U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Diabetes Diagnosis Classification

(Data Mining Project)

Master's degree in Data Science & Engineering – FEUP

Introduction to Machine Learning and Data Mining

December 2022

Authors: Ian Karkles and Sónia Ferreira

1. Table of Contents

List of Tables

List of Figures

List of Appendix Figures

2. Introduction

"A correct diagnosis is a three-fourths the remedy" – Mahatma Gandi

Diabetes is considered one of the deadliest and most chronic diseases, that occurs either when the pancreas does not produce enough insulin or when the body cannot effectively use the insulin it produces, which in turn makes the metabolism of carbohydrate metabolism abnormal and raises the levels of glucose in the blood. In diabetes, a person generally suffers from high blood sugar. Intensify thirst, hunger, and frequent urination are some of the symptoms of the disease (Vijayan, V.V., Anjali, C.,).

Diabetes diagnosis is not only affected by several factors like height, weight, hereditary factors, and insulin, but the major reason considered is sugar concentration among all factors. The measure of sugar substances cannot be controlled, and this is what makes the process of identifying the disease morose and tedious.

In 2014, 8,5% of adults over 18 years old had diabetes. In 2019, diabetes was the direct cause of 1.5 million deaths and 48% of all deaths due to diabetes occurred before the age of 70 years. In addition, 460 000 kidney disease deaths were caused by diabetes, and raised blood causes around 20% of cardiovascular deaths. Between 2000 and 2019, there was a 3% increase in age-standardized mortality rates from diabetes (WHO).

Many complications, like blindness, kidney failure, heart attacks, strokes, and lower limb amputation can occur if diabetes remains untreated or unidentified. A healthy diet combined with regular exercise, plus maintaining a normal body weight are some of the ways to prevent or delay to have diabetes. Consequently, early identification is the only remedy to stay away from complications. However, without the proper diagnosis, there isn't much that can be done, and the occurrence of diabetes has been rising more rapidly in low-and middle-income countries than in high-income countries (WHO).

The process to identify the disease is wearisome and costly, as it requires the patient to visit a doctor or diagnostic center and run several exams. And it's here that the growth of Machine Learning approaches can solve this critical problem since it would not require a patient's visit to a doctor or diagnosis center or to run several exams. Plus, a Machine Learning process could be applied worldwide at a cheaper cost than the traditional diagnosis of the disease.

Today, many researchers are conducting experiments into diagnosing diseases using various classification algorithms of machine learning approaches, since this type of algorithm was proven to

work better for these cases. Classification strategies are broadly used for classifying data into different classes according to some constraints compared to an individual classifier.

Data mining and machine learning algorithms gain their strength due to their capabilities to manage large amounts of data, combine data from different sources, and integrate background information into the study.

Therefore, this research is performed on Pima Indians Diabetes Database (PIDD) which was sourced from "Kaggle: Your Machine Learning and Data Science Community". This database focuses on pregnant women, older than 21 years, and suffering from diabetes. Several machine learning classification algorithms, (e.g.: Decision Tree, KNN, and AdaBoost) are used to find the prediction of diabetes in a patient. The performance of all algorithms is then evaluated on various measures like precision, accuracy, f-measure, and recall. And these results are then verified using the Receiver Operating Characteristic (ROC) curves properly and systematically.

The remaining of this report is organized as follows: Section III briefs the materials and method used, Section IV introduces the data source used and its origin, Section V gives a summary of what CRISP-DM is and how it's implemented in this research (its relation between report and technical solution), Section VI focus on understanding the research and project objectives and requirements from a business point of view, Section VII explores the raw data of the dataset, Section VIII details all activities to construct the final dataset from the raw data, Section IX will approach and discuss the evaluations done on the modeling techniques, Section X discusses and evaluates the results, Section XI determines the Conclusion of the research work and Section XII approach's Future Work that can be done to further continue or improve this research.

3. Materials and Methods

For this data mining project, the methodology used to approach its execution was CRISP-DM (phases one to five will be described in detail at the core of this report).

In this data mining project, Python programming language was used through Jupyter Notebook.

4. Data Source

The present data mining project will use a dataset to diagnose diabetes, that was collected from the website "Kaggle: Your Machine Learning and Data Science Community".

The dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases and its objective are to predict whether a patient has diabetes, based on certain diagnostic measurements that are included in the dataset. This dataset has the following constraints in place:

- All patients are female.
- They are at least 21 years old.
- They are of Pima Indian heritage.2.

5. What is the CRISP-DM methodology?

CRISP-DM which stands for **CR**oss **I**ndustry **S**tandard **P**rocess for **D**ata **M**ining is an industry-proven modeling process that serves as a basis for data sciences processes that can be applied in any type of business without the dependency on software or service to be executed.

Published in 1999 to standardize data mining processes across industries, it has since become the most common methodology for data mining, analytics, and data science projects.

CRISP-DM consists of six sequential phases, to conceive a Data Mining project and these phases can have cycle iterations according to the developer's needs. The six sequential phases are:

1. Business understanding – What does the business need?
2. Data understanding – What data do we have/need? Is it clean?
3. Data preparation – How do we organize the data for modeling?
4. Modeling – What modeling techniques should we apply?
5. Evaluation – Which model best meets the business objectives?
6. Deployment – How do stakeholders access the results?

5.1. What are the six CRISP-DM phases?

The first phase, **Business Understanding** focuses on understanding the project objectives and requirements from a business perspective. The analyst formulates this knowledge as a data mining problem and develops a preliminary plan.

The second phase is **Data Understanding,** and its objective is to know what can be expected and achieved from the data. It checks the quality of the data, in several terms, such as data completeness, values distributions, and data governance compliance. In this phase, the analyst might also detect interesting subsets to form hypotheses for hidden information.

This phase is a crucial part of the project because it defines how viable and trustworthy can be the result. It can be necessary to step back, to understand the business point of view and how that piece of information can be beneficial.

The third phase, **Data Preparation** and involves the ELTs or ETLs process that will cover all activities to construct the final dataset, based on the initial raw data (select, clean, construct, integrate, and format/re-format data).

Sometimes data governance policies are not respected or set in on the organizations, human errors happen, and to give meaning to data, it becomes necessary to standardize the information. Likewise,

some algorithms perform better under certain parameters, some do not accept numerical or non-numerical values or large values, then again is necessary to normalize this data.

Data Preparation is, therefore, one of the most phases of a data mining project and the step that consumes more time.

The fourth phase, **Modelling** is the core phase of any machine learning project. This phase will be responsible for the results that should satisfy or help satisfy the project goals.

This part of the project should however be the shortest phase of the project, as if everything previously is done correctly, there is little to adjust. In the cases that the results are improvable, the methodology is set to step back to data preparation and improve the available data.

The fifth phase, **Evaluation**, looks more broadly at which model best meets the business and what to do next. Here the analyst will evaluate if the results are valid and correct. If determined that the results are wrong, the methodology allows the review back to the first step, to understand why the results are incorrect.

The sixth phase (and last phase) is **Deployment** and depending on the requirements, the deployment phase can be as simple as creating a report or as complex as implementing a repeatable data mining process across the enterprise (CRISP-DM).

5.2. CRISP-DM phases & Diabetes Diagnosis Classification Mining Project phases

The below table offers a comparison between the CRISP-DM phases and the technical steps implemented in this data mining project.

Table 1 - CRISP-DM phases & Diabetes Diagnosis Classification Mining Project phases

CRISP-DM Phases	Technical Project Phases
1. Business Understanding	1. Understand and set goals from a business perspective for the project (Jupiter notebook chapter: Business Context / Data Context)
2. Data Understanding	2. Check if the available data can meet the objectives of the project and its quality (Jupiter notebook chapter: Exploratory Data Analysis - EDA)

3. Data Preparation	3. The raw data is transformed, in the cases that are justifiable for this data mining project (Jupiter notebook chapter: Pre-processing, Feature Engineering, Feature Selection)
4. Modeling	4. Execute the algorithms that satisfied the project objectives (Jupiter notebook chapter: Models)
5. Evaluation	5. The results are presented, analyzed and evaluated (Jupiter notebook chapter: Evaluation)
6. Deployment[1]	6. Not Applicable[1]

[1] Phase six was not required for the conclusion of this project/experiment.

6. Business Understanding (CRISP-DM - phase 1)

Any good project starts with a deep understanding of the customer's needs. Data mining projects are no exception and CRISP-DM recognizes this. Thus, the first phase of this project focused on understanding the current context of the topic in the study: Diabetes disease, and understanding the project objectives and requirements from a business perspective.

Diabetes is a worldwide killer and ranks among the top causes of premature deaths. More than 500 million adults (20 – 79 years) are living with diabetes (nearly one in ten) and it's predicted that these numbers will keep ascending in the coming years. By 2030 is estimated that more than 600 million will suffer from diabetes and by 2045 more than 783 million (IDF).

Between 2000 and 2019, diabetes mortality rates by age increased by 3% (WHO). And just in 2021, diabetes was responsible for more than 6 million deaths (one every five seconds) (IDF).

A very concerning point is that three out of four people leaving with this disease, live in low- and middle-income countries that do not have the resources or money to map and treat rightfully this disease, which may affect the daily life of a person who is not aware of this illness. If diabetes is diagnosed correctly and timely, it can be treated, and its consequences are avoided or delayed with diet, physical activity, medication, and regular screening and treatment for complications (WHO).

Diabetes caused at least 966 billion dollars (approximately more than 900 million euros) in health expenditure – a 316% increase over the last 15 years (IDF).

There are several reasons, for the lack of the right diagnosis, including that the process is expensive and tedious as it includes several visits of the patient to a doctor or a diagnostic center, plus doing some exams. But one of the main reasons is the lack of technical skills to interpret the results of exams, especially in low and middle-income countries.

Nevertheless, if a machine learning approach could solve or improve this critical problem of diagnosing when a patient has or is prone to have diabetes, this would represent a win globally in the quality of health service and diagnosis, especially for low- and middle-income countries where the problem is more persistent, and they have less money and resources to invest. Another win, less related to the patient of diabetes would be that the resources (especially money) that are today spent to diagnose, prevent, and treat diabetes disease would be needed less and therefore could be allocated to other problems.

In this way, the main goal of this experiment with a classifier is to guide especially low and medium-income health professionals to classify a Diabetes case with simple exams in a person who most probably would be unaware of the disease until would be too late.

In the context of Data Classification, multiple classification methods can help to solve this type of problem and will be used in this experiment. For this type of project, the dataset was collected from the website "Kaggle: Your Machine Learning and Data Science Community" and the experiments are performed on the Pima Indians Diabetes Database, originally from the National Institute of Diabetes and Digestive and Kidney Diseases. This dataset has the following constraints in place:

- All patients are female.
- They are at least 21 years old.
- They are of Pima Indian heritage.2.

The dataset contains multiple features, described in the table below, that can help the classifier model to rightfully predict diabetes in a patient.

Table 2 - Dataset features

Columns (features)	Description
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
Blood Pressure	Diastolic blood pressure (mm/Hg)
Skin Thickness	The Triceps skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	Body Mass Index (weight in Kg/(height in m)^2)
Diabetes Pedigree Function	Diabetes pedigree function
Age	Age of the Patient (in years)
Outcome	Class variable

7. Data Understanding (CRISP-DM - phase 2)

During phase 1 (Business Understanding), several questions related to the data that would be available in the dataset arose, therefore in this section (phase 2 – Data Understanding), the data is going to be analyzed and visualized for easy understanding and followed with appropriate statistical analysis for empirical findings within the data. And the following Business questions will be answered within this phase:

- From the total of patients in this study how many are diabetic and non-diabetic?
- Is there a difference in the average age of patients with diabetes and without?
- Do diabetic and non-diabetic patients have similar levels of glucose, insulin, blood pressure, and BMI?
- Are the skin thickness and the diabetes pedigree function similar between patients with and without diabetes?

Pima Indians Diabetes Dataset (PIDD), comprises 768 observations and 9 characteristics (features)[2].

Out of which one is a dependent variable, and the remaining are independent:

- Dependent variable: outcome.
- Independent variable: all remaining features.

500 of the patients are non-diabetic and 268 have diabetes[3].

Apart from BMI and Diabetes Pedigree Function which are of type float, all remaining features are of type int64[4].

The below figure shows a small extract of the raw dataset, where it can be observed the type of data and structure that is available in the dataset.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

Figure 1 - Raw Dataset (extract)

[2] Appendix - Figure 1 - Dataframe Shape
[3] **Error! Reference source not found.**
[4] Appendix - Figure 2 - Dataframe columns and corresponding data

The following conclusions were taken after a toughly analysis of the dataset:

- No missing values[5] are present, as can be observed in figure 2.
- No redundant data exist[6].
- No negative values exist in the dataset.
- As it can be observed in figure 3, some parameters (features) have zero values, and they can be considered as NaN values classified as zero, for example in Skin Thickness (that represents the triceps skin fold thickness in mm) and Glucose.

  Each feature was checked individually, to understand better the values present in each and see if there was some justifiable reason for the zeros[7].
- Outliers were detected in the dataset with the use of Z-Score and Winsorization. Both pointed out some potential outliers and these results can be observed in figure 5 and figure 6, respectively, for Z-Score and Winsorization. Z-Score test was conducted with 3 standard deviations from the mean.

  However even though these values can be considered outliers, after further research, it was concluded that in rare or extreme medical cases these values could be achieved (Mayo Clinic).
- When looking at the dataset statistics[8], was observed that for some features (e.g. glucose, blood pressure, BMI, etc.) the mean and standard deviation (std) values are not close, which indicates that the values in the data set are most probably not very consistent and can indicate, the possible presence of outliers and already checked with Z-Score and Winsorization.

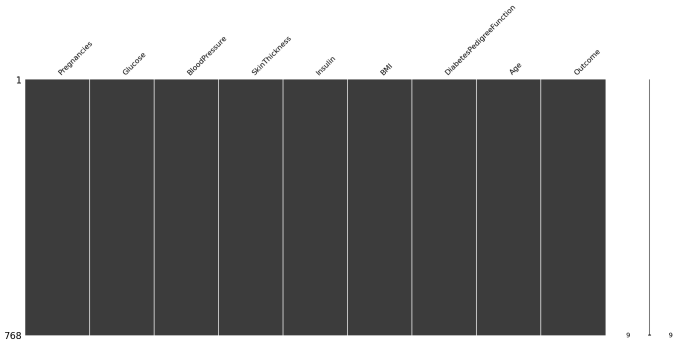


Figure 2 - Dataframe Missing Values[9]

[5] Appendix - Figure 3 - Missing Values
[6] Appendix - Figure 5 - Redundant Data
[7] Appendix - Figure 8 – Glucose 0 values; Appendix - Figure 9 – Blood Pressure 0 values; Appendix - Figure 10 – Skin Thickness 0 values; Appendix - Figure 11 - Insulin 0 values; Appendix - Figure 12 - BMI 0 values; Appendix - Figure 13 – Diabetes Pedrigree Function values; Appendix - Figure 14 - Age values
[8] Appendix - Figure 15 - Statistics of the dataset
[9] Appendix - Figure 4 – Missing Values Matrix

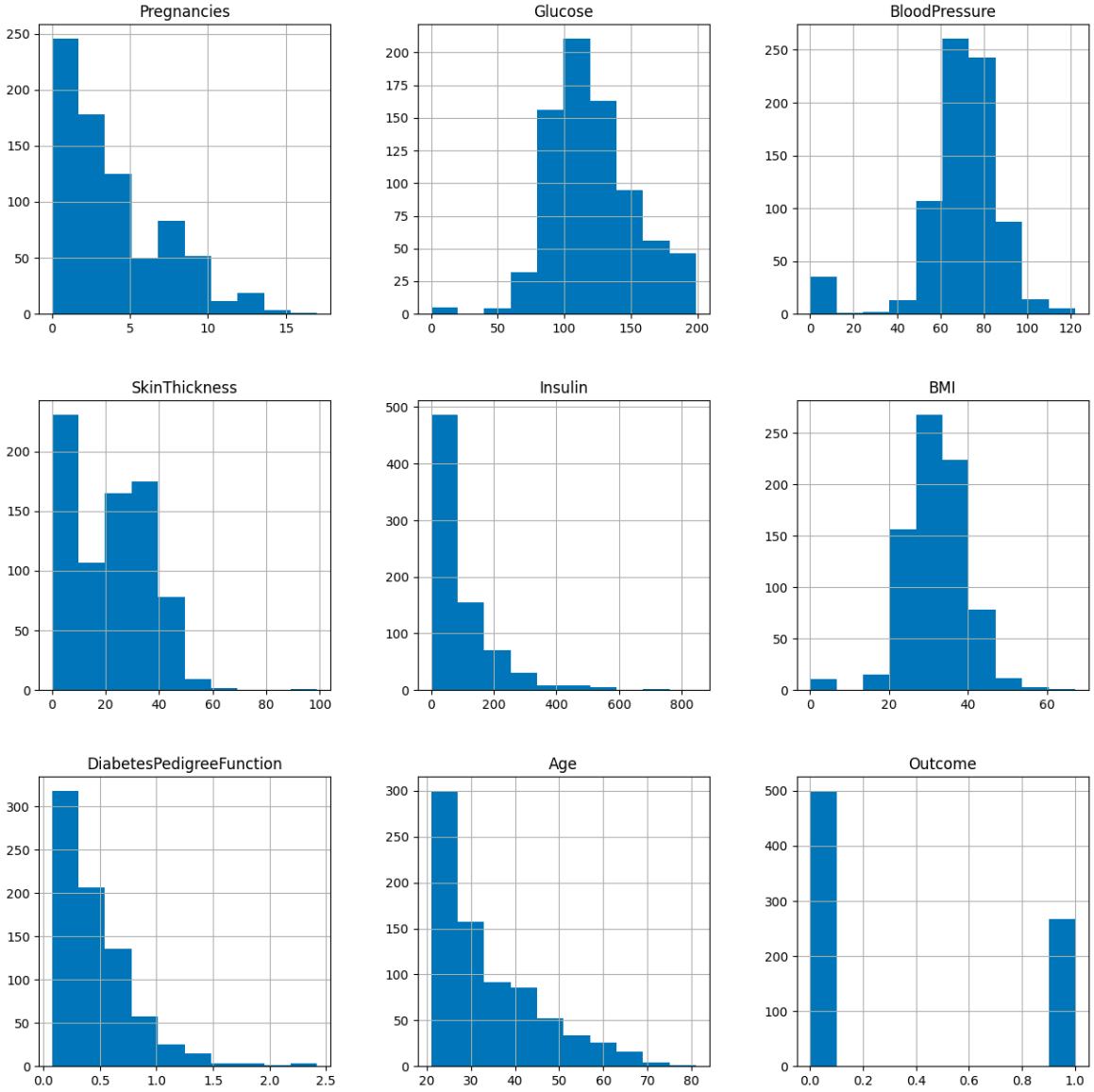


Figure 3 - Plots of individual variables of the raw dataset

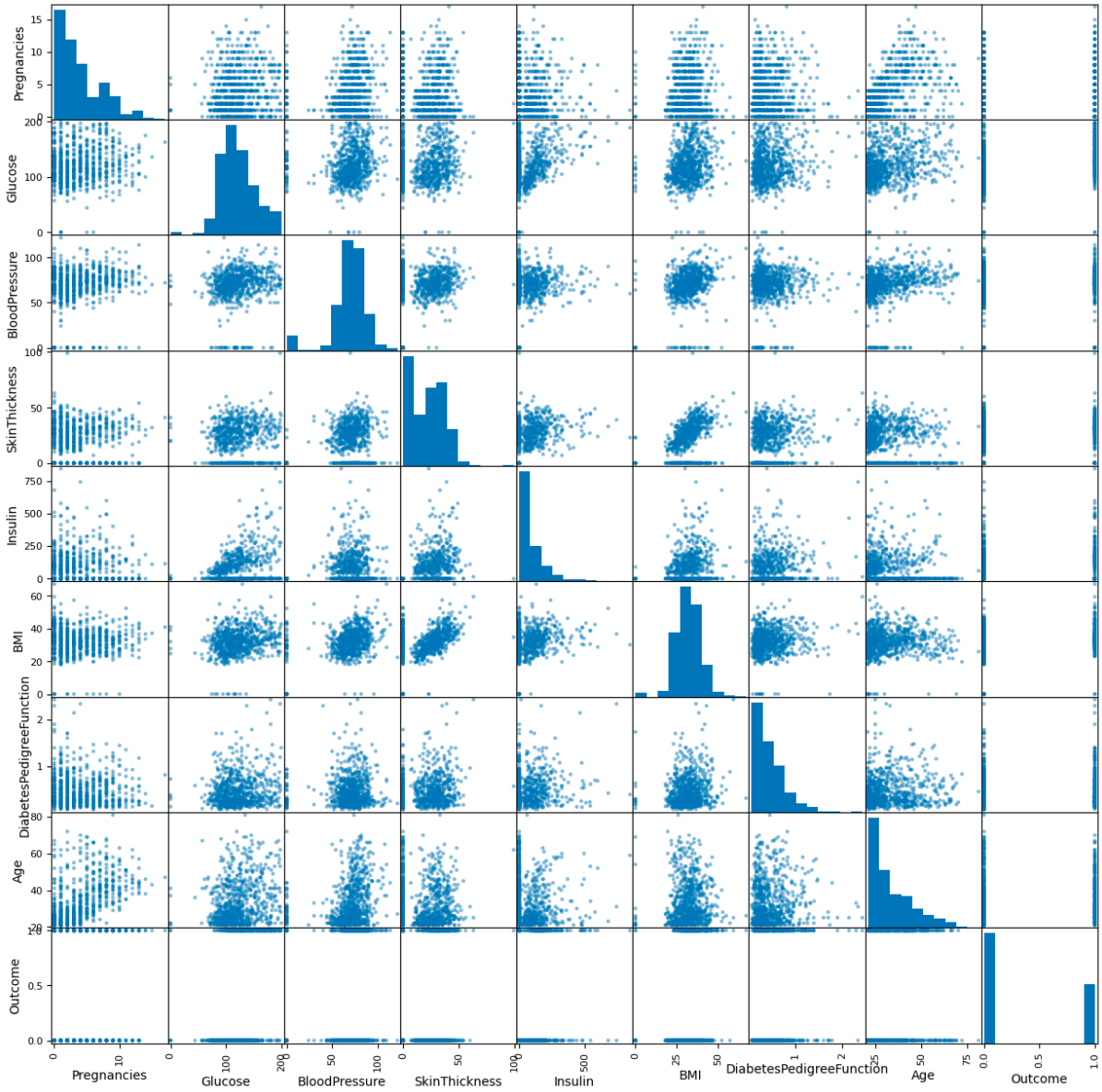


Figure 4 - Plots of scatter plots of individual variables of the raw dataset

```
From the Column: Pregnancies
Outliers: [15, 17, 14, 14]
From the Column: Glucose
Outliers: [0, 0, 0, 0, 0]
From the Column: BloodPressure
Outliers: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
From the Column: SkinThickness
Outliers: [99]
From the Column: Insulin
Outliers: [543, 846, 495, 485, 495, 478, 744, 680, 545, 465, 579, 474, 480, 600, 440, 540, 480, 510]
From the Column: BMI
Outliers: [0.0, 0.0, 0.0, 0.0, 0.0, 67.1, 0.0, 0.0, 59.4, 0.0, 0.0, 57.3, 0.0, 0.0]
From the Column: DiabetesPedigreeFunction
Outliers: [2.288, 1.893, 1.781, 2.329, 1.476, 2.137, 1.731, 1.6, 2.42, 1.699, 1.698]
From the Column: Age
Outliers: [69, 72, 81, 70, 69]
From the Column: Outcome
Outliers: []
```

Figure 5 - Outliers: Z-Score result

```
From the Column: Pregnancies
Outliers: [15, 17, 14, 14]
From the Column: Glucose
Outliers: [197, 44, 0, 0, 197, 0, 0, 197, 0, 198, 197, 199, 56]
From the Column: BloodPressure
Outliers: [110, 108, 122, 110, 108, 110, 114]
From the Column: SkinThickness
Outliers: [60, 54, 56, 54, 52, 63, 52, 99]
From the Column: Insulin
Outliers: [543, 846, 744, 680, 545, 579, 600, 540]
From the Column: BMI
Outliers: [53.2, 55.0, 67.1, 52.3, 52.3, 52.9, 59.4, 57.3]
From the Column: DiabetesPedigreeFunction
Outliers: [2.288, 1.893, 1.781, 0.088, 0.085, 0.084, 2.329, 0.089, 0.092, 0.078, 2.137, 1.731, 2.42, 0.085, 1.699, 0.088]
From the Column: Age
Outliers: [69, 72, 81, 70, 68, 69]
From the Column: Outcome
Outliers: []
```

Figure 6 - Outliers: Winsorization result

7.1. Data Visualization before treatment (Pre-Processing)

7.1.1. Univariate Analysis

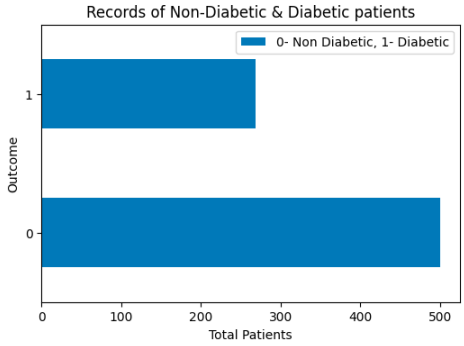


Figure 7 - Univariate Analysis: Records of Non-Diabetic & Diabetic patients

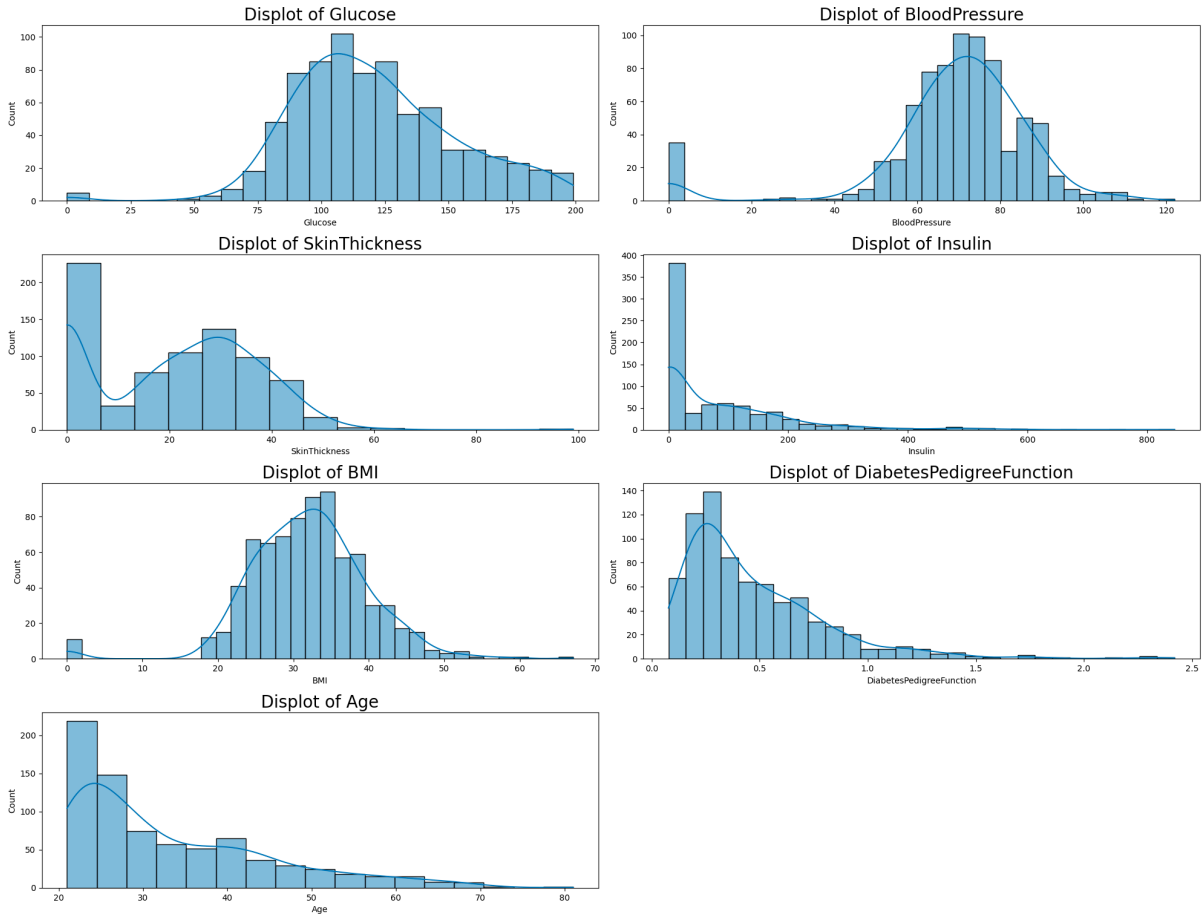


Figure 8 - Univariate Analysis: Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, and Age

The following conclusions were reached about the previous graphics: some columns are normally distributed, and some columns are right skewed:

- Column distributions that are normally or roughly normally distributed are glucose, blood pressure, skin thickness, and BMI.
- Column distributions that are rightly skewed are Insulin, Diabetes Pedigree Function, and Age.

7.1.2. Bivariate Analysis related to the target column

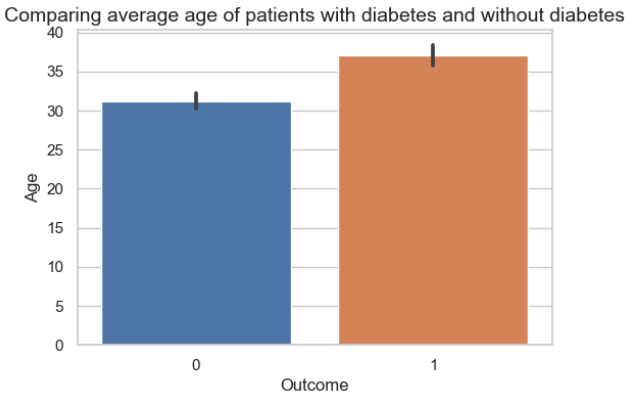


Figure 9 - Bivariate Analysis related to target column: Comparing the average age of patients with diabetes and without diabetes

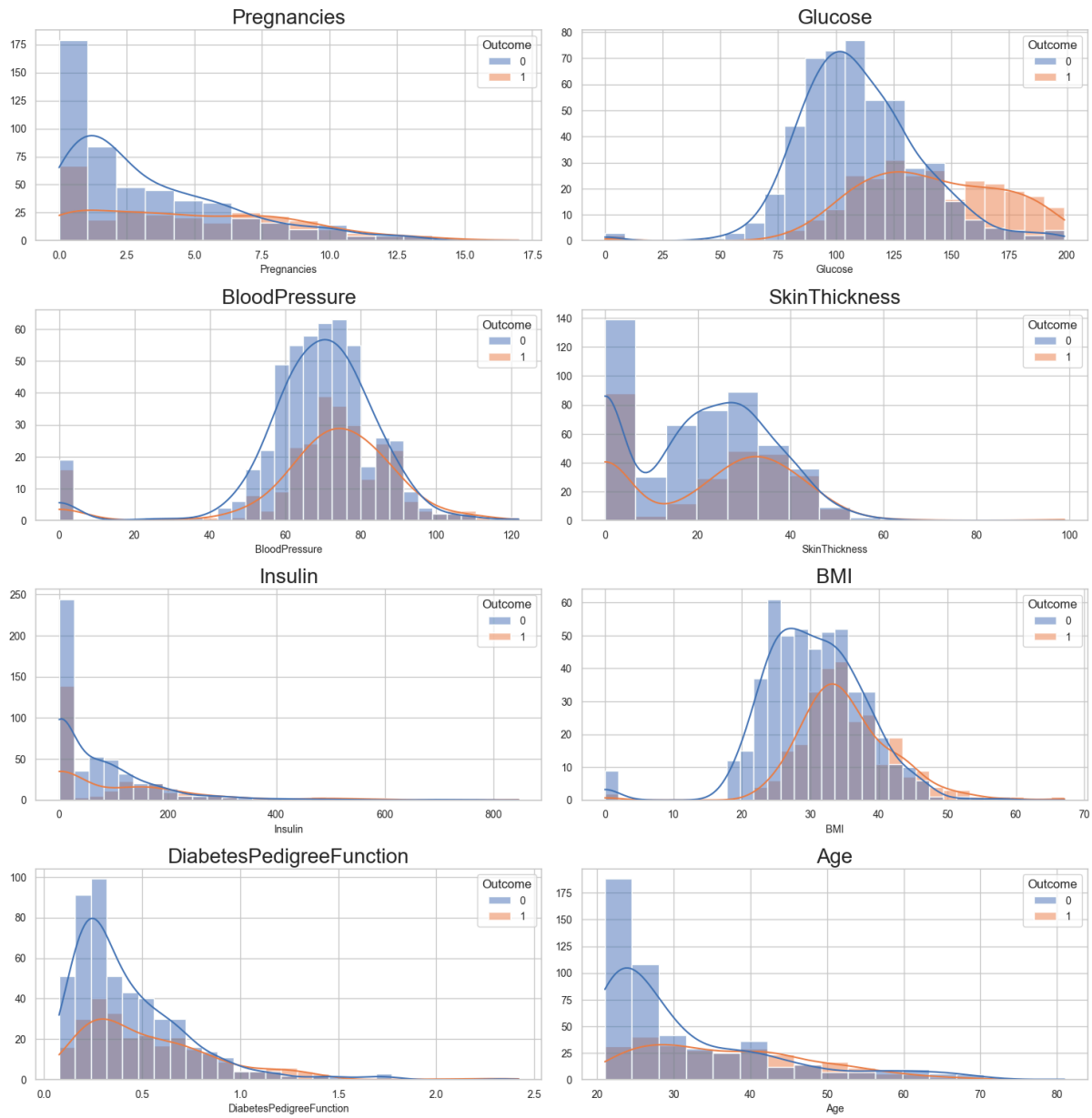


Figure 10 - Bivariate Analysis related to target column: Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, and Age

When observing the above graphics, it seems that <u>there is a difference:</u>

- In the age of the patient with and without diabetes.
- In the diabetes pedigree function of patients with and without diabetes.
- In the BMI of patients with diabetes and without diabetes.
- In the average insulin level of the patient with diabetes and without diabetes.
- In the average skin thickness of patients with diabetes and without diabetes.
- In the Blood pressure of patients with diabetes and without diabetes.
- The glucose level of patients with diabetes and without diabetes.

To confirm if these differences are significant or not and can have an impact further in the experiment a statistical test (T-Test)[10] was done to assess further this.

T-Test determined if there was a significant difference in the average per feature for diabetic or non-diabetic patients. The results of the T-Test are as follows:

- We can reject the null hypothesis. There is a difference in the average of patients with diabetes and without diabetes, for:
    - Age, Diabetes Pedigree Function, BMI, Insulin, and Glucose.
- We cannot reject the null hypothesis. There is no difference in the average of patients with diabetes and without diabetes, for:
    - Blood Pressure.

[10] Appendix - Figure 18: T-Test: Age; Appendix - Figure 19: T-Test: Diabetes Pedigree Function; Appendix - Figure 20: T-Test: BMI; Appendix - Figure 21: T-Test: Insulin; Appendix - Figure 22 - T-Test: Skin Thickness; Appendix - Figure 23: T-Test: Blood Pressure; Appendix - Figure 24 - T-Test: Glucose

7.1.3. Multivariate Graphics

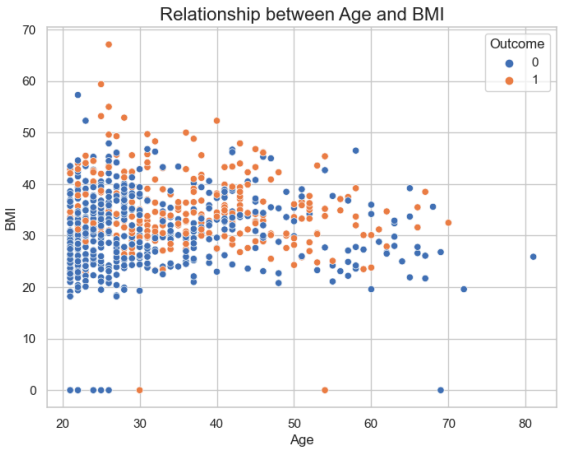


Figure 11 - Multivariate Graphics: relationship between Age and BMI

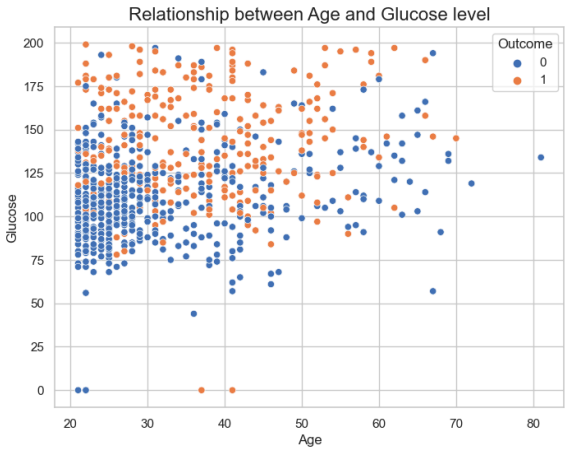


Figure 12 - Multivariate Graphics: relationship between Age and glucose

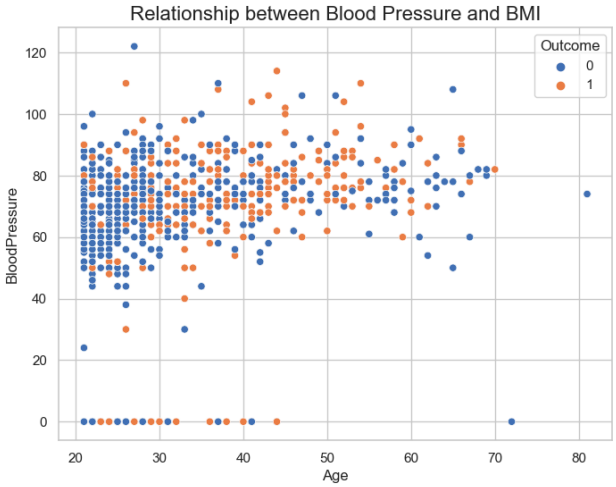


Figure 13- Multivariate Graphics: relationship between blood pressure and BMI

Main conclusions: as observed in the graphics for each variable, it's possible to observe that there is a difference in the mean of a variable for a diabetic and a non-diabetic patient. By applying the T-Test to determine if the difference suggested by the graph is real or not, it possible to conclude/observe that there is a tendency for a patient that suffers from diabetes to have the following characteristics:

- They are older.
- Have high glucose levels.
- Have high blood pressure.
- Have high BMI.
- Have high diabetes pedigree function.

Moreover, at the begging of this chapter, several business questions were listed, and all these questions were answered and represented in the present chapter with the resource of graphical

representation or using functional and technical analysis and describing the results and presenting them on the appendix chapter.

8. Data Preparation (CRISP-DM - phase 3)

During the previous phase of CRISP-DM (Data Understanding), when analyzing the raw dataset, it was detected some possible issues with the original dataset. Therefore, this phase will cover all activities to construct the final dataset, based on the initial raw data (select, clean, construct, integrate, and format/re-format data).

- Null values (zero values) were removed for glucose, blood pressure, skin thickness, insulin, and BMI. For glucose and blood pressure, these values were replaced by the mean, and the remaining were replaced by the median[11].

  Figure 15, shows the plots of individual variables after the removal of null values (0 values).
- Outliers were treated based on the previous finds, described in chapter 7 – Data Understanding[12].
- Min Max Scaler algorithm[13] was then used to rescale data feature values to a common range [0, 1].
- Algorithm Extra Trees Classifier was used to detect the most relevant features in dataset[14].

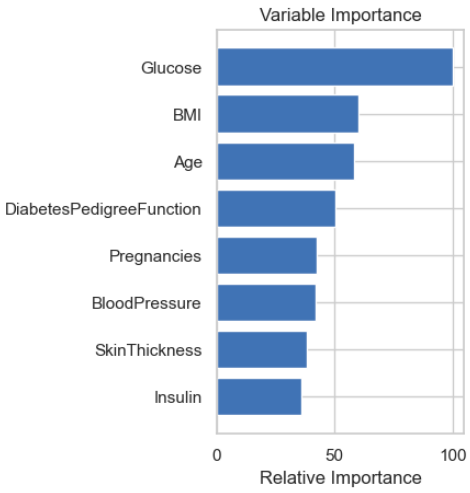


Figure 14 - Extra Trees Classifier, features importance

Based on the finds, of this algorithm, only the top 4 features (glucose, BMI, age and diabetes pedigree function) were selected for the next phase[15].

[11] Appendix - Figure 25: Treatment and removal of null values (0 values)
[12] Appendix - Figure 26 - Treatment and removal of Outliers
[13] Appendix - Figure 27 - MinMaxScaler algorithm
[14] Appendix - Figure 28 – Feature Selection: Check the most relevant features in the dataset for the modelsAppendix - Figure 28 – Feature Selection: Check the most relevant features
[15] Appendix - Figure 29 - Feature Selection: Extract the most relevant features in the dataset for the modelsAppendix - Figure 29 - Feature Selection: Extract the most relevant features in the dataset for the models

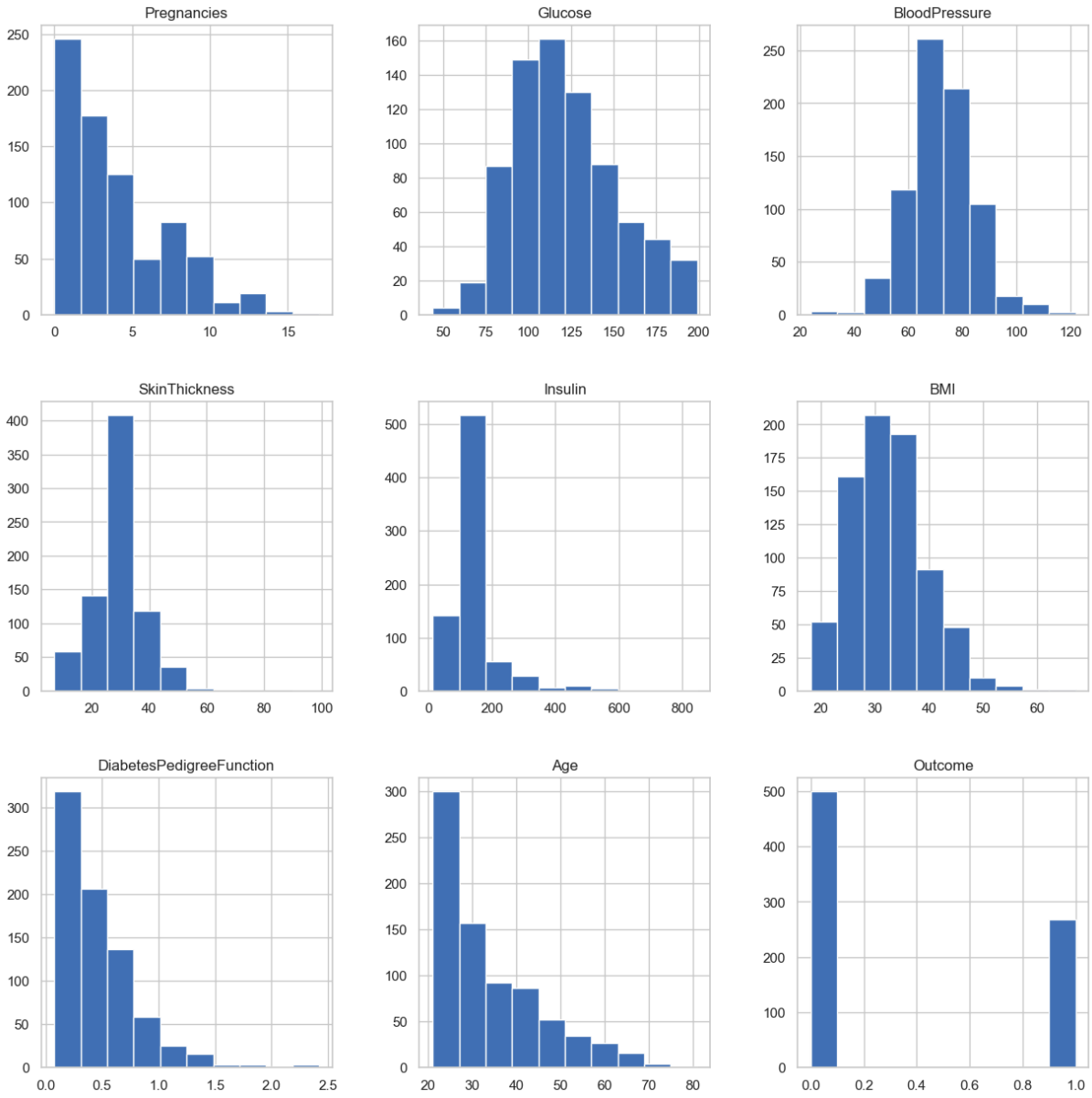


Figure 15 - Plots of individual variables after the removal of null values (0 values)

9. Modeling (CRISP-DM - phase 4)

In the modeling phase of this problem, multiple classification models were selected and compared in the foundation of what could be a Data Pipeline, with the generation of the model mainly through libraries in python like *Sklearn* and *XGBoost* as a Standalone Library of the process. Although was pondered the possibility to add Neural Networks in this pipelined test of models, due to the sake of the process and the project itself, the classification models used for this process were mainly traditional Machine Learning classifiers that were used for the process.

The classifiers used in this model were:

- Logistic Regression
- Linear Discriminant Analysis
- K-Neighbors Classifier
- Decision Tree Classifier
- Gaussian NB
- Support Vector Machine Classifier
- Adaboost Classifier
- Gradient Boosting Classifier
- Random Forest Classifier
- Extra Tree Classifiers
- Extreme Gradient Boost Classifier (XGBoost)

Of all of these, the most promising of them was the XGBoost Classifier, due to its track record of success in many Machine Learning competitions and experiments. However, surprisingly when this was not exactly the result got in the evaluation of each one of the respective models.

For this experiment (project), and for the sake of effectivity, all the models run with their standard parameters. The dataset for testing was split into a proportion of 25% to 75%[16]. And a 10-Fold cross-validation was used[17].

[16] Appendix - Figure 31 – Classifiers used in this model
[17] Appendix - Figure 32 – 10-Fold Cross Validation

```
LR: 0.764035 (0.060710)
LDA: 0.764065 (0.058034)
KNN: 0.718784 (0.087651)
CART: 0.692650 (0.061552)
NB: 0.758863 (0.067307)
SVM: 0.769147 (0.072132)
AB: 0.736207 (0.063703)
GBM: 0.741319 (0.075488)
RF: 0.730853 (0.075273)
ET: 0.729310 (0.070007)
XGB: 0.723956 (0.065987)
```

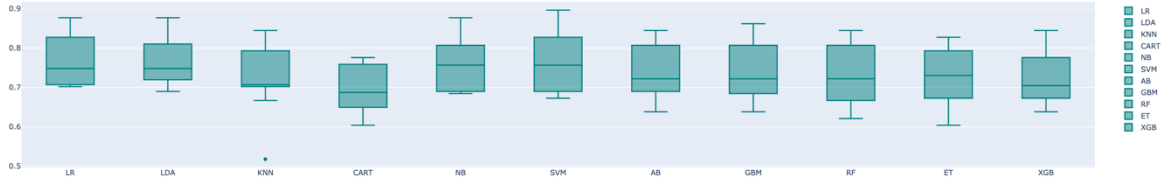


Figure 16 - Classifier generic results

10. Evaluation (CRISP-DM - phase 5)

For the evaluation of the respective classification models, the metrics used were Accuracy, F1-Score, and ROC/AUC Curves

F1-Score is a measure of a model's accuracy that considers both the precision and the recall of the model. Precision is the proportion of correct positive predictions made by the model, while recall is the proportion of actual positive cases that the model was able to identify. The F1-Score is calculated as the harmonic mean of the precision and recall and is a useful metric for comparing the performance of different models.

ROC, or Receiver Operating Characteristic, is a plot that shows the relationship between the false positive rate and the true positive rate of a classification model. The false positive rate is the proportion of negative cases that are incorrectly classified as positive, while the true positive rate is the proportion of positive cases that are correctly classified as such. The ROC curve is a useful tool for evaluating a model's performance and comparing it to other models.

Finally, the accuracy of a classification model is simply the proportion of data points that the model correctly classifies. This is a commonly used metric, but it can be misleading, especially when the data is imbalanced (i.e., when there are unequal numbers of positive and negative cases). In such cases, the F1-Score and ROC should be more informative measures of a model's performance.

In this experiment, many models measured, in terms of performance, almost the same among them. As the following values returned for the respective models, in mean accuracy after the 10-fold process and its standard deviation between parentheses.

The below table shows the accuracy, of each model tested in this experiment:

Table 3 – Model Accuracy and mean Standard Deviation

Model Acronym	Mean Accuracy	Mean Standard Deviation
LR	0.764035	0.060710
LDA	0.764065	0.058034
KNN	0.718784	0.087651
CART	0.692650	0.061552
NB	0.758863	0.067307
SVM	0.769147	0.072132
AB	0.736207	0.063703
GBM	0.741319	0.075488
RF	0.730853	0.075273
ET	0.729310	0.070007
XGB	0.723956	0.065987

With this result, it's possible to confirm the information returned from the experiment (notebook) where most of the models had a similar result. In other aspects, it's possible to notice that some models have slightly better results in the outcome of the models. The main conclusions are the following:

- The Gradient Boost Classifier has a better classification in a ROC metric (0.757), mostly due to its good tolerance for imbalanced data, and its ability to combine multiple weak classifications into a strong one, which is the case in this dataset, which is the strongest one of the classifiers correlate with the Outcome variable of 54%.
- The Linear Discriminant Analysis should also be highlighted due to its good accuracy result and F1-Score concerning other classifiers (0.7640 and 0.78 respectively). This slightly better accuracy has a foundation in the logic behind the classifier itself. This specific classifier works with a linear method of resolution where you can generate linear methods between the classes and ensemble them afterward. The Classification also works with a Generative model, that stacks multiple linear regressors into a classifier, different than a method like an SVC, that utilizes discriminative bounds to split the classes among the model.

More details about the metrics obtained during the evaluation phase and the testes conducted can be found in the appendix section, in the following figures:

- Appendix - Figure 34 – Evaluation results, for precision, recall, F1-score, and support for each classifier model
- Appendix - Figure 35 – ROC Curve for each classifier model

11. Conclusions

This project allowed the possibility to evaluate the importance of Data Science and Data Mining in real-world problems and how they could directly affect and improve day-to-day world decisions.

For the specific problem of dataset classification based on health and physical parameters, it was possible to get through testing multiple classifiers at a reasonably good accuracy of 75% on most cases, mostly in denying a case of Diabetes and then confirming it, due to its imbalanced dataset. A systematic effort was made to design a system that results in the prediction of disease like diabetes and evaluate their results.

However, even with an imbalanced dataset, it was managed to understand how the preprocessing steps work and are done and how it's possible to evaluate machine learning models for future problems. This will also allow for future work (and for the team), to have a more critical view of future classification problems, that will be faced in the future and are possible to counsel and directional the efforts of the team for a better outcome even before the models are applicable.

12. Future Work

An important real-world medical problem is the detection of diabetes at its early stage. For future works of this experiment, one of the main targets should be to enhance each one of the models with hyperparameters that are more suitable for each of them. Another important enhancement would be to create a step process in the notebook to evaluate the results and outcomes of the model at each step of the pre-processing process, with the intent to understand how each step affects the outcome and which steps just add complexity to the final output without any gain on accuracy or performance.

The work can also be extended and improved with other machine learning algorithms like Neural Networks.

13. Bibliography

CRISP-DM. n.d. <https://web.archive.org/web/20220401041957/https://www.the-modeling-agency.com/crisp-dm.pdf>.

Vijayan, V.V., Anjali, C., . "Prediction and diagnosis of diabetes mellitus A machine learning approach." *IEEE Recent Advances in Intelligent Computational Systems (RAICS)*. 122–127doi:10.1109/RAICS.2015.7488400 vols. 2015.

IDF, International Diabetes Federation. *IDF Diabetes Atlas 10th Edition 2021*. 2021. <https://diabetesatlas.org/>.

WHO. *World Health Organization*. 2022. <https://www.who.int/news-room/fact-sheets/detail/diabetes>.

Mayo Clinic. n.d. 10 2022. <https://www.mayoclinic.org/diseases-conditions/diabetes/diagnosis-treatment/drc-20371451>.

14. Appendix

```
shape = df.shape #total nr of rows and colums
print(f"Dataframe shape: {shape}")
```
✓ 0.4s

```
Dataframe shape: (768, 9)
```

Appendix - Figure 1 - Dataframe Shape

```
df.info(verbose=True) #colums and corresponding data
```
✓ 0.7s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Appendix - Figure 2 - Dataframe columns and corresponding data

```
df.isna().sum()
```
✓ 0.4s

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Appendix - Figure 3 - Missing Values

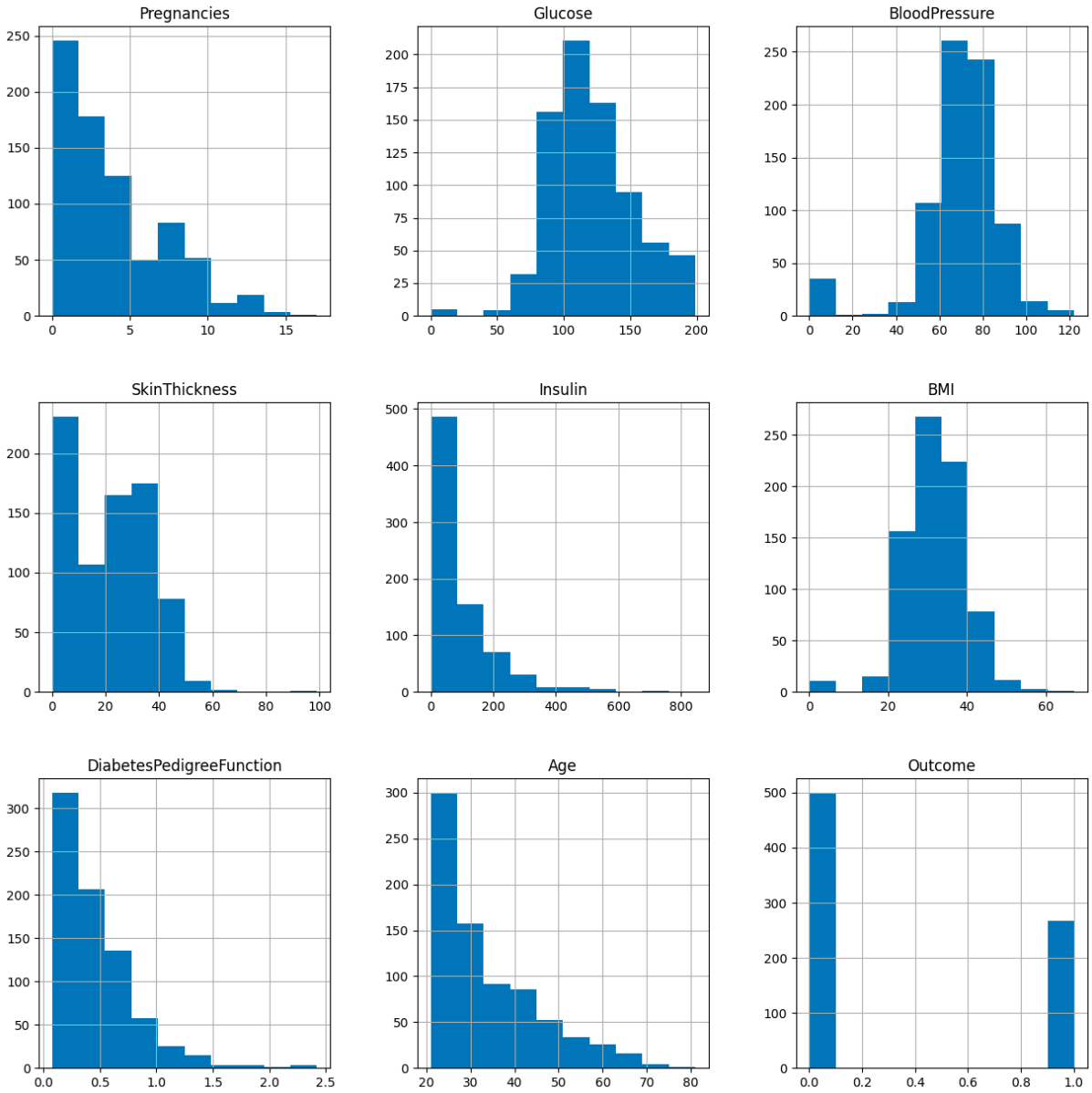
Appendix - Figure 4 – Missing Values Matrix

```
df.duplicated().value_counts() # check for duplicate values
```

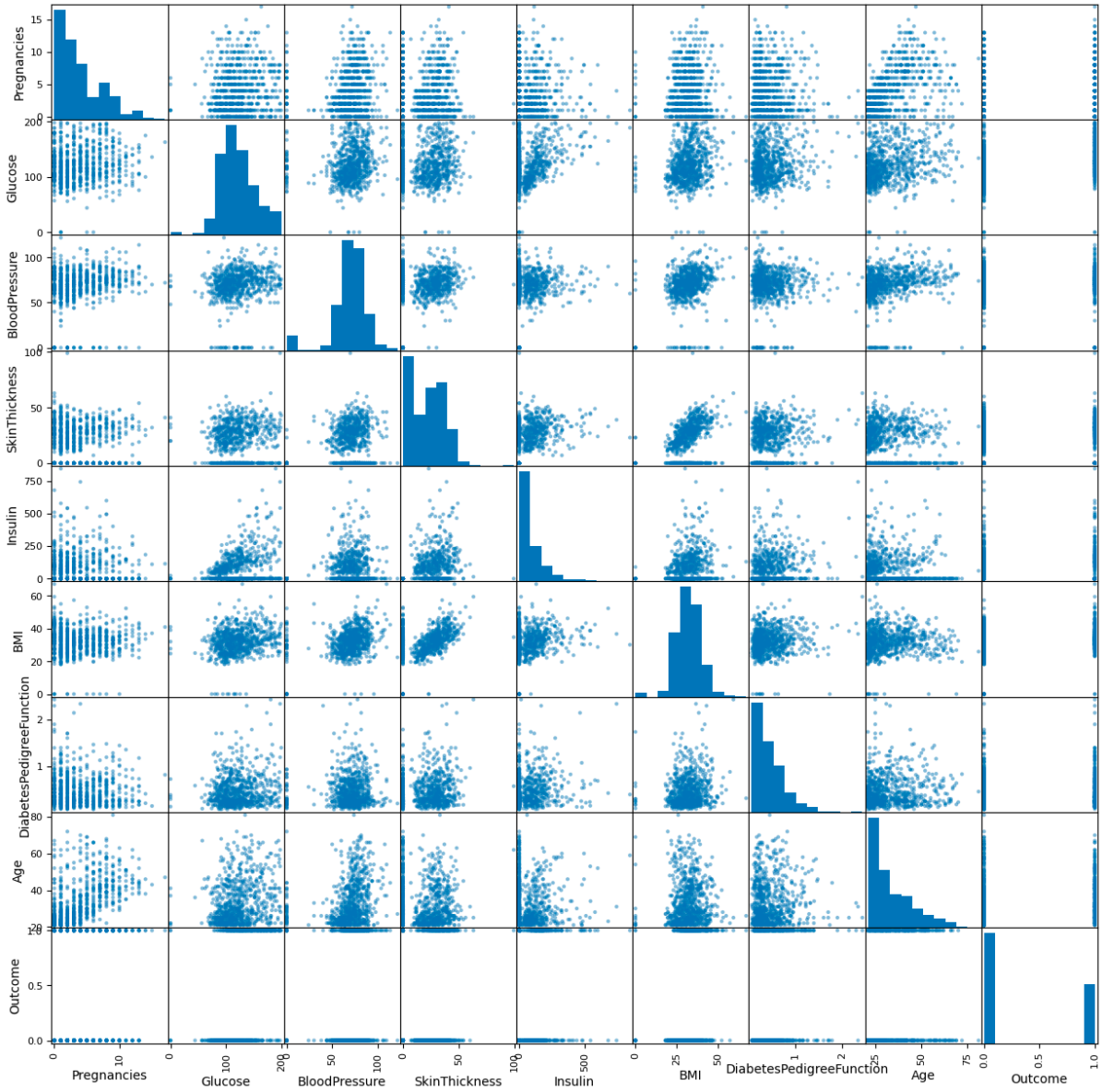
✓ 0.3s

```
False    768
dtype: int64
```

Appendix - Figure 5 - Redundant Data



Appendix - Figure 6 – Plots of individual variables of the raw dataset



Appendix - Figure 7 - Plots of scatter plots of individual variables of the raw dataset

```
print("Glucose Values:",sorted(df.Glucose.unique()))
```

✓ 0.3s Python

```
Glucose Values: [0, 44, 56, 57, 61, 62, 65, 67, 68, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102,
103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176,
177, 178, 179, 180, 181, 182, 183, 184, 186, 187, 188, 189, 190, 191, 193, 194, 195, 196, 197, 198, 199]
```

```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df[cols][df['Glucose']==0]
```

✓ 0.4s Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
75	1	0	48	20	0	24.7	0.140	22	0
182	1	0	74	20	23	27.7	0.299	21	0
342	1	0	68	35	0	32.0	0.389	22	0
349	5	0	80	32	0	41.0	0.346	37	1
502	6	0	68	41	0	39.0	0.727	41	1

Appendix - Figure 8 – Glucose 0 values

```
print("Blood Presure Values:",sorted(df.BloodPressure.unique()))
```
✓ 0.3s Python

```
Blood Presure Values: [0, 24, 30, 38, 40, 44, 46, 48, 50, 52, 54, 55, 56, 58, 60, 61, 62, 64, 65, 66, 68, 70, 72, 74, 75, 76, 78, 80, 82, 84, 85, 86, 88, 90, 92, 94, 95, 96, 98, 100, 102, 104, 106, 108, 110, 114, 122]
```

```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df[cols][df['BloodPressure']==0].head(15)
```
✓ 0.4s Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	10	115	0	0	0	35.3	0.134	29	0
15	7	100	0	0	0	30.0	0.484	32	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0
78	0	131	0	0	0	43.2	0.270	26	1
81	2	74	0	0	0	0.0	0.102	22	0
172	2	87	0	23	0	28.9	0.773	25	0
193	11	135	0	0	0	52.3	0.578	40	1
222	7	119	0	0	0	25.2	0.209	37	0
261	3	141	0	0	0	30.0	0.761	27	1
266	0	138	0	0	0	36.3	0.933	25	1
269	2	146	0	0	0	27.5	0.240	28	1
300	0	167	0	0	0	32.3	0.839	30	1
332	1	180	0	0	0	43.3	0.282	41	1
336	0	117	0	0	0	33.8	0.932	44	0

Appendix - Figure 9 – Blood Pressure 0 values

```
print("Skin Thickness Values in mm:",sorted(df.SkinThickness.unique()))
```
✓ 0.4s Python

```
Skin Thickness Values in mm: [0, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 54, 56, 60, 63, 99]
```

```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df[cols][df['SkinThickness']==0].head(15)
```
✓ 0.4s Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
2	8	183	64	0	0	23.3	0.672	32	1
5	5	116	74	0	0	25.6	0.201	30	0
7	10	115	0	0	0	35.3	0.134	29	0
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
15	7	100	0	0	0	30.0	0.484	32	1
17	7	107	74	0	0	29.6	0.254	31	1
21	8	99	84	0	0	35.4	0.388	50	0
22	7	196	90	0	0	39.8	0.451	41	1
26	7	147	76	0	0	39.4	0.257	43	1
29	5	117	92	0	0	34.1	0.337	38	0
33	6	92	92	0	0	19.9	0.188	28	0
36	11	138	76	0	0	33.2	0.420	35	0

Appendix - Figure 10 – Skin Thickness 0 values

```
print("Insulin Values:",sorted(df.Insulin.unique()))
```
✓ 0.4s Python

```
Insulin Values: [0, 14, 15, 16, 18, 22, 23, 25, 29, 32, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 70, 71,
72, 73, 74, 75, 76, 77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 94, 95, 96, 99, 100, 105, 106, 108, 110, 112, 114, 115, 116, 119, 120, 122, 125, 126, 127, 128, 129, 130,
132, 135, 140, 142, 144, 145, 146, 148, 150, 152, 155, 156, 158, 159, 160, 165, 166, 167, 168, 170, 171, 175, 176, 178, 180, 182, 183, 184, 185, 188, 190, 191, 192, 193, 194, 196, 200,
204, 205, 207, 210, 215, 220, 225, 228, 230, 231, 235, 237, 240, 245, 249, 250, 255, 258, 265, 270, 271, 272, 274, 275, 277, 278, 280, 284, 285, 291, 293, 300, 304, 310, 318, 321, 325,
326, 328, 330, 335, 342, 360, 370, 375, 387, 392, 402, 415, 440, 465, 474, 478, 480, 485, 495, 510, 540, 543, 545, 579, 600, 680, 744, 846]
```

```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df[cols][df['Insulin']==0].head(10)
```
✓ 0.8s Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
5	5	116	74	0	0	25.6	0.201	30	0
7	10	115	0	0	0	35.3	0.134	29	0
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
15	7	100	0	0	0	30.0	0.484	32	1

Appendix - Figure 11 - Insulin 0 values

```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df[cols][df['BMI']==0].head(10)
```
✓ 0.4s Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
9	8	125	96	0	0	0.0	0.232	54	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0
81	2	74	0	0	0	0.0	0.102	22	0
145	0	102	75	23	0	0.0	0.572	21	0
371	0	118	64	23	89	0.0	1.731	21	0
426	0	94	0	0	0	0.0	0.256	25	0
494	3	80	0	0	0	0.0	0.174	22	0
522	6	114	0	0	0	0.0	0.189	26	0
684	5	136	82	0	0	0.0	0.640	69	0

- Diabetes Pedigree

```
print("Diabetes Pedigree Function Values :",sorted(df.DiabetesPedigreeFunction.unique()))
```
✓ 0.4s Python

```
Diabetes Pedigree Function Values : [0.078, 0.084, 0.085, 0.088, 0.089, 0.092, 0.096, 0.1, 0.101, 0.102, 0.107, 0.108, 0.115, 0.118, 0.121, 0.122, 0.123, 0.126, 0.127, 0.128, 0.129,
0.13, 0.133, 0.134, 0.135, 0.136, 0.137, 0.138, 0.14, 0.141, 0.142, 0.143, 0.144, 0.145, 0.147, 0.148, 0.149, 0.15, 0.151, 0.153, 0.154, 0.155, 0.156, 0.157, 0.158, 0.159, 0.16, 0.161,
0.162, 0.163, 0.164, 0.165, 0.166, 0.167, 0.17, 0.171, 0.173, 0.174, 0.175, 0.176, 0.177, 0.178, 0.179, 0.18, 0.181, 0.182, 0.183, 0.186, 0.187, 0.188, 0.189, 0.19, 0.191, 0.192, 0.194,
0.196, 0.197, 0.198, 0.199, 0.2, 0.201, 0.203, 0.204, 0.205, 0.206, 0.207, 0.209, 0.21, 0.212, 0.215, 0.217, 0.218, 0.219, 0.22, 0.221, 0.222, 0.223, 0.225, 0.226, 0.227, 0.229, 0.23,
0.231, 0.232, 0.233, 0.234, 0.235, 0.236, 0.237, 0.238, 0.239, 0.24, 0.241, 0.243, 0.244, 0.245, 0.246, 0.247, 0.248, 0.249, 0.251, 0.252, 0.253, 0.254, 0.255, 0.256, 0.257, 0.258,
0.259, 0.26, 0.261, 0.262, 0.263, 0.264, 0.265, 0.267, 0.268, 0.269, 0.27, 0.271, 0.272, 0.277, 0.278, 0.279, 0.28, 0.282, 0.283, 0.284, 0.285, 0.286, 0.287, 0.289, 0.29, 0.292, 0.293,
0.294, 0.295, 0.296, 0.297, 0.299, 0.3, 0.302, 0.303, 0.304, 0.305, 0.306, 0.307, 0.313, 0.314, 0.315, 0.317, 0.318, 0.319, 0.323, 0.324, 0.325, 0.326, 0.328, 0.329, 0.33, 0.331, 0.332,
0.334, 0.335, 0.336, 0.337, 0.338, 0.34, 0.341, 0.342, 0.343, 0.344, 0.345, 0.346, 0.347, 0.349, 0.351, 0.352, 0.355, 0.356, 0.358, 0.361, 0.362, 0.364, 0.365, 0.366, 0.368, 0.37, 0.371,
0.374, 0.375, 0.376, 0.378, 0.38, 0.381, 0.382, 0.383, 0.385, 0.388, 0.389, 0.391, 0.393, 0.394, 0.395, 0.396, 0.398, 0.399, 0.4, 0.401, 0.402, 0.403, 0.404, 0.407, 0.408, 0.409, 0.411,
0.412, 0.415, 0.416, 0.417, 0.419, 0.42, 0.421, 0.422, 0.423, 0.426, 0.427, 0.43, 0.431, 0.432, 0.433, 0.434, 0.435, 0.439, 0.441, 0.443, 0.444, 0.446, 0.447, 0.451, 0.452, 0.453, 0.454,
0.455, 0.457, 0.46, 0.463, 0.464, 0.465, 0.466, 0.467, 0.471, 0.472, 0.479, 0.482, 0.483, 0.484, 0.485, 0.487, 0.488, 0.491, 0.493, 0.495, 0.496, 0.497, 0.498, 0.499, 0.501, 0.502,
0.503, 0.507, 0.509, 0.51, 0.512, 0.514, 0.515, 0.516, 0.52, 0.525, 0.526, 0.527, 0.528, 0.529, 0.532, 0.534, 0.536, 0.537, 0.539, 0.542, 0.543, 0.545, 0.546, 0.547, 0.549, 0.551, 0.554,
0.557, 0.559, 0.56, 0.561, 0.564, 0.565, 0.569, 0.571, 0.572, 0.575, 0.578, 0.58, 0.582, 0.583, 0.586, 0.587, 0.588, 0.591, 0.593, 0.595, 0.597, 0.598, 0.6, 0.601, 0.605, 0.607, 0.61,
0.612, 0.613, 0.614, 0.615, 0.619, 0.624, 0.626, 0.627, 0.629, 0.63, 0.631, 0.637, 0.64, 0.645, 0.646, 0.647, 0.649, 0.652, 0.654, 0.655, 0.658, 0.66, 0.661, 0.665, 0.666, 0.672, 0.673,
0.674, 0.677, 0.678, 0.68, 0.682, 0.686, 0.687, 0.692, 0.693, 0.695, 0.696, 0.698, 0.699, 0.702, 0.703, 0.704, 0.705, 0.709, 0.711, 0.717, 0.718, 0.719, 0.721, 0.722, 0.725, 0.727, 0.73,
0.731, 0.732, 0.733, 0.734, 0.735, 0.738, 0.741, 0.742, 0.743, 0.744, 0.745, 0.748, 0.757, 0.759, 0.761, 0.766, 0.767, 0.771, 0.773, 0.785, 0.787, 0.801, 0.803, 0.804, 0.805, 0.808,
0.813, 0.816, 0.817, 0.821, 0.825, 0.826, 0.828, 0.831, 0.832, 0.833, 0.839, 0.84, 0.845, 0.851, 0.855, 0.856, 0.867, 0.871, 0.874, 0.875, 0.878, 0.88, 0.881, 0.886, 0.892, 0.893, 0.904,
0.905, 0.917, 0.925, 0.926, 0.93, 0.932, 0.933, 0.944, 0.947, 0.949, 0.955, 0.956, 0.962, 0.966, 0.968, 0.97, 0.997, 1.001, 1.021, 1.022, 1.034, 1.057, 1.072, 1.076, 1.095, 1.096, 1.101,
1.114, 1.127, 1.136, 1.138, 1.144, 1.154, 1.159, 1.162, 1.174, 1.182, 1.189, 1.191, 1.213, 1.222, 1.224, 1.251, 1.258, 1.268, 1.282, 1.292, 1.318, 1.321, 1.353, 1.39, 1.391, 1.394, 1.4,
1.441, 1.461, 1.476, 1.6, 1.698, 1.699, 1.731, 1.781, 1.893, 2.137, 2.288, 2.329, 2.42]
```

Appendix - Figure 12 - BMI 0 values

```
print("Diabetes Pedigree Function Values :",sorted(df.DiabetesPedigreeFunction.unique()))
```
✓ 0.4s Python

```
Diabetes Pedigree Function Values : [0.078, 0.084, 0.085, 0.088, 0.089, 0.092, 0.096, 0.1, 0.101, 0.102, 0.107, 0.108, 0.115, 0.118, 0.121, 0.122, 0.123, 0.126, 0.127, 0.128, 0.129,
0.13, 0.133, 0.134, 0.135, 0.136, 0.137, 0.138, 0.14, 0.141, 0.142, 0.143, 0.144, 0.145, 0.147, 0.148, 0.149, 0.15, 0.151, 0.153, 0.154, 0.155, 0.156, 0.157, 0.158, 0.159, 0.16, 0.161,
0.162, 0.163, 0.164, 0.165, 0.166, 0.167, 0.17, 0.171, 0.173, 0.174, 0.175, 0.176, 0.177, 0.178, 0.179, 0.18, 0.181, 0.182, 0.183, 0.186, 0.187, 0.188, 0.189, 0.19, 0.191, 0.192, 0.194,
0.196, 0.197, 0.198, 0.199, 0.2, 0.201, 0.203, 0.204, 0.205, 0.206, 0.207, 0.209, 0.21, 0.212, 0.215, 0.217, 0.218, 0.219, 0.22, 0.221, 0.222, 0.223, 0.225, 0.226, 0.227, 0.229, 0.23,
0.231, 0.232, 0.233, 0.234, 0.235, 0.236, 0.237, 0.238, 0.239, 0.24, 0.241, 0.243, 0.244, 0.245, 0.246, 0.247, 0.248, 0.249, 0.251, 0.252, 0.253, 0.254, 0.255, 0.256, 0.257, 0.258,
0.259, 0.26, 0.261, 0.262, 0.263, 0.264, 0.265, 0.267, 0.268, 0.269, 0.27, 0.271, 0.272, 0.277, 0.278, 0.279, 0.28, 0.282, 0.283, 0.284, 0.285, 0.286, 0.287, 0.289, 0.29, 0.292, 0.293,
0.294, 0.295, 0.296, 0.297, 0.299, 0.3, 0.302, 0.303, 0.304, 0.305, 0.306, 0.307, 0.313, 0.314, 0.315, 0.317, 0.318, 0.319, 0.323, 0.324, 0.325, 0.326, 0.328, 0.329, 0.33, 0.331, 0.332,
0.334, 0.335, 0.336, 0.337, 0.338, 0.34, 0.341, 0.342, 0.343, 0.344, 0.345, 0.346, 0.347, 0.349, 0.351, 0.352, 0.355, 0.356, 0.358, 0.361, 0.362, 0.364, 0.365, 0.366, 0.368, 0.37, 0.371,
0.374, 0.375, 0.376, 0.378, 0.38, 0.381, 0.382, 0.383, 0.385, 0.388, 0.389, 0.391, 0.393, 0.394, 0.395, 0.396, 0.398, 0.399, 0.4, 0.401, 0.402, 0.403, 0.404, 0.407, 0.408, 0.409, 0.411,
0.412, 0.415, 0.416, 0.417, 0.419, 0.42, 0.421, 0.422, 0.423, 0.426, 0.427, 0.43, 0.431, 0.432, 0.433, 0.434, 0.435, 0.439, 0.441, 0.443, 0.444, 0.446, 0.447, 0.451, 0.452, 0.453, 0.454,
0.455, 0.457, 0.46, 0.463, 0.464, 0.465, 0.466, 0.467, 0.471, 0.472, 0.479, 0.482, 0.483, 0.484, 0.485, 0.487, 0.488, 0.491, 0.493, 0.495, 0.496, 0.497, 0.498, 0.499, 0.501, 0.502,
0.503, 0.507, 0.509, 0.51, 0.512, 0.514, 0.515, 0.516, 0.52, 0.525, 0.526, 0.527, 0.528, 0.529, 0.532, 0.534, 0.536, 0.537, 0.539, 0.542, 0.543, 0.545, 0.546, 0.547, 0.549, 0.551, 0.554,
0.557, 0.559, 0.56, 0.561, 0.564, 0.565, 0.569, 0.571, 0.572, 0.575, 0.578, 0.58, 0.582, 0.583, 0.586, 0.587, 0.588, 0.591, 0.593, 0.595, 0.597, 0.598, 0.6, 0.601, 0.605, 0.607, 0.61,
0.612, 0.613, 0.614, 0.615, 0.619, 0.624, 0.626, 0.627, 0.629, 0.63, 0.631, 0.637, 0.64, 0.645, 0.646, 0.647, 0.649, 0.652, 0.654, 0.655, 0.658, 0.66, 0.661, 0.665, 0.666, 0.672, 0.673,
0.674, 0.677, 0.678, 0.68, 0.682, 0.686, 0.687, 0.692, 0.693, 0.695, 0.696, 0.698, 0.699, 0.702, 0.703, 0.704, 0.705, 0.709, 0.711, 0.717, 0.718, 0.719, 0.721, 0.722, 0.725, 0.727, 0.73,
0.731, 0.732, 0.733, 0.734, 0.735, 0.738, 0.741, 0.742, 0.743, 0.744, 0.745, 0.748, 0.757, 0.759, 0.761, 0.766, 0.767, 0.771, 0.773, 0.785, 0.787, 0.801, 0.803, 0.804, 0.805, 0.808,
0.813, 0.816, 0.817, 0.821, 0.825, 0.826, 0.828, 0.831, 0.832, 0.833, 0.839, 0.84, 0.845, 0.851, 0.855, 0.856, 0.867, 0.871, 0.874, 0.875, 0.878, 0.88, 0.881, 0.886, 0.892, 0.893, 0.904,
0.905, 0.917, 0.925, 0.926, 0.93, 0.932, 0.933, 0.944, 0.947, 0.949, 0.955, 0.956, 0.962, 0.966, 0.968, 0.97, 0.997, 1.001, 1.021, 1.022, 1.034, 1.057, 1.072, 1.076, 1.095, 1.096, 1.101,
1.114, 1.127, 1.136, 1.138, 1.144, 1.154, 1.159, 1.162, 1.174, 1.182, 1.189, 1.191, 1.213, 1.222, 1.224, 1.251, 1.258, 1.268, 1.282, 1.292, 1.318, 1.321, 1.353, 1.39, 1.391, 1.394, 1.4,
1.441, 1.461, 1.476, 1.6, 1.698, 1.699, 1.731, 1.781, 1.893, 2.137, 2.288, 2.329, 2.42]
```

Appendix - Figure 13 – Diabetes Pedrigree Function values

```
print("Age Values :",sorted(df.Age.unique()))
```
✓ 0.4s Python

```
Age Values : [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
64, 65, 66, 67, 68, 69, 70, 72, 81]
```

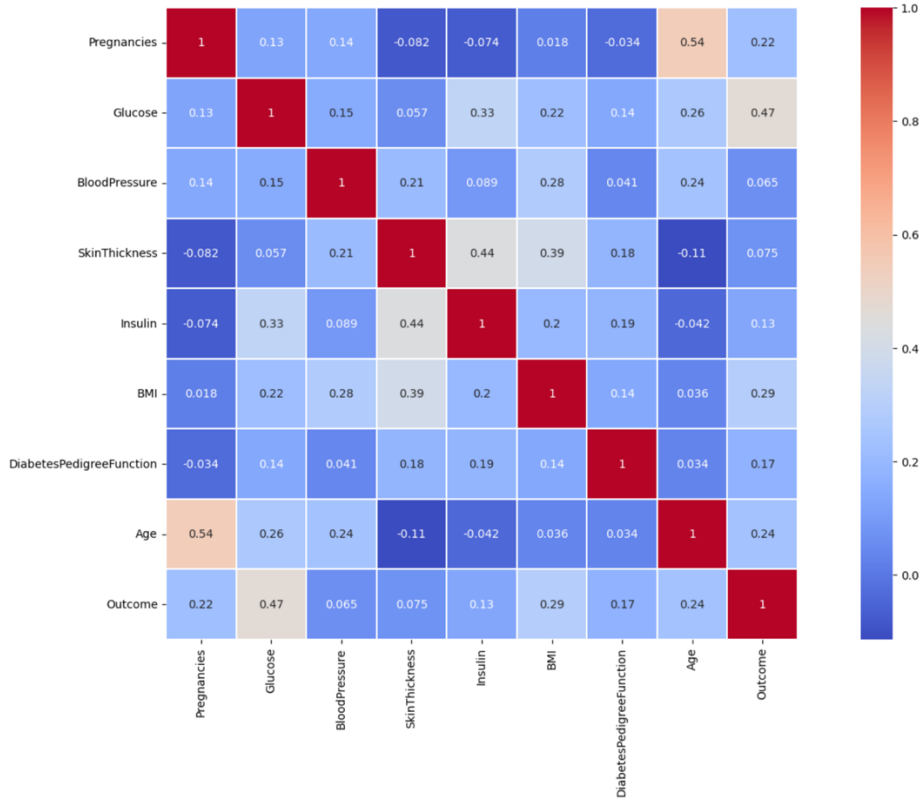
Appendix - Figure 14 - Age values

```
df.describe()
```
✓ 0.5s

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Appendix - Figure 15 - Statistics of the dataset

```
plt.subplots(figsize=(20, 10))
sns.heatmap(df.corr(),  square=True,  annot=True,linewidths=0.1, cmap="coolwarm")
plt.show()
```
✓ 0.2s



Appendix - Figure 16 - Correlation Maof the variables

```
def Zscore_outlier(df):
    out = []
    m = np.mean(df)
    sd = np.std(df)
    for i in df:
        z = (i-m)/sd
        if np.abs(z) > 3:
            out.append(i)
    print(f"Outliers:",out)
    out = []

def Winsorization_outliers(df):
    out = []
    q1 = np.percentile(df , 1)
    q3 = np.percentile(df , 99)
    for i in df:
        if i > q3 or i < q1:
            out.append(i)
    print(f"Outliers:",out)
    #Remove the value if its an outlier
    df = df[~df.isin(out)]
    return df
    out = []
```
✓ 0.3s

```
for i in df.columns:
    print(f'From the Column: {i}')
    Zscore_outlier(df[i])
```
✓ 0.4s

```
From the Column: Pregnancies
Outliers: [15, 17, 14, 14]
From the Column: Glucose
Outliers: [0, 0, 0, 0, 0]
From the Column: BloodPressure
Outliers: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
From the Column: SkinThickness
Outliers: [99]
From the Column: Insulin
Outliers: [543, 846, 495, 485, 495, 478, 744, 680, 545, 465, 579, 474, 480, 600, 440, 540, 480, 510]
From the Column: BMI
Outliers: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 67.1, 0.0, 0.0, 59.4, 0.0, 0.0, 57.3, 0.0, 0.0]
From the Column: DiabetesPedigreeFunction
Outliers: [2.288, 1.893, 1.781, 2.329, 1.476, 2.137, 1.731, 1.6, 2.42, 1.699, 1.698]
From the Column: Age
Outliers: [69, 72, 81, 70, 69]
From the Column: Outcome
Outliers: []
```

```
for i in df.columns:
    print(f'From the Column: {i}')
    Winsorization_outliers(df[i])
    #df_outliers = Winsorization_outliers(df[i])
    #df = df.append(df_outliers)
```
✓ 0.4s

```
From the Column: Pregnancies
Outliers: [15, 17, 14, 14]
From the Column: Glucose
Outliers: [197, 44, 0, 0, 197, 0, 0, 197, 0, 198, 197, 199, 56]
From the Column: BloodPressure
Outliers: [110, 108, 122, 110, 108, 110, 114]
From the Column: SkinThickness
Outliers: [60, 54, 56, 54, 52, 63, 52, 99]
From the Column: Insulin
Outliers: [543, 846, 744, 680, 545, 579, 600, 540]
From the Column: BMI
Outliers: [53.2, 55.0, 67.1, 52.3, 52.3, 52.9, 59.4, 57.3]
From the Column: DiabetesPedigreeFunction
Outliers: [2.288, 1.893, 1.781, 0.088, 0.085, 0.084, 2.329, 0.089, 0.092, 0.078, 2.137, 1.731, 2.42, 0.085, 1.699, 0.088]
From the Column: Age
Outliers: [69, 72, 81, 70, 68, 69]
From the Column: Outcome
Outliers: []
```

Appendix - Figure 17 – Outliers detection (Z-Score and Winsorization)

```
Applying student's t-test to determine if there is significant difference in the average age of people with diabetes and without diabetes.<br>
💡H0 : There is no difference in the average age of patients with diabetes and without diabetes<br>
- H1: There is difference in the average age of patients with diabetes and without diabetes
```

Markdown

```
# applying t-test to determine is their significant difference in age distribution of diabetic and non diabetic patients
age_with_diabetes = df[df["Outcome"] == 1]["Age"]
age_without_diabetes = df[df["Outcome"] == 0]["Age"]

# applying t-test
stat,p = ttest_ind(age_with_diabetes,age_without_diabetes)
if p > 0.05:
    print("P-value {}".format(p))
    print("We cannot reject the null hypothesis.There is no difference in the average age of patients with diabetes and without diabetes ")
else:
    print("P-value {}".format(p))
    print("We can reject the null hypothesis.There is a difference in the average age of patients with diabetes and without diabetes ")
```

✓ 0.1s Python

```
P-value 2.2099754606654358e-11
We can reject the null hypothesis.There is a difference in the average age of patients with diabetes and without diabetes
```

Appendix - Figure 18: T-Test: Age

```
ped_func_with_diabetes = df[df["Outcome"] == 1]["DiabetesPedigreeFunction"]
ped_func_without_diabetes = df[df["Outcome"] == 0]["DiabetesPedigreeFunction"]


stat,p = ttest_ind(ped_func_with_diabetes,ped_func_without_diabetes)
if p > 0.05:
    print("P-value {}".format(p))
    print("We cannot reject the null hypothesis.There is no difference in the average diabetes pedigree function of patients with diabetes and without diabetes ")
else:
    print("P-value {}".format(p))
    print("We can reject the null hypothesis.There is a difference in the average diabetes pedigree function of patients with diabetes and without diabetes ")
```

✓ 0.5s Python

```
P-value 1.254607010148809e-06
We can reject the null hypothesis.There is a difference in the average diabetes pedigree function of patients with diabetes and without diabetes
```

Appendix - Figure 19: T-Test: Diabetes Pedigree Function

```
bmi_with_diabetes = df[df["Outcome"] == 1]["BMI"]
bmi_without_diabetes = df[df["Outcome"] == 0]["BMI"]


stat,p = ttest_ind(ped_func_with_diabetes,ped_func_without_diabetes)
if p > 0.05:
    print("P-value {}".format(p))
    print("We cannot reject the null hypothesis.There is no difference in the average BMI of patients with diabetes and without diabetes ")
else:
    print("P-value {}".format(p))
    print("We can reject the null hypothesis.There is a difference in the average BMI of patients with diabetes and without diabetes ")
```

✓ 0.5s Python

```
P-value 1.254607010148809e-06
We can reject the null hypothesis.There is a difference in the average BMI of patients with diabetes and without diabetes
```

Appendix - Figure 20: T-Test: BMI

```
insulin_with_diabetes = df[df["Outcome"] == 1]["Insulin"]
insulin_without_diabetes = df[df["Outcome"] == 0]["Insulin"]


stat,p = ttest_ind(insulin_with_diabetes,insulin_without_diabetes)
if p > 0.05:
    print("P-value {}".format(p))
    print("We cannot reject the null hypothesis.There is no difference in the average insulin level of patients with diabetes and without diabetes ")
else:
    print("P-value {}".format(p))
    print("We can reject the null hypothesis.There is a difference in the average insulin level of patients with diabetes and without diabetes ")
```

✓ 0.6s Python

```
P-value 0.00028618646036031987
We can reject the null hypothesis.There is a difference in the average insulin level of patients with diabetes and without diabetes
```

Appendix - Figure 21: T-Test: Insulin

```python
skin_thick_with_diabetes = df[df["Outcome"] == 1]["SkinThickness"]
skin_thick_without_diabetes = df[df["Outcome"] == 0]["SkinThickness"]


stat,p = ttest_ind(skin_thick_with_diabetes,skin_thick_without_diabetes)
if p > 0.05:
    print("P-value {}".format(p))
    print("We cannot reject the null hypothesis.There is no difference in the average Skin Thickness of patients with diabetes and without diabetes ")
else:
    print("P-value {}".format(p))
    print("We can reject the null hypothesis.There is a difference in the average Skin Thickness of patients with diabetes and without diabetes ")
```

✓ 0.2s Python

```
P-value 0.03834770482049123
We can reject the null hypothesis.There is a difference in the average Skin Thickness of patients with diabetes and without diabetes
```

Appendix - Figure 22 - T-Test: Skin Thickness

```python
blood_pressure_with_diabetes = df[df["Outcome"] == 1]["BloodPressure"]
blood_pressure_without_diabetes = df[df["Outcome"] == 0]["BloodPressure"]


stat,p = ttest_ind(blood_pressure_with_diabetes,blood_pressure_without_diabetes)
if p > 0.05:
    print("P-value {}".format(p))
    print("We cannot reject the null hypothesis.There is no difference in the average Blood Pressure of patients with diabetes and without diabetes ")
else:
    print("P-value {}".format(p))
    print("We can reject the null hypothesis.There is  difference in the average Blood Pressure of patients with diabetes and without diabetes ")
```

✓ 0.3s Python

```
P-value 0.0715139000977608
We cannot reject the null hypothesis.There is no difference in the average Blood Pressure of patients with diabetes and without diabetes
```

Appendix - Figure 23: T-Test: Blood Pressure

```python
glucose_with_diabetes = df[df["Outcome"] == 1]["Glucose"]
glucose_without_diabetes = df[df["Outcome"] == 0]["Glucose"]


stat,p = ttest_ind(glucose_with_diabetes,glucose_without_diabetes)
if p > 0.05:
    print("P-value {}".format(p))
    print("We cannot reject the null hypothesis.There is no difference in the average glucose of patients with diabetes and without diabetes ")
else:
    print("P-value {}".format(p))
    print("We can reject the null hypothesis.There is a difference in the average glucose of patients with diabetes and without diabetes ")
```

✓ 0.4s Python

```
P-value 8.935431645289913e-43
We can reject the null hypothesis.There is a difference in the average glucose of patients with diabetes and without diabetes
```

Appendix - Figure 24 - T-Test: Glucose

```python
df_copy = df.copy(deep = True)
df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

## showing the count of Nans
print(df_copy.isnull().sum())
```

✓ 0.3s Python

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```python
# Inputing the missing values with their most useful parameters: mean or median.
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace = True)
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(), inplace = True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace = True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace = True)
df_copy['BMI'].fillna(df_copy['BMI'].median(), inplace = True)
```

✓ 0.1s Python

Appendix - Figure 25: Treatment and removal of null values (0 values)

2. Removing Outliers
The outliers will be treated/removed, based on the results of EDA step, where Z-Score and Winsorization test were used to acess the outliers in the dataset.

```
for i in df.columns:
    df_outliers = Winsorization_outliers(df[i])
    #df_outliers = df_outliers.append(df_outliers)
```

✓ 0.1s Python

```
Outliers: [15, 17, 14, 14]
Outliers: [197, 44, 0, 0, 197, 0, 0, 197, 0, 198, 197, 199, 56]
Outliers: [110, 108, 122, 110, 108, 110, 114]
Outliers: [60, 54, 56, 54, 52, 63, 52, 99]
Outliers: [543, 846, 744, 680, 545, 579, 600, 540]
Outliers: [53.2, 55.0, 67.1, 52.3, 52.3, 52.9, 59.4, 57.3]
Outliers: [2.288, 1.893, 1.781, 0.088, 0.085, 0.084, 2.329, 0.089, 0.092, 0.078, 2.137, 1.731, 2.42, 0.085, 1.699, 0.088]
Outliers: [69, 72, 81, 70, 68, 69]
Outliers: []
```

```
df_outliers = pd.DataFrame()
for i in df_copy.columns:
    print(f'From the Column: {i}')
    column_outliers = Winsorization_outliers(df_copy[i])
    df_outliers = df_outliers.append(column_outliers)
```

✓ 0.2s Python

```
From the Column: Pregnancies
Outliers: [15, 17, 14, 14]
From the Column: Glucose
Outliers: [197.0, 44.0, 62.0, 57.0, 197.0, 61.0, 197.0, 57.0, 198.0, 197.0, 67.0, 199.0, 56.0, 65.0]
From the Column: BloodPressure
Outliers: [40.0, 30.0, 110.0, 108.0, 122.0, 30.0, 110.0, 108.0, 110.0, 24.0, 38.0, 114.0]
From the Column: SkinThickness
Outliers: [60.0, 54.0, 56.0, 54.0, 7.0, 52.0, 8.0, 8.0, 63.0, 7.0, 52.0, 99.0]
From the Column: Insulin
Outliers: [543.0, 846.0, 23.0, 18.0, 23.0, 744.0, 680.0, 545.0, 579.0, 14.0, 18.0, 600.0, 15.0, 540.0, 22.0, 16.0]
From the Column: BMI
Outliers: [19.4, 19.1, 53.2, 55.0, 67.1, 52.3, 18.4, 52.3, 52.9, 19.3, 18.2, 18.2, 59.4, 18.2, 57.3]
From the Column: DiabetesPedigreeFunction
Outliers: [2.288, 1.893, 1.781, 0.088, 0.085, 0.084, 2.329, 0.089, 0.092, 0.078, 2.137, 1.731, 2.42, 0.085, 1.699, 0.088]
From the Column: Age
Outliers: [69, 72, 81, 70, 68, 69]
From the Column: Outcome
Outliers: []
```

```
df_outliers = df_outliers.transpose()
df_outliers.sort_index(inplace=True)
```

✓ 0.5s Python

```
df_outliers.head()
```

✓ 0.3s Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6.0	148.0	72.0	35.0	125.0	33.6	0.627	50.0	1.0
1	1.0	85.0	66.0	29.0	125.0	26.6	0.351	31.0	0.0
2	8.0	183.0	64.0	29.0	125.0	23.3	0.672	32.0	1.0
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	0.0
4	0.0	137.0	NaN	35.0	168.0	43.1	NaN	33.0	1.0

Filling up the gaps that appear from the Outliers with mean and median values

```
df_outliers['Glucose'].fillna(df_outliers['Glucose'].mean(), inplace = True)
df_outliers['BloodPressure'].fillna(df_outliers['BloodPressure'].mean(), inplace = True)
df_outliers['SkinThickness'].fillna(df_outliers['SkinThickness'].median(), inplace = True)
df_outliers['Insulin'].fillna(df_outliers['Insulin'].median(), inplace = True)
df_outliers['BMI'].fillna(df_outliers['BMI'].median(), inplace = True)
df_outliers['DiabetesPedigreeFunction'].fillna(df_outliers['DiabetesPedigreeFunction'].median(), inplace = True)
df_outliers['Age'].fillna(df_outliers['Age'].median(), inplace = True)
df_outliers['Pregnancies'].fillna(df_outliers['Pregnancies'].mean(), inplace = True)
```

✓ 0.4s Python

```
df_outliers.isnull().sum()
```

✓ 0.3s Python

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Appendix - Figure 26 - Treatment and removal of Outliers

3 - MinMaxScaler

MinMaxScaler to Enhance the Dataset Classification

```
try:
    MinMaxScaler = MinMaxScaler()
except:
    None
df_minmax = MinMaxScaler.fit_transform(df_outliers)
```

✓ 0.4s Python

```
df_minmax = pd.DataFrame(df_minmax, columns = df_outliers.columns)
df_minmax.head()
```

✓ 0.3s Python

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.461538	0.625000	0.451613	0.609756	0.206186	0.462295	0.331461	0.630435	1.0
1	0.076923	0.132812	0.354839	0.463415	0.206186	0.232787	0.159176	0.217391	0.0
2	0.615385	0.898438	0.322581	0.463415	0.206186	0.124590	0.359551	0.239130	1.0
3	0.076923	0.164062	0.354839	0.317073	0.142268	0.281967	0.044320	0.000000	0.0
4	0.000000	0.539062	0.456545	0.609756	0.294845	0.773770	0.172597	0.260870	1.0

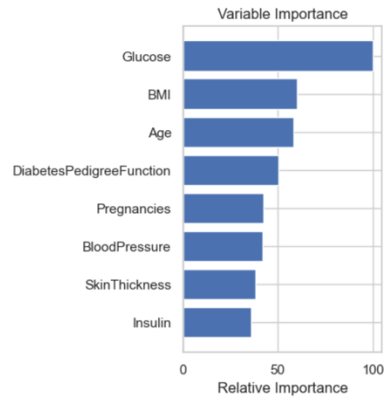
Appendix - Figure 27 - MinMaxScaler algorithm

```
x = df_minmax[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age']]
y = df_minmax['Outcome']

#ExtraTreesClassifier to select features based on importance of features in the dataset

clf = ExtraTreesClassifier(n_estimators=250,
                           random_state=SEED)
clf.fit(x , y)
# ##############################################################################
# Plot feature importance
feature_importance = clf.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, df.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```

✓ 0.2s Python



Appendix - Figure 28 – Feature Selection: Check the most relevant features in the dataset for the models

```
df_selected = df_minmax[[ 'Glucose','BMI','Age','DiabetesPedigreeFunction','Outcome']]
```

✓ 0.2s Python

Appendix - Figure 29 - Feature Selection: Extract the most relevant features in the dataset for the models

```
# Train Test Split of the data
from sklearn.model_selection import train_test_split
x = df_selected[['Glucose','BMI','Age','DiabetesPedigreeFunction',]]
y = df_selected['Outcome']
X_train, X_test, y_train, y_test =train_test_split(x,y,test_size=0.25,random_state=0,stratify=df_copy['Outcome'])
```

✓ 0.3s

Appendix - Figure 30 - Dataset split

```
def GetBasedModel():
    basedModels = []
    basedModels.append(('LR'   , LogisticRegression(max_iter=100)))
    basedModels.append(('LDA'  , LinearDiscriminantAnalysis()))
    basedModels.append(('KNN'  , KNeighborsClassifier()))
    basedModels.append(('CART' , DecisionTreeClassifier()))
    basedModels.append(('NB'   , GaussianNB()))
    basedModels.append(('SVM'  , SVC(probability=True)))
    basedModels.append(('AB'   , AdaBoostClassifier()))
    basedModels.append(('GBM'  , GradientBoostingClassifier()))
    basedModels.append(('RF'   , RandomForestClassifier()))
    basedModels.append(('ET'   , ExtraTreesClassifier()))
    basedModels.append(('XGB'  , XGBClassifier()))


    return basedModels
```

✓ 0.3s

Appendix - Figure 31 – Classifiers used in this model

10-Fold Cross Validation for larger test and more trustworthy result

```
def BasedLine2(X_train, y_train,models):
    # Test options and evaluation metric
    num_folds = 10
    scoring = 'accuracy'

    results = []
    names = []
    data_pred = []
    for name, model in models:
        kfold = StratifiedKFold(n_splits=num_folds,shuffle=True, random_state=SEED)
        cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
        y_pred = cross_val_predict(model, X_train, y_train, cv=kfold)
        data_pred.append(y_pred)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

    return names, results, data_pred
```

✓ 0.3s

```
class PlotBoxR(object):

    def __Trace(self,nameOfFeature,value):

        trace = go.Box(
            y=value,
            name = nameOfFeature,
            marker = dict(
                color = 'rgb(0, 128, 128)',
            )
        )
        return trace

    def PlotResult(self,names,results):

        data = []

        for i in range(len(names)):
            data.append(self.__Trace(names[i],results[i]))


        py.iplot(data)
```

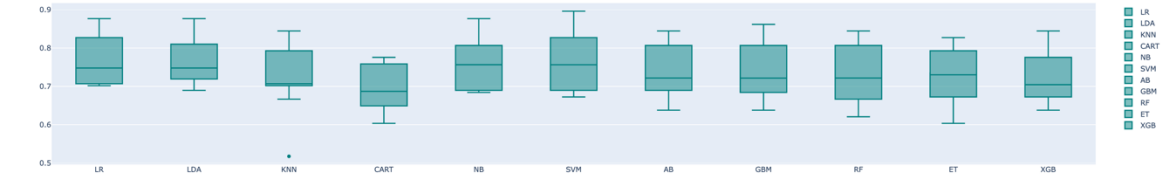
✓ 0.3s

Appendix - Figure 32 – 10-Fold Cross Validation

```
models = GetBasedModel()
names,results,data_pred = BasedLine2(X_train, y_train,models)
PlotBoxR().PlotResult(names,results)
```

✓ 7.1s

```
LR: 0.764035 (0.060710)
LDA: 0.764065 (0.058034)
KNN: 0.718784 (0.087651)
CART: 0.692650 (0.061552)
NB: 0.758063 (0.067307)
SVM: 0.769147 (0.072132)
AB: 0.736207 (0.063703)
GBM: 0.741319 (0.075488)
RF: 0.730853 (0.075273)
ET: 0.729310 (0.070007)
XGB: 0.723956 (0.065987)
```



Appendix - Figure 33 - Classifiers general results

```python
def evaluation(names,models,X_train,y_train,X_test,y_test):
    models = GetBasedModel()
    for name, model in models:
        y_pred = model.fit(X_train, y_train).predict(X_test)
        print('')
        print(f'Model Tested: {name}')
        print('Classification Report:')
        print(classification_report(y_test, y_pred))
        #print(confusion_matrix(y_test, y_pred))
        print(f'Accurary Score: {accuracy_score(y_test, y_pred)}')
        crosstab = pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
        print('')
        print('Confusion Matrix for the Algorithm:')
        print(crosstab)
        print('')
        print('*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*')
    #return y_pred
```

✓ 0.8s Python

```python
evaluation(names,models,X_train,y_train,X_test,y_test)
```

✓ 0.5s Python

```
Model Tested: LR
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.89      0.84       125
         1.0       0.73      0.57      0.64        67

    accuracy                           0.78       192
   macro avg       0.76      0.73      0.74       192
weighted avg       0.77      0.78      0.77       192

Accurary Score: 0.7760416666666666

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        111   14  125
1.0         29   38   67
All        140   52  192

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Model Tested: LDA
Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.87      0.84       125
         1.0       0.71      0.60      0.65        67

    accuracy                           0.78       192
   macro avg       0.76      0.73      0.74       192
weighted avg       0.77      0.78      0.77       192

Accurary Score: 0.7760416666666666

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        109   16  125
1.0         27   40   67
All        136   56  192

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Model Tested: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.76      0.78       125
         1.0       0.58      0.63      0.60        67

    accuracy                           0.71       192
   macro avg       0.69      0.69      0.69       192
weighted avg       0.72      0.71      0.72       192

Accurary Score: 0.7135416666666666

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0         95   30  125
1.0         25   42   67
All        120   72  192

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Model Tested: CART
Classification Report:
              precision    recall  f1-score   support

         0.0       0.74      0.79      0.77       125
         1.0       0.56      0.49      0.52        67

    accuracy                           0.69       192
   macro avg       0.65      0.64      0.65       192
weighted avg       0.68      0.69      0.68       192

Accurary Score: 0.6875

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0         99   26  125
1.0         34   33   67
All        133   59  192

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Model Tested: NB
Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.86      0.83       125
         1.0       0.71      0.61      0.66        67

    accuracy                           0.78       192
   macro avg       0.76      0.74      0.74       192
weighted avg       0.77      0.78      0.77       192

Accurary Score: 0.7760416666666666

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        108   17  125
1.0         26   41   67
All        134   58  192

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Model Tested: SVM
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.86      0.82       125
         1.0       0.68      0.57      0.62        67

    accuracy                           0.76       192
   macro avg       0.73      0.71      0.72       192
weighted avg       0.75      0.76      0.75       192

Accurary Score: 0.7552083333333334

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        107   18  125
1.0         29   38   67
All        136   56  192
```

```
Model Tested: AB
Classification Report:
              precision    recall  f1-score   support

         0.0       0.82      0.85      0.83       125
         1.0       0.70      0.66      0.68        67

    accuracy                           0.78       192
   macro avg       0.76      0.75      0.76       192
weighted avg       0.78      0.78      0.78       192

Accurary Score: 0.78125

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        106   19  125
1.0         23   44   67
All        129   63  192

*-----------------------------------------------------*

Model Tested: GBM
Classification Report:
              precision    recall  f1-score   support

         0.0       0.82      0.86      0.84       125
         1.0       0.72      0.64      0.68        67

    accuracy                           0.79       192
   macro avg       0.77      0.75      0.76       192
weighted avg       0.78      0.79      0.78       192

Accurary Score: 0.7864583333333334

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        108   17  125
1.0         24   43   67
All        132   60  192

*-----------------------------------------------------*

Model Tested: RF
Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.84      0.83       125
         1.0       0.68      0.64      0.66        67

    accuracy                           0.77       192
   macro avg       0.75      0.74      0.74       192
weighted avg       0.77      0.77      0.77       192

Accurary Score: 0.7708333333333334

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        105   20  125
1.0         24   43   67
All        129   63  192
```

```
Model Tested: ET
Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.82      0.81       125
         1.0       0.66      0.63      0.64        67

    accuracy                           0.76       192
   macro avg       0.73      0.73      0.73       192
weighted avg       0.75      0.76      0.75       192

Accurary Score: 0.7552083333333334

Confusion Matrix for the Algorithm:
Predicted  0.0  1.0  All
True
0.0        103   22  125
1.0         25   42   67
All        128   64  192

*-----------------------------------------------------*

Model Tested: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0       0.78      0.82      0.80       125
         1.0       0.62      0.57      0.59        67

    accuracy                           0.73       192
   macro avg       0.70      0.69      0.70       192
weighted avg       0.72      0.73      0.73       192

Accurary Score: 0.7291666666666666

Confusion Matrix for the Algorithm:
Predicted    0   1  All
True
0.0        102  23  125
1.0         29  38   67
All        131  61  192

*-----------------------------------------------------*
```

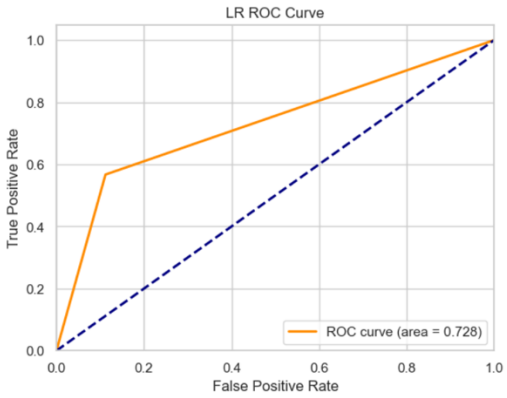
Appendix - Figure 34 – Evaluation results, for precision, recall, F1-score, and support for each classifier model

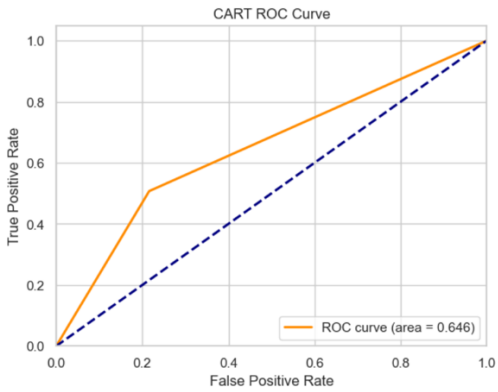
```python
#ROC Curve
def ROC_Curve(names,models,X_train,y_train,X_test,y_test):
    models = GetBasedModel()
    for name, model in models:
        y_pred = model.fit(X_train, y_train).predict(X_test)
        fpr, tpr, thresholds = roc_curve(y_test, y_pred)
        roc_auc = auc(fpr, tpr)
        plt.figure()
        plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.3f)' % roc_auc)
        plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'{name} ROC Curve')
        plt.legend(loc="lower right")
        plt.show()
```

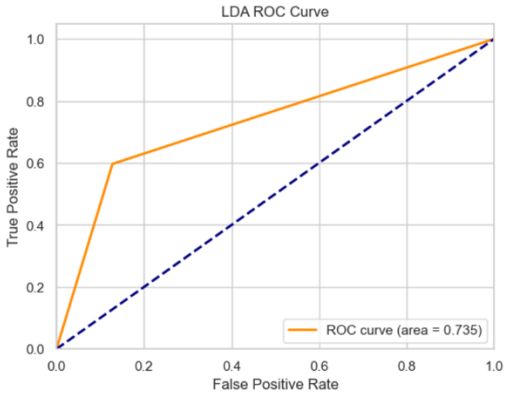
✓ 0.3s Python

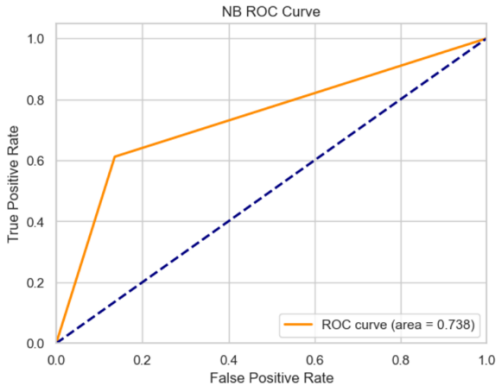
```python
ROC_Curve(names,models,X_train,y_train,X_test,y_test)
```

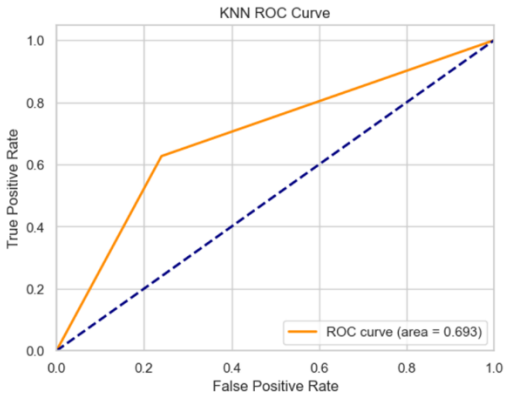
✓ 1.2s Python

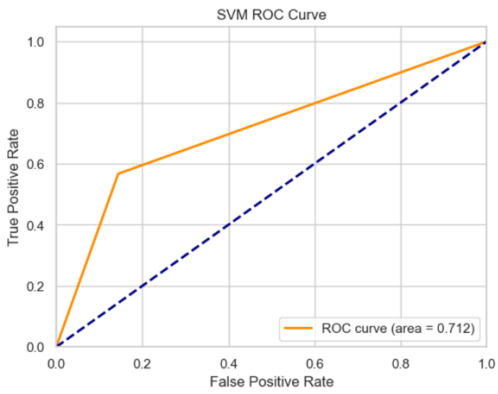

LR ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.728)

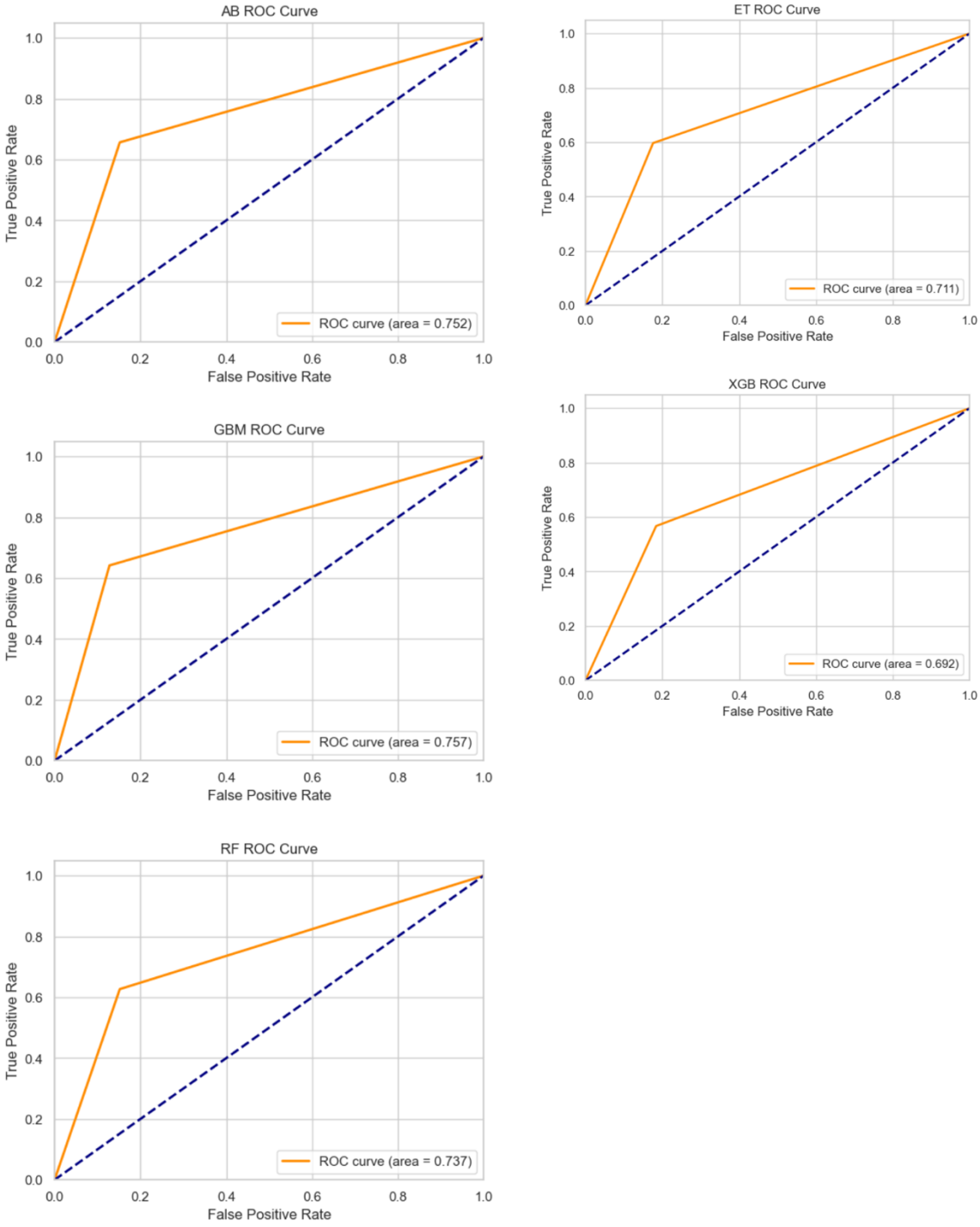

CART ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.646)


LDA ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.735)


NB ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.738)


KNN ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.693)


SVM ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.712)



Appendix - Figure 35 – ROC Curve for each classifier model