notebook

August 21, 2024

1 D.R.I.P. Diabetes Risk Identification and Prediction

1.1 Introduction

Diabetes affections millions worldwide, leading to health complications and even death if not managed properly. Early identification and prediction prevents onset and progression of the disease. D.R.I.P. is a predictive model that with 70% accuracy can identify individuals at risk of developing diabetes, and biases toward false positives in order to catch as many people as possible.

1.2 Central Problem

Using data on health indicators, can we predict an individual's risk of developing diabetes? This prediction could enable timely interventions that may prevent or delay the onset of diabetes.

1.3 Data Understanding

This project leveraged the publically available `CDC Diabetes Health Indicators` data set to build statistical models to predict risk for diabetes. It is based on survey questions, and every one of the over 220,000 respondents were labeled with having diabetes or not.

1.4 Business Understanding

A tool using self reported health indicators to predict your risk of diabetes can significantly help preventative healthcare efforts. Identifying more high-risk individuals before they have full onset diabetes diabetes lets healthcare providers offer targeted lifestyle interventions, monitoring, and treatments to mitigate the risk. It also lowers insurance premiums and saves taxpayers money by lowering healthcare costs for everyone.

1.5 Stakeholders

1.6 Project Overview

This project follows a structured machine learning pipeline outlined below.

1. **Data Acquisition and Loading:** The dataset is loaded using an import from the UCI libary. This ensures data can be downloaded in any project and is reproducable.
2. **Data Understanding:** I examine the variables and understand their distributions, missing values, and relationships with the target variable.

3. **Data Preprocessing:** This step involves handling missing values, dealing with duplicates, and ensuring correct data types. We also perform scaling and binning where necessary.
4. **Feature Engineering:** I create new features through binning and interaction terms to improve model performance. Additionally, we drop or transform features based on their importance and correlation.
5. **Exploratory Data Analysis (EDA):** I conduct thorough EDA to identify patterns, distributions, and relationships that are critical for model selection and tuning.
6. **Model Development:** I experiment with various models including Logistic Regression, Random Forest, CatBoost, and TabNet. Each model is tuned using techniques like Optuna and GridSearchCV to find the best parameters.
7. **Model Evaluation:** The models are evaluated on various metrics such as accuracy, recall, precision, F1 score, and AUC-ROC to ensure robust performance. Threshold tuning is performed to balance the trade-off between precision and recall, with recall preferred.
8. **Final Model Selection and Interpretation:** Based on evaluation metrics, the best model is selected. I interpret its predictions and suggest practical applications for healthcare interventions.
9. **Saving Model For Deployment:** I create a new class which lets CatBoost be exported to ONNX Runtime Web and work as a model in a web browser. Without this modification it is not compatible. This deployment lets someone self report their stats and get a prediction on their diabetes risk.
10. **Conclusion and Recommendations:** I summarize the model's effectiveness and discuss its potential impact on healthcare cost savings and patient outcomes. Recommendations for future work and potential deployment strategies are also provided.

1.6.1 Conclusion

This project aims not only to build a robust predictive model for diabetes risk but also to highlight the importance of early detection and the potential impact of machine learning in preventive healthcare. The results of this project could identify more people at risk for diabetes, in a low cost way. Preventing more bad health outcomes improves lives, saves lives, lower costs for insurance, save taxpayers money, and most important of all, saves more people from having diabetes.

2 About the Dataset

Selected Variables Encoding Information: https://www.cdc.gov/pcd/issues/2019/19_0109.htm

Codebook: https://www.cdc.gov/brfss/annual_data/2015/pdf/codebook15_llcp.pdf

3 Imports

[ ]:
```
# Data Handling and Manipulation
import os
import json
import math
```

```python
import pickle
import pandas as pd
from scipy.stats import pointbiserialr, spearmanr
from sklearn.utils import resample

# Data Visualization
import sweetviz as sv
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display, Markdown

# Warning Handling
import warnings
from sklearn.exceptions import DataConversionWarning

# Machine Learning Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from catboost import CatBoostClassifier, Pool, cv
import xgboost as xgb
from pytorch_tabnet.tab_model import TabNetClassifier

# Model Selection and Evaluation
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit,␣
↪GridSearchCV, cross_val_score
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    roc_curve,
    confusion_matrix,
    precision_recall_curve,
    auc,
    precision_score,
    recall_score,
    f1_score,
    accuracy_score
)

# Preprocessing and Pipelines
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.pipeline import Pipeline

# Hyperparameter Optimization
import optuna
from optuna.samplers import RandomSampler
```

```python
# Hierarchical Clustering
from scipy.cluster.hierarchy import linkage, leaves_list

# Data Fetching (Specific to UCI ML Repository)
from ucimlrepo import fetch_ucirepo

# Suppress specific warnings
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

import re
from sklearn.base import BaseEstimator, ClassifierMixin


from onnx.helper import get_attribute_value
from skl2onnx import update_registered_converter
from skl2onnx.common.shape_calculator import␣
↪calculate_linear_classifier_output_shapes
from skl2onnx.common.data_types import Int64TensorType, guess_tensor_type
from skl2onnx._parse import _apply_zipmap, _get_sklearn_operator_name
from catboost.utils import convert_to_onnx_object
import onnxruntime as rt
import numpy as np

from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

import plotly.graph_objects as go
from plotly.offline import plot

from sklearn.metrics import precision_recall_fscore_support, accuracy_score
from matplotlib.colors import LinearSegmentedColormap

warnings.filterwarnings('ignore')
```

4 Data Acquisition and Loading

[ ]:
```python
data_dir = "data"  # You can change this to your preferred directory
feature_path = os.path.join(data_dir, "cdc_diabetes_data_features.pkl")
target_path = os.path.join(data_dir, "cdc_diabetes_data_targets.pkl")
metadata_path = os.path.join(data_dir, "cdc_diabetes_data_metadata.json")
variables_path = os.path.join(data_dir, "cdc_diabetes_data_variables.pkl")  #␣
↪Changed to .pkl for pickle

def load_dataset():
    # Ensure the directory exists
    if not os.path.exists(data_dir):
```

```python
        os.makedirs(data_dir)

    # Check if the dataset is already downloaded and saved locally
    if not (os.path.exists(feature_path) and os.path.exists(target_path) and
            os.path.exists(metadata_path) and os.path.exists(variables_path)):

        # Assuming fetch_ucirepo is a function that downloads the dataset
        cdc_diabetes_health_indicators = fetch_ucirepo(id=891)
        display(cdc_diabetes_health_indicators.keys())

        # Save the features, targets, metadata, and variables locally
        cdc_diabetes_health_indicators.data.features.to_pickle(feature_path)
        cdc_diabetes_health_indicators.data.targets.to_pickle(target_path)

        # Serialize metadata using JSON
        metadata = cdc_diabetes_health_indicators.metadata
        with open(metadata_path, 'w') as f:
            json.dump(metadata, f)

        # Serialize variables using pickle
        variables = cdc_diabetes_health_indicators.variables
        with open(variables_path, 'wb') as f:
            pickle.dump(variables, f)

        print("Downloaded and saved the dataset locally.")
        features = cdc_diabetes_health_indicators.data.features
        targets = cdc_diabetes_health_indicators.data.targets

    else:
        # Load the features, targets, metadata, and variables from local storage
        features = pd.read_pickle(feature_path)
        targets = pd.read_pickle(target_path)

        with open(metadata_path, 'r') as f:
            metadata = json.load(f)

        with open(variables_path, 'rb') as f:
            variables = pickle.load(f)

        print("Loaded dataset from local storage.")

    return features, targets, metadata, variables

# Use the function to load your dataset
X, y, metadata, variables = load_dataset()

# Display metadata and variables to ensure they're correctly loaded
```

```python
print("Metadata:", metadata)
display("Variables:", variables)
from sklearn.metrics import roc_auc_score, roc_curve, classification_report,␣
↪confusion_matrix, accuracy_score

import numpy as np
```

```
Loaded dataset from local storage.
Metadata: {'uci_id': 891, 'name': 'CDC Diabetes Health Indicators',
'repository_url':
'https://archive.ics.uci.edu/dataset/891/cdc+diabetes+health+indicators',
'data_url': 'https://archive.ics.uci.edu/static/public/891/data.csv',
'abstract': 'The Diabetes Health Indicators Dataset contains healthcare
statistics and lifestyle survey information about people in general along with
their diagnosis of diabetes. The 35 features consist of some demographics, lab
test results, and answers to survey questions for each patient. The target
variable for classification is whether a patient has diabetes, is pre-diabetic,
or healthy. ', 'area': 'Health and Medicine', 'tasks': ['Classification'],
'characteristics': ['Tabular', 'Multivariate'], 'num_instances': 253680,
'num_features': 21, 'feature_types': ['Categorical', 'Integer'], 'demographics':
['Sex', 'Age', 'Education Level', 'Income'], 'target_col': ['Diabetes_binary'],
'index_col': ['ID'], 'has_missing_values': 'no', 'missing_values_symbol': None,
'year_of_dataset_creation': 2017, 'last_updated': 'Fri Nov 03 2023',
'dataset_doi': '10.24432/C53919', 'creators': [], 'intro_paper': {'title':
'Incidence of End-Stage Renal Disease Attributed to Diabetes Among Persons with
Diagnosed Diabetes - United States and Puerto Rico, 2000-2014', 'authors':
'Nilka Rios Burrows, MPH; Israel Hora, PhD; Linda S. Geiss, MA; Edward W. Gregg,
PhD; Ann Albright, PhD', 'published_in': 'Morbidity and Mortality Weekly
Report', 'year': 2017, 'url':
'https://www.cdc.gov/mmwr/volumes/66/wr/mm6643a2.htm', 'doi': None},
'additional_info': {'summary': 'Dataset link:
https://www.cdc.gov/brfss/annual_data/annual_2014.html', 'purpose': 'To better
understand the relationship between  lifestyle and diabetes in the US',
'funded_by': 'The CDC', 'instances_represent': 'Each row represents a person
participating in this study.', 'recommended_data_splits': 'Cross validation or a
fixed train-test split could be used.', 'sensitive_data': '- Gender\n- Income\n-
Education level', 'preprocessing_description': 'Bucketing of age',
'variable_info': '- Diabetes diagnosis\n- Demographics (race, sex)\n- Personal
information (income, educations)\n- Health history (drinking, smoking, mental
health, physical health)', 'citation': None}, 'external_url':
'https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset'}

'Variables:'

                    name     role     type      demographic  \
0                     ID       ID  Integer             None
1        Diabetes_binary   Target   Binary             None
2                 HighBP  Feature   Binary             None
```

```
3              HighChol  Feature   Binary             None
4             CholCheck  Feature   Binary             None
5                   BMI  Feature  Integer             None
6                Smoker  Feature   Binary             None
7                Stroke  Feature   Binary             None
8   HeartDiseaseorAttack  Feature   Binary             None
9          PhysActivity  Feature   Binary             None
10               Fruits  Feature   Binary             None
11              Veggies  Feature   Binary             None
12    HvyAlcoholConsump  Feature   Binary             None
13        AnyHealthcare  Feature   Binary             None
14          NoDocbcCost  Feature   Binary             None
15              GenHlth  Feature  Integer             None
16             MentHlth  Feature  Integer             None
17             PhysHlth  Feature  Integer             None
18             DiffWalk  Feature   Binary             None
19                  Sex  Feature   Binary              Sex
20                  Age  Feature  Integer              Age
21            Education  Feature  Integer  Education Level
22               Income  Feature  Integer           Income

                                        description units missing_values
0                                        Patient ID  None             no
1        0 = no diabetes 1 = prediabetes or diabetes  None             no
2                         0 = no high BP 1 = high BP  None             no
3       0 = no high cholesterol 1 = high cholesterol  None             no
4   0 = no cholesterol check in 5 years 1 = yes ch…  None            no
5                                   Body Mass Index  None             no
6   Have you smoked at least 100 cigarettes in you…  None            no
7        (Ever told) you had a stroke. 0 = no 1 = yes  None             no
8   coronary heart disease (CHD) or myocardial inf…  None            no
9   physical activity in past 30 days - not includ…  None            no
10  Consume Fruit 1 or more times per day 0 = no 1…  None            no
11  Consume Vegetables 1 or more times per day 0 =…  None            no
12  Heavy drinkers (adult men having more than 14 …  None            no
13  Have any kind of health care coverage, includi…  None            no
14  Was there a time in the past 12 months when yo…  None            no
15  Would you say that in general your health is: …  None            no
16  Now thinking about your mental health, which i…  None            no
17  Now thinking about your physical health, which…  None            no
18  Do you have serious difficulty walking or clim…  None            no
19                                0 = female 1 = male  None             no
20  13-level age category (_AGEG5YR see codebook) …  None            no
21  Education level (EDUCA see codebook) scale 1-6…  None            no
22  Income scale (INCOME2 see codebook) scale 1-8 …  None            no
```

[ ]:
```
display(X)
```

```
        HighBP  HighChol  CholCheck  BMI  Smoker  Stroke  \
0            1         1          1   40       1       0
1            0         0          0   25       1       0
2            1         1          1   28       0       0
3            1         0          1   27       0       0
4            1         1          1   24       0       0
…          …         …          …   …      …      …
253675       1         1          1   45       0       0
253676       1         1          1   18       0       0
253677       0         0          1   28       0       0
253678       1         0          1   23       0       0
253679       1         1          1   25       0       0

        HeartDiseaseorAttack  PhysActivity  Fruits  Veggies  …  \
0                          0             0       0        1  …
1                          0             1       0        0  …
2                          0             0       1        0  …
3                          0             1       1        1  …
4                          0             1       1        1  …
…                        …           …      …      …  …
253675                     0             0       1        1  …
253676                     0             0       0        0  …
253677                     0             1       1        0  …
253678                     0             0       1        1  …
253679                     1             1       1        0  …

        AnyHealthcare  NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  \
0                   1            0        5        18        15         1
1                   0            1        3         0         0         0
2                   1            1        5        30        30         1
3                   1            0        2         0         0         0
4                   1            0        2         3         0         0
…                 …          …      …        …        …       …
253675              1            0        3         0         5         0
253676              1            0        4         0         0         1
253677              1            0        1         0         0         0
253678              1            0        3         0         0         0
253679              1            0        2         0         0         0

        Sex  Age  Education  Income
0         0    9          4       3
1         0    7          6       1
2         0    9          4       8
3         0   11          3       6
4         0   11          5       4
…     …  …         …     …
253675    1    5          6       7
253676    0   11          2       4
```

```
253677    0    2          5       2
253678    1    7          5       1
253679    0    9          6       2

[253680 rows x 21 columns]
```

[ ]:
```python
display(y)
```

```
        Diabetes_binary
0                     0
1                     0
2                     0
3                     0
4                     0
…                   …
253675                0
253676                1
253677                0
253678                0
253679                1

[253680 rows x 1 columns]
```

5 Data Understanding

5.0.1 Plot Single Feature Function

[ ]:
```python
def plot_single_feature(col, df=X):

    if col not in df.columns:
        return
    # Set the style with a more widely available font and a light blue␣
↪background
    sns.set(style="whitegrid", font="DejaVu Sans", rc={"axes.facecolor":␣
↪"#EAF2F8", "figure.facecolor": "#EAF2F8"})
    custom_palette = ["#E1D7C6", "#4A90E2"]  # Medical light blue and gray

    plt.figure(figsize=(5, 3))

    if df[col].nunique() == 2:  # Binary feature
        # Calculate the proportion of each class
        value_counts = df[col].value_counts(normalize=True).sort_index()

        # Determine the labels
        labels = value_counts.index

        # Create a DataFrame for plotting
```

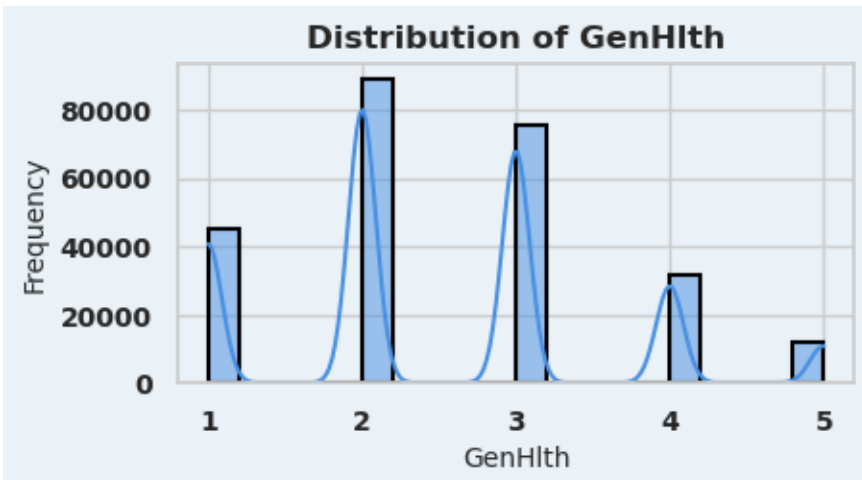
```
        plot_data = pd.DataFrame({
            'Label': labels,
            'Proportion': value_counts.values,
            'Category': labels
        })

        # Create a bar plot with the actual labels using hue and palette
        sns.barplot(x='Label', y='Proportion', hue='Category', data=plot_data,
                    palette=custom_palette, dodge=False, edgecolor=".2",␣
↪linewidth=1.5)

        # Add title and labels
        plt.title(f'Class Balance for {col}', fontsize=12, fontweight='bold')
        plt.xlabel('', fontsize=10)
        plt.ylabel('Proportion', fontsize=10)
        plt.ylim(0, 1.05)  # Set y-axis to go slightly above 1 to prevent␣
↪percentage overlap

        # Adjust x-tick labels for size and weight
        plt.xticks(fontsize=10, fontweight='bold')

        # Display the percentages on the bars with padding to avoid bleeding
        for index, value in enumerate(value_counts.values):
            plt.text(index, value + 0.02, f'{value:.2%}', ha='center',␣
↪fontsize=10, color='black', fontweight='bold')

    else:  # Numeric feature
        sns.histplot(df[col], bins=20, kde=True, color="#4A90E2",␣
↪edgecolor="black", linewidth=1.5)

        # Add title and labels
        plt.title(f'Distribution of {col}', fontsize=12, fontweight='bold')
        plt.xlabel(col, fontsize=10)
        plt.ylabel('Frequency', fontsize=10)

        # Adjust x-tick labels for size and weight
        plt.xticks(fontsize=10, fontweight='bold')
        plt.yticks(fontsize=10, fontweight='bold')

    # Adjust layout to prevent overlap
    plt.tight_layout(pad=2.0)

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

5.0.2 Display Variable Desc Function

[ ]:
```
def display_variable_desc(varname):
    try:
        vardesc = variables[variables['name'] == varname]['description'].
 ↪values[0]
        display(Markdown(f"**{varname} Description:**\n\n{vardesc}"))
        plot_single_feature(varname)
    except Exception as e:
        display(f"Could not find description for {varname}")

display_variable_desc('GenHlth')
```

GenHlth Description:

Would you say that in general your health is: scale 1-5 1 = excellent 2 = very good 3 = good 4 = fair 5 = poor

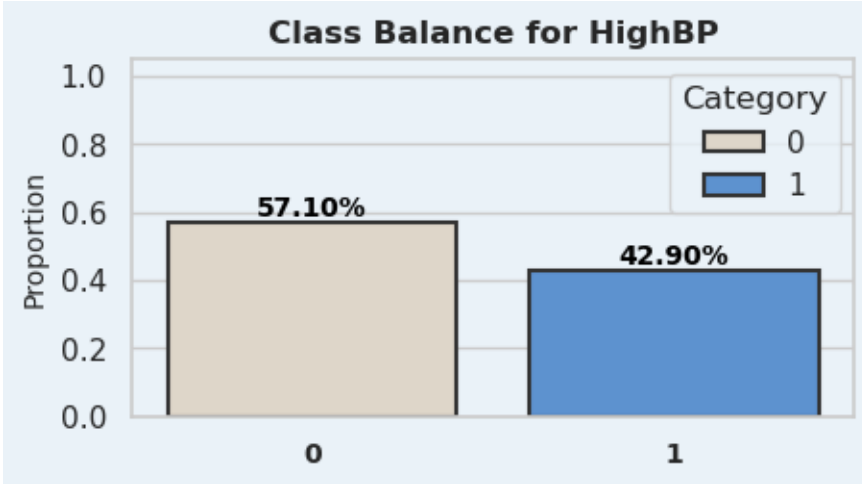


5.0.3 Feature List and Descriptions

[ ]:
```
for col in X.columns:
    colname=col.replace("_binned","")
    display_variable_desc(colname)
```

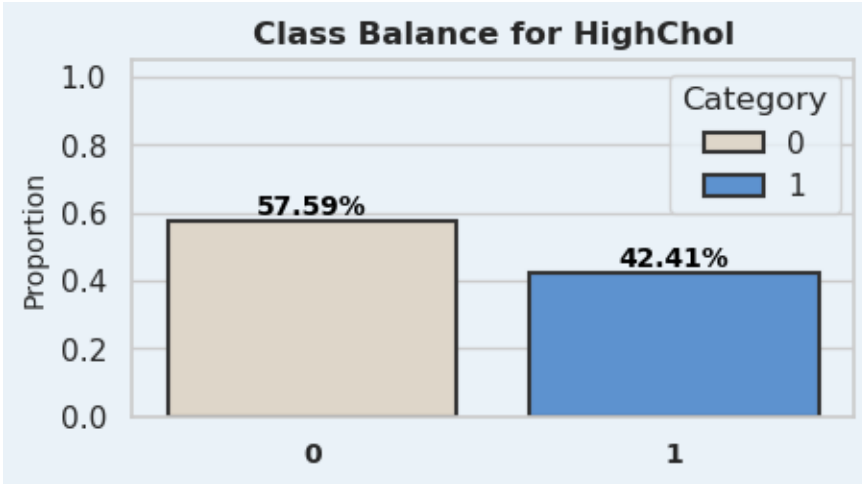
HighBP Description:

0 = no high BP 1 = high BP



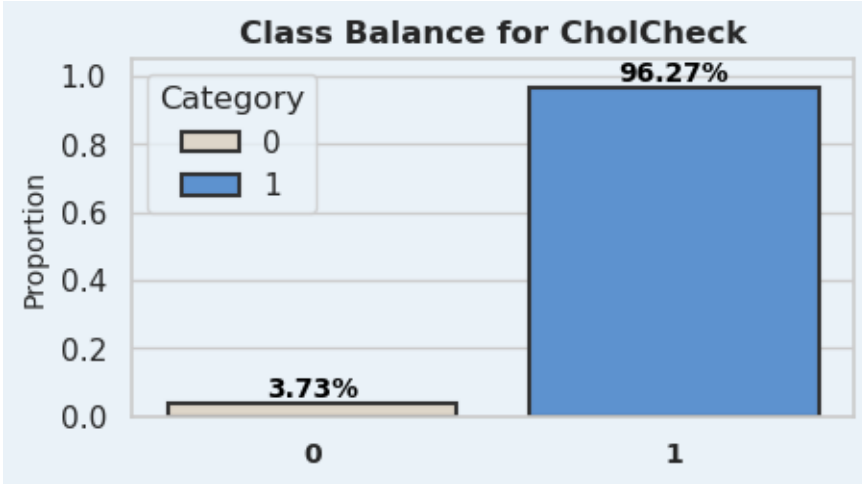
HighChol Description:

0 = no high cholesterol 1 = high cholesterol



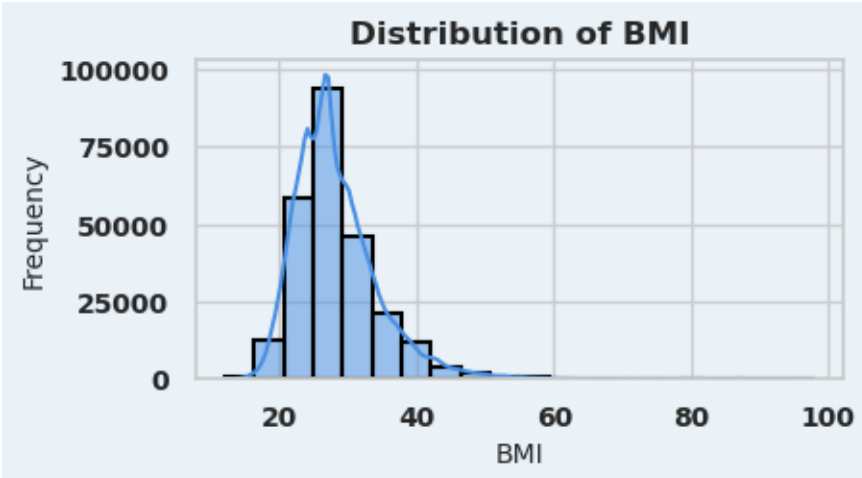
CholCheck Description:

0 = no cholesterol check in 5 years 1 = yes cholesterol check in 5 years



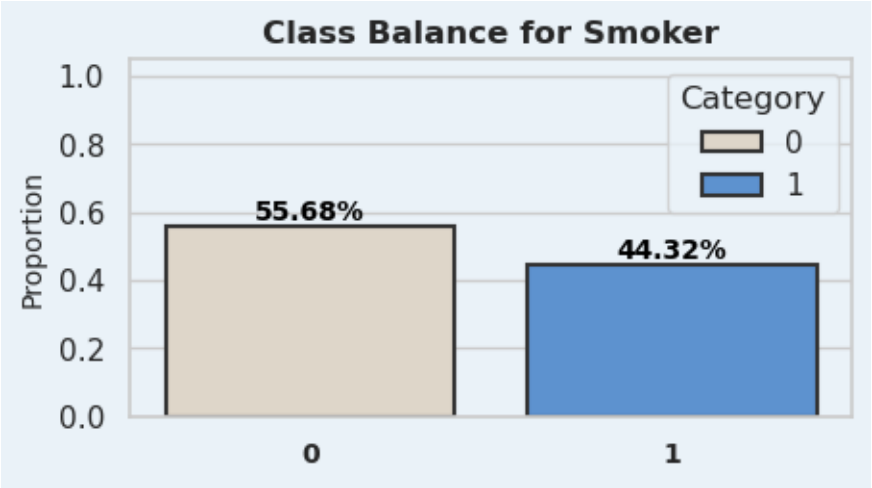
BMI Description:

Body Mass Index



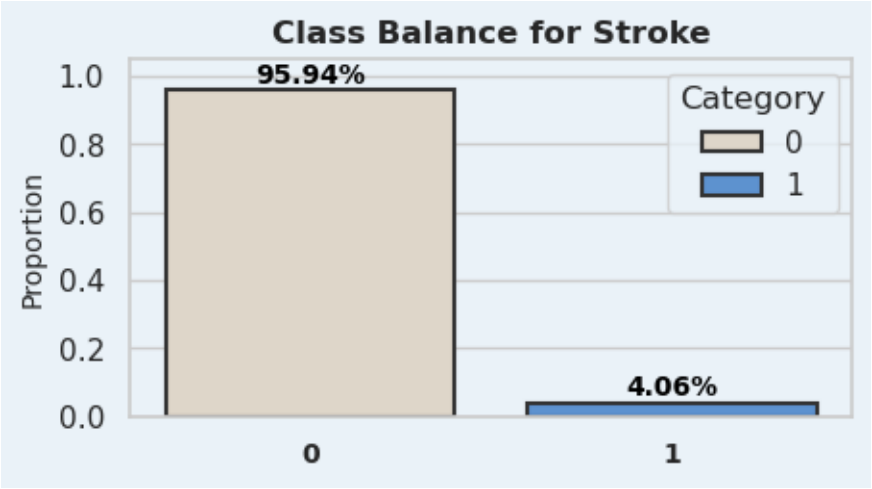
Smoker Description:

Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes] 0 = no 1 = yes



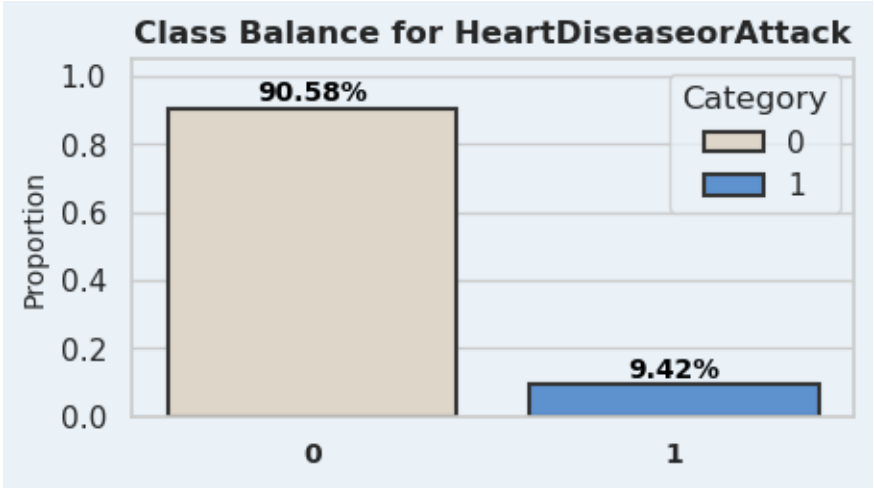
Stroke Description:

(Ever told) you had a stroke. 0 = no 1 = yes



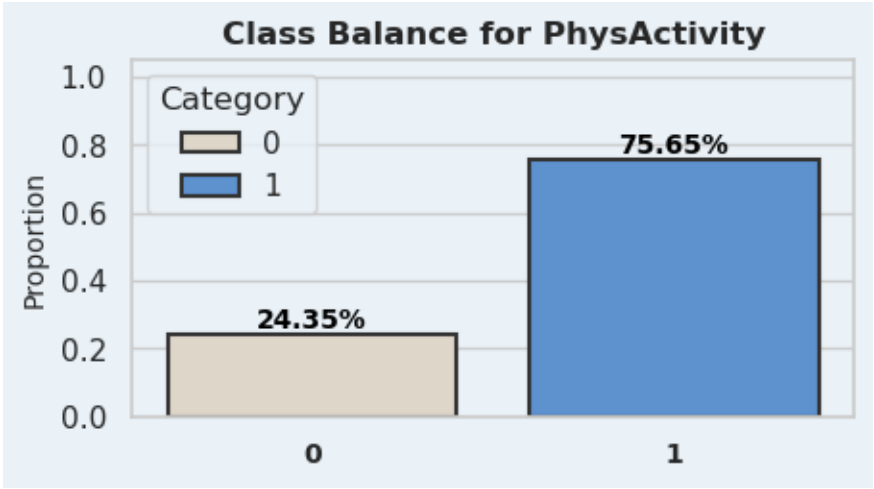
HeartDiseaseorAttack Description:

coronary heart disease (CHD) or myocardial infarction (MI) 0 = no 1 = yes



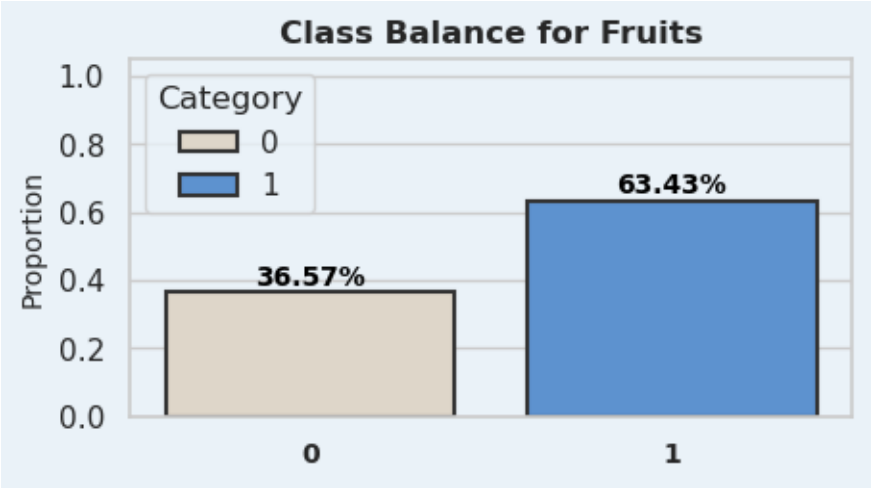
PhysActivity Description:

physical activity in past 30 days - not including job 0 = no 1 = yes



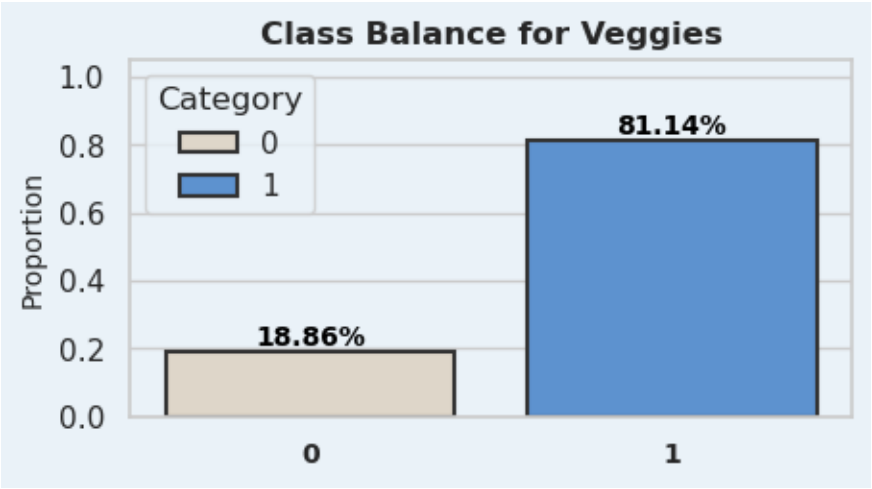
Fruits Description:

Consume Fruit 1 or more times per day 0 = no 1 = yes



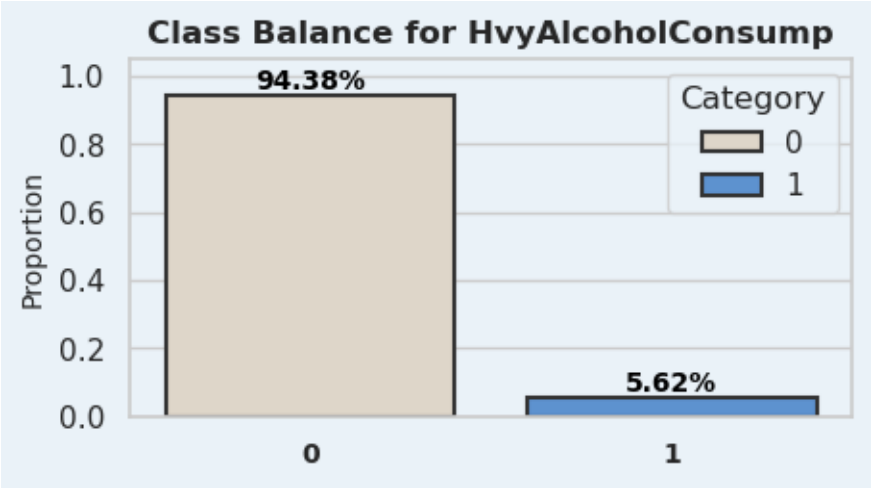
Veggies Description:

Consume Vegetables 1 or more times per day 0 = no 1 = yes



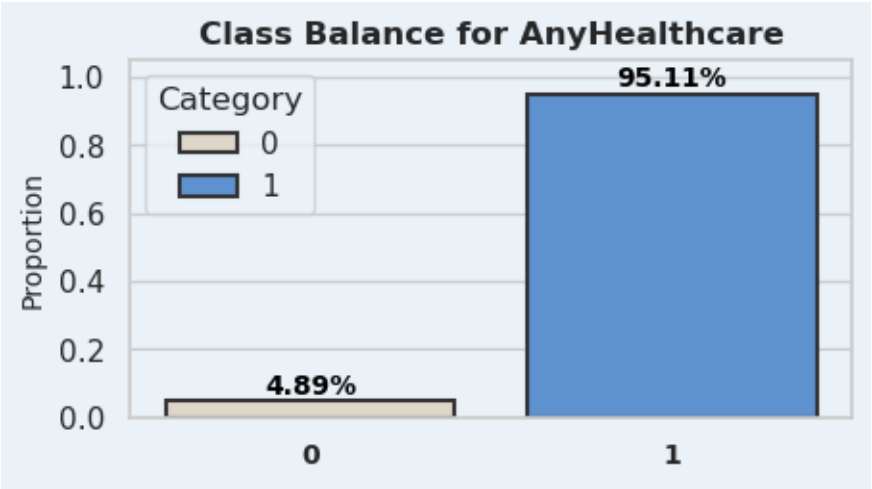
HvyAlcoholConsump Description:

Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week) 0 = no 1 = yes



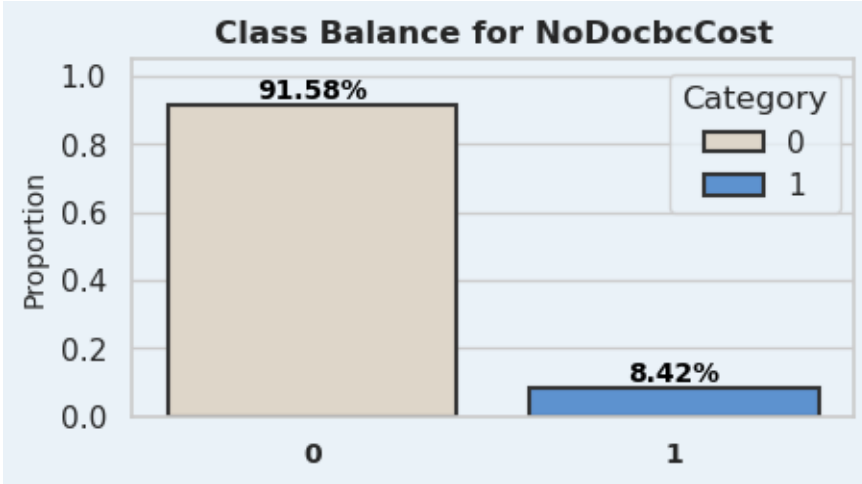
AnyHealthcare Description:

Have any kind of health care coverage, including health insurance, prepaid plans such as HMO, etc. 0 = no 1 = yes



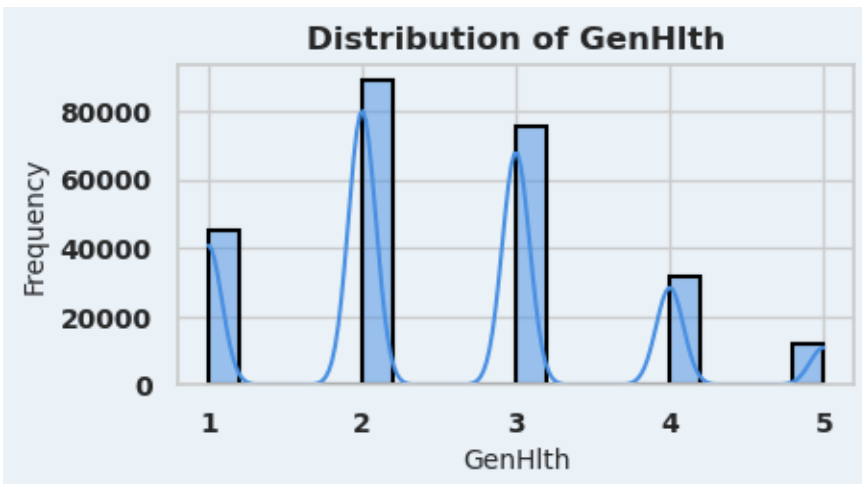
NoDocbcCost Description:

Was there a time in the past 12 months when you needed to see a doctor but could not because of cost? 0 = no 1 = yes



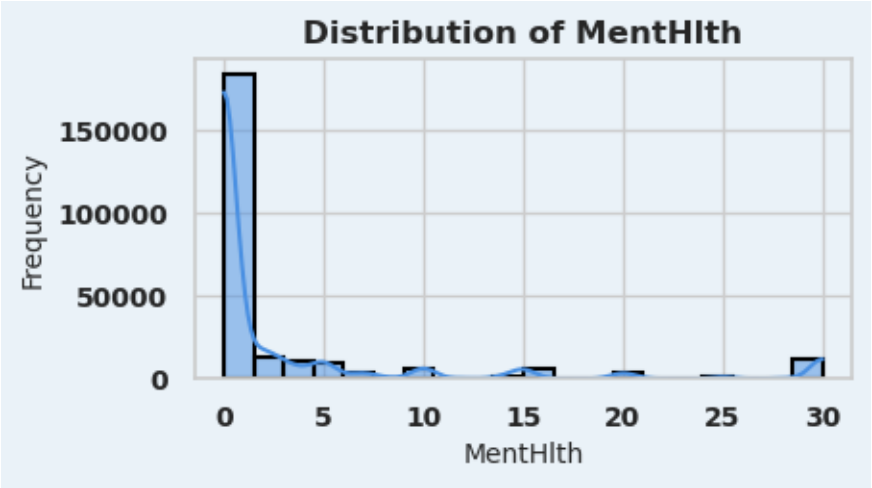
GenHlth Description:

Would you say that in general your health is: scale 1-5 1 = excellent 2 = very good 3 = good 4 = fair 5 = poor



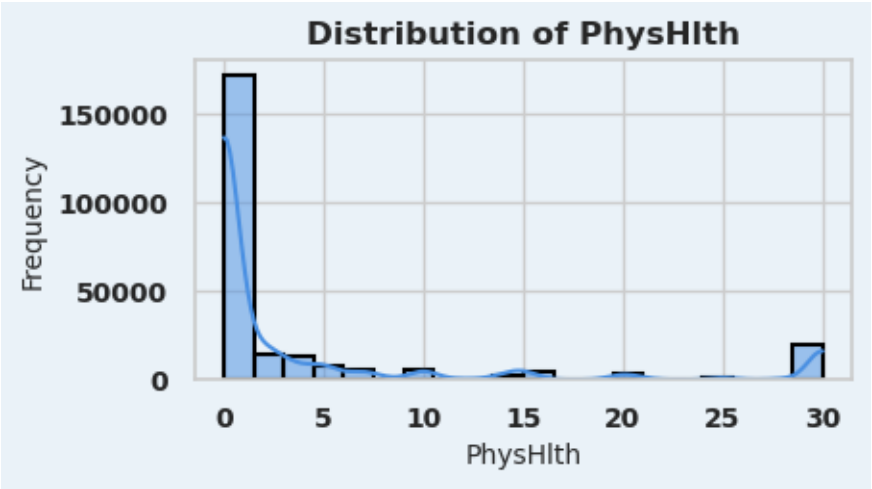
MentHlth Description:

Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good? scale 1-30 days



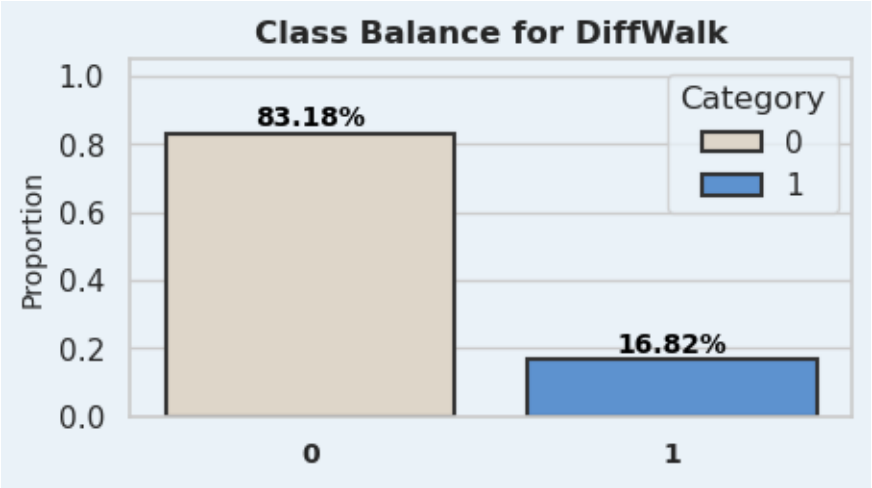
PhysHlth Description:

Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? scale 1-30 days



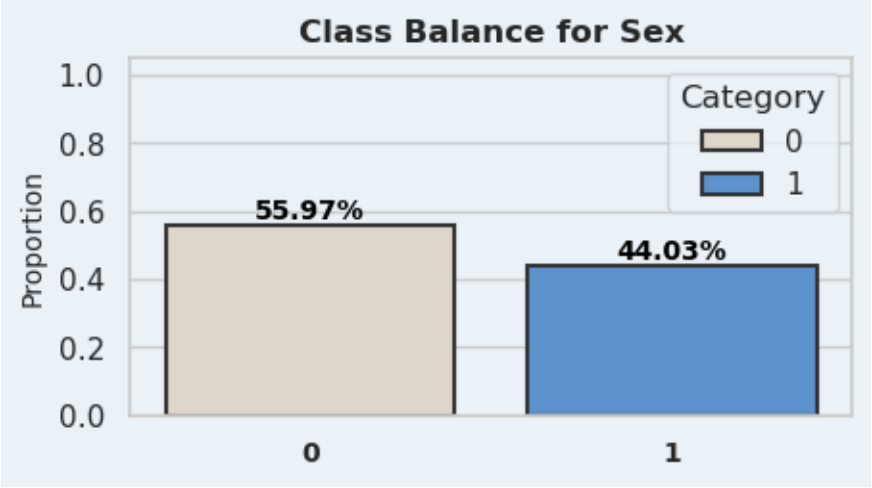
DiffWalk Description:

Do you have serious difficulty walking or climbing stairs? 0 = no 1 = yes



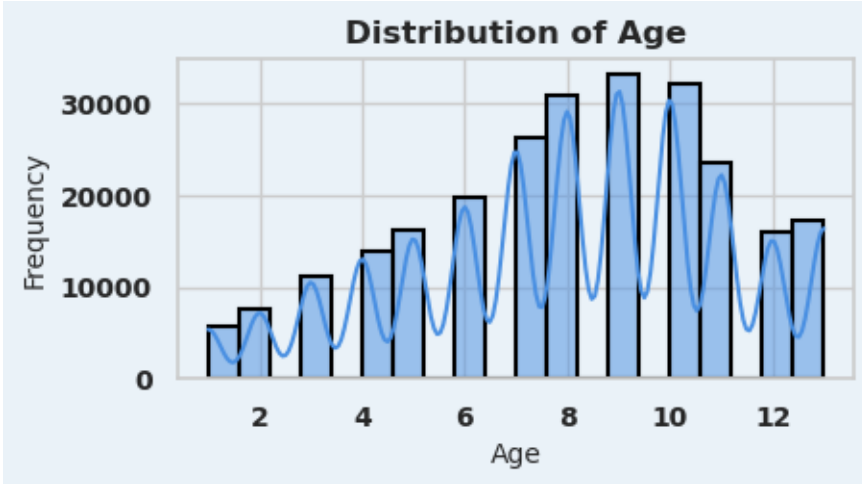
Sex Description:

0 = female 1 = male



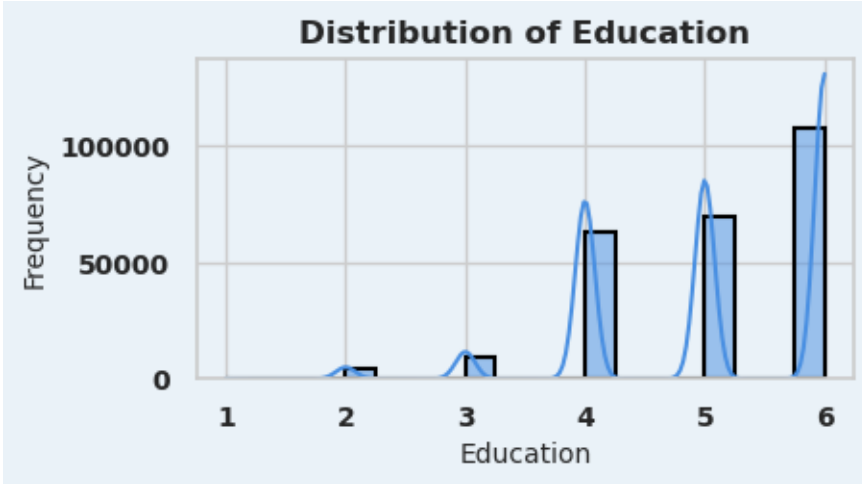
Age Description:

13-level age category (_AGEG5YR see codebook) 1 = 18-24 9 = 60-64 13 = 80 or older



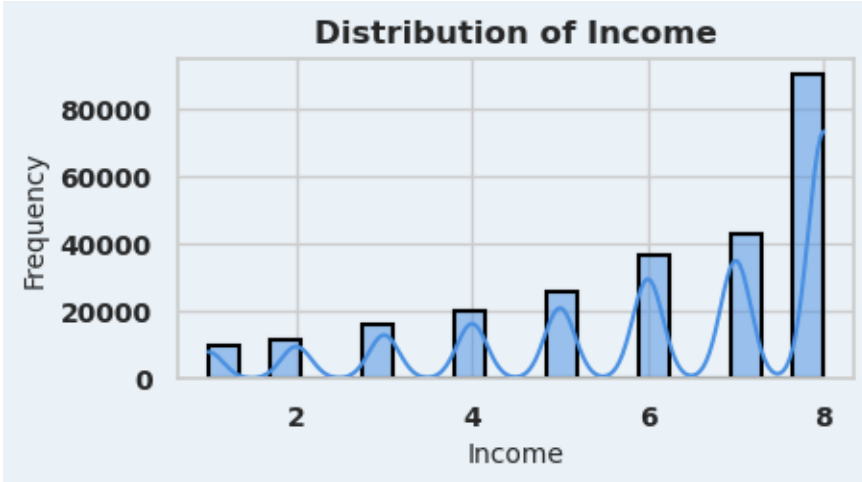
Education Description:

Education level (EDUCA see codebook) scale 1-6 1 = Never attended school or only kindergarten 2 = Grades 1 through 8 (Elementary) 3 = Grades 9 through 11 (Some high school) 4 = Grade 12 or GED (High school graduate) 5 = College 1 year to 3 years (Some college or technical school) 6 = College 4 years or more (College graduate)



Income Description:

Income scale (INCOME2 see codebook) scale 1-8 1 = less than $10,000 5 = less than $35,000 8 = $75,000 or more



6 SweetViz

[ ]:
```
if 'report' not in globals():
    # Generate a Sweetviz report
    report = sv.analyze(X)

# Save the report to an HTML file
#report.show_html("sweetviz_report.html")
report.show_notebook()
```

```
Done! Use 'show' commands to display/save.   |          | [100%]   00:00 ->
(00:00 left)
```

```
<IPython.core.display.HTML object>
```

7 EDA and Data Prep

7.1 Missing Values

There are no missing values! This will save us a big headache.

There are some that are less informative though. (Other) exists in `Income` and `Education`

7.2 Distinct and Data Types

Many values are binary. Distinct also shows missingness isn't encoded for binary so the missing values really just don't exist.

7.2.1 Binary

Since I verified they don't have missing this should be an easy check `HighBP`, `HighChol`, `CholCheck`, `Smoker`, `Stroke`, `HeartDiseaseorAttack`, `PhysActivity`, `Fruits`, `Veggies`, `HvyAlcoholConsump`,

`AnyHealthcare`, `NoDocbcCost`, `DiffWalk`, `Sex` are the binary columns

[ ]:
```
unique_counts = X.nunique()
bin_cols = unique_counts[unique_counts == 2].index.tolist()
bin_cols
```

[ ]:
```
['HighBP',
 'HighChol',
 'CholCheck',
 'Smoker',
 'Stroke',
 'HeartDiseaseorAttack',
 'PhysActivity',
 'Fruits',
 'Veggies',
 'HvyAlcoholConsump',
 'AnyHealthcare',
 'NoDocbcCost',
 'DiffWalk',
 'Sex']
```

7.2.2 Continuous

`BMI`, `PhysHlth`, `MentHlth` These columns can't be encoded as anything they just are what they are.

[ ]:
```
cont_cols = ['BMI','PhysHlth','MentHlth']
cont_cols
```

[ ]:
```
['BMI', 'PhysHlth', 'MentHlth']
```

7.2.3 Discrete

`GenHlth`, `Age`, `Education`, `Income` Age was binned so it is discrete here. Education and Income are categories already encoded.

[ ]:
```
disc_cols = ['GenHtlh','Age','Education','Income']
```

7.2.4 Validate Cols

[ ]:
```
all_cols = bin_cols + disc_cols + cont_cols
display(len(all_cols))
if all_cols.sort() == X.columns.to_list().sort():
    print("\n All Columns Categorized")
```

```
21

 All Columns Categorized
```

7.3 Duplicates

Why are there so many duplicate rows?

[ ]:
```
X[X.duplicated()]
```

[ ]:
```
        HighBP  HighChol  CholCheck  BMI  Smoker  Stroke  \
1242         1         1          1   27       1       0
1563         0         0          1   21       1       0
2700         0         0          1   32       0       0
3027         1         0          1   38       0       0
3160         0         0          1   21       0       0
…          …         …          …   …      …      …
253492       1         1          1   33       0       0
253550       0         0          1   25       0       0
253563       0         1          1   24       1       0
253597       0         0          1   24       0       0
253638       0         0          1   24       0       0

        HeartDiseaseorAttack  PhysActivity  Fruits  Veggies  …  \
1242                       0             0       0        1  …
1563                       0             1       1        1  …
2700                       0             1       1        1  …
3027                       0             1       1        1  …
3160                       0             1       1        1  …
…                        …             …      …       …  …
253492                     0             1       1        1  …
253550                     0             1       1        1  …
253563                     0             1       1        1  …
253597                     0             1       1        1  …
253638                     0             1       1        1  …

        AnyHealthcare  NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  \
1242                1            0        5         0        30         1
1563                1            0        1         0         0         0
2700                1            0        2         0         0         0
3027                1            0        3         0         0         0
3160                1            0        1         0         0         0
…                 …            …       …        …        …        …
253492              1            0        3         0         0         0
253550              1            0        1         0         0         0
253563              1            0        2         0         0         0
253597              1            0        2         0         0         0
253638              1            0        2         0         0         0

        Sex  Age  Education  Income
1242      0   10          4       5
1563      0    4          6       8
```

```
2700       0    5          6       8
3027       1    7          6       8
3160       0    4          6       8
…      …   …         …      …
253492     1    9          6       6
253550     0    7          6       8
253563     1    8          6       8
253597     0    5          6       8
253638     1    1          4       6

[25772 rows x 21 columns]
```

7.3.1 Statistically Likely?

[ ]:
```
X.duplicated().sum() / len(X)
```

[ ]:
```
0.10159255755282245
```

10% of rows being duplicate among how many features is how likely?

Lets assume they are all binary to just make the math easy

7.3.2 Broad Estimate

[ ]:
```
num_combinations = 1
for column in X.columns:
    distinct_values = X[column].nunique()
    num_combinations *= distinct_values

print(f"Total number of unique combinations possible: {num_combinations:,}")
```

```
Total number of unique combinations possible: 4,126,455,889,920
```

[ ]:
```
n = len(X)
# Calculating the expected number of duplicates
expected_duplicates = (n * (n - 1)) / (2 * num_combinations)
print(f"Expected number of duplicate rows: {expected_duplicates:,}")
```

```
Expected number of duplicate rows: 0.007797646507890773
```

7.3.3 There shouldn't be any duplicates but 10% of the dataset has them

The expected duplicates being close to zero strongly indicates that having 10% of the dataset be duplicates is extremely unlikely by chance. So there has to be another reason.

The duplicates could be for a slew of reasons, such as sampling strategies in surveys, repeated surveys including the same people or errors in aggergating surveys with scripts.

In any case, I should either delete all of them or keep only 1 of each. Since their data integrity is in quesiton and there's so many rows I can just drop them.

[ ]:
```
drop_masq = X.duplicated()
y = y[~drop_masq]
X = X[~drop_masq]
X.duplicated().sum()
```

[ ]: 0

7.4 Data Type Checking

[ ]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 227908 entries, 0 to 253679
Data columns (total 21 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   HighBP                227908 non-null  int64
 1   HighChol              227908 non-null  int64
 2   CholCheck             227908 non-null  int64
 3   BMI                   227908 non-null  int64
 4   Smoker                227908 non-null  int64
 5   Stroke                227908 non-null  int64
 6   HeartDiseaseorAttack  227908 non-null  int64
 7   PhysActivity          227908 non-null  int64
 8   Fruits                227908 non-null  int64
 9   Veggies               227908 non-null  int64
 10  HvyAlcoholConsump     227908 non-null  int64
 11  AnyHealthcare         227908 non-null  int64
 12  NoDocbcCost           227908 non-null  int64
 13  GenHlth               227908 non-null  int64
 14  MentHlth              227908 non-null  int64
 15  PhysHlth              227908 non-null  int64
 16  DiffWalk              227908 non-null  int64
 17  Sex                   227908 non-null  int64
 18  Age                   227908 non-null  int64
 19  Education             227908 non-null  int64
 20  Income                227908 non-null  int64
dtypes: int64(21)
memory usage: 38.3 MB
```

Many of these are boolean but all are in int64 lets fix that

7.4.1 Binary Cols Should be Boolean

[ ]:
```
X[bin_cols] = X[bin_cols].astype('bool')
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 227908 entries, 0 to 253679
Data columns (total 21 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   HighBP                227908 non-null  bool
 1   HighChol              227908 non-null  bool
 2   CholCheck             227908 non-null  bool
 3   BMI                   227908 non-null  int64
 4   Smoker                227908 non-null  bool
 5   Stroke                227908 non-null  bool
 6   HeartDiseaseorAttack  227908 non-null  bool
 7   PhysActivity          227908 non-null  bool
 8   Fruits                227908 non-null  bool
 9   Veggies               227908 non-null  bool
 10  HvyAlcoholConsump     227908 non-null  bool
 11  AnyHealthcare         227908 non-null  bool
 12  NoDocbcCost           227908 non-null  bool
 13  GenHlth               227908 non-null  int64
 14  MentHlth              227908 non-null  int64
 15  PhysHlth              227908 non-null  int64
 16  DiffWalk              227908 non-null  bool
 17  Sex                   227908 non-null  bool
 18  Age                   227908 non-null  int64
 19  Education             227908 non-null  int64
 20  Income                227908 non-null  int64
dtypes: bool(14), int64(7)
memory usage: 17.0 MB
```

7.4.2 max value of int64 checks

We can improve memory usage and computation speed if we lower the values to their sizes

[ ]:
```
int_cols = X.select_dtypes('int64').columns.to_list()
for col in int_cols:
    print(f"{col} range is {X[col].min()} - {X[col].max()}")
```

```
BMI range is 12 - 98
GenHlth range is 1 - 5
MentHlth range is 0 - 30
PhysHlth range is 0 - 30
Age range is 1 - 13
Education range is 1 - 6
```

```
Income range is 1 - 8
```

[ ]:
```python
int_cols
```

[ ]:
```
['BMI', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Education', 'Income']
```

7.4.3 Cast to uint8

They are all less than 255 by a lot, which means uint8 is big enough for them.

[ ]:
```python
cast_dict = {}
for col in int_cols:
    cast_dict[col] = 'uint8'
display(cast_dict)
X = X.astype(cast_dict)
X.info()
```

```
{'BMI': 'uint8',
 'GenHlth': 'uint8',
 'MentHlth': 'uint8',
 'PhysHlth': 'uint8',
 'Age': 'uint8',
 'Education': 'uint8',
 'Income': 'uint8'}
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 227908 entries, 0 to 253679
Data columns (total 21 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   HighBP                227908 non-null  bool
 1   HighChol              227908 non-null  bool
 2   CholCheck             227908 non-null  bool
 3   BMI                   227908 non-null  uint8
 4   Smoker                227908 non-null  bool
 5   Stroke                227908 non-null  bool
 6   HeartDiseaseorAttack  227908 non-null  bool
 7   PhysActivity          227908 non-null  bool
 8   Fruits                227908 non-null  bool
 9   Veggies               227908 non-null  bool
 10  HvyAlcoholConsump     227908 non-null  bool
 11  AnyHealthcare         227908 non-null  bool
 12  NoDocbcCost           227908 non-null  bool
 13  GenHlth               227908 non-null  uint8
 14  MentHlth              227908 non-null  uint8
 15  PhysHlth              227908 non-null  uint8
 16  DiffWalk              227908 non-null  bool
 17  Sex                   227908 non-null  bool
```

```
 18  Age                    227908 non-null  uint8
 19  Education              227908 non-null  uint8
 20  Income                 227908 non-null  uint8
dtypes: bool(14), uint8(7)
memory usage: 6.3 MB
```

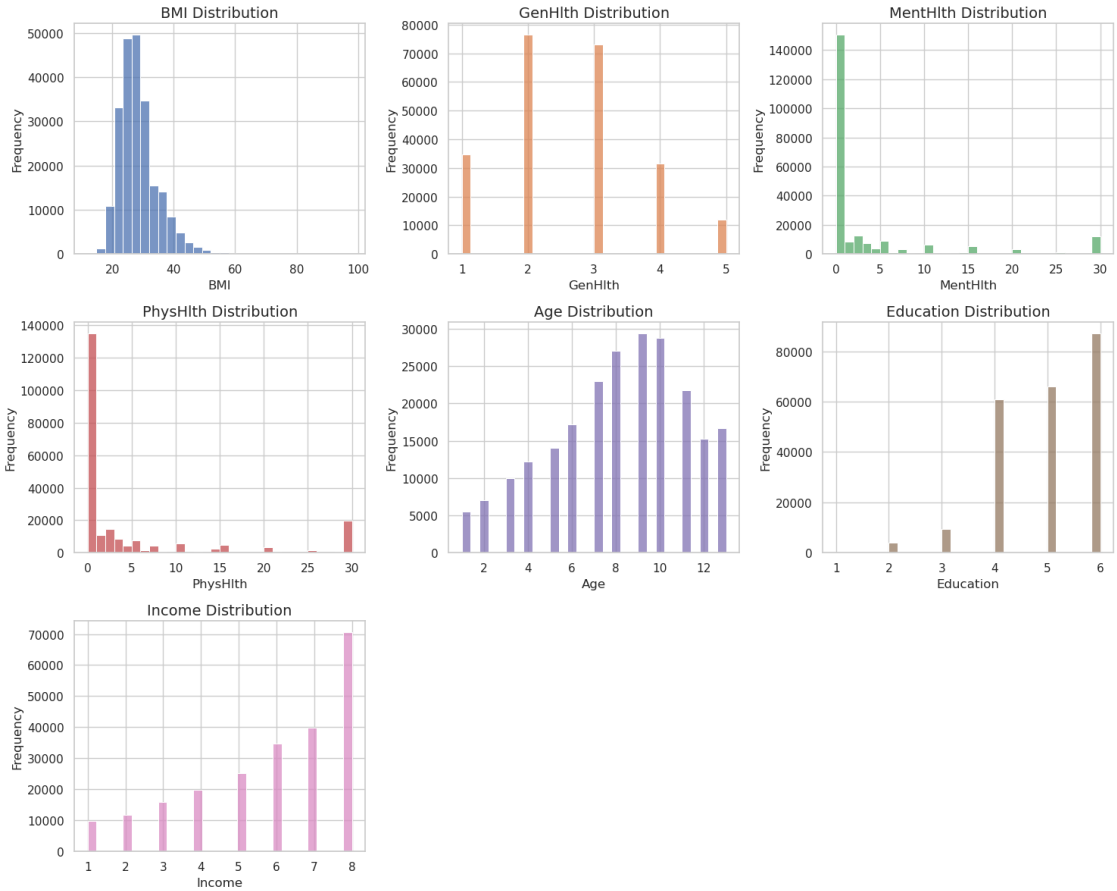
7.4.4 Analysis of Casting

great! we went from `memory usage: 38.3 MB` to `memory usage: 6.3 MB` with zero information loss! Feels great. This should really help our model training and efficiency overall, as well as let models with special handling of booleans benefit from it!

7.5 Int Cols Distributions and Class Imbalances

I need to analyze the distributions of all the int cols

[ ]:
```
def hist_plot_cols(cols):
    # Set the style and color palette
    sns.set(style="whitegrid")
    palette = sns.color_palette("deep",n_colors=len(cols))

    # Create a grid of plots (3x3 grid to accommodate all the columns)
    n_rows = 3
    n_cols = 3

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 12))

    # Flatten the axes array for easy iteration
    axes = axes.flatten()

    # Plot histograms for each integer column with different colors
    for i, col in enumerate(cols):
        sns.histplot(X[col], ax=axes[i], kde=False, color=palette[i], bins=30)
        axes[i].set_title(f'{col} Distribution', fontsize=14)
        axes[i].set_xlabel(col, fontsize=12)
        axes[i].set_ylabel('Frequency', fontsize=12)
        axes[i].grid(True)

    # Remove any unused subplots (if any)
    for j in range(i + 1, n_rows * n_cols):
        fig.delaxes(axes[j])

    # Adjust layout to prevent overlapping
    plt.tight_layout()

    # Show the plots
    plt.show()
hist_plot_cols(int_cols)
```



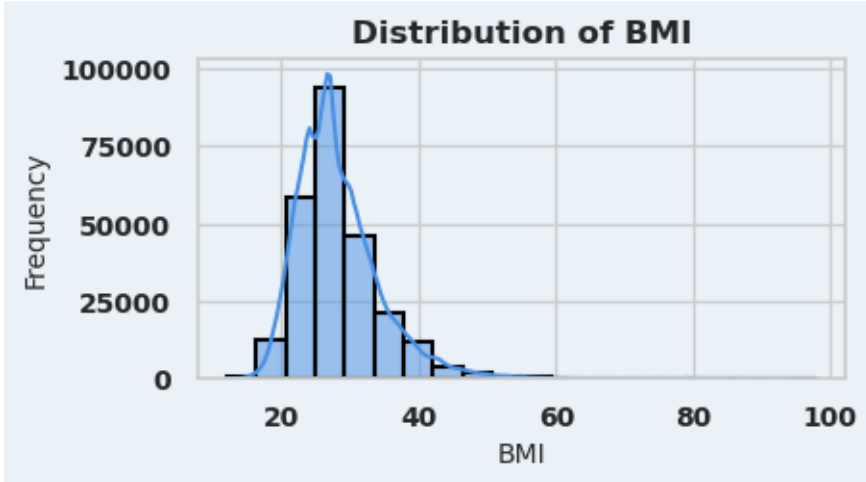

```
[ ]: do_rebins = True
```

BMI BMI has a skewed distribution, the number of people really overweight are probably outliers.

```
[ ]: display_variable_desc('BMI')
```

BMI Description:

Body Mass Index

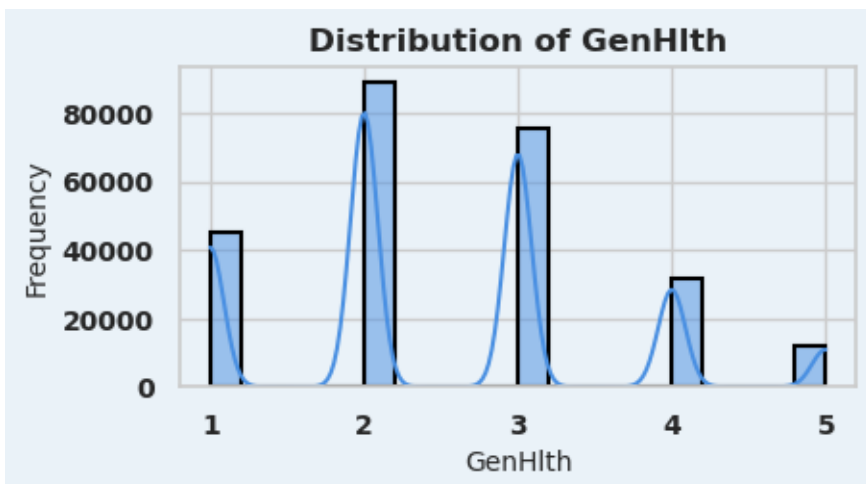


GenHlth Lets see what the encodings mean

```
[ ]: display_variable_desc('GenHlth')
```

GenHlth Description:

Would you say that in general your health is: scale 1-5 1 = excellent 2 = very good 3 = good 4 = fair 5 = poor



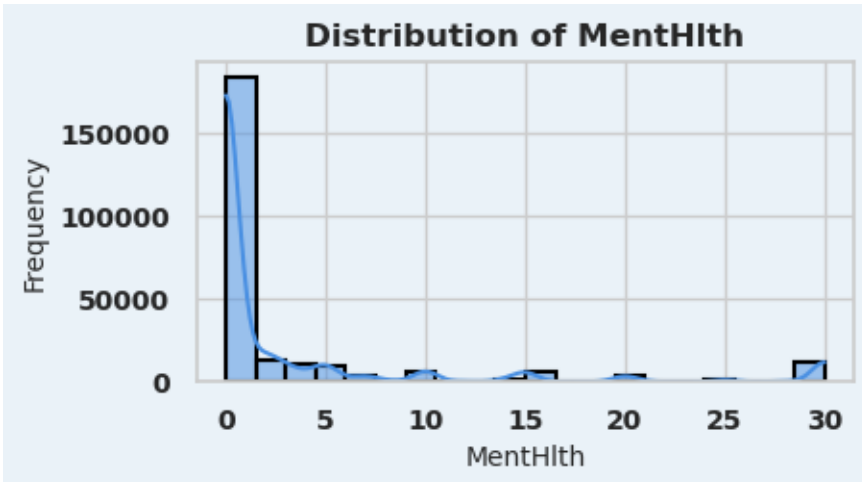
It seems most people rate their health very good or good. Since this is an encoding the distribution is more of a classification.

MentHlth

[ ]:
```
display_variable_desc('MentHlth')
X['MentHlth'].value_counts(normalize=True).head()
```

MentHlth Description:

Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good? scale 1-30 days



[ ]:
```
MentHlth
0     0.661574
2     0.055650
30    0.052999
5     0.039090
1     0.036423
Name: proportion, dtype: float64
```

66% of people say 0 days of their mental health was not good in the last month. So a pretty big imbalance. Consider binning since there's so few people in each bin other than 0.

Bin MentHlth

[ ]:
```
if do_rebins:
    # Create the bins and labels
    bins = [-1, 0, 7, 30]  # -1 to include 0 in its own bin
    labels = [1, 2, 3]  # 1 for 0, then 2, 3, 4 for the other bins

    # Apply the binning to the MentHlth column
    X['MentHlth_binned'] = pd.cut(X['MentHlth'], bins=bins, labels=labels,␣
 ↪include_lowest=True)

    # Display the resulting binned column
```

```
    print(X[['MentHlth', 'MentHlth_binned']].head())
```

```
   MentHlth MentHlth_binned
0        18               3
1         0               1
2        30               3
3         0               1
4         3               2
```

[ ]:
```
if do_rebins:
    X['MentHlth_binned'].value_counts(normalize=True).head()
```

I have binned it into 1: people who have zero mental health issues 2: people who reported 1-7 days with mental health issues in a month 3: people who reported 8-30 days with mental health issues in a month

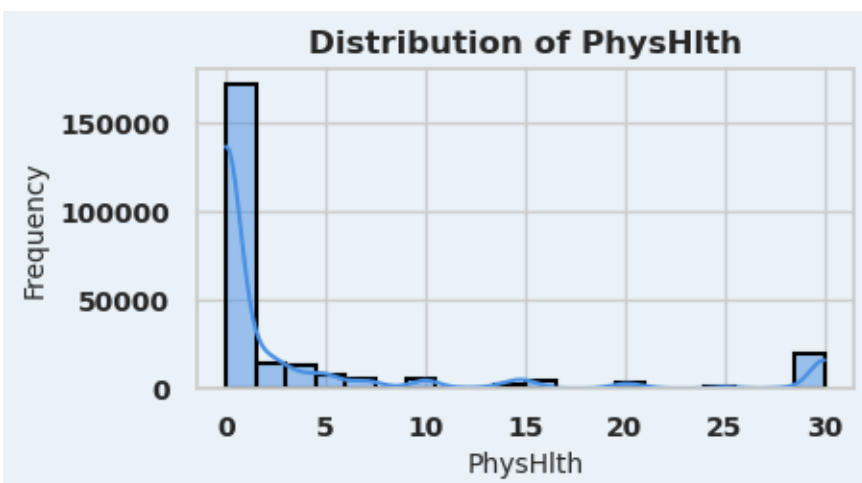
The reason for this is two fold. One is class balance, the other is, it's basically very healthy, struggling, and miserable.

PhysHlth

[ ]:
```
display_variable_desc('PhysHlth')
X['PhysHlth'].value_counts(normalize=True).head()
```

PhysHlth Description:

Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? scale 1-30 days



[ ]:
```
PhysHlth
0     0.592599
30    0.085030
2     0.063495
```

```
1     0.048533
3     0.036989
Name: proportion, dtype: float64
```

Bin PhysHlth

[ ]:
```
if do_rebins:
    # Create the bins and labels
    bins = [-1, 0, 7, 30]  # -1 to include 0 in its own bin
    labels = [1, 2, 3]  # 1 for 0, then 2, 3, 4 for the other bins

    # Apply the binning to the PhysHlth column
    X['PhysHlth_binned'] = pd.cut(X['PhysHlth'], bins=bins, labels=labels,␣
 ↪include_lowest=True)

    # Display the resulting binned column
    print(X[['PhysHlth', 'PhysHlth_binned']].head())
```

```
   PhysHlth PhysHlth_binned
0        15               3
1         0               1
2        30               3
3         0               1
4         0               1
```

[ ]:
```
if do_rebins:
    X['PhysHlth_binned'].value_counts(normalize=True).head()
```

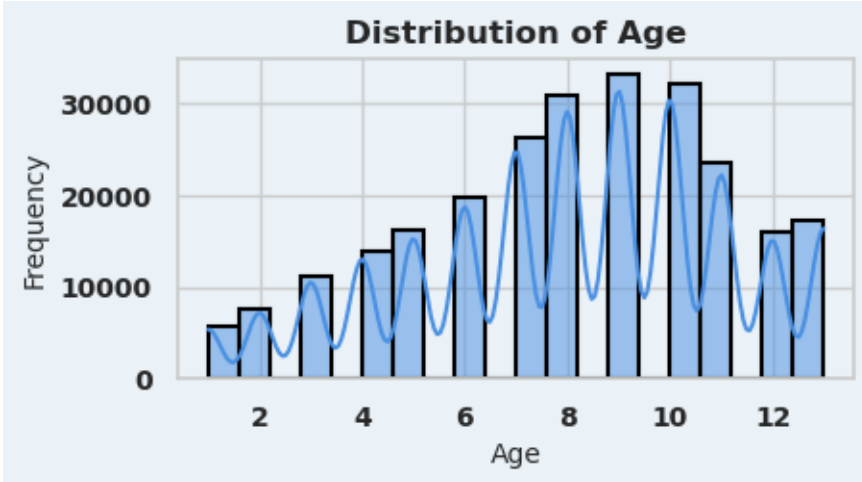
Same reasoning as with Mental Health

Age

[ ]:
```
display_variable_desc('Age')
X['Age'].value_counts(normalize=True).head()
```

Age Description:

13-level age category (_AGEG5YR see codebook) 1 = 18-24 9 = 60-64 13 = 80 or older



[ ]:
```
Age
9     0.129004
10    0.126051
8     0.118807
7     0.100892
11    0.095464
Name: proportion, dtype: float64
```

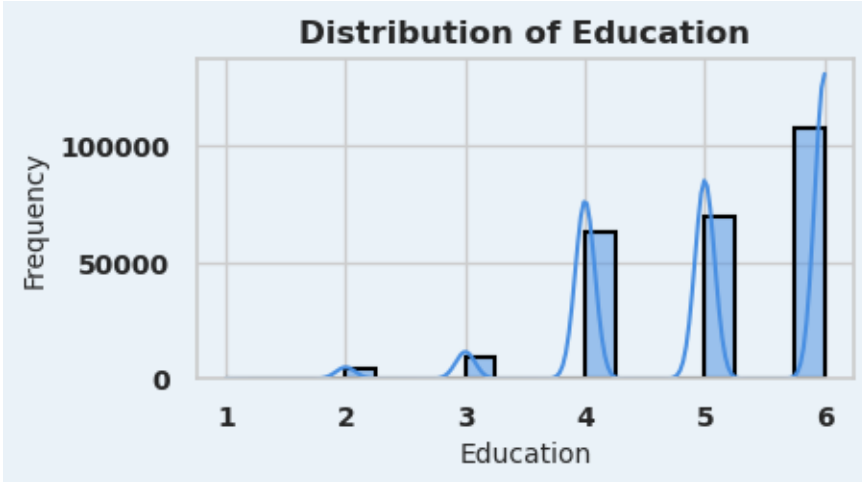
Age looks great for our needs. It is slightly not normal but it's ok.

Education

[ ]:
```
display_variable_desc('Education')
X['Education'].value_counts(normalize=True).head()
```

Education Description:

Education level (EDUCA see codebook) scale 1-6 1 = Never attended school or only kindergarten 2 = Grades 1 through 8 (Elementary) 3 = Grades 9 through 11 (Some high school) 4 = Grade 12 or GED (High school graduate) 5 = College 1 year to 3 years (Some college or technical school) 6 = College 4 years or more (College graduate)



[ ]:
```
Education
6    0.382611
5    0.290078
4    0.267283
3    0.041539
2    0.017726
Name: proportion, dtype: float64
```

1 is an empty class we can just get rid of it

2 and 3 are super small consider binning

4,5, and 6 are fine but we can make this a better encoding by rebinning

Rebinning Education

[ ]:
```
if do_rebins:
    education_mapping = {
        1: 1,  # Combine these three into a single bin
        2: 1,
        3: 1,
        4: 2,  # Keep the original class for 4, 5, and 6, but reassign them to␣
 ↪new values
        5: 3,
        6: 4
    }

    # Apply the mapping to the Education column
    X['Education_binned'] = X['Education'].map(education_mapping)

    # Display the resulting binned column
    print(X[['Education', 'Education_binned']].head())
```

```
   Education  Education_binned
0          4                 2
1          6                 4
2          4                 2
3          3                 1
4          5                 3
```

That's better!

[ ]:
```python
if do_rebins:
    display(X['Education_binned'].value_counts(normalize=True).head())
    plot_single_feature('Education_binned')
```

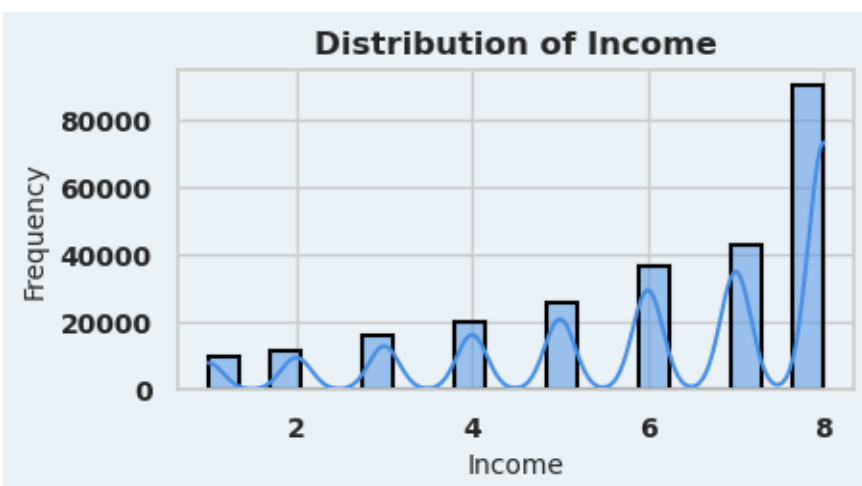
```
Education_binned
4    0.382611
3    0.290078
2    0.267283
1    0.060029
Name: proportion, dtype: float64
```

Income

[ ]:
```python
display_variable_desc('Income')
X['Income'].value_counts(normalize=True)
```

Income Description:

Income scale (INCOME2 see codebook) scale 1-8 1 = less than $10,000 5 = less than $35,000 8 = $75,000 or more



[ ]:
```
Income
8    0.310046
7    0.174921
```

```
6    0.152588
5    0.110782
4    0.087377
3    0.069769
2    0.051560
1    0.042956
Name: proportion, dtype: float64
```

More imbalances and less meaningful categories. Lets fix it.

Rebin Income

[ ]:
```
if do_rebins:
    income_mapping = {
        1: 1,
        2: 1,
        3: 1,
        4: 1,
        5: 2,
        6: 2,
        7: 3,
        8: 4,
    }

    # Apply the mapping to the income column
    X['Income_binned'] = X['Income'].map(income_mapping)

    # Display the resulting binned column
    print(X[['Income', 'Income_binned']].head())
```

```
   Income  Income_binned
0       3              1
1       1              1
2       8              4
3       6              2
4       4              1
```

Eval Rebin

[ ]:
```
if do_rebins:
    X['Income_binned'].value_counts(normalize=True)
```

This seems a better grouping, we can now categorize as: * 1: poverty * 2: poor ($35,000) * 3: mid income * 4: high income ($75,000+)

7.5.1 Fix Datatypes

[ ]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 227908 entries, 0 to 253679
Data columns (total 25 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   HighBP                227908 non-null  bool
 1   HighChol              227908 non-null  bool
 2   CholCheck             227908 non-null  bool
 3   BMI                   227908 non-null  uint8
 4   Smoker                227908 non-null  bool
 5   Stroke                227908 non-null  bool
 6   HeartDiseaseorAttack  227908 non-null  bool
 7   PhysActivity          227908 non-null  bool
 8   Fruits                227908 non-null  bool
 9   Veggies               227908 non-null  bool
 10  HvyAlcoholConsump     227908 non-null  bool
 11  AnyHealthcare         227908 non-null  bool
 12  NoDocbcCost           227908 non-null  bool
 13  GenHlth               227908 non-null  uint8
 14  MentHlth              227908 non-null  uint8
 15  PhysHlth              227908 non-null  uint8
 16  DiffWalk              227908 non-null  bool
 17  Sex                   227908 non-null  bool
 18  Age                   227908 non-null  uint8
 19  Education             227908 non-null  uint8
 20  Income                227908 non-null  uint8
 21  MentHlth_binned       227908 non-null  category
 22  PhysHlth_binned       227908 non-null  category
 23  Education_binned      227908 non-null  int64
 24  Income_binned         227908 non-null  int64
dtypes: bool(14), category(2), int64(2), uint8(7)
memory usage: 10.2 MB
```

[ ]:
```python
if do_rebins:
    for col in X.columns:
        if "_binned" in col:
            X[col] = X[col].astype('uint8')
    X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 227908 entries, 0 to 253679
Data columns (total 25 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   HighBP                227908 non-null  bool
 1   HighChol              227908 non-null  bool
 2   CholCheck             227908 non-null  bool
 3   BMI                   227908 non-null  uint8
```

```
 4   Smoker                227908 non-null  bool
 5   Stroke                227908 non-null  bool
 6   HeartDiseaseorAttack  227908 non-null  bool
 7   PhysActivity          227908 non-null  bool
 8   Fruits                227908 non-null  bool
 9   Veggies               227908 non-null  bool
 10  HvyAlcoholConsump     227908 non-null  bool
 11  AnyHealthcare         227908 non-null  bool
 12  NoDocbcCost           227908 non-null  bool
 13  GenHlth               227908 non-null  uint8
 14  MentHlth              227908 non-null  uint8
 15  PhysHlth              227908 non-null  uint8
 16  DiffWalk              227908 non-null  bool
 17  Sex                   227908 non-null  bool
 18  Age                   227908 non-null  uint8
 19  Education             227908 non-null  uint8
 20  Income                227908 non-null  uint8
 21  MentHlth_binned       227908 non-null  uint8
 22  PhysHlth_binned       227908 non-null  uint8
 23  Education_binned      227908 non-null  uint8
 24  Income_binned         227908 non-null  uint8
dtypes: bool(14), uint8(11)
memory usage: 7.2 MB
```

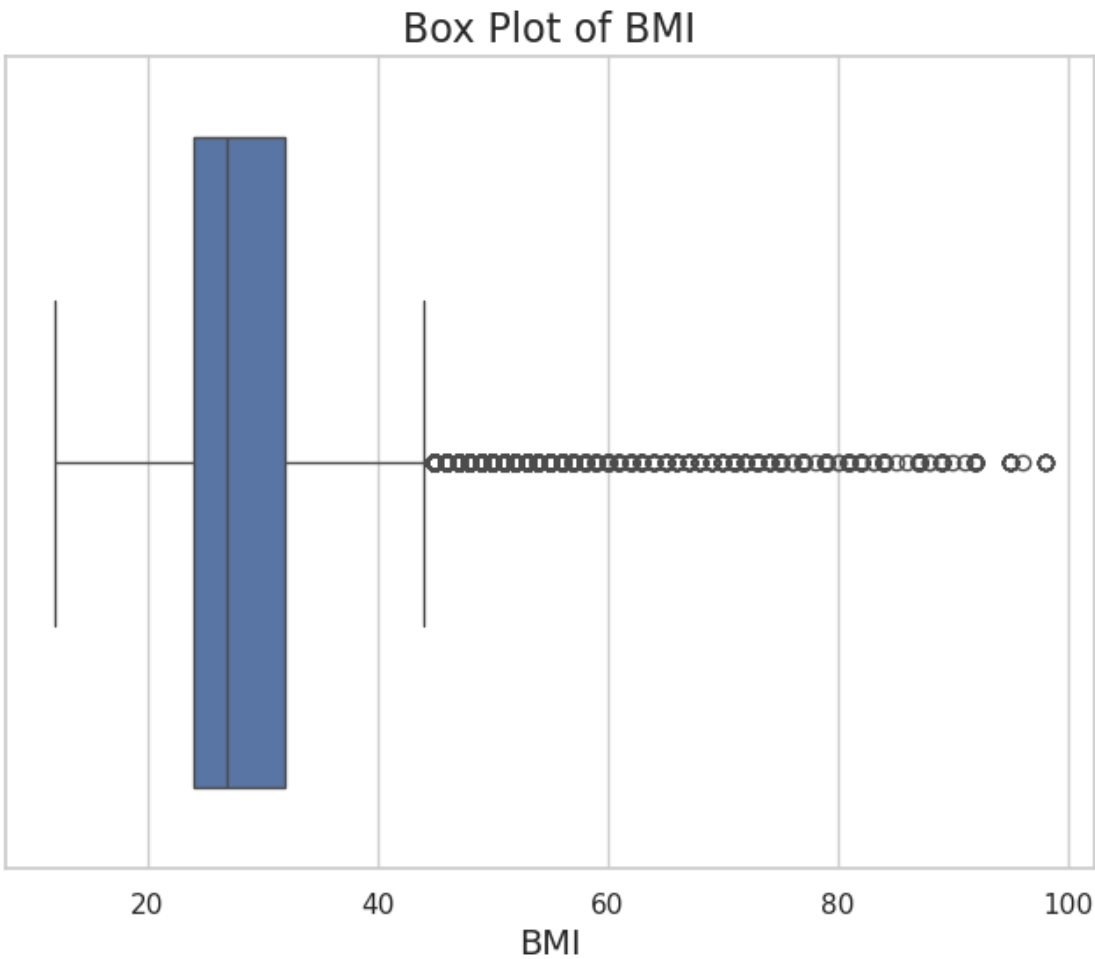
7.5.2 Lists of Feature Changes

[ ]:
```python
# List of features that were re-binned
rebinned_features = [
    'PhysHlth_binned',
    'MentHlth_binned',
    'Income_binned',
    'Education_binned'
]

# List of original features that were not re-binned
unrebinned_features = [
    'PhysHlth',
    'MentHlth',
    'Income',
    'Education'
]
```

7.6 Outliers

BMI is the only column I'm worried about outliers lets check it

[ ]:
```
sns.set(style="whitegrid")
palette = sns.color_palette("deep")

# Create the box plot for the 'BMI' column
plt.figure(figsize=(8, 6))
sns.boxplot(x=X['BMI'], color=palette[0])

# Add title and labels for clarity
plt.title('Box Plot of BMI', fontsize=16)
plt.xlabel('BMI', fontsize=14)

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



Well, I know from domain knowledge that heavy weights are actually pretty important for predicting diabetes so they must stay.

7.7 Binary Class Imbalances

7.7.1 Extract Labels for True/False Utility Function

[ ]:
```
# Function to extract labels for binary columns
def extract_labels_for_bin_cols(bin_cols):
    labels = {}
    for col in bin_cols:
        description = variables[variables['name'] == col]['description'].
↪values[0]
        meanings = re.findall(r'(\d) = ([^\d]+?)(?= \d|$)', description)
        false_meaning = meanings[0][1].strip() if len(meanings) > 0 else "False"
        true_meaning = meanings[1][1].strip() if len(meanings) > 1 else "True"
        labels[col] = (false_meaning, true_meaning)
    return labels
```

[ ]:
```
binary_labels = extract_labels_for_bin_cols(bin_cols)
```

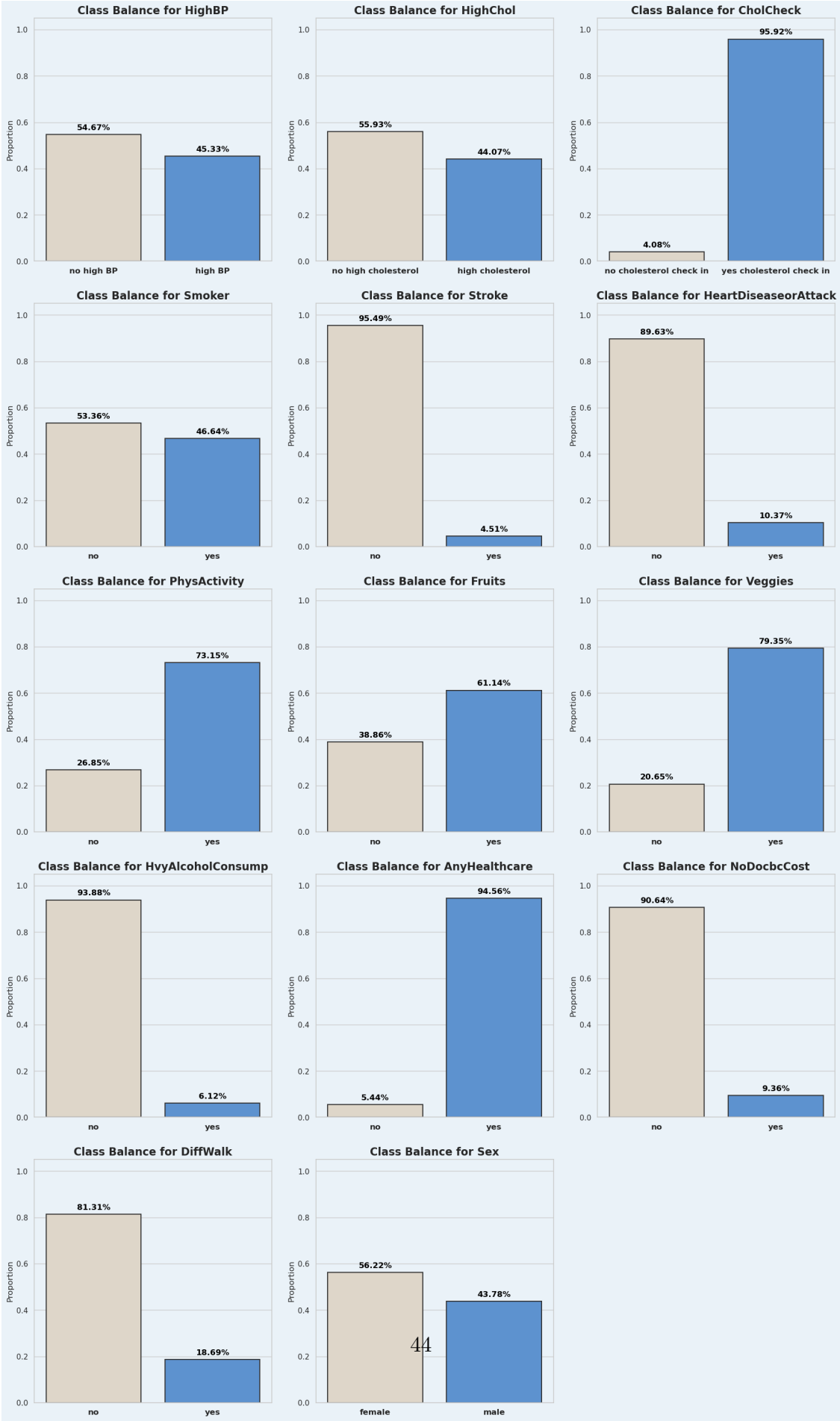
7.7.2 Plot Binary Class Balances

[ ]:
```
def plot_binary_class_balances(bin_cols,dfX=X):
    # Set the style with a more widely available font and a light blue␣
↪background
    sns.set(style="whitegrid", font="DejaVu Sans", rc={"axes.facecolor":␣
↪"#EAF2F8", "figure.facecolor": "#EAF2F8"})
    custom_palette = ["#E1D7C6","#4A90E2"]  # Medical light blue and gray

    # Calculate the number of rows needed based on the number of columns
    n_cols = 3  # Three plots per row
    n_rows = (len(bin_cols) + n_cols - 1) // n_cols  # Ceiling division to␣
↪determine rows

    # Create a figure to hold all subplots
    plt.figure(figsize=(18, n_rows * 6))

    # Iterate through each binary column
    for i, col in enumerate(bin_cols):
        plt.subplot(n_rows, n_cols, i + 1)  # Create a subplot for each binary␣
↪column

        # Calculate the proportion of each class
        value_counts = dfX[col].value_counts(normalize=True).sort_index()

        # Get the actual labels from the global binary_labels variable
        false_label, true_label = binary_labels[col]
        labels = [false_label, true_label]
```

```
        # Create a DataFrame for plotting
        plot_data = pd.DataFrame({
            'Label': labels,
            'Proportion': value_counts.values,
            'Category': labels
        })

        # Create a bar plot with the actual labels using hue and palette
        sns.barplot(x='Label', y='Proportion', hue='Category', data=plot_data,
                    palette=custom_palette, dodge=False, edgecolor=".2",␣
↪linewidth=1.5)

        # Add title and labels
        plt.title(f'Class Balance for {col}', fontsize=16, fontweight='bold')
        plt.xlabel('', fontsize=12)
        plt.ylabel('Proportion', fontsize=12)
        plt.ylim(0, 1.05)  # Set y-axis to go slightly above 1 to prevent␣
↪percentage overlap
        # Adjust x-tick labels for size and weight
        plt.xticks(fontsize=12, fontweight='bold')

        # Display the percentages on the bars with padding to avoid bleeding
        for index, value in enumerate(value_counts.values):
            plt.text(index, value + 0.02, f'{value:.2%}', ha='center',␣
↪fontsize=12, color='black', fontweight='bold')

    # Adjust layout to prevent overlap
    plt.tight_layout(pad=2.0)  # Add padding around the subplots

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

7.7.3 Binary Class Balances

[ ]:
```
plot_binary_class_balances(bin_cols)
```


Class Balance for HighBP
54.67%
45.33%
no high BP
high BP
Class Balance for HighChol
55.93%
44.07%
no high cholesterol
high cholesterol
Class Balance for CholCheck
4.08%
95.92%
no cholesterol check in
yes cholesterol check in
Class Balance for Smoker
53.36%
46.64%
Class Balance for Stroke
95.49%
4.51%
Class Balance for HeartDiseaseorAttack
89.63%
10.37%
Class Balance for PhysActivity
26.85%
73.15%
Class Balance for Fruits
38.86%
61.14%
Class Balance for Veggies
20.65%
79.35%
Class Balance for HvyAlcoholConsump
93.88%
6.12%
Class Balance for AnyHealthcare
5.44%
94.56%
Class Balance for NoDocbcCost
90.64%
9.36%
Class Balance for DiffWalk
81.31%
18.69%
Class Balance for Sex
56.22%
43.78%
female
male
no
yes
Proportion

7.7.4 Analysis of Class Balances

HighBP HighBP shows a fairly even distribution.

HighChol HighChol shows a fairly even distribution.

CholCheck CholCheck is highly imbalanced, with 95.92% having had a cholesterol check. Important but really hard to balance.

Smoker Smoker is fairly balanced (53.36% no, 46.64% yes).

Stroke Stroke is extremely imbalanced, with only 4.51% reporting a stroke. It is however really useful info. Can try a drop possibly.

HeartDiseaseorAttack HeartDiseaseorAttack has a notable imbalance (89.63% no, 10.37% yes), but is a critical predictor.

PhysActivity PhysActivity shows a slight imbalance with 73.15% reporting physical activity. However it is critical.

Fruits Fruits consumption is somewhat imbalanced, with 61.14% reporting daily consumption. Keep as is.

Veggies Veggies consumption is more imbalanced (79.35% yes, 20.65% no). Still critical to keep.

HvyAlcoholConsump Heavy alcohol consumption is highly imbalanced, with only 6.12% reporting it. It is however important to keep.

AnyHealthcare AnyHealthcare is imbalanced, with 94.56% having healthcare coverage. This is a really hard one to call, it could be an indicator for sure though.

NoDocbcCost NoDocbcCost indicates a strong imbalance, with only 9.36% reporting inability to see a doctor due to cost. This is still maybe important but it and AnyHealthCare may be candidates for dropping if model performance is in question.

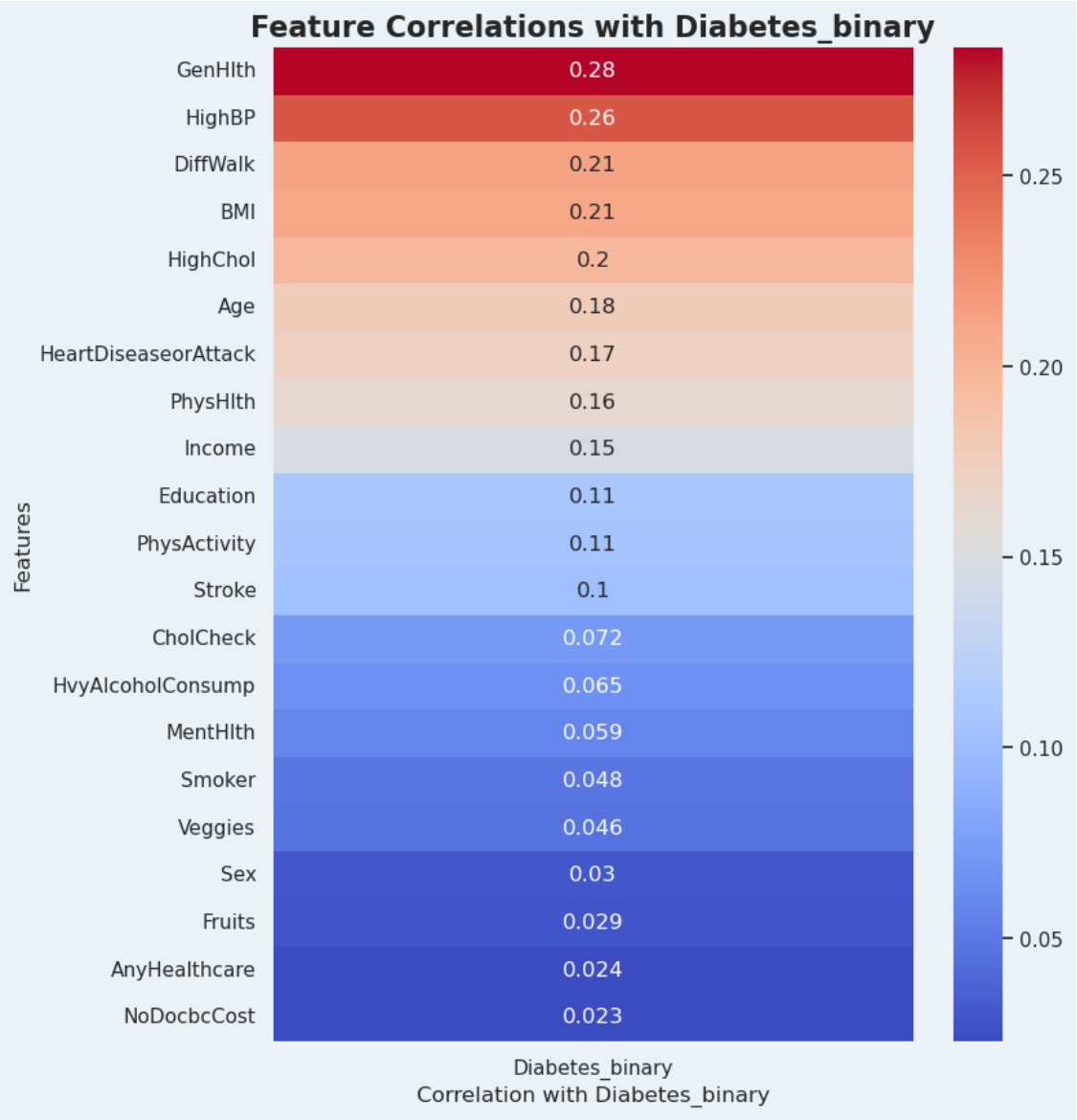
DiffWalk DiffWalk shows an imbalance where 18.69% report difficulty walking. This is an important feature.

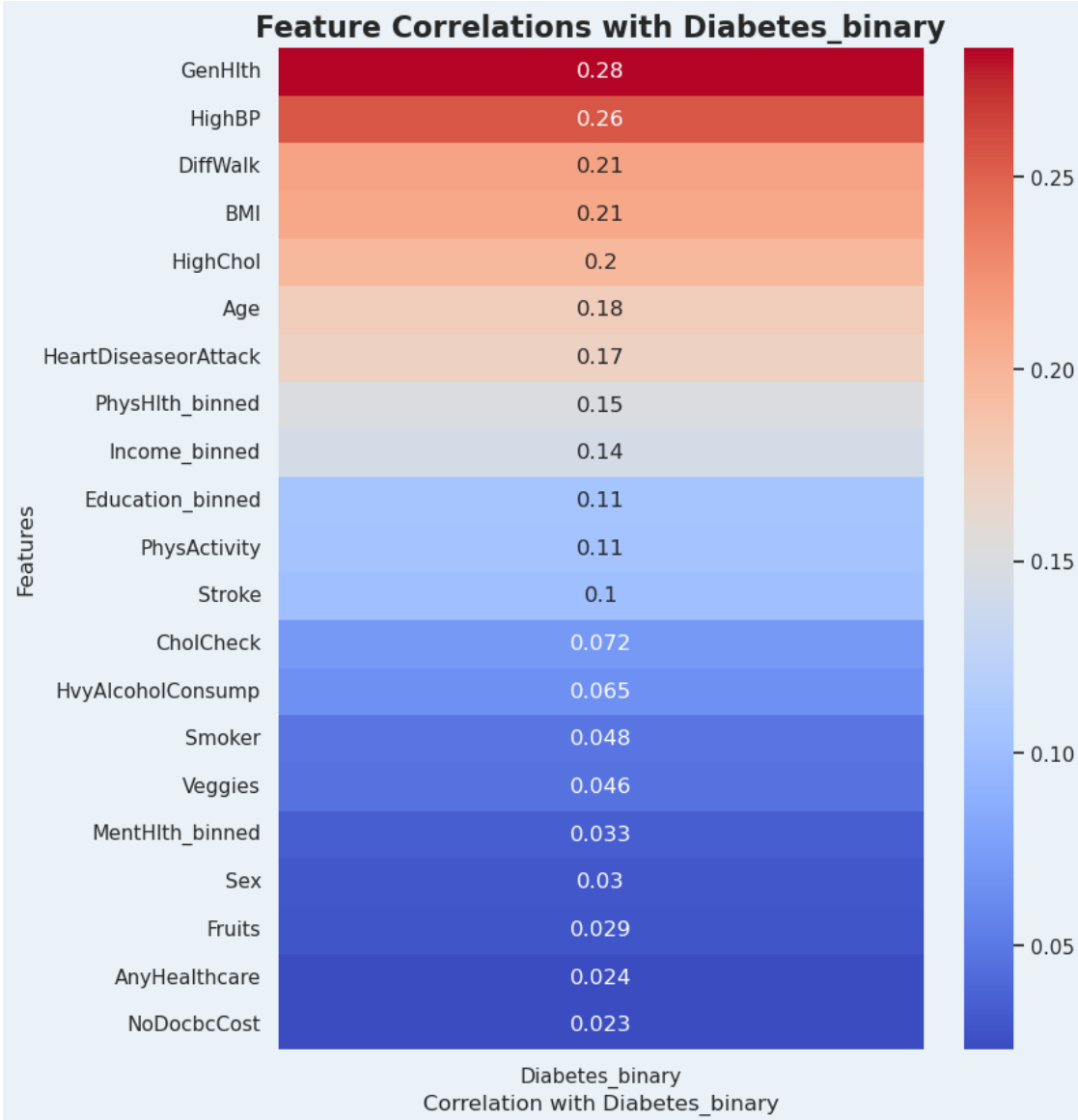
Sex Sex is relatively balanced (56.22% female, 43.78% male), we'll have to see how much of a predictor it is, some diseases are more common in one sex or the other.

7.8 Feature Importance and Correlations

7.8.1 Target to Features Correlations

[ ]:
```
def feature_to_target_correlation_heatmap(X, y):
    # Combine X and y into a single DataFrame
    df = pd.concat([X, y], axis=1)

    # Calculate the correlation matrix
    corr_matrix = df.corr()

    # Select the correlation of features with the target variable␣
↪'Diabetes_binary'
    target_corr = corr_matrix['Diabetes_binary']

    # Drop the target itself from the correlation list
    target_corr = target_corr.drop(index='Diabetes_binary')

    # Sort the correlations by absolute value
    target_corr_sorted = target_corr.abs().sort_values(ascending=False)

    # Plot the heatmap
    plt.figure(figsize=(8, 10))
    sns.heatmap(target_corr_sorted.to_frame(), annot=True, cmap='coolwarm',␣
↪cbar=True)
    plt.title('Feature Correlations with Diabetes_binary', fontsize=16,␣
↪fontweight='bold')
    plt.xlabel('Correlation with Diabetes_binary')
    plt.ylabel('Features')
    plt.show()

# Assuming X is your features DataFrame and y is your target variable
feature_to_target_correlation_heatmap(X.drop(columns=rebinned_features), y)
feature_to_target_correlation_heatmap(X.drop(columns=unrebinned_features), y)
```

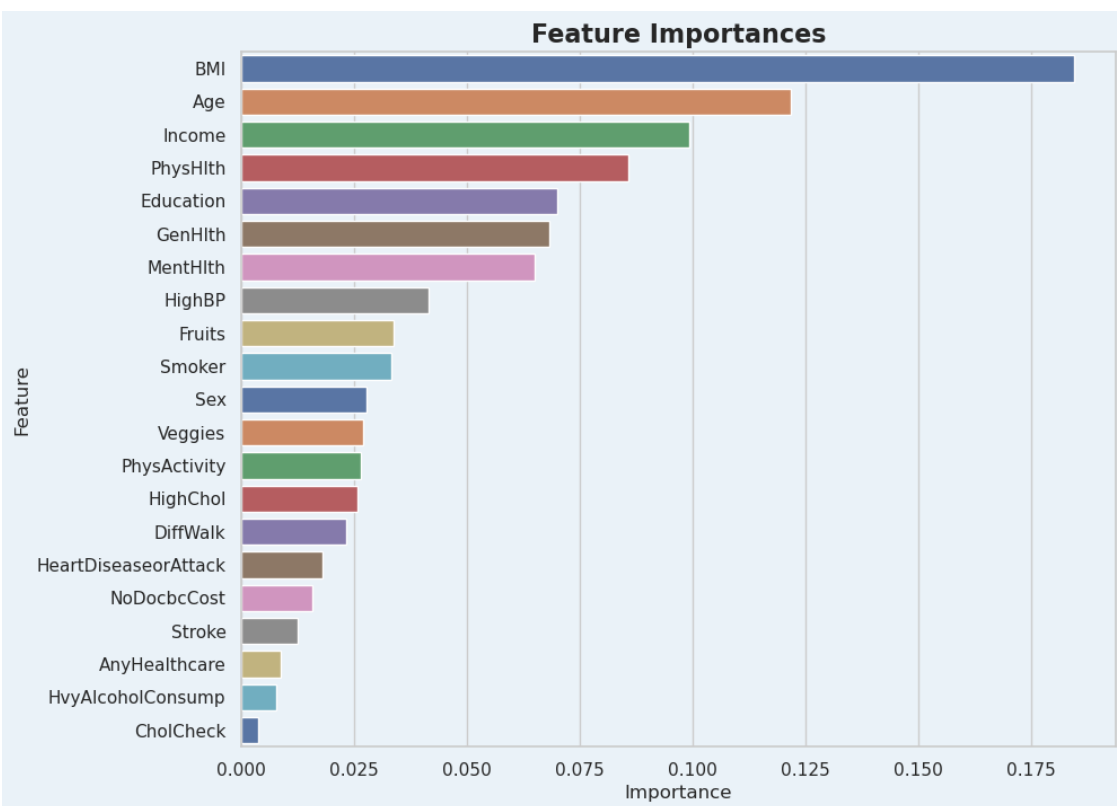

Feature Correlations with Diabetes_binary
GenHlth
HighBP
DiffWalk
BMI
HighChol
Age
HeartDiseaseorAttack
PhysHlth
Income
Education
PhysActivity
Stroke
CholCheck
HvyAlcoholConsump
MentHlth
Smoker
Veggies
Sex
Fruits
AnyHealthcare
NoDocbcCost
0.28
0.26
0.21
0.21
0.2
0.18
0.17
0.16
0.15
0.11
0.11
0.1
0.072
0.065
0.059
0.048
0.046
0.03
0.029
0.024
0.023
Features
Diabetes_binary
Correlation with Diabetes_binary
0.25
0.20
0.15
0.10
0.05

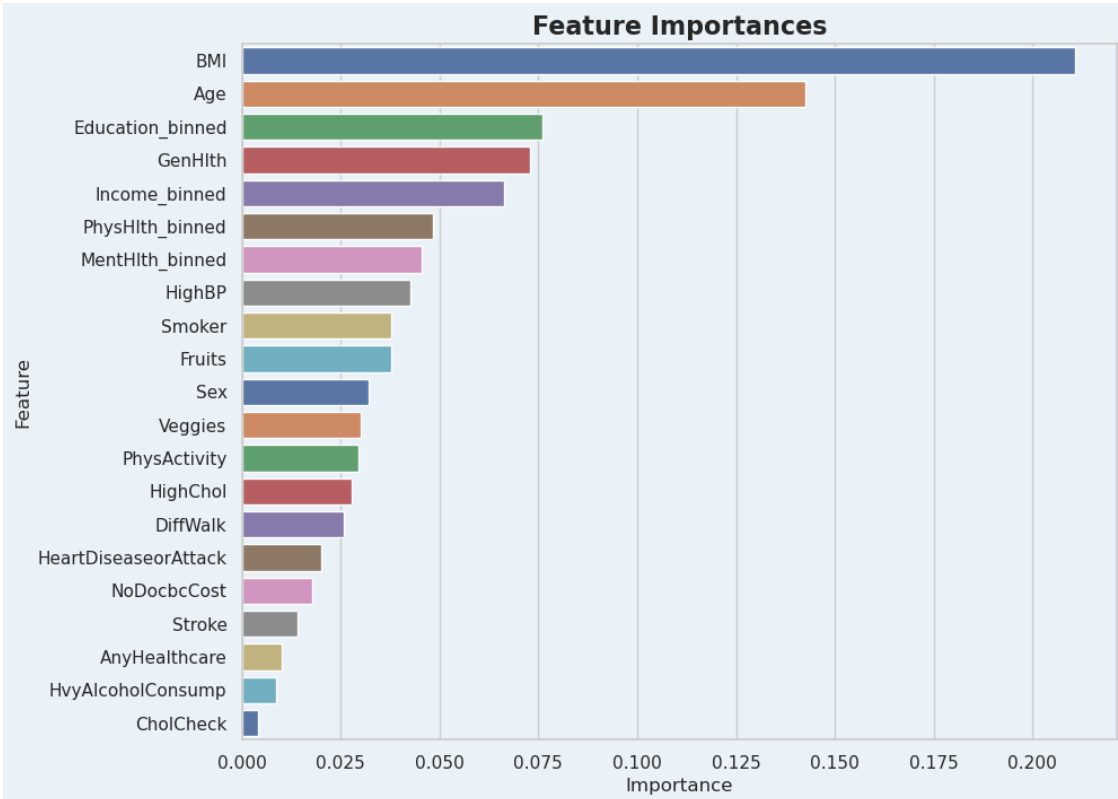


7.8.2 Feature Importance

[ ]:
```
def calculate_feature_importance(X, y):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

    # Initialize and train the model
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
```

```
    # Calculate feature importances
    importances = model.feature_importances_
    feature_importance_df = pd.DataFrame({
        'Feature': X.columns,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False)

    # Plot the feature importances
    plt.figure(figsize=(10, 8))
    sns.barplot(x='Importance', y='Feature', data=feature_importance_df,␣
 ↪palette="deep", legend=False)

    plt.title('Feature Importances', fontsize=16, fontweight='bold')
    plt.show()

    return feature_importance_df

# Calculate and display the feature importances
orig_feature_importances = calculate_feature_importance(X.
 ↪drop(columns=rebinned_features), y)
feature_importances = calculate_feature_importance(X.
 ↪drop(columns=unrebinned_features), y)
```





7.8.3 Drop rebinned

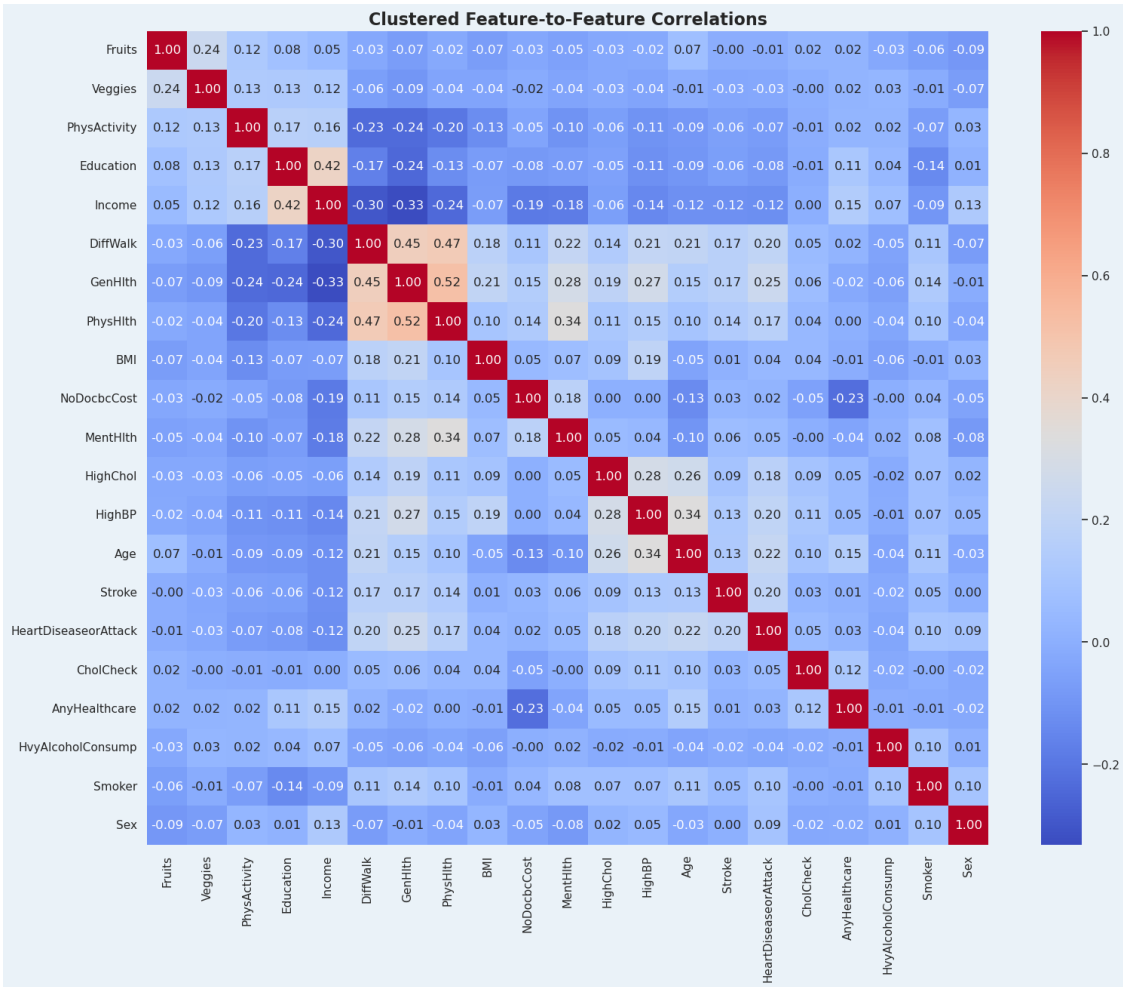
[ ]:
```
X.drop(columns=rebinned_features,inplace=True,errors='ignore')
X.columns.to_list()
```

[ ]:
```
['HighBP',
 'HighChol',
 'CholCheck',
 'BMI',
 'Smoker',
 'Stroke',
 'HeartDiseaseorAttack',
 'PhysActivity',
 'Fruits',
 'Veggies',
 'HvyAlcoholConsump',
 'AnyHealthcare',
 'NoDocbcCost',
 'GenHlth',
 'MentHlth',
```

```
 'PhysHlth',
 'DiffWalk',
 'Sex',
 'Age',
 'Education',
 'Income']
```

7.8.4 Feature to Feature Correlations

[ ]:
```
def feature_to_feature_correlations(X):
    # Calculate the correlation matrix
    corr_matrix = X.corr()

    # Perform hierarchical clustering on the correlation matrix
    linkage_matrix = linkage(corr_matrix, method='ward')  # You can also use␣
 ↪other methods like 'average'

    # Get the order of the features after clustering
    ordered_indices = leaves_list(linkage_matrix)
    ordered_columns = corr_matrix.columns[ordered_indices]

    # Reorder the correlation matrix
    ordered_corr_matrix = corr_matrix.loc[ordered_columns, ordered_columns]

    # Use a heatmap to visualize the reordered correlations
    plt.figure(figsize=(18, 14))
    sns.heatmap(ordered_corr_matrix, annot=True, fmt=".2f", cmap='coolwarm',␣
 ↪cbar=True)
    plt.title("Clustered Feature-to-Feature Correlations", fontsize=16,␣
 ↪fontweight='bold')
    plt.show()

    return ordered_corr_matrix

# Calculate and display the feature to feature correlations
feature_correlations = feature_to_feature_correlations(X)
```



7.9 Dropping Features

7.9.1 Heavy Alcohol Consumption

This feature seems a great one to keep but in my final model outputs it was causing too much issues. It would lower the class 1 probablity when turned on, which isn't what broader research reports.

[ ]:
```
alcohol_masq = X['HvyAlcoholConsump'].copy()
alcohol_df = X[alcohol_masq]
alcohol_df['target'] = y.loc[alcohol_df.index].copy()
alcohol_and_diabetes = alcohol_df[alcohol_df['target']==1]
display(alcohol_and_diabetes)
display(len(alcohol_and_diabetes)/ len(X))
```

```
        HighBP  HighChol  CholCheck  BMI  Smoker  Stroke  \
92        True      True       True   29    True   False
144       True      True       True   29    True   False
```

```
420       True      True       True   23    True    True
454      False     False       True   31    True   False
650       True      True       True   28    True   False
…         …        …         …  …     …      …
252614    True      True       True   23    True   False
253267    True     False       True   33    True   False
253377    True     False       True   27    True   False
253573    True     False       True   36    True   False
253603    True     False       True   37   False   False

        HeartDiseaseorAttack  PhysActivity  Fruits  Veggies  …  NoDocbcCost  \
92                     False          True    True     True  …        False
144                    False          True   False     True  …        False
420                    False          True   False    False  …        False
454                    False          True    True     True  …        False
650                     True         False   False     True  …        False
…                      …           …     …      …  …          …
252614                 False          True   False     True  …        False
253267                 False         False    True    False  …         True
253377                  True          True   False     True  …        False
253573                 False          True    True     True  …        False
253603                 False         False    True     True  …        False

        GenHlth  MentHlth  PhysHlth  DiffWalk    Sex  Age  Education  Income  \
92            3         0         0     False   True   11          5       7
144           3         0         0     False   True    8          5       8
420           4         2         6      True   True    9          4       5
454           2         0         0     False  False   10          6       8
650           3         0        30     False   True   10          5       6
…          …        …        …        …     …  …        …     …
252614        3         0         0     False   True   12          5       1
253267        3         0         0     False   True    8          4       1
253377        4         0         0     False   True    7          5       4
253573        3         0         0     False   True   10          6       8
253603        2         0         3      True   True   11          6       7

        target
92           1
144          1
420          1
454          1
650          1
…         …
252614       1
253267       1
253377       1
253573       1
253603       1
```

```
[828 rows x 22 columns]
```

```
0.003633044912859575
```

Because only .3% of respondents reported both diabetes and heavy drinking, it is making the association that diabetes and heavy drinking are not related.

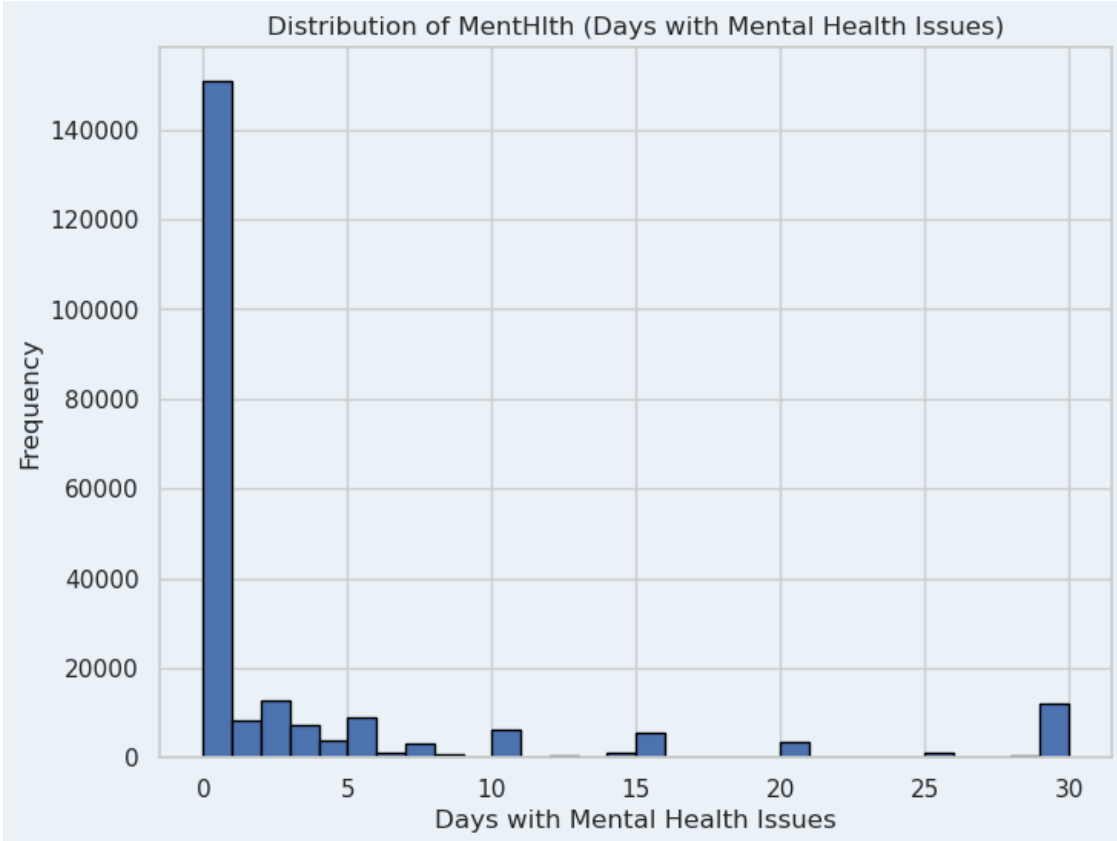
Due to insufficient data this is being dropped. In different contexts or with more co-occurence this feature could be highly valuable.

[ ]:
```
X.drop(columns=['HvyAlcoholConsump'],inplace=True)
```

7.9.2 Mental Health

Almost all repondants said 0 bad mentalh health days in the last 30 days. This leads to sparsity and the feature also isn't really having much impact, or worse, it will have the opposite if slight impact that one would expect. Bad mental health lowering your chance of diabetes extremely slightly is not really a conclusion that makes sense or is supported. If the patterns were stronger and this feature wasn't so sparse it'd be a different story. We could rely on its pattern more. As is, it is better to drop it.

[ ]:
```
if 'MentHlth' in X:
    saved_menthlth = X['MentHlth'].copy()

# Plotting the histogram
plt.figure(figsize=(8, 6))
plt.hist(saved_menthlth, bins=30, edgecolor='black')
plt.title('Distribution of MentHlth (Days with Mental Health Issues)')
plt.xlabel('Days with Mental Health Issues')
plt.ylabel('Frequency')
plt.grid(True)

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



[ ]:
```
#if 'MentHlth' in X:
    #X.drop(columns='MentHlth',inplace=True,errors='ignore')
```

7.10 Interaction Effects

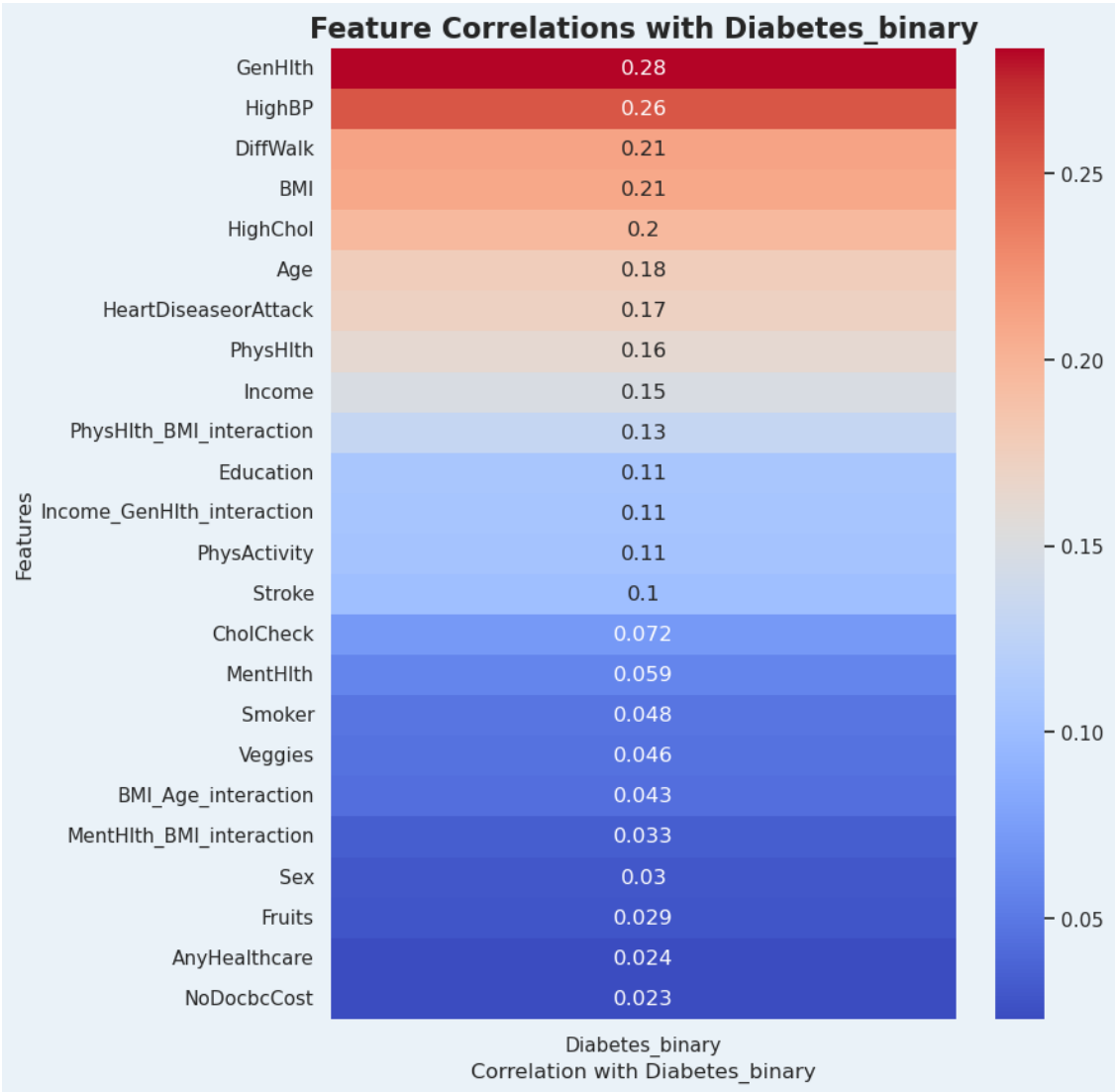
[ ]:
```
# Define the dictionary with features to interact
interaction_dict = {
    'BMI': 'Age',
    'Income': 'Education',
    #'GenHlth': 'Age',  # Too highly correlated with Age and GenHlth
    #'BMI': 'GenHlth',  # Too highly correlated with BMI and GenHtlh
    #'MentHlth': 'Age',  # Highly correlated with MentHlth
    'Income': 'GenHlth',
    'PhysHlth': 'BMI',  # Highly correlated with PhysHealth
    'MentHlth': 'BMI',  # Highly correlated with MentHlth
}

# Initialize an empty list to store the names of the interaction columns
interaction_cols = []
```

```python
# Loop through the dictionary to create interaction features
for key, value in interaction_dict.items():
    if key in X and value in X:
        interaction_name = f'{key}_{value}_interaction'
        X[interaction_name] = X[key] * X[value]
        interaction_cols.append(interaction_name)

# Display the list of created interaction columns
print(interaction_cols)
```

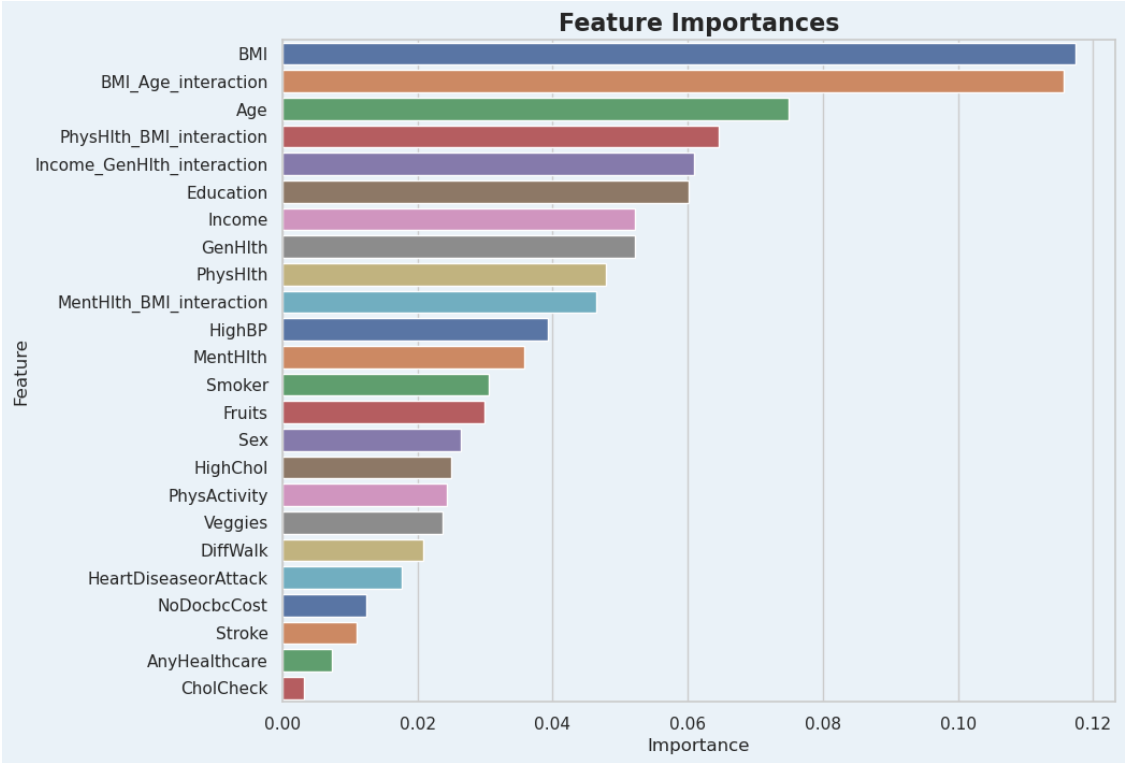
```
['BMI_Age_interaction', 'Income_GenHlth_interaction',
'PhysHlth_BMI_interaction', 'MentHlth_BMI_interaction']
```

7.10.1 Interaction Importances

[ ]:
```python
feature_to_target_correlation_heatmap(X,y)
```

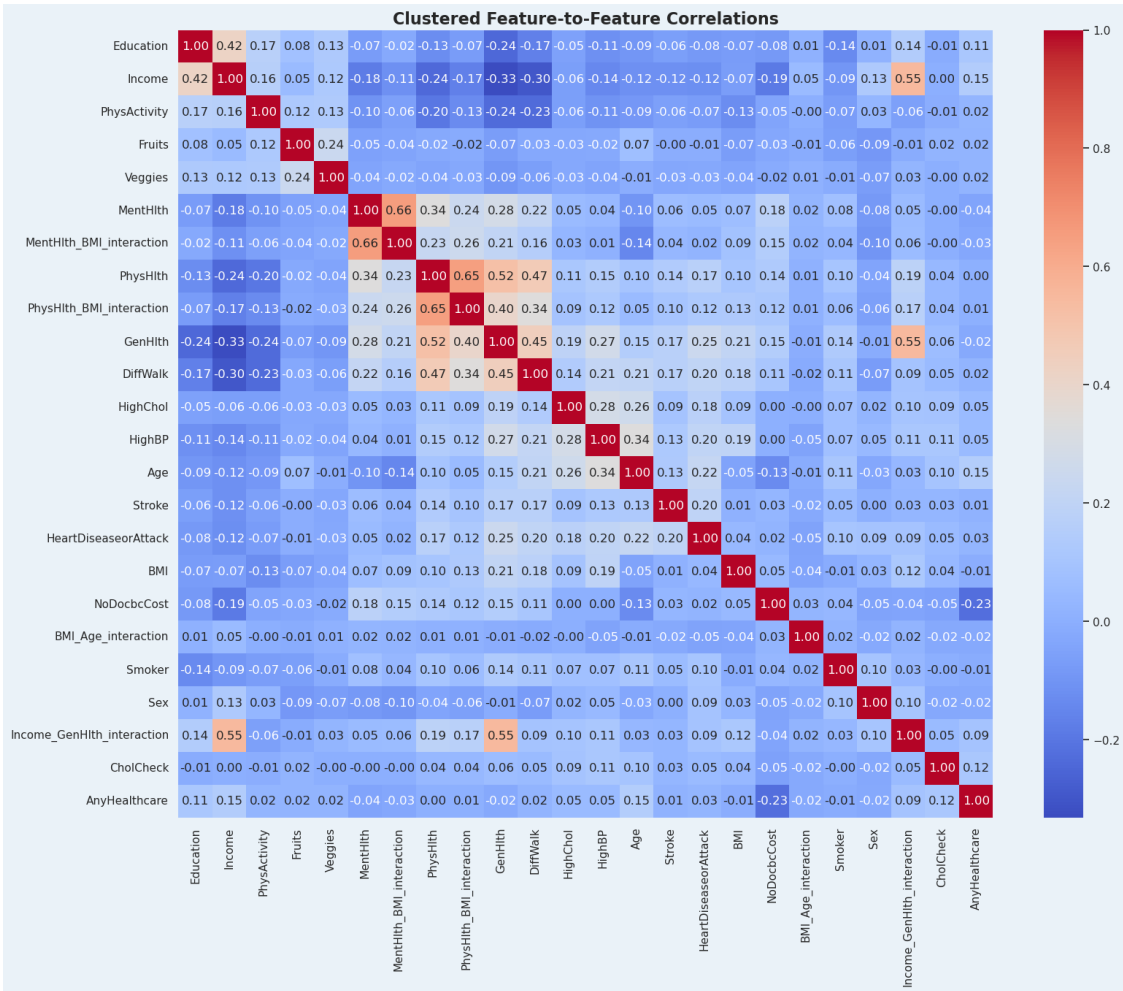


```
[ ]: int_feature_importances = calculate_feature_importance(X, y)
```



7.10.2 New Feature Correlation Matrix

```
[ ]: new_feature_correlations = feature_to_feature_correlations(X)
```



[ ]:
```
display(new_feature_correlations)
```

```
                          Education    Income  PhysActivity    Fruits  \
Education                  1.000000  0.417535      0.169558  0.083770
Income                     0.417535  1.000000      0.164141  0.049598
PhysActivity               0.169558  0.164141      1.000000  0.124056
Fruits                     0.083770  0.049598      0.124056  1.000000
Veggies                    0.130447  0.123646      0.134155  0.242411
MentHlth                  -0.074696 -0.184259     -0.104626 -0.051235
MentHlth_BMI_interaction  -0.022379 -0.114544     -0.056078 -0.040681
PhysHlth                  -0.126253 -0.239457     -0.198083 -0.023217
PhysHlth_BMI_interaction  -0.072603 -0.167852     -0.130892 -0.016050
GenHlth                   -0.244402 -0.331527     -0.237046 -0.070749
DiffWalk                  -0.168257 -0.297958     -0.234697 -0.028977
HighChol                  -0.051189 -0.063466     -0.064538 -0.026862
HighBP                    -0.114373 -0.141637     -0.105882 -0.020494
Age                       -0.094490 -0.117921     -0.089567  0.072509
```

Stroke	-0.063604	-0.116470	-0.058706	-0.003984
HeartDiseaseorAttack	-0.082023	-0.122363	-0.072775	-0.006564
BMI	-0.074248	-0.069088	-0.127653	-0.067231
NoDocbcCost	-0.082337	-0.186694	-0.045451	-0.031692
BMI_Age_interaction	0.011225	0.048654	-0.001034	-0.011742
Smoker	-0.135266	-0.094754	-0.066600	-0.061450
Sex	0.013806	0.129017	0.032356	-0.089216
Income_GenHlth_interaction	0.144111	0.547547	-0.057010	-0.012681
CholCheck	-0.010738	0.001062	-0.005264	0.017393
AnyHealthcare	0.110804	0.145466	0.023189	0.022122

	Veggies	MentHlth	MentHlth_BMI_interaction	\
Education	0.130447	-0.074696	-0.022379	
Income	0.123646	-0.184259	-0.114544	
PhysActivity	0.134155	-0.104626	-0.056078	
Fruits	0.242411	-0.051235	-0.040681	
Veggies	1.000000	-0.041049	-0.019924	
MentHlth	-0.041049	1.000000	0.655508	
MentHlth_BMI_interaction	-0.019924	0.655508	1.000000	
PhysHlth	-0.043859	0.339225	0.228810	
PhysHlth_BMI_interaction	-0.026937	0.244856	0.255989	
GenHlth	-0.093575	0.283591	0.207340	
DiffWalk	-0.062131	0.217725	0.155967	
HighChol	-0.028306	0.051411	0.030077	
HighBP	-0.044020	0.038591	0.008810	
Age	-0.005103	-0.102375	-0.144479	
Stroke	-0.032503	0.061403	0.035879	
HeartDiseaseorAttack	-0.026837	0.052171	0.020886	
BMI	-0.043843	0.068436	0.085504	
NoDocbcCost	-0.019029	0.180979	0.147045	
BMI_Age_interaction	0.010578	0.024439	0.018003	
Smoker	-0.013348	0.077416	0.036836	
Sex	-0.067222	-0.083029	-0.101889	
Income_GenHlth_interaction	0.027184	0.050374	0.058551	
CholCheck	-0.001225	-0.000793	-0.000507	
AnyHealthcare	0.019898	-0.042880	-0.030430	

	PhysHlth	PhysHlth_BMI_interaction	GenHlth	…	\
Education	-0.126253	-0.072603	-0.244402	…	
Income	-0.239457	-0.167852	-0.331527	…	
PhysActivity	-0.198083	-0.130892	-0.237046	…	
Fruits	-0.023217	-0.016050	-0.070749	…	
Veggies	-0.043859	-0.026937	-0.093575	…	
MentHlth	0.339225	0.244856	0.283591	…	
MentHlth_BMI_interaction	0.228810	0.255989	0.207340	…	
PhysHlth	1.000000	0.646708	0.516665	…	
PhysHlth_BMI_interaction	0.646708	1.000000	0.397747	…	
GenHlth	0.516665	0.397747	1.000000	…	

DiffWalk	0.466148	0.341392	0.446799	…
HighChol	0.112496	0.094796	0.189127	…
HighBP	0.146311	0.121854	0.274234	…
Age	0.097181	0.052928	0.147063	…
Stroke	0.140262	0.095627	0.169703	…
HeartDiseaseorAttack	0.170155	0.121153	0.246408	…
BMI	0.102659	0.128699	0.208347	…
NoDocbcCost	0.135519	0.115104	0.149368	…
BMI_Age_interaction	0.009369	0.006603	-0.008330	…
Smoker	0.100259	0.059745	0.135173	…
Sex	-0.043347	-0.057188	-0.009818	…
Income_GenHlth_interaction	0.186681	0.170682	0.547694	…
CholCheck	0.041509	0.041407	0.063448	…
AnyHealthcare	0.003747	0.006366	-0.022862	…

	Stroke	HeartDiseaseorAttack	BMI	\
Education	-0.063604	-0.082023	-0.074248	
Income	-0.116470	-0.122363	-0.069088	
PhysActivity	-0.058706	-0.072775	-0.127653	
Fruits	-0.003984	-0.006564	-0.067231	
Veggies	-0.032503	-0.026837	-0.043843	
MentHlth	0.061403	0.052171	0.068436	
MentHlth_BMI_interaction	0.035879	0.020886	0.085504	
PhysHlth	0.140262	0.170155	0.102659	
PhysHlth_BMI_interaction	0.095627	0.121153	0.128699	
GenHlth	0.169703	0.246408	0.208347	
DiffWalk	0.168852	0.202626	0.182618	
HighChol	0.090077	0.177053	0.090060	
HighBP	0.125442	0.202400	0.194577	
Age	0.129090	0.224860	-0.048771	
Stroke	1.000000	0.198895	0.010917	
HeartDiseaseorAttack	0.198895	1.000000	0.039785	
BMI	0.010917	0.039785	1.000000	
NoDocbcCost	0.028171	0.021622	0.045672	
BMI_Age_interaction	-0.019869	-0.050596	-0.044642	
Smoker	0.054349	0.104957	-0.009371	
Sex	0.004195	0.090186	0.030863	
Income_GenHlth_interaction	0.029920	0.091402	0.120772	
CholCheck	0.028263	0.050441	0.042657	
AnyHealthcare	0.013979	0.026338	-0.008390	

	NoDocbcCost	BMI_Age_interaction	Smoker	\
Education	-0.082337	0.011225	-0.135266	
Income	-0.186694	0.048654	-0.094754	
PhysActivity	-0.045451	-0.001034	-0.066600	
Fruits	-0.031692	-0.011742	-0.061450	
Veggies	-0.019029	0.010578	-0.013348	
MentHlth	0.180979	0.024439	0.077416	

MentHlth_BMI_interaction	0.147045	0.018003	0.036836
PhysHlth	0.135519	0.009369	0.100259
PhysHlth_BMI_interaction	0.115104	0.006603	0.059745
GenHlth	0.149368	-0.008330	0.135173
DiffWalk	0.105400	-0.016996	0.108013
HighChol	0.003607	-0.001015	0.074867
HighBP	0.002976	-0.050335	0.074514
Age	-0.129546	-0.009949	0.107864
Stroke	0.028171	-0.019869	0.054349
HeartDiseaseorAttack	0.021622	-0.050596	0.104957
BMI	0.045672	-0.044642	-0.009371
NoDocbcCost	1.000000	0.025736	0.037105
BMI_Age_interaction	0.025736	1.000000	0.018767
Smoker	0.037105	0.018767	1.000000
Sex	-0.045888	-0.023538	0.096800
Income_GenHlth_interaction	-0.041277	0.017421	0.032798
CholCheck	-0.053707	-0.022708	-0.003489
AnyHealthcare	-0.227076	-0.020887	-0.013727

	Sex	Income_GenHlth_interaction	CholCheck	\
Education	0.013806	0.144111	-0.010738	
Income	0.129017	0.547547	0.001062	
PhysActivity	0.032356	-0.057010	-0.005264	
Fruits	-0.089216	-0.012681	0.017393	
Veggies	-0.067222	0.027184	-0.001225	
MentHlth	-0.083029	0.050374	-0.000793	
MentHlth_BMI_interaction	-0.101889	0.058551	-0.000507	
PhysHlth	-0.043347	0.186681	0.041509	
PhysHlth_BMI_interaction	-0.057188	0.170682	0.041407	
GenHlth	-0.009818	0.547694	0.063448	
DiffWalk	-0.072443	0.090482	0.049789	
HighChol	0.022220	0.100061	0.094752	
HighBP	0.046650	0.114281	0.111253	
Age	-0.032902	0.034028	0.095594	
Stroke	0.004195	0.029920	0.028263	
HeartDiseaseorAttack	0.090186	0.091402	0.050441	
BMI	0.030863	0.120772	0.042657	
NoDocbcCost	-0.045888	-0.041277	-0.053707	
BMI_Age_interaction	-0.023538	0.017421	-0.022708	
Smoker	0.096800	0.032798	-0.003489	
Sex	1.000000	0.102191	-0.024959	
Income_GenHlth_interaction	0.102191	1.000000	0.052045	
CholCheck	-0.024959	0.052045	1.000000	
AnyHealthcare	-0.021922	0.092719	0.115200	

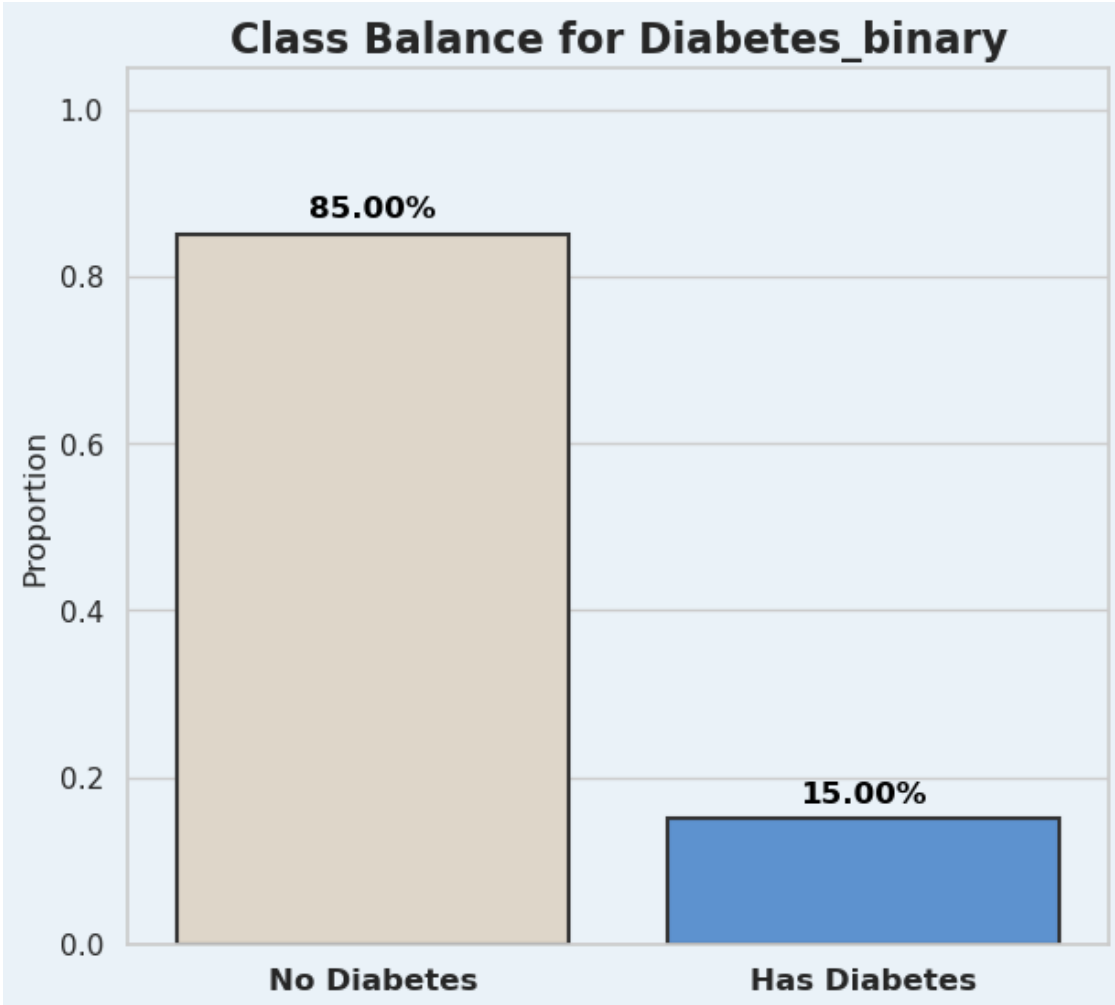
	AnyHealthcare
Education	0.110804
Income	0.145466

```
PhysActivity                     0.023189
Fruits                           0.022122
Veggies                          0.019898
MentHlth                        -0.042880
MentHlth_BMI_interaction        -0.030430
PhysHlth                         0.003747
PhysHlth_BMI_interaction         0.006366
GenHlth                         -0.022862
DiffWalk                         0.018463
HighChol                         0.052122
HighBP                           0.051749
Age                              0.147154
Stroke                           0.013979
HeartDiseaseorAttack             0.026338
BMI                             -0.008390
NoDocbcCost                     -0.227076
BMI_Age_interaction             -0.020887
Smoker                          -0.013727
Sex                             -0.021922
Income_GenHlth_interaction       0.092719
CholCheck                        0.115200
AnyHealthcare                    1.000000

[24 rows x 24 columns]
```

8 Sampling For Balance

8.1 Initial Balance

[ ]:
```
binary_labels['Diabetes_binary'] = ('No Diabetes','Has Diabetes')
plot_binary_class_balances(['Diabetes_binary'], y)
```



Great! Now we are ready to split the data and then model

9 Test Train Split on Full Data

[ ]:
```
#Split the full dataset into a temporary set and a holdout set
X_temp, X_holdout, y_temp, y_holdout = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)

#Further split the temporary set into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_temp, y_temp, test_size=0.
↪25, random_state=42, stratify=y_temp)

# Display the shape of the resulting splits
print("Train set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
print("Holdout set size:", X_holdout.shape)
```

```
Train set size: (136744, 24)
Test set size: (45582, 24)
Holdout set size: (45582, 24)
```

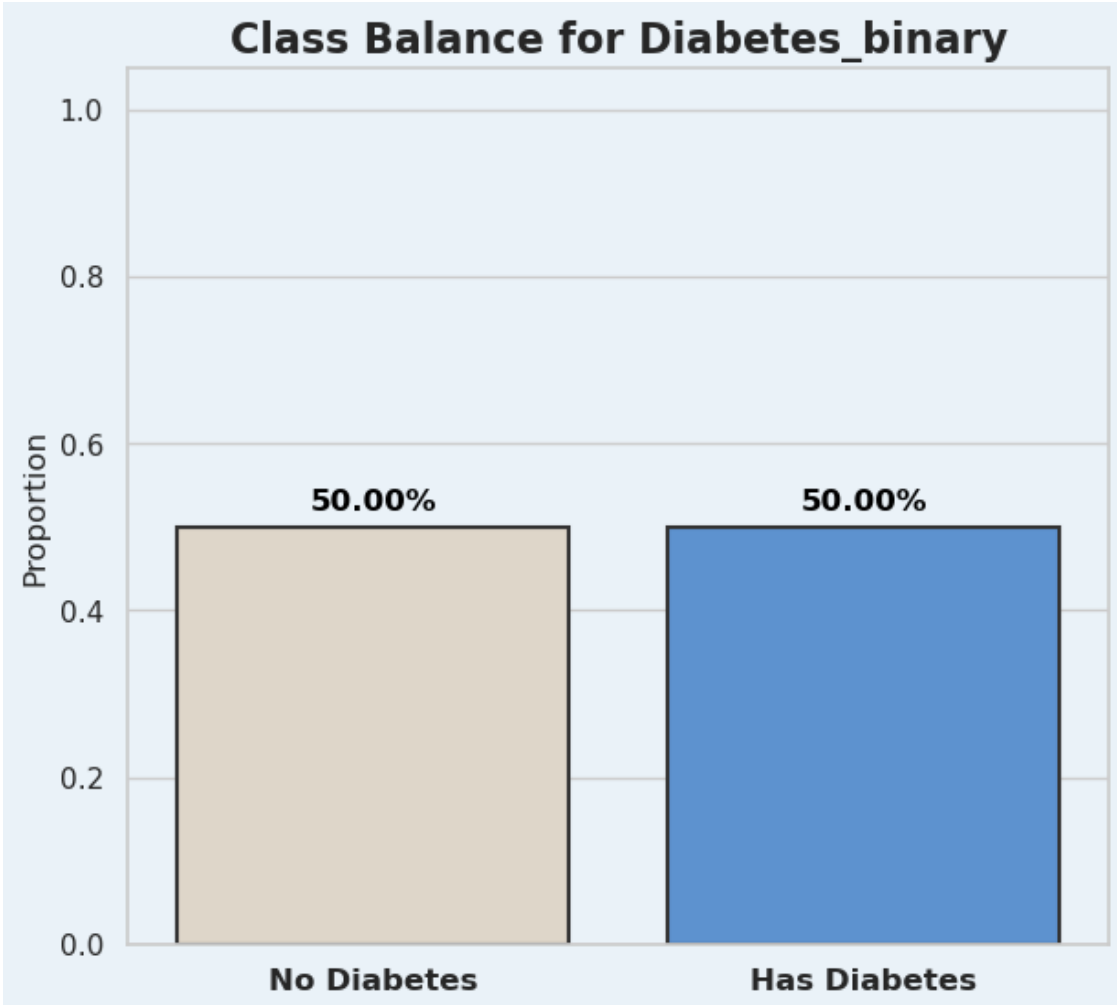
[ ]:
```
y_train.value_counts(normalize=True)
```

[ ]:
```
Diabetes_binary
0                  0.850034
1                  0.149966
Name: proportion, dtype: float64
```

9.1 Undersample Balance

[ ]:
```
# First, combine X_temp and y_temp into a single DataFrame to make resampling␣
↪easier
temp_df = pd.concat([X_temp, y_temp], axis=1)

# Separate the majority and minority classes
majority_class = temp_df[temp_df['Diabetes_binary'] == 0]
minority_class = temp_df[temp_df['Diabetes_binary'] == 1]

# Undersample the majority class to match the size of the minority class
majority_class_undersampled = resample(majority_class,
                                       replace=False,  # sample without␣
↪replacement
                                       n_samples=len(minority_class),  # match␣
↪minority class
                                       random_state=42)  # reproducible results

# Combine the undersampled majority class with the minority class to create the␣
↪undersampled dataset
undersampled_dataset = pd.concat([majority_class_undersampled, minority_class])

# Separate the features and target from the undersampled dataset
uX = undersampled_dataset.drop(columns=['Diabetes_binary'])
uy = undersampled_dataset['Diabetes_binary']

# Display the shape of the undersampled dataset
print("Undersampled dataset size:", undersampled_dataset.shape)

# Plot the class balance for the target variable
# Assuming plot_binary_class_balances is a function you have defined elsewhere
plot_binary_class_balances(['Diabetes_binary'], undersampled_dataset)
```

```
Undersampled dataset size: (54686, 25)
```



9.2 Test Train Split Undersampled Set

[ ]:
```
# Split the undersampled dataset into training and test sets
uX_train, uX_test, uy_train, uy_test = train_test_split(uX, uy, test_size=0.2,␣
↪random_state=42, stratify=uy)

# Display the shape of the resulting splits
print("uX_train set size:", uX_train.shape)
print("uX_test set size:", uX_test.shape)
print("uy_train set size:", uy_train.shape)
print("uy_test set size:", uy_test.shape)

# Check the class distribution in the training set
print(uy_train.value_counts(normalize=True))
```

```
uX_train set size: (43748, 24)
```

```
uX_test set size: (10938, 24)
uy_train set size: (43748,)
uy_test set size: (10938,)
Diabetes_binary
1    0.5
0    0.5
Name: proportion, dtype: float64
```

10 Utility Functions

10.1 Classification Report

[ ]:
```
def display_classification_report(y_true, y_pred, title="Classification␣
↪Report"):
    # Generate the classification report as a dictionary
    report = classification_report(y_true, y_pred, output_dict=True,␣
↪target_names=["No Diabetes", "Diabetes"])

    # Convert the dictionary to a pandas DataFrame for nicer formatting
    report_df = pd.DataFrame(report).transpose()

    # Display the classification report
    print(title)
    display(report_df)
```

10.2 Evaluate Model

[ ]:
```
# Initialize a global dictionary to store scores for different models
model_performance = {}
```

[ ]:
```
def evaluate_model(model, X_train, y_train, X_test, y_test, title="Model␣
↪Evaluation"):
    # Predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Probabilities for ROC AUC and Precision-Recall Curves
    y_train_prob = model.predict_proba(X_train)[:, 1]
    y_test_prob = model.predict_proba(X_test)[:, 1]

    # Scores
    train_auc = roc_auc_score(y_train, y_train_prob)
    test_auc = roc_auc_score(y_test, y_test_prob)

    # Classification reports
    train_report = classification_report(y_train, y_train_pred,␣
↪output_dict=True, target_names=["Train Class 0", "Train Class 1"])
```

```
    test_report = classification_report(y_test, y_test_pred, output_dict=True,␣
↪target_names=["Test Class 0", "Test Class 1"])


    # Store recall, precision, and AUC scores for the test set in the global␣
↪dictionary
    model_performance[title] = {
        "Test Class 0 Precision": test_report["Test Class 0"]["precision"],
        "Test Class 0 Recall": test_report["Test Class 0"]["recall"],
        "Test Class 1 Precision": test_report["Test Class 1"]["precision"],
        "Test Class 1 Recall": test_report["Test Class 1"]["recall"],
        "Test AUC": test_auc,
        "Test Accuracy": test_report['accuracy'],
    }

    # Define colors based on Seaborn palettes
    train_color = sns.color_palette("deep")[0]  # Typically blue in deep palette
    test_color = sns.color_palette("deep")[2]   # Typically red in deep palette

    # Plotting
    fig = plt.figure(figsize=(14, 25))  # Increased height to accommodate extra␣
↪row
    gs = fig.add_gridspec(4, 2, height_ratios=[1, 1, 1, 1])

    # ROC AUC Curve
    ax1 = fig.add_subplot(gs[0, 0])
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_prob)
    ax1.plot(fpr_train, tpr_train, label=f'Train ROC AUC: {train_auc:.4f}',␣
↪color=train_color)
    ax1.plot([0, 1], [0, 1], 'k--', lw=2)
    ax1.set_title('ROC AUC Curve - Train', fontsize=16)
    ax1.set_xlabel('False Positive Rate', fontsize=12)
    ax1.set_ylabel('True Positive Rate', fontsize=12)
    ax1.legend(loc='lower right')

    ax2 = fig.add_subplot(gs[0, 1])
    fpr_test, tpr_test, _ = roc_curve(y_test, y_test_prob)
    ax2.plot(fpr_test, tpr_test, label=f'Test ROC AUC: {test_auc:.4f}',␣
↪color=test_color)
    ax2.plot([0, 1], [0, 1], 'k--', lw=2)
    ax2.set_title('ROC AUC Curve - Test', fontsize=16)
    ax2.set_xlabel('False Positive Rate', fontsize=12)
    ax2.set_ylabel('True Positive Rate', fontsize=12)
    ax2.legend(loc='lower right')

    # Precision-Recall Curves
    ax3 = fig.add_subplot(gs[1, 0])
```

```
    precision_train, recall_train, _ = precision_recall_curve(y_train,␣
↪y_train_prob)
    ax3.plot(recall_train, precision_train, label=f'Train PR AUC:␣
↪{auc(recall_train, precision_train):.4f}', color=train_color)
    ax3.set_title('Precision-Recall Curve - Train', fontsize=16)
    ax3.set_xlabel('Recall', fontsize=12)
    ax3.set_ylabel('Precision', fontsize=12)
    ax3.legend(loc='lower left')

    ax4 = fig.add_subplot(gs[1, 1])
    precision_test, recall_test, _ = precision_recall_curve(y_test, y_test_prob)
    ax4.plot(recall_test, precision_test, label=f'Test PR AUC:␣
↪{auc(recall_test, precision_test):.4f}', color=test_color)
    ax4.set_title('Precision-Recall Curve - Test', fontsize=16)
    ax4.set_xlabel('Recall', fontsize=12)
    ax4.set_ylabel('Precision', fontsize=12)
    ax4.legend(loc='lower left')

    # Confusion Matrices
    ax5 = fig.add_subplot(gs[2, 0])
    train_conf_matrix = confusion_matrix(y_train, y_train_pred)
    sns.heatmap(train_conf_matrix, annot=True, fmt="d", cmap=sns.
↪light_palette(train_color, as_cmap=True), ax=ax5)
    ax5.set_title("Train Confusion Matrix", fontsize=16)
    ax5.set_xlabel("Predicted Label", fontsize=12)
    ax5.set_ylabel("True Label", fontsize=12)

    ax6 = fig.add_subplot(gs[2, 1])
    test_conf_matrix = confusion_matrix(y_test, y_test_pred)
    sns.heatmap(test_conf_matrix, annot=True, fmt="d", cmap=sns.
↪light_palette(test_color, as_cmap=True), ax=ax6)
    ax6.set_title("Test Confusion Matrix", fontsize=16)
    ax6.set_xlabel("Predicted Label", fontsize=12)
    ax6.set_ylabel("True Label", fontsize=12)

    # Classification Report Plot
    ax7 = fig.add_subplot(gs[3, :])
    categories = ["precision", "recall", "f1-score", "accuracy"]
    train_metrics = [
        train_report["Train Class 0"]["precision"],
        train_report["Train Class 0"]["recall"],
        train_report["Train Class 0"]["f1-score"],
        train_report["Train Class 1"]["precision"],
        train_report["Train Class 1"]["recall"],
        train_report["Train Class 1"]["f1-score"],
        train_report["accuracy"]
    ]
```

```
    test_metrics = [
        test_report["Test Class 0"]["precision"],
        test_report["Test Class 0"]["recall"],
        test_report["Test Class 0"]["f1-score"],
        test_report["Test Class 1"]["precision"],
        test_report["Test Class 1"]["recall"],
        test_report["Test Class 1"]["f1-score"],
        test_report["accuracy"]
    ]

    bar_width = 0.35
    index = np.arange(len(train_metrics))

    ax7.bar(index, train_metrics, bar_width, label='Train', color=train_color)
    ax7.bar(index + bar_width, test_metrics, bar_width, label='Test',␣
↪color=test_color)

    ax7.set_xticks(index + bar_width / 2)
    ax7.set_xticklabels(["0 Precision", "0 Recall", "0 F1", "1 Precision", "1␣
↪Recall", "1 F1", "Accuracy"], rotation=45)
    ax7.set_title("Classification Report Metrics", fontsize=16)
    ax7.legend()

    # Add the title to the entire figure
    fig.suptitle(title, fontsize=20, fontweight='bold')

    plt.tight_layout(rect=[0, 0, 1, 0.97])  # Adjust layout to make space for␣
↪the suptitle
    plt.show()

    test_class_report = classification_report(y_test, y_test_pred,␣
↪target_names=["Test Class 0", "Test Class 1"])
    print("Test Classification Report:\n", test_class_report)
```

10.3 Model Confusion Matrix

```
[ ]: def model_confusion_matrix(model, X_test, y_test, threshold=.5, title="Model␣
↪Evaluation"):
    # Predict probabilities
    y_test_prob = model.predict_proba(X_test)[:, 1]

    # Apply custom threshold to binary predictions
    y_test_pred = (y_test_prob >= threshold).astype(int)

    # Generate confusion matrix
```

```
    test_conf_matrix = confusion_matrix(y_test, y_test_pred)

    # Plotting the confusion matrix
    fig, ax = plt.subplots(figsize=(6, 5))
    sns.heatmap(test_conf_matrix, annot=True, fmt="d", cmap='Blues', ax=ax)
    ax.set_title("Model Confusion Matrix", fontsize=16)
    ax.set_xlabel("Predicted Label", fontsize=12)
    ax.set_ylabel("True Label", fontsize=12)
    plt.show()
```

10.4 Evaluation Summary

```
[ ]: def evaluation_summary():
    # Extract and sort the keys (model names) by AUC score in ascending order
    model_names = sorted(model_performance.keys(), key=lambda x:␣
 ↪model_performance[x]["Test AUC"])

    # Extracting the metrics for each model in the sorted order
    test_class_0_precision = [model_performance[model]["Test Class 0␣
 ↪Precision"] for model in model_names]
    test_class_0_recall = [model_performance[model]["Test Class 0 Recall"] for␣
 ↪model in model_names]
    test_class_1_precision = [model_performance[model]["Test Class 1␣
 ↪Precision"] for model in model_names]
    test_class_1_recall = [model_performance[model]["Test Class 1 Recall"] for␣
 ↪model in model_names]
    test_auc = [model_performance[model]["Test AUC"] for model in model_names]
    test_accuracy = [model_performance[model]["Test Accuracy"] for model in␣
 ↪model_names]

    # Increase the bar height to make the bars taller
    bar_height = 0.18  # Adjusted for taller bars
    group_spacing = bar_height * 8  # Increase this value to add more space␣
 ↪between groups
    index = np.arange(len(model_names))

    # Set the color palette to 'deep'
    sns.set_palette("deep")

    # Create a figure and axis with appropriate size for full-width plotting
    fig, ax = plt.subplots(figsize=(18, 28))  # Adjusting size for large␣
 ↪monitors

    # Plotting the bars for each metric in the correct order
    plt.barh(index * group_spacing + 3 * bar_height, test_auc, bar_height,␣
 ↪label="Test AUC")
```

```
    plt.barh(index * group_spacing + 2 * bar_height, test_class_1_recall,␣
↪bar_height, label="Test Class 1 Recall")
    plt.barh(index * group_spacing + 1 * bar_height, test_class_0_recall,␣
↪bar_height, label="Test Class 0 Recall")
    plt.barh(index * group_spacing + 0 * bar_height, test_class_1_precision,␣
↪bar_height, label="Test Class 1 Precision")
    plt.barh(index * group_spacing - 1 * bar_height, test_class_0_precision,␣
↪bar_height, label="Test Class 0 Precision")
    plt.barh(index * group_spacing - 2 * bar_height, test_accuracy, bar_height,␣
↪label="Test Accuracy")

    # Adding labels, title, and ticks
    ax.set_ylabel('Model', fontsize=14)
    ax.set_xlabel('Score', fontsize=14)
    ax.set_title('Model Performance Summary', fontsize=20, fontweight='bold',␣
↪pad=20,fontfamily='DejaVu Sans')

    # Setting y-ticks with model names, placing them in the center of the group
    ax.set_yticks(index * group_spacing)
    ax.set_yticklabels(model_names, fontsize=14,␣
↪fontweight='bold',fontfamily='DejaVu Sans')

    # Adjusting x-axis limit assuming all scores are between 0 and 1
    ax.set_xlim(0, 1)

    # Adding the actual values on top of the bars with a font size of 10
    for i in range(len(model_names)):
        ax.text(test_auc[i] + 0.01, index[i] * group_spacing + 3 * bar_height,␣
↪f'{test_auc[i]:.3f}', va='center', fontsize=12)
        ax.text(test_class_1_recall[i] + 0.01, index[i] * group_spacing + 2 *␣
↪bar_height, f'{test_class_1_recall[i]:.3f}', va='center', fontsize=12)
        ax.text(test_class_0_recall[i] + 0.01, index[i] * group_spacing + 1 *␣
↪bar_height, f'{test_class_0_recall[i]:.3f}', va='center', fontsize=12)
        ax.text(test_class_1_precision[i] + 0.01, index[i] * group_spacing + 0␣
↪* bar_height, f'{test_class_1_precision[i]:.3f}', va='center', fontsize=12)
        ax.text(test_class_0_precision[i] + 0.01, index[i] * group_spacing - 1␣
↪* bar_height, f'{test_class_0_precision[i]:.3f}', va='center', fontsize=12)
        ax.text(test_accuracy[i] + 0.01, index[i] * group_spacing - 2 *␣
↪bar_height, f'{test_accuracy[i]:.3f}', va='center', fontsize=12)

    # Adjusting the legend to be below the title and span the full width, with␣
↪3 rows
    ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.005), ncol=3,␣
↪fontsize=10, frameon=True, facecolor='white', edgecolor='black')

    # Adjust layout to minimize extra space
```

```
    plt.subplots_adjust(top=0.85, bottom=0.1, left=0.05, right=0.95, hspace=0.2)

    # Display the plot
    plt.show()
```

10.5 Evaluation Summary Simplified

```
[ ]: def evaluation_summary_simplified(perf_dict=model_performance):
    # Extract and sort the keys (model names) by AUC score in descending order
    model_names = sorted(perf_dict.keys(), key=lambda x: perf_dict[x]["Test␣
 ↪AUC"], reverse=False)

    # Extracting the Test AUC for each model in the sorted order
    test_auc = [perf_dict[model]["Test AUC"] for model in model_names]

    # Adjust the bar height and spacing between bars
    bar_height = 0.1
    spacing = 0.05  # Adding space between bars
    index = np.arange(len(model_names))

    # Define a custom color gradient excluding the brightest colors
    custom_colors = ["#00008B", "#1E90FF", "#32CD32", "#FFD700", "#FF4500",␣
 ↪"#FF0000"]
    cmap = LinearSegmentedColormap.from_list("custom_cmap", custom_colors)
    colors = cmap(np.linspace(0, 1, len(model_names)))

    # Create a figure and axis with appropriate size for full-width plotting
    fig, ax = plt.subplots(figsize=(16, 8))

    # Plotting the bars for AUC in the correct order with gaps in between
    bars = plt.barh(index * (bar_height + spacing), test_auc, bar_height,␣
 ↪color=colors, label="AUC")

    # Adding labels, title, and ticks
    ax.set_ylabel('Model', fontsize=14, fontweight='bold')
    ax.set_xlabel('Score', fontsize=14, fontweight='bold')
    ax.set_title('Model Performance Summary (ROC AUC)', fontsize=18,␣
 ↪fontweight='bold', pad=20)

    # Setting y-ticks with model names, accounting for the spacing
    ax.set_yticks(index * (bar_height + spacing))
    ax.set_yticklabels(model_names, fontsize=12, fontweight='bold')

    # Adjusting x-axis limit assuming all scores are between 0 and 1
    ax.set_xlim(0, 1)
```

```
    # Adding the actual values inside the bars
    for bar, auc in zip(bars, test_auc):
        ax.text(auc - 0.05, bar.get_y() + bar.get_height()/2, f'{auc:.3f}',
                va='center', ha='center', fontsize=12, color='white',␣
↪fontweight='bold')

    # Adding vertical gridlines for better readability
    ax.xaxis.grid(True, linestyle='--', which='both', color='gray', alpha=0.7)

    # Adjust layout to minimize extra space
    plt.subplots_adjust(top=0.85, bottom=0.1, left=0.15, right=0.95)

    # Display the plot
    plt.show()
```

10.6 Threshold evaluation

[ ]:
```
def evaluate_threshold(y_true, y_prob, threshold):
    y_pred = (y_prob >= threshold).astype(int)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    auc = roc_auc_score(y_true, y_prob)

    return {
        'Threshold': threshold,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'Accuracy': accuracy,
        'AUC': auc
    }
def model_thresholds(model,tX_test,ty_test,title='',showplot=False):
    y_test_prob = model.predict_proba(tX_test)[:, 1]

    thresholds = np.arange(0.1, 0.9, 0.05)  # Define a range of thresholds to␣
↪test
    results = []

    for threshold in thresholds:
        metrics = evaluate_threshold(ty_test, y_test_prob, threshold)
        results.append(metrics)

    # Convert to a DataFrame for easier analysis
    results_df = pd.DataFrame(results)
    if showplot:
```

```
        plt.figure(figsize=(10, 6))
        plt.plot(results_df['Threshold'], results_df['Precision'],␣
↪label='Precision')
        plt.plot(results_df['Threshold'], results_df['Recall'], label='Recall')
        plt.plot(results_df['Threshold'], results_df['F1 Score'], label='F1␣
↪Score')
        plt.plot(results_df['Threshold'], results_df['Accuracy'],␣
↪label='Accuracy')
        plt.xlabel('Threshold')
        plt.ylabel('Score')
        plt.title(title)
        plt.legend()
        plt.grid(True)
        plt.show()

    return results_df, title

def report_at_threshold(best_threshold,model,tX_test,ty_test):
    y_test_prob = model.predict_proba(tX_test)[:, 1]
    y_test_pred_adjusted = (y_test_prob >= best_threshold).astype(int)

    # Confusion Matrix
    conf_matrix = confusion_matrix(ty_test, y_test_pred_adjusted)
    print("Confusion Matrix:\n", conf_matrix)

    # Classification Report
    class_report = classification_report(ty_test, y_test_pred_adjusted)
    print("Classification Report:\n", class_report)

def plot_subplots(models_data):
    # Determine the number of rows needed
    n_cols = 3# Fixed number of columns
    n_rows = math.ceil(len(models_data) / n_cols)  # Calculate the number of␣
↪rows needed# Create the figure and axes
    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(16, n_rows *␣
↪6))
    axes = axes.flatten()

    for i, (model_data, title) in enumerate(models_data):
        ax = axes[i]
        ax.plot(model_data['Threshold'], model_data['Precision'],␣
↪label='Precision')
        ax.plot(model_data['Threshold'], model_data['Recall'], label='Recall')
        ax.plot(model_data['Threshold'], model_data['F1 Score'], label='F1␣
↪Score')
```

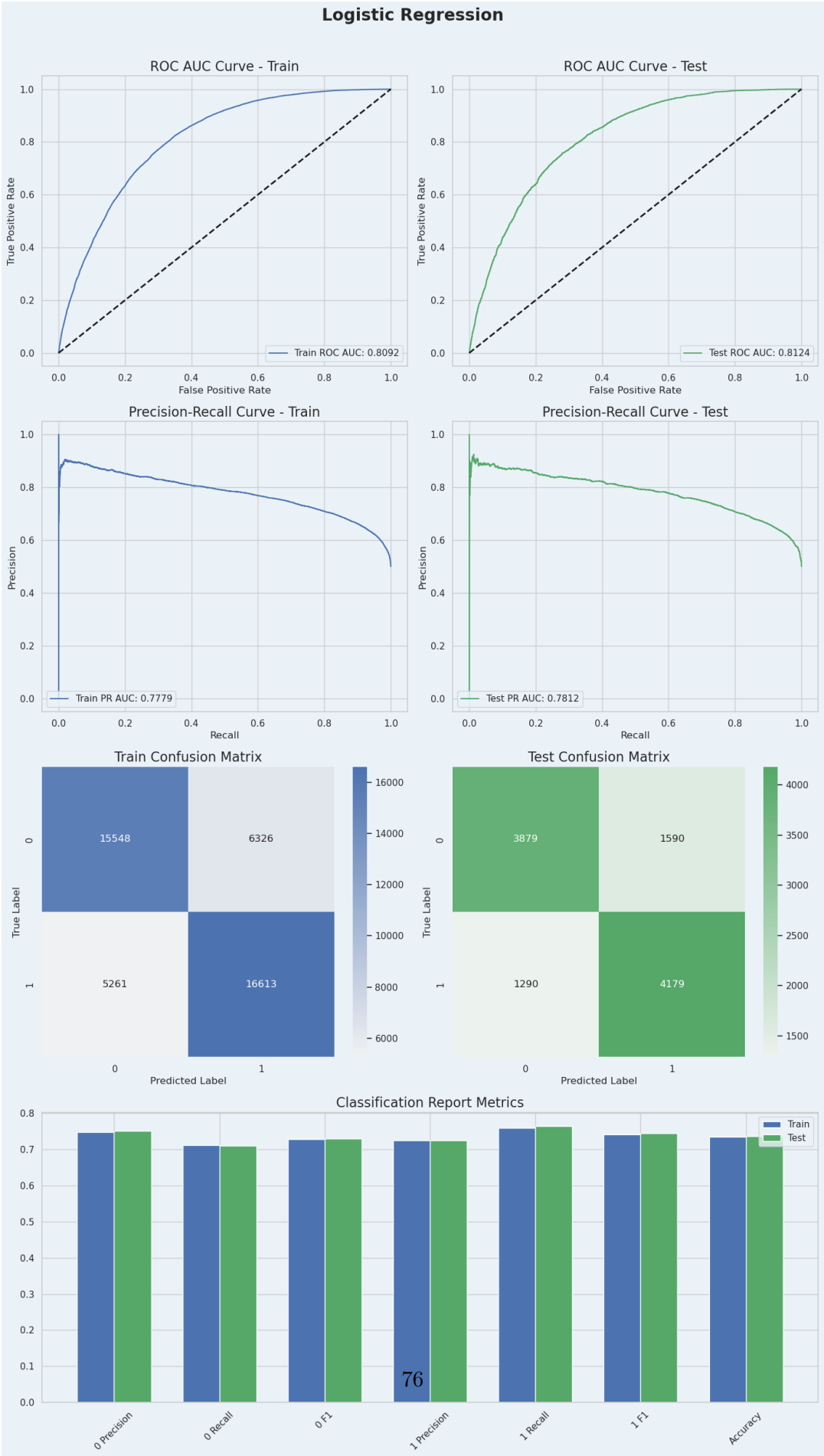
```
        ax.plot(model_data['Threshold'], model_data['Accuracy'],␣
↪label='Accuracy')
        ax.set_title(title)
        ax.set_xlabel('Threshold')
        ax.set_ylabel('Score')
        ax.legend()
        ax.grid(True)

    # Remove any unused axes (if any)
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

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

11 First Simple Model

[ ]:
```
# Initialize the Logistic Regression model
log_reg = LogisticRegression(random_state=42, max_iter=1000)

# Train the model on the training data
log_reg.fit(uX_train, uy_train)
```

[ ]:
```
LogisticRegression(max_iter=1000, random_state=42)
```

[ ]:
```
# Evaluate the model
evaluate_model(log_reg, uX_train, uy_train, uX_test, uy_test,'Logistic␣
↪Regression')
```


Logistic Regression
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8092
ROC AUC Curve - Test
Test ROC AUC: 0.8124
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.7779
Precision-Recall Curve - Test
Test PR AUC: 0.7812
Train Confusion Matrix
15548
6326
5261
16613
True Label
Predicted Label
Test Confusion Matrix
3879
1590
1290
4179
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.75      0.71      0.73      5469
Test Class 1       0.72      0.76      0.74      5469

    accuracy                           0.74     10938
   macro avg       0.74      0.74      0.74     10938
weighted avg       0.74      0.74      0.74     10938
```

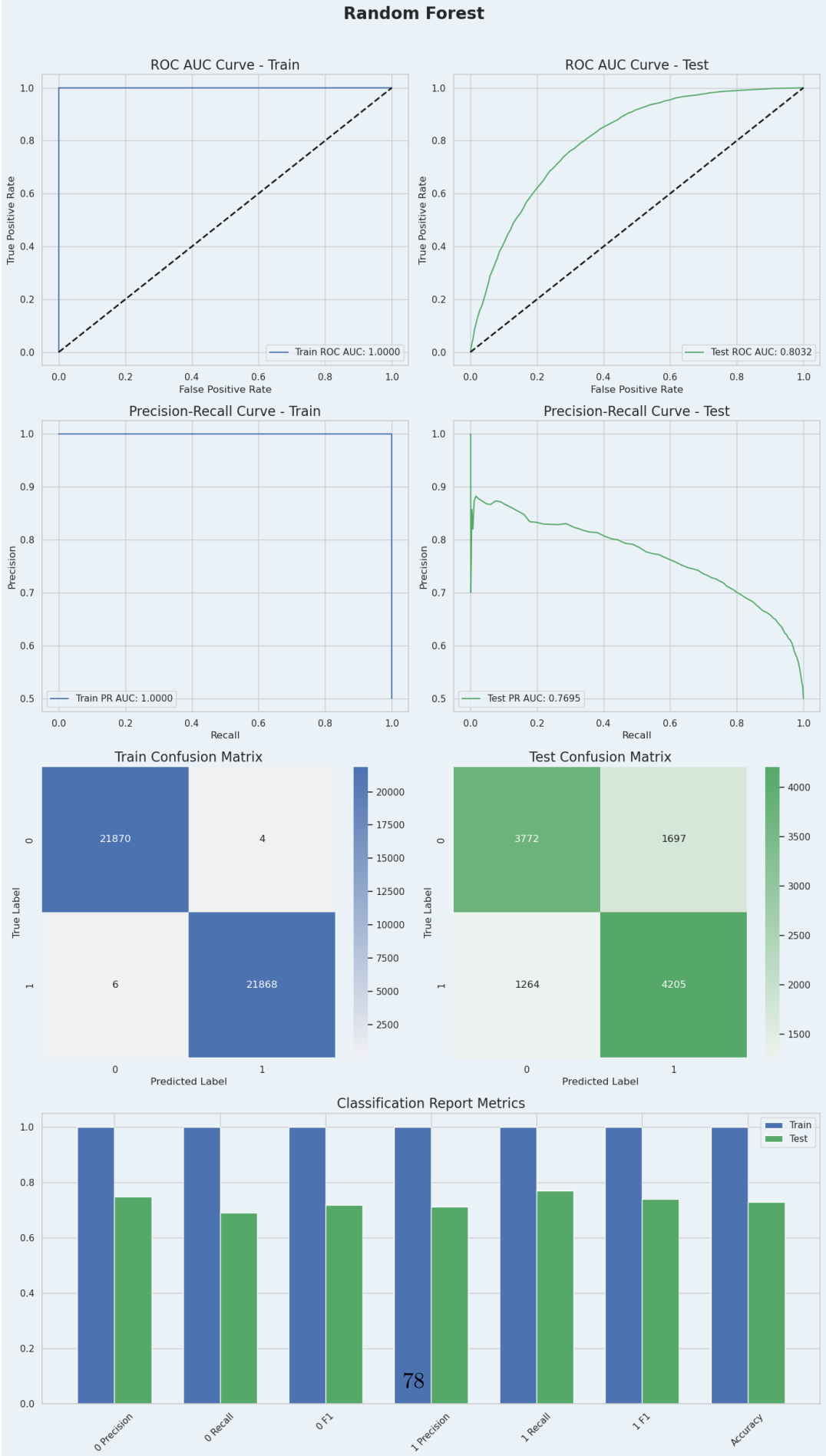
12 Sample Models

Before spending a lot of time tuning models it's best to sample some promising ones for the data set and see which shows the best results before tuning.

12.1 Random Forest

I selected the Random Forest model for its robust performance across multiple types of data sets. Random Foreset aggregates predictions from multiple decision trees to enhance accuracy and handle complex datasets.

[ ]:
```
# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)

# Train the model on the training data
rf_model.fit(uX_train, uy_train)

# Evaluate the model
evaluate_model(rf_model, uX_train, uy_train, uX_test, uy_test,'Random Forest')
```


Random Forest
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 1.0000
ROC AUC Curve - Test
Test ROC AUC: 0.8032
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 1.0000
Precision-Recall Curve - Test
Test PR AUC: 0.7695
Train Confusion Matrix
21870
4
6
21868
True Label
Predicted Label
Test Confusion Matrix
3772
1697
1264
4205
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.75      0.69      0.72      5469
Test Class 1       0.71      0.77      0.74      5469

    accuracy                           0.73     10938
   macro avg       0.73      0.73      0.73     10938
weighted avg       0.73      0.73      0.73     10938
```

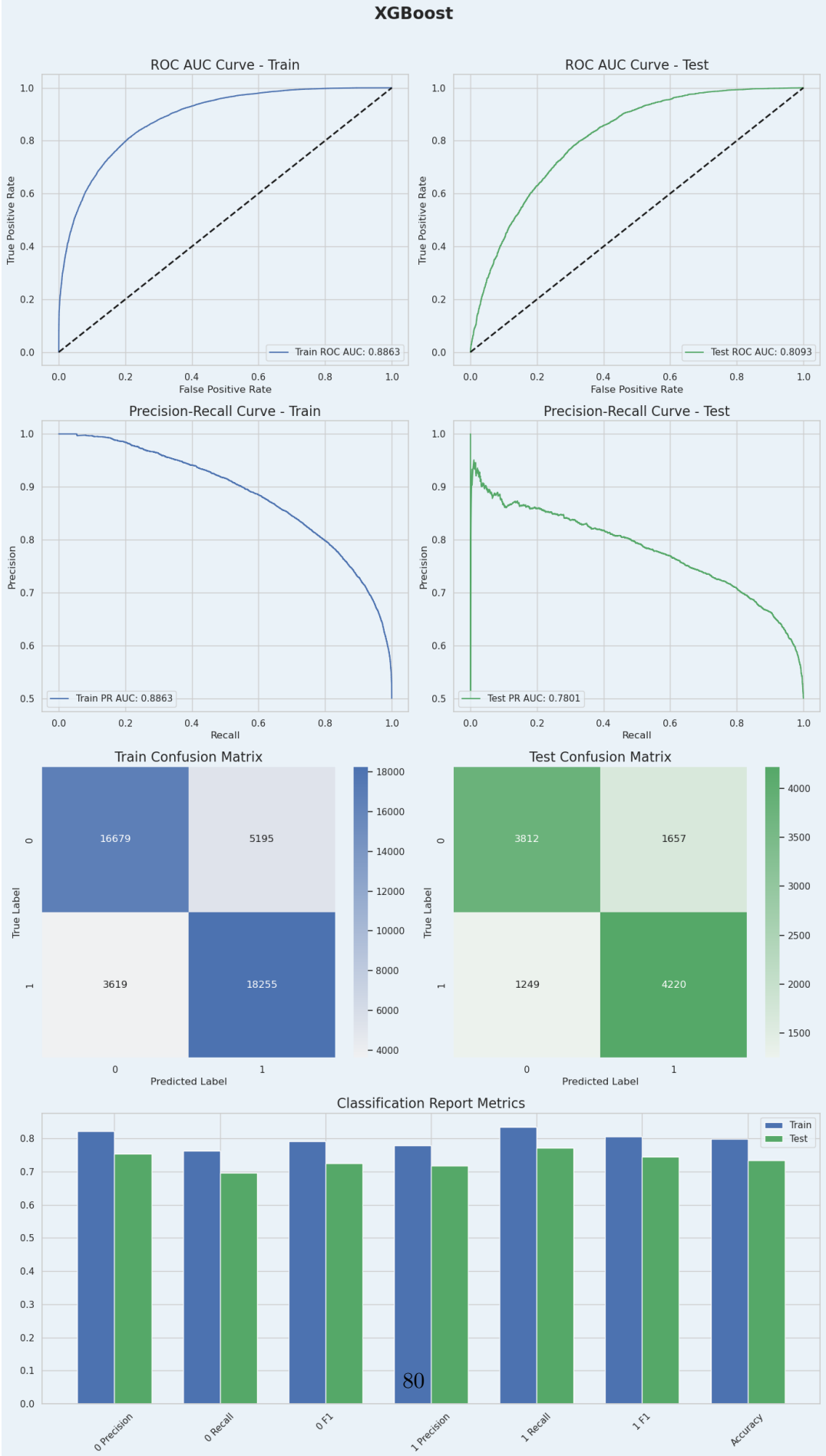
12.2 XGBoost

I selected the XGBoost model for its efficiency in processing large and complex datasets with a gradient boosting framework designed to optimize speed and model performance. It runs effectively on GPUs and excels at identifying key patterns and interactions within the dataset.

[ ]:
```
# Initialize the XGBoost model with GPU support
xgb_model = xgb.XGBClassifier(
    random_state=42,
    eval_metric='logloss',
    tree_method='hist',  # Enable GPU support
    device='cuda'
)

# Train the model on the training data
xgb_model.fit(uX_train, uy_train)
```

[ ]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device='cuda', early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=42, …)
```

[ ]:
```
# Evaluate the model
evaluate_model(xgb_model, uX_train, uy_train, uX_test, uy_test,'XGBoost')
```


XGBoost
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8863
ROC AUC Curve - Test
Test ROC AUC: 0.8093
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.8863
Precision-Recall Curve - Test
Test PR AUC: 0.7801
Train Confusion Matrix
16679
5195
3619
18255
True Label
Predicted Label
Test Confusion Matrix
3812
1657
1249
4220
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.75      0.70      0.72      5469
Test Class 1       0.72      0.77      0.74      5469

    accuracy                           0.73     10938
   macro avg       0.74      0.73      0.73     10938
weighted avg       0.74      0.73      0.73     10938
```

12.3 Catboost

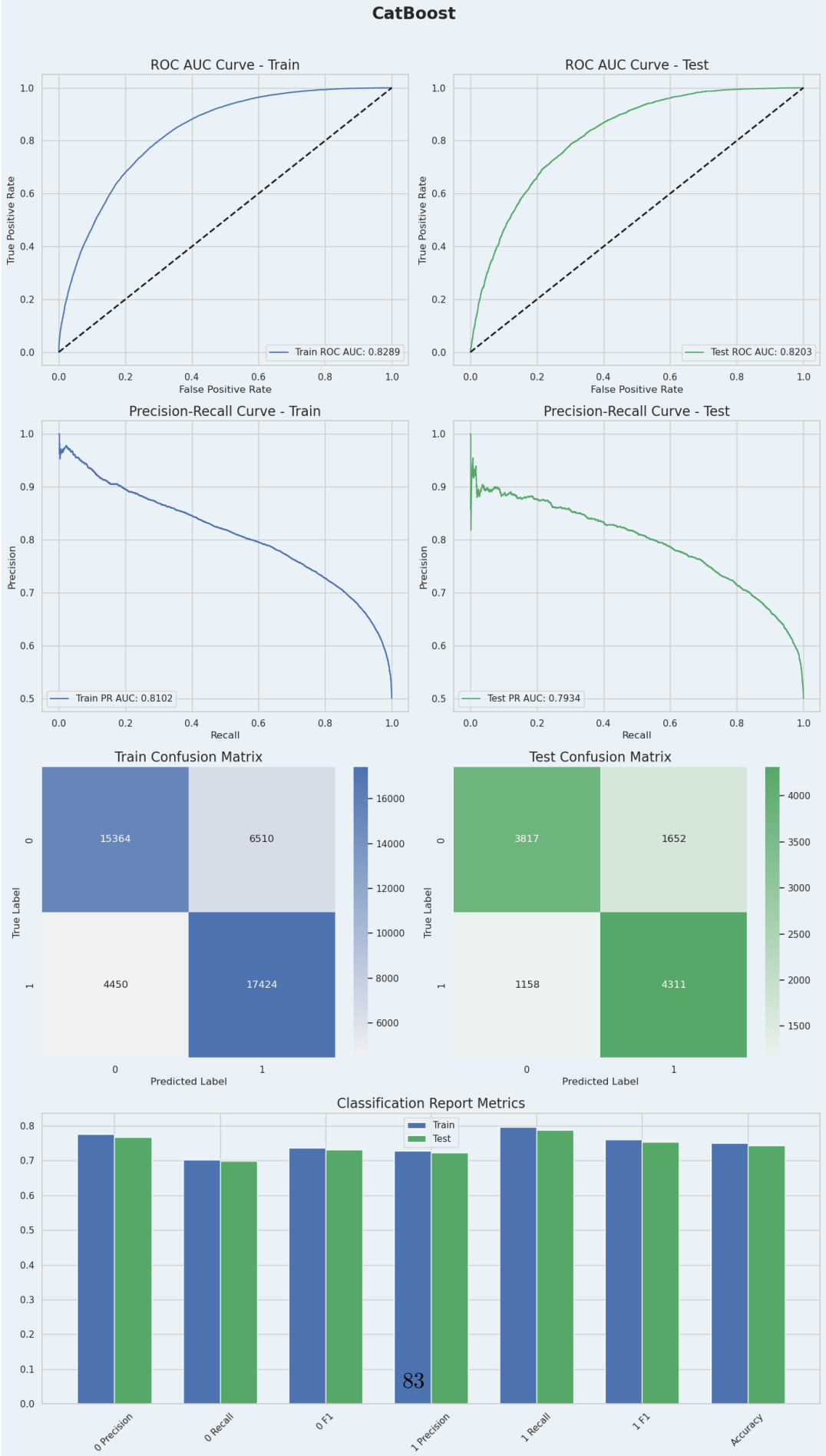
I selected the CatBoost model for its advanced handling of complex feature interactions and its gradient boosting framework, which effectively reduces overfitting. It has proven high accuracy and reliability, as demonstrated by its frequent use and success in many data science competitions. Its reputation as a robust model is well-established in the field.

[ ]:
```
# Initialize the CatBoost Classifier
catboost_model = CatBoostClassifier(
    iterations=1000,
    learning_rate=0.1,
    depth=6,
    eval_metric='F1',
    random_seed=42,
    task_type="GPU"  # Ensures GPU usage
)

# Train the CatBoost model
catboost_model.fit(uX_train, uy_train, eval_set=(uX_test, uy_test),␣
 ↪verbose=100, early_stopping_rounds=100)
```

```
Warning: less than 75% gpu memory available for training. Free: 684.8125 Total:
3803.1875

0:      learn: 0.7405065        test: 0.7405750 best: 0.7405750 (0)     total:
86.8ms   remaining: 1m 26s
100:    learn: 0.7577266        test: 0.7512681 best: 0.7526788 (41)    total:
2.4s     remaining: 21.4s
200:    learn: 0.7614475        test: 0.7531059 best: 0.7541987 (187)   total:
4.76s    remaining: 18.9s
bestTest = 0.7541987404
bestIteration = 187
Shrink model to first 188 iterations.
```

[ ]:
```
<catboost.core.CatBoostClassifier at 0x7ca5c88da6b0>
```

[ ]:
```
# Evaluate the model
evaluate_model(catboost_model, uX_train, uy_train, uX_test, uy_test,'CatBoost')
```


CatBoost
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8289
ROC AUC Curve - Test
Test ROC AUC: 0.8203
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.8102
Precision-Recall Curve - Test
Test PR AUC: 0.7934
Train Confusion Matrix
15364
6510
4450
17424
True Label
Predicted Label
Test Confusion Matrix
3817
1652
1158
4311
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.77      0.70      0.73      5469
Test Class 1       0.72      0.79      0.75      5469

    accuracy                           0.74     10938
   macro avg       0.75      0.74      0.74     10938
weighted avg       0.75      0.74      0.74     10938
```

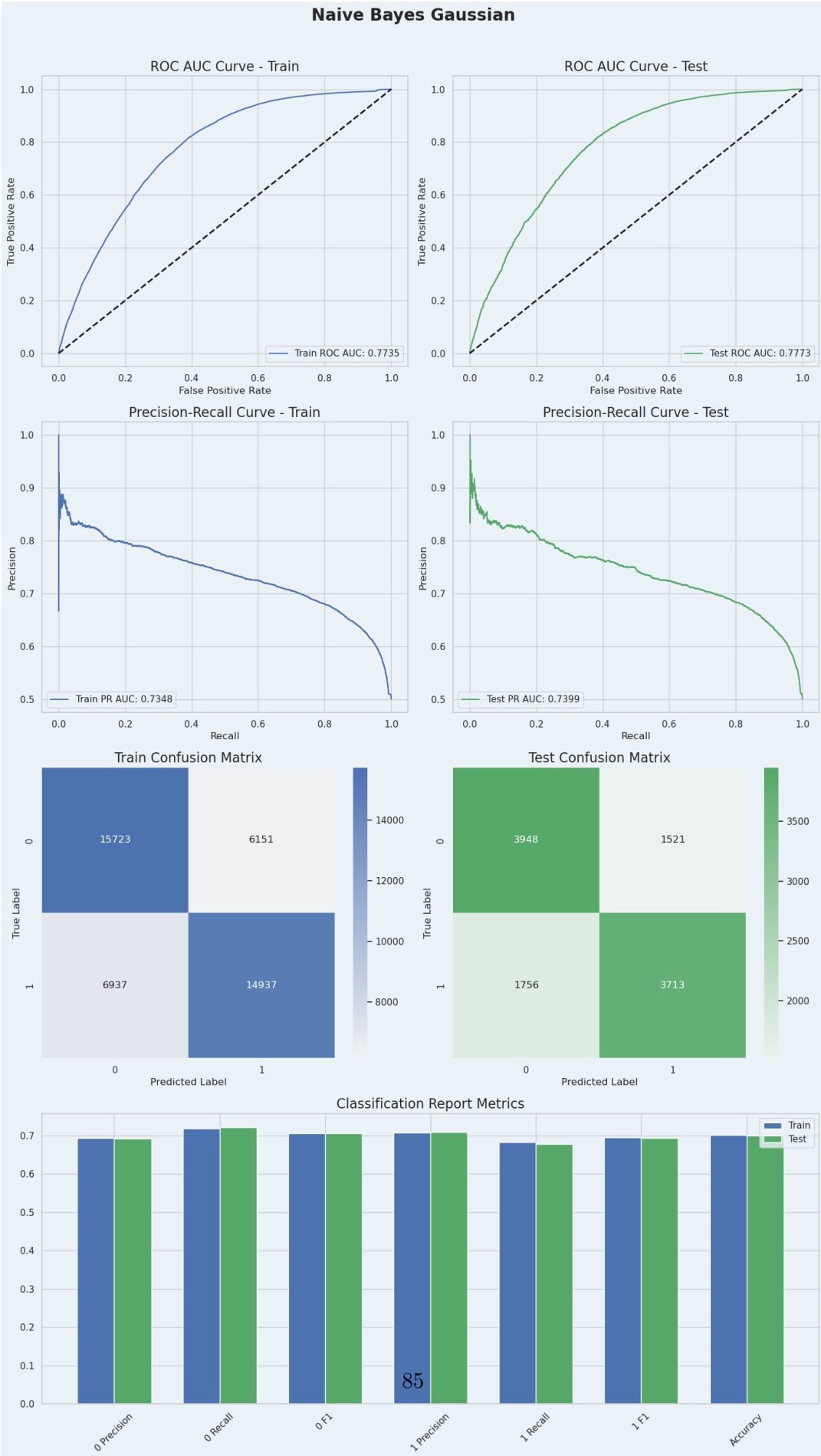
12.4 Naive Bayes Gaussian

I chose the Gaussian Naive Bayes model for its established use in scientific and medical fields, particularly for datasets where features are assumed to follow a normal gaussian distribution. This assumption simplifies the probability calculations for model training and prediction. Its simplicity and computational speed make it suitable for initial baseline evaluations.

[ ]:
```
# Initialize the Gaussian Naive Bayes model
gnb = GaussianNB()

# Train the model on the training data
gnb.fit(uX_train, uy_train)
```

[ ]:
```
GaussianNB()
```

[ ]:
```
# Evaluate the model using your custom evaluate_model function
evaluate_model(gnb, uX_train, uy_train, uX_test, uy_test,'Naive Bayes Gaussian')
```


Naive Bayes Gaussian
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.7735
ROC AUC Curve - Test
Test ROC AUC: 0.7773
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.7348
Precision-Recall Curve - Test
Test PR AUC: 0.7399
Train Confusion Matrix
15723
6151
6937
14937
True Label
Predicted Label
Test Confusion Matrix
3948
1521
1756
3713
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.69      0.72      0.71      5469
Test Class 1       0.71      0.68      0.69      5469

    accuracy                           0.70     10938
   macro avg       0.70      0.70      0.70     10938
weighted avg       0.70      0.70      0.70     10938
```

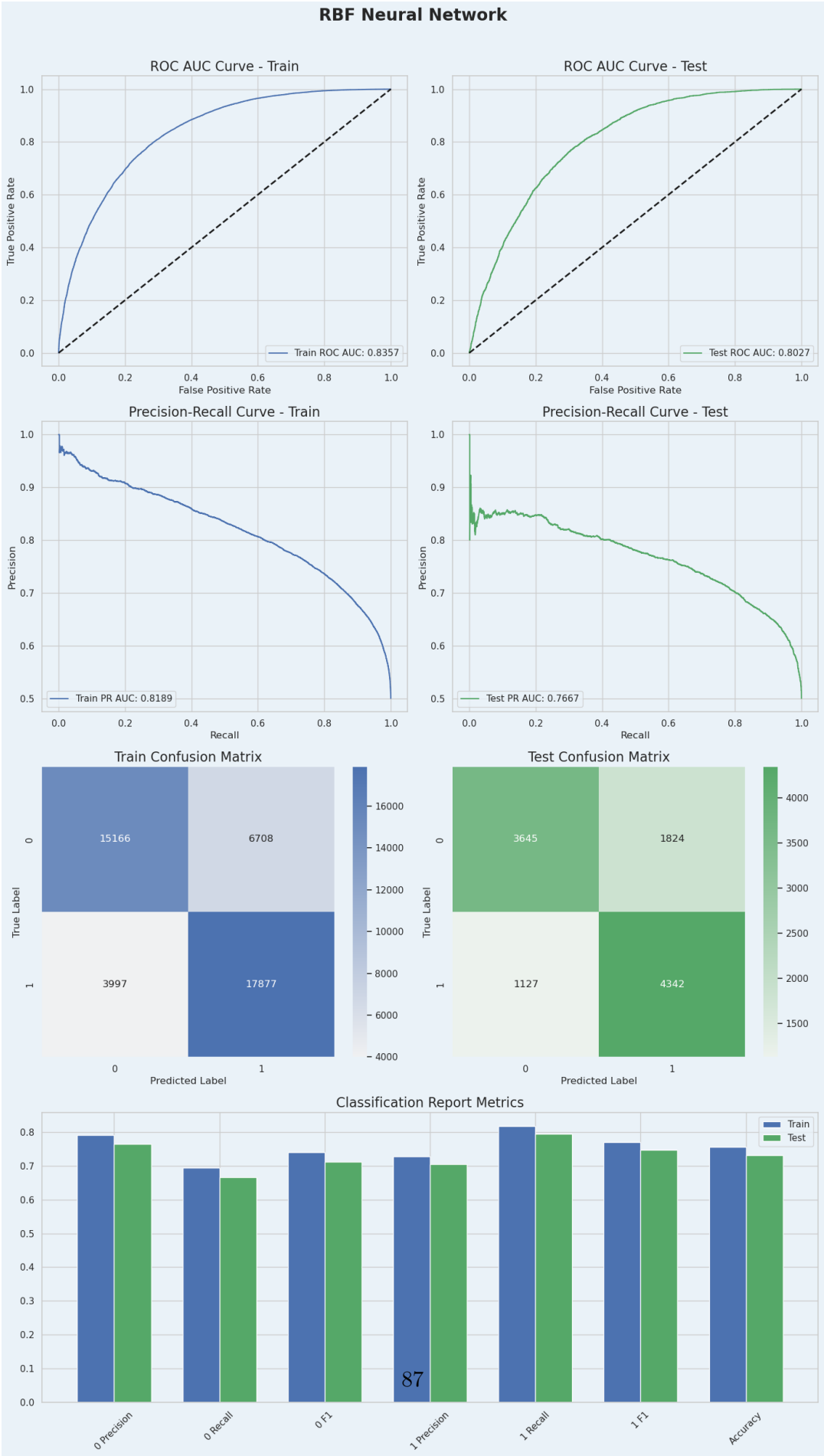
12.5 MLP

MLP Classifier as a Baseline for Diabetes Classification: I selected the Multi-Layer Perceptron (MLP) because it deciphers non-linear interactions in medical datasets. The tanh activation function and the adam solver are effective for managing the complexities inherent in this type of data.

[ ]:
```
from sklearn.neural_network import MLPClassifier

# Initialize the RBF Neural Network model
rbf_nn = MLPClassifier(hidden_layer_sizes=(100,), activation='tanh',␣
 ↪solver='adam', random_state=42)

# Train the model on the training data
rbf_nn.fit(uX_train, uy_train)
```

[ ]:
```
MLPClassifier(activation='tanh', random_state=42)
```

[ ]:
```
# Evaluate the model
evaluate_model(rbf_nn, uX_train, uy_train, uX_test, uy_test, 'MLP')
```


RBF Neural Network
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8357
ROC AUC Curve - Test
Test ROC AUC: 0.8027
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.8189
Precision-Recall Curve - Test
Test PR AUC: 0.7667
Train Confusion Matrix
15166
6708
3997
17877
True Label
Predicted Label
Test Confusion Matrix
3645
1824
1127
4342
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.76      0.67      0.71      5469
Test Class 1       0.70      0.79      0.75      5469

    accuracy                           0.73     10938
   macro avg       0.73      0.73      0.73     10938
weighted avg       0.73      0.73      0.73     10938
```

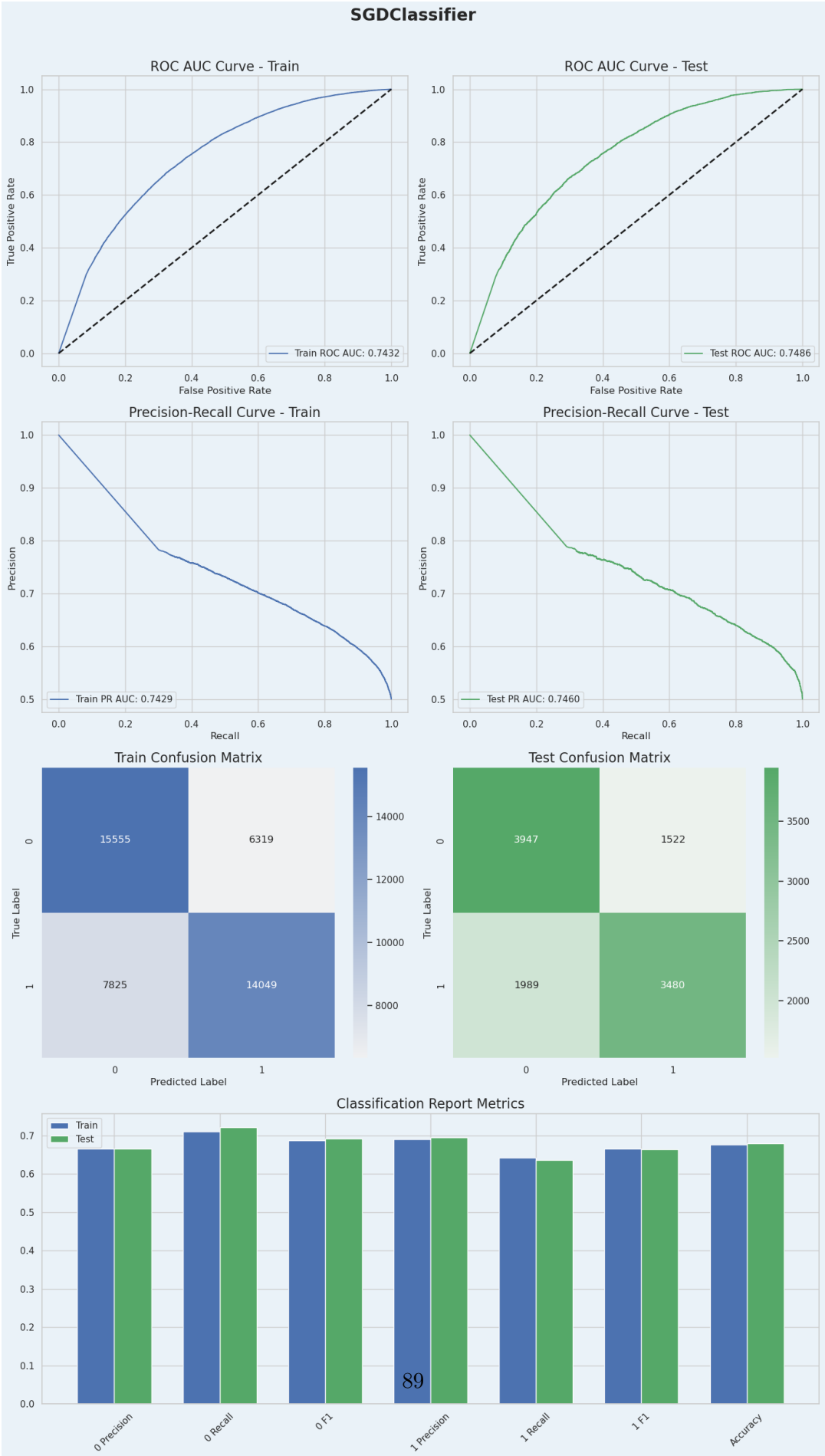
12.6 SGDClassifier

SGDClassifier (Logistic Regression via SGD): Uses stochastic gradient descent, which updates the model parameters using only a small subset of the data (a mini-batch) at each iteration. This process allows it to sometimes escape local minimization but it does introduce noise to the optimization. This was chosen in case there was a local minimum it might escape.

[ ]:
```
from sklearn.linear_model import SGDClassifier

# Initialize the SGDClassifier with 'log_loss' (equivalent to Logistic␣
↪Regression with SGD)
sgd_clf = SGDClassifier(loss='log_loss', penalty='l2', max_iter=1000,␣
↪random_state=42)

# Train the model on the training data
sgd_clf.fit(uX_train, uy_train)
```

[ ]:
```
SGDClassifier(loss='log_loss', random_state=42)
```

[ ]:
```
# Evaluate the model
evaluate_model(sgd_clf, uX_train, uy_train, uX_test, uy_test, 'SGDClassifier')
```


SGDClassifier
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.7432
ROC AUC Curve - Test
Test ROC AUC: 0.7486
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.7429
Precision-Recall Curve - Test
Test PR AUC: 0.7460
Train Confusion Matrix
15555
6319
7825
14049
True Label
Predicted Label
Test Confusion Matrix
3947
1522
1989
3480
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.66      0.72      0.69      5469
Test Class 1       0.70      0.64      0.66      5469

    accuracy                           0.68     10938
   macro avg       0.68      0.68      0.68     10938
weighted avg       0.68      0.68      0.68     10938
```

12.7 TabNet

I selected the TabNet model for its ability to handle tabular data using sequential attention mechanisms. TabNet selects the most relevant features at each decision step, improving prediction accuracy. This helps in datasets where feature selection significantly impacts performance.

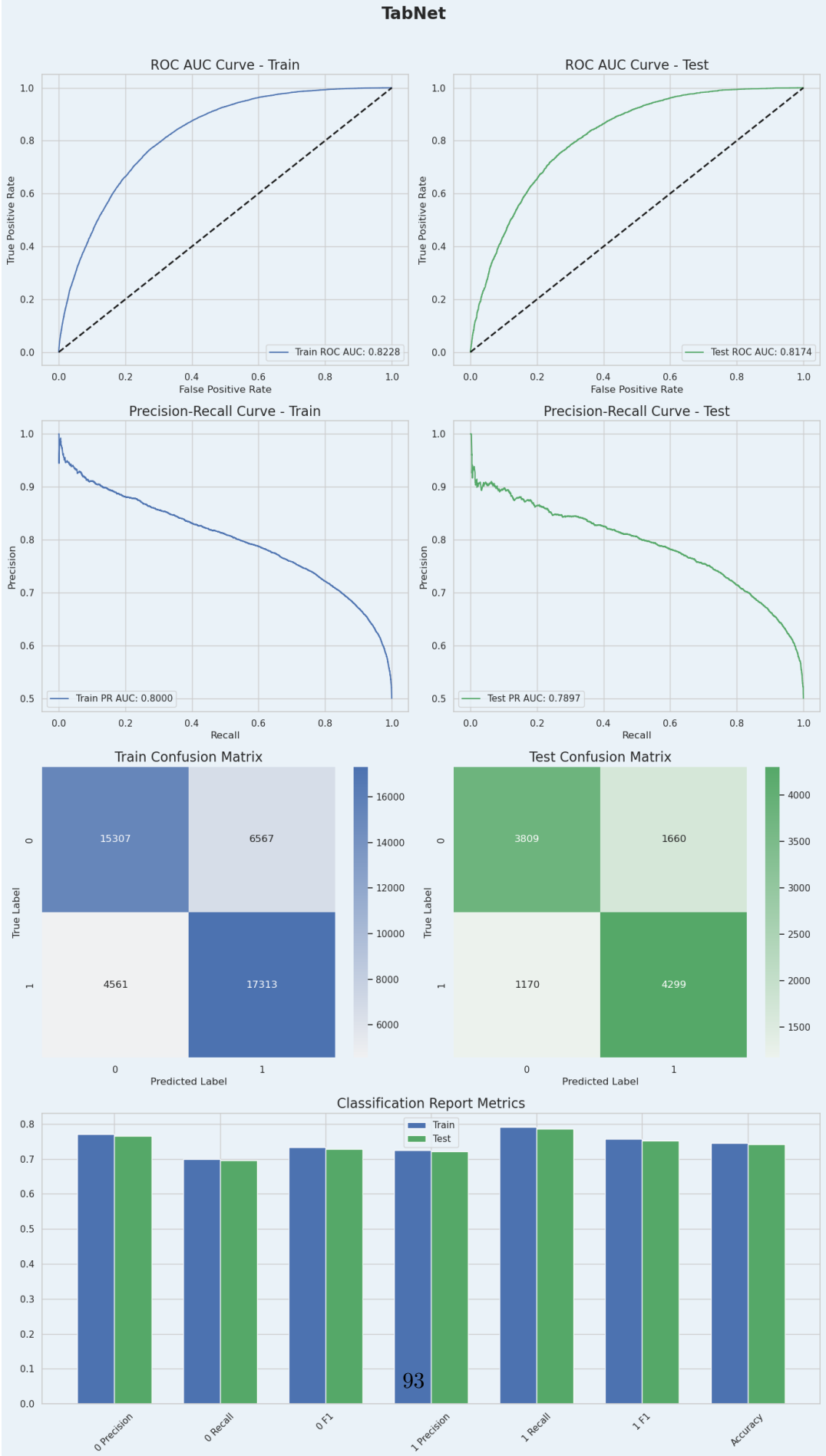
[ ]:
```
run_tabnet = True
if run_tabnet:
    # Create copies of the original data frames for TabNet
    uX_train_tabnet = uX_train.copy()
    uX_test_tabnet = uX_test.copy()

    # Convert all boolean columns to integers in the copies
    uX_train_tabnet = uX_train_tabnet.astype({col: 'uint8' for col in␣
 ↪uX_train_tabnet.select_dtypes('bool').columns})
    uX_test_tabnet = uX_test_tabnet.astype({col: 'uint8' for col in␣
 ↪uX_test_tabnet.select_dtypes('bool').columns})
```

[ ]:
```
if run_tabnet:
    # Flatten the target arrays to make them 1-dimensional
    uy_train_tabnet = uy_train.values.ravel()
    uy_test_tabnet = uy_test.values.ravel()

    # Initialize the TabNet model
    tabnet_clf = TabNetClassifier()

    # Train the model on the training data
    tabnet_clf.fit(uX_train_tabnet.values, uy_train_tabnet, max_epochs=50,␣
 ↪patience=5, batch_size=1024, virtual_batch_size=128)

    # Evaluate the model
    uy_test_pred = tabnet_clf.predict(uX_test_tabnet.values)
    uy_test_prob = tabnet_clf.predict_proba(uX_test_tabnet.values)[:, 1]
```

```
epoch 0  | loss: 0.65362 |  0:00:01s
epoch 1  | loss: 0.58002 |  0:00:02s
epoch 2  | loss: 0.55795 |  0:00:04s
epoch 3  | loss: 0.55013 |  0:00:05s
epoch 4  | loss: 0.54672 |  0:00:07s
epoch 5  | loss: 0.54086 |  0:00:08s
epoch 6  | loss: 0.53621 |  0:00:10s
epoch 7  | loss: 0.53521 |  0:00:11s
epoch 8  | loss: 0.53412 |  0:00:13s
epoch 9  | loss: 0.53187 |  0:00:14s
epoch 10 | loss: 0.53157 |  0:00:16s
epoch 11 | loss: 0.53032 |  0:00:17s
epoch 12 | loss: 0.53005 |  0:00:19s
epoch 13 | loss: 0.5302  |  0:00:20s
epoch 14 | loss: 0.52782 |  0:00:21s
epoch 15 | loss: 0.52736 |  0:00:23s
epoch 16 | loss: 0.52756 |  0:00:24s
epoch 17 | loss: 0.52541 |  0:00:26s
epoch 18 | loss: 0.5245  |  0:00:27s
epoch 19 | loss: 0.52548 |  0:00:29s
epoch 20 | loss: 0.52374 |  0:00:30s
epoch 21 | loss: 0.52337 |  0:00:32s
epoch 22 | loss: 0.52487 |  0:00:33s
epoch 23 | loss: 0.52258 |  0:00:35s
epoch 24 | loss: 0.52276 |  0:00:36s
epoch 25 | loss: 0.52277 |  0:00:38s
epoch 26 | loss: 0.5218  |  0:00:39s
epoch 27 | loss: 0.52212 |  0:00:41s
epoch 28 | loss: 0.52757 |  0:00:42s
epoch 29 | loss: 0.52403 |  0:00:43s
epoch 30 | loss: 0.52223 |  0:00:45s
epoch 31 | loss: 0.52104 |  0:00:46s
epoch 32 | loss: 0.52067 |  0:00:48s
epoch 33 | loss: 0.52097 |  0:00:49s
epoch 34 | loss: 0.52049 |  0:00:51s
epoch 35 | loss: 0.52041 |  0:00:52s
epoch 36 | loss: 0.51974 |  0:00:53s
epoch 37 | loss: 0.51943 |  0:00:55s
epoch 38 | loss: 0.51933 |  0:00:56s
epoch 39 | loss: 0.51817 |  0:00:58s
epoch 40 | loss: 0.5189  |  0:00:59s
epoch 41 | loss: 0.51863 |  0:01:01s
epoch 42 | loss: 0.51826 |  0:01:02s
epoch 43 | loss: 0.51735 |  0:01:04s
epoch 44 | loss: 0.51776 |  0:01:05s
epoch 45 | loss: 0.51702 |  0:01:07s
epoch 46 | loss: 0.51693 |  0:01:08s
epoch 47 | loss: 0.5175  |  0:01:10s
```

```
epoch 48 | loss: 0.51641 |  0:01:11s
epoch 49 | loss: 0.51654 |  0:01:12s
```

[ ]:
```
if run_tabnet:
    evaluate_model(tabnet_clf, uX_train_tabnet.values, uy_train_tabnet,␣
 ↪uX_test_tabnet.values, uy_test_tabnet, 'TabNet')
```


TabNet
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8228
ROC AUC Curve - Test
Test ROC AUC: 0.8174
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.8000
Precision-Recall Curve - Test
Test PR AUC: 0.7897
Train Confusion Matrix
15307
6567
4561
17313
True Label
Predicted Label
Test Confusion Matrix
3809
1660
1170
4299
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.77      0.70      0.73      5469
Test Class 1       0.72      0.79      0.75      5469

    accuracy                           0.74     10938
   macro avg       0.74      0.74      0.74     10938
weighted avg       0.74      0.74      0.74     10938
```

12.8 Stacking TabNet to CatBoost

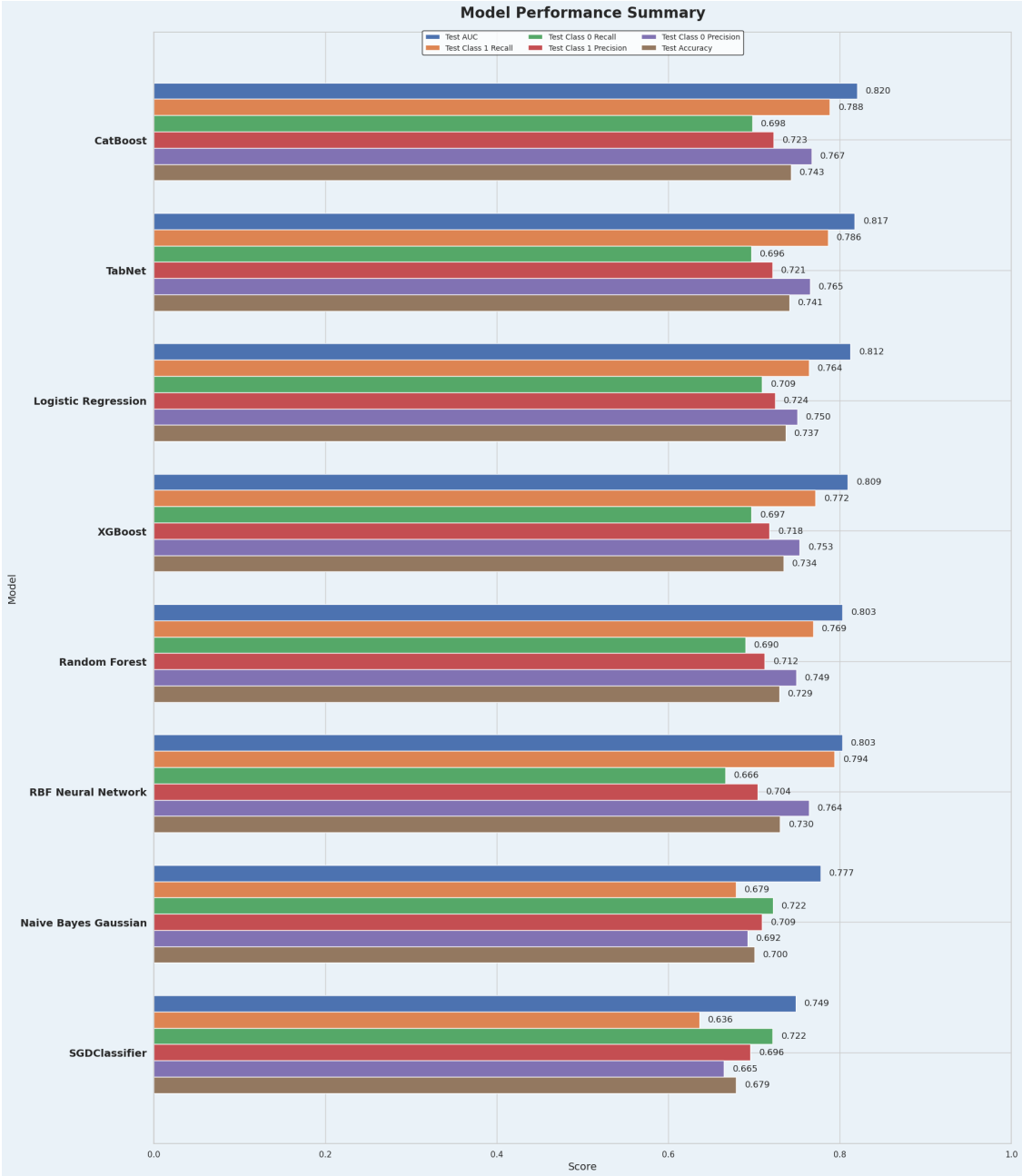
I chose to stack TabNet and CatBoost to combine their strengths of gradient boosting and selecting features through an attention mechanism.

[ ]:
```
run_stacked = False
if run_stacked:
    # Step 1: Train TabNet
    tabnet_clf = TabNetClassifier()
    tabnet_clf.fit(uX_train_tabnet.values, uy_train_tabnet, max_epochs=50,␣
 ↪patience=20, batch_size=1024, virtual_batch_size=128)

    # Step 2: Generate TabNet predictions
    train_tabnet_preds = tabnet_clf.predict_proba(uX_train_tabnet.values)[:, 1]
    test_tabnet_preds = tabnet_clf.predict_proba(uX_test_tabnet.values)[:, 1]

    # Step 3: Add TabNet predictions as a new feature to the original features
    uX_train_stacked = pd.DataFrame(uX_train_tabnet)
    uX_train_stacked['TabNet_Preds'] = train_tabnet_preds

    uX_test_stacked = pd.DataFrame(uX_test_tabnet)
    uX_test_stacked['TabNet_Preds'] = test_tabnet_preds

    # Step 4: Train CatBoost on the stacked features
    catboost_clf = CatBoostClassifier(iterations=1000, depth=6, learning_rate=0.
 ↪1, eval_metric='Recall', random_seed=42)
    catboost_clf.fit(uX_train_stacked, uy_train_tabnet,␣
 ↪eval_set=(uX_test_stacked, uy_test_tabnet), verbose=100,␣
 ↪early_stopping_rounds=100)
```

[ ]:
```
if run_stacked:
    # Step 5: Evaluate the stacked model
    evaluate_model(catboost_clf, uX_train_stacked, uy_train_tabnet,␣
 ↪uX_test_stacked, uy_test_tabnet, 'Stacked TabNet to CatBoost')
```

12.9 Evaluation Summary Visualizations

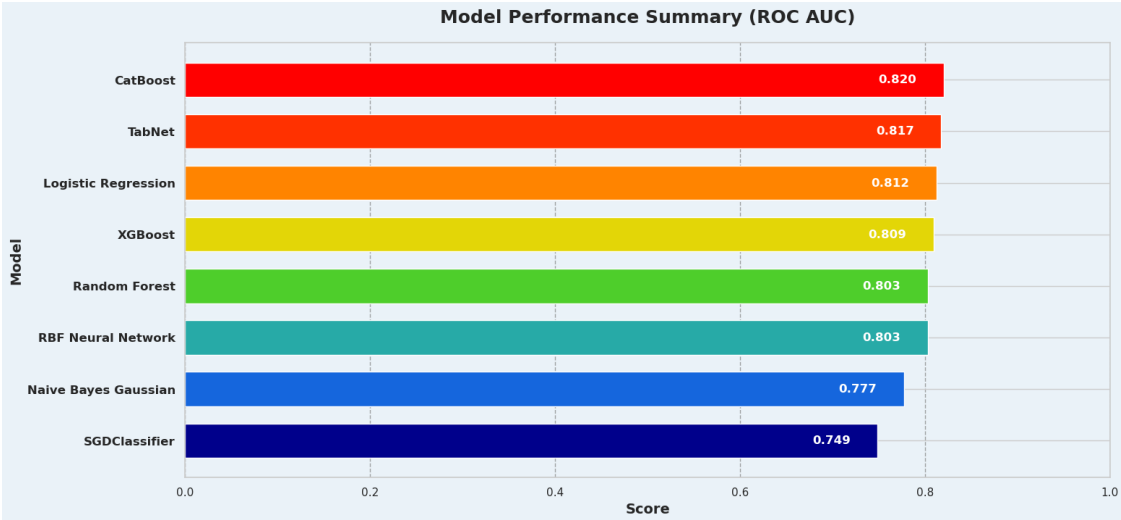
[ ]:
```
evaluation_summary()
```



12.9.1 Simplified

[ ]:
```
baseline_model_list = ['Logistic Regression', 'Random Forest', 'XGBoost',␣
↪'CatBoost', 'Naive Bayes Gaussian', 'RBF Neural Network',␣
↪'SGDClassifier','TabNet']
baseline_model_dict = {key: model_performance[key] for key in␣
↪baseline_model_list if key in model_performance}
#baseline_model_dict
```

[ ]:
```
evaluation_summary_simplified(baseline_model_dict)
```



13 Hypertuned Models

Now that we know which models seem to fit the problem best from using a wide selection, lets pick the top 3: CatBoost, Logistic and TabNet. Then we can try stacking them later to see if it improves things.

13.1 Optuna Objective

I will use optuna to find the optimal params in a way GridSearchCV can't, it also usually finishes much quicker. The only downside is sometimes it will miss things an exhaustive search doesn't. However, Optuna can search over a wider range and find what is working and what isn't. This makes it powerful to search over a much larger space than you could do with GridSearch in anywhere near the same time.

Not every model is good with the optuna setup, and you can get a lot of convergence warnings. Logistic tends to do poor with it for me, but Catboost is really easy. I will use GridSearchCV Iteratively to find the best logistic params.

13.2 CatBoost

13.2.1 Flag

[ ]:
```
use_optimal_catboost = True
```

13.2.2 Catboost Objective

[ ]:
```
def catboost_objective(trial):
    try:
        # Define the hyperparameter search space
        param = {
            'iterations': trial.suggest_int('iterations', 500, 2000),
            'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3,␣
↪log=True),
            'depth': trial.suggest_int('depth', 4, 10),
            'bagging_temperature': trial.suggest_float('bagging_temperature', 0.
↪0, 1.0),
            'border_count': trial.suggest_int('border_count', 32, 255),
            'random_strength': trial.suggest_float('random_strength', 1e-9, 10.
↪0, log=True),
            'eval_metric': 'F1',  # Use F1 directly since Recall might not be␣
↪available
            'loss_function': 'Logloss',  # Specify the loss function for␣
↪cross-validation
            'task_type': "GPU",  # Ensures GPU usage
            'random_seed': 42,  # Ensures reproducibility
            'logging_level': 'Silent',  # Suppresses CatBoost output
            'early_stopping_rounds': 100  # Enable early stopping
        }

        # Conditionally add 'l2_leaf_reg' based on a binary choice
        if trial.suggest_categorical('use_l2_leaf_reg', [True, False]):
            param['l2_leaf_reg'] = trial.suggest_float('l2_leaf_reg', 1e-5, 10.
↪0, log=True)

        # Create the CatBoost Pool object for training data
        train_pool = Pool(uX_train, uy_train)

        # Perform cross-validation using CatBoost's built-in function
        cv_results = cv(
            params=param,
            pool=train_pool,
            fold_count=5,  # Number of folds
            shuffle=True,  # Shuffle the dataset before splitting
            partition_random_seed=42,  # Ensures reproducibility
            stratified=True,  # For classification tasks
```

```
            verbose=False,  # Suppress output
            plot=False  # Disable plotting
        )

        # Get the mean of the cross-validated F1 score
        mean_f1 = np.max(cv_results['test-F1-mean'])

        return mean_f1

    except FloatingPointError:
        return float('nan')  # Return NaN to skip this trial
```

13.2.3 Study

[ ]:
```
if use_optimal_catboost:
    best_params = {
        'iterations': 1124,
        'learning_rate': 0.04133450880038945,
        'depth': 8,
        'bagging_temperature': 0.2633451677082204,
        'border_count': 39,
        'random_strength': 0.0005954796194102114,
        'use_l2_leaf_reg': False
    }
else:
    # Create an Optuna study to maximize the scoring metric
    study = optuna.create_study(direction="maximize")
    #more than 30 gets a lot of errors
    study.optimize(catboost_objective, n_trials=30, timeout=3600)
    best_params = study.best_params

print("Best Hyperparameters: ", best_params)
```

```
Best Hyperparameters:  {'iterations': 1124, 'learning_rate':
0.04133450880038945, 'depth': 8, 'bagging_temperature': 0.2633451677082204,
'border_count': 39, 'random_strength': 0.0005954796194102114, 'use_l2_leaf_reg':
False}
```

13.2.4 Train Best

[ ]:
```
# Display the best hyperparameters
display(best_params)
if 'use_l2_leaf_reg' in best_params.keys():
    del best_params['use_l2_leaf_reg']

# Note: Cross-validation has already been performed during hyperparameter␣
↪tuning
```

```
# to identify the best parameters. There's no need to perform CV again here.
# Instead, we will directly train the final model on the entire training␣
↪dataset
# using the best hyperparameters.

# Train the model on the full dataset using the best hyperparameters
best_catboost_model = CatBoostClassifier(
    **best_params,  # Unpack the best hyperparameters
    eval_metric='Recall',  # Evaluation metric
    loss_function='Logloss',  # Specify the loss function for cross-validation
    task_type="GPU",  # Ensures GPU usage
    random_seed=42,  # Ensures reproducibility
    logging_level='Silent',  # Suppresses CatBoost output
    early_stopping_rounds=100  # Enable early stopping
)

# Fit the model on the full training data
best_catboost_model.fit(uX_train, uy_train, eval_set=(uX_test, uy_test),␣
↪verbose=100, early_stopping_rounds=100)
```

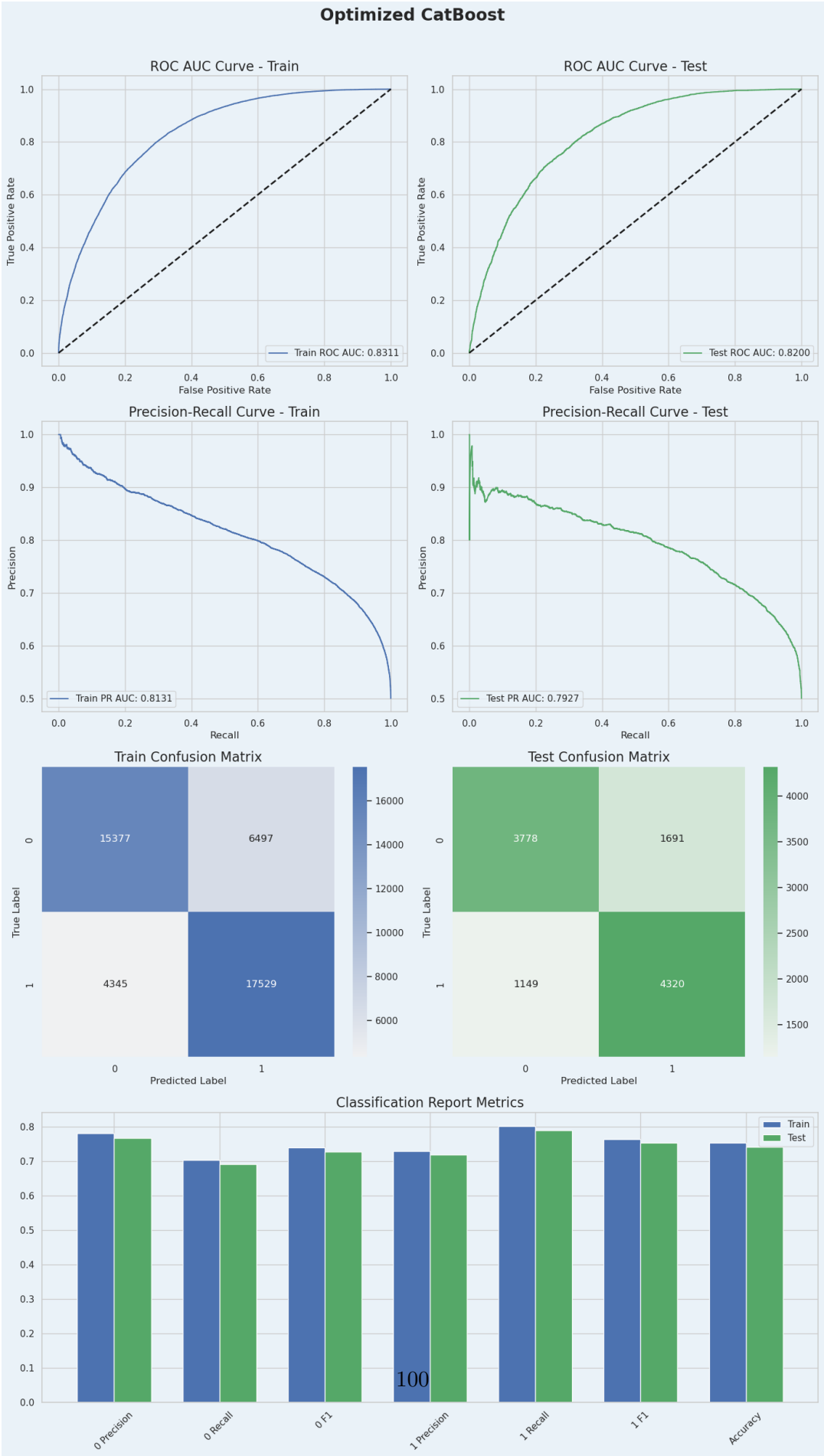
```
{'iterations': 1124,
 'learning_rate': 0.04133450880038945,
 'depth': 8,
 'bagging_temperature': 0.2633451677082204,
 'border_count': 39,
 'random_strength': 0.0005954796194102114,
 'use_l2_leaf_reg': False}
```

```
Warning: less than 75% gpu memory available for training. Free: 678.8125 Total:
3803.1875
```

[ ]:
```
<catboost.core.CatBoostClassifier at 0x7ca5b3410610>
```

13.2.5 Evaluate

[ ]:
```
evaluate_model(best_catboost_model, uX_train, uy_train, uX_test, uy_test,␣
↪'Optimized CatBoost')
```


Optimized CatBoost
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8311
ROC AUC Curve - Test
Test ROC AUC: 0.8200
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.8131
Precision-Recall Curve - Test
Test PR AUC: 0.7927
Train Confusion Matrix
15377
6497
4345
17529
True Label
Predicted Label
Test Confusion Matrix
3778
1691
1149
4320
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.77      0.69      0.73      5469
Test Class 1       0.72      0.79      0.75      5469

    accuracy                           0.74     10938
   macro avg       0.74      0.74      0.74     10938
weighted avg       0.74      0.74      0.74     10938
```

13.3 CatBoost Full Data

I wanted to test if class weights would perform better than using the undersampled data set.

13.3.1 Flag

[ ]:
```
fd_use_optimal_catboost = True
```

13.3.2 Catboost Objective

[ ]:
```
def fd_catboost_objective(trial):
    try:
        # Define the hyperparameter search space
        param = {
            'iterations': trial.suggest_int('iterations', 500, 2000),
            'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3,␣
 ↪log=True),
            'depth': trial.suggest_int('depth', 4, 10),
            'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1e-5, 10.0,␣
 ↪log=True),
            'bagging_temperature': trial.suggest_float('bagging_temperature', 0.
 ↪0, 1.0),
            'border_count': trial.suggest_int('border_count', 32, 255),
            'random_strength': trial.suggest_float('random_strength', 1e-9, 10.
 ↪0, log=True),
            #'rsm': trial.suggest_float('rsm', 0.5, 1.0),  # Subspace of␣
 ↪features used per iteration
            'bootstrap_type': trial.suggest_categorical('bootstrap_type',␣
 ↪['Poisson']),
            'subsample': trial.suggest_float('subsample', 0.5, 1.0),  #␣
 ↪Fraction of samples per iteration
            'eval_metric': 'Recall',
            'loss_function': 'Logloss',
```

```
            'task_type': "GPU",  # Ensures GPU usage
            'random_seed': 42,  # Ensures reproducibility
            'logging_level': 'Silent',  # Suppresses CatBoost output
            'early_stopping_rounds': 100,  # Enable early stopping
        }

        # Add auto_class_weights only if it's set to 'Balanced'
        auto_class_weights = trial.suggest_categorical('auto_class_weights',␣
↪['Balanced', None])
        if auto_class_weights == 'Balanced':
            param['auto_class_weights'] = 'Balanced'

        # Optionally remove l2_leaf_reg based on some condition
        use_l2_leaf_reg = trial.suggest_categorical('use_l2_leaf_reg', [True,␣
↪False])
        if not use_l2_leaf_reg:
            del param['l2_leaf_reg']

        # Create the CatBoost Pool object for training data
        train_pool = Pool(X_train, y_train)

        # Perform cross-validation using CatBoost's built-in function
        cv_results = cv(
            params=param,
            pool=train_pool,
            fold_count=5,  # Number of folds
            shuffle=True,  # Shuffle the dataset before splitting
            partition_random_seed=42,  # Ensures reproducibility
            stratified=True,  # For classification tasks
            verbose=False,  # Suppress output
            plot=False  # Disable plotting
        )

        # Get the mean of the cross-validated metric (in this case, recall)
        mean_recall = np.max(cv_results['test-Recall-mean'])

        return mean_recall

    except FloatingPointError:
        return float('nan')  # Return NaN to skip this trial
```

13.3.3 Study

[ ]:
```
if fd_use_optimal_catboost:
    best_params = {
        'iterations': 1328,
```

```
        'learning_rate': 0.2398037404808512,
        'depth': 8,
        'l2_leaf_reg': 0.0020016866010072057,
        'bagging_temperature': 0.6854454937290867,
        'border_count': 32,
        'random_strength': 2.588526332596195e-09,
        'bootstrap_type': 'Poisson',
        'subsample': 0.6802590715134494,
        'auto_class_weights': 'Balanced',
        'use_l2_leaf_reg': True
    }
else:
    # Create an Optuna study to maximize the recall
    study = optuna.create_study(direction="maximize")
    #more than 30 gets a lot of errors
    study.optimize(fd_catboost_objective, n_trials=30, timeout=3600)
    best_params = study.best_params

#best_params['class_weights'] = [1, 7]  # Ensuring class weights are set

print("Best Hyperparameters: ", best_params)
```

```
Best Hyperparameters:  {'iterations': 1328, 'learning_rate': 0.2398037404808512,
'depth': 8, 'l2_leaf_reg': 0.0020016866010072057, 'bagging_temperature':
0.6854454937290867, 'border_count': 32, 'random_strength':
2.588526332596195e-09, 'bootstrap_type': 'Poisson', 'subsample':
0.6802590715134494, 'auto_class_weights': 'Balanced', 'use_l2_leaf_reg': True}
```

13.3.4 Train Best

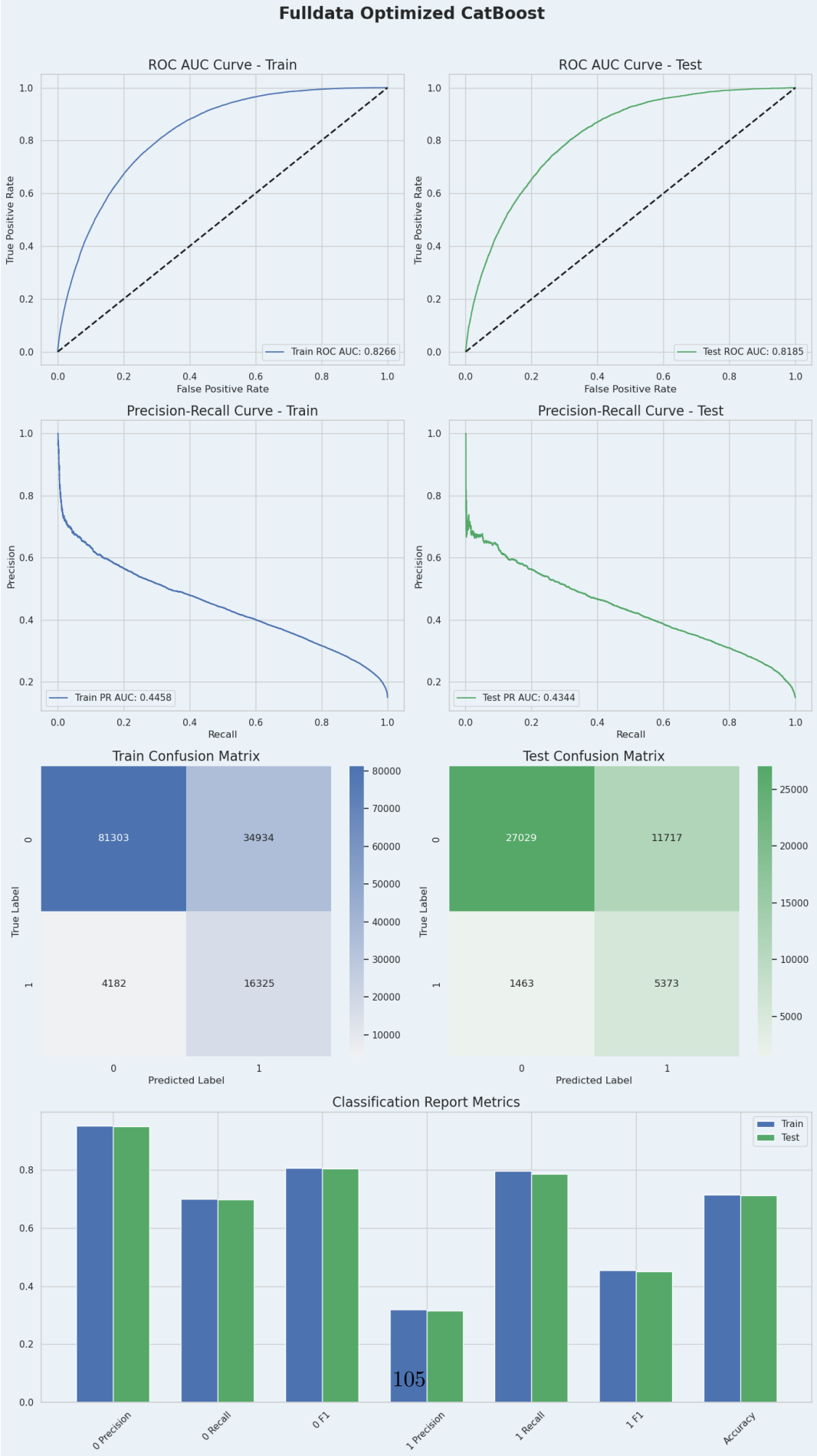
[ ]:
```
if 'use_l2_leaf_reg' in best_params:
    del best_params['use_l2_leaf_reg']
# Train the model on the full dataset using the best hyperparameters
fd_best_catboost_model = CatBoostClassifier(
    **best_params,  # Unpack the best hyperparameters
    eval_metric='Recall:use_weights=True',  # Evaluation metric
    task_type="GPU",  # Ensures GPU usage
    random_seed=42,  # Ensures reproducibility
    logging_level='Silent',  # Suppresses CatBoost output
    early_stopping_rounds=100  # Enable early stopping
)

# Fit the model on the full training data
fd_best_catboost_model.fit(X_train, y_train, eval_set=(X_test, y_test),␣
 ↪verbose=100, early_stopping_rounds=100)
```

```
Warning: less than 75% gpu memory available for training. Free: 676.8125 Total:
3803.1875
```

[ ]: <catboost.core.CatBoostClassifier at 0x7ca5b2a4d8a0>

13.3.5 Evaluate

[ ]:
```
evaluate_model(fd_best_catboost_model, X_train, y_train, X_test, y_test,␣
↪'Fulldata Optimized CatBoost')
```


Fulldata Optimized CatBoost
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8266
ROC AUC Curve - Test
Test ROC AUC: 0.8185
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.4458
Precision-Recall Curve - Test
Test PR AUC: 0.4344
Train Confusion Matrix
81303
34934
4182
16325
True Label
Predicted Label
Test Confusion Matrix
27029
11717
1463
5373
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.95      0.70      0.80     38746
Test Class 1       0.31      0.79      0.45      6836

    accuracy                           0.71     45582
   macro avg       0.63      0.74      0.63     45582
weighted avg       0.85      0.71      0.75     45582
```

13.4 Logistic Tuned

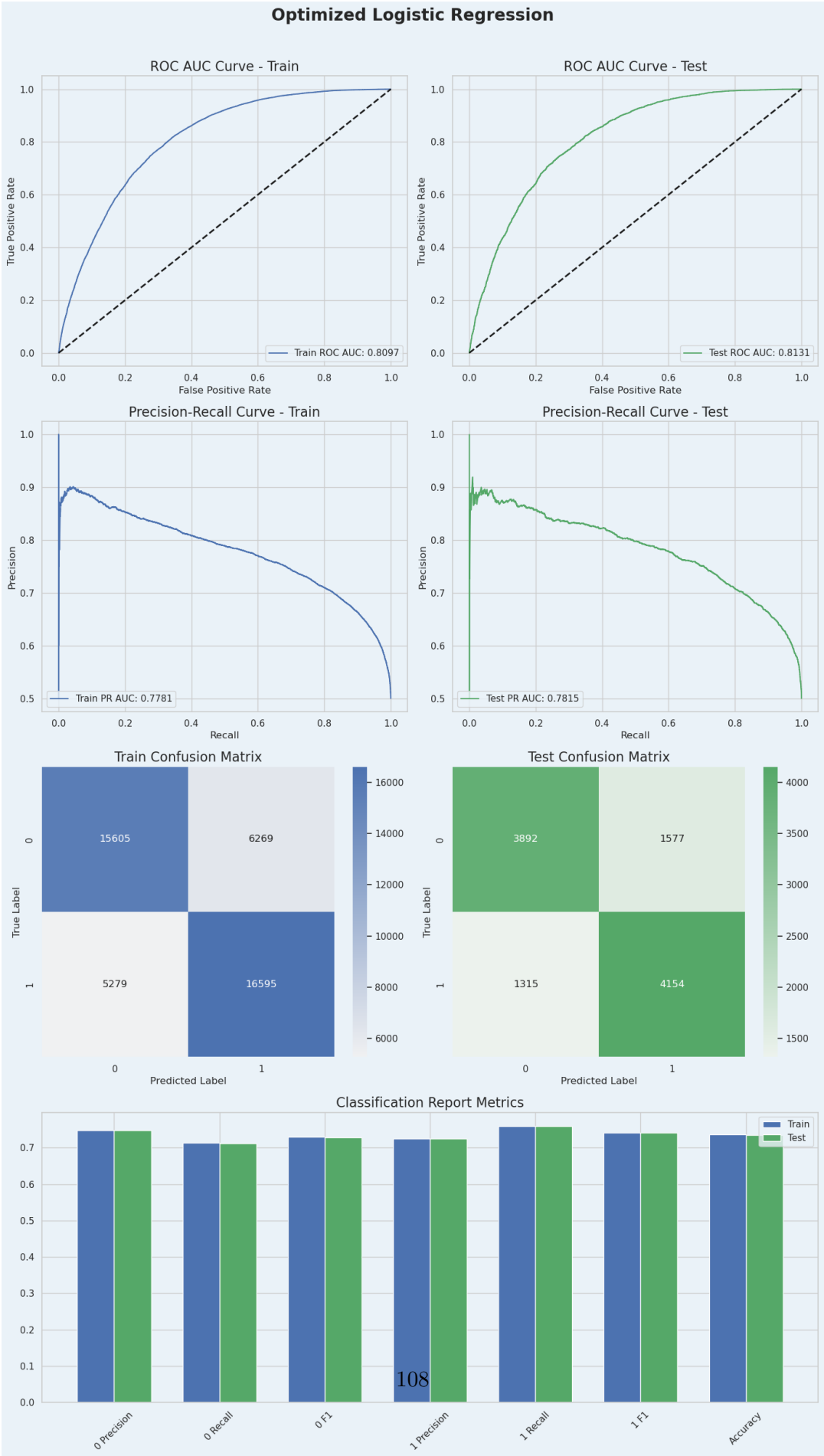
GridSearchCV is used instead of optuna as it seems to have a lot of issues with the optuna study I wrote for it.

[ ]:
```
log_use_optimal_params = True
if log_use_optimal_params:
    # Iterated down to these values after many grid searches
    param_grid = {
        'model__C': [0.01],  # Single value in a list
        'model__l1_ratio': [0.3],  # Single value in a list
        'model__max_iter': [100],  # Single value in a list
        'model__penalty': ['elasticnet'],  # Single value in a list
        'scaler': [StandardScaler()]  # Single instance in a list
    }
else:
    # Full Search, over 15 mins to run
    param_grid = [
        {
            'scaler': [None, StandardScaler(), MinMaxScaler(), RobustScaler()],
            'model__penalty': ['none'],
            'model__C': [1],  # C is irrelevant when penalty is 'none', but␣
 ↪must be set
            'model__max_iter': [100, 500,1000]
        },
        {
            'scaler': [None, StandardScaler(), MinMaxScaler(), RobustScaler()],
            'model__penalty': ['l1', 'l2'],
            'model__C': [0.01, 0.1, 1, 10, 100],
            'model__max_iter': [100, 500, 1000]
        },
        {
            'scaler': [None, StandardScaler(), MinMaxScaler(), RobustScaler()],
            'model__penalty': ['elasticnet'],
```

```
            'model__C': [0.01, 0.1, 1, 10, 100],
            'model__max_iter': [100, 500, 1000],
            'model__l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
        }
    ]

# Create a pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Placeholder for scaler
    ('model', LogisticRegression(solver='saga', random_state=42))  # Use saga␣
 ↪to handle elasticnet
])

# Initialize the grid search
grid_search = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring='roc_auc', error_score=np.nan)

# Fit the grid search
grid_search.fit(uX_train, uy_train.values.ravel())  # Ensuring uy_train is␣
 ↪flattened correctly

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)

# Evaluate on the test set
tuned_logistic_model = grid_search.best_estimator_
test_score = tuned_logistic_model.score(uX_test, uy_test.values.ravel())
print("Test Set Score:", test_score)
```

```
Best Parameters: {'model__C': 0.01, 'model__l1_ratio': 0.3, 'model__max_iter':
100, 'model__penalty': 'elasticnet', 'scaler': StandardScaler()}
Best Cross-Validation Score: 0.8090617159107527
Test Set Score: 0.7356006582556226
```

[ ]:
```
# Evaluate the model using your evaluate_model function
evaluate_model(tuned_logistic_model, uX_train, uy_train, uX_test, uy_test,␣
 ↪'Optimized Logistic Regression')
```


Optimized Logistic Regression
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8097
ROC AUC Curve - Test
Test ROC AUC: 0.8131
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.7781
Precision-Recall Curve - Test
Test PR AUC: 0.7815
Train Confusion Matrix
15605
6269
5279
16595
True Label
Predicted Label
Test Confusion Matrix
3892
1577
1315
4154
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.75      0.71      0.73      5469
Test Class 1       0.72      0.76      0.74      5469

    accuracy                           0.74     10938
   macro avg       0.74      0.74      0.74     10938
weighted avg       0.74      0.74      0.74     10938
```

13.5 TabNet Tuned

TabNet takes a lot of time to train so utlizing Optuna smartly was key.

13.5.1 TabNet Format

[ ]:
```
use_best_params_tabnet = True
# Prepare the data for TabNet
uX_train_tabnet = uX_train.copy()
uX_test_tabnet = uX_test.copy()

# Convert all boolean columns to integers in the copies
uX_train_tabnet = uX_train_tabnet.astype({col: 'uint8' for col in␣
 ↪uX_train_tabnet.select_dtypes('bool').columns})
uX_test_tabnet = uX_test_tabnet.astype({col: 'uint8' for col in uX_test_tabnet.
 ↪select_dtypes('bool').columns})

# Flatten the target arrays to make them 1-dimensional
uy_train_tabnet = uy_train.values.ravel()
uy_test_tabnet = uy_test.values.ravel()
```

13.5.2 TabNet Objective

[ ]:
```
def tabnet_objective(trial):
    try:
        # Suggest hyperparameters for TabNet
        n_d = trial.suggest_int('n_d', 8, 64)
        n_a = trial.suggest_int('n_a', 8, 64)
        n_steps = trial.suggest_int('n_steps', 3, 10)
        gamma = trial.suggest_float('gamma', 1.0, 2.0)
        lambda_sparse = trial.suggest_float('lambda_sparse', 1e-6, 1e-3)
        momentum = trial.suggest_float('momentum', 0.01, 0.4)
        learning_rate = trial.suggest_float('learning_rate', 1e-4, 1e-1,␣
 ↪log=True)
```

```
        tabnet_clf = TabNetClassifier(
            n_d=n_d,
            n_a=n_a,
            n_steps=n_steps,
            gamma=gamma,
            lambda_sparse=lambda_sparse,
            momentum=momentum,
            optimizer_params=dict(lr=learning_rate),  # Use the suggested␣
↪learning rate
            seed=42
        )

        # Train the model with early stopping
        tabnet_clf.fit(
            uX_train_tabnet.values, uy_train_tabnet,
            eval_set=[(uX_test_tabnet.values, uy_test_tabnet)],
            eval_name=['val'],
            eval_metric=['auc'],
            max_epochs=200, patience=20,
            batch_size=1024, virtual_batch_size=128,
            num_workers=0,
            drop_last=False
        )

        # Predict on the test set
        uy_test_prob = tabnet_clf.predict_proba(uX_test_tabnet.values)[:, 1]

        # Calculate and return the evaluation metric
        score = roc_auc_score(uy_test_tabnet, uy_test_prob)
        return score

    except Exception as e:
        # Handle any exceptions that occur during the training or evaluation
        print(f"An error occurred: {e}")
        return float('nan')  # Return NaN to indicate the trial failed
```

13.5.3 Study

[ ]:
```
if not use_best_params_tabnet:
    # Create an Optuna study and optimize
    study = optuna.create_study(direction='maximize')
    study.optimize(tabnet_objective, n_trials=50)

    # Best parameters and score
    print("Best Parameters:", study.best_params)
    print("Best ROC AUC Score:", study.best_value)
```

13.5.4 Train Best

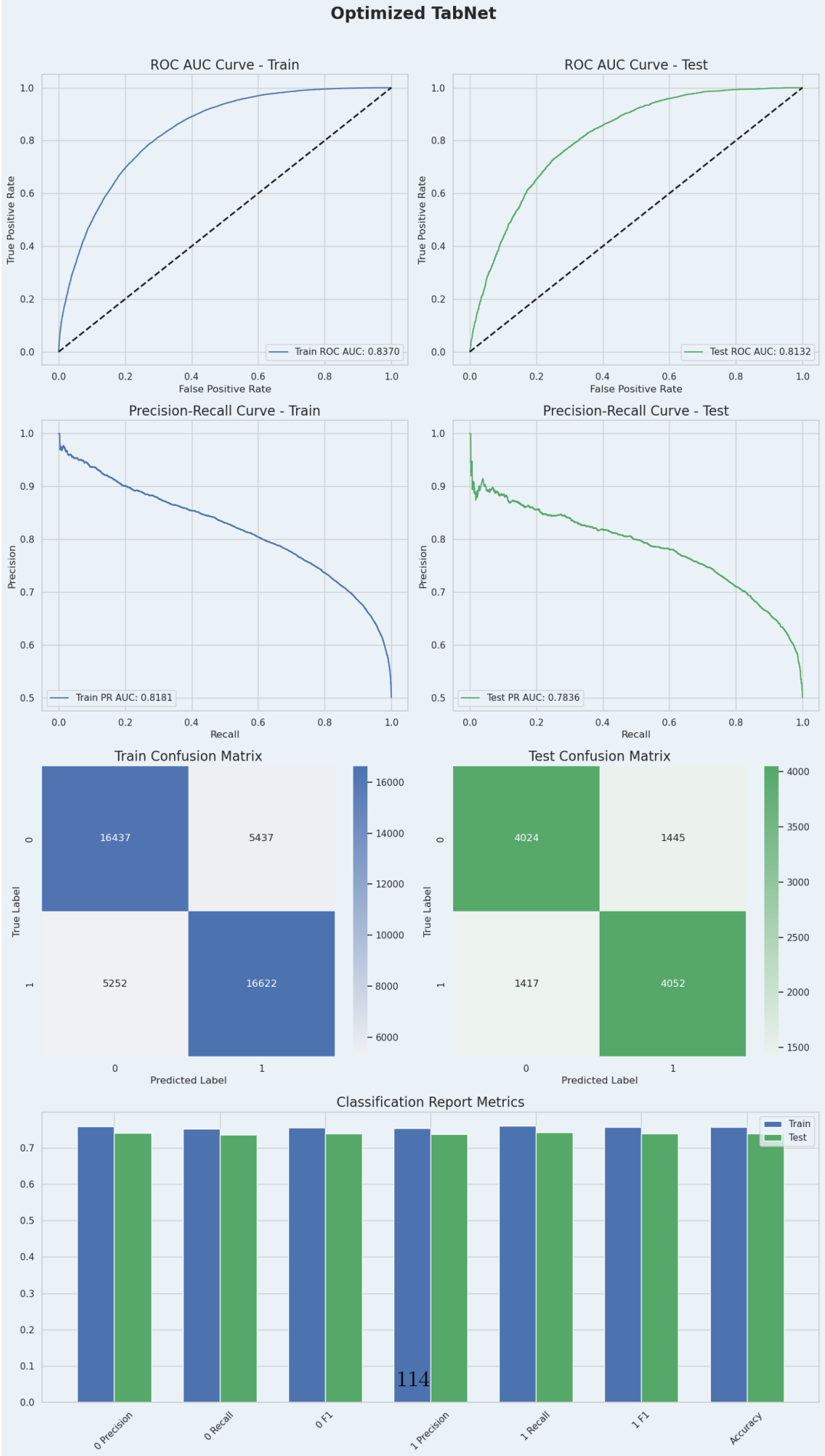
[ ]:
```
if use_best_params_tabnet:
    best_params = {
        'n_d': 53,
        'n_a': 53,
        'n_steps': 4,
        'gamma': 1.5677023508488426,
        'lambda_sparse': 7.369693723068355e-05,
        'momentum': 0.3130307693656363,
        'learning_rate': 0.06790698411953208
    }
else:
    best_params = study.best_params

# Provide default values for n_independent and n_shared if not found
n_independent = best_params.get('n_independent', 2)  # Default to 2 if not found
n_shared = best_params.get('n_shared', 2)  # Default to 2 if not found

# Initialize the TabNet model with the best hyperparameters
best_tabnet_clf = TabNetClassifier(
    n_d=best_params['n_d'],
    n_a=best_params['n_a'],
    n_steps=best_params['n_steps'],
    gamma=best_params['gamma'],
    lambda_sparse=best_params['lambda_sparse'],
    optimizer_params=dict(lr=best_params['learning_rate']),
    n_independent=n_independent,
    n_shared=n_shared,
    seed=42
)

# Train the model on the entire training data
best_tabnet_clf.fit(
    uX_train_tabnet.values, uy_train_tabnet,
    max_epochs=50, patience=20,
    batch_size=1024, virtual_batch_size=128
)
```

```
epoch 0  | loss: 0.75436 |  0:00:02s
epoch 1  | loss: 0.55815 |  0:00:04s
epoch 2  | loss: 0.55535 |  0:00:06s
epoch 3  | loss: 0.54998 |  0:00:09s
epoch 4  | loss: 0.54265 |  0:00:11s
epoch 5  | loss: 0.54047 |  0:00:14s
epoch 6  | loss: 0.53318 |  0:00:17s
epoch 7  | loss: 0.53162 |  0:00:19s
epoch 8  | loss: 0.5269  |  0:00:22s
epoch 9  | loss: 0.52772 |  0:00:25s
epoch 10 | loss: 0.52642 |  0:00:27s
epoch 11 | loss: 0.52823 |  0:00:30s
epoch 12 | loss: 0.52808 |  0:00:32s
epoch 13 | loss: 0.5321  |  0:00:35s
epoch 14 | loss: 0.52513 |  0:00:38s
epoch 15 | loss: 0.52531 |  0:00:40s
epoch 16 | loss: 0.52572 |  0:00:43s
epoch 17 | loss: 0.52697 |  0:00:45s
epoch 18 | loss: 0.52406 |  0:00:48s
epoch 19 | loss: 0.5219  |  0:00:51s
epoch 20 | loss: 0.5208  |  0:00:53s
epoch 21 | loss: 0.52002 |  0:00:56s
epoch 22 | loss: 0.52052 |  0:00:58s
epoch 23 | loss: 0.51819 |  0:01:01s
epoch 24 | loss: 0.51904 |  0:01:04s
epoch 25 | loss: 0.51738 |  0:01:06s
epoch 26 | loss: 0.51812 |  0:01:09s
epoch 27 | loss: 0.52241 |  0:01:12s
epoch 28 | loss: 0.51824 |  0:01:14s
epoch 29 | loss: 0.51952 |  0:01:17s
epoch 30 | loss: 0.51864 |  0:01:20s
epoch 31 | loss: 0.51834 |  0:01:22s
epoch 32 | loss: 0.51682 |  0:01:25s
epoch 33 | loss: 0.51687 |  0:01:27s
epoch 34 | loss: 0.51518 |  0:01:30s
epoch 35 | loss: 0.51652 |  0:01:33s
epoch 36 | loss: 0.51515 |  0:01:35s
epoch 37 | loss: 0.51346 |  0:01:38s
epoch 38 | loss: 0.51385 |  0:01:41s
epoch 39 | loss: 0.51159 |  0:01:43s
epoch 40 | loss: 0.51113 |  0:01:46s
epoch 41 | loss: 0.50976 |  0:01:48s
epoch 42 | loss: 0.51073 |  0:01:51s
epoch 43 | loss: 0.50917 |  0:01:54s
epoch 44 | loss: 0.50869 |  0:01:56s
epoch 45 | loss: 0.50862 |  0:01:59s
epoch 46 | loss: 0.50823 |  0:02:02s
epoch 47 | loss: 0.50692 |  0:02:04s
```

```
epoch 48 | loss: 0.50536 |  0:02:07s
epoch 49 | loss: 0.50241 |  0:02:09s
```

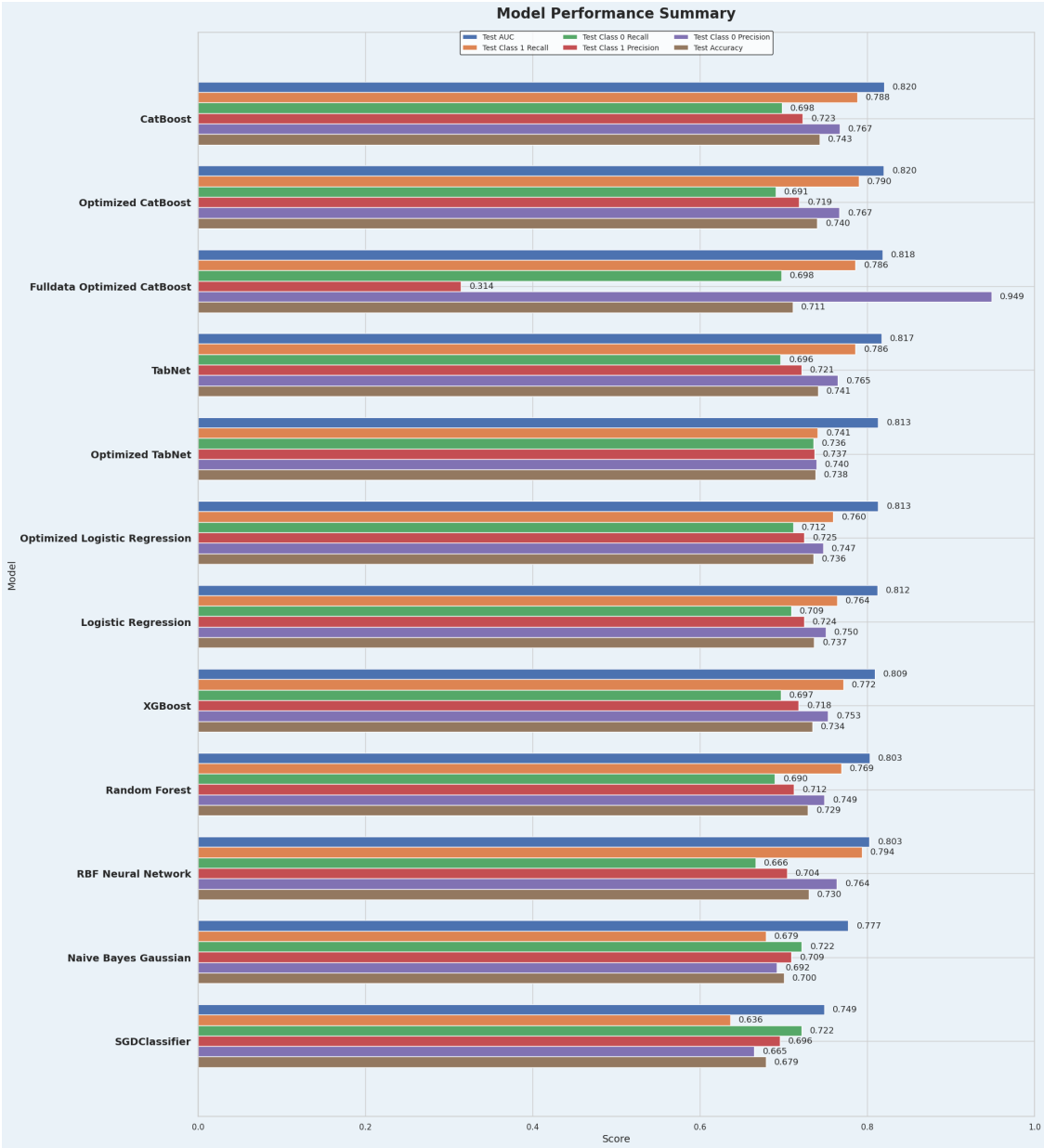
13.5.5 Evaluate

[ ]:
```
# Evaluate the model using your custom evaluation function
evaluate_model(best_tabnet_clf, uX_train_tabnet.values, uy_train_tabnet,␣
↪uX_test_tabnet.values, uy_test_tabnet, 'Optimized TabNet')
```


Optimized TabNet
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8370
ROC AUC Curve - Test
Test ROC AUC: 0.8132
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.8181
Precision-Recall Curve - Test
Test PR AUC: 0.7836
Train Confusion Matrix
16437
5437
5252
16622
True Label
Predicted Label
Test Confusion Matrix
4024
1445
1417
4052
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.74      0.74      0.74      5469
Test Class 1       0.74      0.74      0.74      5469

    accuracy                           0.74     10938
   macro avg       0.74      0.74      0.74     10938
weighted avg       0.74      0.74      0.74     10938
```

[ ]:
```
evaluation_summary()
```



13.6 Summary

CatBoost is said to have strong out of the box defaults and tuning did not improve it despite training on it multiple times for long periods. However I later find reason to believe the optimized version is more robust. This was found through plotting for finding optimal thresholds.

For TabNet, although it achieved a slightly higher recall, this gain was minimal and came at the cost of a lower ROC AUC score, and tweaking the threshold is probably a better course.

Logistic regression showed a marginal improvement after tuning with grid search, but for the time and effort spent feature engineering might have been time better spent. It is something you have

to do to know the value of for a dataset however.

Overall, this hyperparameter tuning process was time-consuming, requiring extensive setup, execution, and evaluation. Despite these efforts, the results suggest that this dataset may not respond well to hyperparameter tuning overall and get minimal benefit from it.

This dataset benefits more from feature engineering, binning, and balancing techniques than from hyperparameter tuning.

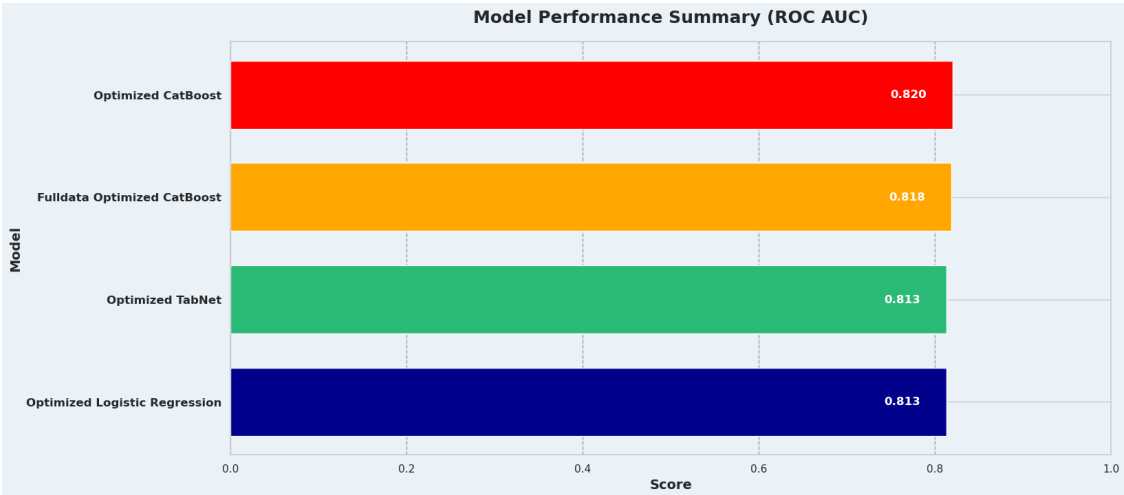
13.6.1 Summary Visuals

[ ]:
```
model_performance.keys()
```

[ ]:
```
dict_keys(['Logistic Regression', 'Random Forest', 'XGBoost', 'CatBoost', 'Naive
Bayes Gaussian', 'RBF Neural Network', 'SGDClassifier', 'TabNet', 'Optimized
CatBoost', 'Fulldata Optimized CatBoost', 'Optimized Logistic Regression',
'Optimized TabNet'])
```

[ ]:
```
tuned_model_list = ['Optimized CatBoost', 'Fulldata Optimized CatBoost',␣
↪'Optimized Logistic Regression', 'Optimized TabNet']
tuned_model_dict = {key: model_performance[key] for key in tuned_model_list if␣
↪key in model_performance}
#baseline_model_dict
```

[ ]:
```
evaluation_summary_simplified(tuned_model_dict)
```



14 Finetune, Stack and Average Models

14.1 Finetune CatBoost on Full Data

Can I combine the undersampled class balanced model with a fine tune adjustment on the unbalanced data to make it more robust? I can, and I did. It also ended up my best model!

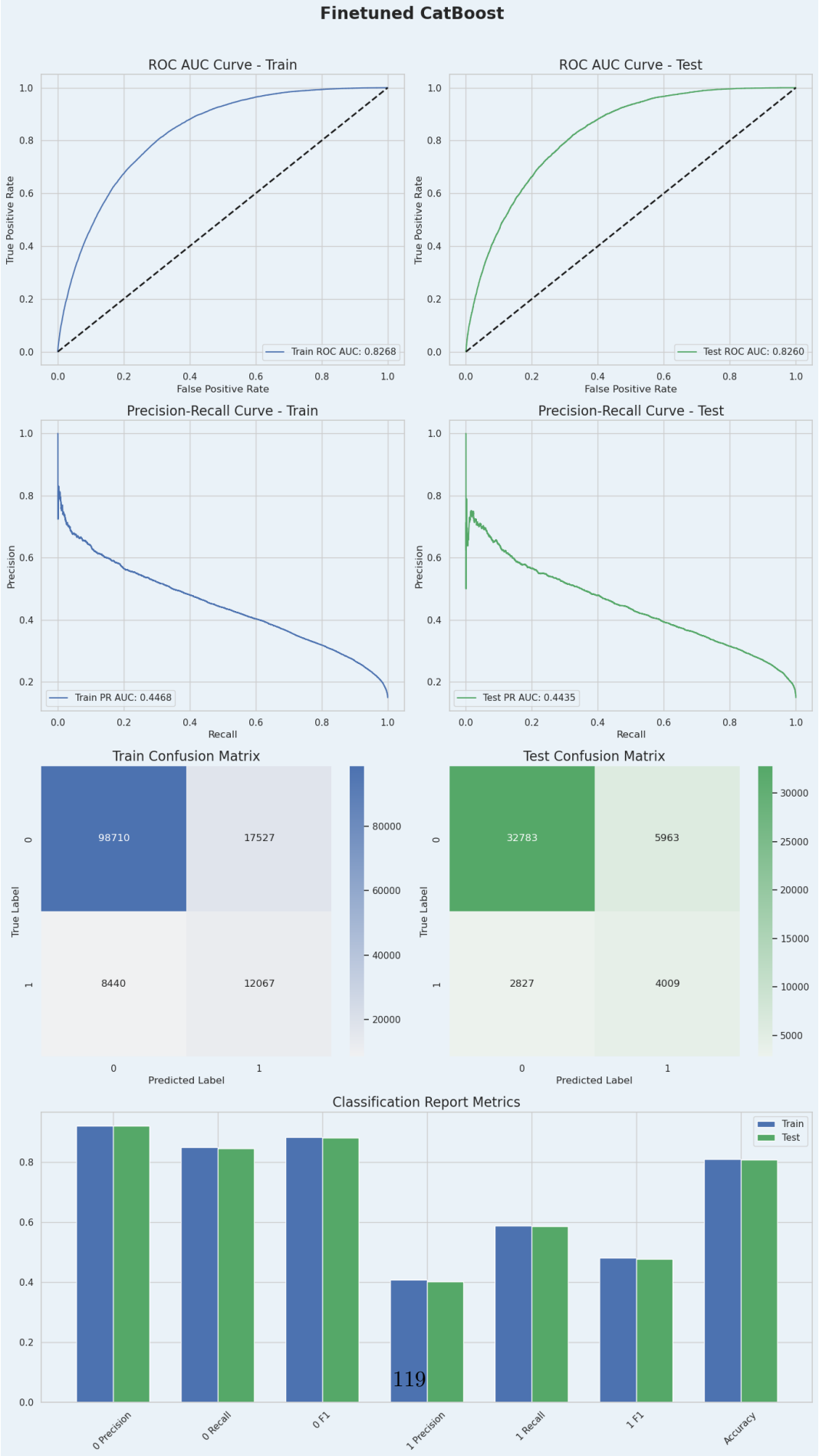
14.1.1 CatBoost

[ ]:
```
# Initialize the CatBoost model
full_catboost_model = CatBoostClassifier(
    iterations=20000,
    learning_rate=0.01,  # Base Rate was .1, sometimes you want to reduce the␣
 ↪learning rate for fine-tuning
    depth=7,  # Same Depth
    # maxxing recall tends to make the overall model suffer.
    # Adjusting later with finetuning makes a stronger model for us in this case
    eval_metric='F1',
    random_seed=42,  # Ensure reproducibility
    task_type="CPU",  # Use CPU because CatBoost does not support training␣
 ↪continuation with GPU
    logging_level='Silent'  # Suppress detailed output
)

# Fine-tune the model on the entire training set, before it was trained only on␣
 ↪the undersampled set
full_catboost_model.fit(
    X_train, y_train,
    eval_set=(X_test, y_test), #This is used for early stopping only
    verbose=100,  # Print training progress every 100 iterations
    early_stopping_rounds=1000,  # Stop early if there is no improvement
    init_model=best_catboost_model  # Continue training from the prior best␣
 ↪model
    #init_model=catboost_model  # Continue training from the prior best model
)
```

[ ]:
```
<catboost.core.CatBoostClassifier at 0x7ca5c8a3ee90>
```

14.1.2 Evaluate

[ ]:
```
evaluate_model(full_catboost_model, X_train, y_train, X_test, y_test,␣
 ↪'Finetuned CatBoost')
```


Finetuned CatBoost
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8268
ROC AUC Curve - Test
Test ROC AUC: 0.8260
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.4468
Precision-Recall Curve - Test
Test PR AUC: 0.4435
Train Confusion Matrix
98710
17527
8440
12067
True Label
Predicted Label
Test Confusion Matrix
32783
5963
2827
4009
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.92      0.85      0.88     38746
Test Class 1       0.40      0.59      0.48      6836

    accuracy                           0.81     45582
   macro avg       0.66      0.72      0.68     45582
weighted avg       0.84      0.81      0.82     45582
```

It seems the changes are really minimal to ROC AUC but worse.

What really changes is the balance of the classifications. 0 Precision shoots up. 1 Precision and 1 F1 score go down. Accuracy is worse. Though it does boost 1 Recall some.

I tried this with lots of iterations too and a bigger learning rate the results are almost always the same.

What makes this model stand out is how it does improve robustness but also requires threshold tuning.

14.2 Stack CatBoost and Logistic Regression

This is a more traditional way to stack models. A simpler model on top of a more complex one underneath.

14.2.1 Generate Predictions from CatBoost Model

[ ]:
```
catboost_train_preds = full_catboost_model.predict_proba(X_train)[:, 1].
 ↪reshape(-1, 1)
catboost_test_preds = full_catboost_model.predict_proba(X_test)[:, 1].
 ↪reshape(-1, 1)
stacked_train_features = np.hstack((X_train, catboost_train_preds))
stacked_test_features = np.hstack((X_test, catboost_test_preds))
```

14.2.2 Train Stacked

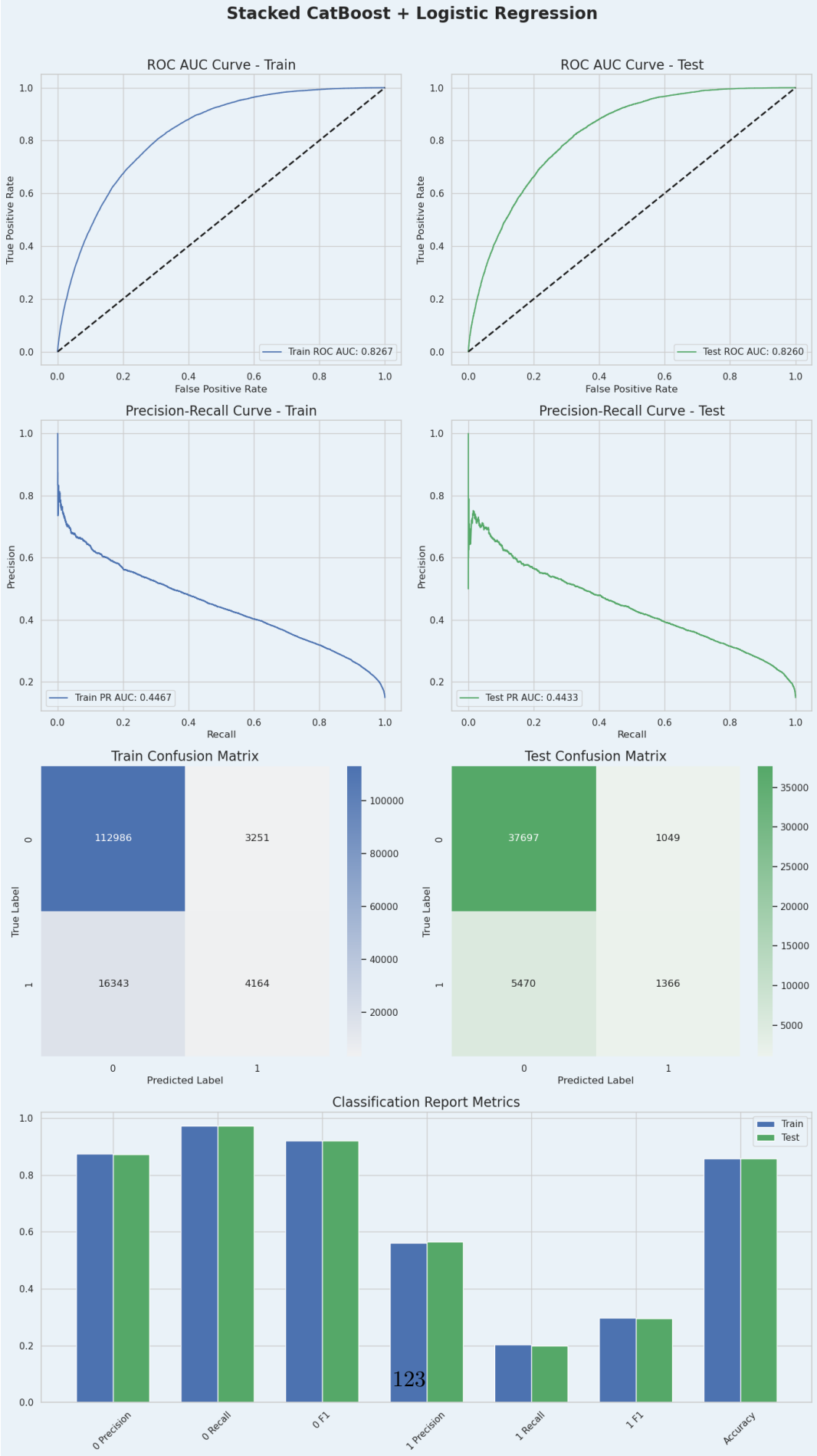
[ ]:
```
# Define the logistic regression hyperparameter grid
stacked_logistic_optimal = True

if stacked_logistic_optimal:
    param_grid = {
        'model__C': [0.01],  # Single value in a list
        'model__max_iter': [100],  # Single value in a list
        'model__penalty': ['l1'],  # Single value in a list
        'scaler': [MinMaxScaler()]  # Single instance in a list
```

```
    }
else:
    param_grid = [
        {
            'scaler': [None, StandardScaler(), MinMaxScaler(), RobustScaler()],
            'model__penalty': ['none'],
            'model__C': [1],
            'model__max_iter': [100, 500, 1000]
        },
        {
            'scaler': [None, StandardScaler(), MinMaxScaler(), RobustScaler()],
            'model__penalty': ['l1', 'l2'],
            'model__C': [0.01, 0.1, 1, 10, 100],
            'model__max_iter': [100, 500, 1000]
        },
        {
            'scaler': [None, StandardScaler(), MinMaxScaler(), RobustScaler()],
            'model__penalty': ['elasticnet'],
            'model__C': [0.01, 0.1, 1, 10, 100],
            'model__max_iter': [100, 500, 1000],
            'model__l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
        }
    ]

# Create the pipeline for logistic regression
stacked_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Placeholder for scaler
    ('model', LogisticRegression(solver='saga', random_state=42))  # Use saga␣
 ↪to handle elasticnet
])

# Initialize the grid search with the parameter grid
grid_search = GridSearchCV(stacked_pipeline, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring='roc_auc', error_score=np.nan)

# Fit the grid search on the stacked features
grid_search.fit(stacked_train_features, y_train.values.ravel())

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)

# Evaluate on the test set
stacked_model = grid_search.best_estimator_
test_score = stacked_model.score(stacked_test_features, y_test.values.ravel())
print("Test Set Score:", test_score)
```

```
Best Parameters: {'model__C': 0.01, 'model__max_iter': 100, 'model__penalty':
'l1', 'scaler': MinMaxScaler()}
Best Cross-Validation Score: 0.8267081388840645
Test Set Score: 0.8569830196130052
```

Best Parameters: {'model__C': 0.01, 'model__max_iter': 100, 'model__penalty': 'l1', 'scaler': MinMaxScaler()} Best Cross-Validation Score: 0.8215216729619217 Test Set Score: 0.8554912026677197

[ ]:
```
evaluate_model(stacked_model, stacked_train_features, y_train,␣
↪stacked_test_features, y_test, 'Stacked CatBoost + Logistic Regression')
```


Stacked CatBoost + Logistic Regression
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8267
ROC AUC Curve - Test
Test ROC AUC: 0.8260
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.4467
Precision-Recall Curve - Test
Test PR AUC: 0.4433
Train Confusion Matrix
112986
3251
16343
4164
True Label
Predicted Label
Test Confusion Matrix
37697
1049
5470
1366
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.87      0.97      0.92     38746
Test Class 1       0.57      0.20      0.30      6836

    accuracy                           0.86     45582
   macro avg       0.72      0.59      0.61     45582
weighted avg       0.83      0.86      0.83     45582
```

14.3 Average CatBoost and Logistic Regression

This is a way to simply use multiple models and average the result. This did help! But it didn't end up being the best solution.

[ ]:
```
# Generate predictions (probabilities) for the training and test sets from both␣
↪models
catboost_train_preds = full_catboost_model.predict_proba(X_train)[:, 1]
catboost_test_preds = full_catboost_model.predict_proba(X_test)[:, 1]

logistic_train_preds = tuned_logistic_model.predict_proba(X_train)[:, 1]
logistic_test_preds = tuned_logistic_model.predict_proba(X_test)[:, 1]
```

14.3.1 Averaged Predictions

[ ]:
```
# Average the predictions
averaged_train_preds = (catboost_train_preds + logistic_train_preds) / 2
averaged_test_preds = (catboost_test_preds + logistic_test_preds) / 2
```

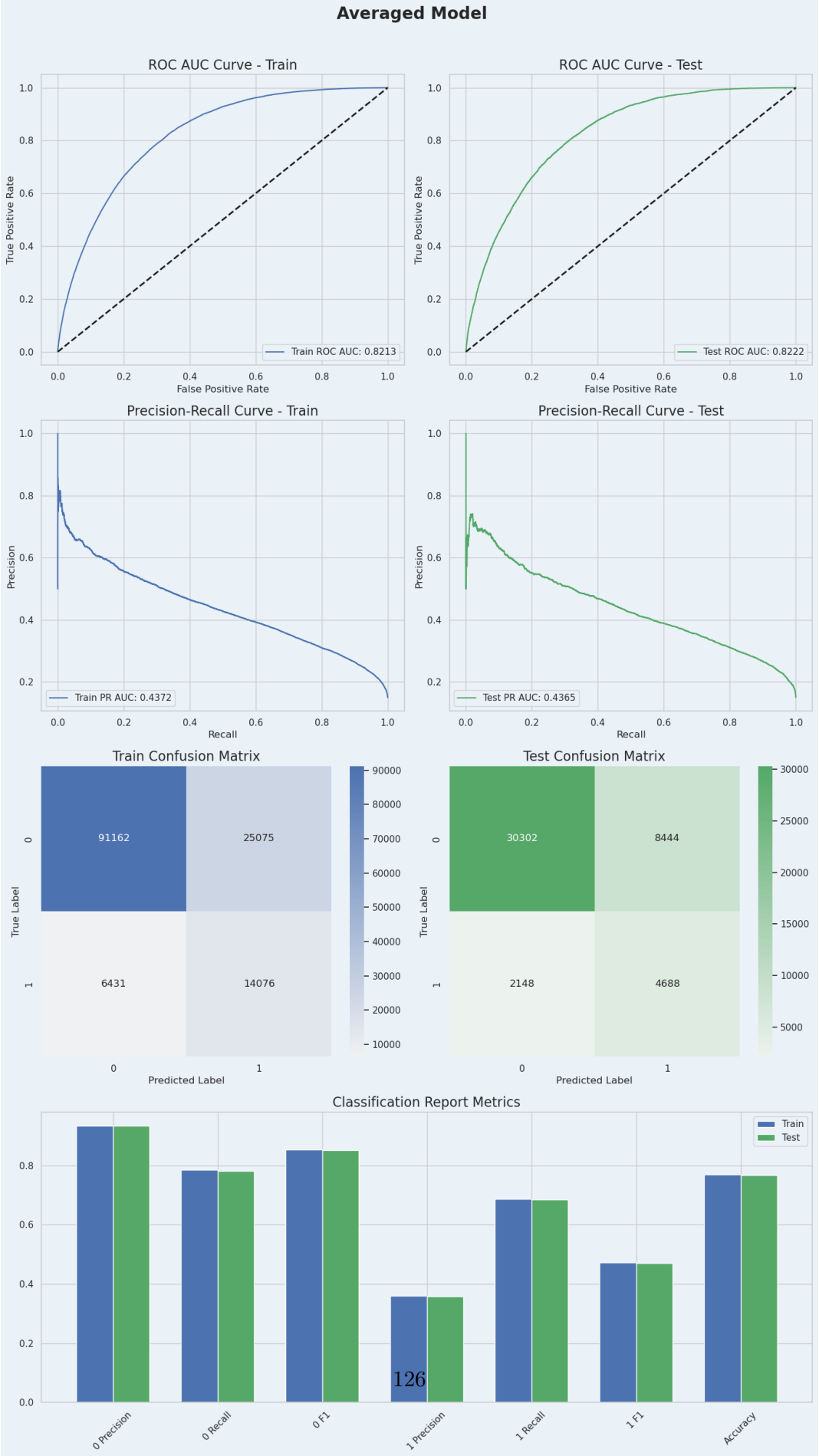
14.3.2 Averaged Classifier

[ ]:
```
class AveragedModel(BaseEstimator, ClassifierMixin):
    def __init__(self, model1, model2, threshold=0.5):
        self.model1 = model1
        self.model2 = model2
        self.threshold = threshold

    def predict_proba(self, X):
        # Compute the predictions dynamically based on the input X
        model1_probs = self.model1.predict_proba(X)[:, 1]
        model2_probs = self.model2.predict_proba(X)[:, 1]
        averaged_probs = (model1_probs + model2_probs) / 2
        return np.vstack([1 - averaged_probs, averaged_probs]).T
```

```
    def predict(self, X):
        # Apply the threshold to determine class labels
        probs = self.predict_proba(X)[:, 1]
        return (probs >= self.threshold).astype(int)

# Instantiate the averaged model with actual models
averaged_model = AveragedModel(model1=full_catboost_model,␣
 ↪model2=tuned_logistic_model)
```

14.3.3 Evaluation

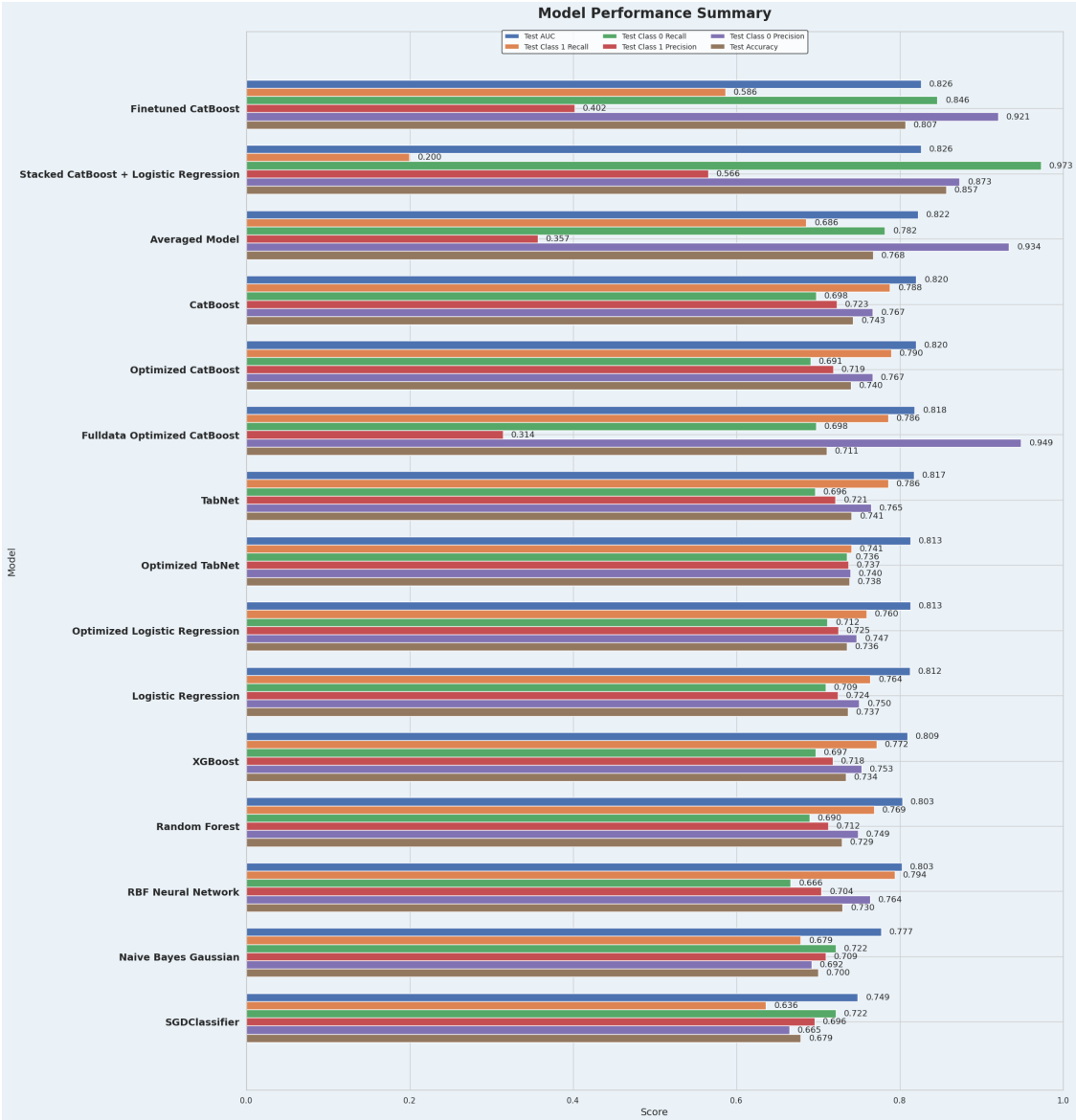
[ ]:
```
# Evaluate the averaged model
evaluate_model(averaged_model, X_train, y_train, X_test, y_test, 'Averaged␣
 ↪Model')
```


Averaged Model
ROC AUC Curve - Train
True Positive Rate
False Positive Rate
Train ROC AUC: 0.8213
ROC AUC Curve - Test
Test ROC AUC: 0.8222
Precision-Recall Curve - Train
Precision
Recall
Train PR AUC: 0.4372
Precision-Recall Curve - Test
Test PR AUC: 0.4365
Train Confusion Matrix
91162
25075
6431
14076
True Label
Predicted Label
Test Confusion Matrix
30302
8444
2148
4688
Classification Report Metrics
Train
Test
0 Precision
0 Recall
0 F1
1 Precision
1 Recall
1 F1
Accuracy

```
Test Classification Report:
               precision    recall  f1-score   support

Test Class 0       0.93      0.78      0.85     38746
Test Class 1       0.36      0.69      0.47      6836

    accuracy                           0.77     45582
   macro avg       0.65      0.73      0.66     45582
weighted avg       0.85      0.77      0.79     45582
```

15 All Models Summary

[ ]:
```
evaluation_summary()
```



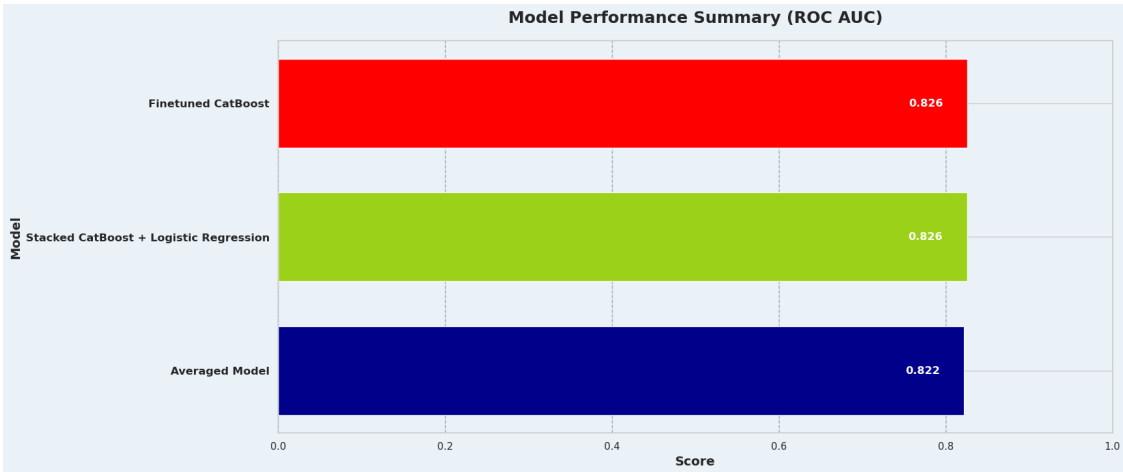
15.0.1 Summary Visuals

[ ]:
```
model_performance.keys()
```

[ ]:
```
dict_keys(['Logistic Regression', 'Random Forest', 'XGBoost', 'CatBoost', 'Naive
Bayes Gaussian', 'RBF Neural Network', 'SGDClassifier', 'TabNet', 'Optimized
CatBoost', 'Fulldata Optimized CatBoost', 'Optimized Logistic Regression',
'Optimized TabNet', 'Finetuned CatBoost', 'Stacked CatBoost + Logistic
Regression', 'Averaged Model'])
```

```
[ ]: finetuned_model_list = ['Finetuned CatBoost', 'Stacked CatBoost + Logistic␣
     ↪Regression', 'Averaged Model']
     finetuned_model_dict = {key: model_performance[key] for key in␣
     ↪finetuned_model_list if key in model_performance}
     #baseline_model_dict
```

```
[ ]: evaluation_summary_simplified(finetuned_model_dict)
```



16 Threshold Tuning and Model Evaluations

Threshold tuning is essential for any medical model in my opinion because you often care about recall more than you do other metrics.

16.1 Subplot Model Threshold Evaluations

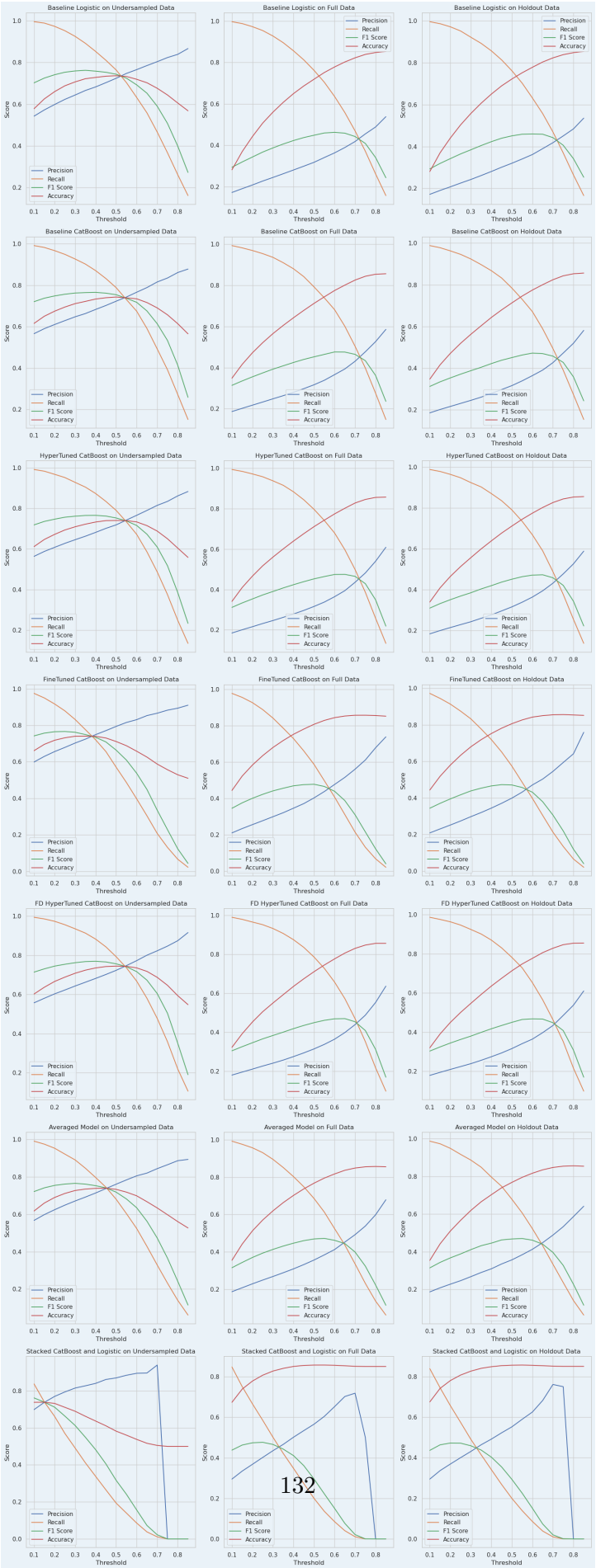
```
[ ]: # Collect data for all models and datasets
     models_data = []
     # Define the models and their descriptions
     models = [
         (log_reg, "Baseline Logistic"),
         (catboost_model, "Baseline CatBoost"),
         (best_catboost_model, "HyperTuned CatBoost"),
         (full_catboost_model, "FineTuned CatBoost"),
         (fd_best_catboost_model, "FD HyperTuned CatBoost"),
         (averaged_model, "Averaged Model")
     ]

     # Define the datasets and their descriptions
     datasets = [
         (uX_test, uy_test, "Undersampled Data"),
```

```
    (X_test, y_test, "Full Data"),
    (X_holdout, y_holdout, "Holdout Data")
]

# Collect data for all models and datasets
models_data = []

# Loop over the models and datasets
for model, model_name in models:
    for X, y, data_desc in datasets:
        # Generate the title and apply the model_thresholds function
        title = f'{model_name} on {data_desc}'
        thresh_df, title = model_thresholds(model, X, y, title)
        models_data.append((thresh_df, title))

# Special handling for the stacked model
# Stack the predictions with the original undersampled test features
catboost_undersampled_preds = full_catboost_model.predict_proba(uX_test)[:, 1].
 ↪reshape(-1, 1)
stacked_undersampled_features = np.hstack((uX_test,␣
 ↪catboost_undersampled_preds))

thresh_df_undersampled, title_undersampled = model_thresholds(stacked_model,␣
 ↪stacked_undersampled_features, uy_test, 'Stacked CatBoost and Logistic on␣
 ↪Undersampled Data')
models_data.append((thresh_df_undersampled, title_undersampled))

# Stack the predictions with the original full test features
catboost_full_preds = full_catboost_model.predict_proba(X_test)[:, 1].
 ↪reshape(-1, 1)
stacked_full_features = np.hstack((X_test, catboost_full_preds))

thresh_df_full, title_full = model_thresholds(stacked_model,␣
 ↪stacked_full_features, y_test, 'Stacked CatBoost and Logistic on Full Data')
models_data.append((thresh_df_full, title_full))

# Stack the predictions with the original holdout features
catboost_holdout_preds = full_catboost_model.predict_proba(X_holdout)[:, 1].
 ↪reshape(-1, 1)
stacked_holdout_features = np.hstack((X_holdout, catboost_holdout_preds))

thresh_df_holdout, title_holdout = model_thresholds(stacked_model,␣
 ↪stacked_holdout_features, y_holdout, 'Stacked CatBoost and Logistic on␣
 ↪Holdout Data')
models_data.append((thresh_df_holdout, title_holdout))
```

16.2 Threshold Visualizations

This lets you see not only the thresholds you should tune to, but how stable the model is over a wide range of thresholds. Sharp falls or rises in a score can be reason to believe a model might not be as robust.

[ ]:
```
# Plot all subplots
plot_subplots(models_data)
```


Baseline Logistic on Undersampled Data
Baseline Logistic on Full Data
Baseline Logistic on Holdout Data
Baseline CatBoost on Undersampled Data
Baseline CatBoost on Full Data
Baseline CatBoost on Holdout Data
HyperTuned CatBoost on Undersampled Data
HyperTuned CatBoost on Full Data
HyperTuned CatBoost on Holdout Data
FineTuned CatBoost on Undersampled Data
FineTuned CatBoost on Full Data
FineTuned CatBoost on Holdout Data
FD HyperTuned CatBoost on Undersampled Data
FD HyperTuned CatBoost on Full Data
FD HyperTuned CatBoost on Holdout Data
Averaged Model on Undersampled Data
Averaged Model on Full Data
Averaged Model on Holdout Data
Stacked CatBoost and Logistic on Undersampled Data
Stacked CatBoost and Logistic on Full Data
Stacked CatBoost and Logistic on Holdout Data
Precision
Recall
F1 Score
Accuracy
Threshold
Score

16.2.1 Threshold Compare

[ ]:
```
report_at_threshold(.32,full_catboost_model,X_holdout,y_holdout)
```

```
Confusion Matrix:
 [[26191 12555]
 [ 1285  5551]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.68      0.79     38746
           1       0.31      0.81      0.45      6836

    accuracy                           0.70     45582
   macro avg       0.63      0.74      0.62     45582
weighted avg       0.86      0.70      0.74     45582
```

17 Save Model

- Use ONNX and make a single web page js with forms that will report your diabetes risk

17.1 Wrappers

[ ]:
```
# See https://onnx.ai/sklearn-onnx/auto_tutorial/plot_gexternal_catboost.html
# the only place that had a converter for catboost that works
def skl2onnx_parser_catboost_classifier(scope, model, inputs,␣
 ↪custom_parsers=None):
    options = scope.get_options(model, dict(zipmap=True))
    no_zipmap = isinstance(options["zipmap"], bool) and not options["zipmap"]

    alias = _get_sklearn_operator_name(type(model))
    this_operator = scope.declare_local_operator(alias, model)
    this_operator.inputs = inputs

    label_variable = scope.declare_local_variable("label", Int64TensorType())
    prob_dtype = guess_tensor_type(inputs[0].type)
    probability_tensor_variable = scope.declare_local_variable(
        "probabilities", prob_dtype
    )
    this_operator.outputs.append(label_variable)
    this_operator.outputs.append(probability_tensor_variable)
    probability_tensor = this_operator.outputs
```

```
    if no_zipmap:
        return probability_tensor

    return _apply_zipmap(
        options["zipmap"], scope, model, inputs[0].type, probability_tensor
    )

def skl2onnx_convert_catboost(scope, operator, container):
    onx = convert_to_onnx_object(operator.raw_operator)
    opsets = {d.domain: d.version for d in onx.opset_import}
    if "" in opsets and opsets[""] >= container.target_opset:
        raise RuntimeError("CatBoost uses an opset more recent than the target␣
 ↪one.")
    if len(onx.graph.initializer) > 0 or len(onx.graph.sparse_initializer) > 0:
        raise NotImplementedError(
            "CatBoost returns a model initializers. This option is not␣
 ↪implemented yet."
        )
    if (
        len(onx.graph.node) not in (1, 2)
        or not onx.graph.node[0].op_type.startswith("TreeEnsemble")
        or (len(onx.graph.node) == 2 and onx.graph.node[1].op_type != "ZipMap")
    ):
        types = ", ".join(map(lambda n: n.op_type, onx.graph.node))
        raise NotImplementedError(
            f"CatBoost returns {len(onx.graph.node)} != 1 (types={types}). "
            f"This option is not implemented yet."
        )
    node = onx.graph.node[0]
    atts = {}
    for att in node.attribute:
        atts[att.name] = get_attribute_value(att)
    container.add_node(
        node.op_type,
        [operator.inputs[0].full_name],
        [operator.outputs[0].full_name, operator.outputs[1].full_name],
        op_domain=node.domain,
        op_version=opsets.get(node.domain, None),
        **atts,
    )
```

17.2 Register Converter

[ ]:
```
# Register the converter
update_registered_converter(
    CatBoostClassifier,
```

```
    "CatBoostCatBoostClassifier",
    calculate_linear_classifier_output_shapes,
    skl2onnx_convert_catboost,
    parser=skl2onnx_parser_catboost_classifier,
    options={"nocl": [True, False], "zipmap": [True, False, "columns"]}
)
```

17.3 Convert Model To ONNX

[ ]:
```
# Define the input type based on your training data shape
initial_type = [("input", FloatTensorType([None, X_train.shape[1]]))]

# Set options to disable ZipMap
options = {
    CatBoostClassifier: {
        "zipmap": False  # Disable ZipMap
    }
}

# Convert the trained CatBoost model to ONNX with ZipMap disabled
model_onnx = convert_sklearn(
    full_catboost_model,
    "catboost_model",
    initial_types=initial_type,
    target_opset={"": 12, "ai.onnx.ml": 2},
    options=options  # Pass the options here
)

# Save the ONNX model to a file
catboost_onnx_file = os.path.join("docs","deployment", "models",␣
 ↪"catboost_model_no_zipmap.onnx")
with open(catboost_onnx_file, "wb") as f:
    f.write(model_onnx.SerializeToString())
```

17.4 Test ONNX

[ ]:
```
# Convert DataFrame to NumPy array and ensure float32 type
X_test_np = X_test.to_numpy().astype(np.float32)

# Load the ONNX model
sess = rt.InferenceSession(catboost_onnx_file)

# Test with the converted NumPy array using the correct input name
pred_onx = sess.run(None, {"input": X_test_np})

# Print predictions
```

```
print("Predicted Labels:", pred_onx[0])
print("Predicted Probabilities:", pred_onx[1])
```

```
Predicted Labels: [0 0 1 … 1 1 0]
Predicted Probabilities: [[0.7719998  0.22800022]
 [0.9543823  0.0456177 ]
 [0.35632342 0.6436766 ]
 …
 [0.4947396  0.5052604 ]
 [0.42117465 0.57882535]
 [0.9625573  0.03744268]]
```

[ ]:
```
# Save the model in Core ML format
#full_catboost_model.save_model('models/catboost_model.onnx', format="onnx")
```

18 Model Feature Accuracy Evaluation

[ ]:
```
def analyze_feature_impact(model, aX, feature_ranges):
    """
    Analyze the impact of increasing to max or decreasing to min each feature␣
 ↪on the probability of class 1.

    Parameters:
    model: Trained model object with a predict_proba method
    aX: Pandas DataFrame containing the holdout data features
    feature_ranges: Dictionary specifying the range of values for each feature

    Returns:
    DataFrame: A report showing the impact of each feature on class 1␣
 ↪probability
    """
    results = []
    aX = aX.copy()

    for feature in feature_ranges:
        if feature in aX.columns:
            original_values = aX[feature].copy()

            # Lower bound impact
            aX[feature] = feature_ranges[feature]['min']
            aX = update_interaction_terms(aX)
            low_probs = model.predict_proba(aX)[:, 1].mean()

            # Upper bound impact
            aX[feature] = feature_ranges[feature]['max']
            aX = update_interaction_terms(aX)
```

```
            high_probs = model.predict_proba(aX)[:, 1].mean()

            # Reset the feature to its original value
            aX[feature] = original_values

            # Calculate the change
            prob_change = high_probs - low_probs
            results.append({
                'Feature': feature,
                'Increases Class 1 Probability': high_probs > low_probs,
                'Change in Class 1 Probability': prob_change
            })

    report_df = pd.DataFrame(results)
    return report_df

def update_interaction_terms(df):
    """
    Update the interaction terms in the DataFrame based on the current values.
    """
    df['BMI_GenHlth_interaction'] = df['BMI'] * df['GenHlth']
    df['Income_GenHlth_interaction'] = df['Income'] * df['GenHlth']
    df['GenHlth_Age_interaction'] = df['GenHlth'] * df['Age']
    df['Age_PhysHlth_interaction'] = df['Age'] * df['PhysHlth']
    return df

def plot_fi_report(fi_report):
    fi_report = fi_report.sort_values(by='Change in Class 1 Probability',␣
 ↪ascending=False)
    plt.figure(figsize=(10, 8))
    sns.barplot(x='Change in Class 1 Probability', y='Feature', data=fi_report,␣
 ↪palette="viridis")
    plt.title('Feature Impact on Class 1 Probability')
    plt.show()

# Define the ranges for each feature based on your form and data
feature_ranges = {
    'HighBP': {'min': 0, 'max': 1},
    'HighChol': {'min': 0, 'max': 1},
    'CholCheck': {'min': 0, 'max': 1},
    'BMI': {'min': 12, 'max': 60},
    'Smoker': {'min': 0, 'max': 1},
    'Stroke': {'min': 0, 'max': 1},
    'HeartDiseaseorAttack': {'min': 0, 'max': 1},
    'PhysActivity': {'min': 0, 'max': 1},
    'Fruits': {'min': 0, 'max': 1},
    'Veggies': {'min': 0, 'max': 1},
```

```
    'HvyAlcoholConsump': {'min': 0, 'max': 1},
    'AnyHealthcare': {'min': 0, 'max': 1},
    'NoDocbcCost': {'min': 0, 'max': 1},
    'GenHlth': {'min': 1, 'max': 5},
    'DiffWalk': {'min': 0, 'max': 1},
    'Sex': {'min': 0, 'max': 1},
    'Age': {'min': 1, 'max': 13},
    'MentHlth': {'min': 1, 'max': 3},
    'PhysHlth': {'min': 1, 'max': 3},
    'Education': {'min': 1, 'max': 4},
    'Income': {'min': 1, 'max': 4}
}
```

18.1 Run Test

18.1.1 Save Reports

[ ]:
```
reports = {}
```

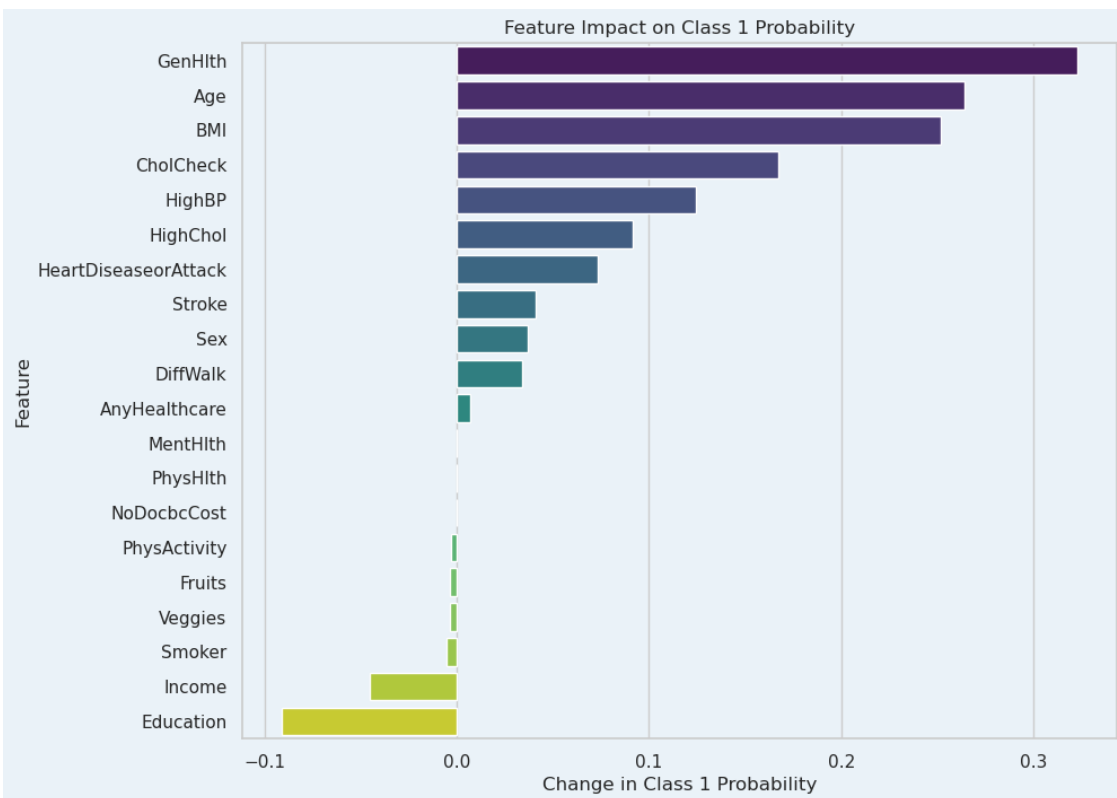
18.1.2 Baseline CatBoost

[ ]:
```
fi_report = analyze_feature_impact(catboost_model, X_holdout, feature_ranges)
display(fi_report)
reports['catboost_model'] = fi_report
```

```
                 Feature  Increases Class 1 Probability  \
0                 HighBP                           True
1               HighChol                           True
2              CholCheck                           True
3                    BMI                           True
4                 Smoker                          False
5                 Stroke                           True
6   HeartDiseaseorAttack                           True
7           PhysActivity                          False
8                 Fruits                          False
9                Veggies                          False
10         AnyHealthcare                           True
11           NoDocbcCost                          False
12               GenHlth                           True
13              DiffWalk                           True
14                   Sex                           True
15                   Age                           True
16              MentHlth                          False
17              PhysHlth                          False
18             Education                          False
19                Income                          False
```

```
    Change in Class 1 Probability
0                        0.124121
1                        0.091775
2                        0.167138
3                        0.251877
4                       -0.005600
5                        0.040999
6                        0.073060
7                       -0.002958
8                       -0.003705
9                       -0.003874
10                       0.007122
11                      -0.000394
12                       0.322664
13                       0.033880
14                       0.036975
15                       0.263925
16                       0.000000
17                       0.000000
18                      -0.091179
19                      -0.045295
```

[ ]:
```
plot_fi_report(fi_report)
```

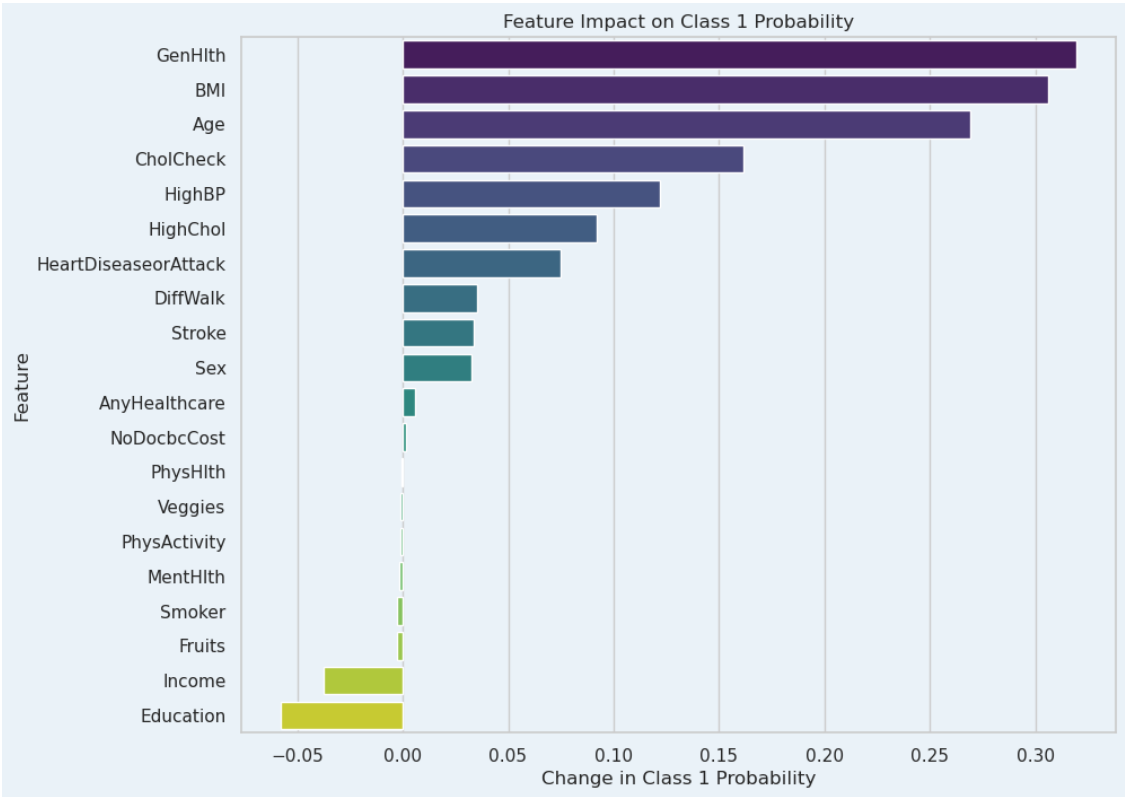


18.1.3 Hypertune CatBoost

[ ]:
```
fi_report = analyze_feature_impact(best_catboost_model, X_holdout,␣
↪feature_ranges)
display(fi_report)

reports['best_catboost_model'] = fi_report
plot_fi_report(fi_report)
```

```
                 Feature  Increases Class 1 Probability  \
0                 HighBP                           True
1               HighChol                           True
2              CholCheck                           True
3                    BMI                           True
4                 Smoker                          False
5                 Stroke                           True
6   HeartDiseaseorAttack                           True
7           PhysActivity                          False
8                 Fruits                          False
9                Veggies                          False
10         AnyHealthcare                           True
11           NoDocbcCost                           True
12               GenHlth                           True
13              DiffWalk                           True
14                   Sex                           True
15                   Age                           True
16              MentHlth                          False
17              PhysHlth                          False
18             Education                          False
19                Income                          False

    Change in Class 1 Probability
0                        0.121729
1                        0.092041
2                        0.161424
3                        0.305775
4                       -0.002939
5                        0.033605
6                        0.074728
7                       -0.001154
8                       -0.002969
9                       -0.001142
10                       0.005833
11                       0.001365
12                       0.319083
13                       0.035039
```

```
14                        0.032512
15                        0.268887
16                       -0.001551
17                       -0.000393
18                       -0.057824
19                       -0.037278
```



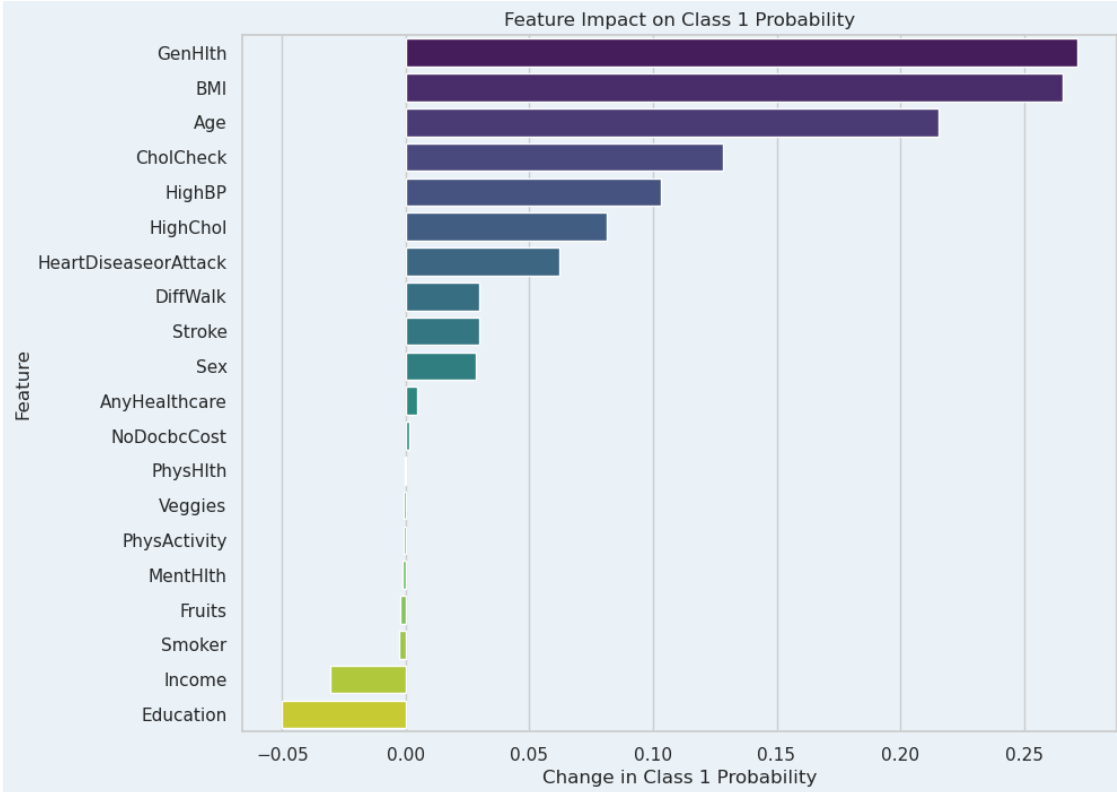
18.1.4 Finetuned CatBoost

[ ]:
```
fi_report = analyze_feature_impact(full_catboost_model, X_holdout,␣
↪feature_ranges)
display(fi_report)
final_fi_report = fi_report.copy()

reports['full_catboost_model'] = fi_report

plot_fi_report(fi_report)
```

```
                 Feature  Increases Class 1 Probability  \
0                 HighBP                           True
1               HighChol                           True
2              CholCheck                           True
```

3	BMI	True
4	Smoker	False
5	Stroke	True
6	HeartDiseaseorAttack	True
7	PhysActivity	False
8	Fruits	False
9	Veggies	False
10	AnyHealthcare	True
11	NoDocbcCost	True
12	GenHlth	True
13	DiffWalk	True
14	Sex	True
15	Age	True
16	MentHlth	False
17	PhysHlth	False
18	Education	False
19	Income	False

	Change in Class 1 Probability
0	0.103117
1	0.081174
2	0.128232
3	0.265598
4	-0.002813
5	0.029679
6	0.061900
7	-0.001025
8	-0.002171
9	-0.001021
10	0.004665
11	0.001447
12	0.271403
13	0.029743
14	0.028544
15	0.215589
16	-0.001209
17	-0.000363
18	-0.050175
19	-0.030733



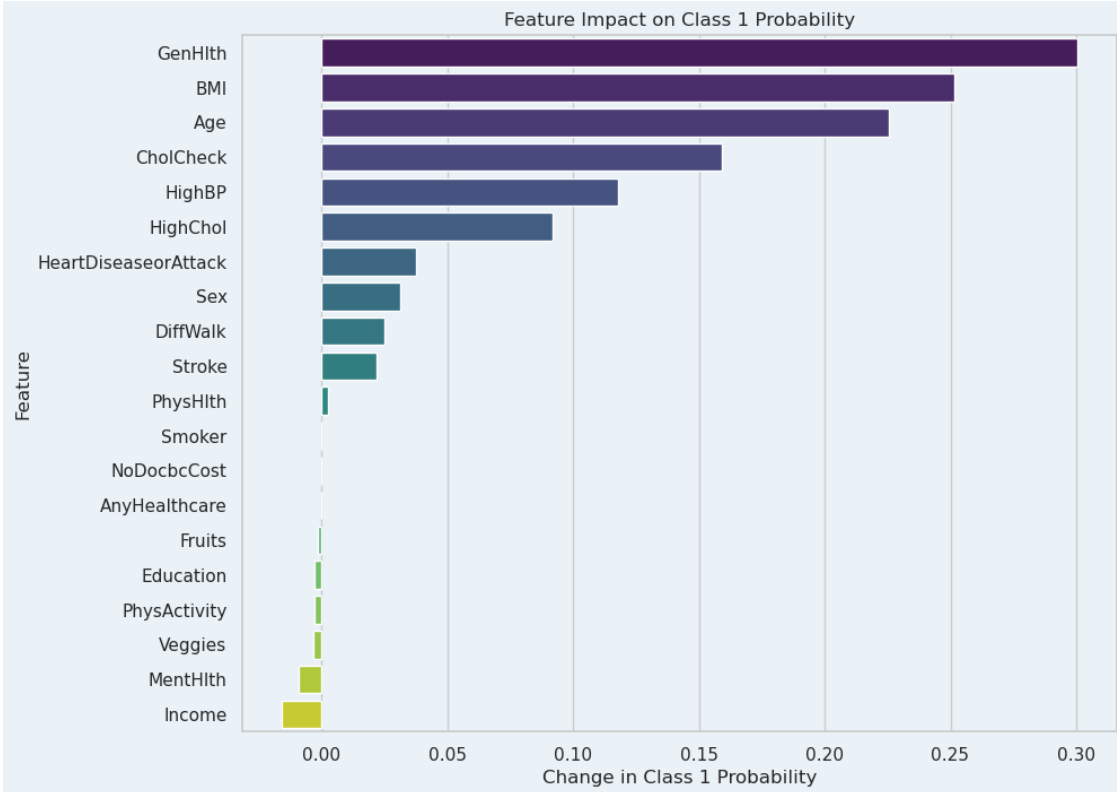
18.1.5 FullData HyperTuned CatBoost

[ ]:
```
fi_report = analyze_feature_impact(fd_best_catboost_model, X_holdout,␣
↪feature_ranges)
display(fi_report)

reports['fd_best_catboost_model'] = report

plot_fi_report(fi_report)
```

```
                Feature  Increases Class 1 Probability  \
0                 HighBP                           True
1               HighChol                           True
2              CholCheck                           True
3                    BMI                           True
4                 Smoker                          False
5                 Stroke                           True
6   HeartDiseaseorAttack                           True
7           PhysActivity                          False
8                 Fruits                          False
9                Veggies                          False
10         AnyHealthcare                          False
```

11	NoDocbcCost	False
12	GenHlth	True
13	DiffWalk	True
14	Sex	True
15	Age	True
16	MentHlth	False
17	PhysHlth	True
18	Education	False
19	Income	False

	Change in Class 1 Probability
0	0.117690
1	0.092023
2	0.159207
3	0.251595
4	0.000000
5	0.021640
6	0.037314
7	-0.003051
8	-0.001632
9	-0.003460
10	0.000000
11	0.000000
12	0.300243
13	0.025148
14	0.031293
15	0.225659
16	-0.009361
17	0.002461
18	-0.002645
19	-0.015885



18.1.6 Tuned Logistic

[ ]:
```
#fi_report = analyze_feature_impact(tuned_logistic_model,X_holdout,␣
↪feature_ranges)
#display(fi_report)

#reports['tuned_logistic_model'] = fi_report
```

18.1.7 TabNet

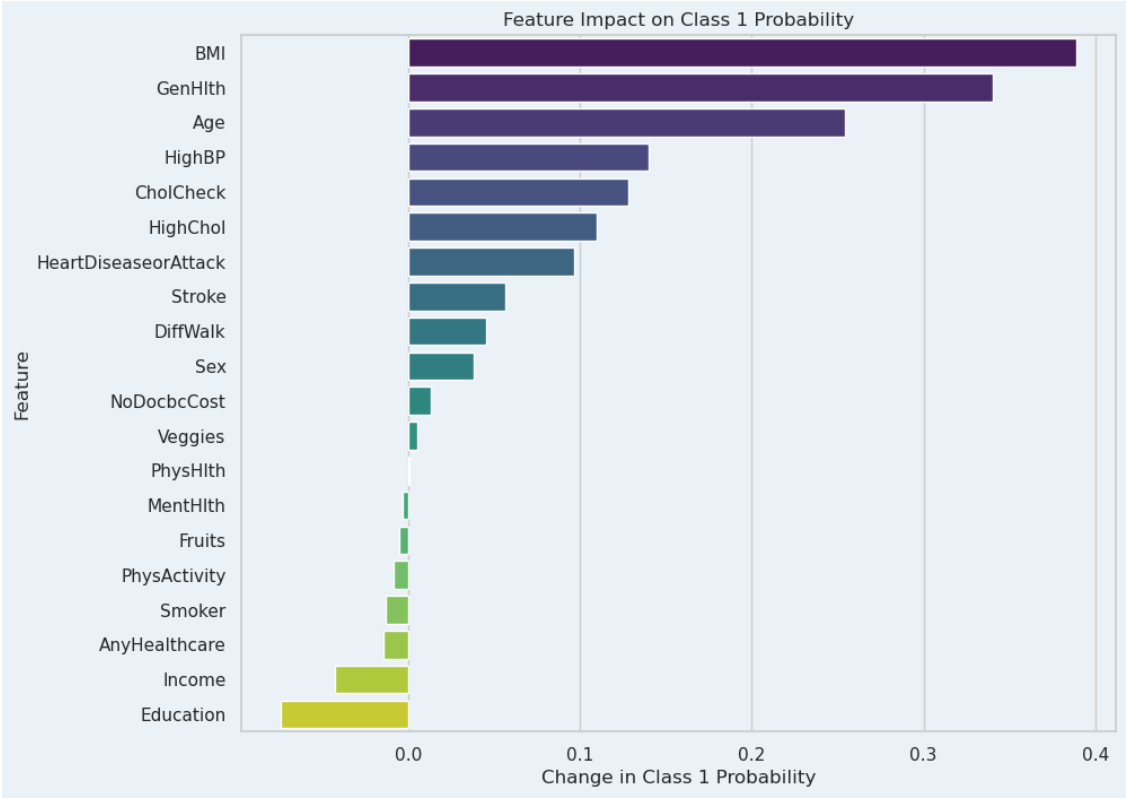
[ ]:
```
def analyze_tabnet_feature_impact(model, aX, feature_ranges):
    """
    Analyze the impact of increasing or decreasing each feature on the␣
↪probability of class 1 for the entire dataset for TabNet models.

    Parameters:
    model: Trained TabNet model object with a predict_proba method
    aX: Pandas DataFrame containing the holdout data features
    feature_ranges: Dictionary specifying the range of values for each feature

    Returns:
```

```
    DataFrame: A report showing the impact of each feature on class 1␣
↪probability for the entire dataset.
    """
    results = []

    # Convert the DataFrame to a numpy array for TabNet compatibility
    original_data = aX.to_numpy(dtype=np.float32)

    # Get the original probabilities for class 1
    original_probs = model.predict_proba(original_data)[:, 1]

    for feature in feature_ranges:
        if feature in aX.columns:
            # Prepare a copy of the data for modification
            data_copy = original_data.copy()

            # Set feature to its minimum value and predict
            min_val = feature_ranges[feature]['min']
            data_copy[:, aX.columns.get_loc(feature)] = min_val
            low_probs = model.predict_proba(data_copy)[:, 1]

            # Set feature to its maximum value and predict
            max_val = feature_ranges[feature]['max']
            data_copy[:, aX.columns.get_loc(feature)] = max_val
            high_probs = model.predict_proba(data_copy)[:, 1]

            # Calculate the average change in probability
            prob_change = high_probs.mean() - low_probs.mean()

            # Append the result for this feature
            results.append({
                'Feature': feature,
                'Increases Class 1 Probability': high_probs.mean() > low_probs.
↪mean(),
                'Change in Class 1 Probability': prob_change
            })

    # Create a DataFrame from the results
    report_df = pd.DataFrame(results)
    return report_df
```

[ ]:
```
fi_report = analyze_tabnet_feature_impact(best_tabnet_clf,X_holdout,␣
↪feature_ranges)
display(fi_report)

reports['best_tabnet_clf'] = fi_report
```

```
                 Feature  Increases Class 1 Probability  \
0                 HighBP                           True
1               HighChol                           True
2              CholCheck                           True
3                    BMI                           True
4                 Smoker                          False
5                 Stroke                           True
6   HeartDiseaseorAttack                           True
7           PhysActivity                          False
8                 Fruits                          False
9                Veggies                           True
10         AnyHealthcare                          False
11           NoDocbcCost                           True
12               GenHlth                           True
13              DiffWalk                           True
14                   Sex                           True
15                   Age                           True
16              MentHlth                          False
17              PhysHlth                           True
18             Education                          False
19                Income                          False

    Change in Class 1 Probability
0                        0.140042
1                        0.109510
2                        0.128362
3                        0.388686
4                       -0.013176
5                        0.056650
6                        0.096616
7                       -0.008191
8                       -0.005104
9                        0.005162
10                      -0.014457
11                       0.013217
12                       0.340467
13                       0.045502
14                       0.038408
15                       0.254392
16                      -0.003160
17                       0.000701
18                      -0.074102
19                      -0.042270
```

[ ]:
```
plot_fi_report(fi_report)
```



18.2 Summary

It all makes sense except the heavy drinking reducing your chance of diabetes.

Light Drinking is supposed to help vs diabetes but heavy drinking is expected to make it higher risk.

Lets analyze the heavy drinking data.

[ ]:
```
X_train.columns
```

[ ]:
```
Index(['HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'Stroke',
       'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'AnyHealthcare', 'NoDocbcCost', 'GenHlth', 'MentHlth', 'PhysHlth',
       'DiffWalk', 'Sex', 'Age', 'Education', 'Income', 'BMI_Age_interaction',
       'Income_GenHlth_interaction', 'PhysHlth_BMI_interaction',
       'MentHlth_BMI_interaction'],
      dtype='object')
```

19 Dashboard

19.1 Feature Impact

[ ]:
```
final_fi_report = final_fi_report.sort_values(by='Change in Class 1␣
↪Probability', ascending=True)

medical_blue = '#4A90E2'  # A soft medical blue
darker_medical_red = '#A52A2A'   # A darker, more muted medical red

# Define the gradient color scale
colorscale = [
    [0, '#FFEDA0'],   # Light yellow for values close to zero
    [0.5, '#FEB24C'], # Orange for mid-range values
    [1, '#F03B20']    # Dark red for values far from zero
]

# Create the horizontal bar chart with a gradient color scale
fig = go.Figure(data=[
    go.Bar(
        x=final_fi_report['Change in Class 1 Probability'],
        y=final_fi_report['Feature'],
        orientation='h',
        marker=dict(
            color=final_fi_report['Change in Class 1 Probability'],
            colorscale=colorscale,  # Apply the gradient color scale
            colorbar=dict(title='Change in Probability')  # Add a color bar␣
↪legend
        ),
        hoverinfo='x+y+text',  # Customize hover info
        hovertext=final_fi_report['Feature'],  # Add feature name to hover text
        hoverlabel=dict(
            bgcolor='white',  # Set hover label background color to white
            font_size=14,
            font_family="Arial"
        )
    )
])

# Update the layout for improved appearance
fig.update_layout(
    title='Mean Impact on Diabetes Risk Chance in Min vs Max Values Per Feature␣
↪For Final Model',
    title_font=dict(size=24),  # Increase the title font size
    xaxis=dict(
        title='Change in Class 1 Probability',
        title_font=dict(size=18),
```

```
        tickfont=dict(size=16, color=darker_medical_red, family='Arial'),  #␣
↪Increase x-axis tick label font size, color, and boldness
        automargin=True,  # Automatically adjust margins
        ticks='outside',  # Place ticks outside the plot area
        ticklen=10,  # Increase the tick length
        ticklabelposition='outside',  # Ensure tick labels are outside the axis
        ticklabelstandoff=10,  # Add standoff distance from the axis to the␣
↪tick labels
        showgrid=True,  # Show grid lines
        gridcolor='lightgrey'  # Set a light grey grid color
    ),
    yaxis=dict(
        title='Feature',
        title_font=dict(size=18),
        tickfont=dict(size=16, color=darker_medical_red, family='Arial'),  #␣
↪Increase y-axis tick label font size, color, and boldness
        automargin=True,  # Automatically adjust margins
        ticks='outside',
        ticklen=10,  # Increase the tick length
        ticklabelposition='outside',  # Ensure tick labels are outside the axis
        ticklabelstandoff=10,  # Add standoff distance from the axis to the␣
↪tick labels
        showgrid=True,  # Show grid lines
        gridcolor='lightgrey',  # Set a light grey grid color
    ),
    margin=dict(l=200, r=50, t=100, b=50),  # Adjust margins for better spacing
    plot_bgcolor='#f9f9f9',  # Set a light grey background for the plot area to␣
↪differentiate from the grid
    paper_bgcolor='white',  # Set paper background to white
    font=dict(color=darker_medical_red)  # Apply the darker medical red color␣
↪to the text
)

filename = os.path.join("docs","dashboard_files","mean_impact.html")
# Save as standalone HTML
plot(fig, filename=filename, include_plotlyjs='cdn')
```

[ ]: 'docs/dashboard_files/mean_impact.html'

19.2 ROC AUC Curve

[ ]:
```
# Step 1: Generate predicted probabilities for the positive class (class 1)
y_proba = full_catboost_model.predict_proba(X_test)[:, 1]

# Step 2: Calculate the ROC curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_proba)
```

```
roc_auc = auc(fpr, tpr)
display(roc_auc)

# Step 3: Create the ROC AUC chart with customizations
fig = go.Figure()

# Add the ROC curve with gradient color
fig.add_trace(go.Scatter(
    x=fpr, y=tpr,
    mode='lines',
    line=dict(
        color='black',  # Base line color before applying gradient
        width=2,
        shape='spline'
    ),
    fill='tozeroy',  # Fill the area under the curve
    fillcolor='rgba(240, 59, 32, 0.5)',  # Semi-transparent red fill
    hoverinfo='x+y+text',
    hovertext=f'ROC AUC = {roc_auc:.4f}',
    hoverlabel=dict(
        bgcolor='white',  # Set hover label background color to white
        font_size=14,
        font_family="Arial"
    ),
    showlegend=False  # Remove the legend for this trace
))

# Add diagonal line for reference
fig.add_trace(go.Scatter(
    x=[0, 1], y=[0, 1],
    mode='lines',
    line=dict(color='grey', dash='dash'),
    showlegend=False  # Remove the legend for this trace
))

# Update the layout for improved appearance
fig.update_layout(
    title=f'Receiver Operating Characteristic Areac Under Curve for Final␣
 ↪CatBoost Model | AUC = {roc_auc:.4f}',
    title_font=dict(size=24),
    xaxis=dict(
        title='False Positive Rate',
        title_font=dict(size=18),
        tickfont=dict(size=16, color=darker_medical_red, family='Arial'),  #␣
 ↪Tick styling
        automargin=True,
        showgrid=True,
```

```
        gridcolor='lightgrey',
        zeroline=False
    ),
    yaxis=dict(
        title='True Positive Rate',
        title_font=dict(size=18),
        tickfont=dict(size=16, color=darker_medical_red, family='Arial'),  #␣
 ↪Tick styling
        automargin=True,
        showgrid=True,
        gridcolor='lightgrey',
        zeroline=False
    ),
    margin=dict(l=100, r=50, t=100, b=50),
    plot_bgcolor='#f9f9f9',  # Set a light grey background for the plot area
    paper_bgcolor='white',
    font=dict(color=darker_medical_red)
)

filename = os.path.join("docs","dashboard_files","roc_auc.html")
# Save as standalone HTML
plot(fig, filename=filename, include_plotlyjs='cdn')
```

```
0.8260328905542169
```

[ ]:
```
'docs/dashboard_files/roc_auc.html'
```

19.3 Threshold Analysis

[ ]:
```
y_proba = full_catboost_model.predict_proba(X_holdout)[:, 1]

thresholds = np.arange(0.1, 0.9, 0.05)  # Match the threshold range and steps␣
 ↪to the first chart
precision_scores = []
recall_scores = []
f1_scores = []
accuracy_scores = []

for threshold in thresholds:
    y_pred = (y_proba >= threshold).astype(int)
    precision, recall, f1, _ = precision_recall_fscore_support(y_holdout,␣
 ↪y_pred, average='binary')
    accuracy = accuracy_score(y_holdout, y_pred)

    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)
```

```
    accuracy_scores.append(accuracy)

# Step 3: Create the line chart
fig = go.Figure()

# Plot F1 Score
fig.add_trace(go.Scatter(
    x=thresholds, y=f1_scores,
    mode='lines',
    name='F1 Score',
    line=dict(color='blue', width=2)
))

# Plot Accuracy
fig.add_trace(go.Scatter(
    x=thresholds, y=accuracy_scores,
    mode='lines',
    name='Accuracy',
    line=dict(color='green', width=2)
))

# Plot Recall
fig.add_trace(go.Scatter(
    x=thresholds, y=recall_scores,
    mode='lines',
    name='Recall',
    line=dict(color='red', width=2)
))

# Plot Precision
fig.add_trace(go.Scatter(
    x=thresholds, y=precision_scores,
    mode='lines',
    name='Precision',
    line=dict(color='purple', width=2)
))

# Add a vertical line at threshold 0.32
fig.add_shape(
    type="line",
    x0=0.32, y0=0, x1=0.32, y1=1,
    line=dict(color="black", width=2, dash="dot"),
    xref="x", yref="paper"
)

# Add rotated annotation for the chosen threshold
fig.add_annotation(
```

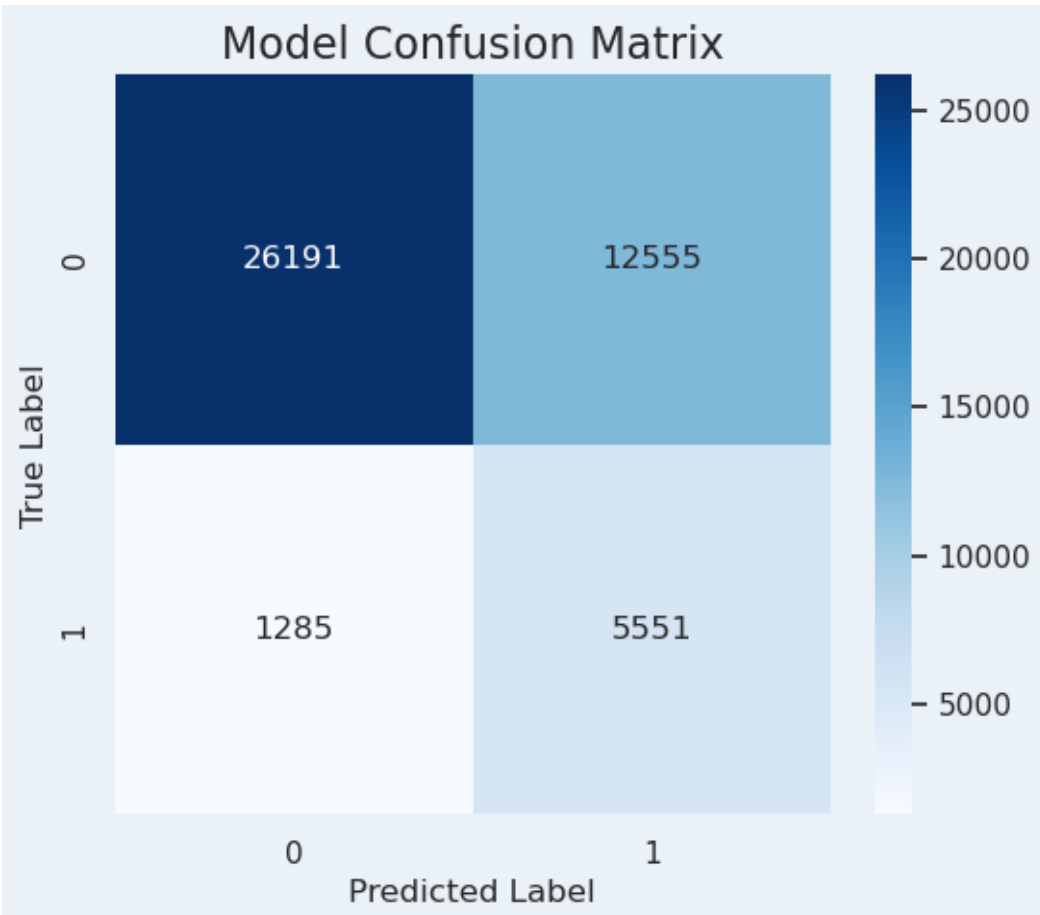
```
    x=0.315, y=0.5,  # Position of the annotation
    xref="x", yref="paper",
    text="Chosen Threshold = .32",  # Text to display
    showarrow=False,  # No arrow needed since it's aligned with the line
    font=dict(size=12, color="black"),
    align="center",
    textangle=-90,  # Rotate text to run parallel to the line
    ax=-10,  # Adjust the annotation position 10px to the left
    ay=0     # Keep the vertical alignment
)

# Update the layout
fig.update_layout(
    title='Threshold Analysis for Final CatBoost Model',
    title_font=dict(size=24),
    xaxis=dict(
        title='Threshold',
        title_font=dict(size=18),
        tickfont=dict(size=16, color='black', family='Arial'),
        automargin=True,
        showgrid=True,
        gridcolor='lightgrey',
        zeroline=False
    ),
    yaxis=dict(
        title='Score',
        title_font=dict(size=18),
        tickfont=dict(size=16, color='black', family='Arial'),
        automargin=True,
        showgrid=True,
        gridcolor='lightgrey',
        zeroline=False
    ),
    margin=dict(l=100, r=50, t=100, b=50),
    plot_bgcolor='white',
    paper_bgcolor='white',
    font=dict(color='black'),
    legend=dict(
        x=0.7, y=0.1,
        bgcolor='white',
        bordercolor='lightgrey',
        borderwidth=1
    )
)

# Save as standalone HTML
filename = os.path.join("docs","dashboard_files","threshold_analysis.html")
```

```
plot(fig, filename=filename, include_plotlyjs='cdn')
```

[ ]: 'docs/dashboard_files/threshold_analysis.html'

20 Confusion Matrix for Final Model

[ ]:
```
model_confusion_matrix(full_catboost_model,X_holdout,y_holdout,.32,"Final Model␣
↪Confusion Matrix")
```



[ ]:
```
report_at_threshold(.32,full_catboost_model,X_holdout,y_holdout)
```

```
Confusion Matrix:
 [[26191 12555]
 [ 1285  5551]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.68      0.79     38746
```

1	0.31	0.81	0.45	6836
accuracy			0.70	45582
macro avg	0.63	0.74	0.62	45582
weighted avg	0.86	0.70	0.74	45582