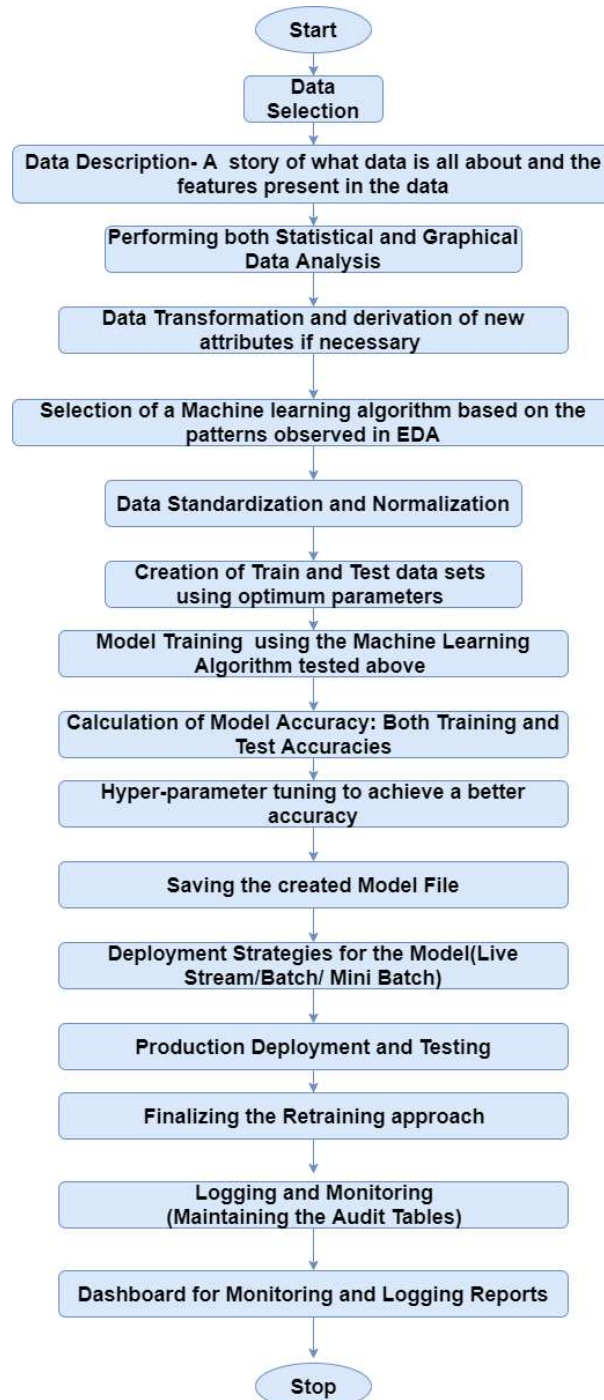


Application Flow

Before proceeding with the algorithm, let's first discuss the lifecycle of any machine learning model. This diagram explains the creation of a Machine Learning model from scratch and then taking the same model further with hyperparameter tuning to increase its accuracy, deciding the deployment strategies for that model and once deployed setting up the logging and monitoring frameworks to generate reports and dashboards based on the client requirements. A typical lifecycle diagram for a machine learning model looks like:



Bayes's Theorem

According to the Wikipedia, In probability theory and statistics, ** Bayes's theorem** (alternatively **Bayes's law* or *Bayes's rule*) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Mathematically, it can be written as:

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

Where A and B are events and $P(B) \neq 0$

- $P(A|B)$ is a conditional probability: the likelihood of event A occurring given that B is true.
- $P(B|A)$ is also a conditional probability: the likelihood of event B occurring given that A is true.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B respectively; they are known as the marginal probability.

Let's understand it with the help of an example:

The problem statement:

There are two machines which manufacture bulbs. Machine 1 produces 30 bulbs per hour and machine 2 produce 20 bulbs per hour. Out of all bulbs produced, 1 % turn out to be defective. Out of all the defective bulbs, the share of each machine is 50%. What is the probability that a bulb produced by machine 2 is defective?

We can write the information given above in mathematical terms as:

The probability that a bulb was made by Machine 1, $P(M1)=30/50=0.6$

The probability that a bulb was made by Machine 2, $P(M2)=20/50=0.4$

The probability that a bulb is defective, $P(Defective)=1\%=0.01$

The probability that a defective bulb came out of Machine 1, $P(M1 | Defective)=50\%=0.5$

The probability that a defective bulb came out of Machine 2, $P(M2 | Defective)=50\%=0.5$

Now, we need to calculate the probability of a bulb produced by machine 2 is defective i.e., $P(Defective | M2)$. Using the Bayes Theorem above, it can be written as:

$$P(Defective|M2) = \frac{P(M2|Defective)*P(Defective)}{P(M2)}$$

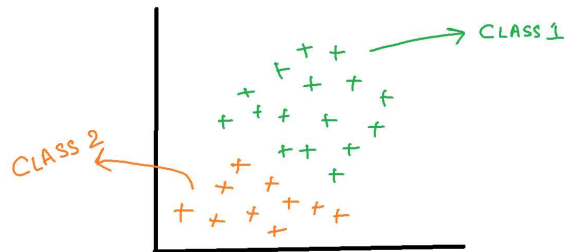
$$\text{Substituting the values, we get: } P(Defective|M2) = \frac{0.5*0.01}{0.4} = 0.0125$$

Task for you is to calculate the probability that a bulb produced by machine 1 is defective.

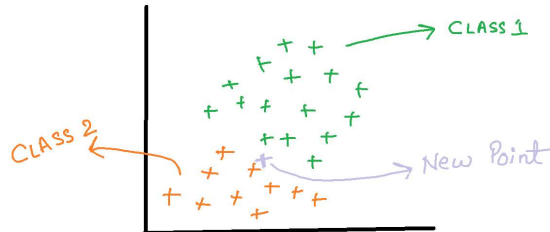
We'll extend this same understanding to understand the Naïve Bayes's Algorithm.

Algorithm steps:

1. Let's consider that we have a binary classification problem i.e., we have two classes in our data as shown below.



2. Now suppose if we are given with a new data point, to which class does that point belong to?



3. The formula for a point 'X' to belong in class1 can be written as:

$$P(\text{CLASS 1} | X) = \frac{P(X | \text{CLASS 1}) * P(\text{CLASS 1})}{P(X)}$$

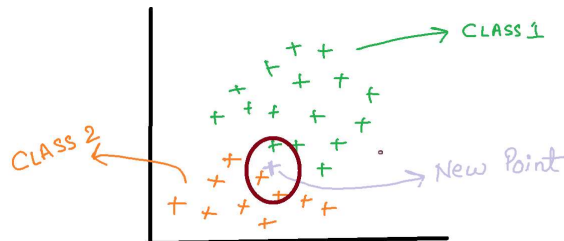
3. Likelihood
1. Prior Probability
4. Posterior Probability
2. Marginal Likelihood

Where the numbers represent the order in which we are going to calculate different probabilities.

4. A similar formula can be utilised for class 2 as well.

5. Probability of class 1 can be written as: $P(\text{class1}) = \frac{\text{Number of points in class1}}{\text{Total number of points}} = \frac{16}{26} = 0.62$

6. For calculating the probability of X, we draw a circle around the new point and see how many points(excluding the new point) lie inside that circle.



The points inside the circle are considered to be similar points. $P(X) = \frac{\text{Number of similar observation}}{\text{Total Observations}} = \frac{3}{26} = 0.12$

7. Now, we need to calculate the probability of a point to be in the circle that we have made given that it's of class 1. $P(X | \text{Class1}) = \frac{\text{Number of points in class1 inside the circle}}{\text{Total number of points in class1}} = \frac{1}{16} = 0.06$

8. We can substitute all the values into the formula in step 3. We get: $P(\text{Class1} | X) = \frac{0.06 * 0.62}{0.12} = 0.31$

9. And if we calculate the probability that X belongs to Class2, we'll get 0.69. It means that our point belongs to class 2.

The Generalization for Multiclass:

The approach discussed above can be generalised for multiclass problems as well. Suppose, $P_1, P_2, P_3 \dots P_n$ are the probabilities for the classes $C_1, C_2, C_3 \dots C_n$, then the point X will belong to the class for which the probability is maximum. Or mathematically the point belongs to the result of : $\text{argmax}(P_1, P_2, P_3 \dots P_n)$

The Difference

You can notice a major difference in the way in which the Naïve Bayes algorithm works from other classification algorithms. It does not first try to learn how to classify the points. It directly uses the label to identify the two separate classes and then it predicts the class to which the new point shall belong.

Why it is called Naïve Bayes?

The entire algorithm is based on Bayes's theorem to calculate probability. So, it also carries forward the assumptions for the Bayes's theorem. But those assumptions(that the features are independent) might not always be true when implemented over a real-world dataset. So, those assumptions are considered *Naïve* and hence the name.

Gaussian Naive Bayes

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. Go back to the normal distribution lecture to review the formulas for the Gaussian/Normal Distribution.

For example of using the Gaussian Distribution, suppose the training data contain a continuous attribute, x . We first segment the data by the class, and then compute the mean and variance of x in each class. Let μ_c be the mean of the values in x associated with class c , and let σ_c^2 be the variance of the values in x associated with class c . Then, the probability distribution of some value given a class, $p(x=v|c)$, can be computed by plugging v into the equation for a Normal distribution parameterized by μ_c and σ_c^2 . That is:

$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

Python Implementation

In [1]:

```
1 #Let's start with importing necessary libraries
2
3 import pandas as pd
4 import numpy as np
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.linear_model import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet, ElasticNetCV
7 from sklearn.model_selection import train_test_split
8 from statsmodels.stats.outliers_influence import variance_inflation_factor
9 from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
10 import matplotlib.pyplot as plt
11 import seaborn as sns
12 import scikitplot as skl
13 sns.set()
```

In [2]:

```
1 data = pd.read_csv("diabetes.csv") # Reading the Data
2 data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.621
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.281

In [3]:

```
1 data.describe()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
count	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	0.392003
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.471404
min	0.000000	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	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	0.990108

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 it with their respective mean values. Let's do the latter.

In [4]:

```
1 # replacing zero values with the mean of the column
2 data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
3 data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
4 data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
5 data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
6 data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [5]:

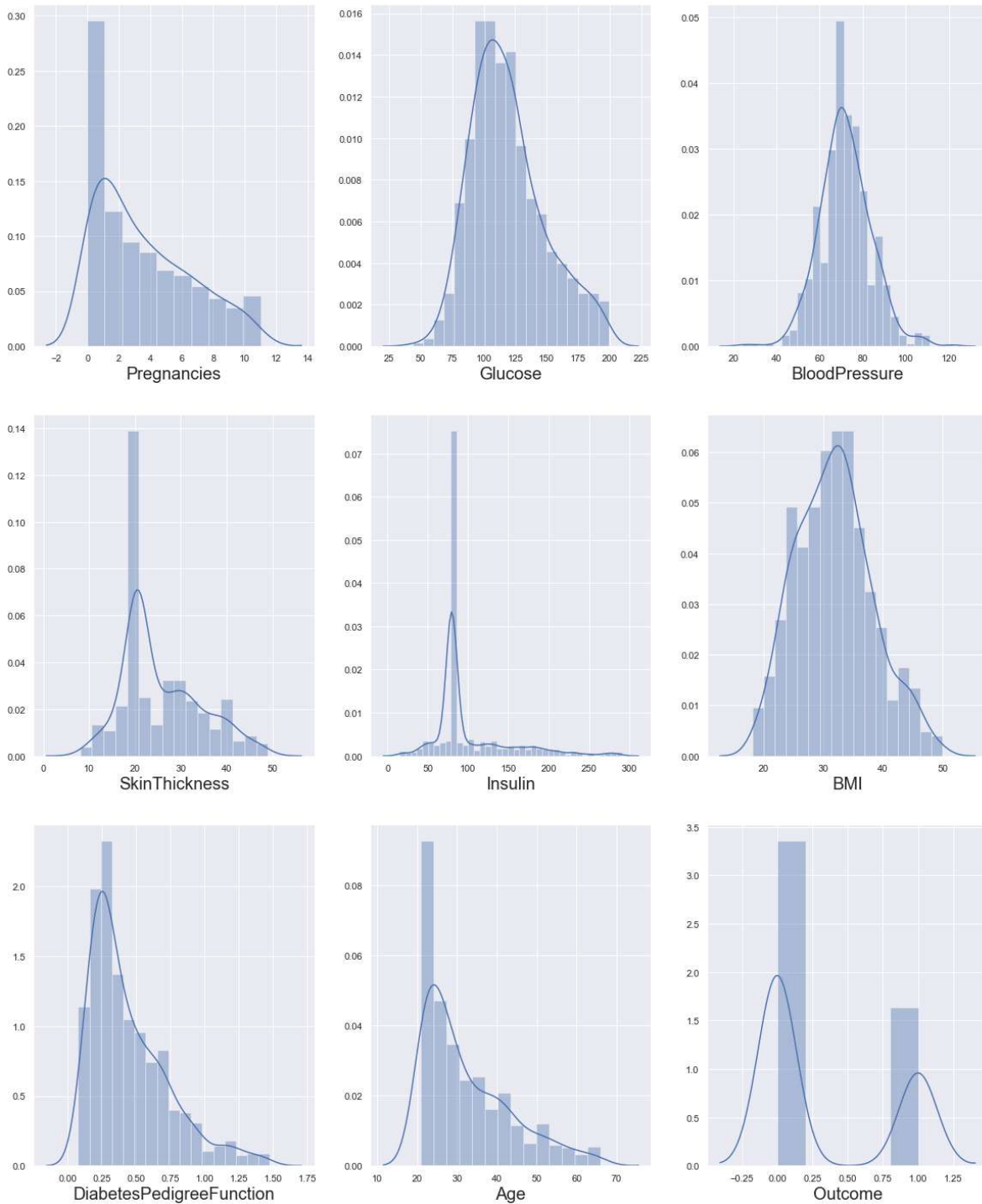
```
1 # Handling the Outliers
2
3 q = data['Pregnancies'].quantile(0.98)
4 # we are removing the top 2% data from the Pregnancies column
5 data_cleaned = data[data['Pregnancies']<q]
6 q = data_cleaned['BMI'].quantile(0.99)
7 # we are removing the top 1% data from the BMI column
8 data_cleaned = data_cleaned[data_cleaned['BMI']<q]
9 q = data_cleaned['SkinThickness'].quantile(0.99)
10 # we are removing the top 1% data from the SkinThickness column
11 data_cleaned = data_cleaned[data_cleaned['SkinThickness']<q]
12 q = data_cleaned['Insulin'].quantile(0.95)
13 # we are removing the top 5% data from the Insulin column
14 data_cleaned = data_cleaned[data_cleaned['Insulin']<q]
15 q = data_cleaned['DiabetesPedigreeFunction'].quantile(0.99)
16 # we are removing the top 1% data from the DiabetesPedigreeFunction column
17 data_cleaned = data_cleaned[data_cleaned['DiabetesPedigreeFunction']<q]
18 q = data_cleaned['Age'].quantile(0.99)
19 # we are removing the top 1% data from the Age column
20 data_cleaned = data_cleaned[data_cleaned['Age']<q]
```

In [6]:

```

1  # Let's see how data is distributed for every column
2  plt.figure(figsize=(20,25), facecolor='white')
3  plotnumber = 1
4
5  for column in data_cleaned:
6      if plotnumber<=9 :
7          ax = plt.subplot(3,3,plotnumber)
8          sns.distplot(data_cleaned[column])
9          plt.xlabel(column,fontsize=20)
10         #plt.ylabel('Salary',fontsize=20)
11         plotnumber+=1
12 plt.show()

```



In [7]:

```
1 X = data.drop(columns = ['Outcome'])
2 y = data['Outcome']
```

In [8]:

```
1 # we need to scale our data as well
2
3 scalar = StandardScaler()
4 X_scaled = scalar.fit_transform(X)
```

```
C:\Users\virat\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:64
5: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by StandardScaler.
    return self.partial_fit(X, y)
C:\Users\virat\Anaconda3\lib\site-packages\sklearn\base.py:464: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by StandardScaler.
    return self.fit(X, **fit_params).transform(X)
```


In [9]:

```
1 # This is how our data looks now after scaling.
2 X_scaled
```

Out[9]:

```
array([[ 0.63994726,  0.86527574, -0.0210444 , ...,  0.16725546,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.20598931, -0.51658286, ..., -0.85153454,
        -0.36506078, -0.19067191],
       [ 1.23388019,  2.01597855, -0.68176235, ..., -1.33182125,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 , -0.02240928, -0.0210444 , ..., -0.90975111,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.14197684, -1.01212132, ..., -0.34213954,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.94297153, -0.18622389, ..., -0.29847711,
        -0.47378505, -0.87137393]])
```

In [10]:

```
1 # now we will check for multicollinearity using VIF(Variance Inflation factor)
2 vif = pd.DataFrame()
3 vif["vif"] = [variance_inflation_factor(X_scaled,i) for i in range(X_scaled.shape[1])]
4 vif["Features"] = X.columns
5
6 #Let's check the values
7 vif
```

Out[10]:

	vif	Features
0	1.431075	Pregnancies
1	1.347308	Glucose
2	1.247914	BloodPressure
3	1.450510	SkinThickness
4	1.262111	Insulin
5	1.550227	BMI
6	1.058104	DiabetesPedigreeFunction
7	1.605441	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. Before that, let's split our data in test and training set.

In [11]:

```
1 x_train,x_test,y_train,y_test = train_test_split(X_scaled,y, test_size= 0.25, random_st
```

In [12]:

```
1 from sklearn.naive_bayes import GaussianNB
2 model = GaussianNB()
```

In [13]:

```
1 model.fit(x_train,y_train)
```

Out[13]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [18]:

```
1 import pickle
2 # Writing different model files to file
3 with open( 'modelForPrediction.sav', 'wb') as f:
4     pickle.dump(model,f)
5
6 with open('standardScalar.sav', 'wb') as f:
7     pickle.dump(scalar,f)
```

In [20]:

```
1 y_pred = model.predict(x_test)
```

In [21]:

```
1 print(accuracy_score(y_test, y_pred))
```

```
0.7864583333333334
```

In [22]:

```
1 # Confusion Matrix
2 conf_mat = confusion_matrix(y_test,y_pred)
3 conf_mat
```

Out[22]:

```
array([[109, 16],
       [ 25, 42]], dtype=int64)
```

In [23]:

```
1 true_positive = conf_mat[0][0]
2 false_positive = conf_mat[0][1]
3 false_negative = conf_mat[1][0]
4 true_negative = conf_mat[1][1]
```

In [24]:

```
1 # Breaking down the formula for Accuracy
2 Accuracy = (true_positive + true_negative) / (true_positive + false_positive + false_negative)
3 Accuracy
```

Out[24]:

```
0.7864583333333334
```

In [25]:

```
1 # Precision
2 Precision = true_positive/(true_positive+false_positive)
3 Precision
```

Out[25]:

0.872

In [26]:

```
1 # Recall
2 Recall = true_positive/(true_positive+false_negative)
3 Recall
```

Out[26]:

0.8134328358208955

In [27]:

```
1 # F1 Score
2 F1_Score = 2*(Recall * Precision) / (Recall + Precision)
3 F1_Score
```

Out[27]:

0.8416988416988417

In [28]:

```
1 # Area Under Curve
2 auc = roc_auc_score(y_test, y_pred)
3 auc
```

Out[28]:

0.7494328358208956

So far we have been doing grid search to maximise the accuracy of our model. Here, we'll follow a different approach. We'll create two models, one with Logistic regression and other with Naïve Bayes and we'll compare the AUC. The algorithm having a better AUC shall be considered for production deployment.

In [29]:

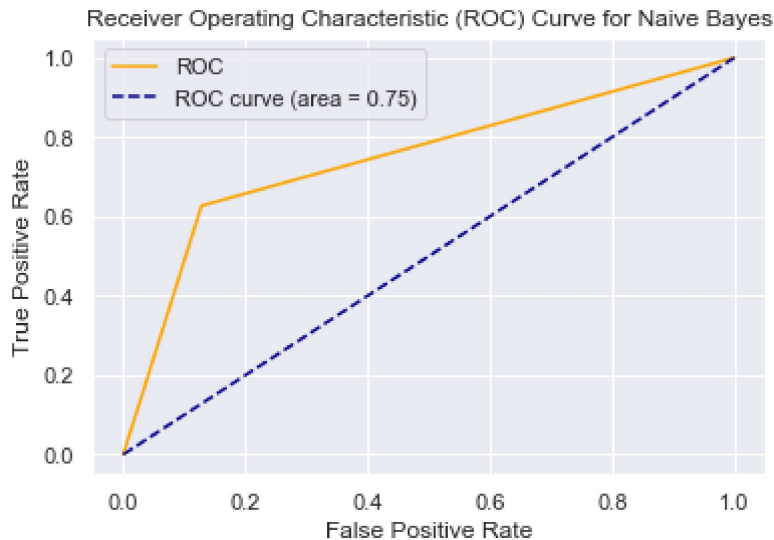
```
1 fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

In [46]:

```

1 plt.plot(fpr, tpr, color='orange', label='ROC')
2 plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve (area = %0.2)'
3 plt.xlabel('False Positive Rate')
4 plt.ylabel('True Positive Rate')
5 plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes')
6 plt.legend()
7 plt.show()

```



In [32]:

```

1 from sklearn.linear_model import LogisticRegression
2 log_reg = LogisticRegression()
3
4 log_reg.fit(x_train,y_train)

```

C:\Users\virat\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:
 433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
 FutureWarning)

Out[32]:

```

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
  intercept_scaling=1, max_iter=100, multi_class='warn',
  n_jobs=None, penalty='l2', random_state=None, solver='warn',
  tol=0.0001, verbose=0, warm_start=False)

```

In [33]:

```
1 y_pred_logistic = log_reg.predict(x_test)
```

In [35]:

```
1 accuracy_logistic = accuracy_score(y_test,y_pred_logistic)
2 accuracy_logistic
```

Out[35]:

```
0.7552083333333334
```

In [36]:

```
1 # Confusion Matrix
2 conf_mat = confusion_matrix(y_test,y_pred_logistic)
3 conf_mat
```

Out[36]:

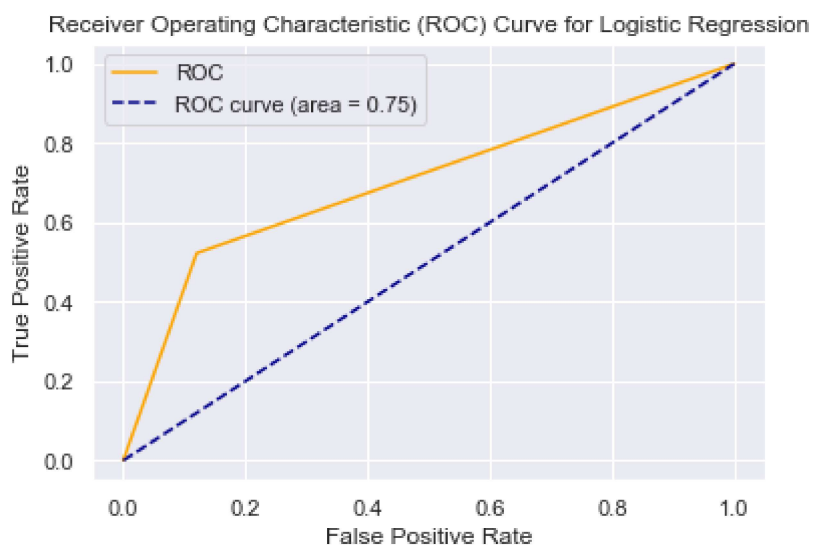
```
array([[110,  15],
       [ 32,  35]], dtype=int64)
```

In [37]:

```
1 # ROC
2 fpr_logistic, tpr_logistic, thresholds_logistic = roc_curve(y_test, y_pred_logistic)
3
```

In [38]:

```
1 plt.plot(fpr_logistic, tpr_logistic, color='orange', label='ROC')
2 plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)' % auc(fpr_logistic, tpr_logistic))
3 plt.xlabel('False Positive Rate')
4 plt.ylabel('True Positive Rate')
5 plt.title('Receiver Operating Characteristic (ROC) Curve for Logistic Regression')
6 plt.legend()
7 plt.show()
```



In [40]:

```
1 from sklearn.metrics import roc_auc_score
```

In [41]:

```
1 auc_naive=roc_auc_score(y_test,y_pred)
2 auc_naive
```

Out[41]:

0.7494328358208956

In [42]:

```
1 auc_logistic=roc_auc_score(y_test,y_pred_logistic)
2 auc_logistic
```

Out[42]:

0.7011940298507463

Here, you can see that the AUC for Naïve Bayes is more. So, we'll take that as our production-ready model.

Advantages:

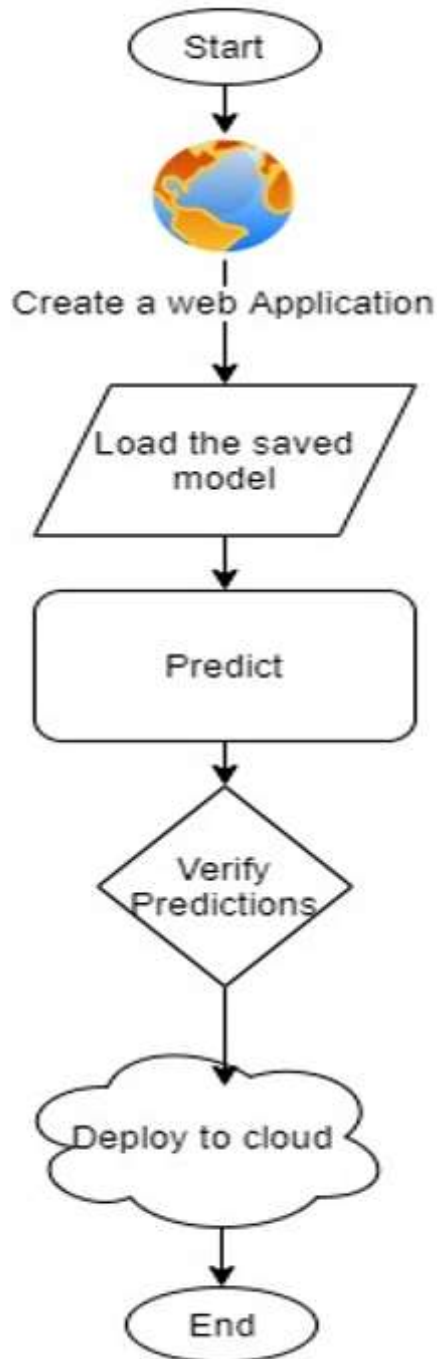
- Naive Bayes is extremely fast for both training and prediction as they not have to learn to create separate classes.
- Naive Bayes provides a direct probabilistic prediction.
- Naive Bayes is often easy to interpret.
- Naive Bayes has fewer (if any) parameters to tune

Disadvantages:

- The algorithm assumes that the features are independent which is not always the scenario
- Zero Frequency i.e. if the category of any categorical variable is not seen in training data set even once then model assigns a zero probability to that category and then a prediction cannot be made.

Cloud Deployment

Once the training is completed, we need to expose the trained model as an API for the user to consume it. For prediction, the saved model is loaded first and then the predictions are done using it. If the web app works fine, the same app is deployed to the cloud platform. The flow for that can be shown as:



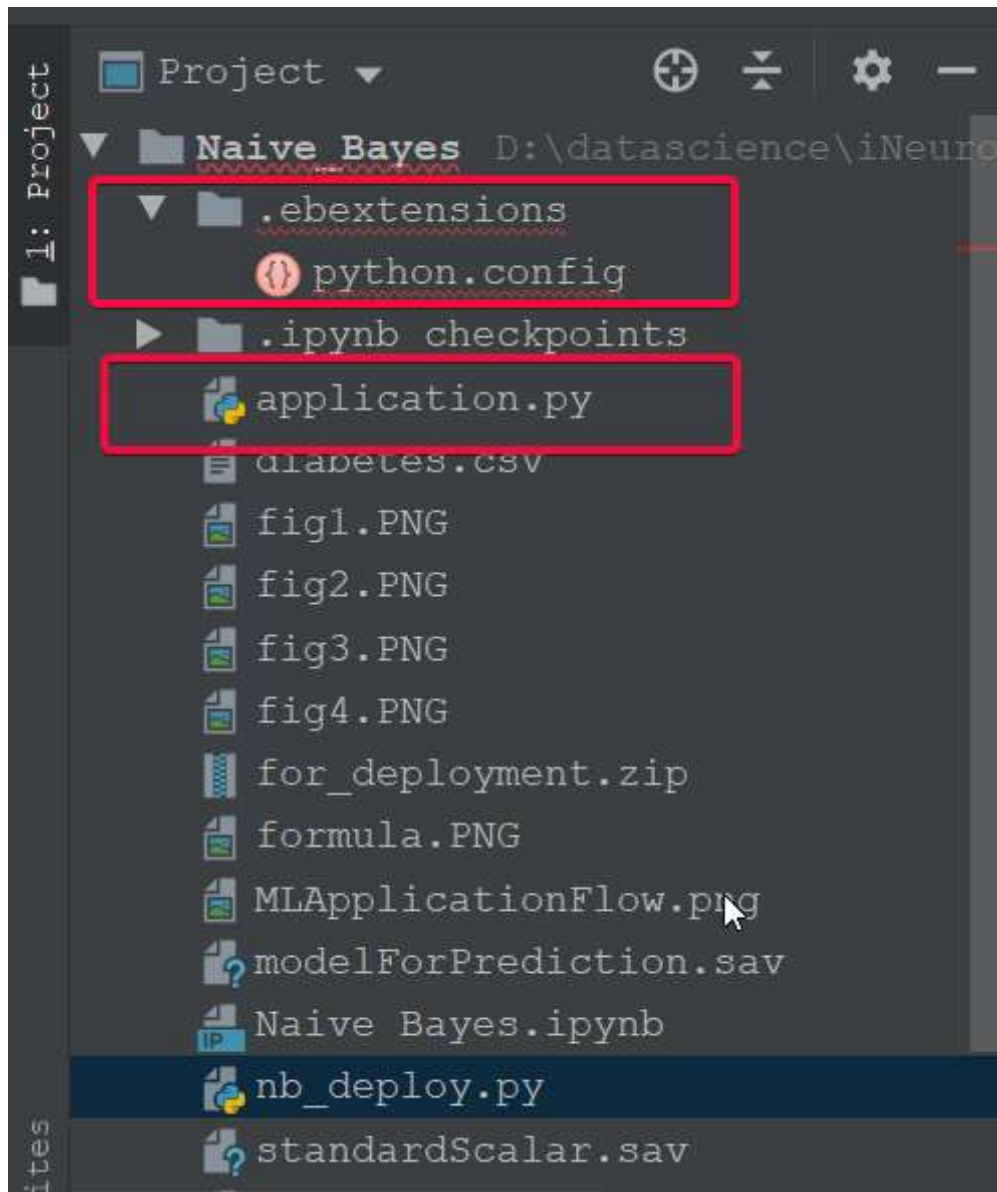
We'll deploy this model to the AWS Cloud Platform.

Pre-requisites for Cloud Deployment:

- Basic knowledge of flask framework.
- Any Python IDE installed(we are using PyCharm).
- An AWS account.
- Basic understanding of HTML.

The Flask App: As we'll expose the created model as a web API to be consumed by the client/client APIs, we'd do it using the flask framework.

- Create the project structure, as shown below:



The content for **application.py** is:

```
1 from flask import Flask, request, app
2 from flask import Response
3 from flask_cors import CORS
4 from nb_deploy import predObj
5
6 application = Flask(__name__) # initializing a flask app
7 app=application
8 CORS(app)
9 app.config['DEBUG'] = True
10
11
12 class ClientApi:
13
14     def __init__(self):
15         self.predObj = predObj()
16
17 @app.route("/predict", methods=['POST'])
18 def predictRoute():
19     try:
20         if request.json['data'] is not None:
21             data = request.json['data']
```



```

22         print('data is:      ', data)
23         pred=predObj()
24         res = pred.predict_log(data)
25
26         #result = clntApp.predObj.predict_log(data)
27         print('result is      ',res)
28         return Response(res)
29     except ValueError:
30         return Response("Value not found")
31     except Exception as e:
32         print('exception is    ',e)
33         return Response(e)
34
35
36 if __name__ == "__main__":
37     clntApp = ClientApi()
38     host = '0.0.0.0'
39     port = 5000
40     app.run(debug=True)
41     #httpd = simple_server.make_server(host, port, app)
42     # print("Serving on %s %d" % (host, port))
43     #httpd.serve_forever()

```

The content for **nb_deploy.py** is:

```

1 #Let's start with importing necessary libraries
2 import pickle
3 import pandas as pd
4
5 class predObj:
6
7     def predict_log(self, dict_pred):
8         with open("standardScalar.sav", 'rb') as f:
9             scalar = pickle.load(f)
10
11         with open("modelForPrediction.sav", 'rb') as f:
12             model = pickle.load(f)
13         data_df = pd.DataFrame(dict_pred,index=[1,])
14         scaled_data = scalar.transform(data_df)
15         predict = model.predict(scaled_data)
16         #predict = model.predict(data_df)
17         if predict[0] ==1 :
18             result = 'Diabetic'
19         else:
20             result ='Non-Diabetic'
21
22         return result
23
24
25
26

```

The content for **python.config** is:

```

1 option_settings:
2     "aws:elasticbeanstalk:container:python":
3         WSGIPath: application.py
4
5 files:

```

```

6  "/etc/httpd/conf.d/wsgi_custom.conf":
7  mode: "000644"
8  owner: root
9  group: root
10 content: |
11     WSGIApplicationGroup %{GLOBAL}

```

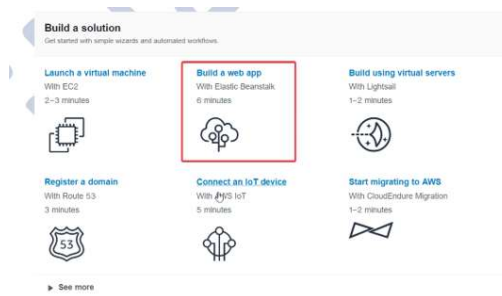
Points to consider before deployment

- The python application file should be named application.py
- Create a requirements.txt using **pip freeze > requirements.txt** from the project folder
- Create a folder **.ebextensions** and create a file **python.config** inside it. Make sure to populate the content of python.config, as shown above.
- Create the zip file from the project folder itself.

Name	Date modified	Type	Size
.ebextensions	1/21/2020 7:10 PM	File folder	
.idea	1/21/2020 7:46 PM	File folder	
.ipynb_checkpoints	1/21/2020 4:49 PM	File folder	
__pycache__	1/21/2020 7:22 PM	File folder	
application	1/21/2020 7:18 PM	Python File	2 KB
diabetes	1/2/2020 10:16 PM	Microsoft Excel Co...	24 KB
fig1	1/21/2020 4:33 PM	PNG File	12 KB
fig2	1/21/2020 4:37 PM	PNG File	15 KB
fig3	1/21/2020 4:44 PM	PNG File	19 KB
fig4	1/21/2020 5:00 PM	PNG File	16 KB
for_deployment	1/21/2020 7:38 PM	WinRAR ZIP archive	4 KB
formula	1/21/2020 3:56 PM	PNG File	6 KB
MLApplicationFlow	1/6/2020 6:58 PM	PNG File	75 KB
modelForPrediction.sav	1/21/2020 6:07 PM	SAV File	1 KB
Naive Bayes	1/21/2020 7:45 PM	IPYNB File	284 KB
nb_deploy	1/21/2020 7:16 PM	Python File	1 KB
project_structure	1/21/2020 7:40 PM	PNG File	32 KB
requirements	1/21/2020 7:45 PM	Text Document	1 KB
standardScaler.sav	1/21/2020 6:07 PM	SAV File	1 KB
testing_pipeline	12/27/2019 6:19 PM	PNG File	71 KB

Deployment Process

- Go to <https://aws.amazon.com/> (<https://aws.amazon.com/>) and create an account if already don't have one.
- Go to the console and go to the 'Build a web app' section and click it.



- Give the name of the application, give platform as python, and select the option to upload your code.

Create a web app

Create a new application and environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

Application information

Application name: Naive-Bayes-academics
Up to 100 Unicode characters, not including forward slash (/)

Application tags

Base configuration

Platform: Python
Choose Configure more options for more platform configuration options.

Application code

Get started right away with sample code

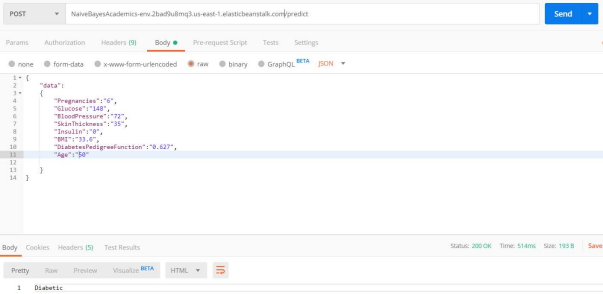
☒ Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

ZIP or WAR

☐ Application code tags

- Click on Create application to upload your code and create the app

Final Result:



In []:

1