UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING

SECB3203

PROGRAMMING FOR BIOINFORMATICS

PROJECT REPORT

LECTURER NAME

DR. SEAH CHOON SEN

SECTION 01

GROUP 06

STUDENT NAME & MATRIC ID

RAJA NUR ALLEA DEWI MAHSURI BINTI RAJA MOHD YUSRI
A24CS0294

IRDINA SOFIA BINTI ROHAIDI
A24CS0253

NURZULAIKHA BINTI MOHD ISA
A24CS0291

TABLE OF CONTENTS

1.0 INTRODUCTION

Nowadays, chronic diseases such as heart diseases, diabetes, and cancer are the leading causes of death not only in Malaysia but also worldwide. According to Ramli, A., & Taher, S. (2008), it states that chronic diseases are the major cause of death and disability in Malaysia, with 71% of overall cases of death and 69% of the total burden of disease meanwhile it killed at least 43 million people around the world based on the data from World Health Statistics (WHO) on 2021. This study focuses on diabetes itself, which is a common disease that might occur in people nearby. Diabetes happens when the pancreas itself fails to produce enough insulin or when the blood sugar is too high and the body is not properly utilizing it, also known as Hyperglycemia. Diabetes is most likely to affect people of all ages, depending on the type. Type 1 diabetes happens when the pancreas produces insufficient insulin or no insulin at all. This type of diabetes has no cure and depends on how well the insulin. Meanwhile, type 2 diabetes happens when the body stops using insulin properly, which can lead to serious damage to the body, especially nerves or blood vessels. Gestational diabetes happens during pregnancy, where the blood sugar level is above the normal sugar level value but below the diagnostic of diabetes. The symptoms of those types can be detected at early stages by doing regular check-ups and blood tests. However, this traditional diagnostic method in most cases relies on clinical tests, which might not be accessible in certain areas, or the cost is not cheap, especially for people in urban areas. These are the reasons why it is important to use data driven methods to detect individuals at high risk of diabetes with the help of simple demographic and lifestyle information.

In this context, machine learning and data analysis are useful tools to predict health conditions and support clinical decision-making by analyzing the large volume of health-related data. By using machine learning, it can discover the patterns from the information and come up with the relationships between the factors such as body mass index (BMI), sugar level, age and smoking status. The results enable healthcare institutions to predict the risk at the earliest stage and come up with a preventive intervention. This project will apply the machine learning techniques to a dataset from Kaggle to predict the risk factors of diabetes.

1.1 PROBLEM BACKGROUND

The area of medical research has been improving due to high-end health technology as well as the advancement of data science. However, global cases due to diabetes keep rising, which causes worries if it cannot be prevented. Diabetes happens when the pancreas itself fails to produce enough insulin or when the blood sugar is too high and the body is not properly utilized it which can lead to other chronic conditions such as cardiovascular diseases, kidney failure and damage to the body's nerves. Even with that progress, diabetes remains as one of the leading causes of death and disabilities such as vision impairment.

The increasing of large datasets has made it easier to analyze the risk factors of diabetes by using machine learning. These computer-based methods can find patterns in health data that are hard to find with standard statistical methods. Machine learning models can look at a lot of different variables at once and figure out how they are related to each other in a complicated way. This study seeks to utilize supervised machine-learning algorithms on a global health dataset to forecast the probability of diabetes in individuals. The study seeks to determine the most critical factors affecting diabetes by examining various models, including Logistic Regression, Decision Tree, and Random Forest, and to assess model accuracy for early prediction. The insights derived from this analysis are anticipated to aid in the development of effective digital health applications that facilitate early intervention and enhance healthcare outcomes.

1.2 PROBLEM STATEMENT

Diabetes is a long-lasting chronic disease that causes millions of people in the world and is dangerous when not noticed and treated at an early age. Despite the huge amounts of medical and health-related information, a great number of patients are still diagnosed at a late stage despite the fact that current methods of diagnosis are based on manual clinical evaluation and set threshold criteria. Such conventional approaches tend to be time-consuming and subject to error, and do not reflect the complex relationships between several health indicators like glucose level, body mass index (BMI), age and blood pressure.

As a result, patients who exhibit moderate but combined risk factors may not be identified as high-risk, even though meaningful patterns indicating diabetes exist within medical datasets. This puts a disconnect between the existence of healthcare data and the utilization of the same to make accurate and timely predictions of diabetes.

Therefore, there is a growing need for a data-driven predictive framework that can analyze multiple health indicators simultaneously, identify the most significant risk factors, and improve the accuracy of diabetes risk detection. Using machine learning and statistical methods on a dataset of diabetes, this study will produce valuable information and also create a predictive model that may be useful to healthcare professionals by making earlier diagnoses and improving the decisions they make concerning prevention healthcare.

1.3 AIM AND OBJECTIVES

AIM

To investigate and develop a machine learning model that uses data to enhance the early detection of diabetes through the learning of complicated interactions among a variety of health measures, overcoming the restrictions of conventional rule-based diabetes diagnosis.

OBJECTIVES

- To investigate the relationships between health measures (such as BMI, blood pressure, and age) to identify the most influential risk factors for diabetes.
- To design a predictive model, which is capable of identifying the risk of diabetes more accurately than the conventional threshold-based diagnostic techniques.
- To make a comparison of various machine learning algorithms based on performance metrics like accuracy, sensitivity, and specificity to identify the model that is the most reliable.
- To present the model outputs as a way of explaining the different health indicators as contributors to the risk of diabetes.

1.4 SCOPES

DATASET

The project uses the Diabetes Health Indicators Dataset from Kaggle, derived from the Behavioral Risk Factor Surveillance System (BRFSS 2015). The dataset includes over 70,692 records representing health survey responses. It focuses on health behaviors, medical history, and demographic information that may influence the likelihood of causing diabetes. Each record includes a binary target variable “Diabetes_binary”, indicating whether an individual has diabetes (1) or not (0).

FEATURES

The Diabetes Health Indicators Dataset includes 21 features representing demographic, behavioral, and medical factors related to diabetes risk. These features cover aspects such as age, sex, BMI, blood pressure, cholesterol levels, physical activity, smoking and alcohol habits, sleep duration, and general health status. The dataset also includes a comprehensive view of how lifestyle and health indicators can cause diabetes. This dataset has 22 columns, but this project only uses 10 columns.

TECHNIQUES

This project applies data preprocessing, exploratory data analysis, and machine learning methods in Python to forecast the occurrence of diabetes. Analysis starts with cleaning and preparation of data, and then moves on to the exploratory analysis of data to determine trends and associations between health indicators. Gradient Boosting is taken as the better classification model in the case of predictive modelling. Gradient Boosting is a type of ensemble learning method that trains decision trees in a sequential manner, with the successive trees aiming to rectify the mistakes committed by the previous trees. Gradient Boosting is much less prone to overfitting and can classify data with much greater accuracy in comparison to a simple Decision Tree Model, because it uses a combination of weak learners to build a strong one.

The Decision Tree model is initially adopted as a baseline to understand the data and establish initial performance. However, due to its tendency to overfit and its limited generalization capability, Gradient Boosting is introduced as an improvement model to achieve better performance and more stable predictions. Model performance is measured by accuracy, precision, recall, F1-score and ROC-AUC. The analysis of importance of features is also enough to determine the most significant health indicators that cause diabetes. Python libraries like Pandas, Numpy, Matplotlib, and scikit learn are used to analyze all the data, develop, and evaluate all models.

LIMITATIONS

There are a number of limitations associated with this project. The data is based on self-reported of people, which is not always true or accurate. It further fails to provide valuable information like genetic, environmental, or medical test data, which restrains the depth of the analysis. The data was taken only once. Therefore, it cannot demonstrate the way in which the health of a person should vary as time passes. There might be a close relationship between some of the features, and this can have an impact on the learning and predictive performance of the model. Even though the advanced models of machine learning, such as the Random Forest, could produce a high accuracy, they are hard to understand and interpret. The sample also may not be quite applicable to any other population. Finally, the reliance on cloud computing providers, such as Azure or AWS, can be resource-constrained or too expensive, or inaccessible due to the internet.

2.0 DATA COLLECTION AND PRE-PROCESSING

This section explains how the dataset was acquired, imported, and prepared for modelling and analysis. For machine learning models to produce reliable and valid results, proper data collection and preprocessing are crucial.

2.1 IMPORTING DATASET

The dataset used in this project is from Kaggle:
https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_5050split_health_indicators_BRFSS2015.csv

This dataset contains 70,692 survey responses and 22 health-related variables, representing various demographic, lifestyle, and health conditions of individuals. The variables include, but are not limited to:

- Body Mass Index (BMI)
- High blood pressure (HighBP)
- High cholesterol (HighChol)
- General health status (GenHlth)
- Physical health (PhysHlth)
- Mental health (MentHlth)
- Physical activity (PhysActivity)
- Age
- Difficulty walking (DiffWalk)
- Diabetes status (Diabetes_012)

The target variable, Diabetes_012, categorizes individuals into three groups:

- 0: No diabetes or diabetes only during pregnancy
- 1: Pre-diabetes or borderline diabetes
- 2: Diagnosed diabetes

```python
# Load dataset
df = pd.read_csv(
    "diabetes_dataset.csv",
    encoding="latin1",
    engine="python",
    on_bad_lines="skip"
)
print("Shape of dataset:", df.shape)


df.head()
df.info()
```

```
Shape of dataset: (70692, 22)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70692 entries, 0 to 70691
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Diabetes_012          70692 non-null  int64
 1   HighBP                70692 non-null  int64
 2   HighChol              70692 non-null  int64
 3   CholCheck             70692 non-null  int64
 4   BMI                   70692 non-null  int64
 5   Smoker                70692 non-null  int64
 6   Stroke                70692 non-null  int64
 7   HeartDiseaseorAttack  70692 non-null  int64
 8   PhysActivity          70692 non-null  int64
 9   Fruits                70692 non-null  int64
 10  Veggies               70692 non-null  int64
 11  HvyAlcoholConsump     70692 non-null  int64
 12  AnyHealthcare         70692 non-null  int64
 13  NoDocbcCost           70692 non-null  int64
 14  GenHlth               70692 non-null  int64
 15  MentHlth              70692 non-null  int64
 16  PhysHlth              70692 non-null  int64
 17  DiffWalk              70692 non-null  int64
 18  Sex                   70692 non-null  int64
 19  Age                   70692 non-null  int64
 20  Education             70692 non-null  int64
 21  Income                70692 non-null  int64
dtypes: int64(22)
memory usage: 11.9 MB
```

The dataset, loaded using the Pandas library, consists of 70,692 rows and 22 columns, reflecting a comprehensive array of survey responses focused on health-related variables, preliminary examination utilizing df.info indicates that all variables contain no missing values, reaffirming the dataset's completeness. Each variable is represented as an integer data type (int64), predominantly comprising binary indicators or ordinal categories related to health conditions.

Some of the important health-related data in the dataset are body mass index (BMI), blood pressure, cholesterol level, general health, physical activity, age, education, income, and diabetes status. The dataset is not very big since it only takes up about 11.9 MB of memory. A regular computer can analyze and train it is using machine learning. The data is generally clean and well-organized, and it can be used for data analysis, data exploration, and model building.

2.2 DATA WRANGLING

Remove extra spaces in the columns

```
# Removes extra spaces and standardizes column names
df.columns = df.columns.str.strip()
```

Identify the missing values

```
# Check if there are missing values

df.isna().sum() # no missing values
```

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

In this part of code, the analysis is checking on the missing values. The check showed that all the variables had complete data, so there was no need to fill in or remove any records. This helped keep the whole dataset safe for analysis.

Separating numerical and binary features

```
# ----------------------------------------
# Separate feature columns by data type
# ----------------------------------------

# Define the target (label) column
target = 'Diabetes_binary'

# Identify binary feature columns:
# - Columns that contain exactly two unique values (e.g., 0 and 1)
# - Exclude the target variable from this list
binary_col = [
    col for col in df.columns
    if df[col].nunique() == 2 and col != target
]

# Identify numerical / non-binary feature columns:
# - All remaining columns excluding binary features and the target variable
# - This helps distinguish features that may require different preprocessing steps
num_col = [
    col for col in df.columns.difference(binary_col)
    if col != target
]

# Display the separated feature groups
print('Binary Columns: ', binary_col)
print('Numerical Columns: ', num_col)
```

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

This code divides the features of the dataset into binary and numerical groups to assist in the proper preprocessing and modelling. The first step is identifying Diabetes_binary as the target variable and excluding it from feature selection. These are then narrowed down to binary feature columns which have only two distinct values and represent yes/no or true/false health indicators. The remaining non targets columns that are not binary are categorized as numerical features, which are normally continuous or ordinal measurements. Lastly, generated binomial and numerical columns are printed to ensure transparency and provide a clear foundation for applying different preprocessing techniques in subsequent analysis.

Verify diabetes value

```
# Check the target variables of the original dataset
# 0 is for no diabetes or only during pregnancy,
# 1 is for pre-diabetes or borderline diabetes,
# and 2 is for yes diabetes
df['Diabetes_012'].value_counts().sort_index()
```

```
Diabetes_012
0    35346
1    35346
Name: count, dtype: int64
```

To gain an idea of the balance of classes prior to modelling, this code considers the distribution of the original target variables Diabetes 012. The variable encodes diabetes status into three categories: 0 for no diabetes or gestational diabetes only, 1 for pre-diabetes or borderline diabetes, and 2 for diagnosed diabetes. The code counts the instances of each category with value counts sorted by index, which can be effectively compared across the classes. The output indicates equal counts of the categories found in the dataset, which indicates that the target distribution is balanced, which is good in minimizing bias and enhancing the accuracy of downstream classification models.

Transform Diabetes_012 to Diabetes_binary

```
In [7]: # Convert 1 and 2 into diabetic class (1)
        df['Diabetes_012'] = df['Diabetes_012'].replace({1:1, 2:1})

In [8]: # Change the column name to Diabetes_binary
        df = df.rename(columns = {'Diabetes_012': 'Diabetes_binary'})

In [9]: # Print the count od the diabtes cases
        # 0 - non diabetic
        # 1 - diabetic
        print("\nTarget distribution (Diabetes_binary):")
        print(df["Diabetes_binary"].value_counts())
```

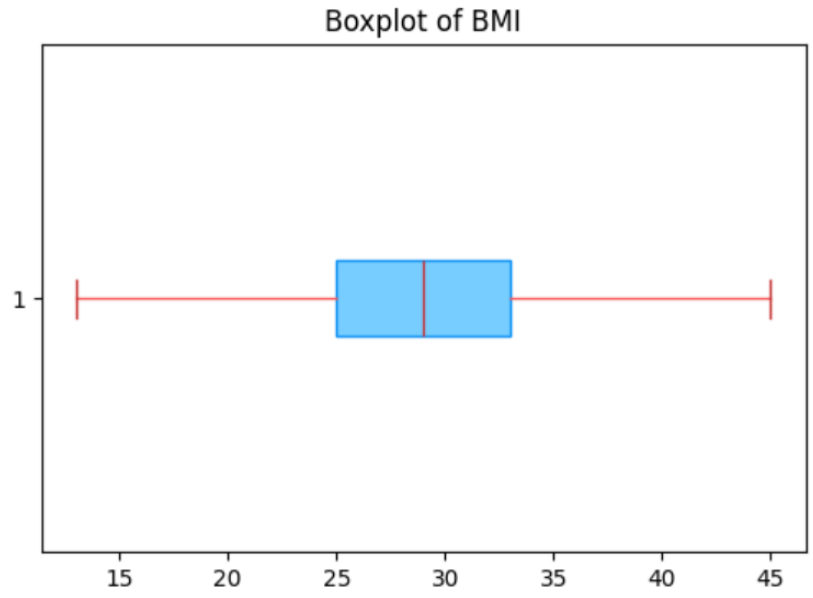
```
Target distribution (Diabetes_binary):
Diabetes_binary
0    35346
1    35346
Name: count, dtype: int64
```

Next, renaming or assigning the target variable Diabetes_012 in terms of its categories to learn more about it. Since the first data set has three types of diabetes (no diabetes, pre-diabetes, and diabetes), the target variable was changed to binary form so that it would be easier to predict the observation. In this instance, individuals with pre-diabetes or diabetes were categorised as high-risk cases, while those without diabetes were classified as low-risk cases.

Handle the BMI outliers

```
# Define a function to winzorize outliers
def winsorize_iqr(df, col, k=1.5):

    # Set the lower quartile and the upper quartile
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)

    # Set the lower bound and upper bound
    # Based on IQR and parameter k
    IQR = Q3 - Q1
    lower_bound = Q1 - k * IQR
    upper_bound = Q3 + k * IQR

    print('Lower Bound:', lower_bound)
    print('Upper Bound:', upper_bound)

    # Replace values below/above the bounds with the boundary values
    return df[col].clip(lower=lower_bound, upper=upper_bound)

df['BMI'] = winsorize_iqr(df, 'BMI')

# Boxplot of winsorized BMI
plot_boxplot(df, 'BMI')
```

```
Lower Bound: 13.0
Upper Bound: 45.0
```



This code implements an interquartile range (IQR) based winsorization procedure to handle outlier in the BMI variable. The first step involves the calculation of the first and third quartile (Q1 and Q3) followed by the calculation of the IQR based on them and finally obtaining lower and upper bounds based on a scaling factor (k= 1.5). any BMI values that are below these limits or above them are limited to the respective boundary values instead of being eliminated to maintain the total size of the dataset and minimize the effect of extreme values in the data set. The printed boundaries give a visibility on the thresholds applied and the resultant boxplot assures that outliers have been well kept in check and the resultant distributions give a stronger and more reliable BMI distribution to be used in further modeling.

Split data into training testing

```
X_train, X_test, y_train, y_test = train_test_split(
    X_all, y, test_size=0.2, random_state=42, stratify=y
)
```

In this code, the entire dataset is divided into training and testing subsets in order to facilitate the evaluation of the model without biases. In particular, 80 percent of the data are placed in the training set and 20 percent in the test set, and random state=42 so that the split can be replicated. The stratify= y argument maintains the original distribution of classes of the target variable in both subsets which is especially essential in classifying tasks to avoid the risk of class imbalance to impact model performance.

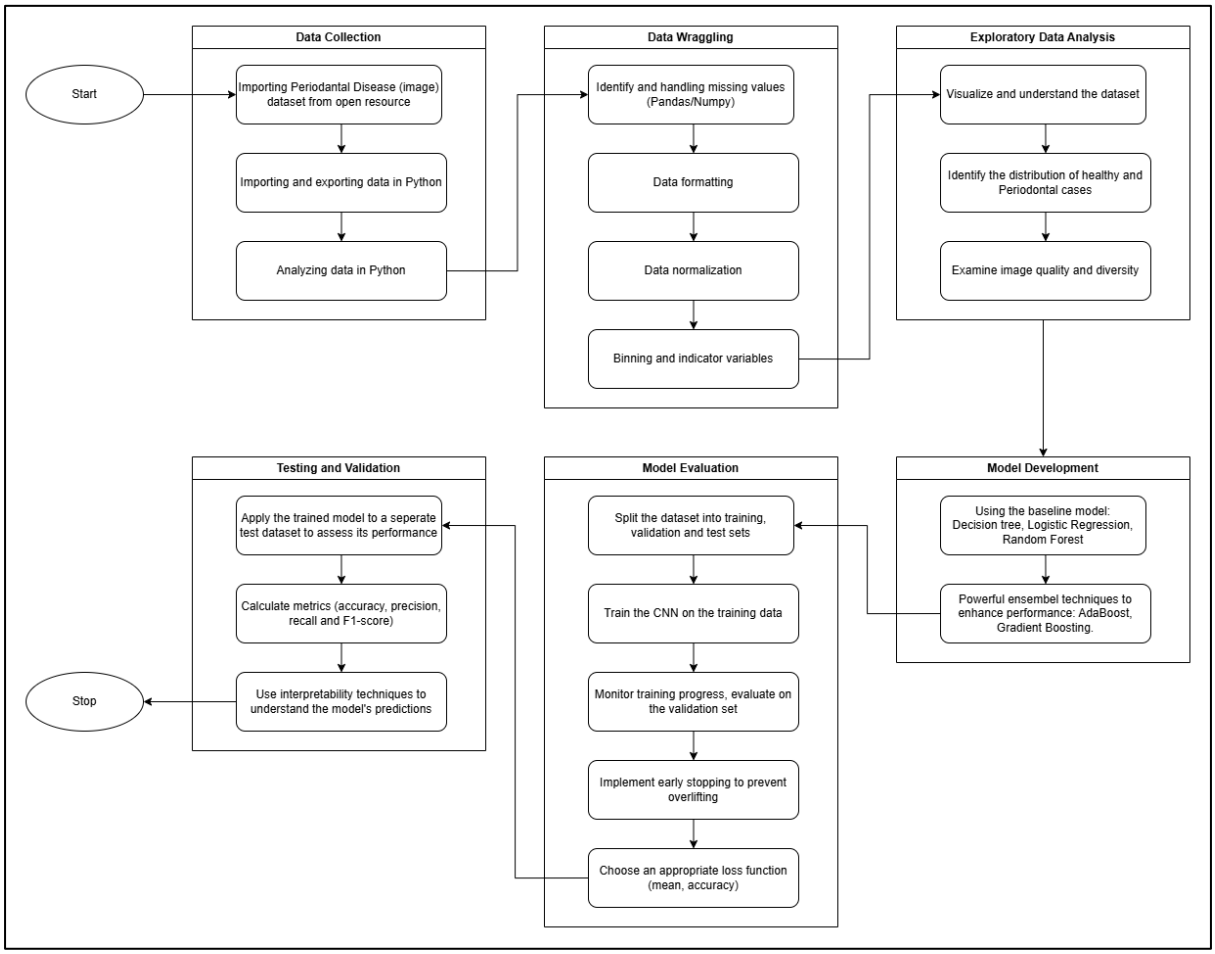
2.3 SOFTWARE AND HARDWARE REQUIREMENTS

Hardware	Specification	Descriptions
Processor	11th Gen Intel® Core™ i5-1140H @ 2.70GHz	To speed up data processing and analysis
RAM	32GB	Sufficient storage in order to handle large and complex datasets
System Type	64-bit operating system, x64-based processor	Support simulation, virtualization, and process large data at once
Graphics Processing Unit (GPU)	4GB	For faster computations

Software	Descriptions
Python	A programming language that is being used for data preprocessing, analysis, and performing machine learning for this project.
Pandas Library	Used for data cleaning, manipulation, and analysis for the diabetes dataset.
NumPy	Supports numerical computations and array-based operations required for efficient data processing.
Matplotlib	Used to visualize basic plots such as histograms and bar charts.
Seaborn	Generate advanced visualization, such as correlation heatmaps.

Scikit-learn	Used for machine learning model implementation, including Decision Tree and Random Forest.
Warnings	Control warning messages that appear when running code.
Patch	Create custom legend entries.
Jupyter Notebook	Provides an interactive environment for coding, data visualization, analysis, and documenting results in an organized manner.
Visual Code	Used as a code editor to collaboratively write, debug, and manage Python scripts among team members.
GitHub	Used for version control, collaboration, project progress tracking, and sharing code and documentation as required by the course.

3.0 FLOWCHART OF PROPOSED APPROACH



3.1 EXPLORATORY DATA ANALYSIS

Descriptive Statistic

```
df.describe(include='all').T
```

	count	mean	std	min	25%	50%	75%	max
Diabetes_012	70692.0	0.500000	0.500004	0.0	0.0	0.5	1.0	1.0
HighBP	70692.0	0.563458	0.495960	0.0	0.0	1.0	1.0	1.0
HighChol	70692.0	0.525703	0.499342	0.0	0.0	1.0	1.0	1.0
CholCheck	70692.0	0.975259	0.155336	0.0	1.0	1.0	1.0	1.0
BMI	70692.0	29.856985	7.113954	12.0	25.0	29.0	33.0	98.0
Smoker	70692.0	0.475273	0.499392	0.0	0.0	0.0	1.0	1.0
Stroke	70692.0	0.062171	0.241468	0.0	0.0	0.0	0.0	1.0
HeartDiseaseorAttack	70692.0	0.147810	0.354914	0.0	0.0	0.0	0.0	1.0
PhysActivity	70692.0	0.703036	0.456924	0.0	0.0	1.0	1.0	1.0
Fruits	70692.0	0.611795	0.487345	0.0	0.0	1.0	1.0	1.0
Veggies	70692.0	0.788774	0.408181	0.0	1.0	1.0	1.0	1.0
HvyAlcoholConsump	70692.0	0.042721	0.202228	0.0	0.0	0.0	0.0	1.0
AnyHealthcare	70692.0	0.954960	0.207394	0.0	1.0	1.0	1.0	1.0
NoDocbcCost	70692.0	0.093914	0.291712	0.0	0.0	0.0	0.0	1.0
GenHlth	70692.0	2.837082	1.113565	1.0	2.0	3.0	4.0	5.0
MentHlth	70692.0	3.752037	8.155627	0.0	0.0	0.0	2.0	30.0
PhysHlth	70692.0	5.810417	10.062261	0.0	0.0	0.0	6.0	30.0
DiffWalk	70692.0	0.252730	0.434581	0.0	0.0	0.0	1.0	1.0
Sex	70692.0	0.456997	0.498151	0.0	0.0	0.0	1.0	1.0
Age	70692.0	8.584055	2.852153	1.0	7.0	9.0	11.0	13.0
Education	70692.0	4.920953	1.029081	1.0	4.0	5.0	6.0	6.0
Income	70692.0	5.698311	2.175196	1.0	4.0	6.0	8.0	8.0

Target variable distribution

```
print(df[target].value_counts(normalize=True).sort_index().map(lambda x: f'{x: .3%}'))
```

```
Diabetes_binary
0     50.000%
1     50.000%
Name: proportion, dtype: object
```

Grouping Analysis

```
# --- Basic Grouping ---
# Calculate mean values of key health indicators by diabetes status

key_features = [
    'BMI',
    'GenHlth',
    'PhysHlth',
    'MentHlth',
    'Age',
    'HighBP',
    'HighChol',
    'PhysActivity'
]

grouped_key = df.groupby('Diabetes_binary')[key_features].mean()

grouped_key
```

ANOVA Test

```
target = "Diabetes_binary"
features = df.drop(columns=[target]).columns

anova_results = []

for feature in features:
    group_0 = df[df[target] == 0][feature]
    group_1 = df[df[target] == 1][feature]

    f_stat, p_value = f_oneway(group_0, group_1)

    anova_results.append({
        "Feature": feature,
        "F-statistic": f_stat,
        "p-value": p_value,
        "Significant (p<0.05)": "Yes" if p_value < 0.05 else "No"
    })

anova_df = pd.DataFrame(anova_results).sort_values("p-value")
anova_df
```

Correlation Analysis

```
target_col = "Diabetes_binary"
# 1) Keep only numeric columns (safe for correlation)
df_num = df.select_dtypes(include=[np.number]).copy()

# 2) Correlation matrix
corr_matrix = df_num.corr(method="pearson")  # Pearson correlation

print("Correlation matrix shape:", corr_matrix.shape)
display(corr_matrix)

# 3) Heatmap of correlation matrix
plt.figure(figsize=(12, 9))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    square=True,
    linewidths=0.5
)
plt.title("Correlation Heatmap (Pearson)")
plt.tight_layout()
plt.show()

# 4) Correlation with the target variable (sorted)
corr_with_target = (
    corr_matrix[target_col]
    .drop(target_col)
    .sort_values(ascending=False)
)

print("\nCorrelation with target (sorted):")
display(corr_with_target)
```

```
# 5) Bar chart: correlation with target
plt.figure(figsize=(10, 4))
corr_with_target.plot(kind="bar")
plt.title(f"Feature Correlation with {target_col}")
plt.ylabel("Pearson correlation")
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.tight_layout()
plt.show()
```

3.2 MODEL DEVELOPMENT

Training the data

```
# Target
target = "Diabetes_binary"

# Features (21 features excluding target)
features = [c for c in df.columns if c != target]

X = df[features]
y = df[target]

# Split (stratify keeps class ratio consistent)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
print("Train shape:", X_train.shape, " Test shape:", X_test.shape)
print("Target distribution (train):\n", y_train.value_counts(normalize=True).sort_index())
```

This part prepares and trains the data by first defining the target variables (Diabetes-binary) and all the other columns as input features. A feature matrix (X) and target vector (y) are then constructed and divided in terms of 80/20 ratio (training and testing sets) where stratification is used to ensure equal number of class representatives in both subsets. The shape of the resulting dataset can be printed to ensure the split is correct and the normalized target distribution in the training set can be displayed to ensure that the balance of the classes has not been violated, providing a solid basis with which to conduct further model training and testing.

Training 5 Models

```
results = []
roc_curves = {}   # store (fpr, tpr, auc) for each model that supports probabilities

for name, model in models.items():
    # Train
    model.fit(X_train, y_train)

    # Predict class labels
    y_pred = model.predict(X_test)

    # Predict probabilities for ROC-AUC + ROC curve (if supported)
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
        auc = roc_auc_score(y_test, y_proba)

        fpr, tpr, _ = roc_curve(y_test, y_proba)
        roc_curves[name] = (fpr, tpr, auc)
    else:
        auc = np.nan

    # Save metrics
    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, zero_division=0),
        "Recall": recall_score(y_test, y_pred, zero_division=0),
        "F1-score": f1_score(y_test, y_pred, zero_division=0),
        "ROC-AUC": auc
    })
```

The code tests and measures five classification models along a standard pipeline so that they can be compared. Each model is trained upon the training data and applied to give prediction on the data. Predicted class probabilities with support are then computed to give the ROC-AUC score and produce false positive and true positive rates to be used in the analysis of the ROC curve. Along with ROC-AUC, accuracy, precision, recall, and F1-score are also computed and stored when each model and the controls to address the problem of zero division are taken into account. The findings are tabulated in an organized manner, which gives a holistic foundation of comparing model performance with various measures of evaluation.

3.3 MODEL EVALUATION

```
models = {

    # Logistic Regression:
    # A baseline and interpretable model commonly used in medical prediction.
    # It helps understand how different health indicators influence diabetes risk.
    "Logistic Regression": LogisticRegression(
        max_iter=1000,           # Increase iterations to ensure convergence
        class_weight="balanced" # Handle class imbalance between diabetic and non-diabetic cases
    ),

    # Decision Tree:
    # A rule-based model that splits data using decision rules.
    # Easy to interpret but may overfit on training data.
    "Decision Tree": DecisionTreeClassifier(
        random_state=42          # Ensures reproducibility of results
    ),

    # Random Forest:
    # An ensemble model made of multiple decision trees.
    # Reduces overfitting and improves prediction accuracy.
    "Random Forest": RandomForestClassifier(
        n_estimators=200,        # Number of trees in the forest
        random_state=42,         # Ensures reproducible results
        n_jobs=-1                # Use all available CPU cores
    ),

    # Gradient Boosting:
    # Builds trees sequentially to correct errors from previous trees.
    # Effective for capturing complex relationships in data.
    "Gradient Boosting": GradientBoostingClassifier(
        random_state=42
    ),

    # AdaBoost:
    # Boosting algorithm that focuses more on misclassified samples.
    # Useful for improving weak learners in classification tasks.
    "AdaBoost": AdaBoostClassifier(
        random_state=42
    )
```

Figure above shows the code to evaluate each trained classification model on the test dataset. For each model, it predicts class labels (y_pred) and predicted probabilities (y_proba) of the positive class (high diabetes risk). These measures of performance such as accuracy, precision recall, F1-score and ROC-AUC are calculated and stored in the list of the results. After that, the results are stored into the DataFrame and sorted by F1-score to compare the performance result. It is important to use variety methods since accuracy alone may hide weakness. For medical prediction, Recall is important to reduce false negatives (missing high-risk individuals), while F1-score is a good way to balance precision and recall.

```
    # Confusion matrix plot
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(values_format='d')
    plt.title(f"Confusion Matrix: {name}")
    plt.tight_layout()
    plt.show()
```

This code will generate a few confusions matrix for each model based on the predicted labels. The confusion matrix will show the number of True Positive (TP), True Negatives (TN), False Positives, and False Negatives (FN). Based on the plotted confusion matrix, the model sorts of people into low-risk and high-risk groups. False Negatives cases are important in the medical area since these cases depict the high-risk population that is misdiagnosed as low risk. Misclassification can cause late detection and treatment which could result in serious health problems. Therefore, an effective predictive model should minimize false negatives while maintaining a reasonable number of false positives to ensure reliable early detection.

```
print("=== Model Comparison Table (sorted by F1-score) ===")
display(results_df)

# Identify the best model
best_model_row = results_df.iloc[0]
best_model_name = best_model_row["Model"]
best_model = models[best_model_name]

print("\n=== Best Model Based on F1-score ===")
print("Model:", best_model_name)
print("Accuracy:", round(best_model_row["Accuracy"], 4))
print("Precision:", round(best_model_row["Precision"], 4))
print("Recall:", round(best_model_row["Recall"], 4))
print("F1-score:", round(best_model_row["F1-score"], 4))
print("ROC-AUC:", round(best_model_row["ROC-AUC"], 4))

# Optional: Highlight best row in table
def highlight_best(row):
    return ["background-color: blue"] * len(row) if row["Model"] == best_model_name else [""] * len(row)

display(results_df.style.apply(highlight_best, axis=1))
```

The code selects the most performing model according to the highest F1-scrore of results as the best performance model. Then, it is shown with its metric values. This step helps with “decision making” by choosing the most reliable model to predict the diabetes risk. This study will use F1-score since it gives a balanced recall and precision measure. Recall is important to identify high-risk individuals, while precision helps to minimize excessive false alarms. The F1-score effectively balances both aspects and therefore, it is a suitable score to use in comparing models.

```python
# ROC Curve Plot (all models that support predict_proba)

if len(roc_curves) > 0:
    plt.figure(figsize=(7, 6))

    for name, (fpr, tpr, auc) in roc_curves.items():
        plt.plot(fpr, tpr, label=f"{name} (AUC={auc:.3f})")

    # Diagonal baseline
    plt.plot([0, 1], [0, 1], linestyle="--", label="Random guess")

    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curves (Model Comparison)")
    plt.legend()
    plt.tight_layout()
    plt.show()
else:
    print("No ROC curves plotted: none of the models support predict_proba().")
```

This code plots ROC curves for all models that have the predict_proba(). The ROC curve shows the trade-off between True Positive Rate (TPR) and False Positive Rate (FPR) at different classification thresholds. ROC-AUC simplifies this curve into one value which a higher ROC-AUC shows the production of a better ability to distinguish between high and low risk cases. The model with the ROC curve closest to the top-left corner and has the largest Area Under Cover (AUC) has better classification performance since it has a higher true positive rate and a lower false positive rate at different decision thresholds.

4.0 TESTING AND VALIDATION

```python
def evaluate_model(name, model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)

    # Predictions
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    # Probabilities for ROC-AUC (if available)
    y_proba_train = model.predict_proba(X_train)[:, 1] if hasattr(model, "predict_proba") else None
    y_proba_test  = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

    # Metrics (train)
    train_metrics = {
        "Accuracy_train": accuracy_score(y_train, y_pred_train),
        "Precision_train": precision_score(y_train, y_pred_train, zero_division=0),
        "Recall_train": recall_score(y_train, y_pred_train, zero_division=0),
        "F1_train": f1_score(y_train, y_pred_train, zero_division=0),
        "ROC_AUC_train": roc_auc_score(y_train, y_proba_train) if y_proba_train is not None else np.nan
    }

    # Metrics (test)
    test_metrics = {
        "Accuracy_test": accuracy_score(y_test, y_pred_test),
        "Precision_test": precision_score(y_test, y_pred_test, zero_division=0),
        "Recall_test": recall_score(y_test, y_pred_test, zero_division=0),
        "F1_test": f1_score(y_test, y_pred_test, zero_division=0),
        "ROC_AUC_test": roc_auc_score(y_test, y_proba_test) if y_proba_test is not None else np.nan
    }

    # Overfitting check (gap between train and test F1)
    gap = train_metrics["F1_train"] - test_metrics["F1_test"]

    return { "Model": name, **train_metrics, **test_metrics, "F1_gap(train-test)": gap }, model
```

```python
baseline_results = []
fitted_models = {}

for name, model in models.items():
    row, fitted = evaluate_model(name, model, X_train, y_train, X_test, y_test)
    baseline_results.append(row)
    fitted_models[name] = fitted

baseline_df = pd.DataFrame(baseline_results).sort_values(by="F1_test", ascending=False)
baseline_df
```

This code compares how well the models' performance in training the data compares to testing the data using accuracy or F1-score. When training performance is significantly higher than test performance, then the model is probably overfitting. When the training performance and the test performance are low, then the model can be underfitting. This step is important to validate whether the model generalizes quality of the model to the unknown data. It is necessary to guarantee a good performance to ensure the diabetes prediction works well.

```
param_grids = {
    "Logistic Regression": {
        "C": [0.01, 0.1, 1, 10],
        "penalty": ["l2"],
        "solver": ["lbfgs"]
    },
    "Decision Tree": {
        "max_depth": [None, 3, 5, 10],
        "min_samples_split": [2, 10, 20],
        "min_samples_leaf": [1, 5, 10]
    },
    "Random Forest": {
        "n_estimators": [200],
        "max_depth": [None, 10, 20],
        "min_samples_split": [2, 10],
        "min_samples_leaf": [1, 5]
    },
    "Gradient Boosting": {
        "n_estimators": [100, 200],
        "learning_rate": [0.05, 0.1],
        "max_depth": [2, 3]
    },
    "AdaBoost": {
        "n_estimators": [100, 200],
        "learning_rate": [0.05, 0.1, 0.5]
    }
}
```

As the project is implementing a binary classification task, standard Ridge Regression is not applicable in this project. However, in Logistic Regression (penalty = “l2”), the ridge is used to regularization. It helps to reduce the overfitting by penalizing large coefficients, which makes the model stable to generalize.

```
best_models = {}
grid_summary = []

for name, model in models.items():
    print(f"\nRunning GridSearchCV for: {name}")

    if name not in param_grids:
        raise KeyError(f"param_grids does not contain a grid for '{name}'. Add it to param_grids first.")

    grid = GridSearchCV(
        estimator=model,
        param_grid=param_grids[name],
        scoring="f1",      # Good for imbalanced classification (balances precision & recall)
        cv=cv,
        n_jobs=-1
    )

    grid.fit(X_train, y_train)

    best_models[name] = grid.best_estimator_

    grid_summary.append({
        "Model": name,
        "Best_Params": grid.best_params_,
        "Best_CV_F1": grid.best_score_
    })
```

```
# Create GridSearch summary dataframe
grid_df = pd.DataFrame(grid_summary).sort_values(by="Best_CV_F1", ascending=False).reset_index(drop=True)

# 2) Show FULL GridSearch results (no truncation)
pd.set_option("display.max_colwidth", None)
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)


# 3) Identify the best tuned model (by CV F1-score)
best_grid_row = grid_df.iloc[0]
best_model_name = best_grid_row["Model"]
best_model = best_models[best_model_name]

print("\n=== Best Tuned Model (GridSearchCV) ===")
print("Model:", best_model_name)
print("Best Parameters:", best_grid_row["Best_Params"])
print("Best CV F1:", round(float(best_grid_row["Best_CV_F1"]), 4))
```

These codes use GridSearchCV to test different combinations of hyperparameters, such as the number of estimators, the learning rate, and the depth of tree. By using cross validation, Grid Search finds the hyperparameter setting with the highest validation, normally through ROC-AUC. The result of the tuned model shows better predictive performance over the default model, which contributes to the need for the better model in a more evidence based.

```
best_name = refined_df.iloc[0]["Model"]
final_model = best_models[best_name]

print(" >> Final selected model:", best_name)

final_model.fit(X_train, y_train)
y_pred = final_model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm).plot(values_format="d")
plt.title(f"Confusion Matrix - {best_name}")
plt.show()

# ROC curve (only if predict_proba exists)
if hasattr(final_model, "predict_proba"):
    RocCurveDisplay.from_estimator(final_model, X_test, y_test)
    plt.title(f"ROC Curve - {best_name}")
    plt.show()
```

This code measures the best estimator of tuning on the test set that was achieved by Grid Search. It reports updates performance metrics and visual outputs, such as the confusion matrix and ROC-AUC. This step is the last step to validate the process. It ensures that the tuned model performs better and also reliable to use on the unseen data. The drawbacks of the baseline Decision Tree model include overfitting and lack of ability to generalize to unknown data. The problems can be overcome by using Gradient Boosting as better model and subsequent refinement of this model with the help of hyperparameter tuning. The improved model makes more stable predictions and shows a higher diabetes risk prediction performance, which is a valid solution to the problem of inaccurate and inconsistent risk prediction.

Result and Interpretation

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
Gradient Boosting	0.754226	0.733619	0.798274	0.764582	0.829702
Logistic Regression	0.747295	0.738082	0.766587	0.752064	0.824979
AdaBoost	0.744819	0.734580	0.766587	0.750242	0.822319
Random Forest	0.734918	0.718343	0.772811	0.744582	0.811973
Decision Tree	0.649975	0.652474	0.641675	0.647029	0.650314

The results table of the performance as in Table were obtained only based on the testing of (X_test, y_test), which represents unseen data. Using the test set ensures that the metrics give an unbiased assessment of the generalization capabilities and show how well they can predict the diabetes risk.

Accuracy. Precision, recall, F1-scrore and ROC-AUC were used to evaluate these models. The results were sorted based on the F1-score since it helps to balance between precision and recall. In medical applications, recall is important in identifying the high-risk people, whereas precision can help to reduce unnecessary false alarms. Hence, F1-score was chosen as the main feature of model comparison.

The Gradient Boosting model recorder the best F1-score (0.7646), which means that it is the best model for classification. The value of recall (0.7983) indicates that the model is successful at identifying high-risk people, whereas the precision is reasonable (0.7336). The baseline model, Decision Tree, on the other hand, had the lower F1-score (0.6740) and ROC-AUC (0.6503) values, which indicates its inability to generalize and overfitting.

Model	Pros	Cons
Decision Tree (Baseline)	• Simple and easy to interpret • Fast training time • Useful for understanding feature-based decision rules	• Prone to overfitting • Poor generalization to unseen data • Lower recall leading to missed high-risk cases
Gradient Boosting (Improved)	• Higher predictive accuracy and F1-score • Better generalization through ensemble learning • Higher recall, reducing false negatives • Strong discrimination ability (high ROC-AUC)	• More computationally expensive • Less interpretable than a single tree • Requires hyperparameter tuning

Overall, the test-set analysis suggests that Gradient Boosting is more reliable and balanced in predicting the diabetes risk compared to the baseline model and the other model. Consequently, the Gradient Boosting was chosen as the final model in predicting the diabetes risk.

5.0 CONCLUSION

This project was able to illustrate how data analysis and machine learning methods can be used to determine the risk of diabetes based on health-related survey data in the BRFSS 2015 dataset. The valuable health indicators related to diabetes were identified and analyzed through proper data preprocessing, exploration, data analysis, and the development of the model. Several classification models have been implemented and compared, which are Logistic Regression, Decision Tree, Random Forest, AdaBoost, and Gradient Boosting. The Decision Tree model was first used as a benchmark to learn the data structure and fundamental decision rules. But it exhibited weaknesses in overfitting and the inability to extrapolate to unknown data. Gradient Boosting as a better model was proposed to deal with this problem. According to such evaluation measures as accuracy, precision, recall, F1-score, and ROC-AUC, the gradient Boosting model performed the best in general, especially in F1-score and recall. This means that the model can be useful in predicting high-risk people and has a good balance of false positives and negatives, which is vital in medical prediction activities. Further analysis of features showed that the variables like Body Mass Index (BMI), General Health (GenHlth), High Blood Pressure (HighBP), Age, and Difficult Walking (DiffWalk) are among the most effective contributing factors of diabetes risk. These results not only confirm the current medical understanding but also indicate the applicability of information-based methods in the risk estimation of health conditions. To sum up, the application of Gradient Boosting as a better model of machine learning gives a stable and efficient solution in predicting the risk of diabetes. The findings indicate that machine learning can support early detection and preventive health-related decisions with the help of basic demographic and lifestyle data. It could be applicable in future work to add more clinical or longitudinal data to better predict the accuracy as well as model interpretability.

REFERENCES

Ramli, A., & Taher, S. (2008). Managing chronic diseases in the Malaysian primary health care - a need for change. *Malaysian family physician : the official journal of the Academy of Family Physicians of Malaysia*, *3*(1), 7–13.

World Health Organization. (2024, December 23). *Noncommunicable diseases*. World Health Organization. https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases