Pima Indian Diabetes Prediction

The aim of this project to analyze the medical factors of a patient such as Glucose Level, Blood Pressure, Skin Thickness, Insulin Level and many others to predict whether the patient has diabetes or not.

About the Dataset 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 or not 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.

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.

Data Dictionary

Feature	Description
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	Triceps skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	Body mass index (weight in kg/(height in m)^2)
DiabetesPedigreeFunction	Diabetes pedigree function
Age	Age (years)
Outcome	Class variable (0 or 1)

Impact

The Pima Indian Diabetes Prediction project holds significant potential for impacting healthcare outcomes for Pima Indian females. Early detection and diagnosis of diabetes can play a crucial role in managing the condition effectively and preventing complications. By developing an accurate predictive model, healthcare providers can identify individuals at higher risk of diabetes and offer timely interventions and personalized treatment plans. This project's successful implementation may lead to improved health management strategies, reduced healthcare costs, and enhanced overall well-being for the Pima Indian female community. Additionally, the insights gained from this study may contribute to broader research on diabetes risk factors and aid in formulating targeted public health initiatives for diabetes prevention and awareness within the Pima Indian population. The ethical and responsible use of data in this project will be ensured to protect patient privacy and promote transparency in the application of predictive modeling in healthcare settings.

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

```
df=pd.read_csv('diabetes.csv')
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
df.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

```
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
```

```
df.shape
```

```
(768, 9)
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcom
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.34895
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.47695
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.00000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.00000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.00000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.00000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.00000

```
df.nunique()
```

```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

Checking the unique values for each variable in the dataset

```
#checking unique values
variables = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']
for i in variables:
    print(df[i].unique())
```

```
[ 6  1  8  0  5  3 10  2  4  7  9 11 13 15 17 12 14]
[148  85 183  89 137 116  78 115 197 125 110 168 139 189 166 100 118 107
 103 126  99 196 119 143 147  97 145 117 109 158  88  92 122 138 102  90
 111 180 133 106 171 159 146  71 105 101 176 150  73 187  84  44 141 114
  95 129  79   0  62 131 112 113  74  83 136  80 123  81 134 142 144  93
 163 151  96 155  76 160 124 162 132 120 173 170 128 108 154  57 156 153
 188 152 104  87  75 179 130 194 181 135 184 140 177 164  91 165  86 193
 191 161 167  77 182 157 178  61  98 127  82  72 172  94 175 195  68 186
 198 121  67 174 199  56 169 149  65 190]
[ 72  66  64  40  74  50   0  70  96  92  80  60  84  30  88  90  94  76
  82  75  58  78  68 110  56  62  85  86  48  44  65 108  55 122  54  52
  98 104  95  46 102 100  61  24  38 106 114]
[35 29  0 23 32 45 19 47 38 30 41 33 26 15 36 11 31 37 42 25 18 24 39 27
 21 34 10 60 13 20 22 28 54 40 51 56 14 17 50 44 12 46 16  7 52 43 48  8
 49 63 99]
[  0  94 168  88 543 846 175 230  83  96 235 146 115 140 110 245  54 192
 207  70 240  82  36  23 300 342 304 142 128  38 100  90 270  71 125 176
  48  64 228  76 220  40 152  18 135 495  37  51  99 145 225  49  50  92
 325  63 284 119 204 155 485  53 114 105 285 156  78 130  55  58 160 210
```

```
  318  44 190 280  87 271 129 120 478  56  32 744 370  45 194 680 402 258
  375 150  67  57 116 278 122 545  75  74 182 360 215 184  42 132 148 180
  205  85 231  29  68  52 255 171  73 108  43 167 249 293  66 465  89 158
   84  72  59  81 196 415 275 165 579 310  61 474 170 277  60  14  95 237
  191 328 250 480 265 193  79  86 326 188 106  65 166 274  77 126 330 600
  185  25  41 272 321 144  15 183  91  46 440 159 540 200 335 387  22 291
  392 178 127 510  16 112]
 [33.6 26.6 23.3 28.1 43.1 25.6 31.  35.3 30.5  0.  37.6 38.  27.1 30.1
  25.8 30.  45.8 29.6 43.3 34.6 39.3 35.4 39.8 29.  36.6 31.1 39.4 23.2
  22.2 34.1 36.  31.6 24.8 19.9 27.6 24.  33.2 32.9 38.2 37.1 34.  40.2
  22.7 45.4 27.4 42.  29.7 28.  39.1 19.4 24.2 24.4 33.7 34.7 23.  37.7
  46.8 40.5 41.5 25.  25.4 32.8 32.5 42.7 19.6 28.9 28.6 43.4 35.1 32.
  24.7 32.6 43.2 22.4 29.3 24.6 48.8 32.4 38.5 26.5 19.1 46.7 23.8 33.9
  20.4 28.7 49.7 39.  26.1 22.5 39.6 29.5 34.3 37.4 33.3 31.2 28.2 53.2
  34.2 26.8 55.  42.9 34.5 27.9 38.3 21.1 33.8 30.8 36.9 39.5 27.3 21.9
  40.6 47.9 50.  25.2 40.9 37.2 44.2 29.9 31.9 28.4 43.5 32.7 67.1 45.
  34.9 27.7 35.9 22.6 33.1 30.4 52.3 24.3 22.9 34.8 30.9 40.1 23.9 37.5
  35.5 42.8 42.6 41.8 35.8 37.8 28.8 23.6 35.7 36.7 45.2 44.  46.2 35.
  43.6 44.1 18.4 29.2 25.9 32.1 36.3 40.  25.1 27.5 45.6 27.8 24.9 25.3
  37.9 27.  26.  38.7 20.8 36.1 30.7 32.3 52.9 21.  39.7 25.5 26.2 19.3
  38.1 23.5 45.5 23.1 39.9 36.8 21.8 41.  42.2 34.4 27.2 36.5 29.8 39.2
  38.4 36.2 48.3 20.  22.3 45.7 23.7 22.1 42.1 42.4 18.2 26.4 45.3 37.
  24.5 32.2 59.4 21.2 26.7 30.2 46.1 41.3 38.8 35.2 42.3 40.7 46.5 33.5
  37.3 30.3 26.3 21.7 36.4 28.5 26.9 38.6 31.3 19.5 20.1 40.8 23.4 28.3
  38.9 57.3 35.6 49.6 44.6 24.1 44.5 41.2 49.3 46.3]
 [0.627 0.351 0.672 0.167 2.288 0.201 0.248 0.134 0.158 0.232 0.191 0.537
  1.441 0.398 0.587 0.484 0.551 0.254 0.183 0.529 0.704 0.388 0.451 0.263
  0.205 0.257 0.487 0.245 0.337 0.546 0.851 0.267 0.188 0.512 0.966 0.42
  0.665 0.503 1.39  0.271 0.696 0.235 0.721 0.294 1.893 0.564 0.586 0.344
  0.305 0.491 0.526 0.342 0.467 0.718 0.962 1.781 0.173 0.304 0.27  0.699
  0.258 0.203 0.855 0.845 0.334 0.189 0.867 0.411 0.583 0.231 0.396 0.14
  0.391 0.37  0.307 0.102 0.767 0.237 0.227 0.698 0.178 0.324 0.153 0.165
  0.443 0.261 0.277 0.761 0.255 0.13  0.323 0.356 0.325 1.222 0.179 0.262
  0.283 0.93  0.801 0.207 0.287 0.336 0.247 0.199 0.543 0.192 0.588 0.539
  0.22  0.654 0.223 0.759 0.26  0.404 0.186 0.278 0.496 0.452 0.403 0.741
  0.361 1.114 0.457 0.647 0.088 0.597 0.532 0.703 0.159 0.268 0.286 0.318
  0.272 0.572 0.096 1.4   0.218 0.085 0.399 0.432 1.189 0.687 0.137 0.637
  0.833 0.229 0.817 0.204 0.368 0.743 0.722 0.256 0.709 0.471 0.495 0.18
  0.542 0.773 0.678 0.719 0.382 0.319 0.19  0.956 0.084 0.725 0.299 0.244
```

In the dataset the variables except Pregnancies and Outcome cannot have value as 0, because it is not possible to have 0 Glucose Level or to have 0 Blood Pressure. So, this will be counted as incorrect information

Checking the count of value 0 in the variables

```
variables = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age',]
for i in variables:
    c = 0
    for x in (df[i]):
        if x == 0:
            c = c + 1
    print(i,c)
```

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

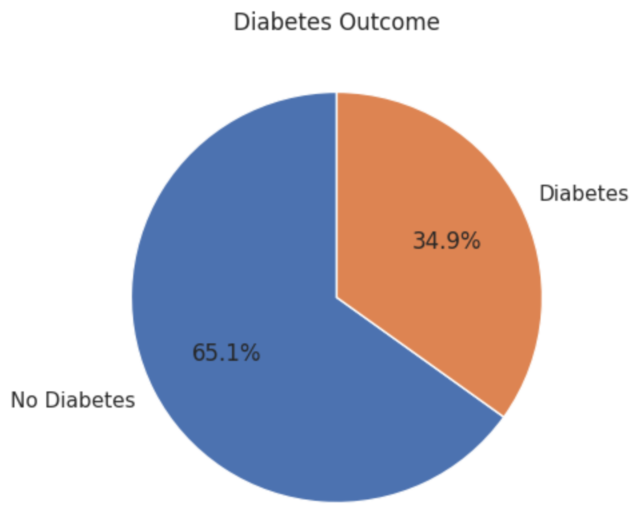
Replacing the 0 value in the variables - Glucose, BloodPressure, SkinThickness, Insulin, BMI

```
# Fill missing values in each column with the mean of that column
variables = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
for column in variables:
    mean_value = df[column].mean()
    df[column].fillna(mean_value, inplace=True)


missing_value_counts = {}

# Loop through columns and count missing values
for column in variables:
    missing_count = df[column].isnull().sum()
    missing_value_counts[column] = missing_count


print(missing_value_counts)
```

```
{'Glucose': 0, 'BloodPressure': 0, 'SkinThickness': 0, 'Insulin': 0, 'BMI': 0}
```

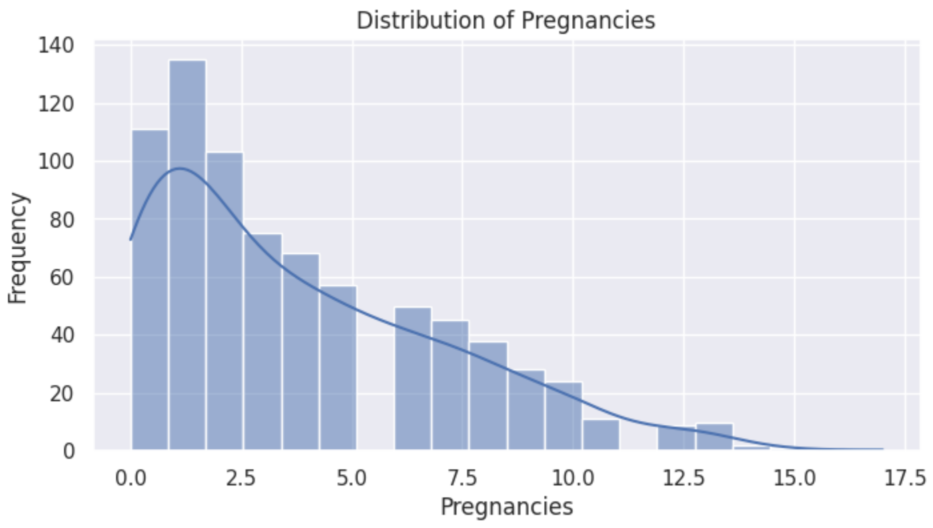
Exploratory Data Analysis

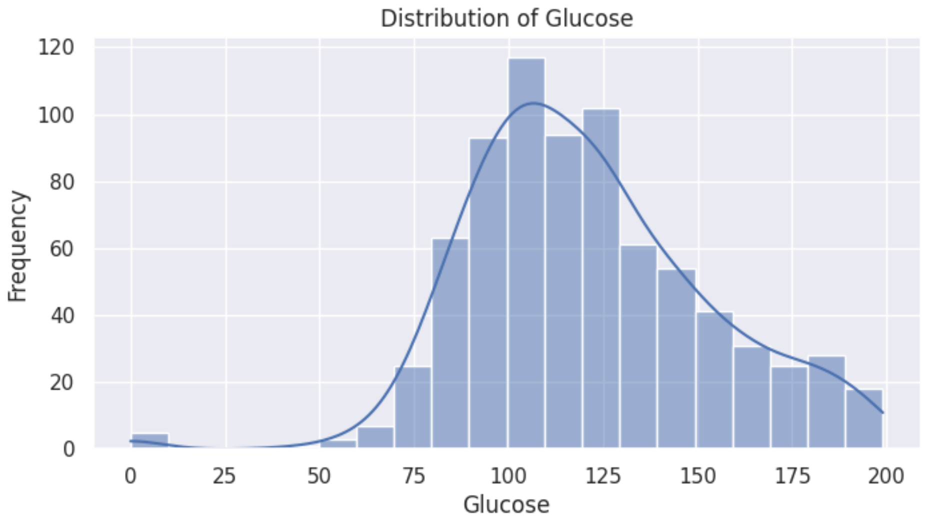
```
plt.figure(figsize=(5,5))
plt.pie(df['Outcome'].value_counts(), labels=['No Diabetes', 'Diabetes'], autopct='%1.1f%%', shadow=False, startangle=90)
plt.title('Diabetes Outcome')
plt.show()
```

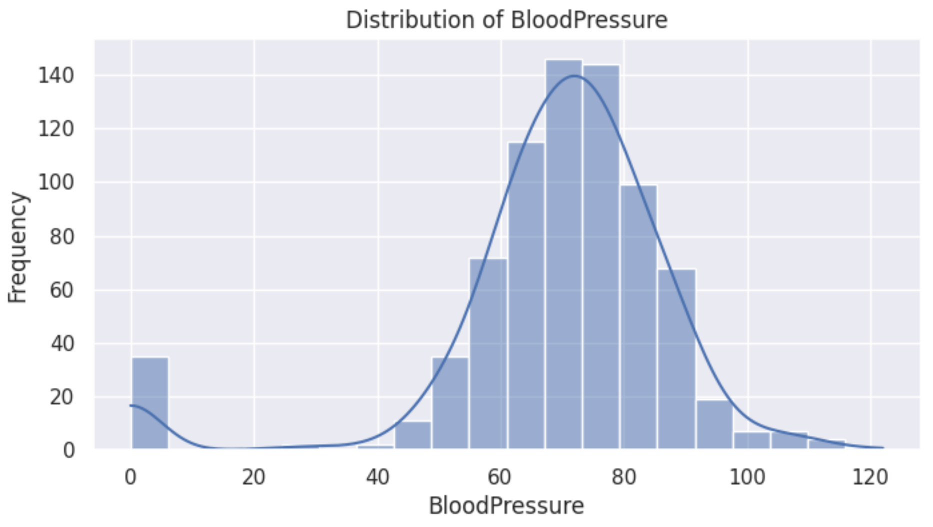


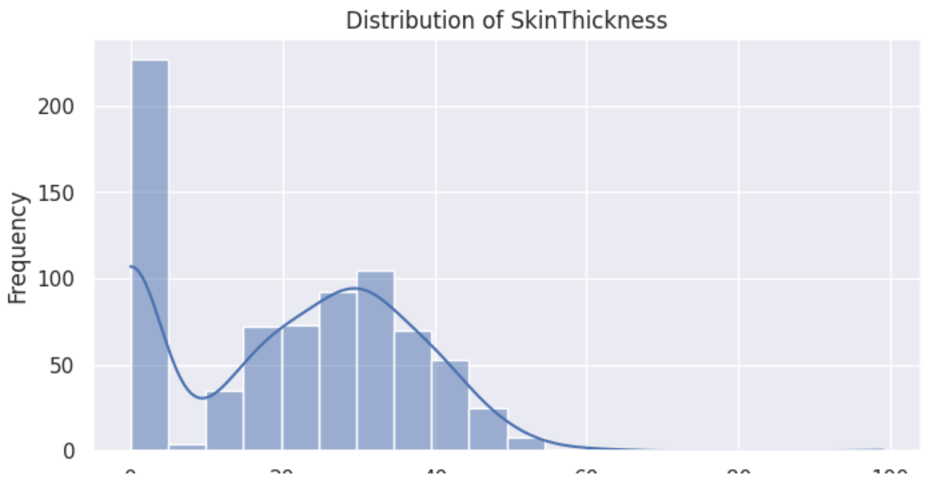
```
numeric_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
summary_stats = df[numeric_columns].describe()
```

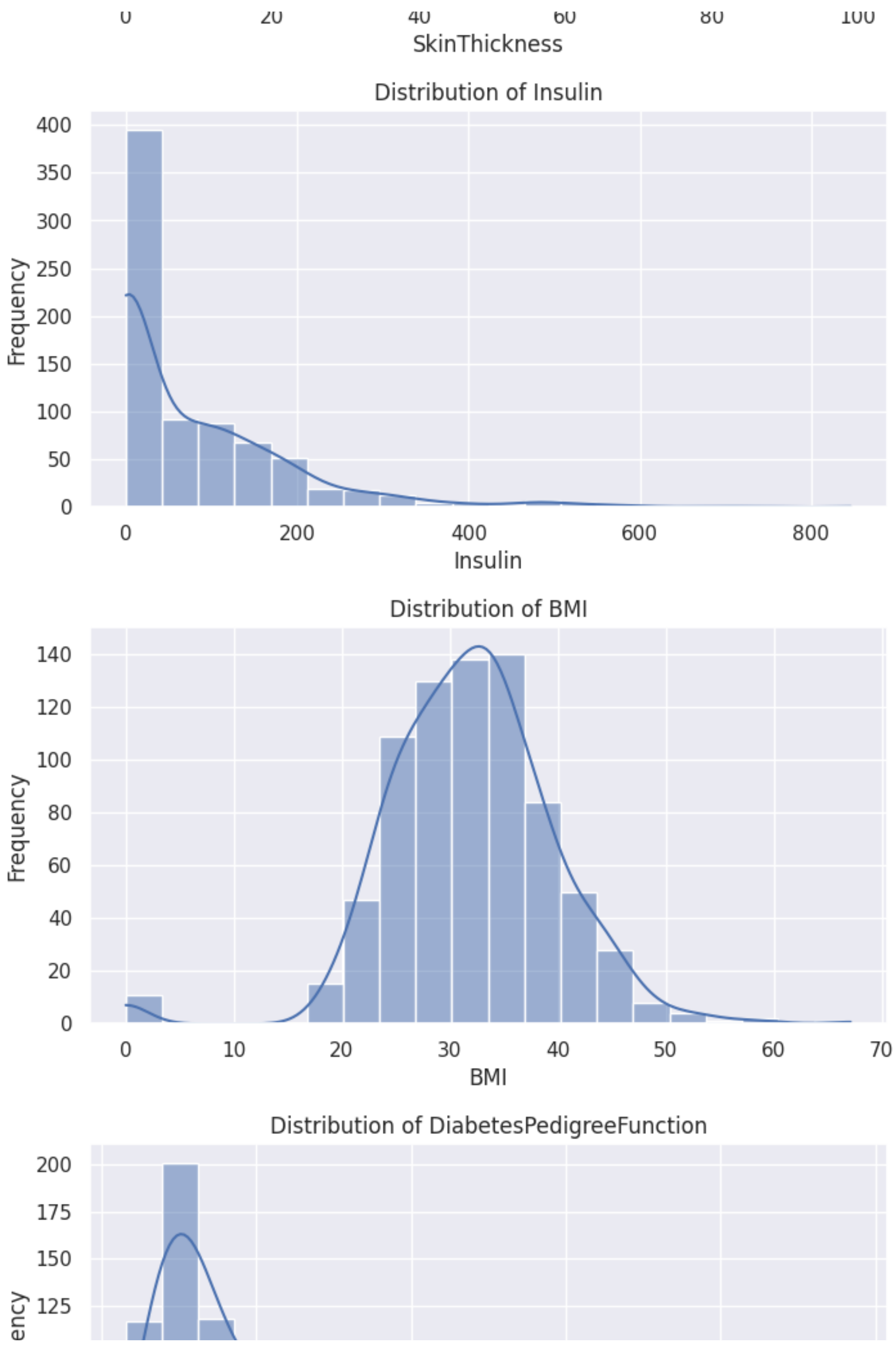
```
# Distribution plots for numeric columns

for column in numeric_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[column], kde=True, bins=20)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```


Distribution of Pregnancies
Frequency
0
20
40
60
80
100
120
140
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Pregnancies

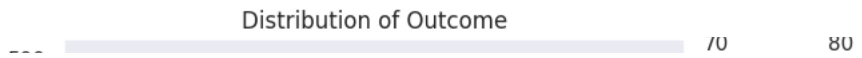

Distribution of Glucose
Frequency
0
20
40
60
80
100
120
0
25
50
75
100
125
150
175
200
Glucose


Distribution of BloodPressure
Frequency
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
BloodPressure


Distribution of SkinThickness
Frequency
0
50
100
150
200



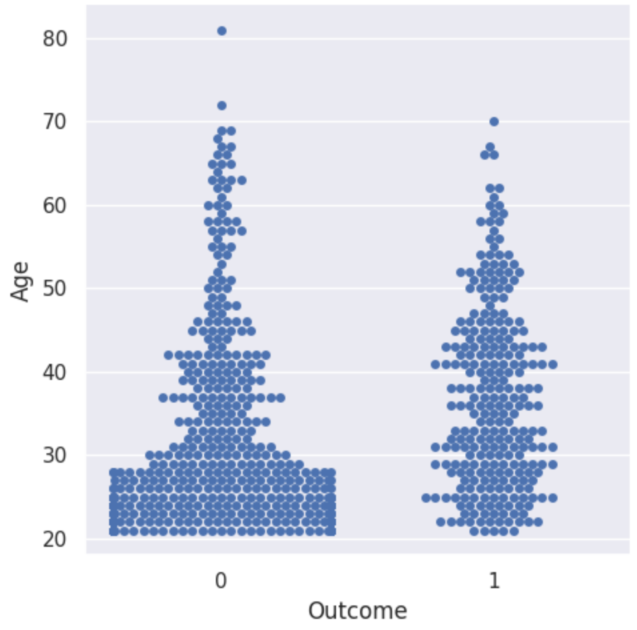
```
# Distribution of the 'Outcome' column (assuming it's binary)
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='Outcome')
plt.title('Distribution of Outcome')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.show()
```



Age Distribution and Diabetes

```
sns.catplot(x="Outcome", y="Age", kind="swarm", data=df)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 7.2% of the points cannot be placed; you may want
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.FacetGrid at
0x798aecc54d00>/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 23.2% of the points cannot be plac
  warnings.warn(msg, UserWarning)
```

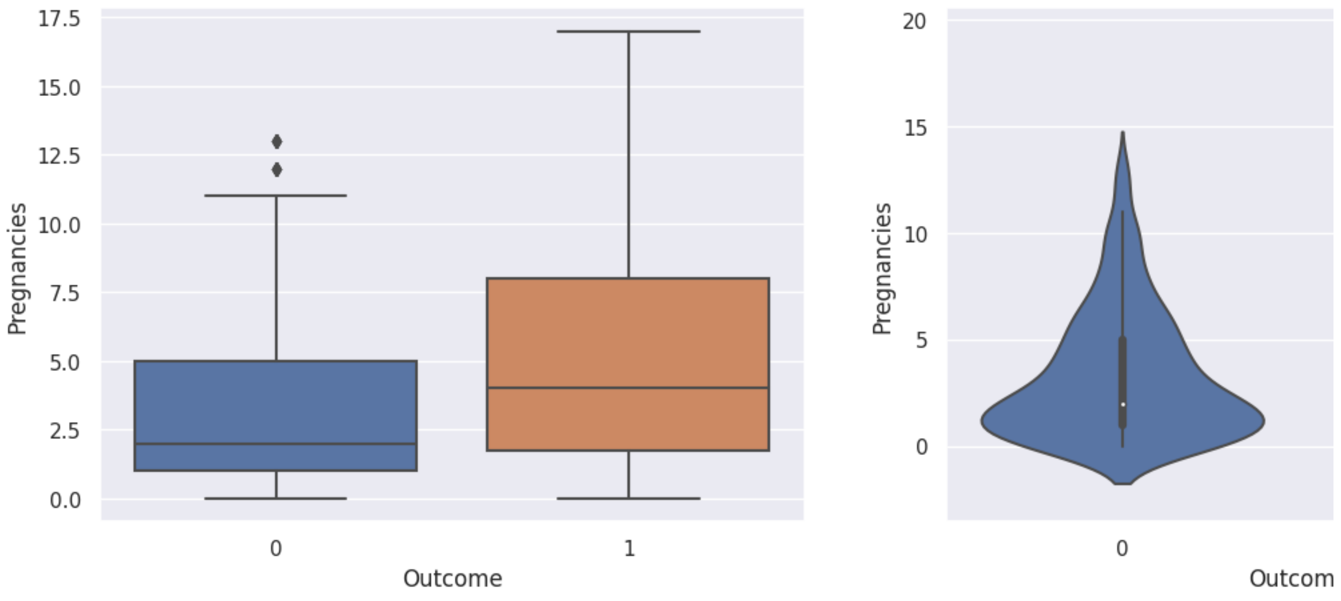


From the graph, it is quite clear that majority of the patients are adult within the age group of 20-30 years. Patients in the age range 40-55 years are more prone to diabetes, as compared to other age groups. Since the number adults in the age group 20-30 years is more, the number of patients with diabetes is also more as compared of other age groups.

Pregnancies and Diabetes

```
fig,ax = plt.subplots(1,2,figsize=(15,5))
sns.boxplot(x='Outcome',y='Pregnancies',data=df,ax=ax[0])
sns.violinplot(x='Outcome',y='Pregnancies',data=df,ax=ax[1])
```

```
<Axes: xlabel='Outcome', ylabel='Pregnancies'>
```

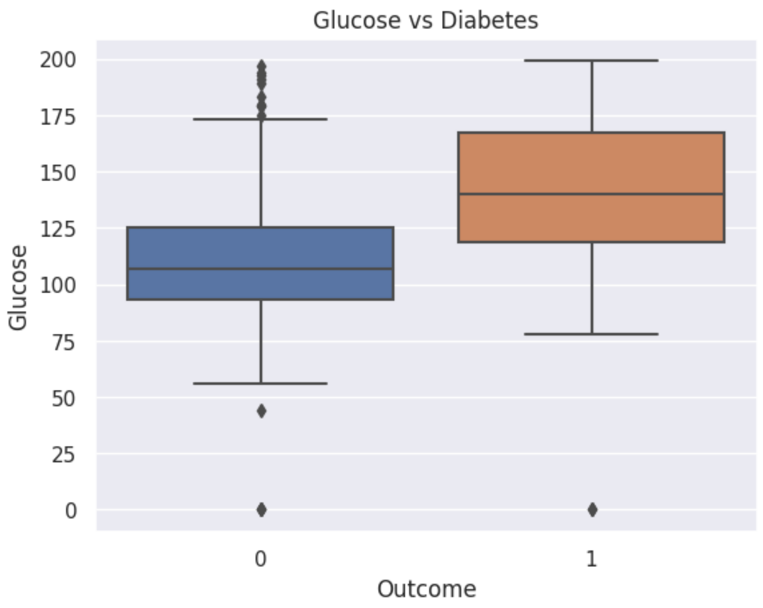


Both boxplot and violinplot shows strange relation between the number of preganacies and diabetes. According to the graphs the increased number of pregnancies highlights increased risk of diabetes.

Glucose and Diabetes

```
sns.boxplot(x='Outcome', y='Glucose', data=df).set_title('Glucose vs Diabetes')
```

```
Text(0.5, 1.0, 'Glucose vs Diabetes')
```

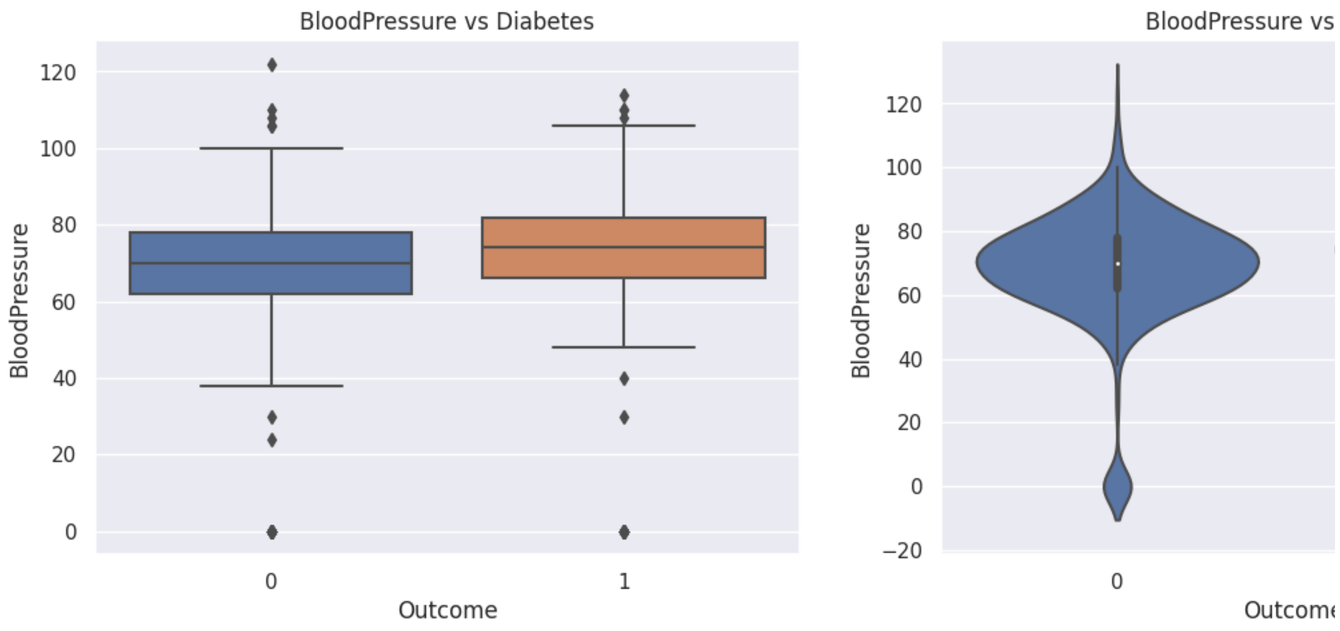


Glucose level plays a major role in determine whether the patient is diabetic or not. The patients with median gluocse level less than 120 are more likely to be non-diabetic. The patients with median gluocse level greather than 140 are more likely to be diabetic. Therefore, high gluocose levels is a good indicator of diabetes.

Blood Pressuse and Diabetes

```
fig,ax = plt.subplots(1,2,figsize=(15,5))
sns.boxplot(x='Outcome', y='BloodPressure', data=df, ax=ax[0]).set_title('BloodPressure vs Diabetes')
sns.violinplot(x='Outcome', y='BloodPressure', data=df, ax=ax[1]).set_title('BloodPressure vs Diabetes')
```

```
Text(0.5, 1.0, 'BloodPressure vs Diabetes')
```

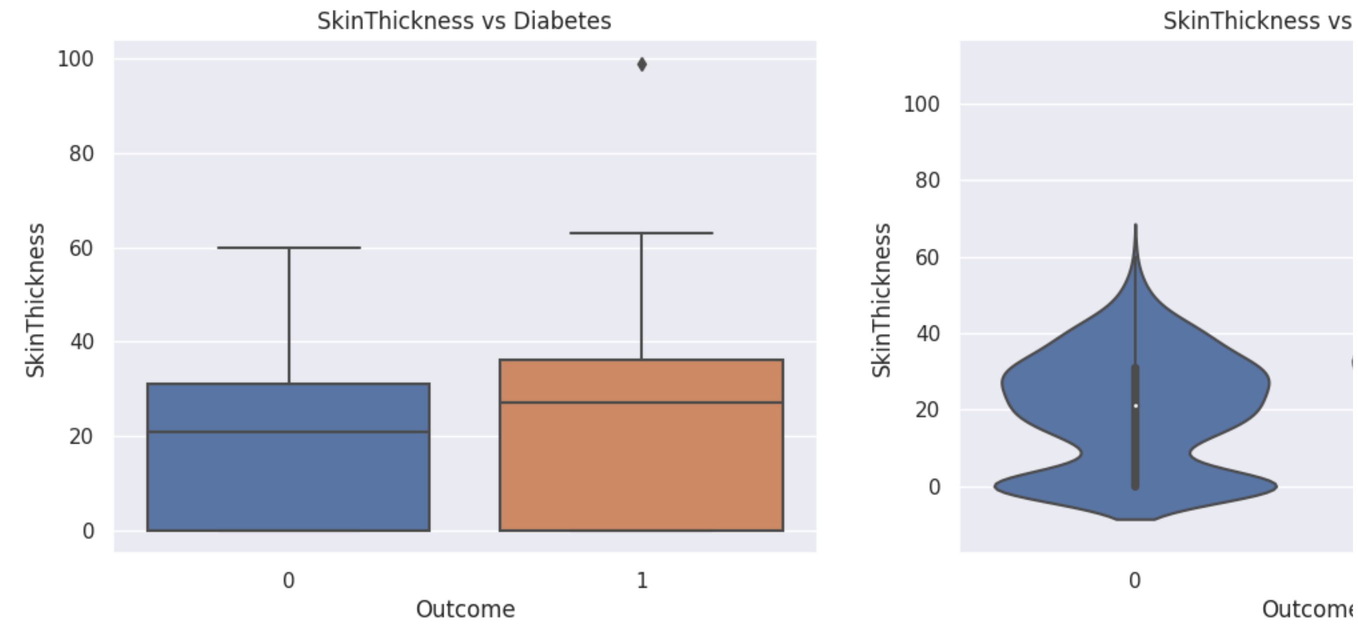


Both the boxplot and voilinplot provides clear understanding of the realtion between the blood pressure and diabetes. The boxplot shows that the median of the blood pressure for the diabetic patients is slightly higher than the non-diabetic patients. The voilinplot shows that the distribution of the blood pressure for the diabetic patients is slightly higher than the non-diabetic patients. But there has been not enough evidence to conclude that the blood pressure is a good predictor of diabetes.

Skin Thickness and Diabetes

```
fig,ax = plt.subplots(1,2,figsize=(15,5))
sns.boxplot(x='Outcome', y='SkinThickness', data=df,ax=ax[0]).set_title('SkinThickness vs Diabetes')
sns.violinplot(x='Outcome', y='SkinThickness', data=df,ax=ax[1]).set_title('SkinThickness vs Diabetes')
```

```
    Text(0.5, 1.0, 'SkinThickness vs Diabetes')
```



Here both the boxplot and violinplot reveals the effect of diabetes on skin thickness. As obserevd in the boxplot, the median of skin thickness is higher for the diabetic patients than the non-diabetic patients, where non diabetic patients have median skin thickness near 20 in comparison to skin thickness nearly 30 in diabetic patients. The voilinpplot shows the distribution of patients' skin thickness amoung the patients, where the non diabetic ones have greater distribution near 20 and diabetic much less distribution near 20 and increased distribution near 30. Therefore, skin thickness can be a indicator of diabetes.

Insulin and Diabetes

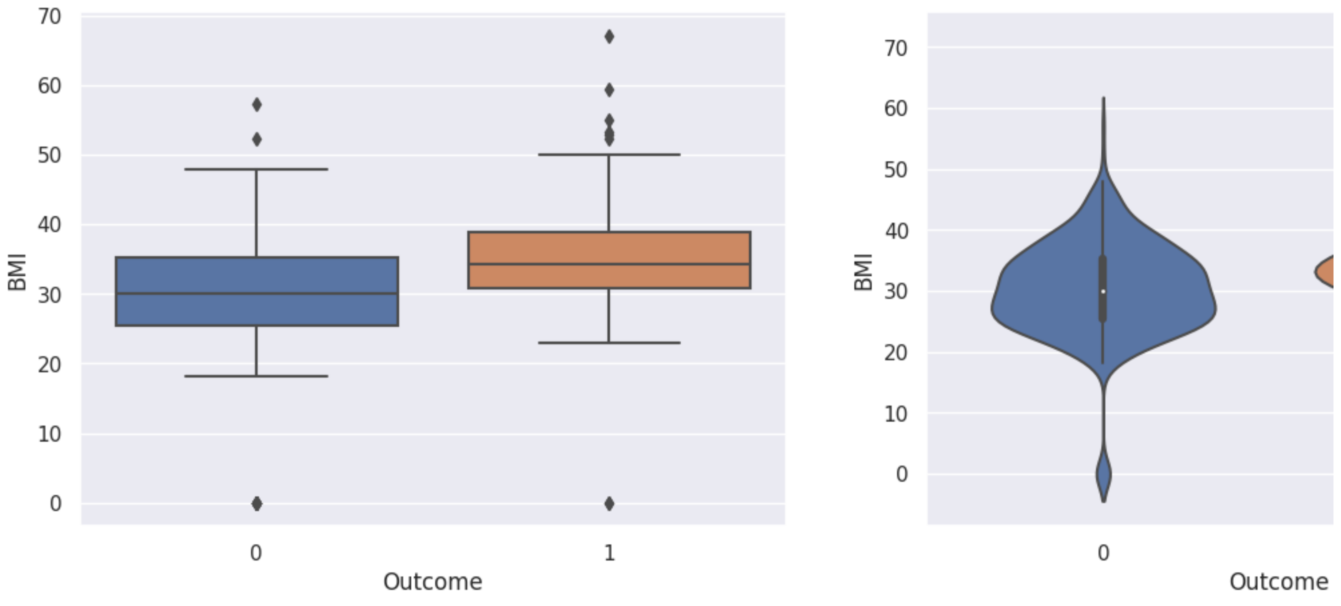
```
fig,ax = plt.subplots(1,2,figsize=(15,5))
sns.boxplot(x='Outcome',y='Insulin',data=df,ax=ax[0]).set_title('Insulin vs Diabetes')
sns.violinplot(x='Outcome',y='Insulin',data=df,ax=ax[1]).set_title('Insulin vs Diabetes')
```

Insulin is a major body hormone that regulates glucose metabolism. Insulin is required for the body to efficiently use sugars, fats and proteins. Any change in insulin amount in the body would result in change glucose levels as well. Here the boxplot and violinplot shows the distribution of insulin level in patients. In non diabetic patients the insulin level is near to 100, whereas in diabetic patients the insulin level is near to 200. In the voilinplot we can see that the distribution of insulin level in non diabetic patients is more spread out near 100, whereas in diabetic patients the distribution is contracted and shows a little bit spread in higher insulin levels. This shows that the insulin level is a good indicator of diabetes.

BMI and Diabetes

```
fig,ax = plt.subplots(1,2,figsize=(15,5))
sns.boxplot(x='Outcome',y='BMI',data=df,ax=ax[0])
sns.violinplot(x='Outcome',y='BMI',data=df,ax=ax[1])
```

```
<Axes: xlabel='Outcome', ylabel='BMI'>
```



Both graphs highlights the role of BMI in diabetes prediction. Non diabetic patients have a normal BMI within the range of 25-35 whereas the diabetic patients have a BMI greater than 35. The violinplot reveals the BMI distribution, where the non dibetic patients have a increased spread from 25 to 35 with narrows after 35. However in diabetic patients there is increased spread at 35 and increased spread 45-50 as compared to non diabetic patients.Therefore BMI is a good predictor of diabetes and obese people are more likely to be diabetic.

Diabetes Pedigree Function and Diabetes Outcome

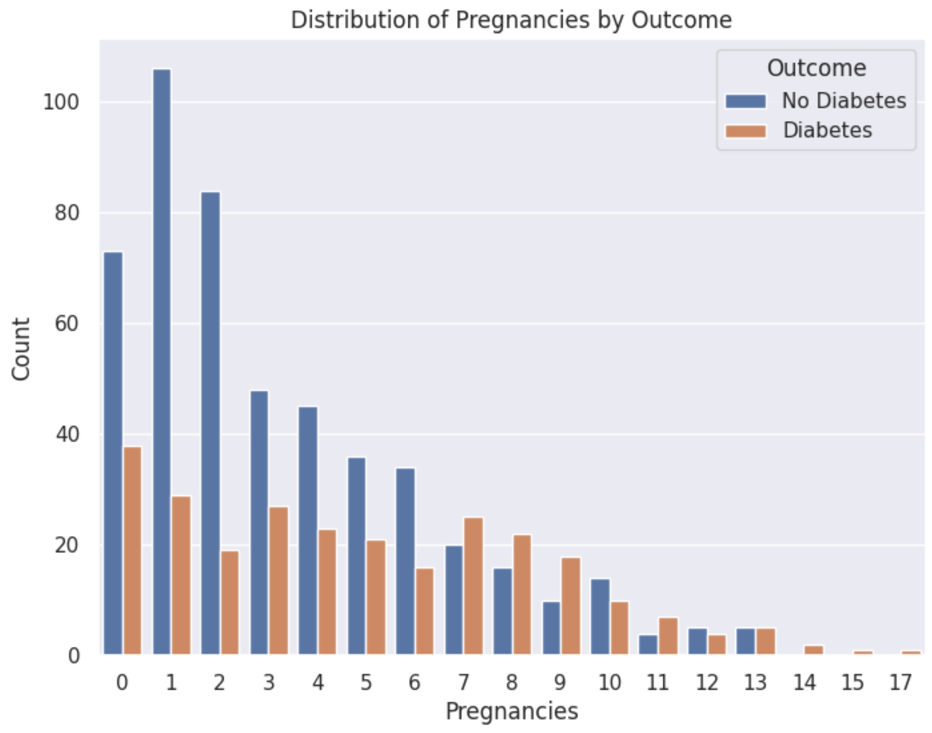
```
fig,ax = plt.subplots(1,2,figsize=(15,5))
sns.boxplot(x='Outcome',y='DiabetesPedigreeFunction',data=df,ax=ax[0]).set_title('Diabetes Pedigree Function')
sns.violinplot(x='Outcome',y='DiabetesPedigreeFunction',data=df,ax=ax[1]).set_title('Diabetes Pedigree Function')
```

```
Text(0.5, 1.0, 'Diabetes Pedigree Function')
```

Diabetes Pedigree Function (DPF) calculates diabetes likelihood depending on the subject's age and his/her diabetic family history. From the boxplot, the patients with lower DPF, are much less likely to have diabetes. The patients with higher DPF, are much more likely to have diabetes. In the violinplot, majority of the non diabetic patients have a DPF of 0.25-0.35, whereas the diabetic patients have a increased DPF, which is shown by the their distribution in the violinplot where there is a increased spread in the DPF from 0.5 -1.5. Therefore the DPF is a good indicator of diabetes.

Coorelation Matrix Heatmap

```
# 1. Categorical vs. Categorical
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Pregnancies', hue='Outcome')
plt.title('Distribution of Pregnancies by Outcome')
plt.xlabel('Pregnancies')
plt.ylabel('Count')
plt.legend(title='Outcome', labels=['No Diabetes', 'Diabetes'])
plt.show()
```



```
from scipy.stats import chi2_contingency, ttest_ind
# Chi-squared test for independence
contingency_table = pd.crosstab(df['Pregnancies'], df['Outcome'])
chi2, p, _, _ = chi2_contingency(contingency_table)
print(f"Chi-squared p-value: {p}")
```

```
Chi-squared p-value: 8.648349123362548e-08
```

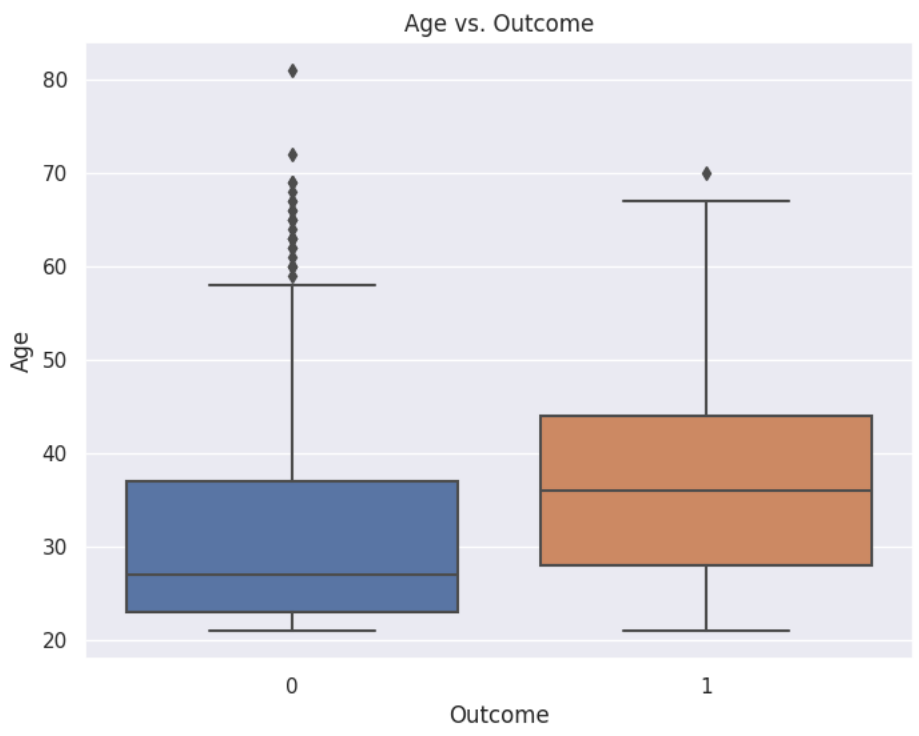
The p-value from a chi-squared test for independence represents the probability of observing the observed association (or more extreme) between two categorical variables, such as 'Pregnancies' and 'Outcome,' under the assumption that there is no actual association (independence) between them.

where the p-value is approximately 8.65e-08 (very close to zero), this extremely small p-value indicates the following:

Statistical Significance: The chi-squared test has found strong evidence to reject the null hypothesis. The null hypothesis in this context would typically be that there is no association or independence between 'Pregnancies' and 'Outcome.'

Association or Dependence: The small p-value suggests that there is a significant association or dependence between the number of pregnancies ('Pregnancies') and the outcome of the diabetes test ('Outcome') in your dataset.

```
# 2. Categorical vs. Numerical
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='Outcome', y='Age')
plt.title('Age vs. Outcome')
plt.xlabel('Outcome')
plt.ylabel('Age')
plt.show()
```



```
# t-test for difference in means
no_diabetes_age = df[df['Outcome'] == 0]['Age']
diabetes_age = df[df['Outcome'] == 1]['Age']
t_stat, p_value = ttest_ind(no_diabetes_age, diabetes_age)
print(f"t-test p-value for Age and Outcome: {p_value}")
```

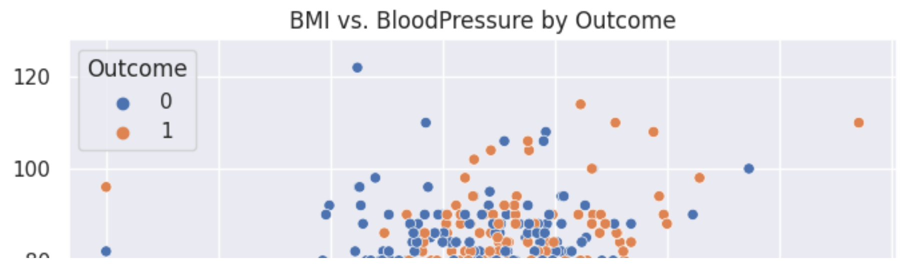
```
    t-test p-value for Age and Outcome: 2.2099754606654358e-11
```

The p-value you obtained from the t-test for 'Age' and 'Outcome' is approximately 2.21e-11, which is an extremely small number. In simple terms, here's what you can understand from this result:

Statistical Significance: The small p-value (much less than 0.05 or any common significance level) indicates strong evidence against the idea that there is no difference in the average ages between the two groups (diabetes and no diabetes).

Difference in Means: A small p-value suggests that there is a statistically significant difference in the average ages of individuals with diabetes and those without diabetes.

```
# 3. Numerical vs. Numerical
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='BMI', y='BloodPressure', hue='Outcome')
plt.title('BMI vs. BloodPressure by Outcome')
plt.xlabel('BMI')
plt.ylabel('BloodPressure')
plt.show()
```



```
#Correlation between BMI and BloodPressure
correlation = df[['BMI', 'BloodPressure']].corr(method='pearson')
print("Correlation matrix between BMI and BloodPressure:")
print(correlation)
```

```
Correlation matrix between BMI and BloodPressure:
                    BMI  BloodPressure
BMI            1.000000       0.281805
BloodPressure  0.281805       1.000000
```

Correlation Coefficient Values:

The correlation coefficient between 'BMI' and 'BloodPressure' is approximately 0.282. This value is between -1 and 1. A positive correlation coefficient (0.282) indicates a positive linear relationship, meaning that as 'BMI' increases, 'BloodPressure' tends to increase as well. However, the correlation is relatively weak, as the coefficient is closer to 0 than to 1. Strength of Relationship:

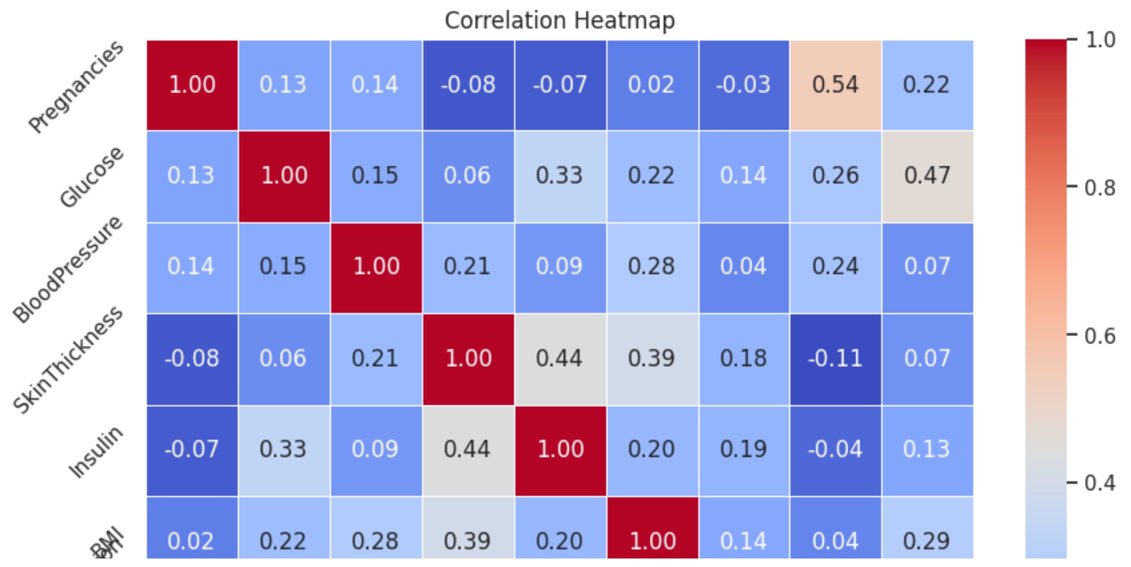
The correlation coefficient of 0.282 suggests a relatively weak positive relationship between 'BMI' and 'BloodPressure.' This means that while there is a tendency for these variables to increase together, the relationship is not very strong. Interpretation:

In practical terms, this correlation suggests that there is some degree of association between higher BMI values and higher blood pressure values. This is consistent with the common understanding that excess body weight (higher BMI) can be a risk factor for elevated blood pressure. Magnitude of Correlation:

The magnitude of the correlation coefficient (0.282) indicates that while there is a relationship, it may not be the only factor influencing blood pressure. Other factors could also play a significant role.

In summary, the correlation matrix suggests a weak positive linear relationship between 'BMI' and 'BloodPressure' in your dataset. It's important to note that correlation does not imply causation. While these variables are correlated, it doesn't necessarily mean that one directly causes the other; other factors and mechanisms may be involved.

```
correlation_matrix=df.corr()


plt.figure(figsize=(10, 8))  # Adjust the figure size as needed

# Create a heatmap using Seaborn
sns.set(font_scale=1.0)  # Adjust the font size as needed
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)

# Customize the plot
plt.title("Correlation Heatmap")
plt.xticks(rotation=45)
plt.yticks(rotation=45)

# Display the heatmap
plt.show()
```



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.drop('Outcome',axis=1),df['Outcome'],test_size=0.2,random_state=42)
```

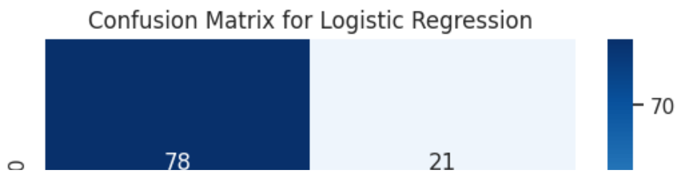
Logistic Regression

```
#building model
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr

#training the model
lr.fit(X_train,y_train)
#training accuracy
lr.score(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.7719869706840391
```

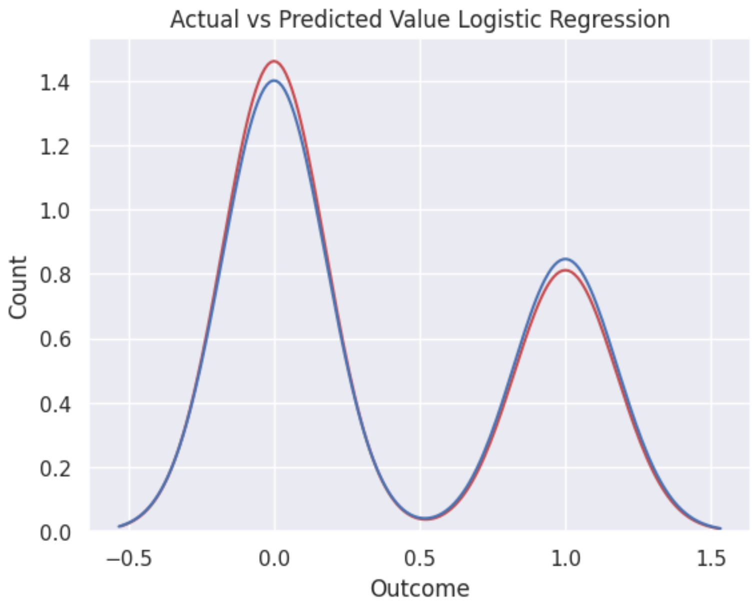
```
#predicted outcomes
lr_pred = lr.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, cmap='Blues')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```



```
ax = sns.distplot(y_test, color='r',  label='Actual Value',hist=False)
sns.distplot(lr_pred, color='b', label='Predicted Value',hist=False,ax=ax)
plt.title('Actual vs Predicted Value Logistic Regression')
plt.xlabel('Outcome')
plt.ylabel('Count')
```

```
<ipython-input-48-bc104d462945>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(y_test, color='r',  label='Actual Value',hist=False)
<ipython-input-48-bc104d462945>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(lr_pred, color='b', label='Predicted Value',hist=False,ax=ax)
Text(0, 0.5, 'Count')
```



These distribution plot clearly visualizes the accuracy of the model. The red color represents the actual values and the blue color represents the predicted values. The more the overlapping of the two colors, the more accurate the model is.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, lr_pred))
```

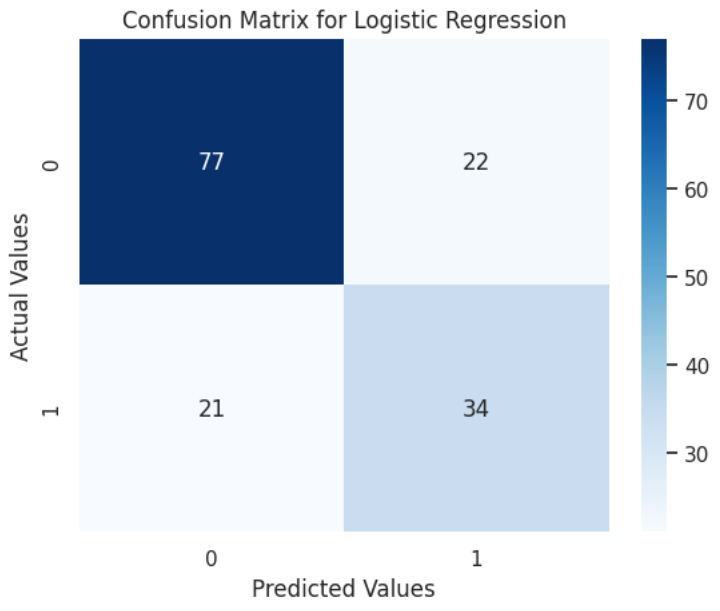
```
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        99
           1       0.64      0.67      0.65        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154
```

Random Forest Classifier

```
#buidling model
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100,random_state=42)
rfc

#training model
rfc.fit(X_train, y_train)
#training accuracy
rfc.score(X_train, y_train)

     1.0
```

```
#predicted outcomes
rfc_pred = rfc.predict(X_test)
```

```
sns.heatmap(confusion_matrix(y_test, rfc_pred), annot=True, cmap='Blues')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```



```
ax = sns.distplot(y_test, color='r',  label='Actual Value',hist=False)
sns.distplot(rfc_pred, color='b', label='Predicted Value',hist=False,ax=ax)
plt.title('Actual vs Predicted Value Logistic Regression')
plt.xlabel('Outcome')
plt.ylabel('Count')
```

```
<ipython-input-53-9669e741e5cd>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(y_test, color='r',  label='Actual Value',hist=False)
<ipython-input-53-9669e741e5cd>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
```

```
print(classification_report(y_test, rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

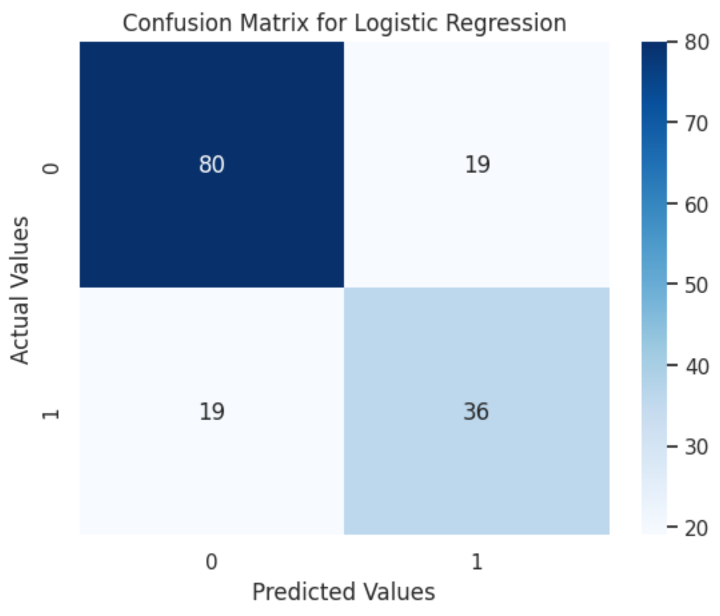
Support Vector Machine (SVM)

```
#building model
from sklearn.svm import SVC
svm = SVC(kernel='linear', random_state=0)
svm

#training the model
svm.fit(X_train, y_train)
#training the model
svm.score(X_test, y_test)
```

```
0.7532467532467533
```

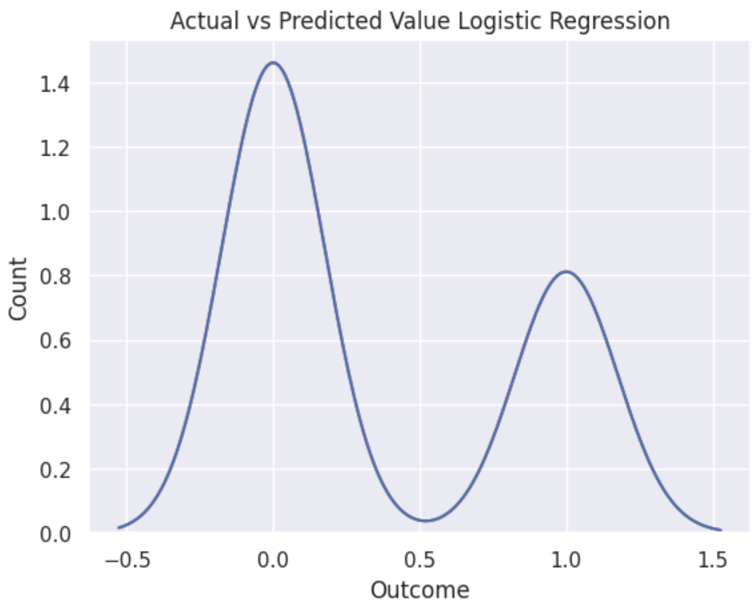
```
#predicting outcomes
svm_pred = svm.predict(X_test)
```

```
sns.heatmap(confusion_matrix(y_test, svm_pred), annot=True, cmap='Blues')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```



```
ax = sns.distplot(y_test, color='r',  label='Actual Value',hist=False)
sns.distplot(svm_pred, color='b', label='Predicted Value',hist=False,ax=ax)
plt.title('Actual vs Predicted Value Logistic Regression')
plt.xlabel('Outcome')
plt.ylabel('Count')
```

```
<ipython-input-59-b9a6ee476682>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(y_test, color='r',  label='Actual Value',hist=False)
<ipython-input-59-b9a6ee476682>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(svm_pred, color='b', label='Predicted Value',hist=False,ax=ax)
Text(0, 0.5, 'Count')
```



```
print(classification_report(y_test, rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

Comparing the models

```
#comparing the accuracy of different models
sns.barplot(x=['Logistic Regression', 'RandomForestClassifier', 'SVM'], y=[0.7792207792207793,0.7662337662337663,0.7597402597402597])
plt.xlabel('Classifier Models')
plt.ylabel('Accuracy')
plt.title('Comparison of different models')
```