Title:

**2nd Homework (Machine Learning)**

Course:

**Artificial Intelligence**

Author:

**Hatef Mohammadi Alashti**

810601123

Course Instructor:

**Dr. Shariat Panahi**

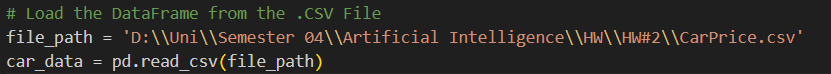
**May 2024**

# Regression Methods

The goal of this section is to develop and train a model that can predict a car's price using 24 numerical or categorical features.

## Studying Raw Data

The first step is to read the CSV file.

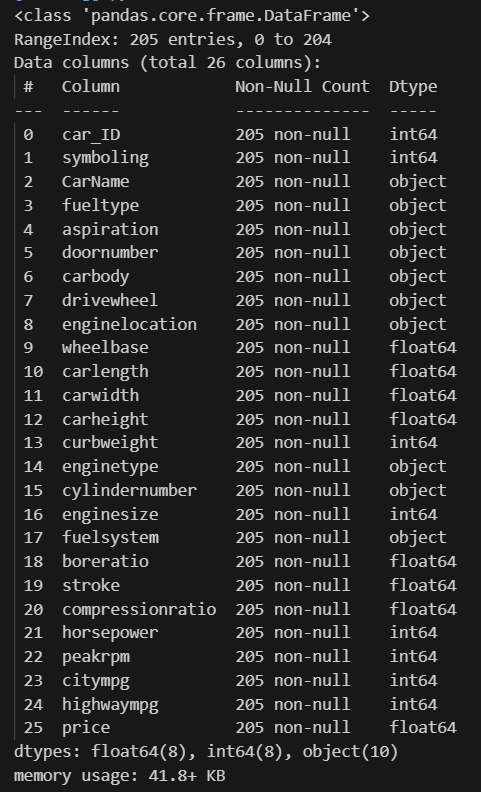


## A-1) Data Framework

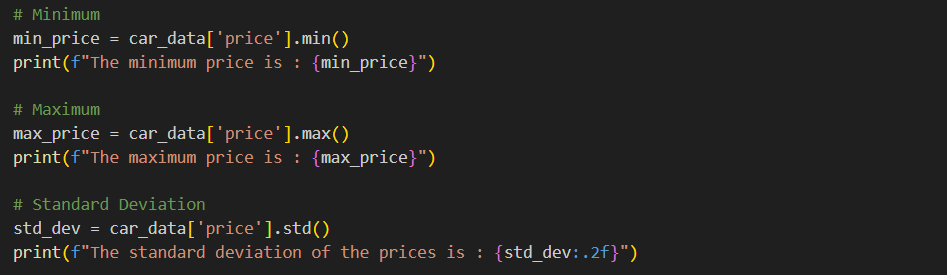
To obtain an overview of the data structure, the “info” function was used.



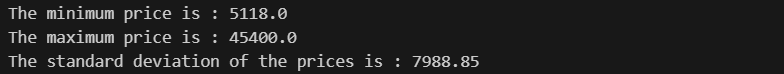
Output: We can identify numerical (int64 and float64), and categorical (object) features.



## A-2) Statistical Analysis (Min, Max, Std Dev)

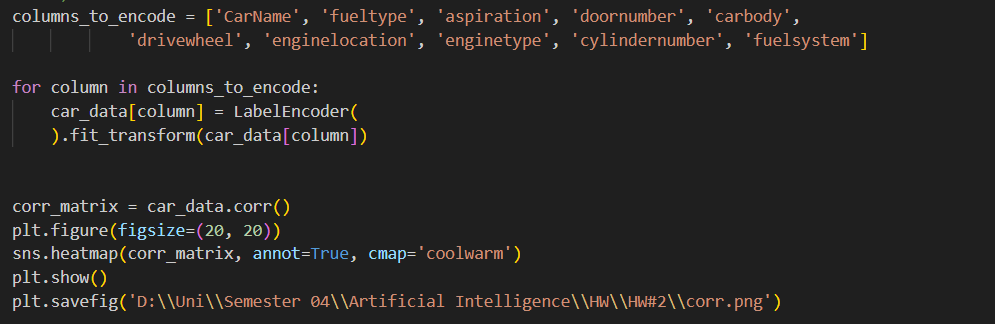


Output:



## A-3) Correlation Matrix

At first, we assign numbers to categorical data to have a homogeneous data frame.

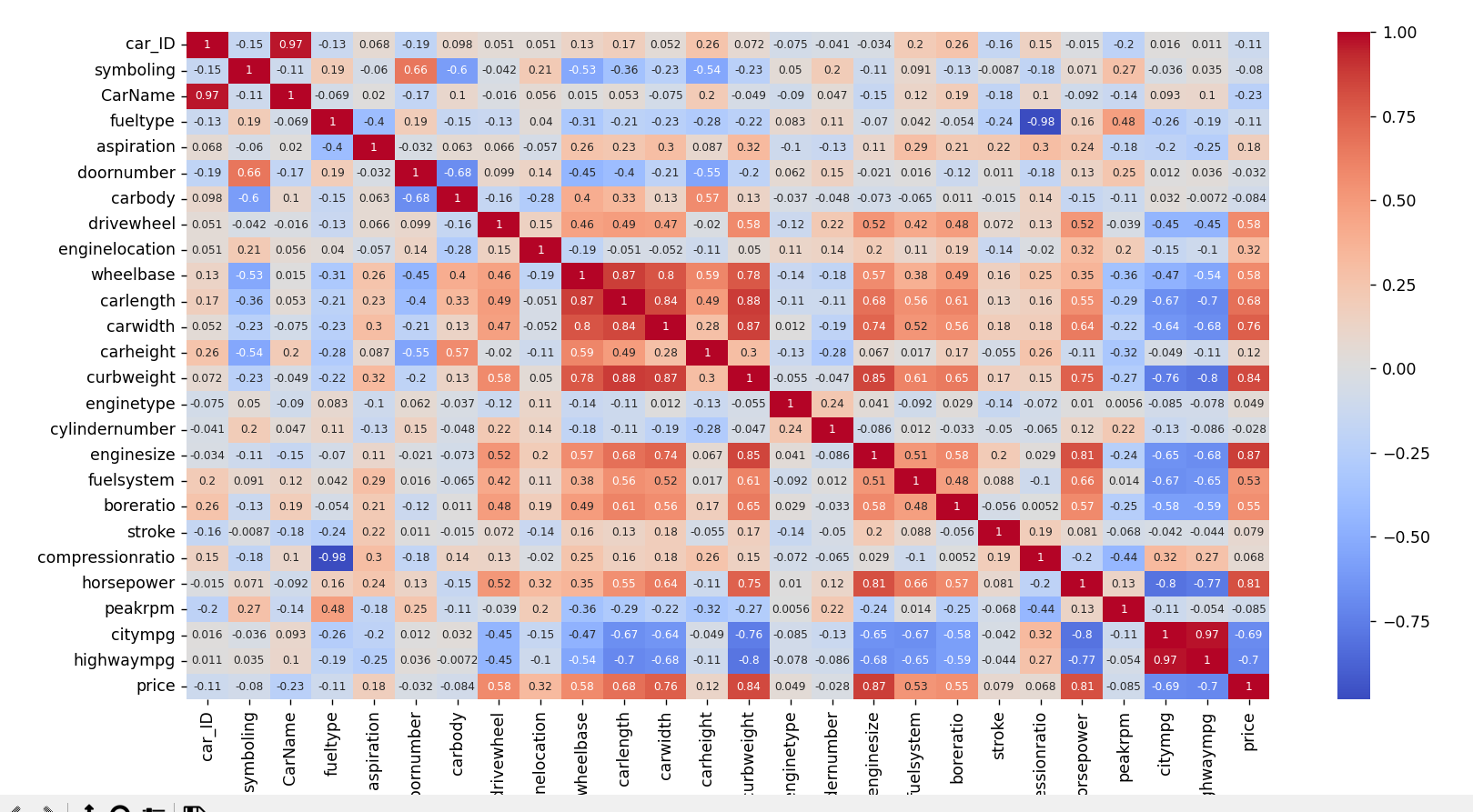


Output:

From the figure below, the most important features (to the price) can be selected:

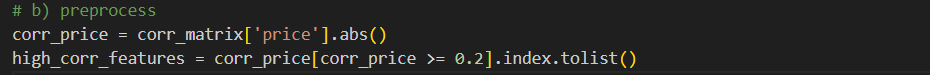
Enginsize (0.87), curbweight (0.84), horsepower (0.81), carwidth (0.76), highwaympg (0.70), citympg (0.69), and carlength (0.68).

Also, it can be seen that the correlation factors between highwaympg and citympg, and between fueltype and compressionratio are 0.97, and -0.98 respectively. This means that they are strongly colinear, so it is justifiable to keep only one of each pair.

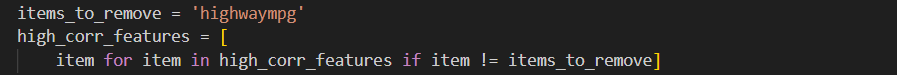


## Preprocess

In this section, we aim to neglect as much data as possible to reduce the calculation load. Firstly, we only keep features that have correlation factors of over 0.2 to the price.



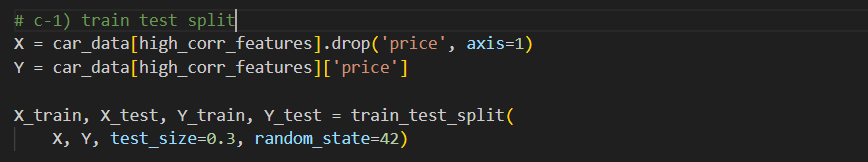
Then, for pairs that have high colinearity, we only keep one of each pair. We only have to remove one from highwaympg and citympg, as the other pair (fueltype and compressionration) did not make the list to this point due to having a correlation factor lower than 0.2.



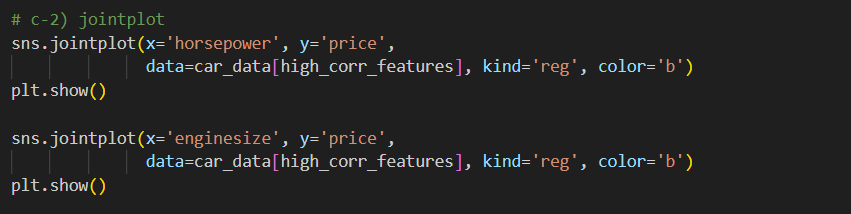
## Selection, Training, and Evaluation

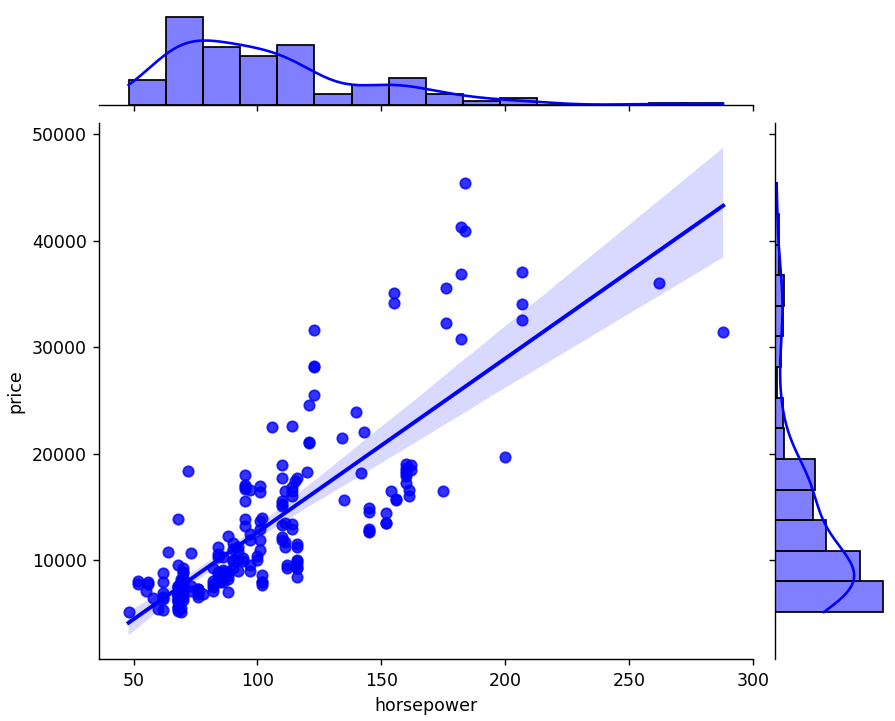
## C-1) Train and Test Split

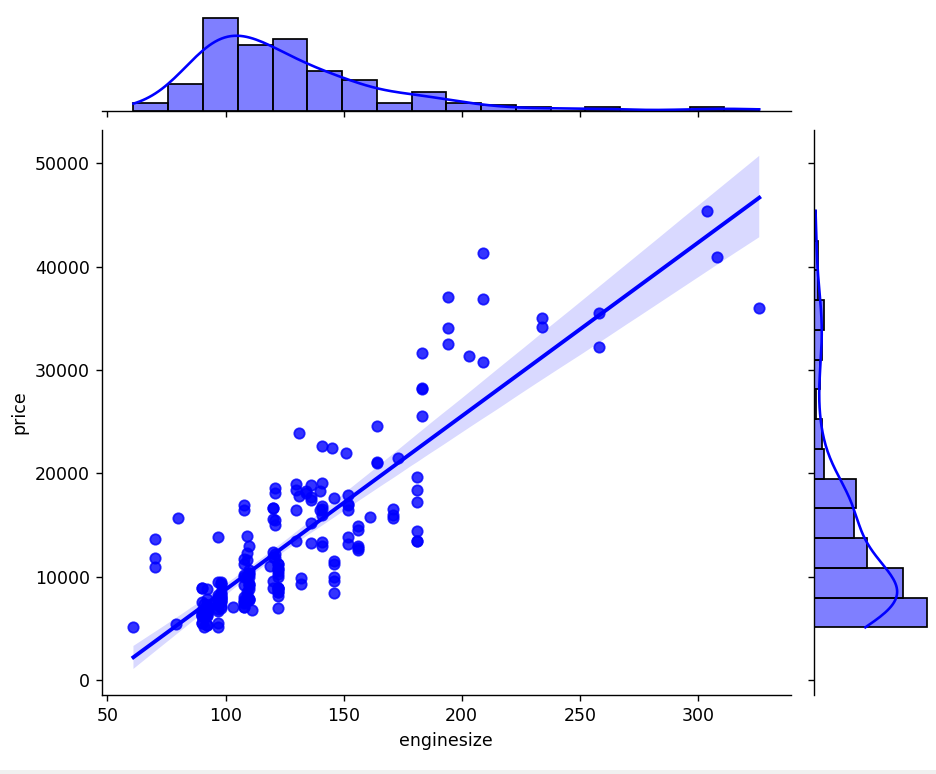
We have been asked to split the data frame into test (30%) and train (70%) sets. To do this, first, we only keep the features (we exclude price). Then, from the scikit-learn module, we implement the train\_test\_split function with test size = 0.3 and random\_state = 42.



## C-2) Jointplot (Horsepower and Enginesize)

In this stage, we are inclined to ensure whether horsepower and enginesize are suitable to be implemented in a linear regression model. To do so, we are asked to use jointplot finction from the seaborn module. 

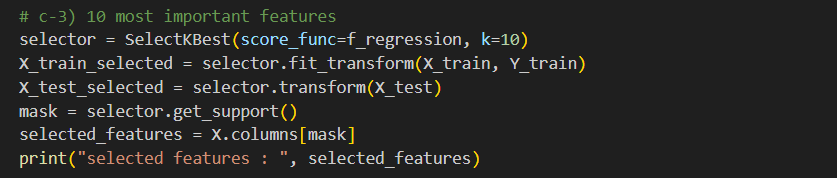




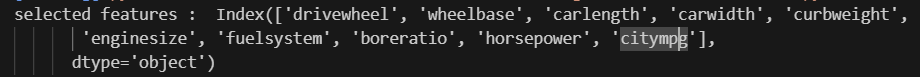
It can easily be deduced that both these features have a fair linearity with the price. Therefore, they have no problem to be taken into a linear regression model.

## C-3) Selection of Best Features

Now, we select 10 of the most impactful features on the price by using the SelectKBest function from the sci-kit-learn module. We set the score function to be f\_regression.

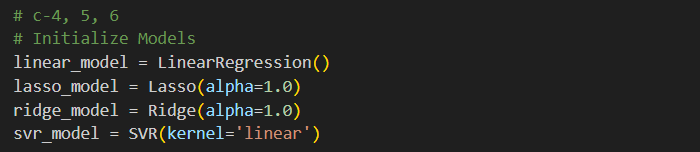


Output:

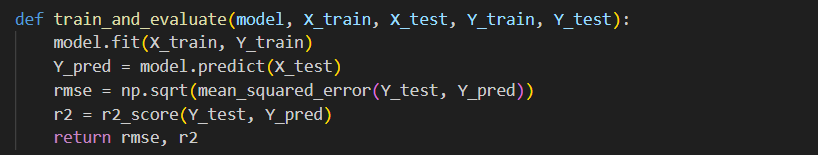


## C-4, C-5, C-6) Train and Evaluate

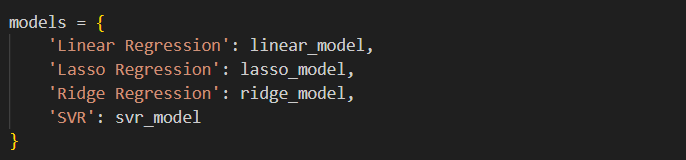
This is the section in which we train the four models in question. First, we need to initialize the models. We set the alpha parameter to be one for both the Lasso and the Ridge models. Also, we set the kernel parameter to linear for the Support Vector Regression (SVR) model.



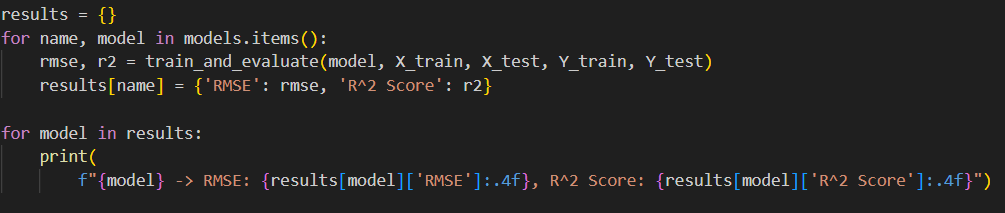
Then, we define a function that takes the model and the data sets as the input. Next, it fits the model according to the train and test sets. In addition, the evaluation of the model (RMSE and score) is calculated.



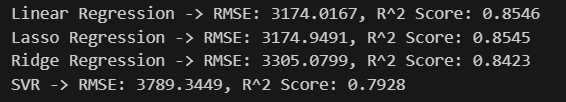
Resultantly, we set a dictionary of the models to be implemented in a for loop to be trained and evaluated.



For the last section, in a for loop that runs through the aforementioned dictionary of the models, we train and evaluate them. Ultimately, we print the RMSE and score of the models.



Output:

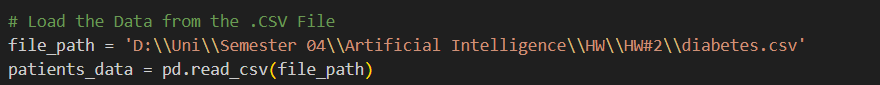


# Classification

The second half of the homework is designated to develop and train a model to predict whether one has diabetes based on 8 features.

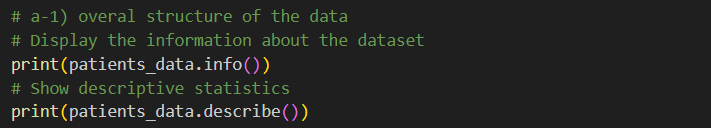
## A) Studying Raw Data

The first step is to read the spreadsheet file.

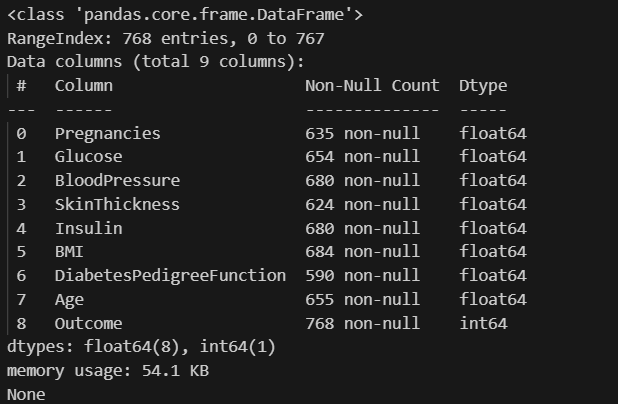


## A-1) Data Framework

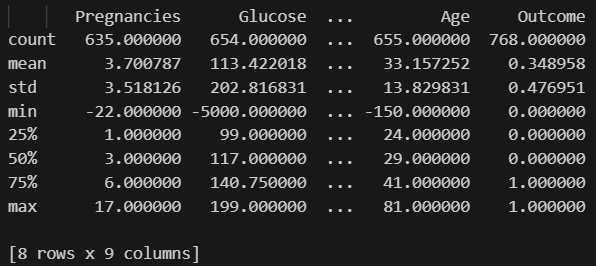
To get an overview of the data frame, we use info and describe functions.



Info function’s Output:

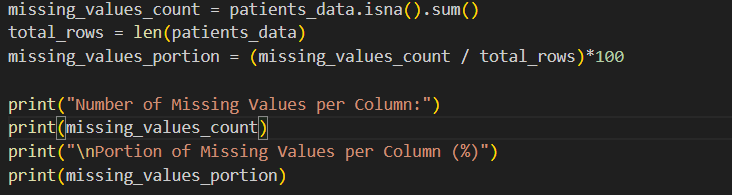


Describe Function’s Output:

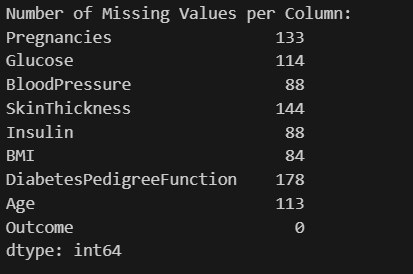
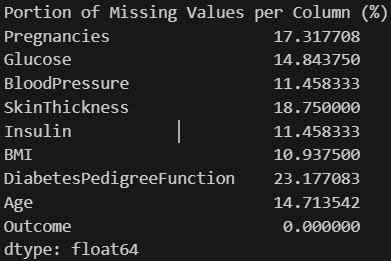


## A-2) Missing Values

We used the sum function to calculate the number of empty cells in each row.

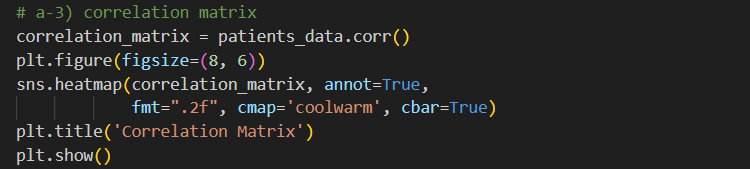


Output:

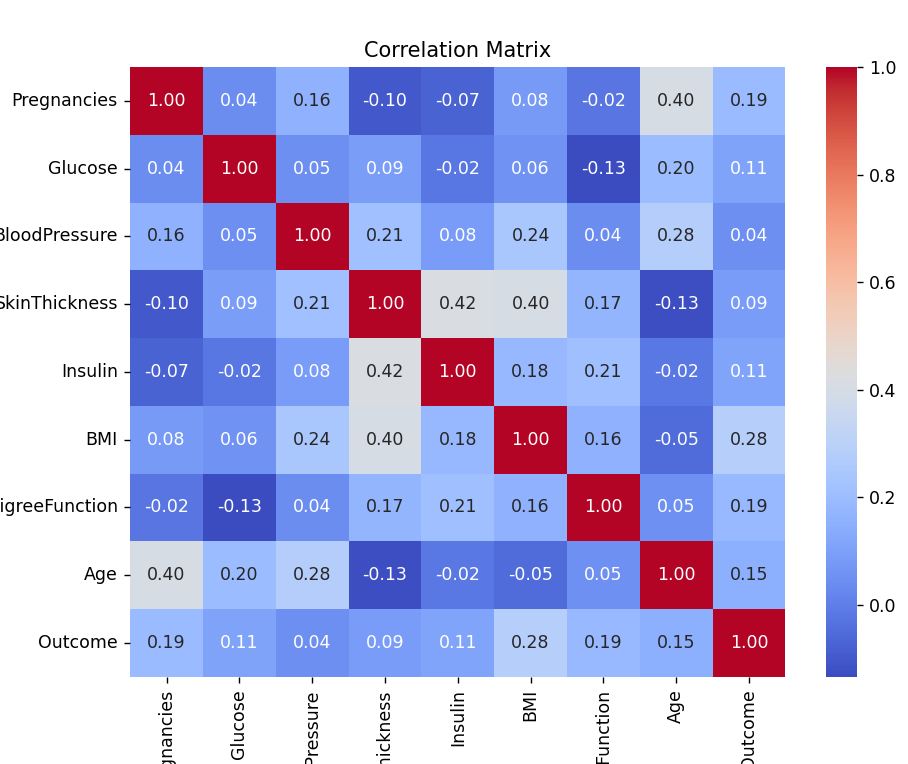
 

## A-3) Correlation Matrix

Then, we calculated the correlation factor and showed its figure.



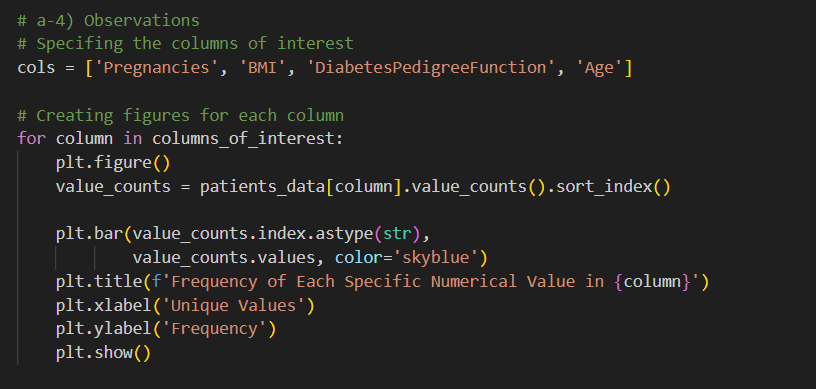
Output:



It is apparent that BMI (0.28), DiabetesPedigreeFunction (0.19), number of pregnancies (0.19), and Age (0.15) are of the most importance among other features.

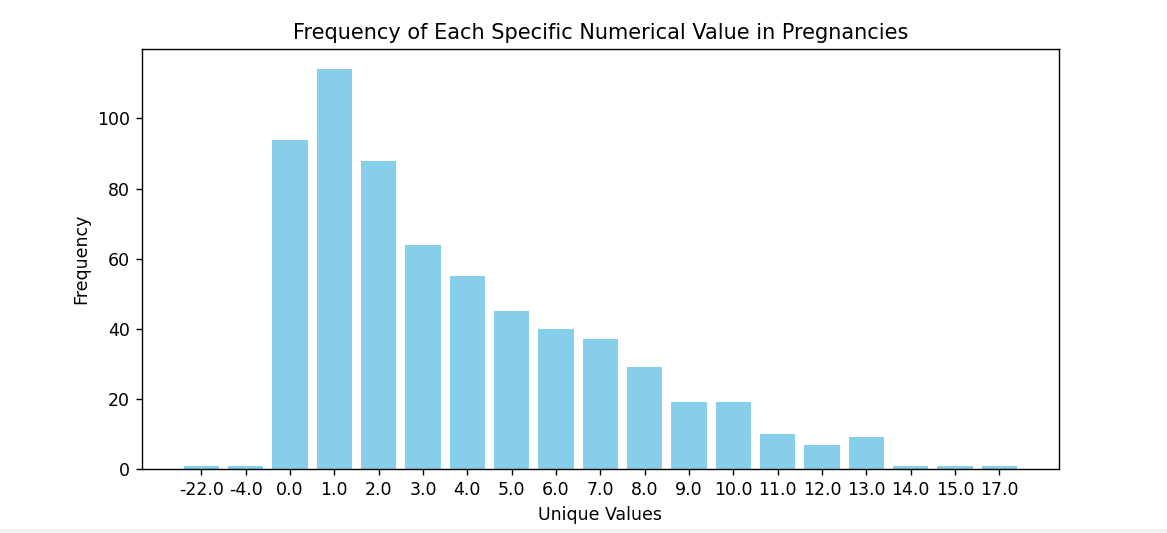
## A-4) Number of Observations

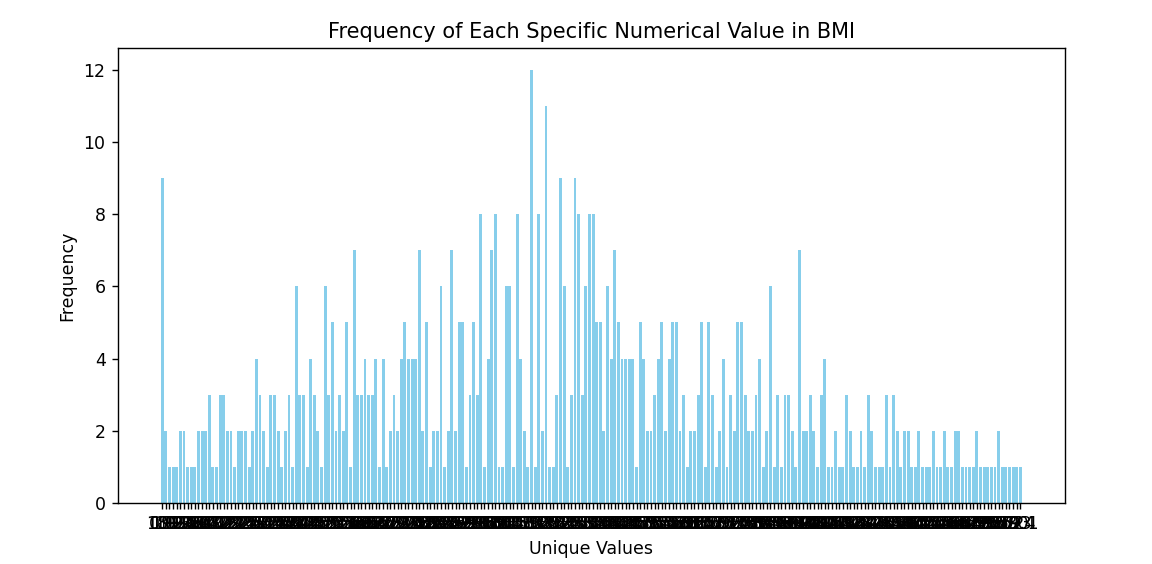
In question, it is stated to show the number of observations per each specific numerical value for four columns with the most importance. Since we need to obtain the number of frequencies for each unique number, the histogram of the data frame would not be of any help. Therefore, we use the value\_counts function instead.

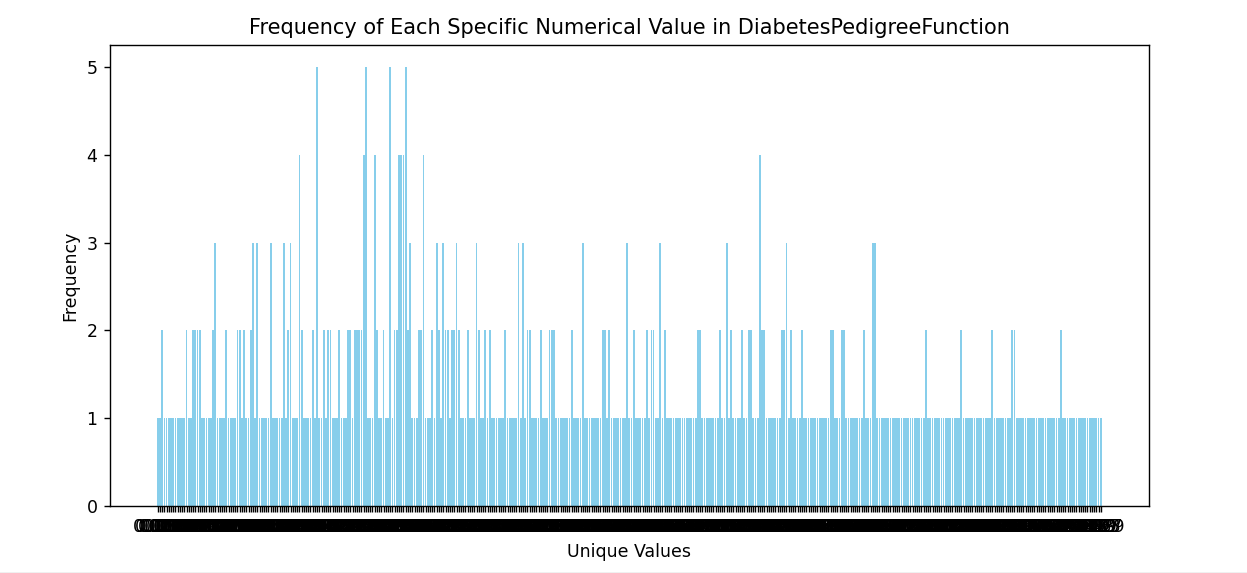


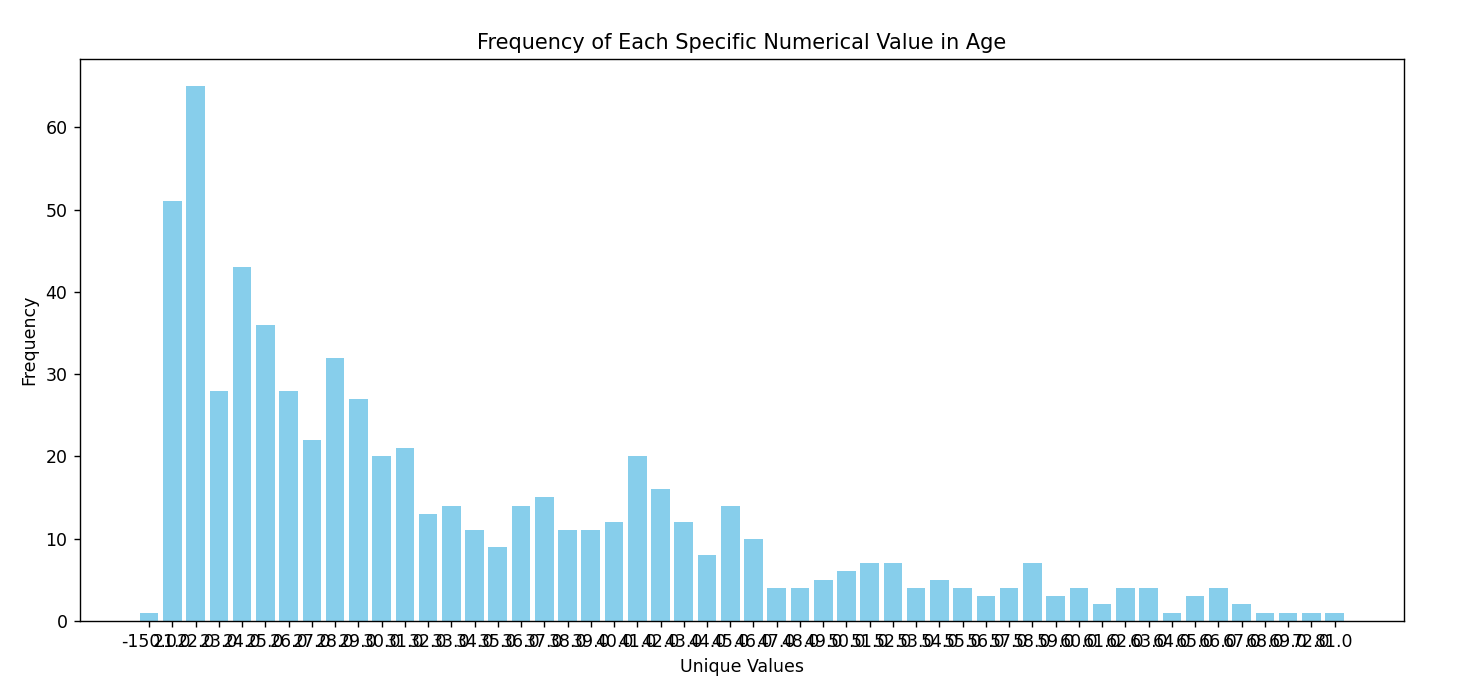
Since we wanted to obtain the number of repetitions for each number (not numbers in a specific range as in histogram), and there are numerous different values in the data frame, the x label of the plots may be not feasibly readable. This issue would not be problematic owing to the accessibility to the list of all the frequencies, in case of need.

Outputs:



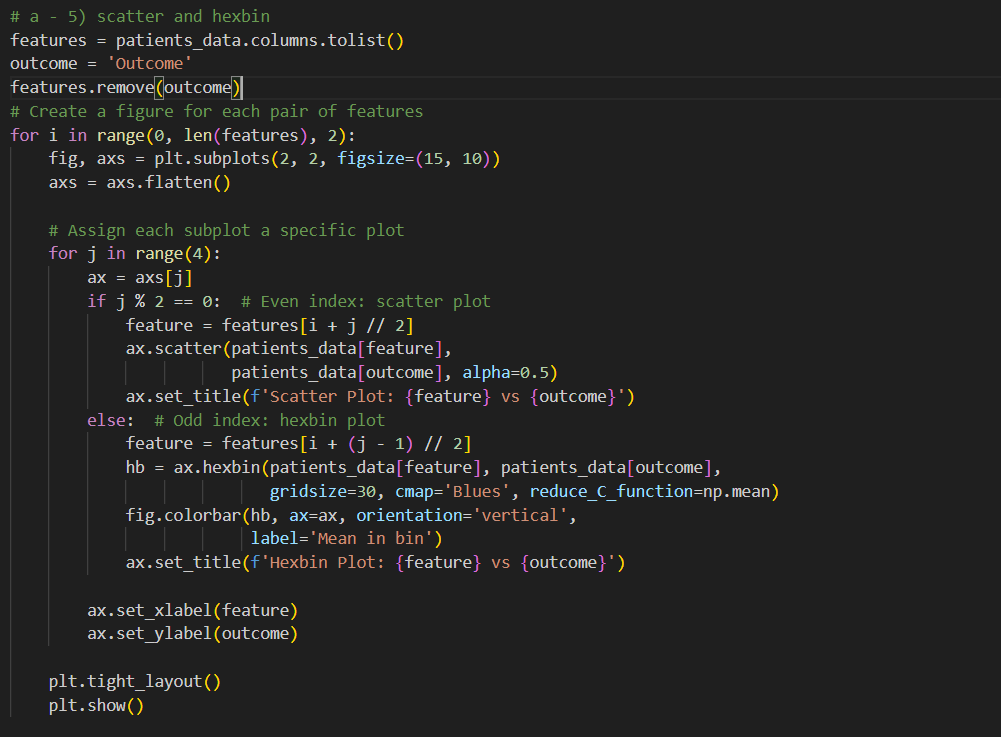




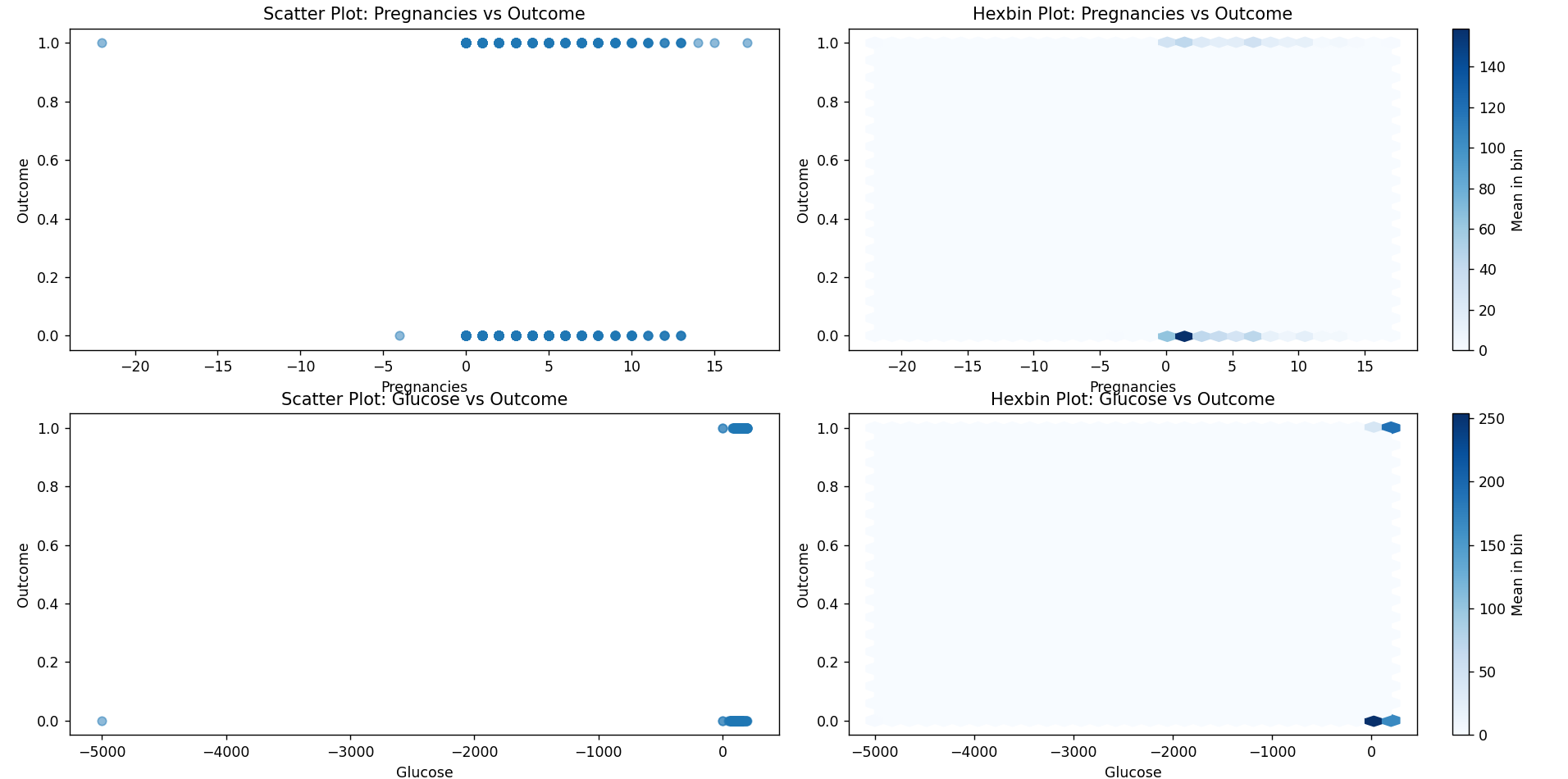


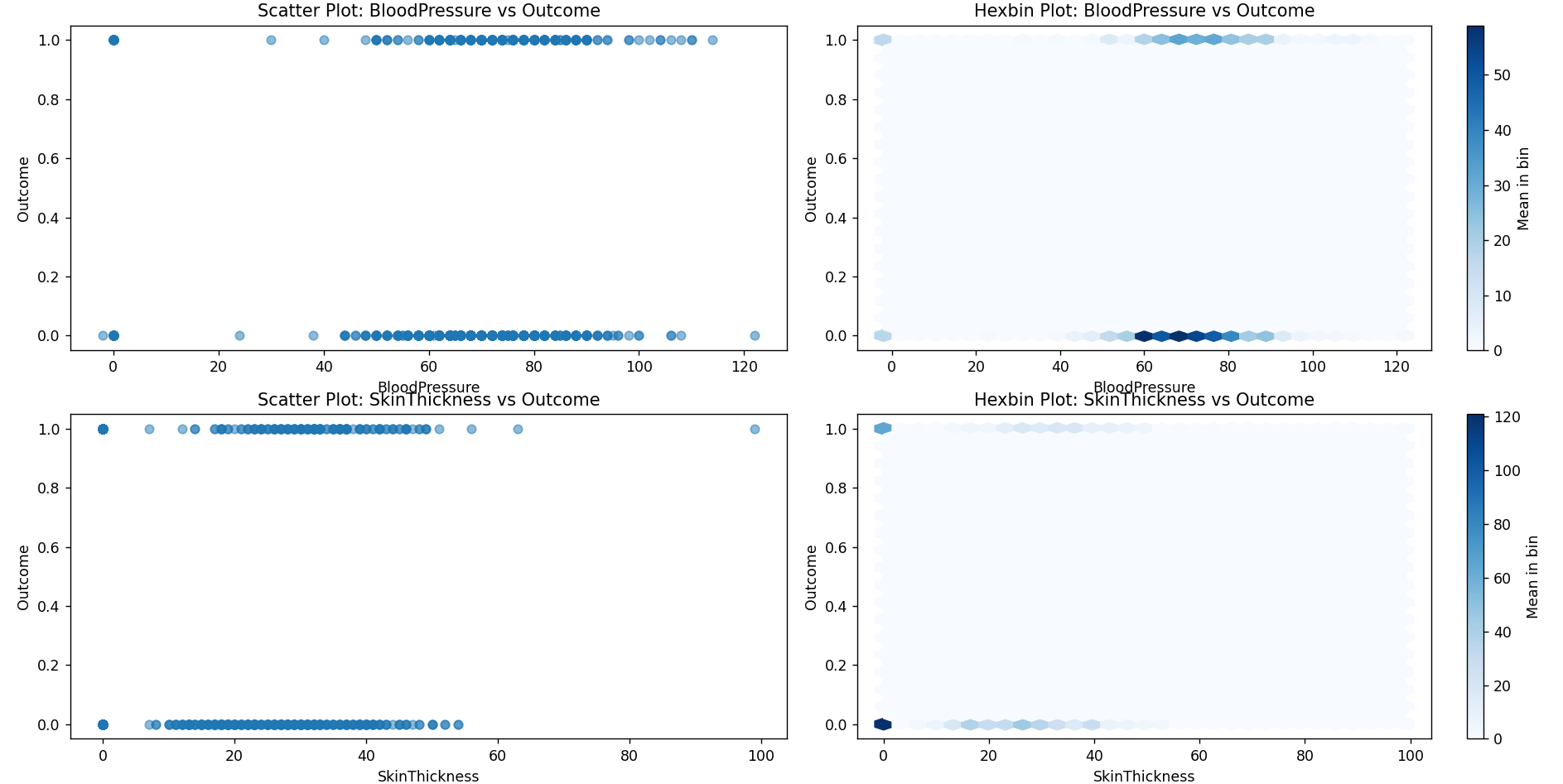
**Note: The objective of this section was rather vague owing to stating “the number of observations per specific values”. We reported the number of frequencies for each different value in the data frame. If the objective was to show the histogram figures of the selected features, they are exhibited in section B-2.**

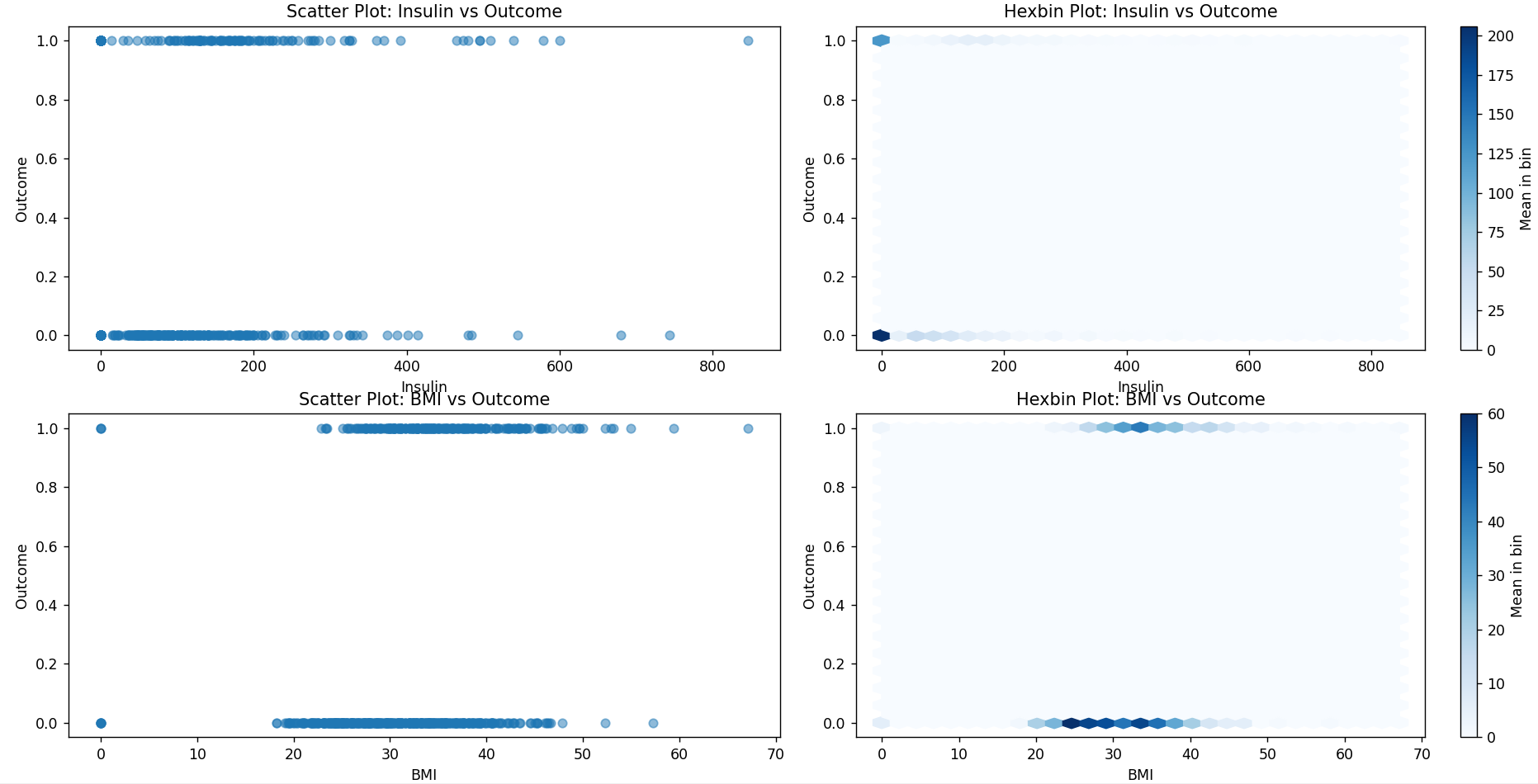
## A-5) Scatter and Hexbin Figures

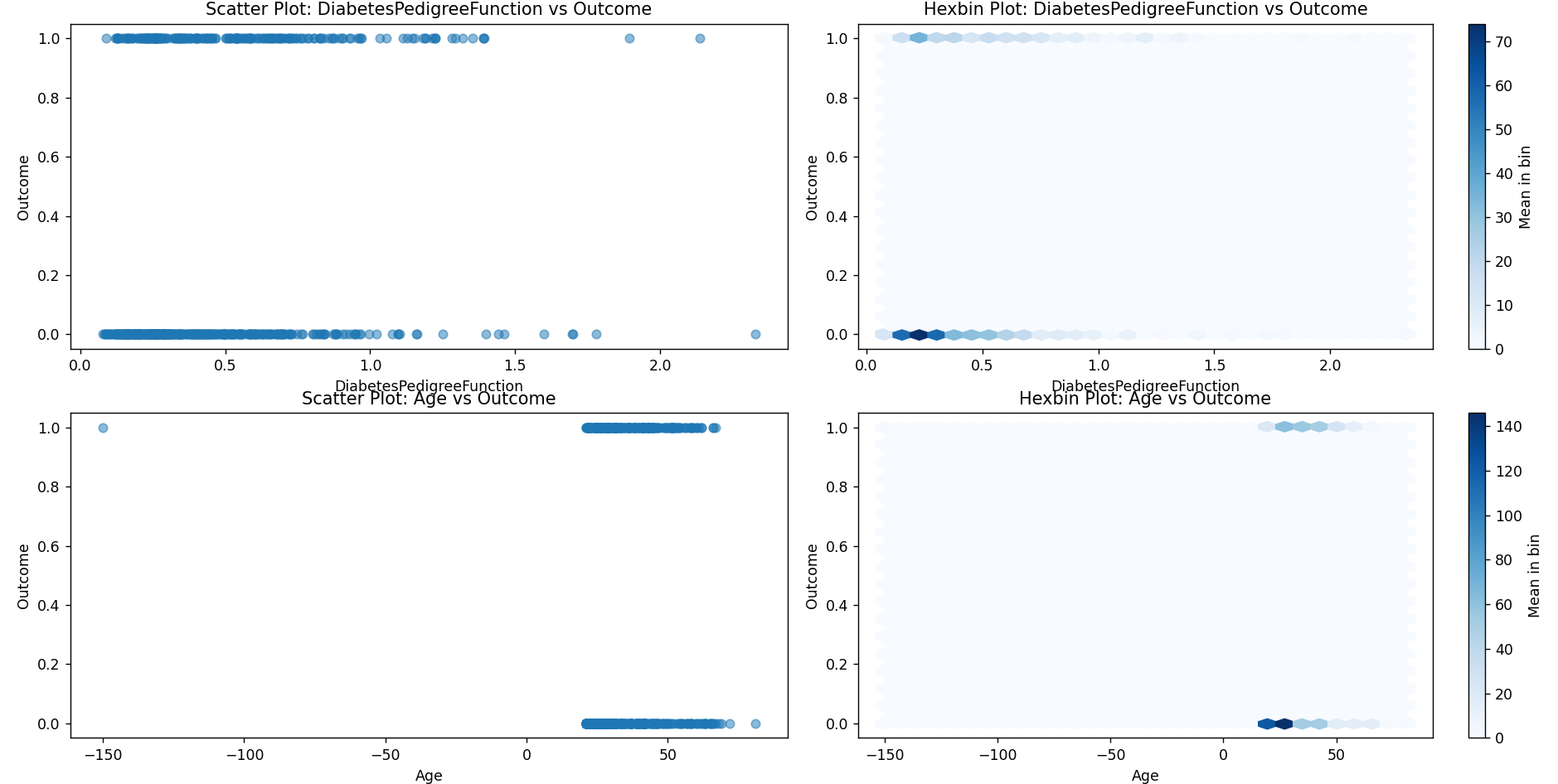


Outcome:





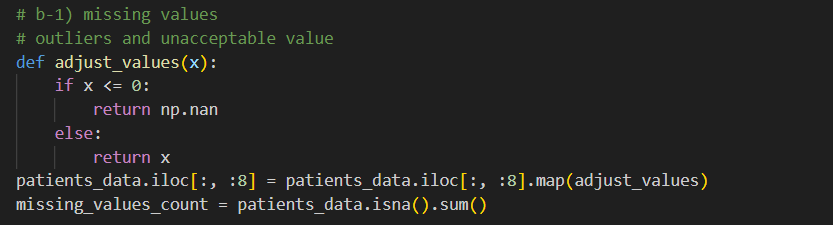




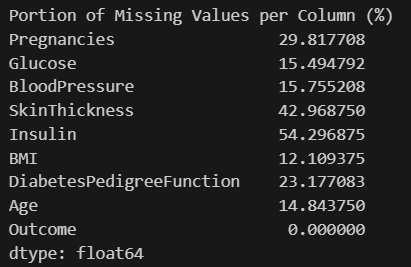
## B) Preprocessing

## B-1) Missing Values

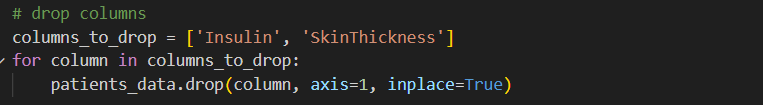
As can be seen from the previous visualizations, some data should be removed. For all columns, negative and zero values cannot be accepted. Hence, we remove them by assigning NaN values to them.



Once again, like before, we calculate the number of missing values.

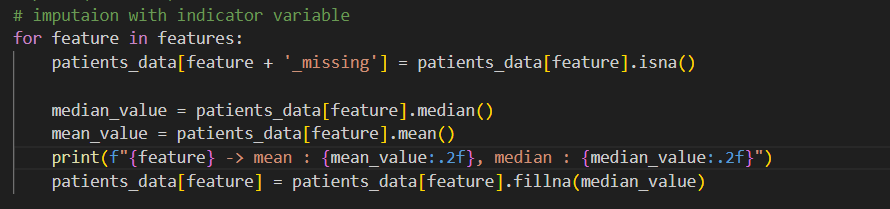


It is safe to say that perhaps, the SkinThickness and Insulin columns cannot help much with training our models due to the high portion of missing values. As a result, we remove them from our data frame.

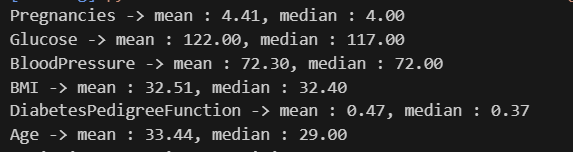


Now, that we have our data frame (containing 6 features) with some missing values, we opt for the imputation method with an indicator variable to mark values that did not exist before.

After, adding new columns for missing values, we have to assign a numerical datum to each empty cell. The two most popular options are the mean and the median of each column. Here we compared these two values.

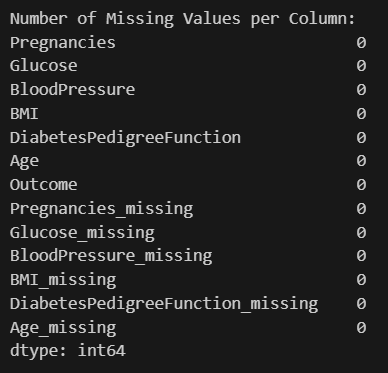


Comparison between median and mean:



Since no meaningful difference between these two statistical measurements was seen, we can be assured that assigning the median (or mean) of each column to its empty values probably would not cause any inconsistencies.

At this stage, to check our work, we counted the number of missing values. As expected, no column has any missing value anymore.



## B-2) Normalizing, and Standardizing

Normalizing and standardizing are two common preprocessing steps to prepare numerical data for analysis, especially when using machine learning algorithms that are sensitive to the scale of input data.

Standardizing, often called z-score normalization, transforms the data to have zero mean and a standard deviation of one. It's computed using the formula:

where 𝑥 is the original value, μ is the mean of the feature, and 𝜎 is the standard deviation. Standardizing does not bind values to a specific range, which might be necessary for some algorithms or methodologies.

Normalizing, often referred to as min-max scaling, rescales the data to a fixed range, typically 0 to 1, using the formula:

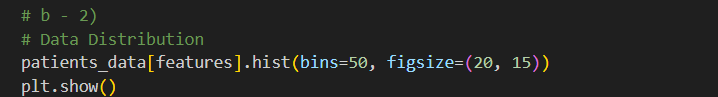
Where and are the minimum and maximum values of the feature, respectively. This transformation is sensitive to outliers because an outlier can compress the majority of the data into a very narrow range in the transformed space.

Standardizing is more commonly used in scenarios where the data needs to be normalized around the mean, removing the bias of different means and variances in the data. It’s beneficial for methods that assume data is normally distributed within each feature and work well even if the data isn't strictly normal.

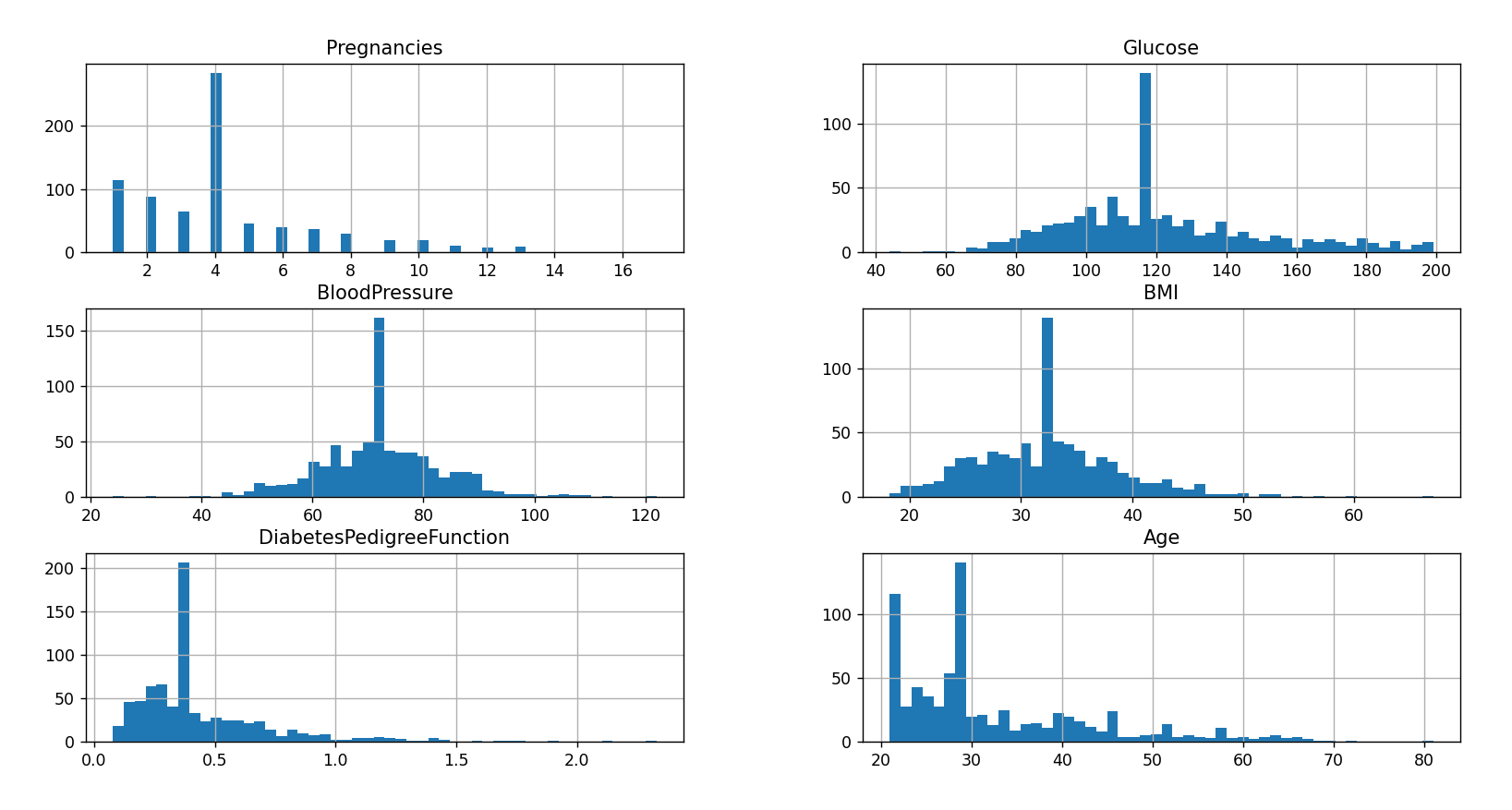
Normalizing is particularly useful when the data needs to be bound within a range, such as with neural networks that use Sigmoid or Tanh activation functions, and for algorithms that compute distances between data points.

To determine whether our data frame needs standardizing, normalizing, or neither, we have to examine the data distribution. If features are on different scales, standardizing can help equalize the influence of each feature. If outliers are a concern and should not disproportionately influence the model, we should consider robust scaling methods (like using median and quantile range instead of mean and standard deviation in standardization).

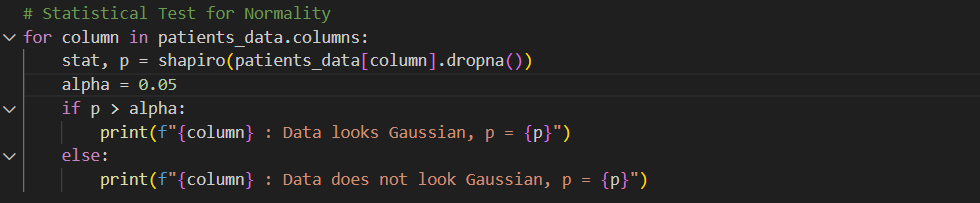
To do so, the first thing to investigate should be the histogram of the data set.



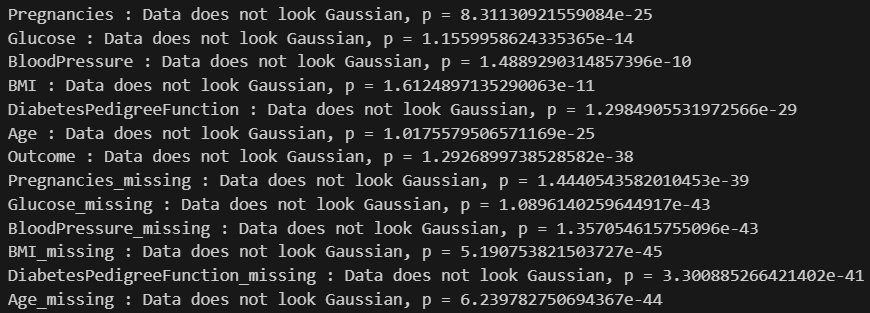
Output:



One of the most popular tests for normality of the data is the Shapiro-Wilk test. This test is used to assess whether a set of data is normally distributed. Also, this test is sensitive to NaN values. We set the significance level threshold at 5%. If the p-value of a data set is greater than the set threshold, we will have a rather Gaussian distribution.

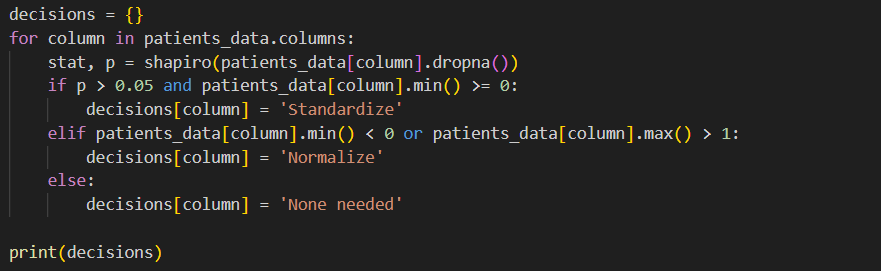


Output:

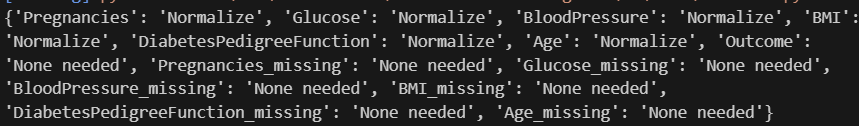


This test alone is not sufficient to deduct if normalizing or standardizing is needed. However, if we look at our results, it can be easily interpreted that the scale of the values of each feature differs significantly. Resultantly, normalizing our data can be a big help.

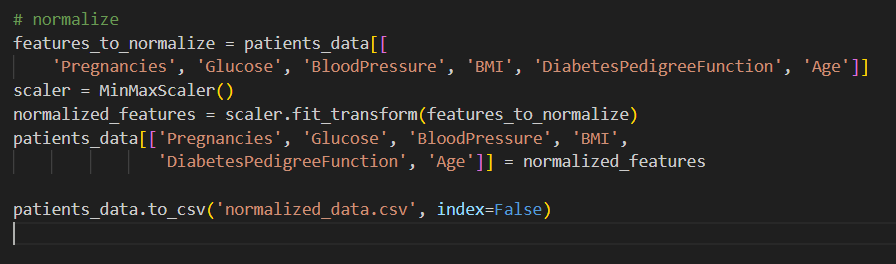
Also, in the code below, we determined which set needs to be normalized. We expect the distribution of the missing indicator columns to need nothing, as they only contain True (1) and False (0).



Output:



Hence, we need to normalize all the features (6 columns) except the indicators.

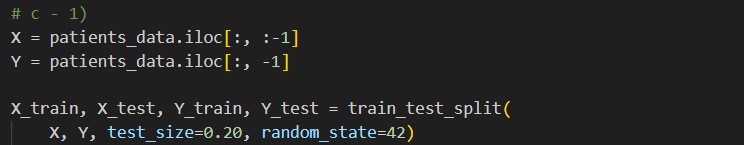


At this stage, our data frame only has values between 0 and 1.

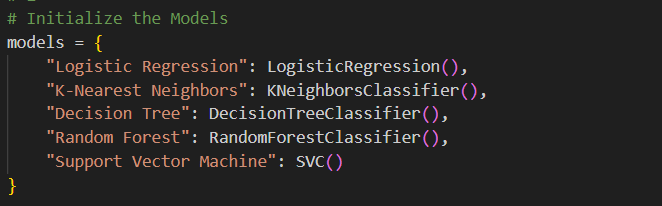
## Selection, Training, and Evaluation

## C-1, C-2) Split and Train

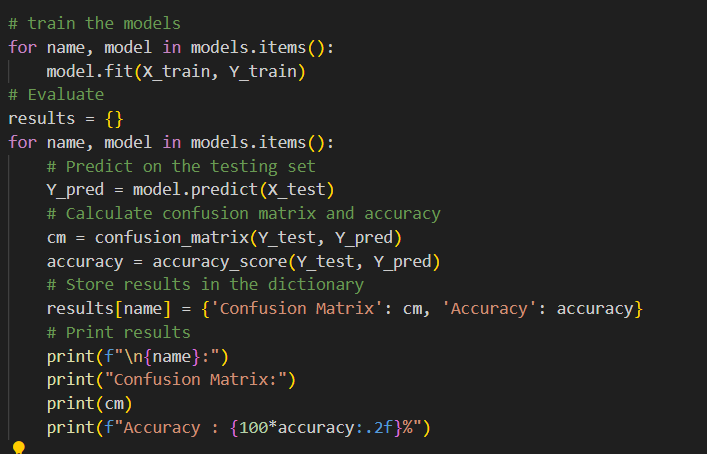
In this section we are asked to randomly split our data frame into a train (80%) and test (20%) set. Then by using the sci-kit-learn module train 5 models: Logistic Regression (LR), K-Nearest-Neighbors (KNN), Decision Tree (DT), Random Forest (RF), and Support Vector Machine (SVM). First, we split the data:



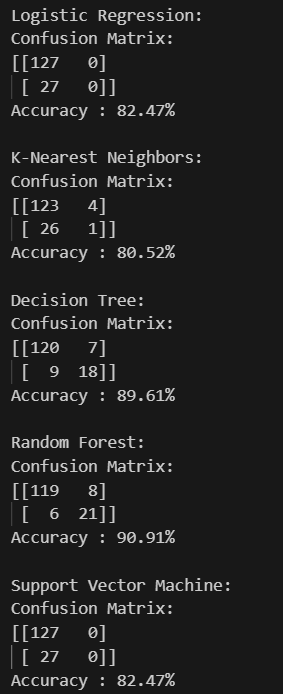
Then we initialize the models into a dictionary to loop through:



Then we train and evaluate the models. Also, we calculate the confusion matrix and accuracy for each model.

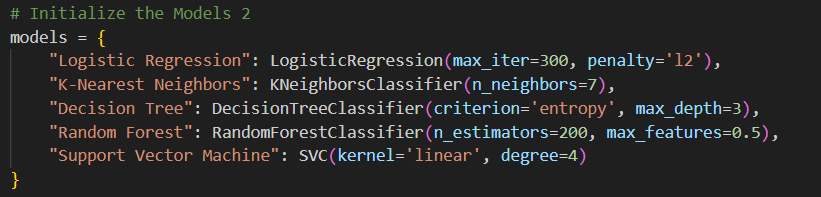


Output:

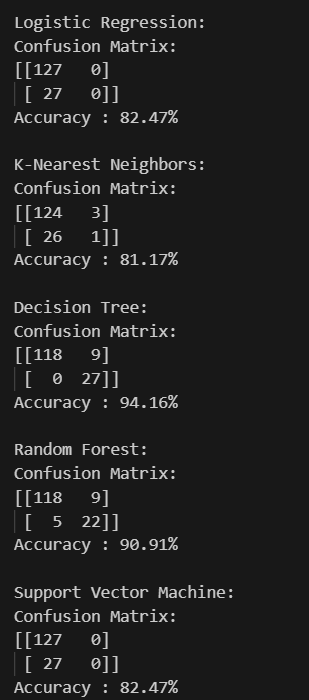


## C-3) Change Hyperparameters and Optimization

First, we changed two parameters of each model, except for the KNN model in which we were asked to only change the number of neighbors. The only difference between this section and the former one is in the initializing part:



The output differed as well, as we expected:

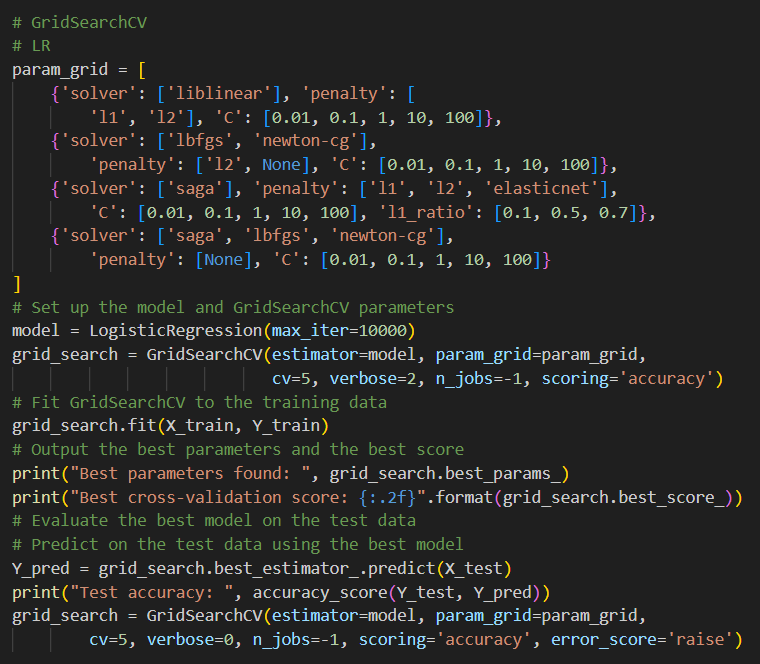


Hence, our models experienced no changes in accuracy for 3 of the models (LR, Rf, and SVM). However, the other two (KNN, and DT) got slightly more accurate.

Now, we go through the process of optimizing.

We have done this section for each model separately.

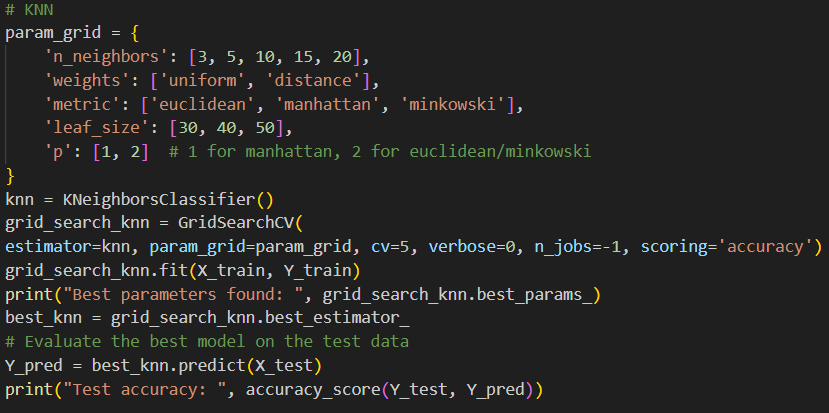
Linear Regression:



Output:



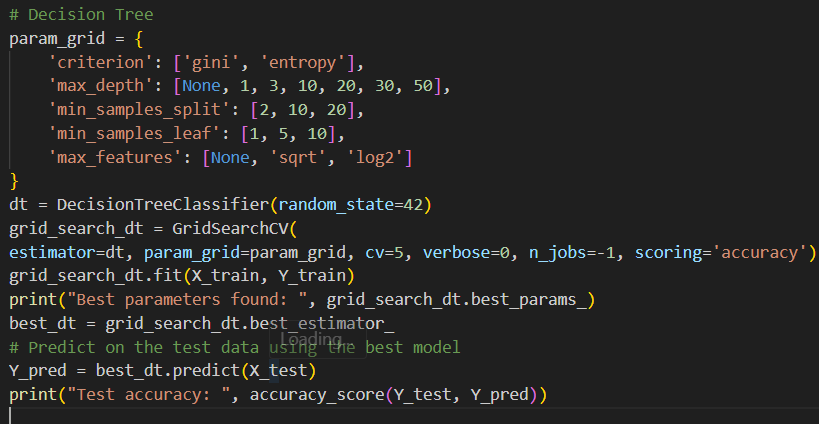
K-Nearest-Neighbor:



Output:



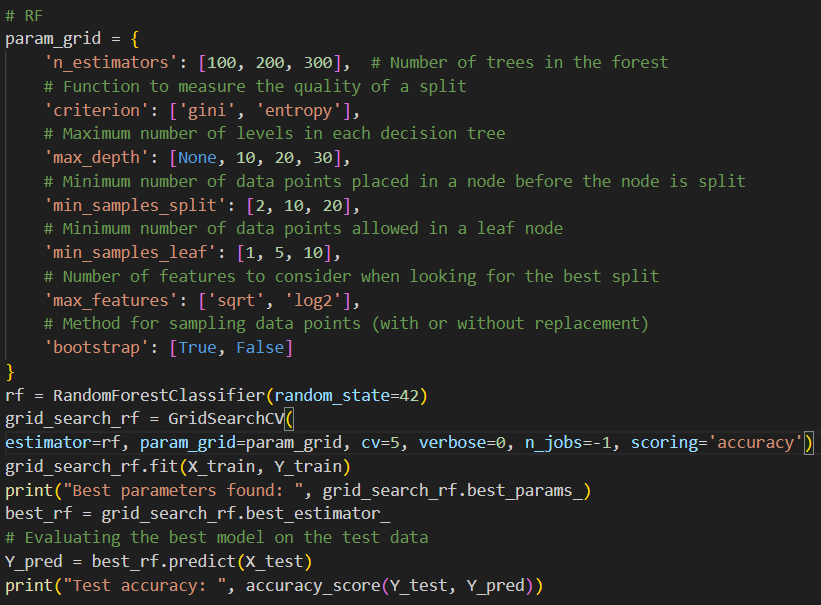
Decision Tree:



Output:



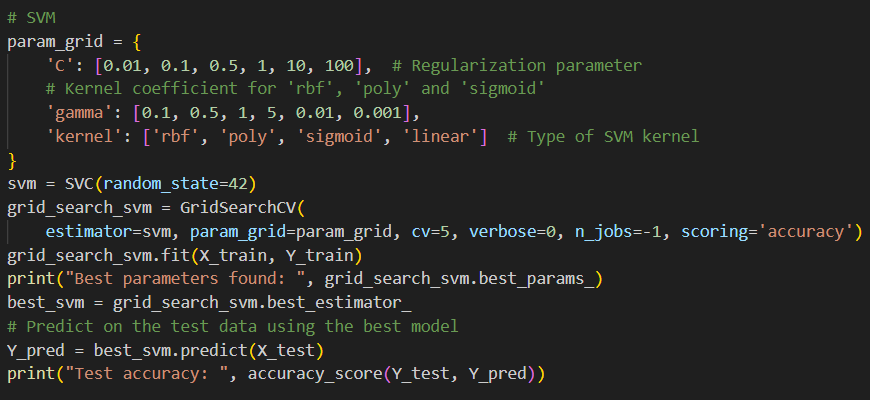
Random Forest:



Output:



SVM:

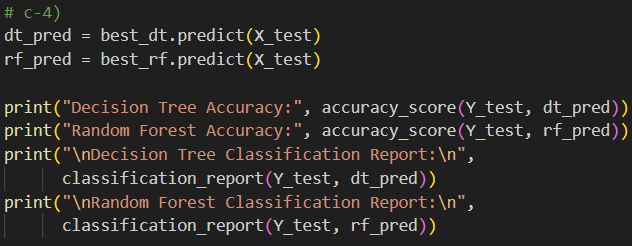


Output:

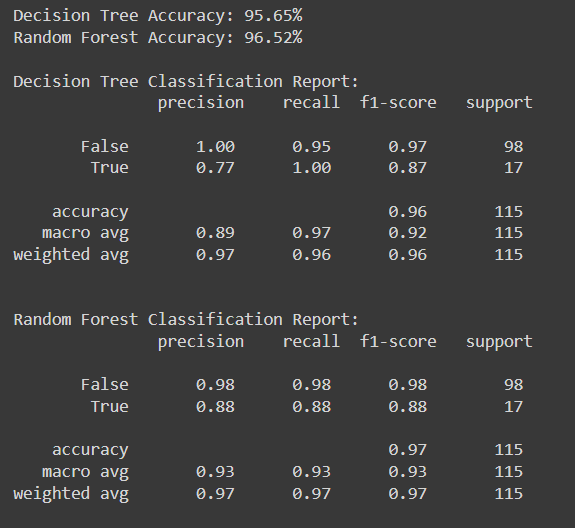


## C-4) Decision Tree vs. Random Forest

To compare the best results of our Decision Tree and Random Forest model, we wrote the code below to show their accuracy and classification report.



Output:



Bias and variance are two fundamental concepts in statistics and machine learning that describe different types of error in predictive models. Understanding the trade-off between bias and variance is crucial to developing models that generalize well from training data to unseen data.

Bias refers to the error introduced by approximating a real-world problem, which may be quite complicated, by a much simpler model. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting). The model is not complex enough to capture the underlying pattern of the data and hence has a high error on training and test data. On the other hand, Low bias means the model is flexible enough to capture the main trends in the data and hence has a low error on training data.

Variance refers to the amount by which the model's predictions would change if it were trained on a different dataset from the same overall population. Variance captures how much the model is influenced by the randomness in the training data. High variance means the model's predictions vary widely based on the specific observations in the training data, and it captures a lot of noise from the data (overfitting). Such a model performs well on training data but poorly on unseen test data. However, Low variance implies the model makes consistent predictions across different datasets, but it might not capture all the nuances of the data if the model is too simple.

The relationship between bias and variance is commonly described by the bias-variance trade-off. This trade-off is a fundamental problem that faces any supervised learning algorithm. The goal is to find a balance where both bias and variance are minimized to achieve the most generalizable model. Ideally, one wants a model with both low bias (accurately models the real data) and low variance (performs well on any input data from the population). However, these two properties are often at odds. Improving bias usually increases variance and vice versa.

In the context of bias and variance, Decision Trees typically exhibit high variance and low bias, while Random Forests, which are ensembles of decision trees, generally show lower variance and slightly higher bias. Here’s how this plays out:

**Decision Tree:**

High Variance: A single decision tree is highly sensitive to the data on which it is trained, and small changes to the data can lead to different tree structures. This is why decision trees are prone to overfitting, especially if they are allowed to grow deeply or complexly.

Low Bias: Decision trees make very few assumptions about the structure of the data; hence they are extremely flexible. They can capture complex patterns which can lead to a very accurate performance on the training data, but possibly poor generalization to unseen data.

**Random Forest:**

Lower Variance: Random forests create several decision trees on randomly selected data samples, then aggregate their predictions to make a final decision.

Slightly Higher Bias: In introducing randomness, random forests sacrifice a bit of the low-bias nature of individual decision trees. The combined effect of multiple trees in the forest leads to the averaging of results, which can smooth out the predictions, leading to slightly increased bias.

For the Decision Tree model, high recall and precision for the 'True' class suggest the tree model captured most of the positive cases perfectly, possibly due to overfitting to the training data.

For the Random Forest model, more balanced precision and recall across both classes and a slightly higher overall accuracy indicate better generalization to the data. The random forest model is less likely to overfit compared to a single decision tree.

The reported results align with this understanding: Random Forest shows slightly better performance in terms of overall accuracy and maintains more consistency across different metrics like precision and recall for both classes.

## C-5) Overfitting

This section is included in the attached code. However, since this section is omitted, we will not explain it in the report.