IS424 - Data Mining and Business Analytics

Spotting the Sweet Signs: Predicting Diabetes using Health Indicators
G1T2

Members:

Abhisit Anupapphan (01442701, abhisita.2021)
Ahmad Mirza Bin Fazili (01391751, ahmadmirzaf.2021)
Anirban Ghosh (01397458, anirbang.2021)
Lee Jun Hui (01379175, junhui.lee.2020)
Lim En-Rei Renee Ashley (01441101, reneelim.2021)
Lee Shao Ming (01418136, smlee.2020)
Vera Keh Qi (01387228, verakeh.2019)
Vighnesh Ramasamy (01394515, ramasamyv.2021)

Table of Contents

1. Introduction

Diabetes is one of the most prevalent chronic diseases today. According to the World Health Organization, “about 422 million people worldwide have diabetes, the majority living in low-and middle-income countries, and 1.5 million deaths are directly attributed to diabetes each year” and “both the number of cases and the prevalence of diabetes have been steadily increasing over the past few decades”, (World Health Organization, n.d.). In addition, statistics also show that nearly one in two persons, aged 20 to 79 years old, with diabetes were unaware of their diabetic status in 2021 and the highest prevalence of undiagnosed diabetes (53.6%) were found in the Africa, Western Pacific (52.8%) and South-East Asia (51.3%) areas, (Rice & Galbraith, 2008). Furthermore, if people are not aware of their conditions, it will get worse over time and more serious complications like kidney failure and heart disease will occur, increasing the risk of early death. Just in 2017 alone, the IDF estimated that diabetes was responsible for four million deaths, with half of them occurring prematurely before the age of 60 on average, (Open Access Government, 2018). Therefore, predicting if a person has diabetes or no diabetes based on health indicators can help to prevent more tragedy from happening.

2. Motivation

Why should we care?

If a person who is more likely to develop diabetes is not properly diagnosed and treated, the risks of developing diabetes complications are significantly higher. Premature heart disease, stroke, blindness, limb amputation, and kidney failure are among these complications. It is possible that diabetic symptoms may not be obvious for a long time, and the condition may have gotten worse without the person knowing.

This is precisely why it is important to diagnose pre-diabetes early, so that patients can be informed of their condition and take action to avoid serious complications that can lower quality of life (Apollo Diagnostics, 2020).

Why is the problem relevant?

Prediction of diabetes in the early stages is important because it reduces the lethal repercussions of diabetes. With the advances in technology such as machine learning and data mining, diabetes can now be predicted more accurately. Not only have these methods been proven to be beneficial in the

diagnosis and prognosis of a disease, but it has also proven to be helpful in formulating personalised treatment, behavioural modification, manufacturing of drugs and discovering new patterns resulting in new breakthroughs with regards to medications, treatments, clinical trial research and smart electronic health records (Saxena et al., 2022).

3. Literature Review

Several studies using machine learning techniques have shown to be promising in predicting diabetes in patients.

Case Study 1: Predicting Diabetes Mellitus With Machine Learning Techniques

After collecting physical examination data from a hospital in Luzhou, China, researchers used several algorithms including decision trees, random forests, and neural networks on a diabetes dataset. The study utilised principal component analysis (PCA) to reduce dimensionality and improve the performance of the algorithms. The results show that the random forest algorithm performed the best in terms of accuracy (ACC = 0.8084) in predicting diabetes.

Points for Improvement:

1. **Adding more recent data** and updates in the field of diabetic prediction models, as the knowledge cutoff of the paper is 2018.
2. **Model comparison:** The study could compare the performance of other machine learning algorithms beyond decision trees, random forests, and neural networks to find the best-performing algorithm for predicting diabetes.
3. **Hyperparameter tuning:** The study could investigate the impact of hyperparameter tuning on the performance of the machine learning algorithms.
4. **Feature selection:** The study could evaluate different methods of feature selection to improve the performance of the algorithms and potentially reduce the complexity of the model.
5. **Performance evaluation:** The study could use other performance metrics beyond accuracy to evaluate the performance of the algorithms, such as precision, recall, F1 score, and AUC.

These potential improvements will enhance the robustness and generalizability of the results and provide a more comprehensive analysis of the effectiveness of machine learning techniques in predicting diabetes mellitus.

Case Study 2: A Comprehensive Review of Various Diabetic Prediction Models: A Literature Survey

It is a study that reviewed various diabetic prediction models and analysed their strengths and weaknesses. The authors conducted a literature survey to gather information on different prediction models and analysed their accuracy, sensitivity, specificity, and other important factors. The study found that various models, including artificial neural networks, decision trees, and logistic regression, can be effectively used for diabetic prediction, but each has its own strengths and weaknesses. The authors concluded that further research is needed to develop more accurate and efficient diabetic prediction models.

Points for improvement:

1. **Dataset size:** The study could benefit from using a larger and more diverse dataset to improve the generalizability of the results as the current training data only includes 459 patients, and testing data includes 128 patients.
2. Including a **broader range of diabetic prediction models**, including newer and more advanced techniques such as deep learning.
3. Adding **more detailed analysis** of the performance of the models and a comparison of their accuracy, sensitivity, and specificity.
4. Providing **more practical applications and real-world examples** of the use of the models.
    a. Integrating case studies and examples of real-world applications into the paper to illustrate the practical usefulness of the models.
5. Including a **discussion of the limitations** of the study and suggestions for future research.

4. Dataset

Every year, the Centers for Disease Control and Prevention survey has collected responses from more than 400,000 Americans on health-related risk behaviours, chronic health conditions, and the use of preventative services. The diabetes dataset is a clean dataset consisting **70,692** survey responses and **22 dimensions**.

Target Variable

1. Diabetes_binary (0 = no diabetes | 1 = prediabetes | 2 = diabetes)

Independent Variables

2. HighBP (0 = no high BP | 1 = high BP)
3. HighChol (0 = no high cholesterol | 1 = high cholesterol)
4. CholCheck (0 = no cholesterol check in 5 years | 1 = yes cholesterol check in 5 years)
5. BMI (Body Mass Index)
6. Smoker (Have you smoked at least 100 cigarettes in your entire life? 0 = no | 1 = yes)
7. Stroke (Have you ever had a stroke? 0 = no | 1 = yes)
8. HeartDiseaseorAttack (Coronary heart disease (CHD) or myocardial infarction (MI)? 0 = no | 1 = yes)
9. PhysActivity (Physical activity in the past 30 days? 0 = no | 1 = yes)
10. Fruits (Consume Fruit 1 or more times per day? 0 = no | 1 = yes)
11. Veggies (Consume Vegetables 1 or more times per day? 0 = no | 1 = yes)
12. HvyAlcoholConsump (adult men >=14 drinks per week | adult women>=7 drinks per week? 0 = no | 1 = yes)
13. AnyHealthcare (Have any kind of health care coverage, including health insurance, prepaid plans such as HMO? 0 = no | 1 = yes)
14. NoDocbcCost (Was there a time in the past 12 months when you needed to see a doctor but could not because of cost? 0 = no | 1 = yes)
15. GenHlth (Would you say that in general your health is? 1 = excellent | 2 = very good | 3 = good | 4 = fair | 5 = poor)
16. MentHlth (Days of poor mental health? Scale of 1-30 days)
17. PhysHlth (Physical illness or injury days in the past 30 days? Scale of 1-30 days)
18. DiffWalk (Do you have serious difficulty walking or climbing stairs? 0 = no | 1 = yes)
19. Sex (0 = female | 1 = male)
20. Age (refer to codebook)

21. Education (refer to codebook)
22. Income (refer to codebook)

Exploratory Data Analysis

Outliers

Used the Boxplot method to visualise which dimensions have outliers in them. The dimensions with multiple outliers are “BMI”, “MentHlth” and “PhysHlth” respectively. “BMI” was found to have the greatest and most extreme outliers hence we proceeded to remove the outliers and conduct Decision Tree analysis. Comparing the Decision Tree based on the dataset with outliers removed, with the Decision Tree based on the original dataset, we found that removing the outliers **worsens the accuracy** instead. As a result, we decided that the outliers are in fact important data that contributes to predicting “diabetes_binary”.

We concluded that a reason for the high importance of the “BMI” outliers could be due to the high correlation between “BMI” and “Diabetes_binary”. Those with high BMI are likely to have diabetes so it would be unfeasible to remove these values and reduce the performance of our models.

```
df.boxplot("BMI")
```

```
<AxesSubplot:>
```

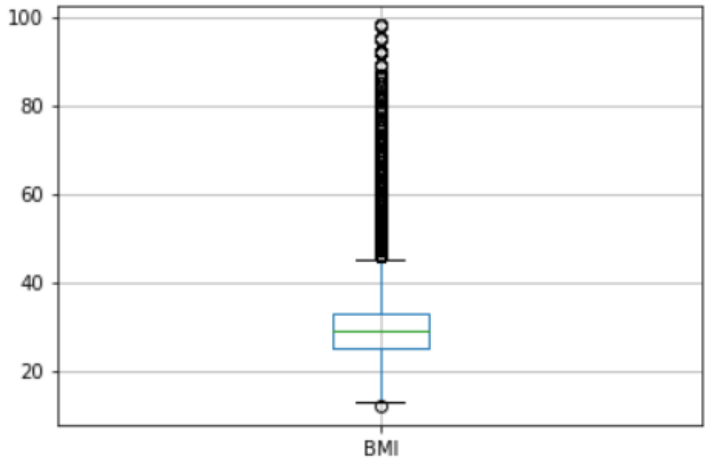


Fig 1. Boxplot diagram based on variable ‘BMI’ to display outliers

```
df.boxplot("MentHlth")
```

```
<AxesSubplot:>
```

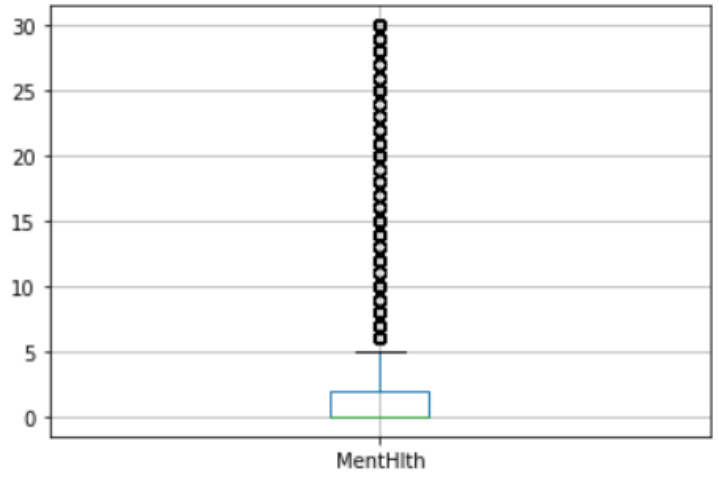


Fig 2. Boxplot diagram based on variable 'MentHlth' to display outliers

```
df.boxplot("PhysHlth")
```

```
<AxesSubplot:>
```

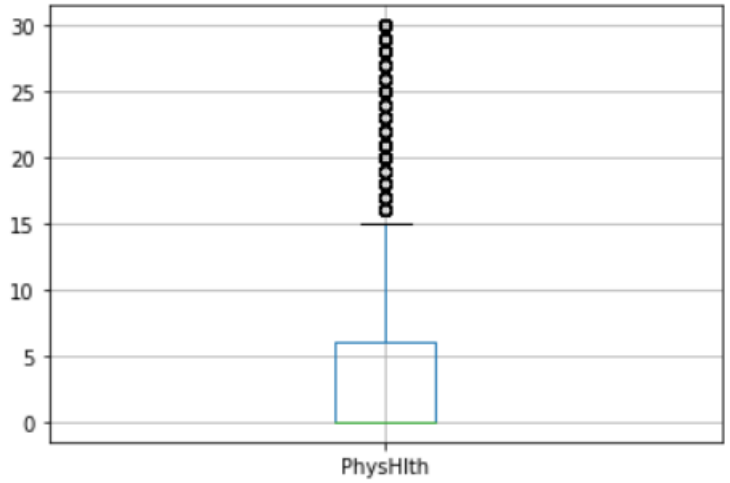


Fig 3. Boxplot diagram based on variable 'PhysHlth' to display outliers

Binning

To reduce data complexity, we decided to perform binning on variables. This would make it easier for us to understand and also reduce the impact of outliers by grouping data points that are close together into the same bin. Hence, the data can become more robust and less sensitive to fluctuations.

Upon analysing the data types within our dataset, we found that only 1 variable in our data set is continuous ("BMI"). In order to **convert "BMI" into ordinal data**, we decided to use Binning according to the different ranges of BMI. The binning criteria we used are as follows:

- Bin 0: If BMI is less than 18.5, it falls within the Underweight range.
- Bin 1: If BMI is 18.5 to 24.9, it falls within the Healthy Weight range.
- Bin 2: If BMI is 25.0 to 29.9, it falls within the Overweight range.
- Bin 3: If BMI is 30.0 or higher, it falls within the Obese range.

Correlation

To find dimensions with **greater significance**, we calculated the correlation coefficient of all the attributes and focused on dimensions with coefficient **greater than 0.25.** We plotted a heatmap to illustrate the correlation between each of the variables to the target variable ("Diabetes_binary") as seen in the figure below.

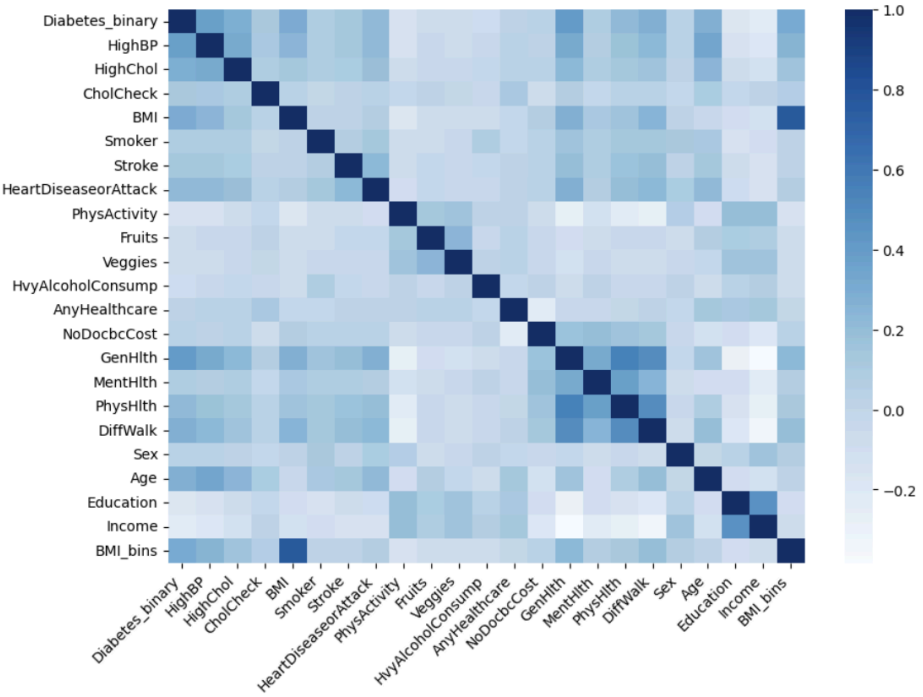


Fig 4. SNS Heatmap of Correlation Coefficient with Target Variable

From the correlation coefficients, we found the top 6 correlated variables to be: HighBP, HighChol, GenHlth, DiffWalk, Age and BMI_bins.

Covariance Matrix

Based on the covariance matrix, the indicators with the highest covariances are **HighBP, HighChol** and **BMI** with the values of 0.381516, 0.289213 and 0.293373 respectively.

Mutual Information Feature Selection

Mutual information from the field of information theory is the application of information gain to feature selection. The output of the Mutual Information Feature Selection is as shown in the figure below. The top 6 features selected are the same as that of the features from Correlation Coefficient, thus affirming the choice of features to conduct our evaluation on.

```
Feature 13: 0.094776
Feature 0: 0.079139
Feature 3: 0.054909
Feature 21: 0.051729
Feature 18: 0.046085
Feature 1: 0.045939
Feature 16: 0.038542
Feature 20: 0.031316
Feature 15: 0.023861
Feature 6: 0.020263
Feature 7: 0.017344
Feature 19: 0.014312
Feature 2: 0.010560
Feature 9: 0.009437
Feature 4: 0.008899
Feature 5: 0.008528
Feature 11: 0.008415
Feature 8: 0.007854
Feature 17: 0.004060
Feature 10: 0.003677
Feature 14: 0.001455
Feature 12: 0.000000
```

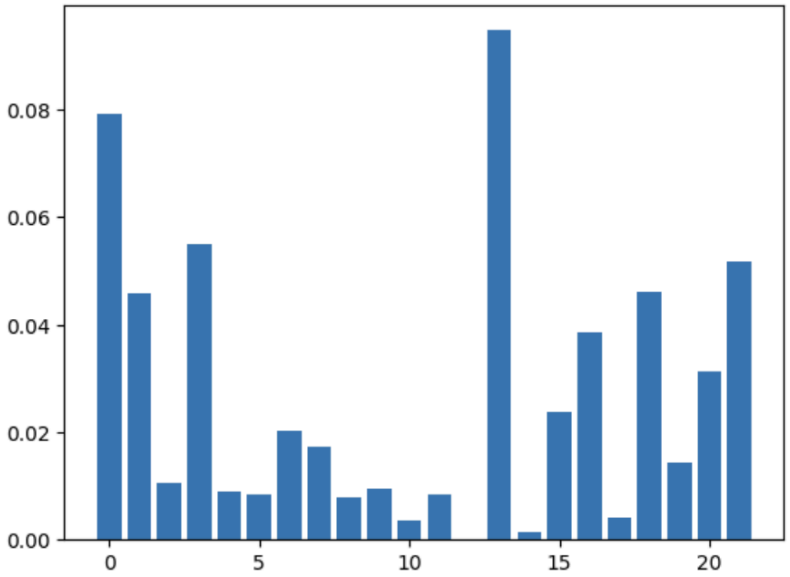


Fig 5. Visualisation of Mutual Information Score for the features

Dataset Splitting

We decided to split the dataset into 13% validation, 4% unseen and 83% training and test data. This is done via train test split.

Stratified k-Fold Train Test Split

The train test data is used to train and evaluate the model's performance. Helps to preserve the distribution and proportion of classes in the dataset and reduce bias and variance. It also provides a more robust estimate of the model's performance.

Validation Set

Validation datasets are used to evaluate the performance of a model after training. The model is evaluated on this data to check how well it is generalising to new data and to make decisions about model hyperparameters, such as the number of hidden layers in a neural network or n estimators in our boosting models

The validation data is used for the optimization of the models

Unseen Set

Unseen dataset is the subset of data that the model has never seen before. This is the data that the model will encounter in the real world when making predictions. The performance of the model on unseen data is the measure of how well it will generalise to new, real-world situations.

The unseen data is used to assess the generalizability of the models.

5. Methodology

Feature Selection Methods

Correlation Coefficient

Top 3 dimensions correlated to diabetes:

1. GenHlth: 40.761
2. HighBP: 38.151
3. BMI: 29.337

Recursive Feature Elimination Cross Validation (RFECV)

It is a feature selection method that can be used to improve the accuracy of machine learning models. In the context of diabetes prediction, researchers have proposed using RFECV to improve the classification accuracy of Type-II diabetes prediction (Misra, 2020). By implementing RFECV on a dataset, the most critical features can be identified and used to improve the accuracy of the model.

Recursive Feature Elimination

It is a feature selection method that works by recursively removing features and building a model on the remaining features. It uses an external estimator that assigns weights to features (e.g., the coefficients of a linear model) to select features by recursively considering smaller and smaller sets of features. The goal of RFE is to **select the most relevant features** for use in a machine learning model, in our case, predicting diabetes.

Evaluation of Model

i. Sensitivity

   Formula: TP / (TP + FN)

   In the context of a model predicting diabetes, it is a measure of how often the model correctly predicts positive cases out of all the actual positive cases.

   Sensitivity is the most important metric as we can decrease the likelihood of overlooking positive predictions for diabetes, which could have significant health consequences for patients if left undiagnosed in the early stages.

ii. Accuracy

Formula: (TP +TN) / (TP + TN + FP + FN)

Measure of how accurate the model correctly predicts true cases out of all the cases

iii. Precision

Formula: TP / (TP + FP)

Measure of how precise the model correctly predicts true positive cases out of all the positive cases.

iv. Specificity

Formula: TN / (TN + FP)

In the context of a model predicting diabetes, it is a measure of how often the model correctly predicts negative cases (i.e. individuals without diabetes) out of all the actual negative cases.

v. Classification Error

Formula: (FP+FN)/(TP+TN+FP+FN) = 1- Accuracy

In the context of a model predicting diabetes, classification error occurs when the model misclassified an individual as either having diabetes when they do not (a false positive) or not having diabetes when they do (a false negative).

vi. F1 Score

Formula: (2 * sensitivity * precision) / (sensitivity + precision)

A combined measure that is derived from Precision (P) and Recall (R) is the F measure. Measures a model's accuracy and computes the number of times a model correctly predicted the entire dataset

vii. Area under ROC curve (receiver operating characteristic curve)

ROC is a probability curve and AUC represents the measure of separability. It tells how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. The ROC curve is plotted with True Positive Rate (Sensitivity) against the False Positive Rate(1 - Specificity) where TPR is on the y-axis and FPR is on the x-axis. This can be seen in the figure below.

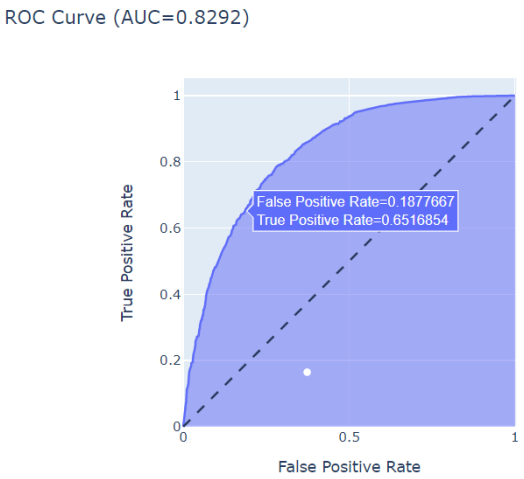


Fig 6. Example ROC Curve to illustrate Area under ROC metric

Parameter Tuning

To identify best hyperparameters for a defined model to predict diabetes

- Best features are selected based on the respective models
- X and y are created to contain the selected features and target variables, respectively, from a validation dataset
- A dictionary of hyperparameters is defined to be tuned to the respective models
- Randomised Search Cross-Validation (RSCV) object is created to find the best set of hyperparameters
- Best hyperparameters and corresponding score are printed

RSCV explores a subset of possible hyperparameters. The best parameters found by RSCV can be used to train the final model with optimal hyperparameters

6. Results and Discussion

In this section, the different models are trained based on the features found to have a correlation coefficient of more than 0.25 and evaluated using stratified k-fold train test split to find the average results of each fold. The models are also assessed using Recursive Feature Elimination Cross Validation (RFECV) to find optimal features and using Recursive Feature Elimination, the weakest features are removed until the specified number of features is reached. This is followed by hyperparameter tuning to find the best parameters, with which, the best performing model's results are displayed below.

6.1. Logistic Regression

6.1.1. Correlation Coefficient Features

Variables Used : Top 6 variables based on Correlation Coefficient

['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity and F1 score) for each fold. After appending each metric to a list and computed the average, the results are as follows:

```
print('The average accuracy is:', statistics.mean(k_fold_accuracy))
print('The average classification error is:', statistics.mean(k_fold_classification_error))
print('The average sensitivity is:', statistics.mean(k_fold_sensitivity))
print('The average precision is:', statistics.mean(k_fold_precision))
print('The average specificity is:', statistics.mean(k_fold_specificity))
print('The average f1 score is:', statistics.mean(k_fold_f1_score))
```

```
The average accuracy is: 0.73973
The average classification error is: 0.26027
The average sensitivity is: 0.75227
The average precision is: 0.7259599999999999
The average specificity is: 0.7277
The average f1 score is: 0.73885
```

Fig 7. Correlation Coefficient Feature Results

6.1.2. RFECV

a) Train the model on the entire feature set, create a LogisticRegression model, fit into RFECV and get the optimal features

```python
from sklearn.feature_selection import RFECV

X = train_test_data[['HighBP', 'HighChol', 'CholCheck', 'Smoker',
       'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income'
       'BMI_bins']]
y = train_test_data["Diabetes_binary"]

# Create a LogisticRegression model
model = LogisticRegression()

rfecv = RFECV(model, cv = 5, scoring='f1_weighted')
rfecv = rfecv.fit(X,y)

print("Feature ranking: ", rfecv.ranking_)

# Get the names of the optimal features
RFECV_selected = []
for index in range(len(X.columns)):
    if rfecv.ranking_[index] == 1:
        RFECV_selected.append(X.columns[index])

RFECV_selected
```

Fig 8. Code of RFECV for Logistic Regression

b) Selecting the features that have been ranked 1

```
In [26]: selected_features_indices = np.where(rfecv.support_ == True)[0]
         selected_features = X.columns[selected_features_indices]
         print("Selected features:", selected_features)

         Selected features: Index(['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Strok
         e',
                'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
                'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
                'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
                'BMI_bins'],
               dtype='object')
```

Fig 9. Features ranked 1 according to RFECV

c) Plotting the Mean Test Accuracy against the number of features selected

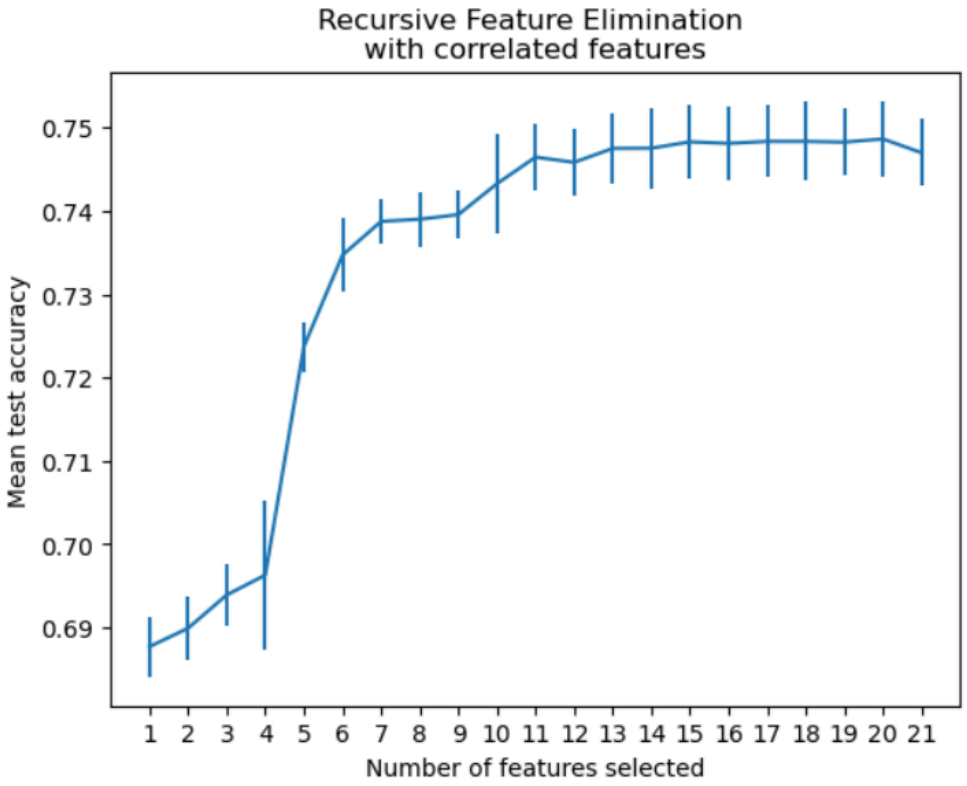


Fig 10. Mean Test Accuracy against the Number of Features Selected

RFECV Evaluation

Based on RFECV, the following variables have all been ranked 1. ['HighBP','HighChol','CholCheck','Smoker','Stroke','HeartDiseaseorAttack','PhysActivity','Fruits','Veggies','HvyAlcoholConsump','AnyHealthcare','NoDocbcCost','GenHlth','PhysHlth','DiffWalk','Sex', 'Age','Education', 'Income','BMI_bins']

But after plotting the mean accuracy against the number of features selected, we can see that the optimal number of features is 6. However, this does not tell us which 6 features have been selected

6.1.3. RFE

Afterwards, we leveraged RFE to remove the weakest features until the specified number of features was reached. Using numbers ranging from 4 to 9 to specify the number of features to select for each iteration, we gathered different sets of features in order to pick the optimal number of features to perform on my model evaluation later.

```
# Create a Logistic Regression model
model = LogisticRegression()
```

```
# Create an RFE object with the Logistic Regression model and number of features to select
rfe = RFE(model, n_features_to_select=n)

# Fit the RFE object on your dataset
fit = rfe.fit(X, y)

# Get the selected features
selected_features = X.columns[rfe.support_]

print("Num Features: %d" % rfe.n_features_)
print("Selected Features: %s" % rfe.support_)
print("Selected Features: %s" % selected_features)
print("Feature Ranking: %s" % fit.ranking_)
```

In terms of sensitivity, the optimal number of features is **7** as shown in the table below with the following variables: ['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']

Table 2: Logistic Regression Model Evaluation Results with varying no. of Features

No. of Features	**Features**	**Sensitivity**
From Correlation Coefficient	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']	0.75227
4	['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump']	0.74374
5	['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump', 'GenHlth']	0.73697
6	['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump', 'GenHlth'', 'BMI_bins']	0.75932
7	**['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']**	**0.76569**
9	['HighBP', 'HighChol', 'CholCheck', 'Stroke', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'DiffWalk', 'BMI_bins']	0.7549

6.1.4. Hyper Parameter Tuning

We then proceeded to do hyper parameter tuning to find a set of optimal hyper parameters values for my model using the 7 features identified by RFE earlier on.

```
# Best Features
features = ['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump',
'GenHlth', 'BMI_bins']
X = val_data[features]
y = val_data["Diabetes_binary"]

# Define the hyperparameter space
params = {
    'penalty': ['l1', 'l2'],
    'C': uniform(loc=0, scale=4),
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [100, 500, 1000],
}
# Create a Logistic Regression Classifier model
model = LogisticRegressionr()

# Define the Random Search CV object
random_search = RandomizedSearchCV(
    model,
    param_distributions=params,
    n_iter=40,
    cv=5,
    random_state=424
)
# Fit the Random Search CV object to the data
random_search.fit(X, y)

# Print the best hyperparameters and the corresponding score
print('Best Hyperparameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)
```

Results:

```
Best Hyperparameters: {'C': 2.743430272273789, 'max_iter': 1000, 'penalty':
'l1', 'solver': 'saga'}
Best Score: 0.7306855277475517
```

Fig 11. Results for hyperparameter tuning

6.1.5. Model Evaluation

Last but not least, we evaluated the model using the 7 best features selected by RFE and best parameters by Random Search CV and calculated the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score).

```
print('The accuracy on unseen data is: ', accuracy)
print('The classification_error on unseen data is: ', classification_error)
print('The sensitivity on unseen data is: ', sensitivity)
print('The precision on unseen data is: ', precision)
print('The specificity on unseen data is: ', specificity)
print('The f1_score on unseen data is: ', f1_score)
```

```
The accuracy on unseen data is:  0.7325088339222615
The classification_error on unseen data is:  0.2674911660777385
The sensitivity on unseen data is:  0.7429775280898876
The precision on unseen data is:  0.7301587301587301
The specificity on unseen data is:  0.7219061166429588
The f1_score on unseen data is:  0.7365123564218585
```

Fig 12. Final Model Evaluation for Logistic Regression

Based on sensitivity, we conclude that this model gives a 74.29% accuracy in detecting as many positive cases of diabetes as possible which is not considered high enough. This means that there is a 25.71% chance that some people with diabetes will be left out. This could lead to delayed diagnosis and treatment.

6.2. Decision Tree (Gini > Entropy)

Exploration the diff max depths & methods:

1. Default and Entropy shows the exact same results
2. Gini returns a higher sensitivity than Entropy

Entropy Results:

```
The average accuracy is: 0.739
The average classification error is: 0.261
The average sensitivity is: 0.77279
The average precision is: 0.7239599999999999
The average specificity is: 0.70519
The average f1 score is: 0.74752
```

Fig 13. Model Evaluation using Entropy

Gini Results:

```
The average accuracy is: 0.73892
The average classification error is: 0.26108
The average sensitivity is: 0.77282
The average precision is: 0.72381
The average specificity is: 0.70497
The average f1 score is: 0.74748
```

Fig 14. Model Evaluation using Gini

When comparison was done between Entropy and Gini, we see that Gini returns a slightly higher sensitivity which was the top priority taken into consideration for predicting diabetes.

6.2.1. Correlation Coefficient Features

Variables Used : Top 6 variables based on Correlation Coefficient

['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']

6.2.2. RFECV

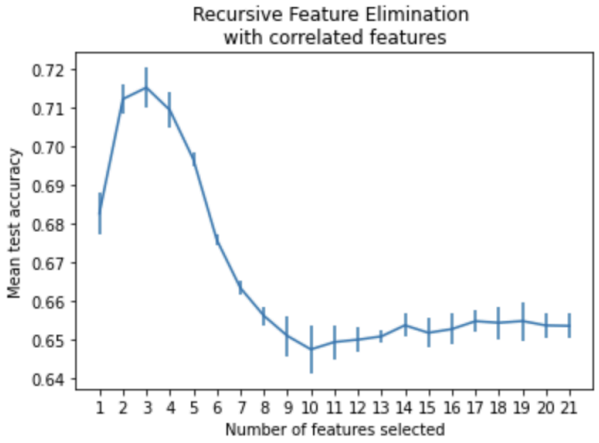


Fig 15. Mean Test Accuracy against the Number of Features Selected

RFECV Evaluation

Based on the Recursive Feature Elimination with Cross-Validation (RFECV), 3 variables were identified to have the highest ranking, there are 'HighBP', 'GenHlth', and ‘'Age' respectively. However, this resulted in too few features being selected, so we decided to increase the number of features by selecting the plateau of the mean test accuracy divided by the number of features selected which is at 5. To identify the optimal number of features, RFE was used to select features ranging from 3 to 7 as well as the 18 top-ranked features from RFECV.

6.2.3. RFE

Table 3: Decision Tree Model Evaluation Results with varying no. of Features

No. of Features	**Features**	**Sensitivity**
From Correlation Coefficient (Gini)	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']	0.77282
3	**['HighBP', 'GenHlth', 'Age']**	**0.80248**
4	['HighBP', 'GenHlth', 'PhysHlth', 'Age']	0.77318
5	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age']	0.74225
6	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Income']	0.67308
7	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Education', 'Income']	0.64431

Optimal number of features for Decision Tree Classifier Gini are 3 with **sensitivity score of 0.80248** using the following features:

['HighBP', 'GenHlth', 'Age',]

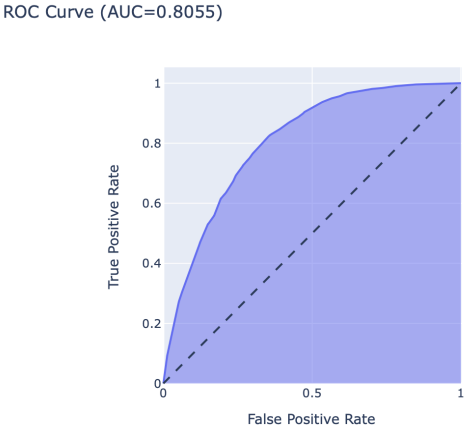
6.2.4. Hyper Parameter Tuning

```
X = val_data[features]
y = val_data["Diabetes_binary"]

# Define the hyperparameter space
params = {
    'max_depth': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 4, 6],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}

# Create a Adaboost Classifier model
model = DecisionTreeClassifier(criterion = 'gini', random_state = 424)
# Define the Random Search CV object
random_search = RandomizedSearchCV(
    model,
    param_distributions=params,
    n_iter=40,
    cv=5,
    random_state=424
)
# Fit the Random Search CV object to the data
random_search.fit(X, y)

# Print the best hyperparameters and the corresponding score
print('Best Hyperparameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)
```

Next, we use hyperparameter tuning to find the optimal parameter for n_estimators and learning rate which we obtained the results of '**min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 6, and 'criterion': 'gini'.**

6.2.5. Model Evaluation

Lastly, we evaluated the model using the unseen datasets and the final result for Decision Tree Classifier Gini is:




```
The accuracy on unseen data is:  0.734982332155477
The classification_error on unseen data is:  0.26501766784452296
The sensitivity on unseen data is:  0.8061797752808989
The precision on unseen data is:  0.7077681874229347
The specificity on unseen data is:  0.662873399715505
The f1_score on unseen data is:  0.7537754432042024
```

Fig 16: Final Model Evaluation for Decision Tree

6.3. Adaptive Boosting

6.3.1 Correlation Coefficient Features

Variables Used : Top 6 variables based on Correlation Coefficient

['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']

6.3.2. RFECV

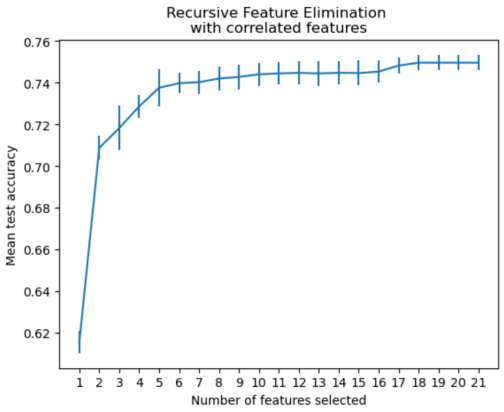


Fig 17: Mean Test Accuracy against the Number of Features Selected

RFECV Evaluation

Based on the Recursive Feature Elimination with Cross-Validation (RFECV), 18 variables were identified to have the highest ranking. However, this resulted in too many features being selected, so the team decided to reduce the number of features by selecting the plateau of the mean test accuracy divided by the number of features selected which is at 5. To identify the optimal number of features, RFE was used to select features ranging from 5 to 9 as well as the 18 top-ranked features from RFECV.

6.3.3. RFE (Decision Tree)

Table 4: Adaboost (Decision Tree) Model Evaluation Results with varying no. of Features

No. of Features	Features	Sensitivity
From Correlation Coefficient	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']	0.76982
5	['HighBP', 'GenHlth', 'Age', 'Income', 'BMI_bins']	0.76516
6	['HighBP', 'GenHlth', 'Age', 'Education', 'Income', 'BMI_bins']	0.76553
7	['HighBP', 'GenHlth', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']	0.77173

8	['HighBP', 'HighChol', 'GenHlth', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']	0.7742
18	**['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Veggies', 'HvyAlcoholConsump', 'GenHlth', 'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']**	**0.77432**

Optimal number of features for Adaboost are 18 with **sensitivity score of 0.77432** using the following features:
['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Veggies', 'HvyAlcoholConsump', 'GenHlth', 'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']

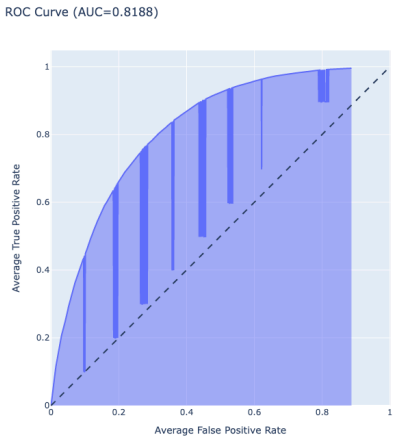
6.3.4. Hyper Parameter Tuning (Decision Tree)

```
X = val_data[features]
y = val_data["Diabetes_binary"]

# Define the hyperparameter space
params = {
    'n_estimators': [50, 100, 150, 200, 250, 300, 350, 400],
    'learning_rate': [0.1,0.2,0.3,0.4,0.5],
}

# Create a Adaboost Classifier model
model = AdaBoostClassifier()
# Define the Random Search CV object
random_search = RandomizedSearchCV(
    model,
    param_distributions=params,
    n_iter=40,
    cv=5,
    random_state=424
)
# Fit the Random Search CV object to the data
random_search.fit(X, y)

# Print the best hyperparameters and the corresponding score
print('Best Hyperparameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)
```

Next, using hyperparameter tuning, we find the optimal parameter for n_estimators and learning rate which we obtained the results of **n_estimators=350 and learning_rate=0.2**

6.3.5. Model Evaluation (Decision Tree)

Lastly, an evaluation of the model using the unseen dataset is conducted and the final result for Adaboost using decision tree as its base estimator is:




```
The accuracy on unseen data is:  0.7501766784452297
The classification_error on unseen data is:  0.24982332155477027
The sensitivity on unseen data is:  0.7591292134831461
The precision on unseen data is:  0.7480968858131488
The specificity on unseen data is:  0.7411095305832148
The f1_score on unseen data is:  0.7535726734053677
```

Fig 18. Final Model Evaluation for Adaboost (Decision Tree)

6.3.6. RFE (Naive Bayes)

Table 5: AdaBoost (Naive Bayes) *Model Evaluation Results with varying no. of Features*

No. of Features	Features	Sensitivity
From Correlation Coefficient	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']	0.76982
5	['HighBP', 'GenHlth', 'Age', 'Income', 'BMI_bins']	0.76907
6	['HighBP', 'GenHlth', 'Age', 'Education', 'Income', 'BMI_bins']	0.76982
7	['HighBP', 'GenHlth', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']	0.77155
8	**['HighBP', 'HighChol', 'GenHlth', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']**	**0.77495**
18	['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Veggies', 'HvyAlcoholConsump', 'GenHlth', 'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']	0.74706

Optimal number of features for Adaboost is 8 with sensitivity score of Sensitivity score = 0.77495 using the following features:

['HighBP', 'HighChol', 'GenHlth', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']

6.3.7. Hyper Parameter Tuning (Naive Bayes)

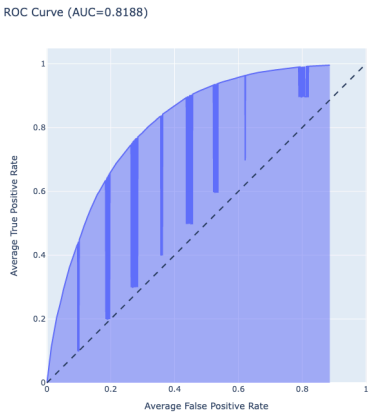
```
X = val_data[features]
y = val_data["Diabetes_binary"]
```

```
# Define the hyperparameter space
params = {
    'n_estimators': [50, 100, 150, 200, 250, 300, 350, 400],
    'learning_rate': [0.1,0.2,0.3,0.4,0.5],
}

# Create a Adaboost Classifier model
model = AdaBoostClassifier()
# Define the Random Search CV object
random_search = RandomizedSearchCV(
    model,
    param_distributions=params,
    n_iter=40,
    cv=5,
    random_state=424
)
# Fit the Random Search CV object to the data
random_search.fit(X, y)

# Print the best hyperparameters and the corresponding score
print('Best Hyperparameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)
```

Next, we use hyperparameter tuning to find the optimal parameter for n_estimators and learning rate which we obtained the results of **n_estimators=350 and learning_rate=0.2**

6.3.8. Model Evaluation (Naive Bayes)

Lastly, we evaluated the model using the unseen datasets and the final result for Adaboost using decision tree as its base estimator is:




```
The accuracy on unseen data is:  0.7501766784452297
The classification_error on unseen data is:  0.24982332155477027
The sensitivity on unseen data is:  0.7591292134831461
The precision on unseen data is:  0.7480968858131488
The specificity on unseen data is:  0.7411095305832148
The f1_score on unseen data is:  0.7535726734053677
```

Fig 19. Final Model Evaluation for Adaboost (Naive Bayes)

6.4. XGBoost

6.4.1 Correlation Coefficient Features

Variables Used : Top 6 variables based on Correlation Cofficient

['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity and F1 score) for each fold. After appending each metric to a list and computed the average, the results are as follows:

```
print('The average accuracy is:', statistics.mean(k_fold_accuracy))
print('The average classification error is:', statistics.mean(k_fold_classification_error))
print('The average sensitivity is:', statistics.mean(k_fold_sensitivity))
print('The average precision is:', statistics.mean(k_fold_precision))
print('The average specificity is:', statistics.mean(k_fold_specificity))
print('The average f1 score is:', statistics.mean(k_fold_f1_score))
```

```
The average accuracy is: 0.74214
The average classification error is: 0.25786
The average sensitivity is: 0.78578
The average precision is: 0.72277
The average specificity is: 0.69848
The average f1 score is: 0.75292
```

Fig 20. XGBoost Correlation Coefficient Results

6.4.2 RFECV

a) Train the model on the entire feature set, create a XGBoost model, then fit it into RFECV and get the names of the optimal features

```
from sklearn.feature_selection import RFECV

X = train_test_data[['HighBP', 'HighChol', 'CholCheck', 'Smoker',
       'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'BMI_bins']]
y = train_test_data["Diabetes_binary"]

# Create a XGBoost classifier model
model = XGBClassifier()

# Create an RFE object with the XGBoost model and number of features to select
rfecv = RFECV(model, cv=5, scoring='f1_weighted')

# Fit the RFE object on your dataset
rfecv.fit(X, y)

print("Feature ranking: ", rfecv.ranking_)

# Get the names of the optimal features
RFECV_selected = []
for index in range(len(X.columns)):
    if rfecv.ranking_[index] == 1:
        RFECV_selected.append(X.columns[index])

RFECV_selected
```

Fig 21. XGBoost RFECV Results

b) Selecting the features that have been ranked 1

```
selected_features_indices = np.where(rfecv.support_ == True)[0]
selected_features = X.columns[selected_features_indices]
print("Selected features:", selected_features)
```

```
Selected features: Index(['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack',
       'HvyAlcoholConsump', 'GenHlth', 'DiffWalk', 'Sex', 'Age', 'Income',
       'BMI_bins'],
      dtype='object')
```

Fig 22. XGBoost RFECV Ranking Results

c) Plotting the Mean Test Accuracy against the number of features selected

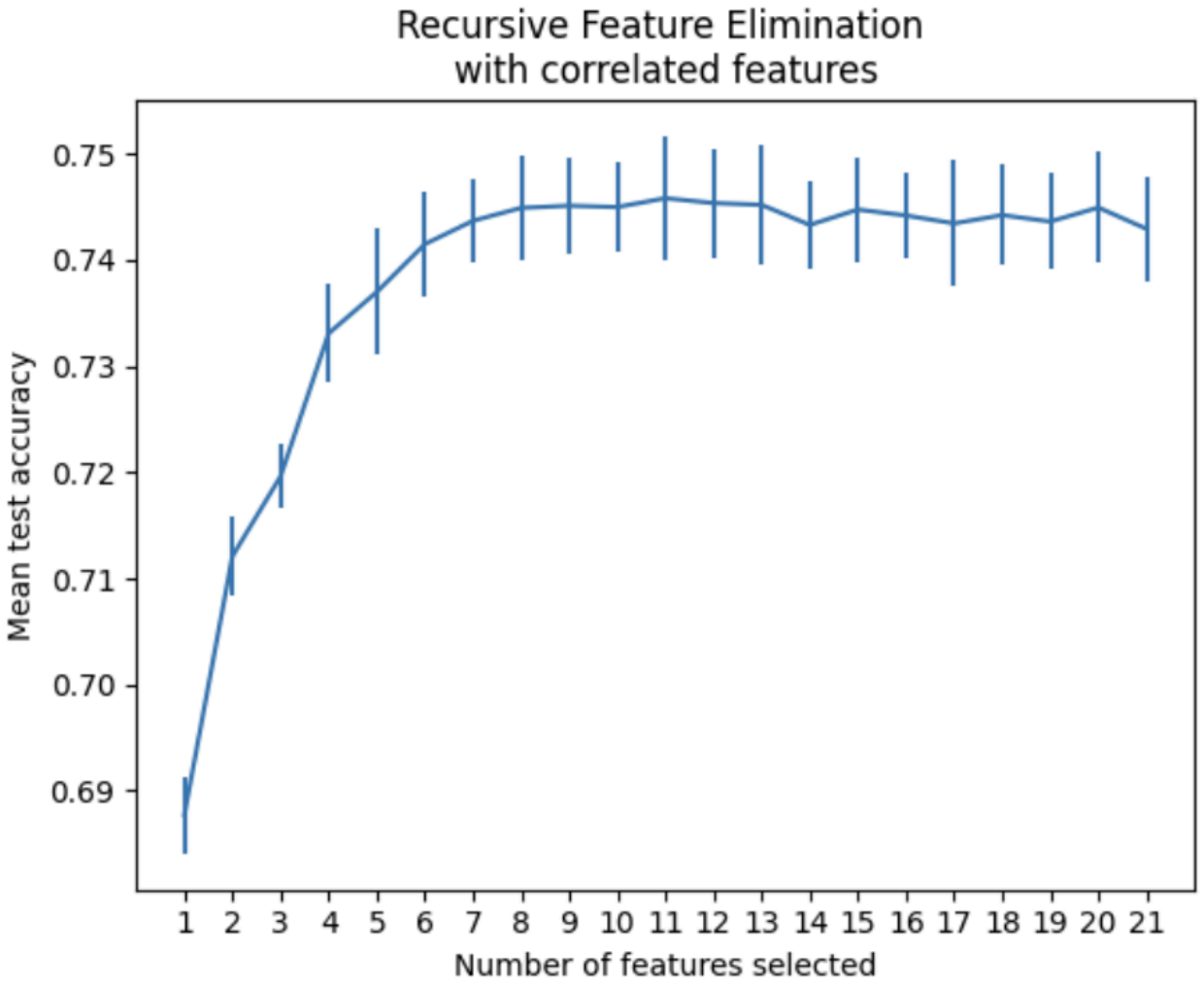


Fig 23. XGBoost RFECV with Correlated Features Graph

RFECV Evaluation

Based on RFECV, the following variables have all been ranked 1. ['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'DiffWalk', 'Sex', 'Age', 'Income', 'BMI_bins']

After plotting the mean accuracy against the number of features selected, we can see that the optimal number of features is 6 from the plateau in the graph. However, this does not tell us which 6 features have been selected.

6.4.3 RFE

We then leveraged RFE to remove the weakest features until the specified number of features was reached. Using numbers ranging from 4 to 9 to specify the number of features to select for each iteration, we gathered different sets of features in order to pick the optimal number of features to perform on my model evaluation later.

```
# Create a XGBoost classifier model
model = XGBClassifier()

# Create an RFE object with the XGBoost model and number of features to select
rfe = RFE(model, n_features_to_select=n)

# Fit the RFE object on your dataset
fit = rfe.fit(X, y)

# Get the selected features
selected_features = X.columns[rfe.support_]

print("Num Features: %d" % rfe.n_features_)
print("Selected Features: %s" % rfe.support_)
print("Selected Features: %s" % selected_features)
print("Feature Ranking: %s" % fit.ranking_)
```

Comparing sensitivity, the optimal number of features is **6** as shown in the table below with the following variables: ['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']

Table 6: XGBoost Model Evaluation Results with varying no. of Features

No. of Features	**Features**	**Sensitivity**
From Correlation Coefficient	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']	0.78578
4	['HighBP', 'HighChol', 'GenHlth', 'BMI_bins']	0.74167
5	['HighBP', 'HighChol', 'GenHlth', 'Age', 'BMI_bins']	0.78171
6	**['HighBP', 'HighChol', 'CholCheck', 'GenHlth', 'Age', 'BMI_bins']**	**0.78633**
7	['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'GenHlth', 'Age', 'BMI_bins']	0.78273
9	['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'DiffWalk', 'Age', 'BMI_bins']	0.78626

6.4.4 Hyper Parameter Tuning

Using the 6 features identified by RFE earlier on, Hyperparameter tuning was performed to find the optimal parameters for the XGBoost model.

```
# Best Features
features = ['HighBP', 'HighChol', 'CholCheck', 'GenHlth', 'Age', 'BMI_bins']
X = val_data[features]
y = val_data["Diabetes_binary"]
```

```
# Define the hyperparameter space
params = {
    'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'n_estimators': [50, 100, 150, 200, 250, 300, 350, 400],
}
# Create a XGBoost Classifier model
model = XGBoostClassifier()

# Define the Random Search CV object
random_search = RandomizedSearchCV(
    model,
    param_distributions=params,
    n_iter=40,
    cv=5,
    random_state=424
)
# Fit the Random Search CV object to the data
random_search.fit(X, y)

# Print the best hyperparameters and the corresponding score
print('Best Hyperparameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)
```

Results:

```
Best Hyperparameters: {'n_estimators': 50, 'max_depth': 4}
Best Score: 0.7326441784548423
```

6.4.5 Model Evaluation

To evaluate the model's best performance, we used the 6 best features selected by RFE and the best parameters by Random Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

```
print('The accuracy on unseen data is: ', accuracy)
print('The classification_error on unseen data is: ', classification_error)
print('The sensitivity on unseen data is: ', sensitivity)
print('The precision on unseen data is: ', precision)
print('The specificity on unseen data is: ', specificity)
print('The f1_score on unseen data is: ', f1_score)
```

```
The accuracy on unseen data is:  0.7491166077738516
The classification_error on unseen data is:  0.2508833922261484
The sensitivity on unseen data is:  0.7844101123595506
The precision on unseen data is:  0.7348684210526316
The specificity on unseen data is:  0.713371266002845
The f1_score on unseen data is:  0.7588315217391304
```

Fig 24. Final Model Evaluation for XGBoost

Based on sensitivity, we conclude that this model gives a 78.44% accuracy in detecting as many positive cases of diabetes as possible.

6.5. CatBoost

6.5.1. Correlation Coefficient Features

Variables Used : Top 6 variables based on Correlation Coefficient

['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity and F1 score) for each fold. After appending each metric to a list and computed the average, the results are as follows:

```
print('The average accuracy is:', statistics.mean(k_fold_accuracy))
print('The average classification error is:', statistics.mean(k_fold_classification_error))
print('The average sensitivity is:', statistics.mean(k_fold_sensitivity))
print('The average precision is:', statistics.mean(k_fold_precision))
print('The average specificity is:', statistics.mean(k_fold_specificity))
print('The average f1 score is:', statistics.mean(k_fold_f1_score))

```

```
The average accuracy is: 0.74263
The average classification error is: 0.25737
The average sensitivity is: 0.78501
The average precision is: 0.72378
The average specificity is: 0.70024
The average f1 score is: 0.75312
```

Fig 25. CatBoost Correlation Coefficient Results

6.5.2. RFECV

a) Train the model on the entire feature set, create a CatBoost model, then fit it into RFECV and get the names of the optimal features

```
from sklearn.feature_selection import RFECV

X = train_test_data[['HighBP', 'HighChol', 'CholCheck', 'Smoker',
       'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'BMI_bins']]
y = train_test_data["Diabetes_binary"]

# Create a CatBoost classifier model
model = CatBoostClassifier()

# Create an RFE object with the CatBoost model and number of features to select
rfecv = RFECV(model, cv=5, scoring='f1_weighted')

# Fit the RFE object on your dataset
rfecv.fit(X, y)

print("Feature ranking: ", rfecv.ranking_)

# Get the names of the optimal features
RFECV_selected = []
for index in range(len(X.columns)):
    if rfecv.ranking_[index] == 1:
        RFECV_selected.append(X.columns[index])

RFECV_selected
```

Fig 26. CatBoost RFECV Results

b) Selecting the features that have been ranked 1

```
selected_features_indices = np.where(rfecv.support_ == True)[0]
selected_features = X.columns[selected_features_indices]
print("Selected features:", selected_features)
```

```
Selected features: Index(['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke',
       'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'BMI_bins'],
      dtype='object')
```

Fig 27. CatBoost RFECV Ranking Results

c) Plotting the Mean Test Accuracy against the number of features selected

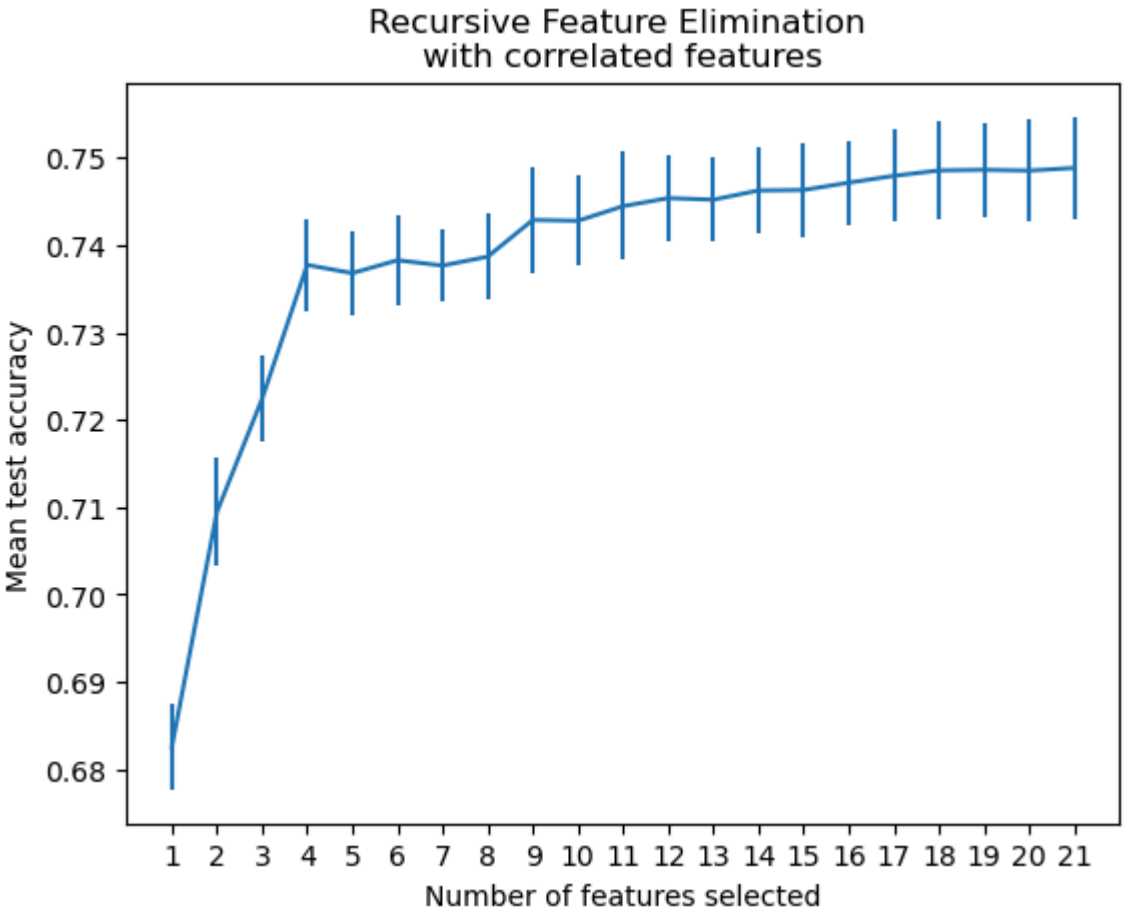


Fig 28. CatBoost RFECV with Correlated Features Graph

RFECV Evaluation

Based on RFECV, All 21 features were ranked one. Using all 21 features (High dimension) may lead to the curse of dimensionality. As such, we plotted the mean accuracy against the number of features selected to identify the optimal number of features (where it plateaus).

After plotting the mean accuracy against the number of features selected, we can see that the optimal number of features is 4 from the plateau in the graph. However, this does not tell us which 6 features have been selected.

6.5.3. RFE

We then leveraged RFE to remove the weakest features until the specified number of features was reached. Using numbers ranging from 4 to 9 to specify the number of features to select for each iteration, we gathered different sets of features in order to pick the optimal number of features to perform on my model evaluation later.

```
# Create a CatBoost classifier model
model = CatBoostClassifier()

# Create an RFE object with the CatBoost model and number of features to select
rfe = RFE(model, n_features_to_select=4)

# Fit the RFE object on your dataset
fit = rfe.fit(X, y)

# Get the selected features
selected_features = X.columns[rfe.support_]

print("Num Features: %d" % rfe.n_features_)
print("Selected Features: %s" % rfe.support_)
print("Selected Features: %s" % selected_features)
print("Feature Ranking: %s" % fit.ranking_)
```

Table 7 : CatBoost Model Evaluation Results with varying no. of Features from RFE

No. of Features	Features	Sensitivity
4	['HighBP', 'GenHlth', 'Age', 'BMI_bins']	0.78803
5	['HighBP', 'GenHlth', 'PhysHlth', 'Age', 'BMI_bins']	0.78864
6	['HighBP', 'GenHlth', 'PhysHlth', 'Age', 'Income', 'BMI_bins']	0.78642
7	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Income','BMI_bins']	0.78481
9	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Income','BMI_bins','HighChol,'Education]	0.78926

In Number of features 9, features 'GenHlth', 'MentHlth' and 'PhysHlth' were present. Since all three were related to each other in terms of health, we decided to remove some of the features and run the model on the following set of features below.

Table 8: CatBoost Model Evaluation Results with other set of variables

No. of Features	Features	Sensitivity
Only 'GenHlth'	['HighBP', 'GenHlth', 'Age', 'Income', 'BMI_bins', 'HighChol, 'Education]	0.79015
Remove Health Conditions	['HighBP', 'Age', 'Income', 'BMI_bins', 'HighChol, 'Education]	0.77819

From Correlation Coefficient	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age', 'BMI_bins']	0.78501

After running the models, we identified ['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age', 'BMI_bins'] from correlation coefficient to be the best set of variables. Despite it not having the highest sensitivity score, there was not much difference between the other set of variables. Furthermore, the 6 features performed significantly better in the other metrics as well (in appendix).

6.5.4. Hyper Parameter Tuning

Using the 6 features identified by RFE earlier on, Hyperparameter tuning was performed to find the optimal parameters for the XGBoost model.

```
# Best Features
features =['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins'] #input best features
X = val_data[features]
y = val_data["Diabetes_binary"]

# Define the hyperparameter space
params = {
    'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'n_estimators': [50, 100, 150, 200, 250, 300, 350, 400],
    'l2_leaf_reg': uniform(0.1, 1.0),
}

# Create a CatBoost Classifier model
model = CatBoostClassifier()

# Define the Random Search CV object
random_search = RandomizedSearchCV(
    model,
    param_distributions=params,
    n_iter=40,
    cv=5,
    random_state=424
)

# Fit the Random Search CV object to the data
random_search.fit(X, y)

# Print the best hyperparameters and the corresponding score
print('Best Hyperparameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)
```

Results:

```
Best Hyperparameters: {'l2_leaf_reg': 0.17865843160768383, 'max_depth': 4, 'n_estimators': 200}
Best Score: 0.7387377584330794
```

Fig 29. Results for hyperparameter tuning

6.5.5. Model Evaluation

To evaluate the model's best performance, we used the 6 best features and the best parameters by Random Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

```
print('The accuracy on unseen data is: ', accuracy)
print('The classification_error on unseen data is: ', classification_error)
print('The sensitivity on unseen data is: ', sensitivity)
print('The precision on unseen data is: ', precision)
print('The specificity on unseen data is: ', specificity)
print('The f1_score on unseen data is: ', f1_score)
```

```
The accuracy on unseen data is:  0.7515901060070671
The classification_error on unseen data is:  0.2484098939929329
The sensitivity on unseen data is:  0.7893258426966292
The precision on unseen data is:  0.7360838244924689
The specificity on unseen data is:  0.713371266002845
The f1_score on unseen data is:  0.7617756692646561
```

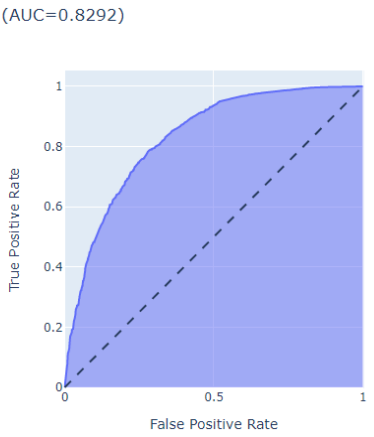


Fig 30. Final Model Evaluation for CatBoost

Based on sensitivity, we conclude that this model gives a 78.93% accuracy in detecting as many positive cases of diabetes as possible.

6.6. Random Forest

Random Forest is a bagging method that has multiple decision trees built on different subsets of the training data. This will help to reduce overfitting and improve the accuracy and robustness of the model. On top of that, it is also less sensitive to the noise and outliers in the data.

6.6.1 Correlation Coefficient Features

Variables Used : Top 6 variables based on Correlation Coefficient

['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']

These are the results that we got from running the model with the top 6 variables.

```
The average accuracy is: 0.73924
The average classification error is: 0.26076
The average sensitivity is: 0.77929
The average precision is: 0.72158
The average specificity is: 0.69916
The average f1 score is: 0.74928
```

Figure 31. Correlation Coefficient Feature Results

6.6.2 RFECV

a) Train the model on the entire feature set, create a Random Forest model and fit into RFECV and get the names of the optimal features

b) Selecting the features that have been ranked 1

```
Selected features: Index(['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke',
       'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'BMI_bins'],
      dtype='object')
```

Figure 32. Features ranked 1 according to RFECV

c) Plotting the Mean Test Accuracy against the number of features selected

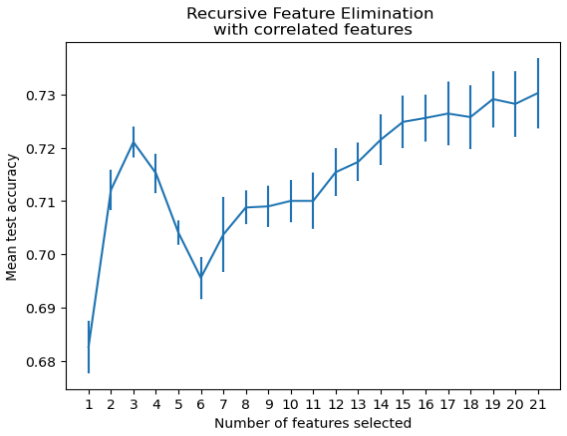


Figure 33. Mean Test Accuracy against the Number of Features Selected

RFECV Evaluation

Based on RFECV, there were 21 variables ranked 1 and this will not be beneficial for us to determine which features are important for this model. However, after plotting the mean accuracy against the number of features selected, we can see that the accuracy drops after 3 features. Hence, we determined that 3 is the optimal number of features for the Random Forest model.

6.6.3 RFE

Table 9: Random Forest Model Evaluation Results with varying no. of Features

No. of Features	**Features**	**Sensitivity**
From Correlation Coefficient	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']	0.77929
3	**['HighBP', 'GenHlth', 'Age']**	**0.80444**
4	['HighBP', 'GenHlth', 'PhysHlth', 'Age']	0.78275
6	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Income']	0.76855
7	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Education', 'Income']	0.72646
Only Gen Health	['HighBP', 'GenHlth', 'Age', 'Income']	0.775
21 (All)	['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth', 'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']	0.76855

We ran the model with varied numbers of features and recorded down the sensitivity for each. We also included 21 features as according to RFECV, all the features are of equal importance. However, you can see that the sensitivity of each model starts to drop as we include more features into the model.

6.6.4 Hyper Parameter Tuning

With the best features selected, we then move on to get the best hyperparameters for this model. For Random Forest, our hyperparameters are: 'n_estimators', 'min_samples_split', 'min_samples_leaf', 'max_depth' and 'bootstrap'.

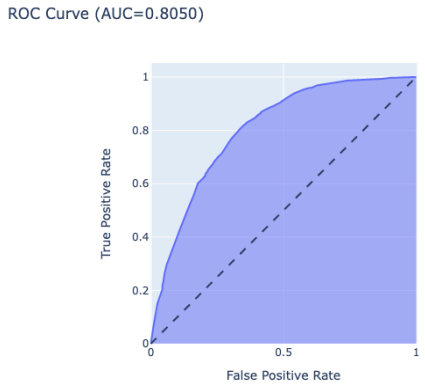
Results:

```
Best Hyperparameters: {'n_estimators': 250, 'min_samples_split': 10, 'min_samples_leaf': 5,
'max_depth': 3, 'bootstrap': False}
Best Score: 0.7171926006528835
```

Fig 34. Results for hyperparameter tuning

6.6.5 Model Evaluation

Using the best features which are: 'HighBP', 'GenHlth', 'Age' and the best parameters we got above, we then ran the model with our unseen data. The results are as follows, with a sensitivity of 0.8013 and an accuracy of 0.735.




```
The accuracy on unseen data is:  0.734982332155477
The classification_error on unseen data is:  0.26501766784452296
The sensitivity on unseen data is:  0.8012640449438202
The precision on unseen data is:  0.7095771144278606
The specificity on unseen data is:  0.6678520625889047
The f1_score on unseen data is:  0.7526385224274406
```

Fig 35. Final Model Evaluation for Logistic Regression

6.7. Artificial Neural Network

The architecture of the ANN involves many parameters such as the number of layers, neurons and the activation function. We used a feedforward neural network using Keras API in python. The code can be explained in the following steps:

Table 10: ANN Code Explanation

Code	**Explanation**
model = Sequential()	This creates a new instance of the Sequential class, which represents a sequence of layers in the neural network.
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))	This adds a Dense layer to the model, with 64 neurons which are linked to the previous sequential layer. The input shape is the number of features in the training data. The activation function is ReLu.
model.add(Dense(32, activation='relu'))	This adds another Dense layer to the model with 32 neurons and a ReLU activation function.
model.add(Dense(1, activation='sigmoid'))	This adds a final Dense layer with a single neuron and a sigmoid activation function. Since this is a binary classification problem (diabetes or no diabetes), the sigmoid function outputs a probability between 0 and 1 indicating the likelihood of a positive class (diabetes).
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])	This compiles the model by specifying the loss function to be optimised, the optimizer to use, and any evaluation metrics to track during training. Here, the loss function is binary cross-entropy, which is commonly used for binary classification problems. The optimizer is Adam, which is an efficient variant of stochastic gradient descent (SGD), and the metric to track is accuracy.
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))	This trains the model on the training data (X_train and y_train) for 100 epochs. Since the batch size is set to 32, the weights of the model are updated every 32 samples. The validation_data parameter is used to evaluate the model's performance on the test set (X_test and y_test) after each epoch. The fit method returns a history object that contains the loss and accuracy values for each epoch.

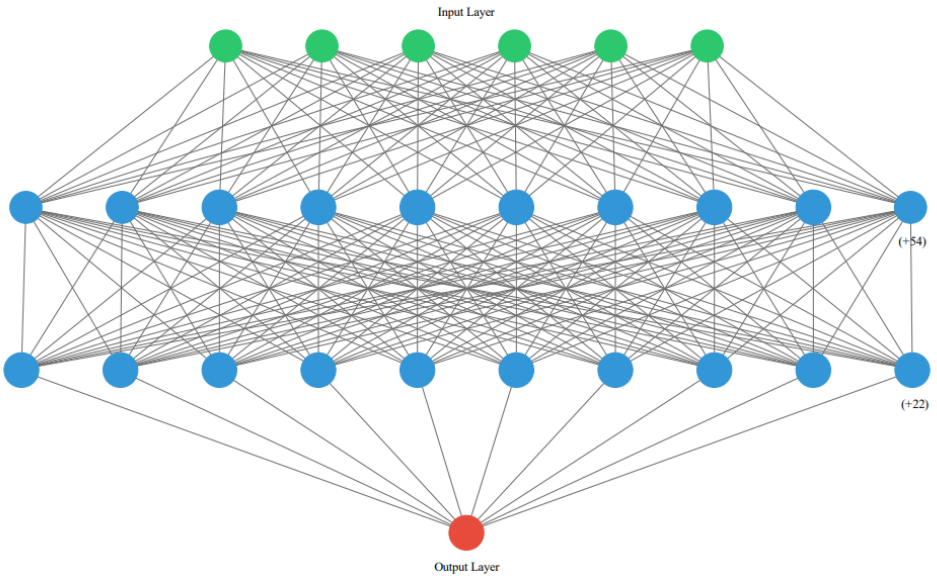


Fig 36. Visualization of ANN model using ann_visualizer library

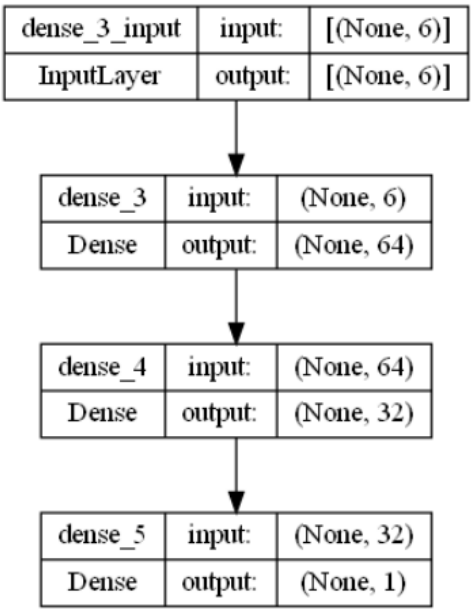


Fig 37. Visualization of ANN model using plot_model library

Since the ANN is incompatible with Stratified k-Fold Train Test Split and Recursive Feature Elimination, the model was tested with different feature sets and parameters to find the optimal model.

6.7.1. Feature Selection

The model's performance was evaluated based on 3 feature sets to decide which features worked best to predict the presence of diabetes.

1. ['HighBP', 'GenHlth', 'Age', 'Income', 'BMI_bins', 'HighChol', 'Education']
   These features were selected based on their correlation coefficient score with the target variable ‘diabetes_binary’.

2. ['HighBP', 'HighChol', 'Age', 'BMI_bins', 'Income', 'Education']
   This feature set was created without health-related features such as GenHlth, PhysHlth and DiffWalk.

3. ['HighBP', 'HighChol', 'Age', 'BMI_bins', 'GenHlth', 'DiffWalk']
   This feature set includes GenHlth and DiffWalk and removes education and income.

Based on each dataset, a neural network was compiled and fitted, to find the set that the model works best on. The sensitivity scores of Set 1, Set 2 and Set 3 are 0.7995, 0.8026 and 0.8029 respectively. Hence, feature set 3 was used for the parameter tuning.

6.7.2. Parameter Tuning

Loss Function

The binary cross-entropy loss function was selected since it was most suitable. The categorical cross-entropy loss function was tested with poor results since it is not appropriate for our dataset.

Number of Epochs

The model’s performance was evaluated based on different numbers of epochs at 50, 100 and 150 to determine the optimal number. The batch_size is constant at 32 for all models.

Table 11: Model Performance for Number of Epochs

Number of Epochs	50	100	150
Performance based on train-test data	Area Under the ROC Curve (AUC-ROC): 0.7397 Accuracy: 0.7392 Classification error: 0.2608 **Sensitivity: 0.8352** Precision: 0.6990 Specificity: 0.6442 F1 score: 0.7610	Area Under the ROC Curve (AUC-ROC): 0.7423 Accuracy: 0.7421 Classification error: 0.2579 **Sensitivity: 0.7824** Precision: 0.7221 Specificity: 0.7022 F1 score: 0.7510	Area Under the ROC Curve (AUC-ROC): 0.7440 Accuracy: 0.7436 Classification error: 0.2564 **Sensitivity: 0.8136** Precision: 0.7120 Specificity: 0.6744 F1 score: 0.7594
Performance	Area Under the ROC Curve	Area Under the ROC Curve	Area Under the ROC Curve

based on unseen data	(AUC-ROC): 0.7408 Accuracy: 0.7413 Classification error: 0.2587 **Sensitivity: 0.8079** Precision: 0.7089 Specificity: 0.6572 F1 score: 0.7623	(AUC-ROC): 0.7483 Accuracy: 0.7484 Classification error: 0.2516 **Sensitivity: 0.7725** Precision: 0.7392 Specificity: 0.7240 F1 score: 0.7555	(AUC-ROC): 0.74983 Accuracy: 0.7502 Classification error: 0.2498 **Sensitivity: 0.8054** Precision: 0.7273 Specificity: 0.6942 F1 score: 0.7644
Model Selected	This model performed best in terms of sensitivity and generalizability.		

Batch Size

The model's performance was evaluated based on different batch sizes at 32 and 64 to determine which size performs better. The number of epochs are left constant at 100.

Table 12: Model Performance for Batch Size

Batch Size	32	64
Performance based on train-test data	Area Under the ROC Curve (AUC-ROC): 0.7423 Accuracy: 0.7421 Classification error: 0.2579 **Sensitivity: 0.7824** Precision: 0.7221 Specificity: 0.7022 F1 score: 0.7510	Area Under the ROC Curve (AUC-ROC): 0.7440 Accuracy: 0.7437 Classification error: 0.2563 **Sensitivity: 0.7990** Precision: 0.7176 Specificity: 0.6889 F1 score: 0.7561
Performance based on unseen data	Area Under the ROC Curve (AUC-ROC): 0.7483 Accuracy: 0.7484 Classification error: 0.2516 **Sensitivity: 0.7725** Precision: 0.7392 Specificity: 0.7240 F1 score: 0.7555	Area Under the ROC Curve (AUC-ROC): 0.7495 Accuracy: 0.7498 Classification error: 0.2502 **Sensitivity: 0.7935** Precision: 0.7319 Specificity: 0.7055 F1 score: 0.7615
Model Selected		This model performed best in terms of sensitivity and generalizability.

6.7.3. Model Evaluation

Based on the best suited dataset (Set 3), different models were created to determine the best input parameters for the ANN. The model was created with 50 epochs, batch size 32 and binary cross-entropy loss function performed with the following scores and ROC Curve on unseen data.

Table 13. Best Performing ANN Model Results

Performance Metric	**Result**
Sensitivity	0.8079
Accuracy	0.7413
Classification Error	0.2587
Precision	0.7089
Specificity	0.6572
F1 Score	0.7623

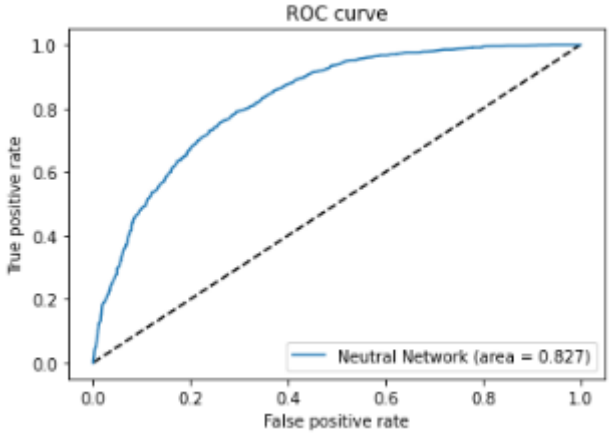


Fig 38. Final Model Evaluation

6.8. SVM

6.8.1. Correlation Coefficient Features

Variables Used : Top 6 variables based on Correlation Coefficient

['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']

```
print('The average accuracy is:', statistics.mean(k_fold_accuracy))
print('The average classification error is:', statistics.mean(k_fold_classification_error))
print('The average sensitivity is:', statistics.mean(k_fold_sensitivity))
print('The average precision is:', statistics.mean(k_fold_precision))
print('The average specificity is:', statistics.mean(k_fold_specificity))
print('The average f1 score is:', statistics.mean(k_fold_f1_score))
```

```
The average accuracy is: 0.7427
The average classification error is: 0.2573
The average sensitivity is: 0.78577
The average precision is: 0.72354
The average specificity is: 0.69963
The average f1 score is: 0.75336
```

Fig 39. SVM Correlation Coefficient Results

6.8.2. RFECV

a) Train the model on the entire feature set, create a SVM model and fit into RFECV and get the names of the optimal features

```
from sklearn.feature_selection import RFECV

X = train_test_data[['HighBP', 'HighChol', 'CholCheck', 'Smoker',
       'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'BMI_bins']]
y = train_test_data["Diabetes_binary"]

# Create a CatBoost classifier model
model = svm.SVC()

# Create an RFE object with the CatBoost model and number of features to select
rfecv = RFECV(model, cv=5, scoring='f1_weighted')

# Fit the RFE object on your dataset
rfecv.fit(X, y)

print("Feature ranking: ", rfecv.ranking_)

# Get the names of the optimal features
RFECV_selected = []
for index in range(len(X.columns)):
    if rfecv.ranking_[index] == 1:
        RFECV_selected.append(X.columns[index])

RFECV_selected
```

Fig 40. Code to train SVM model and use RFECV to find optimal features

b) Selecting the features that have been ranked 1

```
selected_features_indices = np.where(rfecv.support_ == True)[0]
selected_features = X.columns[selected_features_indices]
print("Selected features:", selected_features)
```

```
Selected features: Index(['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke',
       'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
       'BMI_bins'],
      dtype='object')
```

Fig 41. Features ranked 1 according to RFECV

c) Plotting the Mean Test Accuracy against the number of features selected

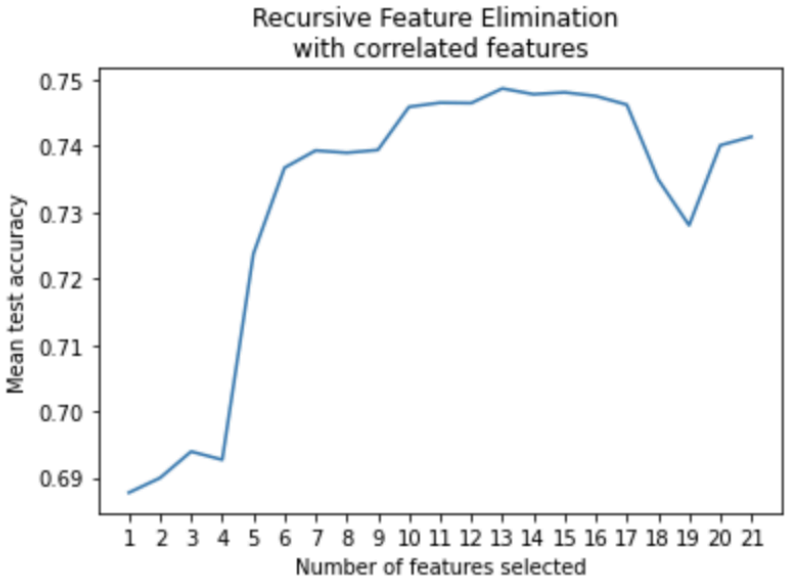


Fig 42. Mean Test Accuracy against the Number of Features Selected

RFECV Evaluation

Based on RFECV, the following variables have all been ranked 1. ['HighBP', 'HighChol',‘'CholCheck', 'Stroke', 'HeartDiseaseorAttack', 'Veggies', 'HvyAlcoholConsump', 'GenHlth', 'DiffWalk', ‘'Sex', ‘'Age', 'Income', 'BMI_bins']

But after plotting the mean accuracy against the number of features selected, we can see that the optimal number of features is 6. However, this does not tell us which 6 features have been selected

6.8.3. RFE

Afterwards, I leveraged on RFE to remove the weakest features until the specified number of features was reached. Using numbers ranging from 4 to 7 to specify the number of features to select for each iteration, I gathered different sets of features in order to pick the optimal number of features to perform on my model evaluation later.

In terms of sensitivity, the optimal number of features is **6** with the following variables:
['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump’, ‘GenHlth’, 'BMI_bins']

Table 13: SVM Model Evaluation Results with varying no. of Features

No. of Features	**Features**	**Sensitivity**
From Correlation Coefficient	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']	0.69963

4	['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump']	0.65634
5	['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump', 'GenHlth']	0.69292
6	**['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']**	**0.69671**
7	['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']	0.6908

6.8.4. Hyper Parameter Tuning

Then, I proceeded to do hyper parameter tuning to find a set of optimal hyper parameters values for my model using the 6 features identified by RFE earlier on.

```
# Best Features
features = ['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']
X = val_data[features]
y = val_data["Diabetes_binary"]

# Define the hyperparameter space
params = {
    'C': uniform(0.1, 10),
    'kernel': ['linear'],
    'degree': [2, 3, 4, 5],
    'gamma': ['scale', 'auto'] + list(np.logspace(-3, 3, 7)),
    'coef0': np.linspace(-1, 1, 21)
}
# Create a Logistic Regression Classifier model
model = LogisticRegressionr()

# Define the Random Search CV object
random_search = RandomizedSearchCV(
    model,
    param_distributions=params,
    n_iter=40,
    cv=5,
    random_state=424
)
# Fit the Random Search CV object to the data
random_search.fit(X, y)
```

```
# Print the best hyperparameters and the corresponding score
print('Best Hyperparameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)
```

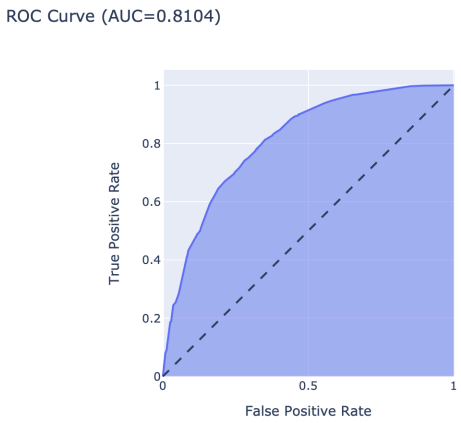
Results:

```
Best Hyperparameters: {'C': 2.350427529842045, 'coef0': -0.5, 'degree': 4, 'gamma': 1000.0, 'kernel': 'linear'}
Best Score: 0.7276387377584331
```

Figure 43: Best Hyperparameters

6.8.5. Model Evaluation

Last but not least, the model is evaluated using the 6 best features selected by RFE and best parameters by Random Search CV and calculated the performance metrics such as accuracy, classification error, sensitivity, precision, specificity and F1_score.




```
The accuracy on unseen data is:  0.7289752650176679
The classification_error on unseen data is:  0.27102473498233215
The sensitivity on unseen data is:  0.7752808988764045
The precision on unseen data is:  0.7117988394584139
The specificity on unseen data is:  0.682076813655761
The f1_score on unseen data is:  0.7421848739495799
```

Figure 44: Final Model Evaluation for SVM

Based on sensitivity, I can conclude that this model gives a 72.90% accuracy in detecting as many cases of diabetes as possible.

7. Conclusion and Future Work

Model Evaluation & Feature Selection

Based on different features and base classifiers we evaluated the models' performance and compiled the results of the best performing models as shown below.

Table 14: Evaluation of all models

Model	Base Classifier	Sensitivity	Attributes Used

SVM	-	0.7753	['HighBP', 'HighChol', 'CholCheck', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']
Decision Tree (Gini)	**-**	**0.8202**	**['HighBP', 'GenHlth', 'Age']**
Random Forest	Decision Tree	0.8013	['HighBP', 'GenHlth', 'Age']
ANN	-	0.8079	['HighBP', 'HighChol', 'CholCheck', 'GenHlth', 'Age', 'BMI_bins']
AdaBoost	Decision Tree	0.7591	['HighBP','HighChol','CholCheck','Smoker','Stroke','HeartDiseaseorAttack','PhysActivity','Veggies','HvyAlcoholConsump','GenHlth','MentHlth','PhysHlth','DiffWalk','Sex','Age','Education','Income','BMI_bins']
AdaBoost	Naive Bayes	0.7605	['HighBP', 'HighChol', 'GenHlth', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins']
Catboost	Decision Tree	0.7893	['HighBP', 'HighChol', 'GenHlth', 'DiffWalk', 'Age','BMI_bins']
XGBoost	Decision Tree	0.7844	['HighBP', 'HighChol', 'CholCheck', 'GenHlth', 'Age', 'BMI_bins']
Logistic Regression	-	0.7430	['HighBP', 'HighChol', 'CholCheck', 'HeartDiseaseorAttack', 'HvyAlcoholConsump', 'GenHlth', 'BMI_bins']

We found that the **decision tree classifier** produced the highest sensitivity out of all the models that we ran hence it is the **best suitable model for predicting diabetes**.

We also discovered that the 3 features: HighBP, GenHlth and Age were the **most sensitive** in its predictions and we should take note of these features when predicting diabetes.

By leveraging the power of the decision tree algorithm and focusing on these critical features, we were able to develop a highly accurate predictive model that can aid in early detection and intervention for individuals at risk of developing diabetes. Overall, our findings suggest that the decision tree classifier is a reliable and effective tool for predicting diabetes, and can provide

valuable insights for healthcare practitioners and policymakers seeking to improve diabetes prevention and management strategies.

Improvements from Literature Review

We implemented learning points from the literature review into our analytics to improve the robustness of our study.

1. From Case Study 1 - Predicting Diabetes Mellitus With ML Techniques

The paper used models such as decision trees, random forest & neural network, so we leveraged on other ML algorithms such as the different boost models. For feature selection, we not only used correlation coefficient to find relevant features but also Recursive Feature Elimination and Recursive Feature Elimination with Cross Validation. Lastly, we found that for performance evaluation we prioritise sensitivity but also generated classification error to evaluate the model.

2. From Case Study 2 - A Comprehensive Review of Various Diabetic Prediction Models

The paper used the Pima Indians diabetes dataset for its research which is small (at 768 entries) so we used a larger dataset with 70692 records for more robust algorithms. Where the study failed to conclude the best performing model, we selected our best model based on sensitivity score.

What can be done further?

With the results gathered from the models, we found that we can continue to improve our model performance with the following methods.

1. Explore new and advanced machine learning models: Keep up with the latest developments in machine learning algorithms and try new techniques to improve prediction accuracy.
2. Incorporate domain knowledge: Collaborate with healthcare professionals or researchers to incorporate domain knowledge into your models. This can help identify relevant features and improve your model's interpretability.
3. Include additional data sources: Combine your current dataset with other sources of information, such as patient demographics, genetic data, or environmental factors, to create a more comprehensive understanding of the factors affecting diabetes risk.
4. Personalised predictions: Explore the possibility of tailoring the models to individual patients based on their unique characteristics, which could lead to more accurate and personalised risk assessments.
5. Using different combinations of train-test split: Getting the best scores possible from different combinations of train-test split and unseen data. The current data splitting has insufficient unseen data for testing.
6. Use more recent data: The current dataset is based on survey responses from 2015, but we can still use more recent data after COVID-19 to see any changes in the relevant indicators.

References

Apollo Diagnostics. (2020, June 04). *Importance Of Early Diabetes Diagnosis And Screening*. Apollo Diagnostics. Retrieved April 1, 2023, from https://www.apollodiagnostics.in/blog/importance-of-early-diabetes-diagnosis-and-screening

Brownlee, J. (2019, September 4). *Feature Selection with Categorical Data.* Machine Learning Mastery. Retrieved April 1, 2023, from https://machinelearningmastery.com/feature-selection-with-categorical-data/

Misra, P. (2020, May 1). *Improving the classification accuracy using recursive feature*. ResearchGate. Retrieved April 3, 2023, from https://www.researchgate.net/publication/344181117_Improving_the_Classification_Accuracy_using_Recursive_Feature_Elimination_with_Cross-Validation

Open Access Government. (2018, June 25). *Diabetes: A Global Health Challenge*. Open Access Government. Retrieved April 1, 2023, from https://www.openaccessgovernment.org/diabetes-a-global-health-challenge/46992/

Rice, D., & Galbraith, M. (2008, November 16). *IDF diabetes Atlas: Global estimates of undiagnosed diabetes in adults for 2021*. ScienceDirect. Retrieved April 1, 2023, from https://www.sciencedirect.com/science/article/abs/pii/S0168822721004770

Saxena, R., Kumar Sharma, S., Gupta, M., & Sampada, G. C. (2022, April 12). *A Comprehensive Review of Various Diabetic Prediction Models: A Literature Survey*. NCBI. Retrieved April 1, 2023, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9018179/

Teboul, A. (2021). *Diabetes Health Indicators Dataset [Data set]*. Kaggle. Retrieved April 1, 2023, from https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_5050split_health_indicators_BRFSS2015.csv

World Health Organization. (n.d.). *Diabetes*. World Health Organization (WHO). Retrieved April 1, 2023, from https://www.who.int/health-topics/diabetes#tab=tab_1

Zou, Q., Qu, K., Luo, Y., Yin, D., Ju, Y., & Tang, H. (2018, October 12). *Predicting diabetes mellitus with machine learning techniques*. Frontiers. Retrieved April 1, 2023 from https://www.frontiersin.org/articles/10.3389/fgene.2018.00515/full

Appendix

Decision Tree Classifier Model (Gini > Entropy)

RFECV Results

Rank	Features
1	HighBP, GenHlth, Age
2	PhysHlth
3	MentHlth
4	Income
5	Education
6	Fruits
7	Smoker
8	PhysActivity
9	BMI_bins
10	Veggies
11	Sex
12	DiffWalk
13	HighChol
14	HeartDiseaseorAttack
15	NoDocbcCost
16	Stroke
17	HvyAlcoholConsump
18	AnyHealthcare
19	CholCheck

RFE Results

Selected Num Features	Features
3	HighBP, GenHlth, Age
4	HighBP, GenHlth, PhysHlth, Age

5	HighBP, GenHlth, MentHlth, PhysHlth, Age
6	HighBP, GenHlth, MentHlth, PhysHlth, Age, Income
7	HighBP, GenHlth, MentHlth, PhysHlth, Age, Education, Income

6 Different Model Results

Model	**Variables**	**Classification Accuracy**	**Classification Error**	**Sensitivity**	**Precision**	**Specificity**	**F1 score**	**AUC Score**
1	HighBP HighChol GenHlth DiffWalk Age BMI_bins	0.73892	0.73892	0.77282	0.72381	0.70497	0.74748	0.8094
2	HighBP GenHlth Age	0.72327	0.27673000000000003	0.80248	0.6929	0.64407	0.74357	0.7957
3	HighBP GenHlth PhysHlth Age	0.71599	0.28401	0.77318	0.69397	0.65878	0.73136	0.7804
4	HighBP GenHlth MentHlth PhysHlth Age	0.69919	0.30081	0.74225	0.68352	0.65618	0.71161	0.7434
5	HighBP GenHlth MentHlth PhysHlth Age Income	0.67908	0.32092	0.67308	0.681410000000001	0.68512	0.67719	0.7052
6	HighBP GenHlth MentHlth PhysHlth Age Education Income	0.66459	0.33541	0.64431	0.67163	0.68486	0.65764	0.6814

Parameter Tuning Results

Best Hyperparameters	Best Score
‘min_samples_split': 2 'min_samples_leaf': 4 'max_features': 'sqrt' 'max_depth': 6 'criterion': 'gini'	0.7159956474428727

Unseen Data Results

Variables	Classification Accuracy	Classification Error	Sensitivity	Precision	Specificity	F1 score	AUC Score
HighBP GenHlth Age	0.7339222614840989	0.2660777385159011	0.8202247191011236	0.7015015015015015	0.6465149359886202	0.7562317902233733	0.8022

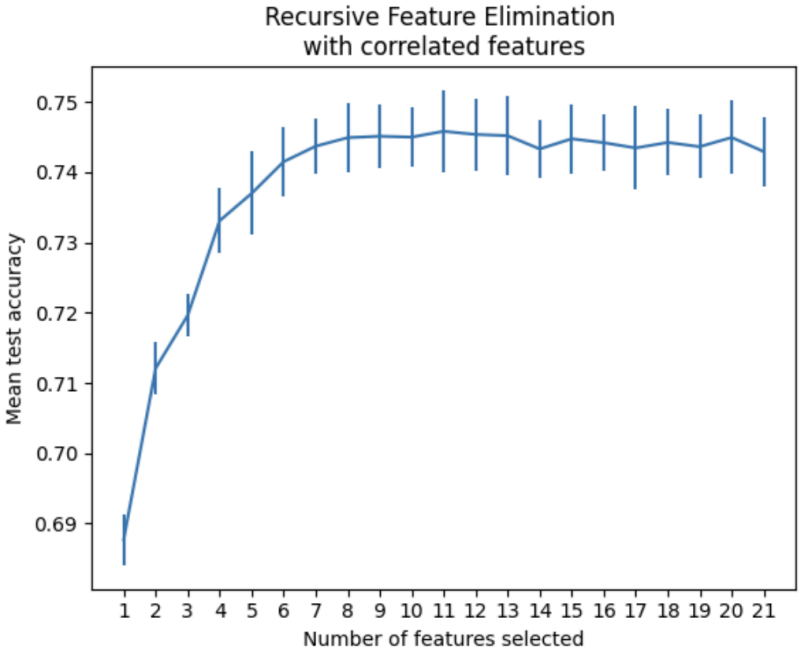
XGBoost

RFECV Results

Rank	Features
1	HighBP, HighChol, CholCheck, HeartDiseaseorAttack, HvyAlcoholConsump, GenHlth, DiffWalk, Sex, Age, Income, BMI_bins
2	Stroke
3	Smoker
4	Education
5	PhysActivity
6	PhysHlth
7	NoDocbcCost
8	MentHlth
9	Fruits
10	AnyHealthcare
11	Veggies

Recursive Feature Elimination with Correlated Features

Plotting the mean accuracy against the number of features, we can see that it plateaus at 6 features as seen in the graph below. This tells us that the optimal number of features to use for XGBoost is 6



RFE Results

Selected Num Features	Features
4	HighBP, HighChol, GenHlth, BMI_bins
5	HighBP, HighChol, GenHlth, Age, BMI_bins
6	HighBP, HighChol, CholCheck, GenHlth, Age, BMI_bins
7	HighBP, HighChol, CholCheck, HeartDiseaseorAttack, GenHlth, Age, BMI_bins
9	HighBP, HighChol, CholCheck, HeartDiseaseorAttack, HvyAlcoholConsump, GenHlth, DiffWalk, Age, BMI_bins

8 diff model results

Model	Variables	Classification Accuracy	Classification Error	Sensitivity	Precision	Specificity	F1 score	AUC Score
1	HighBP HighChol GenHlth DiffWalk Age BMI_bins	0.74214	0.25786	0.78578	0.72277	0.69848	0.75292	0.8167
2	HighBP, HighChol, GenHlth, BMI_bins	0.73338	0.26662	0.7416699999999999	0.72967	0.72510000000000001	0.73556	0.8050
3	HighBP, HighChol, GenHlth, Age, BMI_bins	0.74112	0.25888	0.78171	0.72311	0.70055	0.75125	0.8170
4	HighBP, HighChol, CholCheck, GenHlth, Age, BMI_bins	0.7420599999999999	0.25794	0.78633	0.722470000000001	0.6978	0.75299	0.8184
5	HighBP, HighChol, CholCheck, HeartDiseaseor Attack, GenHlth,	0.7371099999999999	0.26289	0.7827299999999999	0.71743	0.69156	0.748620000000001	0.8091

	Age, BMI_bins							
6	HighBP, HighChol, CholCheck, HeartDiseaseorAttack, HvyAlcoholConsump, GenHlth, DiffWalk, Age, BMI_bins	0.74501	0.25499	0.78626	0.72644	0.70375	0.75511	0.8208
7	HighBP, HighChol, CholCheck, HeartDiseaseorAttack, Age, BMI_bins (removed GenHlth)	0.72245	0.27755	0.77632	0.70088	0.66858	0.73664	0.7901

Parameter Tuning

Best Hyperparameters	**Best Score**
'n_estimators': 50 'max_depth': 4	0.7326441784548423

Unseen Data Results

Variables	**Classification Accuracy**	**Classification Error**	**Sensitivity**	**Precision**	**Specificity**	**F1 score**	**AUC Score**
HighBP, HighChol, CholCheck, GenHlth, Age, BMI_bins	0.7491166077738516	0.2508833922261484	0.7844101123595506	0.7348684210526316	0.7133712660028457	0.7588315217391304	0.8282

CatBoost

8 diff model results

Model	Variables	Classification Accuracy	Classification Error	Sensitivity	Precision	Specificity	F1 score	AUC Score
1	HighBP HighChol GenHlth DiffWalk Age BMI_bins	0.74263	0.25737	0.78501	0.72378	0.70024	0.75312	0.8174
2	HighBP GenHlth Age BMI_bins	0.73833	0.26167	0.78803	0.71689	0.6886	0.75071	0.8117
3	HighBP GenHlth PhysHlth Age BMI_bins	0.73783	0.26217	0.78864	0.71601	0.68703	0.75054	0.8109
4	HighBP GenHlth PhysHlth Age Income BMI_bins	0.73868	0.26132	0.78642	0.71802	0.69094	0.75062	0.8118
5	HighBP GenHlth MentHlth PhysHlth Age Income BMI_bins	0.73805	0.26194	0.78481	0.71781	0.69131	0.7498	0.8116
6	HighBP GenHlth MentHlth PhysHlth Age Income BMI_bins HighChol Education	0.74329	0.25671	0.78926	0.72294	0.6973	0.75459	0.8178
7	HighBP GenHlth Age Income BMI_bins	0.74312	0.25688	0.79015	0.72235	0.69608	0.75467	0.8169

	HighChol Education							
8	HighBP Age Income BMI_bins HighChol Education	0.723	0.277	0.77819	0.700929	0.66784	0.73751	0.7919

Parameter Tuning

Best Hyperparameters	**Best Score**
'l2_leaf_reg': 0.17865843160768383, 'max_depth':4, 'n_estimators': 200	0.7387377584330794

Unseen Data Results

Variables	**Classification Accuracy**	**Classification Error**	**Sensitivity**	**Precision**	**Specificity**	**F1 score**	**AUC Score**
HighBP, HighChol, GenHlth, DiffWalk, Age, BMI_bins	0.7515901060070671	0.2484098939929329	0.7893258426966292	0.7360838244924689	0.713371266002845	0.7617756692646561	0.8292

<u>Random Forest</u>

RFECV Results

Rank	**Features**
1	'HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth', 'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income', 'BMI_bins'

RFE Results

Selected Num	Features

Features	
3	'HighBP', 'GenHlth', 'Age'
4	'HighBP', 'GenHlth', 'PhysHlth', 'Age'
6	'HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Income'
7	['HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Income', 'BMI_bins']

7 diff model results

Model	Variables	Classification Accuracy	Classification Error	Sensitivity	Precision	Specificity	F1 score	AUC Score
1	HighBP HighChol GenHlth DiffWalk Age BMI_bins	0.73924	0.26076	0.77929	0.72158	0.69916	0.74928	0.8115
2	'HighBP', 'GenHlth', 'Age'	0.72319	0.27681	0.80444	0.69218	0.64193	0.74396	0.7957
3	'HighBP', 'GenHlth', 'PhysHlth', 'Age'	0.71777	0.28223	0.78275	0.6928	0.65277	0.735	0.7840
4	'HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Income'	0.71777	0.28223	0.78275	0.6928	0.65277	0.735	0.7840
5	'HighBP', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Education', 'Income'	0.69595	0.30405	0.72646	0.68482	0.66543	0.70498	0.7547
6	'HighBP', 'GenHlth','Age', 'Income'	0.72566	0.27434	0.775	0.70548	0.6763	0.73858	0.7913

7	'HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth', 'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income','BMI_bins'	0.7317	0.2683	0.76855	0.7159	0.69489	0.74124	0.7997

Parameter Tuning

Best Hyperparameters	**Best Score**
'n_estimators': 250, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_depth': 3, 'bootstrap': False	0.7173014145810663

Unseen Data Results

Variables	**Classification Accuracy**	**Classification Error**	**Sensitivity**	**Precision**	**Specificity**	**F1 score**	**AUC Score**
'HighBP', 'GenHlth', 'Age'	0.7339	0.2660	0.80827	0.7057	0.6586	0.7535	0.8038