CS60050 : Machine Learning

Report

ASSIGNMENT - 3 [E]

Prepared by:

Group No. 51

Shristi Singh (19CS10057)

Mayank Kumar (19CS30029)

November 10, 2021

I. Analysis of the dataset

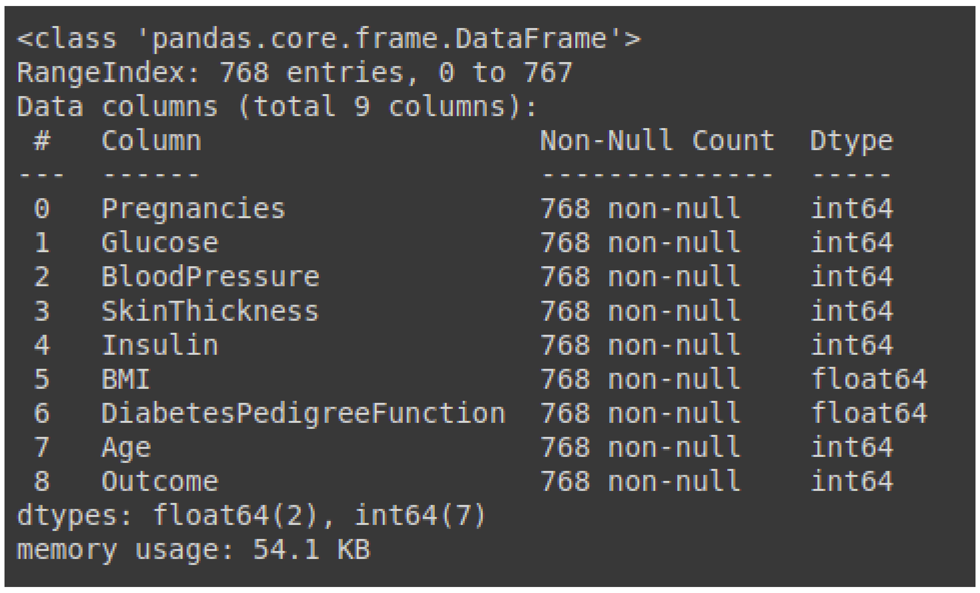
Shape of dataset: (768, 9)

Basic details about the dataset:

The dataset has 8 attributes that decide the Outcome column, i.e., presence (1) or absence (0) of diabetes.

- 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 skinfold 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 in years
- Outcome: Class variable (0 or 1)

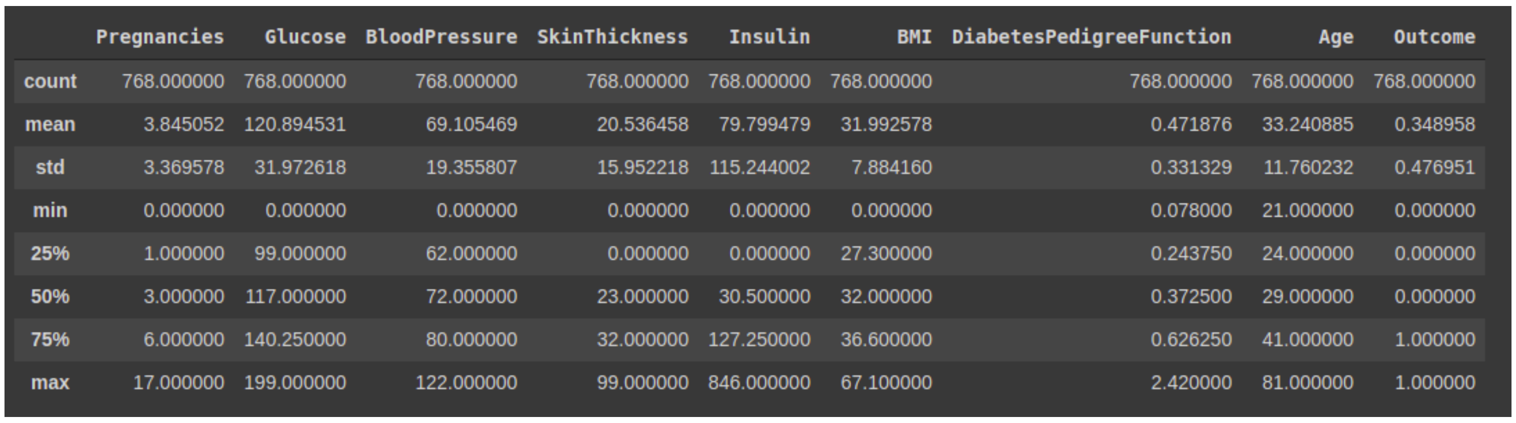
Overview of the dataset:



```
<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
```

BMI and DiabetesPedigreeFunction columns have float values, while all other columns have integer values.

Descriptive Statistics of the dataset:

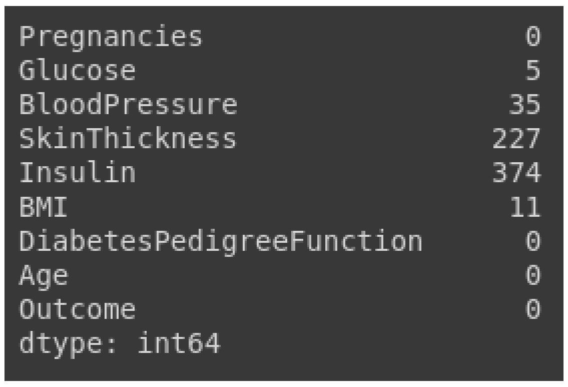


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578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

This is a bit strange because the description shows that the minimum values of Glucose, BloodPressure, SkinThickness, Insulin, and BMI as 0. We know that this is impossible or very rare. This suggests that the null values might have been replaced by zeros in the dataset.

Missing Data:

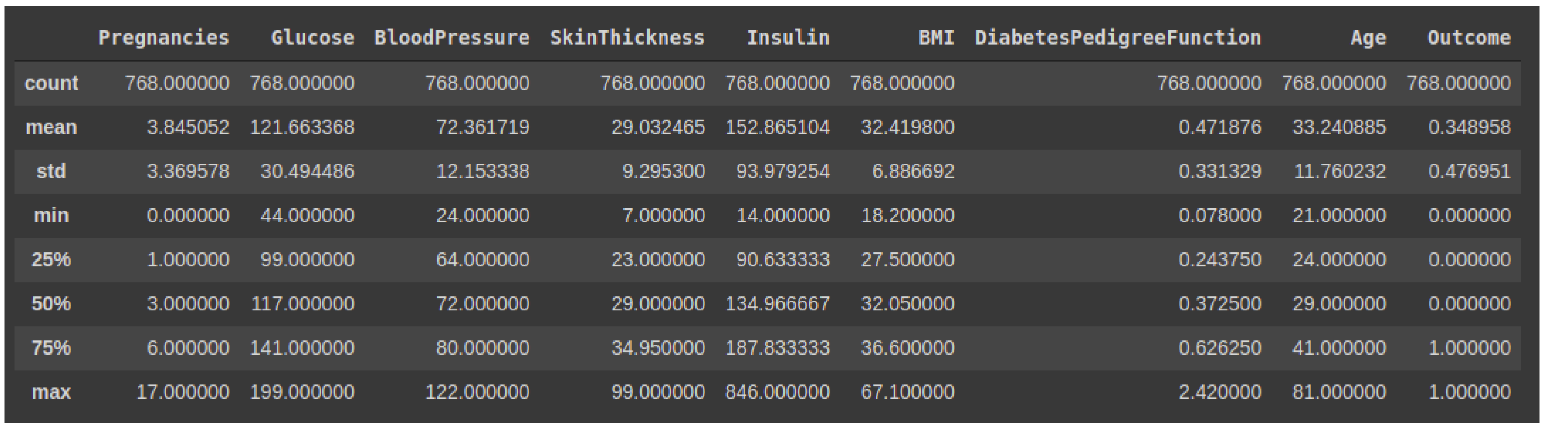
Upon replacing the zeros in Glucose, BloodPressure, SkinThickness, Insulin, and BMI by NaN, we can see that there are, in fact, many rows having zero value.



```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

Even after fasting, glucose levels can not be as low as zero. No living human can have a diastolic blood pressure of zero. Skin Thickness of less than 10 mm is not possible as even 10 mm is an extreme case. In a rare situation, one can have zero insulin, but there are 374 zero insulin values, which comprise almost 50% of the dataset. Zero BMI is also not possible even in an extremely underweight person. These suggest that the readings are invalid or missing, and we need to replace them with valid values. Some columns have almost half of their values missing, so we would use an appropriate method to fill those rows appropriately. One of the ways to ensure if the null values have been filled decently is by checking the values for different statistical metrics before and after this step.

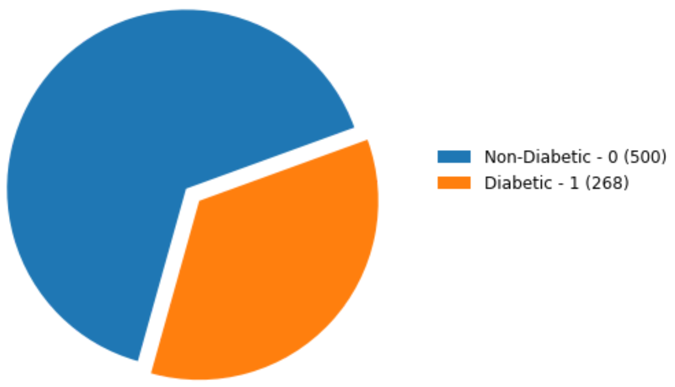
Descriptive Statistics of the dataset after cleaning:



	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	121.663368	72.361719	29.032465	152.865104	32.419800	0.471876	33.240885	0.348958
std	3.369578	30.494486	12.153338	9.295300	93.979254	6.886692	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	64.000000	23.000000	90.633333	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	29.000000	134.966667	32.050000	0.372500	29.000000	0.000000
75%	6.000000	141.000000	80.000000	34.950000	187.833333	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

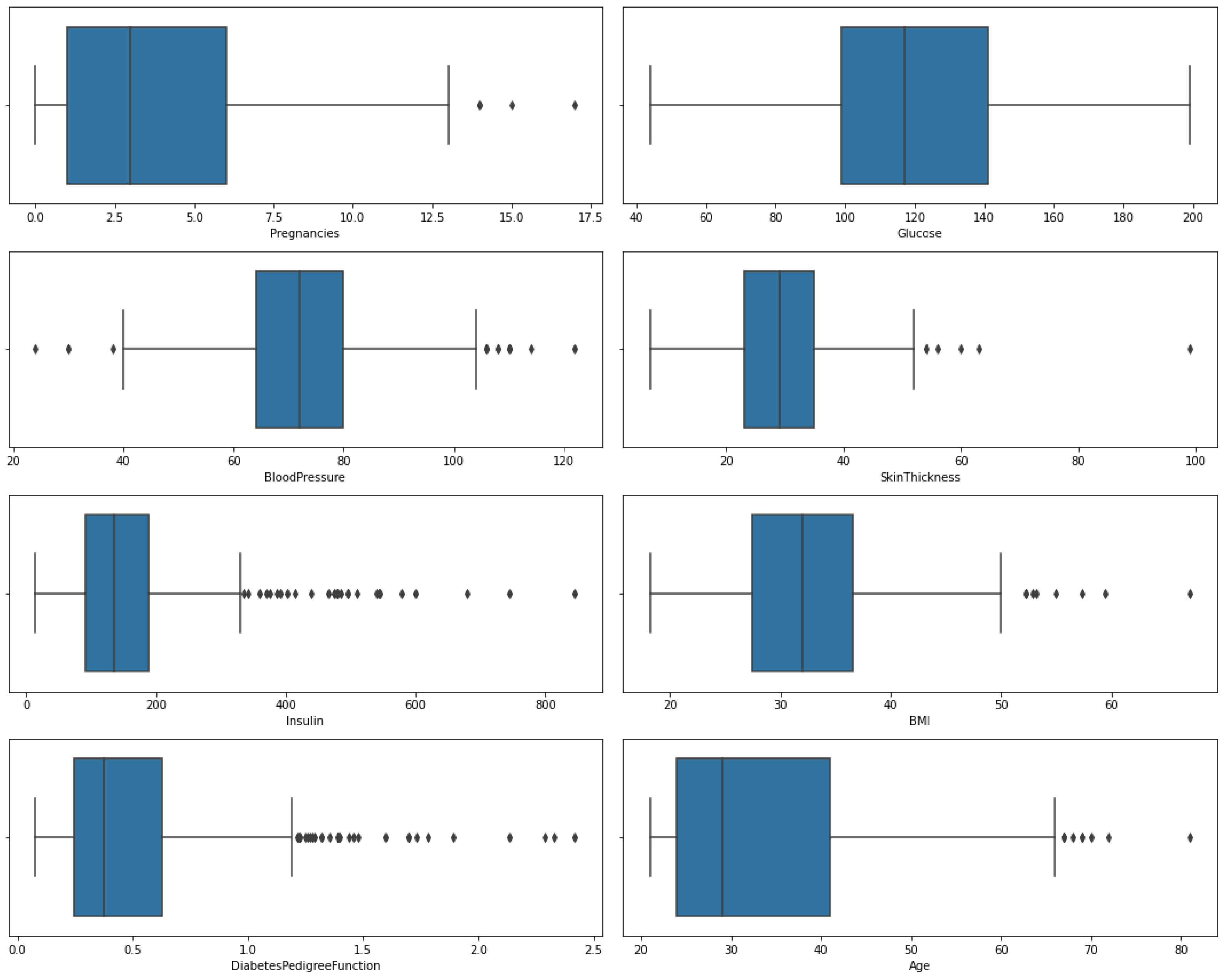
Here, we can see that the minimum values for the aforementioned columns are no longer zero but a positive constant. Also, there is very little change in metrics that evaluate statistics like mean, median, std, etc., and this ensures that the method used for replacing missing data worked fine.

Count for Target Values (Class Labels):



Most of the rows have their Outcome as 0, i.e., most of them are non-diabetic.

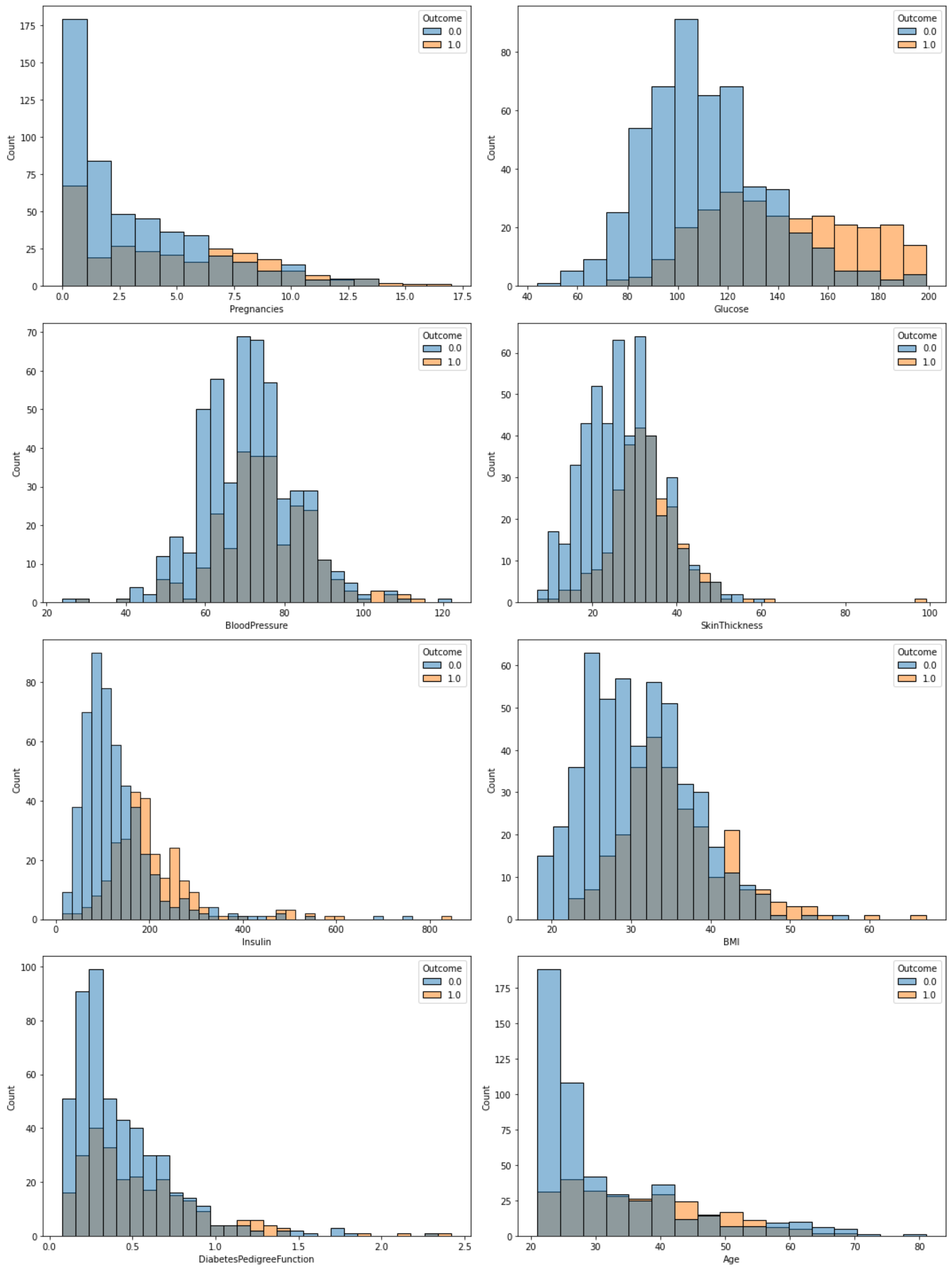
Distribution of attribute values (univariate):



The boxplot displays the distribution of data for an attribute based on a five-number summary - minimum, first quartile, median, third quartile, and maximum. It helps in analyzing if data is symmetrical or skewed or how they are grouped. In the plots, we can see the range in which an attribute's value mostly lies, along with outliers in some of the columns. We would have to overcome the presence of outliers so that it does not have a negative effect on the model's performance. These outliers cannot be removed directly, as seen in the box plot, because an extreme value might give us crucial information to decide if the person is diabetic or not.

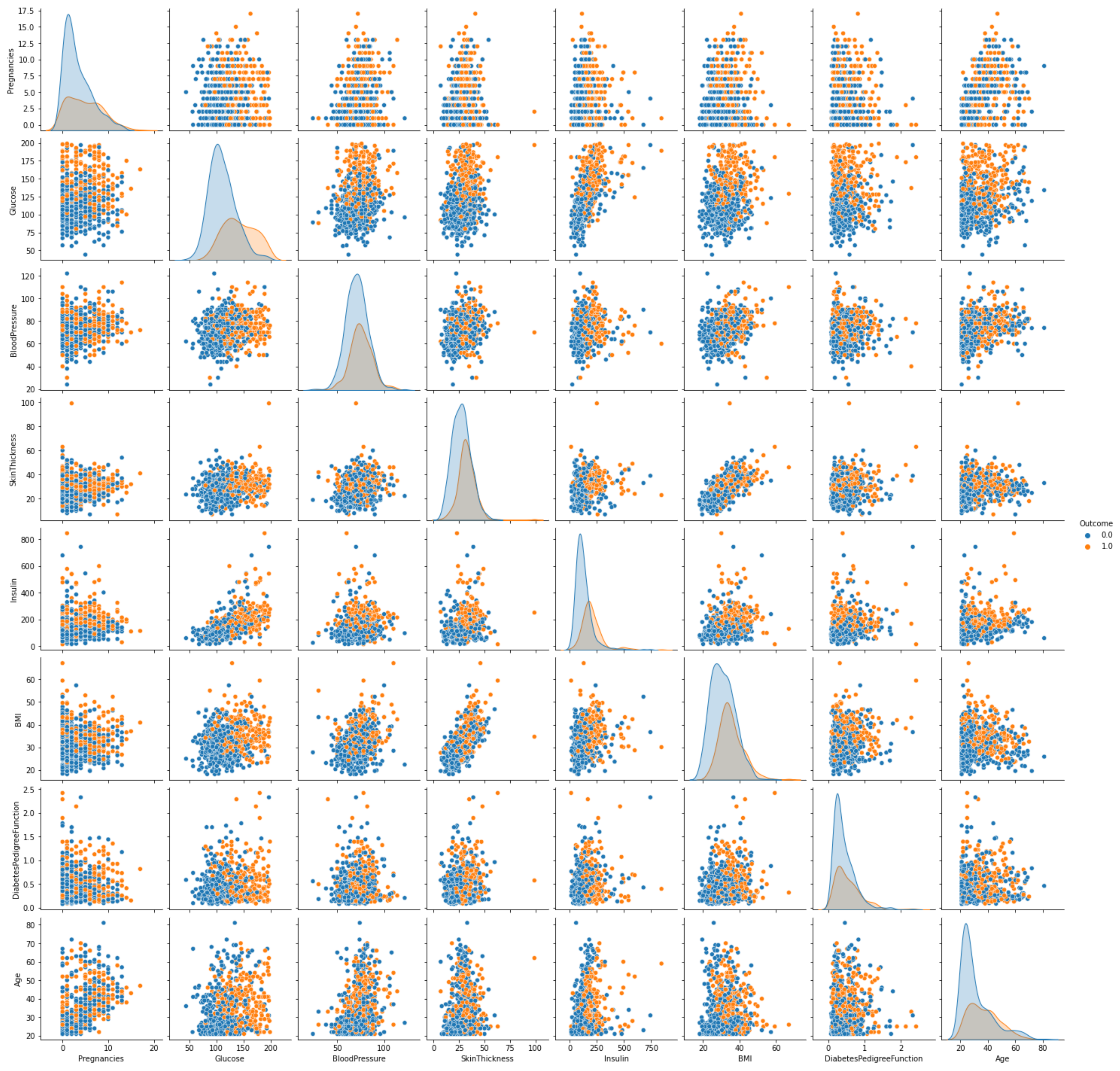
Distribution of attribute values (bivariate):

Hist Plot:



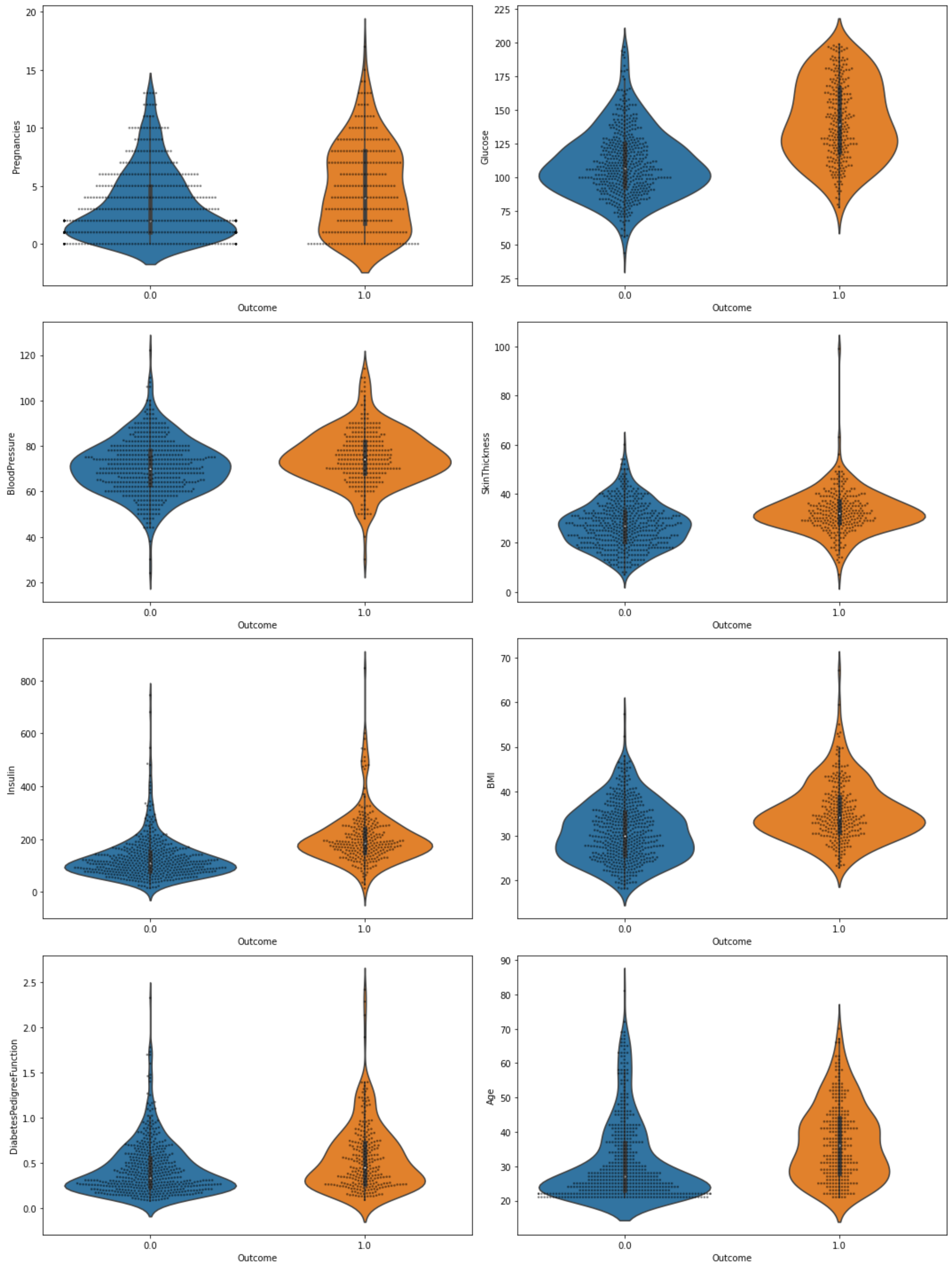
These bivariate plots describe the distribution of data in a detailed manner according to the two outcomes - 1 (diabetic) and 0 (non-diabetic). For most of the attributes given here, it is clear that the probability of being diabetic is high if its value is higher.

Pair Plot:



These are scatter plots that plot multiple pairwise bivariate distributions in a dataset. This plot gives us valuable insights and roughly tries to show what should be expected as we use PCA to reduce the dimensionality and work with two attributes. The distribution along with the diagonal shows how the values for two outcomes vary for the attributes.

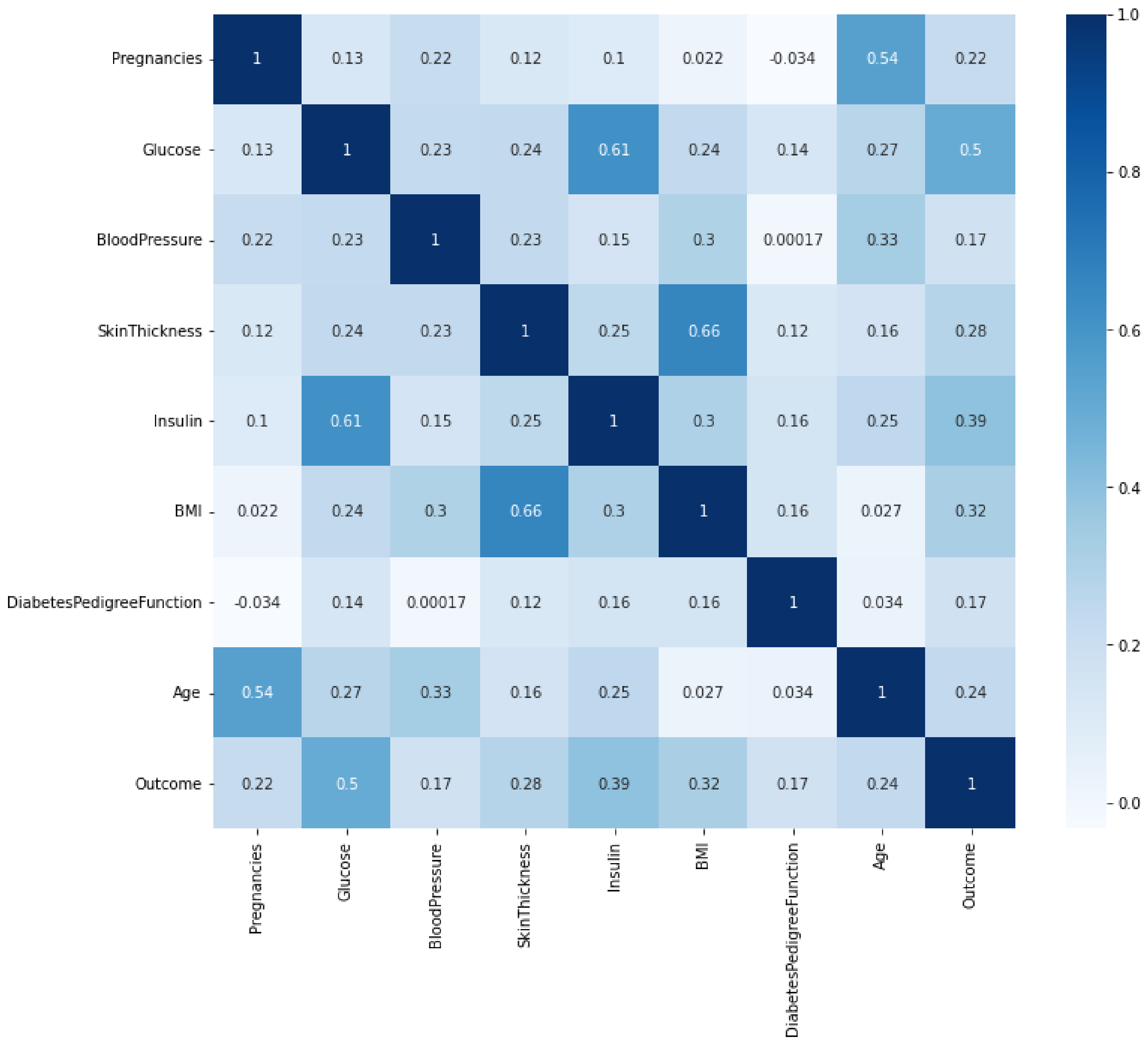
Violin and Swarm Plot:



These violin plots show the probability density of the data for the two outcomes at different values while being smoothed by a kernel density estimator. Here, the width of each curve corresponds to the approximate frequency of data points in each region. The swarm plot is overlaid onto the violin plot in order to see the distribution and samples of the points comprising that distribution.

Heatmap (Multi-Collinearity of Attributes):

Heatmap is used to better visualize the magnitude of a phenomenon in a dataset and assist by directing towards areas that matter the most through color-codings.



Among the attributes, it can be seen that there is a very strong correlation between Age & Pregnancies; Glucose & Insulin; and BMI & SkinThickness. Also, as all the attributes positively correlate with Outcome, none of them could negatively impact while predicting results.

II. Procedure

Missing Data:

We found that there are 5 values missing in Glucose, 35 in BloodPressure, 227 in SkinThickness, 374 in Insulin, and 11 in BMI. This comprises a lot from the dataset, and removing them would mean loss of valuable data. They can also not be replaced by any statistical metric like median or mean, as this would generate a bias for final results towards the measure. So, we will use KNNImputer - a sklearn class used to fill the missing values in a dataset. This works on the basic approach of KNN algorithm, where we predict the missing value in reference to the mean of the neighbors.

Scaling Dataset:

We know that PCA seeks the direction that maximizes the variance, and scaling the data differently will change the PCA vectors. As the attributes do not have the same units of measure, scaling is needed so that PCA does not emphasize those attributes with higher variances.

Normalizing the dataset could skew the distribution and obscure the physical meaning of the variables. If there is an anomaly in any random variable, the normalization will skew it massively, and hence, we might end up with undesirable PCA vectors. On the other hand, standardizing removes the mean and scales the dataset to unit variance. We will consider standardization before applying PCA.

Now, the StandardScaler gets influenced by outliers as it involves the estimation of empirical mean and standard deviation of each feature, and we have seen from the box plot that there are certainly some outliers in our dataset. So, we will use RobustScaler as it scales the data according to the quantile range. Using a different statistic makes this scaler robust to the outliers.

In the case of LDA, mathematically, it finds a set of discriminant axes by computing eigenvectors of $W^{-1}B$, where W and B are within- and between-class scatter matrices. Although the scatter matrices and even the eigenvectors will be different depending on whether the features were scaled or not, the eigenvalues, which show how much the classes are separated, will remain precisely the same. Thus, despite the transformations, the feature scaling does not affect the overall results of an LDA. We will use the scaled datasets in this case as well for no particular reason, as it would not change the results anyway, even if we use non-scaled data.

Splitting Dataset:

As mentioned, the dataset is randomly split, and the ratio of train, validation and test splits should be 70:10:20. At first, we will split in ratio 80:20 (test_size = 20/100 = 0.2), considering the smaller portion as the test set. Then, we will split in ratio 70:10 (test_size = 10/80 = 0.125), where the bigger portion would be the train set while the smaller one would be the validation set.

Reducing to two dimensions (PCA):

The feature dimension of the train, validation and test datasets is reduced into a two-dimensional feature space by applying Principle Component Analysis (PCA). We will use PCA from the sklearn package (sklearn.decomposition.PCA) with n_components = 2 to implement the same. Then, the reduced training data is plotted on a 2d plane such that the red color shows the non-diabetic samples (0) while the blue color shows the diabetic samples (1).

Then, we use the SVM classifier implemented using SVC from the sklearn library. A function is defined earlier, which takes the respective parameters for the SVC class along with the train set, trains an SVC model using them, and returns the model and its validation accuracy. This is done for ease of computation in later steps.

For hyperparameter tuning, we consider kernel, gamma, C and degree of the SVC class. We vary the kernel for {'linear', 'poly', 'rbf', 'sigmoid'}, gamma for {'scale', 'auto'}, C for {0.1, 1, 10, 100} and degrees for {0, 1, 2, 3, 4, 5, 6}. Do note that we have used the values for C and degrees in descending order while finding our optimal model so that we get the simplest model having the highest validation accuracy.

The degree hyperparameter is varied only for the 'poly' kernel and defaulted at 3 for others as it is ignored by all other kernels. Also, in the 'poly' kernel, we will not consider beyond degree 4 if C is 100 and beyond degree 5 if C is 10, as this takes a lot of computation time without much difference in results. We adjudge a model as best if it has the highest validation accuracy, and use it to compute the test accuracy.

Reducing to one dimension (LDA):

The feature dimension of the train, validation and test datasets is now reduced into a one-dimensional feature space by applying Linear Discriminant Analysis (LDA). We will use LinearDiscriminantAnalysis from the sklearn package (sklearn.discriminant_analysis.LinearDiscriminantAnalysis) with n_components = 1 to implement the same. Then, the reduced training data is plotted such that the red color shows the non-diabetic samples (0) while the blue color shows the diabetic samples (1). Since it is one-dimensional data, there may be overlapping while plotting, and it won't be shown in one line. So, we will also plot another graph with two lines for the two outcomes to understand the variations properly.

Similar to what we did in PCA, we will use the SVM classifier and tune the hyperparameters. The kernel, gamma, C and degree of the SVC class are varied in the same way as earlier. We adjudge a model as best if it has the highest validation accuracy, and use it to compute the test accuracy.

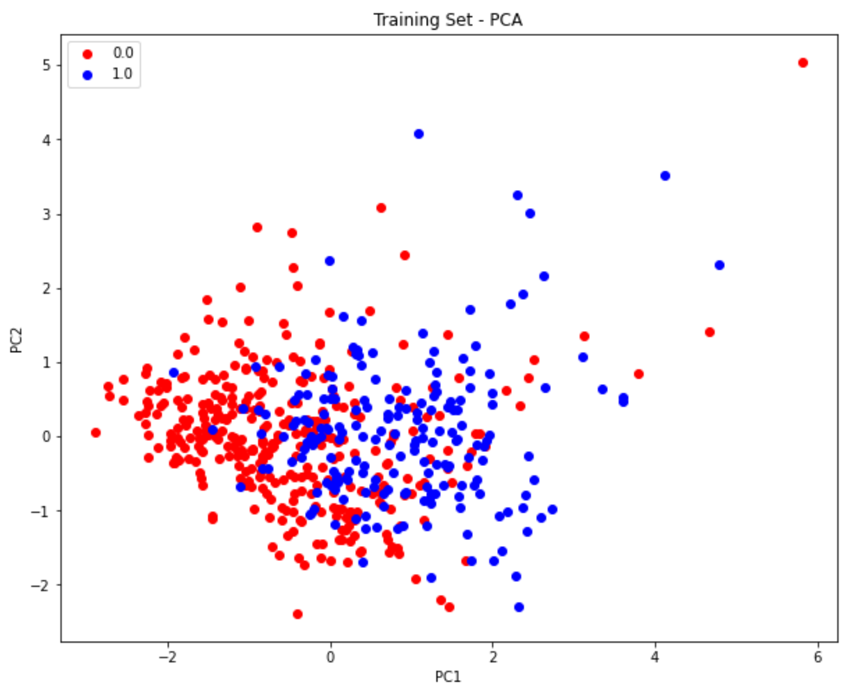
Difference between final test accuracies:

To compare the difference, we will have a look at the accuracy and confusion matrix on the test set for the two methods. We will also plot the predictions made by the two models. These will give us an idea about how the predictions in the two cases differ from each other and the actual outcome.

III. Results and Analysis

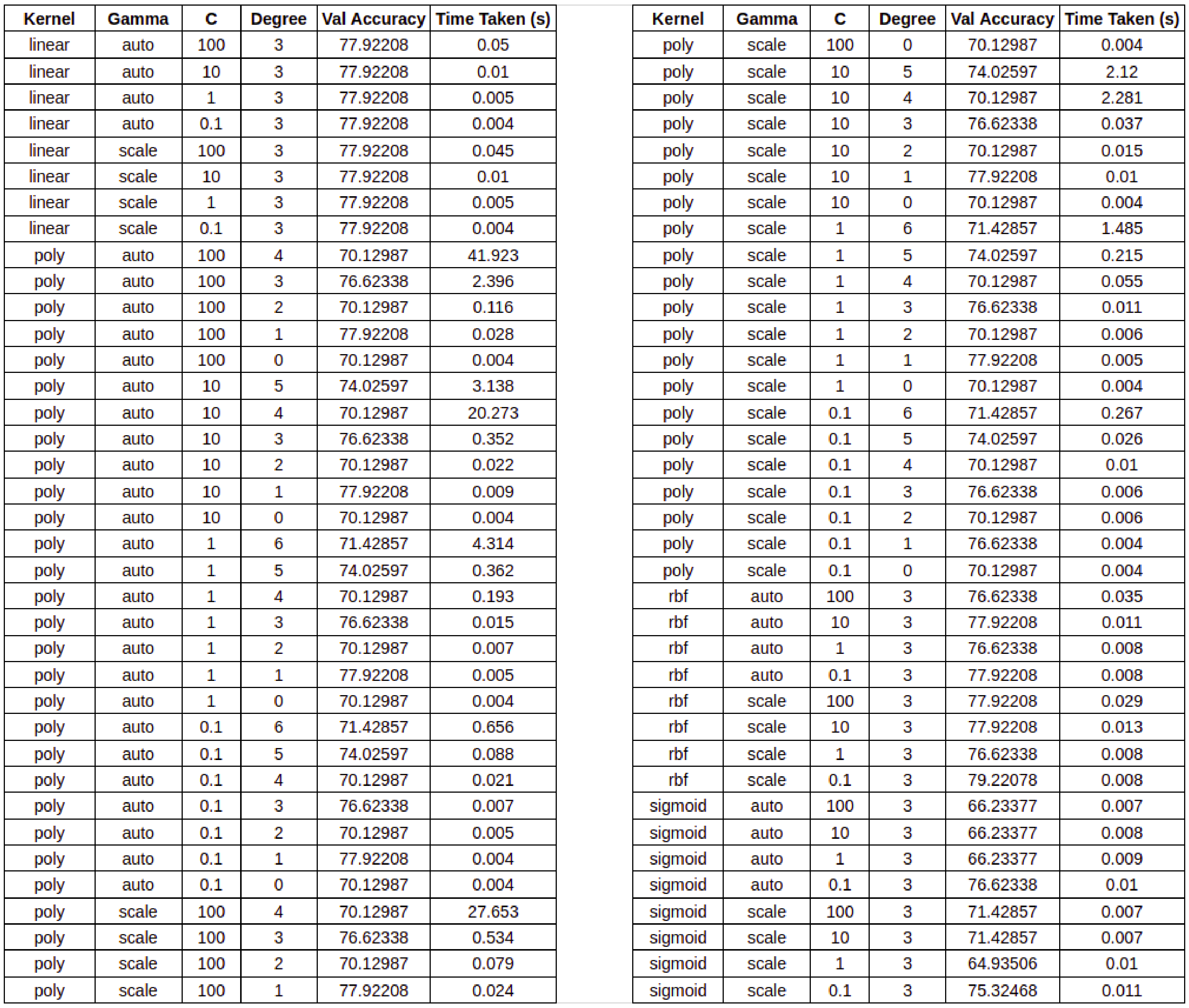
Reducing to two dimensions (PCA):

The plot obtained for reduced training data is shown below.



Here, the red color shows the non-diabetic samples (0), while the blue color shows the diabetic samples (1).

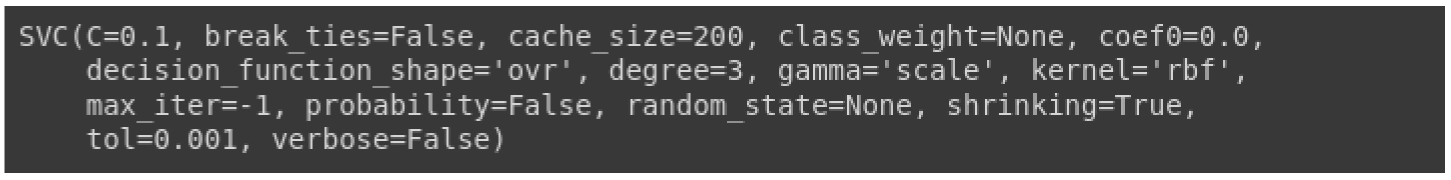
The validation accuracies achieved while tuning hyperparameters of the SVC class are as follows:



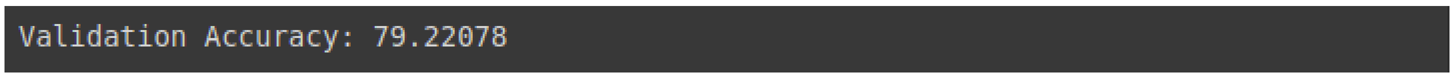
Kernel	Gamma	C	Degree	Val Accuracy	Time Taken (s)
linear	auto	100	3	77.92208	0.05
linear	auto	10	3	77.92208	0.01
linear	auto	1	3	77.92208	0.005
linear	auto	0.1	3	77.92208	0.004
linear	scale	100	3	77.92208	0.045
linear	scale	10	3	77.92208	0.01
linear	scale	1	3	77.92208	0.005
linear	scale	0.1	3	77.92208	0.004
poly	auto	100	4	70.12987	41.923
poly	auto	100	3	76.62338	2.396
poly	auto	100	2	70.12987	0.116
poly	auto	100	1	77.92208	0.028
poly	auto	100	0	70.12987	0.004
poly	auto	10	5	74.02597	3.138
poly	auto	10	4	70.12987	20.273
poly	auto	10	3	76.62338	0.352
poly	auto	10	2	70.12987	0.022
poly	auto	10	1	77.92208	0.009
poly	auto	10	0	70.12987	0.004
poly	auto	1	6	71.42857	4.314
poly	auto	1	5	74.02597	0.362
poly	auto	1	4	70.12987	0.193
poly	auto	1	3	76.62338	0.015
poly	auto	1	2	70.12987	0.007
poly	auto	1	1	77.92208	0.005
poly	auto	1	0	70.12987	0.004
poly	auto	0.1	6	71.42857	0.656
poly	auto	0.1	5	74.02597	0.088
poly	auto	0.1	4	70.12987	0.021
poly	auto	0.1	3	76.62338	0.007
poly	auto	0.1	2	70.12987	0.005
poly	auto	0.1	1	77.92208	0.004
poly	auto	0.1	0	70.12987	0.004
poly	scale	100	4	70.12987	27.653
poly	scale	100	3	76.62338	0.534
poly	scale	100	2	70.12987	0.079
poly	scale	100	1	77.92208	0.024
poly	scale	100	0	70.12987	0.004
poly	scale	10	5	74.02597	2.12
poly	scale	10	4	70.12987	2.281
poly	scale	10	3	76.62338	0.037
poly	scale	10	2	70.12987	0.015
poly	scale	10	1	77.92208	0.01
poly	scale	10	0	70.12987	0.004
poly	scale	1	6	71.42857	1.485
poly	scale	1	5	74.02597	0.215
poly	scale	1	4	70.12987	0.055
poly	scale	1	3	76.62338	0.011
poly	scale	1	2	70.12987	0.006
poly	scale	1	1	77.92208	0.005
poly	scale	1	0	70.12987	0.004
poly	scale	0.1	6	71.42857	0.267
poly	scale	0.1	5	74.02597	0.026
poly	scale	0.1	4	70.12987	0.01
poly	scale	0.1	3	76.62338	0.006
poly	scale	0.1	2	70.12987	0.006
poly	scale	0.1	1	76.62338	0.004
poly	scale	0.1	0	70.12987	0.004
rbf	auto	100	3	76.62338	0.035
rbf	auto	10	3	77.92208	0.011
rbf	auto	1	3	76.62338	0.008
rbf	auto	0.1	3	77.92208	0.008
rbf	scale	100	3	77.92208	0.029
rbf	scale	10	3	77.92208	0.013
rbf	scale	1	3	76.62338	0.008
rbf	scale	0.1	3	79.22078	0.008
sigmoid	auto	100	3	66.23377	0.007
sigmoid	auto	10	3	66.23377	0.008
sigmoid	auto	1	3	66.23377	0.009
sigmoid	auto	0.1	3	76.62338	0.01
sigmoid	scale	100	3	71.42857	0.007
sigmoid	scale	10	3	71.42857	0.007
sigmoid	scale	1	3	64.93506	0.01
sigmoid	scale	0.1	3	75.32468	0.011

The table is also attached as a csv file with the name 'PCA_Accuracies.csv'.

From the table, we can see that the best model found and its validation accuracy are:

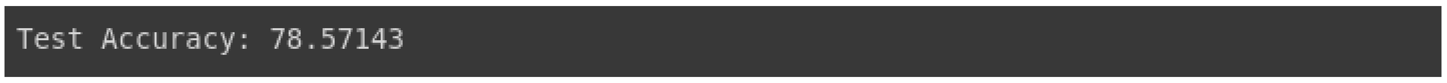


```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

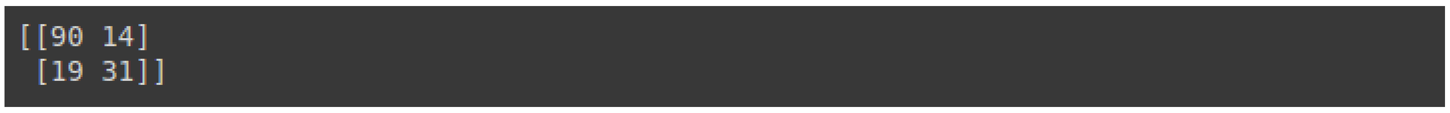


```
Validation Accuracy: 79.22078
```

Upon using this model to predict the output of the test set, the test accuracy achieved and the confusion matrix are:

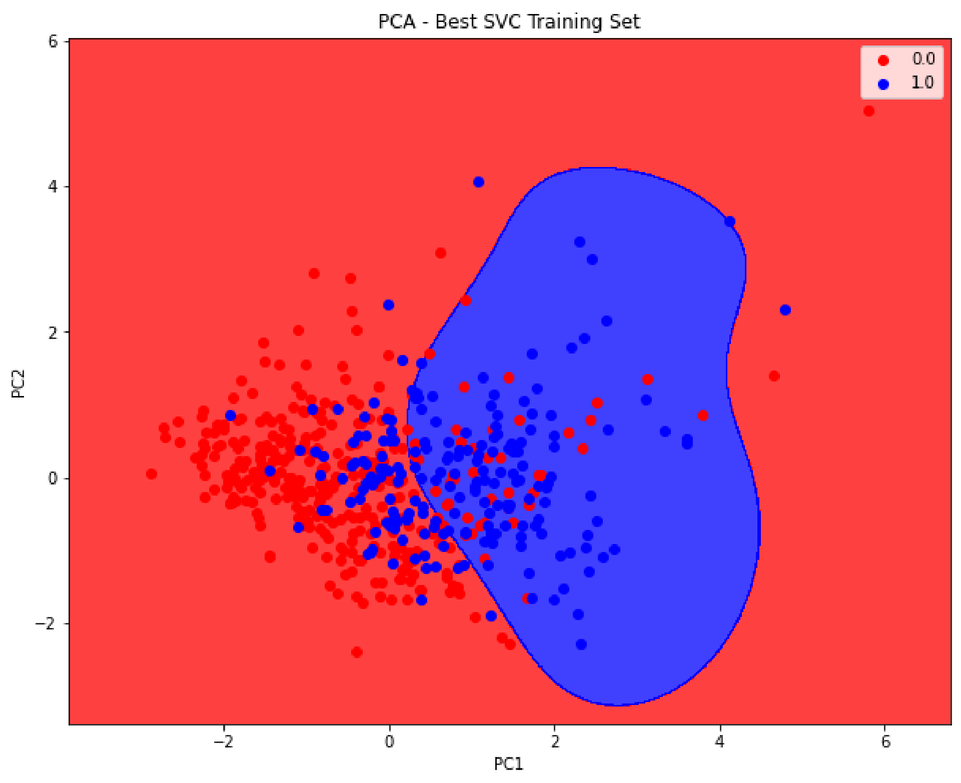


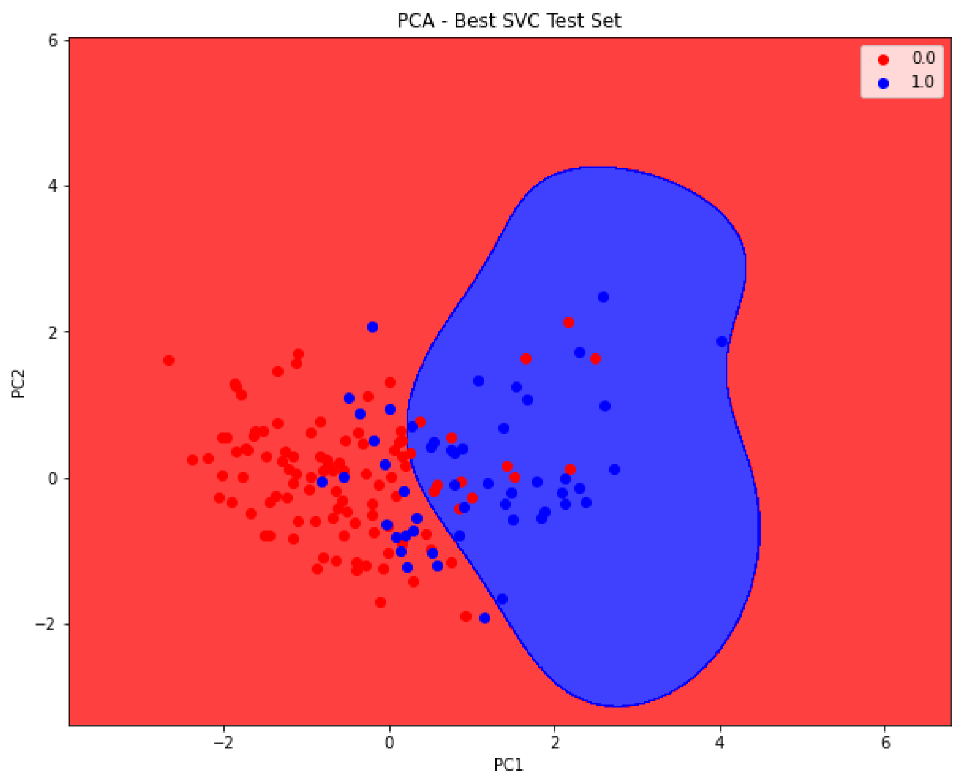
```
Test Accuracy: 78.57143
```



```
[[90 14]
 [19 31]]
```

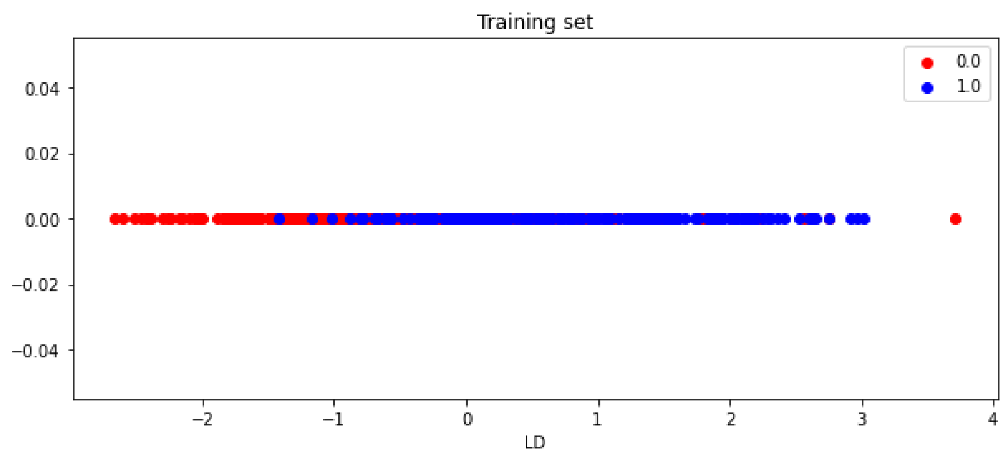
The output by the chosen model at various points on a 2d plane and the train set is shown below. The red area has the non-diabetic samples, while the blue area has the diabetic samples, as predicted by the model. In both plots (i.e., for train and test set), the blue dots lying in red area and the red dots in blue area are the misclassified samples.

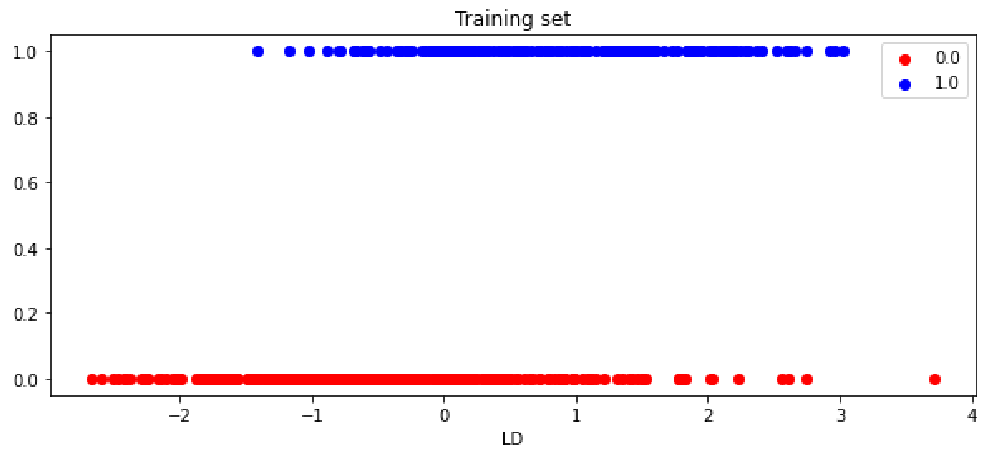




Reducing to one dimension (LDA):

The plot obtained for reduced training data is shown below. As the one-line plot might have overlapping, the two-line plot below plots points at 0 or 1 for the two outcomes to understand the variations properly.





Here, the red color shows the non-diabetic samples (0), while the blue color shows the diabetic ones (1). We can see that most non-diabetic samples have their value at the left-and-middle part of the plot (less than 1) while most diabetic ones are at the middle-and-right part of the plot (greater than 0). As expected, we have overlapping between the two outcomes in the plot at many points. Also, there are a few non-diabetic samples on the right side as well.

The results obtained while tuning hyperparameters of the SVC class are as follows:

Kernel	Gamma	C	Degree	Val Accuracy	Time Taken (s)
linear	auto	100	3	80.51948	0.022
linear	auto	10	3	80.51948	0.006
linear	auto	1	3	80.51948	0.003
linear	auto	0.1	3	80.51948	0.003
linear	scale	100	3	80.51948	0.021
linear	scale	10	3	80.51948	0.007
linear	scale	1	3	80.51948	0.004
linear	scale	0.1	3	80.51948	0.003
poly	auto	100	4	70.12987	22.584
poly	auto	100	3	75.32468	1.965
poly	auto	100	2	70.12987	0.055
poly	auto	100	1	80.51948	0.022
poly	auto	100	0	70.12987	0.004
poly	auto	10	5	75.32468	5.535
poly	auto	10	4	70.12987	0.391
poly	auto	10	3	75.32468	0.044
poly	auto	10	2	70.12987	0.01
poly	auto	10	1	80.51948	0.007
poly	auto	10	0	70.12987	0.003
poly	auto	1	6	70.12987	12.368
poly	auto	1	5	75.32468	3.395
poly	auto	1	4	70.12987	0.052
poly	auto	1	3	75.32468	0.01
poly	auto	1	2	70.12987	0.006
poly	auto	1	1	80.51948	0.004
poly	auto	1	0	70.12987	0.003
poly	auto	0.1	6	70.12987	0.236
poly	auto	0.1	5	75.32468	0.031
poly	auto	0.1	4	70.12987	0.01
poly	auto	0.1	3	75.32468	0.005
poly	auto	0.1	2	70.12987	0.004
poly	auto	0.1	1	80.51948	0.004
poly	auto	0.1	0	70.12987	0.003
poly	scale	100	4	70.12987	2.681
poly	scale	100	3	75.32468	0.757
poly	scale	100	2	70.12987	0.029
poly	scale	100	1	80.51948	0.017
poly	scale	100	0	70.12987	0.004
poly	scale	10	5	75.32468	2.224
poly	scale	10	4	70.12987	0.087
poly	scale	10	3	75.32468	0.017
poly	scale	10	2	70.12987	0.01
poly	scale	10	1	80.51948	0.006
poly	scale	10	0	70.12987	0.003
poly	scale	1	6	70.12987	0.225
poly	scale	1	5	75.32468	0.044
poly	scale	1	4	70.12987	0.013
poly	scale	1	3	75.32468	0.006
poly	scale	1	2	70.12987	0.005
poly	scale	1	1	80.51948	0.004
poly	scale	1	0	70.12987	0.003
poly	scale	0.1	6	70.12987	0.023
poly	scale	0.1	5	75.32468	0.01
poly	scale	0.1	4	70.12987	0.007
poly	scale	0.1	3	75.32468	0.004
poly	scale	0.1	2	70.12987	0.004
poly	scale	0.1	1	80.51948	0.004
poly	scale	0.1	0	70.12987	0.004
rbf	auto	100	3	80.51948	0.075
rbf	auto	10	3	80.51948	0.014
rbf	auto	1	3	79.22078	0.009
rbf	auto	0.1	3	80.51948	0.008
rbf	scale	100	3	80.51948	0.034
rbf	scale	10	3	79.22078	0.013
rbf	scale	1	3	80.51948	0.009
rbf	scale	0.1	3	80.51948	0.008
sigmoid	auto	100	3	67.53247	0.006
sigmoid	auto	10	3	68.83117	0.007
sigmoid	auto	1	3	67.53247	0.007
sigmoid	auto	0.1	3	68.83117	0.008
sigmoid	scale	100	3	68.83117	0.006
sigmoid	scale	10	3	68.83117	0.006
sigmoid	scale	1	3	68.83117	0.007
sigmoid	scale	0.1	3	70.12987	0.007

The table is also attached as a csv file with the name ‘LDA_accuracies.csv’.

From the table, we can see that the best model found and its validation accuracy are:

```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

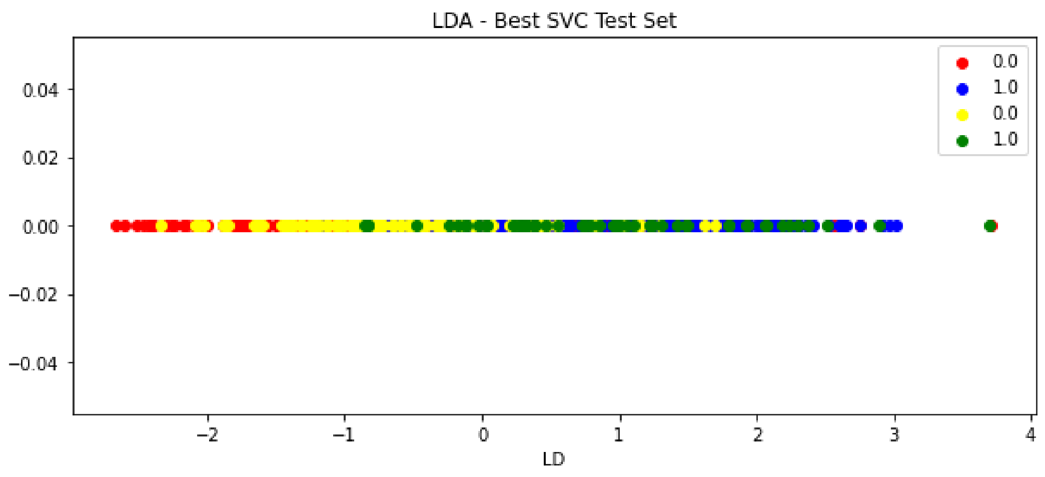
```
Validation Accuracy: 80.51948
```

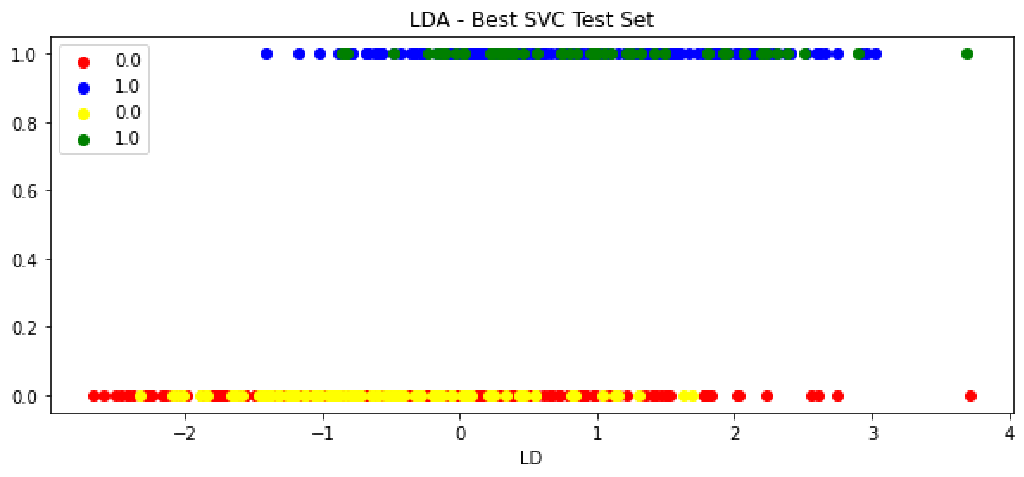
Upon using this model to predict the output of the test set, the test accuracy achieved and the confusion matrix are:

```
Test Accuracy: 80.51948
```

```
[[95  9]
 [21 29]]
```

The one-line plot with both train and test sets is shown below, where red and yellow colored points are non-diabetic samples while blue and green colored points are diabetic samples of train and test sets, respectively. Here, again we would have overlapping for points of the two classes, and so, we will plot the corresponding two-line plot as well.

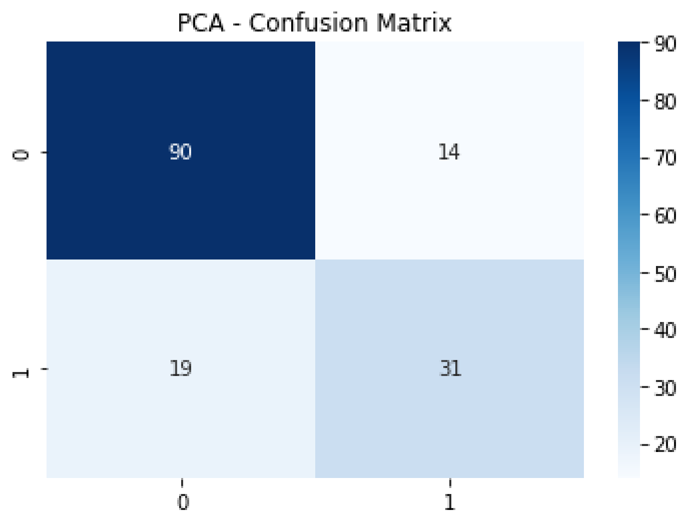


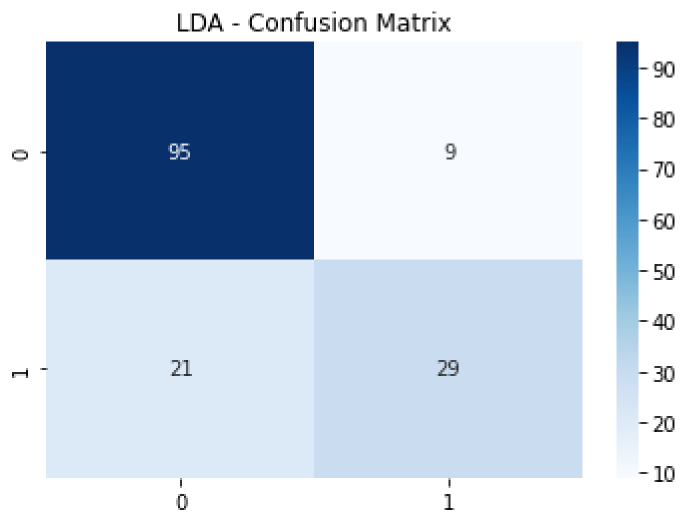


Comparing final test accuracies:

Test Accuracy and Confusion Matrix:

```
Test Accuracy after PCA: 78.57143
Test Accuracy after LDA: 80.51948
```

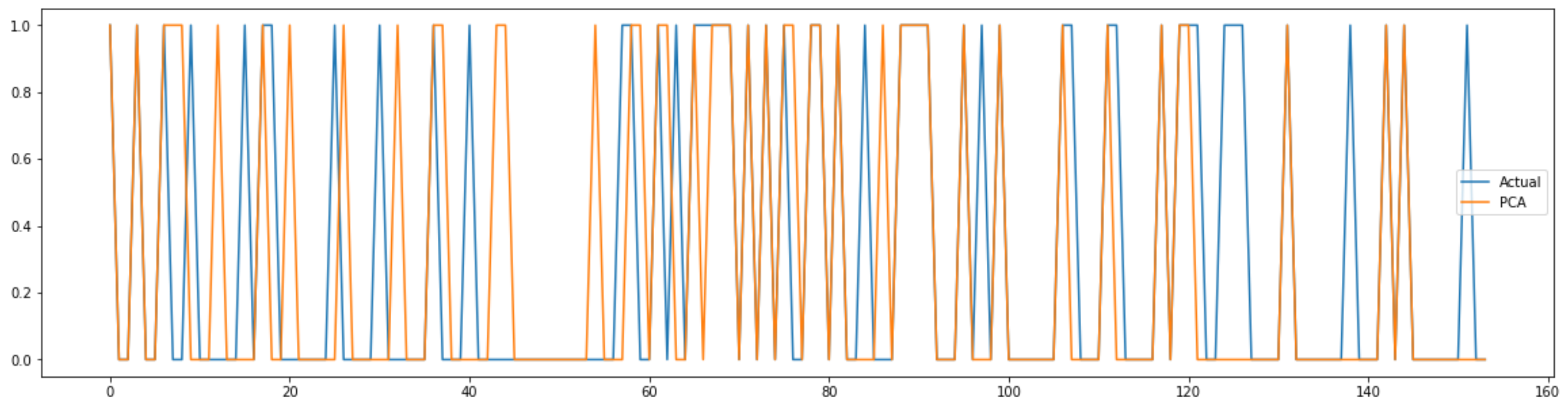




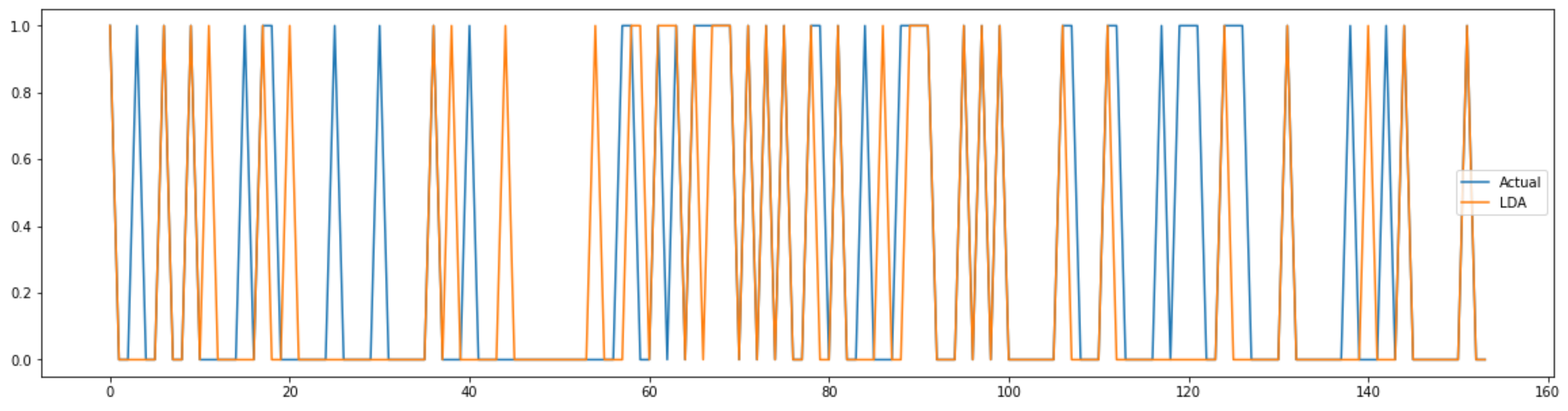
We can see that the test accuracies differ by nearly 2%. An interesting observation here is that the predictions made by model based on one-dimensional feature space (LDA) predicted 116 samples as non-diabetic, while the one based on two-dimensional feature space (PCA) predicted 109 samples as non-diabetic. Also, despite having a decent accuracy, both models are predicting diabetic samples with not much precision. Clearly, the model is getting biased towards predicting a sample as non-diabetic upon reducing the feature dimensions. From this, we can say that accuracy might not be a felicitous metric to evaluate model performance when the dataset has imbalanced classes.

Predictions and Actual Outcomes:

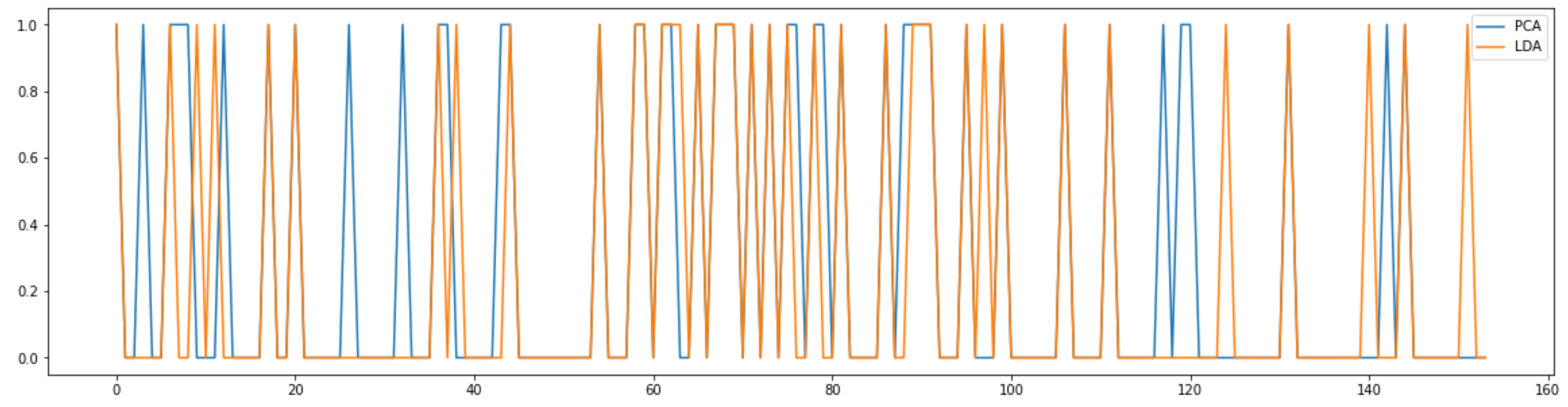
Actual (blue) vs PCA (orange)



Actual (blue) vs LDA (orange)



PCA (blue) vs LDA (orange)



From the above three plots and the confusion matrices, we can see that the predictions made by PCA and LDA are alike except for fifteen instances. Thus, both models are making errors mostly on similar samples. We have already seen that both of them are generalized towards predicting non-diabetic as their outcome. To compare the performance, it is crucial to analyze the salient features behind their working. The above-mentioned points about the two models can be because of two key reasons.

The dimension of the train set used in the two models differs by only 1 while being reduced from 8-dimensional feature space to 2 and 1 in PCA and LDA, respectively. So, the model might not have learned anything particularly different from the two train sets.

Both PCA and LDA rely on decomposing matrices of eigenvalues and eigenvectors. But PCA is an unsupervised technique that reduces dimensions by finding the maximum variance, while LDA is a supervised technique that reduces dimensions by finding a feature subspace that best optimizes class separability. Because of this, LDA is able to give similar or precisely, even better results as compared to PCA despite using only one feature dimension.

Note: Steps to run the notebook are in the README.md file.

IV. References

https://www.kaggle.com/mathchi/diabetes-data-set
https://scikit-learn.org/stable/modules/classes.html
https://moodlecse.iitkgp.ac.in/moodle/pluginfile.php/32205/mod_resource/content/1/LinearDiscrimination.pdf
https://moodlecse.iitkgp.ac.in/moodle/pluginfile.php/32206/mod_resource/content/1/DimensionalityReduction.pdf
https://dl.matlabyar.com/siavash/ML/Book/Ethem%20Alpaydin-Introduction%20to%20Machine%20Learning-The%20MIT%20Press%20(2014).pdf