**AI BASED DIABETIES PREDICTION SYSTEM**

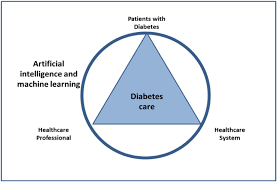
Name: S.M.Jeens Ferroy

Reg.No: 511721106302

**Phase 3 : Development part 1**

**Introduction:**

* Diabetes is a health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar (also called glucose) and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin.
* Without ongoing, careful management, diabetes can lead to a buildup of sugars in the blood, which can increase the risk of dangerous complications, including stroke and heart disease. So that i decide to predict using Machine Learning in Python.



**Installing Libraries:**

In this first step I have imported most common libraries used in python for machine learning such as Pandas, Seaborn, Matplitlib etc.

I am using Python because if very flexible and effective programming language i ever used. I used Python in software development field too.

# Import libraries  
import numpy as np *# linear algebra*  
import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*  
import seaborn as sns *# for data visualization*  
import matplotlib.pyplot as plt *# to plot charts*  
from collections import Counter  
import os  
  
*# Modeling Libraries*  
from sklearn.preprocessing import QuantileTransformer  
from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score  
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import SVC  
from sklearn.model\_selection import GridSearchCV, cross\_val\_score, StratifiedKFold, learning\_curve, train\_test\_split

The sklearn library is very versatile and handy and serves real-world purposes. It provides wide range of ML algorithms and Models.

**Importing Data:**

In this project, i used Diabetes Database from Kaggle. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.

**Database Link: (**<https://www.kaggle.com/datasets/mathchi/diabetes-data-set/download?datasetVersionNumber=1>)

# Import dataset  
df = pd.read\_csv("../input/diabetes-database/diabetes.csv")

*# Get familier with dataset structure*  
df.info()

**Output:**

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

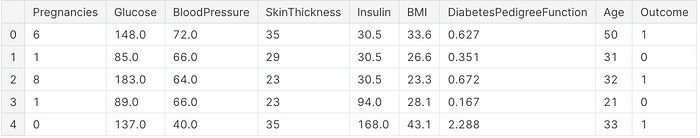
dtypes: float64(2), int64(7)

memory usage: 54.1 KB

Excepting BMI and DiabetesPedigreeFunction all the columns are integer. Outcome is the label containing 1 and 0 values. 1 means person has diabetes and 0 mean person is not diabetic.

*# Show top 5 rows*  
df.head()

**Output:**



**Data Exploration:**

Explore the dataset to understand its structure. Check for missing values, data types, and basic statistics.

**Missing Value Analysis**

Next, i will cleanup the dataset which is the important part of data science. Missing data can lead to wrong statistics during modeling and predictions.

*# Explore missing values*

df.isnull().sum()

**Output:**

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome 0

dtype: int64

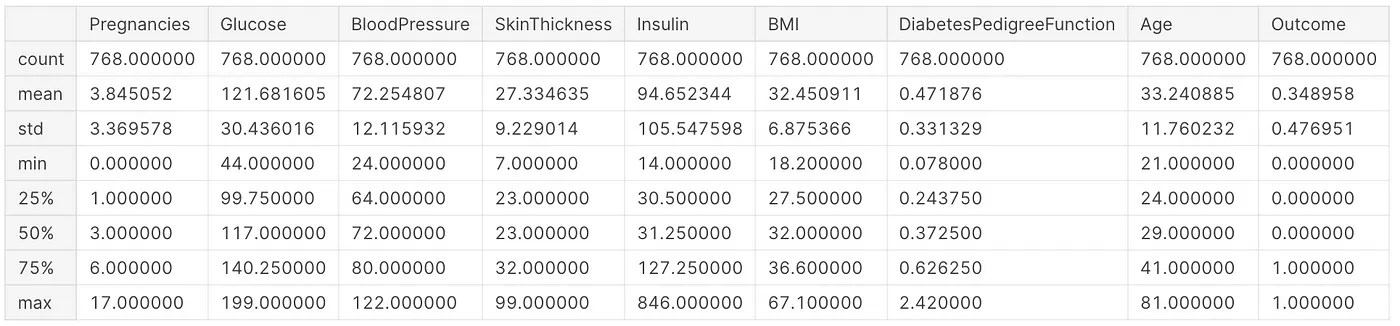
I observed that there is no missing values in dataset however the features like Glucose, BloodPressure, Insulin, SkinThickness has 0 values which is not possible. We have to replace 0 values with either mean or median values of specific column.

df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())*# Correcting missing values in blood pressure*  
df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].mean()) # There are 35 records with 0 BloodPressure in dataset*# Correcting missing values in BMI*  
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())*# Correct missing values in Insulin and SkinThickness*  
  
df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())  
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())

**Dataset statistics:**

Now, lets review the dataset statistics

*# Review dataset statistics*  
df.describe()



Now i have clean dataset without missing values in features which is good.

**Feature Engineering:**

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning.

Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.

**Outlier Detection**

In this part i removed all the records outlined in dataset. Outliers impacts Model accuracy. I used Tukey Method used for outlier detection.

def detect\_outliers(df,n,features):  
 outlier\_indices = []  
 *"""*  
 *Detect outliers from given list of features. It returns a list of the indices*  
 *according to the observations containing more than n outliers according*  
 *to the Tukey method*  
 *"""*  
 *# iterate over features(columns)*  
 for col **in** features:  
 Q1 = np.percentile(df[col], 25)  
 Q3 = np.percentile(df[col],75)  
 IQR = Q3 - Q1  
   
 *# outlier step*  
 outlier\_step = 1.5 \* IQR  
   
 *# Determine a list of indices of outliers for feature col*  
 outlier\_list\_col = df[(df[col] < Q1 - outlier\_step) | (df[col] > Q3 + outlier\_step )].index  
   
 *# append the found outlier indices for col to the list of outlier indices*   
 outlier\_indices.extend(outlier\_list\_col)  
   
 *# select observations containing more than 2 outliers*  
 outlier\_indices = Counter(outlier\_indices)  
 multiple\_outliers = list( k for k, v **in** outlier\_indices.items() if v > n )  
   
 return multiple\_outliers   
  
*# detect outliers from numeric features*  
outliers\_to\_drop = detect\_outliers(df, 2 ,["Pregnancies", 'Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'SkinThickness', 'Insulin', 'Age'])

Here, I find outliers from all the features such as Pregnancies, Glucose, BloodPressure, BMI, DiabetesPedigreeFunction, SkinThickness, Insulin, and Age.

df.drop(df.loc[outliers\_to\_drop].index, inplace=True)

**Modeling**

In this sections, i tried different models and compare the accuracy for each. Then, i performed Hyperparameter Tuning on Models that has high accuracy.

Before i split the dataset i need to transform the data into quantile using

sklearn.preprocessing .

# Data Transformation  
q = QuantileTransformer()  
X = q.fit\_transform(df)  
transformedDF = q.transform(X)  
transformedDF = pd.DataFrame(X)  
transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']# Show top 5 rows  
transformedDF.head()



**Data Splitting**

Next, i split data in test and train dataset. Train dataset will be used in Model training and evaluation and test dataset will be used in prediction. Before i predict the test data, i performed cross validation for various models.

features = df.drop(["Outcome"], axis=1)  
labels = df["Outcome"]x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.30, random\_state=7)

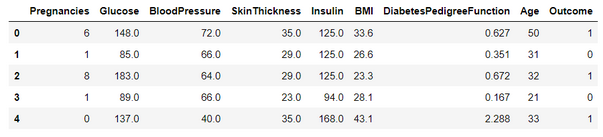
Above code splits dataset into train (70%) and test (30%) dataset.

**Scaling the Data**

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

diabetes\_df\_copy.head()

**Output:**



**After Standard Scaling**

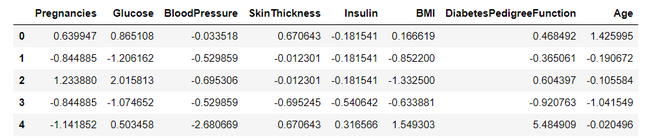
sc\_X = StandardScaler()

X = pd.DataFrame(sc\_X.fit\_transform(diabetes\_df\_copy.drop(["Outcome"],axis = 1),), columns=['Pregnancies',

'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'])

X.head()

**Output:**



That’s how our dataset will be looking like when it is scaled down or we can see every value now is on the same scale which will help our ML model to give a better result.

**Let’s explore our target column**

**Output:**

0 1

1 0

2 1

3 0

4 1

..

763 0

764 0

765 0

766 1

767 0

Name: Outcome, Length: 768, dtype: int64

**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)

**Feature Importance**

Knowing about the feature importance is quite necessary as it shows that how much weightage each feature provides in the model building phase.

**Getting feature importances**

rfc.feature\_importances\_

**Output:**

array([0.07684946, 0.25643635, 0.08952599, 0.08437176, 0.08552636,

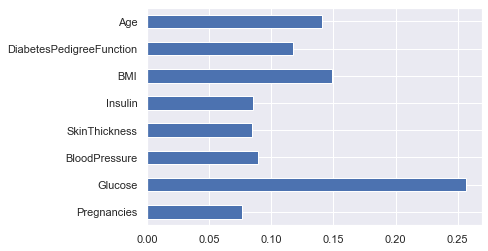
0.14911634, 0.11751284, 0.1406609 ])

From the above output, it is not much clear that which feature is important for that reason we will now make a visualization of the same.

**Plotting feature importances**

(pd.Series(rfc.feature\_importances\_, index=X.columns).plot(kind='barh'))

**Output:**



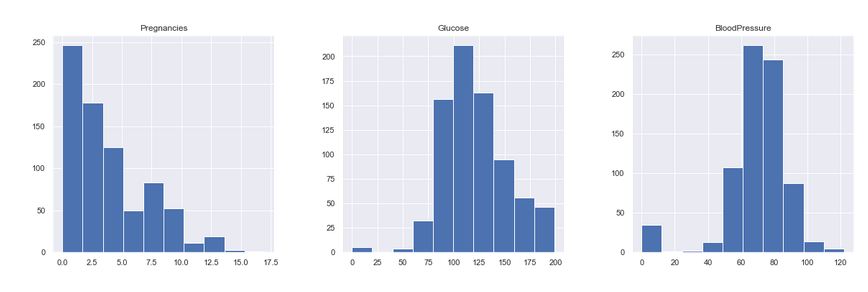
Here from the above graph, it is clearly visible that Glucose as a feature is the most important in this dataset.

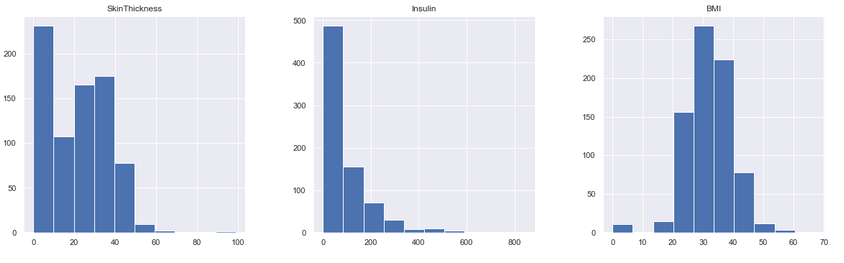
**Data Visualization:**

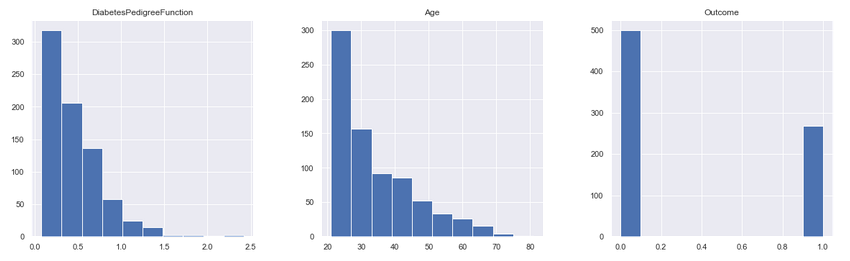
Plotting the data distribution plots before removing null values

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

**Output:**





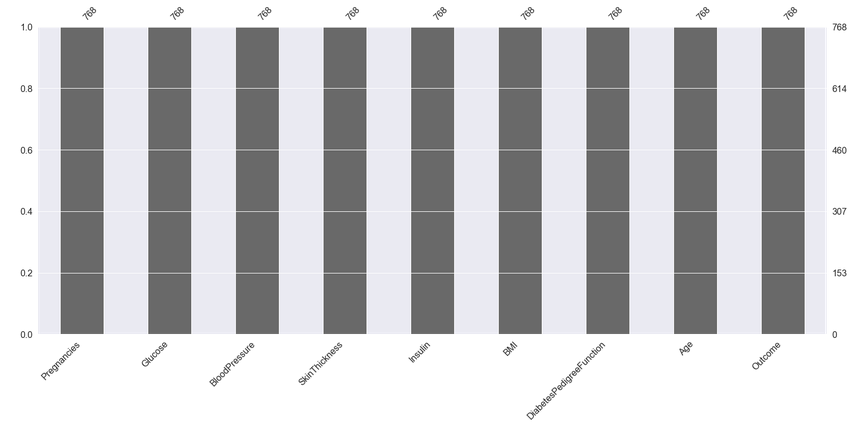


Inference: Here we are again using the hist plot to see the distribution of the dataset but this time we are using this visualization to see the changes that we can see after those null values are removed from the dataset and we can clearly see the difference for example – In age column after removal of the null values, we can see that there is a spike at the range of 50 to 100 which is quite logical as well.

**Plotting Null Count Analysis Plot**

p = msno.bar(diabetes\_df)

**Output:**



Inference: Now in the above graph also we can clearly see that there are no null values in the dataset.

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())

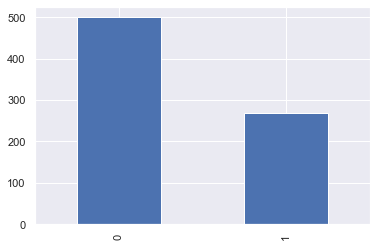
p=diabetes\_df.Outcome.value\_counts().plot(kind="bar")

**Output:**

0 500

1 268

Name: Outcome, dtype: int64



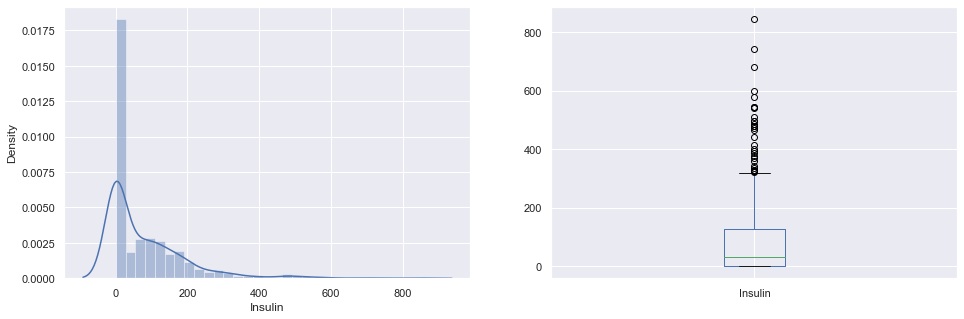
Inference: Here from the above visualization it is clearly visible that our dataset is completely imbalanced in fact the number of patients who are diabetic is half of the patients who are non-diabetic.

plt.subplot(121), sns.distplot(diabetes\_df['Insulin'])

plt.subplot(122), diabetes\_df['Insulin'].plot.box(figsize=(16,5))

plt.show()

**Output:**



**Inference:** That’s how Distplot can be helpful where one will able to see the distribution of the data as well as with the help of boxplot one can see the outliers in that column and other information too which can be derived by the box and whiskers plot.