Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Intelligent Systems Laboratory – ENCS5141

# Case Study #1—Data Cleaning and Feature Engineering for the Diabetes Dataset and Comparative Analysis of Classification Techniques

Prepared By: Mohammad Shreteh - 1201369

Section#: 2

Instructor: Dr. Aziz Qaroush

Assistant teacher: Eng. Ahmad Abbas

Date: 30/11/2024

## Abstract :

This study presents an analysis of the “Diabetes” dataset. The first part focuses on exploring the structure of Set the data and understand its properties, then check for quality issues such as missing items Values and outliers and how to deal with them, as well as partitioning the data set (training and testing sets), the Random Forest model is trained on both raw and pre-processed data to measure improvements in accuracy, precision and recall.

In the second part, we compared three models: Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP). We trained these models on the data we had and evaluated the models based on several foundations: classification accuracy, accuracy, recall, F-measure, and computational efficiency. The goal of this is to find out the best model.

# Table of Contents

# Table of figures

# 1. Introduction

## 1.1 Motivation

The motivation behind this study is to understand data preprocessing and model evaluation in the context of machine learning, especially for bike-sharing demand forecasting. By understanding and analyzing the “giant” dataset and applying preprocessing to it, we aim to improve the performance of the model. There is often a significant amount of noise, missing data, and irrelevant features that can negatively impact the performance of machine learning models. This task aims to provide insight into how effective preprocessing is and which model makes the best predictions.

## 1.2 Background

When preparing data for modeling, methods like as feature selection, scaling, downscaling, handling missing values and outliers, and feature selection are essential since they all affect the dataset's quality and, in turn, the model's performance. The effectiveness of machine learning models is influenced by data preprocessing.

We will compare three techniques: Random Forest (RF), Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP). As each technique has features that distinguish it from others, for example, the Random Forest (RF) is characterized by being very suitable for high-dimensional data, and the SVM technique is characterized by being suitable in high-dimensional spaces and with smaller data sets. MLP, which is a type of artificial neural network, is particularly powerful in capturing relationships. complex data.

## 1.3 Objective

**The primary objective of this assignment is two-fold:**

**Data cleaning and engineering features:**

This study's first objective is to investigate and assess various data preparation methods used on the "diabetes" dataset in order to comprehend the dataset's structure. resolved problems with data quality, carried out feature selection, categorical variable coding, and segmentation. Divide data into training and test sets, use reduction techniques to make a data set simpler, and scale characteristics to guarantee consistency.

**Comparative analysis of classification techniques:**

This study's second objective is to evaluate the performance of three well-known classification methods on a preprocessed dataset: Random Forest (RF), Support Vector Machine (SVM), and Multilayer Perceptron (MLP). For the particular data set, we will assess various methods based on a number of criteria, including accuracy, to determine which model best balances computational efficiency with prediction accuracy.

# 2. Procedure and Discussion

## 2.1 Load the Diabetes dataset

In this step, the Diabetes dataset is loaded, and the header of the dataset is shown in below.

DataSet Describe :

| | Target | Genetic Markers | Autoantibodies | Family History | Environmental Factors | Insulin Levels | Age | BMI | Physical Activity | Dietary Habits | ... | Pulmonary Function | Cystic Fibrosis Diagnosis | Steroid Use History | Genetic Testing | Neurological Assessments | Liver Function Tests |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Steroid-Induced Diabetes | Positive | Negative | No | Present | 40 | 44 | 38 | High | Healthy | ... | 76 | No | No | Positive | 3 | Normal |
| 1 | Neonatal Diabetes Mellitus (NDM) | Positive | Negative | No | Present | 13 | 1 | 17 | High | Healthy | ... | 60 | Yes | No | Negative | 1 | Normal |
| 2 | Prediabetic | Positive | Positive | Yes | Present | 27 | 36 | 24 | High | Unhealthy | ... | 80 | Yes | No | Negative | 1 | Abnormal |
| 3 | Type 1 Diabetes | Negative | Positive | No | Present | 8 | 7 | 16 | Low | Unhealthy | ... | 89 | Yes | No | Positive | 2 | Abnormal |
| 4 | Wolfram Syndrome | Negative | Negative | Yes | Present | 17 | 10 | 17 | High | Healthy | ... | 41 | No | No | Positive | 1 | Normal |

Figure 2.1.1: Header of the Diabetes dataset

Initially, the dataset has 34 columns (features): such as target, Genetic Markers, Family History , Age, BMI and etc. Also, there is no missing values in the dataset.

## 2.2 Data Exploration:

This step's objectives are to comprehend the dataset's properties, structure, and statistics while also gaining new insights.

I used the panda’s libraries. Info (), this method displays the number of entries and columns, each column with the number of non-null entries and its type.

```
Data columns (total 34 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   Target                         70000 non-null  object
 1   Genetic Markers                70000 non-null  object
 2   Autoantibodies                 70000 non-null  object
 3   Family History                 70000 non-null  object
 4   Environmental Factors          70000 non-null  object
 5   Insulin Levels                 70000 non-null  int64 
 6   Age                            70000 non-null  int64 
 7   BMI                            70000 non-null  int64 
 8   Physical Activity              70000 non-null  object
 9   Dietary Habits                 70000 non-null  object
 10  Blood Pressure                 70000 non-null  int64 
 11  Cholesterol Levels             70000 non-null  int64 
 12  Waist Circumference            70000 non-null  int64 
 13  Blood Glucose Levels           70000 non-null  int64 
 14  Ethnicity                      70000 non-null  object
 15  Socioeconomic Factors          70000 non-null  object
 16  Smoking Status                 70000 non-null  object
 17  Alcohol Consumption            70000 non-null  object
 18  Glucose Tolerance Test         70000 non-null  object
 19  History of PCOS                70000 non-null  object
 20  Previous Gestational Diabetes  70000 non-null  object
 21  Pregnancy History              70000 non-null  object
 22  Weight Gain During Pregnancy   70000 non-null  int64 
 23  Pancreatic Health              70000 non-null  int64 
 24  Pulmonary Function             70000 non-null  int64 
 25  Cystic Fibrosis Diagnosis      70000 non-null  object
 26  Steroid Use History            70000 non-null  object
 27  Genetic Testing                70000 non-null  object
 28  Neurological Assessments       70000 non-null  int64 
 29  Liver Function Tests           70000 non-null  object
 30  Digestive Enzyme Levels        70000 non-null  int64 
 31  Urine Test                     70000 non-null  object
 32  Birth Weight                   70000 non-null  int64 
 33  Early Onset Symptoms           70000 non-null  object
dtypes: int64(13), object(21)
memory usage: 18.2+ MB
```

Figure 2.2.1: Dataset information

From the above picture, we can see:

The number of columns (features) is 34.

The number of entries is 70K.

The.df.describe() function provides a summary of the dataset's statistics, including the count, mean, standard deviation, minimum, 25th, 50th, 75th, and maximum values for each numeric column (for numeric columns with datatype int64/float64).

DataSet Describe :

| | Insulin Levels | Age | BMI | Blood Pressure | Cholesterol Levels | Waist Circumference | Blood Glucose Levels | Weight Gain During Pregnancy | Pancreatic Health |
|---|---|---|---|---|---|---|---|---|---|
| count | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 | 70000.000000 |
| mean | 21.607443 | 32.020700 | 24.782943 | 111.339543 | 194.867200 | 35.051657 | 160.701657 | 15.496414 | 47.564243 |
| std | 10.785852 | 21.043173 | 6.014236 | 19.945000 | 44.532466 | 6.803461 | 48.165547 | 9.633096 | 19.984683 |
| min | 5.000000 | 0.000000 | 12.000000 | 60.000000 | 100.000000 | 20.000000 | 80.000000 | 0.000000 | 10.000000 |
| 25% | 13.000000 | 14.000000 | 20.000000 | 99.000000 | 163.000000 | 30.000000 | 121.000000 | 7.000000 | 32.000000 |
| 50% | 19.000000 | 31.000000 | 25.000000 | 113.000000 | 191.000000 | 34.000000 | 152.000000 | 16.000000 | 46.000000 |
| 75% | 28.000000 | 49.000000 | 29.000000 | 125.000000 | 225.000000 | 39.000000 | 194.000000 | 22.000000 | 64.000000 |
| max | 49.000000 | 79.000000 | 39.000000 | 149.000000 | 299.000000 | 54.000000 | 299.000000 | 39.000000 | 99.000000 |

Figure 2.2.2: Dataset statistics

For example " BMI " from the above picture we can see the Count: 70,000 observations, Mean: 24.78, Std: 6.01, Min: 12 , 25%: 20 , Median: 25 , 75%: 29 , Max: 39.

### 2.3 Data Visualization:

The goal of this section is to visualize the features in order to comprehend their relationship.

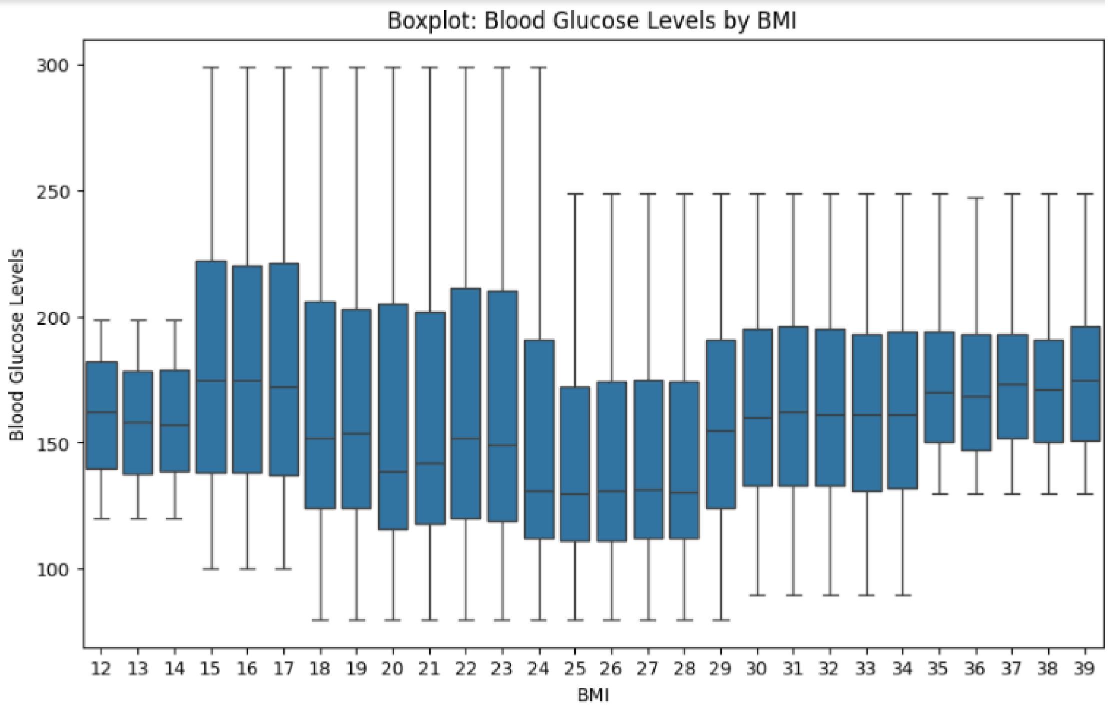


Figure 2.3.1: BMI distribution boxplot

The boxplot shows the distribution of blood glucose levels across different BMI groups. The median is rather steady, with more volatility at lower BMI levels, despite the fact that glucose levels vary significantly across all BMI categories. While mid-range BMI categories (20–25) display tighter glucose level distributions, extreme BMI categories show higher variability and outliers. This implies that blood glucose levels are greatly influenced by variables other than BMI, such as metabolic or lifestyle factors. Further research is needed to fully comprehend the relationship.


Figure 2.3.2: Age distribution scatter-plot

The scatter plot shows how blood glucose levels and age are related, along with the variables of smoking status (smoker or non-smoker) and eating habits (good or unhealthy). There is no discernible pattern in blood glucose levels as people age, and they vary greatly across all age categories. Unlike healthy eating patterns (blue dots), unhealthy eating patterns (orange dots) seem to be associated with higher glucose levels. Additionally, smokers are more likely to have elevated blood glucose levels (black markers), which may indicate a role for lifestyle variables. All things considered, the data points to intricate relationships between nutrition, smoking, aging, and glucose management.

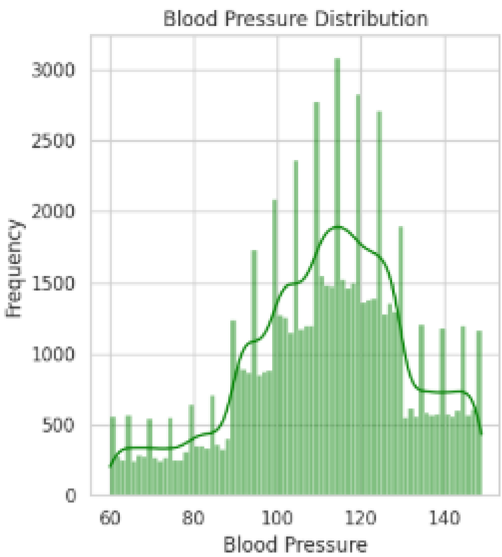


Figure 2.3.3: blood pressure distribution histogram's

The histogram's bell-shaped curve, which shows how blood pressure varies among people, denotes a normal distribution. The majority of the results center on the average blood pressure of the sample, which is 110 mmHg. The data shows a slow ascent and decline with some anomalies at the lower and higher ends. The density curve overlay validates the distribution's smooth, unimodal character. The study might be able to pinpoint blood pressure trends for medical monitoring or treatment.

## 2.4 Data Cleaning:

The goal of this part is to locate and deal with any missing values and possible outliers in the dataset. Furthermore, it entails identifying any outliers and, if required, putting in place suitable management procedures.

### 2.4.1 Detecting Missing Values

I used isnull().sum(). to check the number of missing values in each column of the “Diabetes” dataset. We notice from the picture below that the dataset does not contain any missing values.

| | |
|---|---|
| Target | 0 |
| Genetic Markers | 0 |
| Autoantibodies | 0 |
| Family History | 0 |
| Environmental Factors | 0 |
| Insulin Levels | 0 |
| Age | 0 |
| BMI | 0 |
| Physical Activity | 0 |
| Dietary Habits | 0 |
| Blood Pressure | 0 |
| Cholesterol Levels | 0 |
| Waist Circumference | 0 |
| Blood Glucose Levels | 0 |
| Ethnicity | 0 |
| Socioeconomic Factors | 0 |
| Smoking Status | 0 |
| Alcohol Consumption | 0 |
| Glucose Tolerance Test | 0 |
| History of PCOS | 0 |
| Glucose Tolerance Test | 0 |
| History of PCOS | 0 |
| Previous Gestational Diabetes | 0 |
| Pregnancy History | 0 |
| Weight Gain During Pregnancy | 0 |
| Pancreatic Health | 0 |
| Pulmonary Function | 0 |
| Cystic Fibrosis Diagnosis | 0 |
| Steroid Use History | 0 |
| Genetic Testing | 0 |
| Neurological Assessments | 0 |
| Liver Function Tests | 0 |
| Digestive Enzyme Levels | 0 |
| Urine Test | 0 |
| Birth Weight | 0 |
| Early Onset Symptoms | 0 |

Figure 2.4.1.1: number of missing value in dataset

Some datasets may include completely empty entries, so we must check for it.

```
Number of empty records = 0
Target  Genetic Markers  Autoantibodies  Family History  Environmental Factors  Insulin Levels  Age  BMI  Physical Activity  Dietary Habits  ...  Pulmonary Function  Cystic Fibrosis Diagnosis  Steroid Use History  Genetic Testing  Neurological Assessments  F
0 rows × 34 columns
```

Figure 2.4.1.1: Addressing rows that are completely empty

From above picture we can see The dataset doesn't include any empty rows

### 2.4.2 Handling Missing Values

I decided to use imputation for missing values in order to protect crucial information and avoid bias resulting from information loss. to maintain the core tendency of the data while strengthening its resistance to outliers. In order to preserve the overall distribution and characteristics of categorical variables, the mode, or the most frequent value, was used to replace missing values for non-numeric columns (such object or category). While maintaining the dataset's integrity, this approach lessened the impact of missing data on the analysis and modeling process.

```
Median values used for imputation:
Insulin Levels: 19.0
Age: 31.0
BMI: 25.0
Blood Pressure: 113.0
Cholesterol Levels: 191.0
Waist Circumference: 34.0
Blood Glucose Levels: 152.0
Weight Gain During Pregnancy: 16.0
Pancreatic Health: 46.0
Pulmonary Function: 72.0
Neurological Assessments: 2.0
Digestive Enzyme Levels: 48.0
Birth Weight: 3103.0

Mean values used for imputation:
Insulin Levels: 21.607442857142857
Age: 32.0207
BMI: 24.782942857142856
Blood Pressure: 111.33954285714286
Cholesterol Levels: 194.8672
Waist Circumference: 35.051657142857145
Blood Glucose Levels: 160.70165714285713
Weight Gain During Pregnancy: 15.496414285714286
Pancreatic Health: 47.56424285714286
Pulmonary Function: 70.26467142857143
Neurological Assessments: 1.8041571428571428
Digestive Enzyme Levels: 46.42052857142857
Birth Weight: 3097.0610714285713
<ipython-input-8-8de4961f1968>:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perfo
```

Figure 2.4.2.1: Median and mean values used to fill in missing data

| | |
|---|---|
| Target | 0 |
| Genetic Markers | 0 |
| Autoantibodies | 0 |
| Family History | 0 |
| Environmental Factors | 0 |
| Insulin Levels | 0 |
| Age | 0 |
| BMI | 0 |
| Physical Activity | 0 |
| Dietary Habits | 0 |
| Blood Pressure | 0 |
| Cholesterol Levels | 0 |
| Waist Circumference | 0 |
| Blood Glucose Levels | 0 |
| Ethnicity | 0 |
| Socioeconomic Factors | 0 |
| Smoking Status | 0 |
| Alcohol Consumption | 0 |
| Glucose Tolerance Test | 0 |
| History of PCOS | 0 |
| Glucose Tolerance Test | 0 |
| History of PCOS | 0 |
| Previous Gestational Diabetes | 0 |
| Pregnancy History | 0 |
| Weight Gain During Pregnancy | 0 |
| Pancreatic Health | 0 |
| Pulmonary Function | 0 |
| Cystic Fibrosis Diagnosis | 0 |
| Steroid Use History | 0 |
| Genetic Testing | 0 |
| Neurological Assessments | 0 |
| Liver Function Tests | 0 |
| Digestive Enzyme Levels | 0 |
| Urine Test | 0 |
| Birth Weight | 0 |
| Early Onset Symptoms | 0 |

Figure 2.4.2.2: number of missing value after drop

### 2.4.3 Detecting Outliers

Finding outliers is the main goal of this section, and there are several ways to do so. We will summarize this section using sns.boxenplot(), which is comparable to a box plot but offers additional quantiles for a more thorough picture of the data distribution.

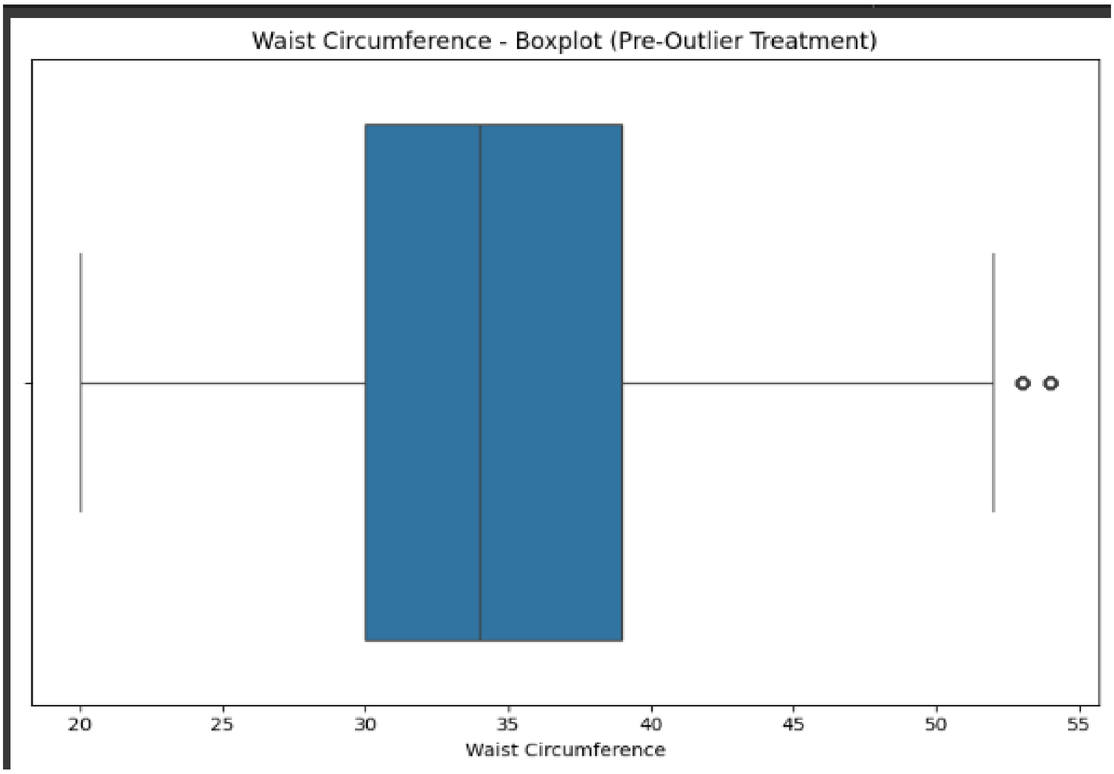


Figure 2.4.3.1: Boxplot of waist circumference column before handling outliers

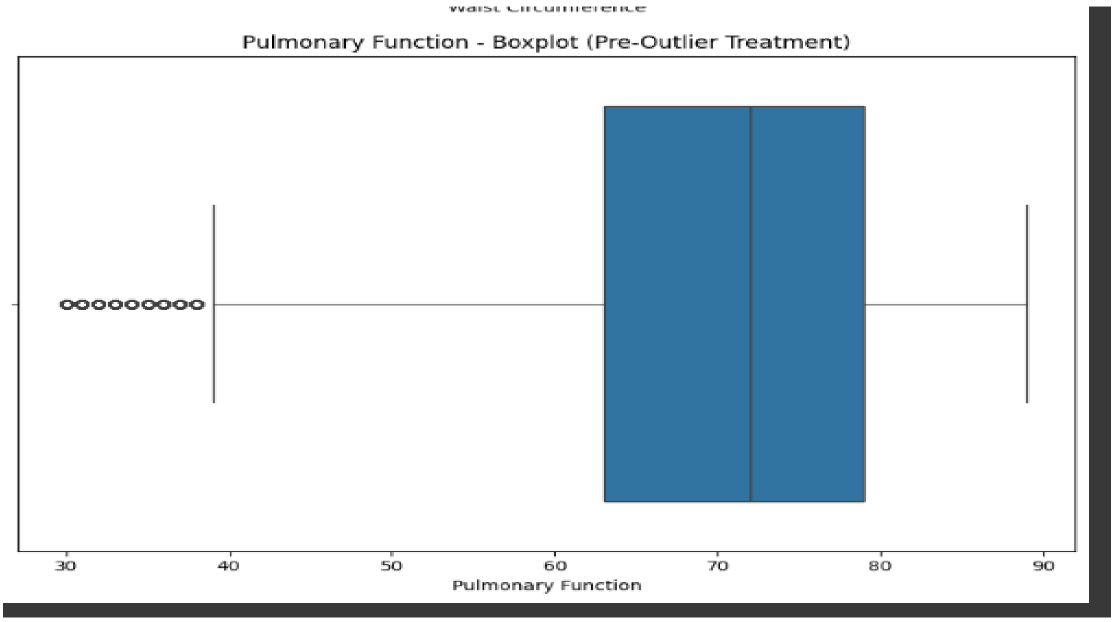


Figure 2.4.3.2: Boxplot of pulmonary function column before handling outliers

I created boxplots for the "Waist Circumference" and "Pulmonary Function" columns so that the visualization highlights the distribution of data points, including any extreme values that might differ significantly from the rest of the data, and so identifies any possible outliers.

Figure above shows that there are outliers for the features of Waist Circumference and Pulmonary Function, which must be eliminated. IQR will be checked in order to compare the results of the two approaches.

```
25th Percentile for Waist Circumference: 30.0
50th Percentile for Waist Circumference: 34.0
75th Percentile for Waist Circumference: 39.0
Interquartile Range (IQR) for Waist_Circumference: 9.0
Lower Bound for Waist Circumference = 16.5,
 and Upper Bound for Waist Circumference = 52.5
```

Figure 2.4.3.3: Waist Circumference percentiles and IQR and lower and upper bounds

```
25th Percentile for Pulmonary Function: 63.0
50th Percentile for Pulmonary Function: 72.0
75th Percentile for Pulmonary Function: 79.0
Interquartile Range (IQR) for Pulmonary Function: 16.0
Lower Bound for Pulmonary Function = 39.0,
 and Upper Bound for Pulmonary Function = 103.0
```

Figure 2.4.3.4: Pulmonary Function percentiles and IQR and lower and upper bounds

I determined the number of outliers for each characteristic by taking into account the fact that a data point is deemed an outlier if it comes above the upper border or below the lower threshold:

```
Number of outliers based on the Interquartile Range for Pulmonary Function: 1206 (1.72%)
Number of outliers based on the Interquartile Range for Waist Circumference: 522 (0.75%)
```

Figure 2.4.3.5: number of outliers based on the IQR before handling

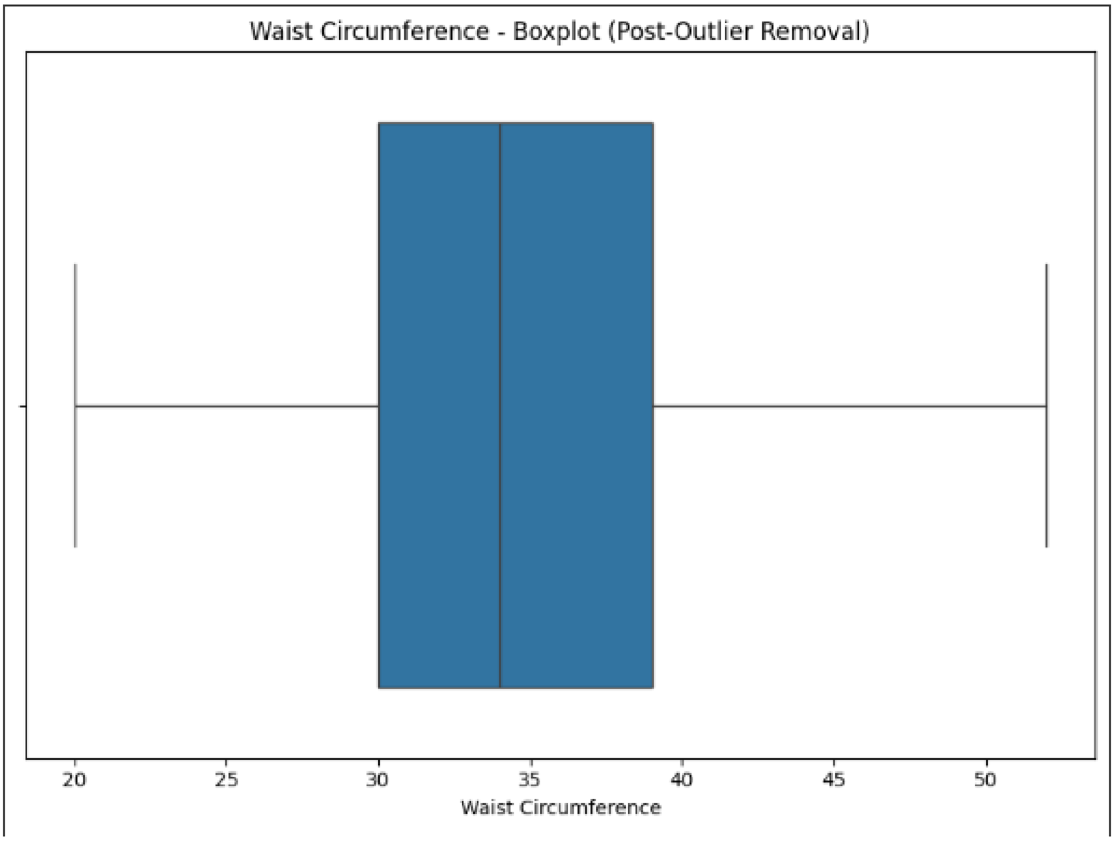


Figure 2.4.3.6: Boxplot of waist circumference column after handling outliers

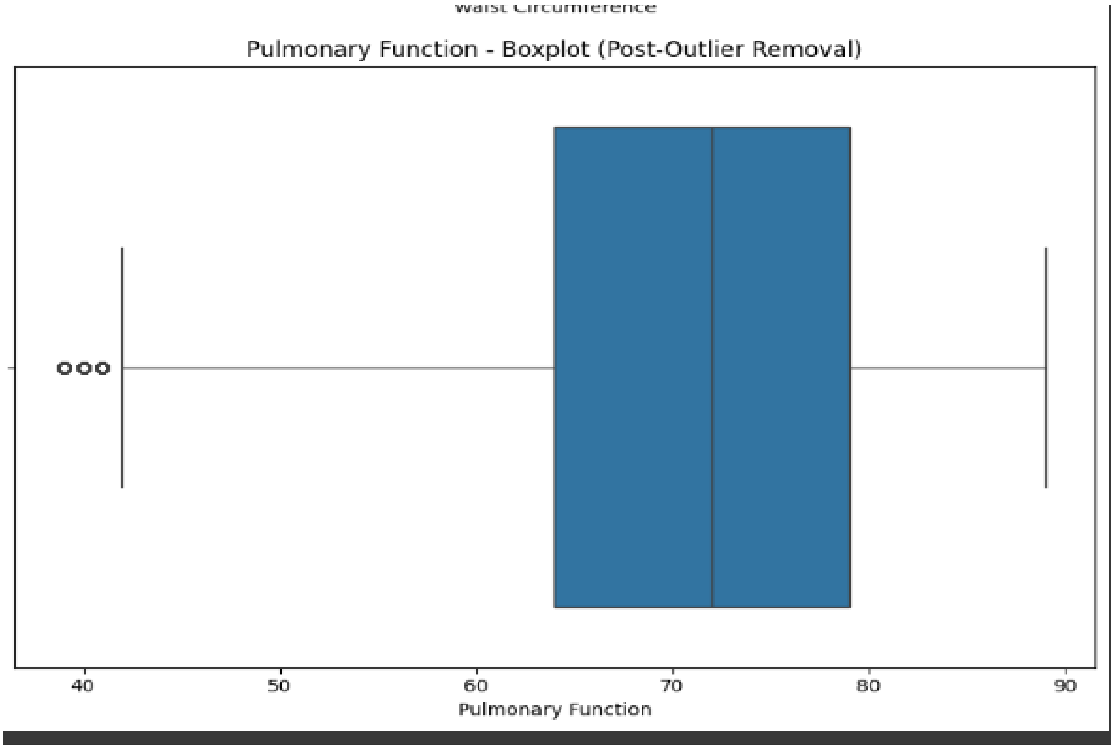


Figure 2.4.3.7: Boxplot of pulmonary function column after handling outliers

```
Number of outliers based on the Interquartile Range for Pulmonary Function: 0 (0.00%)
Number of outliers based on the Interquartile Range for Waist Circumference: 0 (0.00%)
```

Figure 2.4.3.8: number of outliers based on the IQR after handling

I used outlier removal based on the determined constraints after detecting the outliers. As seen in Figure above where the number of outliers was lowered to zero, the procedure effectively managed the outliers, guaranteeing that no extreme values remained in the dataset. This stage improved the quality of the data, increasing its dependability for further modeling and analysis.

## 2.5 Feature Engineering:

### 2.5.1 Analysis of Variance

Since characteristics with little to no variability are unlikely to enhance a machine learning model's prediction abilities, columns were examined for variance. Target_Wolfram Syndrome has the lowest variance, at ~0.0566, suggesting that there is little variation amongst the data. However, as every element was considered essential for the study, no columns were eliminated.

```
Column variances: [4.40000000e+01 7.90000000e+01 2.70000000e+01 8.90000000e+01
 1.99000000e+02 3.20000000e+01 2.19000000e+02 3.90000000e+01
 8.90000000e+01 5.00000000e+01 4.62522695e-01 8.90000000e+01
 2.99900000e+03 7.36275593e-02 7.21481380e-02 7.06501320e-02
 7.47208060e-02 7.29379316e-02 7.25432545e-02 7.38120525e-02
 6.47718372e-02 7.34060400e-02 7.28023109e-02 7.18515123e-02
 7.28393035e-02 5.65634059e-02 2.49996594e-01 2.49996594e-01
 2.49998761e-01 2.49998761e-01 2.49994912e-01 2.49994912e-01
 2.49999253e-01 2.49999253e-01 2.21631596e-01 2.22347397e-01
 2.22682503e-01 2.49999706e-01 2.49999706e-01 2.49999997e-01
 2.49999997e-01 2.22147308e-01 2.22000444e-01 2.22517633e-01
 2.49999506e-01 2.49999506e-01 2.21710487e-01 2.22759917e-01
 2.22191292e-01 2.49983419e-01 2.49983419e-01 2.49998825e-01
 2.49998825e-01 2.49999855e-01 2.49999855e-01 2.49983538e-01
 2.49983538e-01 2.49997811e-01 2.49997811e-01 2.49995173e-01
 2.49995173e-01 2.49980042e-01 2.49980042e-01 2.49999887e-01
 2.49999887e-01 1.86971585e-01 1.86839134e-01 1.87719516e-01
 1.88462983e-01 2.49999253e-01 2.49999253e-01]
Lowest variance: 0.056563405861451976
Column with lowest variance: Target_Wolfram Syndrome
```

Figure 2.5.1.1: Variations of each feature in an array

### 2.5.2 Scaling and Normalization

Numerical data including age, blood pressure, and BMI were scaled to a continuous range of 0 to 1 using min-max scaling. This phase ensures that each feature contributes equally and keeps features with larger numerical ranges from affecting the model.
When converting numbers to a uniform scale, scaling maintained the distribution features, as seen by the original and scaled data histograms (such as those for age and BMI).

```
Range of Scaled Data:
                      min  max
Age                   0.0  1.0
BMI                   0.0  1.0
Blood Pressure        0.0  1.0
Cholesterol Levels    0.0  1.0
Blood Glucose Levels  0.0  1.0
Waist Circumference   0.0  1.0
```

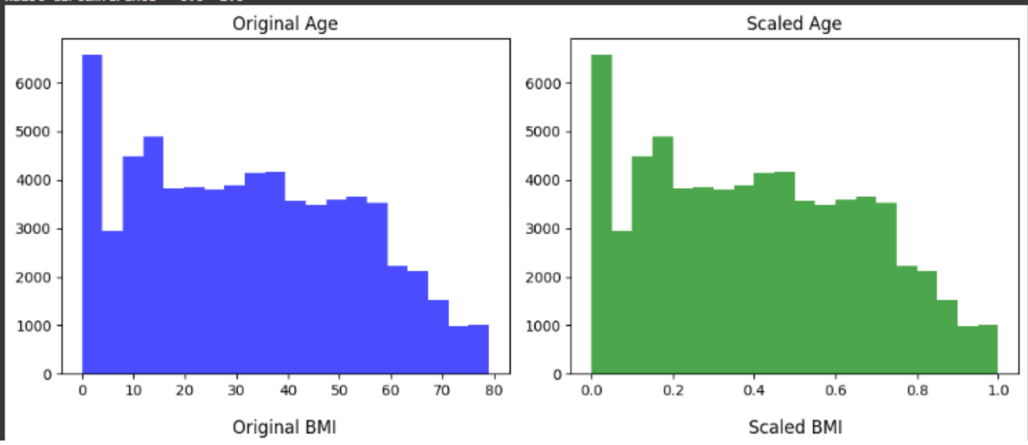


Figure 2.5.2.1: comparison between the original data and the data after the scaling process

## 2.5.3 Encoding the data and sorting it

To convert category data into numerical representations appropriate for machine learning algorithms, methods such as one-hot encoding were used. By removing ordinal relationship bias, this phase ensures that the categorical data adds anything significant to the model.

| | Insulin Levels | Age | BMI | Blood Pressure | Cholesterol Levels | Waist Circumference | Blood Glucose Levels | Weight Gain During Pregnancy | Pancreatic Health | Pulmonary Function | ... | Genetic Testing_Negative | Genetic Testing_Positive | Liver Function Tests_Abnormal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 40 | 44 | 38 | 124 | 201 | 50 | 168 | 18 | 36 | 76 | ... | 0.0 | 1.0 | 0.0 |
| 1 | 13 | 1 | 17 | 73 | 121 | 24 | 178 | 8 | 26 | 60 | ... | 1.0 | 0.0 | 0.0 |
| 2 | 27 | 36 | 24 | 121 | 185 | 36 | 105 | 15 | 56 | 80 | ... | 1.0 | 0.0 | 1.0 |
| 3 | 8 | 7 | 16 | 100 | 151 | 29 | 121 | 12 | 49 | 89 | ... | 0.0 | 1.0 | 1.0 |
| 4 | 17 | 10 | 17 | 103 | 146 | 33 | 289 | 2 | 10 | 41 | ... | 0.0 | 1.0 | 0.0 |

5 rows × 71 columns

Figure 2.5.3.1: data after one-hot encoding

## 2.6 Model Evaluation

In this section we will divide the data in the data set into two groups. Training and Testing data.

I used a function to split the data into 80% training data and 20% test data, then trained the selected data and taught the model based on it and then fed in the selected data. Additionally, setting the random_state parameter to 42 ensures reproducibility, so that the same split is obtained every time the code is executed.

```
Shape of X_train_pca: (54617, 10)
Shape of X_test_pca: (13655, 10)
Shape of y_train: (54617,)
Shape of y_test: (13655,)
```

Figure 2.6.1: shape of testing and training data after reducing

From the above picture we can see :

Training data : 54617

Testing data : 13617

Train a Random Forest model on the preprocessed data.

In this section, we evaluated the accuracy performance of a Random Forest classifier by training it on the preprocessed dataset.

```
Accuracy of Random Forest model on preprocessed data: 0.9674112046869279
```

Figure 2.6.2: Accuracy of Random model on the preprocessed data

As can be seen from the picture above, the model's remarkable accuracy of 96.74% suggests that it had a very high rate of accurate categorization. The Random Forest algorithm's impressive performance shows that it was able to identify the underlying patterns and correlations in the data and use ensemble learning to improve prediction accuracy.

Compare the performance of the model trained on preprocessed data vs. raw data (before applying feature selection and scaling).

```
Random Forest accuracy on unprocessed data: 0.8987857142857143
```

Figure 2.6.3: Accuracy of Random model on the raw data

A filtering technique and the quantity of features to choose from are required in this section. To determine the optimal K (number of features) and accuracy, we will employ grid search in conjunction with the chi2 (chi-squared test).

```
Number of original features: 70
Number of features after chi-squared test filtering: 8
Accuracy with all features: 0.9807396558037349
Accuracy with selected features: 0.969315269132186
```

Figure 2.6.4: Results before and after applying the best parameters

Figure above shows that the accuracy is 96.93 %. The accuracy after removing the features is 96.93%. But because of the grid search, this is the greatest option.

In this section we will compare the final results between the random forest model trained on the preprocessed data and the raw data. From the image above we can see that the model trained on the preprocessed data outperforms the model trained on the raw data significantly on all criteria. From the above picture , The accuracy rate for the preprocessed data reached 0.9673 compared to 0.9051 on the raw data. And the F1 score (0.9639 versus 0.8994), which illustrates the importance of preprocessing techniques.

```
Evaluation Metrics for Model Trained on Preprocessed Data:
Precision: 0.9671
Recall:    0.9674
F1 Score:  0.9640
------------------------------------------------------------

Evaluation Metrics for Model Trained on Raw Data:
Precision: 0.9033
Recall:    0.8988
F1 Score:  0.8978

Comparison of Metrics:
Metric       Preprocessed Data    Raw Data
------------------------------------------
Precision    0.9671               0.9033
Recall       0.9674               0.8988
F1 Score     0.9640               0.8978
```

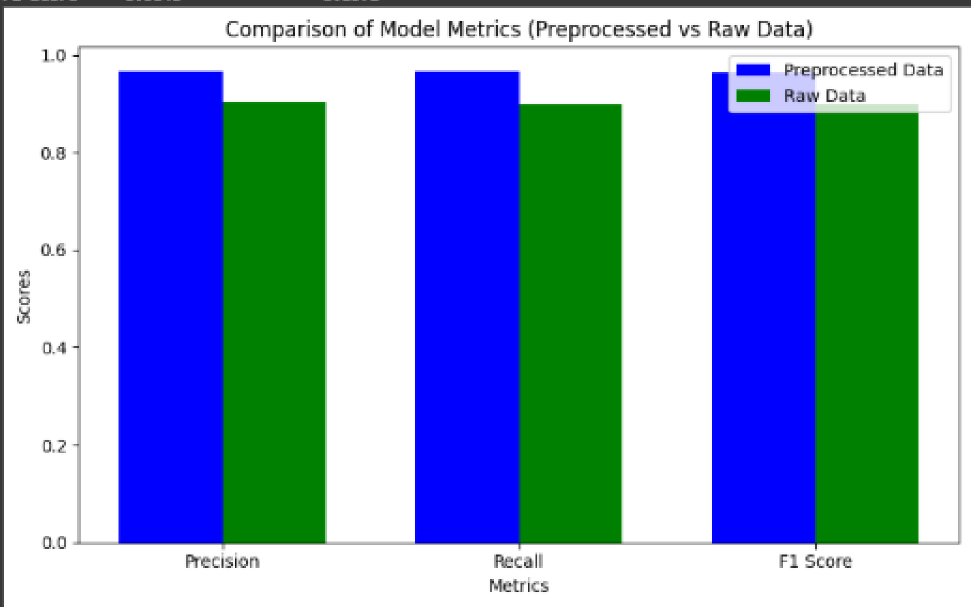

Figure 2.6.5: visualization to compare the Precision and recall and f1 between preprocess data vs raw data

# 3. Comparative Analysis of Classification Techniques

```
<ipython-input-42-ad5926396f97>:5: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping
  df = df.groupby('Target', group_keys=False).apply(
After stratified sampling, dataset shape: (34996, 34)
Training set shape: (27996, 38)
Testing set shape: (7000, 38)
```

Figure 3.1: reduce the data set to half

From the image above we can see that I reduced the data by half due to the time issue. The data size became 35 k. Then I divided the data into two sets, training and test. 80% for training and 20% for testing.

## 3.1 Hyperparamter tuning

The images below show the parameters that we will use to adjust the hyperparameters for each model. The figure 28 shows the RF parameters, the figure 29 shows the SVM parameters, the figure 30 shows the MLP parameters.

Random Forest (RF):

```
param_grids = {
    'Random Forest': {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5]
    },
```

Figure 3.1.1: parameter for RF

SVM:

```
    },
    'SVM': {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    },
```

Figure 3.1.2: parameter for SVM

Multilayer Perceptron (MLP):

```
    'MLP': {
        'hidden_layer_sizes': [(100,), (100, 50)],
        'activation': ['relu', 'tanh'],
        'solver': ['adam', 'sgd'],
        'alpha': [0.0001, 0.001]
    }
```

Figure 3.1.3: parameter for MLP

```
Tuning Random Forest...
Best parameters for Random Forest: {'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 200}

Tuning SVM...
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker stopped while
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearnex/svm/_common.py:239: RuntimeWarning: random_state does not influence oneD
  warnings.warn(
Best parameters for SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}

Tuning MLP...
Best parameters for MLP: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'}
```

Figure 3.1.4: best parameter for three models

The image above shown the best hyperparameter for three models :

The best parameters for RF are max_depth: 20, min_samples_split: 5, and n_estimators: 200.

The best parameters for SVM are: C: 10, gamma: 'scale', and kernel: 'linear'.

The best parameters for MLP are: activation :'relu', alpha: 0.0001, hidden_layer_sizes: (100,), and solver: 'adam'.

## 3.2 Evaluation Metrics:

### 3.2.1 Random Forest (RF):

In below pictures we can see the performance metrics of random forest (RF) before grid search and after

```
Training Random Forest...

Classification Report for Random Forest:
              precision    recall  f1-score   support

           0     0.9955    0.8290    0.9047       538
           1     0.8982    0.8364    0.8662       538
           2     0.9252    0.9201    0.9226       538
           3     0.8977    0.8627    0.8798       539
           4     1.0000    1.0000    1.0000       539
           5     0.8893    1.0000    0.9414       538
           6     0.8098    0.7032    0.7527       539
           7     0.8534    0.7236    0.7831       539
           8     0.8677    1.0000    0.9292       538
           9     0.8939    0.7045    0.7879       538
          10     0.6671    1.0000    0.8003       539
          11     0.9956    0.8383    0.9102       538
          12     0.8606    0.9963    0.9235       539

    accuracy                         0.8780      7000
   macro avg     0.8888    0.8780    0.8771      7000
weighted avg     0.8887    0.8780    0.8770      7000

Accuracy: 0.8780
F1 Score: 0.8770
```

Figure 3.2.1.1: performance for RF before Grid search

```
--- Retraining Random Forest with Best Parameters ---

Classification Report for Random Forest (Tuned):
              precision    recall  f1-score   support

           0     0.9915    0.8717    0.9278       538
           1     0.8821    0.9182    0.8998       538
           2     0.9488    0.9294    0.9390       538
           3     0.9470    0.8627    0.9029       539
           4     1.0000    1.0000    1.0000       539
           5     0.9539    1.0000    0.9764       538
           6     0.8134    0.7199    0.7638       539
           7     0.8352    0.8275    0.8313       539
           8     0.8732    0.9981    0.9315       538
           9     0.8659    0.7323    0.7936       538
          10     0.7834    1.0000    0.8786       539
          11     0.9956    0.8383    0.9102       538
          12     0.8606    0.9963    0.9235       539

    accuracy                         0.8996      7000
   macro avg     0.9039    0.8996    0.8983      7000
weighted avg     0.9039    0.8996    0.8983      7000

Accuracy: 0.8996
F1 Score: 0.8983
```

Figure 3.2.1.2: performance for RF after Grid search

The figure 32 shows the results for the RF model before tuning the parameters, where it achieved an accuracy rate of 87.80% and an F1 score of 87.70%. In the figure 33 .we notice that the model has improved performance with an accuracy of 89.96% and an F1 score of 89.83%. The tuned model also shows better precision, recall and F1 scores for most classes. This shows the positive effect of using the best parameters for the Rf model on all metrics.

```
Training Time (s): 2.2051
Prediction Time (s): 0.0729
Memory Usage (MB): 1.6
```

Figure 3.2.1.3: training and prediction time and memory for RF before grid search

```
Training Time (s): 4.6483
Prediction Time (s): 0.1773
Memory Usage (MB): 1.6
```

Figure 3.2.1.4: training and prediction time and memory for RF after grid search

We can notice from Figures 34 and 35 that after using grid search it requires more time in the training time and prediction time and no change for memory usage while fitting the data into the model.

### 3.2.2 MLP Evaluation

In below pictures we can see the performance metrics of MLP before grid search and after

```
Training MLP...

Classification Report for MLP:
              precision    recall  f1-score   support

           0     0.8256    0.7918    0.8083       538
           1     0.7524    0.7230    0.7374       538
           2     0.7747    0.8755    0.8220       538
           3     0.8052    0.7514    0.7774       539
           4     0.9926    0.9963    0.9944       539
           5     0.8970    0.8420    0.8686       538
           6     0.6204    0.5974    0.6087       539
           7     0.6498    0.6679    0.6587       539
           8     0.8233    0.8922    0.8564       538
           9     0.7201    0.7175    0.7188       538
          10     0.6906    0.6957    0.6932       539
          11     0.8321    0.8290    0.8305       538
          12     0.8367    0.8367    0.8367       539

    accuracy                         0.7859      7000
   macro avg     0.7862    0.7859    0.7855      7000
weighted avg     0.7862    0.7859    0.7855      7000

Accuracy: 0.7859
F1 Score: 0.7855
```

Figure 3.2.2.1: performance for MLP before Grid search

```
--- Retraining MLP with Best Parameters ---

Classification Report for MLP (Tuned):
              precision    recall  f1-score   support

           0     0.8256    0.7918    0.8083       538
           1     0.7524    0.7230    0.7374       538
           2     0.7747    0.8755    0.8220       538
           3     0.8052    0.7514    0.7774       539
           4     0.9926    0.9963    0.9944       539
           5     0.8970    0.8420    0.8686       538
           6     0.6204    0.5974    0.6087       539
           7     0.6498    0.6679    0.6587       539
           8     0.8233    0.8922    0.8564       538
           9     0.7201    0.7175    0.7188       538
          10     0.6906    0.6957    0.6932       539
          11     0.8321    0.8290    0.8305       538
          12     0.8367    0.8367    0.8367       539

    accuracy                         0.7859      7000
   macro avg     0.7862    0.7859    0.7855      7000
weighted avg     0.7862    0.7859    0.7855      7000

Accuracy: 0.7859
F1 Score: 0.7855
```

Figure 3.2.2.2: performance for MLP after Grid search

The figure 36 shows the results for the MLP model before tuning the parameters, where it achieved an accuracy rate of 78.59% and an F1 score of 78.55%. In the figure 37 .we notice that the model has no change accuracy of 78.59% and an F1 score of 78.55%.

```
Training Time (s): 158.1591
Prediction Time (s): 0.0251
Memory Usage (MB): 4.07
```

Figure 3.2.2.3: training and prediction time and memory for MLP before grid search

```
Training Time (s): 153.7742
Prediction Time (s): 0.0134
Memory Usage (MB): 4.07
```

Figure 3.2.2.4: training and prediction time and memory for MLP after grid search

We can notice from Figures 38 and 39 that after using grid search that there is an improvement in training time and prediction time while there is no difference in memory usage.

## 3.2.3 SVM Evaluation

```
  warnings.warn(

Classification Report for SVM:
              precision    recall  f1-score   support

           0     0.8186    0.7379    0.7761       538
           1     0.6733    0.6933    0.6832       538
           2     0.7888    0.7844    0.7866       538
           3     0.7458    0.7403    0.7430       539
           4     0.9908    0.9981    0.9945       539
           5     0.7397    0.7342    0.7369       538
           6     0.5983    0.5250    0.5593       539
           7     0.5620    0.5380    0.5498       539
           8     0.7953    0.8810    0.8360       538
           9     0.6908    0.7100    0.7003       538
          10     0.5589    0.6252    0.5902       539
          11     0.8346    0.8253    0.8299       538
          12     0.8355    0.8386    0.8370       539

    accuracy                         0.7409      7000
   macro avg     0.7409    0.7409    0.7402      7000
weighted avg     0.7409    0.7409    0.7402      7000

Accuracy: 0.7409
F1 Score: 0.7402
```

Figure 3.2.3.1: performance for SVM before Grid search

```
warnings.warn(

Classification Report for SVM (Tuned):
              precision    recall  f1-score   support

           0     0.8093    0.7732    0.7909       538
           1     0.6982    0.7268    0.7122       538
           2     0.8197    0.8030    0.8113       538
           3     0.7677    0.7662    0.7669       539
           4     0.9981    0.9981    0.9981       539
           5     0.7786    0.7714    0.7750       538
           6     0.6030    0.5213    0.5592       539
           7     0.5528    0.5436    0.5482       539
           8     0.8357    0.8699    0.8525       538
           9     0.6753    0.7305    0.7018       538
          10     0.5697    0.6067    0.5876       539
          11     0.8698    0.8569    0.8633       538
          12     0.8611    0.8738    0.8674       539

    accuracy                         0.7570      7000
   macro avg     0.7569    0.7570    0.7565      7000
weighted avg     0.7568    0.7570    0.7565      7000

Accuracy: 0.7570
F1 Score: 0.7565
```

Figure 3.2.3.2: performance for SVM after Grid search

The figure 40 shows the results for the RF model before tuning the parameters, where it achieved an accuracy rate of 74.09% and an F1 score of 74.02%. In the figure 41 .we notice that the model has improved performance with an accuracy of 75.70% and an F1 score of 75.65%. The tuned model also shows better precision, recall and F1 scores for most classes. This shows the positive effect of using the best parameters for the SVM model on all metrics.

```
Training Time (s): 109.11
Prediction Time (s): 5.0219
Memory Usage (MB): 32.17
```

Figure 3.2.3.3: training and prediction time and memory for SVM before grid search

```
Training Time (s): 117.7412
Prediction Time (s): 1.221
Memory Usage (MB): 32.03
```

Figure 3.2.3.4: training and prediction time and memory for SVM after grid search

We can notice from Figures 42 and 43 that after using grid search it requires more time in the training time while there is a clear improvement in the prediction time and memory usage while fitting the data into the model.

## 3.3 compression between three models

```
=== Model Performance Comparison ===
                       Accuracy  F1 Score  Training Time (s)  \
Random Forest          0.878000  0.877038             2.2051
SVM                    0.740857  0.740186           109.1100
MLP                    0.785857  0.785462           158.1591
Random Forest (Tuned)  0.899571  0.898315             4.6483
SVM (Tuned)            0.757000  0.756457           117.7412
MLP (Tuned)            0.785857  0.785462           153.7742

                       Prediction Time (s)  Memory Usage (MB)
Random Forest                       0.0729               1.60
SVM                                 5.0219              32.17
MLP                                 0.0251               4.07
Random Forest (Tuned)               0.1773               1.60
SVM (Tuned)                         1.2210              32.03
MLP (Tuned)                         0.0134               4.07
<ipython-input-48-c0756ee959c4>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
```

Figure 3.3.1: compression between three models

**Accuracy and f1 score:** based on the above figure we can see the random forest tuned (RF) outperforms all others with accuracy 89.96% and F1 89.83% showing the best balance between precision and recall.

**Training Time**: the model random forest have the smallest time in training taking 4.64 seconds. While the SVM taking 109.11 seconds.

**Prediction Time**: we can see the MLP model have the faster prediction time, the prediction time is 0.0251 seconds. While SVM model have 5.02 seconds to prediction.

**Memory Usage**: the random forest is the efficient memory usage at 1.60MB, while the SVM has the 32.17MB and MLP has 4.07MB.

In conclusion, Random Forest (Tuned) is the greatest option for balancing speed, accuracy, and resource efficiency, whereas SVM is excellent at accuracy but comes at a far larger cost in terms of training and prediction timeframes.

## 3.4 Performance visualization

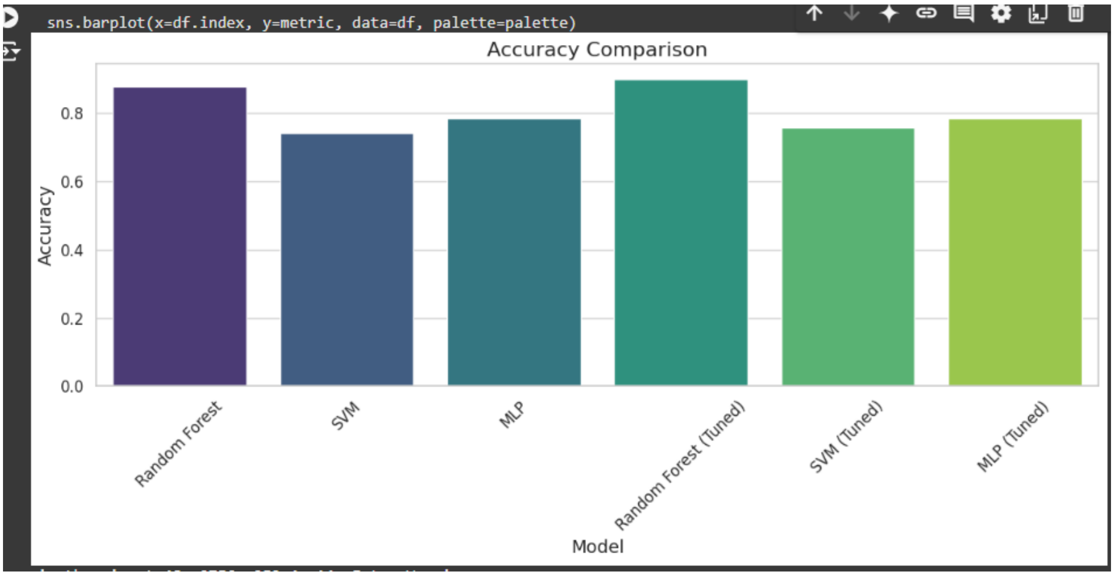


Figure 3.4.1: accuracy visualization

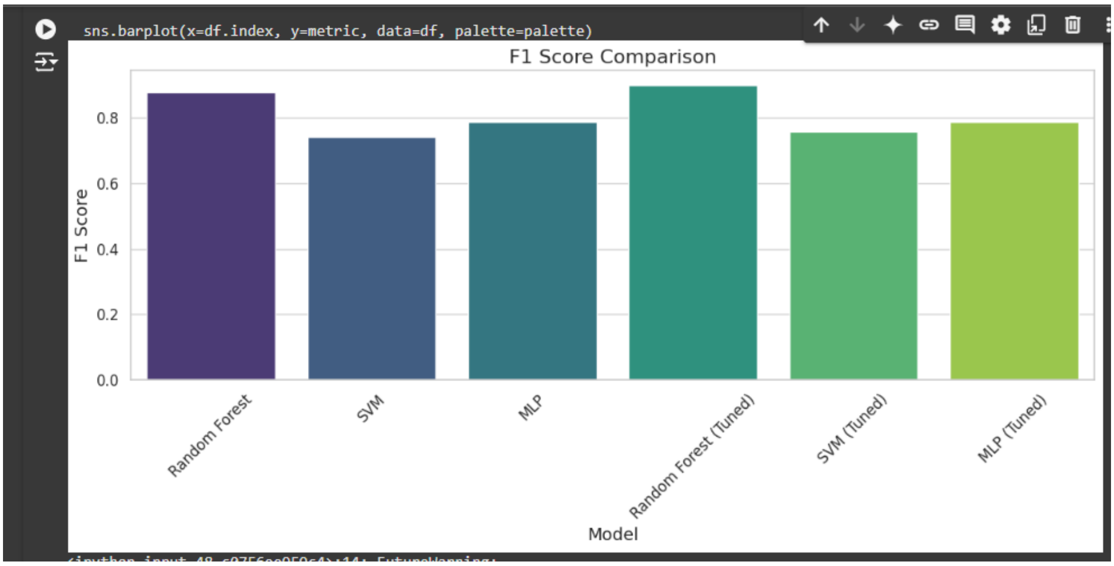


Figure 3.4.2: F1 score visualization

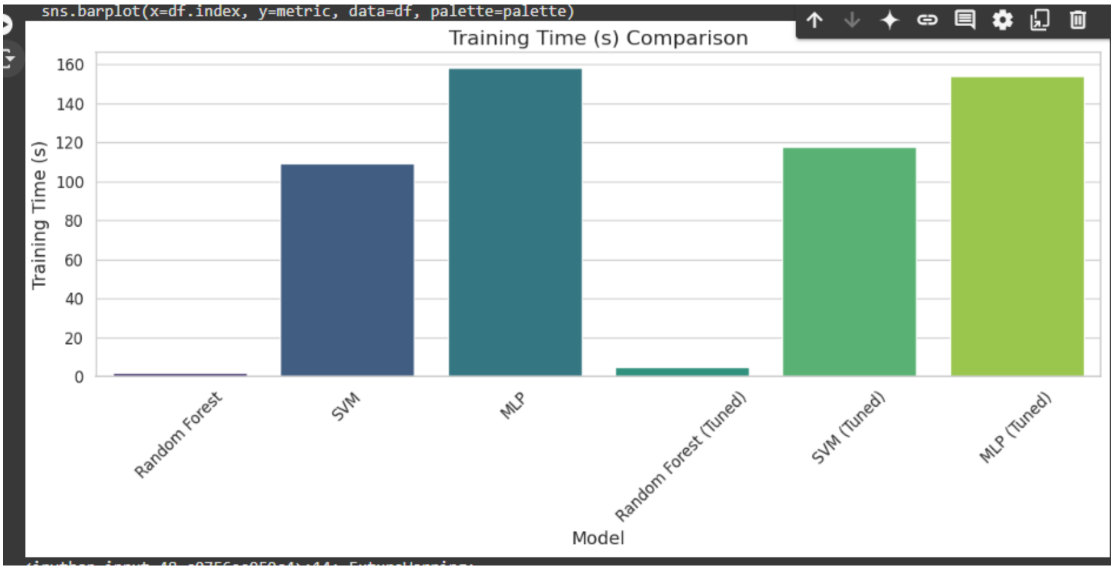


Figure 3.4.3: training time visualization

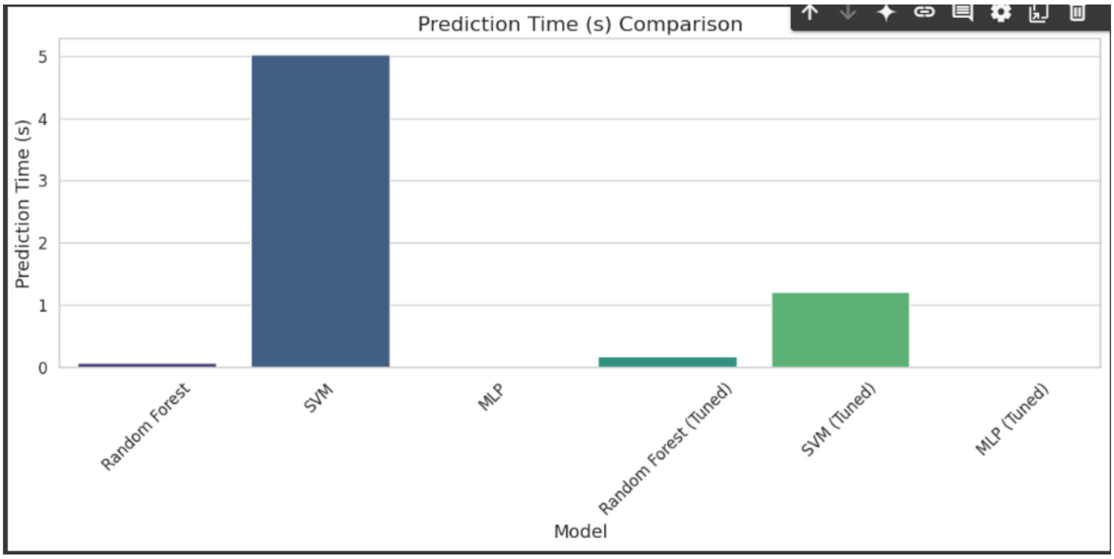


Figure 3.4.4: prediction time visualization

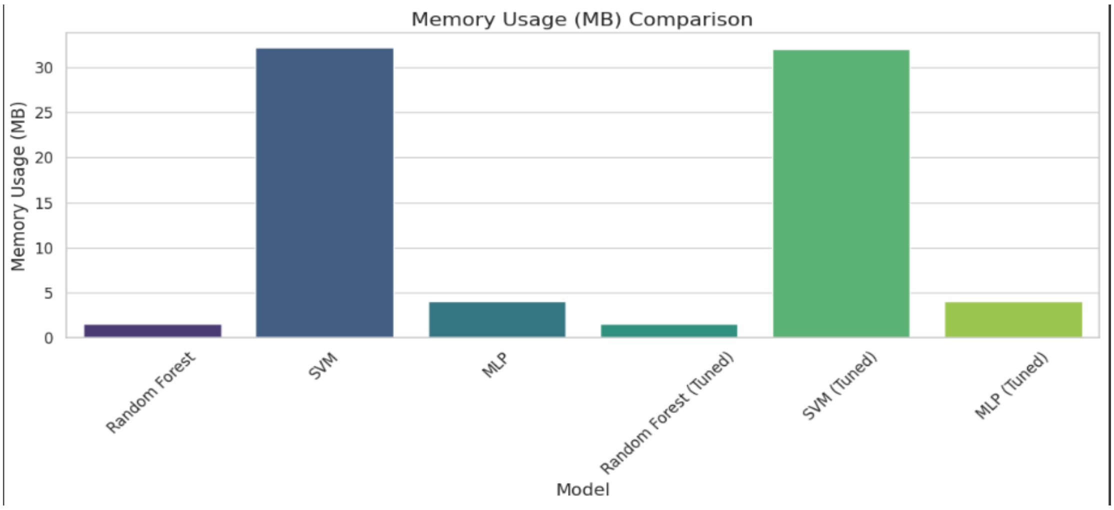


Figure 3.4.5: memory usage visualization

## 4. Conclusion

This assignment emphasizes how vital careful data preparation is to improving the diabetes dataset's quality and suitability for predictive modeling. By methodically addressing lacking values, meticulously choosing with the use of pertinent features, categorical data encoding, and suitable partitioning, we make sure the dataset is ready for machine learning research.
While descriptive statistics provide a first overview necessary for spotting patterns, abnormalities, and potential problems that could impair model efficacy, the use of various visualization tools aids in intuitively comprehending the dataset, facilitating pattern recognition and comprehension. Managing outlier's demands careful consideration to maintain data integrity, and dealing with missing data necessitates making wise decisions depending on the quantity and characteristics of the missing values. Moreover, hiring feature Selection methods contribute to enhancing model performance by reducing dimensionality and eliminating relevant features

The Random Forest (Tuned) model showed the best balance of accuracy, training time, and memory usage, making it ideal for real-world applications. This emphasizes the importance of preprocessing for improving both performance and efficiency, especially when handling diverse data types and ensuring scalability.