Table of Contents

Introduction

Diabetes mellitus, commonly referred to as diabetes, is a chronic medical condition characterized by high levels of blood glucose (blood sugar). It is a major global health concern, affecting over 400 million people worldwide. The disease is primarily categorized into two types: Type 1 diabetes, where the body fails to produce insulin, and Type 2 diabetes, where the body does not use insulin properly. Left unmanaged, diabetes can lead to severe health complications including heart disease, stroke, kidney failure, blindness, and lower limb amputations.

Given the significant impact of diabetes on individuals and healthcare systems, effective prediction and management strategies are critical. Machine learning, with its capability to analyze large datasets and identify patterns, offers promising solutions for early diagnosis, risk assessment, and personalized management of diabetes.

Data Source

The dataset used in this project is derived from the National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK). It contains detailed medical records of patients, including information such as age, BMI, insulin levels, and other relevant health metrics. This dataset is a valuable resource for developing predictive models and has been widely used in academic and professional research. The dataset can be accessed from the UCI Machine Learning Repository: Diabetes Dataset.

Project Overview

This project aims to leverage machine learning techniques to predict the likelihood of diabetes in individuals based on their medical records. The primary objectives of the project are:

1. **Data Exploration and Preprocessing**: Analyzing the dataset to understand its structure and clean any anomalies. This step involves handling missing values, normalizing data, and performing feature selection.

2. **Model Development**: Building and training various machine learning models, including Logistic Regression, Decision Trees, Random Forest, and Support Vector Machines (SVM), to predict diabetes. The models will be evaluated based on their accuracy, precision, recall, and F1-score.

3. **Model Evaluation and Selection**: Comparing the performance of different models to select the best-performing one. Cross-validation techniques will be employed to ensure the robustness of the results.

4. **Feature Importance Analysis**: Identifying the most significant features contributing to diabetes prediction, which can provide insights into the key factors influencing the disease.

5. **Deployment and Application**: Developing a user-friendly application that allows healthcare providers and individuals to input medical data and receive predictions about the likelihood of diabetes. This tool aims to facilitate early diagnosis and personalized management plans.

6. **Documentation and Reporting**: Thoroughly documenting the entire process, from data preprocessing to model deployment, to ensure reproducibility and transparency. The final report will include detailed explanations, visualizations, and interpretations of the results.

Impact

The outcomes of this project have the potential to significantly improve diabetes management by providing a reliable predictive tool for early diagnosis. Early identification of high-risk individuals can lead to timely interventions, ultimately reducing the burden of diabetes-related complications. Additionally, the insights gained from feature importance analysis can inform public health strategies and individual lifestyle modifications to mitigate the risk of diabetes.

By integrating advanced machine learning techniques with comprehensive medical data, this project aims to contribute to the broader effort of combating diabetes and enhancing the quality of life for affected individuals.

References

- National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK). "Diabetes Dataset." UCI Machine Learning Repository. Link.
- World Health Organization. "Diabetes." Link.

  This project exemplifies the practical application of machine learning in healthcare, showcasing the potential to make a meaningful impact on global health outcomes.

Data Description

The **diabetes_012_health_indicators_BRFSS2015.csv** dataset consists of 253,680 survey responses to the CDC's BRFSS2015, capturing various health indicators. The target variable, `Diabetes_012`, has three classes indicating the diabetes status of respondents. The dataset contains 21 feature variables, each representing different health metrics and behaviors. Below is a detailed description of each column in the dataset:

1. **Diabetes_012**: Diabetes status of the respondent.
   - 0 = No diabetes or only during pregnancy

- 1 = Prediabetes
- 2 = Diabetes

2. **HighBP**: High blood pressure status.
    - 0 = No high blood pressure
    - 1 = High blood pressure
3. **HighChol**: High cholesterol status.
    - 0 = No high cholesterol
    - 1 = High cholesterol
4. **CholCheck**: Whether the respondent had a cholesterol check in the past 5 years.
    - 0 = No cholesterol check in 5 years
    - 1 = Yes cholesterol check in 5 years
5. **BMI**: Body Mass Index, a numeric value representing body weight relative to height.
6. **Smoker**: Whether the respondent has smoked at least 100 cigarettes in their entire life.
    - 0 = No
    - 1 = Yes
7. **Stroke**: Whether the respondent has ever been told they had a stroke.
    - 0 = No
    - 1 = Yes
8. **HeartDiseaseorAttack**: Whether the respondent has coronary heart disease (CHD) or myocardial infarction (MI).
    - 0 = No
    - 1 = Yes
9. **PhysActivity**: Physical activity in the past 30 days, not including job-related activities.
    - 0 = No
    - 1 = Yes
10. **Fruits**: Whether the respondent consumes fruit one or more times per day.
    - 0 = No
    - 1 = Yes
11. **Veggies**: Whether the respondent consumes vegetables one or more times per day.
    - 0 = No
    - 1 = Yes
12. **HvyAlcoholConsump**: Heavy drinking status (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week).
    - 0 = No
    - 1 = Yes

13. **AnyHealthcare**: Whether the respondent has any kind of health care coverage, including health insurance, prepaid plans such as HMO, etc.
    - 0 = No
    - 1 = Yes
14. **NoDocbcCost**: Whether there was a time in the past 12 months when the respondent needed to see a doctor but could not because of cost.
    - 0 = No
    - 1 = Yes
15. **GenHlth**: General health status of the respondent.
    - Scale 1-5
        - 1 = Excellent
        - 2 = Very good
        - 3 = Good
        - 4 = Fair
        - 5 = Poor
16. **MentHlth**: Number of days during the past 30 days the respondent's mental health was not good.
    - Scale 1-30 days
17. **PhysHlth**: Number of days during the past 30 days the respondent's physical health was not good.
    - Scale 1-30 days
18. **DiffWalk**: Whether the respondent has serious difficulty walking or climbing stairs.
    - 0 = No
    - 1 = Yes
19. **Sex**: Gender of the respondent.
    - 0 = Female
    - 1 = Male
20. **Age**: 13-level age category.
    - Scale 1-13
        - 1 = 18-24
        - 2 = 25-29
        - 3 = 30-34
        - 4 = 35-39
        - 5 = 40-44
        - 6 = 45-49
        - 7 = 50-54
        - 8 = 55-59

    - 9 = 60-64
    - 10 = 65-69
    - 11 = 70-74
    - 12 = 75-79
    - 13 = 80 or older

21. **Education**: Education level of the respondent.

    - 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)

22. **Income**: Income scale of the respondent.

    - Scale 1-8
        - 1 = Less than $10,000
        - 2 = $10,000 to 14,999
        - 3 = $15,000 to 19,999
        - 4 = $20,000 to 24,999
        - 5 = $25,000 to 34,999
        - 6 = $35,000 to 49,999
        - 7 = $50,000 to 74,999
        - 8 = $75,000 or more

Summary of Columns

Column Name	Description	Data Type	Categories
Diabetes_012	Diabetes status of the respondent	Categorical	0, 1, 2
HighBP	High blood pressure status	Categorical	0, 1
HighChol	High cholesterol status	Categorical	0, 1
CholCheck	Whether the respondent had a cholesterol check in the past 5 years	Categorical	0, 1
BMI	Body Mass Index, a numeric value representing body weight relative to height	Numeric	-
Smoker	Whether the respondent has smoked at least 100 cigarettes in their entire life	Categorical	0, 1
Stroke	Whether the respondent has ever been told they had a stroke	Categorical	0, 1
HeartDiseaseorAttack	Whether the respondent has coronary heart disease (CHD) or myocardial infarction (MI)	Categorical	0, 1

Column Name	Description	Data Type	Categories
PhysActivity	Physical activity in the past 30 days, not including job-related activities	Categorical	0, 1
Fruits	Whether the respondent consumes fruit one or more times per day	Categorical	0, 1
Veggies	Whether the respondent consumes vegetables one or more times per day	Categorical	0, 1
HvyAlcoholConsump	Heavy drinking status (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week)	Categorical	0, 1
AnyHealthcare	Whether the respondent has any kind of health care coverage, including health insurance, prepaid plans such as HMO, etc.	Categorical	0, 1
NoDocbcCost	Whether there was a time in the past 12 months when the respondent needed to see a doctor but could not because of cost	Categorical	0, 1
GenHlth	General health status of the respondent	Categorical	1, 2, 3, 4, 5
MentHlth	Number of days during the past 30 days the respondent's mental health was not good	Numeric	1-30
PhysHlth	Number of days during the past 30 days the respondent's physical health was not good	Numeric	1-30
DiffWalk	Whether the respondent has serious difficulty walking or climbing stairs	Categorical	0, 1
Sex	Gender of the respondent	Categorical	0, 1
Age	13-level age category	Categorical	1-13
Education	Education level of the respondent	Categorical	1-6
Income	Income scale of the respondent	Categorical	1-8

This dataset enables a comprehensive analysis of health indicators and their correlation with diabetes status, providing valuable insights for predictive modeling and public health studies.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from imblearn.over_sampling import SMOTE
from sklearn.metrics import mean_squared_error, accuracy_score, f1_score, confusion
from sklearn.model_selection import train_test_split, KFold
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
from sklearn.ensemble import StackingClassifier
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:
```
sns.set_style('darkgrid')
plt.rcParams['font.size'] = 14
plt.rcParams['figure.figsize'] = (10,6)
plt.rcParams['figure.facecolor'] = '#00000000'
pd.set_option('display.max_columns', None)
%matplotlib inline
```

Data Analysis

Data Loading

In [ ]:
```
data = pd.read_csv("/content/gdrive/MyDrive/diabetes_012_health_indicators_BRFSS201
```

In [ ]:
```
data
```

Out[ ]:

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseasec
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	
...	...	...	...	...	...	...	...	
253675	0.0	1.0	1.0	1.0	45.0	0.0	0.0	
253676	2.0	1.0	1.0	1.0	18.0	0.0	0.0	
253677	0.0	0.0	0.0	1.0	28.0	0.0	0.0	
253678	0.0	1.0	0.0	1.0	23.0	0.0	0.0	
253679	2.0	1.0	1.0	1.0	25.0	0.0	0.0	

253680 rows × 22 columns

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Diabetes_012          253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

In [ ]:
```
data.isna().sum()
```

Out[ ]:
```
Diabetes_012            0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

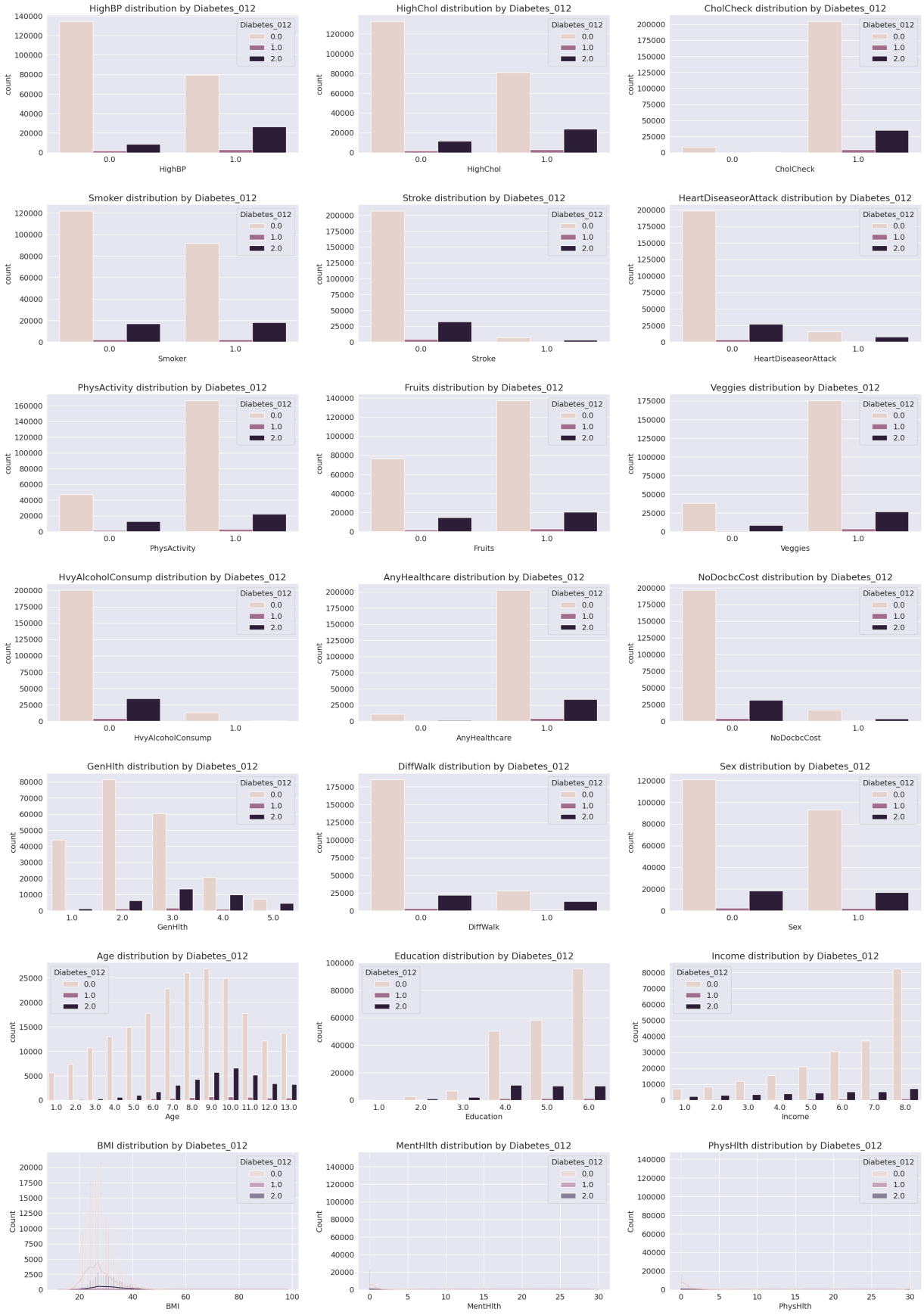
In [ ]:
```
data.describe()
```

Out[ ]:

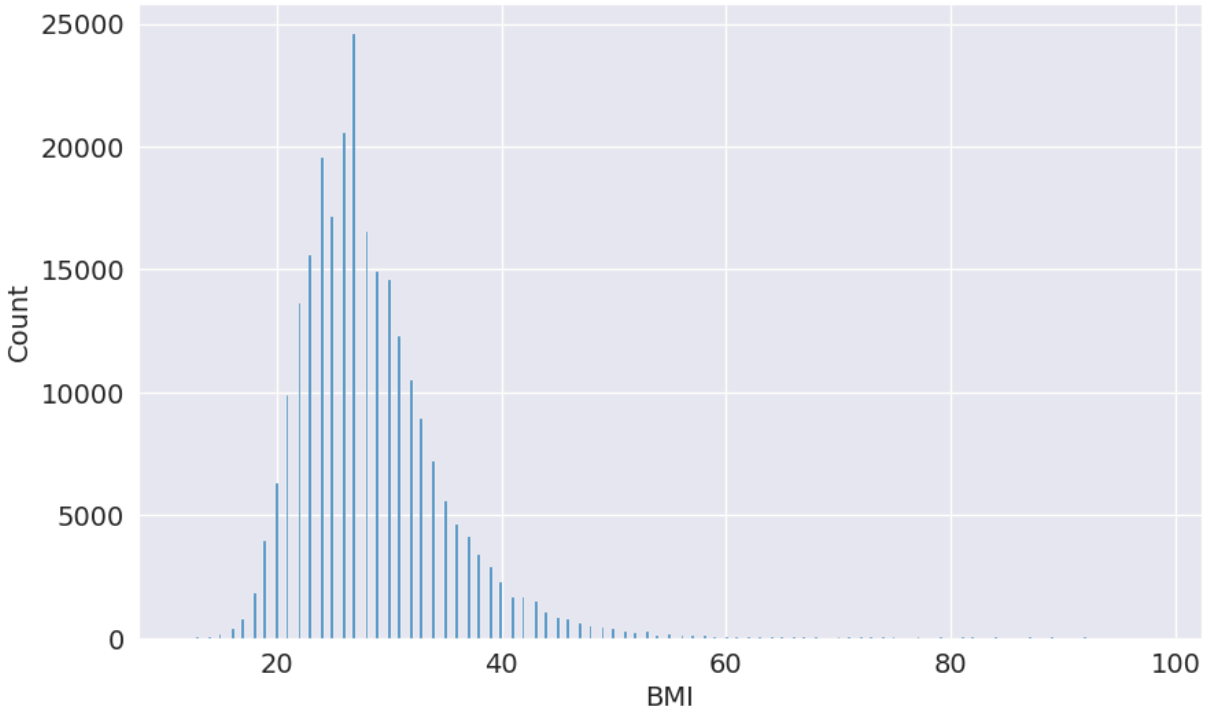
	Diabetes_012	HighBP	HighChol	CholCheck	BMI	S
count	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.
mean	0.296921	0.429001	0.424121	0.962670	28.382364	0.
std	0.698160	0.494934	0.494210	0.189571	6.608694	0.
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.
max	2.000000	1.000000	1.000000	1.000000	98.000000	1.

EDA

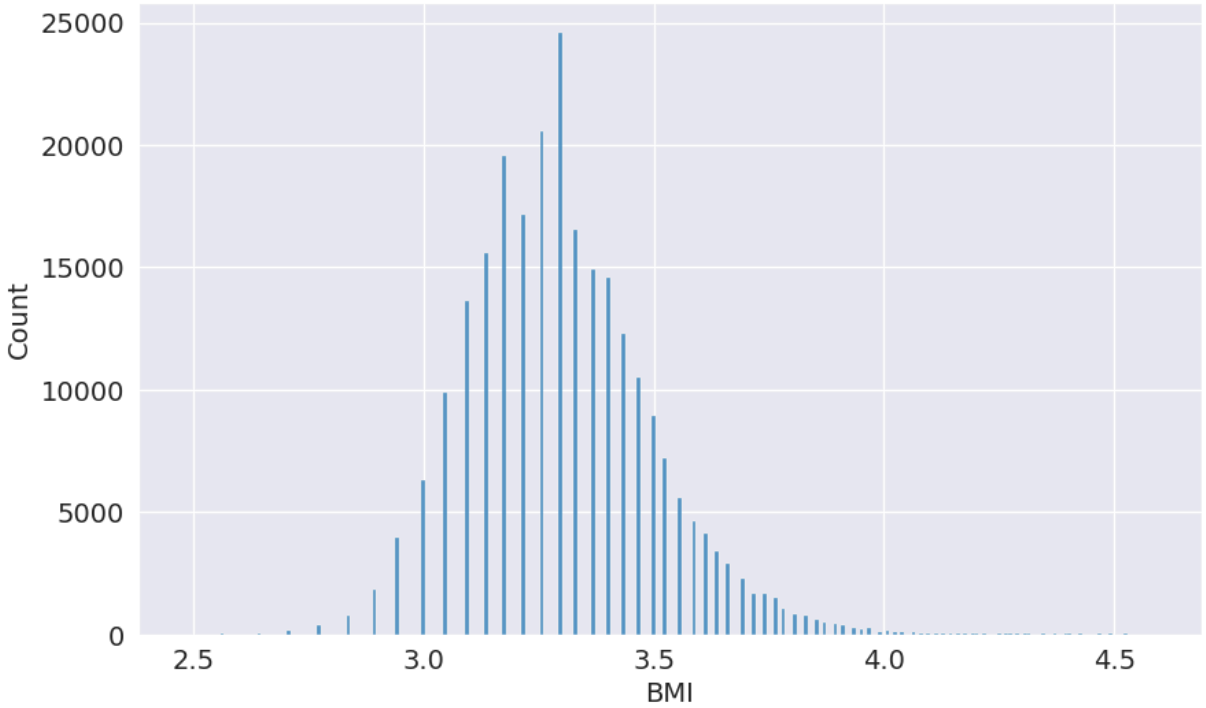
In [ ]:
```
categorical_cols = [
    'HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack',
    'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare',
    'NoDocbcCost', 'GenHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income'
]
numeric_cols = ['BMI', 'MentHlth', 'PhysHlth']

# Create a grid of subplots
fig, axes = plt.subplots(8, 3, figsize=(25, 40))
fig.tight_layout(pad=5.0)

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

# Plot each categorical column
for i, col in enumerate(categorical_cols):
    sns.countplot(data=data, x=col, hue='Diabetes_012', ax=axes[i])
    axes[i].set_title(f'{col} distribution by Diabetes_012')

# Plot each numeric column
for i, col in enumerate(numeric_cols):
    sns.histplot(data=data, x=col, hue='Diabetes_012', kde=True, ax=axes[len(catego
    axes[len(categorical_cols) + i].set_title(f'{col} distribution by Diabetes_012'

# Hide any remaining empty subplots
for j in range(len(categorical_cols) + len(numeric_cols), len(axes)):
    fig.delaxes(axes[j])

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



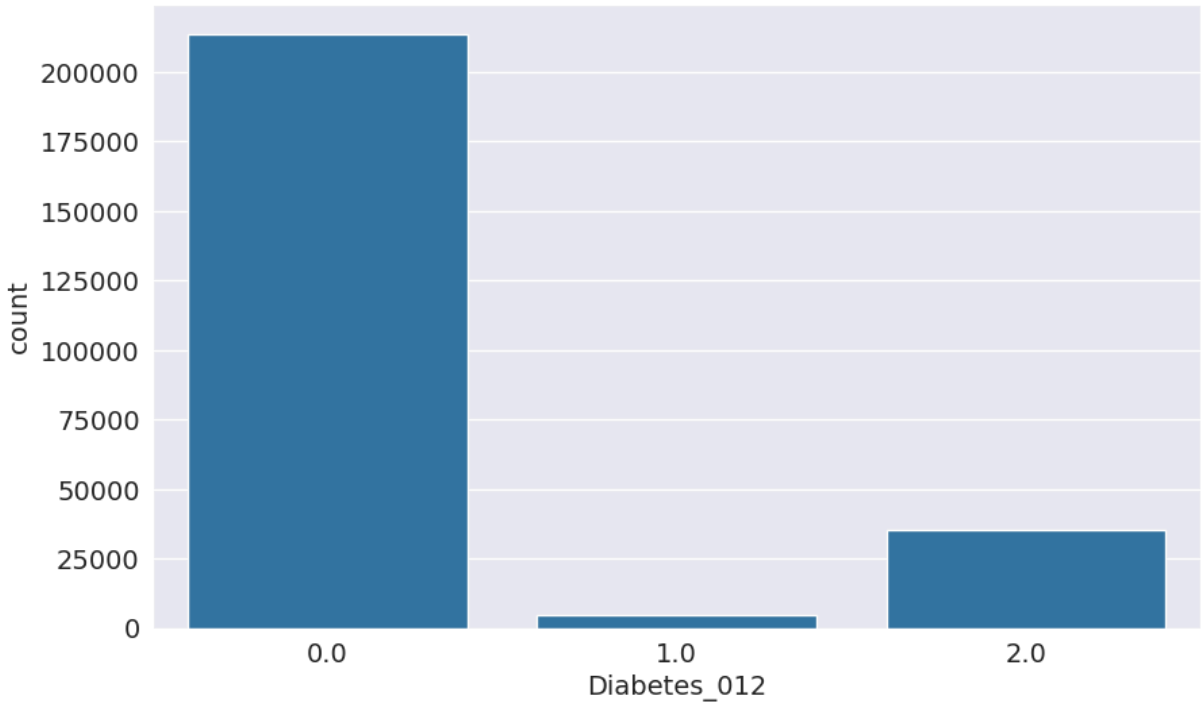
In [ ]:
```
sns.histplot(data, x='BMI')
plt.show()
```



In [ ]:
```
sns.histplot(x=np.log(data['BMI']))
plt.show()
```



In [ ]:
```
sns.countplot(data, x='Diabetes_012')
plt.show()
```



Identifying correlation

In [ ]:
```
data.corr()
```

Out[ ]:

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker
Diabetes_012	1.000000	0.271596	0.209085	0.067546	0.224379	0.062914
HighBP	0.271596	1.000000	0.298199	0.098508	0.213748	0.096991
HighChol	0.209085	0.298199	1.000000	0.085642	0.106722	0.091299
CholCheck	0.067546	0.098508	0.085642	1.000000	0.034495	-0.009929
BMI	0.224379	0.213748	0.106722	0.034495	1.000000	0.013804
Smoker	0.062914	0.096991	0.091299	-0.009929	0.013804	1.000000
Stroke	0.107179	0.129575	0.092620	0.024158	0.020153	0.061173
HeartDiseaseorAttack	0.180272	0.209361	0.180765	0.044206	0.052904	0.114441
PhysActivity	-0.121947	-0.125267	-0.078046	0.004190	-0.147294	-0.087401
Fruits	-0.042192	-0.040555	-0.040859	0.023849	-0.087518	-0.077666
Veggies	-0.058972	-0.061266	-0.039874	0.006121	-0.062275	-0.030678
HvyAlcoholConsump	-0.057882	-0.003972	-0.011543	-0.023730	-0.048736	0.101619
AnyHealthcare	0.015410	0.038425	0.042230	0.117626	-0.018471	-0.023251
NoDocbcCost	0.035436	0.017358	0.013310	-0.058255	0.058206	0.048946
GenHlth	0.302587	0.300530	0.208426	0.046589	0.239185	0.163143
MentHlth	0.073507	0.056456	0.062069	-0.008366	0.085310	0.092196
PhysHlth	0.176287	0.161212	0.121751	0.031775	0.121141	0.116460
DiffWalk	0.224239	0.223618	0.144672	0.040585	0.197078	0.122463
Sex	0.031040	0.052207	0.031205	-0.022115	0.042950	0.093662
Age	0.185026	0.344452	0.272318	0.090321	-0.036618	0.120641
Education	-0.130517	-0.141358	-0.070802	0.001510	-0.103932	-0.161955
Income	-0.171483	-0.171235	-0.085459	0.014259	-0.100069	-0.123937

In [ ]:
```
data.corr()['Diabetes_012'].sort_values(ascending=False)
```

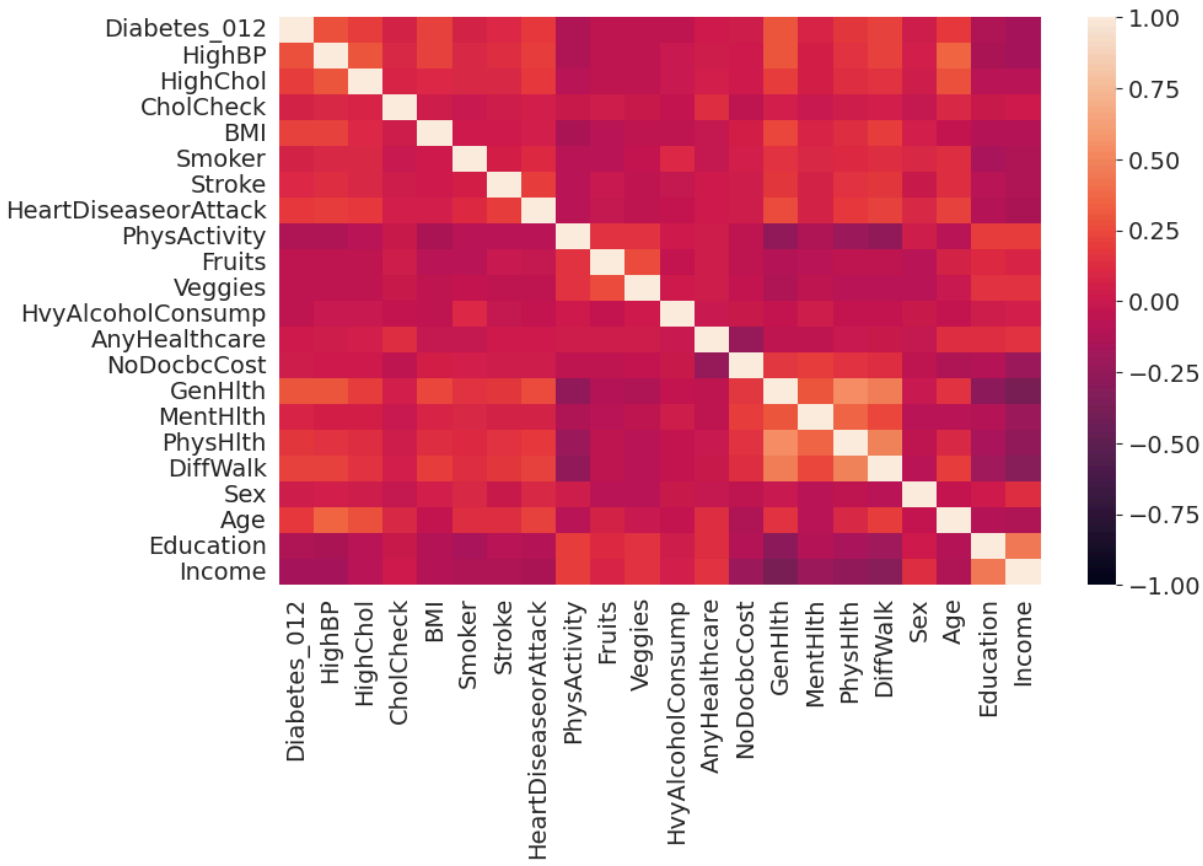
Out[ ]:
```
Diabetes_012            1.000000
GenHlth                 0.302587
HighBP                  0.271596
BMI                     0.224379
DiffWalk                0.224239
HighChol                0.209085
Age                     0.185026
HeartDiseaseorAttack    0.180272
PhysHlth                0.176287
Stroke                  0.107179
MentHlth                0.073507
CholCheck               0.067546
Smoker                  0.062914
NoDocbcCost             0.035436
Sex                     0.031040
AnyHealthcare           0.015410
Fruits                 -0.042192
HvyAlcoholConsump      -0.057882
Veggies                -0.058972
PhysActivity           -0.121947
Education              -0.130517
Income                 -0.171483
Name: Diabetes_012, dtype: float64
```

In [ ]:
```
corr_matrix = data.corr()
corr_pairs = corr_matrix.unstack()
sorted_pairs = corr_pairs.sort_values(kind='quicksort')
high_corr_pairs = sorted_pairs[sorted_pairs!=1]
threshold = 0.35
high_corr_pairs = high_corr_pairs[high_corr_pairs.abs() > threshold].sort_values(as
pd.set_option('display.max_rows', 15)
high_corr_pairs
```

Out[ ]:
```
GenHlth    PhysHlth     0.524364
PhysHlth   GenHlth      0.524364
DiffWalk   PhysHlth     0.478417
PhysHlth   DiffWalk     0.478417
DiffWalk   GenHlth      0.456920
GenHlth    DiffWalk     0.456920
Income     Education    0.449106
Education  Income       0.449106
PhysHlth   MentHlth     0.353619
MentHlth   PhysHlth     0.353619
GenHlth    Income      -0.370014
Income     GenHlth     -0.370014
dtype: float64
```

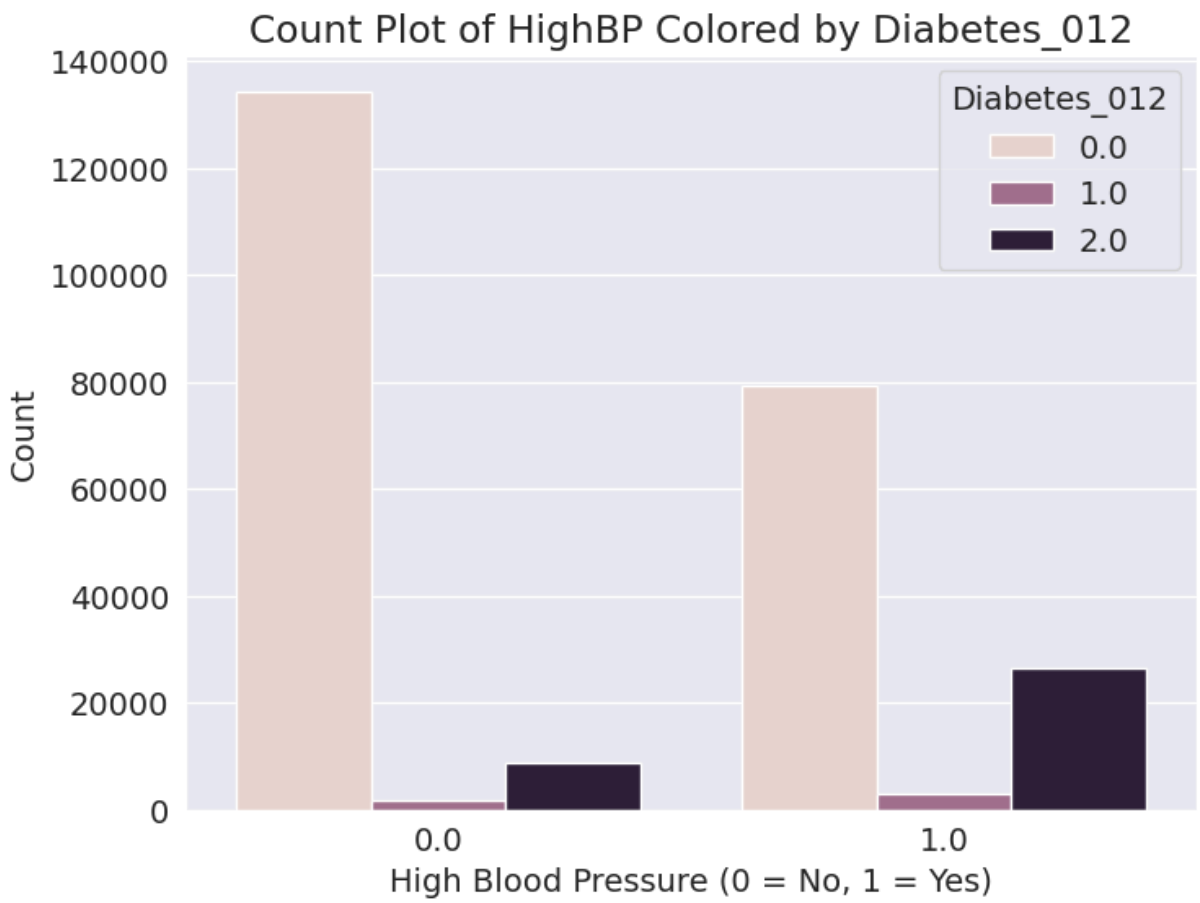
In [ ]:
```
sns.heatmap(data.corr(), vmin=-1, vmax=1)
```

Out[ ]:
```
<Axes: >
```



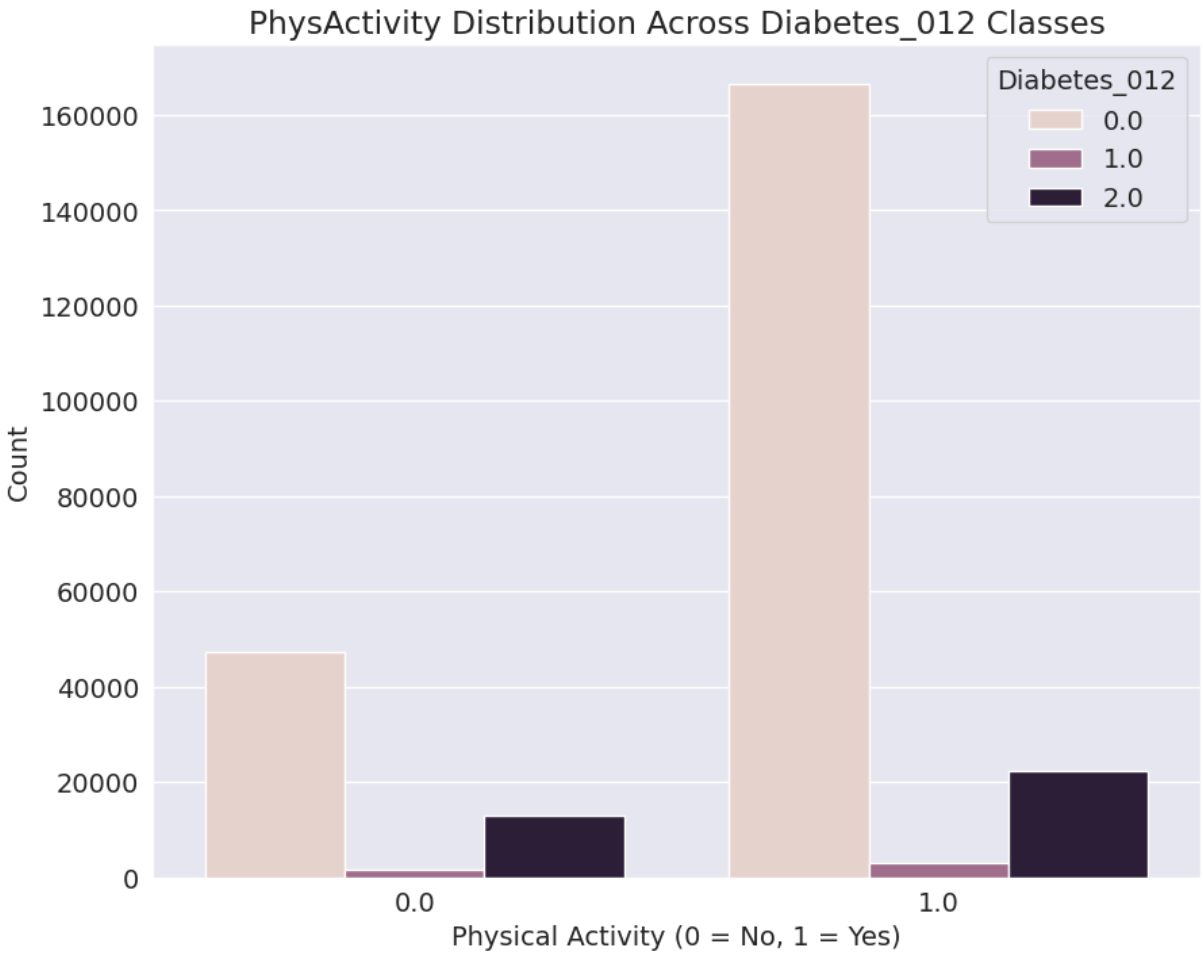
In [ ]:

```
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='HighBP', hue='Diabetes_012')
plt.title('Count Plot of HighBP Colored by Diabetes_012')
plt.xlabel('High Blood Pressure (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.legend(title='Diabetes_012')
plt.show()
```



In [ ]:
```
def plotGrouped(column):
    plt.figure(figsize=(10, 8))
    sns.countplot(data=data, x=column, hue='Diabetes_012')
    plt.title('PhysActivity Distribution Across Diabetes_012 Classes')
    plt.xlabel('Physical Activity (0 = No, 1 = Yes)')
    plt.ylabel('Count')
    plt.legend(title='Diabetes_012')
    plt.show()
```

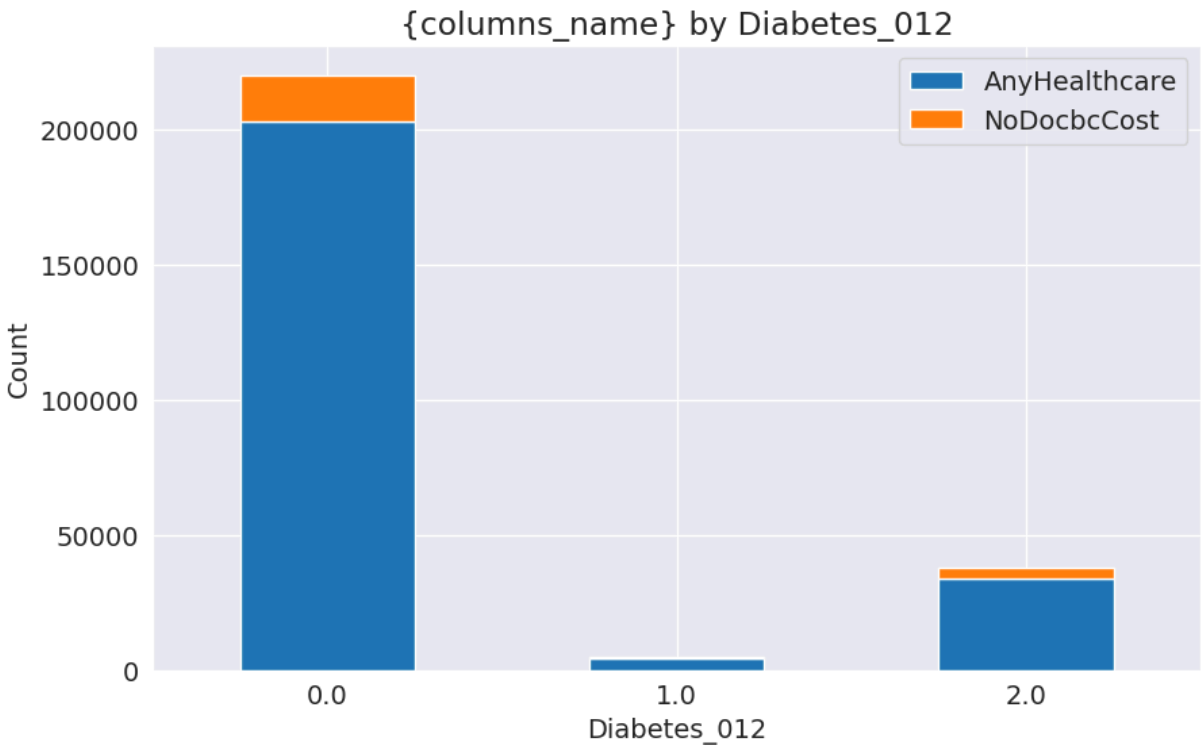
In [ ]:
```
plotGrouped('PhysActivity')
```



In [ ]:
```
def plotStacked(columns, columns_name):
    healthcare_df = data.groupby('Diabetes_012')[columns].sum().reset_index()

    plt.figure(figsize=(8, 6))
    healthcare_df.set_index('Diabetes_012').plot(kind='bar', stacked=True)
    plt.title('{columns_name} by Diabetes_012')
    plt.xlabel('Diabetes_012')
    plt.ylabel('Count')
    plt.xticks(rotation=0)
    plt.legend(columns)
    plt.show()
```

In [ ]:
```
plotStacked(['AnyHealthcare', 'NoDocbcCost'], 'Healthcare Access Rating')
```

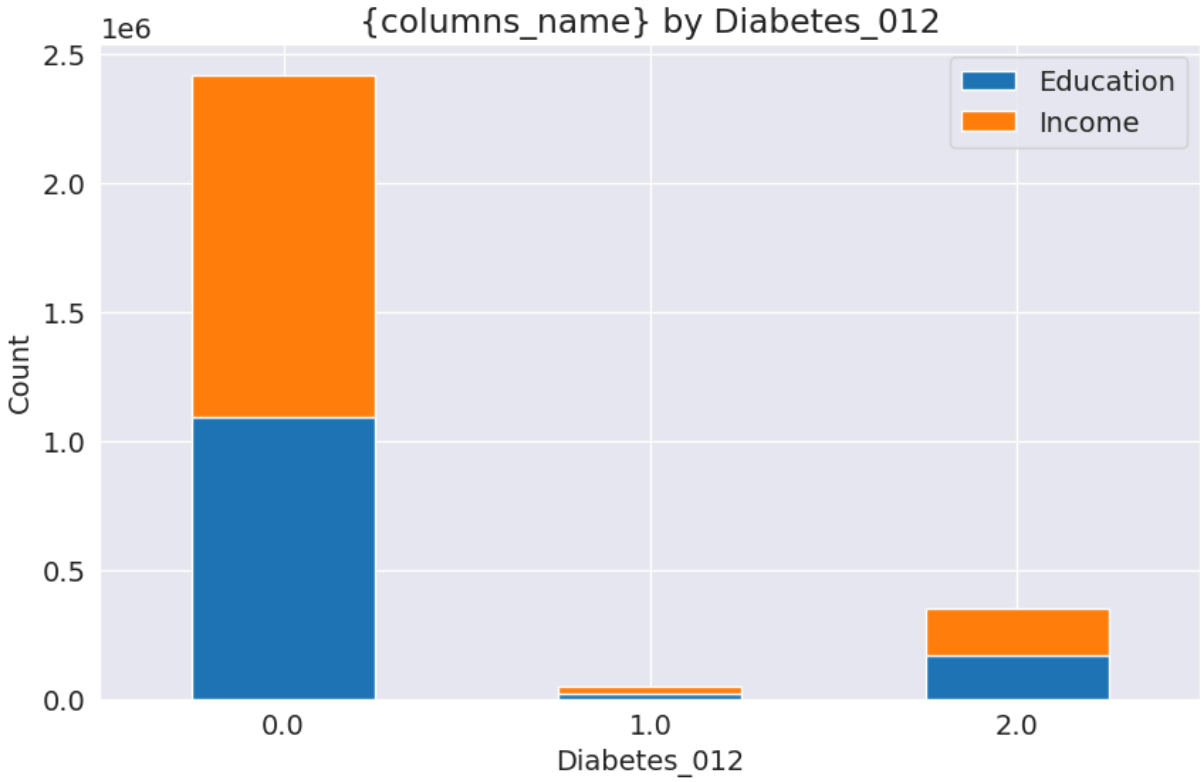
```
<Figure size 800x600 with 0 Axes>
```



In [ ]:

In [ ]:
```
plotStacked(['Education', 'Income'], 'Income and Education Status')
```

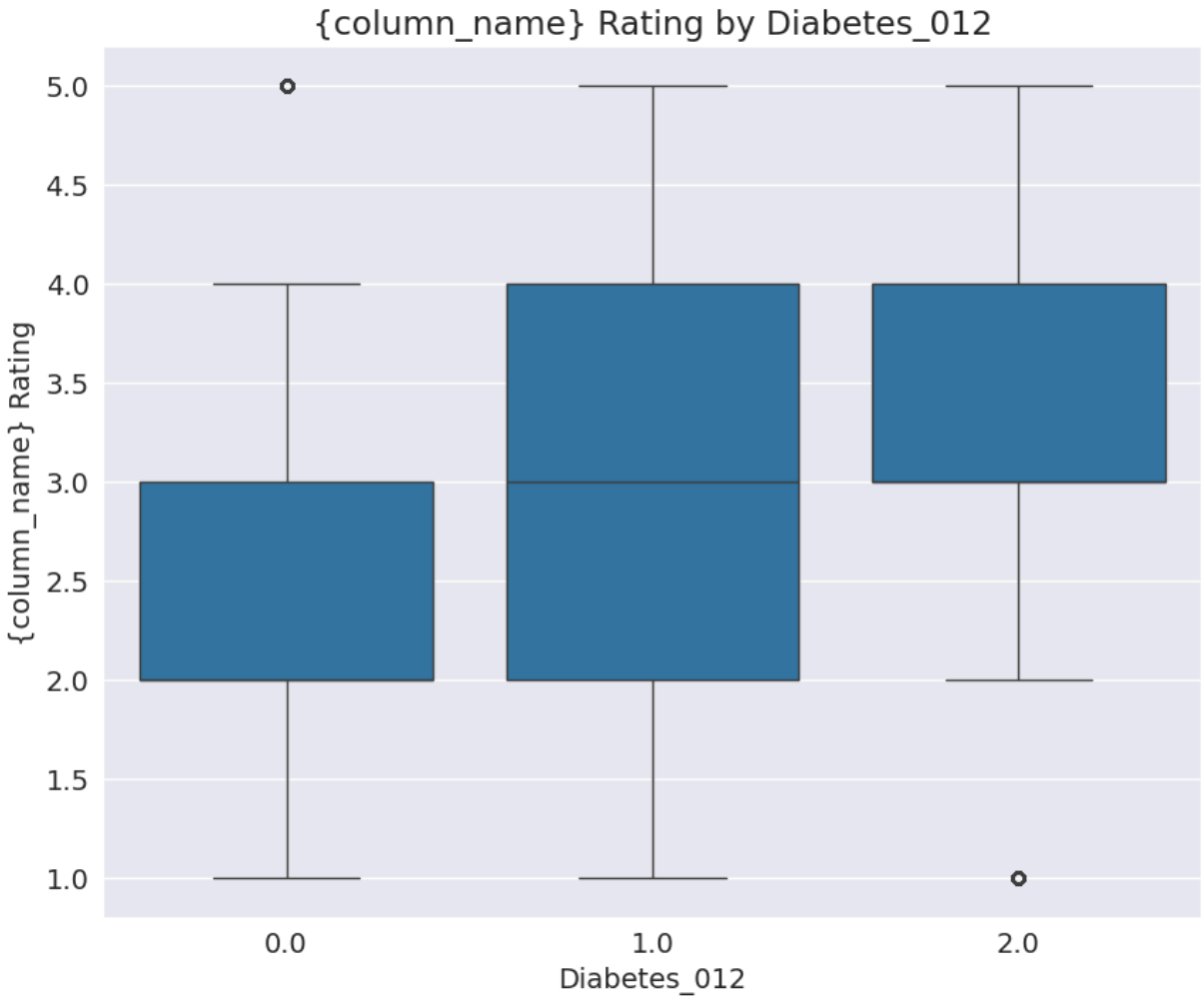
```
<Figure size 800x600 with 0 Axes>
```



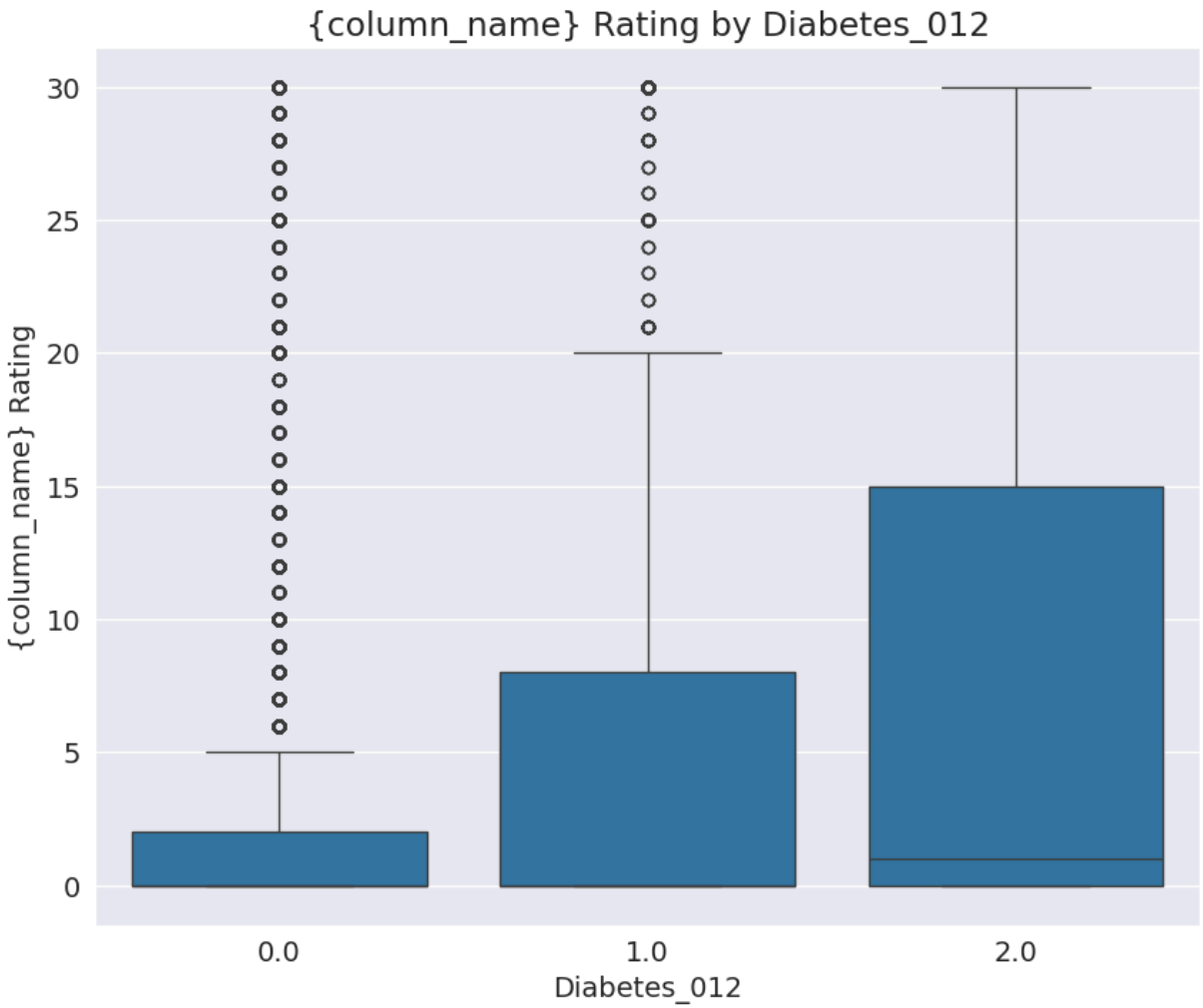
In [ ]:
```
def plotBox(column, column_name):
    plt.figure(figsize=(10, 8))
    sns.boxplot(data=data, x='Diabetes_012', y=column)
```

```
    plt.title('{column_name} Rating by Diabetes_012')
    plt.xlabel('Diabetes_012')
    plt.ylabel('{column_name} Rating')
    plt.show()
```

In [ ]:
```
plotBox('GenHlth', 'General Health')
```



In [ ]:
```
plotBox('PhysHlth', 'Physical Health')
```



Feature Engineering

In [ ]:
```
data['log_BMI'] = np.log(data['BMI'])
data['log_MentHlth'] = np.log(data['MentHlth'])
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:396: RuntimeWarnin
g: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [ ]:
```
data['log_BMI'] = data['log_BMI'].replace([np.inf, -np.inf], np.nan)
data['log_MentHlth'] = data['log_MentHlth'].replace([np.inf, -np.inf], np.nan)
```

In [ ]:
```
data['CardioVascularRisk'] = (data['HeartDiseaseorAttack'])*1.8 + (data['Stroke'])*
```

In [ ]:
```
def mobility_score(row):
    return row['PhysActivity']*(-1.22) + (row['DiffWalk'])*2.24

data['MobilityScore'] = data.apply(mobility_score, axis=1)
```

In [ ]:
```
def substance_use(row):
    return row['Smoker']*0.62 + row['HvyAlcoholConsump']*(-0.57)
```

```
data['SubstanceUse'] = data.apply(substance_use, axis=1)
```

In [ ]:
```
def diet_quality(row):
    return row['Fruits']*(-0.59) + row['Veggies']*(-0.42)

data['DietQuality'] = data.apply(diet_quality, axis=1)
```

In [ ]:
```
def socioeconomic_status(row):
    return ((row['Education']-1)/5)*(-1.71) + ((row['Income']-1)/7)*(-1.30)

data['SocioeconomicStatus'] = data.apply(socioeconomic_status, axis=1)
```

In [ ]:
```
def healthcare_access(row):
    return (row['AnyHealthcare'] * 0.15) + (row['NoDocbcCost'] * 0.35)

data['HealthcareAccess'] = data.apply(healthcare_access, axis=1)
```

In [ ]:
```
data['MetabolicRisk'] = (data['log_BMI'])*2.3 + data['HighBP']*2.71 + data['HighCho
```

In [ ]:
```
data['BMI_Age'] = ((data['BMI']-12)/86) * ((data['Age']-1)/12)
```

In [ ]:
```
data['BP_Chol'] = data['HighBP'] * data['HighChol']
```

In [ ]:
```
data['Sum_Health_Conditions'] = data[['HighBP', 'HighChol', 'HeartDiseaseorAttack',
```

In [ ]:
```
data['HealthSeverity'] = (data['GenHlth']-1)/4*3.02 + ((30 - data['PhysHlth']) / 30
```

In [ ]:
```
data.sample()
```

Out[ ]:

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseor
55289	0.0	1.0	1.0	1.0	25.0	0.0	0.0	

In [ ]:
```
data.corr()['Diabetes_012'].abs().sort_values()
```

Out[ ]:
```
AnyHealthcare            0.015410
Sex                      0.031040
NoDocbcCost              0.035436
HealthcareAccess         0.041490
Fruits                   0.042192
                           ...
GenHlth                  0.302587
BMI_Age                  0.308930
Sum_Health_Conditions    0.321549
MetabolicRisk            0.332414
Diabetes_012             1.000000
Name: Diabetes_012, Length: 35, dtype: float64
```

Data preprocessing and Balancing Classes

In [ ]:
```
data.columns
```

Out[ ]:
```
Index(['Diabetes_012', 'HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker',
       'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'log_BMI', 'log_MentHlth', 'CardioVascularRisk', 'MobilityScore',
       'SubstanceUse', 'DietQuality', 'SocioeconomicStatus',
       'HealthcareAccess', 'MetabolicRisk', 'BMI_Age', 'BP_Chol',
       'Sum_Health_Conditions', 'HealthSeverity'],
      dtype='object')
```

In [ ]:
```
input_cols_simple = ['HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker',
       'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income']
```

In [ ]:
```
input_cols_eng = ['MetabolicRisk', 'Sum_Health_Conditions', 'BMI_Age',
                  'BP_Chol', 'MobilityScore', 'CardioVascularRisk',
                  'SocioeconomicStatus', 'SubstanceUse', 'DietQuality',
                  'GenHlth', 'HealthSeverity', 'HealthcareAccess']
```

In [255…
```
input_cols_all = input_cols_simple + input_cols_eng
```

In [256…
```
target_col = ['Diabetes_012']
```

In [257…
```
imputer = SimpleImputer(strategy='mean').fit(data[input_cols_eng])
data[input_cols_eng] = imputer.transform(data[input_cols_eng])
```

In [258…
```
scaler = MinMaxScaler().fit(data[input_cols_eng])
data[input_cols_eng] = scaler.transform(data[input_cols_eng])
```

In [259…
```
train_val_inputs, test_inputs, train_val_target, test_target = train_test_split(dat
```

In [260…
```
smote = SMOTE(random_state=42)
train_val_inputs_resampled, train_val_target_resampled = smote.fit_resample(train_v
```

In [261…
```
train_val_target_resampled.value_counts()
```

Out[261…
```
Diabetes_012
0.0             170908
1.0             170908
2.0             170908
Name: count, dtype: int64
```

In [262…
```
train_inputs, val_inputs, train_target, val_target = train_test_split(train_val_inp
```

In [263…
```
data['Diabetes_012'].value_counts()
```

Out[263…
```
Diabetes_012
0.0    213703
2.0     35346
1.0      4631
Name: count, dtype: int64
```

In [264…
```
train_target.value_counts()
```

Out[264…
```
Diabetes_012
0.0             153844
1.0             153821
2.0             153786
Name: count, dtype: int64
```

Hardcoded Model

In [ ]:
```
def returnNo(df):
    return np.full(len(df), 0)
```

In [ ]:
```
def returnRandom(df):
    return np.random.choice([0,1,2], len(df))
```

In [ ]:
```
def rmse(inputs, target):
    return mean_squared_error(inputs, target)
```

In [ ]:
```
rmse(train_target, returnNo(train_inputs))
```

Out[ ]:
```
1.6664066173873282
```

In [ ]:
```
accuracy_score(train_target, returnNo(train_inputs))
```

Out[ ]:
```
0.33339184442118447
```

In [ ]:
```
rmse(train_target, returnRandom(train_inputs))
```

Out[ ]:
```
1.33298876803821
```

In [ ]:
```
accuracy_score(train_target, returnRandom(train_inputs))
```

Out[ ]:
```
0.3328825812491467
```

Baseline Models

In [ ]:
```
def predict_model(model):
    model = model.fit(train_inputs, train_target)
    train_preds = model.predict(train_inputs)
    val_preds = model.predict(val_inputs)
    train_score = model.score(train_inputs, train_target)
    val_score = accuracy_score(val_preds, val_target)
    train_rmse = rmse(train_target, train_preds)
    val_rmse = rmse(val_target, val_preds)
    train_f1 = f1_score(train_target, train_preds, average='macro')
```

```
    val_f1 = f1_score(val_target, val_preds, average='macro')
    train_cm = confusion_matrix(train_target, train_preds, normalize='true')
    val_cm = confusion_matrix(train_target, train_preds, normalize='true')
    print('Training Score: ', train_score, ', Training RMSE: ', train_rmse, ', Trai
    print('Validation Score: ', val_score, ', validation RMSE: ', val_rmse, ', Vali
    print('Training F1 Scores: ', f1_score(train_target, train_preds, average=None)
    print('Validation F1 Scores: ', f1_score(val_target, val_preds, average=None))
    print('Training Confusion Matrix: \n', train_cm)
    print('Validation Confusion Matrix: \n', val_cm)
    return model
```

In [ ]:
```
predict_model(LogisticRegression(random_state=42, solver='liblinear'))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Please chang
e the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Training Score:  0.5167179180454696 , Training RMSE:  0.851979950200563 , Training F
1 Score:  0.49760067088986704
Validation Score:  0.5166071811674761 , validation RMSE:  0.8587365670040762 , Valid
ation F1 Score:  0.4977606701361748
Training F1 Scores:  [0.63168023 0.31607822 0.54504356]
Validation F1 Scores:  [0.63005077 0.31912528 0.54410596]
Training Confusion Matrix:
 [[0.6998128  0.13083383 0.16935337]
 [0.31667328 0.24982285 0.43350388]
 [0.19935495 0.20013525 0.6005098 ]]
Validation Confusion Matrix:
 [[0.6998128  0.13083383 0.16935337]
 [0.31667328 0.24982285 0.43350388]
 [0.19935495 0.20013525 0.6005098 ]]
```

Out[ ]:
```
LogisticRegression(random_state=42, solver='liblinear')
```

Decision Tree Models

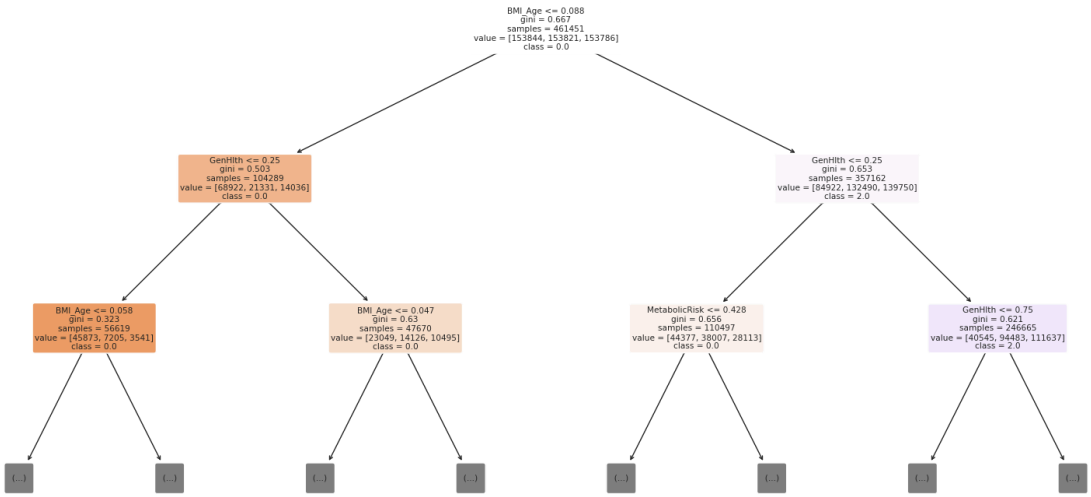
In [ ]:
```
def decision_tree_model(**params):
    return predict_model(DecisionTreeClassifier(random_state=42, **params))
```

In [ ]:
```
%%time
model = decision_tree_model()
```

```
Training Score:  0.9960645875726784 , Training RMSE:  0.013856292434082925 , Trainin
g F1 Score:  0.9960669257018743
Validation Score:  0.8281356659450393 , validation RMSE:  0.4607883291400932 , Valid
ation F1 Score:  0.8276892194545905
Training F1 Scores:  [0.99419703 0.99905656 0.99494719]
Validation F1 Scores:  [0.82224646 0.88831229 0.77250892]
Training Confusion Matrix:
 [[9.98927485e-01 6.50009100e-06 1.06601492e-03]
 [1.73578380e-03 9.98225210e-01 3.90063775e-05]
 [8.85646288e-03 1.04040680e-04 9.91039496e-01]]
Validation Confusion Matrix:
 [[9.98927485e-01 6.50009100e-06 1.06601492e-03]
 [1.73578380e-03 9.98225210e-01 3.90063775e-05]
 [8.85646288e-03 1.04040680e-04 9.91039496e-01]]
CPU times: user 9.05 s, sys: 13 ms, total: 9.06 s
Wall time: 9.16 s
```

In [ ]:
```
%%time
def max_depth_error(md):
    model = DecisionTreeClassifier(max_depth=md, random_state=42)
    model.fit(train_inputs, train_target)
    train_acc = 1 - model.score(train_inputs, train_target)
    val_acc = 1 - model.score(val_inputs, val_target)
    return {'Max Depth': md, 'Training Error': train_acc, 'Validation Error': val_a


errors_df = pd.DataFrame([max_depth_error(md) for md in range(1, 26)])

plt.figure()
plt.plot(errors_df['Max Depth'], errors_df['Training Error'])
plt.plot(errors_df['Max Depth'], errors_df['Validation Error'])
plt.title('Training vs. Validation Error')
plt.xticks(range(0,21, 2))
plt.xlabel('Max. Depth')
plt.ylabel('Prediction Error (1 - Accuracy)')
plt.legend(['Training', 'Validation'])

errors_df
```

In [ ]:
```
class_names = [str(cls) for cls in model.classes_]

plt.figure(figsize=(20, 10))
plot_tree(model, max_depth=2, feature_names=train_inputs.columns, class_names=class
plt.show()
```



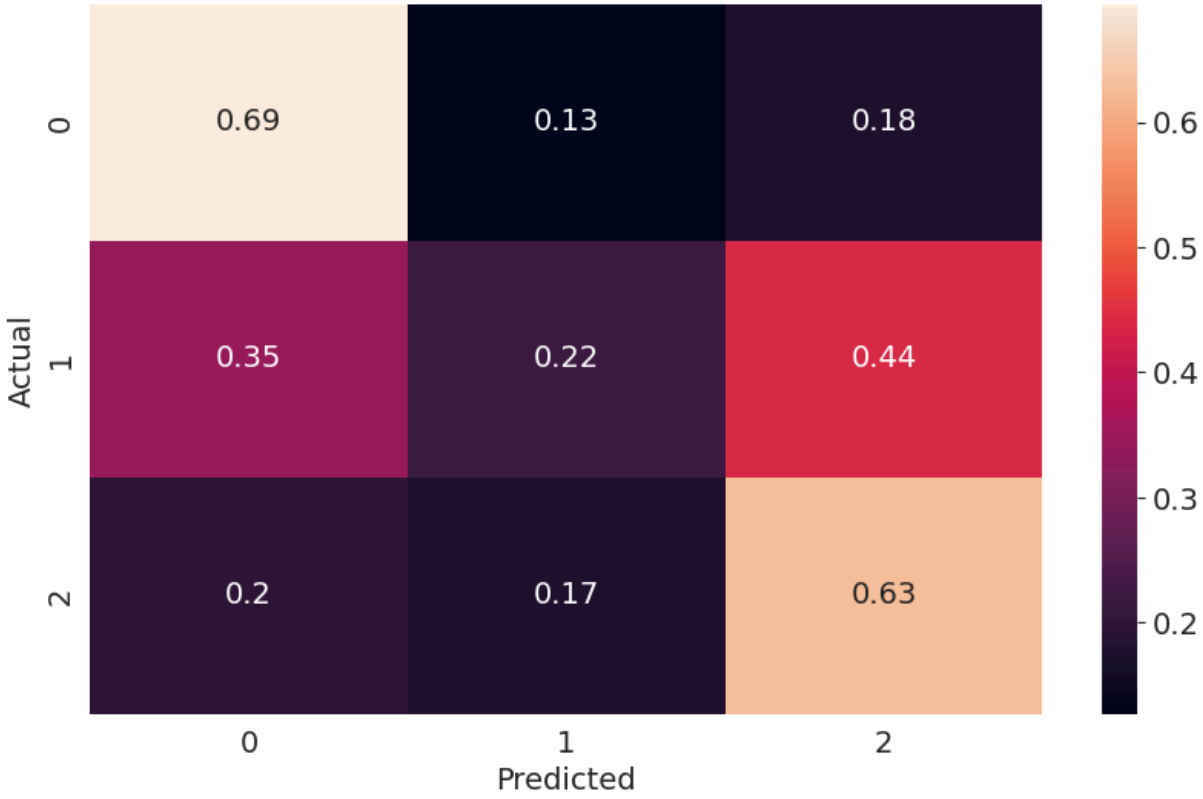
Random Forests

In [ ]:
```
def random_forest_model(**params):
    model = RandomForestClassifier(random_state=42, n_jobs=-1, **params)
    model.fit(train_inputs, train_target)
    train_preds = model.predict(train_inputs)
    val_preds = model.predict(val_inputs)
    trainingScore = model.score(train_inputs, train_target)
    validationScore = accuracy_score(val_preds, val_target)
    train_rmse = rmse(train_target, train_preds)
    val_rmse = rmse(val_target, val_preds)
    train_f1 = f1_score(train_target, train_preds, average='macro')
    val_f1 = f1_score(val_target, val_preds, average='macro')
    train_cm = confusion_matrix(train_target, train_preds, normalize='true')
    val_cm = confusion_matrix(val_target, val_preds, normalize='true')
    val_cmn = val_cm.astype('float') / val_cm.sum(axis=1)[:, np.newaxis]
    train_cmn = train_cm.astype('float') / train_cm.sum(axis=1)[:, np.newaxis]
    print('Training Score: ', trainingScore, ', Training RMSE: ', train_rmse, ', Tr
    print('Validation Score: ', validationScore, ', validation RMSE: ', val_rmse, '
    print('Training F1 Scores: ', f1_score(train_target, train_preds, average=None)
    print('Validation F1 Scores: ', f1_score(val_target, val_preds, average=None))
    print('Training Confusion Matrix: \n', train_cmn)
    print('Validation Confusion Matrix: \n', val_cmn)
    return model
```

In [267…
```
train_target = train_target.values.ravel()
val_target = val_target.values.ravel()
test_target = test_target.values.ravel()

rf_model = RandomForestClassifier(random_state=42, n_jobs=-1,
                                  n_estimators=14, max_depth=8,
                                  min_samples_split= 10, min_samples_leaf= 3,
                                  max_leaf_nodes= 512, max_features= 'sqrt',
                                  bootstrap= False)
rf_model.fit(train_inputs, train_target)
```

```
train_preds = rf_model.predict(train_inputs)
val_preds = rf_model.predict(val_inputs)
trainingScore = rf_model.score(train_inputs, train_target)
validationScore = accuracy_score(val_preds, val_target)
train_rmse = rmse(train_target, train_preds)
val_rmse = rmse(val_target, val_preds)
train_f1 = f1_score(train_target, train_preds, average='macro')
val_f1 = f1_score(val_target, val_preds, average='macro')
train_cm = confusion_matrix(train_target, train_preds, normalize='true')
val_cm = confusion_matrix(val_target, val_preds, normalize='true')
val_cmn = val_cm.astype('float') / val_cm.sum(axis=1)[:, np.newaxis]
train_cmn = train_cm.astype('float') / train_cm.sum(axis=1)[:, np.newaxis]
print('Training Score: ', trainingScore, ', Training RMSE: ', train_rmse, ', Traini
print('Validation Score: ', validationScore, ', validation RMSE: ', val_rmse, ', Va
print('Training F1 Scores: ', f1_score(train_target, train_preds, average=None))
print('Validation F1 Scores: ', f1_score(val_target, val_preds, average=None))
print('Training Confusion Matrix: \n', train_cmn)
print('Validation Confusion Matrix: \n', val_cmn)


test_preds = rf_model.predict(test_inputs)
testScore = rf_model.score(test_inputs, test_target)
test_rmse = rmse(test_target, test_preds)
test_f1 = f1_score(test_target, test_preds, average='macro')
test_cm = confusion_matrix(test_target, test_preds, normalize='true')
test_cmn = test_cm.astype('float') / test_cm.sum(axis=1)[:, np.newaxis]
print('Testing Score: ', testScore, ', Testing RMSE: ', test_rmse, ', Testing F1 Sc
print('Testing F1 Scores: ', f1_score(test_target, test_preds, average=None))
print('Testing Confusion Matrix: \n', test_cmn)
sns.heatmap(test_cmn, annot=True).set(xlabel='Predicted', ylabel='Actual')
plt.show()
```

```
Training Score:  0.6078131805977233 , Training RMSE:  0.8495762339353616 , Training
F1 Score:  0.6065342582900316
Validation Score:  0.604996781932011 , validation RMSE:  0.8540774248367089 , Valida
tion F1 Score:  0.6038973244471664
Training F1 Scores:  [0.68656764 0.54823564 0.5847995 ]
Validation F1 Scores:  [0.68628244 0.54663964 0.57876989]
Training Confusion Matrix:
 [[0.69688776 0.12542576 0.17768649]
 [0.18135365 0.51390252 0.30474383]
 [0.1519059  0.23545706 0.61263704]]
Validation Confusion Matrix:
 [[0.69573371 0.12365213 0.18061416]
 [0.17721075 0.51290455 0.30988471]
 [0.15383717 0.23969162 0.60647121]]
Testing Score:  0.676383632923368 , Testing RMSE:  0.9267457126386072 , Testing F1 S
core:  0.43508193100715914
Testing F1 Scores:  [0.80056032 0.05296311 0.45172237]
Testing Confusion Matrix:
 [[0.69442692 0.12686061 0.17871247]
 [0.34533898 0.21822034 0.43644068]
 [0.20065742 0.17150207 0.6278405 ]]
```



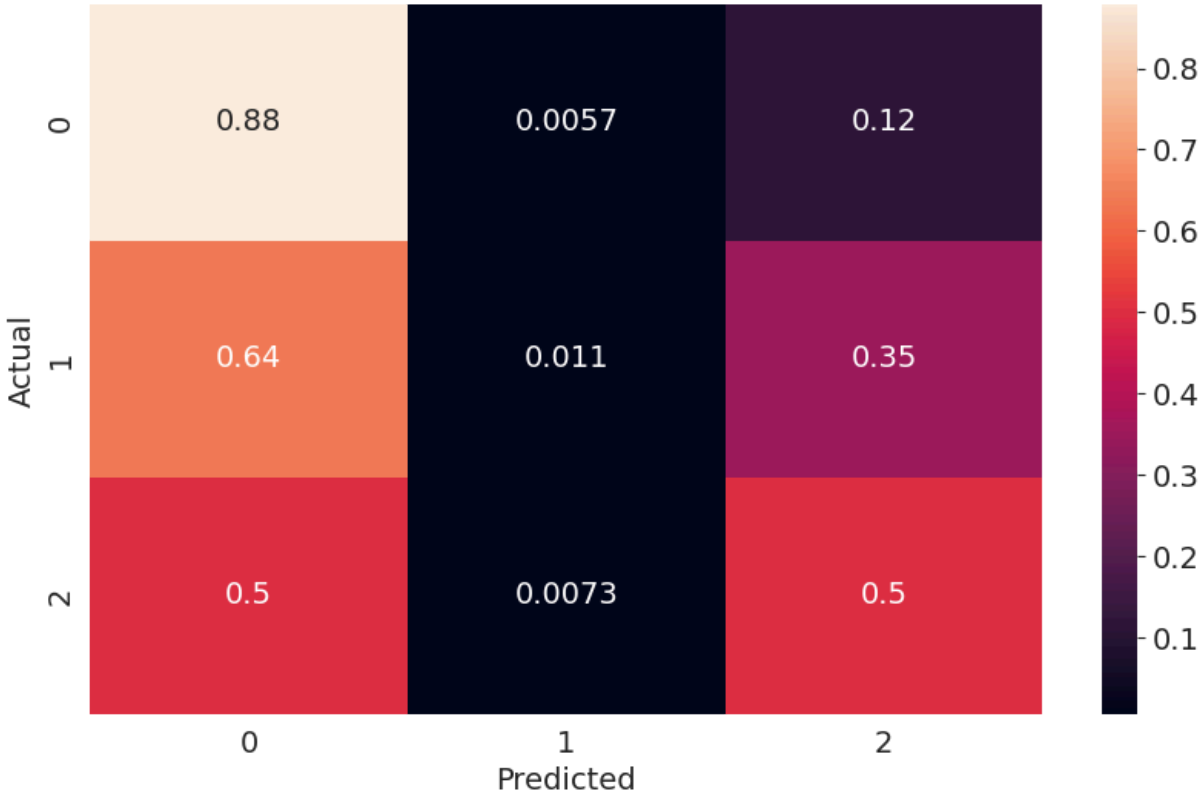
XGBoost

In [268…

```
xgb_model = xgb.XGBClassifier(random_state=42, use_label_encoder=False,
                              eval_metric='mlogloss',
                              subsample= 0.7, reg_lambda= 0.1,
                              reg_alpha= 1, n_estimators= 60,
                              min_child_weight= 0.5, max_depth= 6,
                              learning_rate= 0.5, gamma= 0.05,
                              colsample_bytree= 0.2)
xgb_model.fit(train_inputs, train_target)

xgb_model.fit(train_inputs, train_target)
train_preds = xgb_model.predict(train_inputs)
val_preds = xgb_model.predict(val_inputs)
trainingScore = xgb_model.score(train_inputs, train_target)
validationScore = accuracy_score(val_preds, val_target)
train_rmse = rmse(train_target, train_preds)
val_rmse = rmse(val_target, val_preds)
train_f1 = f1_score(train_target, train_preds, average='macro')
val_f1 = f1_score(val_target, val_preds, average='macro')
train_cm = confusion_matrix(train_target, train_preds, normalize='true')
val_cm = confusion_matrix(val_target, val_preds, normalize='true')
val_cmn = val_cm.astype('float') / val_cm.sum(axis=1)[:, np.newaxis]
train_cmn = train_cm.astype('float') / train_cm.sum(axis=1)[:, np.newaxis]
print('Training Score: ', trainingScore, ', Training RMSE: ', train_rmse, ', Traini
print('Validation Score: ', validationScore, ', validation RMSE: ', val_rmse, ', Va
print('Training F1 Scores: ', f1_score(train_target, train_preds, average=None))
print('Validation F1 Scores: ', f1_score(val_target, val_preds, average=None))
print('Training Confusion Matrix: \n', train_cmn)
print('Validation Confusion Matrix: \n', val_cmn)
```

```
test_preds = xgb_model.predict(test_inputs)
testScore = xgb_model.score(test_inputs, test_target)
test_rmse = rmse(test_target, test_preds)
test_f1 = f1_score(test_target, test_preds, average='macro')
test_cm = confusion_matrix(test_target, test_preds, normalize='true')
test_cmn = test_cm.astype('float') / test_cm.sum(axis=1)[:, np.newaxis]
print('Testing Score: ', testScore, ', Testing RMSE: ', test_rmse, ', Testing F1 Sc
print('Testing F1 Scores: ', f1_score(test_target, test_preds, average=None))
print('Testing Confusion Matrix: \n', test_cmn)
sns.heatmap(test_cmn, annot=True).set(xlabel='Predicted', ylabel='Actual')
plt.show()
```

```
Training Score:  0.7355320499901398 , Training RMSE:  0.7264852701906571 , Training
F1 Score:  0.7330182156388986
Validation Score:  0.7338755290308739 , validation RMSE:  0.7286578149586365 , Valid
ation F1 Score:  0.731523205850249
Training F1 Scores:  [0.83345774 0.72256041 0.64303651]
Validation F1 Scores:  [0.83299598 0.72091813 0.6406555 ]
Training Confusion Matrix:
 [[0.88186085 0.00554458 0.11259458]
 [0.08362967 0.69026336 0.22610697]
 [0.15072894 0.214844   0.63442706]]
Validation Confusion Matrix:
 [[0.88027426 0.00521566 0.11451008]
 [0.08240183 0.68929596 0.22830222]
 [0.1502161  0.21732274 0.63246116]]
Testing Score:  0.8102333648691264 , Testing RMSE:  0.8285617095092179 , Testing F1
Score:  0.44952574917010324
Testing F1 Scores:  [0.89055121 0.01603849 0.44198755]
Testing Confusion Matrix:
 [[0.87907466 0.00565487 0.11527048]
 [0.63983051 0.01059322 0.34957627]
 [0.49564099 0.00728884 0.49707017]]
```



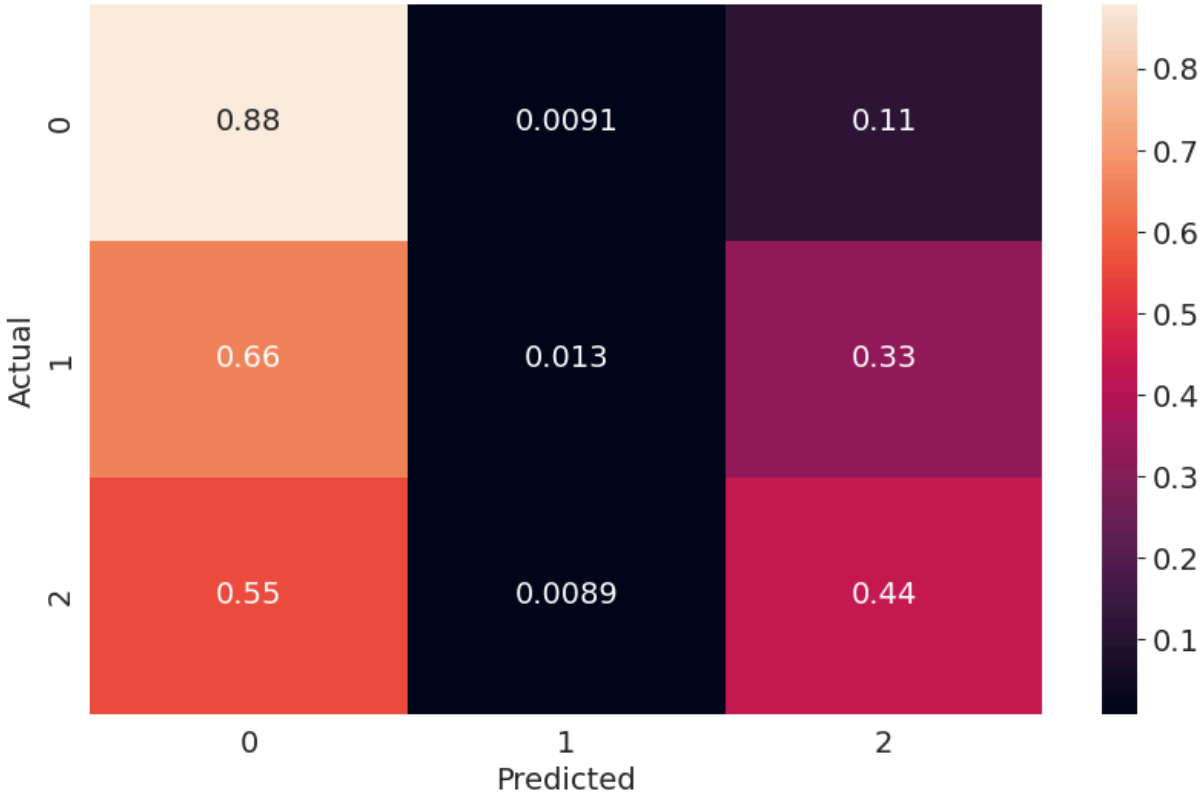
Ensemble Models

In [269…

```
base_models = [
    ('xgb', xgb_model),
    ('rf', rf_model),
]

stack_model = StackingClassifier(
    estimators=base_models,
    final_estimator=LogisticRegression()
)

stack_model.fit(train_inputs, train_target)

train_preds = stack_model.predict(train_inputs)
val_preds = stack_model.predict(val_inputs)
trainingScore = stack_model.score(train_inputs, train_target)
validationScore = accuracy_score(val_preds, val_target)
train_rmse = rmse(train_target, train_preds)
val_rmse = rmse(val_target, val_preds)
train_f1 = f1_score(train_target, train_preds, average='macro')
val_f1 = f1_score(val_target, val_preds, average='macro')
train_cm = confusion_matrix(train_target, train_preds, normalize='true')
val_cm = confusion_matrix(val_target, val_preds, normalize='true')
val_cmn = val_cm.astype('float') / val_cm.sum(axis=1)[:, np.newaxis]
train_cmn = train_cm.astype('float') / train_cm.sum(axis=1)[:, np.newaxis]
print('Training Score: ', trainingScore, ', Training RMSE: ', train_rmse, ', Traini
print('Validation Score: ', validationScore, ', validation RMSE: ', val_rmse, ', Va
print('Training F1 Scores: ', f1_score(train_target, train_preds, average=None))
```

```
print('Validation F1 Scores: ', f1_score(val_target, val_preds, average=None))
print('Training Confusion Matrix: \n', train_cmn)
print('Validation Confusion Matrix: \n', val_cmn)


test_preds = stack_model.predict(test_inputs)
testScore = stack_model.score(test_inputs, test_target)
test_rmse = rmse(test_target, test_preds)
test_f1 = f1_score(test_target, test_preds, average='macro')
test_cm = confusion_matrix(test_target, test_preds, normalize='true')
test_cmn = test_cm.astype('float') / test_cm.sum(axis=1)[:, np.newaxis]
print('Testing Score: ', testScore, ', Testing RMSE: ', test_rmse, ', Testing F1 Sc
print('Testing F1 Scores: ', f1_score(test_target, test_preds, average=None))
print('Testing Confusion Matrix: \n', test_cmn)
sns.heatmap(test_cmn, annot=True).set(xlabel='Predicted', ylabel='Actual')
plt.show()
```

```
Training Score:  0.7425230414496881 , Training RMSE:  0.7117065078898258 , Training
F1 Score:  0.7407542588409601
Validation Score:  0.7396485479687165 , validation RMSE:  0.715503919237355 , Valida
tion F1 Score:  0.7379511107424775
Training F1 Scores:  [0.84838878 0.72941588 0.64445812]
Validation F1 Scores:  [0.8471862  0.72675665 0.63991049]
Training Confusion Matrix:
 [[0.88192585 0.00846312 0.10961103]
 [0.05774894 0.70542384 0.23682722]
 [0.13944702 0.22037767 0.64017531]]
Validation Confusion Matrix:
 [[0.88045007 0.00826301 0.11128692]
 [0.05729502 0.70427811 0.23842687]
 [0.14022895 0.22514893 0.63462212]]
Testing Score:  0.8018960895616525 , Testing RMSE:  0.8429942990350607 , Testing F1
Score:  0.4354432019178674
Testing F1 Scores:  [0.88599724 0.01705757 0.40327479]
Testing Confusion Matrix:
 [[0.87905129 0.00908985 0.11185886]
 [0.66101695 0.01271186 0.32627119]
 [0.55466629 0.00886094 0.43647277]]
```



Binary model of Diabetes = True or False

In [277…
```
def convert_binary(value):
    if value in [1, 2]:
        return 1
    else:
        return 0

train_val_target_binary = train_val_target.applymap(convert_binary)
test_target_binary = np.vectorize(convert_binary)(test_target)
```

In [278…
```
smote = SMOTE(random_state=42)
train_val_inputs_resampled_binary, train_val_target_resampled_binary = smote.fit_re
```

In [279…
```
train_inputs_binary, val_inputs_binary, train_target_binary, val_target_binary = tr
```

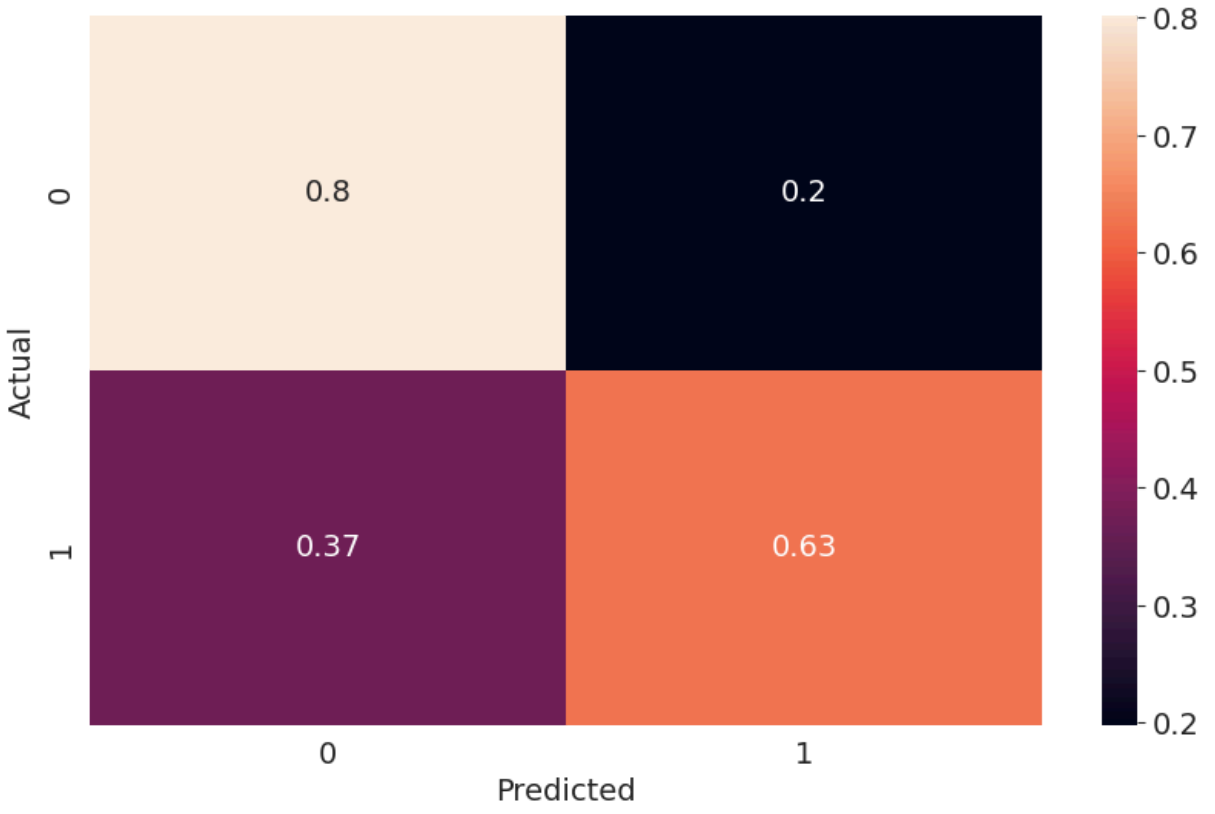
In [280…
```
rf_binary_model = RandomForestClassifier(random_state=42, n_jobs=-1,
                                         n_estimators=7, max_depth=12,
                                         min_samples_split= 5, min_samples_leaf= 4,
                                         max_leaf_nodes= 512, max_features= 'sqrt',
                                         bootstrap= False)

xgb_binary_model = xgb.XGBClassifier(random_state=42, use_label_encoder=False,
                                     eval_metric='mlogloss',
                                     subsample= 1.0, reg_lambda= 5,
                                     reg_alpha= 0.7, n_estimators= 10,
                                     min_child_weight= 7, max_depth= 6,
```

```
                                learning_rate= 0.7, gamma= 0,
                                colsample_bytree= 0.8)
```

In [289…

```
base_models = [
    ('xgb', xgb_binary_model),
    ('rf', rf_binary_model),
]

meta_classifier = LogisticRegression(random_state=42)

stacked_binary_model = StackingClassifier(
    estimators=base_models,
    final_estimator=meta_classifier,
    cv=5,
    n_jobs=-1
)

stacked_binary_model.fit(train_inputs_binary, train_target_binary)

train_preds = stacked_binary_model.predict(train_inputs_binary)
val_preds = stacked_binary_model.predict(val_inputs_binary)
trainingScore = stacked_binary_model.score(train_inputs_binary, train_target_binary
validationScore = accuracy_score(val_preds, val_target_binary.values.ravel())
train_rmse = rmse(train_target_binary.values.ravel(), train_preds)
val_rmse = rmse(val_target_binary.values.ravel(), val_preds)
train_f1 = f1_score(train_target_binary.values.ravel(), train_preds, average='macro
val_f1 = f1_score(val_target_binary.values.ravel(), val_preds, average='macro')
train_cm = confusion_matrix(train_target_binary.values.ravel(), train_preds, normal
val_cm = confusion_matrix(val_target_binary.values.ravel(), val_preds, normalize='t
val_cmn = val_cm.astype('float') / val_cm.sum(axis=1)[:, np.newaxis]
train_cmn = train_cm.astype('float') / train_cm.sum(axis=1)[:, np.newaxis]
print('Training Score: ', trainingScore, ', Training RMSE: ', train_rmse, ', Traini
print('Validation Score: ', validationScore, ', validation RMSE: ', val_rmse, ', Va
print('Training F1 Scores: ', f1_score(train_target_binary.values.ravel(), train_pr
print('Validation F1 Scores: ', f1_score(val_target_binary.values.ravel(), val_pred
print('Training Confusion Matrix: \n', train_cmn)
print('Validation Confusion Matrix: \n', val_cmn)

test_preds = stacked_binary_model.predict(test_inputs)
testScore = stacked_binary_model.score(test_inputs, test_target_binary)
test_rmse = rmse(test_target_binary, test_preds)
test_f1 = f1_score(test_target_binary, test_preds, average='macro')
test_cm = confusion_matrix(test_target_binary, test_preds, normalize='true')
test_cmn = test_cm.astype('float') / test_cm.sum(axis=1)[:, np.newaxis]
print('Testing Score: ', testScore, ', Testing RMSE: ', test_rmse, ', Testing F1 Sc
print('Testing F1 Scores: ', f1_score(test_target_binary, test_preds, average=None)
print('Testing Confusion Matrix: \n', test_cmn)

sns.heatmap(test_cmn, annot=True).set(xlabel='Predicted', ylabel='Actual')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:99: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected. Please ch
ange the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
Training Score:  0.8170780862973533 , Training RMSE:  0.427693714827149 , Training F
1 Score:  0.8170602102148183
Validation Score:  0.8158972558656603 , validation RMSE:  0.4290719568258216 , Valid
ation F1 Score:  0.815841830899631
Training F1 Scores:  [0.81525183 0.81886859]
Validation F1 Scores:  [0.812647   0.81903666]
Training Confusion Matrix:
 [[0.80656372 0.19343628]
 [0.17239113 0.82760887]]
Validation Confusion Matrix:
 [[0.80419539 0.19580461]
 [0.17256406 0.82743594]]
Testing Score:  0.774302270577105 , Testing RMSE:  0.4750765511187592 , Testing F1 S
core:  0.6607746032000347
Testing F1 Scores:  [0.85701799 0.46453121]
Testing Confusion Matrix:
 [[0.80191611 0.19808389]
 [0.37451203 0.62548797]]
```



In [ ]: