# MERISKILL | Data Analyst | Internship | Project No: 02 | Diabetes Patients

## Problem Statement

## Project No# 2: Diabetes Patients

- This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.
- The objective of the dataset is to diagnostically predict whether a patient has diabetes based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.
- From the data set in the (.csv) File We can find several variables, some of them are independent (several medical predictor variables) and only one target dependent variable (Outcome).

**Project No 2: Title: Predicting Diabetes in Pima Indian Patients using Python**

The "Predicting Diabetes in Pima Indian Patients" project is aimed at developing a machine learning model that can predict the likelihood of diabetes in Pima Indian female patients who are at least 21 years old. This model uses diagnostic measurements to provide personalized predictions and assist in early diagnosis and preventive healthcare measures.

**Exploratory Data Analysis (EDA):** The project begins with an EDA to understand the dataset's characteristics and distribution. Visualizations, including histograms and scatterplots, offer insights into the distribution of variables like age, pregnancies, glucose, and insulin. The EDA also includes a correlation matrix heatmap to reveal relationships between variables.

**Data Preprocessing:** Feature scaling is applied to normalize the dataset, and the data is divided into training and testing sets. The choice of a Random Forest Classifier as the predictive model ensures robustness and reliability.

**Machine Learning Model:** The Random Forest Classifier is employed to predict diabetes. This ensemble learning technique excels in handling complex datasets, making it an excellent choice for this medical prediction task. The model's performance is evaluated using key metrics, including accuracy, precision, recall, and a confusion matrix.

**User Interaction:** One of the project's highlights is its interactive nature. A function is developed to allow users to input their medical predictor variables. The model then generates personalized predictions regarding the likelihood of diabetes. This interactive feature makes the project a valuable tool for personalized healthcare decision-making.

**Python Libraries:** Throughout the project, various Python libraries are utilized, including Numpy, Pandas, Matplotlib, Seaborn, and scikit-learn. These libraries streamline data analysis, visualization, preprocessing, and machine learning model development, making the code accessible and adaptable for broader healthcare applications.

**In summary, the "Predicting Diabetes in Pima Indian Patients" Project** is a compelling example of how data science and machine learning can be applied to address healthcare challenges. By leveraging diagnostic measurements, this project aids in the early detection of diabetes and empowers individuals with personalized risk assessments. It serves as a model for utilizing data-driven approaches to improve public health, particularly in populations with specific healthcare needs.

# Predicting Diabetes in Pima Indian Patients | Python Code & Output

## Importing Necessary Libraries

```
#Importing Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

✓ 0.0s Python

## Load the Diabetes Patients dataset

```
# Load the Diabetes Patients dataset
diabetes_data = pd.read_csv("diabetes.csv")
```

✓ 0.4s Python

## EDA and Visualization

```
diabetes_data.head()
```

✓ 0.0s Python

| | 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 |

```
diabetes_data.tail()
```

✓ 0.1s Python

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 763 | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 | 0 |
| 764 | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 | 0 |
| 765 | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 | 0 |
| 766 | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 | 1 |
| 767 | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 | 0 |

## Display basic information about the dataset

```
# Display basic information about the dataset
print("Diabetes Dataset Information: \n")
diabetes_data.info()
```

✓ 0.3s Python

```
Diabetes Dataset Information:

<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
```



## Statistical Summary

```
# Statistical Summary
print("\nStatistical Summary: \n")
diabetes_data.describe()
```

✓ 0.2s Python

Statistical Summary:

| | 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 |

## Shape for Dataset

```
# Shape for dataset
diabetes_data.shape
```

✓ 0.0s Python

```
(768, 9)
```

### Check for missing values

```
# Check for missing values
print("\nMissing Values:")
print(diabetes_data.isnull().sum())
```

✓ 0.1s Python

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

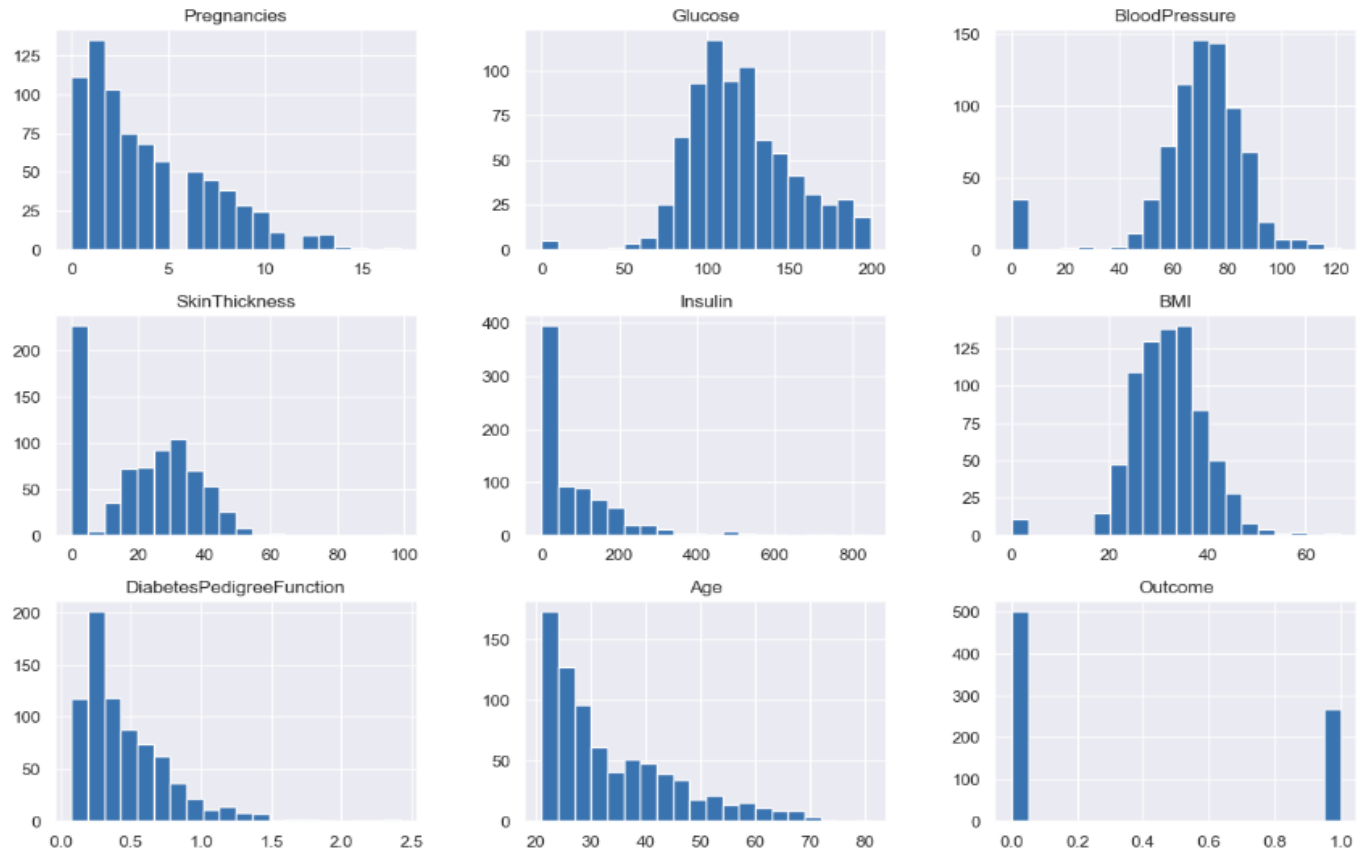
## Data Exploration and Analysis

- Let's start with some data exploration and visualization.

### Visualize the distribution of features

```
# Visualize the distribution of features
diabetes_data.hist(bins=20, figsize=(15, 10))
plt.show()
```

✓ 4.1s Python



## Check for Duplicated Values

```
# Check for Duplicated Values
print("Duplicated Values is:", diabetes_data.duplicated().sum())
```

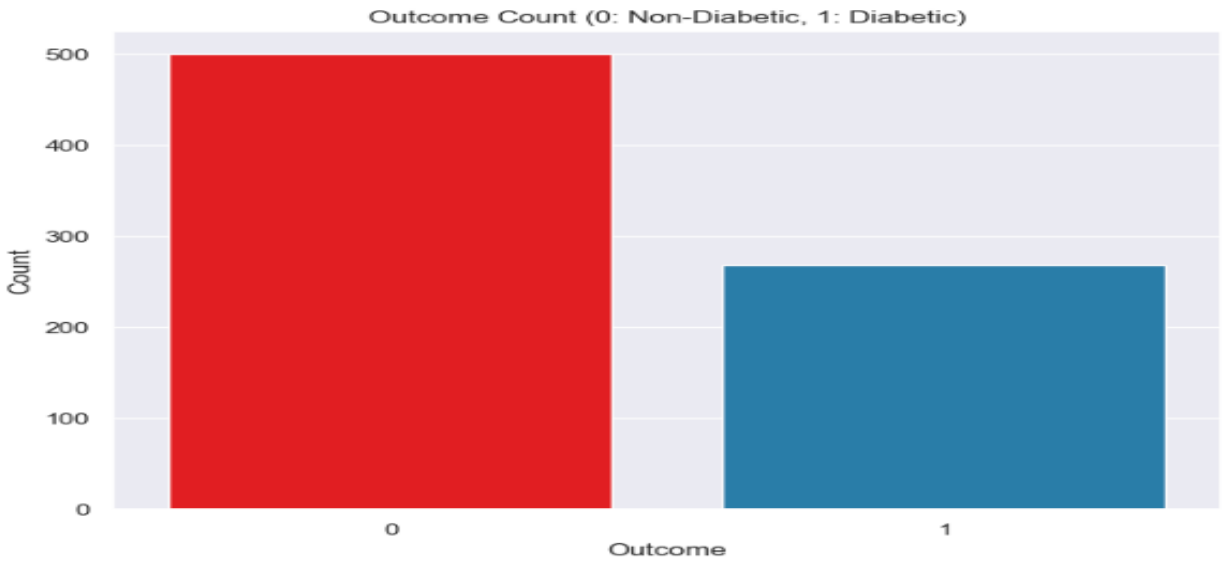
✓ 0.0s Python

```
Duplicated Values is: 0
```

## Check the Distribution of Outcome Feature

```
# Plot the count of Outcomes
sns.set(style="darkgrid")
plt.figure(figsize=(8, 6))
sns.countplot(data=diabetes_data, x='Outcome', palette='Set1')
plt.title('Outcome Count (0: Non-Diabetic, 1: Diabetic)')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.show()
```

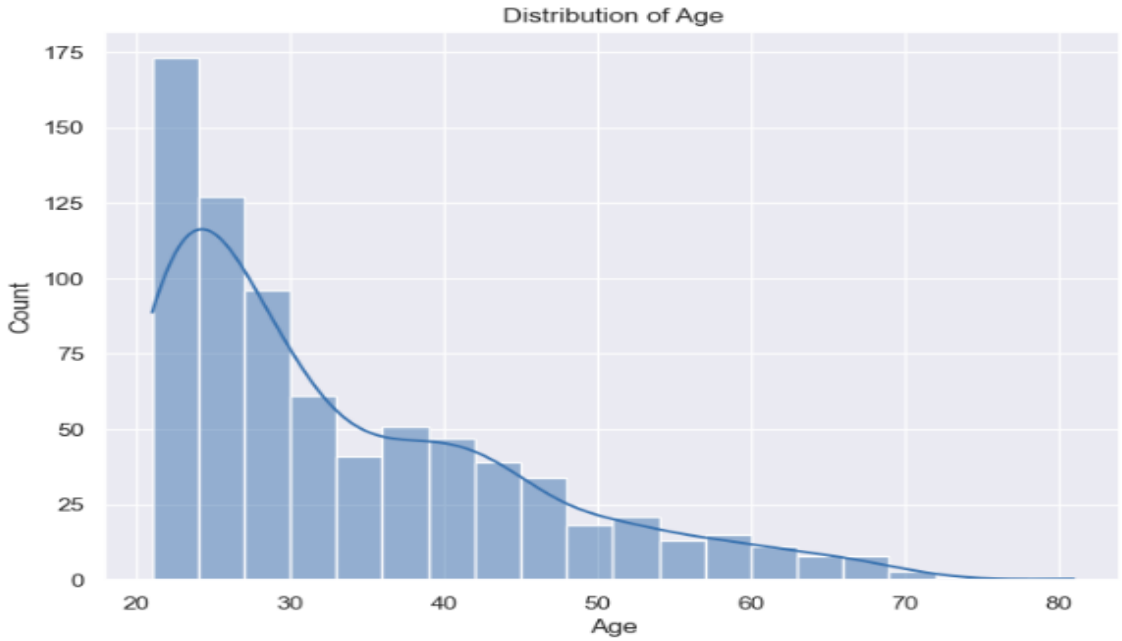
✓ 1.0s




## Distribution of Age

```
# Age distribution of Counts
plt.figure(figsize=(8, 6))
sns.histplot(data=diabetes_data, x='Age', bins=20, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

✓ 0.9s




## Scatter Plot Age and Pregnancies

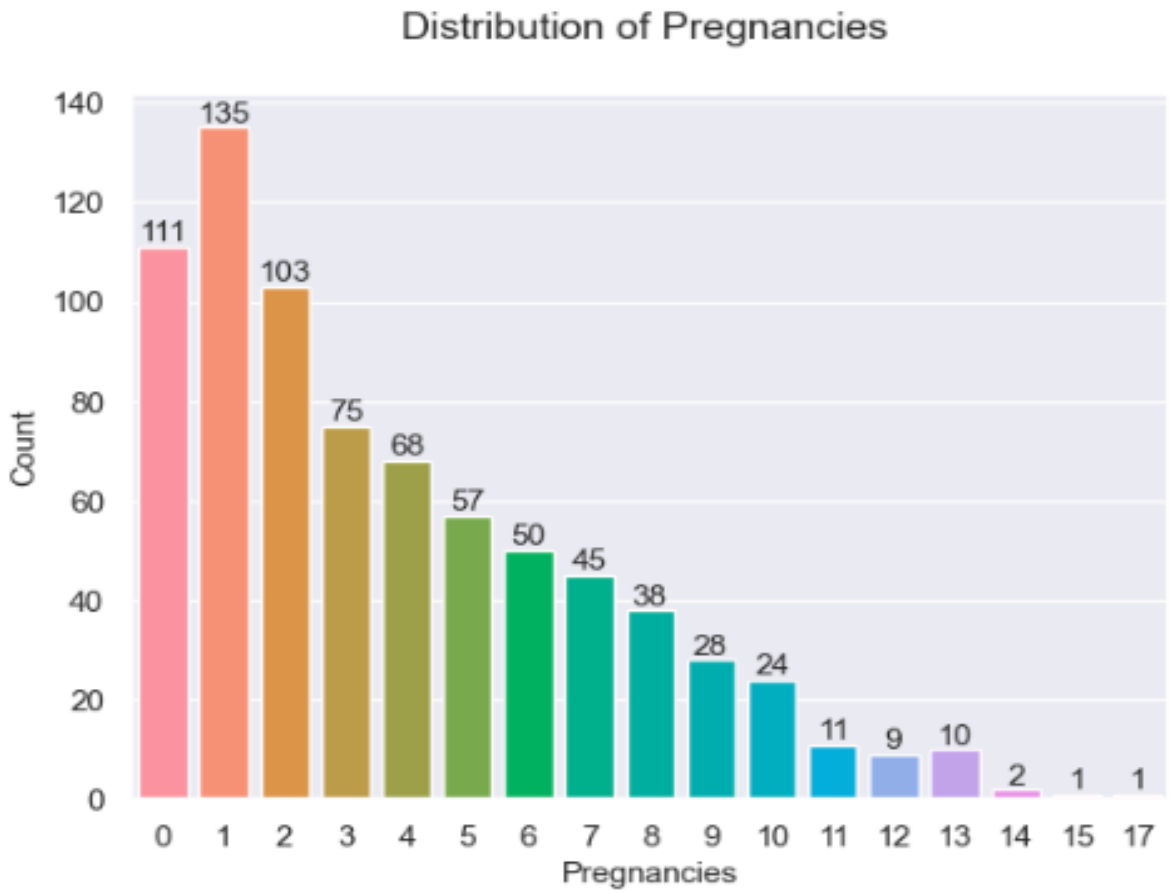
```
plt.figure(figsize=(8,8))
sns.scatterplot(x="Age", y="Pregnancies", hue="Outcome",
                data=diabetes_data)
plt.show()
```

✓ 0.5s

## Distribution of Pregnancies

```
# Distribution of Pregnancies
ax = sns.barplot(x=diabetes_data['Pregnancies'].value_counts().index,
                 y=diabetes_data['Pregnancies'].value_counts())
for bars in ax.containers:
    ax.bar_label(bars,size = 11)
plt.xlabel('Pregnancies', size = 11)
plt.ylabel('Count', size = 11)
plt.title('Distribution of Pregnancies \n', size = 14)
plt.show()
```

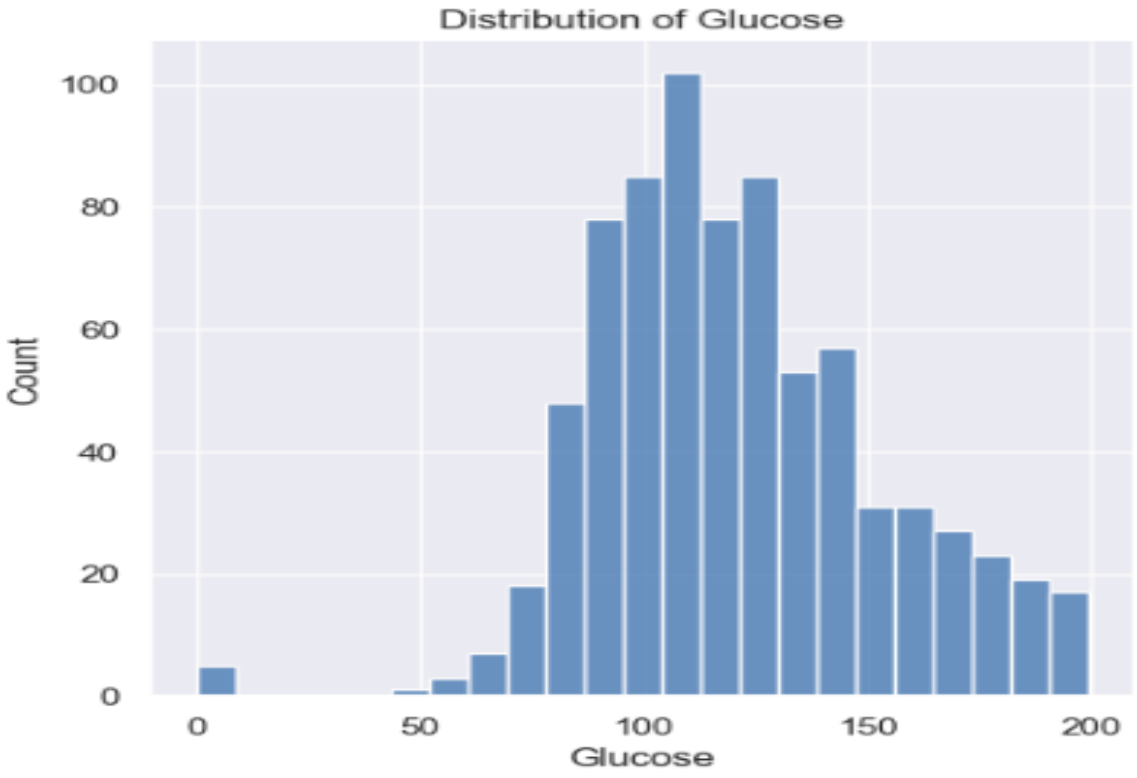
✓ 0.7s



## Distribution of Glucose

```
#  Distribution of Glucose
sns.displot(data=diabetes_data, x='Glucose')
plt.title('Distribution of Glucose')
plt.xlabel('Glucose')
plt.ylabel('Count')
plt.show()
```

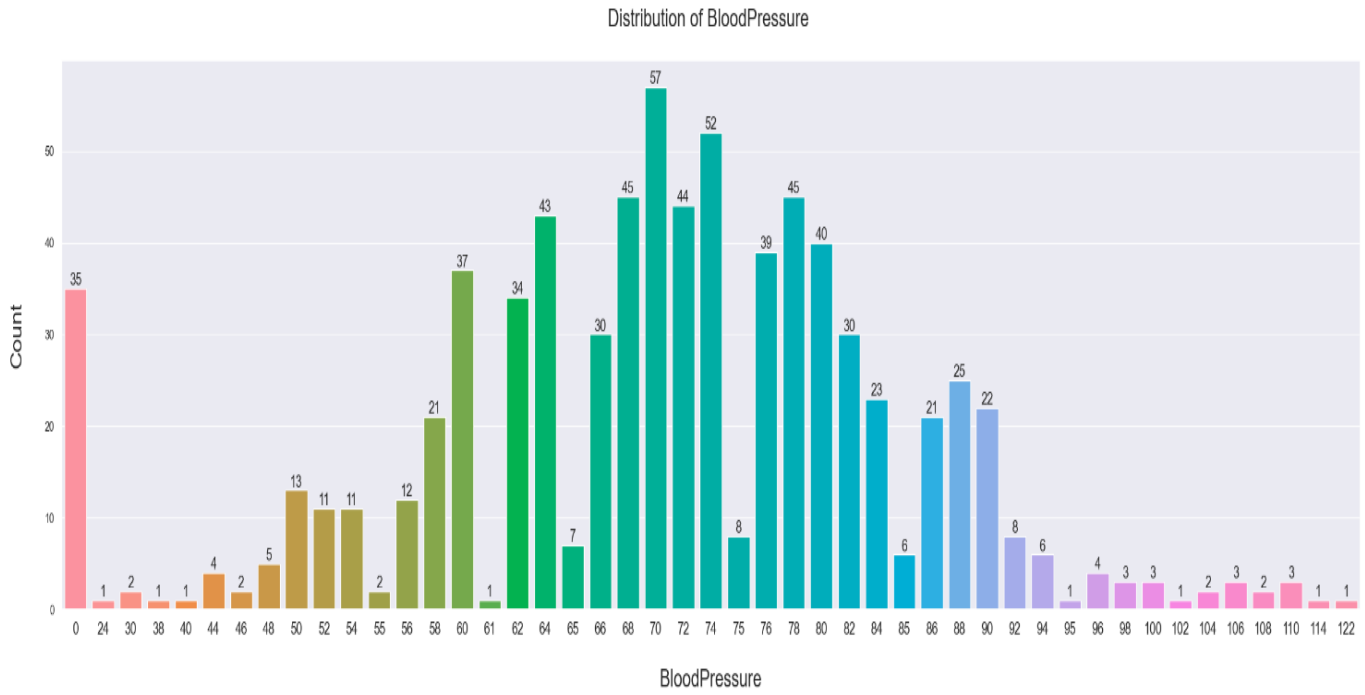
0.7s




## Distribution of BloodPressure

```
# Distribution of BloodPressure
plt.figure(figsize= (30, 8))
ax = sns.countplot(data =diabetes_data, x=diabetes_data.BloodPressure)
for bars in ax.containers:
    ax.bar_label(bars,size = 14)
plt.title('Distribution of BloodPressure \n', size= 20)
plt.xlabel("\nBloodPressure",size = 20)
plt.ylabel("Count\n",size =20)
plt.xticks(rotation = 0,size= 14)
plt.show()
```

1.5s



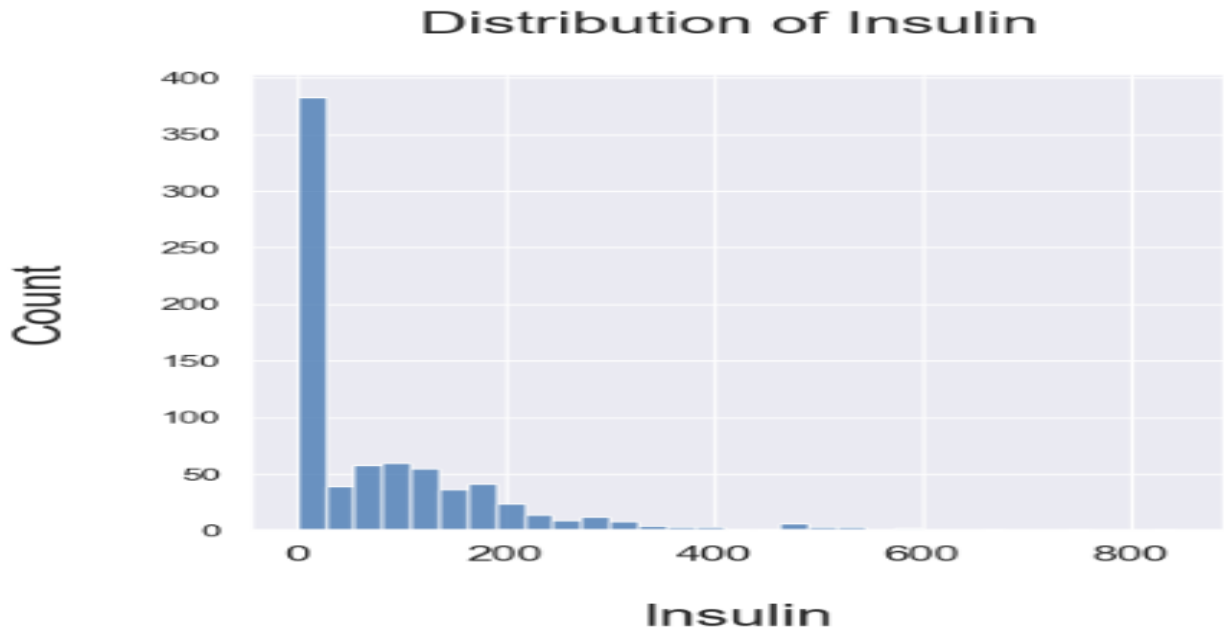

## Distribution of Insulin

```
# Distribution of Insulin

sns.displot(data =diabetes_data, x=diabetes_data.Insulin)

plt.title('Distribution of Insulin \n', size= 20)
plt.xlabel("\Insulin",size = 20)
plt.ylabel("Count\n",size =20)
plt.xticks(rotation = 0,size= 14)
plt.show()
```

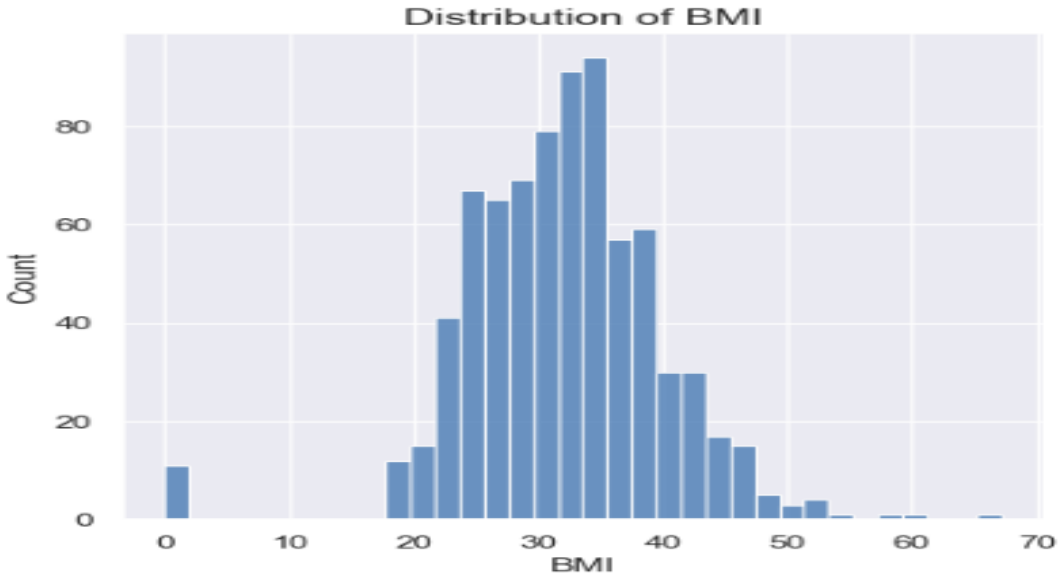
0.7s



## Distribution of BMI

```
# Distribution of BMI
sns.displot(diabetes_data, x="BMI")
plt.title("Distribution of BMI", size = 14)
plt.show()
```

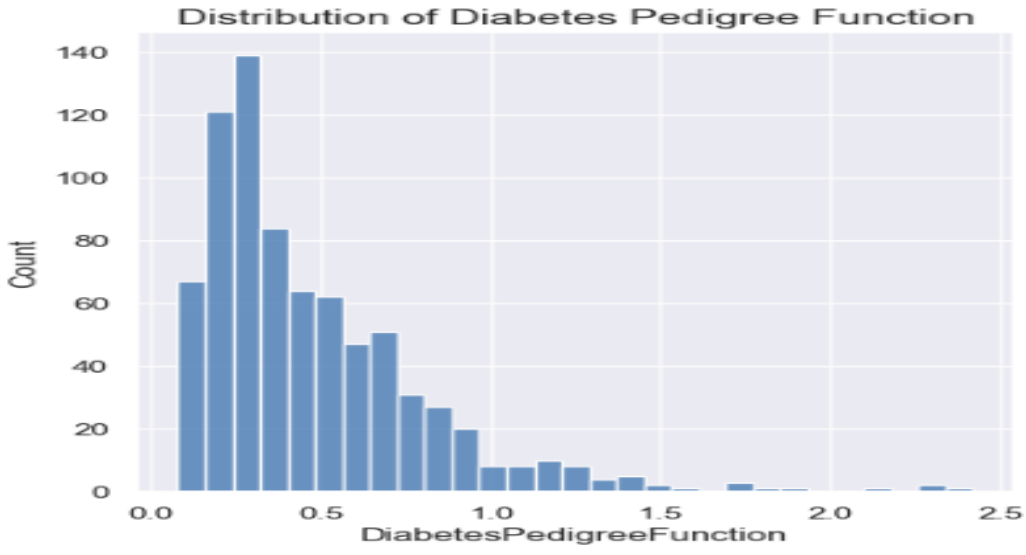
0.8s




## Distribution of Diabetes Pedigree Function

```
# Distribution of DiabetesPedigreeFunction
sns.displot(diabetes_data, x="DiabetesPedigreeFunction")
plt.title("Distribution of Diabetes Pedigree Function", size = 14)
plt.show()
```

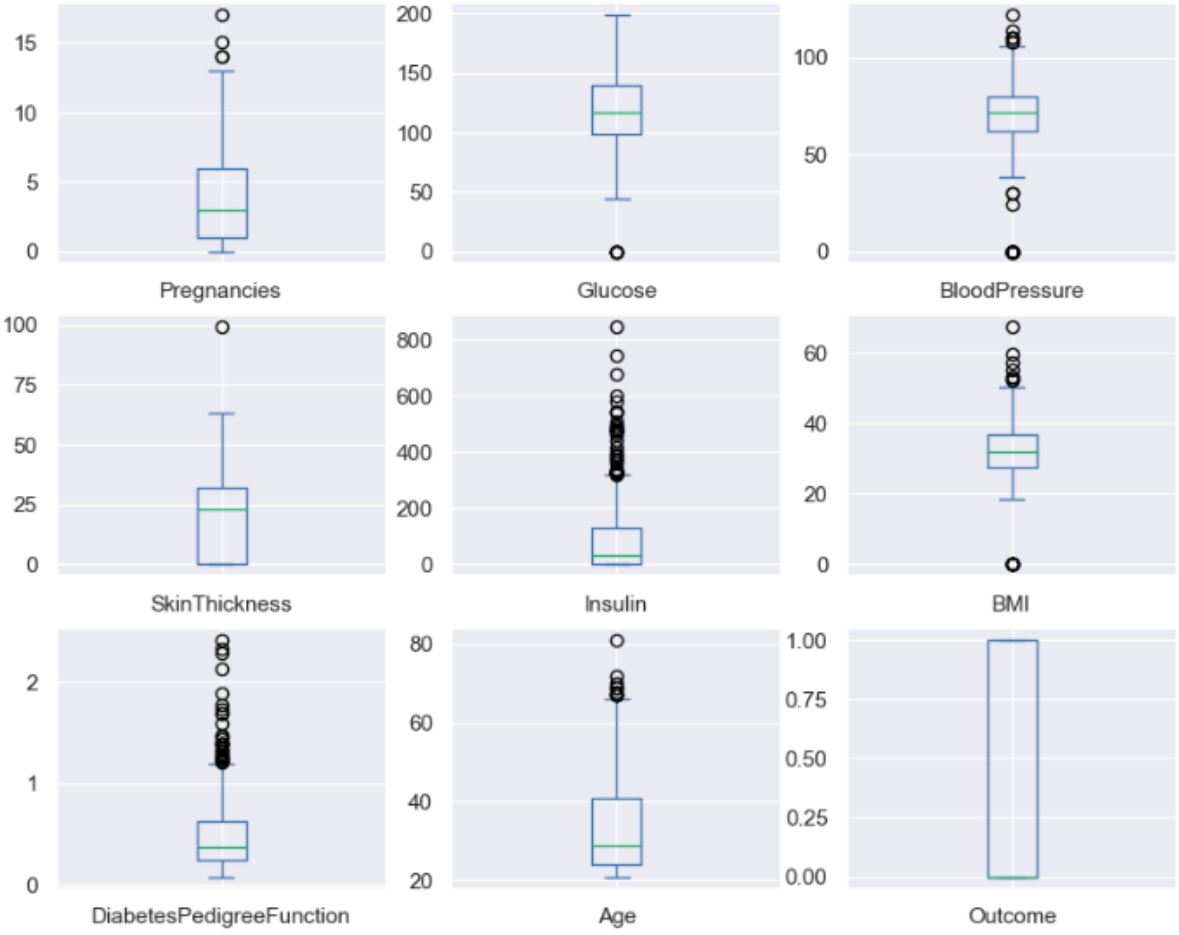
1.0s



## Checking Outliers in DataFrame

```
# Checking Outliers in DataFrame
diabetes_data.plot(kind = 'box',subplots = True, layout = (3,3),
                   sharex=False, sharey=False,figsize=(10,8))
plt.show()
```

✓ 2.0s




## Multivariate Analysis

```
# Multivariate Analysis
sns.pairplot(diabetes_data)
plt.show()
```

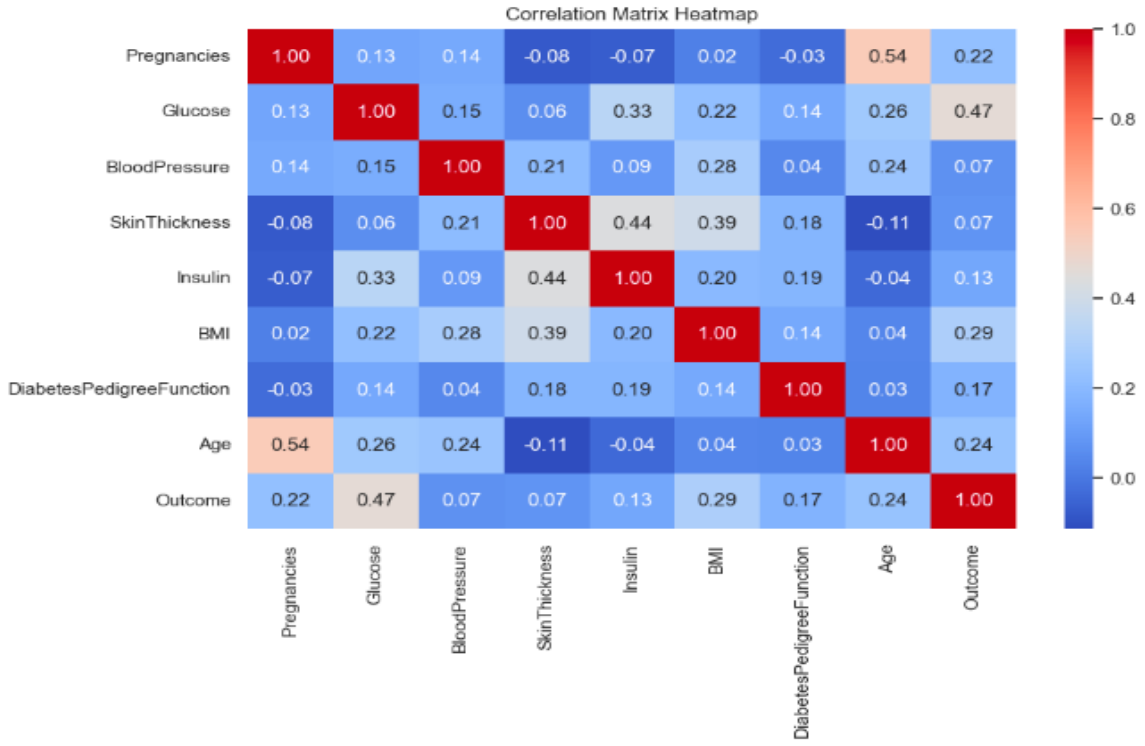
✓ 33.5s


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

## Correlation Matrix Heatmap

```
# Correlation Matrix Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(diabetes_data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()
```

✓ 1.1s




## Data Preprocessing

### Normalize and scale features

```
# Normalize and scale features
scaler = StandardScaler()
X = diabetes_data.drop('Outcome', axis=1)
y = diabetes_data['Outcome']
X = scaler.fit_transform(X)
```

✓ 0.0s

### Split the data into training and testing sets

```
# Split 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)
```

✓ 0.2s

```
print("Shape of X_train: ", X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of y_test: ", y_test.shape)
```

✓ 0.1s

```
Shape of X_train:  (614, 8)
Shape of X_test:  (154, 8)
Shape of y_train:  (614,)
Shape of y_test:  (154,)
```

## Train a Random Forest Classifier

```
# Train a Random Forest Classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

✓ 0.4s

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

## Make predictions

```
# Make predictions
y_pred = model.predict(X_test)
```

✓ 0.0s

```
print(y_pred)
```

✓ 0.1s

```
[0 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1 1 1
 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0
 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 0 1 0 1 0
 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0]
```

## Evaluate the model's performance

```
# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

✓ 0.1s

```
Accuracy: 0.7272727272727273
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

0.1s

```
Confusion Matrix:
 [[78 21]
 [21 34]]
```

```
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

0.1s

```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79        99
           1       0.62      0.62      0.62        55

    accuracy                           0.73       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.73      0.73      0.73       154
```

## Function to get user input and predict

```
# Function to get user input and predict
Comment Code
def predict_diabetes():
    print("\nEnter Medical Predictor variables:\n")
    predictor_vars = []
    for col in diabetes_data.columns[:-1]:
        value = float(input(f"{col}: "))
        predictor_vars.append(value)

    # Create a new input in the same format as the training data
    new_input = pd.DataFrame([predictor_vars], columns=diabetes_data.columns[:-1])
    new_input = scaler.transform(new_input)

    # Predict the outcome
    prediction = model.predict(new_input)
    if prediction[0] == 1:
        print("Prediction of Diabeties Patients: Diabetic")
    else:
        print("Prediction of Diabeties Patients: Non-Diabetic")
```

0.1s

## Allow the user to make predictions

```
# Allow the user to make predictions
while True:
    predict_diabetes()
    again = input("Do you want to predict again? (yes/no): ")
    if again.lower() != "yes":
        break
```

1m 1.9s

```
Enter Medical Predictor variables:
Prediction of Diabeties Patients: Non-Diabetic

Enter Medical Predictor variables:
Prediction of Diabeties Patients: Diabetic
```