Data Capstone Report

GROUP 2:

Amber Sethi (101328584)
Joshua Dmello (101346654)
Mohamed Bennis (101276333)
Muhammad Attique (101292217)
Tanvi Sharma (101265693)

Table of Contents

Introduction & Statement of business problem

Methodology and Approach

To go forward with our diagnosis, we received the raw file from sensory which contained continuous timeline data from their BOB assistant sensor relating to the machines from 2019 to 2021.

We had 6 main variables namely: -

- Ambient humidity
- Surface Temperature
- Rotation speed
- Peak vibrations-vertical
- Peak vibrations-horizontal
- Peak vibrations-axial

Firstly, we ran some descriptive statistics on variables such as surface temperature, Rotation speed, peak vibration, and ambient humidity to get a clear-cut understanding of what the data reflects.

This led us to derive some variables that could act as KPIs. Examples such as the average vibrations indicated most vibrations within a specific range which could be classified as normal, anything above or below this range could be considered as an anomaly and easily be detected setting off an alarm on the Sezary IoT platform. Similarly, with mean humidity, we could find out the ambient humidity range. The

peak vibration measurement Pointed out vibrations with a high displacement amplitude can cause machine components to exceed their yield point and experience catastrophic failures.

A frequency distribution was created for each variable to get an estimate of the number of observations within a range, this could potentially indicate the amount of anomaly occurrence within the variable data. We also derived variables from our source such as Average Vibration (Acceleration), Average Vibration (Velocity), Average Vibration (frequency), RMS Ultrasound Trend Difference, Average Rotation Speed (x, Y and Z) axis, Rotation Speed Trend Diff, Anomalies Trend Diff, Percentage of Humidity, Percentage of Temperature. (These are the variables we could derive from the data which was provided) which led us to our target variables such as Average Vibration (Acceleration, Velocity, Frequency), Average Rotation Speed and Trend Difference (Rotation Speed, Anomalies).

We went further to create a correlation and conduct an exploratory data analysis for our target variables to derive some hidden insights from the data sets, following which A linear regression model was created and run by selecting specific predictors and outcome variables. Which gave us the best fit variables to detect the outcome of our target variables.

Summary of key findings of Data Audit results

We received the dataset (diabetes.csv) file and immediately wanted to get familiar with the data through analysis. We imported the necessary libraries and the dataset into the Jupyter notebook to give us a view of what we were working with, and to build on for the data analytics

The dataset consists of several medical parameters and one dependent (outcome) parameter of binary values; the dataset is of the female gender. 'Outcome' is the column which we are going to predict, which says if the patient is diabetic or not. We can identify that out of the 768 persons, 500 are labelled as 0 (non-diabetic) and 268 as 1 (diabetic).

We first investigated the first 4 key variables or metrics in the data to provide a top-line overview or understanding of the dataset. The variables used are age, BMI, blood pressure and diabetes pedigree function. We applied Panda profiling in Python to explore their mean, median, standard deviation, and total count. Results show:

AGE: Average age 33 whilst median age is 29. The minimum age is 21 and the maximum age is 81.

BMI: Average BMI value is 32 which is also the value for median and we can say that is very close to a normal distribution curve. Zero values were present.

BloodPressure: median was 72(an average person has a diastolic blood pressure reading of 69). Missing values were present.

DiabetesPedigreeFunction: Diabetes Pedigree Function has a mean of 0.47 is a positively skewed variable with no zero values.

This was important to replace missing or null data points and normalize outlier values with the mean values rather than remove the data.

We decided to use a combination of analysis from MS Excel and Jupyter notebook (using python) to create the analytical findings.

Summary of findings of Analytical file

The health data analysis around diabetes was done is a series of exercises.

Firstly, we did data load, data diagnostics and frequency distribution reports for each file or field respectively. In doing so, the data exploration gave us quantifiable dimensions for the prediction model. They also assessed any data fallacies. Luckily, the dataset did not show any non-numeric data.

Key findings showed most of the diabetic patients recorded were in their 40's. The diastolic blood pressure range was recorded to be normal for most diabetic patients. Most of the patients had normal serum levels. They have, however, above-normal glucose levels, which was a concern.

Secondly, we explored 5 source variables and came up with 15 derived variables that we thought were useful for the analysis. Source variables were age, BMI, serum, glucose, and pedigree. Examples of the derived variables were categories or ranges of the source variables. Categories were labelled in plain language to make them easier to read. Categories can make the analysis simpler and can convey messages easier. We also defined the target variable outcome: 0 if non-diabetic, 1 if diabetic. One advantage of a binary variable for outcome was the easiness to see patterns between a single variable (such as age or BMI) and the predicted effect on the target variable (outcome). There was a disadvantage, however.

Diabetes can be classified into 6 different categories. Using binary (0 & 1) for the target variable would not be able to distinguish the 6 different categories. This could pose a limitation of the study.

As the third step, we did correlation analysis, segmentation, and analysis of numerical variables. Correlation analysis measures the strength of association between 2 variables: target variable (outcome) and source or derived variable.

In doing these analyses, we identified that glucose can be a very important variable in future model-building as it shows a strong positive correlation with the target variable outcome.

BMI, Diabetes Pedigree function and Pregnancies are variables that can also have strong relationships with the outcome and their influence needs to be evaluated further.

Lastly, after learning different types of models, we then developed a model for the dataset. The final model used is a random forest classifier. It is a classification algorithm that is made up of multiple decision trees. It uses randomness to build each tree to promote uncorrelated forests, and it merges them to get a more accurate and stable prediction.

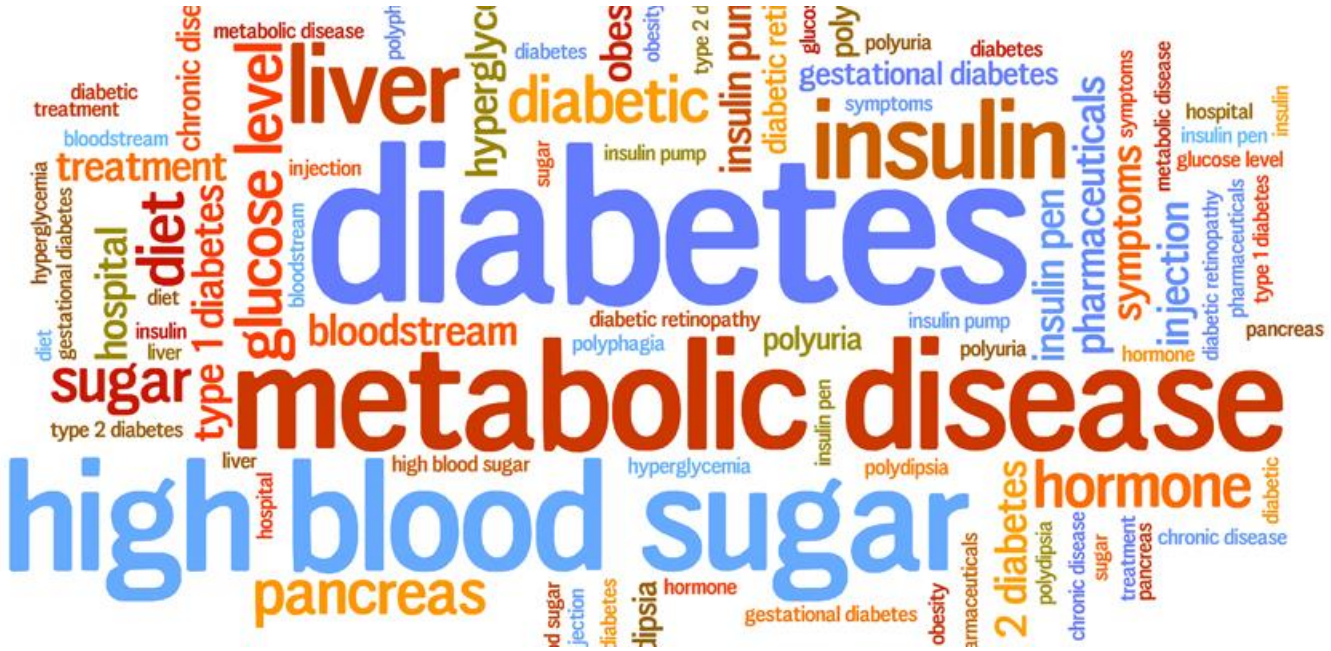
In calculating the accuracy score, the random forest classifier resulted in an accuracy of 81%. We also created a receiver operating characteristic (ROC) curve, which shows the performance of the model at all classification thresholds. In computing ROC AUC from prediction scores, it resulted in 0.88. This means the model is one whose predictions are 88% correct. The higher the AUC value for a classifier, the better it is in distinguishing between positive and negative classes.

Furthermore, we did ANOVA testing for the key variables. ANOVA is a statistical technique used to check if the means of two or more groups are statistically different from each other.

In running the test, it has been identified that glucose has the highest confidence percentage, hence it has the strongest impact on Type II diabetes.

Based on these findings, it supports the narrative in many scientific pieces of literature that diabetes is strongly tied with high glucose levels. As we all know, high glucose levels are heavily influenced by diet and lifestyle, which include consumption of sugary drinks and sugary food. To help lower blood sugar levels, it is pertinent to follow a healthy diet and lifestyle. This includes, but is not limited to, moderate consumption of carbs or sugary items, as well as regular exercise.

Analytical results



Word Cloud

The Data Set-

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

- Pregnancies: No. of times pregnant
- Glucose: Plasma Glucose Concentration (mg/dl)
- Blood Pressure: Diastolic Blood Pressure(mmHg)
- Skin Thickness: A value used to estimate body fat. Normal Triceps SkinFold Thickness in women is 23mm. Higher thickness leads to obesity and the chances of diabetes increase.
- Insulin: 2-Hour Serum Insulin (mu U/ml)
- BMI: Body Mass Index (weight in kg/ height in m2)
- Diabetes Pedigree Function: It provides information about diabetes history in relatives and the genetic relationship of those relatives with patients. Higher Pedigree Function means the patient is more likely to have diabetes.
- Age: Age (years)

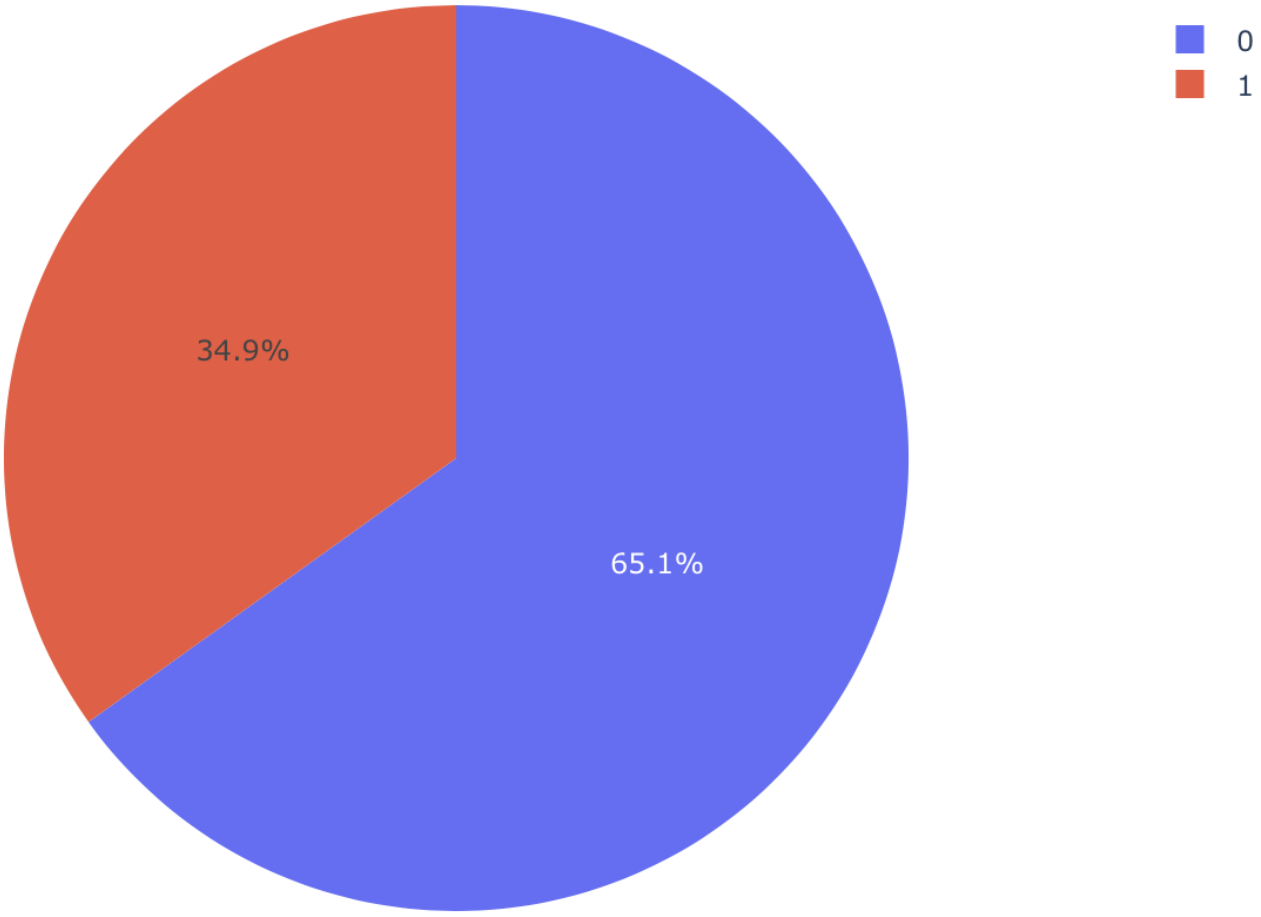
- Outcome: Class Variable (0 or 1) where '0' denotes patient is not having diabetes and '1' denotes patient having diabetes.

```
data[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] =
data[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].re
place(0,np.NaN)
```

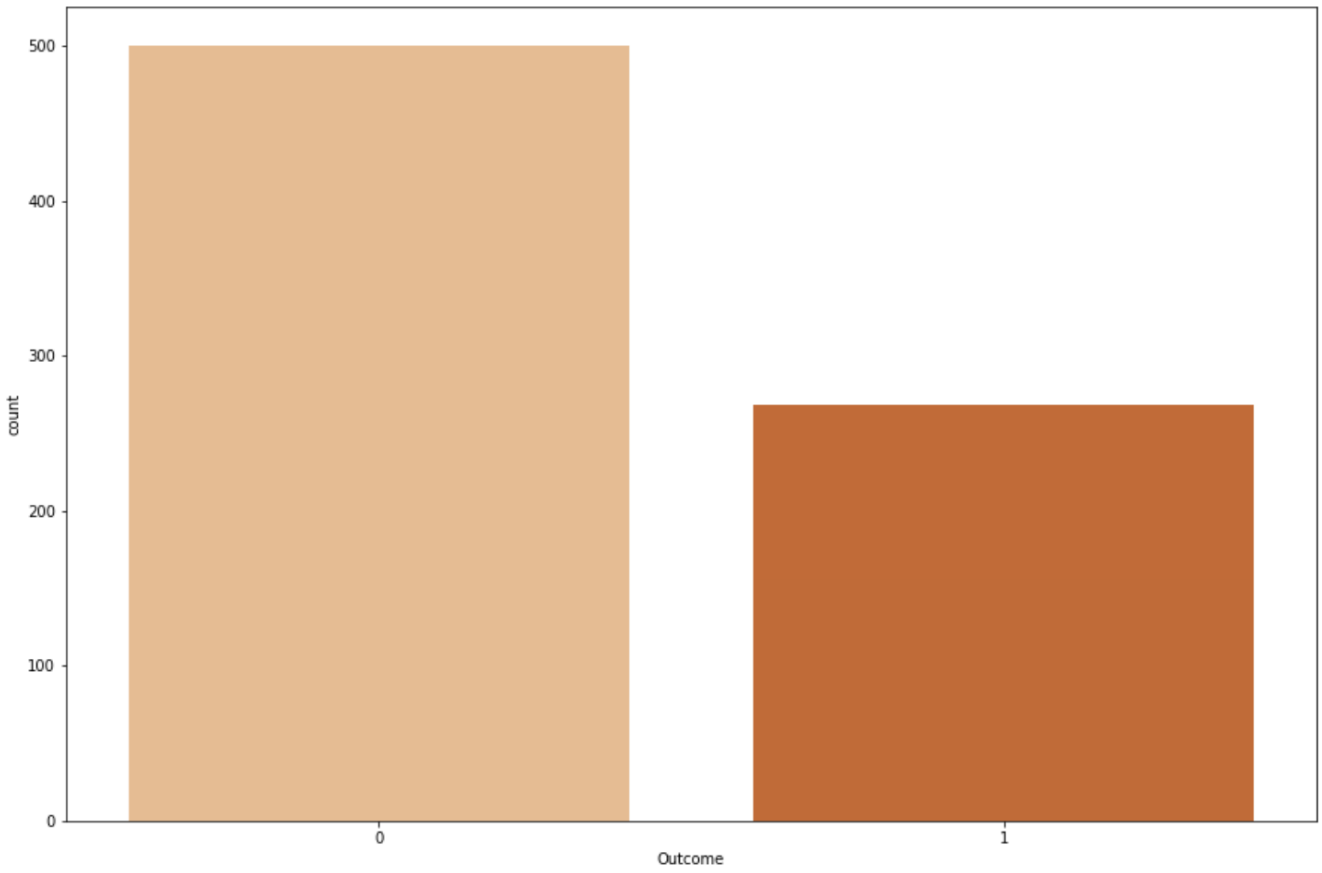
We saw on data.head() that some features contain 0, it doesn't make sense here and this indicates missing value Below we replace 0 value by NaN :

Missing values:

- Insulin = 48.7% - 374
- SkinThickness = 29.56% - 227
- BloodPressure = 4.56% - 35
- BMI = 1.43% - 11
- Glucose = 0.65% - 5



Here we can see 65.1% of the people in this dataset doesn't have Diabetes and 34.9% does.




```
Skewness for the column Pregnancies is 0.9016739791518588
Skewness for the column Glucose is 0.5309885349396285
Skewness for the column BloodPressure is 0.13415273171959252
Skewness for the column SkinThickness is 0.690619013984192
Skewness for the column Insulin is 2.166463843812443
Skewness for the column BMI is 0.5939697505712673
Skewness for the column DiabetesPedigreeFunction is 1.919911066307204
Skewness for the column Age is 1.1295967011444805
```

```
df.skew()
```

```
Pregnancies                 0.901674
Glucose                     0.530989
BloodPressure               0.134153
SkinThickness               0.690619
BMI                         0.593970
DiabetesPedigreeFunction    1.919911
Age                         1.129597
Outcome                     0.635017
dtype: float64
```

For highly skewed values we'll impute the column with **median** else **mean**.

Columns like Pregnancies, Glucose, BloodPressure, SkinThickness and BMI are not that much skewed. We fill null values with the mean for these columns, but for columns like Insulin and DiabetesPedigreeFunction, we will have to replace them will median because of skewness.

Correlation

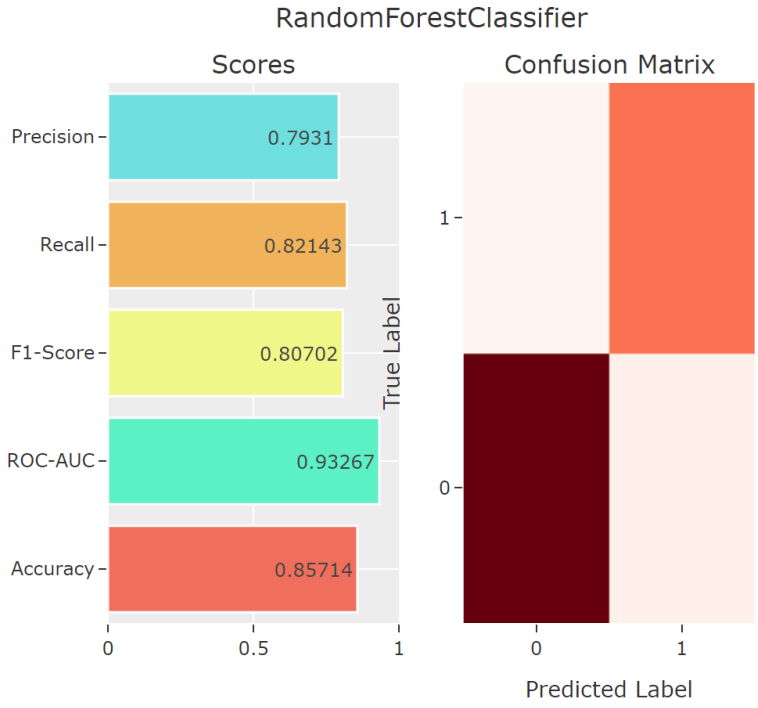
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.128213	0.208615	0.081770	0.025047	0.021565	-0.033523	0.544341	0.221898
Glucose	0.128213	1.000000	0.218937	0.192615	0.419451	0.230862	0.137327	0.266909	0.492782
BloodPressure	0.208615	0.218937	1.000000	0.191892	0.045363	0.281319	-0.002378	0.324915	0.165723
SkinThickness	0.081770	0.192615	0.191892	1.000000	0.155610	0.543162	0.102188	0.126107	0.214873
Insulin	0.025047	0.419451	0.045363	0.155610	1.000000	0.180170	0.126503	0.097101	0.203790
BMI	0.021565	0.230862	0.281319	0.543162	0.180170	1.000000	0.153400	0.025519	0.311924
DiabetesPedigreeFunction	-0.033523	0.137327	-0.002378	0.102188	0.126503	0.153400	1.000000	0.033561	0.173844
Age	0.544341	0.266909	0.324915	0.126107	0.097101	0.025519	0.033561	1.000000	0.238356
Outcome	0.221898	0.492782	0.165723	0.214873	0.203790	0.311924	0.173844	0.238356	1.000000

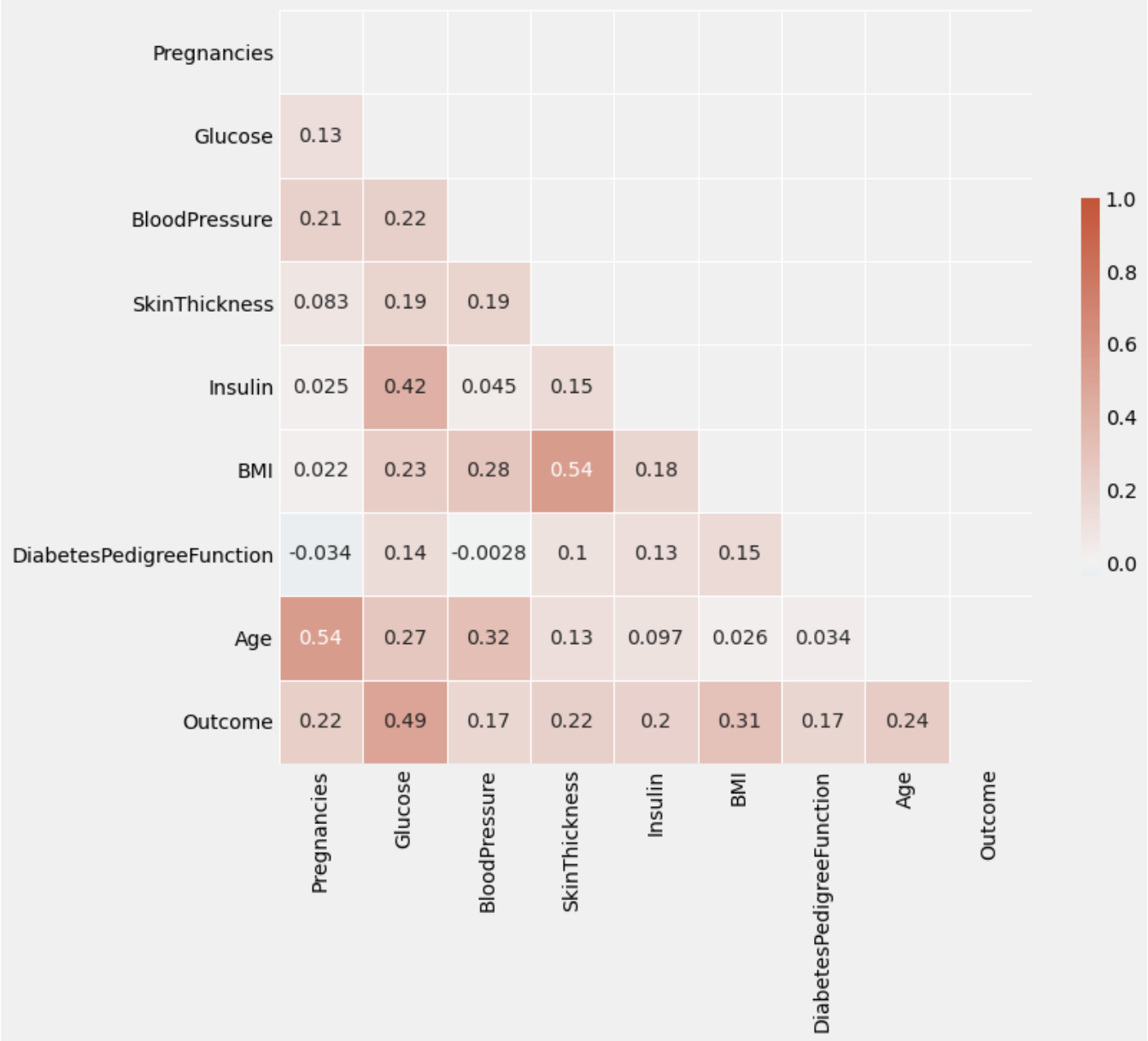
```
df[df.columns[1:]].corr()['Outcome'][:]
```

```
Glucose                     0.304123
BloodPressure               0.133105
SkinThickness               0.127473
BMI                         0.170016
DiabetesPedigreeFunction    0.113516
Age                         0.270412
Outcome                     1.000000
Name: Outcome, dtype: float64
```

Variables within a dataset can be related for lots of reasons. It can be useful in data analysis and modelling to better understand the relationships between variables. The statistical relationship between two variables is referred to as their correlation.

A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease. Correlation can also be neural or zero, meaning that the variables are unrelated.





From the above heatmap, we can observe that all the features are weakly correlated, so that removes multicollinearity out of the equation. Multicollinearity (also collinearity) is a phenomenon in which one predictor variable in a multiple regression model can be linearly predicted from the others with a substantial degree of accuracy. Models like Logistic Regression assumes the presence of non-collinearity among the features, if multicollinearity is present it can lead to the bad performance of such models.

Grouping predictor variables by target variable

```
data.groupby("Outcome")[["Pregnancies","Glucose","BloodPressure"]].agg(['max','min','mean'])
```

	Pregnancies			Glucose			BloodPressure		
	max	min	mean	max	min	mean	max	min	mean
Outcome									
0	13	0	3.298000	197.0	44.0	110.682000	122.0	24.0	70.920000
1	17	0	4.865672	199.0	78.0	142.130597	114.0	30.0	75.123134

```
data.groupby("Outcome")[["SkinThickness","Insulin","BMI","Age"]].agg(['max','min','mean'])
```

	SkinThickness			Insulin			BMI			Age		
	max	min	mean	max	min	mean	max	min	mean	max	min	mean
Outcome												
0	60.0	7.0	27.726000	744.0	15.0	127.792000	57.3	18.2	30.888434	81	21	31.190000
1	99.0	7.0	31.686567	846.0	14.0	164.701493	67.1	22.9	35.384757	70	21	37.067164

Target variable

```
y = data['Outcome']
```

```
# Getting first few observations of target variable

y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

```
# Splitting the matrices into random train & test subsets where test data contains 25% data and rest considered as training data

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.25,random_state = 200)
```

```
# Getting dimensions of train & test subsets

X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((576, 8), (192, 8), (576,), (192,))
```

```
clf = RandomForestClassifier(oob_score = True,n_jobs = -1,random_state = 100)
clf
```

```
RandomForestClassifier(n_jobs=-1, oob_score=True, random_state=100)
```

Cross validation score should between 0 and 1 and as high as possible. Here cross validation has been performed to find how well model is performing in terms of F1 score.

```
# Performing K-fold cross validation with 5 folds

scores = cross_val_score(clf,X_train,y_train,cv = 5,scoring = "f1_macro")
scores.mean()
```

```
0.7139192769714777
```

```
# Building a forest of trees from training set

clf.fit(X_train,y_train)
```

```
RandomForestClassifier(n_jobs=-1, oob_score=True, random_state=100)
```

```
# Predicting on classifier created

train_pred = clf.predict(X_train)
test_pred = clf.predict(X_test)
```

```
# Finding F1 score of training and testing sets

print("The training F1 score is: ",f1_score(train_pred,y_train))
print("The testing F1 score is :",f1_score(test_pred,y_test))
```

```
The training F1 score is:  1.0
The testing F1 score is : 0.6942148760330579
```

```
#  Tuning hyperparameters

parameters = {
              "max_depth":[2,3,4],
              "n_estimators":[100,104,106],
              "min_samples_split":[3,4,5],
              "min_samples_leaf":[4,8,9]
              }

scorer = make_scorer(f1_score)
```

```
# Using Randomized Search CV to find best optimal hyperparameter that best describe a classifier

clf1 = RandomizedSearchCV(clf,parameters,scoring = scorer)

# Fitting the model

clf1.fit(X_train,y_train)

# Getting best estimator having high score

best_clf_random = clf1.best_estimator_
best_clf_random
```

```
RandomForestClassifier(max_depth=4, min_samples_leaf=8, min_samples_split=4,
                       n_estimators=106, n_jobs=-1, oob_score=True,
                       random_state=100)
```

```
# Again, finding cross validation score

scores = cross_val_score(best_clf_random,X_train,y_train,cv = 5,scoring = "f1_macro")
scores.mean()
```

```
0.6993389935821842
```

```
# Fitting the best estimator

best_clf_random.fit(X_train,y_train)
```

```
RandomForestClassifier(max_depth=4, min_samples_leaf=8, min_samples_split=4,
                       n_estimators=106, n_jobs=-1, oob_score=True,
                       random_state=100)
```

```
# Getting first estimator

best_clf_random.estimators_[0]
```

```
DecisionTreeClassifier(max_depth=4, max_features='auto', min_samples_leaf=8,
                       min_samples_split=4, random_state=186422792)
```

```
# Predicting on best estimator

train_pred = best_clf_random.predict(X_train)
test_pred = best_clf_random.predict(X_test)
```

```
# Finding the F1 score of training & testing sets

print("The training F1 score is: ",f1_score(train_pred,y_train))
print("The testing F1 score is :",f1_score(test_pred,y_test))
```

```
The training F1 score is:  0.675603217158177
The testing F1 score is : 0.689655172413793
```

```
# Getting accuracy score

accuracy_score(y_test,test_pred)
```

```
0.8125
```

```
# Computing ROC AUC from prediction scores

roc_auc_score(y_test,best_clf_random.predict_proba(X_test)[:,1])
```

```
0.8840321141837646
```

Standardizing a dataset involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1.

This can be thought of as subtracting the mean value or centering the data. Scaling the features is of utmost importance because different features are in different scales. Let's say the Age has values in double digits, whereas the DPF is of the kind float, the effect of the Age feature will be more as compared to the DPF

The best practice is to use only the training set to figure out how to scale/normalize, then blindly apply the same transform to the test set.

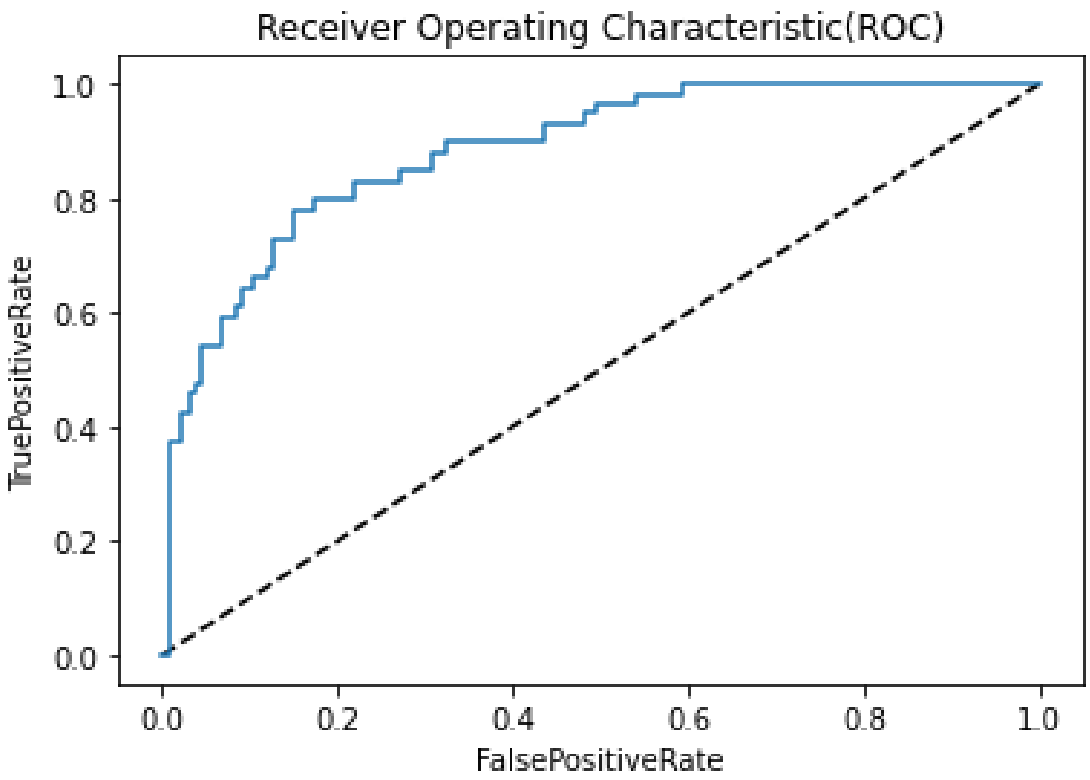
For example, say you're going to normalize the data by removing the mean and dividing out the variance. If you use the whole dataset to figure out the feature mean and variance, you're using knowledge about the distribution of the test set to set the scale of the training set - 'leaking' information.

The right way to do this is to use only the training set to calculate the mean and variance, normalize the training set, and then at test time, use that same (training) mean and variance to normalize the test set.

```
Random Forest Classifier:
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        83
           1       0.65      0.65      0.65        43

    accuracy                           0.76       126
   macro avg       0.74      0.74      0.74       126
weighted avg       0.76      0.76      0.76       126
```

```
# Computing confusion matrix

pd.crosstab(y_test,test_pred,rownames = ['True'],colnames = ['Predicted'],margins = True)
```

Predicted	0	1	All
True			
0	116	17	133
1	19	40	59
All	135	57	192



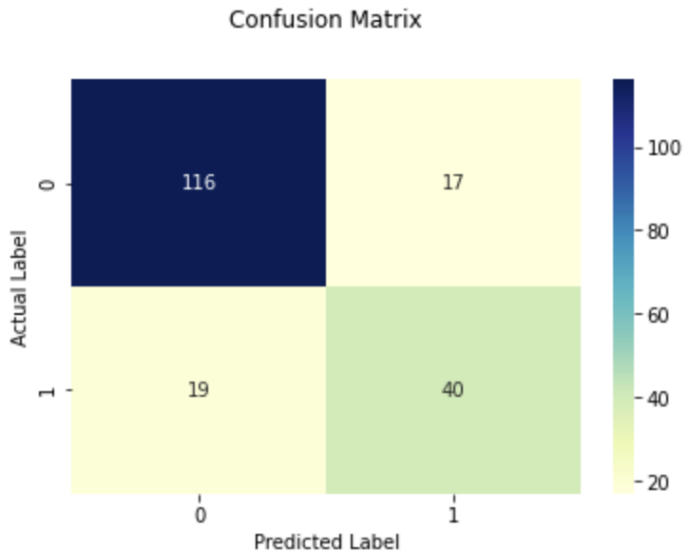
AUC is the percentage of the ROC plot that is underneath the curve:

- AUC is useful as a single number summary of classifier performance, Higher value = better classifier
- AUC of 0.5 is like tossing a coin
- AUC is useful even when there is high class imbalance (unlike classification accuracy) like in Fraud case with a null accuracy almost 99%

A ROC curve can help you to choose a threshold that balances sensitivity and specificity in a way that makes sense for your particular context. You can't actually see the thresholds used to generate the curve on the ROC curve itself. You can use the function and plot below.

```
# Plotting confusion matrix

cnf_matrix = confusion_matrix(y_test,test_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix),annot = True,cmap = "YlGnBu",fmt = 'g')
plt.title("Confusion Matrix",y = 1.1)
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

```
Text(33.0, 0.5, 'Actual Label')
```




```
# Computing the precision

precision_score(y_test,test_pred)
```

```
0.7017543859649122
```

Conclusions / Next Steps