```
! pip install catboost==1.2.3 lightgbm==4.3.0 xgboost==2.0.3
```

```
Collecting catboost==1.2.3
  Downloading catboost-1.2.3-cp311-cp311-manylinux2014_x86_64.whl (98.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.5/98.5 MB 11.3 MB/s eta 0:00:00
Collecting lightgbm==4.3.0
  Downloading lightgbm-4.3.0-py3-none-manylinux_2_28_x86_64.whl (3.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 105.2 MB/s eta 0:00:00
Collecting xgboost==2.0.3
  Downloading xgboost-2.0.3-py3-none-manylinux2014_x86_64.whl (297.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 297.1/297.1 MB 5.0 MB/s eta 0:00:00
Collecting graphviz (from catboost==1.2.3)
  Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.1/47.1 kB 7.0 MB/s eta 0:00:00
Requirement already satisfied: matplotlib in /shared-libs/python3.11/py/lib/python3.11/site-packages (from catboost==1.2.3) (3.8.0)
Requirement already satisfied: numpy>=1.16.0 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from catboost==1.2.3) (1.26.1)
Requirement already satisfied: pandas>=0.24 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from catboost==1.2.3) (2.1.4)
Requirement already satisfied: scipy in /shared-libs/python3.11/py/lib/python3.11/site-packages (from catboost==1.2.3) (1.11.3)
Requirement already satisfied: plotly in /shared-libs/python3.11/py/lib/python3.11/site-packages (from catboost==1.2.3) (5.17.0)
Requirement already satisfied: six in /shared-libs/python3.11/py-core/lib/python3.11/site-packages (from catboost==1.2.3) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /shared-libs/python3.11/py-core/lib/python3.11/site-packages (from pandas>=0.24-
Requirement already satisfied: pytz>=2020.1 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from pandas>=0.24->catboost==1.2.
Requirement already satisfied: tzdata>=2022.1 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from pandas>=0.24->catboost==1.
Requirement already satisfied: contourpy>=1.0.1 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from matplotlib->catboost==1.
Requirement already satisfied: cycler>=0.10 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from matplotlib->catboost==1.2.3)
Requirement already satisfied: fonttools>=4.22.0 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from matplotlib->catboost==1
Requirement already satisfied: kiwisolver>=1.0.1 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from matplotlib->catboost==1
Requirement already satisfied: packaging>=20.0 in /shared-libs/python3.11/py-core/lib/python3.11/site-packages (from matplotlib->catboost
Requirement already satisfied: pillow>=6.2.0 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from matplotlib->catboost==1.2.3
Requirement already satisfied: pyparsing>=2.3.1 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from matplotlib->catboost==1.
Requirement already satisfied: tenacity>=6.2.0 in /shared-libs/python3.11/py/lib/python3.11/site-packages (from plotly->catboost==1.2.3)
Installing collected packages: graphviz, xgboost, lightgbm, catboost
```

```python
import numpy as np
import pandas as pd
import regex as re
from datetime import datetime
import psycopg2
import os
from sqlalchemy import create_engine

import plotly.express as px
import plotly.graph_objs as go
import plotly.figure_factory as ff

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, roc_auc_score, precision_recall_curve, roc_curve, auc, average_precisi
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
```

```python
# Replace these variables with your own PostgreSQL connection details
postgresql_user = 'postgres.mntgperewwogedgozlle'
postgresql_pwd = 'zygf1fiUioCqYtAW'
postgresql_host = 'aws-0-eu-central-1.pooler.supabase.com'
postgresql_port = '5432'
postgresql_db = 'postgres'

# SQLAlchemy engine for PostgreSQL
engine = create_engine(f'postgresql+psycopg2://{postgresql_user}:{postgresql_pwd}@{postgresql_host}:{postgresql_port}/{po
imonitor = pd.read_csv('/work/imonitor_1703.csv')
imonitor.head()
```

```
/tmp/ipykernel_38/2658825398.py:1: DtypeWarning: Columns (1,4,11,15,24,25,42,56,62,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,8
  imonitor = pd.read_csv('/work/imonitor_1703.csv')
```

	Survey ID int64	Created Date object	Facility name and ...	Facility ownership o..	Please specify obj...	County object	What is your mont...	H
0	2390063	04-Dec-23	BABA DOGO HEA...	GOK	nan	Nairobi	1977-09-03	M
1	2390062	04-Dec-23	BABA DOGO HEA...	GOK	nan	Nairobi	1972-08-12	F
2	2390061	04-Dec-23	BABA DOGO HEA...	GOK	nan	Nairobi	1984-08-31	F
3	2390060	04-Dec-23	BABA DOGO HEA...	GOK	nan	Nairobi	1977-05-07	F
4	2390059	04-Dec-23	BABA DOGO HEA...	GOK	nan	Nairobi	1987-06-13	M

5 rows, showing 10 per page   Page 1 of 1

```python
imonitor.shape
```

```
(46549, 85)
```

```python
# Find and drop columns that contain "Please specify" or "Please Specify"
cols_to_drop = [col for col in imonitor.columns if "Please specify" in col or "Please Specify" in col]

# Drop these columns from the DataFrame in a single operation
imonitor.drop(cols_to_drop, axis=1, inplace=True)
```

```python
imonitor.shape
```

```
(46549, 69)
```

```python
imonitor.columns = imonitor.columns.map(lambda x: x.strip())
```

```python
for c in imonitor.columns:
    print(c)
```

```
Survey ID
Created Date
Facility name and MFL Code if applicable
Facility ownership
County
What is your month; and year of birth
How do you consider yourself?
What is the highest level of education you completed?
What is your current marital status?
Which county do you currently live in?
What are your sources of income?
For how long have you been accessing services (based on the expected package of services) in this facility?
Are you aware of the package of services that you are entitled to?
According to you; which HIV related services are you likely to receive in this facility?
Is there a service that you needed that was not provided?
Facility name
For that service that was not provided; were you referred?
If referred; did you receive the service where you were referred to?
If Yes which Service/Test/Medicine
On a scale of 1 to 5; how satisfied are you with the package of services received in this facility? If 1 is VERY UNSATISFIED and 5 is VER
What did you like about the services you received?
What did you not like about the services you received?
In your opinion what would you like to be improved?
Do you face any challenges when accessing the services at the facility?
Common issues that can be added in the drop-down box
In your opinion what can be done to improve access to the services you seek at the facility?
Was confidentiality considered while you were being served?
Are there age-appropriate health services for specific groups?
Does the facility allow you to share your concerns with the administration?
Do you know your health-related rights as a client of this facility?
```

```python
columns_to_drop = [
    "Survey ID",
    "Facility name and MFL Code if applicable",
    "What is your month; and year of birth",
    "How do you consider yourself?",
    "What is the highest level of education you completed?",
    "What is your current marital status?",
    "Which county do you currently live in?",
    "What are your sources of income?",
    "Facility name",
    "What did you like about the services you received?",
    "What did you not like about the services you received?",
    "In your opinion what would you like to be improved?",
    "In your opinion what can be done to improve access to the services you seek at the facility?",
    "Facility name denied service",
    "Why",
    "Were reasons provided as to why these services were not available?",
    "Were reasons provided as to why these services were not available?.1",
    "What are the barriers to uptake of VMMC by males 25+years and above?",
    "What are some of the current site level practices that community members like and would love to maintain for KP/PP ?
    "What would you like this facility to change/do better?",
    "Throughout your visit what did you find interesting/pleasing about this facility that should be emulated by other fa
    "What do you think can be improved",
    "Anything else that you would like to mention?",
    "What are the top 1-3 things you like about this facility with regards to care and treatment?",
    "What are the top 1-3 things you don't like about this facility with regards to care and treatment?",
    "how long do you wait on average to get a service; which service was that?",
    "how long do you wait on average to get your lab test result?",
    "Specify the support group you belong to"
]

# Drop the columns
imonitor.drop(columns=columns_to_drop, axis=1, inplace=True)
```

```
column_name_mapping = {
    "Created Date": "Date",
    "Organization name coordinating the feedback from the clients": "OrgFeedbackCoordinator",
    "Facility ownership": "FacilityOwnership",
    "County": "FacilityCounty",
    "For how long have you been accessing services (based on the expected package of services) in this facility?": "Servi
    "Are you aware of the package of services that you are entitled to?": "ServicesAwareness",
    "According to you; which HIV related services are you likely to receive in this facility?": "ExpectedHIVServices",
    "Is there a service that you needed that was not provided?": "UnprovidedService",
    "Facility name no service": "UnprovidedServiceFacilityName",
    "For that service that was not provided; were you referred?": "ReferralForUnprovidedService",
    "If referred; did you receive the service where you were referred to?": "ReferralServiceReceived",
    "If Yes which Service/Test/Medicine": "ReceivedServiceDetail",
    "On a scale of 1 to 5; how satisfied are you with the package of services received in this facility? If 1 is VERY UNS
    "Do you face any challenges when accessing the services at the facility?": "AccessChallenges",
    "Common issues that can be added in the drop-down box": "CommonIssuesDropdown",
    "Was confidentiality considered while you were being served?": "Confidentiality",
    "Are there age-appropriate health services for specific groups?": "AgeAppropriateServices",
    "Does the facility allow you to share your concerns with the administration?": "ConcernsSharing",
    "Do you know your health-related rights as a client of this facility?": "RightsAwareness",
    "Have you ever been denied services at this facility?": "ServiceDenial",
    "Are you comfortable with getting services at this facility": "ComfortWithServices",
    "Have you ever been counseled?": "CounselingReceived",
    "Did you identify any gaps in the facility when you tried to access the services": "IdentifiedGaps",
    "Service type": "ServiceGapsType",
    "Are the HIV testing services readily available when required?": "HIVTestingAvailability",
    "Have you ever Interrupted your treatment?": "TreatmentInterruption",
    "Are the PMTCT services readily available when required?": "PMTCTServiceAvailability",
    "Are the HIV prevention; testing; treatment and care services adequate for KPs?": "KPServiceAdequacy",
    "Facility Level": "FacilityLevel",
    "Facility Operation times": "OperationTimes",
    "Facility Operation Days": "OperationDays",
    "What are your preferred days of visiting the facility": "PreferredVisitDays",
    "What are your preferred time of visiting the facility": "PreferredVisitTimes",
    "On a scale of 1-5; how clean do you find the facility?": "FacilityCleanliness",
    "How do you reach this facility?": "FacilityAccessMode",
    "How long does it take to reach this facility?": "FacilityAccessTime",
    "On a scale of 1-5; how accessible do you find this facility?": "FacilityAccessibility",
    "Do you consider the waiting time to be seen at this facility long?": "GeneralWaitingTime",
    "Do you consider the waiting time for lab test results long?": "LabResultsWaitingTime",
    "Does the facility offer support groups?": "SupportGroupAvailability",
    "In your opinion are the services offered at this facility youth friendly?": "YouthFriendlyServices",
    "What measures have been put in place to create GBV awareness and its harmful effects within the community?": "GBVAwa
    "PWD In your opinion are the services offered at this facility persons-with-disability friendly?": "PWDFriendlyServic
}

# Assuming imonitor is your DataFrame
df = imonitor.rename(columns=column_name_mapping)
```

```python
for c in df.columns:
    print(c)
```

```
Date
FacilityOwnership
FacilityCounty
ServiceAccessDuration
ServicesAwareness
ExpectedHIVServices
UnprovidedService
ReferralForUnprovidedService
ReferralServiceReceived
ReceivedServiceDetail
ServiceSatisfaction
AccessChallenges
CommonIssuesDropdown
Confidentiality
AgeAppropriateServices
ConcernsSharing
RightsAwareness
ServiceDenial
ComfortWithServices
CounselingReceived
IdentifiedGaps
ServiceGapsType
HIVTestingAvailability
TreatmentInterruption
PMTCTServiceAvailability
KPServiceAdequacy
FacilityLevel
OperationTimes
OperationDays
PreferredVisitDays
```

```python
columns_to_clean1 = [
    'GeneralWaitingTime',
    'LabResultsWaitingTime'
]

def replace_dont_know(df, column):
    df[column] = df[column].replace("Dont Know", "Do not know", regex=False)
    return df

for column in columns_to_clean1:
    df = replace_dont_know(df, column)
```

```python
columns_to_clean2 = [
    'FacilityCleanliness',
    'FacilityAccessibility'
    ]

def replace_mixed_with_text(df, column_name):
    def replace_value(value):
        satisfaction_map = {
            1: 'Very Unsatisfied',
            2: 'Unsatisfied',
            3: 'Okay',
            4: 'Satisfied',
            5: 'Very Satisfied'
        }
        if isinstance(value, str) and value[0].isdigit():
            num = int(value[0])
        elif isinstance(value, int):
            num = value
        else:
            return value

        return satisfaction_map.get(num, value)

    df[column_name] = df[column_name].apply(replace_value)
    return df

for column in columns_to_clean2:
    df = replace_mixed_with_text(df, column)
```

```python
def standardize_satisfaction(df, column_name):
    # Mapping for consolidating variations of satisfaction levels
    satisfaction_map = {
        '5': 'Very Satisfied',
        5.0: 'Very Satisfied',
        '4': 'Satisfied',
        4.0: 'Satisfied',
        '3': 'Okay',
        3.0: 'Okay',
        '2': 'Unsatisfied',
        2.0: 'Unsatisfied',
        '1': 'Very Unsatisfied',
        1.0: 'Very Unsatisfied',
        'Dissatisfied': 'Unsatisfied'
    }

    # Replace values based on the map
    df[column_name] = df[column_name].replace(satisfaction_map)
    return df

df = standardize_satisfaction(df, 'ServiceSatisfaction')
```

```python
print(df['FacilityLevel'].value_counts())
```

```
FacilityLevel
4.0    4802
3.0    4515
2.0    2889
5.0    2240
1.0     556
6.0      14
Name: count, dtype: int64
```

```python
def standardize_facility(df, column_name):
    # Mapping for consolidating variations of satisfaction levels
    satisfaction_map = {
        1.0: 'Community Health Unit',
        2.0: 'Dispensaries and Private Clinics',
        3.0: 'Health Centers',
        4.0: 'Sub-County Hospitals',
        5.0: 'County Referral Hospitals',
        6.0: 'National Referral Hospitals',
    }

    # Replace values based on the map
    df[column_name] = df[column_name].replace(satisfaction_map)
    return df

df = standardize_facility(df, 'FacilityLevel')
```

```python
def replace_symbols_and_words(df, column_name):
    df[column_name] = df[column_name].str.replace('<', 'Less than', regex=False)
    df[column_name] = df[column_name].str.replace('>', 'More than', regex=False)
    df[column_name] = df[column_name].str.replace('minutes', 'mins', regex=False)
    return df

df = replace_symbols_and_words(df, 'FacilityAccessTime')
```

```python
def replace_symbols_and_words(df, column_name):
    df[column_name] = df[column_name].str.replace('Less than 30mins', 'Less than 30 mins', regex=False)
    df[column_name] = df[column_name].str.replace('More than45 mins', 'More than 45 mins', regex=False)
    return df

df = replace_symbols_and_words(df, 'FacilityAccessTime')
```

```python
def convert_mixed_dates(date_column):
    """
    This function takes a Pandas Series of mixed dates and Excel serial dates and converts them to datetime objects.

    Parameters:
    date_column (pd.Series): A pandas Series with mixed date formats and serial dates.

    Returns:
    pd.Series: A pandas Series with all dates converted to datetime objects.
    """
    # Define the epoch start for Excel's serial date format
    excel_epoch = pd.Timestamp('1899-12-30')
    converted_dates = []

    for date in date_column:
        if isinstance(date, str) and re.match(r'^\d+(\.\d+)?$', date):
            # If it's a string that looks like a serial date, convert it
            serial_value = float(date)
            converted_date = excel_epoch + pd.to_timedelta(serial_value, unit='D')
        elif isinstance(date, (int, float)):
            # If it's a numeric type, assume it's a serial date
            converted_date = excel_epoch + pd.to_timedelta(date, unit='D')
        else:
            # Otherwise, try to parse it as a regular date
            converted_date = pd.to_datetime(date, errors='coerce')

        # Append the result, which will be NaT (Not a Time) if parsing failed
        converted_dates.append(converted_date)

    return pd.Series(converted_dates)

# Example usage, assuming 'df' is your DataFrame and 'Date' is the column to be converted:
df['Date'] = convert_mixed_dates(df['Date'])
```

```python
def standardize_gbv_awareness(df, column_name):
    df[column_name] = df[column_name].str.replace('Is there a desk to report GBV as community or individual', 'Presence o
    df[column_name] = df[column_name].str.replace('Are there training events on GBV for the community', 'Community traine
    return df

df = standardize_gbv_awareness(df, 'GBVAwarenessMeasures')
```

```python
def encode_multi_select(df, columns):
    # Iterate over the specified columns
    for col in columns:
        # Remove all whitespaces within each value and split based on ';'
        # This creates a Series of lists
        split_series = df[col].str.replace(' ', '').str.split(';')

        # Use the str.get_dummies() method on the Series of lists to perform one-hot encoding
        # This approach handles the separation and encoding in one step
        encoded = split_series.str.join('|').str.get_dummies()

        # Prefix the encoded column names to indicate their origin
        encoded.columns = [f"{col}_{option}" for option in encoded.columns]

        # Join the encoded dataframe with the original dataframe
        df = df.join(encoded)

        # Optionally, drop the original column if no longer needed
        # df.drop(col, axis=1, inplace=True)

    return df

# Specify the columns to encode
columns_to_encode = ['ExpectedHIVServices', 'OperationTimes', 'OperationDays', 'PreferredVisitDays', 'PreferredVisitTimes

# Apply the function
df2 = encode_multi_select(df, columns_to_encode)
```

```python
df2.drop(columns=columns_to_encode, axis=1, inplace=True)
```

```python
df2.to_sql('cleaned_all_columns', con=engine, if_exists='replace', index=True)
```

```
549
```

```python
df3 = df2.sample(5).copy()

df3.to_csv('/work/sample.csv')
```

```python
missing_percentage = df2.isnull().mean() * 100

threshold = 60

columns_to_drop = missing_percentage[missing_percentage > threshold].index.tolist()

print("Columns to drop:", columns_to_drop)

print("Number of columns to drop:", len(columns_to_drop))

df2.drop(columns=columns_to_drop, axis=1, inplace=True)

print("DataFrame shape after dropping columns:", df2.shape)
```

```
Columns to drop: ['ReferralForUnprovidedService', 'ReferralServiceReceived', 'ReceivedServiceDetail', 'CommonIssuesDropdown', 'ServiceGapsTyp
Number of columns to drop: 19
DataFrame shape after dropping columns: (46549, 66)
```

```python
threshold_percentage = 100

threshold = len(df2.columns) * (threshold_percentage / 100)

data = df2.dropna(thresh=threshold).copy()

print("Original DataFrame shape:", df2.shape)
print("Cleaned DataFrame shape:", data.shape)

rows_dropped = df2.shape[0] - data.shape[0]
print("Rows dropped:", rows_dropped)
```

```
Original DataFrame shape: (46549, 66)
Cleaned DataFrame shape: (39862, 66)
Rows dropped: 6687
```

```python
# df2.to_csv('data/cleanednonull.csv', index=False)
```

```python
# General descriptive statistics
data.describe()
```

	Date object	ExpectedHIVServi...	ExpectedHIVServi...	ExpectedHIVServi...	ExpectedHIVServi...	ExpectedHIVServi...	ExpectedHIVServi...	E
cou...	39862	39862	39862	39862	39862	39862	39862	
me...	2023-05-04 00:4...	0.7472028498	0.08516883247	0.05646982088	0.01763584366	0.02518689479	0.04766444233	
min	2022-07-26 00:0...	0	0	0	0	0	0	
25%	2022-10-31 00:0...	0	0	0	0	0	0	
50%	2023-03-31 07:11...	1	0	0	0	0	0	
75%	2023-11-14 00:0...	1	0	0	0	0	0	
max	2024-02-28 00:0...	1	1	1	1	1	1	
std	nan	0.4346210876	0.2791362698	0.2308296274	0.1316254356	0.1566943875	0.2130579312	

8 rows, showing 10 per page Page 1 of 1

```python
data['ServiceSatisfaction'].value_counts()
```

```
ServiceSatisfaction
Satisfied               22585
Very Satisfied          15966
Unsatisfied               584
Okay                      467
Do not know               127
Very Unsatisfied           74
Prefer not to answer       59
Name: count, dtype: int64
```

```python
recategorization_mapping = {
    'Very Satisfied': 2,
    'Satisfied': 1,
    'Okay': 1,
    'Unsatisfied': 0,
    'Very Unsatisfied': 0,
    #'Unknown': 0,
    'Do not know': 99,
    'Prefer not to answer ': 99
}

data.loc[:, 'ServiceSatisfaction'] = data['ServiceSatisfaction'].replace(recategorization_mapping)

# After replacement, you might want to ensure the data type is what you expect
# For example, if you want to ensure it's an integer (especially if NaN values are not expected)
data['ServiceSatisfaction'] = data['ServiceSatisfaction'].astype(int)

# Verify the changes
print(data['ServiceSatisfaction'].value_counts())
```

```
ServiceSatisfaction
1     23052
2     15966
0       658
99      186
Name: count, dtype: int64
```

```python
model_data = data[data.ServiceSatisfaction != 99]
```

```python
model_data = model_data.drop(['Date', 'FacilityCounty', 'FacilityOwnership'], axis=1)
```

```python
# Assuming subset_df is your DataFrame and 'ServiceSatisfaction' is the column of interest

# Split the dataset into separate groups based on 'ServiceSatisfaction'
class_3_df = model_data[model_data['ServiceSatisfaction'] == 2]
class_2_df = model_data[model_data['ServiceSatisfaction'] == 1]
class_1_df = model_data[model_data['ServiceSatisfaction'] == 0]

# Get the target number of instances to match, which is the number of instances in class 1
target_number = class_1_df.shape[0]

# Randomly sample from classes 3 and 2 to match the number of instances in class 1
class_3_sampled_df = class_3_df.sample(n=target_number, random_state=42)
class_2_sampled_df = class_2_df.sample(n=target_number, random_state=42)

balanced_df = pd.concat([class_3_sampled_df, class_2_sampled_df, class_1_df])
```

```python
balanced_df['ServiceSatisfaction'].value_counts()
```

```
ServiceSatisfaction
2    658
1    658
0    658
Name: count, dtype: int64
```

```python
ordinal_vars = balanced_df['ServiceSatisfaction']
nominal_vars = [col for col in balanced_df.columns if balanced_df[col].dtype == 'object' and col not in ordinal_vars]
encoded_data = pd.get_dummies(balanced_df, columns=nominal_vars)

# This automatically drops the original nominal columns and adds the one-hot encoded columns
print("NaN counts after pandas get_dummies:", encoded_data.isnull().sum().sum())
```

```
NaN counts after pandas get_dummies: 0
```

```python
X = encoded_data.drop('ServiceSatisfaction', axis=1)
y = encoded_data['ServiceSatisfaction']
# Split the data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

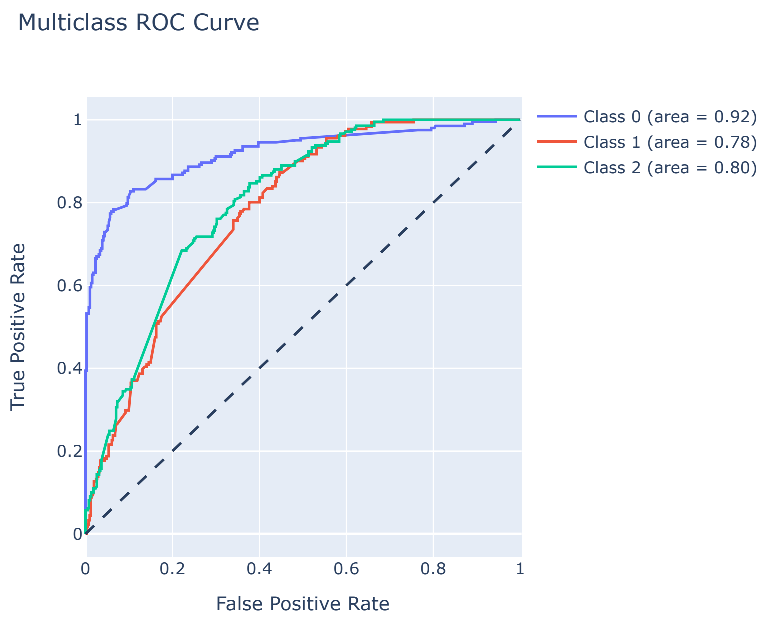
```python
def test_models(X_train, y_train, X_test, y_test):
    models = {
        'CatBoostClassifier': CatBoostClassifier(verbose=0),
        'LGBMClassifier': LGBMClassifier(),
        'XGBClassifier': XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'),
        'RandomForestClassifier': RandomForestClassifier(),
        'LogisticRegression': make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000, multi_class='ovr')),
        'SVC': make_pipeline(StandardScaler(), SVC(probability=True, decision_function_shape='ovr'))
    }

    best_model = None
    best_score = -1
    model_results = []
    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr', average='weighted') if hasattr(mo
        report = classification_report(y_test, y_pred, output_dict=True)

        model_result = {
            'Model': name,
            'ROC AUC': roc_auc,
            'Accuracy': report['accuracy'],
            'Precision': report['weighted avg']['precision'],
            'Recall': report['weighted avg']['recall'],
            'F1 Score': report['weighted avg']['f1-score'],
        }
        model_results.append(model_result)

        # Check if this model is the best based on ROC AUC
        if roc_auc is not None and roc_auc > best_score:
            best_score = roc_auc
            best_model = model

    return pd.DataFrame(model_results), best_model

# To use this function, you need to replace `X_train2`, `y_train2`, `X_test2`, and `y_test2` with your actual dataset.
# Here's how you might call this function:

results_df, best_model = test_models(X_train, y_train, X_test, y_test)
print(results_df)
print("Best model:", best_model)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001011 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 134
[LightGBM] [Info] Number of data points in the train set: 1381, number of used features: 67
[LightGBM] [Info] Start training from score -1.110266
[LightGBM] [Info] Start training from score -1.063047
[LightGBM] [Info] Start training from score -1.123540
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
                    Model   ROC AUC  Accuracy  Precision    Recall  F1 Score
0      CatBoostClassifier  0.835278  0.661046   0.674280  0.661046  0.662522
1          LGBMClassifier  0.829121  0.672850   0.678185  0.672850  0.671965
2           XGBClassifier  0.823444  0.661046   0.670870  0.661046  0.660229
3  RandomForestClassifier  0.831505  0.669477   0.683241  0.669477  0.671800
4      LogisticRegression  0.817806  0.644182   0.649968  0.644182  0.644362
5                     SVC  0.825940  0.647555   0.665978  0.647555  0.642034
Best model: <catboost.core.CatBoostClassifier object at 0x7eff73010490>
```

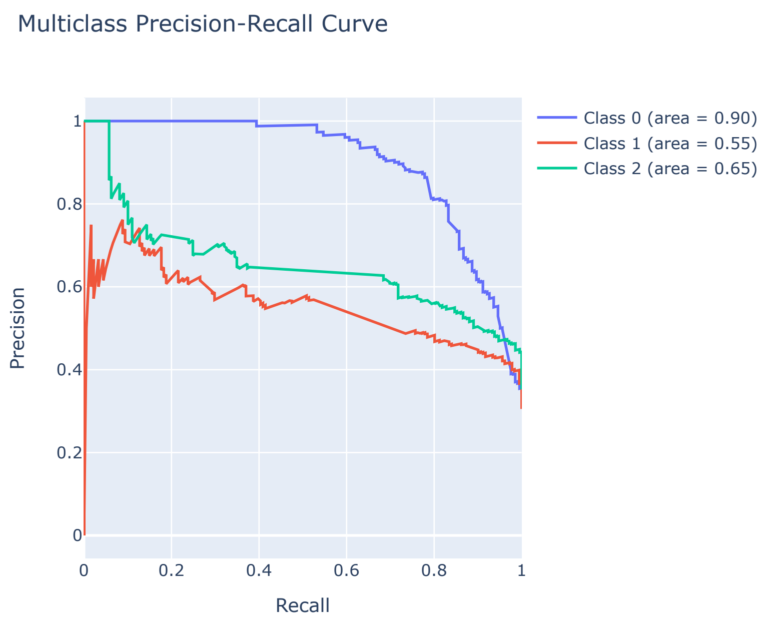
```python
# Predict the values
y_pred = best_model.predict(X_test)

# Assuming y_test2 contains your test labels
unique_classes = np.unique(y_test)

# Now you can use unique_classes for the classification_report
print(classification_report(y_test, y_pred, target_names=[str(cls) for cls in unique_classes]))
```

```
              precision    recall  f1-score   support

           0       0.87      0.75      0.81       203
           1       0.56      0.49      0.52       181
           2       0.58      0.72      0.64       209

    accuracy                           0.66       593
   macro avg       0.67      0.65      0.66       593
weighted avg       0.67      0.66      0.66       593
```

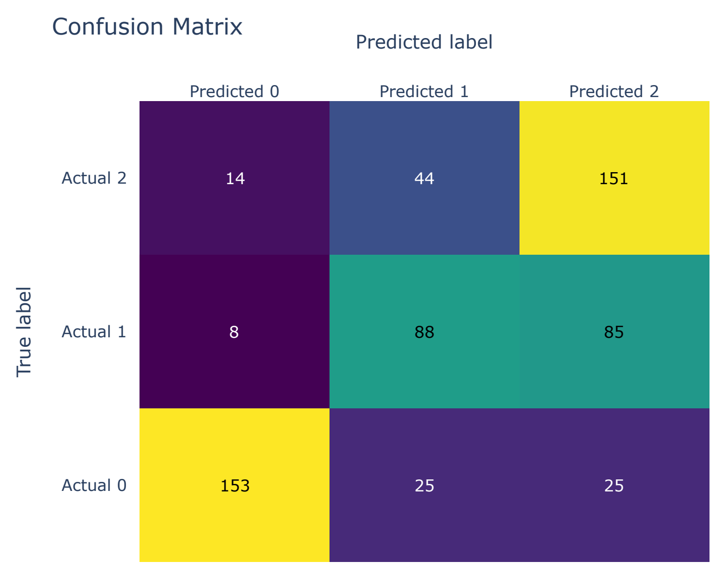
```python
# Assuming y_test2 is not already binarized
n_classes = len(np.unique(y_test))
y_test_bin = label_binarize(y_test, classes=np.arange(n_classes))
y_score = best_model.predict_proba(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve for each class
fig = go.Figure()
for i in range(n_classes):
    fig.add_trace(go.Scatter(x=fpr[i], y=tpr[i], mode='lines', name=f'Class {i} (area = {roc_auc[i]:0.2f})'))

fig.add_shape(type='line', line=dict(dash='dash'), x0=0, x1=1, y0=0, y1=1)
fig.update_layout(title='Multiclass ROC Curve', xaxis_title='False Positive Rate', yaxis_title='True Positive Rate')
fig.show()
```



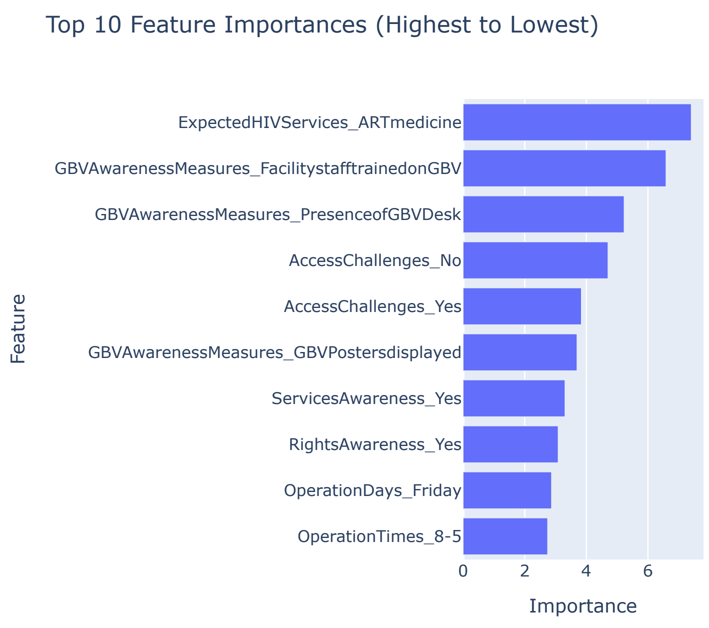
```
# Compute Precision-Recall and plot for each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_score[:, i])
    average_precision[i] = average_precision_score(y_test_bin[:, i], y_score[:, i])

fig = go.Figure()
for i in range(n_classes):
    fig.add_trace(go.Scatter(x=recall[i], y=precision[i], mode='lines', name=f'Class {i} (area = {average_precision[i]:0.

fig.update_layout(title='Multiclass Precision-Recall Curve', xaxis_title='Recall', yaxis_title='Precision')
fig.show()
```



```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
fig = ff.create_annotated_heatmap(z=cm, x=[f'Predicted {i}' for i in range(n_classes)], y=[f'Actual {i}' for i in range(n

fig.update_layout(title='Confusion Matrix', xaxis=dict(title='Predicted label'), yaxis=dict(title='True label'))
fig.show()
```



```python
feature_importances = best_model.feature_importances_

# Create a Series for the feature importances
importances = pd.Series(feature_importances, index=X_train.columns)

# Sort the importances and select the top 10, then reverse the Series for plotting
top_10_importances = importances.sort_values(ascending=False)[:10][::-1]

# Create a bar chart using Plotly
fig = px.bar(top_10_importances, x=top_10_importances.values, y=top_10_importances.index, orientation='h',
             labels={'x': 'Importance', 'index': 'Feature'},
             title='Top 10 Feature Importances (Highest to Lowest)')

# Show the plot
fig.show()
```


Top 10 Feature Importances (Highest to Lowest)
Feature
ExpectedHIVServices_ARTmedicine
GBVAwarenessMeasures_FacilitystafftrainedonGBV
GBVAwarenessMeasures_PresenceofGBVDesk
AccessChallenges_No
AccessChallenges_Yes
GBVAwarenessMeasures_GBVPostersdisplayed
ServicesAwareness_Yes
RightsAwareness_Yes
OperationDays_Friday
OperationTimes_8-5
Importance