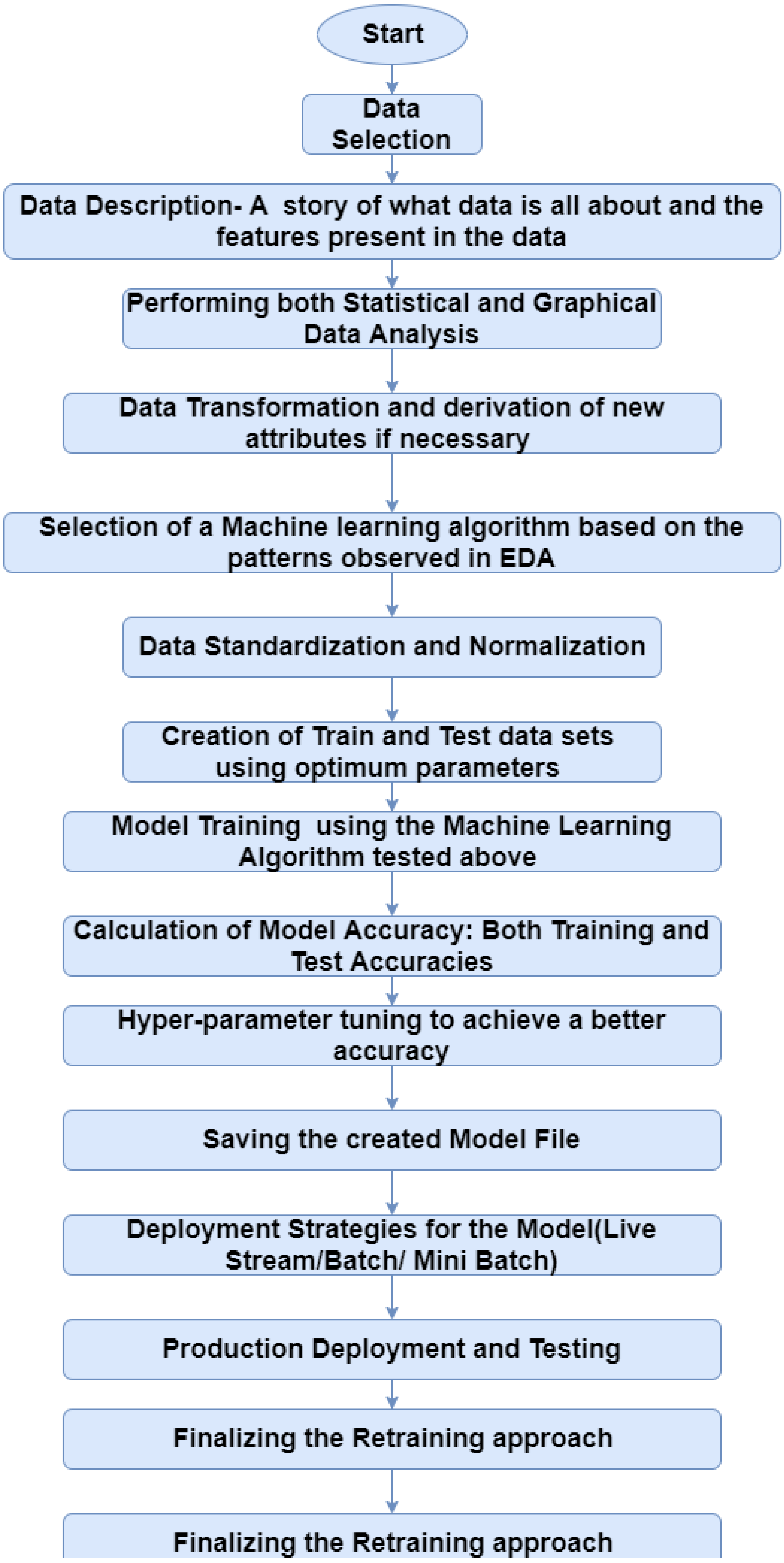
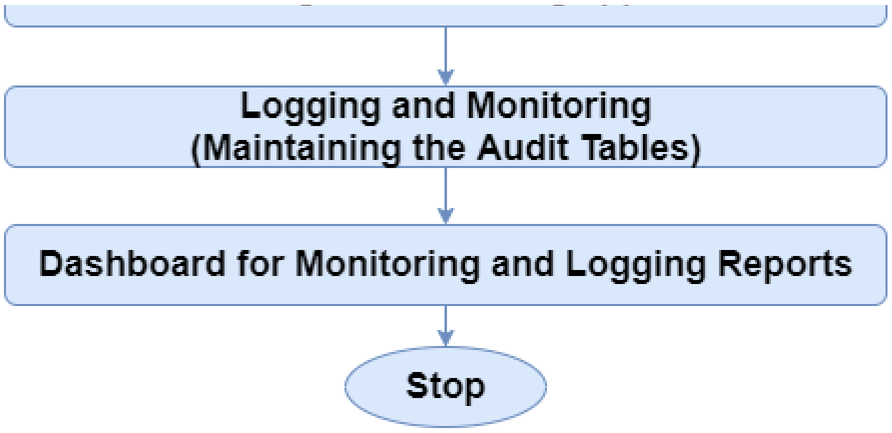
Logistic Regression ¶

Application Flow

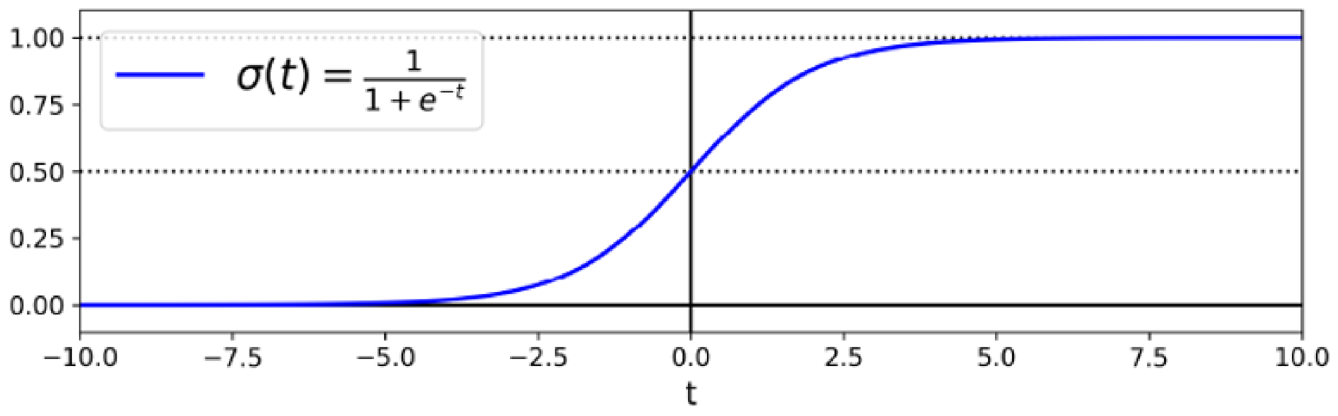
Logistic Regression is one of the most fundamental algorithms for classification in the Machine Learning world. But before proceeding with the algorithm, let’s first discuss the lifecycle of any machine learning model. This diagram explains the creation of aMachine Learning model from scratch and then taking the same model further with hyperparameter tuning to increase its accuracy, deciding thedeployment strategies for that model and once deployed setting up the logging and monitoring frameworks to generate reports and dashboardsbased on the client requirements. A typical lifecycle diagram for a machine learning model looks like: Introduction


Start
Data Selection
Data Description- A story of what data is all about and the features present in the data
Performing both Statistical and Graphical Data Analysis
Data Transformation and derivation of new attributes if necessary
Selection of a Machine learning algorithm based on the patterns observed in EDA
Data Standardization and Normalization
Creation of Train and Test data sets using optimum parameters
Model Training using the Machine Learning Algorithm tested above
Calculation of Model Accuracy: Both Training and Test Accuracies
Hyper-parameter tuning to achieve a better accuracy
Saving the created Model File
Deployment Strategies for the Model(Live Stream/Batch/ Mini Batch)
Production Deployment and Testing
Finalizing the Retraining approach
Finalizing the Retraining approach


Logging and Monitoring
(Maintaining the Audit Tables)
Dashboard for Monitoring and Logging Reports
Stop

```
Introduction
In linear regression, the type of data we deal with is quantitative, whereas we use
classification models to deal with qualitative data or categoricaldata. The algorithms
used for solving a classification problem first predict the probability of each of the
categories of the qualitative variables, as thebasis for making the classification. And,
as the probabilities are continuous numbers, classification using probabilities also
behave like regressionmethods. Logistic regression is one such type of classification
model which is used to classify the dependent variable into two or more classes
orcategories.
Why don’t we use Linear regression for classification problems?
Let’s suppose you took a survey and noted the response of each person as satisfied,
neutral or Not satisfied. Let’s map each category:

Satisfied – 2
Neutral – 1
Not Satisfied – 0
But this doesn’t mean that the gap between Not satisfied and Neutral is same as Neutral
and satisfied. There is no mathematical significance ofthese mapping. We can also map the
categories like:
Satisfied – 0
Neutral – 1
Not Satisfied – 2
It’s completely fine to choose the above mapping. If we apply linear regression to both
the type of mappings, we will get different sets of predictions.Also, we can get
prediction values like 1.2, 0.8, 2.3 etc. which makes no sense for categorical values. So,
there is no normal method to convertqualitative data into quantitative data for use in
linear regression. Although, for binary classification, i.e. when there only two
categorical values,

using the least square method can give decent results. Suppose we have two categories
Black and White and we map them as follows:
Black – 0
White - 1
We can assign predicted values for both the categories such as Y> 0.5 goes to class white
and vice versa. Although, there will be some predictionsfor which the value can be greater
than 1 or less than 0 making them hard to classify in any class. Nevertheless, linear
regression can work decentlyfor binary classification but not that well for multi-class
classification. Hence, we use classification methods for dealing with such problems.
Logistic Regression
Logistic regression is one such regression algorithm which can be used for performing
classification problems. It calculates the probability that agiven value belongs to a
specific class. If the probability is more than 50%, it assigns the value in that
particular class else if the probability is lessthan 50%, the value is assigned to the
other class. Therefore, we can say that logistic regression acts as a binary classifier.
Working of a Logistic Model
For linear regression, the model is defined by:
```

```
- (i)
and for logistic regression, we calculate probability, i.e. y is the probability of a
given variable x belonging to a certain class. Thus, it is obvious thatthe value of y
should lie between 0 and 1.
But, when we use equation(i) to calculate probability, we would get values less than 0 as
well as greater than 1. That doesn’t make any sense . So,we need to use such an equation
which always gives values between 0 and 1, as we desire while calculating the probability.
```
$y = \beta 0 + \beta 1x$
```
Sigmoid function
We use the sigmoid function as the underlying function in Logistic regression.
Mathematically and graphically, it is shown as:
```



Why do we use the Sigmoid Function?

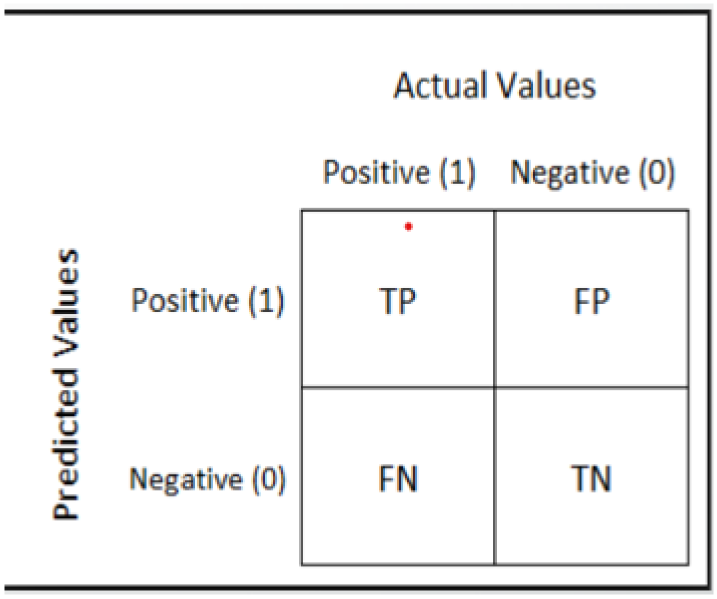
1) The sigmoid function’s range is bounded between 0 and 1. Thus it’s useful in calculating the probability for the Logistic function. 2) It’s derivativeis easy to calculate than other functions which is useful during gradient descent calculation. 3) It is a simple way of introducing non-linearity to themodel.

Although there are other functions as well, which can be used, but sigmoid is the most common function used for logistic regression. The logistic function is given as:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}.$$

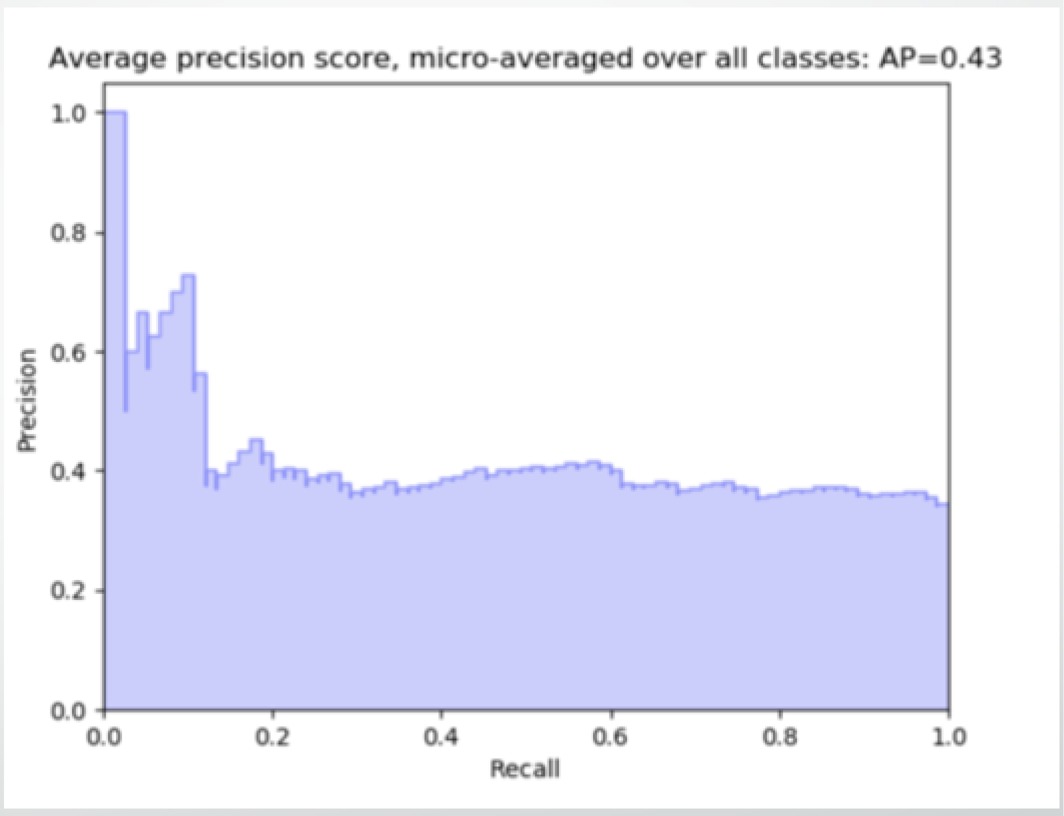
```
Evaluation of a Classification Model
In machine learning, once we have a result of the classification problem, how do we
measure how accurate our classification is? For a regressionproblem, we have different
metrics like R Squared score, Mean Squared Error etc. what are the metrics to measure the
credibility of a classificationmodel?
Metrics In a regression problem, the accuracy is generally measured in terms of the
difference in the actual values and the predicted values. In aclassification problem, the
credibility of the model is measured using the confusion matrix generated, i.e., how
accurately the true positives and truenegatives were predicted. The different metrics used
for this purpose are:
Accuracy
Recall
Precision
```

```
F1 Score
Specifity
AUC( Area Under the Curve)
ROC(Receiver Operator Characteristic)
Confusion Matrix
A typical confusion matrix looks like the figure shown.
```



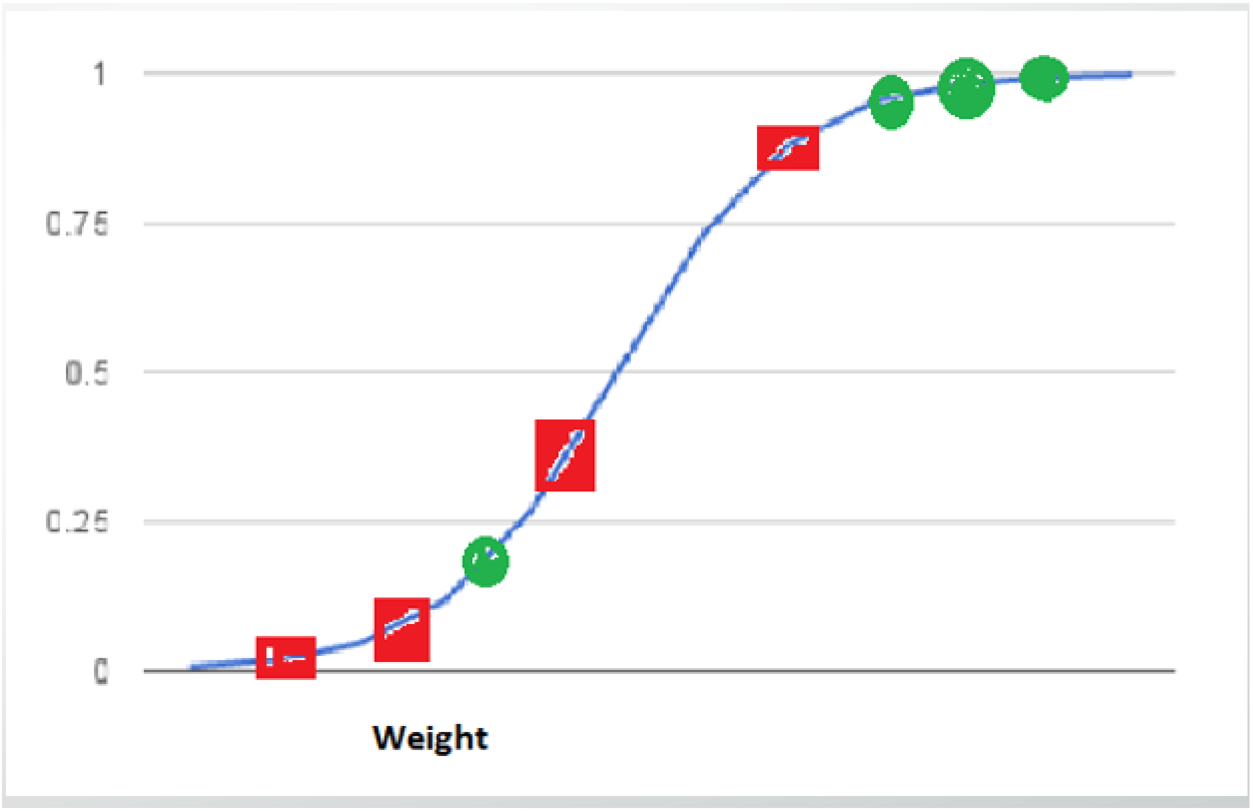
```
# Where the terms have the meaning:

True Positive(TP):
A result that was predicted as positive by the classification model and also is positive

True Negative(TN):
A result that was predicted as negative by the classification model and also is negative

False Positive(FP):
A result that was predicted as positive by the classification model but actually is
negative

False Negative(FN):
A result that was predicted as negative by the classification model but actually is
positive.
The Credibility of the model is based on how many correct predictions did the model do.
Accuracy
The mathematical formula is :
Accuracy
=
Or, it can be said that it’s defined as the total number of correct classifications
divided by the total number of classifications.
```
$(TP+TN)/(TP+TN+FP+FN)$
```
Recall or Sensitivity
The mathematical formula is:
Recall
=
Or, as the name suggests, it is a measure of: from the total number of positive results
how many positives were correctly predicted by the model.
It shows how relevant the model is, in terms of positive results only.
Let’s suppose in the previous model, the model gave 50 correct predictions(TP) but failed
to identify 200 cancer patients(FN). Recall in that case willbe:
Recall=
= 0.2 (The model was able to recall only 20% of the cancer patients)
```
$TP/(TP+FN)$

50
(50+200)
Precision
Precision is a measure of amongst all the positive predictions, how many of them were actually positive. Mathematically,
Precision=
Let’s suppose in the previous example, the model identified 50 people as cancer patients(TP) but also raised a false alarm for 100 patients(FP).Hence,
Precision=
=0.33 (The model only has a precision of 33%)
TP
$(TP+FP)$
50
(50+100)
But we have a problem!!
As evident from the previous example, the model had a very high Accuracy but performed poorly in terms of Precision and Recall. So, necessarily
Accuracy
is not the metric to use for evaluating the model in this case.
Imagine a scenario, where the requirement was that the model recalled all the defaulters who did not pay back the loan. Suppose there were 10such defaulters and to recall those 10 defaulters, and the model gave you 20 results out of which only the 10 are the actual defaulters. Now, therecall of the model is 100%, but the precision goes down to 50%.
A Trade-off?



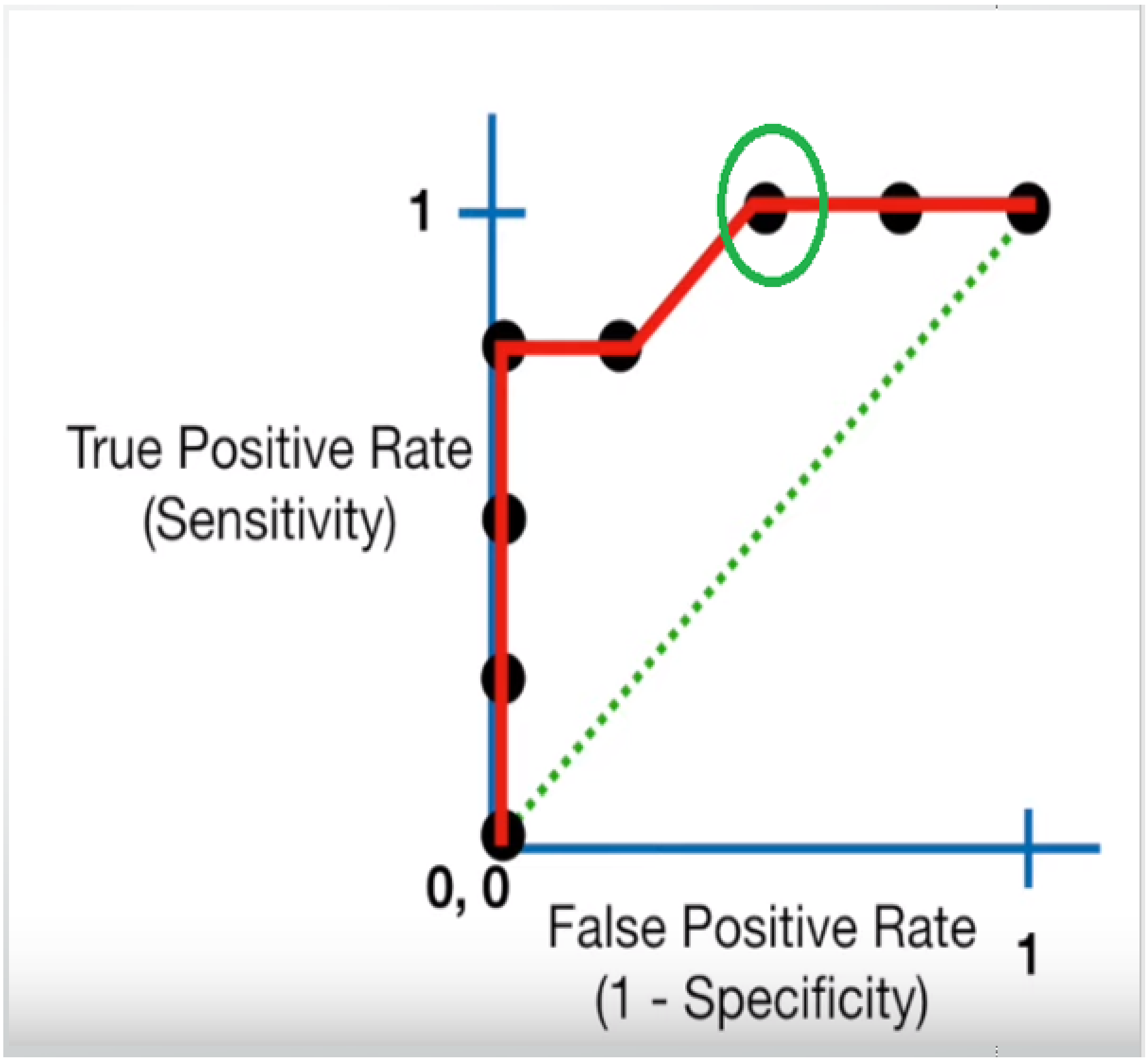
As observed from the graph, with an increase in the Recall, there is a drop in Precision of the model.
So the question is - what to go for? Precision or Recall?
Well, the answer is: it depends on the business requirement.
For example, if you are predicting cancer, you need a 100 % recall. But suppose you are predicting whether a person is innocent or not, you need100% precision.
Can we maximise both at the same time? No
So, there is a need for a better metric then?
Yes. And it’s called an
F1 Score

F1 Score
From the previous examples, it is clear that we need a metric that considers both Precision and Recall for evaluating a model. One such metric isthe F1 score.
F1 score is defined as the harmonic mean of Precision and Recall.
The mathematical formula is: F1 score= 2∗(($Precision*Recall$)
($Precision+Recall$))
Specificity or True Negative Rate
This represents how specific is the model while predicting the True Negatives.
Mathematically,
Specificity=
Or, it can be said that it quantifies the total number of negatives predicted by the model with respect to the total number ofactual negative or non favorable outcomes.
Similarly, False Positive rate can be defined as: (1- specificity) Or,
TN
$(TN+FP)$
FP
$(TN+FP)$
ROC(Receiver Operator Characteristic)
We know that the classification algorithms work on the concept of probability of occurrence of the possible outcomes. A probability value liesbetween 0 and 1. Zero means that there is no probability of occurrence and one means that the occurrence is certain.
But while working with real-time data, it has been observed that we seldom get a perfect 0 or 1 value. Instead of that, we get different decimal valueslying between 0 and 1. Now the question is if we are not getting binary probability values how are we actually determining the class in ourclassification problem?
There comes the concept of Threshold. A threshold is set, any probability value below the threshold is a negative outcome, and anything more thanthe threshold is a favourable or the positive outcome. For Example, if the threshold is 0.5, any probability value below 0.5 means a negative or anunfavourable outcome and any value above 0.5 indicates a positive or favourable outcome.
Now, the question is, what should be an ideal threshold?
The following diagram shows a typical logistic regression curve.



The horizontal lines represent the various values of thresholds ranging from 0 to 1.

```
Let’s suppose our classification problem was to identify the obese people from the given
data.
The green markers represent obese people and the red markers represent the non-obese
people.
Our confusion matrix will depend on the value of the threshold chosen by us.



For Example, if 0.25 is the threshold then
TP(actually obese)=3 TN(Not obese)=2 FP(Not obese but predicted obese)=2(the two red
squares above the 0.25 line) FN(Obese but predicted as not obese )=1(Green circle below
0.25line )


A typical ROC curve looks like the following figure.
```

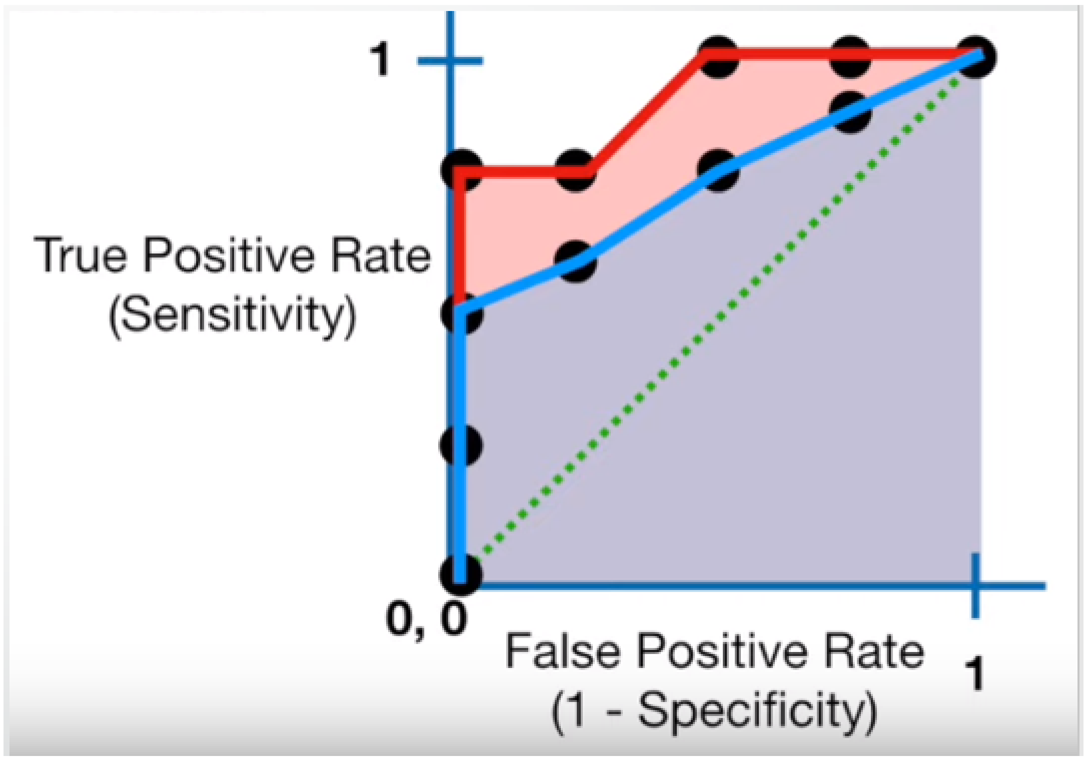


Mathematically, it represents the various confusion matrices for various thresholds. Each black dot is one confusion matrix. The green dotted line represents the scenario when the true positive rate equals the false positive rate. As evident from the curve, as we move from the rightmost dot towards left, after a certain threshold, the false positive rate decreases. After some time, the false positive rate becomes zero. The point encircled in green is the best point as it predicts all the values correctly and keeps the False positive as a minimum. But that is not a rule of thumb. Based on the requirement, we need to select the point of a threshold. The ROC curve answers our question of which threshold to choose.

But we have a confusion!!

Let’s suppose that we used different classification algorithms, and different ROCs for the corresponding algorithms have been plotted. The questionis: which algorithm to choose now? The answer is to calculate the area under each ROC curve.

AUC(Area Under Curve)



It helps us to choose the best model amongst the models for which we have plotted the ROC curves

The best model is the one which encompasses the maximum area under it.

In the adjacent diagram, amongst the two curves, the model that resulted in the red one should be chosen as it clearly covers more area thanthe blue one

Python Implementation

```
For more documentation visit,
scikit-learn.org
```

Problem satement

Based on the pima Indians historical diabetes data, build a machine learning binary classification model to predict if the person is diabetic or Not based on below features

Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,Diabetespedigreefunction,Age

Label

Outcome

In [1]:

```
#Let's start with importing necessary libraries

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
data=pd.read_csv("https://raw.githubusercontent.com/training-ml/Files/main/diabetes.csv") #
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [3]:

```
data.shape
```

Out[3]:

```
(768, 9)
```

In [4]:

```
data.describe()
```

Out[4]:

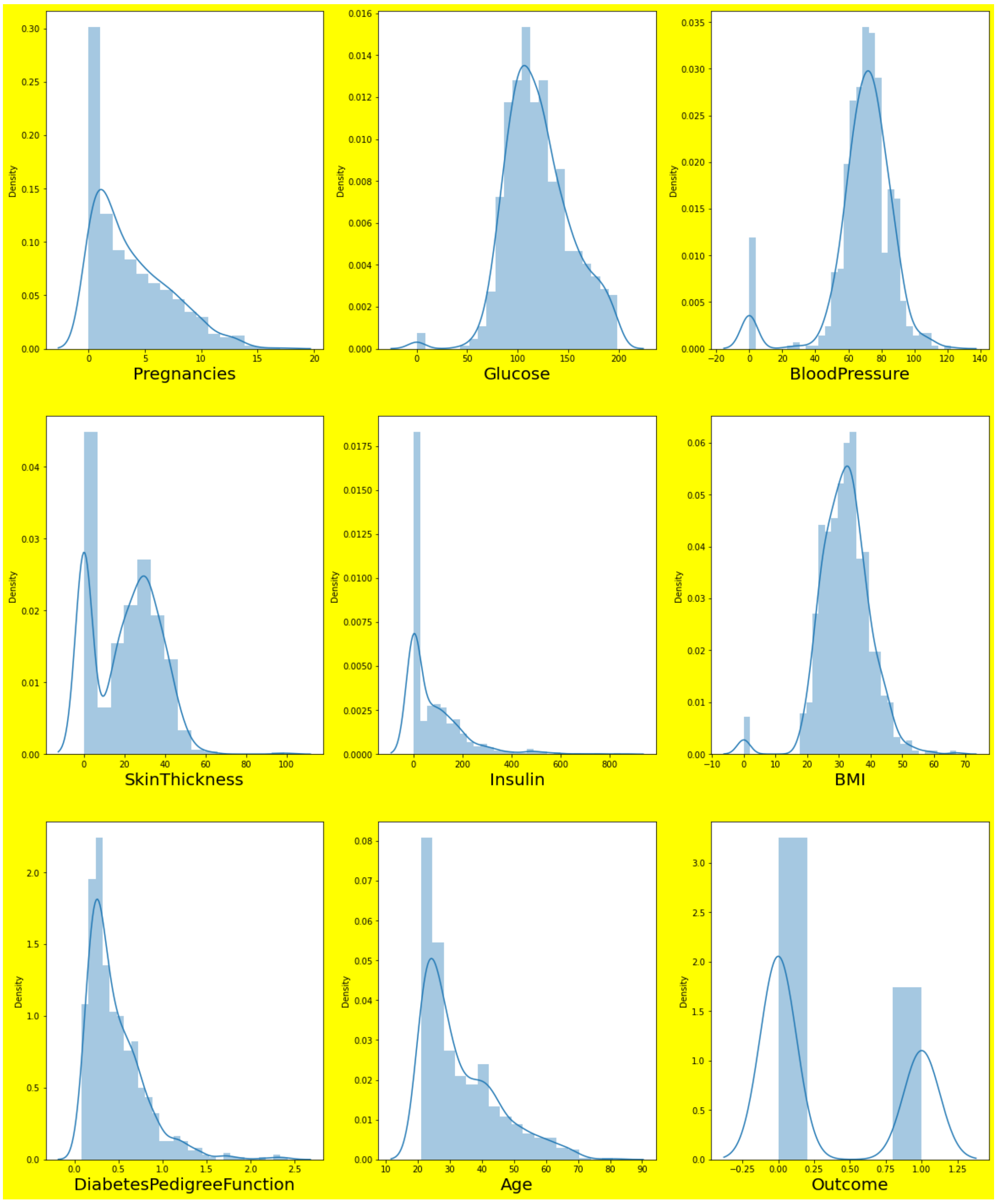
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	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	

It seems that there are no missing values in our data. Great ,let's see the distribution of data

Data preprocessing (Analysis(EDA))

In [5]:

```
#Let's see how data is distributed for every column
plt.figure(figsize=(20,25),facecolor='yellow')
plotnumber=1

for column in data:
    if plotnumber<=9:   # as  there are 9 columns in the data
        ax=plt.subplot(3,3,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize=20)


    plotnumber+=1
plt.show()
```



We can see there is some skewness in the data, let's deal with data.

Also, we can see there few data for columns Glucose, Insulin, skin thickness, BMI and Blood Pressure which have value as 0. That's not possible.You can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace itwith their respective mean values.

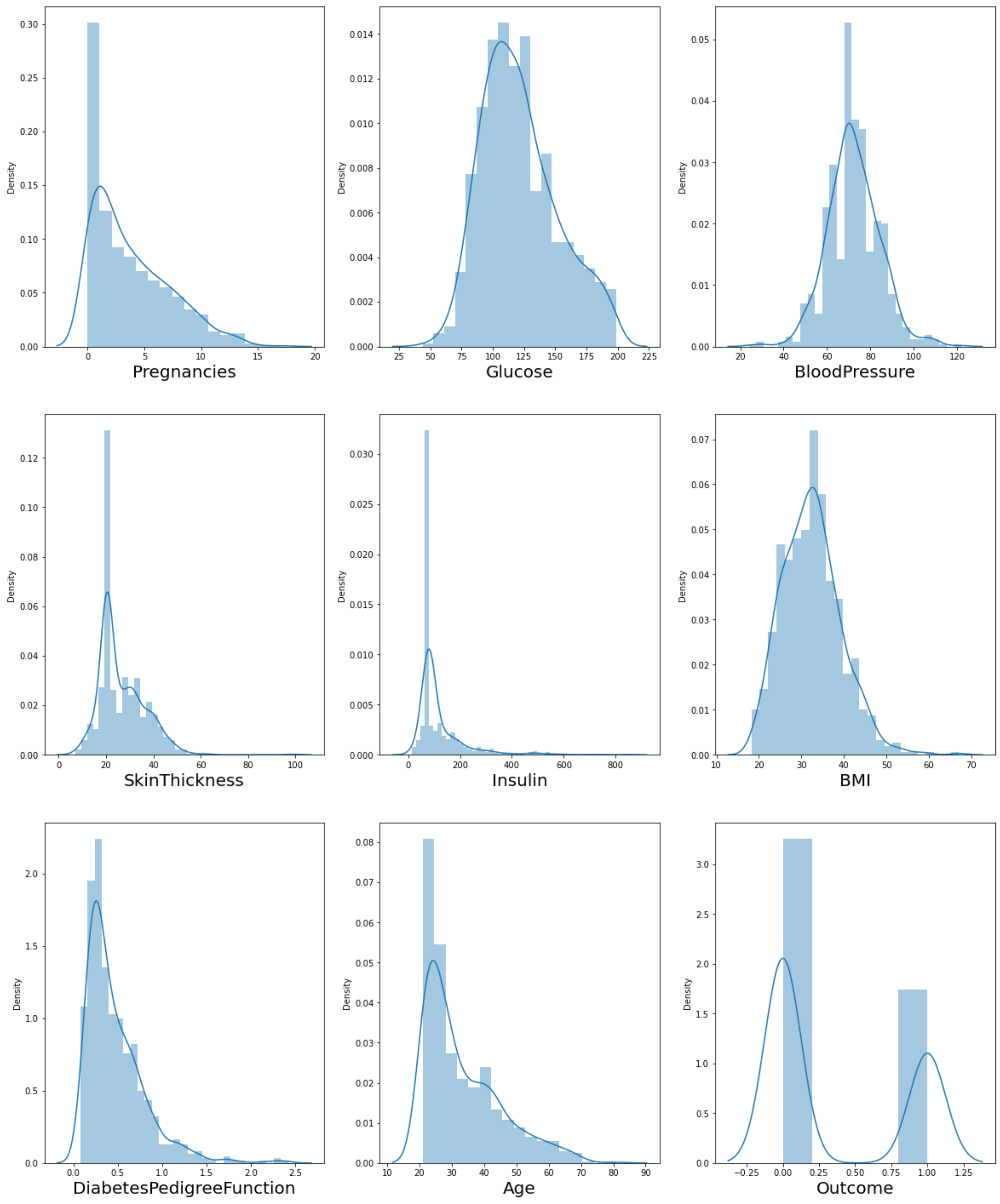
Let's do the latter.

In [6]:

```
# replacing zero values with the mean of the column

data['BMI']=data['BMI'].replace(0,data['BMI'].mean())

data['BloodPressure']=data['BloodPressure'].replace(0,data['BloodPressure'].mean())

data['Glucose']=data['Glucose'].replace(0,data['Glucose'].mean())

data['Insulin']=data['Insulin'].replace(0,data['Insulin'].mean())

data['SkinThickness']=data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [7]:

```
#Let's see how data is distributed for every column
plt.figure(figsize=(20,25))
plotnumber=1

for column in data:
    if plotnumber<=9:   # as  there are 9 columns in the data
        ax=plt.subplot(3,3,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize=20)


    plotnumber+=1
plt.show()
```



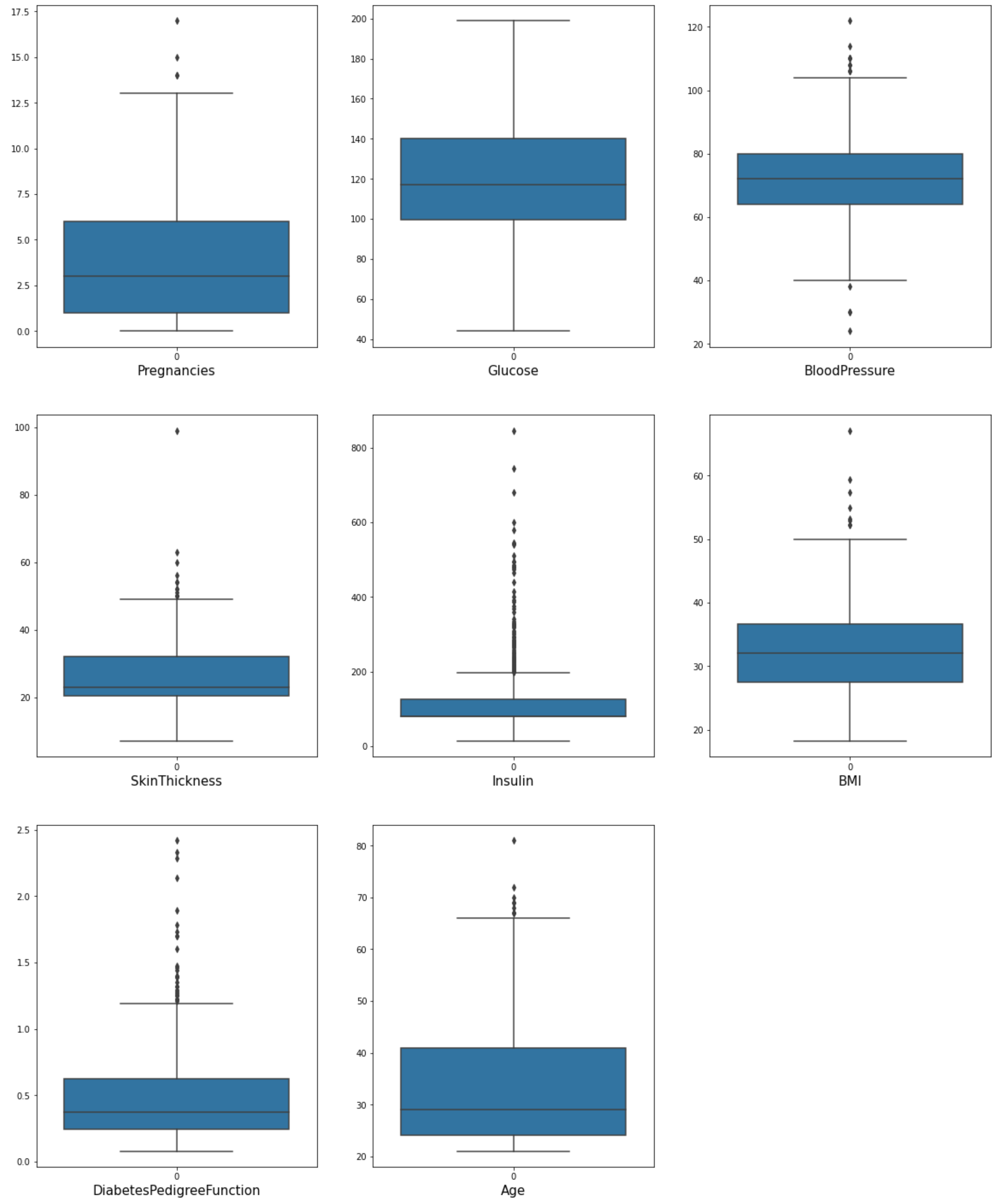
Now we have dealt with the 0 values and data looks better.But, there still are outliers present in some columns.Let's deal with them.

In [8]:

```
df_features= data.drop('Outcome',axis=1)
```

In [9]:

```
# visualize the outliers using boxplot

plt.figure(figsize=(20,25))
graph=1

for column in df_features:
    if graph<=9:   # as  there are 9 columns in the data
        plt.subplot(3,3,graph)
        ax=sns.boxplot(data=df_features[column])
        plt.xlabel(column,fontsize=15)


    graph+=1
plt.show()
```



In [10]:

```
data.shape
```

Out[10]:

```
(768, 9)
```

In [11]:

```
# Find the IQR (inter quantile range) to identify outliers

#1st quantile
q1=data.quantile(0.25)
# 3rd quantile
q3=data.quantile(0.75)

# IQR
iqr=q3-q1
```

outlier detection formula

higher side==> Q3 + (1.5 * IQR)

Lower side==> Q1- (1.5 * IQR)

In [12]:

```
# validating one outlier
preg_high=(q3.Pregnancies + (1.5 * iqr.Pregnancies))
preg_high
```

Out[12]:

```
13.5
```

In [13]:

```
# check the indexes which have higher values
index=np.where(data['Pregnancies']> preg_high)
index
```

Out[13]:

```
(array([ 88, 159, 298, 455], dtype=int64),)
```

In [14]:

```
# Drop the index which we found in the above cell
data=data.drop(data.index[index])
data.shape
```

Out[14]:

```
(764, 9)
```

In [15]:

```
data.reset_index()
```

Out[15]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	4	0	137.0	40.0	35.000000	168.000000	43.1	
...	...	...	...	...	...	...	...	
759	763	10	101.0	76.0	48.000000	180.000000	32.9	
760	764	2	122.0	70.0	27.000000	79.799479	36.8	
761	765	5	121.0	72.0	23.000000	112.000000	26.2	
762	766	1	126.0	60.0	20.536458	79.799479	30.1	
763	767	1	93.0	70.0	31.000000	79.799479	30.4	

764 rows × 10 columns

Type *Markdown* and LaTeX: α^2

In [16]:

```
bp_high=(q3.BloodPressure + (1.5* iqr.BloodPressure))
print(bp_high)

index= np.where(data['BloodPressure']> bp_high)

data  =data.drop(data.index[index])

print(data.shape)

data.reset_index()
```

```
104.0
(754, 9)
```

Out[16]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	4	0	137.0	40.0	35.000000	168.000000	43.1	
...	...	...	...	...	...	...	...	
749	763	10	101.0	76.0	48.000000	180.000000	32.9	
750	764	2	122.0	70.0	27.000000	79.799479	36.8	
751	765	5	121.0	72.0	23.000000	112.000000	26.2	
752	766	1	126.0	60.0	20.536458	79.799479	30.1	
753	767	1	93.0	70.0	31.000000	79.799479	30.4	

754 rows × 10 columns

In [17]:

```

index
```

Out[17]:

```
(array([ 43,  84, 105, 175, 359, 545, 654, 658, 668, 687], dtype=int64),)
```

index

data =data.drop(data.index[index])

print(data.shape)

data.reset_index()

index

In [18]:

```
st_high=(q3.SkinThickness + (1.5 * iqr.SkinThickness))
print(st_high)
index=np.where(data['SkinThickness']> st_high)

data=data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
49.1953125
(742, 9)
```

Out[18]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	4	0	137.0	40.0	35.000000	168.000000	43.1	
...	...	...	...	...	...	...	...	
737	763	10	101.0	76.0	48.000000	180.000000	32.9	
738	764	2	122.0	70.0	27.000000	79.799479	36.8	
739	765	5	121.0	72.0	23.000000	112.000000	26.2	
740	766	1	126.0	60.0	20.536458	79.799479	30.1	
741	767	1	93.0	70.0	31.000000	79.799479	30.4	

742 rows × 10 columns

In [19]:

```
insu_high=(q3.Insulin + (1.5 * iqr.Insulin))
print(insu_high)
index=np.where(data['Insulin']> insu_high)

data=data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
198.42578125
(657, 9)
```

Out[19]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	4	0	137.0	40.0	35.000000	168.000000	43.1	
...	...	...	...	...	...	...	...	
652	763	10	101.0	76.0	48.000000	180.000000	32.9	
653	764	2	122.0	70.0	27.000000	79.799479	36.8	
654	765	5	121.0	72.0	23.000000	112.000000	26.2	
655	766	1	126.0	60.0	20.536458	79.799479	30.1	
656	767	1	93.0	70.0	31.000000	79.799479	30.4	

657 rows × 10 columns

index

In [20]:

```
bmi_high=(q3.BMI + (1.5 * iqr.BMI))
print(bmi_high)
index=np.where(data['BMI']> bmi_high)

data=data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
50.25
(654, 9)
```

Out[20]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	4	0	137.0	40.0	35.000000	168.000000	43.1	
...	...	...	...	...	...	...	...	
649	763	10	101.0	76.0	48.000000	180.000000	32.9	
650	764	2	122.0	70.0	27.000000	79.799479	36.8	
651	765	5	121.0	72.0	23.000000	112.000000	26.2	
652	766	1	126.0	60.0	20.536458	79.799479	30.1	
653	767	1	93.0	70.0	31.000000	79.799479	30.4	

654 rows × 10 columns

In [21]:

```
dpf_high=(q3.DiabetesPedigreeFunction + (1.5 * iqr.DiabetesPedigreeFunction))
print(dpf_high)
index=np.where(data['DiabetesPedigreeFunction']> dpf_high)

data=data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
1.2
(631, 9)
```

Out[21]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	5	5	116.0	74.0	20.536458	79.799479	25.6	
...	...	...	...	...	...	...	...	
626	763	10	101.0	76.0	48.000000	180.000000	32.9	
627	764	2	122.0	70.0	27.000000	79.799479	36.8	
628	765	5	121.0	72.0	23.000000	112.000000	26.2	
629	766	1	126.0	60.0	20.536458	79.799479	30.1	
630	767	1	93.0	70.0	31.000000	79.799479	30.4	

631 rows × 10 columns

In [22]:

```
age_high=(q3.Age + (1.5 * iqr.Age))
print(age_high)
index=np.where(data['Age']> age_high)

data=data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
66.5
(622, 9)
```

Out[22]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	5	5	116.0	74.0	20.536458	79.799479	25.6	
...	...	...	...	...	...	...	...	
617	763	10	101.0	76.0	48.000000	180.000000	32.9	
618	764	2	122.0	70.0	27.000000	79.799479	36.8	
619	765	5	121.0	72.0	23.000000	112.000000	26.2	
620	766	1	126.0	60.0	20.536458	79.799479	30.1	
621	767	1	93.0	70.0	31.000000	79.799479	30.4	

622 rows × 10 columns

In [23]:

```
bp_low=(q1.BloodPressure -(1.5* iqr.BloodPressure))
print(bp_low)

index= np.where(data['BloodPressure'] < bp_low)

data = data.drop(data.index[index])

print(data.shape)

data.reset_index()
```

```
40.0
(619, 9)
```

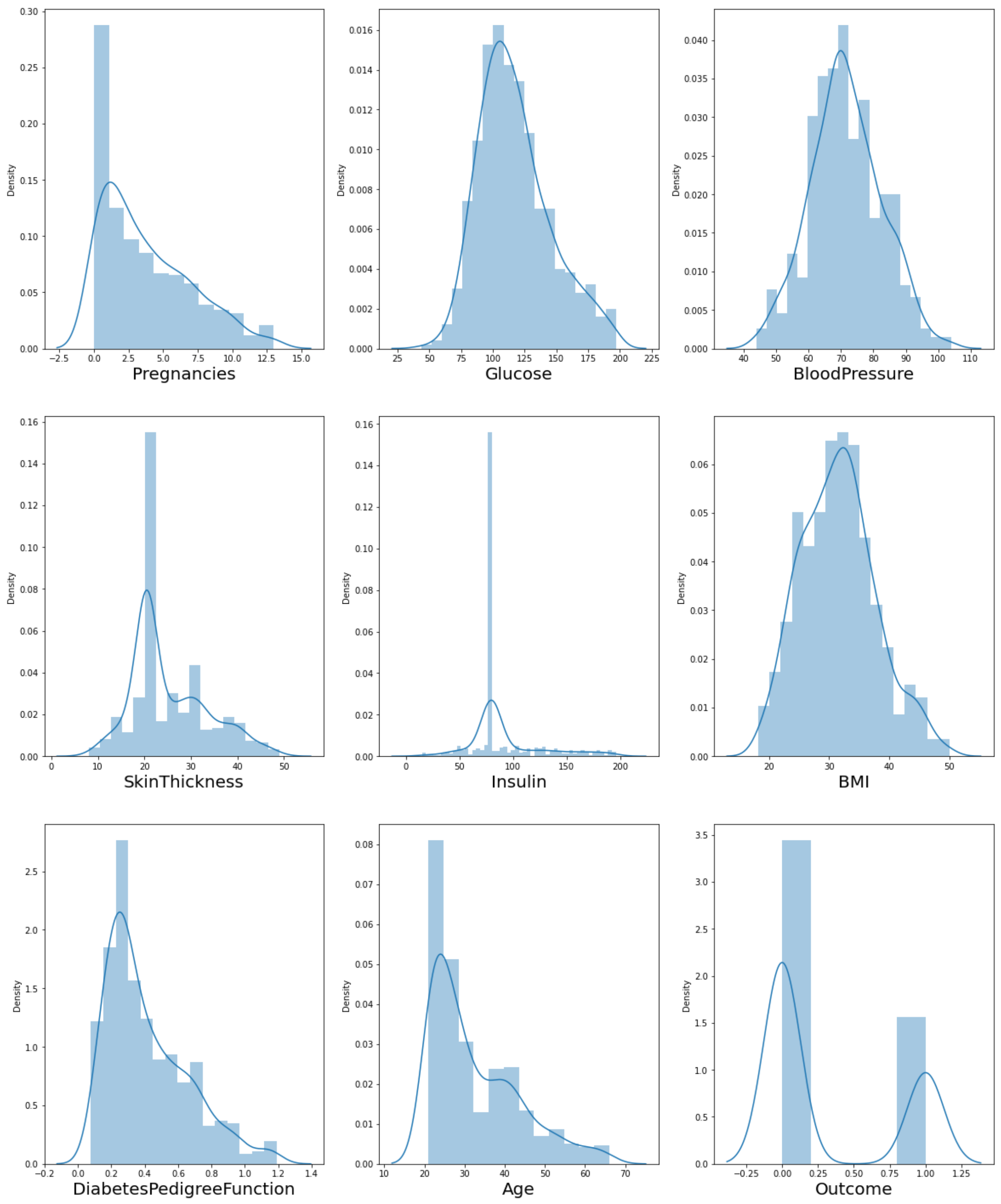
Out[23]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	1	85.0	66.0	29.000000	79.799479	26.6	
2	2	8	183.0	64.0	20.536458	79.799479	23.3	
3	3	1	89.0	66.0	23.000000	94.000000	28.1	
4	5	5	116.0	74.0	20.536458	79.799479	25.6	
...	...	...	...	...	...	...	...	
614	763	10	101.0	76.0	48.000000	180.000000	32.9	
615	764	2	122.0	70.0	27.000000	79.799479	36.8	
616	765	5	121.0	72.0	23.000000	112.000000	26.2	
617	766	1	126.0	60.0	20.536458	79.799479	30.1	
618	767	1	93.0	70.0	31.000000	79.799479	30.4	

619 rows × 10 columns

In [24]:

```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber=1

for column in data:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize=20)

    plotnumber+=1
plt.show()
```



The data looks much better now than before. We will start our analysis with this data now as we don't want to lose important information. If ourmodel doesn't work with accuracy, we will come back for more preprocessing.

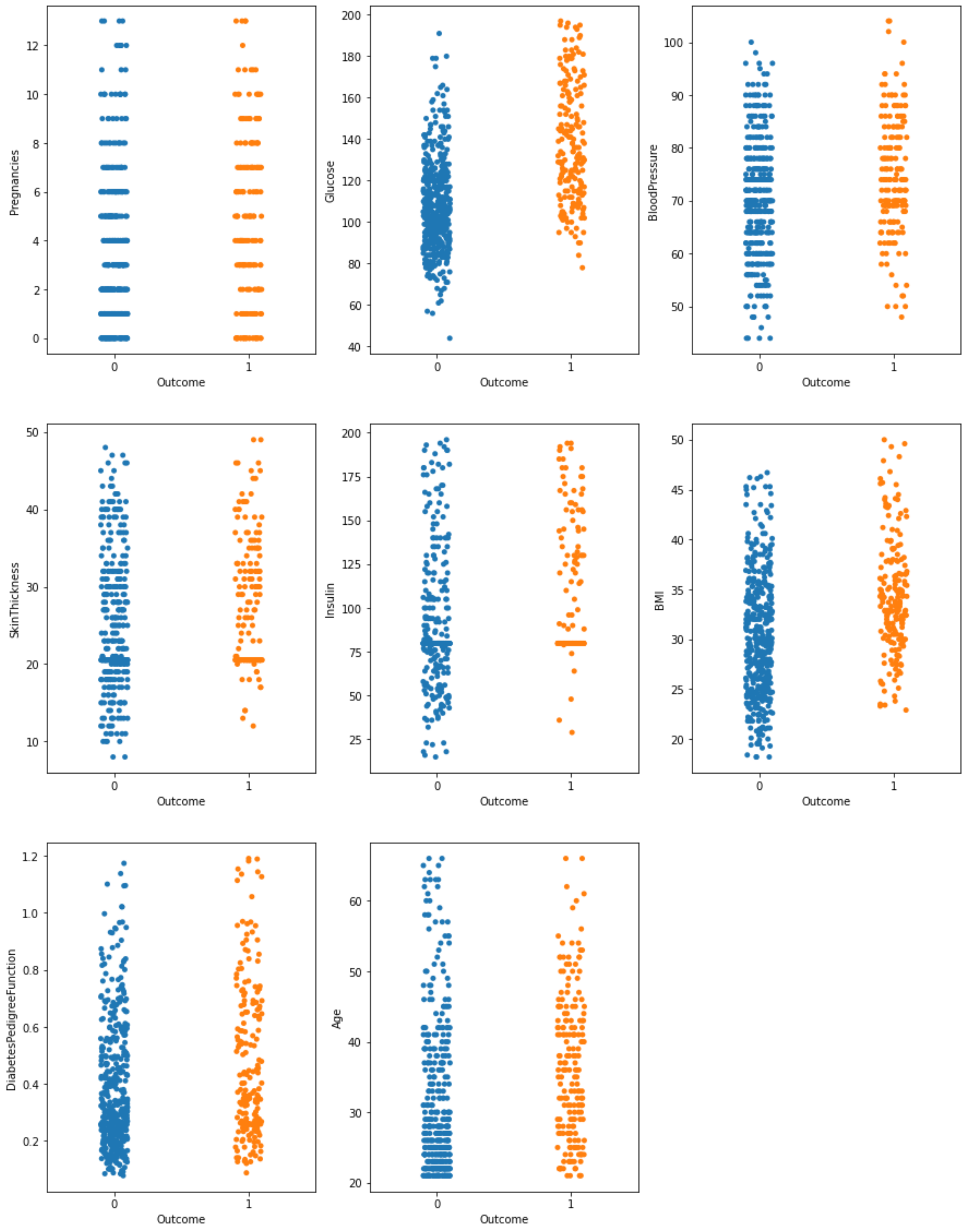
In [25]:

```
x= data.drop(columns=['Outcome'])
y= data['Outcome']
```

Before we fit our data to a model, let's visualize the relationship between our independent variables and the categories.

In [35]:

```
# Let's see how features are related to class

plt.figure(figsize=(15,20))
plotnumber=1

for column in x:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.stripplot(y,x[column])
    plotnumber+=1

plt.show()
```



Great!! Let's proceed by checking multicollinearity in the dependent variables. Before that, we should scale our data. Let's use the standard scaler for that.

In [40]:

```
scalar = StandardScaler()
x_scaled =scalar.fit_transform(x)
```

This is how our data looks now after scaling. Great, now we will check for multicollinearity using VIF(Variance Inflation factor)

In [41]:

```
x_scaled.shape[1]
```

Out[41]:

```
8
```

In [42]:

```
# finding variance inflation factor in each scaled column i.e x_scaled.shape[1] (1/(1-R2))
```

In [43]:

```
vif=pd.DataFrame()
vif["vif"]= [variance_inflation_factor(x_scaled, i) for i in range (x_scaled.shape[1])]
vif["Features"]= x.columns

# Let's check the values

vif
```

Out[43]:

	vif	Features
0	1.448654	Pregnancies
1	1.250247	Glucose
2	1.258898	BloodPressure
3	1.411508	SkinThickness
4	1.200759	Insulin
5	1.447599	BMI
6	1.038530	DiabetesPedigreeFunction
7	1.659799	Age

All the VIF values are less than 5 and are very low. That means no multicollinearity. Now, we can go ahead with fitting our data to the model. Beforethat, let's split our data in test and training set.

In [44]:

```
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.25,random_state=355)
```

In [45]:

```
log_reg=LogisticRegression()
log_reg.fit(x_train,y_train)
```

Out[45]:

```
LogisticRegression()
```

Let's see how well our model performs on the test data set.

In [46]:

```
y_pred=log_reg.predict(x_test)
y_pred
```

Out[46]:

```
array([1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       0], dtype=int64)
```

In [47]:

```
# Model Accuracy

accuracy=accuracy_score(y_test,y_pred)
accuracy
```

Out[47]:

```
0.7612903225806451
```

In [48]:

```
#confusion Matrix

conf_mat=confusion_matrix(y_test,y_pred)
conf_mat
```

Out[48]:

```
array([[94, 11],
       [26, 24]], dtype=int64)
```

In [49]:

```
#Let's check Accuracy manually

(94+24)/(94+24+11+26)
```

Out[49]:

```
0.7612903225806451
```

similarly you can manually calculate recall/precision/F1 score

In [50]:

```
from sklearn.metrics import classification_report
```

In [51]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.90      0.84       105
           1       0.69      0.48      0.56        50

    accuracy                           0.76       155
   macro avg       0.73      0.69      0.70       155
weighted avg       0.75      0.76      0.75       155
```

In [52]:

```
# ROC Curve

fpr,tpr,thresholds=roc_curve(y_test,y_pred)
```

#fpr -False Positive Rate increasing frequency #tpr -True Positive Rate increasing frequency #thresholds - Decreasing thresholds on the decisionfunction used to compute fpr and tpr. fpr, tpr, thresholds = roc_curve(y_test, y_pred)

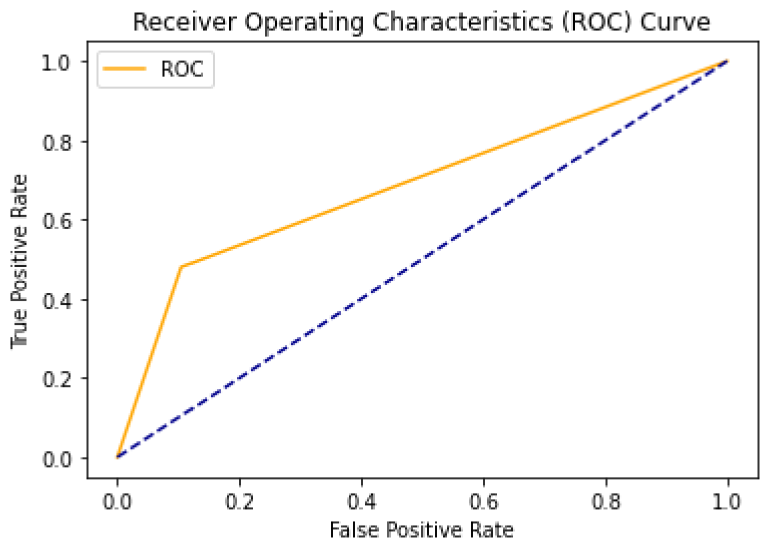
In [53]:

```
# thresholds[0] means no instances predicted (it should be read from 0 - max)
print('Threshold=', thresholds)
print('True Positive rate=', tpr)
print('False Positive rate=', fpr)
```

```
Threshold= [2 1 0]
True Positive rate= [0.   0.48 1.  ]
False Positive rate= [0.        0.1047619 1.       ]
```

In [54]:

```
plt.plot(fpr,tpr,color='orange', label='ROC')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristics (ROC) Curve')
plt.legend()
plt.show()
```



In [55]:

```
# How much area it is covering (AUC)

auc_score = roc_auc_score(y_test,y_pred)
print(auc_score)
```

```
0.6876190476190477
```

What is the significance of Roc curve and AUC?

In real life, we create various models using different algorithms that we can use for classification purpose. We use AUC to determine which model isthe best one to use for a given dataset. Suppose we have created Logistic regression, SVM as well as a clustering model for classification purpose.We will calculate AUC for all the models seperately. The model with highest AUC value will be the best model to use.

Advantages of Logisitic Regression

It is very simple and easy to implement.

The output is more informative than other classification algorithms

It expresses the relationship between independent and dependent variables

Very effective with linearly seperable data

Disadvantages of Logisitic Regression

Not effective with data which are not linearly seperable

Not as powerful as other classification models

Multiclass classifications are much easier to do with other algorithms than logisitic regression

It can only predict categorical outcomes

```
In [ ]:
```