

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)

```
In [ ]: #importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [ ]: #Loading the dataset
df = pd.read_csv("diabetes.csv")
df.head()
```

Out []: **Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeF**

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
----------	---	-----	----	----	-----	------

Data Preprocessing

```
In [ ]: #shape of the dataset  
df.shape
```

Out []: (768, 9)

Checking the unique values for each variable in the dataset

```
In [ ]: #checking unique values  
variables = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', '  
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
0.745 0.615 1.321 0.64 0.142 0.374 0.383 0.578 0.136 0.395 0.187 0.905
0.15 0.874 0.236 0.787 0.407 0.605 0.151 0.289 0.355 0.29 0.375 0.164]
```

```

0.431 0.742 0.514 0.464 1.224 1.072 0.805 0.209 0.666 0.101 0.198 0.652
2.329 0.089 0.645 0.238 0.394 0.293 0.479 0.686 0.831 0.582 0.446 0.402
1.318 0.329 1.213 0.427 0.282 0.143 0.38 0.284 0.249 0.926 0.557 0.092
0.655 1.353 0.612 0.2 0.226 0.997 0.933 1.101 0.078 0.24 1.136 0.128
0.422 0.251 0.677 0.296 0.454 0.744 0.881 0.28 0.259 0.619 0.808 0.34
0.434 0.757 0.613 0.692 0.52 0.412 0.84 0.839 0.156 0.215 0.326 1.391
0.875 0.313 0.433 0.626 1.127 0.315 0.345 0.129 0.527 0.197 0.731 0.148
0.123 0.127 0.122 1.476 0.166 0.932 0.343 0.893 0.331 0.472 0.673 0.389
0.485 0.349 0.279 0.346 0.252 0.243 0.58 0.559 0.302 0.569 0.378 0.385
0.499 0.306 0.234 2.137 1.731 0.545 0.225 0.816 0.528 0.509 1.021 0.821
0.947 1.268 0.221 0.66 0.239 0.949 0.444 0.463 0.803 1.6 0.944 0.196
0.241 0.161 0.135 0.376 1.191 0.702 0.674 1.076 0.534 1.095 0.554 0.624
0.219 0.507 0.561 0.421 0.516 0.264 0.328 0.233 0.108 1.138 0.147 0.727
0.435 0.497 0.23 0.955 2.42 0.658 0.33 0.51 0.285 0.415 0.381 0.832
0.498 0.212 0.364 1.001 0.46 0.733 0.416 0.705 1.022 0.269 0.6 0.571
0.607 0.17 0.21 0.126 0.711 0.466 0.162 0.419 0.63 0.365 0.536 1.159
0.629 0.292 0.145 1.144 0.174 0.547 0.163 0.738 0.314 0.968 0.409 0.297
0.525 0.154 0.771 0.107 0.493 0.717 0.917 0.501 1.251 0.735 0.804 0.661
0.549 0.825 0.423 1.034 0.16 0.341 0.68 0.591 0.3 0.121 0.502 0.401
0.601 0.748 0.338 0.43 0.892 0.813 0.693 0.575 0.371 0.206 0.417 1.154
0.925 0.175 1.699 0.682 0.194 0.4 0.1 1.258 0.482 0.138 0.593 0.878
0.157 1.282 0.141 0.246 1.698 1.461 0.347 0.362 0.393 0.144 0.732 0.115
0.465 0.649 0.871 0.149 0.695 0.303 0.61 0.73 0.447 0.455 0.133 0.155
1.162 1.292 0.182 1.394 0.217 0.631 0.88 0.614 0.332 0.366 0.181 0.828
0.335 0.856 0.886 0.439 0.253 0.598 0.904 0.483 0.565 0.118 0.177 0.176
0.295 0.441 0.352 0.826 0.97 0.595 0.317 0.265 0.646 0.426 0.56 0.515
0.453 0.785 0.734 1.174 0.488 0.358 1.096 0.408 1.182 0.222 1.057 0.766
0.171]
[50 31 32 21 33 30 26 29 53 54 34 57 59 51 27 41 43 22 38 60 28 45 35 46
56 37 48 40 25 24 58 42 44 39 36 23 61 69 62 55 65 47 52 66 49 63 67 72
81 64 70 68]
[1 0]

```

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

```

In [ ]: variables = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'Diabetes
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

```

Now, I have count of incorrect values in the variables, I will be replacing these values

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

```
In [ ]: #replacing the missing values with the mean
variables = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
for i in variables:
    df[i].replace(0, df[i].mean(), inplace=True)
```

```
In [ ]: #checking to make sure that incorrect values are replace
for i in variables:
    c = 0
    for x in (df[i]):
        if x == 0:
            c = c + 1
    print(i, c)
```

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

Now, I have replace the incorrect values

Checking for missing values

```
In [ ]: #missing values
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    float64
2   BloodPressure                        768 non-null    float64
3   SkinThickness                        768 non-null    float64
4   Insulin                              768 non-null    float64
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(6), int64(3)
memory usage: 54.1 KB
```

Descriptive Statistics

```
In [ ]: #checking descriptive statistics
df.describe()
```

Out[]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

In []: `df.head()`

Out[]:

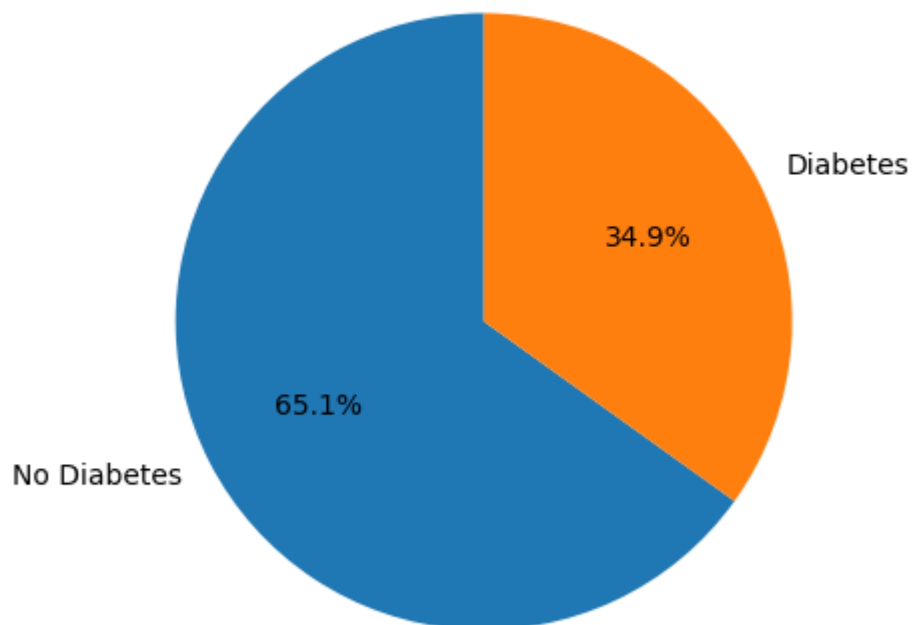
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	85.0	66.0	29.000000	79.799479	26.6	
2	8	183.0	64.0	20.536458	79.799479	23.3	
3	1	89.0	66.0	23.000000	94.000000	28.1	
4	0	137.0	40.0	35.000000	168.000000	43.1	

Exploratory Data Analysis

In the exploratory data analysis, I will be looking at the distribution of the data, the correlation between the features, and the relationship between the features and the target variable. I will start by looking at the distribution of the data, followed by relationship between the target variable and independent variables.

```
In [ ]: plt.figure(figsize=(5,5))
plt.pie(df['Outcome'].value_counts(), labels=['No Diabetes', 'Diabetes'], autopct=
plt.title('Diabetes Outcome')
plt.show()
```

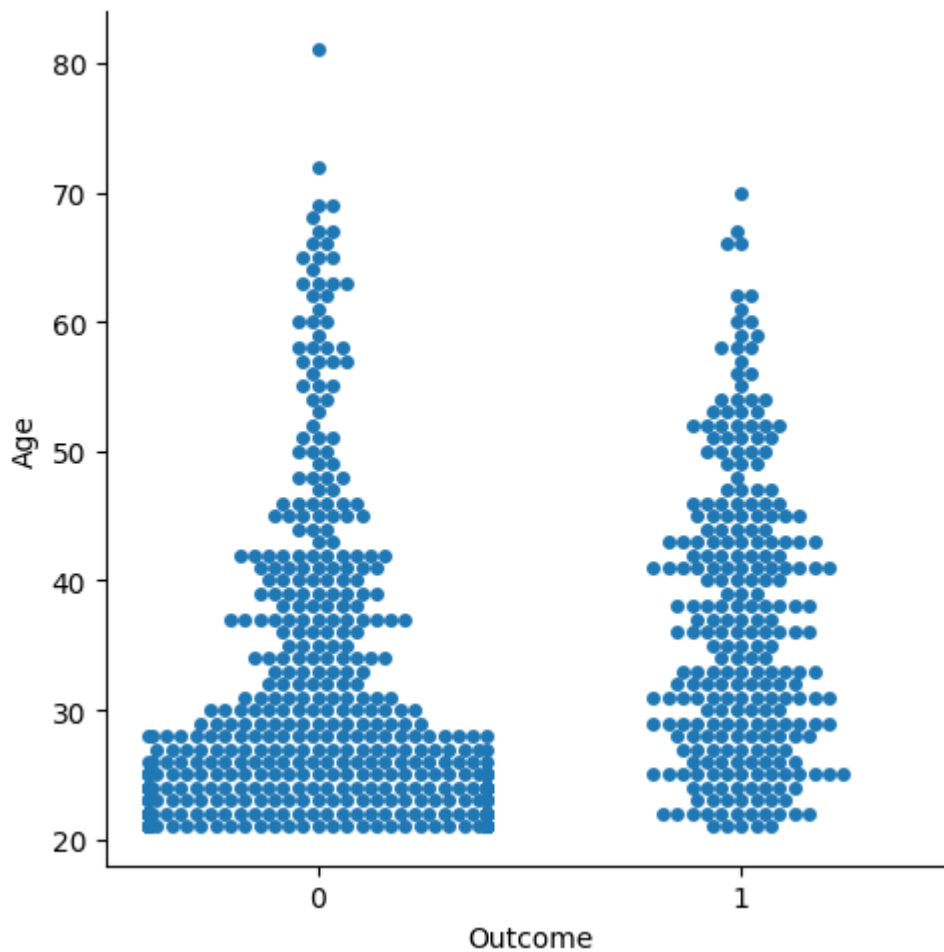
Diabetes Outcome



Age Distribution and Diabetes

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

```
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x2c9111178d0>
```

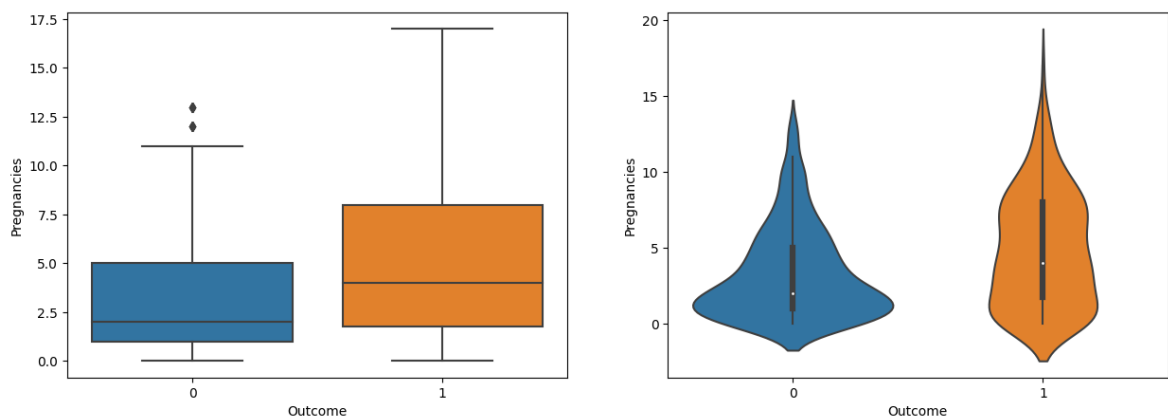


From the graph, it is quite clear that majority of the patients are adult within the age group of 20-30 years. Patients in the age range 40-55 years are more prone to diabetes, as compared to other age groups. Since the number adults in the age group 20-30 years is more, the number of patients with diabetes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

Pregnancies and Diabetes

```
In [ ]: 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])
```

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

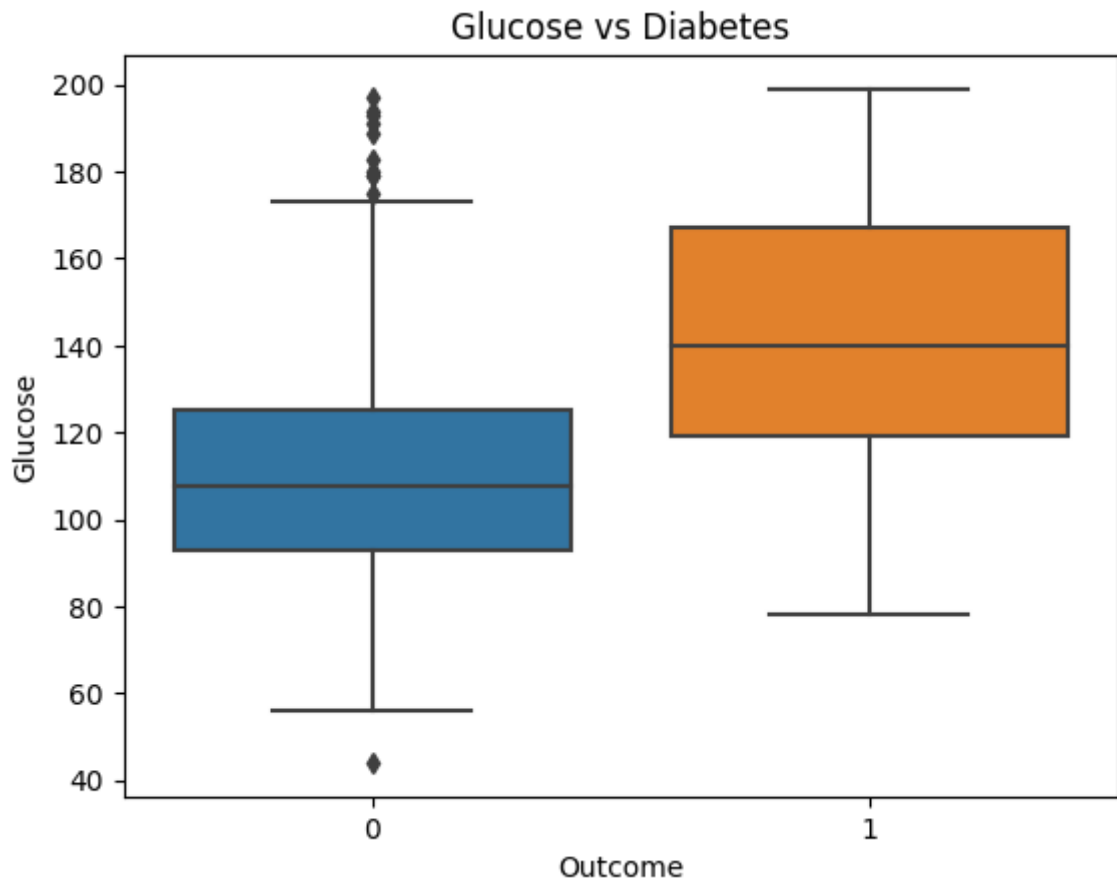


Both boxplot and violinplot shows strange relation between the number of pregnancies and diabetes. According to the graphs the increased number of pregnancies highlights increased risk of diabetes.

Glucose and Diabetes

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

```
Out[ ]: Text(0.5, 1.0, 'Glucose vs Diabetes')
```

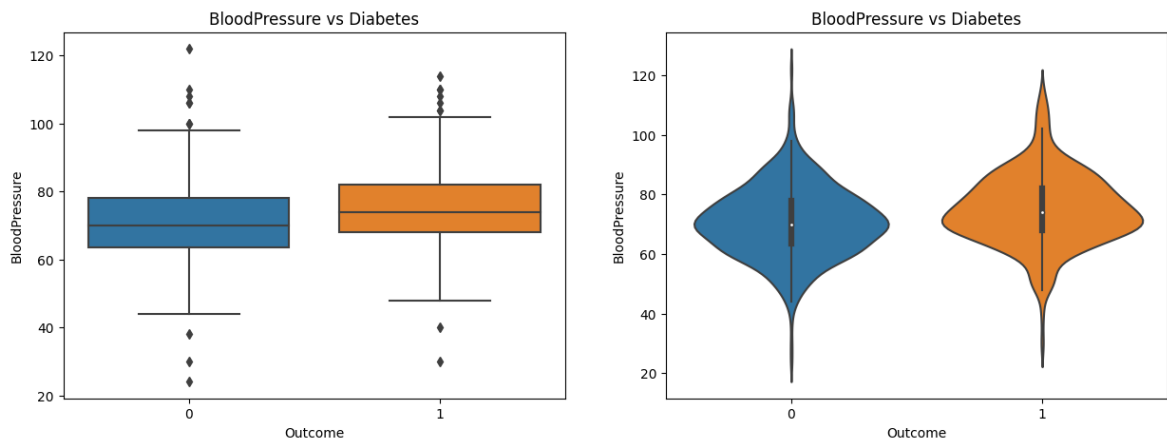


Glucose level plays a major role in determine whether the patient is diabetic or not. The patients with median gluocse level less than 120 are more likely to be non-diabetic. The patients with median gluocse level greather than 140 are more likely to be diabetic. Therefore, high gluocose levels is a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

Blood Pressuse and Diabetes

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

```
Out[ ]: Text(0.5, 1.0, 'BloodPressure vs Diabetes')
```

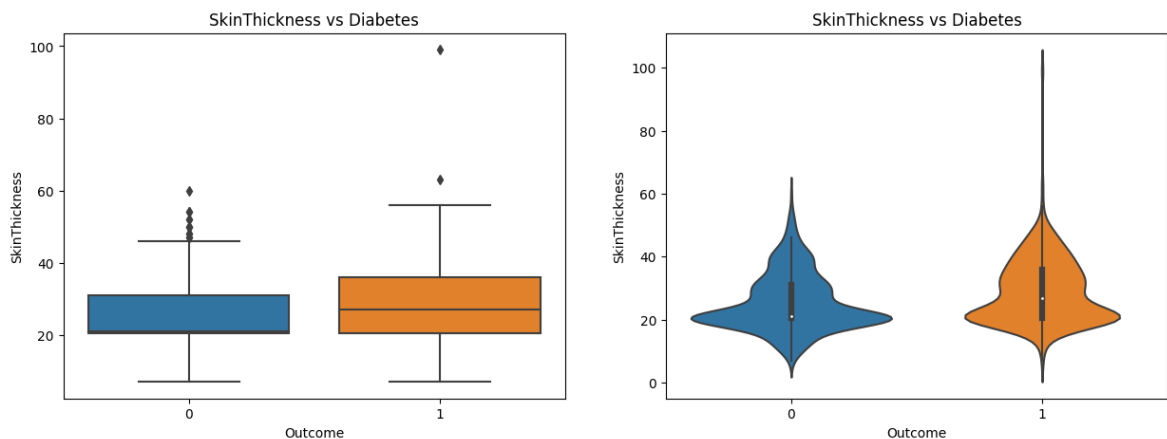


Both the boxplot and violinplot provides clear understanding of the relation between the blood pressure and diabetes. The boxplot shows that the median of the blood pressure for the diabetic patients is slightly higher than the non-diabetic patients. The violinplot shows that the distribution of the 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

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

```
Out[ ]: Text(0.5, 1.0, 'SkinThickness vs Diabetes')
```

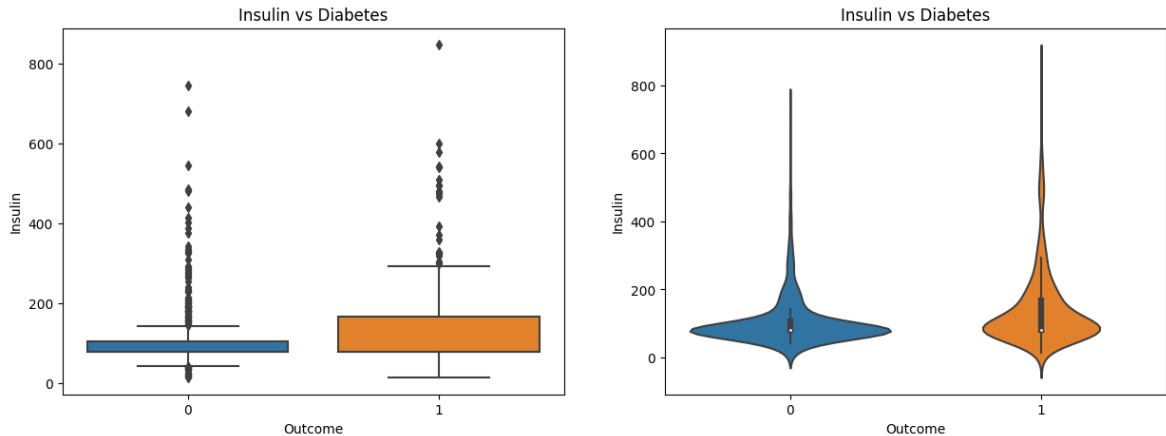


Here both the boxplot and violinplot reveals the effect of diabetes on skin thickness. As observed in the boxplot, the median of skin thickness is higher for the diabetic patients than the non-diabetic patients, where non-diabetic patients have median skin thickness near 20 in comparison to skin thickness nearly 30 in diabetic patients. The violinplot shows the distribution of patients' skin thickness among 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 an indicator of diabetes.

Insulin and Diabetes

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

```
Out[ ]: Text(0.5, 1.0, 'Insulin vs Diabetes')
```

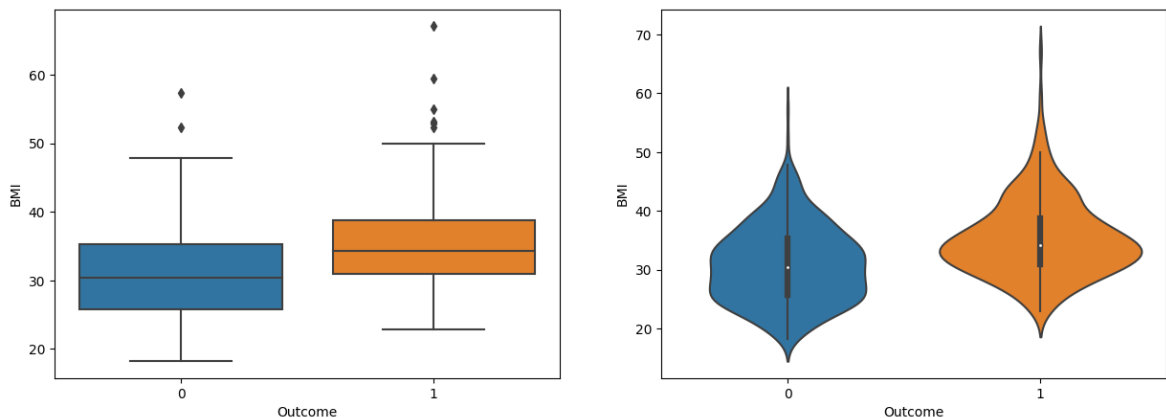


Insulin is a major body hormone that regulates glucose metabolism. Insulin is required for the body to efficiently use sugars, fats and proteins. Any change in insulin amount in the body would result in change glucose levels as well. Here the boxplot and violinplot shows the distribution of insulin level 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 violinplot 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

```
In [ ]: 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])
```

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



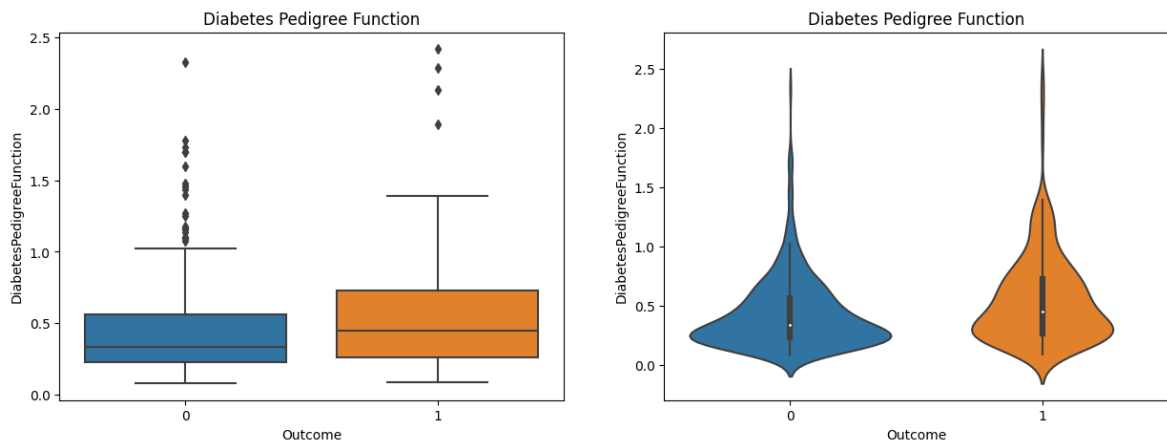
Both graphs highlights the role of BMI in diabetes prediction. Non diabetic patients have a normal BMI within the range of 25-35 whereas the diabetic patients have a BMI greater than 35. The violinplot reveals the BMI distribution, where the non diabetic patients have a increased spread from 25 to 35 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

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

```
Out[ ]: Text(0.5, 1.0, 'Diabetes Pedigree Function')
```

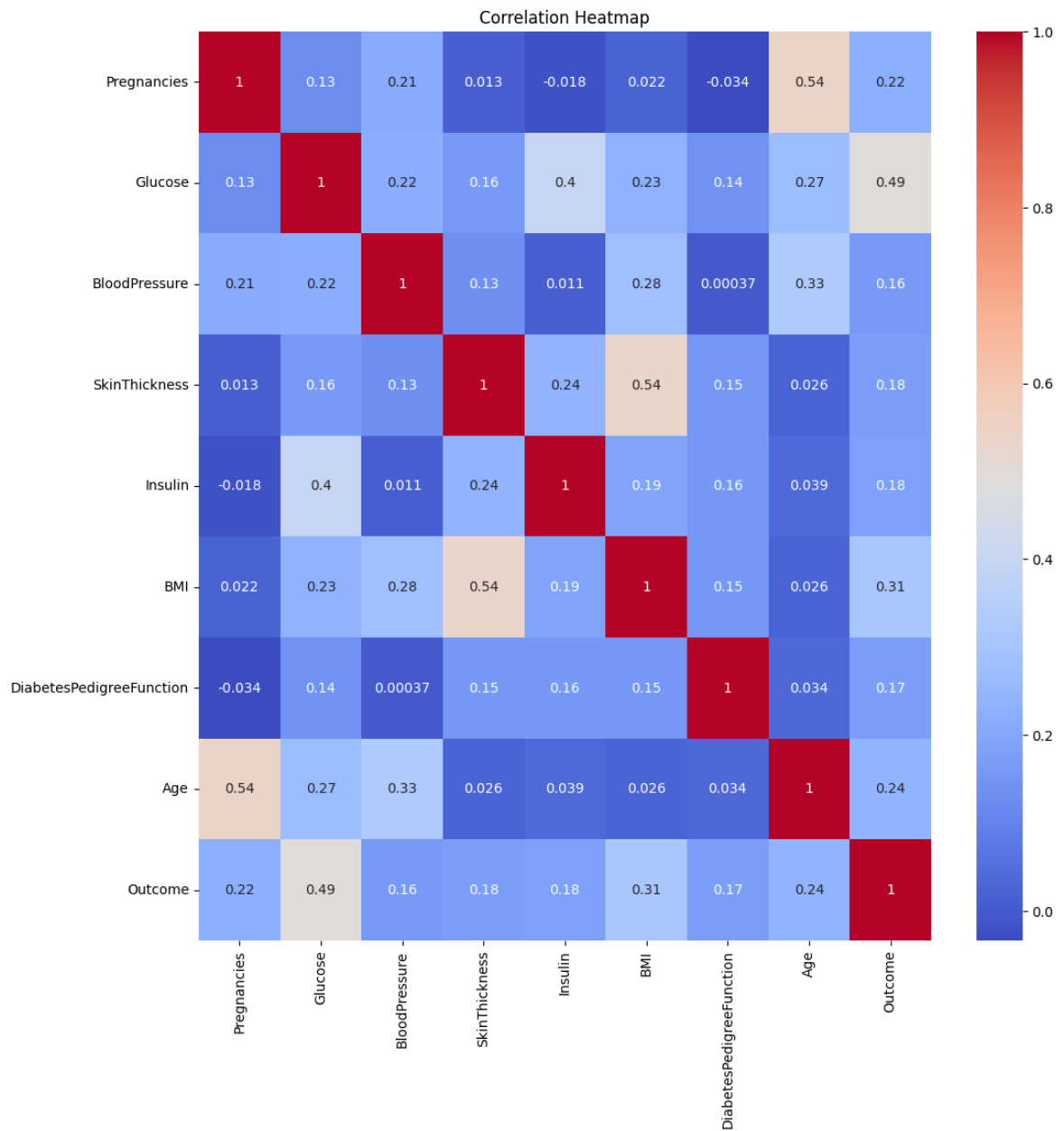


Diabetes Pedigree Function (DPF) calculates diabetes likelihood depending on the subject's age and his/her diabetic family history. From the boxplot, the patients with lower DPF, are much less likely to have diabetes. The patients with higher DPF, are much more likely to have diabetes. In the violinplot, 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

```
In [ ]: #correlation heatmap
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm').set_title('Correlation Heatmap')
```

```
Out[ ]: Text(0.5, 1.0, 'Correlation Heatmap')
```



Train Test Split

```
In [ ]: 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
```

Diabetes Prediction

For predictiong the diabetes, I will be using the following algorithms:

1. Logistic Regression
2. Random Forest Classifier
3. Support Vector Machine

Logistic Regression

```
In [ ]: #building model
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr
```

```
Out[ ]: ▼ LogisticRegression
LogisticRegression()
```

```
In [ ]: #training the model
lr.fit(X_train,y_train)
#training accuracy
lr.score(X_train,y_train)
```

```
Out[ ]: 0.7719869706840391
```

```
In [ ]: #predicted outcomes
lr_pred = lr.predict(X_test)
```

Random Forest Classifier

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

```
Out[ ]: ▼ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
In [ ]: #training model
rfc.fit(X_train, y_train)
#training accuracy
rfc.score(X_train, y_train)
```

```
Out[ ]: 1.0
```

```
In [ ]: #predicted outcomes
rfc_pred = rfc.predict(X_test)
```

Support Vector Machine (SVM)

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

```
Out[ ]: ▼ SVC
SVC(kernel='linear', random_state=0)
```

```
In [ ]: #training the model
svm.fit(X_train, y_train)
```

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

Out[]: 0.7597402597402597

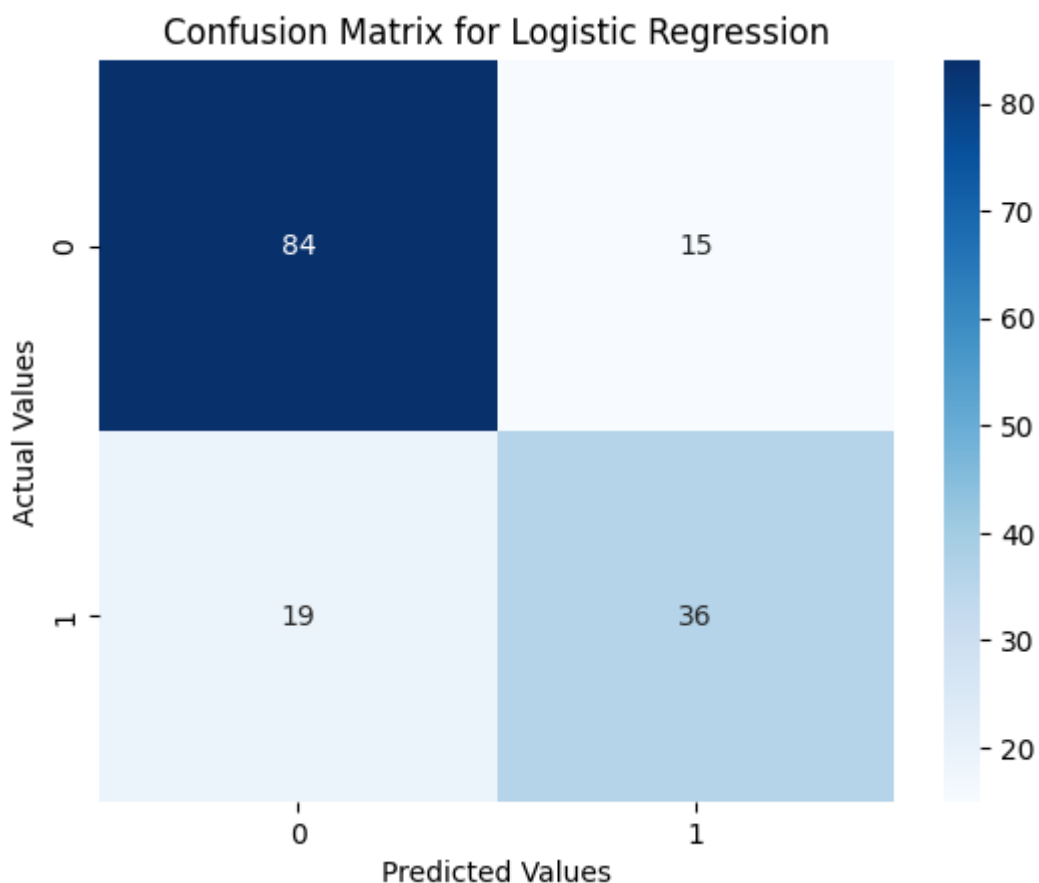
```
In [ ]: #predicting outcomes  
svm_pred = svm.predict(X_test)
```

Model Evaluation

Evaluating Logistic Regression Model

Confusion Matrix Heatmap

```
In [ ]: 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()
```

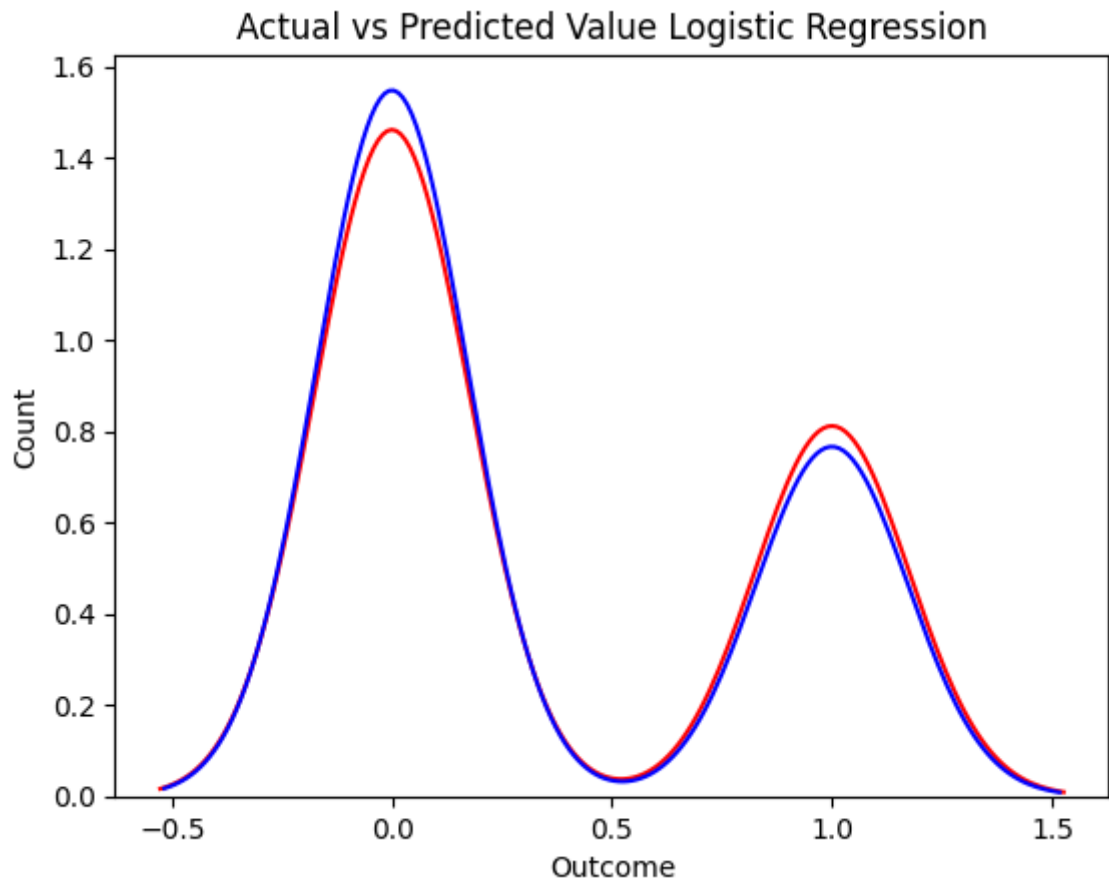


The diagonal boxes shows the count of true positives for each class. The predicted value is given on top while the actual value is given on the left side. The off-diagonal boxes shows the count of false positives.

Distribution plot

```
In [ ]: 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')
```

```
Out[ ]: 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.

Classification Report

```
In [ ]: from sklearn.metrics import classification_report
print(classification_report(y_test, lr_pred))
```

	precision	recall	f1-score	support
0	0.82	0.85	0.83	99
1	0.71	0.65	0.68	55
accuracy			0.78	154
macro avg	0.76	0.75	0.76	154
weighted avg	0.78	0.78	0.78	154

The model has as an average f1 score of 0.755 and accuracy of 78%.

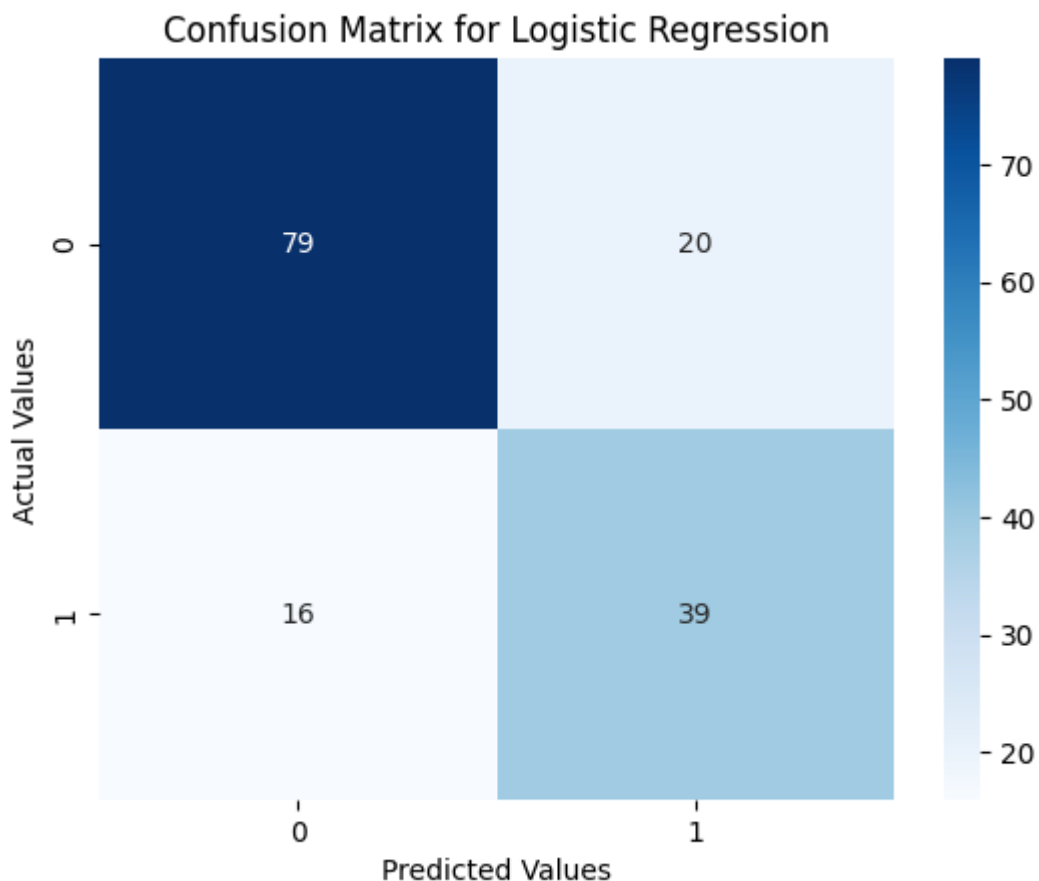

```
In [ ]: from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_error
print('Accuracy Score: ', accuracy_score(y_test, lr_pred))
print('Mean Absolute Error: ', mean_absolute_error(y_test, lr_pred))
print('Mean Squared Error: ', mean_squared_error(y_test, lr_pred))
print('R2 Score: ', r2_score(y_test, lr_pred))
```

Accuracy Score: 0.7792207792207793
Mean Absolute Error: 0.22077922077922077
Mean Squared Error: 0.22077922077922077
R2 Score: 0.038383838383838076

Evaluating Random Forest Classifier

Confusion Matrix Heatmap

```
In [ ]: 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()
```



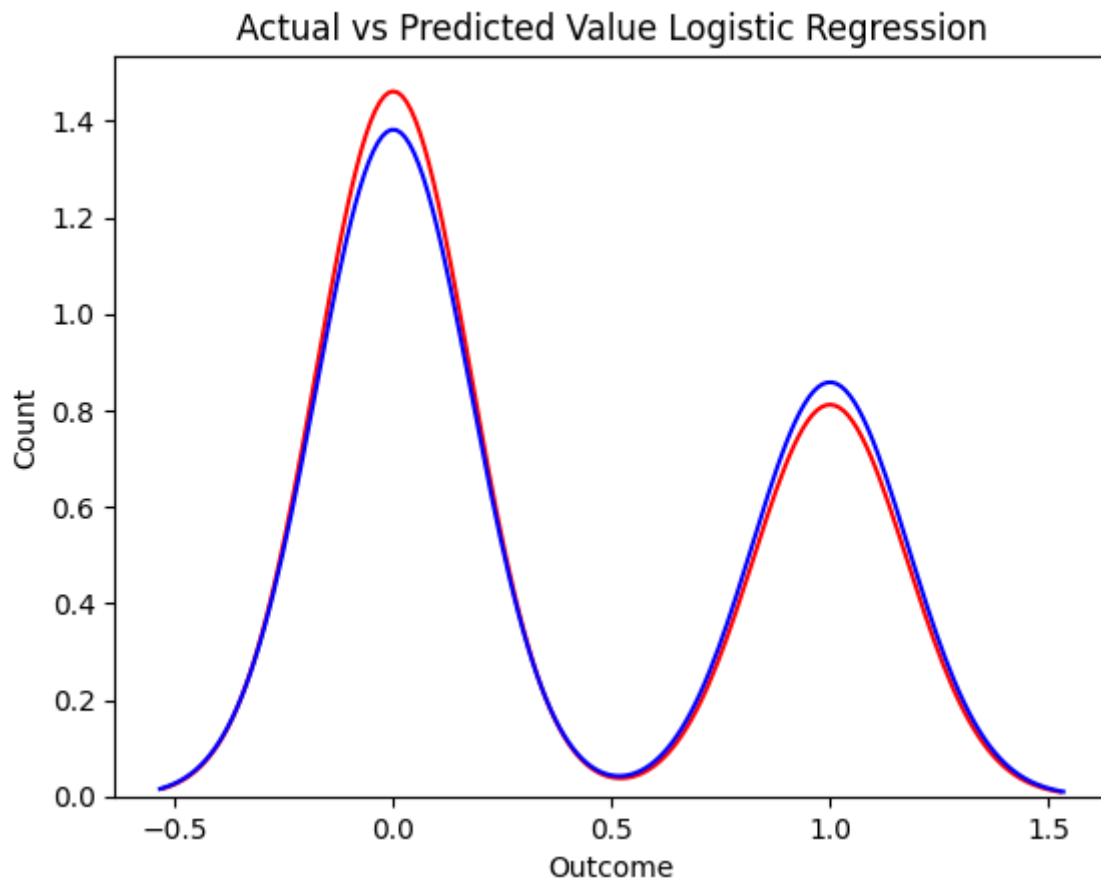
The diagonal boxes shows the count of true positives for each class. The predicted value is given on top while the actual value is given on the left side. The off-diagonal boxes shows the count of false positives.

Distribution Plot

```
In [ ]: 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')
```

Out[]: 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.

Classification Report

```
In [ ]: print(classification_report(y_test, rfc_pred))
```

	precision	recall	f1-score	support
0	0.83	0.80	0.81	99
1	0.66	0.71	0.68	55
accuracy			0.77	154
macro avg	0.75	0.75	0.75	154
weighted avg	0.77	0.77	0.77	154

The model has as an average f1 score of 0.745 and accuracy of 77% which less in comparison to Logistic Regression model.

```
In [ ]: print('Accuracy Score: ',accuracy_score(y_test,rfc_pred))
print('Mean Absolute Error: ',mean_absolute_error(y_test,rfc_pred))
```

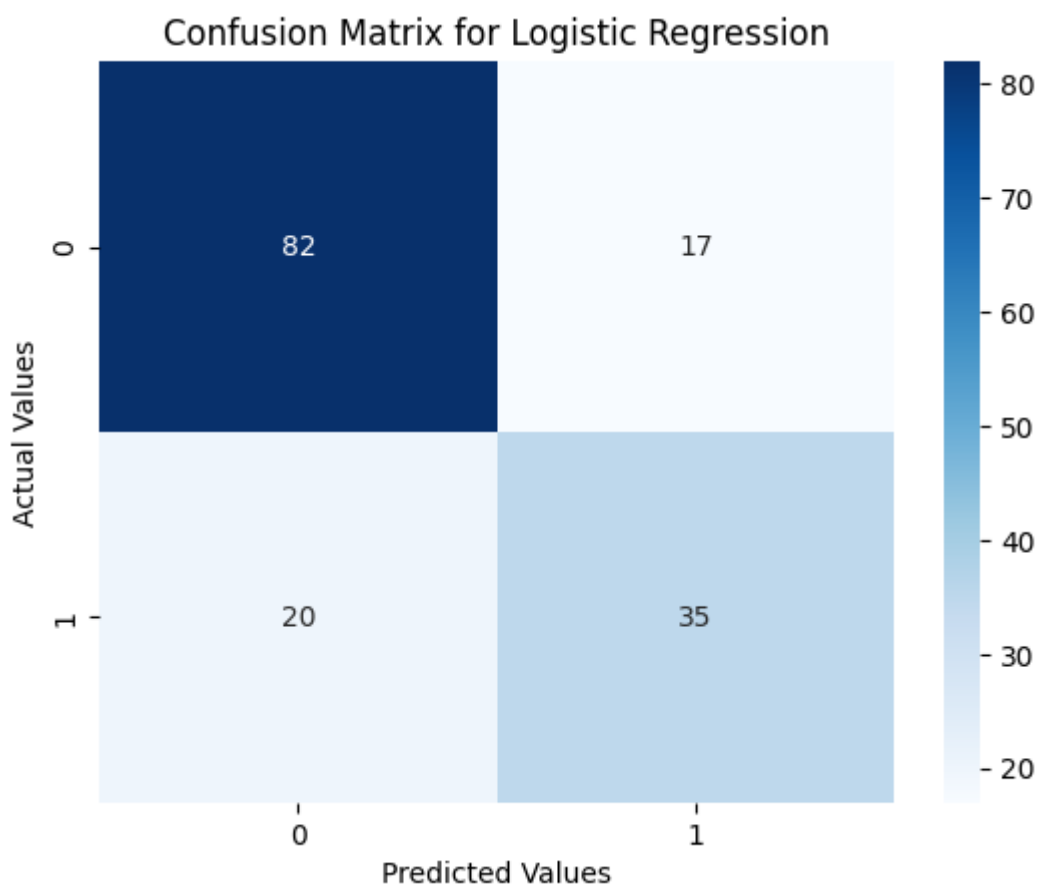
```
print('Mean Squared Error: ',mean_squared_error(y_test,rfc_pred))
print('R2 Score: ',r2_score(y_test,rfc_pred))
```

Accuracy Score: 0.7662337662337663
 Mean Absolute Error: 0.23376623376623376
 Mean Squared Error: 0.23376623376623376
 R2 Score: -0.01818181818181852

Evaluating SVM Model

Confusion Matrix Heatmap

```
In [ ]: 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()
```

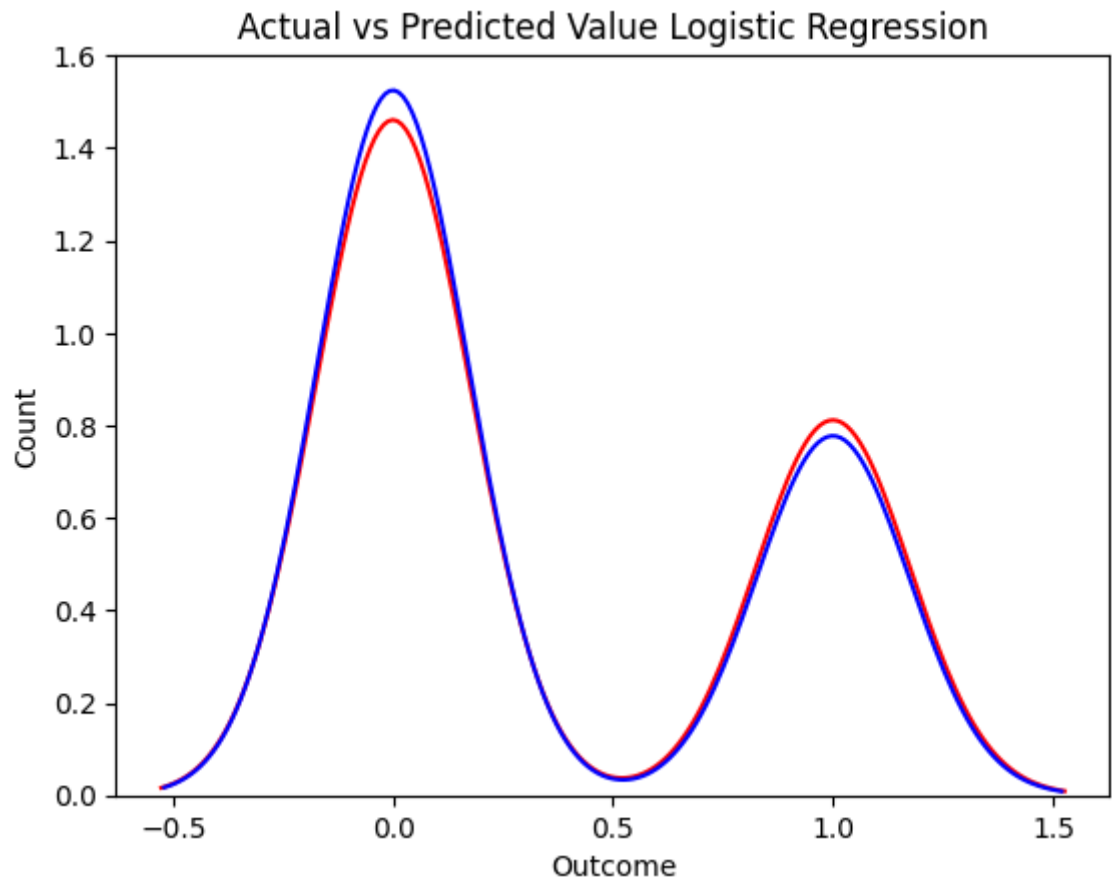


The diagonal boxes shows the count of true positives for each class. The predicted value is given on top while the actual value is given on the left side. The off-diagonal boxes shows the count of false positives.

Distribution Plot

```
In [ ]: 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')
```

```
Out[ ]: 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.

Classification Report

```
In [ ]: print(classification_report(y_test, rfc_pred))
```

	precision	recall	f1-score	support
0	0.83	0.80	0.81	99
1	0.66	0.71	0.68	55
accuracy			0.77	154
macro avg	0.75	0.75	0.75	154
weighted avg	0.77	0.77	0.77	154

The model has as an average f1 score of 0.745 and accuuracy of 77% which is equivalent to previous model.

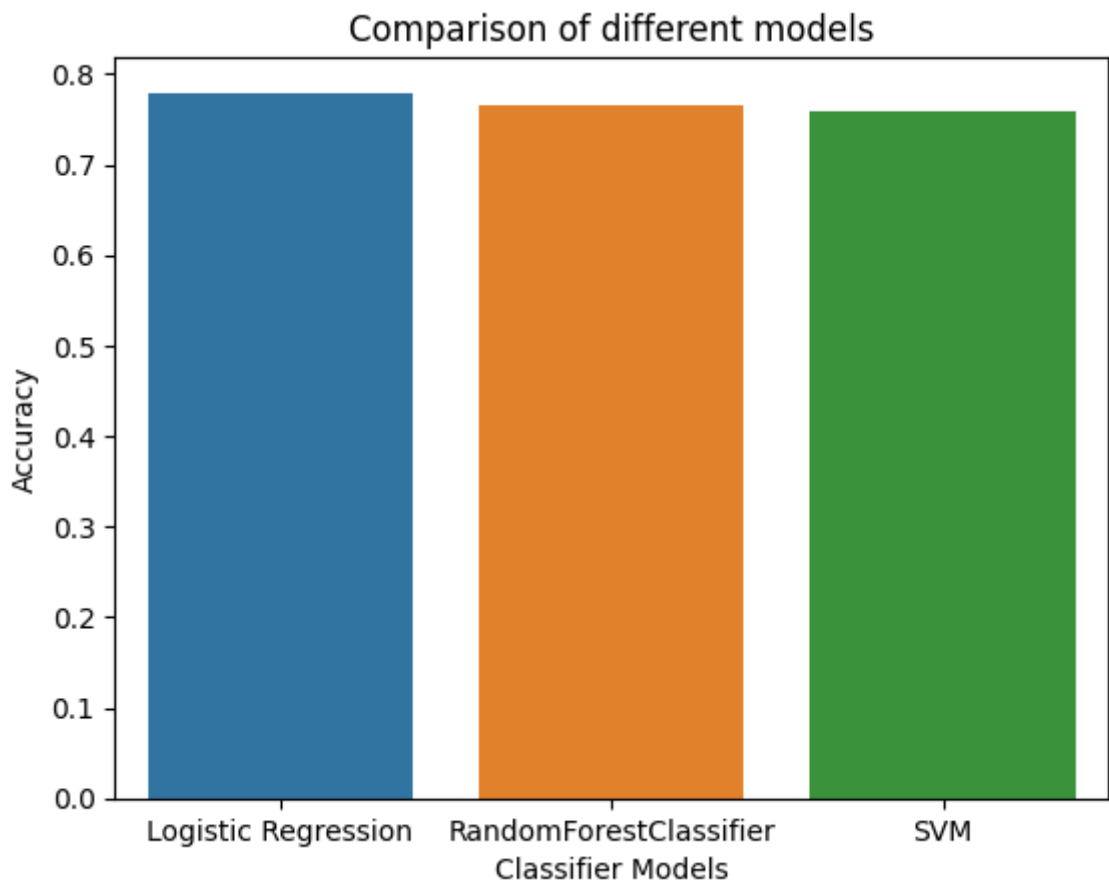
```
In [ ]: print('Accuracy Score: ',accuracy_score(y_test,svm_pred))
print('Mean Absolute Error: ',mean_absolute_error(y_test,svm_pred))
print('Mean Squared Error: ',mean_squared_error(y_test,svm_pred))
print('R2 Score: ',r2_score(y_test,svm_pred))
```

Accuracy Score: 0.7597402597402597
Mean Absolute Error: 0.24025974025974026
Mean Squared Error: 0.24025974025974026
R2 Score: -0.046464646464646764

Comparing the models

```
In [ ]: #comparing the accuracy of different models
sns.barplot(x=['Logistic Regression', 'RandomForestClassifier', 'SVM'], y=[0.779, 0.76, 0.76])
plt.xlabel('Classifier Models')
plt.ylabel('Accuracy')
plt.title('Comparison of different models')
```

Out[]: Text(0.5, 1.0, 'Comparison of different models')



Conclusion

From the exploratory data analysis, I have concluded that the risk of diabetes depends upon the following factors:

1. Glucose level
2. Number of pregnancies
3. Skin Thickness
4. Insulin level
5. BMI

With in increase in Glucose level, insulin level, BMI and number of pregnancies, the risk of diabetes increases. However, the number of pregnancies have strange effect of risk of

diabetes which couldn't be explained by the data. The risk of diabetes also increases with increase in skin thickness.

Coming to the classification models, Logistic Regression outperformed Random Forest and SVM with 78% accuracy. The accuracy of the model can be improved by increasing the size of the dataset. The dataset used for this project was very small and had only 768 rows.