IBM NAAN MUDHALVAN

ARTIFICIAL INTELLIGENCE

PROJECT

AI-BASED DIABETES PREDICTION SYSTEM

PROBLEM DEFINITION:

The problem definition for an AI-based diabetes prediction system is to develop a model that can accurately predict the likelihood of an individual developing diabetes based on their demographic, lifestyle, and health factors. The system should be able to analyze large amounts of data, including medical records, genetic information, and lifestyle choices, to provide personalized predictions. The goal is to assist healthcare professionals in identifying individuals who are at high risk of developing diabetes, allowing for early intervention and prevention strategies. The system should also be user-friendly and easily accessible to both healthcare professionals and individuals looking to assess their own risk of diabetes.

DESIGN THINKING:

Design thinking is a human-centered approach to problem-solving that focuses on understanding the needs and desires of users in order to create innovative solutions. When applying design thinking to the development of an AI-based diabetes prediction system, it is important to empathize with the users, define the problem, ideate potential solutions, prototype and test those solutions, and finally, implement and iterate on the system based on user feedback. Here is a step-by-step guide on how to apply design thinking to the development of an AI-based diabetes prediction system.

DATA COLLECTION:

We need a dataset containing medical features such as glucose levels, blood pressure, BMI, etc., along with information about whether the individual has diabetes or not.

DATA PREPROCESSING:

The medical data needs to be cleaned, normalized, and prepared for training machine learning models.

FEATURE SELECTION:

We will select relevant features that can impact diabetes risk prediction.

MODEL SELECTION:

We can experiment with various machine learning algorithms like Logistic Regression, Random Forest, and Gradient Boosting.

EVALUATION:

We will evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

ITERATIVE IMPROVEMENT:

We will fine-tune the model parameters and explore techniques like feature engineering to enhance prediction accuracy.

DEFINE:

- Synthesize the research findings to define the problem statement and identify the key objectives and goals of the diabetes prediction system.

- Clearly articulate the target user groups and their specific needs and expectations.

IDEATE:

- Brainstorm potential AI-based solutions that can address the identified problem statement and meet the needs of the target users.

- Encourage diverse perspectives and generate a wide range of ideas.

- Prioritize ideas based on feasibility, desirability, and viability.

PROTOTYPE:

- Create low-fidelity prototypes of the potential solutions to visualize and test their functionality and usability.

- Use tools like wireframing software or even paper sketches to create simple representations of the system.

- Focus on the core features and functionalities that are crucial for diabetes prediction.

TEST:

- Conduct usability testing sessions with representative users to gather feedback on the prototypes.

- Observe how users interact with the system, identify pain points, and collect suggestions for improvements.

- Iterate on the prototypes based on user feedback, making necessary adjustments and refinements.

IMPLEMENT:

- Develop the AI-based diabetes prediction system based on the refined prototypes and user feedback.

- Collaborate with developers and data scientists to ensure the accuracy and reliability of the prediction algorithms.

- Integrate the system with existing healthcare systems and data sources.

ITERATE:

- Continuously gather user feedback and iterate on the system to improve its performance, usability, and relevance.

- Regularly update the prediction algorithms based on new research and advancements in AI technology.

- Engage users in the design process through feedback channels and user co-creation sessions.

By following these design thinking principles and steps, you can create an

AI-based diabetes prediction system that truly understands and meets the needs of its users, ultimately improving their health outcomes and quality of life.

DATASET USED:

DATA SOURCE:

A data source for Diabetes prediction system using Artificial intelligence

Dataset link: https://www.kaggle.com/datasets/mathchi/diabetes-data-set

PROGRAM:

```
In [1]:  import pandas as pd
         import numpy as np
         from sklearn.preprocessing import StandardScaler
         from sklearn.linear_model import LogisticRegression
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score, confusion_matrix
         import matplotlib.pyplot as plt
         import seaborn as sns

In[2]:    data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
          data.head()

Out[2]:
```

2	8	183	64	0	0	23.3	0.672	32	1
3	1 0	89	66	23	94	28.1	0.167	21	0
Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
4		137	40	35	168	43.1	2.288	33	1

```
In [3]: data.describe()

Out [3]:
```

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	17.00000	199.000000	122.00000	99.00000	846.0000	67.100000	2.420000	81.00000	1.000000

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
	0		0	0	00			0	

```
In [4]: data.isnull().sum()
Out [ 4 ]:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

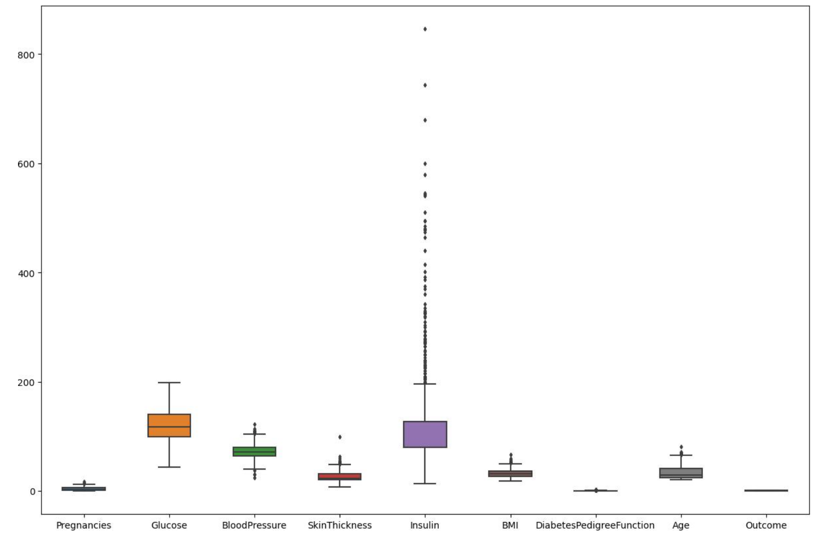
```
In [5]:
#here few misconception is there lke BMI can not be zero, BP can't be zero, glucose,
insuline can't be zero so lets try to fix it
# now replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [6]:

```
#now we have dealt with the 0 values and data looks better. But, there still are outliers
present in some columns.lets visualize it
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

Out [6]:

```
<Axes: >
```



In [7]:

```
Data.head()
```

Out [7]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	1
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	0
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	1
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	0
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	1

In [8]:

```
#segregate the dependent and independent variable
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

In [9]:
```
  # separate dataset into train and test
  X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.25,random_state=0)
  X_train.shape, X_test.shape
```
Out[9]:
```
  ((576, 8), (192, 8))
```
In [10]:
```
  import pickle
  ##standard Scaling- Standardization
  def scaler_standard(X_train, X_test):
    #scaling the data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    #saving the model
    file = open('standardScalar.pkl','wb')
    pickle.dump(scaler,file)
    file.close()

    return X_train_scaled, X_test_scaled
```
In [11]:
```
   X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```
In [12]:
```
   X_train_scaled
```
Out[12]:
```
    array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,
         -0.98325882, -0.04863985],
        [-0.82986389, -0.1331471 , -1.23618124, ...,  0.09272955,
         -0.62493647, -0.88246592],
        [-1.12204091, -1.03283573,  0.61597784, ..., -0.03629955,
          0.39884168, -0.5489355 ],
        ...,
        [ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,
         -0.96519215, -1.04923114],
        [ 2.09190629, -1.23276654,  0.11084355, ..., -0.36604058,
         -0.5075031 ,  0.11812536],
        [ 0.33884418,  0.46664532,  0.78435594, ..., -0.09470985,
          0.51627505,  2.953134  ]])
```
In [13]:
```
   log_reg = LogisticRegression()

   log_reg.fit(X_train_scaled,y_train)
```
Out[13]:
```
  ✓ LogisticRegression
  LogisticRegression()
```
In [14]:
```
  ## Hyperparameter Tuning##
  GridSearch CV
  from sklearn.model_selection import GridSearchCVimport numpy as
  np
  import warnings
```

```
    warnings.filterwarnings('ignore')
    # parameter grid
    parameters = {
        'penalty' : ['l1','l2'],
        'C'               : np.logspace(-3,3,7),
        'solver'   : ['newton-cg', 'lbfgs', 'liblinear'],
}
```

In [15]:

```
    logreg = LogisticRegression()
    clf = GridSearchCV(logreg,                                    # model
                       param_grid = parameters,                   # hyperparameters
                       scoring='accuracy',                        # metric for scoringcv=10)  #
                       number of folds

    clf.fit(X_train_scaled,y_train)
```

Out[15]:

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [16]:

```
    clf.best_params_
```

Out[16]:

```
{'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

In [17]:

```
    clf.best_score_
```

Out[17]:

```
0.763793103448276
```

let's see how well our model performs on the test data set.

In [18]:

```
    y_pred = clf.predict(X_test_scaled)
   accuracy = accuracy_score(y_test,y_pred) accuracy
```

In [19]:

```
    conf_mat = confusion_matrix(y_test,y_pred)conf_mat
```

Out[19]:

```
array([[117,   13],
       [ 26,   36]])
```

In [20]:

```
    true_positive = conf_mat[0][0] false_positive
    = conf_mat[0][1]false_negative =
    conf_mat[1][0]true_negative = conf_mat[1][1]
```

In [21]:

```
    Accuracy = (true_positive + true_negative) / (true_positive
+false_positive +                 false_negative + true_negative)Accuracy
```

Out[21]:

```
0.796875
```

In [22]:
```
Precision = true_positive/(true_positive+false_positive)Precision
```
Out[22]:
```
0.9
```
In [23]:
```
Recall = true_positive/(true_positive+false_negative)Recall
```
Out[23]:
```
0.8181818181818182
```
In [24]:
```
F1_Score = 2*(Recall * Precision) / (Recall + Precision)F1_Score
```
Out[24]:
```
0.8571428571428572
```
In [25]:

linkcode
```
import pickle
file = open('modelForPrediction.pkl','wb')
pickle.dump(log_reg,file)
file.close()
```

OVERVIEW:

In this article, we will be predicting that whether the patient has diabetes or not on the basis of the features we will provide to our machine learning model, and for that, we will be using the famous Pima Indians Diabetes Database.

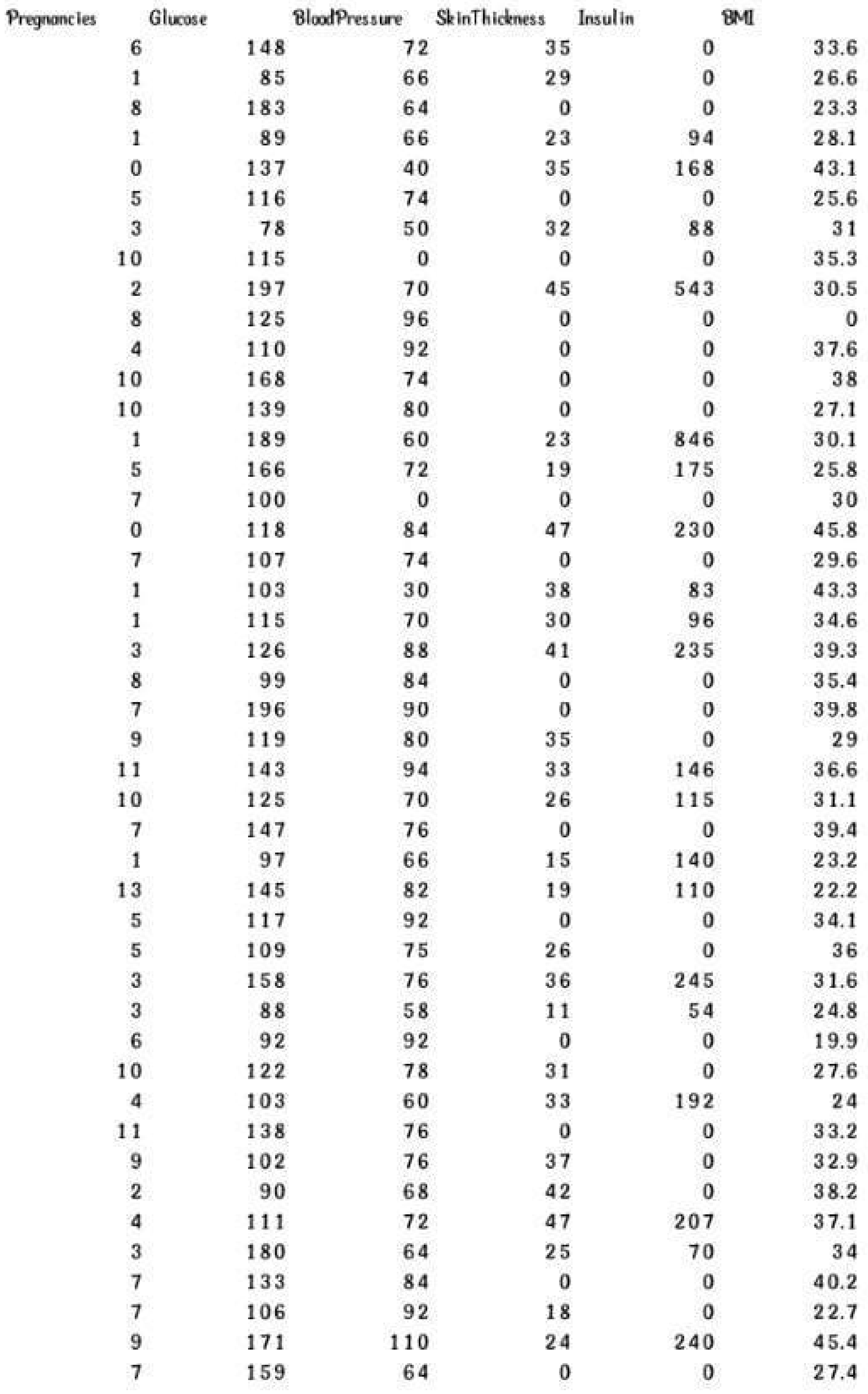
1. Data analysis: Here one will get to know about how the data analysis part is done in a data science life cycle.

2. Exploratory data analysis: EDA is one of the most important steps in the data science project life cycle and here one will need to know that how to make inferences from the visualizations and data analysis

3. Model building: Here we will be using 4 ML models and then we will choose the best performing model.

4. Saving model: Saving the best model using pickle to make the prediction from real data.

DATASET:



Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
6	148	72	35	0	33.6
1	85	66	29	0	26.6
8	183	64	0	0	23.3
1	89	66	23	94	28.1
0	137	40	35	168	43.1
5	116	74	0	0	25.6
3	78	50	32	88	31
10	115	0	0	0	35.3
2	197	70	45	543	30.5
8	125	96	0	0	0
4	110	92	0	0	37.6
10	168	74	0	0	38
10	139	80	0	0	27.1
1	189	60	23	846	30.1
5	166	72	19	175	25.8
7	100	0	0	0	30
0	118	84	47	230	45.8
7	107	74	0	0	29.6
1	103	30	38	83	43.3
1	115	70	30	96	34.6
3	126	88	41	235	39.3
8	99	84	0	0	35.4
7	196	90	0	0	39.8
9	119	80	35	0	29
11	143	94	33	146	36.6
10	125	70	26	115	31.1
7	147	76	0	0	39.4
1	97	66	15	140	23.2
13	145	82	19	110	22.2
5	117	92	0	0	34.1
5	109	75	26	0	36
3	158	76	36	245	31.6
3	88	58	11	54	24.8
6	92	92	0	0	19.9
10	122	78	31	0	27.6
4	103	60	33	192	24
11	138	76	0	0	33.2
9	102	76	37	0	32.9
2	90	68	42	0	38.2
4	111	72	47	207	37.1
3	180	64	25	70	34
7	133	84	0	0	40.2
7	106	92	18	0	22.7
9	171	110	24	240	45.4
7	159	64	0	0	27.4

769 rows*9 columns

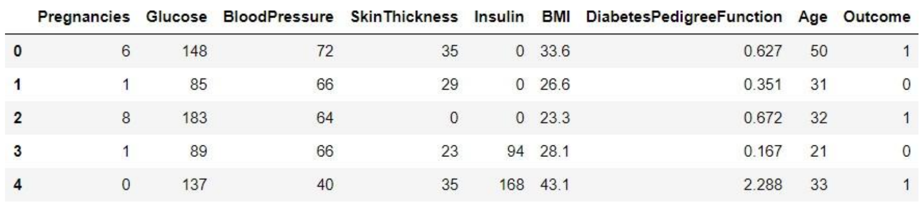
IMPORT LIBRARIES:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

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()
```

Output:



	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

Exploratory Data Analysis (EDA)

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

```
diabetes_df.columns
```

Output:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Information about the dataset

```
diabetes_df.info()
```

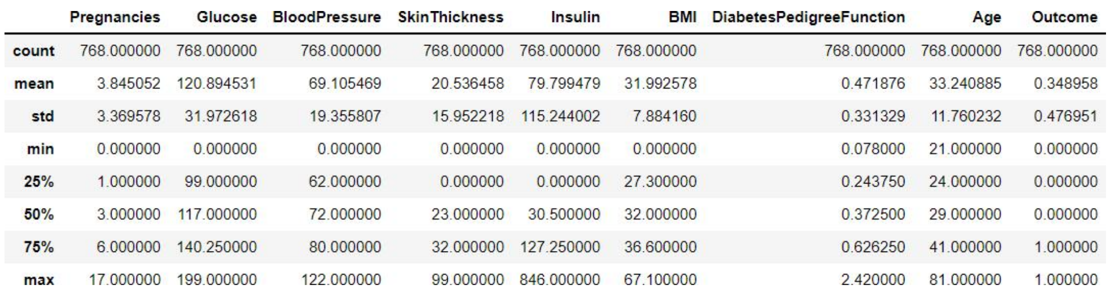
Output:

```
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
```

To know more about the dataset

```
diabetes_df.describe()
```

Output:

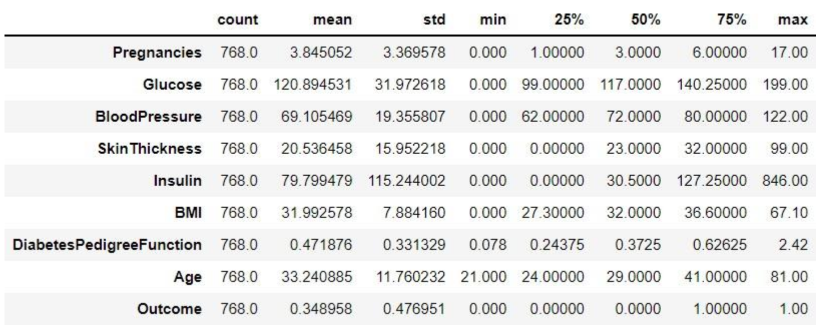


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	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

To know more about the dataset with transpose – here T is for the transpose

```
diabetes_df.describe().T
```

Output:

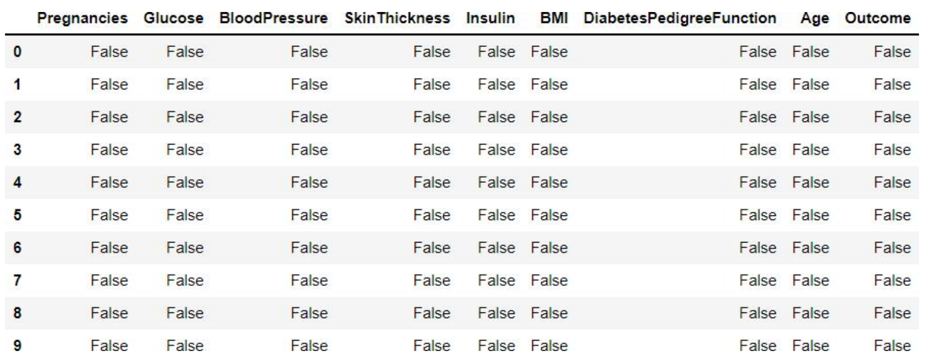


	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

Now let’s check that if our dataset have null values or not

```
diabetes_df.isnull().head(10)
```

Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False

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

```
diabetes_df.isnull().sum()
```

Output:

```
Pregnancies                   0
Glucose                       0
BloodPressure                 0
SkinThickness                 0
Insulin                       0
BMI                           0
```

```
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Here from the above code we first checked that is there any null values from the **IsNull()** function then we are going to take the sum of all those missing values from the **sum()** function and the inference we now get is that there are no missing values but that is actually not a true story as in **this particular dataset all the missing values were given the 0 as a value which is not good for the authenticity of the dataset.** Hence we will first **replace the 0 value with the NAN** value then start the imputation process.

```
diabetes_df_copy = diabetes_df.copy(deep = True)
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insuli
n','BMI']] =
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insuli
n','BMI']].replace(0,np.NaN)


# Showing the Count of NANs
print(diabetes_df_copy.isnull().sum())
```

Output:

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
```

```
Outcome                        0
dtype: int64
```

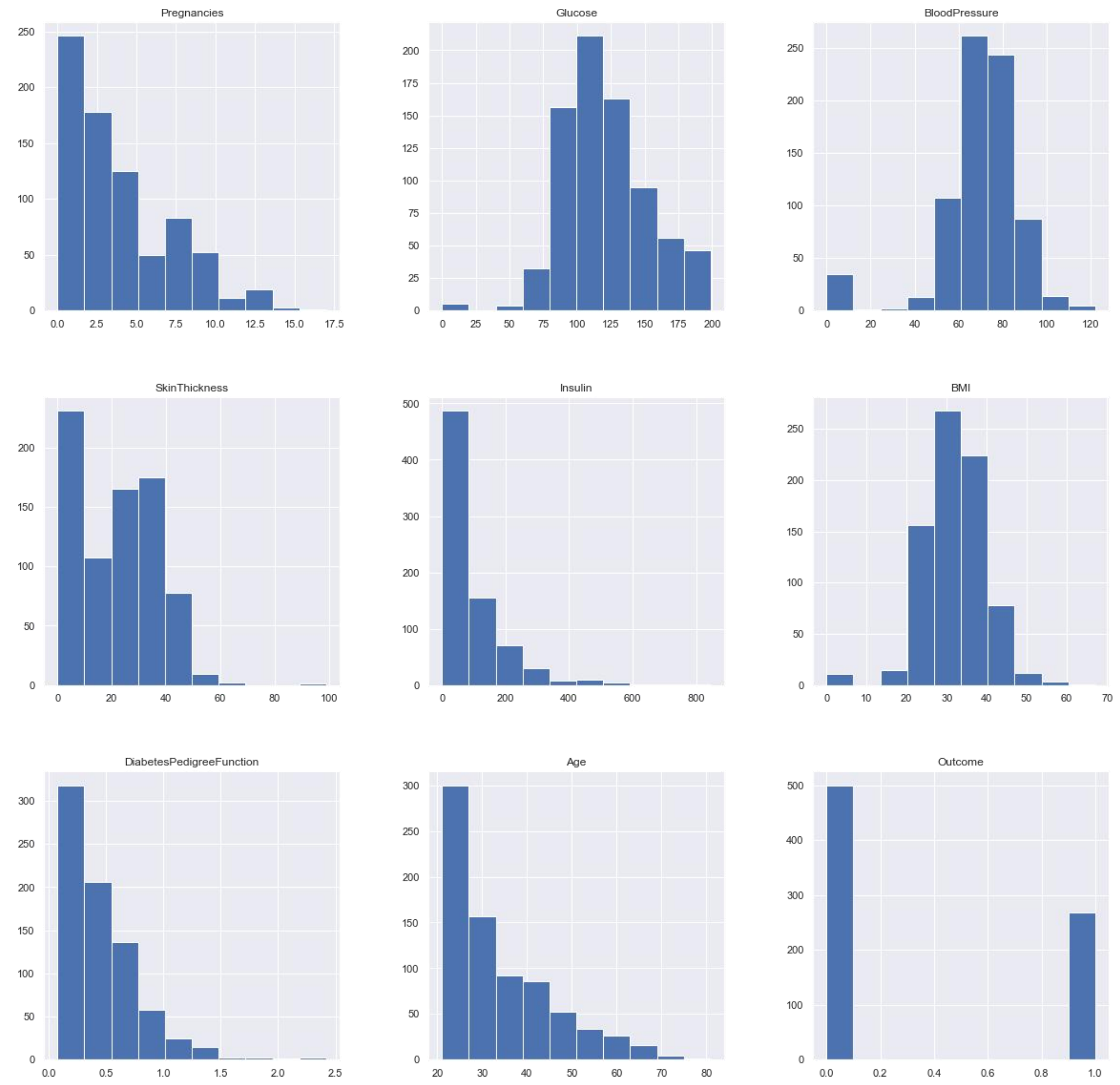
As mentioned above that now **we will be replacing the zeros with the NAN values** so that we can impute it later to maintain the authenticity of the dataset as well as trying to have a better Imputation approach i.e **to apply mean values of each column to the null values of the respective columns.**

Data Visualization

Plotting the data distribution plots before removing null values

```
p = diabetes_df.hist(figsize = (20,20))
```

Output:



Inference: So here we have seen the distribution of each features whether it is dependent data or independent data and one thing which could always strike that **why do we need to see the distribution of data?** So the answer is simple it is the best way to start the analysis of the dataset as **it shows the occurrence of every kind of value in the graphical structure which in turn lets us know the range of the data.**

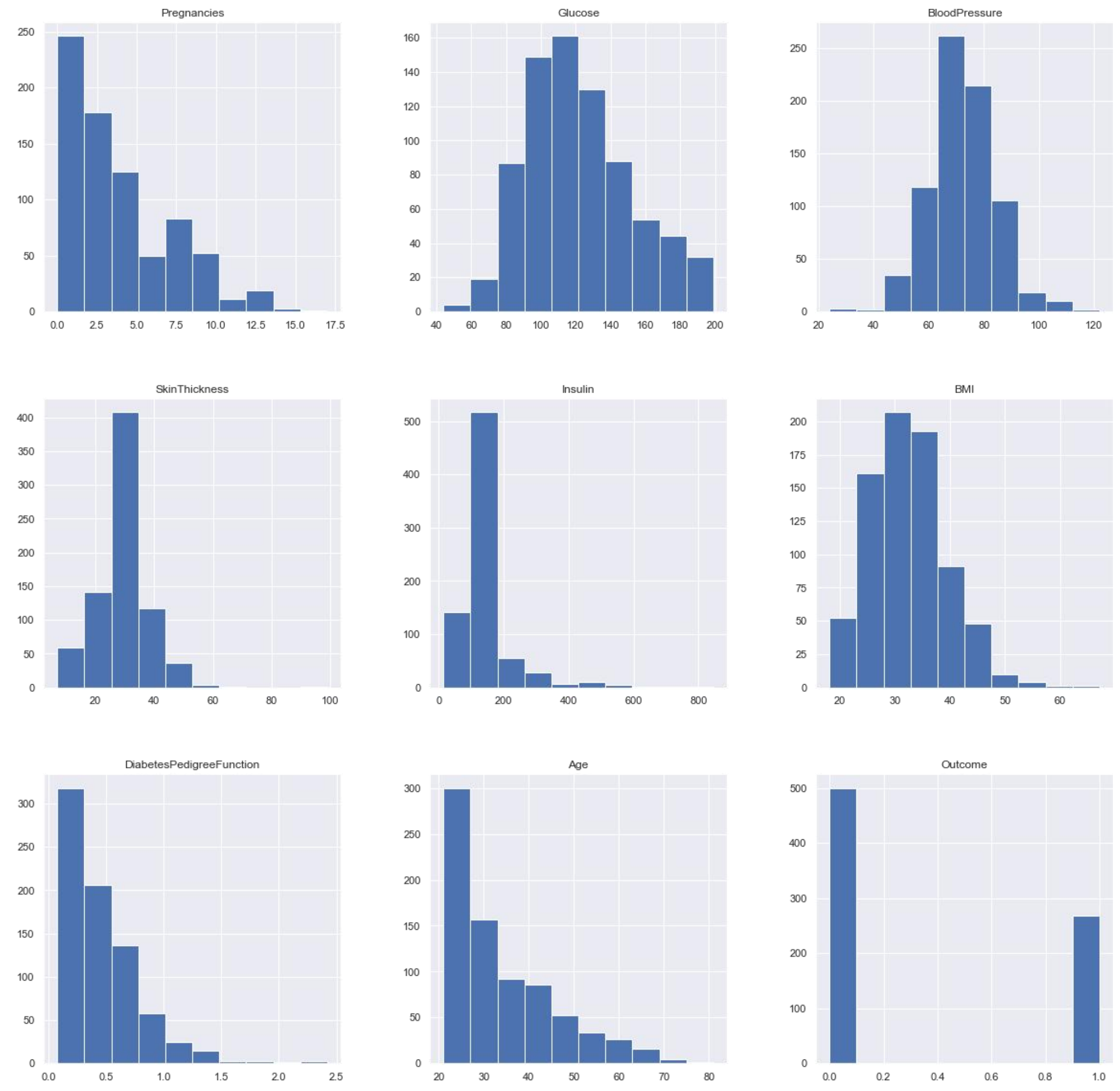
Now we will be imputing the mean value of the column to each missing value of that particular column.

```
diabetes_df_copy['Glucose'].fillna(diabetes_df_copy['Glucose'].mean
(), inplace = True)

diabetes_df_copy['BloodPressure'].fillna(diabetes_df_copy['BloodPre
ssure'].mean(), inplace = True)

diabetes_df_copy['SkinThickness'].fillna(diabetes_df_copy['SkinThic
kness'].median(), inplace = True)

diabetes_df_copy['Insulin'].fillna(diabetes_df_copy['Insulin'].medi
an(), inplace = True)

diabetes_df_copy['BMI'].fillna(diabetes_df_copy['BMI'].median(),
inplace = True)
```

Plotting the distributions after removing the NAN values.

```
p = diabetes_df_copy.hist(figsize = (20,20))
```

Output:

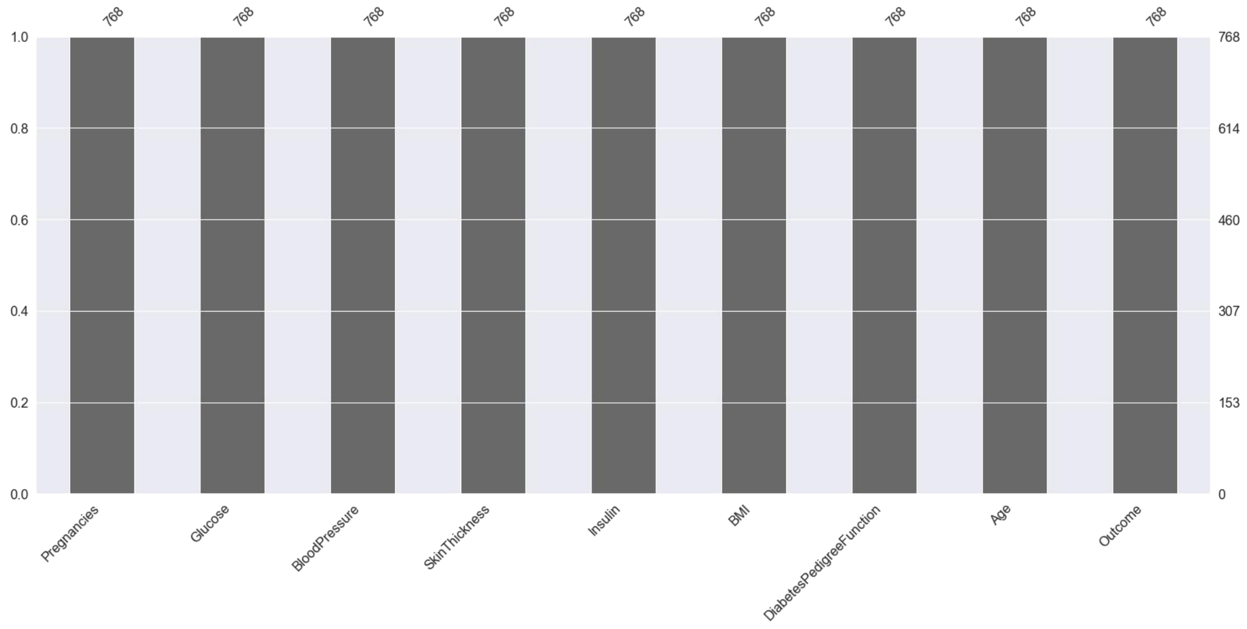


Inference: Here we are again using the hist plot to **see the distribution of the dataset** but this time we are using this visualization to see the changes that we can see after those null values are removed from the dataset and we can clearly see the difference **for example –** In age column after 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:



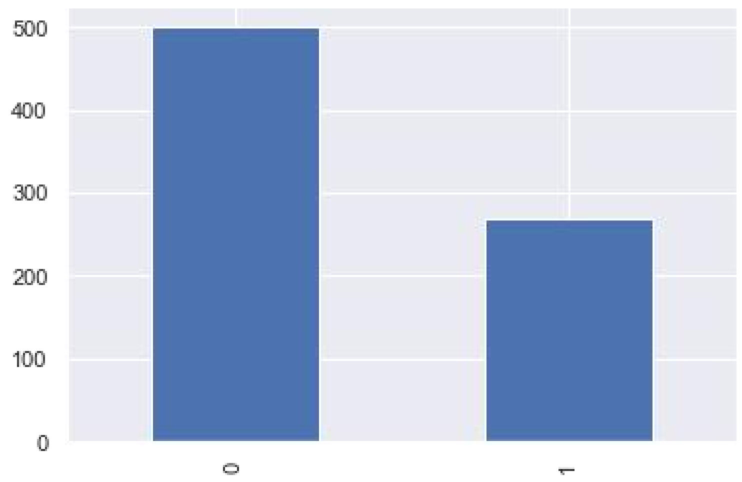
Inference: Now in the above graph also we can clearly see that there are **no null** values in the dataset.

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:

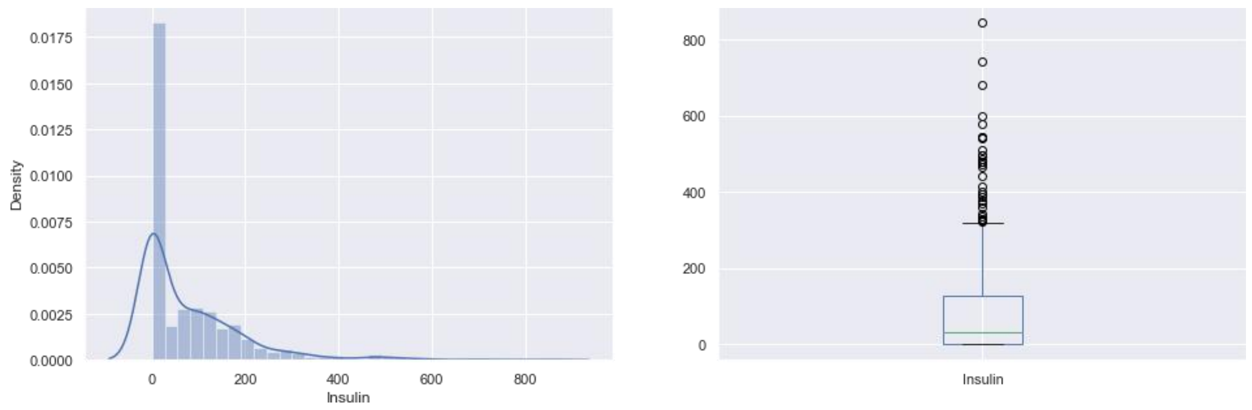
```
0    500
1    268
Name: Outcome, dtype: int64
```



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**

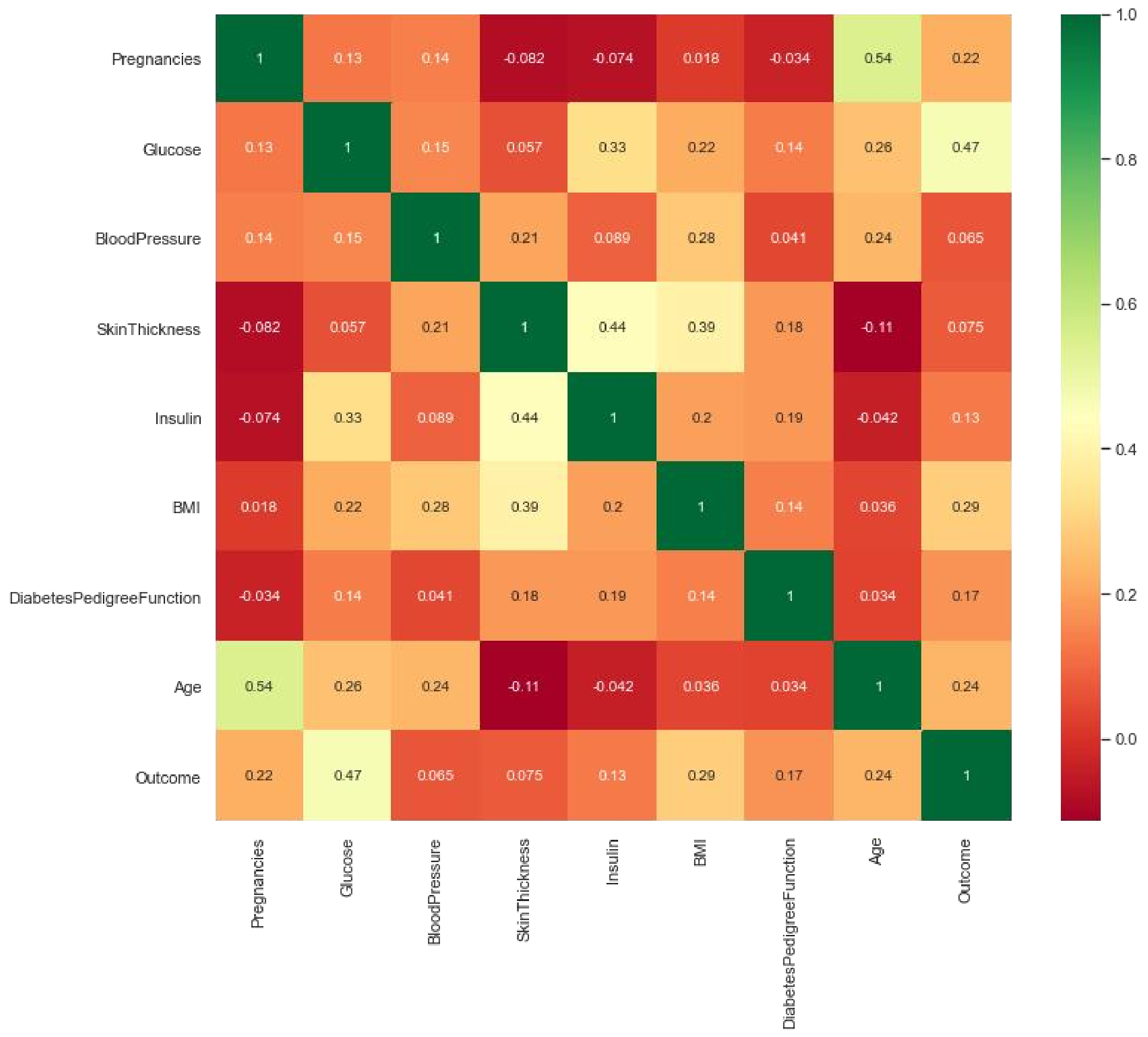
outliers in that column and other information too which can be derived by the **box and whiskers plot.**

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')
```

Output:

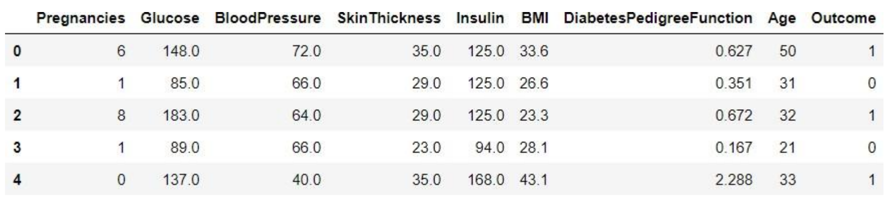


Scaling the Data

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

```
diabetes_df_copy.head()
```

Output:

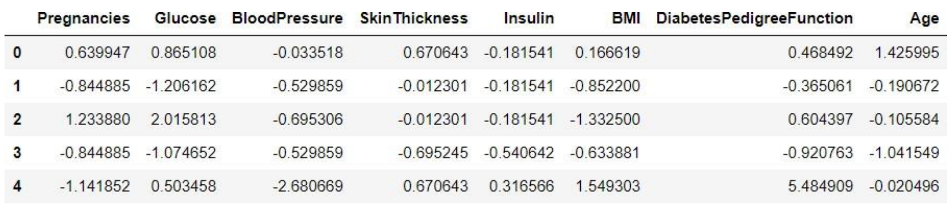


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

After Standard scaling

```
sc_X = StandardScaler()

X =
pd.DataFrame(sc_X.fit_transform(diabetes_df_copy.drop(["Outcome"],a
xis = 1),), columns=['Pregnancies',

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

X.head()
```

Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

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

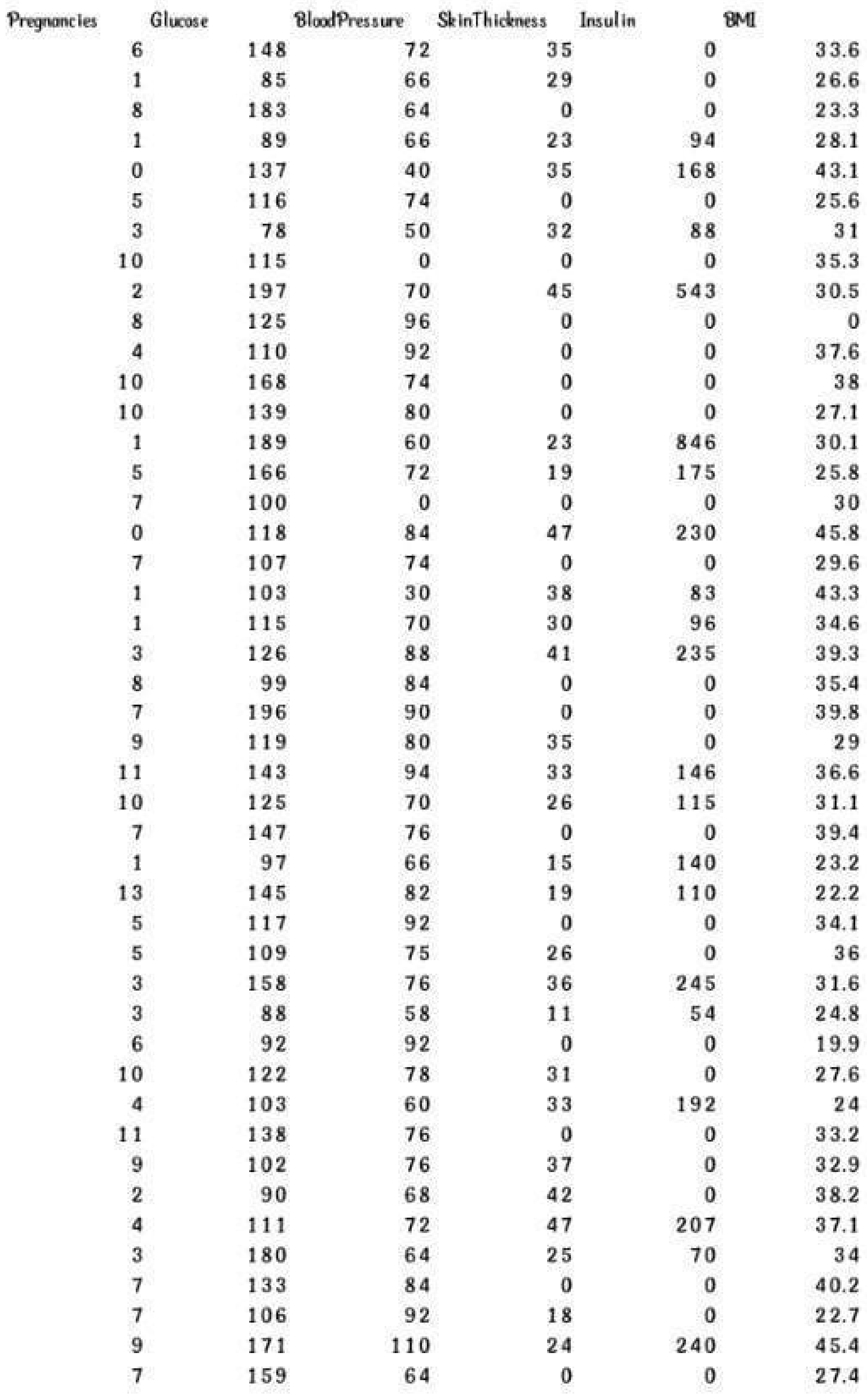
Splitting the dataset

```
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)
```

GIVEN DATASET:



Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
6	148	72	35	0	33.6
1	85	66	29	0	26.6
8	183	64	0	0	23.3
1	89	66	23	94	28.1
0	137	40	35	168	43.1
5	116	74	0	0	25.6
3	78	50	32	88	31
10	115	0	0	0	35.3
2	197	70	45	543	30.5
8	125	96	0	0	0
4	110	92	0	0	37.6
10	168	74	0	0	38
10	139	80	0	0	27.1
1	189	60	23	846	30.1
5	166	72	19	175	25.8
7	100	0	0	0	30
0	118	84	47	230	45.8
7	107	74	0	0	29.6
1	103	30	38	83	43.3
1	115	70	30	96	34.6
3	126	88	41	235	39.3
8	99	84	0	0	35.4
7	196	90	0	0	39.8
9	119	80	35	0	29
11	143	94	33	146	36.6
10	125	70	26	115	31.1
7	147	76	0	0	39.4
1	97	66	15	140	23.2
13	145	82	19	110	22.2
5	117	92	0	0	34.1
5	109	75	26	0	36
3	158	76	36	245	31.6
3	88	58	11	54	24.8
6	92	92	0	0	19.9
10	122	78	31	0	27.6
4	103	60	33	192	24
11	138	76	0	0	33.2
9	102	76	37	0	32.9
2	90	68	42	0	38.2
4	111	72	47	207	37.1
3	180	64	25	70	34
7	133	84	0	0	40.2
7	106	92	18	0	22.7
9	171	110	24	240	45.4
7	159	64	0	0	27.4

769 rows x 9 columns

OVERVIEW OF THE PROCESS:

Building an AI-based diabetes prediction system involves several key steps. Here's an overview of the process:

1. *Data Collection:*

- Gather a comprehensive dataset that includes relevant health and demographic information from individuals.

- Ensure the data is accurate, properly labeled, and covers a diverse population.

2. *Data Preprocessing:*

- Clean the data by addressing missing values, outliers, and inconsistencies.

- Normalize or standardize features to ensure they are on a similar scale.

- Handle class imbalance if present, as diabetes prediction datasets often have more non-diabetic cases.

3. *Feature Engineering:*

- Create or modify features to capture valuable information related to diabetes risk. This may include BMI, blood pressure categories, age groups, and more.

4. *Feature Selection:*

- Choose the most relevant features using various methods, such as filter, wrapper, or embedded techniques, as well as domain knowledge.

5. *Data Split:*

- Divide the dataset into training, validation, and test sets. Common splits are 70-80% for training, 10-15% for validation, and 10-15% for testing.

6. *Model Selection:*

- Choose a suitable machine learning algorithm for binary classification, such as logistic regression, decision trees, random forests, support vector machines, or deep learning models.

7. *Model Training:*

- Train the selected model on the training dataset, tuning hyperparameters to optimize performance.

- Implement k-fold cross-validation to assess generalization performance and prevent overfitting.

8. *Model Evaluation:*

- Use the test set to evaluate the model's predictive accuracy using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

9. *Model Deployment:*

- Deploy the trained model in a healthcare or clinical setting, ensuring compliance with medical and privacy regulations.

The process of building an AI-based diabetes prediction system is iterative and requires collaboration with healthcare professionals and domain experts to ensure its effectiveness and ethical use. Additionally, compliance with healthcare regulations, data privacy standards, and ethical guidelines is crucial throughout the development and deployment phases.

PROCEDURE:

Feature selection:

Feature selection for an AI-based diabetes prediction system involves choosing the most relevant variables or features from your dataset. Here are some steps specific to building a diabetes prediction system:

1. *Data Collection:* Gather a dataset that contains relevant information for diabetes prediction. This may include features like age, BMI, blood pressure, family history, glucose levels, and more.

2. *Data Preprocessing:* Clean the data by handling missing values, outliers, and any inconsistencies. Ensure that your data is properly labeled to distinguish between individuals with and without diabetes.

4. *Feature Selection Methods:* Choose appropriate feature selection methods for your diabetes prediction task:

a. *Filter Methods:* Use statistical measures like correlation coefficients or chi-squared tests to rank the importance of each feature in relation to the target variable (diabetes).

b. *Wrapper Methods:* Employ techniques like forward selection, backward elimination, or recursive feature elimination with a diabetes prediction model to determine the best subset of features.

c. *Embedded Methods:* Some machine learning algorithms for classification tasks, such as decision trees or random forests, can provide feature importance scores. You can use these scores to select relevant features.

d. *Domain Knowledge:* Leverage domain expertise to identify features known to be strongly associated with diabetes, like fasting blood sugar levels or insulin resistance.

6. *Model Selection:* Choose a machine learning algorithm suitable for classification tasks, such as logistic regression, support vector machines, or decision trees.

7. *Model Building:* Train and fine-tune your selected model using the subset of features you've chosen.

9. *Deployment:* Deploy the diabetes prediction system in a healthcare or clinical setting, ensuring compliance with all relevant regulations and data privacy standards.

Feature selection is critical to improve model accuracy and reduce overfitting, especially in cases like diabetes prediction, where not all available features may be equally informative. Remember that the choice of features and the quality of data play a significant role in the effectiveness of your AI-based diabetes prediction system.

Feature selection:

Pearson's correlation method is a popular method to find the most relevant attributes/features. The correlation coefficient is calculated in this method, which correlates with the output and input attributes. The coefficient value remains in the range between −1 and 1. The value above 0.5 and below −0.5 indicates a notable correlation, and the zero value means no correlation. In Weka, the correlation filter is used to find the correlation coefficient, and the results are shown in Table 3. We used 0.2 as a cut-off for relevant attributes. Hence SkinThickness, BP, DPF features are removed. Glucose, BMI, Insulin, Preg, and Age are our most relevant five input attributes.

The correlation between input and output attributes.

Attributes	**Correlation coefficient**
Glucose	0.484

Attributes	Correlation coefficient
BMI	0.316
Insulin	0.261
Preg	0.226
Age	0.224
SkinThickness	0.193
BP	0.183
DPF	0.178

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
RESET = "\033[0m"

df = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
```

# DATA CLEANING

```
  print(BLUE + "\nDATA CLEANING" + RESET)
```

Check for missing values

```
missing_values = df.isnull().sum()
print(GREEN + "Missing Values : " + RESET)
print(missing_values)
```

Handle missing values

mean_fill = df.fillna(df.mean())

df.fillna(mean_fill, inplace=True)

Check for duplicate values

```
    duplicate_values = df.duplicated().sum()
    print(GREEN + "Duplicate Values : " + RESET)
    print(duplicate_values)
```

Drop duplicate values

```
    df.drop_duplicates(inplace=True)
```

Support Vector Machine Modelling

```
print(BLUE + "\nMODELLING" + RESET)
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
# --- Splitting the data into training and testing sets
X_train, X_test, y_train, y_test =
   train_test_split(X, y, test_size=0.2,
   random_state=42
# --- Splitting the data into training and testing sets
X_train, X_test, y_train, y_test =
   train_test_split(X, y, test_size=0.2,
   random_state=42
)
# --- Standardize Features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# --- init and train SVM model
model = svm.SVC(kernel="linear")
model.fit(X_train, y_train)
# --- Predict on test data
y_pred = model.predict(X_test)
# --- Evaluate model performance
accuracy = model.score(X_test, y_test)
print(GREEN + "Model Accuracy : " + RESET)
print(accuracy)
# --- Classification Report and Confusion Matrix
print(GREEN + "Classification Report : " + RESET)
print(classification_report(y_test, y_pred))
print(GREEN + "Confusion Matrix : " + RESET)
cm = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
sns.heatmap(cm.confusion_matrix, annot=True, cmap="Blues")
plt.show()
print("Displayed")

# SAVING THE FILE
df.to_csv("/kaggle/working/cleaned_diabetes.csv", index=False)
print(BLUE + "\nDATA SAVING" + RESET)
print(GREEN + "Data Cleaned and Saved !" + RESET)
print("\n")
```

DATA CLEANING

Missing Values :

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

Duplicate Values :

```
0
```

DATA ANALYSIS

Summary Statistics :

```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

Class Distribution :

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

MODELLING

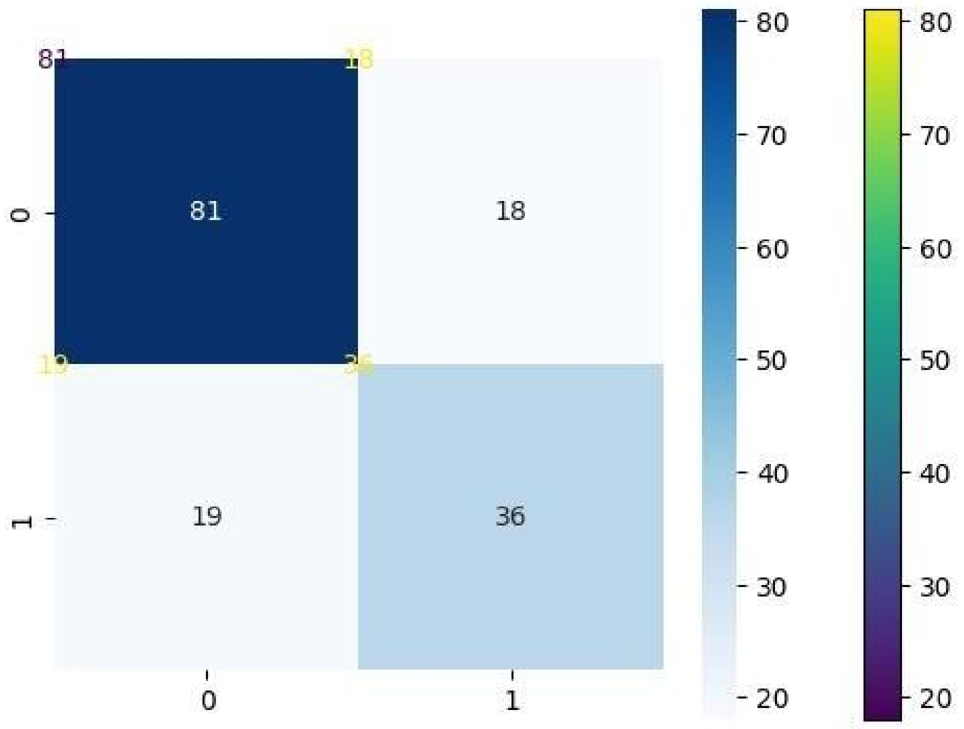
Model Accuracy :

```
0.7597402597402597
```

Classification Report :

```
              precision    recall  f1-score   support

           0       0.81      0.82      0.81        99
           1       0.67      0.65      0.66        55

    accuracy                           0.76       154
   macro avg       0.74      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154
```

Confusion Matrix :



Model Training:

Training an AI-based diabetes prediction system typically involves the following steps:

1. Data Collection: Gather a large and diverse dataset that includes relevant features such as patient demographics, medical history, lifestyle factors, and lab test results. The dataset should also include information about whether each individual has diabetes or not.

2. Data Preprocessing: Clean the data by handling missing values, outliers, and formatting issues. Normalize or standardize the data to ensure that different features have the same scale.

3. Feature Selection: Choose the most relevant features that can contribute to diabetes prediction. Feature engineering may also be performed to create new features based on domain knowledge.

4. Model Selection: Choose an appropriate machine learning or deep learning model for the prediction task. Common choices include logistic regression, decision trees, support vector machines, or neural networks.

5. Model Training: Split the dataset into training and testing sets to evaluate model performance. Train the chosen model on the training data, adjusting hyperparameters and optimizing it for the task.

6. Evaluation: Use evaluation metrics such as accuracy, precision, recall, F1 score, and area under the ROC curve to assess the model's performance on the test data. Cross-validation can also be employed for a more robust evaluation.

7. Fine-tuning: Based on the evaluation results, fine-tune the model by adjusting hyperparameters, trying different algorithms, or increasing the dataset size if necessary.

8. Deployment: Once the model performs well, deploy it in a clinical setting or as part of a diabetes management system. This may involve integrating the model into a mobile app or a web platform.

9. Monitoring and Maintenance: Continuously monitor the model's performance and update it as needed to adapt to changes in data distribution or medical knowledge.

10. Privacy and Ethics: Be mindful of patient data privacy and ethical considerations, such as informed consent and data security.

It's important to collaborate with healthcare professionals to ensure that the AI system aligns with medical standards and can provide valuable insights for diabetes management and prediction.

Model evaluation:

Evaluating an AI-based diabetes prediction system is crucial to ensure its effectiveness and reliability. Here are some common evaluation metrics and methods:

1. *Accuracy*: Calculate the percentage of correctly predicted cases. However, accuracy can be misleading if the dataset is imbalanced.

2. *Precision and Recall*: These metrics are useful when dealing with imbalanced datasets. Precision measures the ratio of true positive predictions to all positive predictions, while recall measures the ratio of true positive predictions to all actual positive cases.

3. *F1 Score*: It combines precision and recall into a single metric, providing a balance between the two. It's especially useful when you need to strike a balance between false positives and false negatives.

4. *Receiver Operating Characteristic (ROC) Curve*: Plot the true positive rate against the false positive rate to assess the model's ability to distinguish between classes. The area under the ROC curve (AUC) is also a valuable metric.

5. *Confusion Matrix*: This matrix provides a detailed breakdown of true positives, true negatives, false positives, and false negatives.

6. *Cross-Validation*: Use techniques like k-fold cross-validation to assess the model's performance on different subsets of the data.

7. *Feature Importance Analysis*: Determine which features have the most significant impact on predictions. This can help in feature selection and model improvement.

8. *Domain Expert Review*: Have domain experts evaluate the model's predictions and provide feedback on its clinical relevance and accuracy.

9. *External Validation*: Test the model on new, unseen data to assess its generalizability.

10. *Ethical and Fairness Evaluation*: Check for biases in the model's predictions and ensure it doesn't disproportionately affect certain demographic groups.

11. *Continuous Monitoring*: After deployment, continuously monitor the model's performance and update it as needed.

Remember that the choice of evaluation metrics and methods should align with the specific goals and requirements of the diabetes prediction system and its intended use in a clinical setting. Additionally, it's important to involve healthcare professionals in the evaluation process to ensure the model meets clinical standards and is safe for patients.

Feature Engineering:

Feature engineering is the process of creating new features or modifying existing ones to improve the performance of an AI-based diabetes prediction system. Here are some feature engineering ideas for such a system:

1. *Body Mass Index (BMI):* Calculate BMI from height and weight features if not already present in your dataset. High BMI is often associated with diabetes risk.

2. *Waist-to-Hip Ratio:* Create a feature that represents the waist-to-hip ratio, which can be an indicator of abdominal obesity and diabetes risk.

3. *Age Groups:* Group individuals into different age categories (e.g., young adults, middle-aged, seniors) to capture age-related diabetes trends.

4. *Blood Pressure Categories:* Categorize blood pressure readings as normal, prehypertension, or hypertension to incorporate the impact of hypertension on diabetes risk.

5. *Glucose Metrics:* Create aggregated metrics for glucose levels, such as the average glucose level over a specific time period or the variation in glucose levels.

6. *Family History Score:* Assign scores based on family history of diabetes,where a higher score indicates a stronger family history of the disease.

7. *Dietary Patterns:* If dietary data is available, engineer features to capture dietary patterns like low-carb, high-fiber, or high-sugar diets.

8. *Physical Activity Levels:* Create a metric that quantifies the level of physical activity for each individual, considering factors like daily steps, exercise frequency, and sedentary behavior.

9. *Sleep Patterns:* Incorporate features related to sleep duration and quality, as poor sleep habits can impact diabetes risk.

10. *Medication History:* If relevant, include features that track an individual's history of diabetes medications or insulin use.

11. *HbA1c Trends:* Calculate trends in HbA1c values over time if your dataset includes multiple measurements for each individual.

12. *Fasting Hours:* Create a feature representing the number of fasting hours before a glucose measurement, which can impact glucose levels.

13. *Microbiome Data:* If available, incorporate features related to gut microbiome composition, as emerging research suggests a link between the microbiome and diabetes.

14. *Social and Economic Factors:* Consider features related to an individual's socioeconomic status, access to healthcare, and lifestyle, as these can influence diabetes risk.

15. *Geographic Factors:* If data includes location information, features related to geographic factors like climate, pollution, or access to healthy food options can be relevant.

16. *Interaction Terms:* Create interaction terms between pairs of features to capture potential synergistic effects, e.g., age multiplied by BMI.

17. *Time-based Features:* Incorporate time-related features like the season, day of the week, or holidays, as they might influence eating and exercise patterns.

18. *Composite Scores:* Develop composite scores that combine multiple features to represent an individual's overall risk, such as a diabetes risk score.

Remember to validate the new features through cross-validation and assess their impact on the performance of your diabetes prediction model. Feature engineering is an iterative process, and domain knowledge plays a crucial role in deciding which features are most relevant to the problem.

Conclusion:

In conclusion, the development of an AI-based diabetes prediction system holds significant potential for improving healthcare outcomes related to diabetes. By leveraging artificial intelligence techniques, such as machine learning and data analysis, the system can provide accurate risk assessments and enable early intervention and prevention strategies.

The AI-based diabetes prediction system offers several advantages. Firstly, it can assist healthcare professionals in identifying individuals at high risk of developing diabetes, allowing for targeted interventions and personalized care plans. Secondly, it can contribute to reducing healthcare costs by focusing resources on prevention and early detection, which can help mitigate the long- term complications associated with diabetes. Additionally, the system can empower individuals to take protective steps towards managing their health and making informed lifestyle choices