Understanding Diabetes Risk

Group 4
Craig Adlam, Kulaphong Jitareerat, Nijiati Abulizi

Introduction

Diabetes is a growing health concern globally, affecting millions each year. Identifying who might be at risk can help in early intervention and potentially prevent the disease from developing. This report uses various computer models to analyze health data from over 250,000 individuals, aiming to uncover what factors might increase someone's risk of developing diabetes.

Our Guiding Questions

To understand diabetes risk, we focused on various personal details (like age and income) and health indicators (like blood pressure and body weight, referred to here as BMI). Our goal was to answer a few key questions:

- Can we use personal and health details to predict someone's risk of diabetes?
- What factors suggest someone might move from being at risk to developing diabetes?
- How can these insights help in making broad health policy decisions?

Results

1. Discovering Connections

Previously, through a method called "correlation analysis," we found certain health indicators, like high blood pressure and BMI, are linked with higher diabetes risk. Interestingly, people who perceive their health as poor also tend to have these health issues.

Our analysis revealed key insights into the factors associated with diabetes risk (Figure 1). We found moderate positive correlations between diabetes and indicators like high blood pressure, high cholesterol, body mass index (BMI), general health perception, and difficulty walking. Notably, a strong link exists between one's general health perception and their physical and mental health, indicating that those who perceive their health as poor often experience broader health and mobility issues. Furthermore, general health perception correlates with other critical health indicators, emphasizing the interconnectedness of health variables.

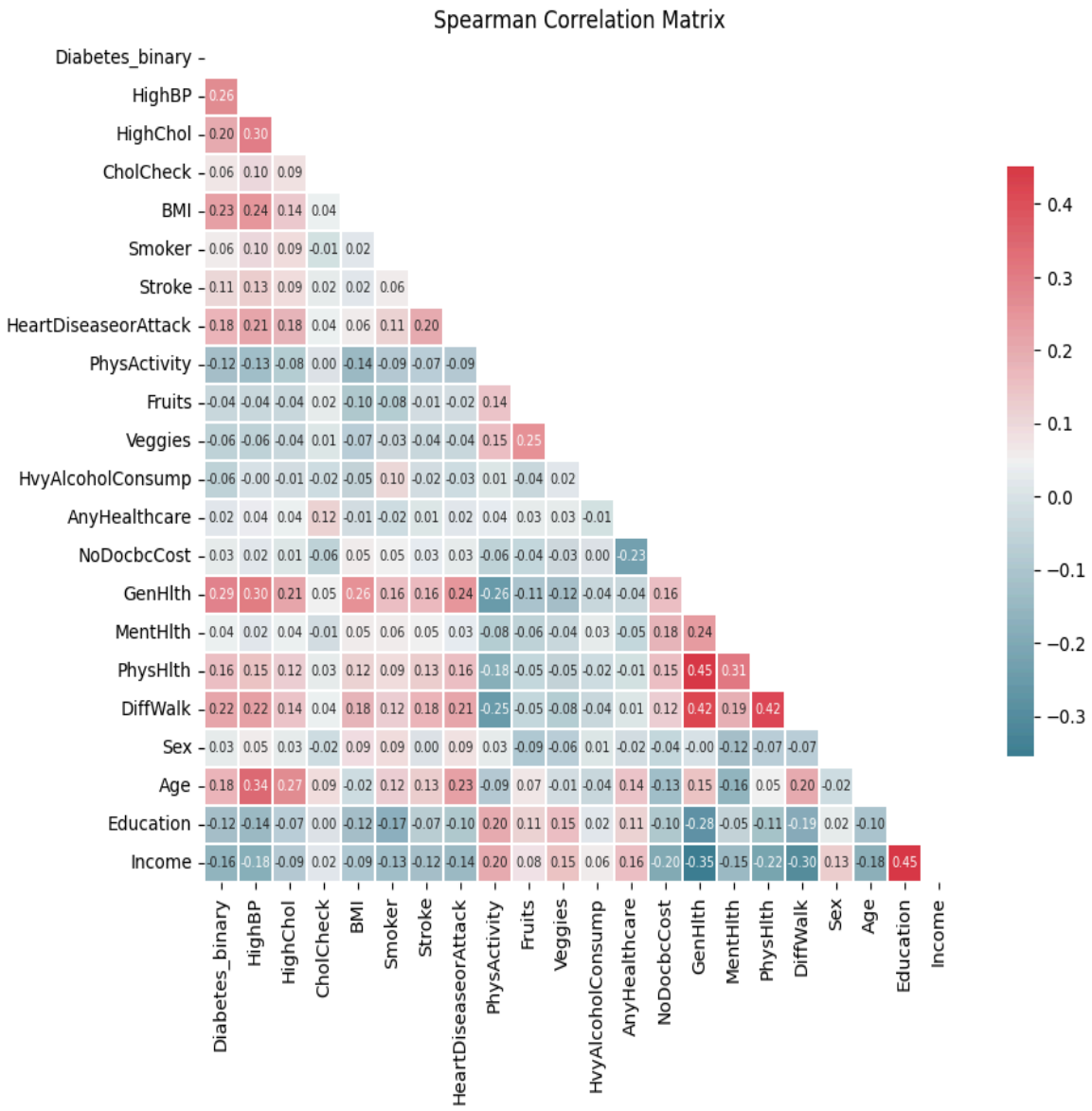


Figure 1 Spearman Correlation Matrix of Variables. This figure presents a correlation matrix utilizing Spearman's rank correlation method to assess the strength and direction of association between variables in the dataset. The matrix visually represents correlations, highlighting relationships ranging from strong positive to strong negative associations, thus providing insights into how variables move in relation to one another in a ranked order.

Interestingly, income shows a strong positive correlation with education, suggesting socioeconomic factors play a significant role in health outcomes. On the flip side, lifestyle factors such as physical activity, fruit and vegetable consumption, and higher education and income levels generally correlate negatively with poor health indicators. This suggests that healthier lifestyle choices and higher socioeconomic status are associated with lower diabetes risk and better overall health perception.

2. Predicting Diabetes Using Personal and Health Details

In this section, we employed three different strategies—similar to medical tests—to gauge the risk of diabetes based on various health and lifestyle factors. These strategies are called Logistic Regression, Decision Tree, and Random Forest. Each one analyzed information differently but aimed to predict who might be more likely to develop diabetes. To see how well each strategy worked, we used a scoring method known as the F1-Score. This score helped us understand how accurately and consistently each strategy could identify individuals at risk for diabetes, based on a combination of health details like weight, age, blood pressure, and lifestyle habits. These models demonstrated varying degrees of accuracy and interpretability. Table 1 summarizes their performance:

Table 1 Model Performance Comparison using F1-Score (Macro) of Testing Dataset

Model	F1-Score (macro)	
	Train Score	Test Score
Random Forest	0.706	0.681
Logistic Regression	0.674	0.673
Decision Tree	0.699	0.671

The Random Forest model, with the highest F1-Score, emerged as the most accurate in our study. It effectively captured a wide array of factors influencing diabetes risk, including traditional health indicators and socio-economic factors. Key insights include the significance of difficulty walking and a history of heart disease or attack, alongside other well-established risk factors such as high blood pressure, high cholesterol, and BMI. These findings in Figure 2 suggest a broad spectrum of considerations that impact diabetes risk, underlining the complexity of disease prediction.

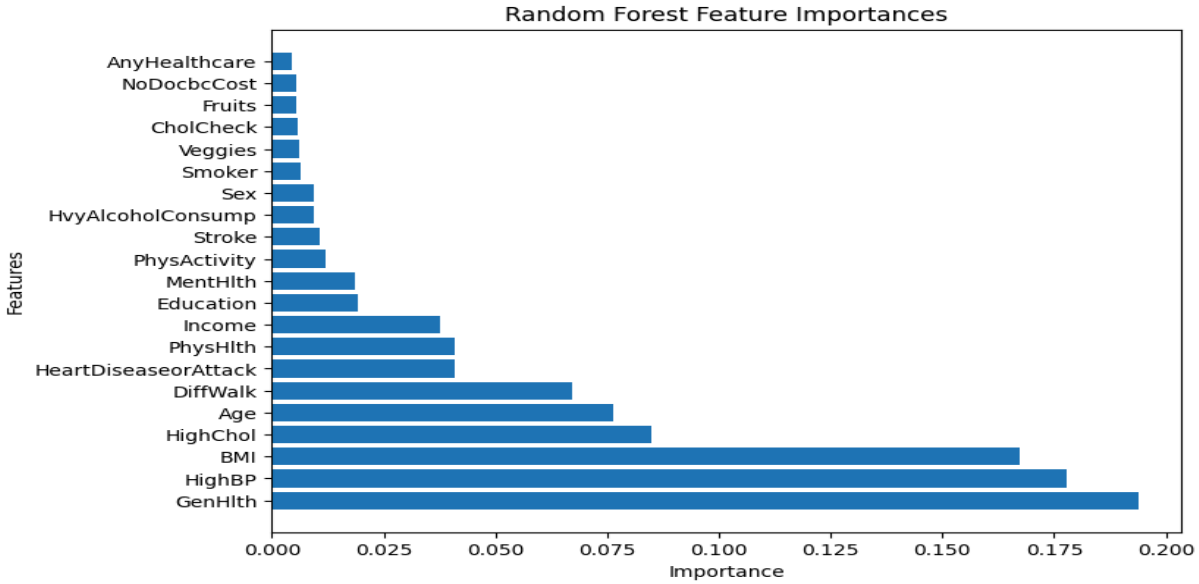


Figure 2 Feature importance horizontal bar chart for the Random Forest model, highlighting the importance of additional characteristics in predicting diabetes: difficulty walking and history of heart disease or attack.

The Random Forest method, which combines insights from many smaller analyses (like piecing together a large puzzle from smaller ones), proved to be very effective, though its complexity makes it a bit like reading a complicated medical chart—it's accurate but can be hard to understand at a glance.

On the other hand, the Logistic Regression and Decision Tree methods, while a bit simpler and less detailed, still provided important clues. Logistic Regression helped us see directly how factors like overall health, body weight in relation to height (BMI), age, blood pressure, and cholesterol levels play into diabetes risk. Meanwhile, the Decision Tree gave us insights into how income and how one perceives their health could affect their diabetes risk, suggesting that financial well-being might also play a role in having access to healthier lifestyle choices and healthcare.

For a comprehensive view of the contributing factors according to the Logistic Regression and Decision Tree models, refer to the supplementary figures in the appendix (Appendix A1 and A2, respectively). These visual representations provide an accessible overview of feature importance, highlighting the varying significance of different diabetes risk factors as identified by each model.

Specifically, through logistic regression, we identified major diabetes risk factors: overall health, body mass index (BMI), age, high blood pressure, and high cholesterol (Appendix A2). Conversely, smoking, healthcare access, and financial barriers to healthcare showed a weaker link to diabetes risk, as indicated by their statistical insignificance (Appendix A3). The Decision Tree model (Appendix A1) highlighted Income as the sixth most important feature, while the Logistic Regression model placed less emphasis on this factor (Appendix A2). This suggests that income, likely representing socioeconomic status, might have a greater impact on diabetes risk through its influence on access to healthcare, healthy food choices, and the ability to maintain a healthy lifestyle.

Interestingly, individuals with heart disease history or mobility issues had a somewhat lower diabetes risk in our analysis, possibly due to increased medical surveillance and management of diabetes-related risk factors.

Key Insights:

- Health Conditions: Poor general health, high blood pressure, and high cholesterol strongly hint at elevated diabetes risk.
- Lifestyle Factors: Higher BMI is a clear risk marker, whereas physical activity serves as a protective factor.
- Demographics: Age and gender play significant roles in diabetes likelihood.

3. Diabetes Risks Patterns: How Advanced Tools Can Predict Health Trends

From our initial investigations using Logistic Regression, Random Forest and Decision Tree, we discovered certain trends that could help identify who's at risk for diabetes. In those analyses, we assumed the data as continuous numbers and treated classes as balanced. To make our findings even more robust, we decided to use another method called CatBoost. Think of CatBoost as a sophisticated tool that's especially good at sifting through various types of information, including simple yes/no answers, to uncover hidden patterns or clarify some of the earlier, puzzling results.

After considering class balance, CatBoost (Figure 3) showed us that factors like overall health, ages, their body mass index (BMI), blood pressure, and cholesterol levels play crucial roles in determining the risk of diabetes. Interestingly, it also revealed that eating fruits and vegetables, while generally good for health, doesn't influence diabetes risk as much as these other factors. This insight helps us focus on what really matters when assessing the risk of diabetes.

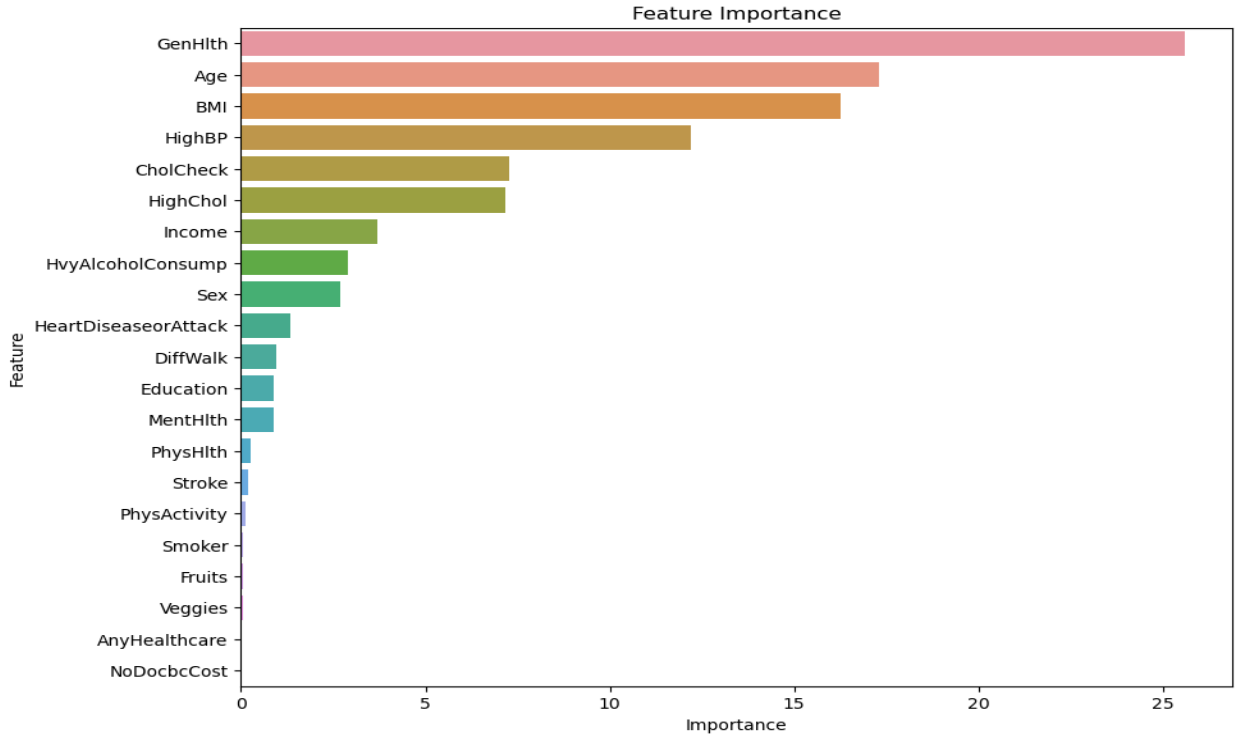


Figure 3 Feature importance bar chart for the CatBoost model, highlighting the dominance of health status, age, and BMI in predicting diabetes.

Following our exploration of feature importance with CatBoost above, we dove deeper into understanding how each factor influences diabetes risk using a SHAP summary plot. This plot, shown as Figure 4, offers a detailed look at the contribution of each factor to the model's predictions. It highlights that general health, body mass index (BMI), age, cholesterol levels, and blood pressure are significant predictors of diabetes risk. Essentially, the higher these values are, the more likely someone is to be at risk for diabetes.

Conversely, the SHAP (SHapley Additive exPlanations) summary also points out factors that could help lower the risk of diabetes. Being physically active and having a diet rich in fruits and vegetables are shown to be beneficial. This analysis provides a clear view of what actions might protect against diabetes, emphasizing the importance of a healthy lifestyle.

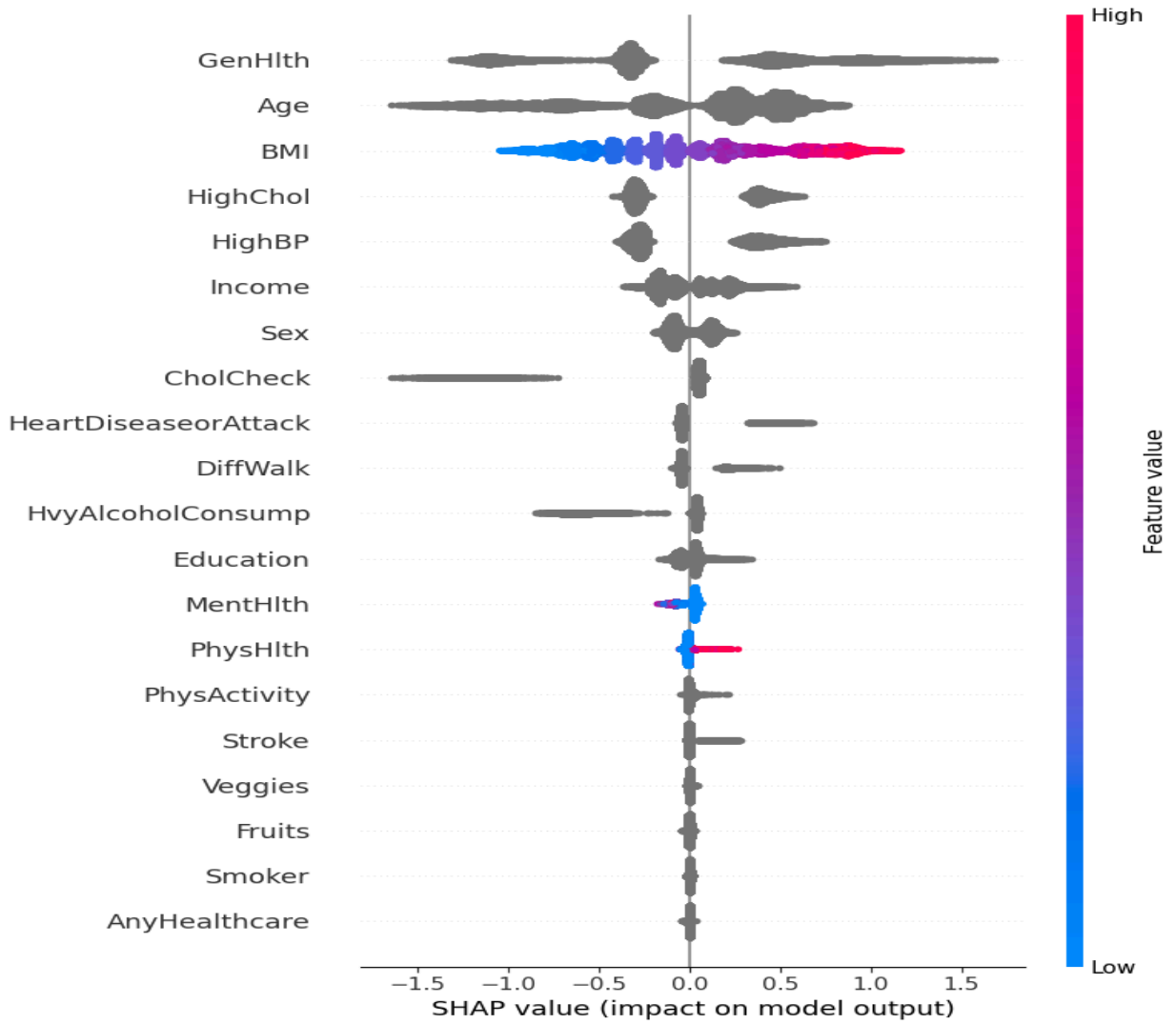


Figure 4 SHAP (SHapley Additive exPlanations) summary plot displaying feature impacts on diabetes risk prediction. Key indicators: general health, BMI, age, high cholesterol and high blood pressure.

We also conducted an XGBoost model and the results largely support what we obtained from the CatBoost model (Appendix Figure A5). This understanding can be useful for someone looking to reduce their diabetes risk. For example, focusing on improving overall health, managing weight, and increasing physical activity can be effective strategies.

To understand the error rates of our model, we looked into the performance of the models.The CatBoost model depicted in the figure 5 demonstrates a capacity to predict non-diabetic cases (true negatives) with a high degree of accuracy. However, its ability to correctly identify diabetic cases (true positives) is less pronounced, which is evidenced by the number of false negatives and a Precision-Recall Curve AUC of 0.43. Despite the model being trained with a balanced

approach to account for the highly imbalanced dataset, there remains a challenge in improving the identification of true positives – crucial for a reliable diabetes prediction tool. The significant number of false positives also suggests that while the model has learned to some extent to cope with class imbalance, there is room for improvement in specificity. Overall, while the model shows potential, these results indicate a need for further refinement to enhance its predictive performance, particularly in correctly identifying diabetic cases in an imbalanced dataset.

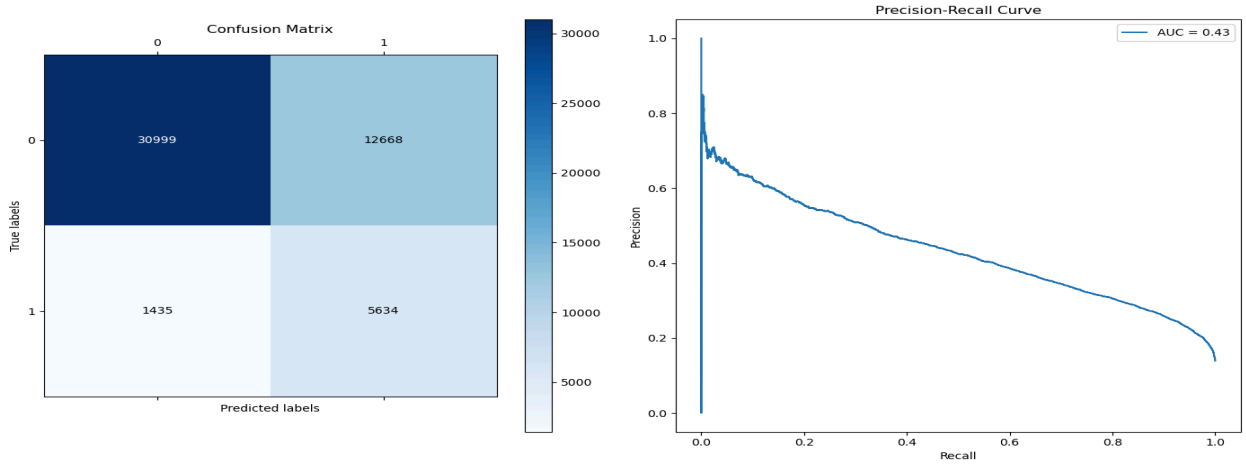


Figure 5 Performance Evaluation of CatBoost model. Left panel shows the Confusion Matrix, depicting actual versus predicted classifications. Right panel presents the Precision-Recall Curve with an area under the curve (AUC) of 0.43, indicating the trade-off between precision and recall for different thresholds.

To refine our understanding due to the data's complexity and the initial model's moderate accuracy, we turned to a more advanced technique: a neural network analysis. However, this approach led to a similar test score of 62% (Table 2).The performance of this model shown in Figure 6 is worse than CatBoost after considering the different class weights. In order to improve the training with advanced models like this, we might need more data to increase the accuracy.

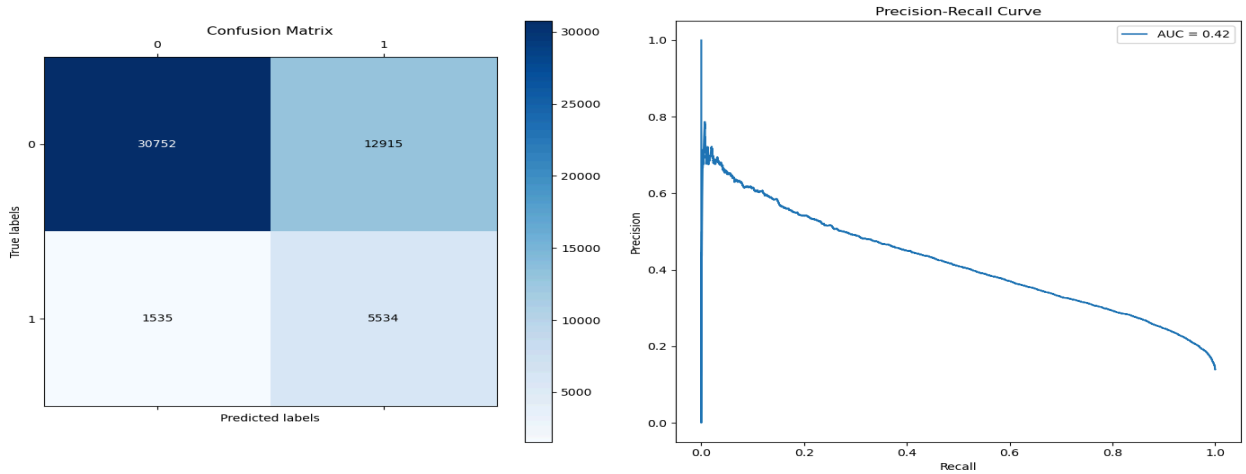


Figure 6 Performance Evaluation of neural network model. Left panel shows the Confusion Matrix, depicting actual versus predicted classifications. Right panel presents the Precision-Recall Curve with an area under the curve (AUC) of 0.42, indicating the trade-off between precision and recall for different thresholds.

Table 2 Model Performance Comparison using F1-Score (Macro) of Testing Dataset

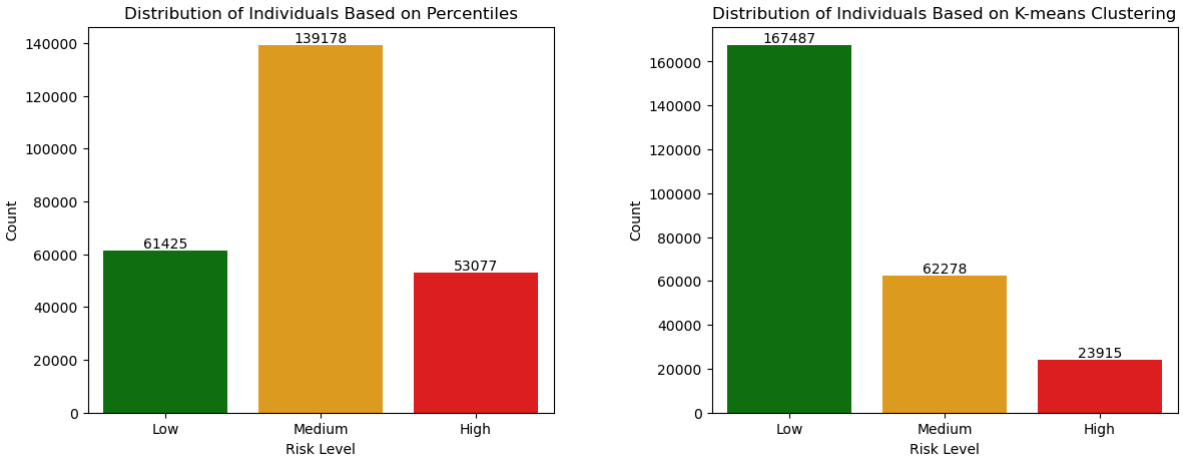
Model	F1-Score (macro)	
	Train Score	Test Score
CatBoost	0.629	0.629
Neural Network	0.631	0.621
Boosting Tree	0.669	0.636

After making sure our models take into account that some types of outcomes are rarer than others, all the models we looked at performed about the same when it comes to their ability to balance precision and recall across different classes, as shown in Table 2. Even though we adjusted the settings for the boosting tree model to try and get the best performance, we haven't done the same for the CatBoost and Neural Network models yet. This means there's a good chance we can make these models work even better by fine-tuning how they're set up.

4. Informing Policy Decisions by Determining Risk Levels

Lastly, we looked at how individual risk factors cluster in the population and what that means for health policy. For example, are there common characteristics among those at higher risk that could guide where to focus health resources?

The logistic regression model showed that certain characteristics significantly affected the probability of having diabetes as discussed in Section 2. Thresholds were established using these predicted probabilities to categorize individuals into low-, medium-, and high-risk groups based on the approximate prevalence of diabetes in the general public (ranging from 11% up to 29%, the middle 20% mark was used)[1], which is displayed in the left bar chart in Figure 7. It is important to note that using the first three models - logistic regression, decision tree, and random forest - more accurately captured high risk individuals, whereas the latter three models - CatBoost, Neural Networks, and Boosting Tree - incorrectly labeled more high risk individuals (increased false positives) and is one reason why logistic regression probabilities are used.



[1] https://www.cdc.gov/diabetes/data/statistics-report/index.html

Figure 7 Bar charts displaying the number of individuals grouped into each risk level based on their predicted probabilities of diabetes using percentiles (left) and K-means clustering (right). Lower/upper thresholds: left (0.03~24th percentile, 0.23~79th percentile); right (0.14~65th percentile, 0.36~90th percentile).

K-means clustering is an alternative method used in machine learning to group similar data points together and determine the thresholds by randomly initializing three centers, and assigning the closest points to these groups in an iterative process until the lowest amount of variability within each group is found. In this case, there are similar characteristics in the low-risk group (e.g., healthier diet, more physical activity, better life choices) that lower the probability of diabetes and increase the lower threshold to capture more individuals as shown in the right bar chart in Figure 7. This method suggests that while certain characteristics remain critical across different risk levels, most people are classified as low risk of diabetes.

As shown in Figure 8, the characteristics that are most representative of the high risk group include increased cholesterol checks, higher blood pressure, higher cholesterol, poorer perceived general health, higher age groups, and higher BMIs, respectively. It is important to note that as an individual rates their general health from 1 to 5 (i.e., as the scale increases from 1 to 5, perception of health decreases), the probability of that individual having diabetes also increases. For example, with each increase of 1 rating on the general health scale, the probability of having diabetes increases by approximately 63%.

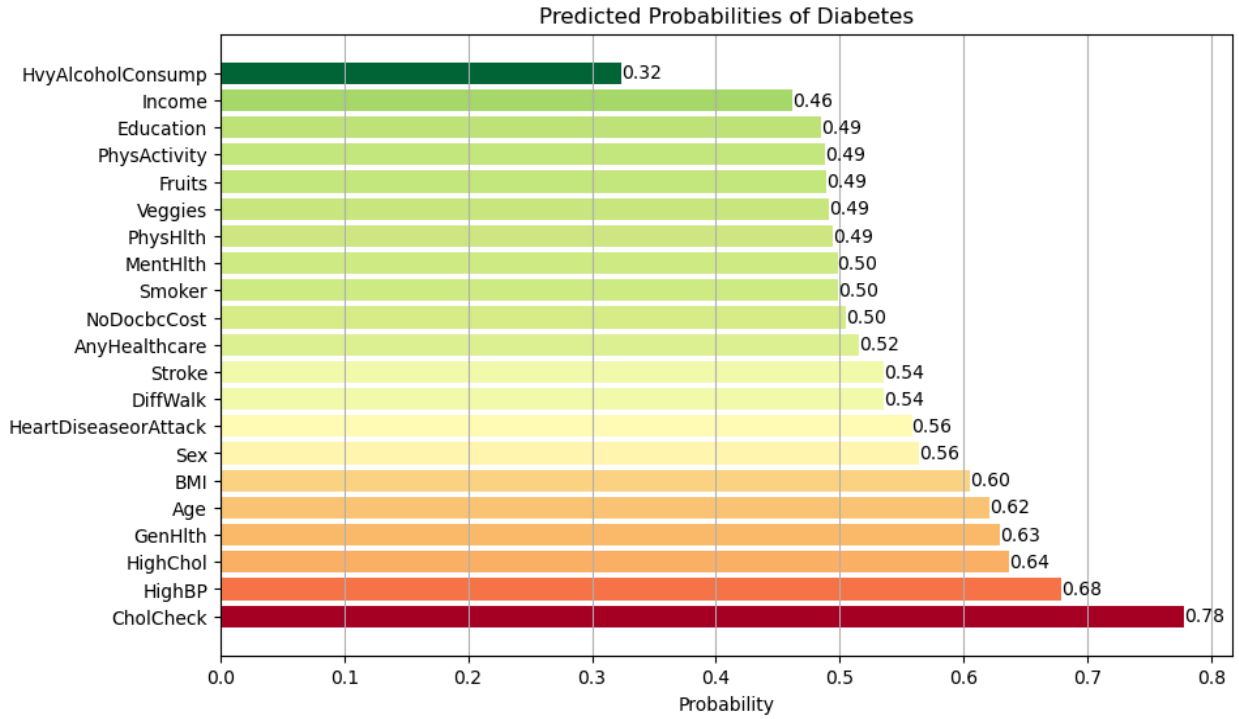


Figure 8 Horizontal bar chart displaying the predicted probabilities from the logistic regression model after converting the log-odds of each predictor variable. Predictors in the orange/red area are considered higher risk characteristics of diabetes and predictors in the green area are considered lower risk characteristics of diabetes.

Conversely, characteristics such as heavy alcohol consumption, higher income and education levels, higher amounts of physical activity, healthier diets containing fruits and vegetables, and

perceived positive physical and mental health are characteristics that represent the low risk group. It is interesting to note that the association between heavy alcohol consumption and diabetes suggests that consuming more than 7 drinks per week for women and 14 drinks per week for men is associated with a lowered risk of diabetes. This could be due to various factors, such as lifestyle choices, metabolic differences, or other confounding variables that are unexplained and not included in this dataset. Additionally, the amount of individuals with diabetes in this dataset is largely imbalanced and heavy alcohol consumption was better represented in the predictive models - CatBoost, Neural Networks, and Boosting Tree - that accounted for this imbalance during the hypertuning stage as discussed in Section 3. As a result, there would be more individuals labeled as high risk, however, the majority of these would be incorrectly predicted (Table A1 Confusion Matrices in Appendix A).

These insights could guide policymaking at various levels. Investing in preventive measures like routine health screenings, affordable healthy food options, and community health facilities might benefit a broader segment of the population (low to medium risk individuals). On the other hand, targeted campaigns with specialized programs such as increased home health nursing, support groups, and educational programs could be implemented to minimize the detrimental effects for those diagnosed with diabetes or are at high risk.

In essence, the findings underscore the potential of using predictive modeling to stratify the population by diabetes risk and implement public health strategies, accordingly, that maximizes the impact of interventions and resource allocation.

Key Insights for Decision makers:

Health Factors: High blood pressure, being overweight, and poor general health perception are significant flags for diabetes risk. Keeping an eye on these can help in early detection and management.

Lifestyle Matters: Engaging in physical activity and having a healthy diet (e.g., fruits and veggies) seem to lower diabetes risk.

Socioeconomic Influences: Higher income and better education are associated with lower diabetes risk. This might be due to better access to healthcare and healthier lifestyle options.

Recommendations Based on Our Analysis

For Individuals:

- Regular health checks focusing on blood pressure, cholesterol, and BMI can help catch early signs of diabetes risk.
- A lifestyle incorporating physical activity and a balanced diet can be powerful in managing or even preventing diabetes.

For Policymakers:

- Focus on communities with lower income and education, as they are at higher risk.
- Public health campaigns promoting physical activity and healthier eating can benefit the entire population, but especially those at higher risk of diabetes.

Conclusion

The analysis of data from hundreds of thousands of individuals highlights key factors that can be used not only to predict diabetes, but also assess the level of risk. Understanding these findings can help everyone from individuals to policymakers that need to make informed decisions to combat the growing diabetes challenge. Whether a healthcare professional, a policymaker, or just someone interested in staying healthy, this report offers insights into how humanity can work towards a healthier future in a proactive way.

Limitations of the analysis

One significant challenge we encountered across all our models pertained to imbalanced classes, significantly impacting model accuracy and their predictions. Addressing this issue in future studies is important, necessitating the inclusion of additional data representing positive diabetes cases. This addition would enable more accurate training of machine learning algorithms.

Appendix A

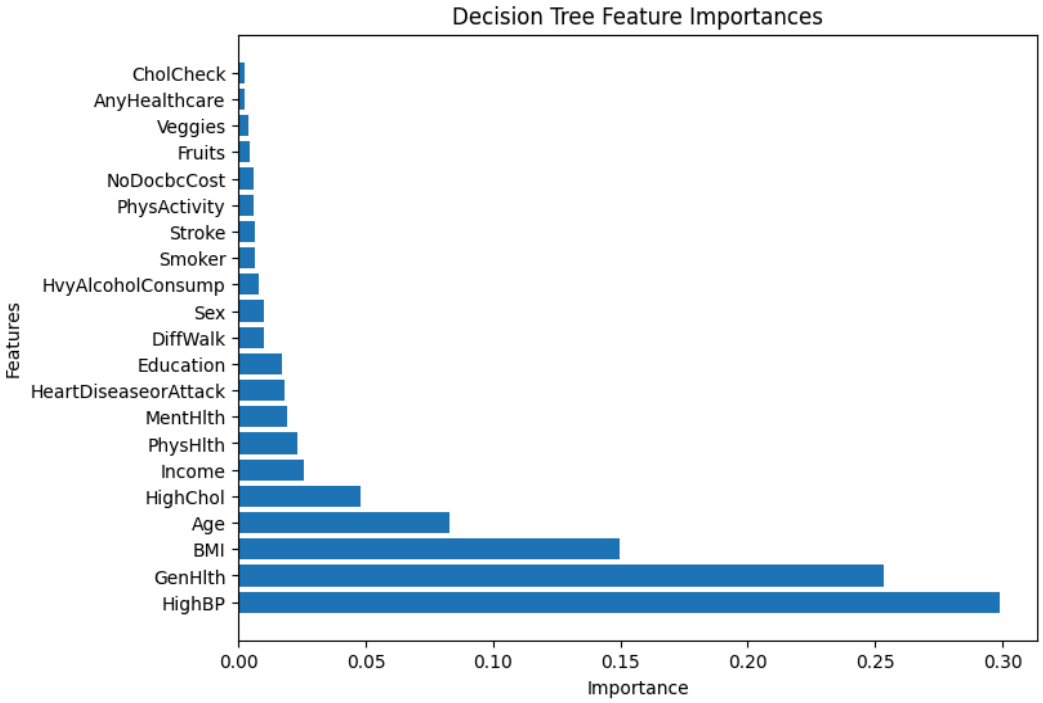


Figure A1 Feature importance horizontal bar chart for the Decision Tree model, highlighting the importance of additional characteristics in predicting diabetes: income, perceived physical and mental health.

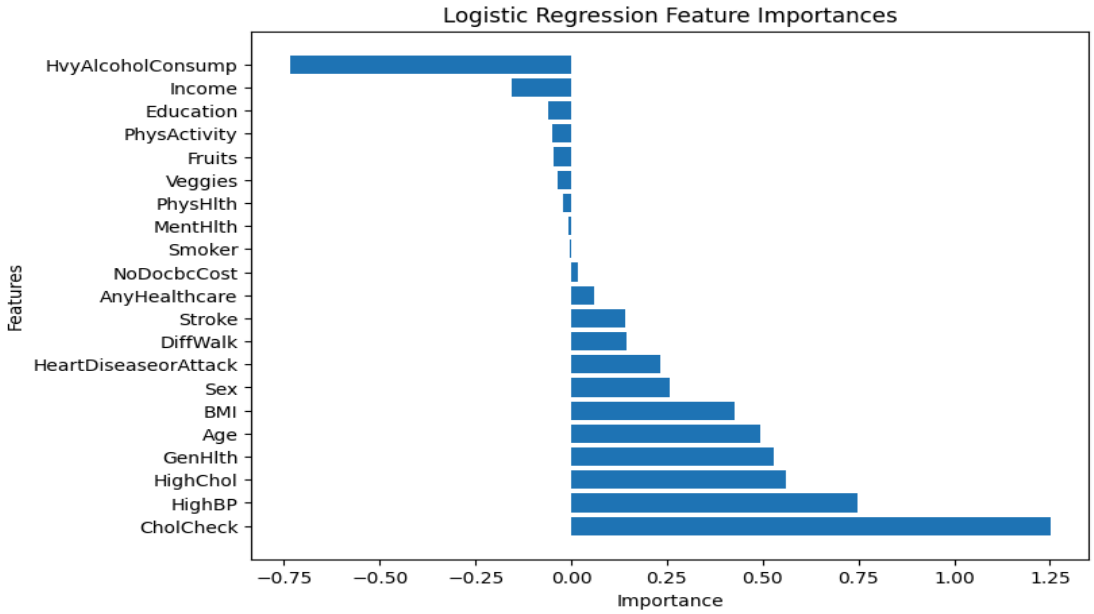


Figure A2 Feature importance horizontal bar chart for the Logistic Regression model, highlighting the dominance of cholesterol checks, high blood pressure and cholesterol , general health, age, and BMI in predicting diabetes.

Logit Regression Results

Dep. Variable:	Diabetes_binary	No. Observations:	202944
Model:	Logit	Df Residuals:	202922
Method:	MLE	Df Model:	21
Date:	Thu, 21 Mar 2024	Pseudo R-squ.:	0.2069
Time:	13:40:45	Log-Likelihood:	-65086.
converged:	True	LL-Null:	-82070.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-6.5022	0.095	-68.352	0.000	-6.689	-6.316
HighBP	0.7491	0.016	45.544	0.000	0.717	0.781
HighChol	0.5615	0.015	37.068	0.000	0.532	0.591
CholCheck	1.2547	0.077	16.385	0.000	1.105	1.405
BMI	5.2330	0.086	60.568	0.000	5.064	5.402
Smoker	-0.0036	0.015	-0.244	0.807	-0.032	0.025
Stroke	0.1419	0.028	5.070	0.000	0.087	0.197
HeartDiseaseorAttack	0.2340	0.020	11.790	0.000	0.195	0.273
PhysActivity	-0.0493	0.016	-3.057	0.002	-0.081	-0.018
Fruits	-0.0446	0.015	-2.914	0.004	-0.075	-0.015
Veggies	-0.0364	0.018	-2.050	0.040	-0.071	-0.002
HvyAlcoholConsump	-0.7360	0.042	-17.329	0.000	-0.819	-0.653
AnyHealthcare	0.0611	0.037	1.641	0.101	-0.012	0.134
NoDocbcCost	0.0181	0.026	0.704	0.482	-0.032	0.068
GenHlth	2.1211	0.036	58.473	0.000	2.050	2.192
MentHlth	-0.1088	0.029	-3.814	0.000	-0.165	-0.053
PhysHlth	-0.2239	0.026	-8.532	0.000	-0.275	-0.172
DiffWalk	0.1441	0.019	7.611	0.000	0.107	0.181
Sex	0.2583	0.015	17.199	0.000	0.229	0.288
Age	1.4788	0.037	39.498	0.000	1.405	1.552
Education	-0.1479	0.039	-3.804	0.000	-0.224	-0.072
Income	-0.3595	0.028	-12.899	0.000	-0.414	-0.305

Figure A3 Logistic Regression results from library statsmodels.

Model	Train Score	Test Score	Params
Random Forest	0.705714	0.680745	{'max_depth': 11, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
Logistic Regression	0.673793	0.672751	{'C': 0.01, 'penalty': None}
Decision Tree	0.699446	0.670886	{'criterion': 'gini', 'max_depth': 11, 'min_samples_leaf': 2, 'min_samples_split': 10}

Figure A4 All model's performance and the best parameters from GridSearch with 5 folds Cross Validation.

Table A1 Confusion Matrix of Each Model (Testing Dataset)

			Prediction	
			Non-diabetes	Diabetes
Target	Logistic Regression	Non-diabetes	37,901	5,838
		Diabetes	3,244	3,753
	Decision Tree	Non-diabetes	38,278	5,461
		Diabetes	3,424	3,573
	Random Forest	Non-diabetes	38,922	4,817
		Diabetes	3,489	3,508

			Prediction	
			Non-diabetes	Diabetes
Target	CatBoost	Non-diabetes	30,999	12,668
		Diabetes	1,435	5,634
	Neural Net	Non-diabetes	30,752	12,915
		Diabetes	1,535	5,534
	Boosting Tree	Non-diabetes	31,882	11,657
		Diabetes	1,790	5,407

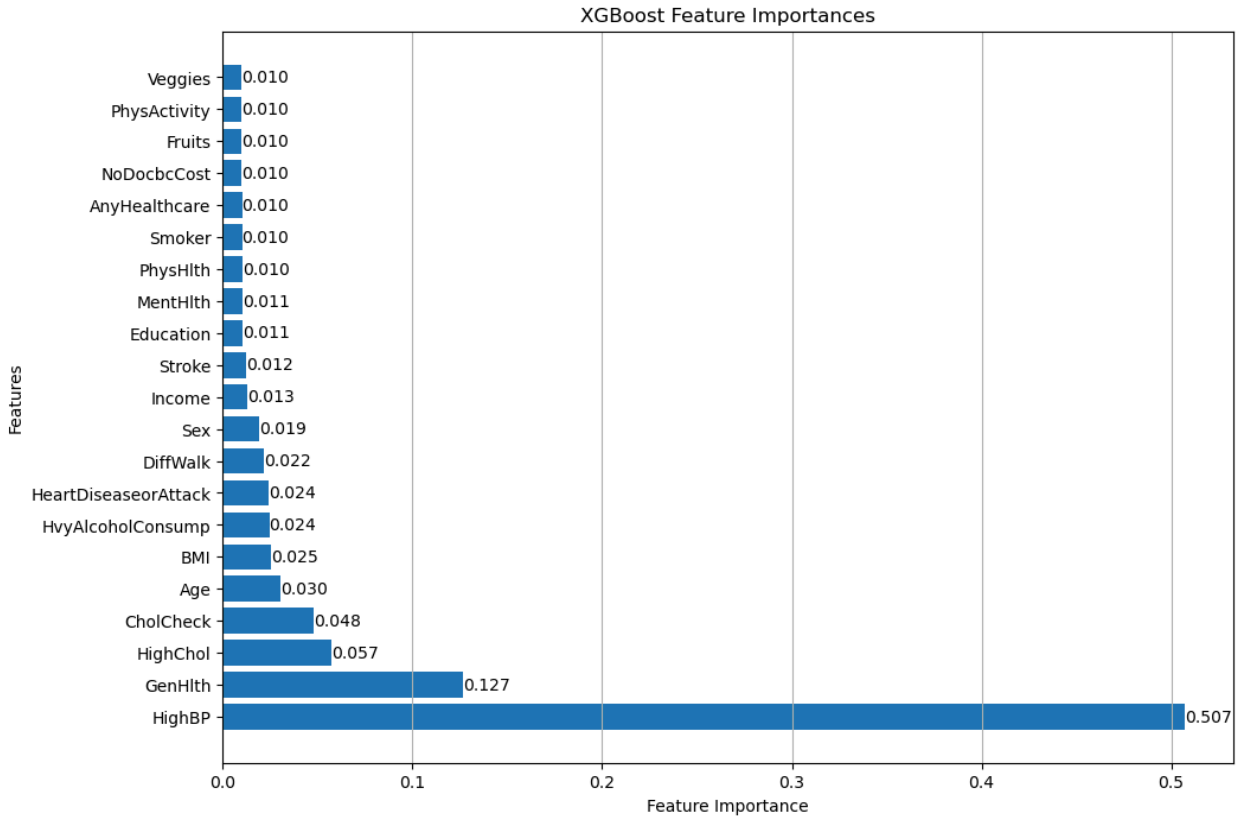


Figure A5 Feature importance horizontal bar chart for the Boosting Tree model (XGBoost), highlighting the importance of considering the imbalanced dataset during the hypertuning step. Similar to CatBoost, heavy alcohol consumption, prior conditions of heart disease or attack, as well as difficult walking are included now.

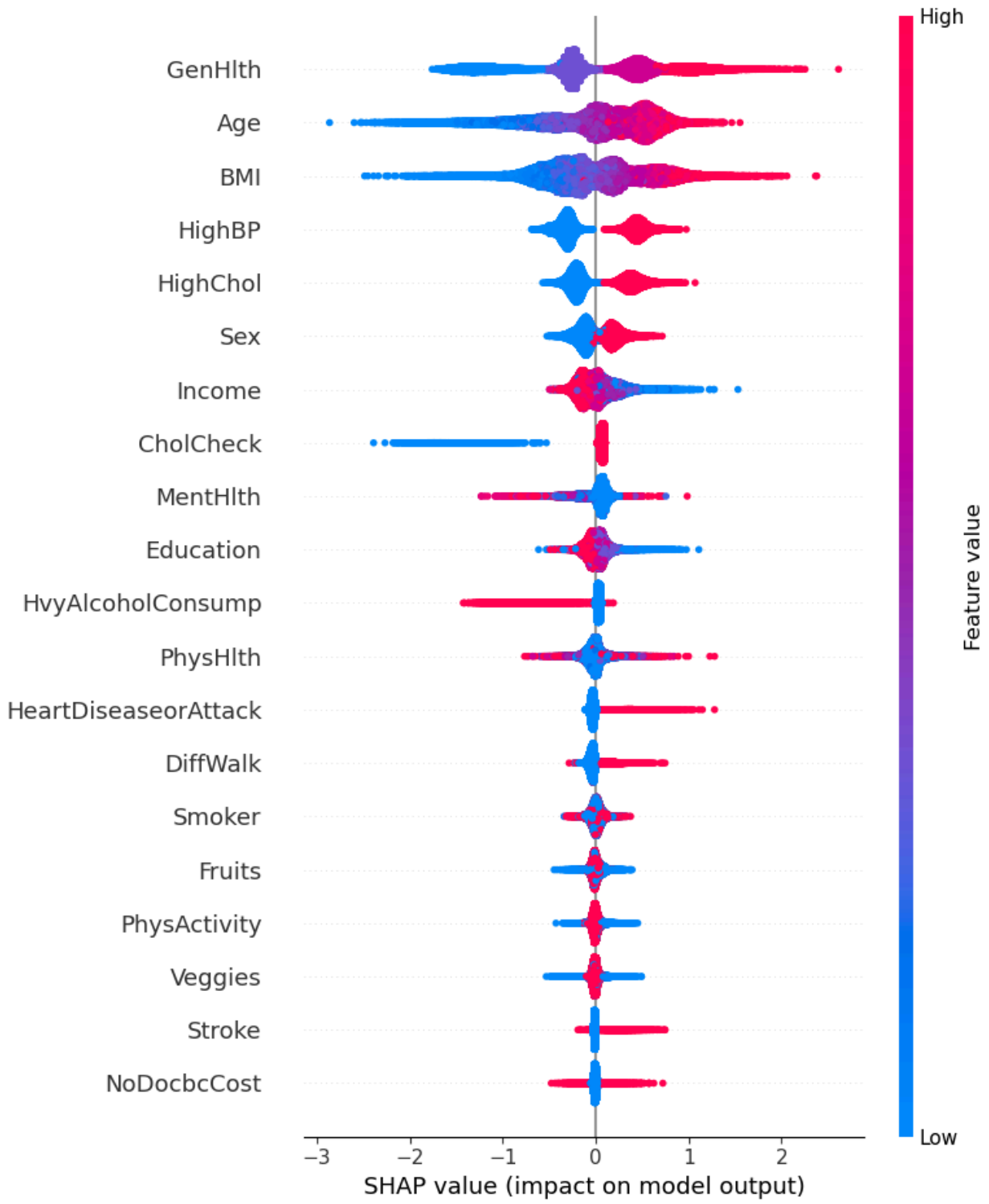


Figure A6 Summary plot of SHAP (SHapley Additive exPlanations) values that visualize the impact of different features on the model's output predictions. As described with CatBoost, the same predictors are impacting the prediction of diabetes with consideration for heavy alcohol consumption, prior heart disease or attack and difficulty walking after hypertuning for the imbalanced response variable.

Python Codes

Section 1. Discovering Connections

```
# import library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# load data
df_dbt = pd.read_csv('./data/diabetes_binary_health_indicators_BRFSS2015.csv')

# Calculate the correlation matrix
corr_matrix = df_dbt.corr(method='spearman')

# Create a mask to hide one half
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Customize colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Generate the heatmap
plt.figure(figsize=(11, 8))
sns.heatmap(corr_matrix, mask=mask, cmap=cmap, square=True,
        annot=True, fmt='.2f', linewidths=.1, annot_kws={"size": 7}, cbar_kws={"shrink": 0.7})
plt.title('Spearman Correlation Matrix')
plt.show()
```

Section 2. Predicting Diabetes

```
# import library
import pandas as pd
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_colwidth', 100)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.metrics import f1_score, roc_auc_score, r2_score
from sklearn.preprocessing import RobustScaler
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import statsmodels.api as sm

import pickle
import warnings
warnings.filterwarnings("ignore")

# load data
df_dbt = pd.read_csv('./data/diabetes_binary_health_indicators_BRFSS2015.csv')
print(df_dbt.shape)
df_dbt.head(2)


df_dbt['Diabetes_binary'].value_counts(dropna=False, normalize=True)


# # Split Data to Train/Test
df_train, df_test = train_test_split(df_dbt, test_size=0.2, random_state=42)

X = df_dbt.drop('Diabetes_binary', axis=1)
y = df_dbt[['Diabetes_binary']]

X_train = df_train.drop('Diabetes_binary', axis=1)
X_test = df_test.drop('Diabetes_binary', axis=1)

y_train = df_train[['Diabetes_binary']]
y_test = df_test[['Diabetes_binary']]

print(X_train.shape, X_test.shape)

y_test.value_counts(normalize=True)


# # Data Preprocessing
scaler = RobustScaler()

# fir scaler
X_train_scaled = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_scaled)
X_train_scaled.columns = X_train.columns
```

```
X_train_scaled.index = X_train.index
X_train_scaled.head()

# apply to test
X_test_scaled = scaler.transform(X_test)
X_test_scaled = pd.DataFrame(X_test_scaled)
X_test_scaled.columns = X_test.columns
X_test_scaled.index = X_test.index
X_test_scaled.head(2)


# # Modeling
results = pd.DataFrame(columns=['Model', 'Train Score', 'CV Score', 'Test Score', 'Params'])


# ## Logistic Regression
model_name = 'Logistic Regression'
model_idx = 0

model = LogisticRegression(random_state=42, max_iter=500)

param_grids = {'penalty': [None, 'l2'],
               'C': [0.01, 0.1, 1, 10]}

grid_search = GridSearchCV(model, param_grids, cv=5, scoring='f1_macro')
grid_search.fit(X_train_scaled, y_train)

# train_score = f1_score(y_train, grid_search.predict(X_train_scaled), average='macro')
cv_score = grid_search.best_score_
model_lr = grid_search.best_estimator_
print(model_lr.intercept_)

# find the right threshold for cutting prediction
y_pred = model_lr.predict_proba(X_train_scaled)[:, 1]
list_f1score_lr = []
for i in np.arange(0, 1, 0.01):
    list_f1score_lr.append(f1_score(y_train, y_pred>=i, average='macro'))

ind_lr = np.argmax(list_f1score_lr)
```

```
f1_thresh_lr = np.arange(0, 1, 0.01)[ind_lr]
print(f1_thresh_lr)

train_score = f1_score(y_train, model_lr.predict_proba(X_train_scaled)[:, 1]>f1_thresh_lr,
average='macro')
test_score = f1_score(y_test, model_lr.predict_proba(X_test_scaled)[:, 1]>f1_thresh_lr,
average='macro')
results.loc[model_idx] = [model_name, train_score, cv_score, test_score,
grid_search.best_params_]
print(results)

y_pred = model_lr.predict_proba(X_test_scaled)[:, 1]
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred>f1_thresh_lr))
print(classification_report(y_test, y_pred>f1_thresh_lr))

# Create Feature Importances DataFrame and sort
feature_df = pd.DataFrame({'feature': model_lr.feature_names_in_, 'importance':
model_lr.coef_[0]})
feature_df = feature_df.sort_values(by='importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(8, 6))
plt.barh(feature_df['feature'], feature_df['importance'])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title(f'Logistic Regression Feature Importances')
plt.show()

feature_df['importance_abs'] = abs(feature_df['importance'])
feature_df.sort_values('importance_abs', ascending=False)
print(feature_df)
```

Section 4. Determining Risk Levels (this is based on logistic regression model)

```
### Inspect the results of the logistic regression model after fitting
intercept = model_lr.intercept_[0]
coefficients = model_lr.coef_[0]

print("Intercept:", intercept)
print("Coefficients:", coefficients)

# Print a portion of the logistic regression equation directly
```

```
print("\nLogistic Regression Equation:")
print(f"p = 1 / (1 + exp(-({intercept} + {coefficients[0]} * x1 + {coefficients[1]} * x2 + ...)))")


### Organize predictor variables and corresponding coefficients to interpret
predictor_variables = X_train.columns

coefficients_df = pd.DataFrame({'Predictor Variable': predictor_variables, 'Coefficient':
coefficients})
coefficients_df_sorted = coefficients_df.sort_values(by='Coefficient', ascending=False)

# Adjust pandas display settings to see full dataframe
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

print(coefficients_df_sorted)

# Reset pandas display settings
pd.reset_option('display.max_rows')
pd.reset_option('display.max_columns')


### Create a dictionary to ensure accuracy when plotting/MANUALLY converting
# New coefficients (log odds)
coefficients = {
    'CholCheck': 1.253008,
    'HighBP': 0.749128,
    'HighChol': 0.561608,
    'GenHlth': 0.530341,
    'Age': 0.492992,
    'BMI': 0.425936,
    'Sex': 0.258160,
    'HeartDiseaseorAttack': 0.233976,
    'DiffWalk': 0.143904,
    'Stroke': 0.142298,
    'AnyHealthcare': 0.060799,
    'NoDocbcCost': 0.018586,
    'Smoker': -0.003604,
    'MentHlth': -0.007258,
    'PhysHlth': -0.022389,
    'Veggies': -0.036432,
    'Fruits': -0.044589,
    'PhysActivity': -0.049266,
    'Education': -0.059140,
    'Income': -0.154018,
    'HvyAlcoholConsump': -0.735539
```

```
}

# Convert log odds (coefficients) of each predictor to probabilities MANUALLY to interpret
probabilities = {key: 1 / (1 + np.exp(-value)) for key, value in coefficients.items()}

# Print probabilities
for key, value in probabilities.items():
    print(f"{key}: {value}")


### Plot the predicted probabilities of each predictor for visually interpreting
# Define colormap from red to yellow to green to represent risk level approximations
cmap = plt.get_cmap('RdYlGn_r') # 'RdYlGn' goes from green to red

# Normalize probabilities to [0, 1] for colormap
norm = plt.Normalize(min(probabilities.values()), max(probabilities.values()))

# Plot probabilities with reversed colormap
plt.figure(figsize=(10, 6))
for predictor, probability in probabilities.items():
    color = cmap(norm(probability))
    plt.barh(predictor, probability, color=color)
    plt.text(probability, predictor, f'{probability:.2f}', va='center') # Add probability beside the bar

plt.xlabel('Probability')
plt.title('Predicted Probabilities of Diabetes')
plt.grid(axis='x')
plt.show()


### Extract predicted probabilities of diabetes for EACH INDIVIDUAL to determine risk levels
# Predict probabilities using the logistic regression model on original data
probabilities = model_lr.predict_proba(scaler.fit_transform(X))

# Extract probabilities for the positive class
positive_class_probabilities = probabilities[:, 1]

len(positive_class_probabilities) # Ensure that all invididuals are included


### Explore the summary statistics to determine approximate percentiles for risk levels
df_probabilities = pd.DataFrame({'Probabilities': positive_class_probabilities})
summary_statistics = df_probabilities.describe()
print(summary_statistics)

# Calculate the 20th and 80th percentiles to approximate risk levels
```

```
percentile_20 = np.percentile(positive_class_probabilities, 20)
percentile_80 = np.percentile(positive_class_probabilities, 80)

print("20th percentile:", percentile_20)
print("80th percentile:", percentile_80)

# Find the approximate percentiles actually used, or from K-means clustering results
value = 0.23 # higher: 0.36918215166172963 # lower: 0.14294478997145707
approx_percentile = np.sum(positive_class_probabilities <= value) /
len(positive_class_probabilities) * 100

print("Approximate percentile:", approx_percentile)


### Labeling EACH INDIVIDUAL in a new Dataframe as low, medium, or high risk
# Adjust thresholds based on the summary statistics/literature
low_threshold = 0.03
high_threshold = 0.23

# Assign risk levels based on predicted probabilities
risk_levels = []
for prob in positive_class_probabilities:
    if prob <= low_threshold:
        risk_levels.append('Low')
    elif prob <= high_threshold:
        risk_levels.append('Medium')
    else:
        risk_levels.append('High')

# Create a copy of the original DataFrame to avoid modifying it
df_dbt_updated = df_dbt.copy()

# Add risk levels to the new DataFrame
df_dbt_updated['Risk_level'] = risk_levels
df_dbt_updated.columns


### Find predictor variables that represent low or high risk class, on average
# Extract the predictor columns from new Dataframe to ensure consistency
predictor_columns = df_dbt_updated.columns[1:-1] # Exclude 'Risk Level' and 'Diabetes_binary'

# Create empty lists to store columns with higher means for low and high risk
higher_mean_low_risk = []
higher_mean_high_risk = []

# Iterate over each predictor column
```

```
for column in predictor_columns:
    # Check if the mean for the low-risk group is higher than the mean for the high-risk group
    if df_dbt_updated.loc[df_dbt_updated['Risk_level'] == 'Low', column].mean() >
df_dbt_updated.loc[df_dbt_updated['Risk_level'] == 'High', column].mean():
        higher_mean_low_risk.append(column)
    else:
        higher_mean_high_risk.append(column)

# Print the columns with higher means for low and high risk
print("Columns with higher means for Low Risk:", higher_mean_low_risk)
print("Columns with higher means for High Risk:", higher_mean_high_risk)


### Plot the number of individuals that were labeled low, medium, or high risk
# Define custom colors for low, medium, and high risk levels
custom_palette = {'Low': 'green', 'Medium': 'orange', 'High': 'red'}

# Create a count plot to visualize the distribution of risk levels
plt.figure(figsize=(6, 5))
ax = sns.countplot(x='Risk_level', data=df_dbt_updated, palette=custom_palette, order=['Low',
'Medium', 'High'])
plt.title('Distribution of Individuals Based on Percentiles')
plt.ylabel('Count')

# Add count labels on top of each bar
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.show()


### Extract the summary statistics for all predictors based on risk level for analysis
# Adjust pandas display settings to see all summary statistics
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

# Group the data by 'Risk Level' and calculate the mean and standard deviation for each predictor
risk_level_stats = df_dbt_updated.groupby('Risk_level').agg(['mean', 'std'])
risk_level_stats_df = pd.DataFrame(risk_level_stats)

# Round the summary stats for export
risk_level_stats_rounded = round(risk_level_stats, 2)
risk_level_stats_rounded_df = pd.DataFrame(risk_level_stats_rounded)
```

```
# Save the DataFrame to an Excel file
risk_level_stats_rounded_df.to_excel('risk_level_statistics.xlsx')

print(risk_level_stats_rounded_df)

# Reset pandas display settings
pd.reset_option('display.max_rows')
pd.reset_option('display.max_columns')


### Perform K-means clustering to determine risk level groups for comparison
from sklearn.cluster import KMeans

# Reshape the probabilities array to have a single feature
positive_class_probabilities_reshaped = positive_class_probabilities.reshape(-1, 1)

# Initialize the KMeans model with 3 clusters
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=33, max_iter=333,
random_state=333)

# Fit the KMeans model on the probabilities
kmeans.fit(positive_class_probabilities_reshaped)

# Get the cluster labels
cluster_labels = kmeans.labels_

# Add cluster labels to the DataFrame
df_dbt_updated['Cluster'] = cluster_labels
df_dbt_updated.columns


### Find the range of values from K-means clustering to determine thresholds/risk level
# Initialize an empty dictionary to store the range of probabilities for each cluster
cluster_probabilities_range = {}

# Iterate over each cluster
for cluster in range(3):
    # Filter the DataFrame to get the positive class probabilities for the current cluster
    cluster_probabilities = positive_class_probabilities[df_dbt_updated['Cluster'] == cluster]

    # Determine the range of probabilities for the current cluster
    min_prob = np.min(cluster_probabilities)
    max_prob = np.max(cluster_probabilities)

    # Store the range in the dictionary
    cluster_probabilities_range[cluster] = (min_prob, max_prob)
```

```
# Print the range of probabilities for each cluster
for cluster, prob_range in cluster_probabilities_range.items():
    print(f"Cluster {cluster} Probabilities Range: {prob_range}")

# Find the approximate percentiles from K-means clustering results
value = 0.14 # higher: 0.36919814140800916 # lower: 0.14294246894407064
approx_percentile = np.sum(positive_class_probabilities <= value) /
len(positive_class_probabilities) * 100

print("Approximate percentile:", approx_percentile)


### Plot the number of individuals that were clustered in each risk level
# Define custom colors for low, medium, and high clusters
custom_palette = {'Low': 'green', 'Medium': 'orange', 'High': 'red'}

# MANUALLY define the mapping dictionary depending on how K-means clusters ABOVE
mapping = {0: 'Low', 2: 'Medium', 1: 'High'}

# Apply the mapping to the cluster labels in the DataFrame
df_dbt_updated['Cluster'] = df_dbt_updated['Cluster'].map(mapping)

# Count the number of occurrences of each cluster after reordering
cluster_counts_reordered = df_dbt_updated['Cluster'].value_counts()

# Reorder the cluster_counts_reordered DataFrame based on the desired order of the levels
cluster_counts_reordered = cluster_counts_reordered.reindex(['Low', 'Medium', 'High'])

# Plot the distribution of reordered clusters with custom palette
plt.figure(figsize=(6, 5))
ax = sns.barplot(x=cluster_counts_reordered.index, y=cluster_counts_reordered.values,
palette=custom_palette)
plt.title('Distribution of Individuals Based on K-means Clustering')
plt.xlabel('Risk Level')
plt.ylabel('Count')

# Add count labels above each bar
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.show()
```

```
### Extract the summary statistics for all predictors based on clusters for analysis
# Adjust pandas display settings to see all summary statistics
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

# Group the data by 'Cluster' and calculate the mean and standard deviation for each cluster
cluster_stats = df_dbt_updated.groupby('Cluster').agg(['mean', 'std'])
cluster_stats_df = pd.DataFrame(cluster_stats)

# Round the summary stats for export
cluster_stats_rounded = round(cluster_stats, 2)
cluster_stats_rounded_df = pd.DataFrame(cluster_stats_rounded)

# Save the DataFrame to an Excel file
cluster_stats_rounded_df.to_excel('cluster_stats.xlsx')

print(cluster_stats_rounded_df)

# Reset pandas display settings
pd.reset_option('display.max_rows')
pd.reset_option('display.max_columns')

### Section 2. Predicting Diabetes CONTINUED HERE
# ### using SM
logit_model = sm.Logit(y_train, sm.add_constant(X_train_scaled))
result = logit_model.fit()
print(result.summary())



# ## Decision Tree
model_name = 'Decision Tree'
model_idx = 1

model = DecisionTreeClassifier(random_state=42)

param_grids = {'max_depth': [3, 5,
                             7, 9, 11],
               'min_samples_leaf': [1, 2, 4],
               'min_samples_split': [2, 5, 10],
               'criterion': ['gini', 'entropy']}

grid_search = GridSearchCV(model, param_grids, cv=5, scoring='f1_macro')
```

```
grid_search.fit(X_train_scaled, y_train)

# train_score = f1_score(y_train, grid_search.predict(X_train_scaled), average='macro')
cv_score = grid_search.best_score_
# test_score = f1_score(y_test, grid_search.predict(X_test_scaled), average='macro')

# results.loc[model_idx] = [model_name, train_score, cv_score, test_score,
grid_search.best_params_]
model_dt = grid_search.best_estimator_

# find the right threshold for cutting prediction
y_pred = model_dt.predict_proba(X_train_scaled)[:, 1]
list_f1score_dt = []
for i in np.arange(0, 1, 0.01):
    list_f1score_dt.append(f1_score(y_train, y_pred>=i, average='macro'))

ind_dt = np.argmax(list_f1score_dt)
f1_thresh_dt = np.arange(0, 1, 0.01)[ind_dt]
print(f1_thresh_dt)

train_score = f1_score(y_train, model_dt.predict_proba(X_train_scaled)[:, 1]>f1_thresh_dt,
average='macro')
test_score = f1_score(y_test, model_dt.predict_proba(X_test_scaled)[:, 1]>f1_thresh_dt,
average='macro')
results.loc[model_idx] = [model_name, train_score, cv_score, test_score,
grid_search.best_params_]
print(results)

y_pred = model_dt.predict_proba(X_test_scaled)[:, 1]
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred>f1_thresh_dt))
print(classification_report(y_test, y_pred>f1_thresh_dt))

# Extract feature importances
importances = model_dt.feature_importances_

# Create DataFrame and sort
feature_df = pd.DataFrame({'feature': X_train_scaled.columns, 'importance': importances})
feature_df = feature_df.sort_values(by='importance', ascending=False)

# Plot the feature importances
```

```
plt.figure(figsize=(8, 6))
plt.barh(feature_df['feature'], feature_df['importance'])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title(f'Decision Tree Feature Importances')
plt.show()

print(feature_df)


# ## Random Forest
# This cell takes time about 35 minutes.
model_name = 'Random Forest'
model_idx = 2

model = RandomForestClassifier(random_state=42, n_jobs=-1)

param_grids = {'n_estimators': [50, 100, 200,
                                300, 500],
               'min_samples_leaf': [1, 2, 4],
               'min_samples_split': [2, 5, 10],
               'max_depth': [3, 5, 7, 9, 11]}

grid_search = GridSearchCV(model, param_grids, cv=5, scoring='f1_macro')
grid_search.fit(X_train_scaled, y_train)

cv_score = grid_search.best_score_
model_rf = grid_search.best_estimator_

# find the right threshold for cutting prediction
y_pred = model_rf.predict_proba(X_train_scaled)[:, 1]
list_f1score_rf = []
for i in np.arange(0, 1, 0.01):
    list_f1score_rf.append(f1_score(y_train, y_pred>=i, average='macro'))

ind_rf = np.argmax(list_f1score_rf)
f1_thresh_rf = np.arange(0, 1, 0.01)[ind_rf]
print(f1_thresh_rf)
```

```
train_score = f1_score(y_train, model_rf.predict_proba(X_train_scaled)[:, 1]>f1_thresh_rf,
average='macro')
test_score = f1_score(y_test, model_rf.predict_proba(X_test_scaled)[:, 1]>f1_thresh_rf,
average='macro')
results.loc[model_idx] = [model_name, train_score, cv_score, test_score,
grid_search.best_params_]
print(results)

y_pred = model_rf.predict_proba(X_test_scaled)[:, 1]
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred>f1_thresh_rf))
print(classification_report(y_test, y_pred>f1_thresh_rf))

# Extract feature importances
importances = model_rf.feature_importances_

# Create DataFrame and sort
feature_df = pd.DataFrame({'feature': X_train_scaled.columns, 'importance': importances})
feature_df = feature_df.sort_values(by='importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(8, 6))
plt.barh(feature_df['feature'], feature_df['importance'])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title(f'Random Forest Feature Importances')
plt.show()

print(feature_df)

# ## Save models
# save trained models
# model_pkl_file = "./models/data583_2classdiabetes_3models.pkl"

# with open(model_pkl_file, 'wb') as file:
#     pickle.dump(model_lr, file)
#     pickle.dump(model_dt, file)
#     pickle.dump(model_rf, file)
#     pickle.dump(scaler, file)

# load trained models
```

```
# model_pkl_file = "./models/data583_2classdiabetes_3models.pkl"

# file = open(model_pkl_file,'rb')
# model_lr = pickle.load(file)
# model_dt = pickle.load(file)
# model_rf = pickle.load(file)
# scaler = pickle.load(file)
# file.close()


# ## All models performance
results = results.sort_values('Test Score', ascending=False)
plt.figure(figsize=(6, 4))
plt.bar(results['Model'], results['Test Score'])
plt.xlabel('Models')
plt.ylabel('F1-Score')
plt.title('Model Performance Comparison (Testing Dataset)')
plt.show()

# CV Score auto-cut the prediction with a threshold equals to 0.5.
results.drop(columns='CV Score')
```

Catboost Code
This section is run separately, please make sure you run this separately

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os


# df = pd.read_csv('/Users/nijat/Downloads/AdvancedProject/diabetes.csv')
df = df_dbt.copy()  # copied from the the previous section
ordinal_info = {
    'GenHlth': ['1.0', '2.0', '3.0', '4.0', '5.0'],
    'Education': ['1.0', '2.0', '3.0', '4.0', '5.0', '6.0'],
    'Income': ['1.0', '2.0', '3.0', '4.0', '5.0', '6.0', '7.0', '8.0'],
    'Age': ['1.0', '2.0', '3.0', '4.0', '5.0', '6.0', '7.0', '8.0', '9.0', '10.0', '11.0', '12.0', '13.0']
}
```

```
for var, order in ordinal_info.items():

    df[var] = df[var].astype(str)
    df[var] = pd.Categorical(df[var], categories=order, ordered=True)

binary_cols = ['Diabetes_binary', 'HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke',
'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump',
'AnyHealthcare', 'NoDocbcCost', 'DiffWalk', 'Sex']
for col in binary_cols:
    df[col] = df[col].astype('category')

from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd


X = df.drop(columns=['Diabetes_binary'])
y = df['Diabetes_binary'].astype('int')
categorical_features_indices = np.where(X.dtypes != np.float64)[0]

categorical_features_names = X.columns[categorical_features_indices]

# Convert only the categorical columns to strings
for col in categorical_features_names:
    X[col] = X[col].astype(str)

# Test, train split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=27
)

new_working_directory = "/Users/nijat/Downloads/CatBoost"  # Update this path
os.chdir(new_working_directory)

# CatBoost Classifier with categorical feature indices
cb_model = CatBoostClassifier(
    iterations=1000,
    learning_rate=0.1,
```

```
    depth=6,
    cat_features=categorical_features_indices,
    eval_metric='Accuracy',
    auto_class_weights='Balanced',
    random_state=27,
    #verbose=100,
    logging_level='Silent',
    train_dir= new_working_directory)
# Fit the model
cb_model.fit(X_train, y_train, eval_set=(X_test, y_test),use_best_model=True)

# Get feature importances
feature_importances = cb_model.get_feature_importance()

# DataFrame for visualization
features_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
})

# Sort by importance in descending order
features_df = features_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)
print(features_df)

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 8))
sns.barplot(x="Importance", y="Feature", data=features_df)
plt.title('Feature Importance')
plt.show()


import shap
from catboost import Pool

# SHAP values explainer
explainer = shap.TreeExplainer(cb_model)

# Compute SHAP values
```

```
shap_values = explainer.shap_values(Pool(X_test, cat_features=categorical_features_indices))

# Summary plot for all features
shap.summary_plot(shap_values, X_test)

# Confusion matrix and Precision-Recall figure for CatBoost
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc
import numpy as np

pred_probs = cb_model.predict_proba(X_test)
precision, recall, _ = precision_recall_curve(y_test, pred_probs[:, 1])
pr_auc = auc(recall, precision)

preds_classes = cb_model.predict(X_test)
# Generate confusion matrix
cm = confusion_matrix(y_test, preds_classes)
# Create subplots for confusion matrix and precision-recall curve
fig, axs = plt.subplots(1, 2, figsize=(14, 7))
# Confusion Matrix
cax = axs[0].matshow(cm, cmap=plt.cm.Blues)
fig.colorbar(cax, ax=axs[0])
axs[0].set_title('Confusion Matrix')
axs[0].set_xlabel('Predicted labels')
axs[0].set_ylabel('True labels')
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        axs[0].text(j, i, format(cm[i, j], 'd'), ha="center", va="center",
                    color="white" if cm[i, j] > cm.max() / 2 else "black")
# Precision-Recall Curve
axs[1].plot(recall, precision, label=f'AUC = {pr_auc:.2f}')
axs[1].set_title('Precision-Recall Curve')
axs[1].set_xlabel('Recall')
axs[1].set_ylabel('Precision')
axs[1].legend(loc='upper right')
plt.tight_layout()
plt.show()
```

 # Neural network analysis
This section is run separately, please make sure you run this separately

```
import pandas as pd
import numpy as np
import tensorflow as tf
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

random.seed(27)
np.random.seed(27)
tf.random.set_seed(27)

#  df = pd.read_csv('/Users/nijat/Downloads/AdvancedProject/diabetes.csv')
df = df_dbt.copy()  # copied from the the previous section

ordinal_info = {
    'GenHlth': ['1.0', '2.0', '3.0', '4.0', '5.0'],
    'Education': ['1.0', '2.0', '3.0', '4.0', '5.0', '6.0'],
    'Income': ['1.0', '2.0', '3.0', '4.0', '5.0', '6.0', '7.0', '8.0'],
    'Age': ['1.0', '2.0', '3.0', '4.0', '5.0', '6.0', '7.0', '8.0', '9.0', '10.0', '11.0', '12.0', '13.0']
}

for var, order in ordinal_info.items():
    df[var] = df[var].astype(str)
    df[var] = pd.Categorical(df[var], categories=order, ordered=True)

binary_cols = ['Diabetes_binary', 'HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke',
'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump',
'AnyHealthcare', 'NoDocbcCost', 'DiffWalk', 'Sex']
for col in binary_cols:
    df[col] = df[col].astype('category')

X = df.drop('Diabetes_binary', axis=1)
y = df['Diabetes_binary'].cat.codes
```

```
scaler = StandardScaler()
X[['BMI', 'MentHlth', 'PhysHlth']] = scaler.fit_transform(X[['BMI', 'MentHlth', 'PhysHlth']])
X = pd.get_dummies(X, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=27)
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train),
y=y_train)
class_weight_dict = dict(enumerate(class_weights))
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2,
class_weight=class_weight_dict, verbose=1)

loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
y_pred = np.argmax(model.predict(X_test), axis=1)
y_true = np.argmax(y_test, axis=1)
f1 = f1_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
print(f'Test accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')

# Confusion matrix and Precision-Recall figure for Neural Network
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc
import matplotlib.pyplot as plt
import numpy as np

predictions = model.predict(X_test)
preds_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_test, axis=1)
```

```
cm = confusion_matrix(true_classes, preds_classes)
precision, recall, _ = precision_recall_curve(true_classes, predictions[:, 1])
pr_auc = auc(recall, precision)

fig, axs = plt.subplots(1, 2, figsize=(14, 7))
# Confusion Matrix Plot
cax = axs[0].matshow(cm, cmap=plt.cm.Blues)
fig.colorbar(cax, ax=axs[0])
axs[0].set_title('Confusion Matrix')
axs[0].set_xlabel('Predicted labels')
axs[0].set_ylabel('True labels')

for i in range(cm.shape[0]):
  for j in range(cm.shape[1]):
    axs[0].text(j, i, format(cm[i, j], 'd'),
            ha="center", va="center",
            color="white" if cm[i, j] > cm.max()/2 else "black")
# Precision-Recall Curve Plot
axs[1].plot(recall, precision, label=f'AUC = {pr_auc:.2f}')
axs[1].set_title('Precision-Recall Curve')
axs[1].set_xlabel('Recall')
axs[1].set_ylabel('Precision')
axs[1].legend(loc='upper right')
plt.tight_layout()
plt.show()
```

Boosting Tree Model

```
from sklearn.preprocessing import StandardScaler, RobustScaler

### Preprocess the data after CatBoost/Neural Network processed X, y etc.
# Create a copy of the DataFrame to avoid modifying the original data
processed_data = df_dbt.copy()

# Select columns to be scaled
columns_to_scale = processed_data.columns.drop('Diabetes_binary')

# Initialize Scalers for comparison
scaler = RobustScaler()
# scaler = StandardScaler()
```

```
# Fit and transform the selected columns
processed_data[columns_to_scale] = scaler.fit_transform(processed_data[columns_to_scale])

# Adjust pandas display settings to see all predictors
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

print(processed_data.head())

# Reset pandas display settings
pd.reset_option('display.max_rows')
pd.reset_option('display.max_columns')


### Fit the model using the parameters determined from grid search hypertuning
import xgboost as xgb
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, classification_report, f1_score

# Prepare the data for XGBoost
X = processed_data.drop('Diabetes_binary', axis=1)
y = processed_data['Diabetes_binary']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=333)

# Define the XGBoost model
# xgb_model = xgb.XGBClassifier(gamma=0.1, max_depth=7, min_child_weight=3,
subsample=0.6, n_jobs=-1, random_state=333) # WITHOUT BALANCING
xgb_model = xgb.XGBClassifier(gamma=0.0, max_depth=7, min_child_weight=1,
subsample=1.0, scale_pos_weight=6.209634445273367, n_jobs=-1, random_state=333) # WITH
BALANCING

# Train the model
xgb_model.fit(X_train, y_train)

# Predict on the training set
y_train_pred = xgb_model.predict(X_train)

# Predict on the test set
y_test_pred = xgb_model.predict(X_test)

# Calculate F1 scores
train_f1_score = f1_score(y_train, y_train_pred, average='macro')
cv_f1_scores = cross_val_score(xgb_model, X_train, y_train, cv=5, scoring='f1_macro')
test_f1_score = f1_score(y_test, y_test_pred, average='macro')
```

```
# Print F1 scores
print("Train F1 Score (Macro):", train_f1_score)
print("CV F1 Score (Macro):", cv_f1_scores.mean())
print("Test F1 Score (Macro):", test_f1_score)

# Confusion matrix and classification report for train set
print("\nConfusion Matrix - Train Set:")
print(confusion_matrix(y_train, y_train_pred))
print("\nClassification Report - Train Set:")
print(classification_report(y_train, y_train_pred))

# Confusion matrix and classification report for test set
print("\nConfusion Matrix - Test Set:")
print(confusion_matrix(y_test, y_test_pred))
print("\nClassification Report - Test Set:")
print(classification_report(y_test, y_test_pred))


### Plot Boosting Tree Feature Importances
import matplotlib.pyplot as plt

# Get feature importances from the XGBoost model
feature_importances = xgb_model.feature_importances_

# Get the names of features
feature_names = X.columns

# Sort feature importances in descending order
sorted_indices = feature_importances.argsort()[::-1]
sorted_feature_importances = feature_importances[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]

# Plot feature importance chart
plt.figure(figsize=(11, 8))
bars = plt.barh(range(len(sorted_feature_importances)), sorted_feature_importances,
tick_label=sorted_feature_names)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('XGBoost Feature Importances')
plt.grid(axis='x')

# Add numbers beside the bars
for bar, importance in zip(bars, sorted_feature_importances):
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{importance:.3f}', va='center')
```

```
plt.show()


### Plot the SHAP Summary Plot
import shap

# Create a SHAP explainer object for the XGBoost model
explainer = shap.Explainer(xgb_model, X_train)

# Compute SHAP values
shap_values = explainer.shap_values(X_test)

# Plot SHAP summary
shap.summary_plot(shap_values, X_test)
```

GRID SEARCH OF BOOSTING TREE CAN TAKE OVER AN HOUR (COMMENTED OUT, BEST PARAMETERS USED ABOVE)

```
# import xgboost as xgb
# from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
# from sklearn.metrics import confusion_matrix, classification_report, f1_score

# # Prepare the data for XGBoost
# X = processed_data.drop('Diabetes_binary', axis=1)
# y = processed_data['Diabetes_binary']

# # Split the data into train and test sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=333)

# # Define the XGBoost model
# xgb_model = xgb.XGBClassifier(random_state=333, n_jobs=-1)

# # Define hyperparameters for GridSearchCV
# # USING ALL parameters below - first/second attempts both took over 60 MINUTES
# # max_depth, min_child_weight, gamma, subsample PORTION TOOK 24 MINUTES
# param_grid = {
#     'max_depth': [3, 5, 7], # maximum depth of each tree in the ensemble. Deeper trees can capture more complex patterns in the data but are more prone to overfitting.
#     'min_child_weight': [1, 3, 5], # minimum sum of instance weight (hessian) needed in a child. It helps to control overfitting by adding regularization to the leaf nodes.
#     'gamma': [0, 0.1, 0.2], # A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split, which acts as regularization by preventing overfitting.
#     'subsample': [0.6, 0.8, 1.0] # fraction of samples used to train each tree. A value less than 1.0 introduces stochasticity into the training process, which helps to prevent overfitting.
```

```
#     # 'colsample_bytree': [0.6, 0.8, 1.0], # fraction of features (columns) used to train each tree. Similar to subsample, it introduces stochasticity into the training process and helps to prevent overfitting.
#     # 'reg_alpha': [0, 0.1, 0.5],
#     # 'reg_lambda': [1, 1.5, 2] # L1 and L2 regularization terms applied to the weights of the features, respectively. They add penalty terms to the objective function, encouraging simpler models and reducing overfitting.

#     # ADDING scale_pos_weight DOUBLED THE TIME FROM 24 MINS TO 48 MINS
----------
#     # 'scale_pos_weight': [1, y_train.value_counts()[0] / y_train.value_counts()[1]] # Ratio of negative to positive class samples
# }

# # Initialize GridSearchCV
# grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, scoring='f1_macro')

# # Fit GridSearchCV to training data
# grid_search.fit(X_train, y_train)

# # Get the best parameters from GridSearchCV
# best_params = grid_search.best_params_
# print("Best Parameters:", best_params)

# # Predict on the training set with the best model
# y_train_pred = grid_search.predict(X_train)

# # Predict on the test set with the best model
# y_test_pred = grid_search.predict(X_test)

# # Calculate F1 scores with the best model
# train_f1_score = f1_score(y_train, y_train_pred, average='macro')
# cv_f1_score = cross_val_score(grid_search.best_estimator_, X_train, y_train, cv=5, scoring='f1_macro').mean()
# test_f1_score = f1_score(y_test, y_test_pred, average='macro')

# # Print F1 scores
# print("Train F1 Score (Macro):", train_f1_score)
# print("CV F1 Score (Macro):", cv_f1_score)
# print("Test F1 Score (Macro):", test_f1_score)

# # Confusion matrix and classification report for train set
# print("\nConfusion Matrix - Train Set:")
# print(confusion_matrix(y_train, y_train_pred))
# print("\nClassification Report - Train Set:")
```

```
# print(classification_report(y_train, y_train_pred))

# # Confusion matrix and classification report for test set
# print("\nConfusion Matrix - Test Set:")
# print(confusion_matrix(y_test, y_test_pred))
# print("\nClassification Report - Test Set:")
# print(classification_report(y_test, y_test_pred))
```