AI BASED DIABETES PREDICTION SYSTEM

PROJECT TITLE	AI based diabetes prediction system
SKILLS TAKEN AWAY	❖ Python script ❖ EDA ❖ UI deployment
DOMAIN	Medical

INTRODUCTION:

Diabetes is a health issue that affects how your body converts food into energy. The majority of the food you consume is broken down into sugar (also known as glucose) and released into your bloodstream. When your blood sugar rises, your pancreas releases insulin. Diabetes, if not managed carefully and continuously, can cause a buildup of sugars in the blood, increasing the risk of serious consequences such as stroke and heart disease. As a result, I decided to anticipate using Machine Learning in Python.

OBJECTIVES:

- Determine whether or not a person has diabetes.
- Discover the most telling signs of diabetes and experiment with several classification methods to get the best accuracy.

DETAILS ABOUT THE DATASET:

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

- **Pregnancies**: Number of times pregnant
- **Glucose**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **Blood Pressure**: Diastolic blood pressure (mm Hg)

- **Skin Thickness**: Triceps skin fold thickness (mm)
- **Insulin**: 2-Hour serum insulin (mu U/ml)
- **BMI**: Body mass index (weight in kg/(height in m)^2)
- **Diabetes Pedigree Function**: Diabetes pedigree function
- **Age**: Age (years)
- **Outcome**: Class variable (0 or 1)

Number of Observation Units: 768

Variable Number: 9

LIBRARY INSTALLATION:

- In this initial stage, I loaded the most often used Python libraries for machine learning, such as Pandas, Seaborn, Matplotlib, and others.
- Python is the most flexible and effective programming language I've ever used. I also used Python in the software development area.

```
#import libraries
import numpy as np #linear algebric operations on numbers
import pandas as pd # data processing (example csv,excel,json,text,xml etc.)
import matplotlib.pyplot as plt #To plot charts and for data visualization
import seaborn as sns #To plot charts and for data visualization
import sklearn #For ML algorithms and models
from collections import Counter #count the number of elements
```

DATA IMPORTING:

In my project I used Pima Indians diabetes database from Kaggle

Original owners: National Institute of Diabetes and Digest Kidney Diseases

```
[ ] df=pd.read_csv('diabetes.csv')
    df.head(10)
```

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

DATA PREPROCESSING:

```
# Size of the dataset(number of rows and columns)
df.shape
```

```
(768, 9)
```

```
df.info()
```

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

MISSING VALUES ANALYSIS:

```
#Missing value analysis
df.isnull().sum()
```

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

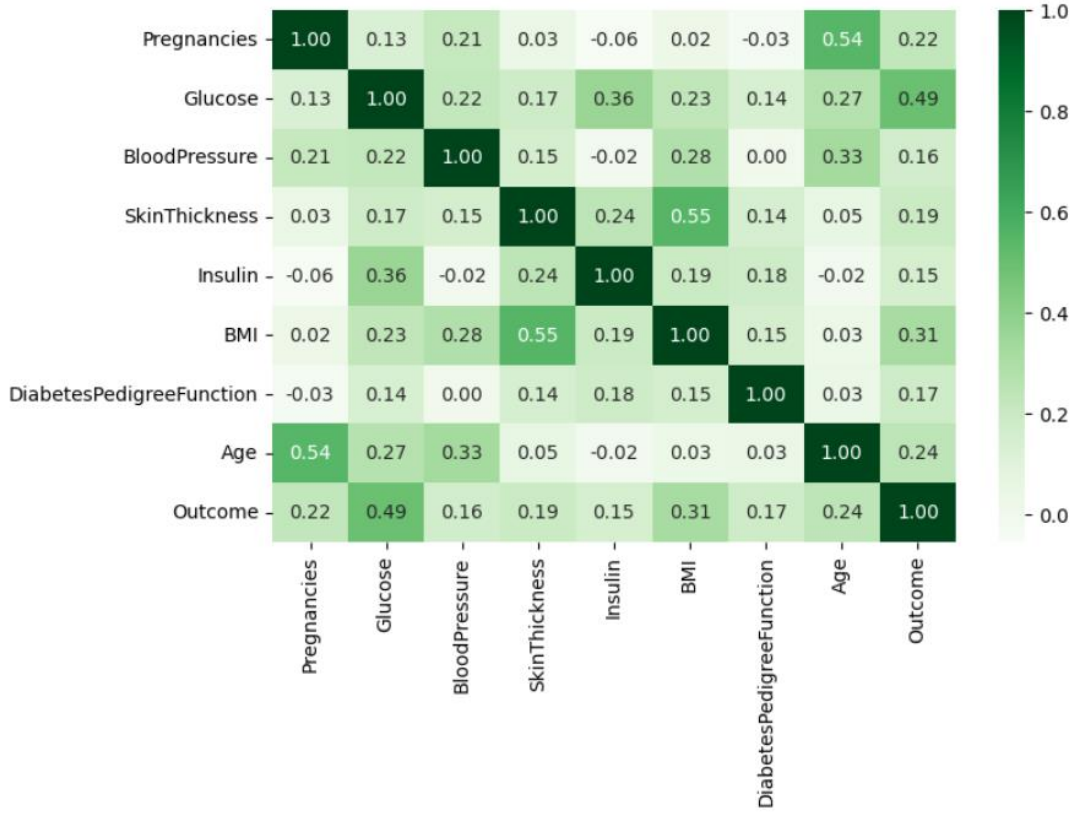
I observed that there is no missing values in dataset however the features like Glucose, Blood Pressure, Insulin, Skin Thickness has 0 values which is not possible. We have to replace 0 values with either mean or median values of specific column.

```
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())
```

EXPLORATORY DATA ANALYSIS:

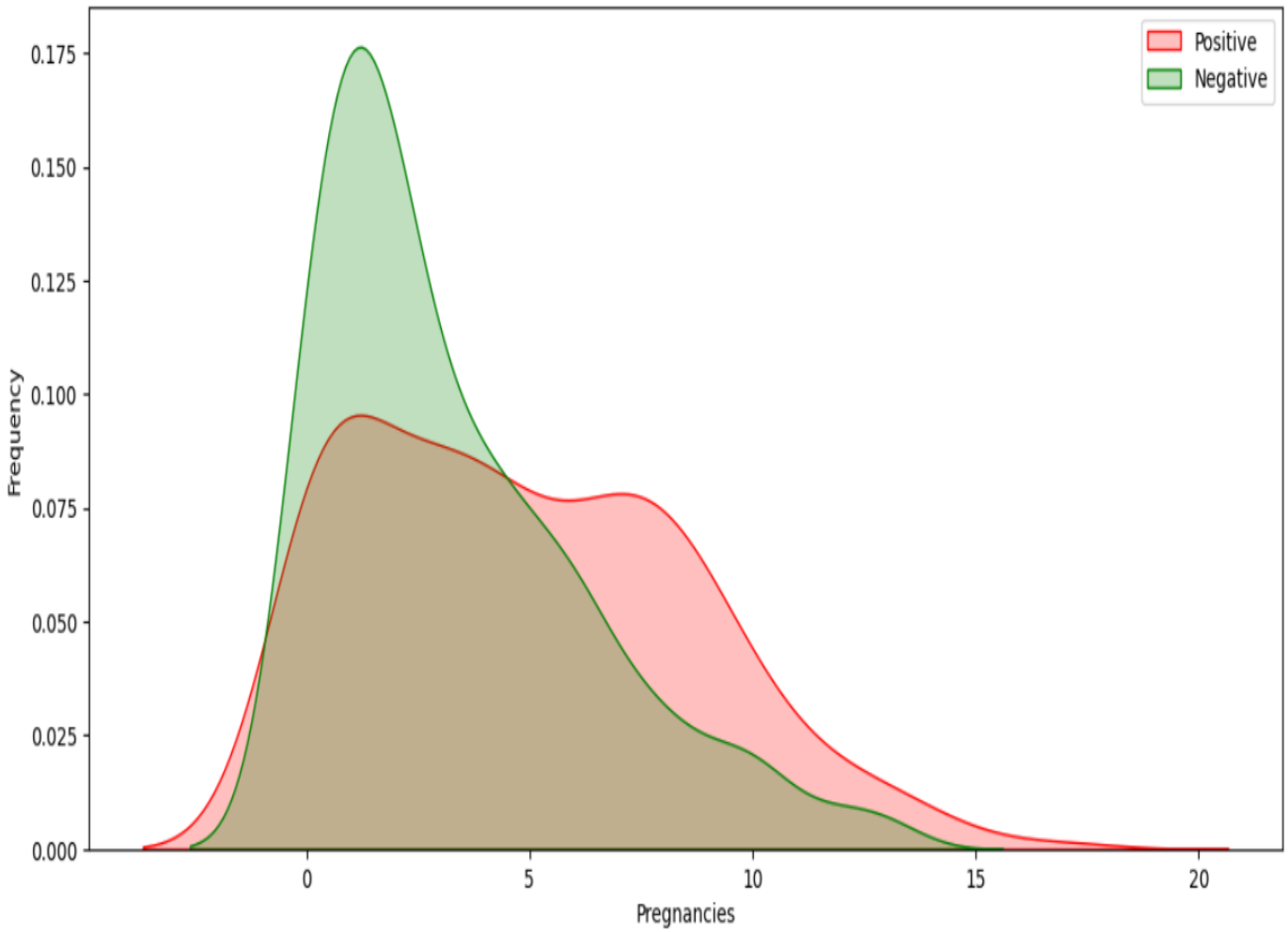
```
# Exploratory data analysis
plt.figure(figsize=(8,5))
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap = "Greens")
```

<Axes: >




```
[13] # Explore Pregnancies vs Outcome
     plt.figure(figsize=(13,6))
     g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1],
         color="Red", shade = True)
     g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0],
         ax =g, color="Green", shade= True)
     g.set_xlabel("Pregnancies")
     g.set_ylabel("Frequency")
     g.legend(["Positive","Negative"])
```

```
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0],
<matplotlib.legend.Legend at 0x7b5896ba80d0>
```



OUTLIER OBSERVATION ANALYSIS:

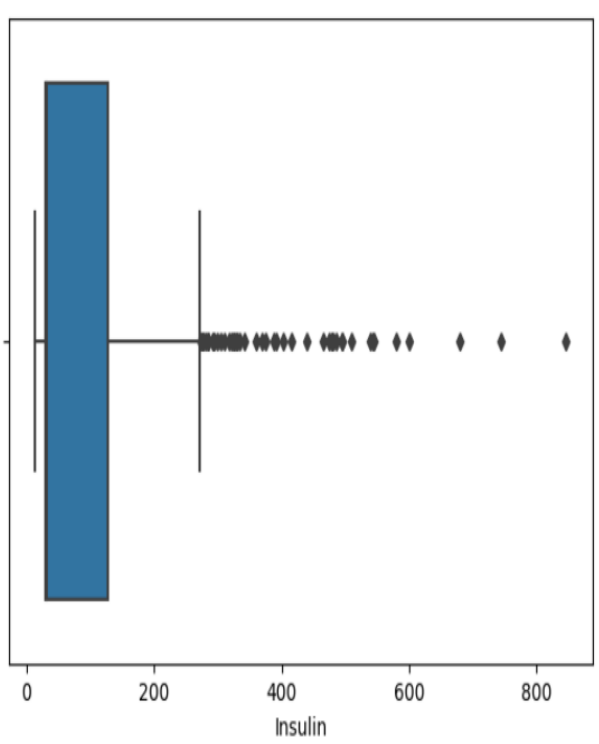
In the data set I am checking is there is any outlier observations compared to the 25% and 75% quarters. It was found to be an outlier observation.

```
for feature in df:


    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1- 1.5*IQR
    upper = Q3 + 1.5*IQR


    if df[(df[feature] > upper)].any(axis=None):
        print(feature,"yes")
    else:
        print(feature, "no")
```

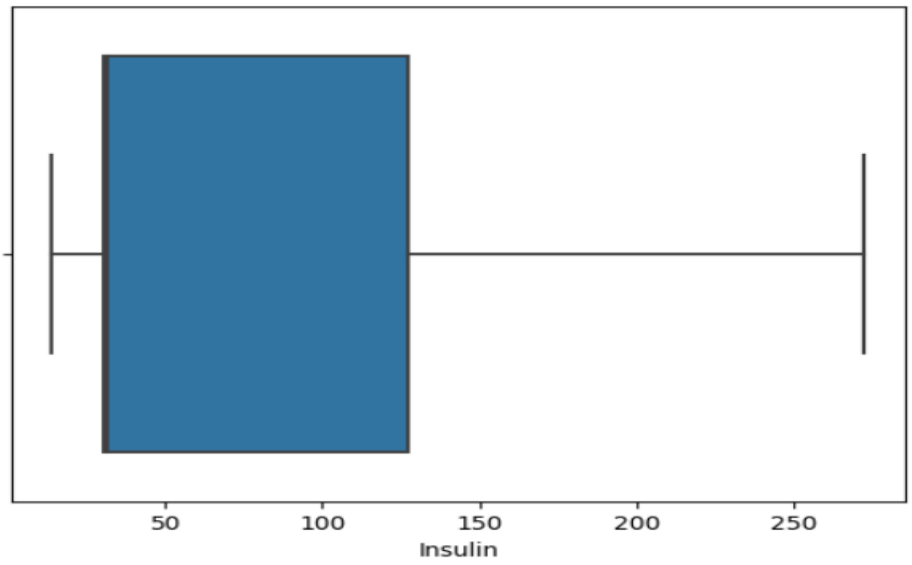
```
Pregnancies yes
Glucose no
BloodPressure yes
SkinThickness yes
Insulin yes
BMI yes
DiabetesPedigreeFunction yes
Age yes
Outcome no
```

The process of visualizing the Insulin variable with boxplot method was done. We find the outlier observations on the chart.

```
sns.boxplot(x = df["Insulin"]);
```




```
#We conduct a stand alone observation review for the Insulin variable
#We suppress contradictory values
Q1 = df.Insulin.quantile(0.25)
Q3 = df.Insulin.quantile(0.75)
IQR = Q3-Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
df.loc[df["Insulin"] > upper,"Insulin"] = upper
sns.boxplot(x = df["Insulin"]);
```



LOCAL OUTLIER FACTOR (LOF):

```python
from sklearn.neighbors import LocalOutlierFactor
lof =LocalOutlierFactor(n_neighbors= 10)
lof.fit_predict(df)
```

```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
       -1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,
       -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1])
```

```
df_scores = lof.negative_outlier_factor_
np.sort(df_scores)[0:30]
```

```
array([-2.97833873, -2.13812857, -2.08004061, -2.07164162, -1.85239939,
       -1.84620717, -1.81951148, -1.81547501, -1.75035734, -1.74379618,
       -1.74124408, -1.72551826, -1.72366681, -1.71639102, -1.61537791,
       -1.61178248, -1.60541668, -1.56976852, -1.56525664, -1.5466146 ,
       -1.52468978, -1.52075927, -1.50902437, -1.50546835, -1.50310522,
       -1.49949252, -1.48775564, -1.47998361, -1.47411255, -1.46775757])
```

We choose the threshold value according to LOF scores and We delete those that are higher than the threshold

```
threshold = np.sort(df_scores)[7]
threshold
```

```
-1.8154750080516624
```

```
[23] outlier = df_scores > threshold
     df = df[outlier]
```

The size of the data set was examined.

```
df.shape
```

```
(760, 9)
```

Number of Observation Units: 760

Variable Number: 9

From this we can infer that after completion of the data pre-processing, the size of the data has been reduced from **(768, 9) to (760, 9).** Here, we can see that the dataset has been organised, cleaned, and transformed so that it may be used for further analysis and to train a machine learning model.