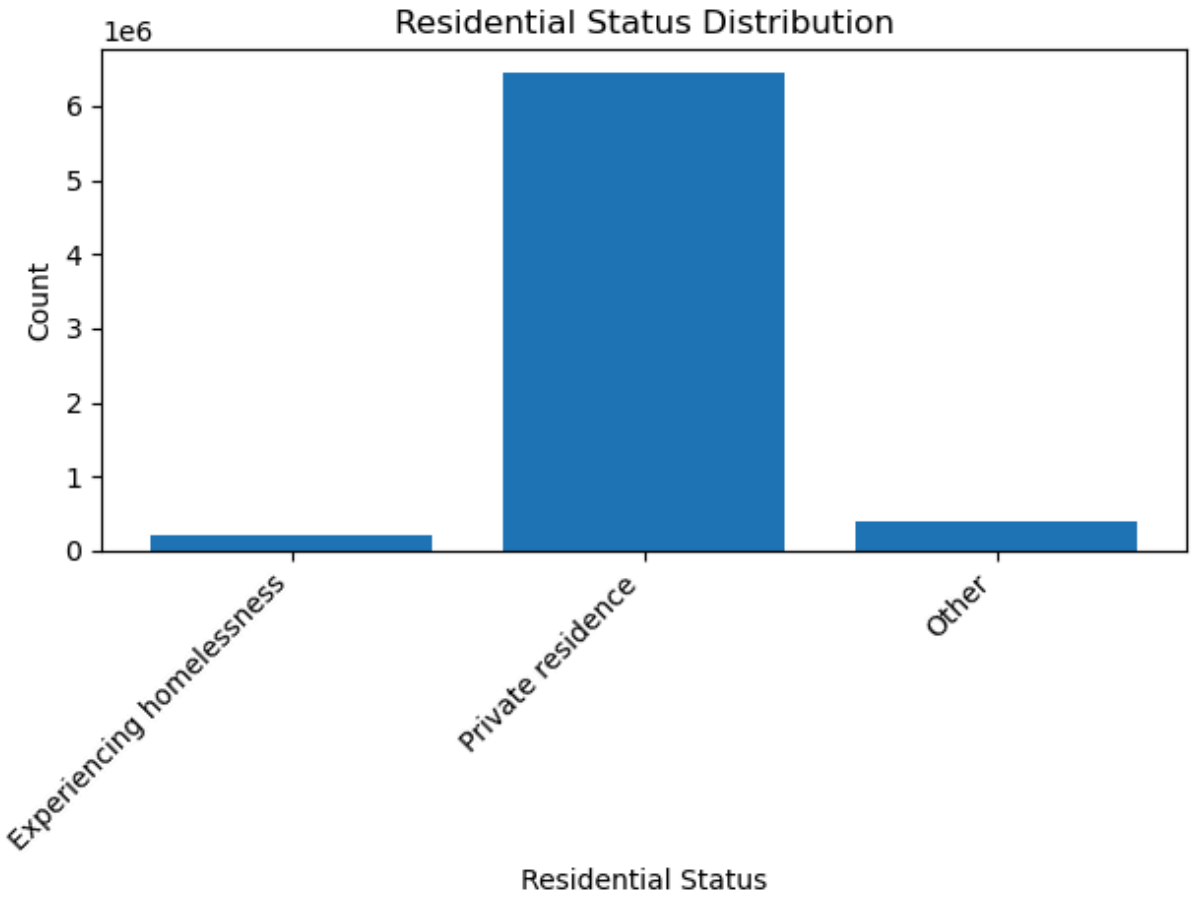
In [71]:
```
# Histogram of EMPLOY

mapped = model_df["EMPLOY"].map(employ_map)

counts = mapped.value_counts().reindex(employ_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Employment Status")
plt.ylabel("Count")
plt.title("Employment Status Distribution")
plt.tight_layout()
plt.show()
```



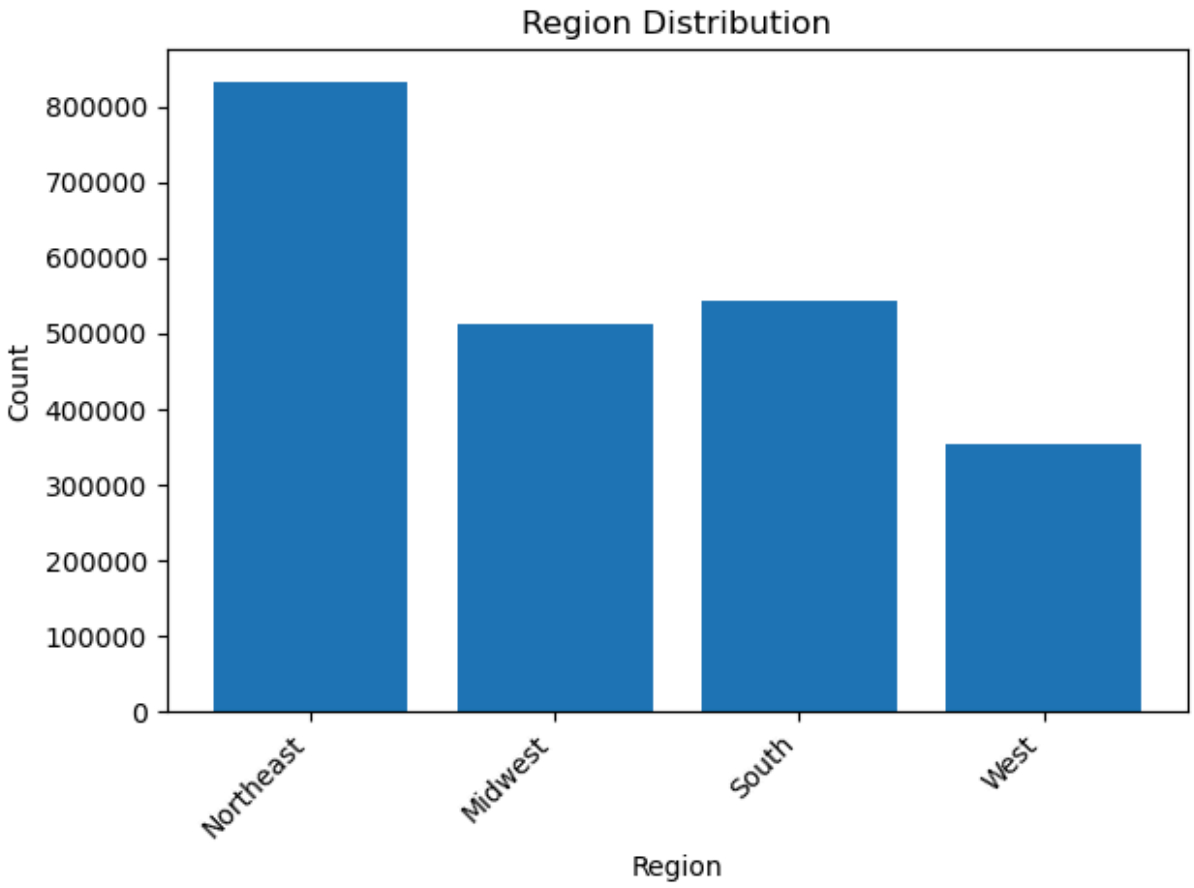
In [72]:
```
# Histogram of VETERAN

mapped = model_df["VETERAN"].map(veteran_map)

counts = mapped.value_counts().reindex(veteran_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Veteran Status")
plt.ylabel("Count")
plt.title("Veteran Status Distribution")
plt.tight_layout()
plt.show()
```



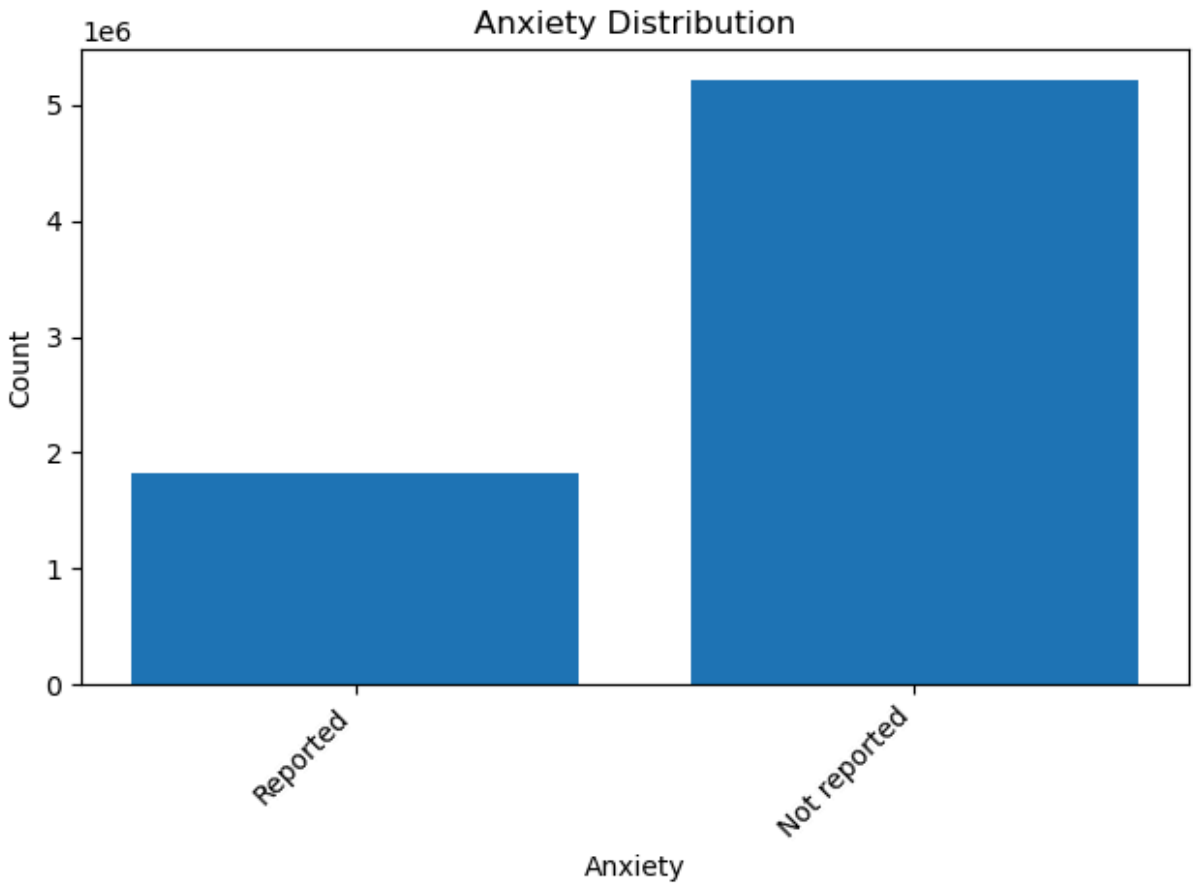
In [73]:
```
# Histogram of LIVARAG

mapped = model_df["LIVARAG"].map(livarag_map)

counts = mapped.value_counts().reindex(livarag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Residential Status")
plt.ylabel("Count")
plt.title("Residential Status Distribution")
plt.tight_layout()
plt.show()
```



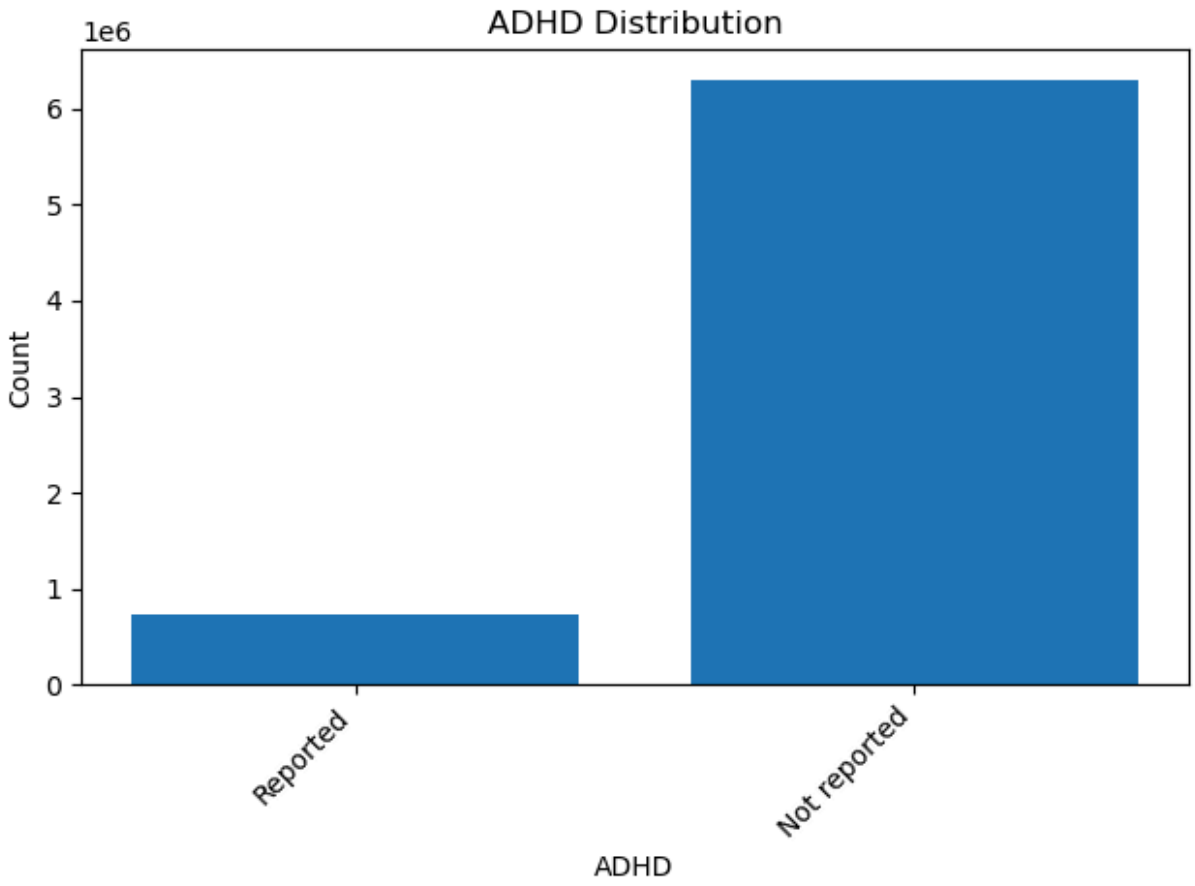
In [74]:
```
# Histogram of REGION

mapped = model_df["AGE"].map(region_map)

counts = mapped.value_counts().reindex(region_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Region")
plt.ylabel("Count")
plt.title("Region Distribution")
plt.tight_layout()
plt.show()
```



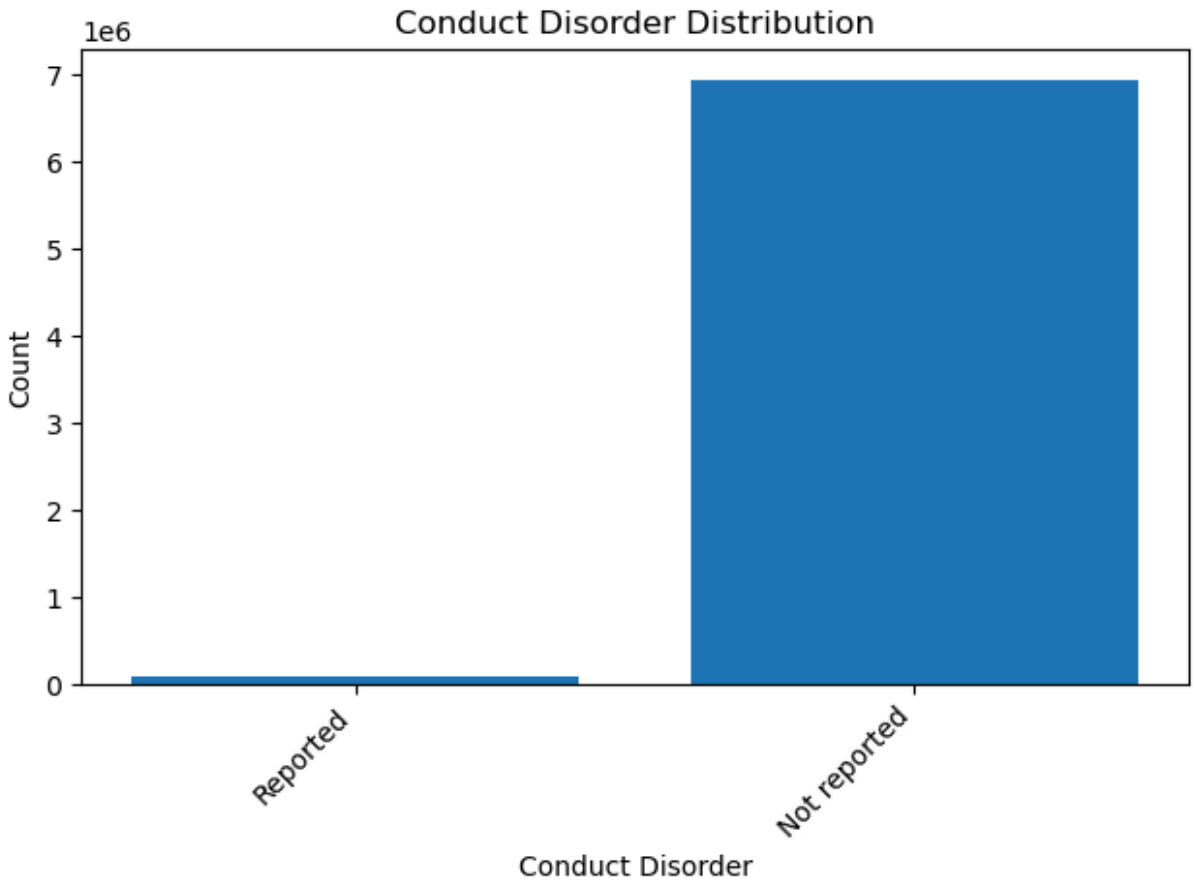
In [75]:
```
# Histogram of ANXIETYFLG

mapped = model_df["ANXIETYFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Anxiety")
plt.ylabel("Count")
plt.title("Anxiety Distribution")
plt.tight_layout()
plt.show()
```



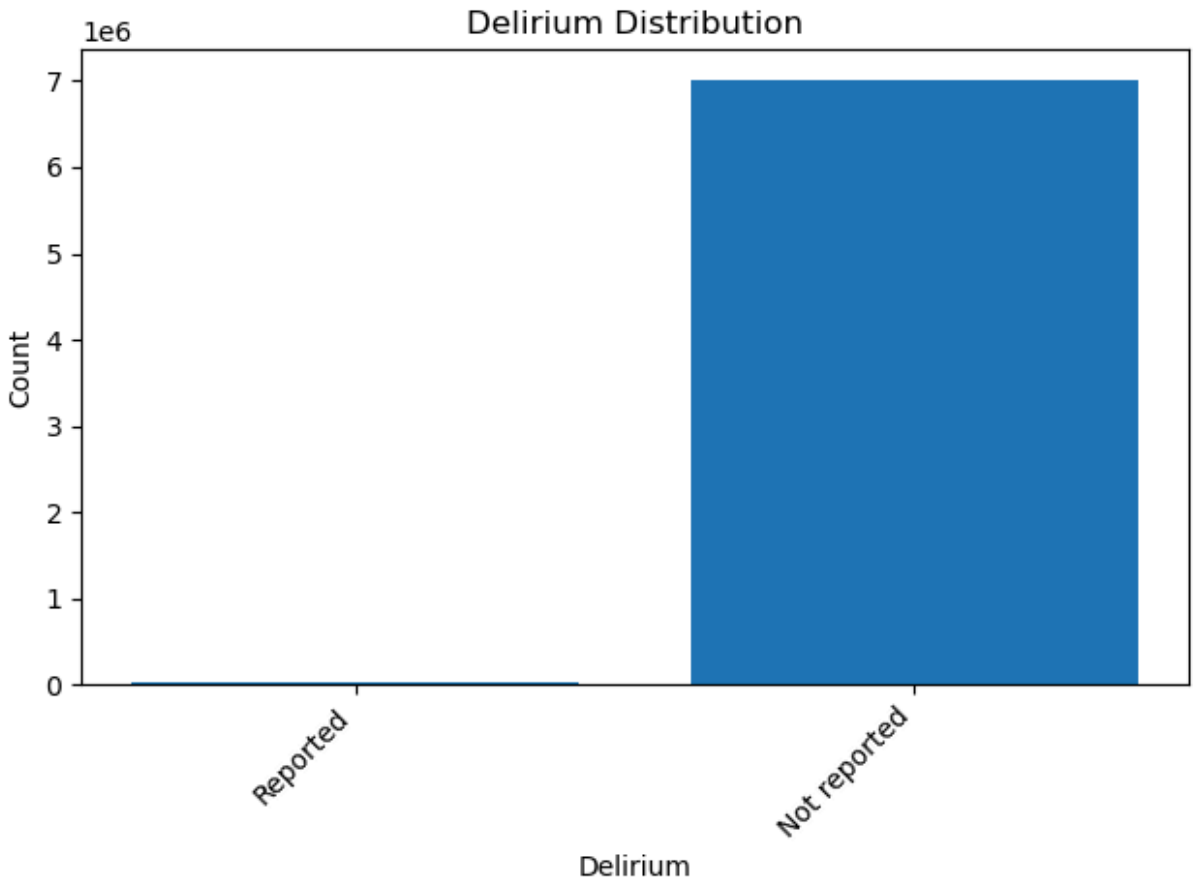
In [76]:
```
# Histogram of ADHDFLG

mapped = model_df["ADHDFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("ADHD")
plt.ylabel("Count")
plt.title("ADHD Distribution")
plt.tight_layout()
plt.show()
```



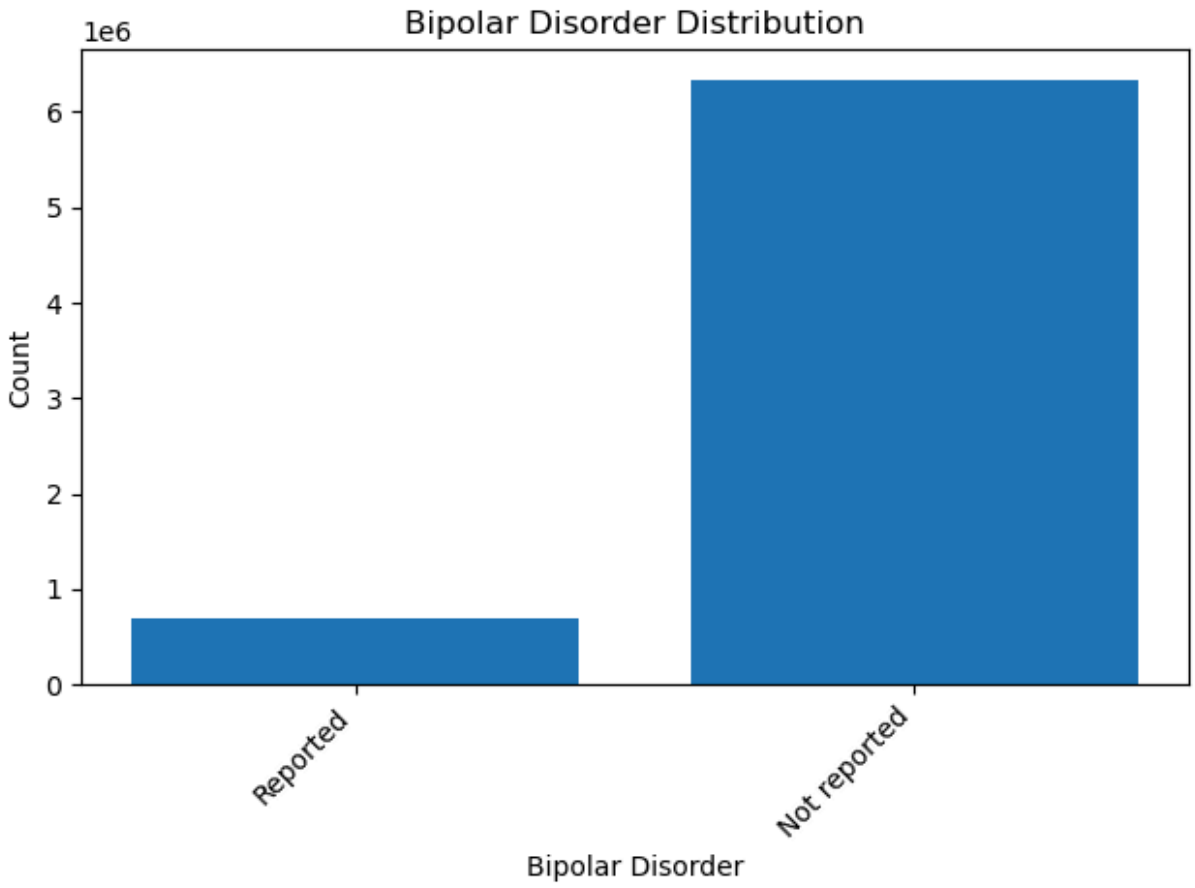
In [77]:
```
# Histogram of CONDUCTFLG

mapped = model_df["CONDUCTFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Conduct Disorder")
plt.ylabel("Count")
plt.title("Conduct Disorder Distribution")
plt.tight_layout()
plt.show()
```



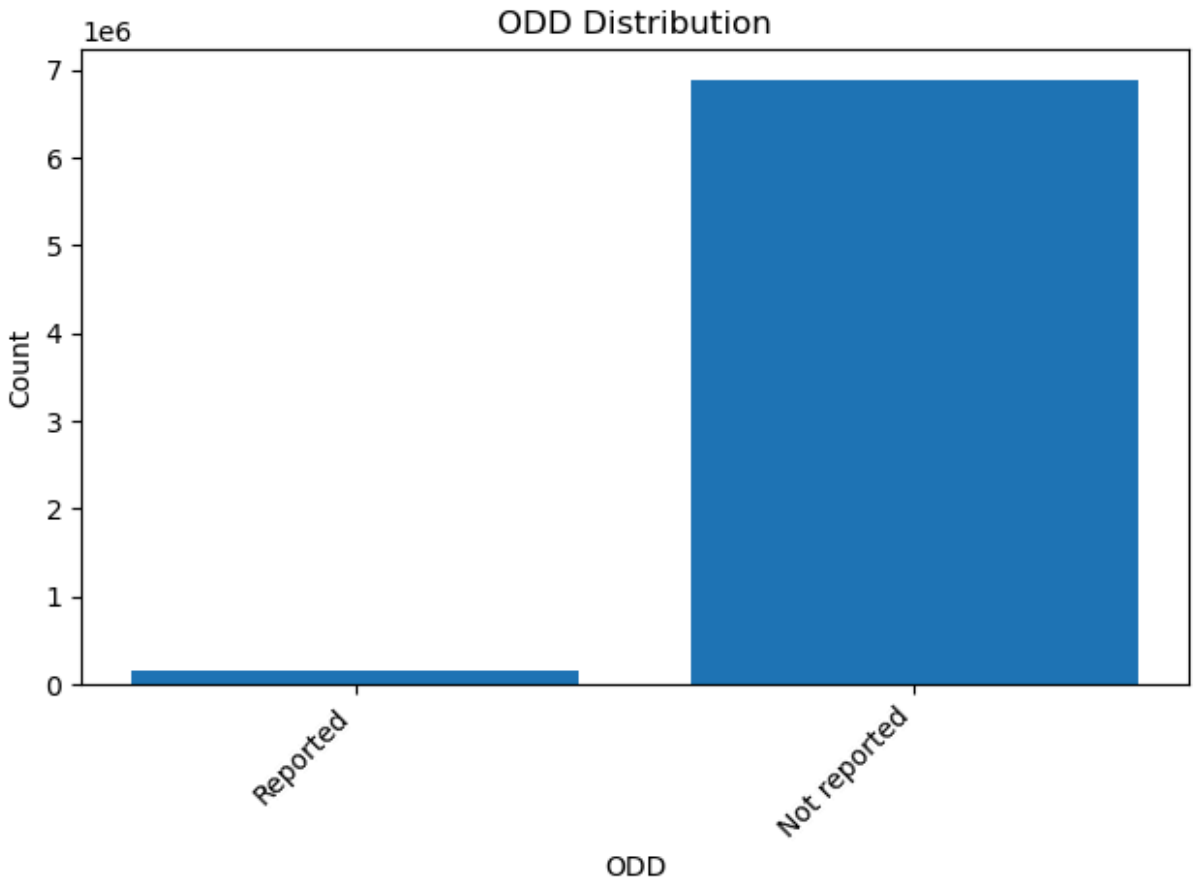
In [78]:
```
# Histogram of DELIRDEMFLG

mapped = model_df["DELIRDEMFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Delirium")
plt.ylabel("Count")
plt.title("Delirium Distribution")
plt.tight_layout()
plt.show()
```



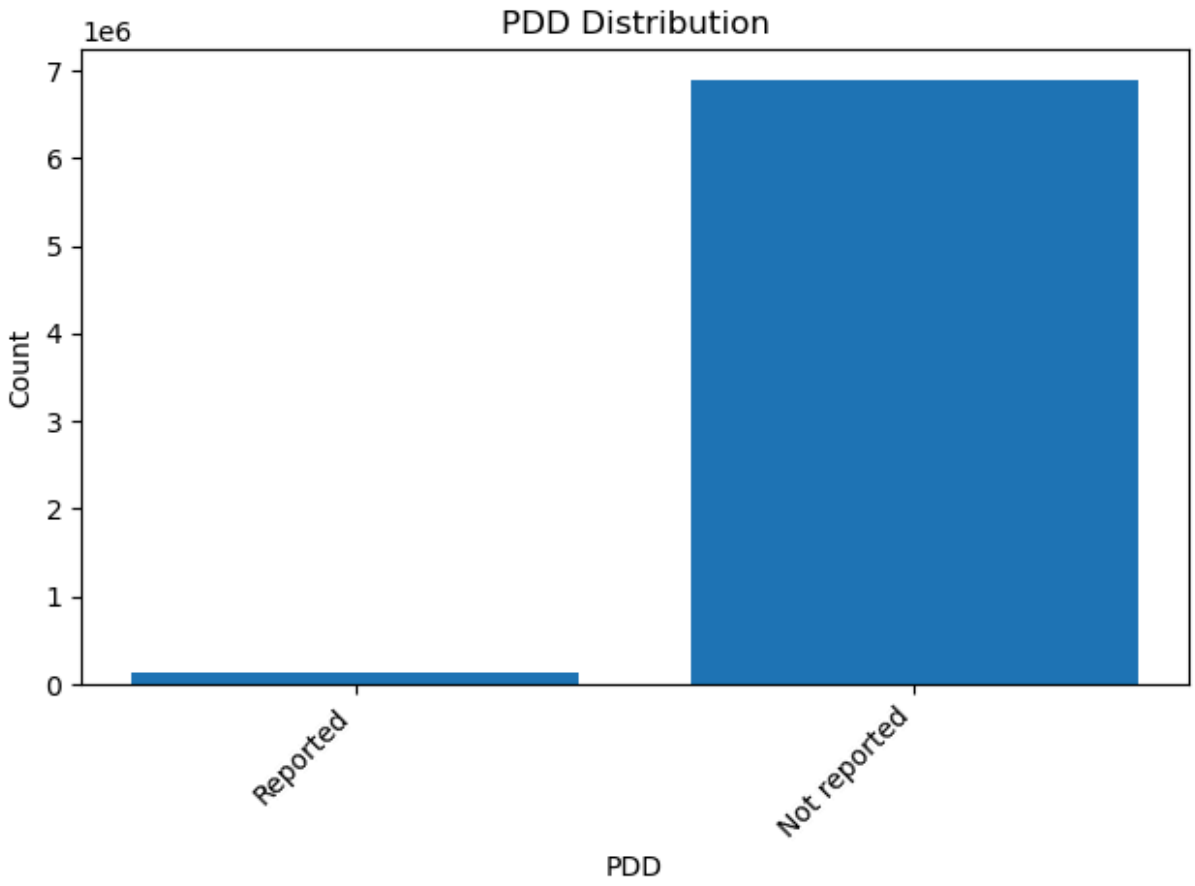
In [79]:
```
# Histogram of BIPOLARFLG

mapped = model_df["BIPOLARFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Bipolar Disorder")
plt.ylabel("Count")
plt.title("Bipolar Disorder Distribution")
plt.tight_layout()
plt.show()
```



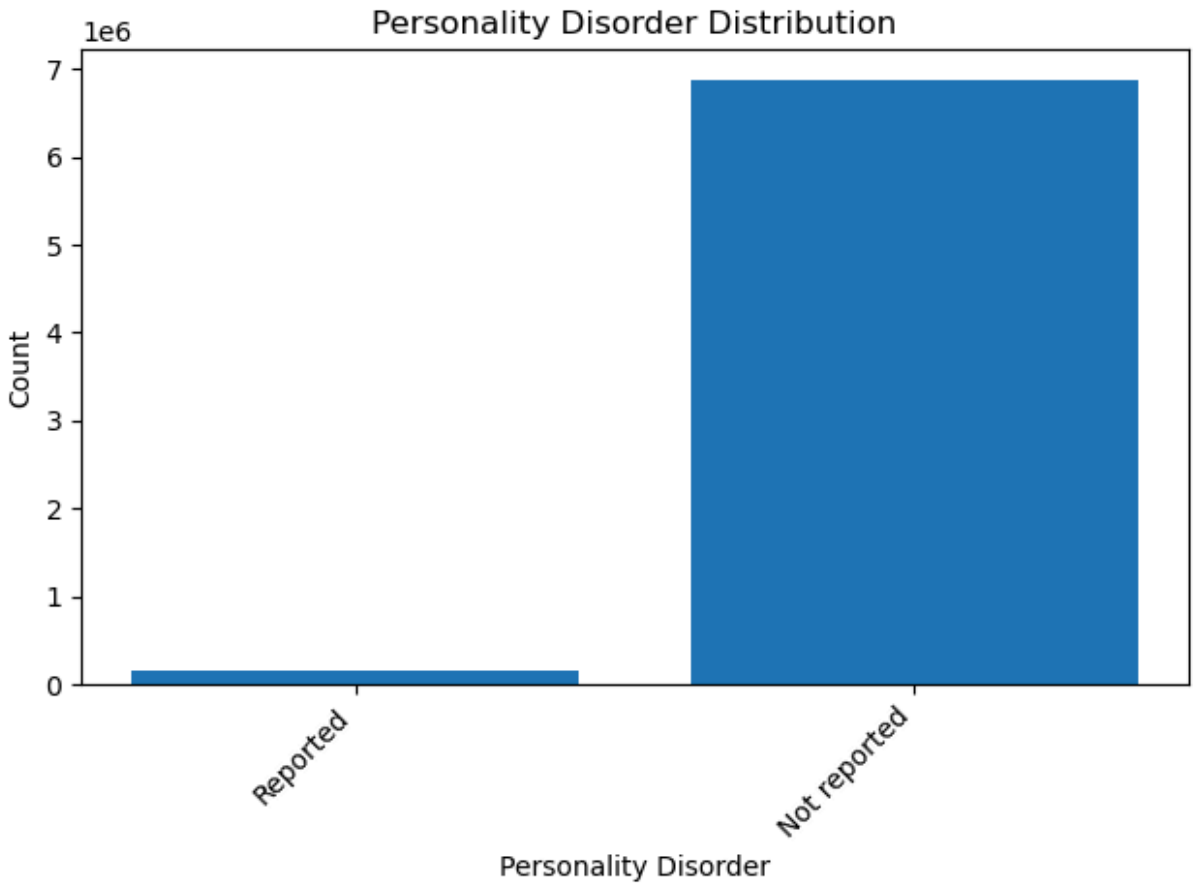
In [80]:
```
# Histogram of ODDFLG

mapped = model_df["ODDFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("ODD")
plt.ylabel("Count")
plt.title("ODD Distribution")
plt.tight_layout()
plt.show()
```



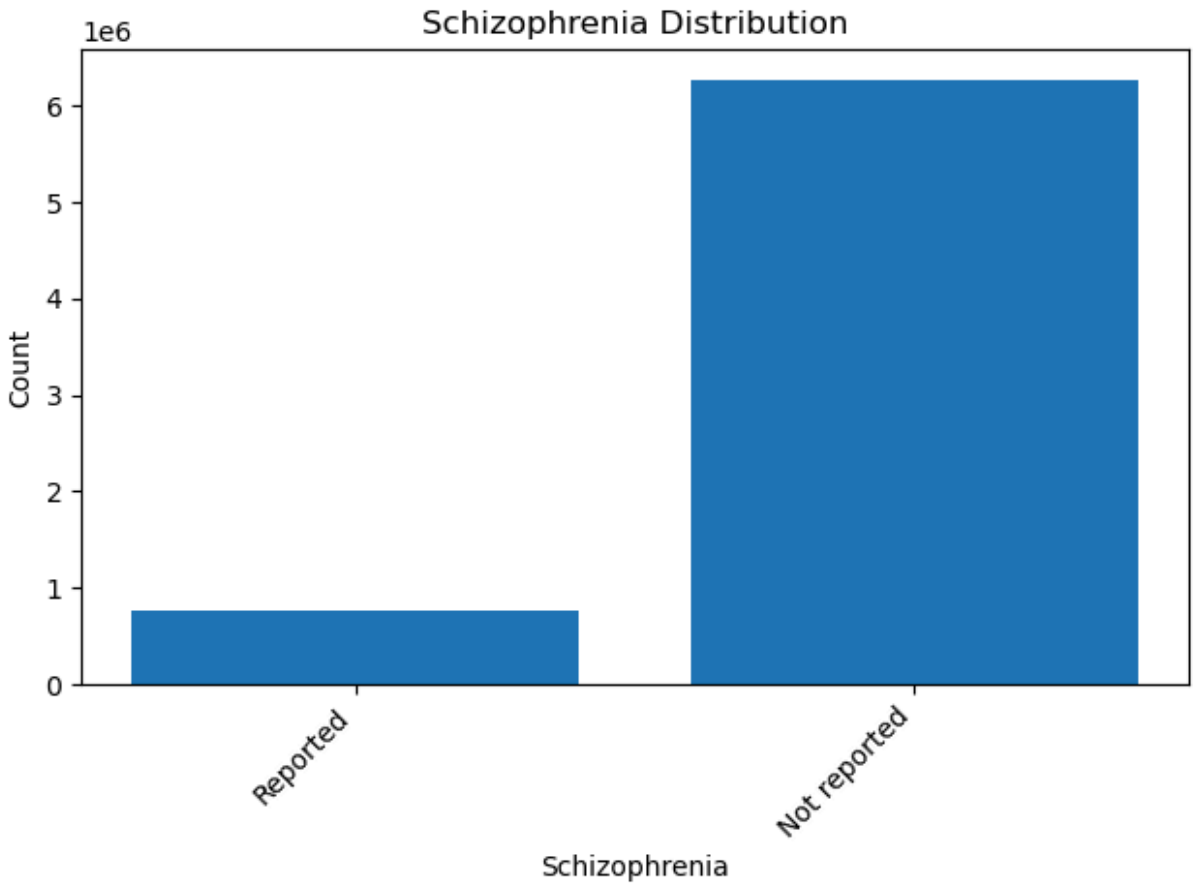
In [81]:
```
# Histogram of PDDFLG

mapped = model_df["PDDFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("PDD")
plt.ylabel("Count")
plt.title("PDD Distribution")
plt.tight_layout()
plt.show()
```



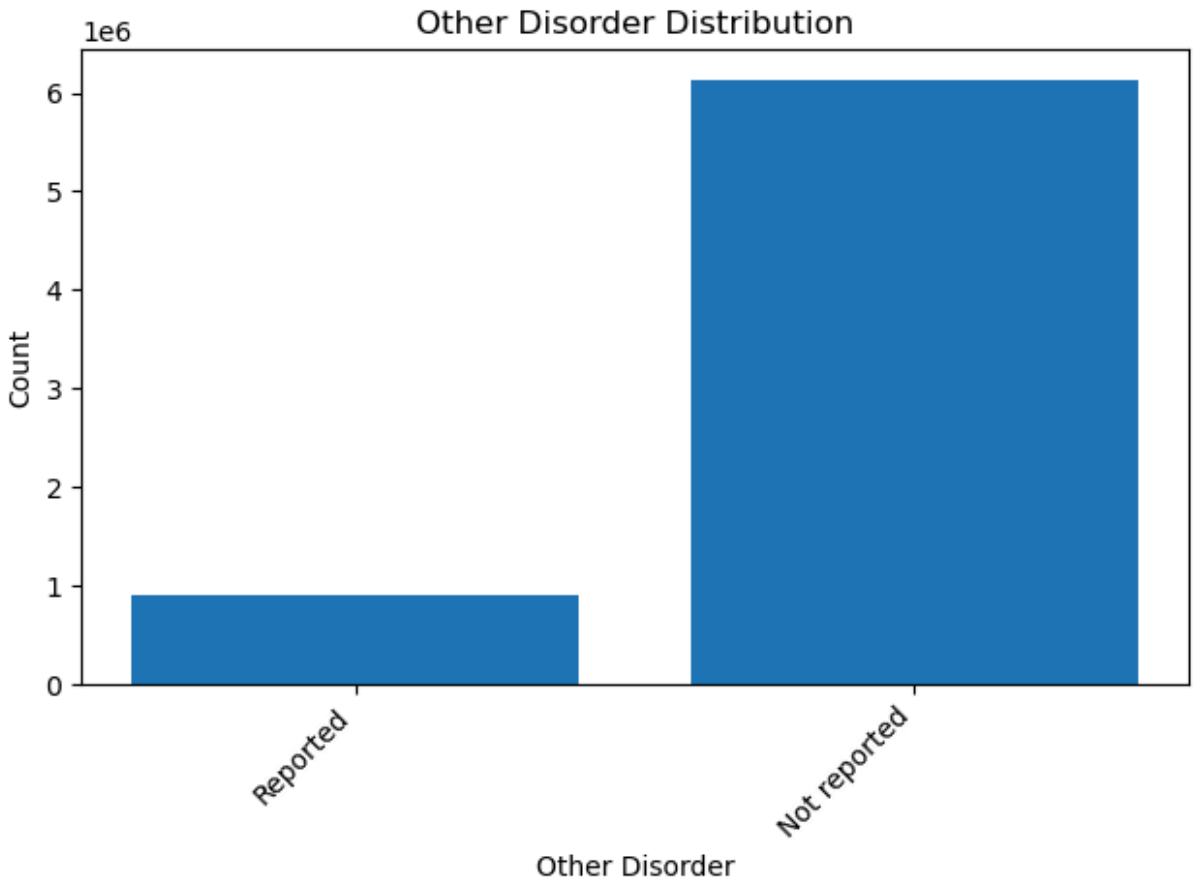
In [82]:
```
# Histogram of PERSONFLG

mapped = model_df["PERSONFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Personality Disorder")
plt.ylabel("Count")
plt.title("Personality Disorder Distribution")
plt.tight_layout()
plt.show()
```



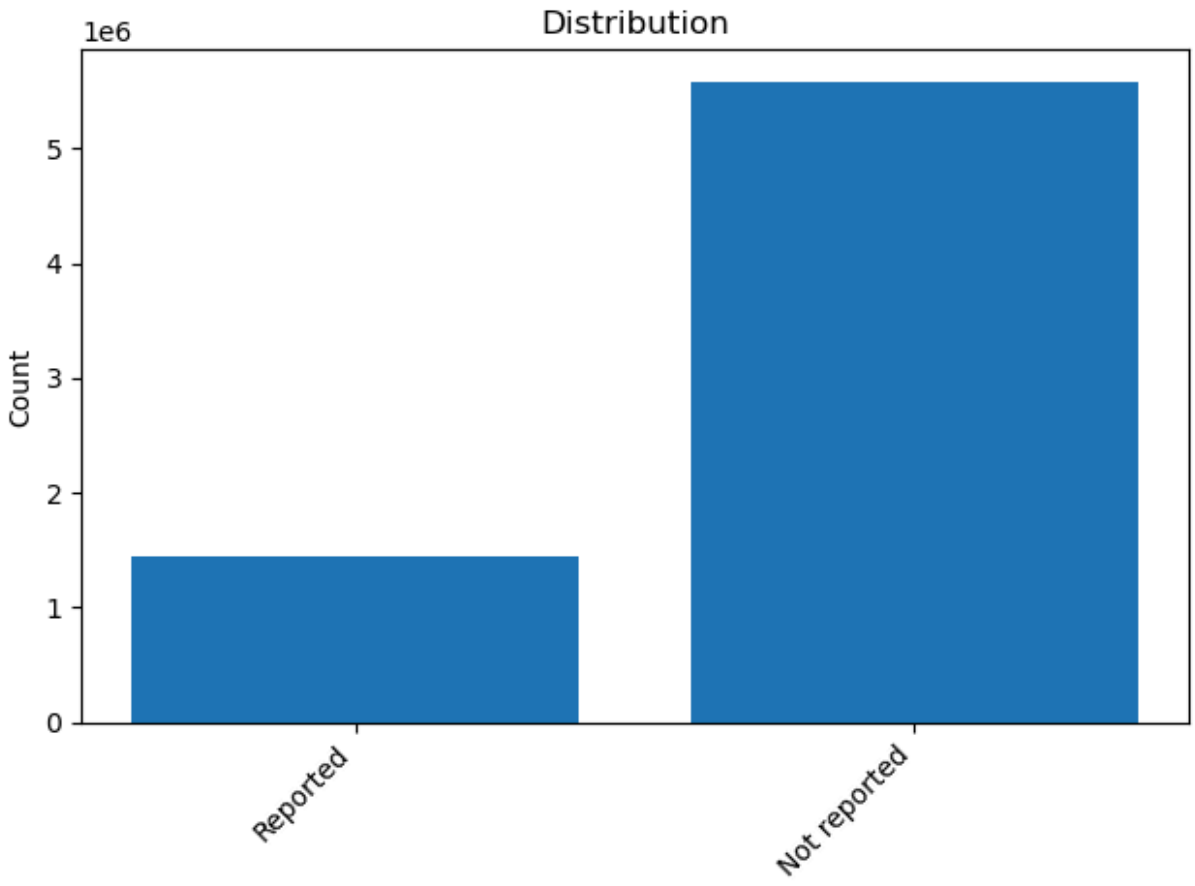
In [83]:
```
# Histogram of SCHIZOFLG

mapped = model_df["SCHIZOFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Schizophrenia")
plt.ylabel("Count")
plt.title("Schizophrenia Distribution")
plt.tight_layout()
plt.show()
```



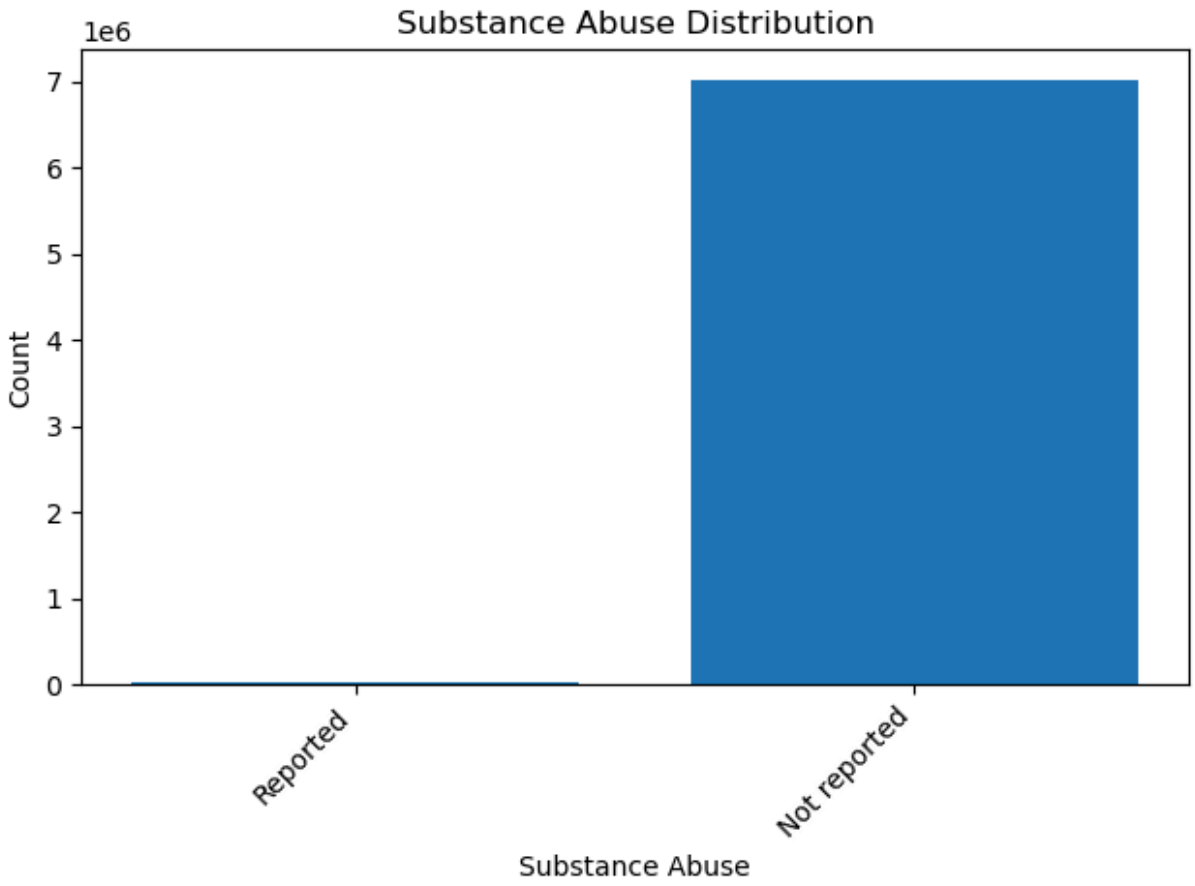
In [84]:
```
# Histogram of OTHERDISFLG

mapped = model_df["OTHERDISFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Other Disorder")
plt.ylabel("Count")
plt.title("Other Disorder Distribution")
plt.tight_layout()
plt.show()
```



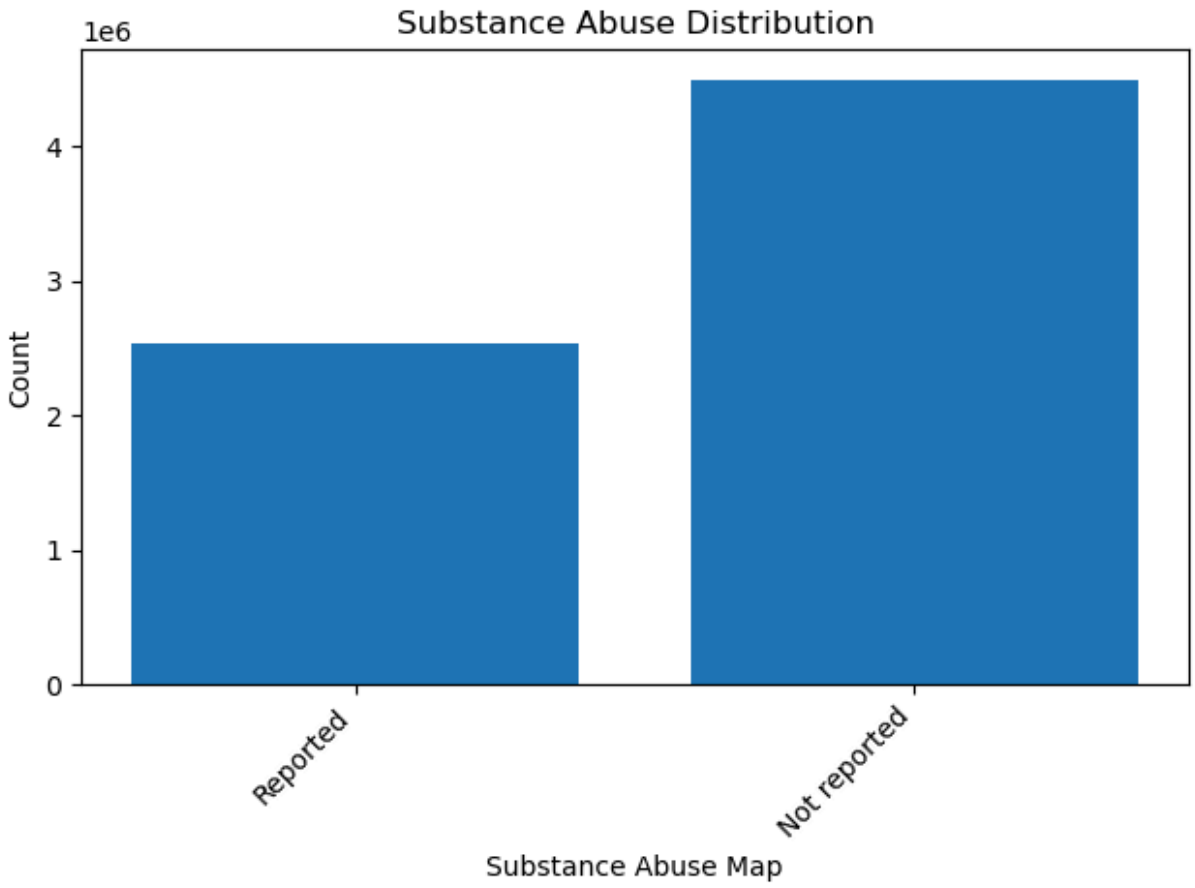
In [85]:
```
# Histogram of TRAUSTREFLG

mapped = model_df["TRAUSTREFLG"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("")
plt.ylabel("Count")
plt.title("Distribution")
plt.tight_layout()
plt.show()
```



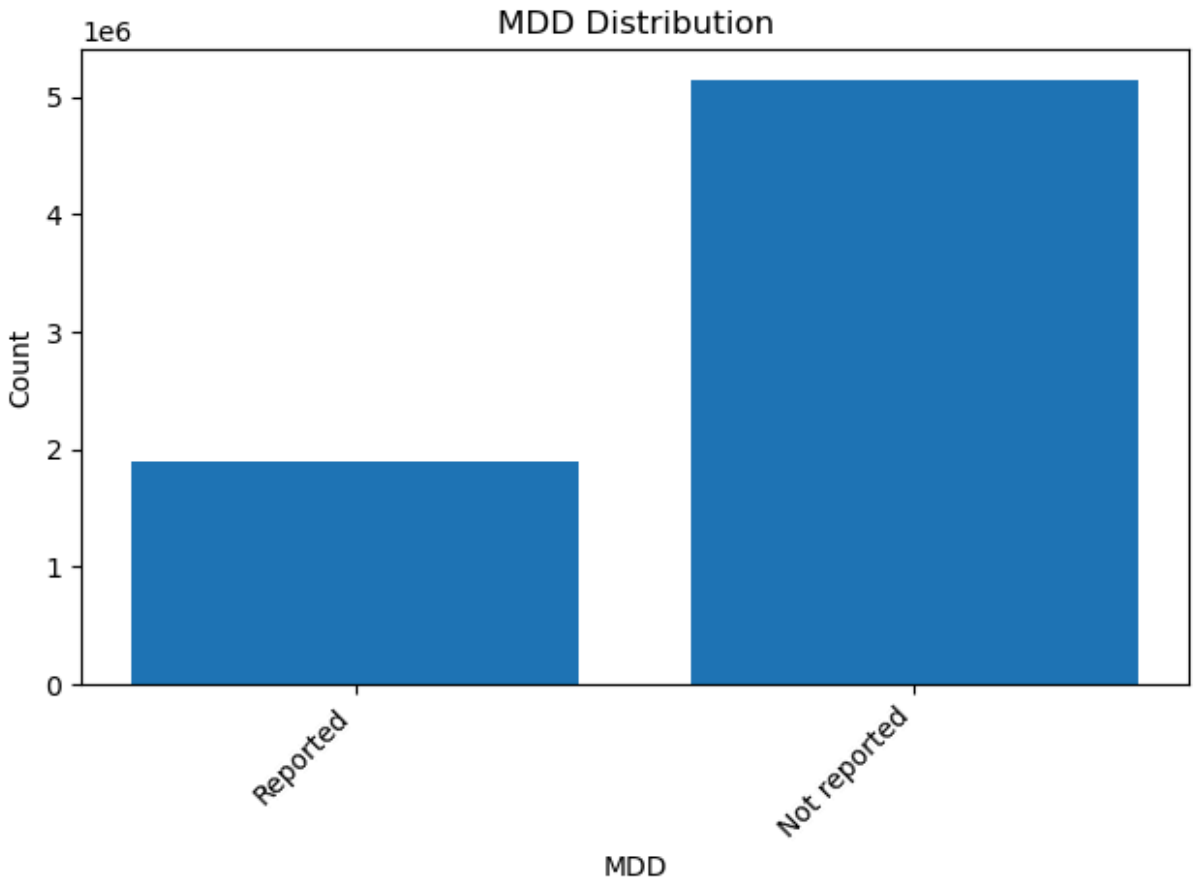
In [86]:
```
# Histogram of HAS_SUBSTANCE_USE

mapped = model_df["HAS_SUBSTANCE_USE"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Substance Abuse")
plt.ylabel("Count")
plt.title("Substance Abuse Distribution")
plt.tight_layout()
plt.show()
```



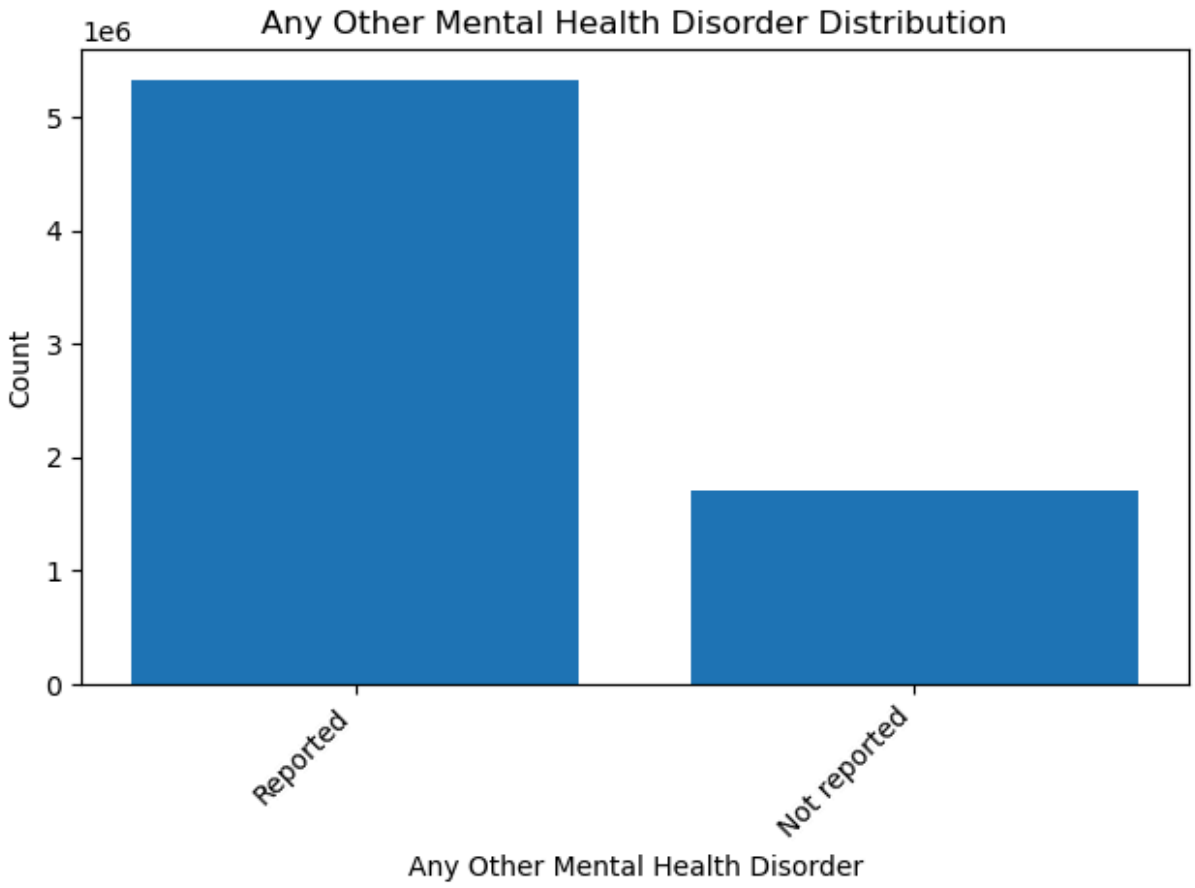
In [87]:
```
# Histogram of HAS_SAP

mapped = model_df["HAS_SAP"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Substance Abuse Map")
plt.ylabel("Count")
plt.title("Substance Abuse Distribution")
plt.tight_layout()
plt.show()
```



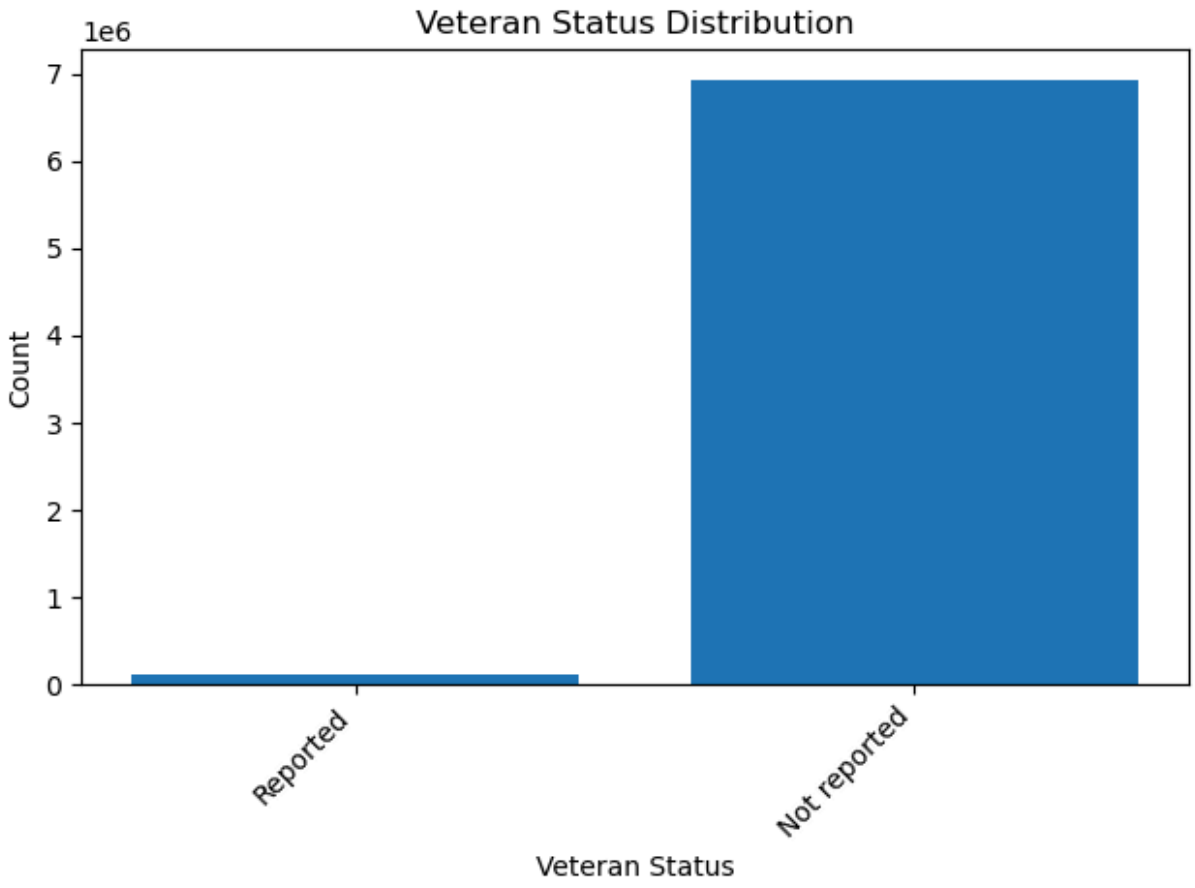
In [88]:
```
# Histogram of MDD

mapped = model_df["MDD"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("MDD")
plt.ylabel("Count")
plt.title("MDD Distribution")
plt.tight_layout()
plt.show()
```



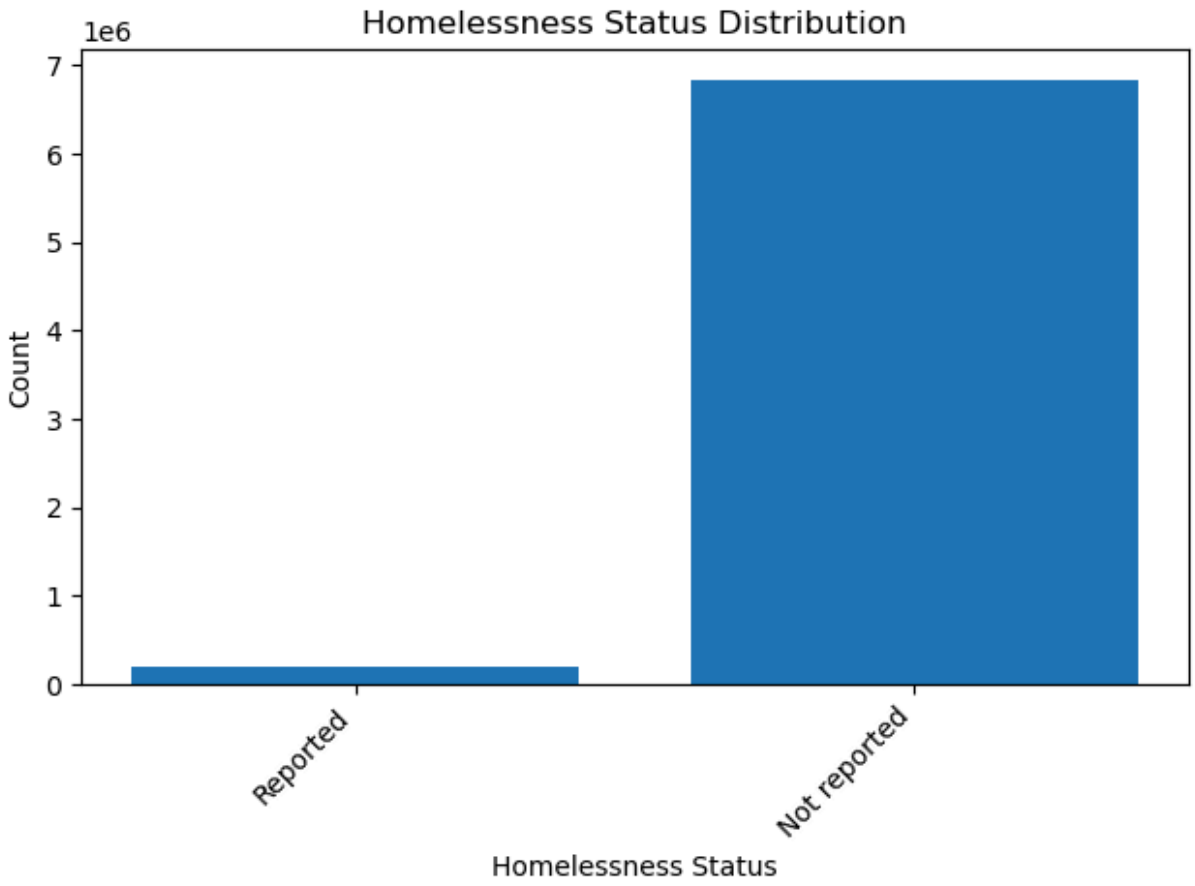
In [89]:
```
# Histogram of ANY_OTHER_MH_DISORDER

mapped = model_df["ANY_OTHER_MH_DISORDER"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Any Other Mental Health Disorder")
plt.ylabel("Count")
plt.title("Any Other Mental Health Disorder Distribution")
plt.tight_layout()
plt.show()
```



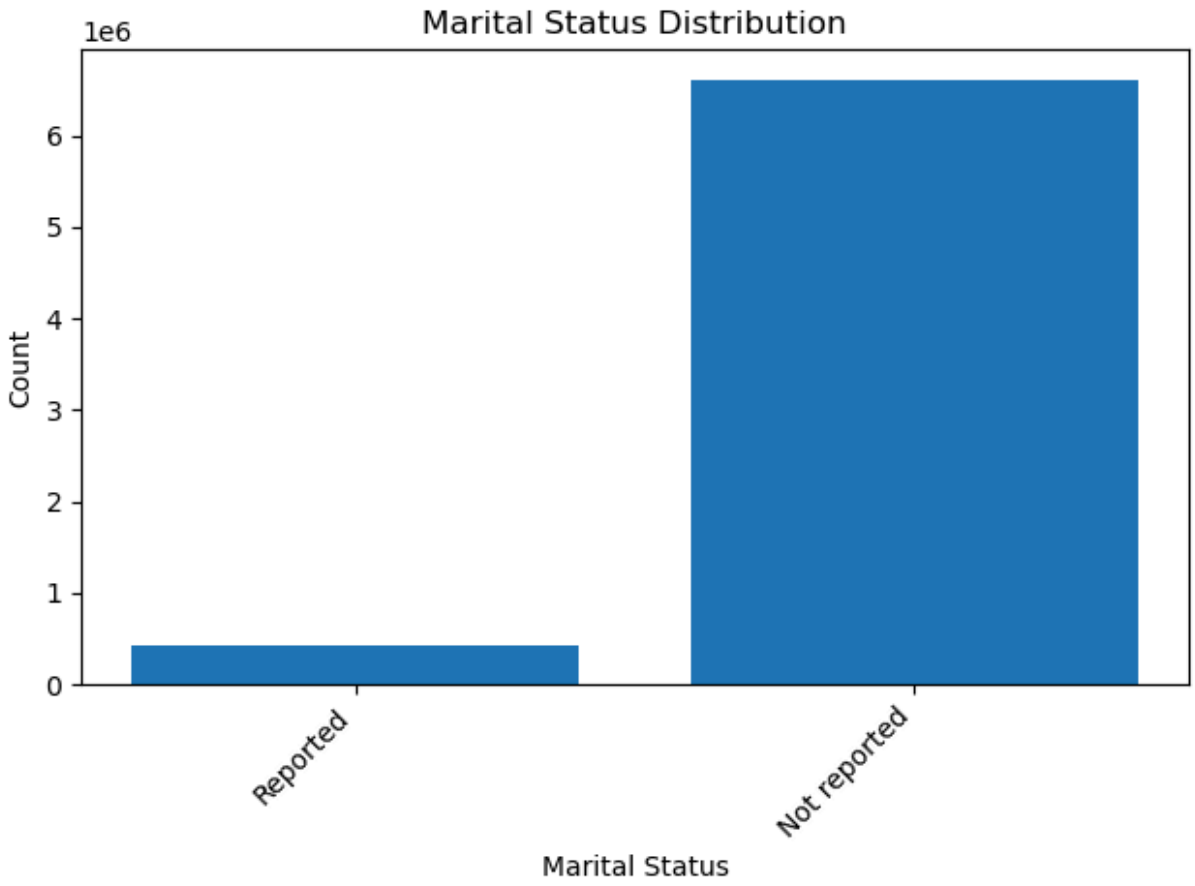
In [90]:
```
# Histogram of IS_VETERAN

mapped = model_df["IS_VETERAN"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Veteran Status")
plt.ylabel("Count")
plt.title("Veteran Status Distribution")
plt.tight_layout()
plt.show()
```



In [91]:
```
# Histogram of IS_HOMELESS

mapped = model_df["IS_HOMELESS"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Homelessness Status")
plt.ylabel("Count")
plt.title("Homelessness Status Distribution")
plt.tight_layout()
plt.show()
```



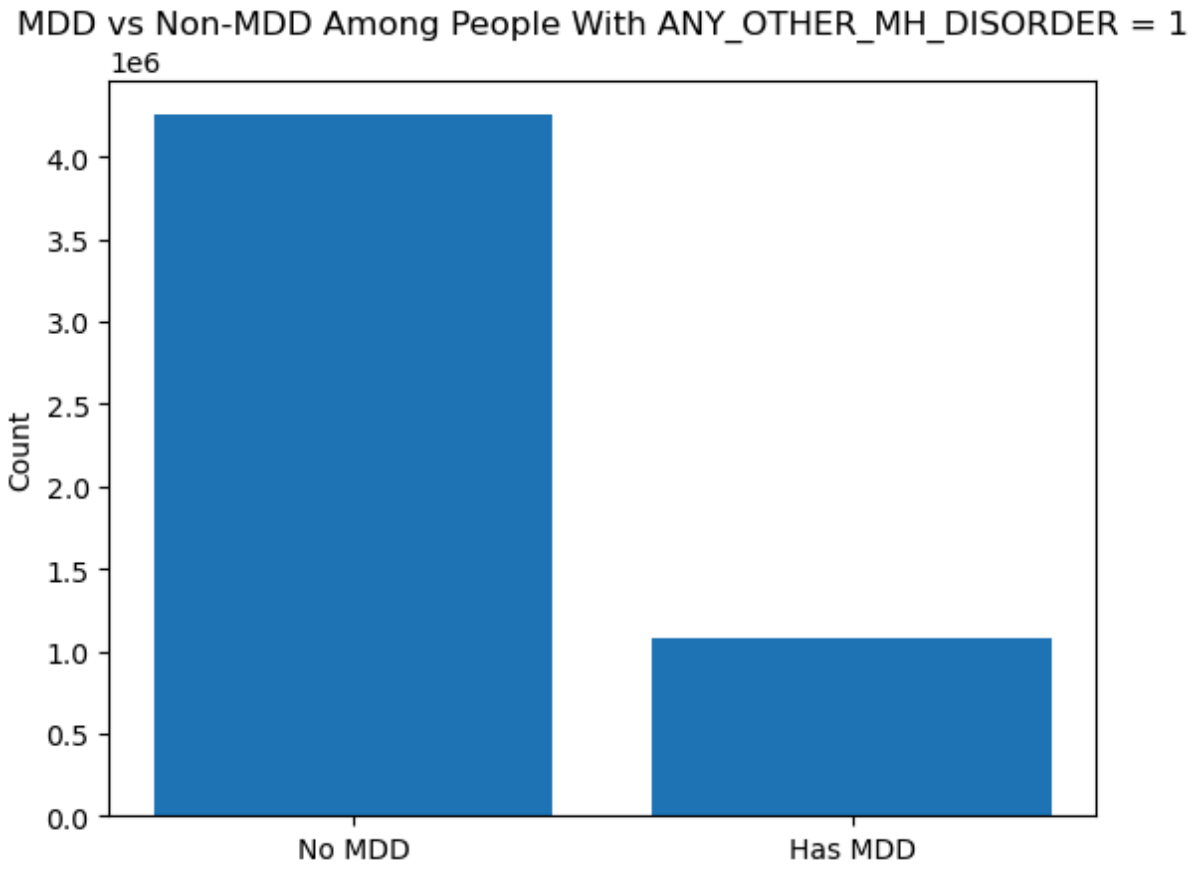
In [92]:
```
# Histogram of IS_MARRIED

mapped = model_df["IS_MARRIED"].map(binary_flag_map)

counts = mapped.value_counts().reindex(binary_flag_map.values())

plt.bar(counts.index, counts.values)
plt.xticks(rotation=45, ha="right")
plt.xlabel("Marital Status")
plt.ylabel("Count")
plt.title("Marital Status Distribution")
plt.tight_layout()
plt.show()
```



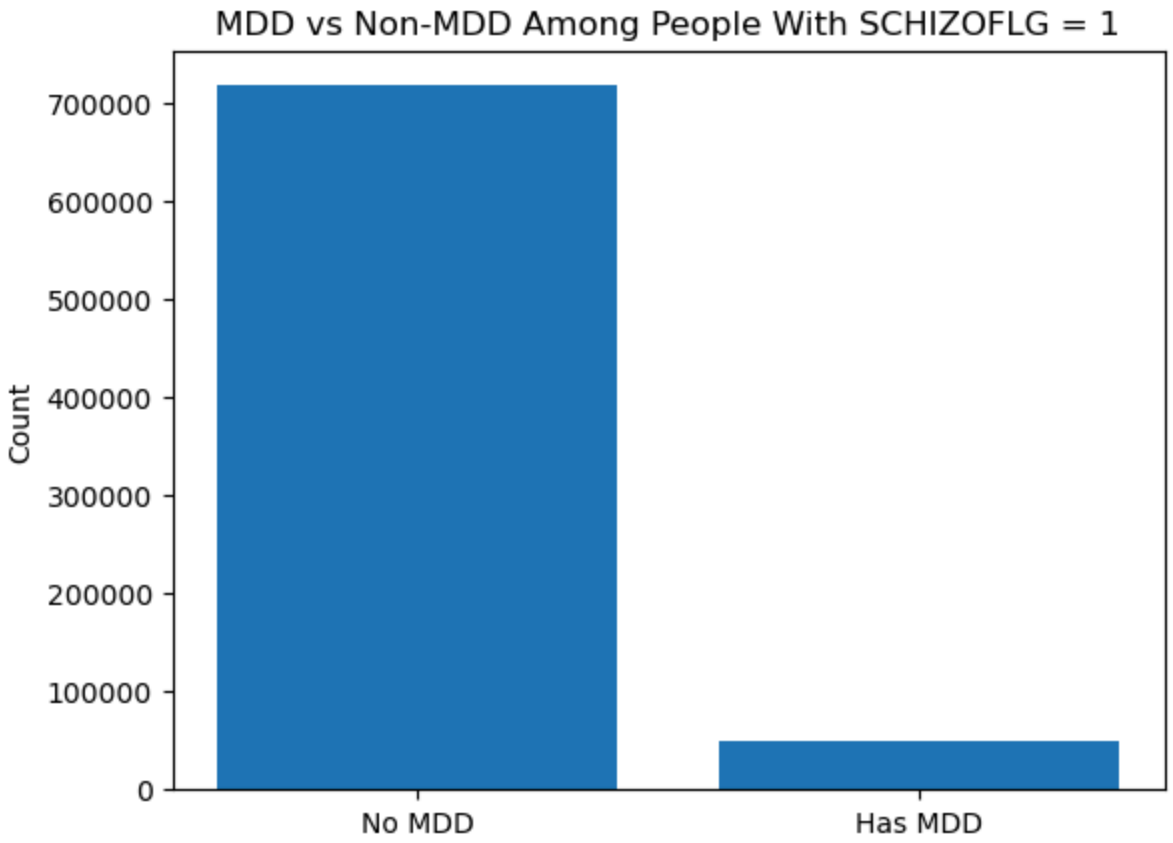
In [93]:
```
# Percentage of people with MDD

subset = model_df[model_df["MDD"] == 1]
percentage_mdd = len(subset)/len(model_df)
print("Percentage of people with MDD: ", percentage_mdd)
```

```
Percentage of people with MDD:  0.2689162224166924
```

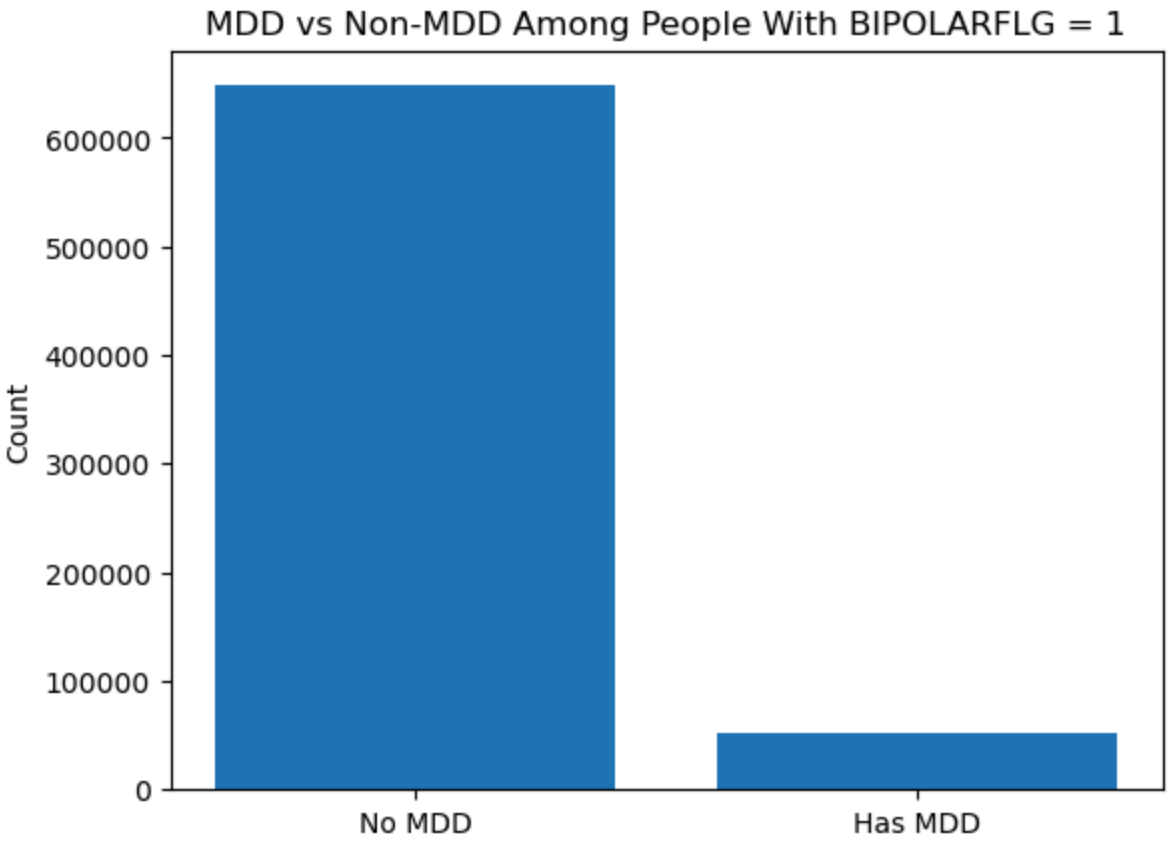
In [94]:
```
# Bar chart of those with any other mental health disorder and MDD

subset = model_df[model_df["ANY_OTHER_MH_DISORDER"] == 1]
counts = subset["MDD"].value_counts().sort_index()
labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People With ANY_OTHER_MH_DISORDER = 1")
plt.show()
```



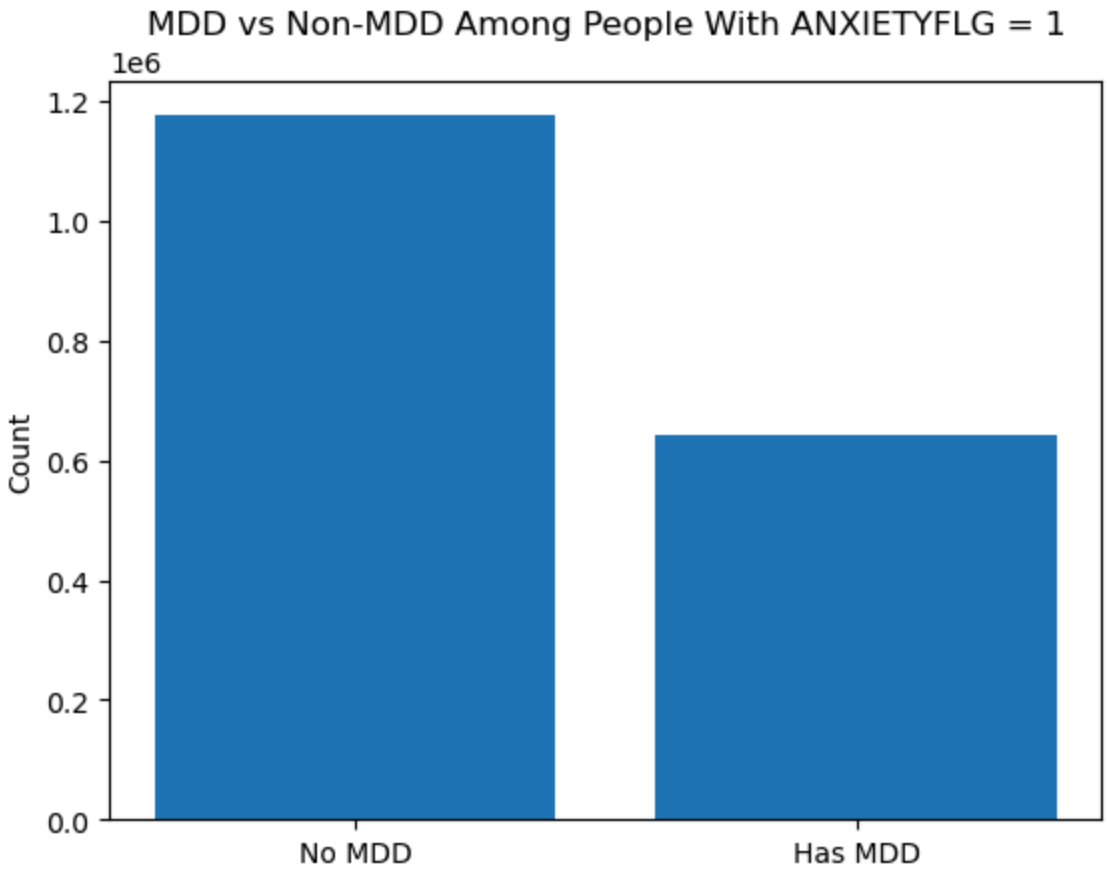
In [95]:
```
# Bar chart of those with SCHIZOFLG and MDD

subset = model_df[model_df["SCHIZOFLG"] == 1]
counts = subset["MDD"].value_counts().sort_index()
labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People With SCHIZOFLG = 1")
plt.show()
```



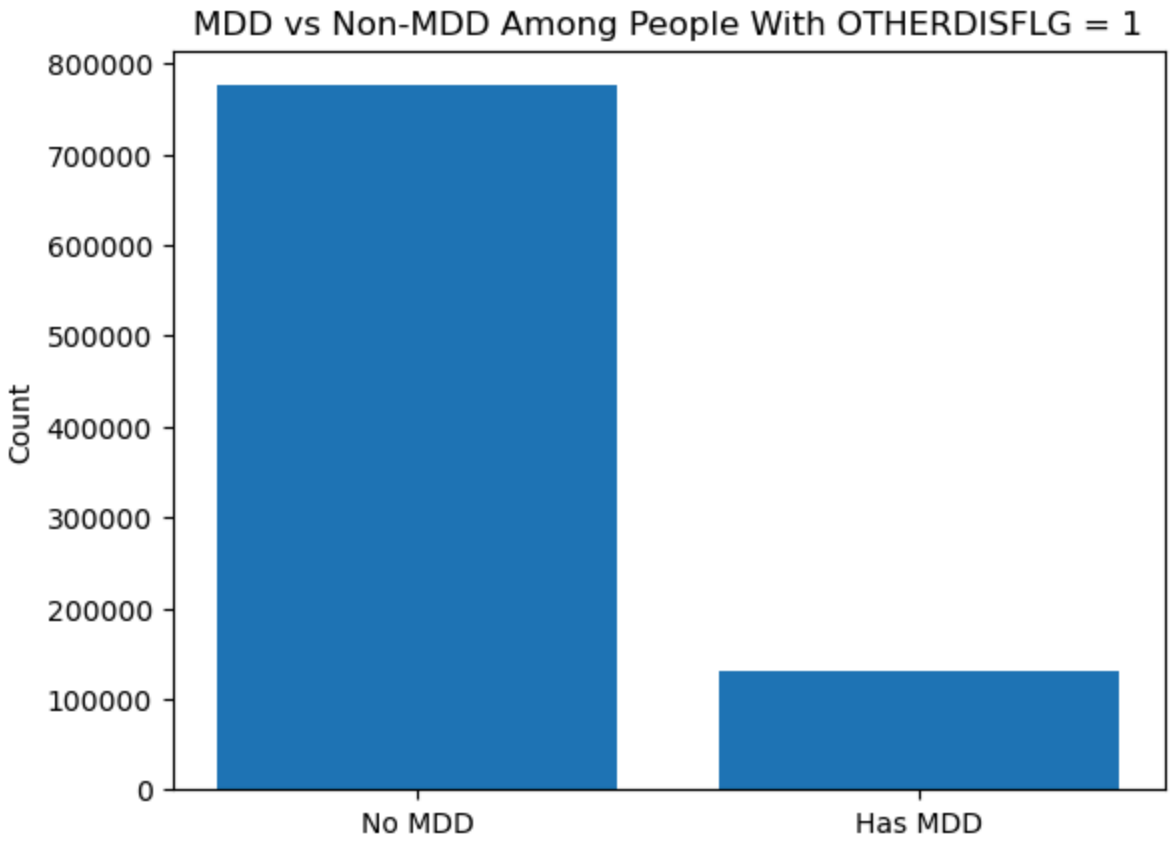
In [96]:
```
# Bar chart of those with BIPOLARFLG and MDD

subset = model_df[model_df["BIPOLARFLG"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People With BIPOLARFLG = 1")
plt.show()
```



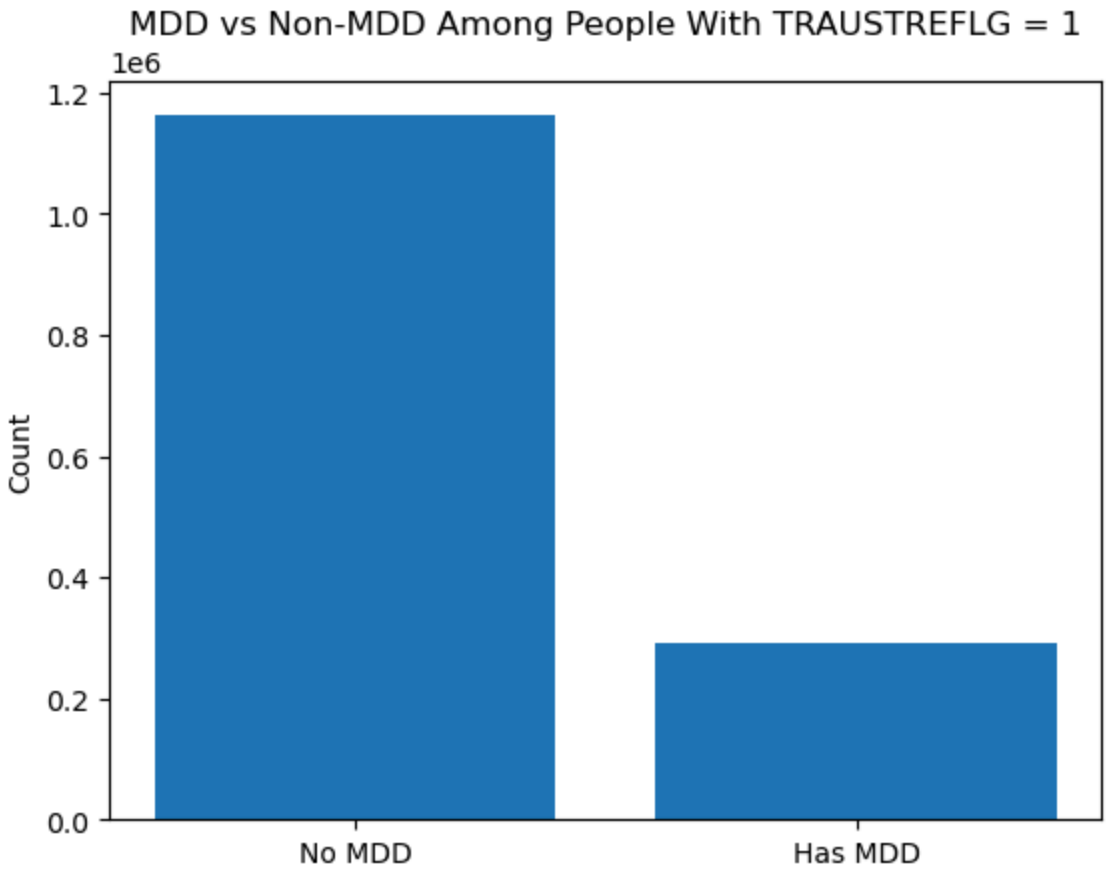
In [97]:
```
# Bar chart of those with ANXIETYFLG and MDD

subset = model_df[model_df["ANXIETYFLG"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People With ANXIETYFLG = 1")
plt.show()
```



In [98]:
```
# Bar chart of those with OTHERDISFLG and MDD

subset = model_df[model_df["OTHERDISFLG"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People With OTHERDISFLG = 1")
plt.show()
```

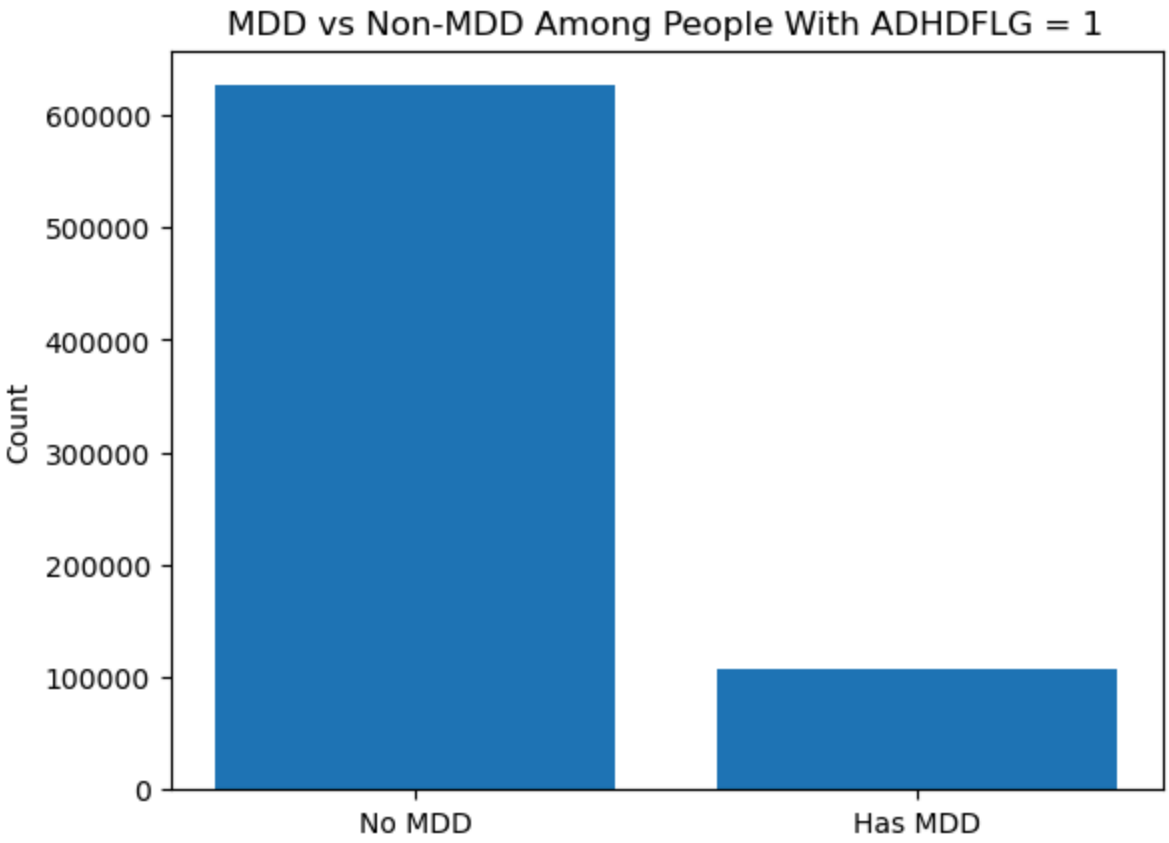


In [99]:
```
# Bar chart of those with TRATUSTREFLG and MDD

subset = model_df[model_df["TRAUSTREFLG"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People With TRAUSTREFLG = 1")
plt.show()
```



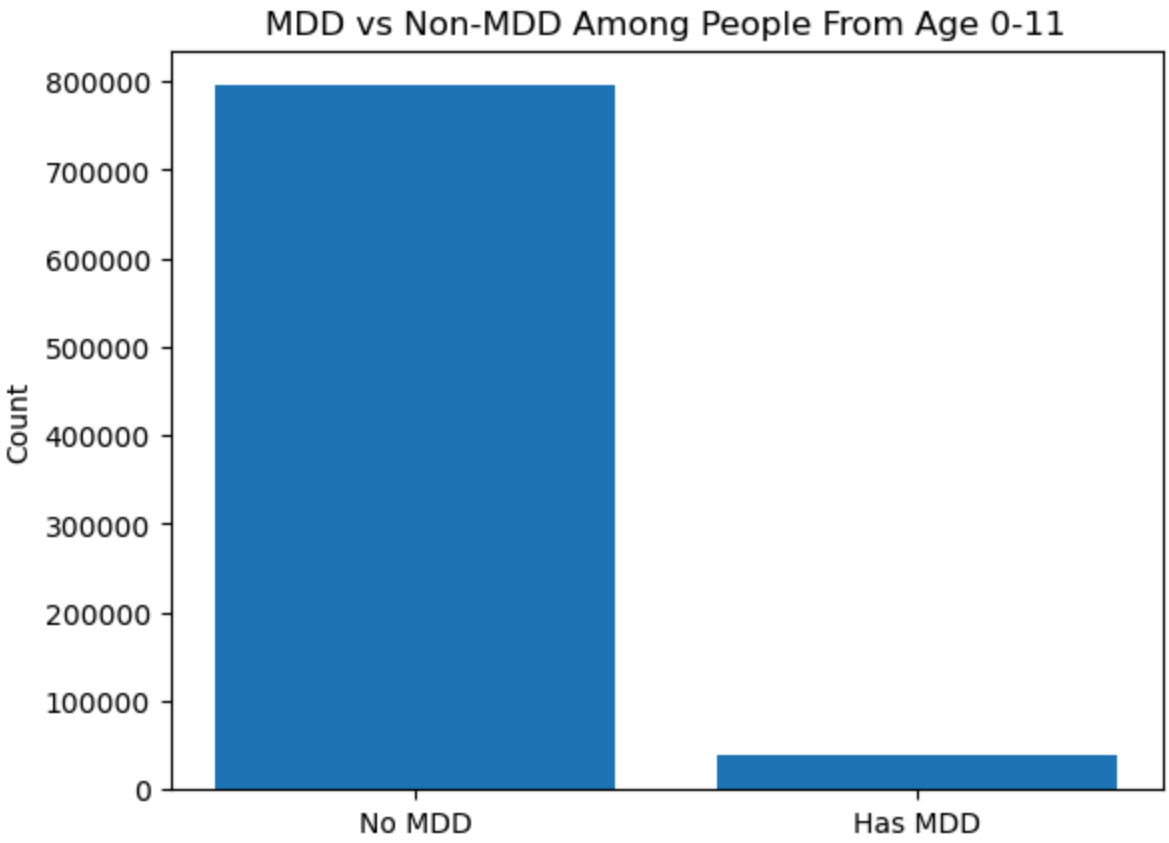
In [100…

```
# Bar chart of those with ADHDFLG and MDD

subset = model_df[model_df["ADHDFLG"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People With ADHDFLG = 1")
plt.show()
```



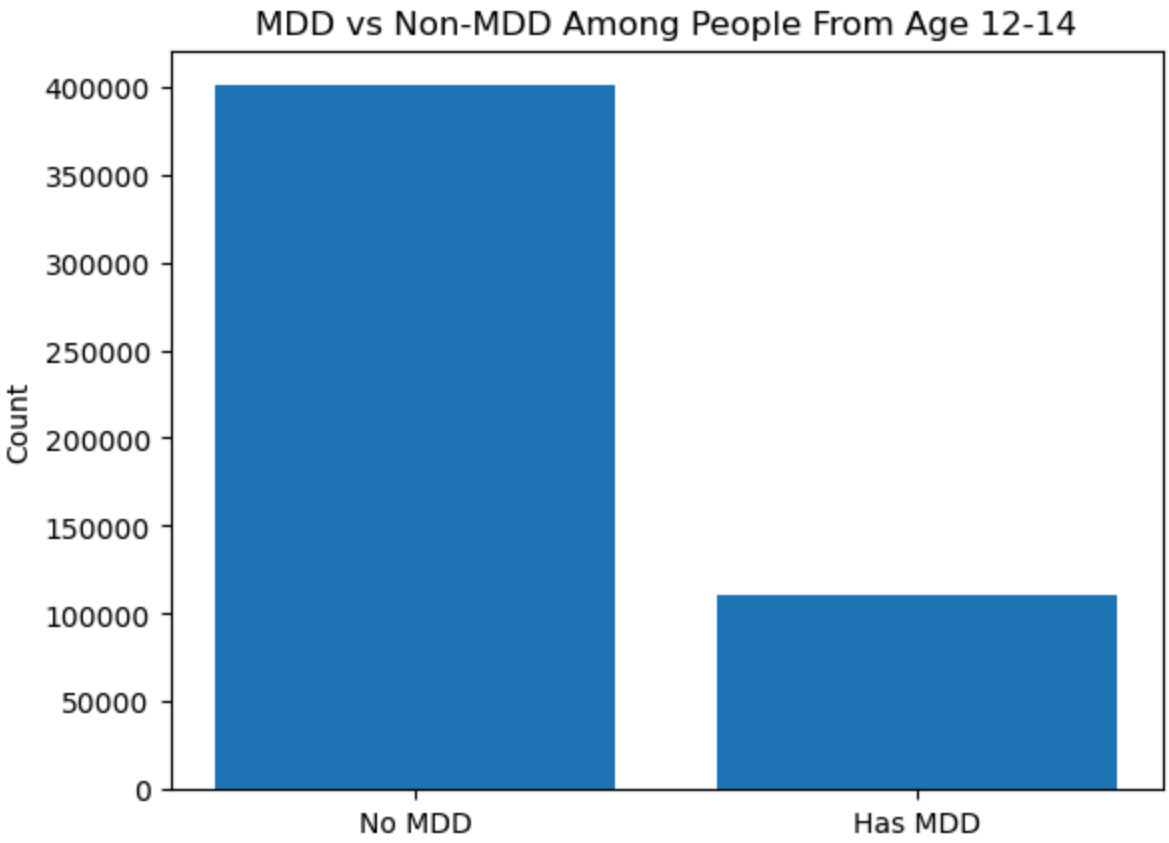
In [101…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 0-11")
plt.show()
```



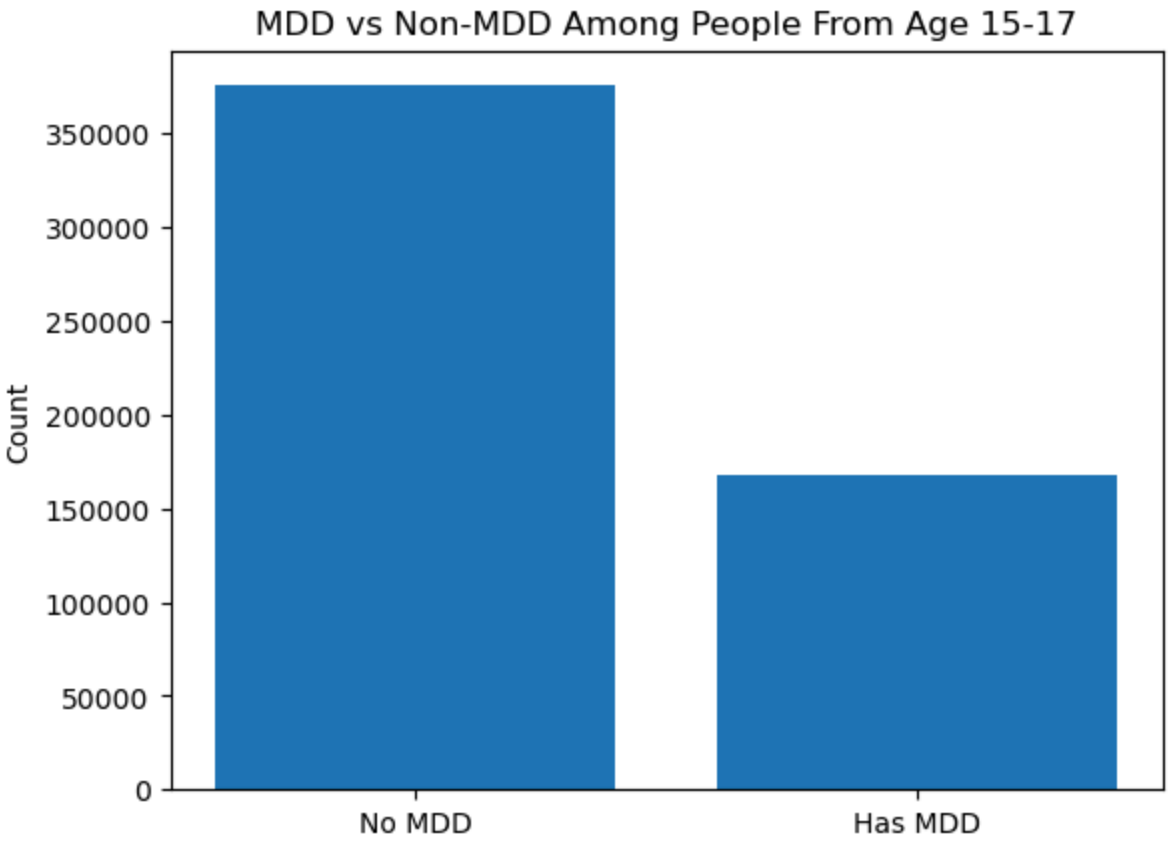
In [102…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 2]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 12-14")
plt.show()
```



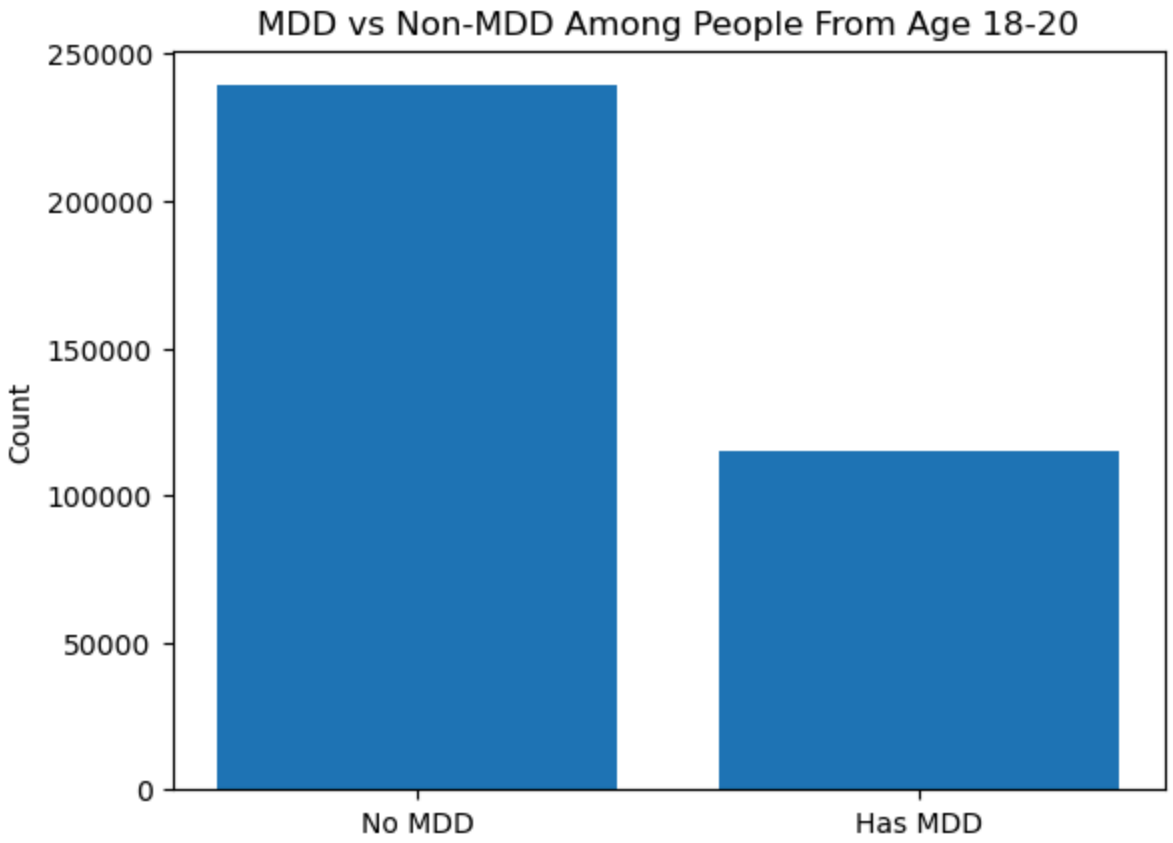
In [103…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 3]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 15-17")
plt.show()
```



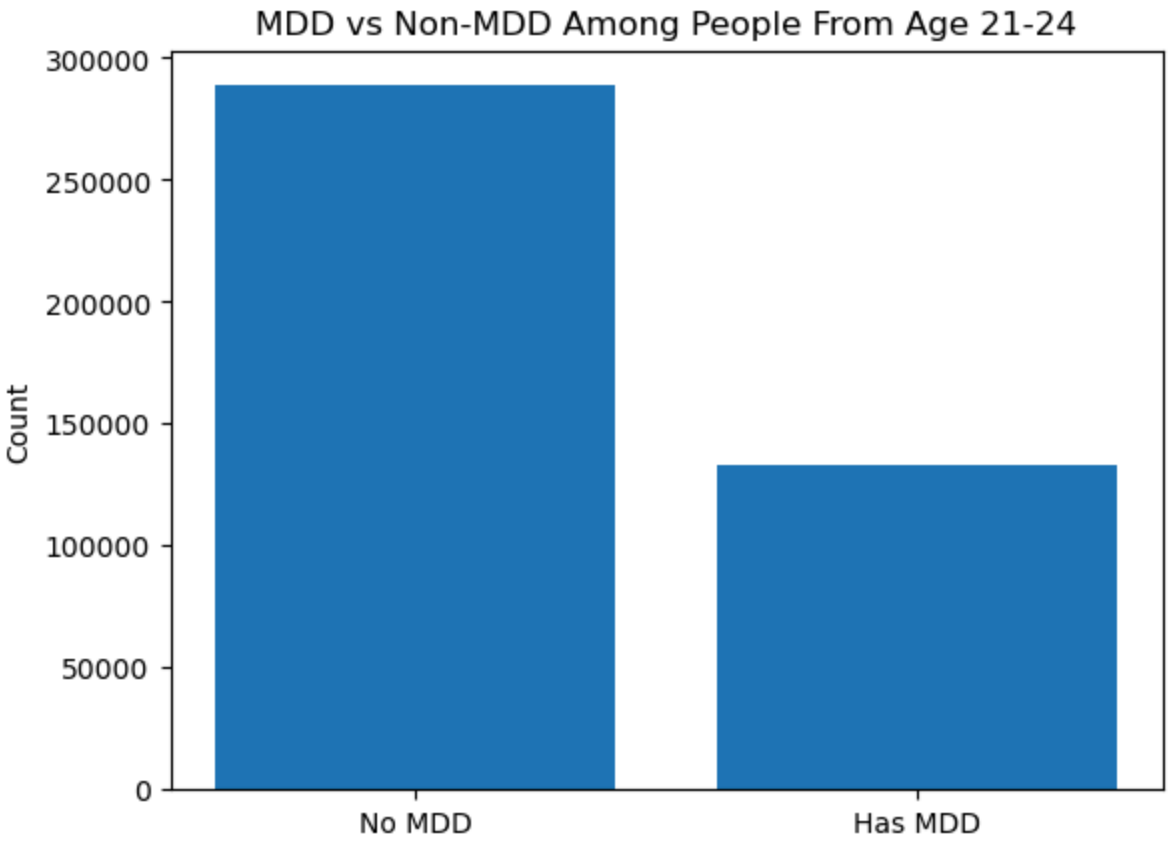
In [104…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 4]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 18-20")
plt.show()
```



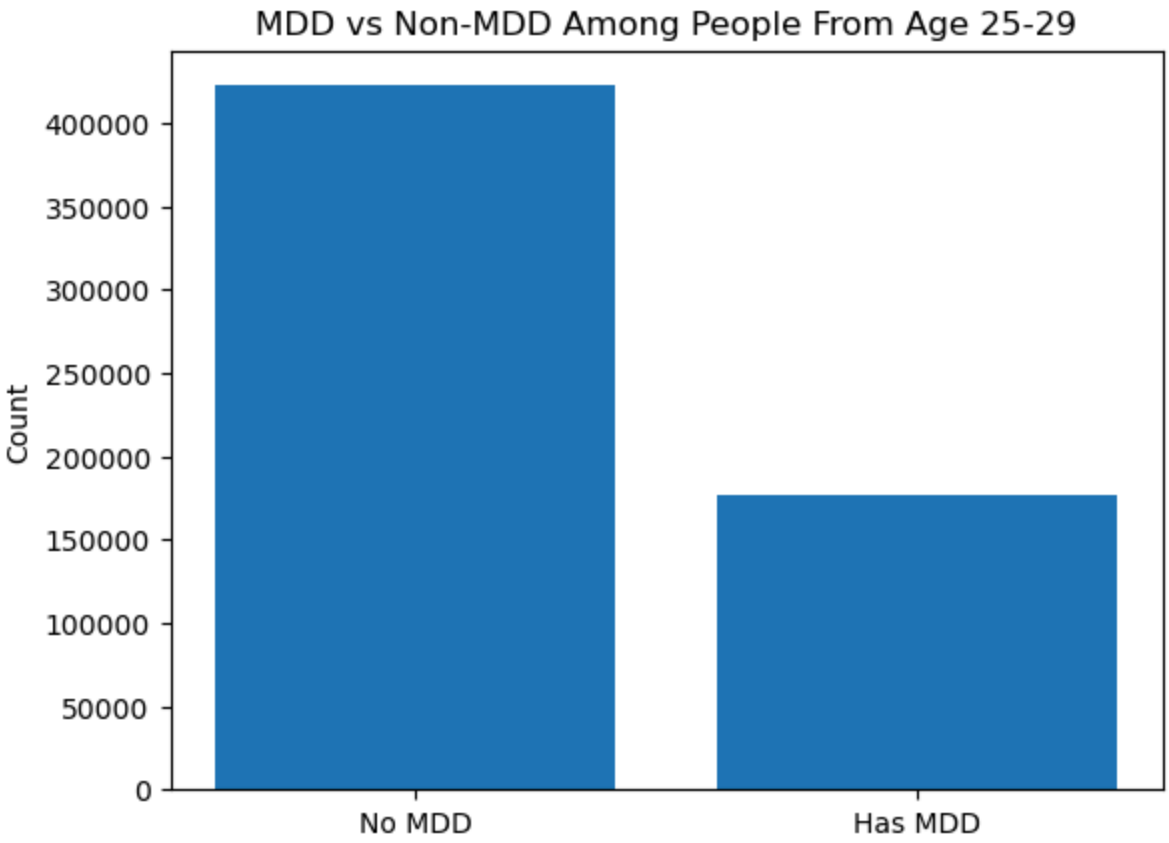
In [105…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 5]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 21-24")
plt.show()
```



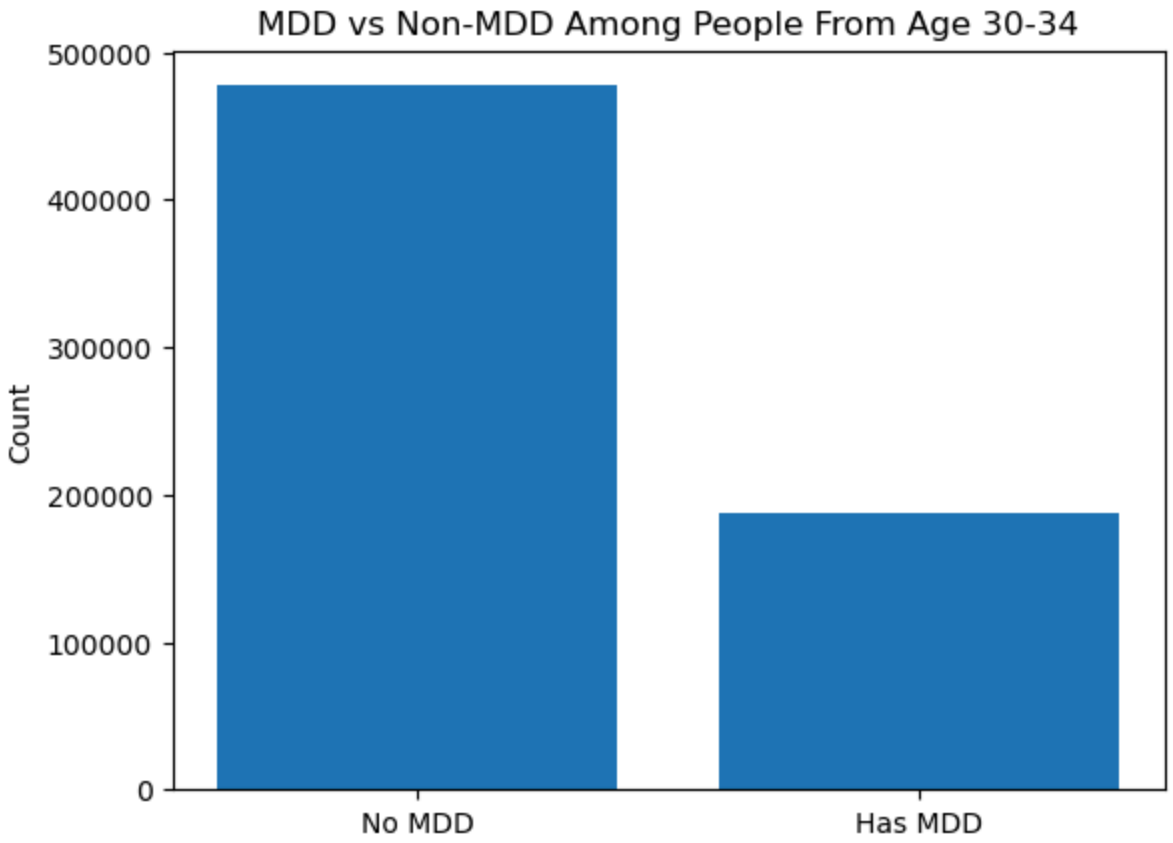
In [106…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 6]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 25-29")
plt.show()
```



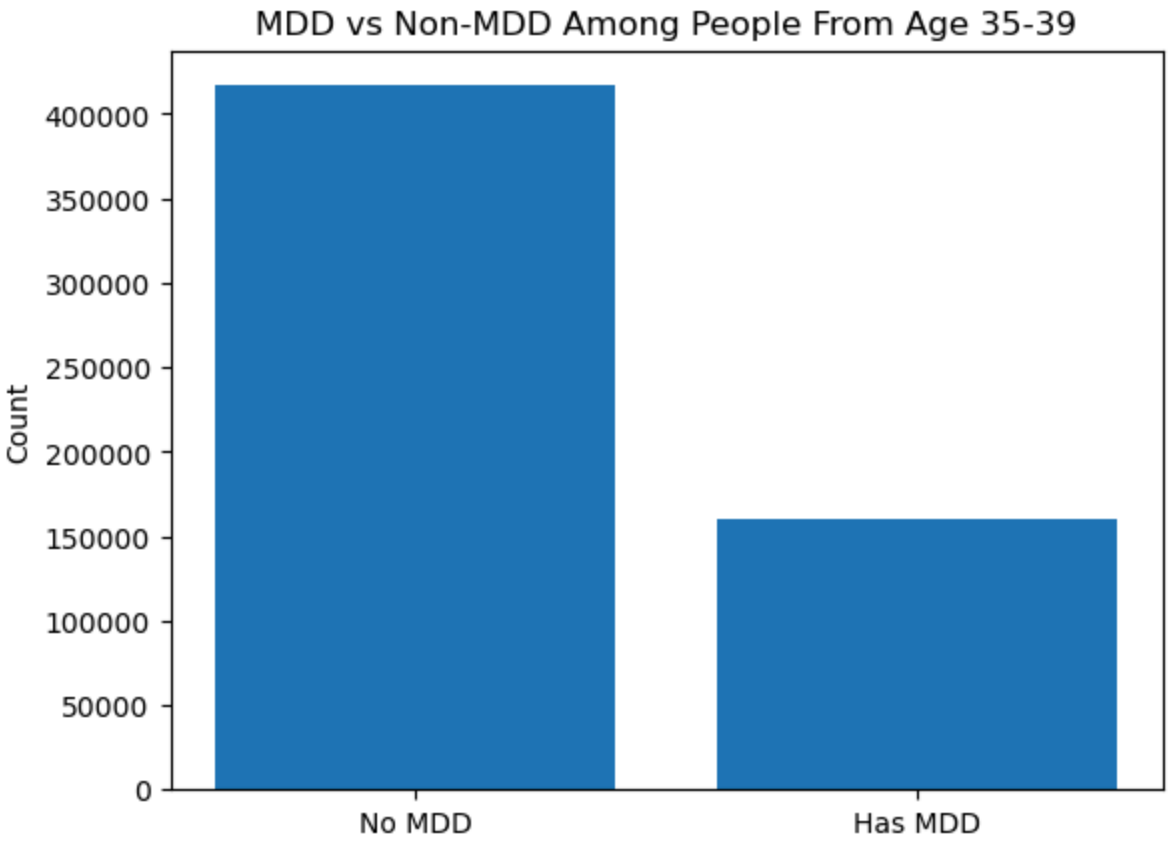
In [107…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 7]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 30-34")
plt.show()
```



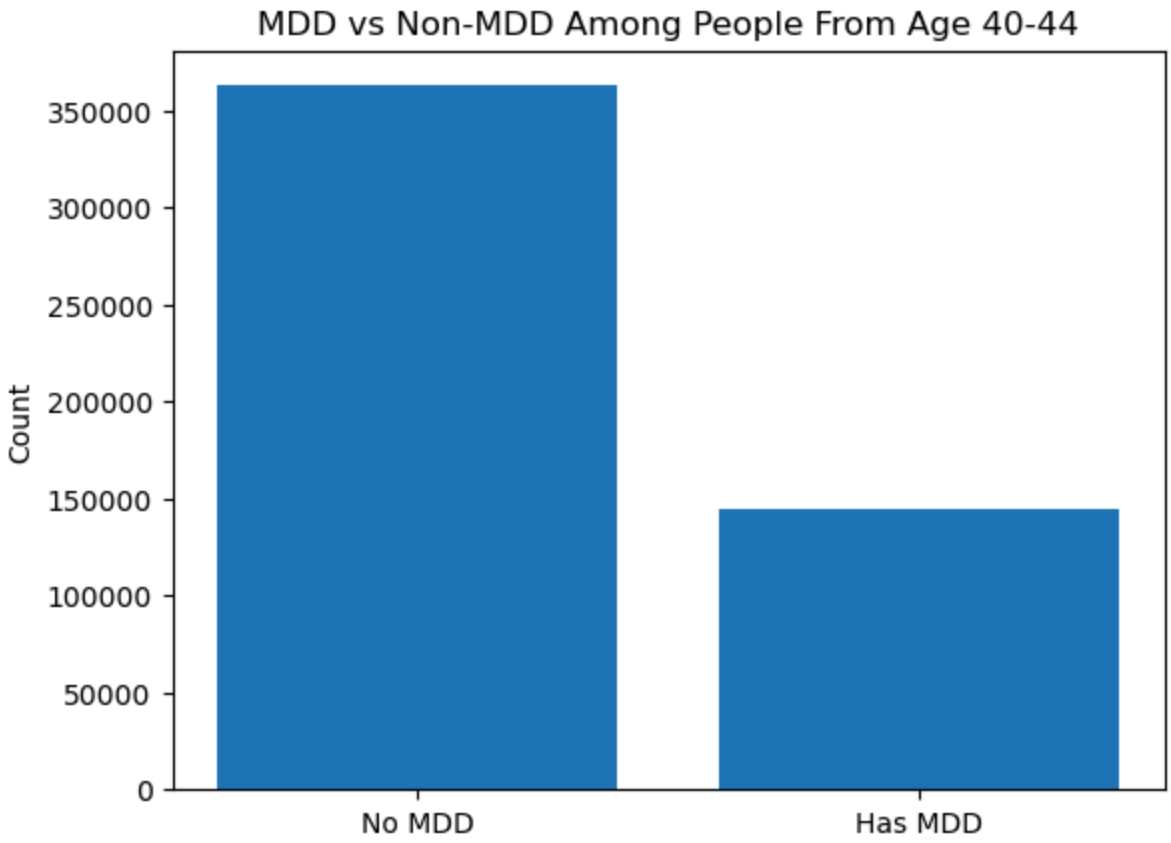
In [108…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 8]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 35-39")
plt.show()
```



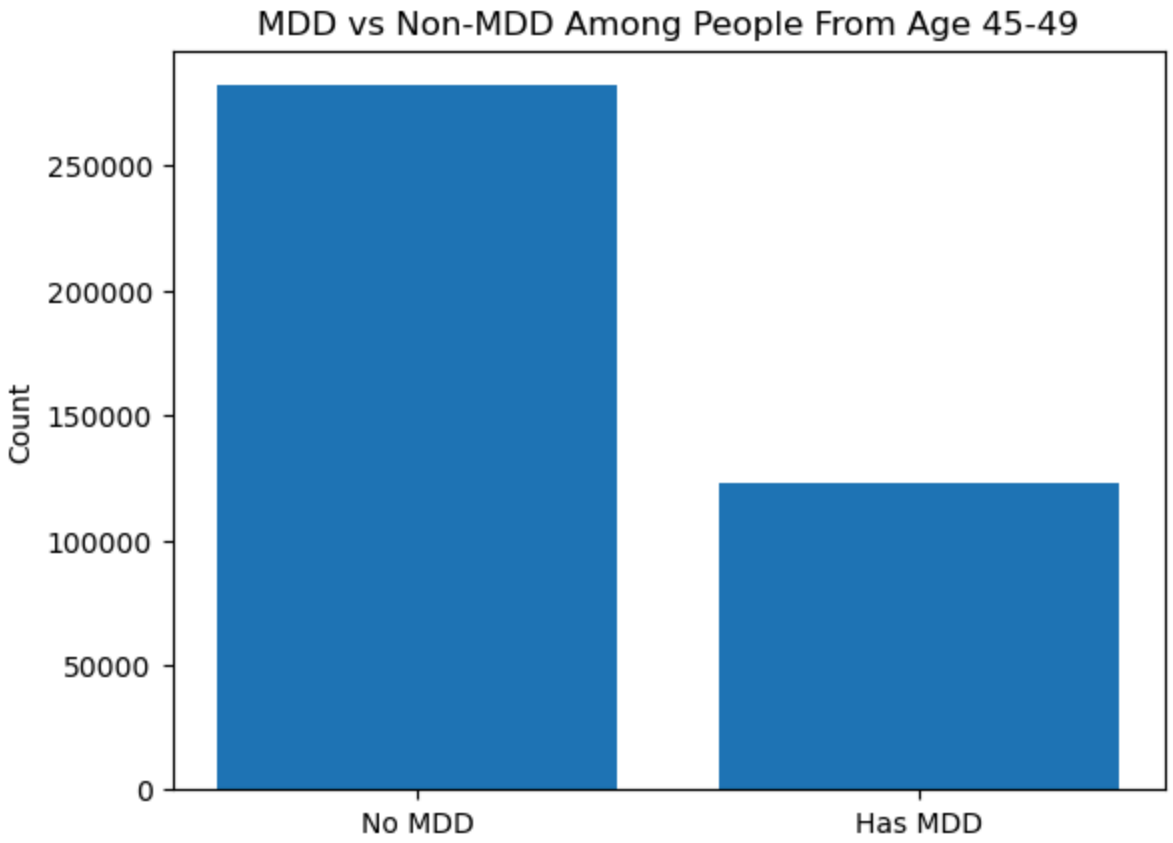
In [109…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 9]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 40-44")
plt.show()
```



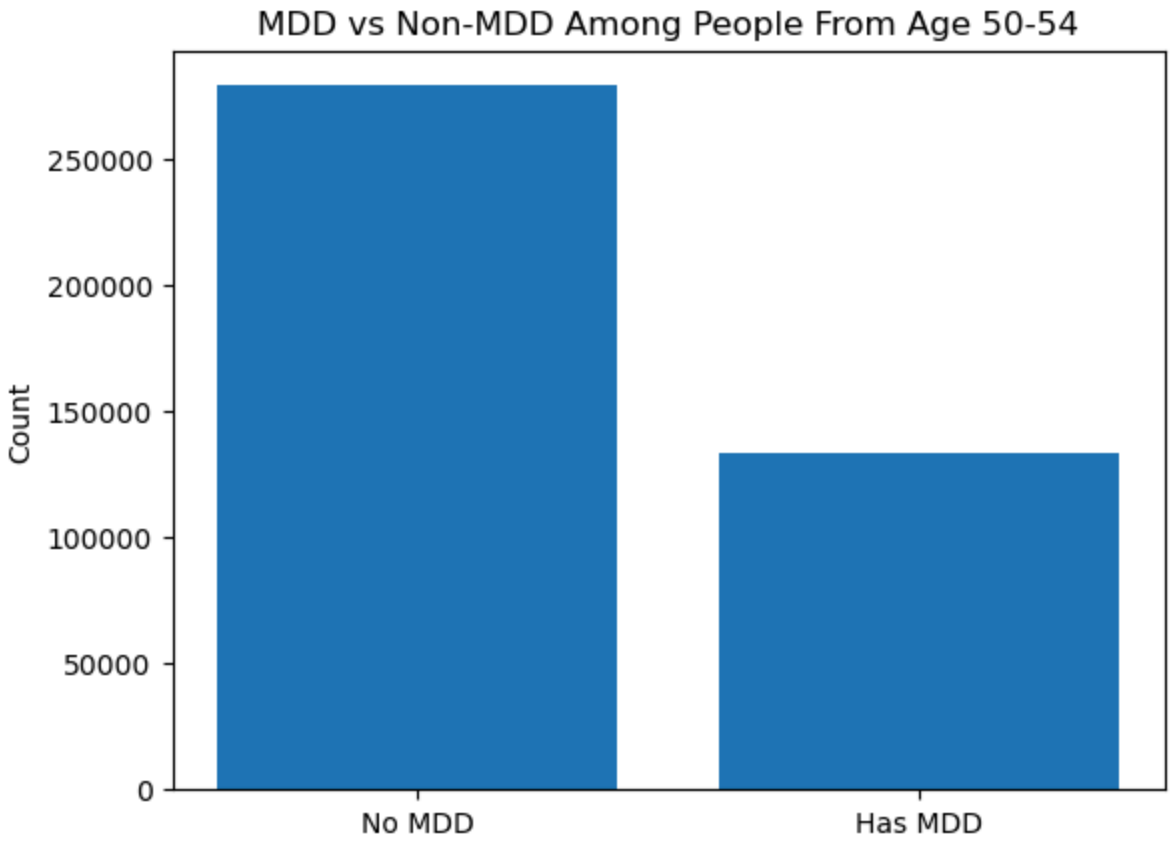
In [110…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 10]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 45-49")
plt.show()
```



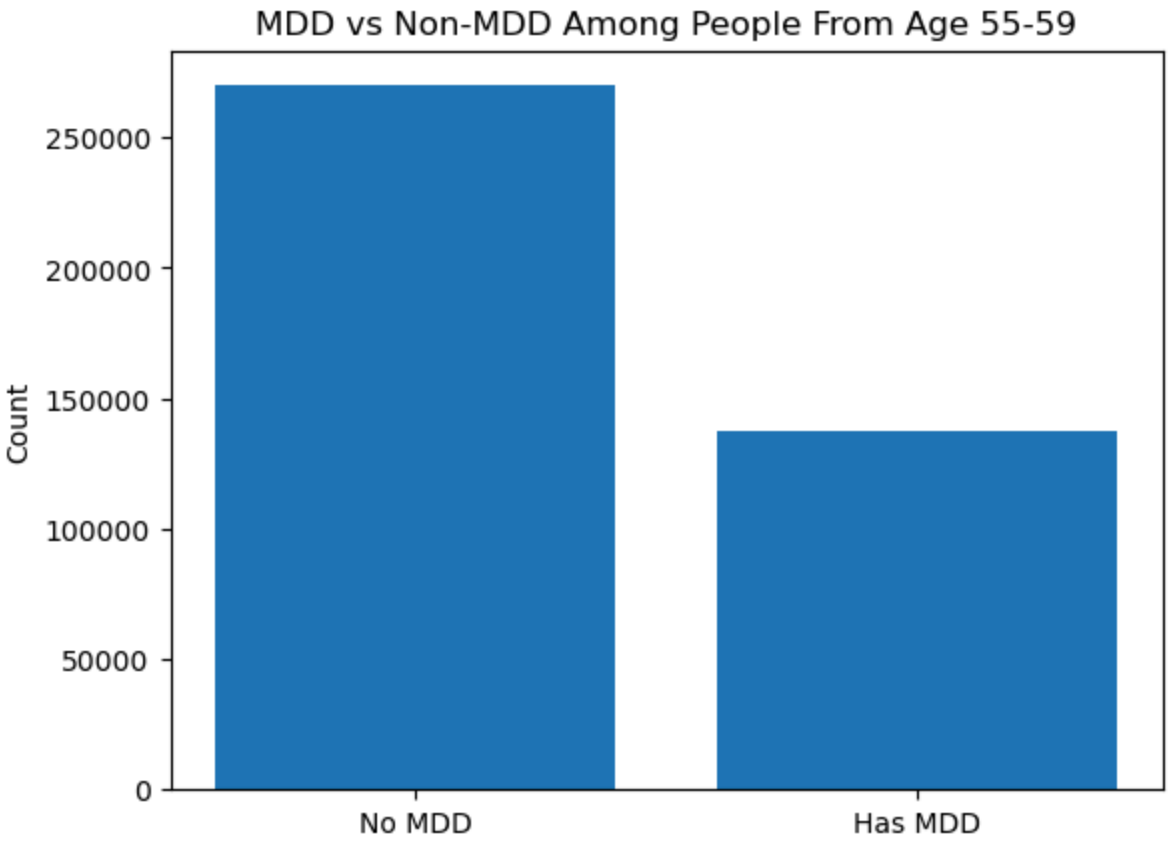
In [111…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 11]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 50-54")
plt.show()
```



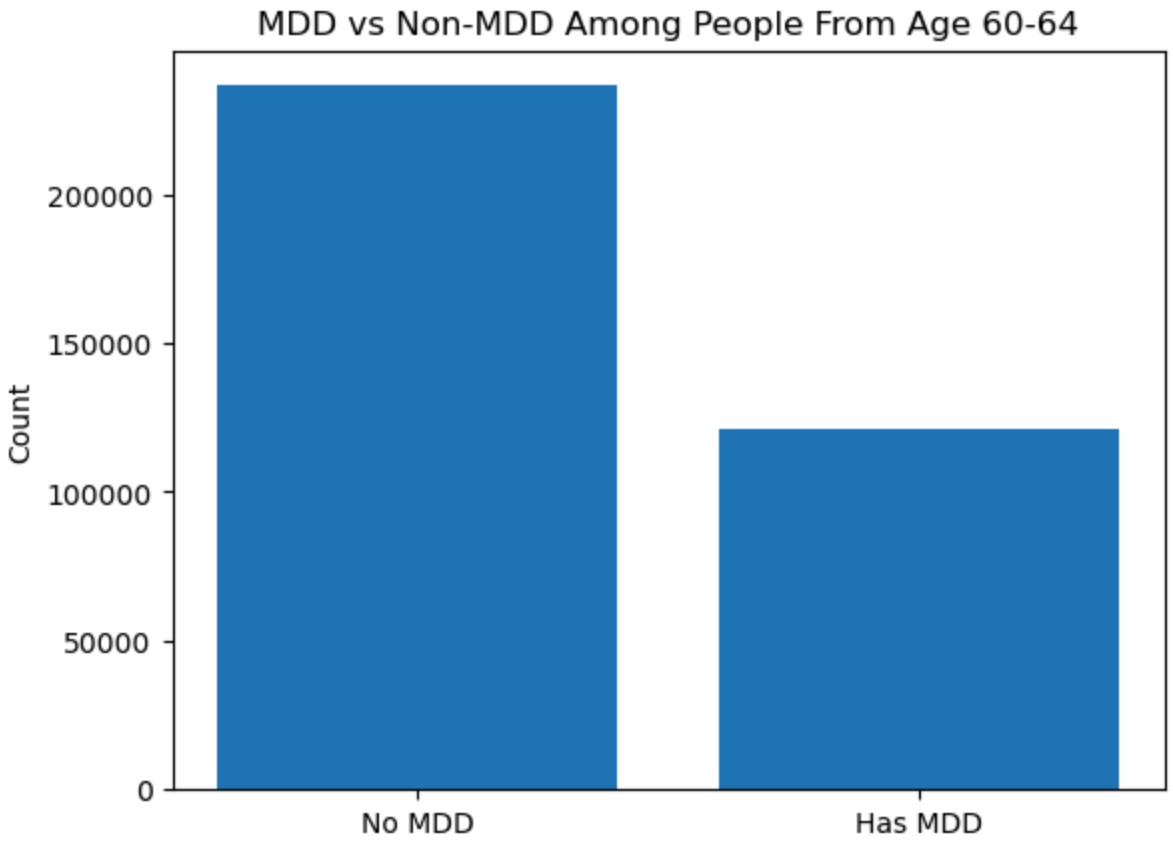
In [112…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 12]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 55-59")
plt.show()
```



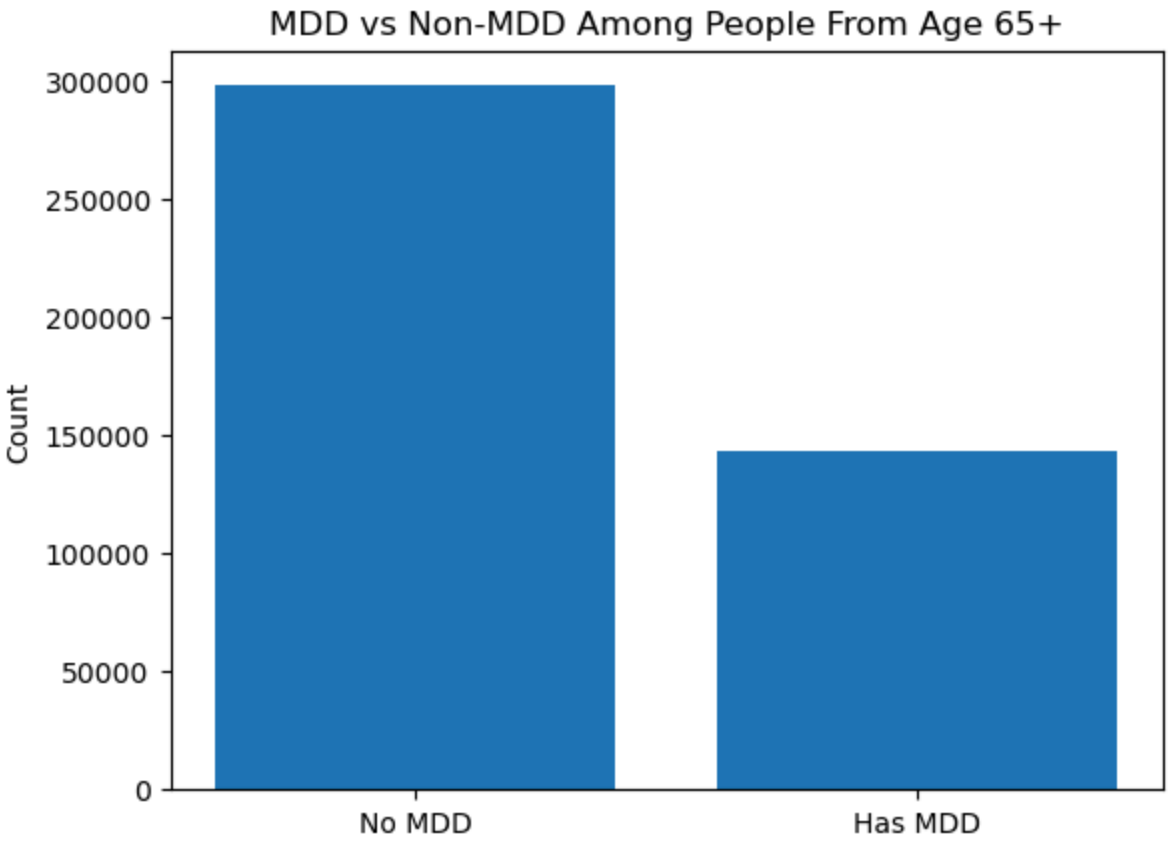
In [113…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 13]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 60-64")
plt.show()
```



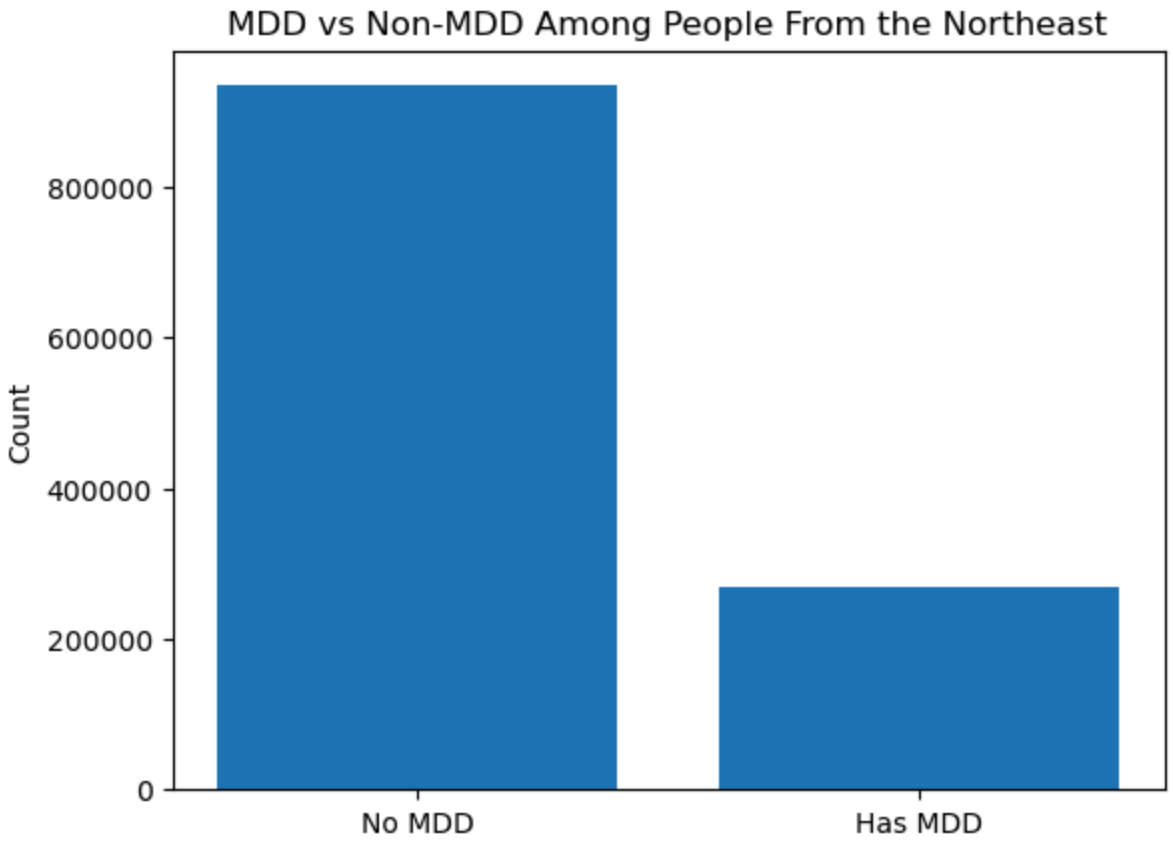
In [114…

```
# Bar chart of AGE and MDD

subset = model_df[model_df["AGE"] == 14]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From Age 65+")
plt.show()
```



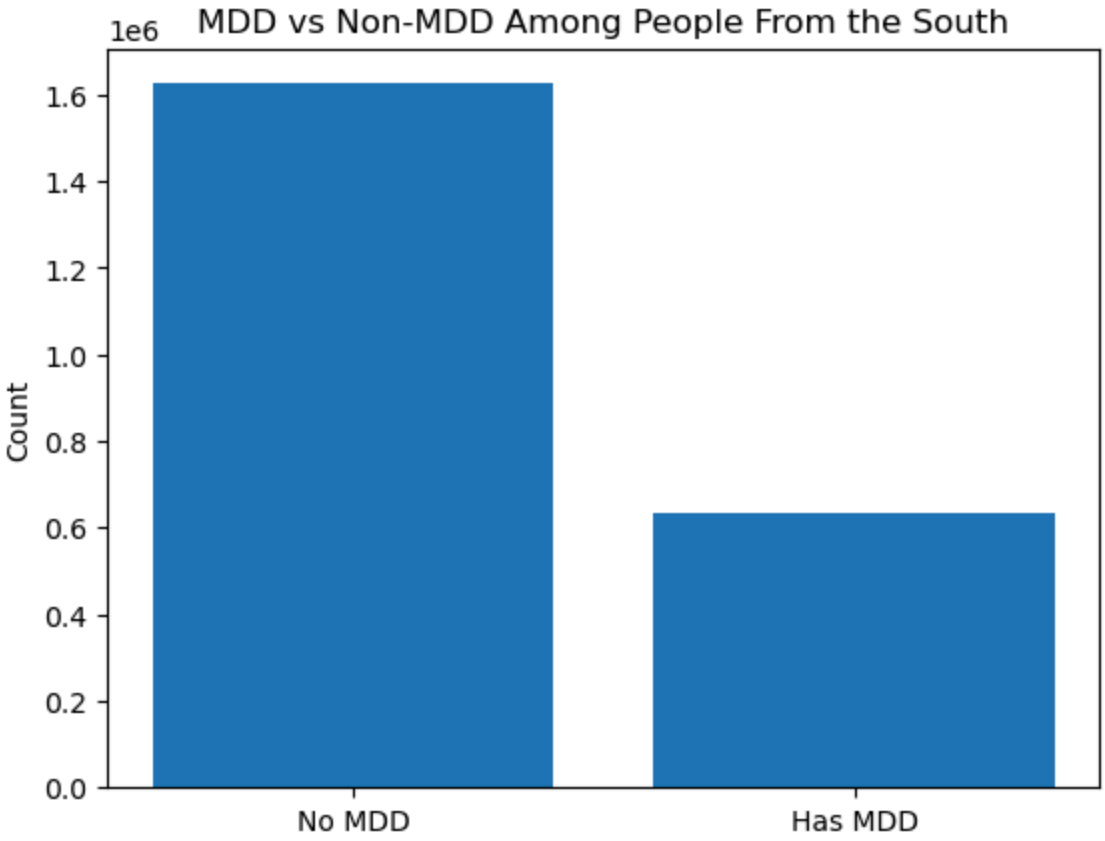
In [115…

```
# Bar chart of REGION and MDD

subset = model_df[model_df["REGION"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From the Northeast")
plt.show()
```



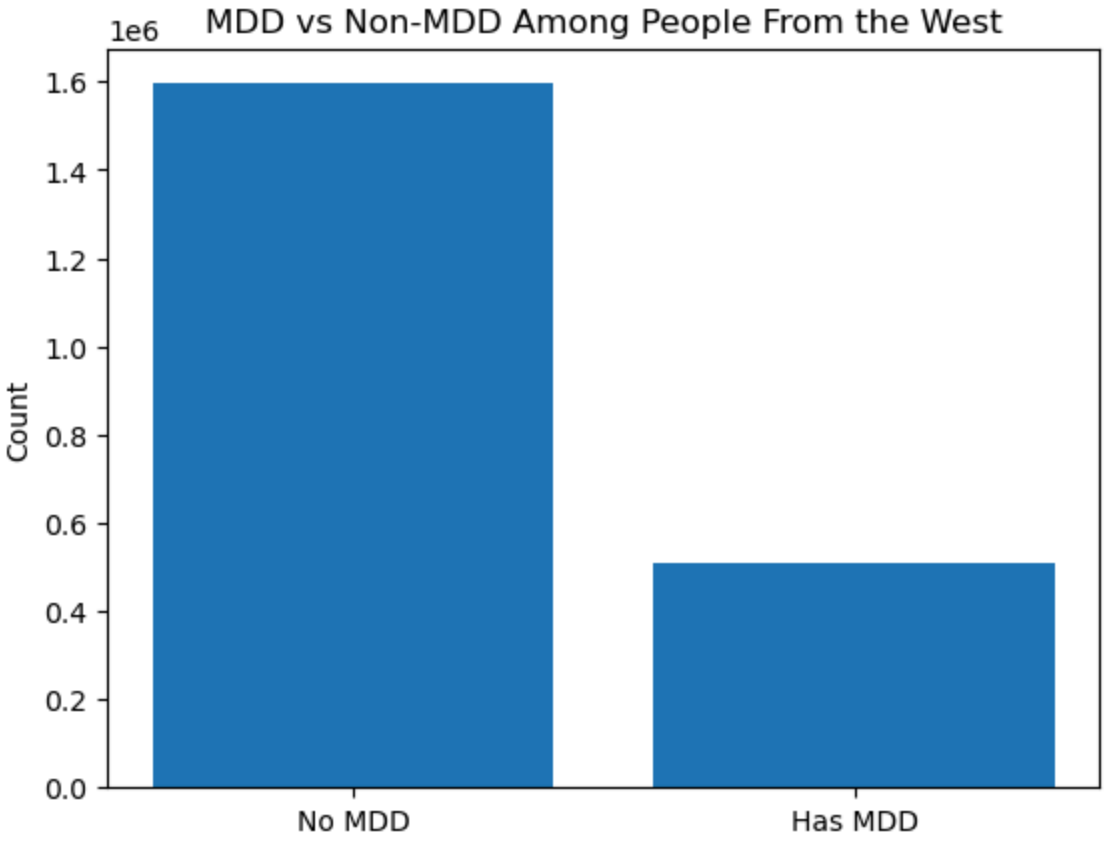
In [116…

```
# Bar chart of REGION and MDD

subset = model_df[model_df["REGION"] == 3]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From the South")
plt.show()
```



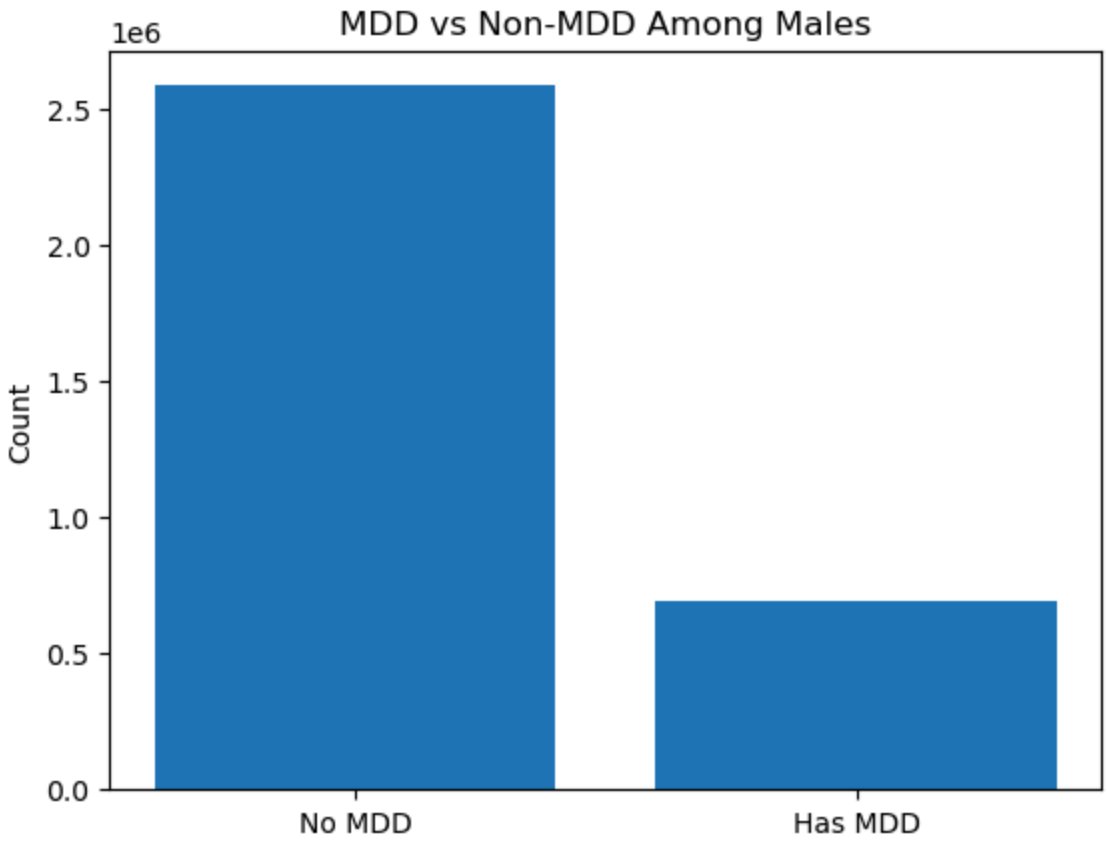
In [117…

```
# Bar chart of REGION and MDD

subset = model_df[model_df["REGION"] == 4]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among People From the West")
plt.show()
```



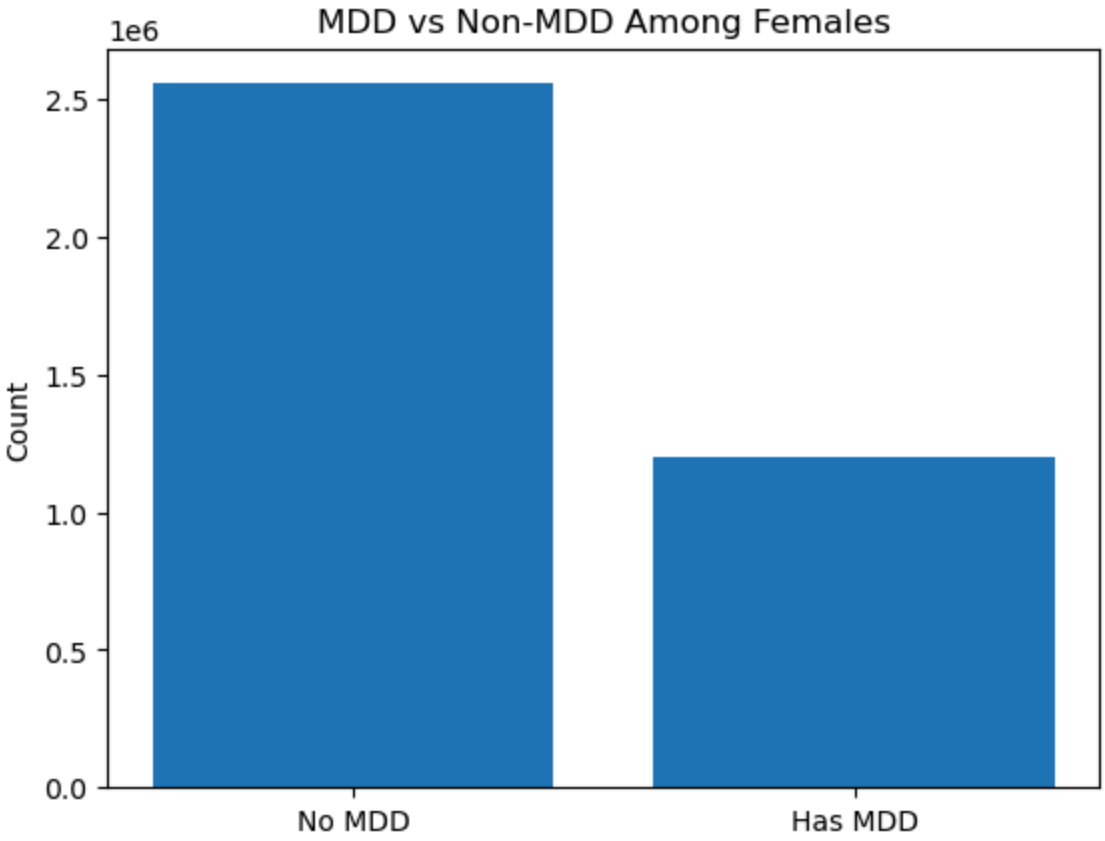
In [118…

```
# Bar chart of SEX and MDD

subset = model_df[model_df["SEX"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among Males")
plt.show()
```



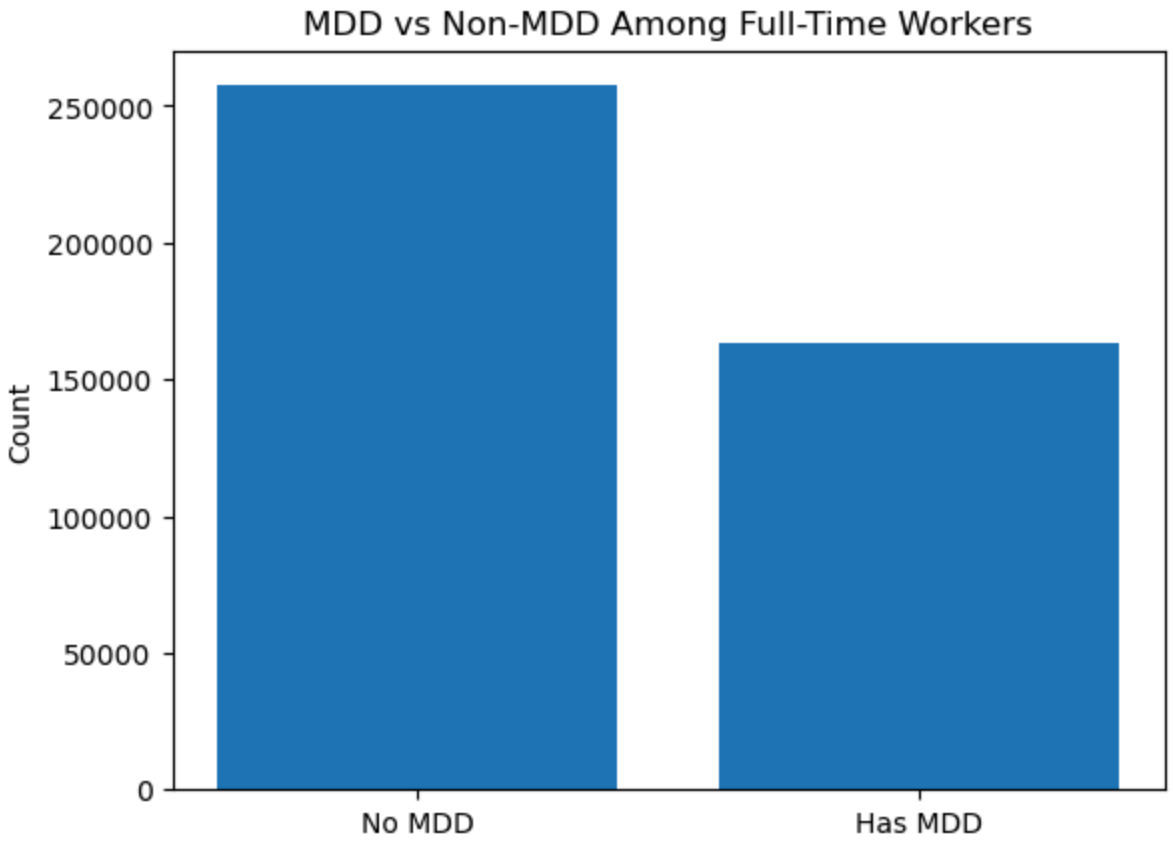
In [119…

```
# Bar chart of SEX and MDD

subset = model_df[model_df["SEX"] == 2]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among Females")
plt.show()
```



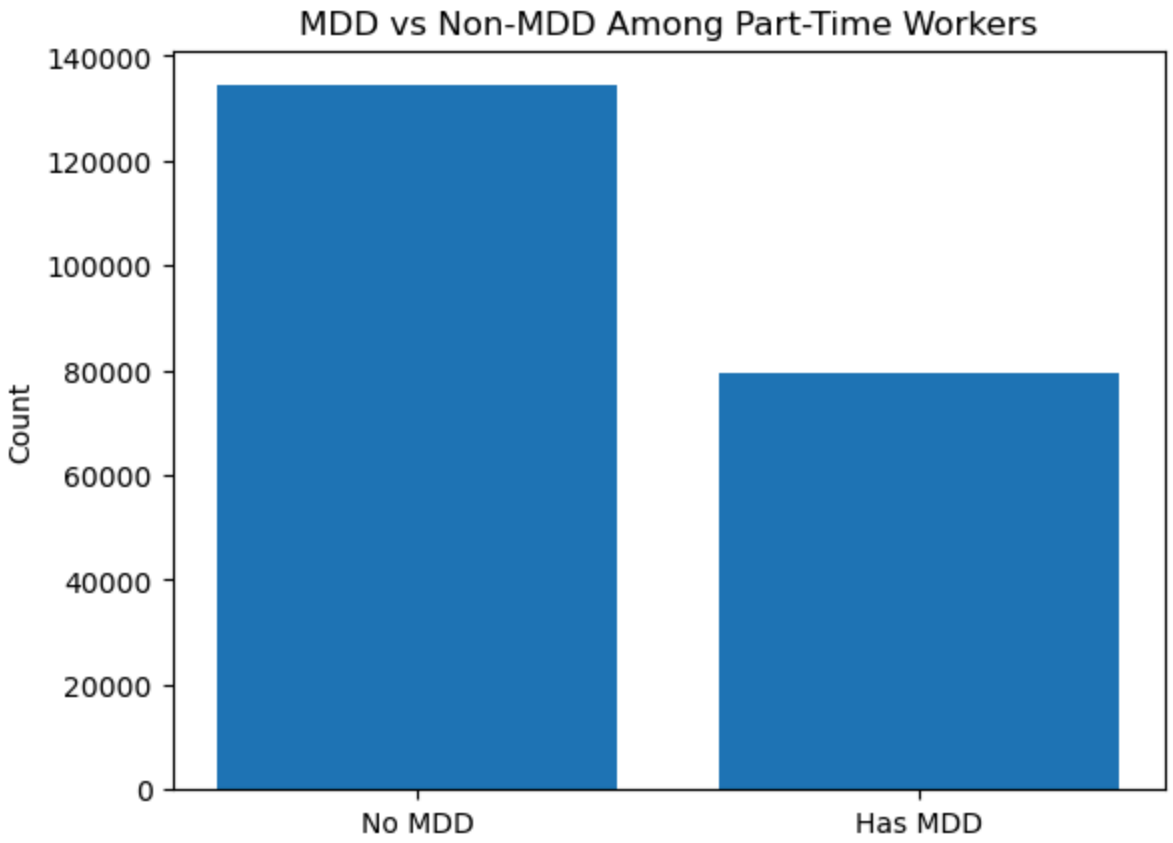
In [120…

```
# Bar chart of EMPLOY and MDD

subset = model_df[model_df["EMPLOY"] == 1]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among Full-Time Workers")
plt.show()
```



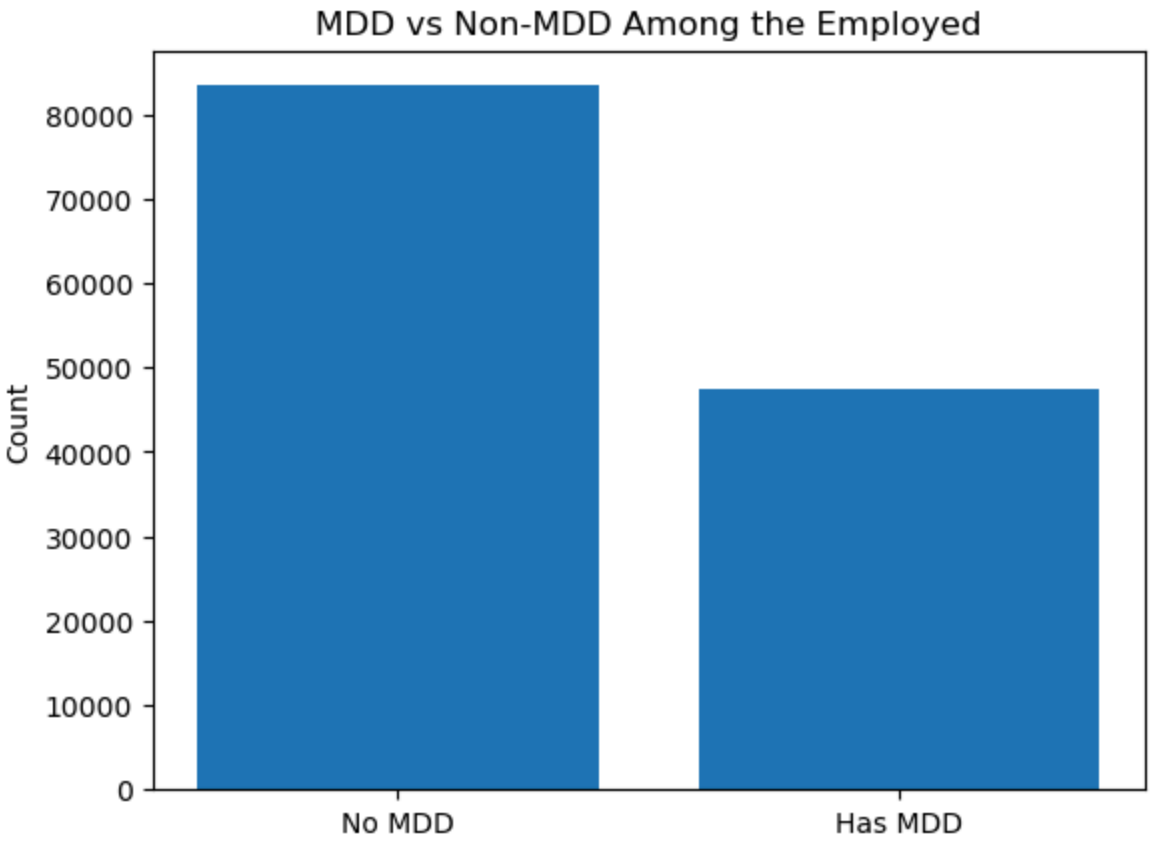
In [121…

```
# Bar chart of EMPLOY and MDD

subset = model_df[model_df["EMPLOY"] == 2]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among Part-Time Workers")
plt.show()
```



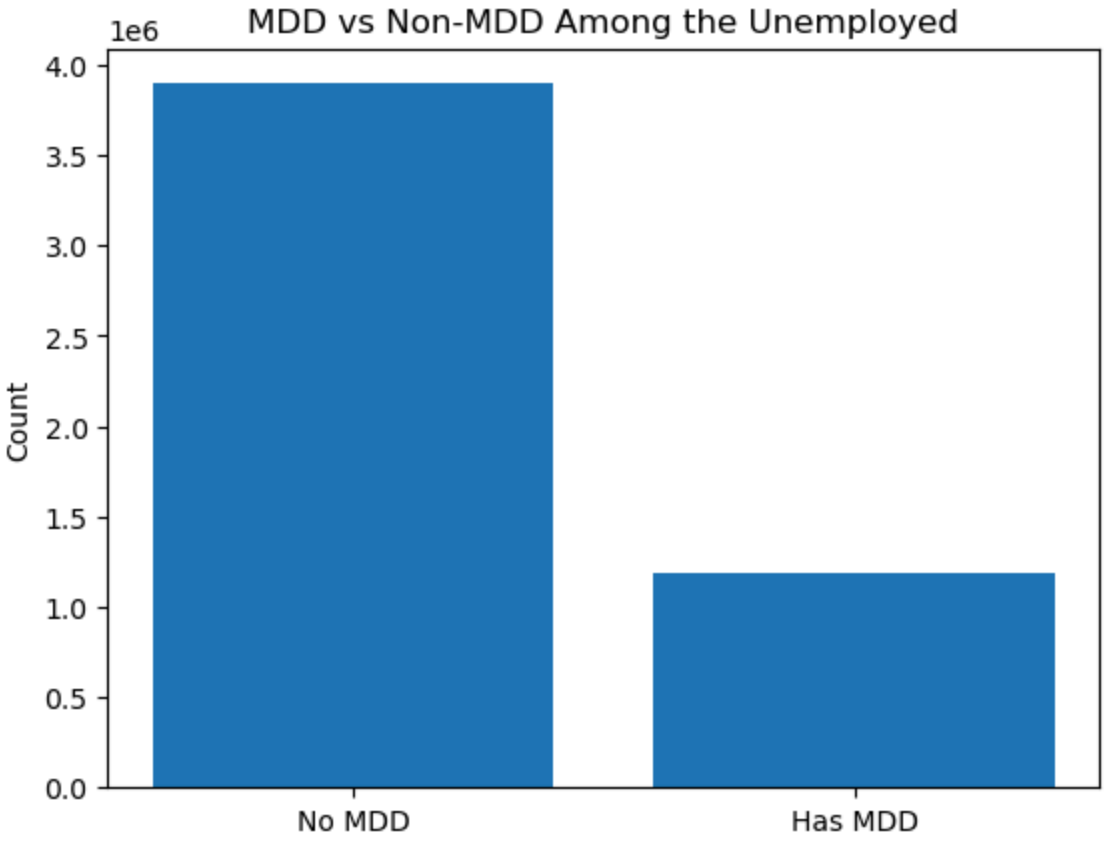
In [122…

```
# Bar chart of EMPLOY and MDD

subset = model_df[model_df["EMPLOY"] == 3]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among the Employed")
plt.show()
```



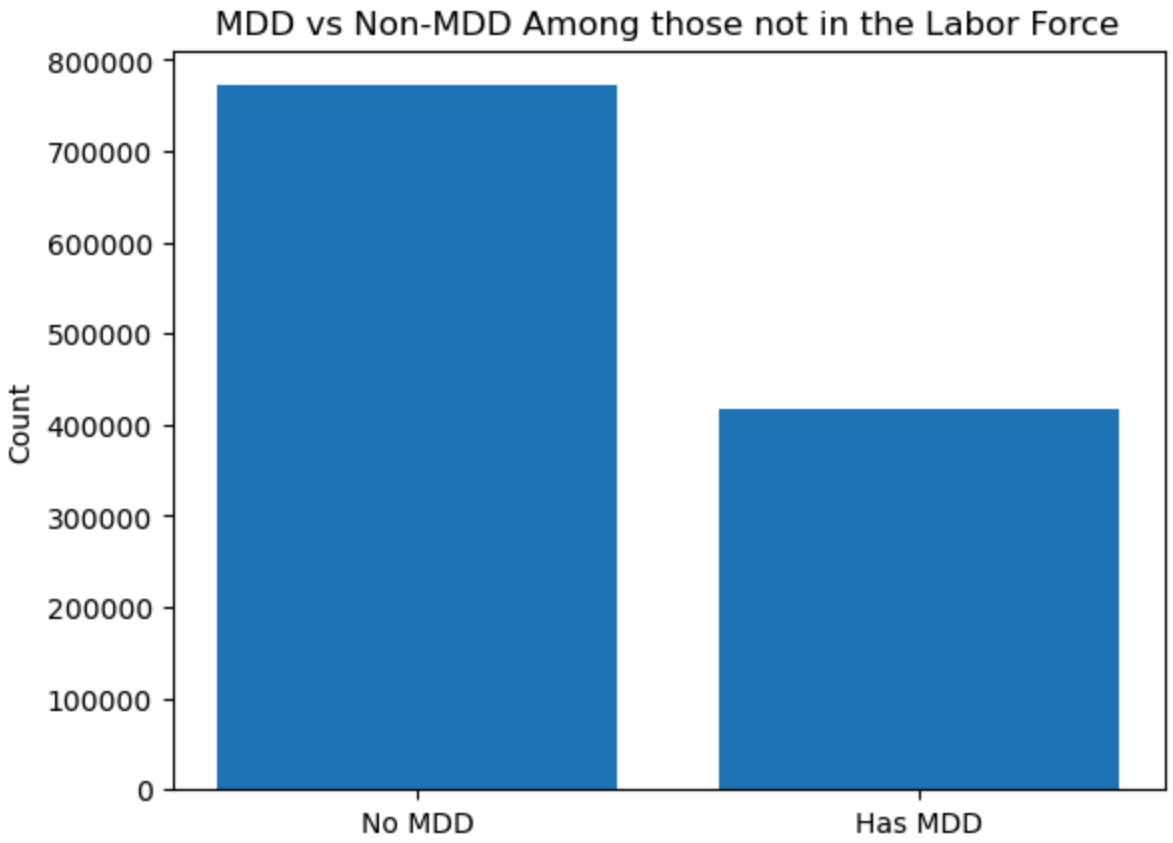
In [123…

```
# Bar chart of EMPLOY and MDD

subset = model_df[model_df["EMPLOY"] == 4]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among the Unemployed")
plt.show()
```



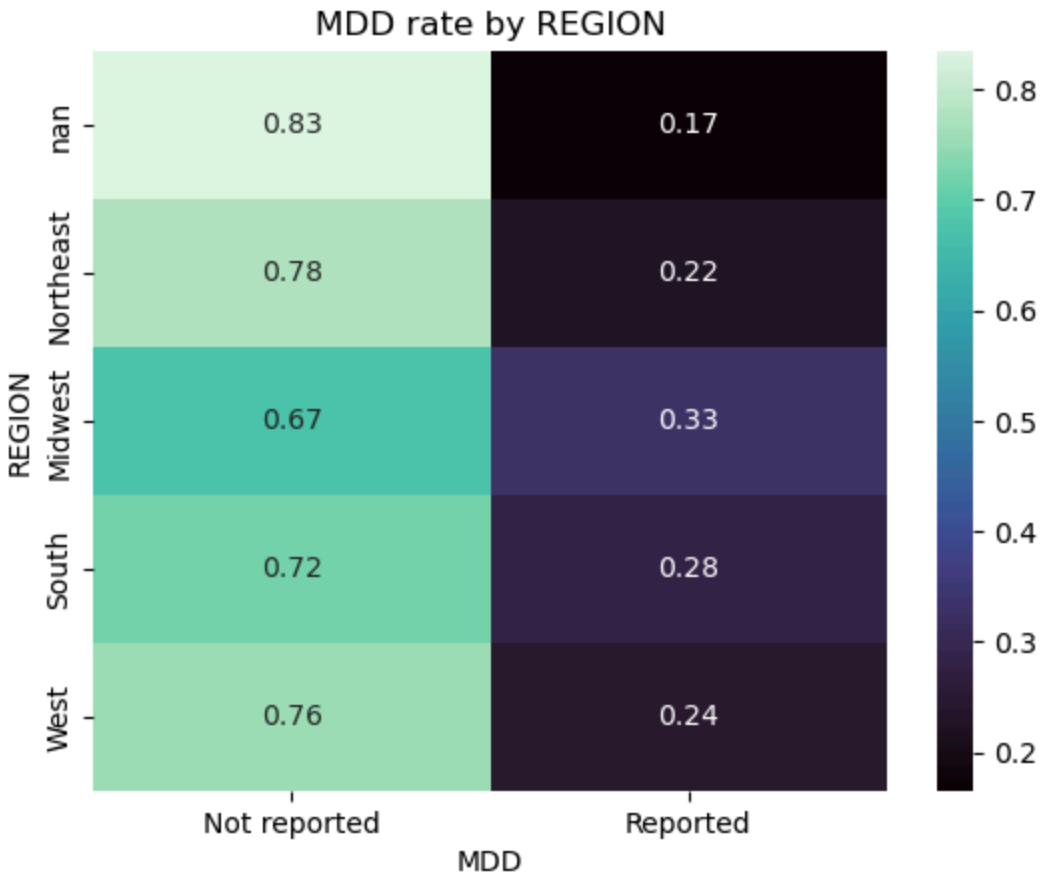
In [124…

```
# Bar chart of EMPLOY and MDD

subset = model_df[model_df["EMPLOY"] == 5]
counts = subset["MDD"].value_counts().sort_index()

labels = ["No MDD", "Has MDD"]

plt.bar(labels, counts.values)
plt.ylabel("Count")
plt.title("MDD vs Non-MDD Among those not in the Labor Force")
plt.show()
```



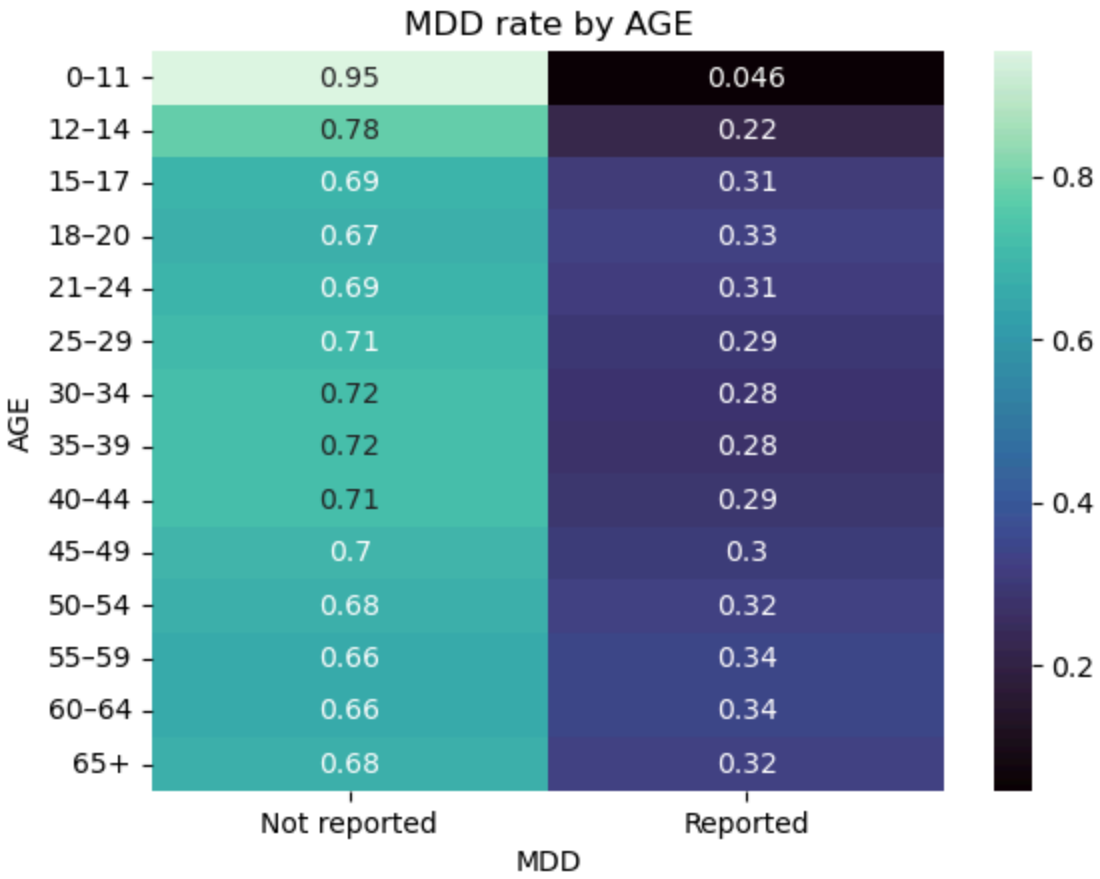
In [125…

```
# Heatmap of REGION and MDD

table = pd.crosstab(model_df["REGION"], model_df["MDD"], normalize='index')
table.index = table.index.map(region_map)
table.columns = table.columns.map(binary_flag_map)
sns.heatmap(table, annot=True, cmap="mako")
plt.title("MDD rate by REGION")
plt.show()
```



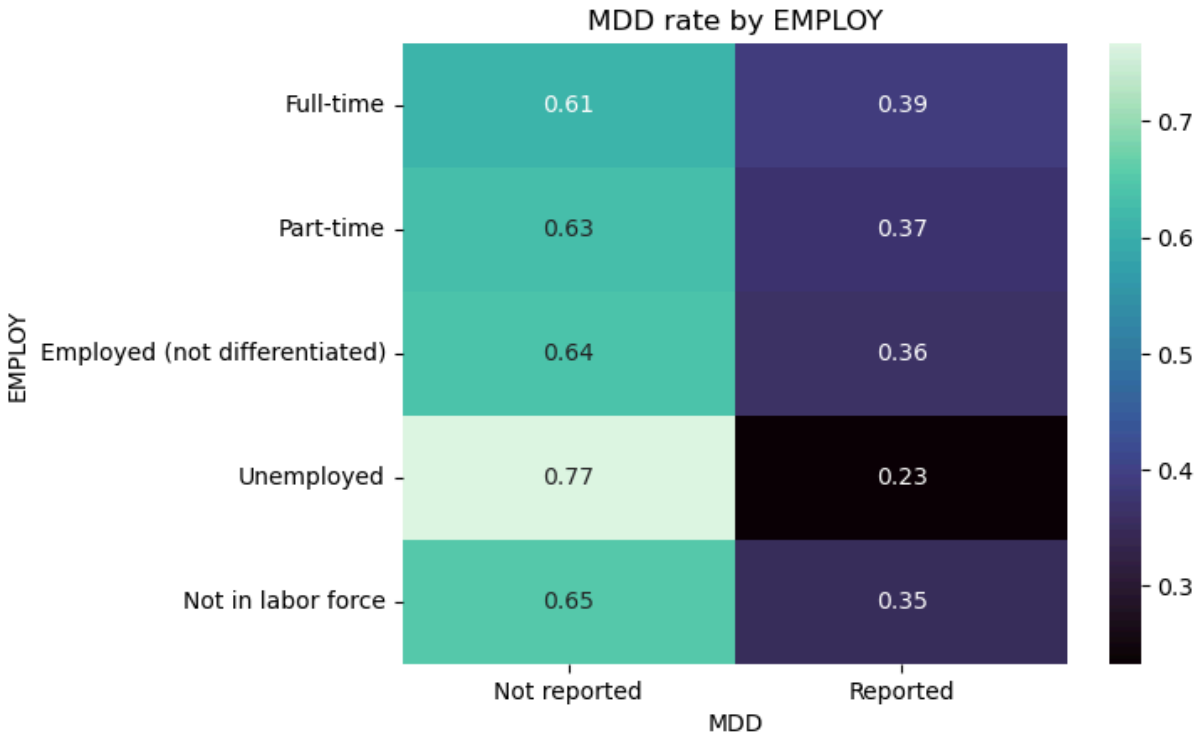
In [126…

```
# Heatmap of AGE and MDD

table = pd.crosstab(model_df["AGE"], model_df["MDD"], normalize='index')
table.index = table.index.map(age_map)
table.columns = table.columns.map(binary_flag_map)
sns.heatmap(table, annot=True, cmap="mako")
plt.title("MDD rate by AGE")
plt.show()
```



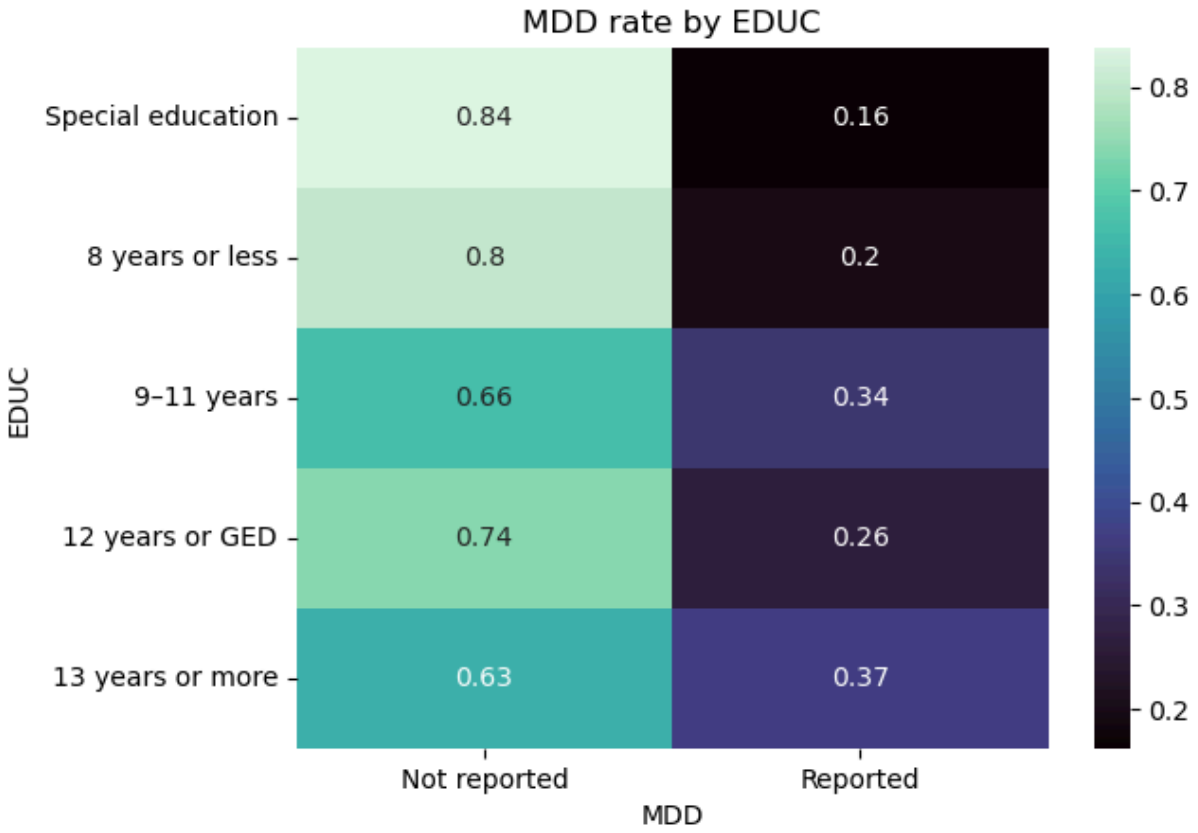
In [127…

```
# Heatmap of EMPLOY and MDD

table = pd.crosstab(model_df["EMPLOY"], model_df["MDD"], normalize='index')
table.index = table.index.map(employ_map)
table.columns = table.columns.map(binary_flag_map)
sns.heatmap(table, annot=True, cmap="mako")
plt.title("MDD rate by EMPLOY")
plt.show()
```



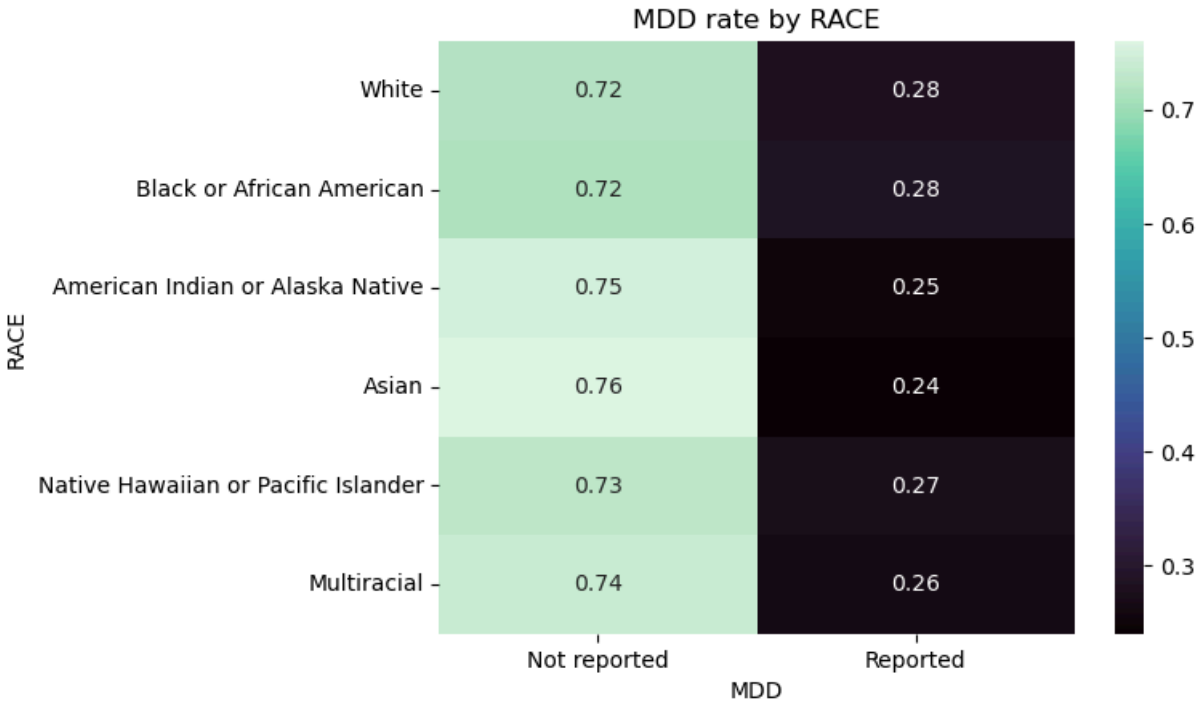
In [128…

```
# Heatmap of EDUC and MDD

table = pd.crosstab(model_df["EDUC"], model_df["MDD"], normalize='index')
table.index = table.index.map(educ_map)
table.columns = table.columns.map(binary_flag_map)
sns.heatmap(table, annot=True, cmap="mako")
plt.title("MDD rate by EDUC")
plt.show()
```



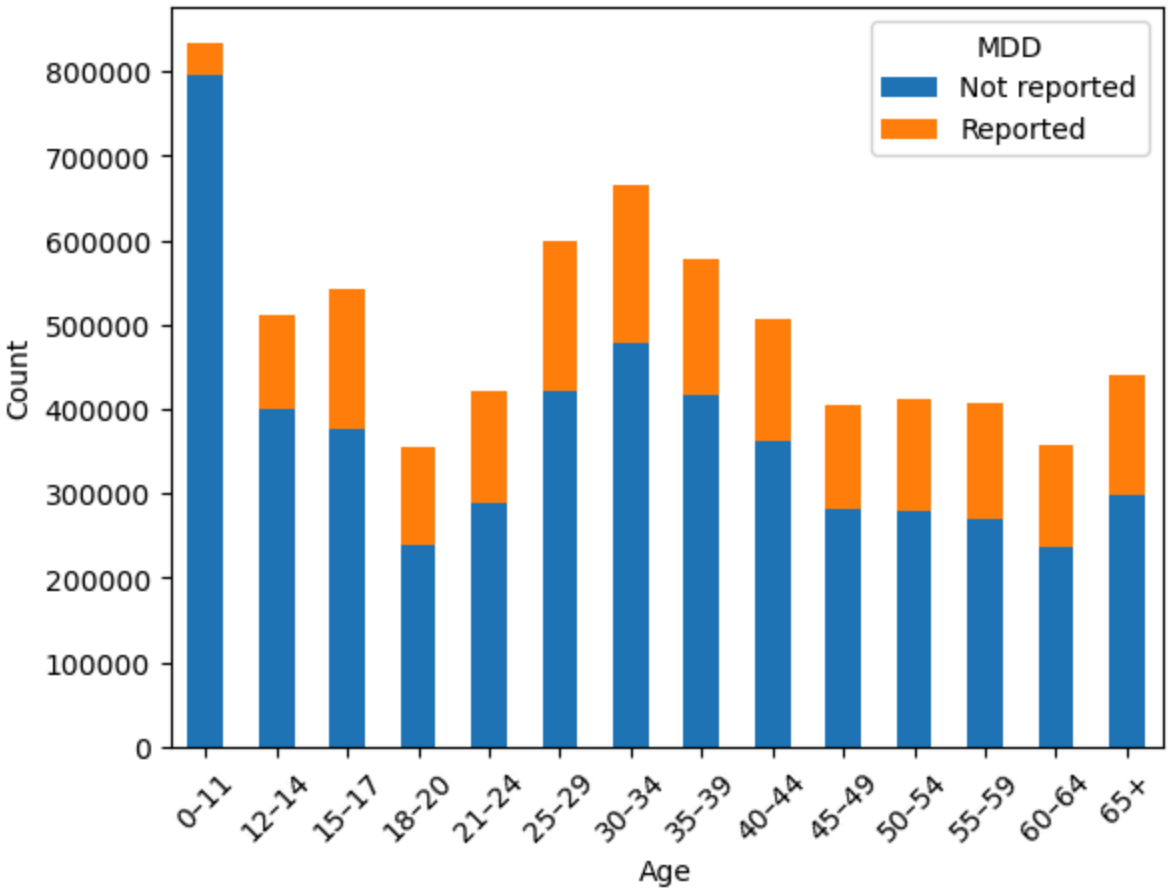
In [129…

```
# Heatmap of RACE and MDD

table = pd.crosstab(model_df["RACE"], model_df["MDD"], normalize='index')
table.index = table.index.map(race_map)
table.columns = table.columns.map(binary_flag_map)
sns.heatmap(table, annot=True, cmap="mako")
plt.title("MDD rate by RACE")
plt.show()
```



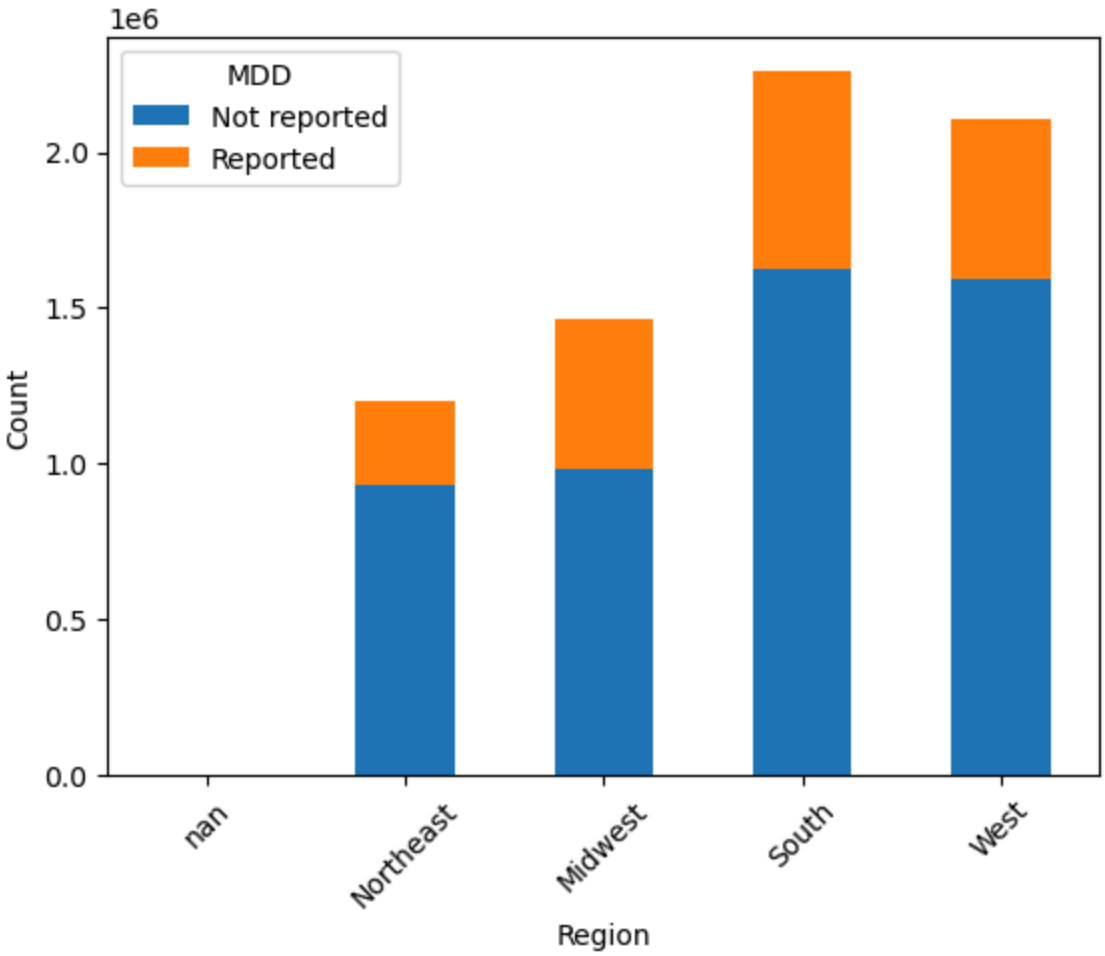
In [130…

```
# Stacked bar chart of AGE and MDD

table = pd.crosstab(model_df['AGE'], model_df['MDD'])
table.index = table.index.map(age_map)
table.columns = table.columns.map(binary_flag_map)
table.plot(kind='bar', stacked=True)

plt.xlabel("Age")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



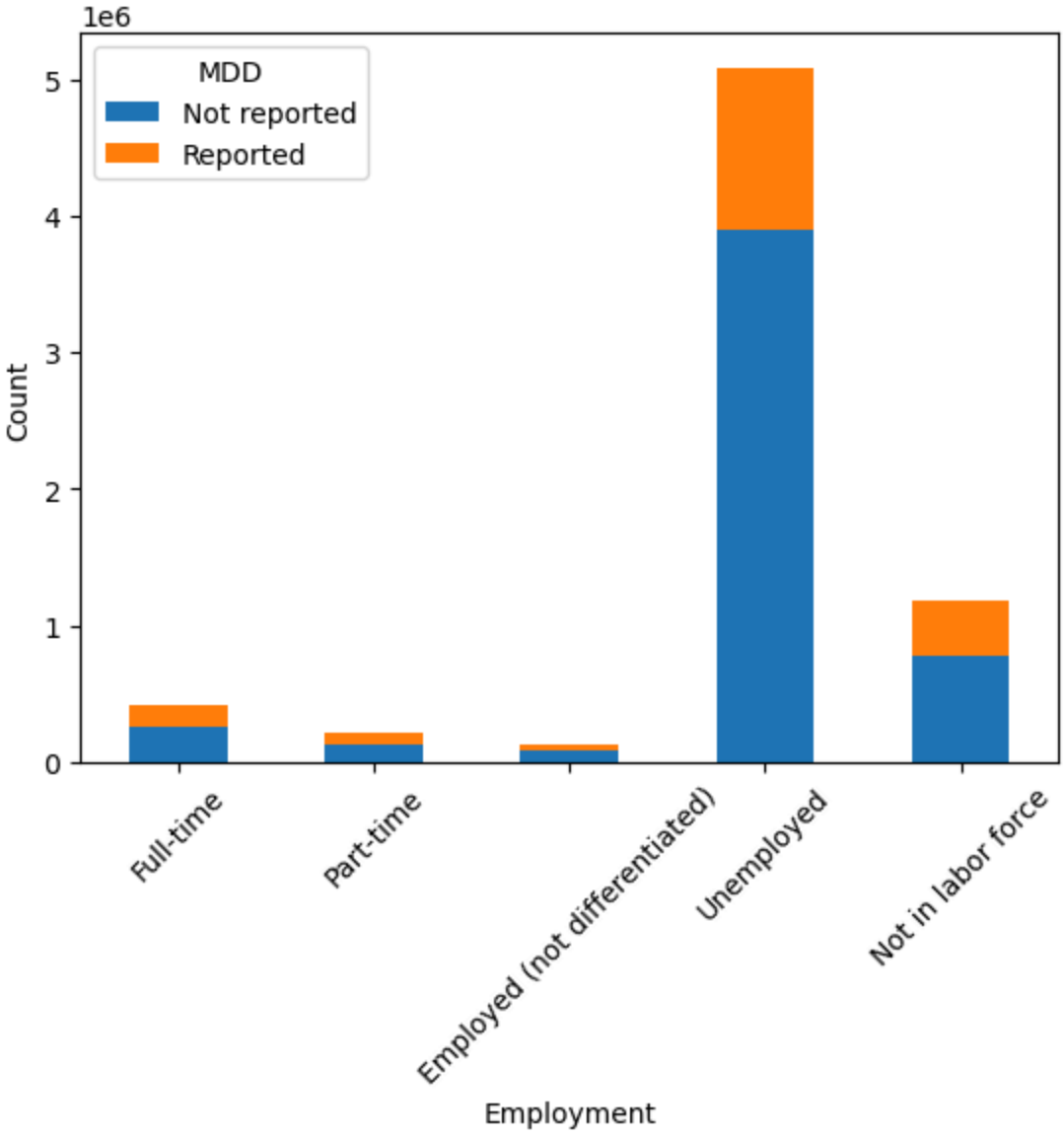
In [131…

```
# Stacked bar chart of REGION and MDD

table = pd.crosstab(model_df['REGION'], model_df['MDD'])
table.index = table.index.map(region_map)
table.columns = table.columns.map(binary_flag_map)
table.plot(kind='bar', stacked=True)

plt.xlabel("Region")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



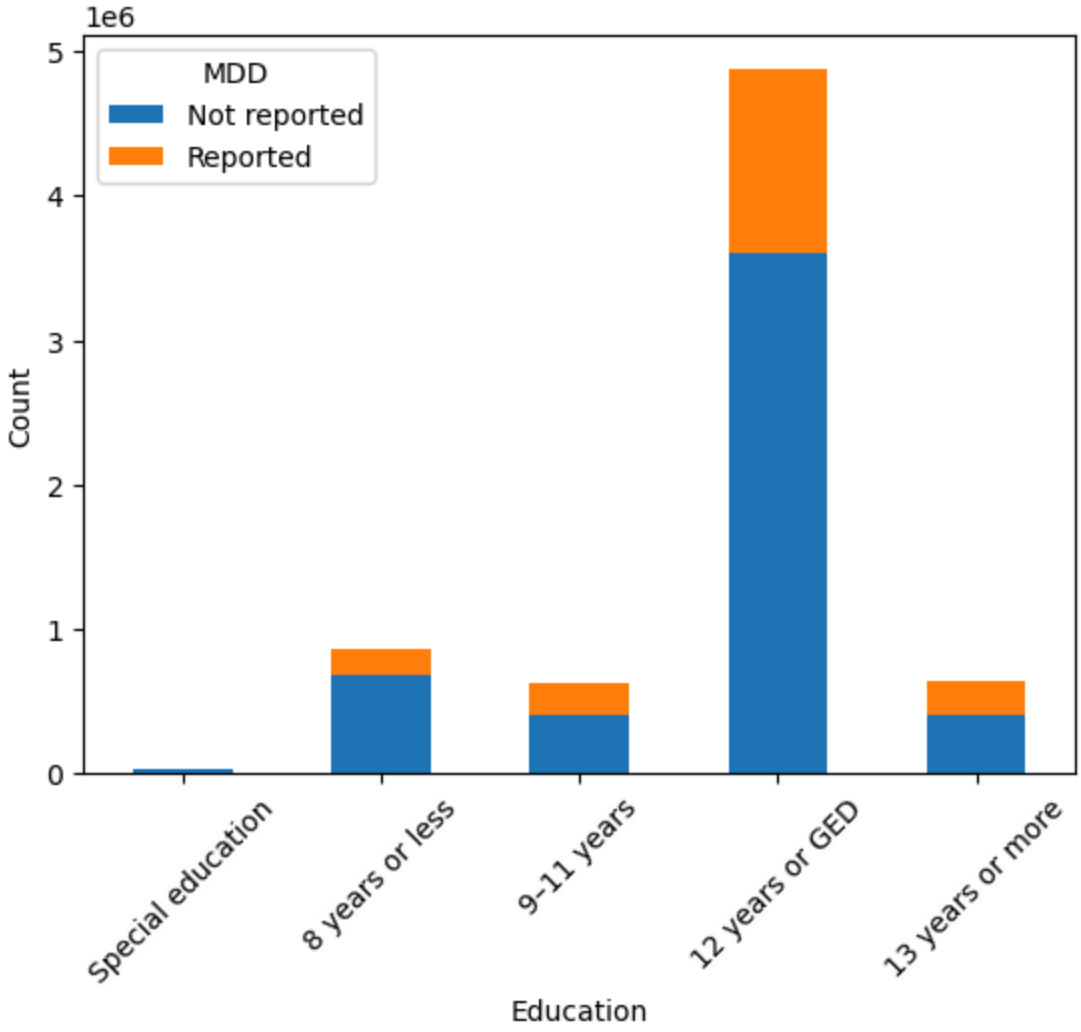
In [132…

```
# Stacked bar chart of EMPLOY and MDD

table = pd.crosstab(model_df['EMPLOY'], model_df['MDD'])
table.index = table.index.map(employ_map)
table.columns = table.columns.map(binary_flag_map)
table.plot(kind='bar', stacked=True)

plt.xlabel("Employment")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



In [133…

```
# Stacked bar chart of EDUC and MDD

table = pd.crosstab(model_df['EDUC'], model_df['MDD'])
table.index = table.index.map(educ_map)
table.columns = table.columns.map(binary_flag_map)
table.plot(kind='bar', stacked=True)

plt.xlabel("Education")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



Multicollinearity Check With VIF

In [135…

```
cols_for_model = X_train_full.columns
X_vif = sm.add_constant(X_train_full[cols_for_model]) ## add constant to design mat

## data frame to store VIF values
vif_df = pd.DataFrame({
    "Variable": X_vif.columns, ## store variable names
    "VIF": [ ## compute vif for each column
    variance_inflation_factor(X_vif.values, i)
    for i in range(X_vif.shape[1])
]
})
vif_df.sort_values("VIF",ascending=False) ## display vif vals (largest -> smallest)
```

Out[135…

	Variable	VIF
0	const	177.419202
22	ANY_OTHER_MH_DISORDER	2.403064
8	ANXIETYFLG	1.566431
7	TRAUSTREFLG	1.510625
16	SCHIZOFLG	1.475688
1	AGE	1.410863
9	ADHDFLG	1.285874
18	OTHERDISFLG	1.266606
12	BIPOLARFLG	1.254667
17	ALCSUBFLG	1.242731
21	HAS_SAP	1.211493
2	EDUC	1.184083
19	REGION	1.151393
13	ODDFLG	1.092693
3	ETHNIC	1.078541
5	SEX	1.071515
25	IS_MARRIED	1.059977
14	PDDFLG	1.053881
4	RACE	1.050702
10	CONDUCTFLG	1.049306
6	EMPLOY	1.044913
15	PERSONFLG	1.028440
24	IS_HOMELESS	1.026987
23	IS_VETERAN	1.024439
11	DELIRDEMFLG	1.015333
20	HAS_SUBSTANCE_USE	1.006624

Logistic Regression

In [137…

```
## Add intercept to training data
X_train_sm = sm.add_constant(X_train_full[cols_for_model])
## Add intercept to test data
```

```
X_test_sm = sm.add_constant(X_test[cols_for_model],has_constant="add")

## define logistic regression model
logit_model = sm.Logit(y_train_full, X_train_sm)
## fit logistic model
logit_res = logit_model.fit()

## summary of logistic regression results
print(logit_res.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.490612
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                    MDD   No. Observations:              5628512
Model:                          Logit   Df Residuals:                  5628486
Method:                           MLE   Df Model:                           25
Date:                Thu, 11 Dec 2025   Pseudo R-squ.:                 0.1573
Time:                        15:20:46   Log-Likelihood:            -2.7614e+06
converged:                       True   LL-Null:                   -3.2768e+06
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
=====
                             coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                    -1.3517      0.014    -97.436      0.000      -1.379      -
1.324
AGE                       0.0908      0.000    303.276      0.000       0.090
0.091
EDUC                     -0.0205      0.001    -14.051      0.000      -0.023      -
0.018
ETHNIC                   -0.0079      0.002     -3.441      0.001      -0.012      -
0.003
RACE                     -0.0093      0.001     -9.076      0.000      -0.011      -
0.007
SEX                       0.4235      0.002    194.280      0.000       0.419
0.428
EMPLOY                   -0.0378      0.001    -34.012      0.000      -0.040      -
0.036
TRAUSTREFLG              -0.1033      0.003    -30.294      0.000      -0.110      -
0.097
ANXIETYFLG                0.9476      0.003    298.803      0.000       0.941
0.954
ADHDFLG                  -0.1509      0.004    -34.016      0.000      -0.160      -
0.142
CONDUCTFLG               -0.6809      0.014    -48.989      0.000      -0.708      -
0.654
DELIRDEMFLG              -0.5271      0.020    -25.982      0.000      -0.567      -
0.487
BIPOLARFLG               -1.6874      0.006   -302.131      0.000      -1.698      -
1.676
ODDFLG                   -0.4604      0.011    -43.025      0.000      -0.481      -
0.439
PDDFLG                   -0.5871      0.011    -53.463      0.000      -0.609      -
0.566
PERSONFLG                 0.4671      0.007     67.832      0.000       0.454
0.481
SCHIZOFLG                -1.6263      0.006   -277.067      0.000      -1.638      -
1.615
ALCSUBFLG                -0.1113      0.005    -20.495      0.000      -0.122      -
0.101
OTHERDISFLG              -0.4653      0.004   -113.792      0.000      -0.473      -
0.457
```

```
REGION                  0.0510      0.001     48.146      0.000       0.049
0.053
HAS_SUBSTANCE_USE      -0.0939      0.019     -4.834      0.000      -0.132      -
0.056
HAS_SAP                 0.2688      0.002    112.651      0.000       0.264
0.274
ANY_OTHER_MH_DISORDER  -1.2286      0.004   -314.887      0.000      -1.236      -
1.221
IS_VETERAN             -0.1545      0.009    -18.017      0.000      -0.171      -
0.138
IS_HOMELESS             0.1771      0.006     27.812      0.000       0.165
0.190
IS_MARRIED              0.0576      0.004     13.614      0.000       0.049
0.066
==============================================================================
=====
```

Odds Ratios

In [139…

```python
## extract estimated coefficients from fitted logit model
params = logit_res.params
## extract confidence intervals
conf = logit_res.conf_int()
## name confidence interval columns
conf.columns = ["2.5%", "97.5%"]

## table of odds ratios and confidence intervals
or_table = pd.DataFrame({
    "Odds_Ratio": np.exp(params), ## log-odds coefficients to odds ratios
    "CI_2.5%": np.exp(conf["2.5%"]), ## lower bound of 95% CI
    "CI_97.5%": np.exp(conf["97.5%"]), ## upper bound of 95% CI
    "p_value": logit_res.pvalues ## corresponding p vals for hypothesis tests
})
## view, sorted by p values
or_table.sort_values("p_value")
```

Out[139…

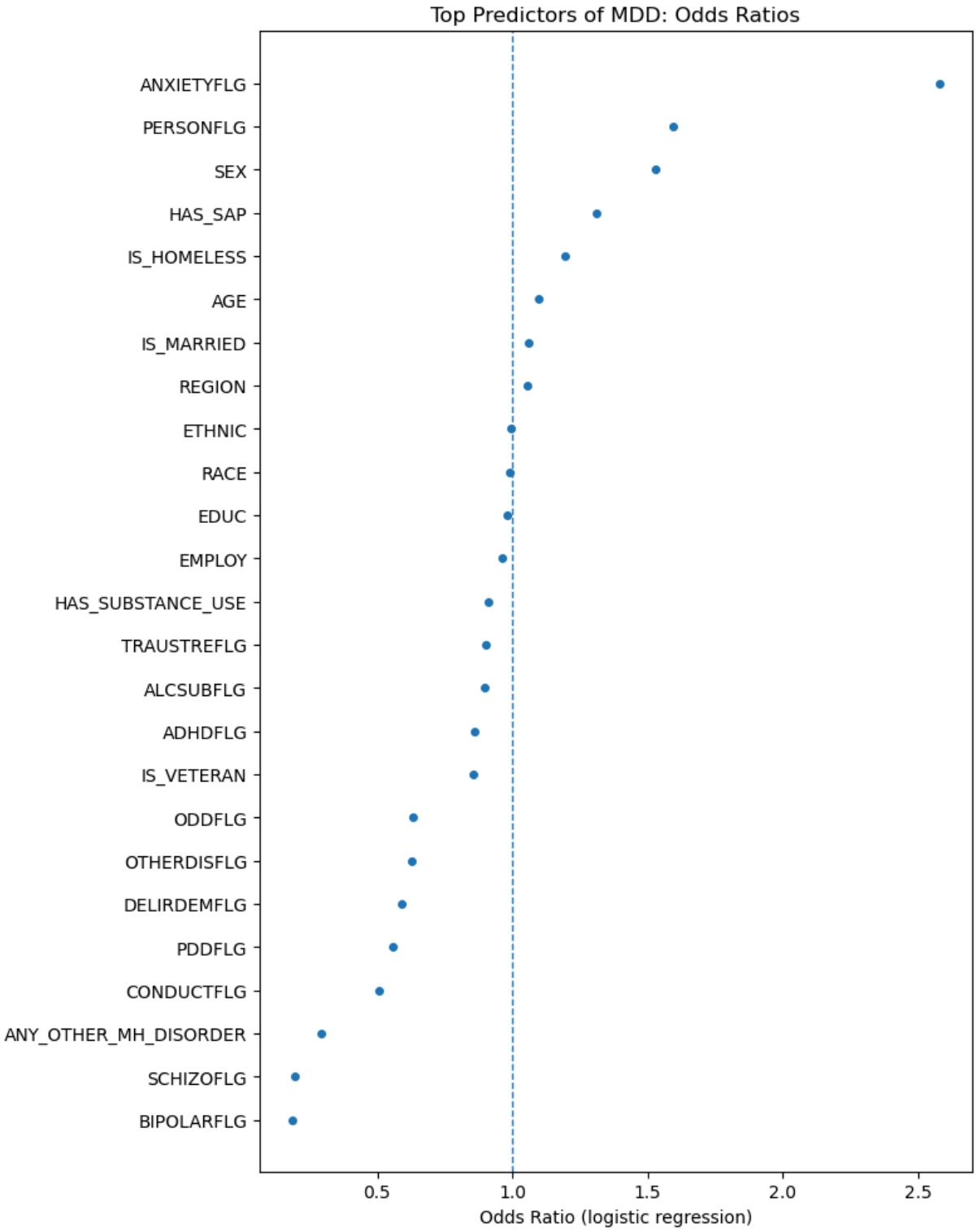
	Odds_Ratio	CI_2.5%	CI_97.5%	p_value
const	0.258803	0.251861	0.265937	0.000000e+00
ANY_OTHER_MH_DISORDER	0.292698	0.290468	0.294944	0.000000e+00
HAS_SAP	1.308451	1.302345	1.314585	0.000000e+00
REGION	1.052271	1.050091	1.054456	0.000000e+00
OTHERDISFLG	0.627925	0.622912	0.632978	0.000000e+00
SCHIZOFLG	0.196657	0.194408	0.198933	0.000000e+00
PERSONFLG	1.595398	1.574009	1.617077	0.000000e+00
PDDFLG	0.555934	0.544097	0.568030	0.000000e+00
ODDFLG	0.631062	0.617965	0.644435	0.000000e+00
CONDUCTFLG	0.506181	0.492578	0.520158	0.000000e+00
BIPOLARFLG	0.185008	0.182994	0.187045	0.000000e+00
ANXIETYFLG	2.579616	2.563631	2.595700	0.000000e+00
SEX	1.527368	1.520855	1.533908	0.000000e+00
AGE	1.095070	1.094427	1.095713	0.000000e+00
ADHDFLG	0.859958	0.852515	0.867467	1.311017e-253
EMPLOY	0.962929	0.960835	0.965027	1.477507e-253
TRAUSTREFLG	0.901844	0.895836	0.907892	1.388096e-201
IS_HOMELESS	1.193736	1.178931	1.208727	3.109690e-170
DELIRDEMFLG	0.590288	0.567276	0.614235	7.952655e-149
ALCSUBFLG	0.894646	0.885171	0.904221	2.391807e-93
IS_VETERAN	0.856880	0.842603	0.871400	1.435739e-72
EDUC	0.979691	0.976891	0.982499	7.548471e-45
IS_MARRIED	1.059312	1.050561	1.068136	3.323589e-42
RACE	0.990792	0.988815	0.992773	1.128874e-19
HAS_SUBSTANCE_USE	0.910342	0.876321	0.945685	1.340037e-06
ETHNIC	0.992133	0.987679	0.996607	5.802098e-04

Odds Ratio Plot (top predictors)

In [141…

```
or_plot = or_table.drop(index="const") ## drop intercept for plotting

or_plot = or_plot.sort_values("Odds_Ratio") ## sort by odds ratio
```

```
## positions for predictors on y axis
y_pos = np.arange(len(or_plot))

## horizontal odds ratio plot 95% CI
plt.figure(figsize=(8,10))

## values for plotting
odds = or_plot["Odds_Ratio"].values

## odds ratios as points
plt.plot(
    odds,
    y_pos,
    "o",
    markersize = 4
)
plt.axvline(1.0,linestyle="--",linewidth=1) ## reference line OR=1 (no effect)

plt.yticks(y_pos, or_plot.index) ## label y axis with predictor names
plt.xlabel("Odds Ratio (logistic regression)")
plt.title("Top Predictors of MDD: Odds Ratios")
plt.tight_layout()
plt.show()
```



Probit Regression

In [143…

```
## define probit regression model
probit_model = sm.Probit(y_train_full, X_train_sm)
## fit probit model
probit_res = probit_model.fit()
```

```
## summary of probit regression
print(probit_res.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.491025
         Iterations 6
                          Probit Regression Results
==============================================================================
Dep. Variable:                    MDD   No. Observations:              5628512
Model:                         Probit   Df Residuals:                  5628486
Method:                           MLE   Df Model:                           25
Date:                Thu, 11 Dec 2025   Pseudo R-squ.:                 0.1566
Time:                        15:21:21   Log-Likelihood:            -2.7637e+06
converged:                       True   LL-Null:                   -3.2768e+06
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
=====
                             coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                    -0.7969      0.008    -98.173      0.000      -0.813      -
0.781
AGE                       0.0545      0.000    307.604      0.000       0.054
0.055
EDUC                     -0.0115      0.001    -13.654      0.000      -0.013      -
0.010
ETHNIC                   -0.0043      0.001     -3.209      0.001      -0.007      -
0.002
RACE                     -0.0046      0.001     -7.763      0.000      -0.006      -
0.003
SEX                       0.2466      0.001    193.897      0.000       0.244
0.249
EMPLOY                   -0.0236      0.001    -36.141      0.000      -0.025      -
0.022
TRAUSTREFLG              -0.0679      0.002    -34.728      0.000      -0.072      -
0.064
ANXIETYFLG                0.5459      0.002    301.385      0.000       0.542
0.549
ADHDFLG                  -0.0829      0.002    -33.386      0.000      -0.088      -
0.078
CONDUCTFLG               -0.3482      0.007    -49.858      0.000      -0.362      -
0.335
DELIRDEMFLG              -0.3112      0.011    -27.414      0.000      -0.333      -
0.289
BIPOLARFLG               -0.9059      0.003   -320.323      0.000      -0.911      -
0.900
ODDFLG                   -0.2310      0.005    -42.126      0.000      -0.242      -
0.220
PDDFLG                   -0.3096      0.006    -54.557      0.000      -0.321      -
0.298
PERSONFLG                 0.2544      0.004     62.868      0.000       0.246
0.262
SCHIZOFLG                -0.8705      0.003   -294.046      0.000      -0.876      -
0.865
ALCSUBFLG                -0.0691      0.003    -22.126      0.000      -0.075      -
0.063
OTHERDISFLG              -0.2622      0.002   -114.812      0.000      -0.267      -
0.258
```

```
REGION                    0.0280      0.001      45.017      0.000       0.027
0.029
HAS_SUBSTANCE_USE        -0.0437      0.011      -3.809      0.000      -0.066      -
0.021
HAS_SAP                   0.1522      0.001     108.800      0.000       0.149
0.155
ANY_OTHER_MH_DISORDER    -0.7290      0.002    -321.989      0.000      -0.733      -
0.725
IS_VETERAN               -0.0943      0.005     -18.521      0.000      -0.104      -
0.084
IS_HOMELESS               0.1034      0.004      27.817      0.000       0.096
0.111
IS_MARRIED                0.0322      0.003      12.775      0.000       0.027
0.037
==============================================================================
=====
```

Logistic vs Probit Coefficient Comparison

In [145…

```
## data frame to compare coefficients and p-values
comparison = pd.DataFrame({
    "Logit_coefficients": logit_res.params, ## estimated coefficients from logistic
    "Logit_p": logit_res.pvalues, ## corresponding p-values
    "Probit_coefficients": probit_res.params, ## estimated coefficients from probit
    "Probit_p": probit_res.pvalues ## corresponding p-values
})
comparison ## display comparison table
```

Out[145...

	Logit_coefficients	Logit_p	Probit_coefficients	Probi
const	-1.351687	0.000000e+00	-0.796946	0.000000e+
AGE	0.090818	0.000000e+00	0.054512	0.000000e+
EDUC	-0.020518	7.548471e-45	-0.011546	1.910119e-
ETHNIC	-0.007898	5.802098e-04	-0.004312	1.333122e-
RACE	-0.009250	1.128874e-19	-0.004626	8.299920e-
SEX	0.423546	0.000000e+00	0.246635	0.000000e+
EMPLOY	-0.037776	1.477507e-253	-0.023584	5.19222
TRAUSTREFLG	-0.103314	1.388096e-201	-0.067941	3.02888
ANXIETYFLG	0.947640	0.000000e+00	0.545917	0.000000e+
ADHDFLG	-0.150872	1.311017e-253	-0.082906	2.15277
CONDUCTFLG	-0.680862	0.000000e+00	-0.348210	0.000000e+
DELIRDEMFLG	-0.527144	7.952655e-149	-0.311192	1.86493
BIPOLARFLG	-1.687354	0.000000e+00	-0.905875	0.000000e+
ODDFLG	-0.460352	0.000000e+00	-0.230970	0.000000e+
PDDFLG	-0.587105	0.000000e+00	-0.309573	0.000000e+
PERSONFLG	0.467123	0.000000e+00	0.254378	0.000000e+
SCHIZOFLG	-1.626294	0.000000e+00	-0.870476	0.000000e+
ALCSUBFLG	-0.111328	2.391807e-93	-0.069128	1.77970
OTHERDISFLG	-0.465335	0.000000e+00	-0.262173	0.000000e+
REGION	0.050951	0.000000e+00	0.027962	0.000000e+
HAS_SUBSTANCE_USE	-0.093934	1.340037e-06	-0.043669	1.396182e-
HAS_SAP	0.268844	0.000000e+00	0.152159	0.000000e+
ANY_OTHER_MH_DISORDER	-1.228616	0.000000e+00	-0.728988	0.000000e+
IS_VETERAN	-0.154457	1.435739e-72	-0.094276	1.392524e-
IS_HOMELESS	0.177088	3.109690e-170	0.103447	2.71885
IS_MARRIED	0.057619	3.323589e-42	0.032170	2.261627e-

Model Fit Metrics

In [147…

```
## predicted probabilities
logit_probs = logit_res.predict(X_test_sm)
probit_probs = probit_res.predict(X_test_sm)

## Area under ROC curve (AUC)
logit_auc = roc_auc_score(y_test, logit_probs)
probit_auc = roc_auc_score(y_test, probit_probs)

print("Logistic AUC:", logit_auc)
print("Probit AUC:", probit_auc)

## Akaike (AIC) / Bayesian (BIC) information criteria
print("Logistic AIC:", logit_res.aic)
print("Logistic BIC:", logit_res.bic)
print("Probit AIC:", probit_res.aic)
print("Probit BIC:", probit_res.bic)
```

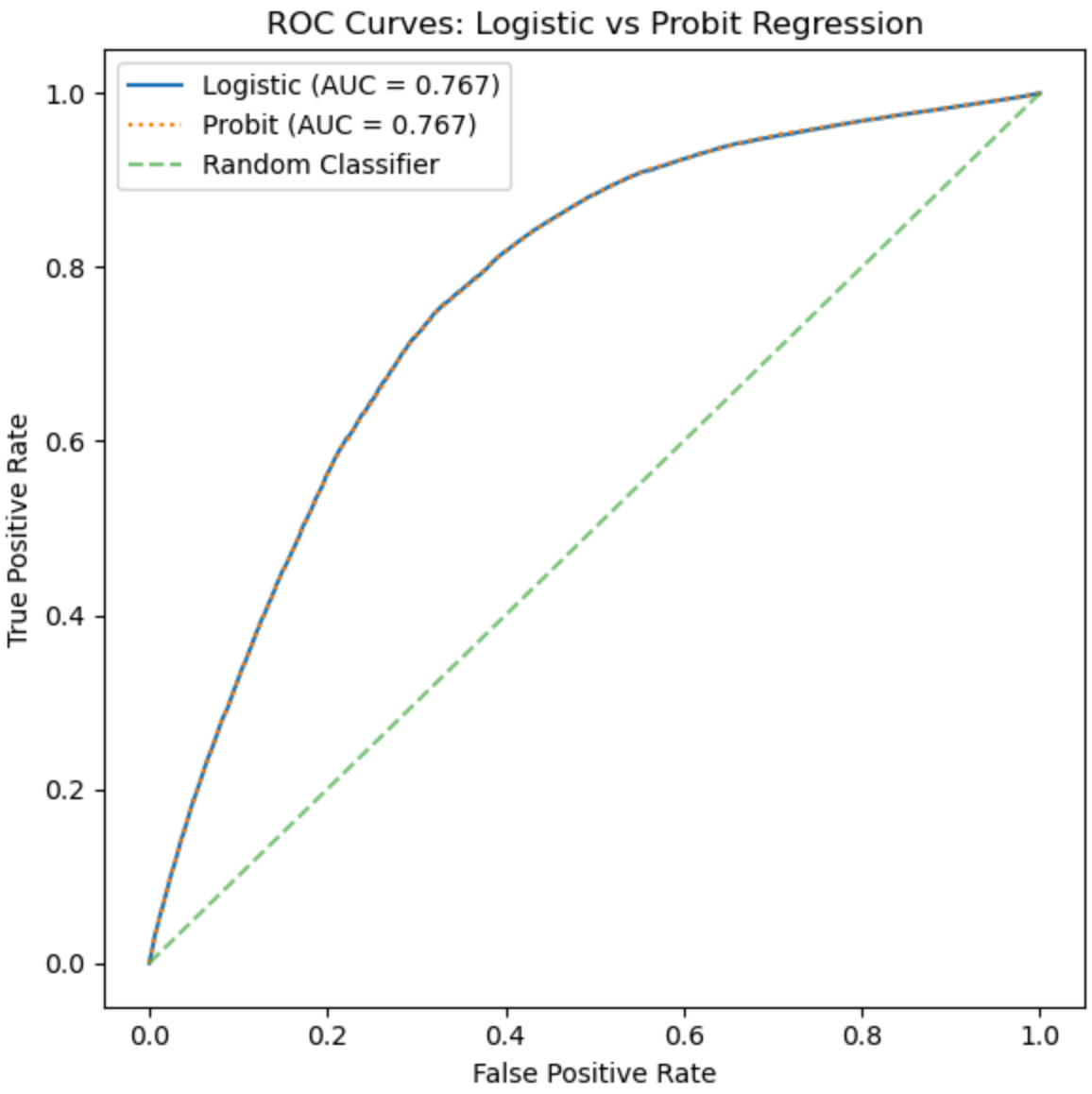
```
Logistic AUC: 0.7671143499944019
Probit AUC: 0.767049165121894
Logistic AIC: 5522885.426124824
Logistic BIC: 5523237.5533721605
Probit AIC: 5527531.43938103
Probit BIC: 5527883.566628367
```

ROC Curve: Logistic vs Probit

In [149…

```
## false positive and true positive values
fpr_logit, tpr_logit, _ = roc_curve(y_test, logit_probs)
fpr_probit, tpr_probit, _ = roc_curve(y_test, probit_probs)

plt.figure(figsize=(6,6))

## plot ROC curves
plt.plot(
    fpr_logit,
    tpr_logit,
    label = f"Logistic (AUC = {logit_auc:.3f})"
)
plt.plot(
    fpr_probit,
    tpr_probit,
    linestyle=":",
    label = f"Probit (AUC = {probit_auc:.3f})"
)

plt.plot([0,1], [0,1], linestyle="--", alpha = 0.6, label="Random Classifier") ## r

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves: Logistic vs Probit Regression")
plt.legend()
```

```
plt.tight_layout()
plt.show()
```



Predictive Modeling

Random Forest Model

In [152…

```
# Defining random forest model
rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=15,
    max_features="sqrt",
    n_jobs=-1,
    class_weight="balanced",
    random_state=42
)
```

In [154…
```
# Fitting random forest model
rf.fit(X_train, y_train)
```

Out[154…
```
RandomForestClassifier(class_weight='balanced', max_depth=15, n_estimators
=200,
                       n_jobs=-1, random_state=42)
```

In [155…
```
# Making predictions on test data
y_pred = rf.predict(X_test)
y_prob = rf.predict_proba(X_test)[:, 1]
```

Stochastic Gradient Descent (SGD) Model

Since we are unable to use a support vector machine model with data of this size, we implement a stochastic gradient descent model instead.

In [158…
```
# Scaling the data and fitting the model
sgd = Pipeline([
    ("scaler", StandardScaler(with_mean=False)),
    ("sgd", SGDClassifier(
        loss="log_loss",
        max_iter=1000,
        tol=1e-3,
        random_state=42
    ))
])
```

In [159…
```
# Fitting the model
sgd.fit(X_train, y_train)
```

Out[159…
```
Pipeline
  StandardScaler
  SGDClassifier
```

In [160…
```
# Making predictions on test data
y_pred = sgd.predict(X_test)
```

Light Gradient Boosting Machine (LIGHTGBM) Model

In [162…
```
# Defining LIGHTBGM model
lgbm = lgb.LGBMClassifier(
    num_leaves=64,
    learning_rate=0.05,
```

```
    n_estimators=500,
    min_child_samples=50,
    subsample=0.8,
    colsample_bytree=0.8
)
```

In [163…
```
# Fitting LIGHTBGM model
lgbm.fit(X_train, y_train,eval_set=[(X_val, y_val)],eval_metric="auc")
```

```
C:\Users\janec\anaconda3\Lib\site-packages\joblib\externals\loky\backend\context.py:
136: UserWarning: Could not find the number of physical cores for the following reas
on:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warning by setti
ng LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "C:\Users\janec\anaconda3\Lib\site-packages\joblib\externals\loky\backend\con
text.py", line 257, in _count_physical_cores
    cpu_info = subprocess.run(
               ^^^^^^^^^^^^^^
  File "C:\Users\janec\anaconda3\Lib\subprocess.py", line 548, in run
    with Popen(*popenargs, **kwargs) as process:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\janec\anaconda3\Lib\subprocess.py", line 1026, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "C:\Users\janec\anaconda3\Lib\subprocess.py", line 1538, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
[LightGBM] [Info] Number of positive: 1210878, number of negative: 3291931
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
s 0.175258 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 83
[LightGBM] [Info] Number of data points in the train set: 4502809, number of used fe
atures: 25
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.268916 -> initscore=-1.000129
[LightGBM] [Info] Start training from score -1.000129
```

Out[163…
```
LGBMClassifier(colsample_bytree=0.8, learning_rate=0.05, min_child_samples
=50,
               n_estimators=500, num_leaves=64, subsample=0.8)
```

In [164…
```
# Making predictions on test data
y_pred = lgbm.predict(X_test)
y_prob = lgbm.predict_proba(X_test)[:,1]
```

CatBoost Model

In [166…
```
# Defining CatBoost model
cat = CatBoostClassifier(
```

```
    iterations=2000,
    learning_rate=0.03,
    depth=6,
    od_type="Iter",
    od_wait=40,
    task_type="CPU",
    verbose=100
)
```

In [167…

```
# Fitting CatBoost model
cat.fit(
    X_train, y_train,
    eval_set=(X_val, y_val),
    use_best_model=True
)
```

```
0:      learn: 0.6772069        test: 0.6773291 best: 0.6773291 (0)     total: 407ms
remaining: 13m 32s
100:    learn: 0.4643240        test: 0.4646228 best: 0.4646228 (100)   total: 24.1s
remaining: 7m 33s
200:    learn: 0.4531630        test: 0.4534787 best: 0.4534787 (200)   total: 47.1s
remaining: 7m 1s
300:    learn: 0.4482528        test: 0.4485497 best: 0.4485497 (300)   total: 1m 10
s       remaining: 6m 36s
400:    learn: 0.4452232        test: 0.4455328 best: 0.4455328 (400)   total: 1m 32
s       remaining: 6m 10s
500:    learn: 0.4429996        test: 0.4433217 best: 0.4433217 (500)   total: 1m 55
s       remaining: 5m 45s
600:    learn: 0.4411004        test: 0.4414250 best: 0.4414250 (600)   total: 2m 18
s       remaining: 5m 22s
700:    learn: 0.4395509        test: 0.4398940 best: 0.4398940 (700)   total: 2m 40
s       remaining: 4m 57s
800:    learn: 0.4382586        test: 0.4386196 best: 0.4386196 (800)   total: 3m 3s
remaining: 4m 34s
900:    learn: 0.4371356        test: 0.4375073 best: 0.4375073 (900)   total: 3m 25
s       remaining: 4m 11s
1000:   learn: 0.4362507        test: 0.4366301 best: 0.4366301 (1000)  total: 3m 48
s       remaining: 3m 47s
1100:   learn: 0.4354789        test: 0.4358811 best: 0.4358811 (1100)  total: 4m 10
s       remaining: 3m 24s
1200:   learn: 0.4347459        test: 0.4351630 best: 0.4351630 (1200)  total: 4m 32
s       remaining: 3m 1s
1300:   learn: 0.4341195        test: 0.4345500 best: 0.4345500 (1300)  total: 4m 54
s       remaining: 2m 38s
1400:   learn: 0.4335553        test: 0.4339995 best: 0.4339995 (1400)  total: 5m 17
s       remaining: 2m 15s
1500:   learn: 0.4330671        test: 0.4335276 best: 0.4335276 (1500)  total: 5m 39
s       remaining: 1m 52s
1600:   learn: 0.4326280        test: 0.4331066 best: 0.4331066 (1600)  total: 6m 1s
remaining: 1m 30s
1700:   learn: 0.4322286        test: 0.4327275 best: 0.4327275 (1700)  total: 6m 23
s       remaining: 1m 7s
1800:   learn: 0.4318661        test: 0.4323769 best: 0.4323769 (1800)  total: 6m 45
s       remaining: 44.8s
1900:   learn: 0.4315336        test: 0.4320570 best: 0.4320570 (1900)  total: 7m 8s
remaining: 22.3s
1999:   learn: 0.4311999        test: 0.4317355 best: 0.4317355 (1999)  total: 7m 30
s       remaining: 0us

bestTest = 0.431735468
bestIteration = 1999
```

Out[167… `<catboost.core.CatBoostClassifier at 0x2b31d094a10>`

In [168…

```
# Making predictions on test data
y_pred = cat.predict(X_test)
y_prob = cat.predict_proba(X_test)[:, 1]
```

Saving Models

In [170…
```
# Saving models
joblib.dump(rf, "random_forest_model.pkl")
joblib.dump(sgd, "sgd_model.pkl")
joblib.dump(lgbm, "lightgbm_model.pkl")
```

Out[170…
```
['lightgbm_model.pkl']
```

In [171…
```
# Loading models
rf = joblib.load("random_forest_model.pkl")
sgd = joblib.load("sgd_model.pkl")
lgbm = joblib.load("lightgbm_model.pkl")
```

In [172…
```
# Saving CatBoost model
cat.save_model("catboost_model.cbm")
```

In [173…
```
# Loading CatBoost model
cat = CatBoostClassifier()
cat.load_model("catboost_model.cbm")
```

Out[173…
```
<catboost.core.CatBoostClassifier at 0x2b1555178c0>
```

Model Evaluation

Creating a 'Model Evaluation Function'

In [176…
```
def evaluate_model(model, X_test, y_test, model_name="Model"):
    print(f"\n{model_name}")

    # Making predictions on the testing set
    y_pred = model.predict(X_test)

    # Obtaining probabilities (if possible)
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)[:, 1]
    elif hasattr(model, "decision_function"):
        y_scores = model.decision_function(X_test)
        # Converting scores to 0-1 using min-max scaling
        y_prob = (y_scores - y_scores.min()) / (y_scores.max() - y_scores.min())
    else:
        y_prob = None

    # Displaying evaluation metrics
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))

    if y_prob is not None:
        print("AUC:", roc_auc_score(y_test, y_prob))
        print("Brier Score:", brier_score_loss(y_test, y_prob))

    print("\nClassification Report:")
```

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

    # Displaying Confusion Matrix
    con_mat = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(6, 5))
    sns.heatmap(
        con_mat,
        annot=True,
        fmt="d",
        cmap="Blues",
        linewidths=.5,
        linecolor='black'
    )
    plt.title(f"{model_name} - Confusion Matrix", fontsize=14)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.tight_layout()
    plt.show()

    # Displaying ROC Curve
    if y_prob is not None:
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        auc = roc_auc_score(y_test, y_prob)
        plt.figure(figsize=(7,6))
        plt.plot(fpr, tpr, color="#5b3eb5", linewidth=2.5, label=f"AUC = {auc:.3f}"
        plt.plot([0, 1], [0, 1], linestyle="--", color="gray", linewidth=1.5)
        plt.title(f"{model_name} - ROC Curve")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.legend(frameon=True)
        plt.grid(alpha=0.3)
        plt.tight_layout()
        plt.show()

    # Displaying Calibration Curve
    if y_prob is not None:
        prob_true, prob_pred = calibration_curve(y_test, y_prob, n_bins=10)
        plt.figure(figsize=(7,6))
        plt.plot(prob_pred, prob_true, marker="o", linestyle="-", color="#5887b0",
        plt.plot([0, 1], [0, 1], "--", color="gray", linewidth=1.5)
        plt.title(f"{model_name} - Calibration Curve")
        plt.xlabel("Predicted Probability")
        plt.ylabel("Actual Probability")
        plt.grid(alpha=0.3)
        plt.tight_layout()
        plt.show()

    # Displaying Probability Curve
    if y_prob is not None:
        plt.figure(figsize=(7,6))
        sns.histplot(y_prob, bins=30, kde=True, color="#819bd4")

        plt.title(f"{model_name} - Predicted Probability Distribution", fontsize=15
        plt.xlabel("Predicted Probability")
        plt.ylabel("Count")
```

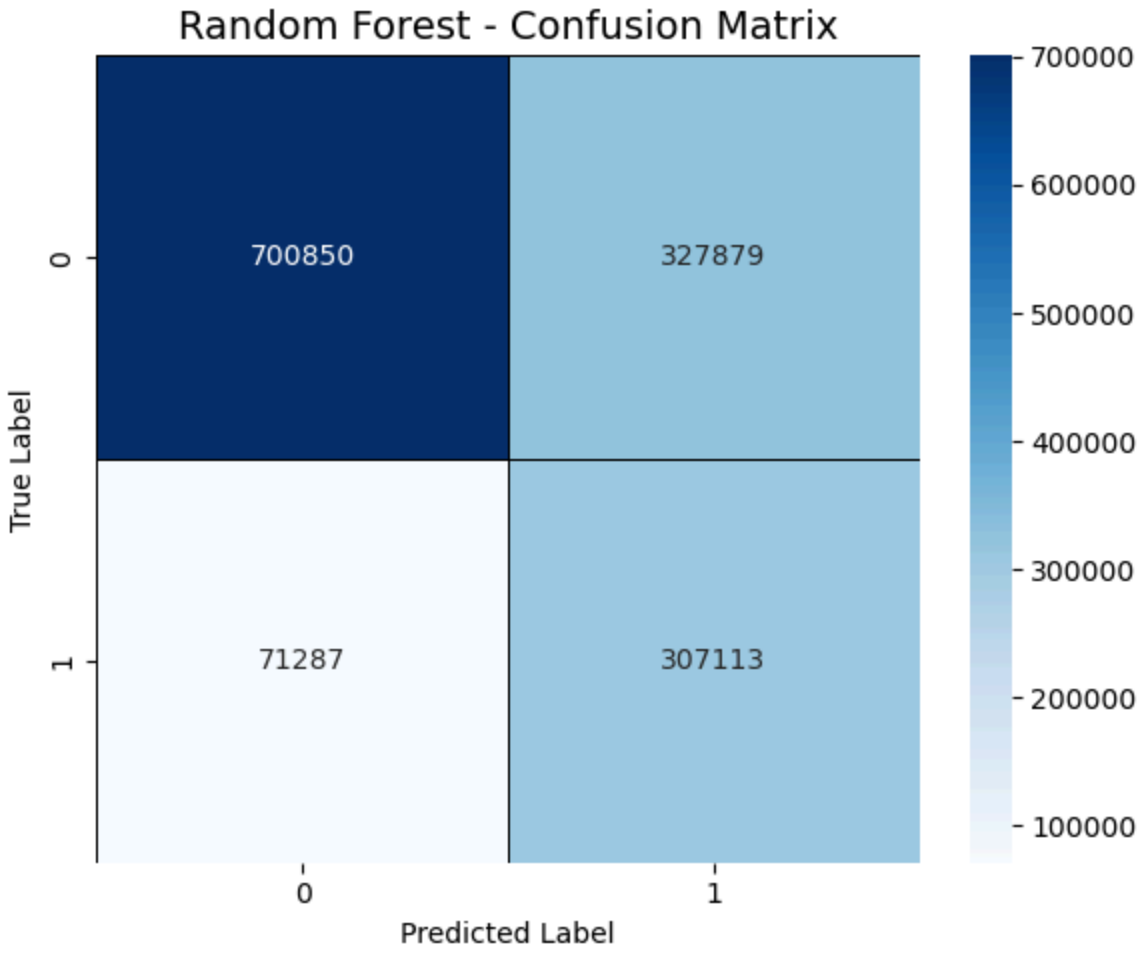
```
        plt.grid(alpha=0.3)
        plt.tight_layout()
        plt.show()

    return y_pred, y_prob
```

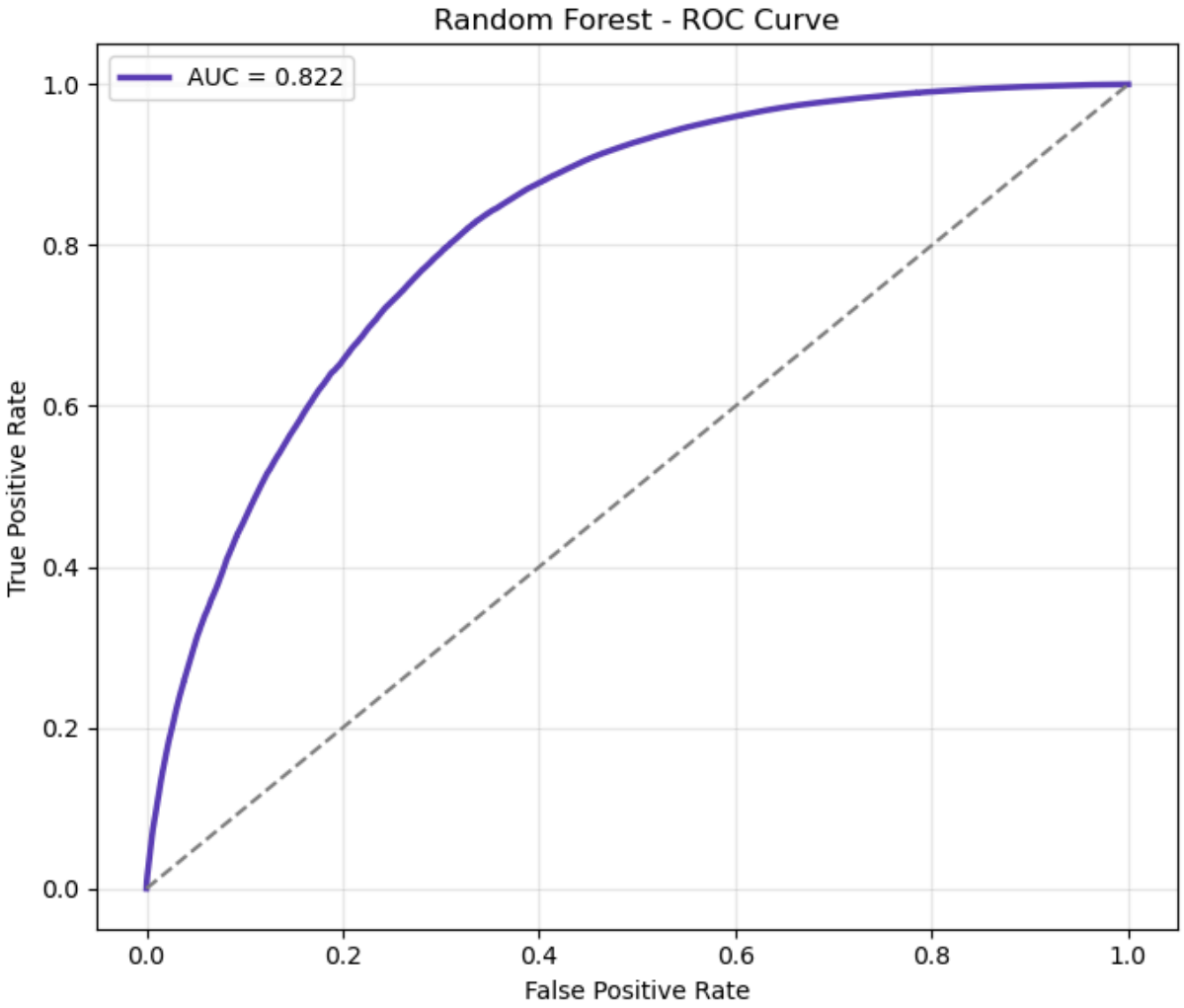
Evaluating Each Model

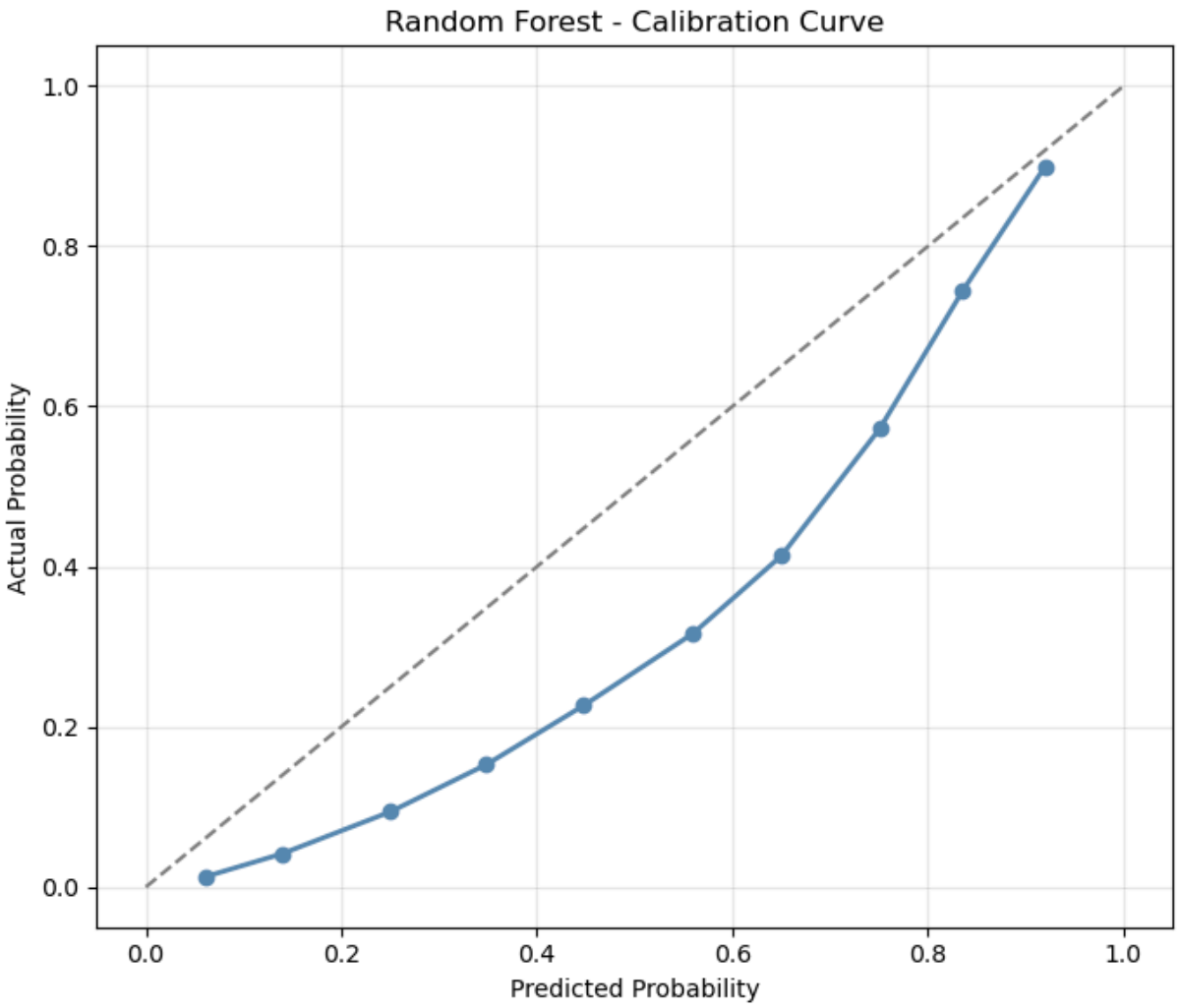
In [178…

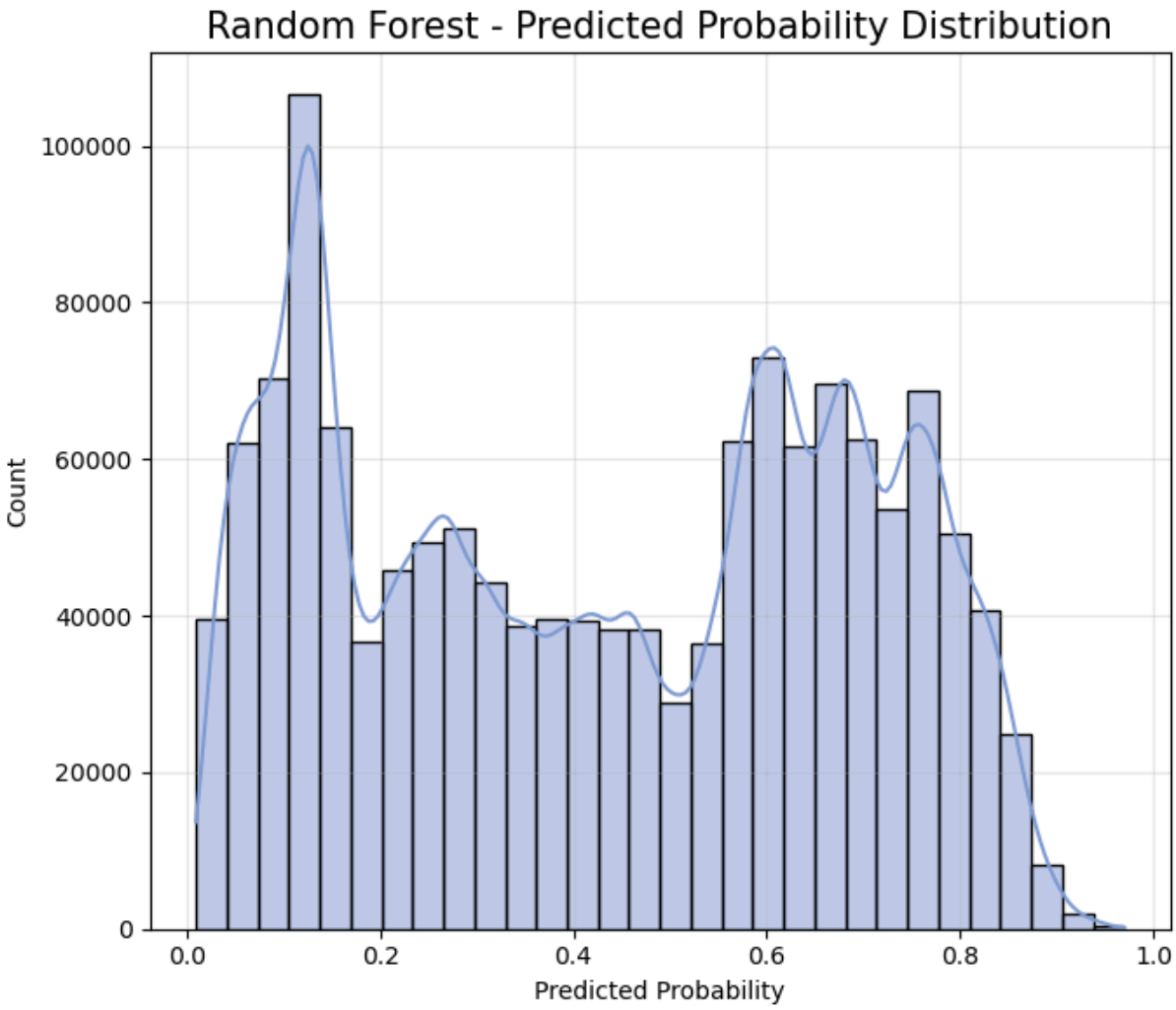
```
evaluate_model(rf, X_test, y_test, model_name="Random Forest")
```

```
Random Forest
Accuracy: 0.716325937422937
Precision: 0.4836486128959105
Recall: 0.8116094080338266
F1 Score: 0.6061089884269858
AUC: 0.8219912018973199
Brier Score: 0.17705954842353622

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.68      0.78   1028729
           1       0.48      0.81      0.61    378400

    accuracy                           0.72   1407129
   macro avg       0.70      0.75      0.69   1407129
weighted avg       0.79      0.72      0.73   1407129
```


Random Forest - Confusion Matrix
700850
327879
71287
307113
True Label
Predicted Label
0
1
0
1
700000
600000
500000
400000
300000
200000
100000


Random Forest - ROC Curve
AUC = 0.822
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

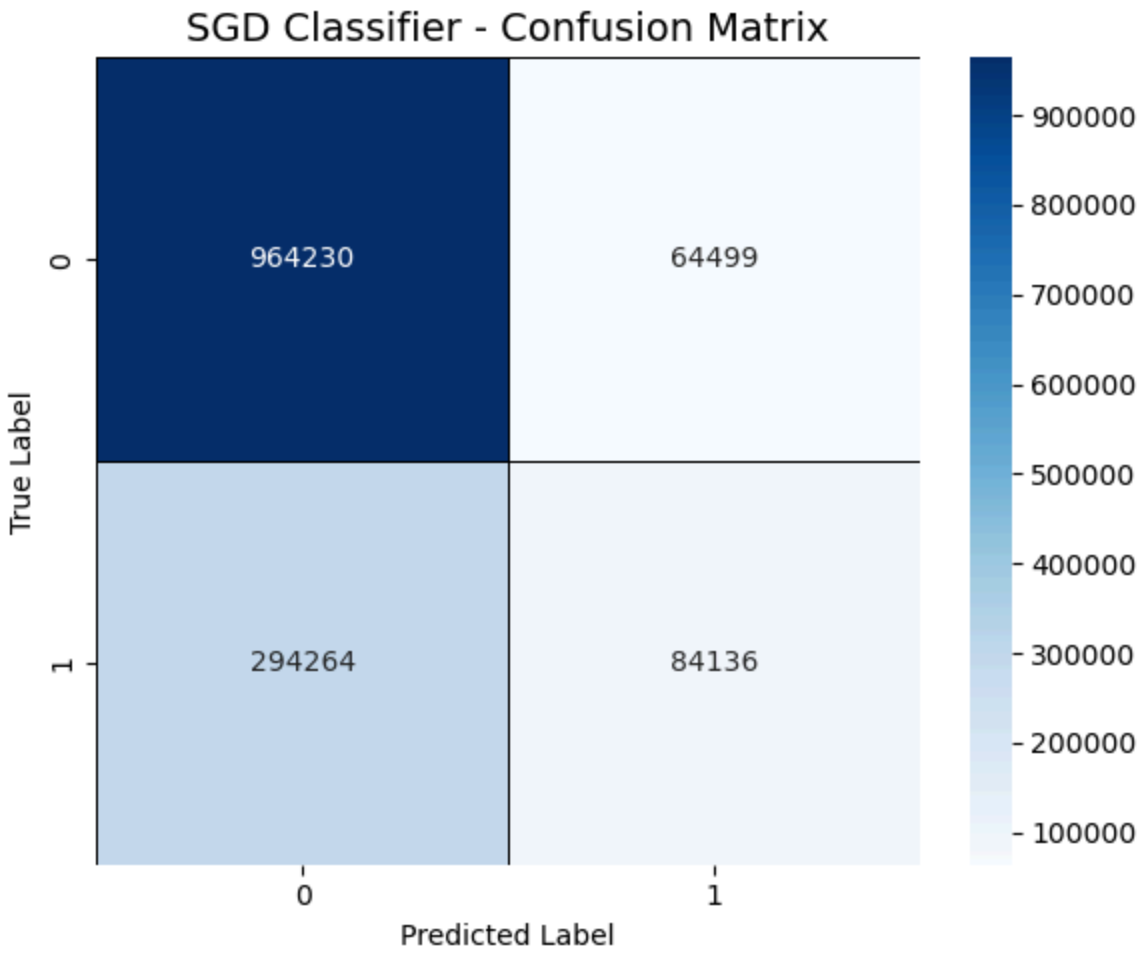

Random Forest - Calibration Curve
Actual Probability
Predicted Probability
0.0
0.2
0.4
0.6
0.8
1.0

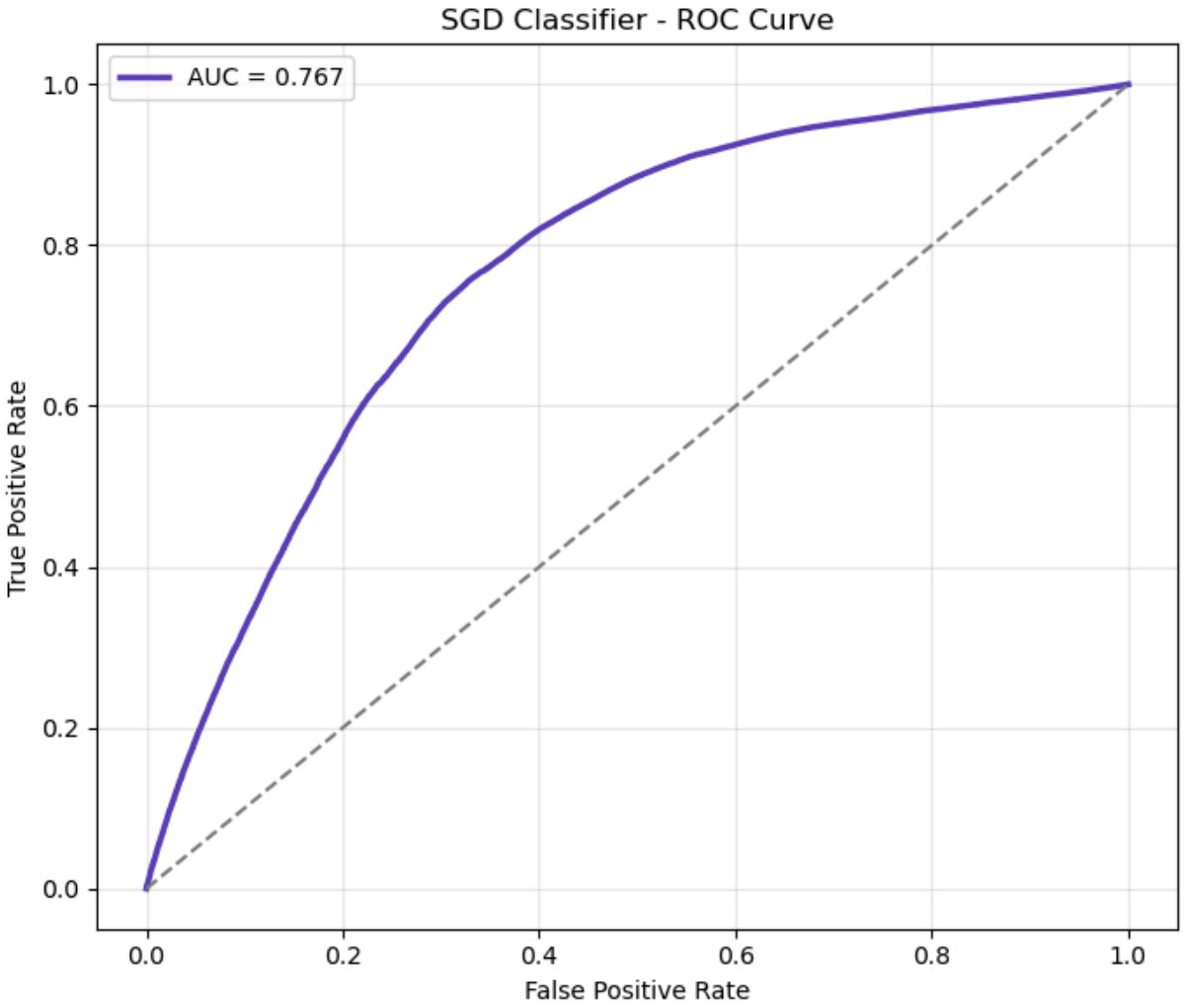


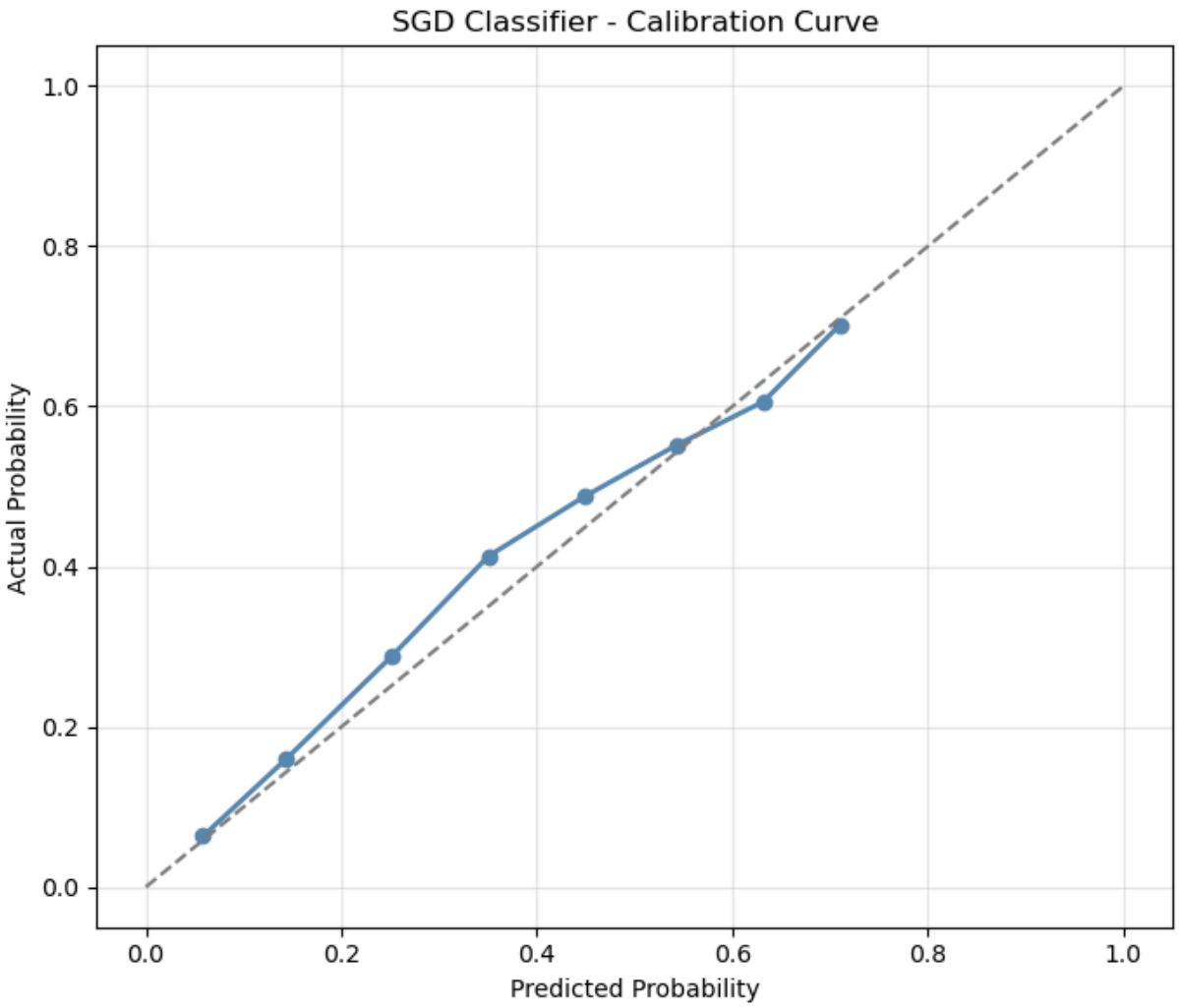
Out[178…
```
(array([1, 1, 0, ..., 0, 1, 1]),
 array([0.81327309, 0.76367856, 0.11888865, ..., 0.11796131, 0.80681424,
        0.65142359]))
```

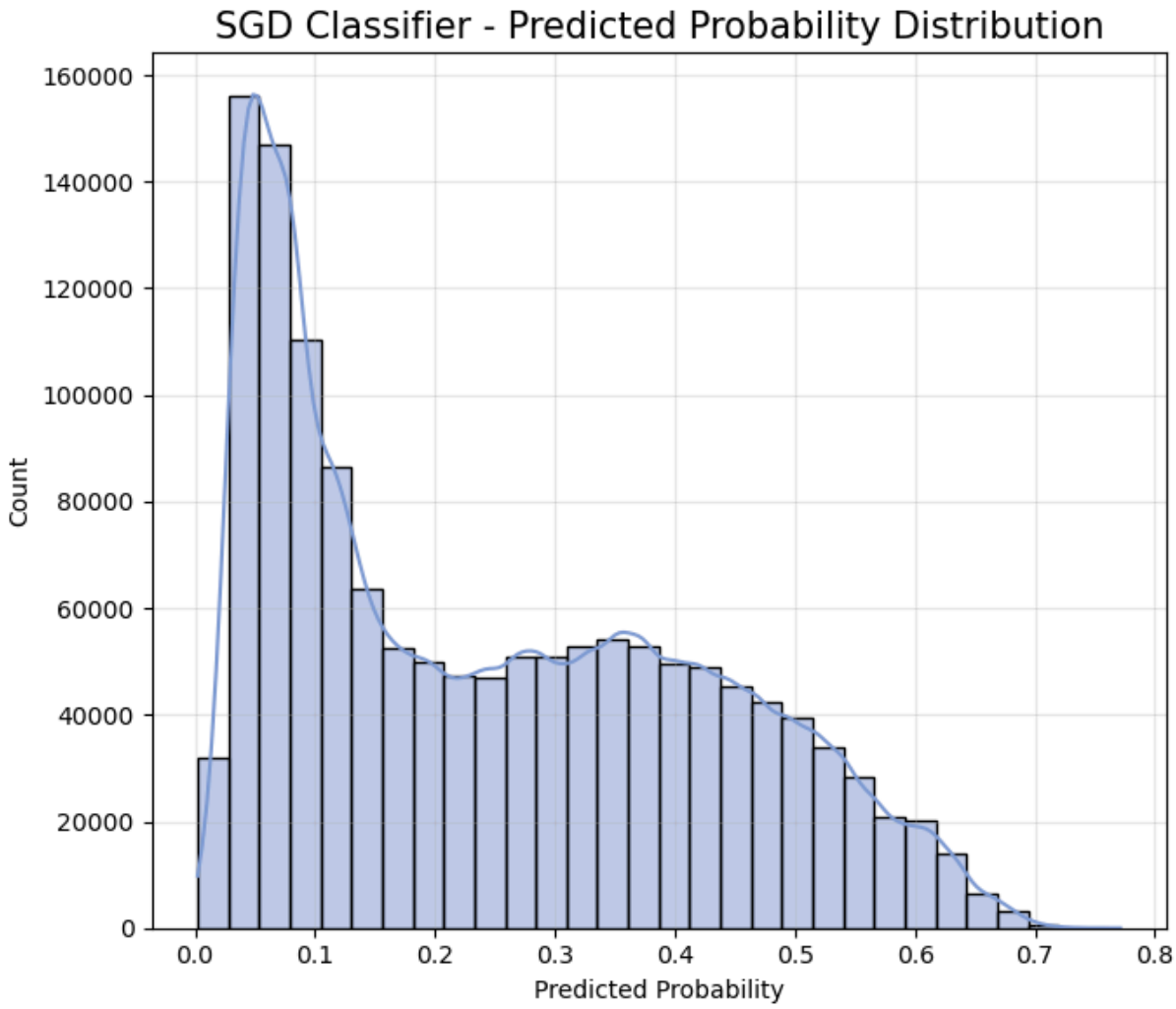
In [179…
```
evaluate_model(sgd, X_test, y_test, model_name="SGD Classifier")
```

```
SGD Classifier
Accuracy: 0.7450390120593066
Precision: 0.5660577925791368
Recall: 0.22234672304439745
F1 Score: 0.3192805031923876
AUC: 0.766953990310776
Brier Score: 0.16342946249934273

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.94      0.84   1028729
           1       0.57      0.22      0.32    378400

    accuracy                           0.75   1407129
   macro avg       0.67      0.58      0.58   1407129
weighted avg       0.71      0.75      0.70   1407129
```


SGD Classifier - Confusion Matrix
964230
64499
294264
84136
True Label
Predicted Label
0
1
0
1
900000
800000
700000
600000
500000
400000
300000
200000
100000


SGD Classifier - ROC Curve
AUC = 0.767
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

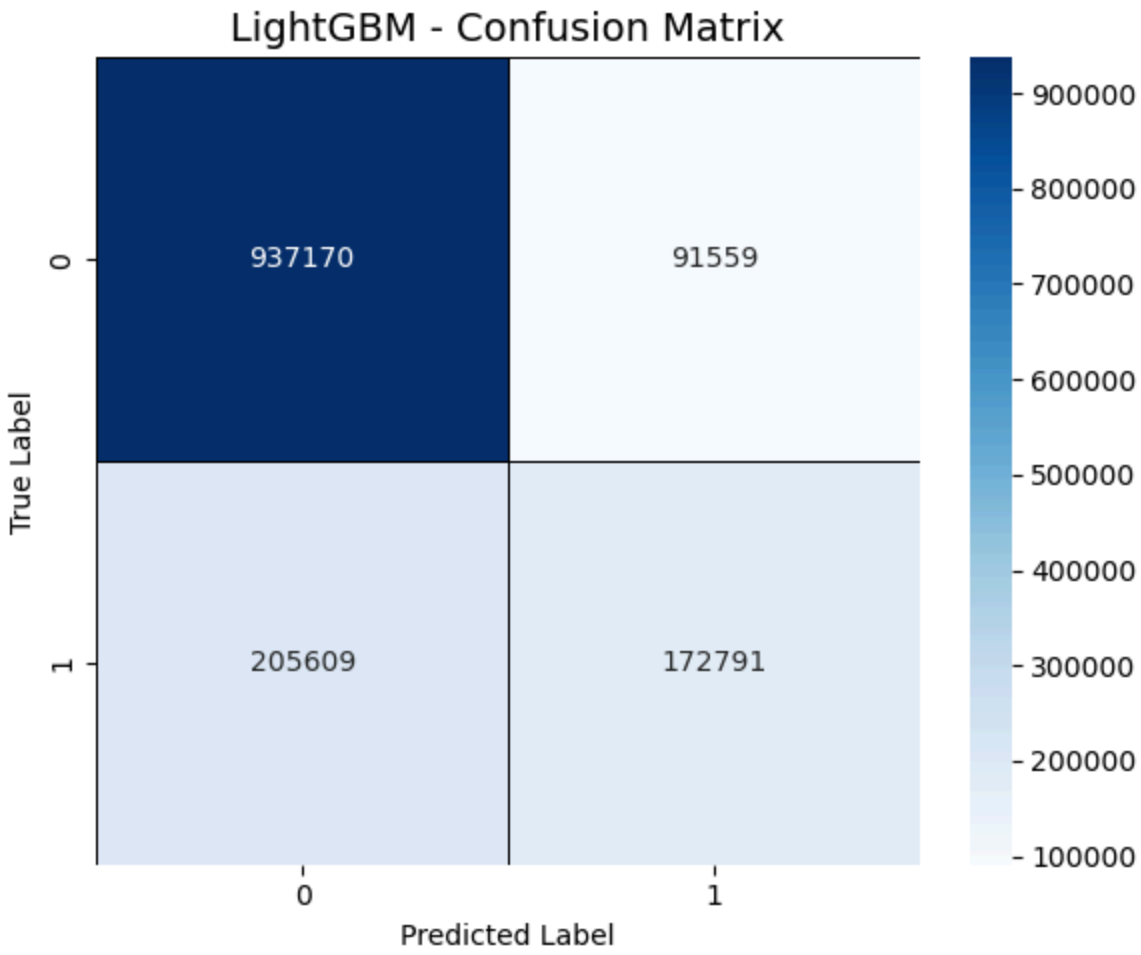

SGD Classifier - Calibration Curve
Actual Probability
Predicted Probability
0.0
0.2
0.4
0.6
0.8
1.0

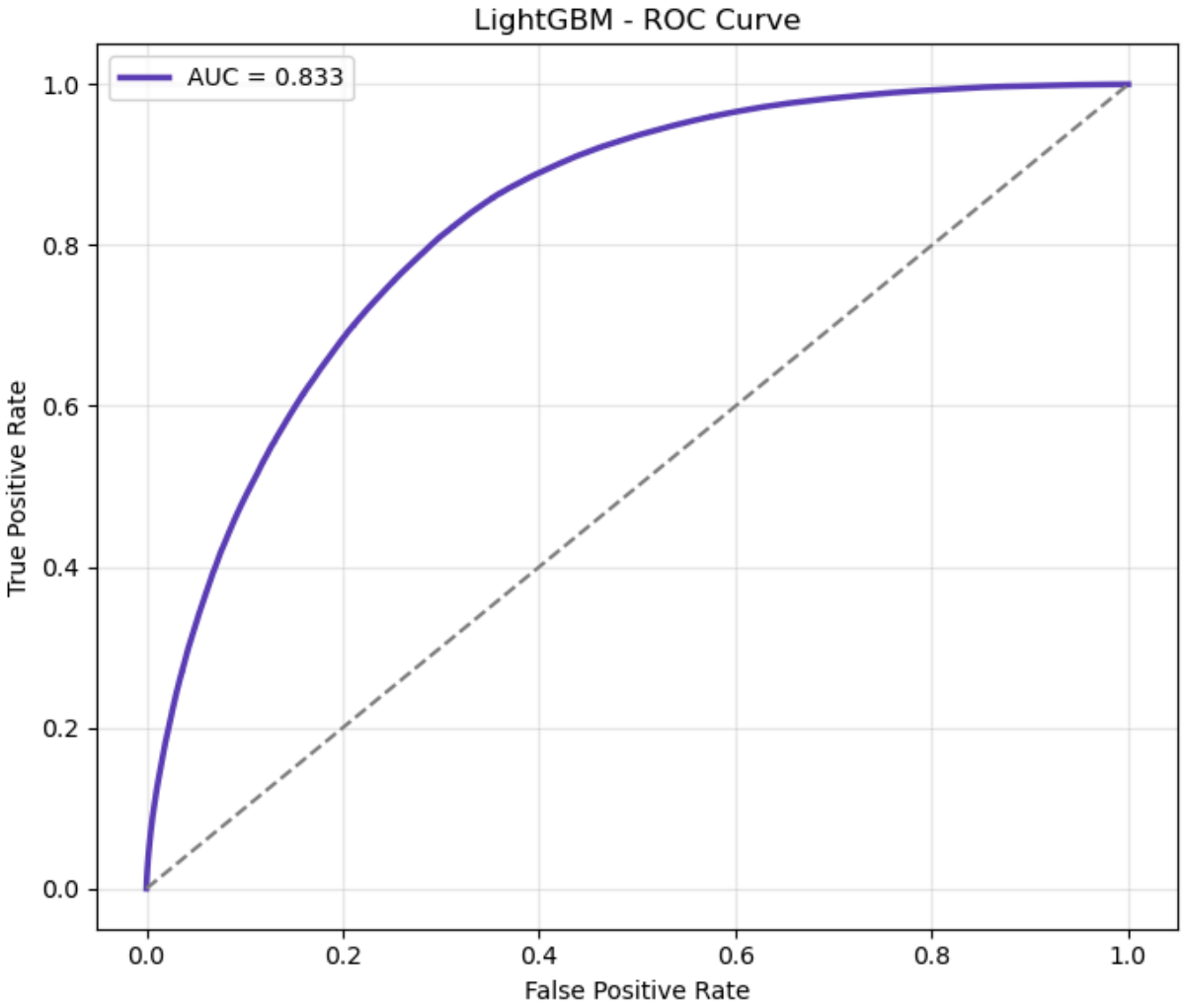


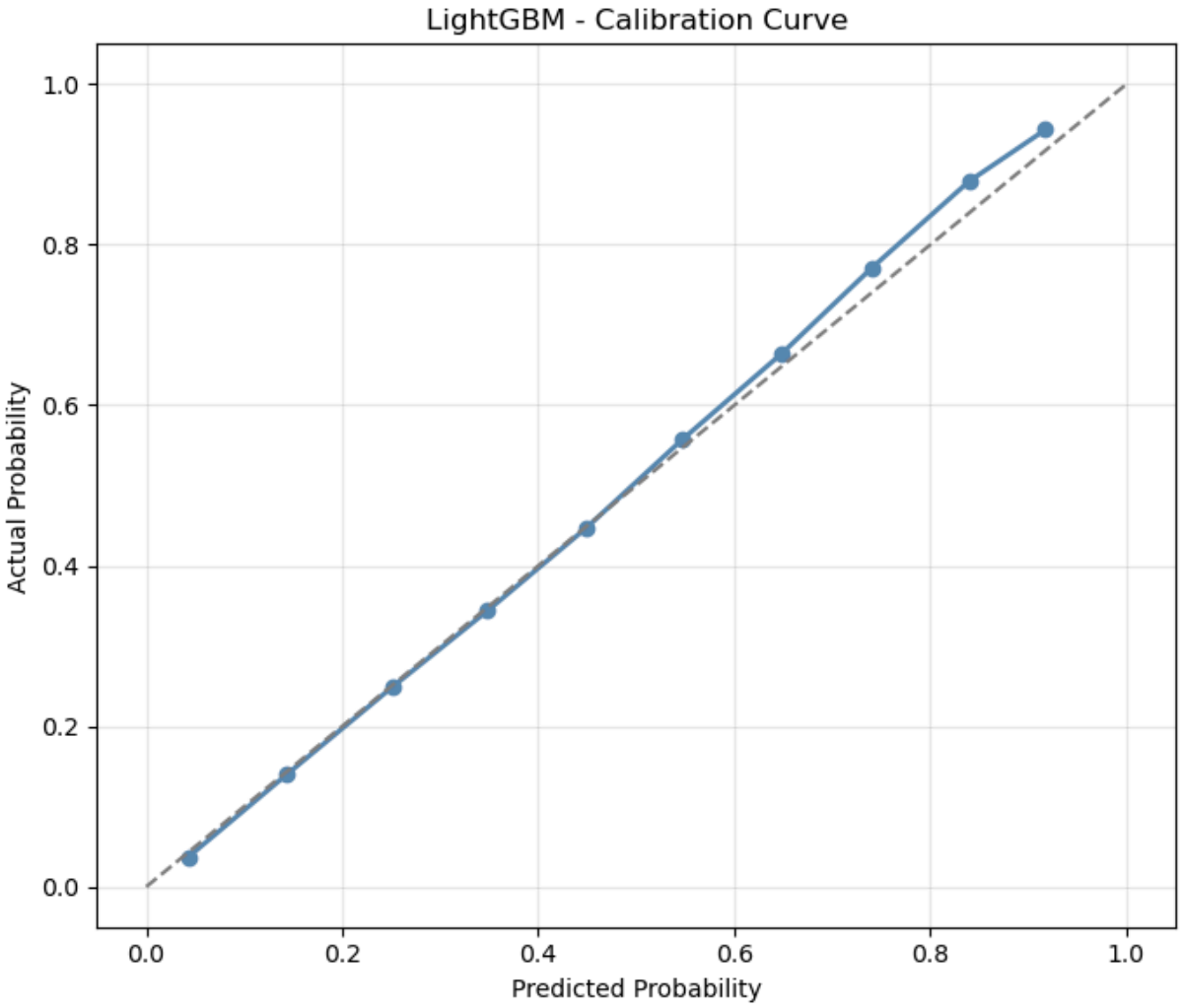
Out[179…
```
(array([0, 1, 0, ..., 0, 1, 0]),
 array([0.45090294, 0.5415523 , 0.03968768, ..., 0.04056335, 0.56261067,
        0.2250938 ]))
```

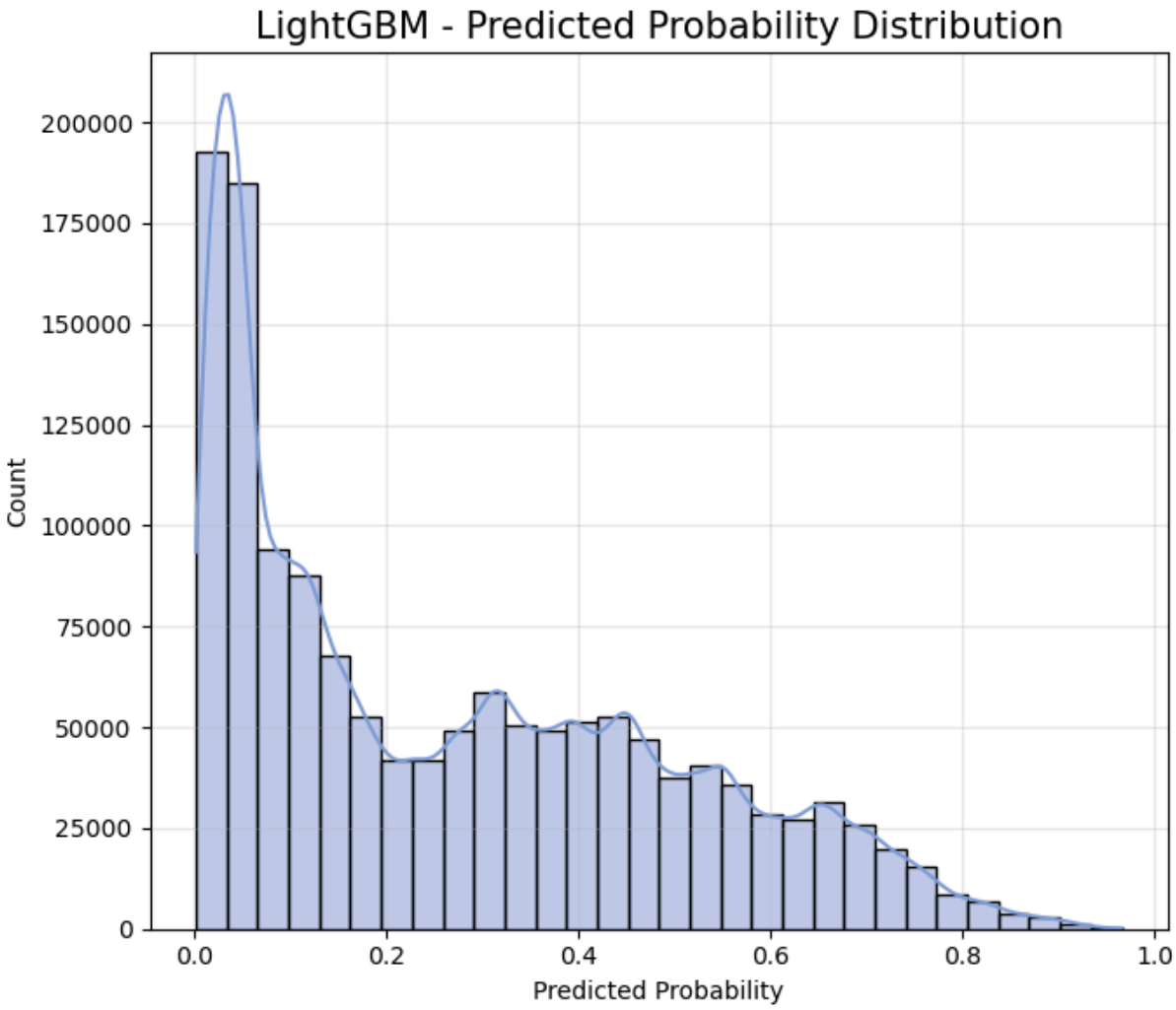
In [180…
```
evaluate_model(lgbm, X_test, y_test, model_name="LightGBM")
```

```
LightGBM
Accuracy: 0.7888125395752629
Precision: 0.6536447891053527
Recall: 0.45663583509513744
F1 Score: 0.5376616102683781
AUC: 0.8332074922151366
Brier Score: 0.14140178947799897

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.91      0.86   1028729
           1       0.65      0.46      0.54    378400

    accuracy                           0.79   1407129
   macro avg       0.74      0.68      0.70   1407129
weighted avg       0.78      0.79      0.78   1407129
```


LightGBM - Confusion Matrix
937170
91559
205609
172791
True Label
Predicted Label
0
1
0
1
900000
800000
700000
600000
500000
400000
300000
200000
100000


LightGBM - ROC Curve
AUC = 0.833
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

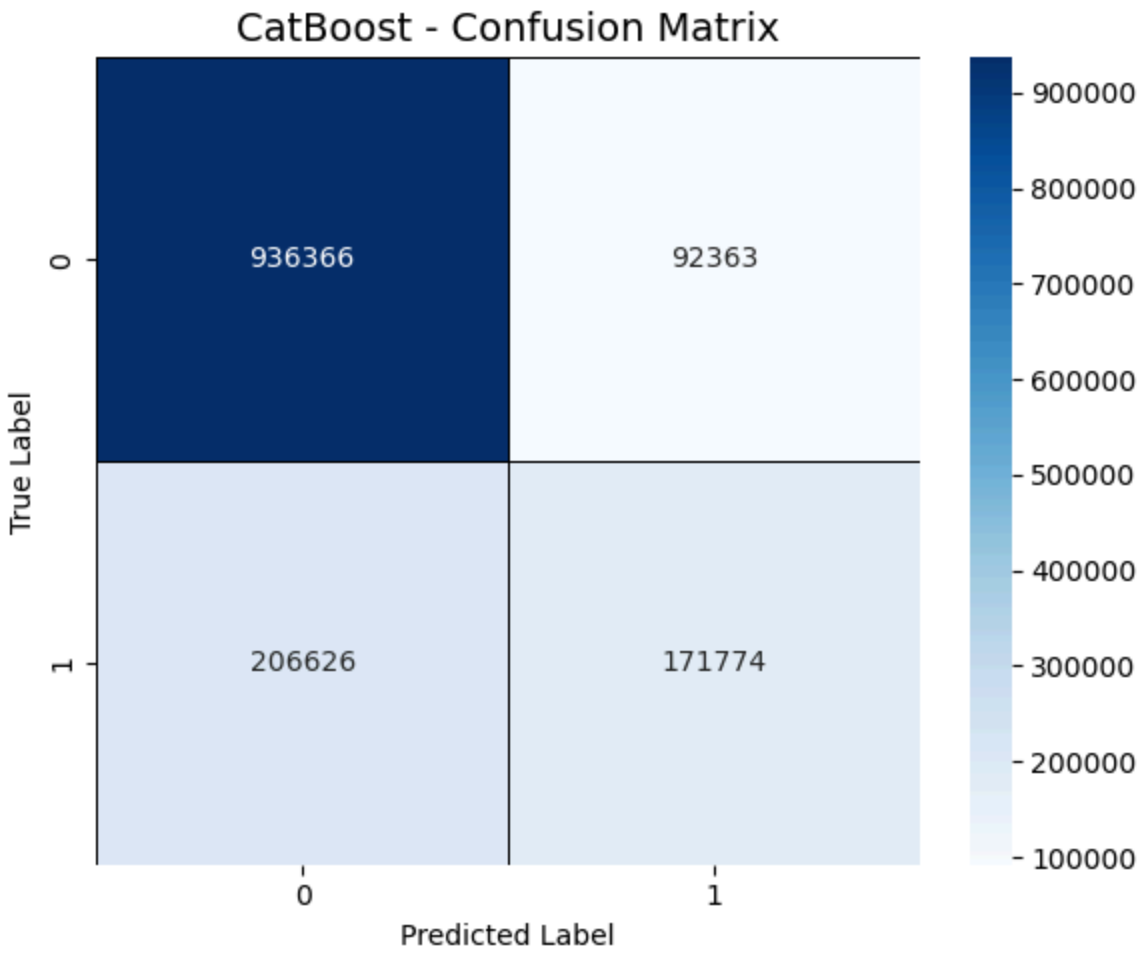

LightGBM - Calibration Curve
Actual Probability
Predicted Probability
0.0
0.2
0.4
0.6
0.8
1.0

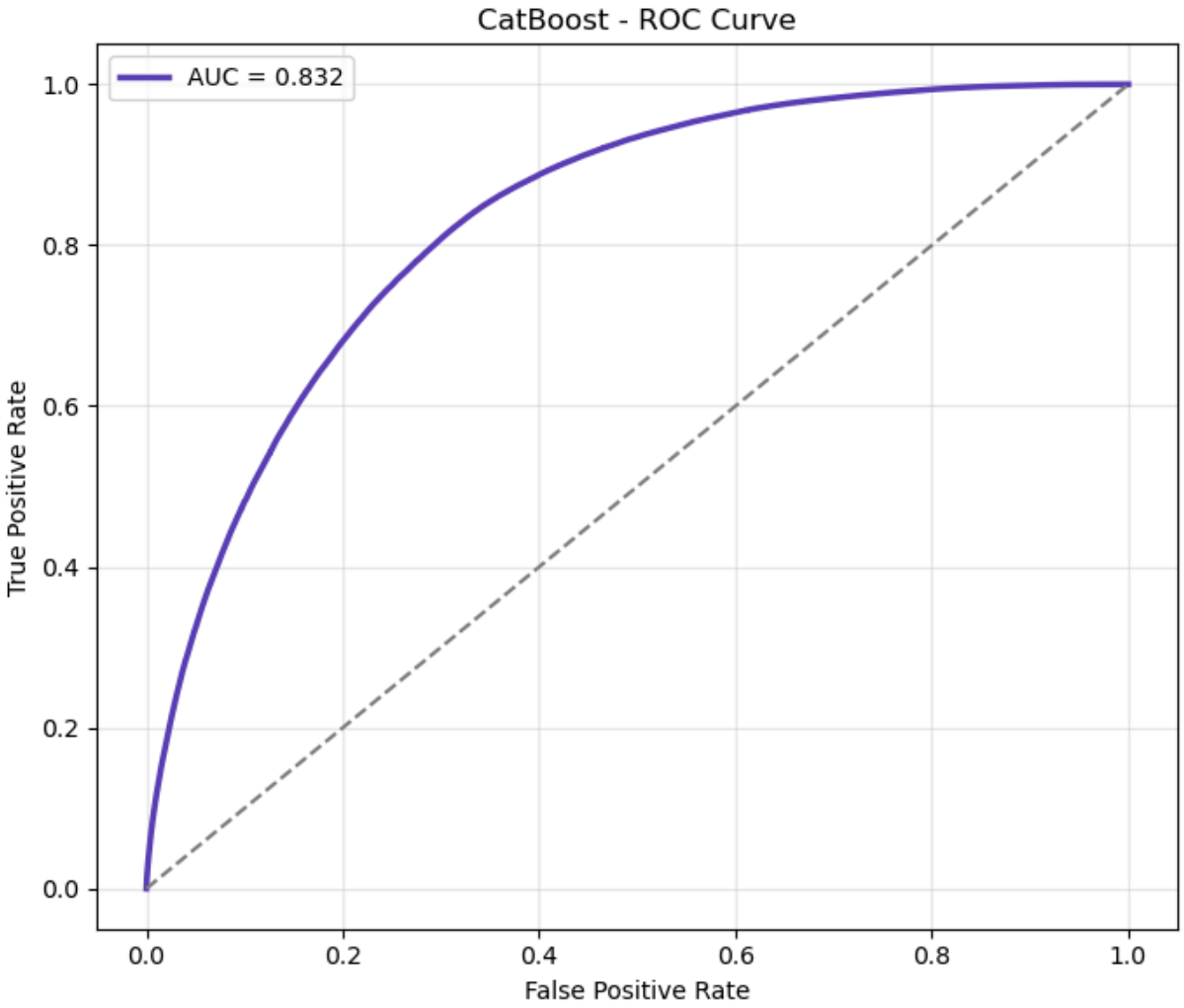


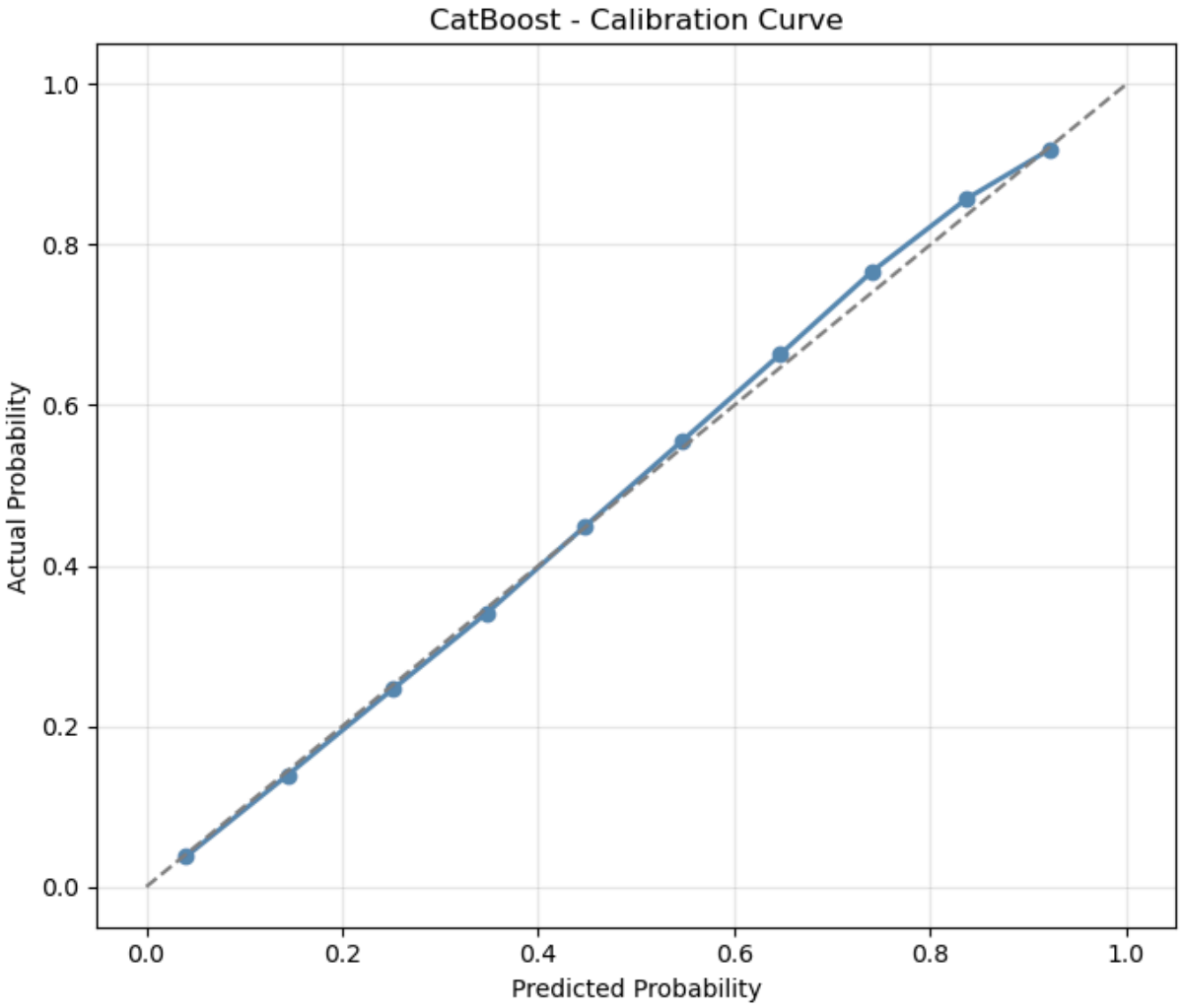
Out[180… (array([1, 1, 0, ..., 0, 1, 0]),
array([0.71526194, 0.53896452, 0.03347226, ..., 0.04177327, 0.61483318,
0.47491318]))

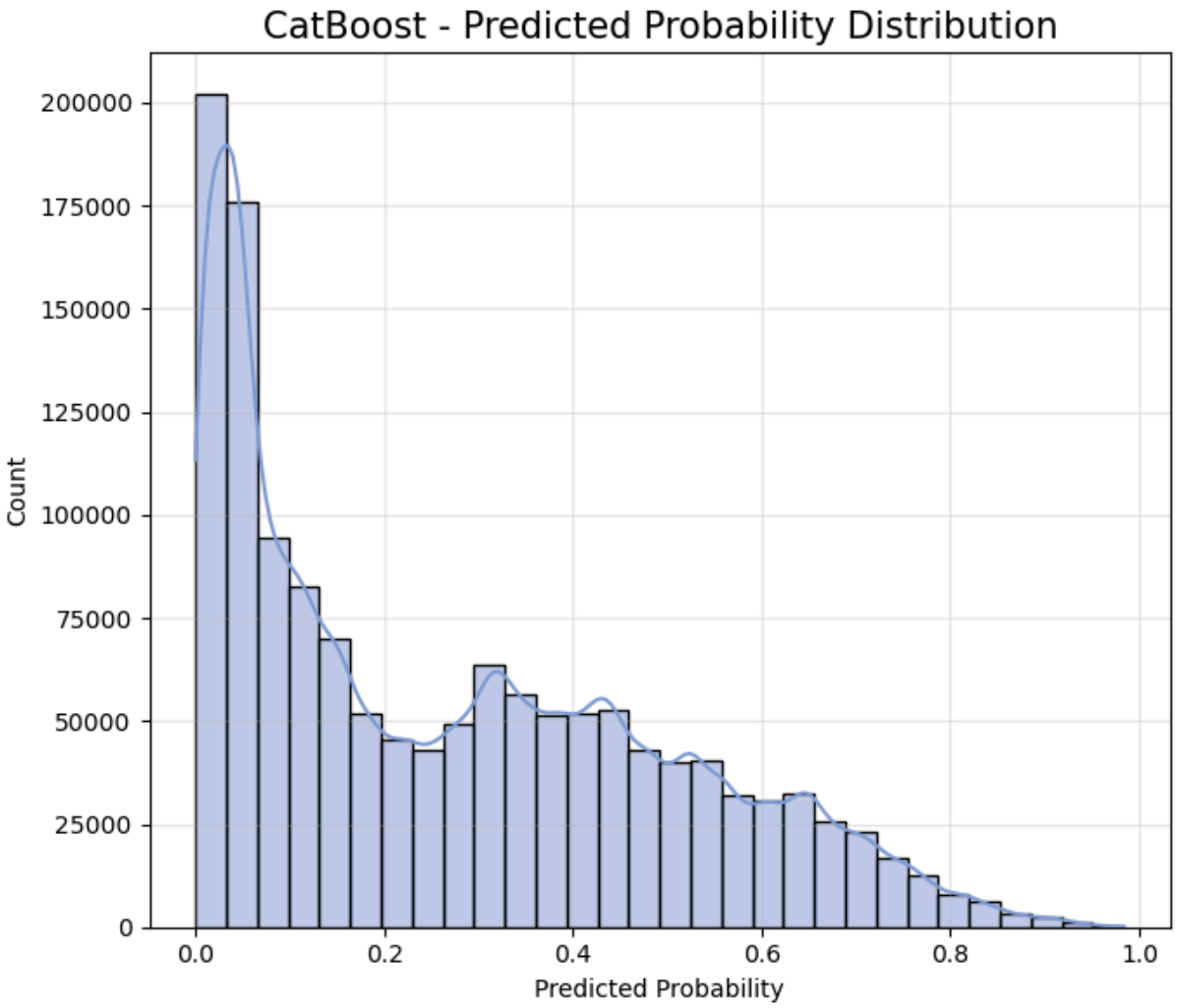
In [181…
```
evaluate_model(cat, X_test, y_test, model_name="CatBoost")
```

```
CatBoost
Accuracy: 0.7875184151559665
Precision: 0.6503216134051647
Recall: 0.45394820295983085
F1 Score: 0.5346742677853571
AUC: 0.8316905718904961
Brier Score: 0.14205196013504273

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.91      0.86   1028729
           1       0.65      0.45      0.53    378400

    accuracy                           0.79   1407129
   macro avg       0.73      0.68      0.70   1407129
weighted avg       0.77      0.79      0.77   1407129
```


CatBoost - Confusion Matrix
936366
92363
206626
171774
True Label
Predicted Label
0
1
0
1
900000
800000
700000
600000
500000
400000
300000
200000
100000


CatBoost - ROC Curve
AUC = 0.832
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


CatBoost - Calibration Curve
Actual Probability
Predicted Probability
0.0
0.2
0.4
0.6
0.8
1.0



Out[181… (array([1, 1, 0, ..., 0, 1, 0], dtype=int64),
array([0.72342368, 0.54602712, 0.03833977, ..., 0.03674687, 0.6319896 ,
0.46122815]))

Feature Importance for Tree-Based Models

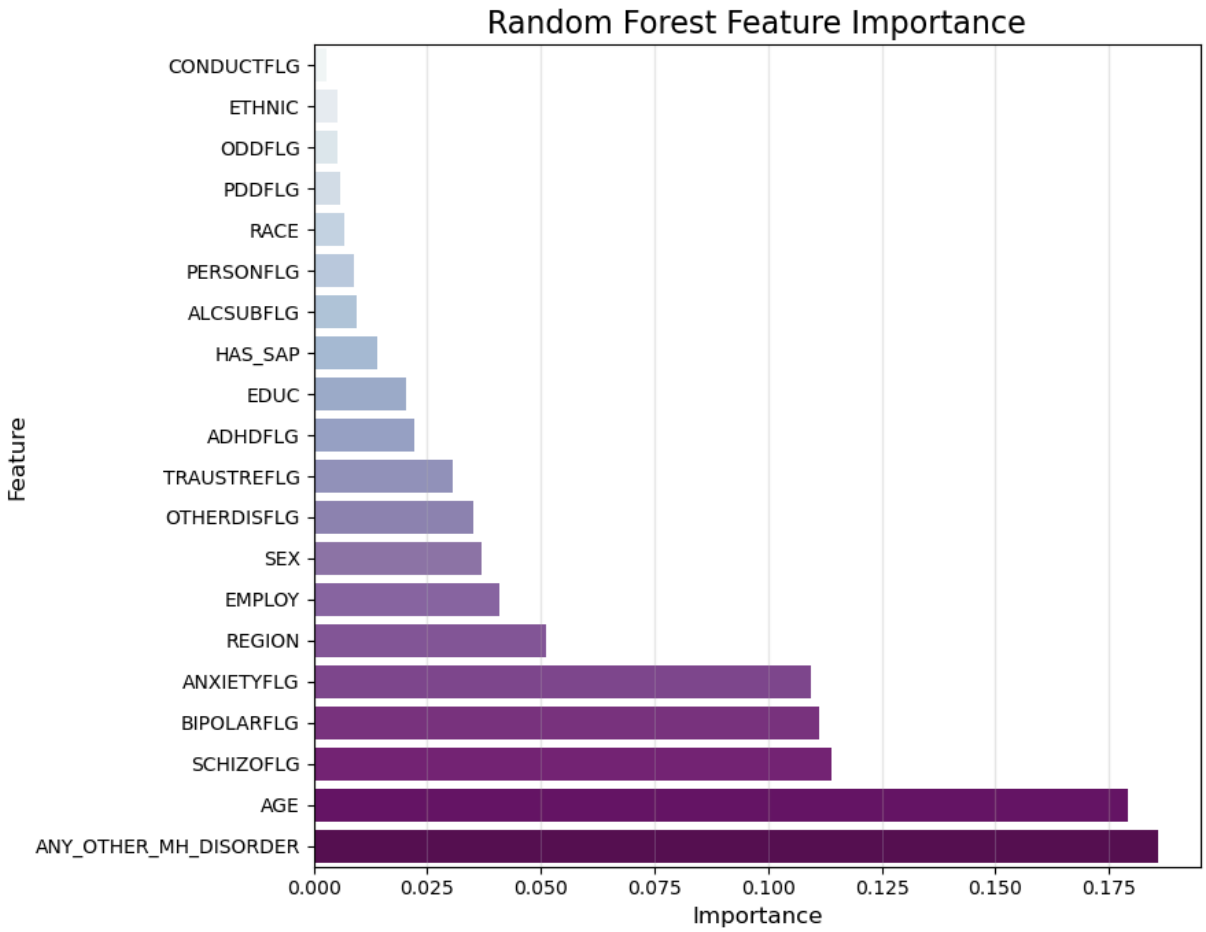
Random Forest

In [184…
```
importances = pd.Series( rf.feature_importances_, index=X_train.columns ).sort_valu
```

In [185…
```
top = importances.head(20).sort_values()

plt.figure(figsize=(9, 7))
sns.barplot(
x=top.values,
y=top.index,
        palette="BuPu"
    )
plt.title("Random Forest Feature Importance", fontsize=16)
plt.xlabel("Importance", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.grid(axis="x", alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```

```
C:\Users\janec\AppData\Local\Temp\ipykernel_8468\2417384146.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(
```



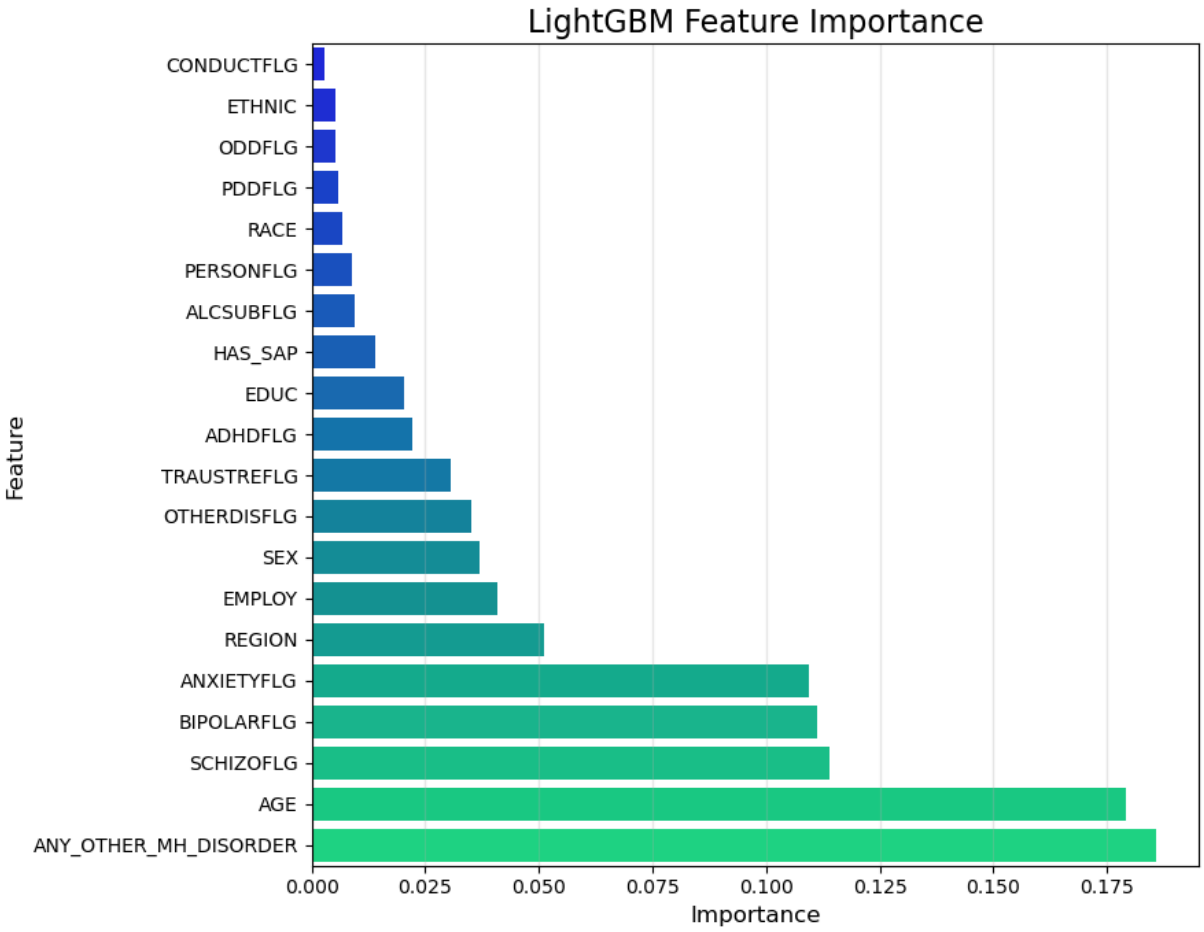
LightGBM

In [187…

```
lgbm_importances = pd.Series(
    lgbm.feature_importances_,
    index=X_train.columns
).sort_values(ascending=False)

top = importances.head(20).sort_values()

plt.figure(figsize=(9, 7))
sns.barplot(
        x=top.values,
        y=top.index,
        palette="winter"
    )

plt.title("LightGBM Feature Importance", fontsize=16)
```

```
plt.xlabel("Importance", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.grid(axis="x", alpha=0.3)
plt.tight_layout()
plt.show()
```

```
C:\Users\janec\AppData\Local\Temp\ipykernel_8468\4184752902.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(
```



CatBoost

In [189…

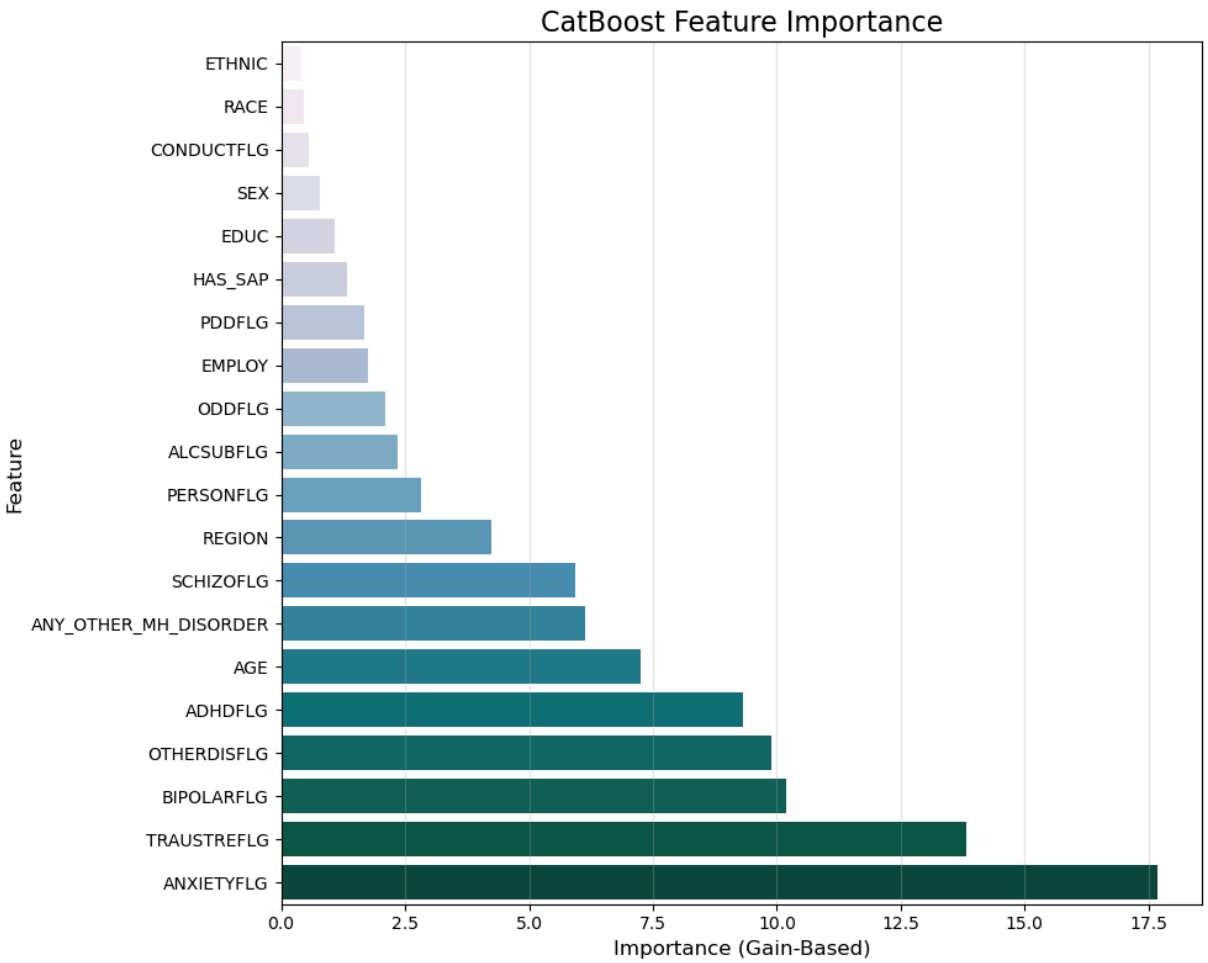
```
# Getting feature importance values from CatBoost model
cat_importances = cat.get_feature_importance(prettified=True)
print(cat_importances)
```

	Feature Id	Importances
0	ANXIETYFLG	17.688612
1	TRAUSTREFLG	13.814850
2	BIPOLARFLG	10.192684
3	OTHERDISFLG	9.880365
4	ADHDFLG	9.318621
5	AGE	7.262040
6	ANY_OTHER_MH_DISORDER	6.121978
7	SCHIZOFLG	5.921620
8	REGION	4.251074
9	PERSONFLG	2.811937
10	ALCSUBFLG	2.354448
11	ODDFLG	2.105787
12	EMPLOY	1.741571
13	PDDFLG	1.674110
14	HAS_SAP	1.324538
15	EDUC	1.083804
16	SEX	0.769795
17	CONDUCTFLG	0.556504
18	RACE	0.466628
19	ETHNIC	0.396699
20	IS_VETERAN	0.084814
21	IS_MARRIED	0.074170
22	DELIRDEMFLG	0.045371
23	IS_HOMELESS	0.037475
24	HAS_SUBSTANCE_USE	0.020504

In [190…
```
# Converting to pandas series
imp_series = pd.Series(
    cat_importances["Importances"].values,
    index=cat_importances["Feature Id"].values
).sort_values(ascending=True)
```

In [191…
```
# Plotting top 20 most important features
top = imp_series.tail(20)

plt.figure(figsize=(10, 8))
sns.barplot(
    x=top.values,
    y=top.index,
    palette="PuBuGn"
)

plt.title("CatBoost Feature Importance", fontsize=16)
plt.xlabel("Importance (Gain-Based)", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.grid(axis="x", alpha=0.3)
plt.tight_layout()
plt.show()
```

```
C:\Users\janec\AppData\Local\Temp\ipykernel_8468\1476501251.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(
```

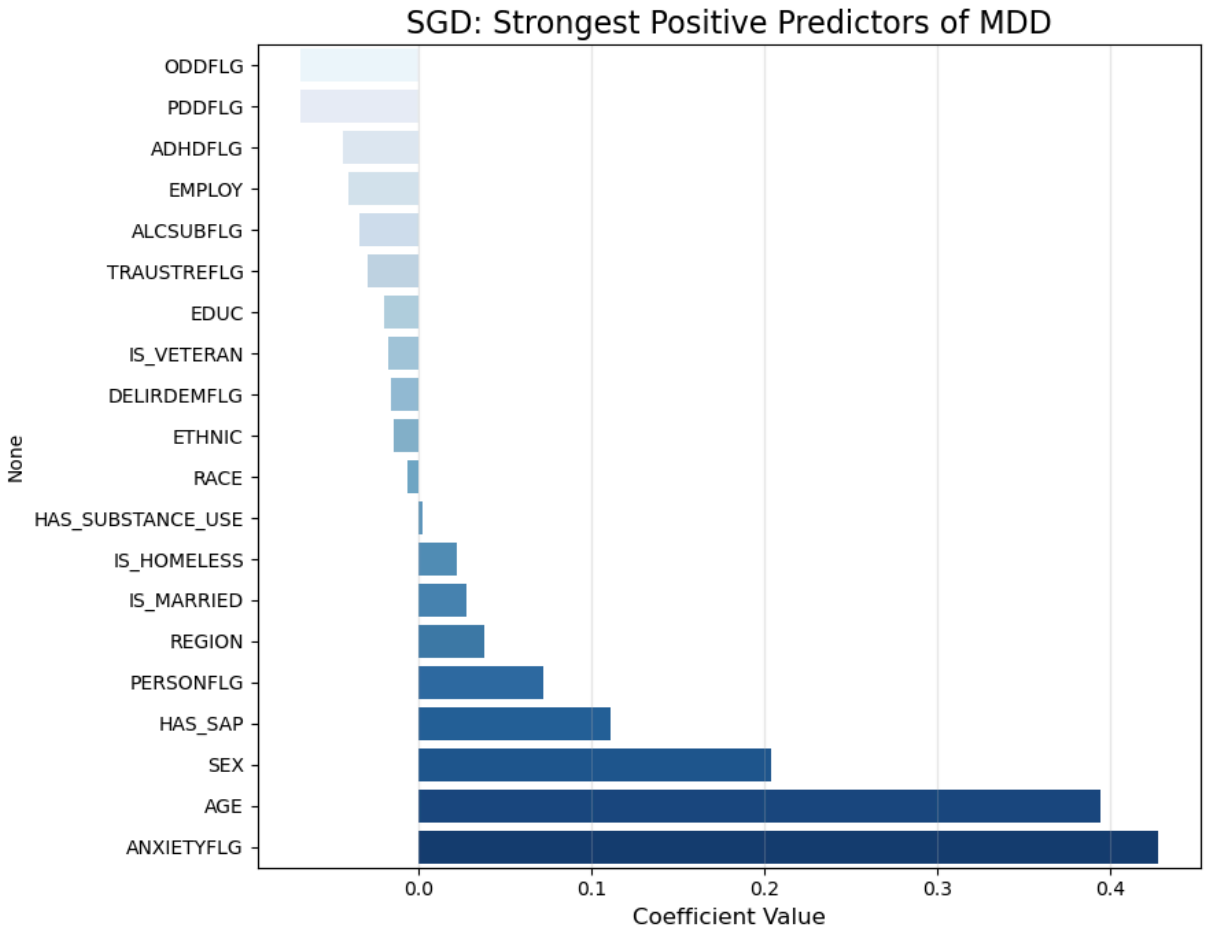


Feature Effects (SGD)

In [193…
```
# Getting feature effects values from sgd model
sgd.named_steps["sgd"].coef_
sgd = sgd.named_steps["sgd"]

# Converting to pandas series
coefs = pd.Series(
    sgd.coef_[0],
    index=X_train.columns
).sort_values()
```

In [194…
```
top_pos = coefs.tail(20).sort_values()

plt.figure(figsize=(9,7))
sns.barplot(x=top_pos.values, y=top_pos.index, palette="Blues")
plt.title("SGD: Strongest Positive Predictors of MDD", fontsize=16)
plt.xlabel("Coefficient Value", fontsize=12)
plt.grid(axis="x", alpha=0.3)
plt.tight_layout()
plt.show()
```

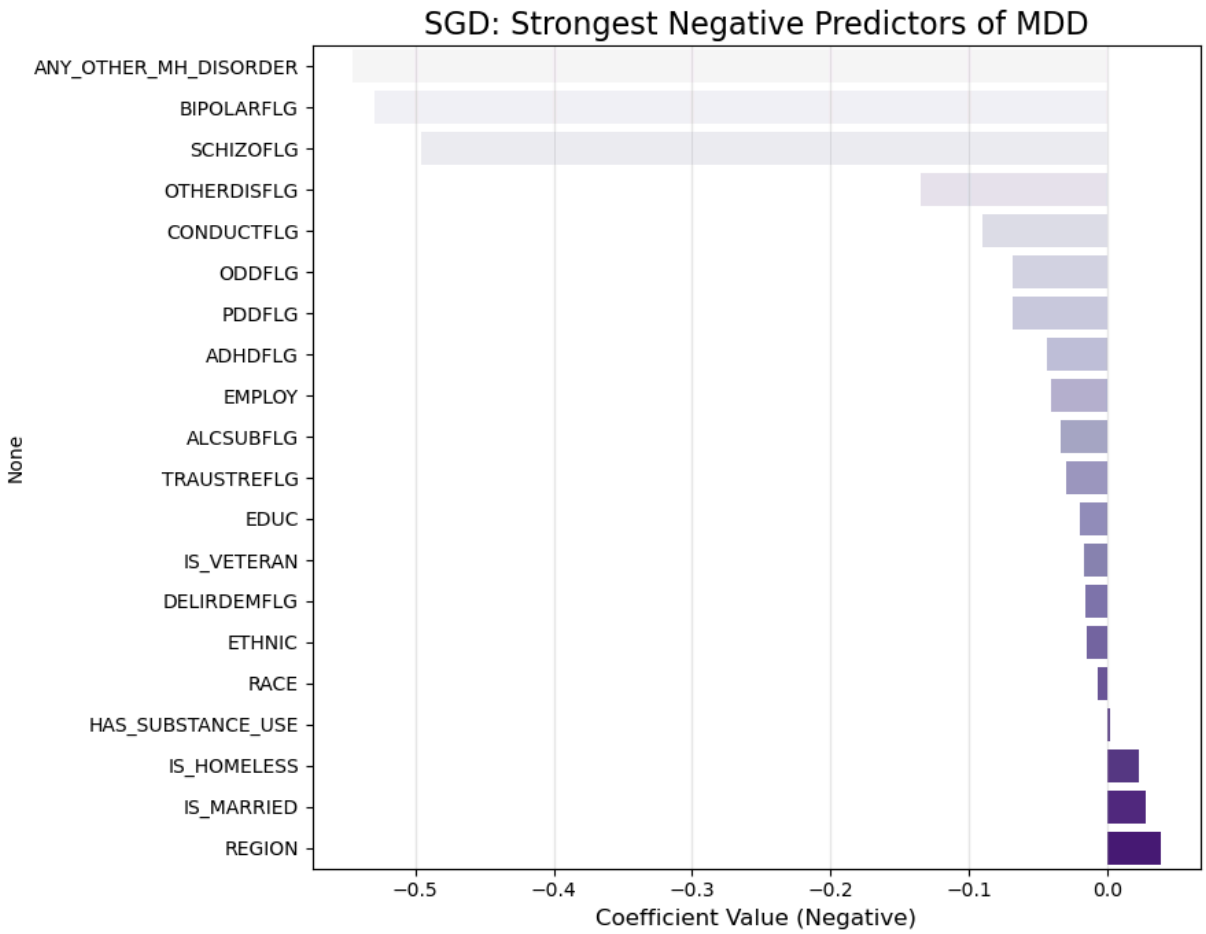
```
C:\Users\janec\AppData\Local\Temp\ipykernel_8468\203848772.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=top_pos.values, y=top_pos.index, palette="Blues")
```



In [195…

```
top_neg = coefs.head(20).sort_values()

plt.figure(figsize=(9,7))
sns.barplot(x=top_neg.values, y=top_neg.index, palette="Purples")
plt.title("SGD: Strongest Negative Predictors of MDD", fontsize=16)
plt.xlabel("Coefficient Value (Negative)", fontsize=12)
plt.grid(axis="x", alpha=0.3)
plt.tight_layout()
plt.show()
```

```
C:\Users\janec\AppData\Local\Temp\ipykernel_8468\1493180783.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=top_neg.values, y=top_neg.index, palette="Purples")
```


SGD: Strongest Negative Predictors of MDD
ANY_OTHER_MH_DISORDER
BIPOLARFLG
SCHIZOFLG
OTHERDISFLG
CONDUCTFLG
ODDFLG
PDDFLG
ADHDFLG
EMPLOY
ALCSUBFLG
TRAUSTREFLG
EDUC
IS_VETERAN
DELIRDEMFLG
ETHNIC
RACE
HAS_SUBSTANCE_USE
IS_HOMELESS
IS_MARRIED
REGION
None
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
Coefficient Value (Negative)

In [ ]:

In [ ]: