**Disbeties**

**Packages used:**

**Numpy:**

NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

**Pandas:**

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" ,

**Matplotlib:**

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Algorithm Used:

**Random Forest:**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

**"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.*"*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

**The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.**

**SVM:**

**Support Vector Machine** or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

**Dataset**: https://www.kaggle.com/datasets/saurabh00007/diabetescsv/

**CODE:**

**Importing Libraries**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from mlxtend.plotting import plot\_decision\_regions

import missingno as msno

from pandas.plotting import scatter\_matrix

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix

from sklearn import metrics

from sklearn.metrics import classification\_report

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline

#### Here we will be reading the dataset which is in the CSV format

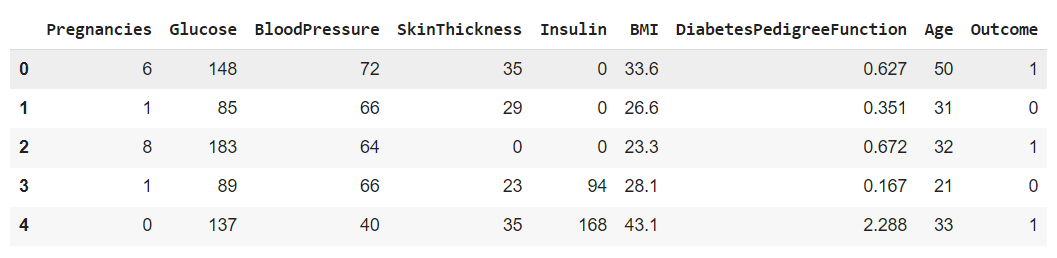
diabetes\_df = pd.read\_csv('diabetes.csv')

diabetes\_df.head()

#### Here we will be reading the dataset which is in the CSV format

diabetes\_df = pd.read\_csv('diabetes.csv')

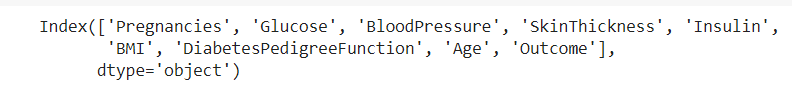
diabetes\_df.head()



## Exploratory Data Analysis (EDA)

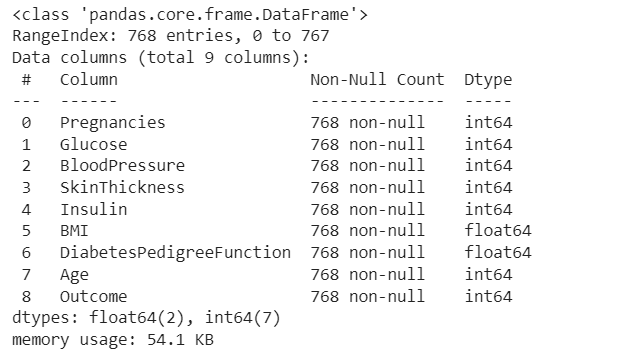
**Now let’ see that what are columns available in our dataset.**

diabetes\_df.columns



**Information about the dataset**

diabetes\_df.info()



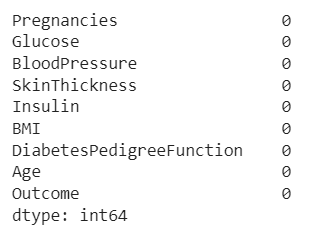
**To know more about the dataset**

diabetes\_df.describe()



**Now let’s check the number of null values our dataset has.**

diabetes\_df.isnull().sum()



diabetes\_df\_copy = diabetes\_df.copy(deep = True)

diabetes\_df\_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes\_df\_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

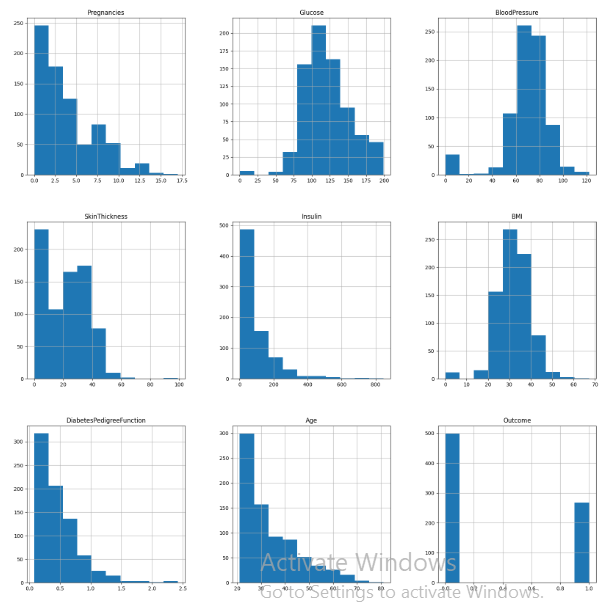
**# Showing the Count of NANs**

print(diabetes\_df\_copy.isnull().sum())

## Data Visualization

**Plotting the data distribution plots before removing null values**

p = diabetes\_df.hist(figsize = (20,20))



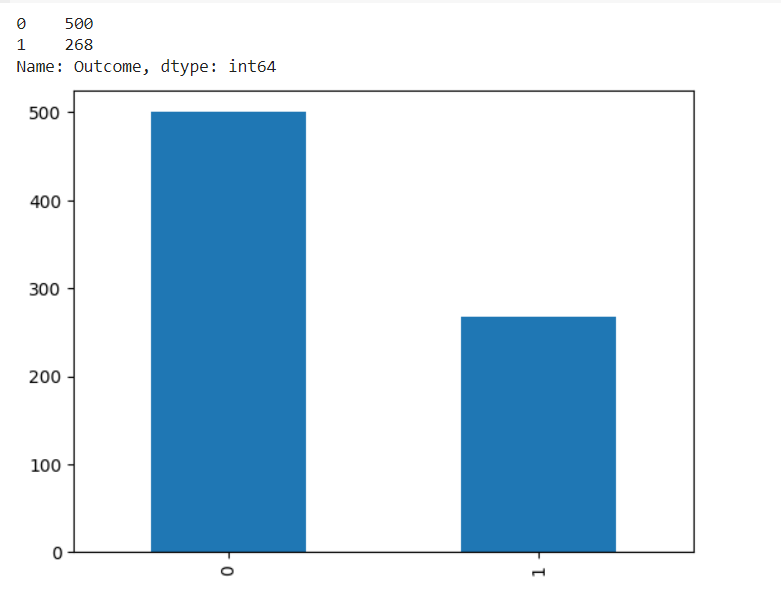
**Now, let’s check that how well our outcome column is balanced**

color\_wheel = {1: "#0392cf", 2: "#7bc043"}

colors = diabetes\_df["Outcome"].map(lambda x: color\_wheel.get(x + 1))

print(diabetes\_df.Outcome.value\_counts())

p=diabetes\_df.Outcome.value\_counts().plot(kind="bar")



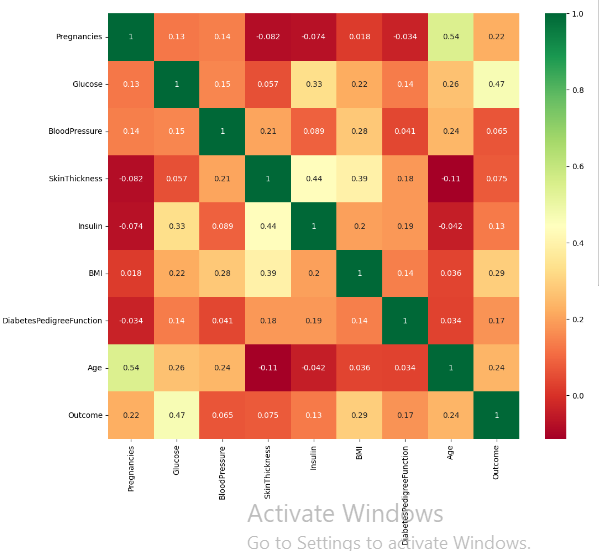
## Correlation between all the features

**Correlation between all the features before cleaning**

plt.figure(figsize=(12,10))

**# seaborn has an easy method to showcase heatmap**

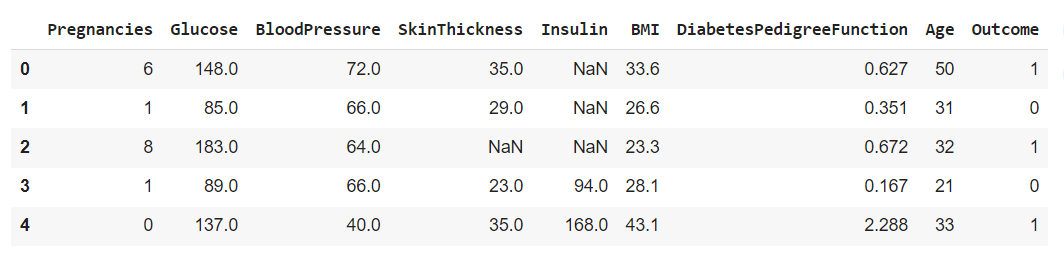
p = sns.heatmap(diabetes\_df.corr(), annot=True,cmap ='RdYlGn')



Scaling the Data

**Before scaling down the data let’s have a look into it**

diabetes\_df\_copy.head()



## Model Building

**Splitting the dataset**

X = diabetes\_df.drop('Outcome', axis=1)

y = diabetes\_df['Outcome']

**Now we will split the data into training and testing data using the train\_test\_split function**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.33, random\_state=7)

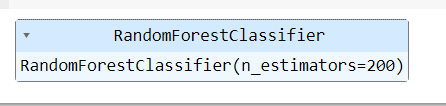
Random Forest

**Building the model using RandomForest**

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=200)

rfc.fit(X\_train, y\_train)



**Now after building the model let’s check the accuracy of the model on the training dataset.**

rfc\_train = rfc.predict(X\_train)

from sklearn import metrics

print("Accuracy\_Score =", format(metrics.accuracy\_score(y\_train, rfc\_train)))



**Getting the accuracy score for Random Forest**

from sklearn import metrics

predictions = rfc.predict(X\_test)

print("Accuracy\_Score =", format(metrics.accuracy\_score(y\_test, predictions)))

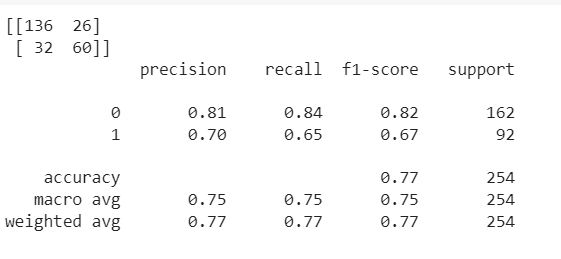


**Classification report and confusion matrix of random forest model**

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, predictions))

print(classification\_report(y\_test,predictions))



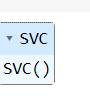
## Support Vector Machine (SVM)

**Building the model using Support Vector Machine (SVM)**

from sklearn.svm import SVC

svc\_model = SVC()

svc\_model.fit(X\_train, y\_train)



**Prediction from support vector machine model on the testing data**

svc\_pred = svc\_model.predict(X\_test)

**Accuracy score for SVM**

from sklearn import metrics

print("Accuracy Score =", format(metrics.accuracy\_score(y\_test, svc\_pred)))

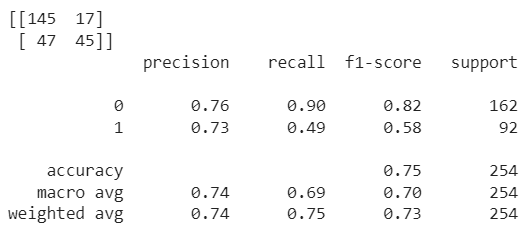


**Classification report and confusion matrix of the SVM classifier**

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, svc\_pred))

print(classification\_report(y\_test,svc\_pred))



**Conclusion:**

After using all these patient records, we are able to build a machine learning model (random forest – best one) to accurately predict whether or not the patients in the dataset have diabetes or not along with that we were able to draw some insights from the data via data analysis and visualization.