Introduction to Data Science

(CSE 372)

"Predicting the level of Maternal Health Risk"

Project by Group 4NN Members:
20UCS147 Pratham Mehta
20UCS185 Shrey Parikh
20UCS187 Shreyash Sharma
20UCS225 Vatsal Mehta

Course Instructors:
Dr. Subrat Dash
Dr. Sakthi Balan
Dr. Aloke Datta

Department of Computer Science and Engineering

The LNM Institute of Information Technology

Contents

Problem Statement

To obtain a dataset from one of the given sources and perform the following steps:

1. Data pre-processing and its visualization
2. Explain all the inferences we got from our data.
3. Explain what ML Classification Algorithms are being used and the reasoning behind using them.
4. Implementing those algorithms.
5. Output the result of the testing set and its visualization.

The Python Libraries used in the process are as follows:

- ➔ Numpy
- ➔ Pandas
- ➔ Matplotlib
- ➔ Scikit_learn
- ➔ Seaborn

The project is implemented fully using a Jupyter Notebook.

Dataset Description

Data has been collected from different hospitals, community clinics, maternal health cares from the rural areas of Bangladesh through the IoT based risk monitoring system. The goal is to model the health risks associated with maternity based on different parameters.

Some Properties of the dataset are:

❖ Data Set Characteristics: Multivariate
❖ Area: Life
❖ Number of instances: 1014
❖ Number of Attributes: 7
❖ Attribute characteristics: Int64, Float64
❖ Associated Tasks: Classification
❖ Date Donated: 2020-12-31

The attributes are Age, Systolic Blood Pressure as SystolicBP, Diastolic BP as DiastolicBP, Blood Sugar as BS, Body Temperature as BodyTemp, HeartRate and RiskLevel. All these are the responsible and significant risk factors for maternal mortality, which is one of the main concerns of the SDG of the UN.

Attribute Description:

- **Age**: Any age in years when a woman is pregnant.
- **SystolicBP**: Upper value of Blood Pressure in mmHg, another significant attribute during pregnancy.
- **DiastolicBP**: Lower value of Blood Pressure in mmHg, another significant attribute during pregnancy.
- **BS**: Blood glucose levels are in terms of a molar concentration, mmol/L.

- **HeartRate**: A normal resting heart rate in beats per minute.
- **RiskLevel**: Predicted Risk Intensity Level during pregnancy considering the previous attribute.

Source: https://archive.ics.uci.edu/ml/datasets/Maternal+Health+Risk+Data+Set

All the above mentioned information regarding the dataset has been taken from this source as well.

Data Analysis

★ Before we can get started on our project, it is necessary to import the required libraries in our notebook.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix,accuracy_score,plot_confusion_matrix
import seaborn as sns
```

★ Then we read the downloaded dataset into our notebook using the **read_csv()** from **Pandas** so that we are able to use it. We also print the first 10 columns.

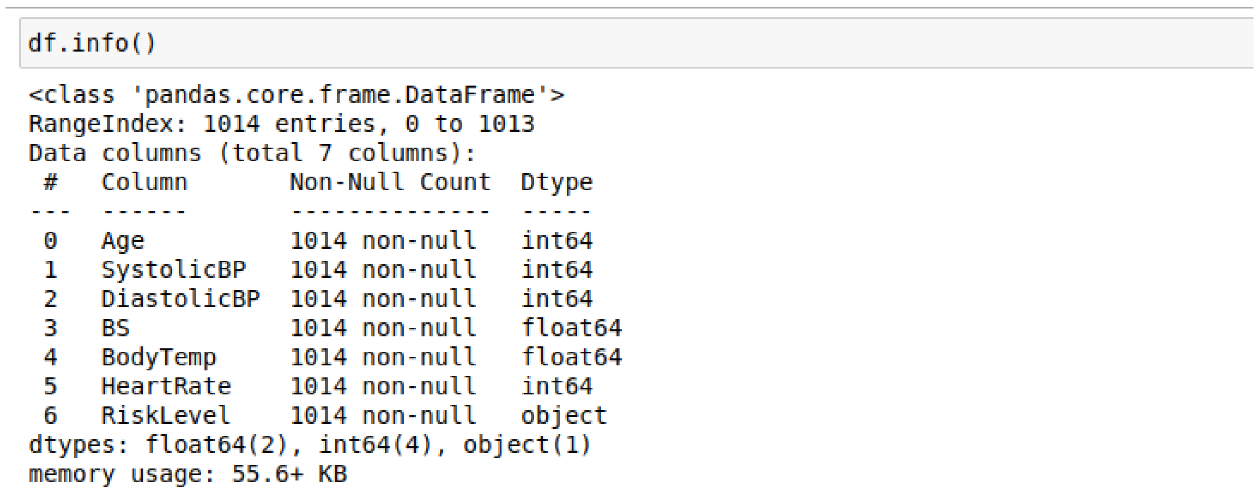
```
df = pd.read_csv("Maternal Health Risk Data Set.csv")
```

```
print(df.shape)
df.head(10)
```

```
(1014, 7)
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.00	98.0	86	high risk
1	35	140	90	13.00	98.0	70	high risk
2	29	90	70	8.00	100.0	80	high risk
3	30	140	85	7.00	98.0	70	high risk
4	35	120	60	6.10	98.0	76	low risk
5	23	140	80	7.01	98.0	70	high risk
6	23	130	70	7.01	98.0	78	mid risk
7	35	85	60	11.00	102.0	86	high risk
8	32	120	90	6.90	98.0	70	mid risk
9	42	130	80	18.00	98.0	70	high risk

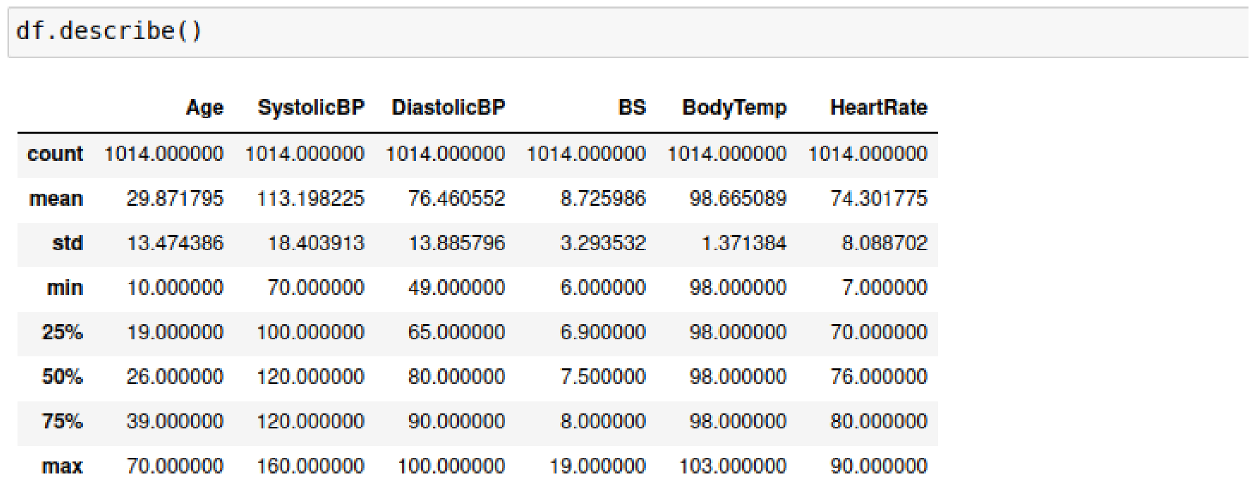
★ **DataFrame.info()** is then used to extract some metadata about our dataset. For eg: The total number of attributes and the datatype of each attribute.



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1014 entries, 0 to 1013
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          1014 non-null   int64  
 1   SystolicBP   1014 non-null   int64  
 2   DiastolicBP  1014 non-null   int64  
 3   BS           1014 non-null   float64
 4   BodyTemp     1014 non-null   float64
 5   HeartRate    1014 non-null   int64  
 6   RiskLevel    1014 non-null   object 
dtypes: float64(2), int64(4), object(1)
memory usage: 55.6+ KB
```

★ **DataFrame.describe()** is then used to extract the basic statistical information about the dataset, such as mean, maximum and minimum values for each attribute.



```
df.describe()
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate
count	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000
mean	29.871795	113.198225	76.460552	8.725986	98.665089	74.301775
std	13.474386	18.403913	13.885796	3.293532	1.371384	8.088702
min	10.000000	70.000000	49.000000	6.000000	98.000000	7.000000
25%	19.000000	100.000000	65.000000	6.900000	98.000000	70.000000
50%	26.000000	120.000000	80.000000	7.500000	98.000000	76.000000
75%	39.000000	120.000000	90.000000	8.000000	98.000000	80.000000
max	70.000000	160.000000	100.000000	19.000000	103.000000	90.000000

What we can infer from this analysis is:

- The dataset has 7 columns and 1014 rows.
- The attributes have data types such as Int64(4), Float64(2), and object(1)
- There are no null values in our dataset.
- There aren't any unnecessary attributes in our dataset which need to be removed at this point.

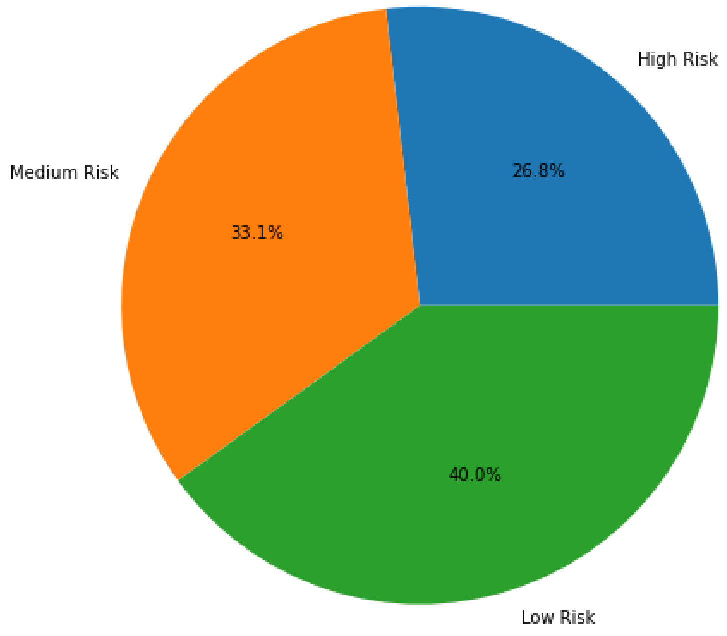
Data Visualization

Pie Chart

```
df['RiskLevel'].value_counts()
```

```
low risk     406
mid risk     336
high risk    272
Name: RiskLevel, dtype: int64
```

```
labels=['High Risk', 'Medium Risk', 'Low Risk']
values=[272, 336, 406]
plt.pie(values, labels=labels, autopct='%1.1f%%', radius=2)
plt.show()
```



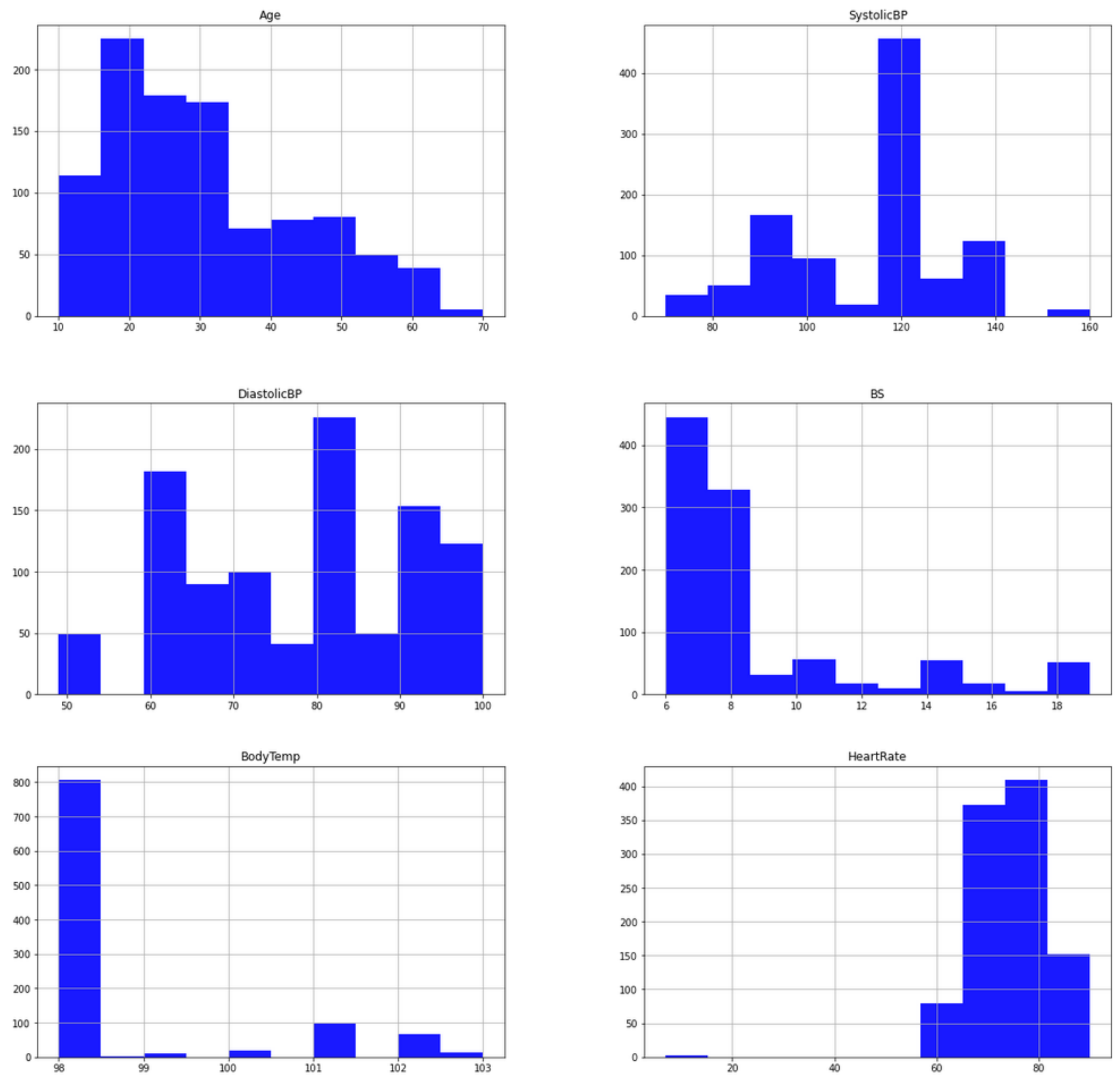
We extract the information about the number of records belonging to the different risk categories and then visualize it in the form of a Pie Chart as shown above.

As we can notice, the maximum records are of the type *Low Risk* at 40%, while the number of records of the type *High Risk* is considerably lower at 26.8%. The number of *Medium Risk* records sit somewhere in between at 33.1%. So, there might be a slight case of Class Imbalance Problem.

Histogram

DataFrame.hist() is used to plot a histogram. This histogram is used to show the ranges of values and their frequencies for the attributes in our dataset.

```
df.hist(figsize=(20,20), alpha=(0.9), color='blue')
plt.show()
```



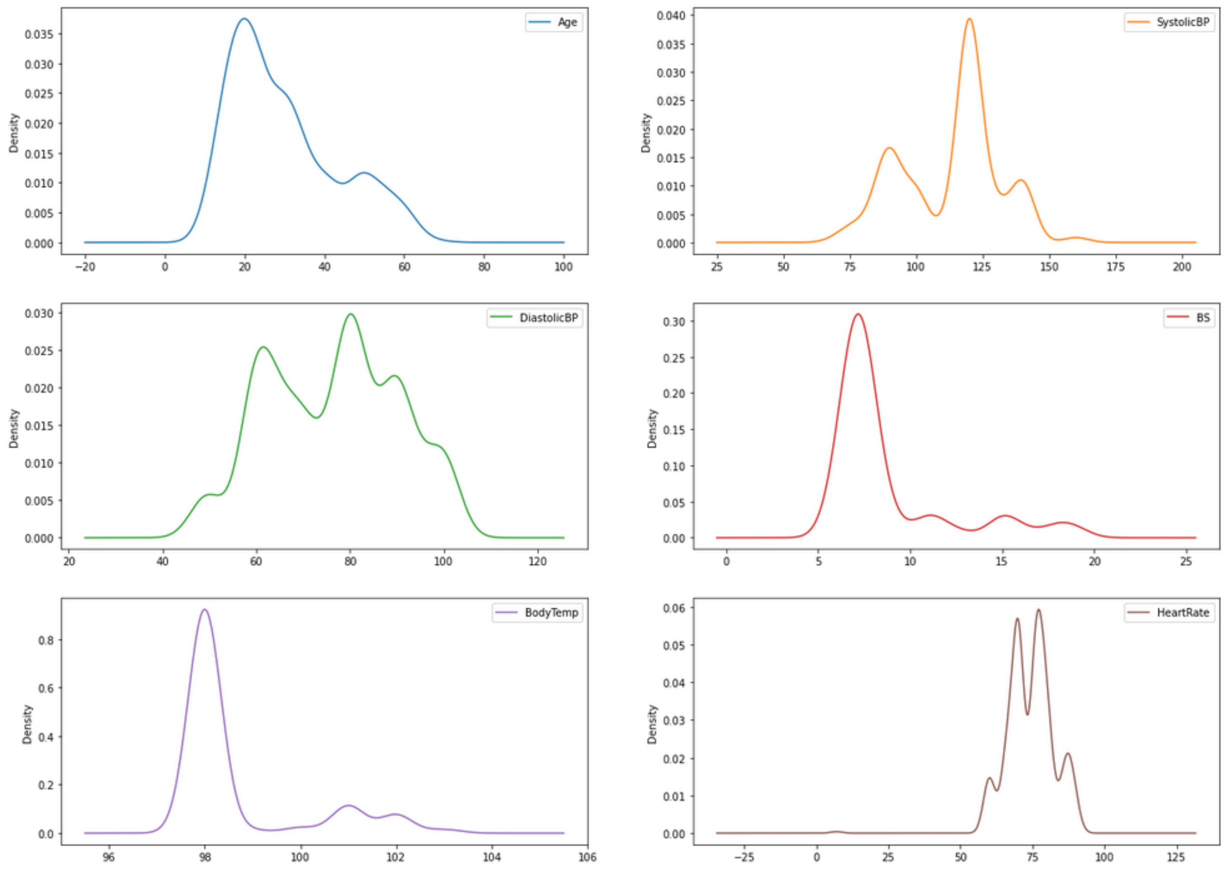
- These histograms give us some useful information about the attributes and their ranges.

- The X-axis denotes the values for the attributes.
- The Y-axis denotes the frequency corresponding to the values.
- It is easy to identify the values with the most occurrences in each attribute by simply watching the histogram corresponding to it.

Mean Plot

```
meanplot = df.plot(kind='density', subplots='true', layout=(4, 2), figsize=(20, 20), sharex=False)
plt.show()
```

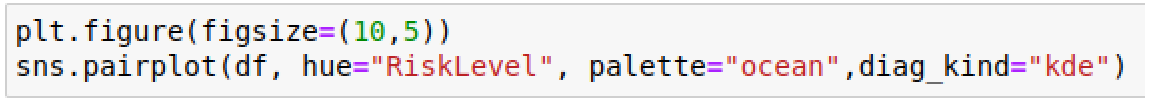
By using the above code, we are able to showcase the density plots for the attributes of our dataset, as shown below.



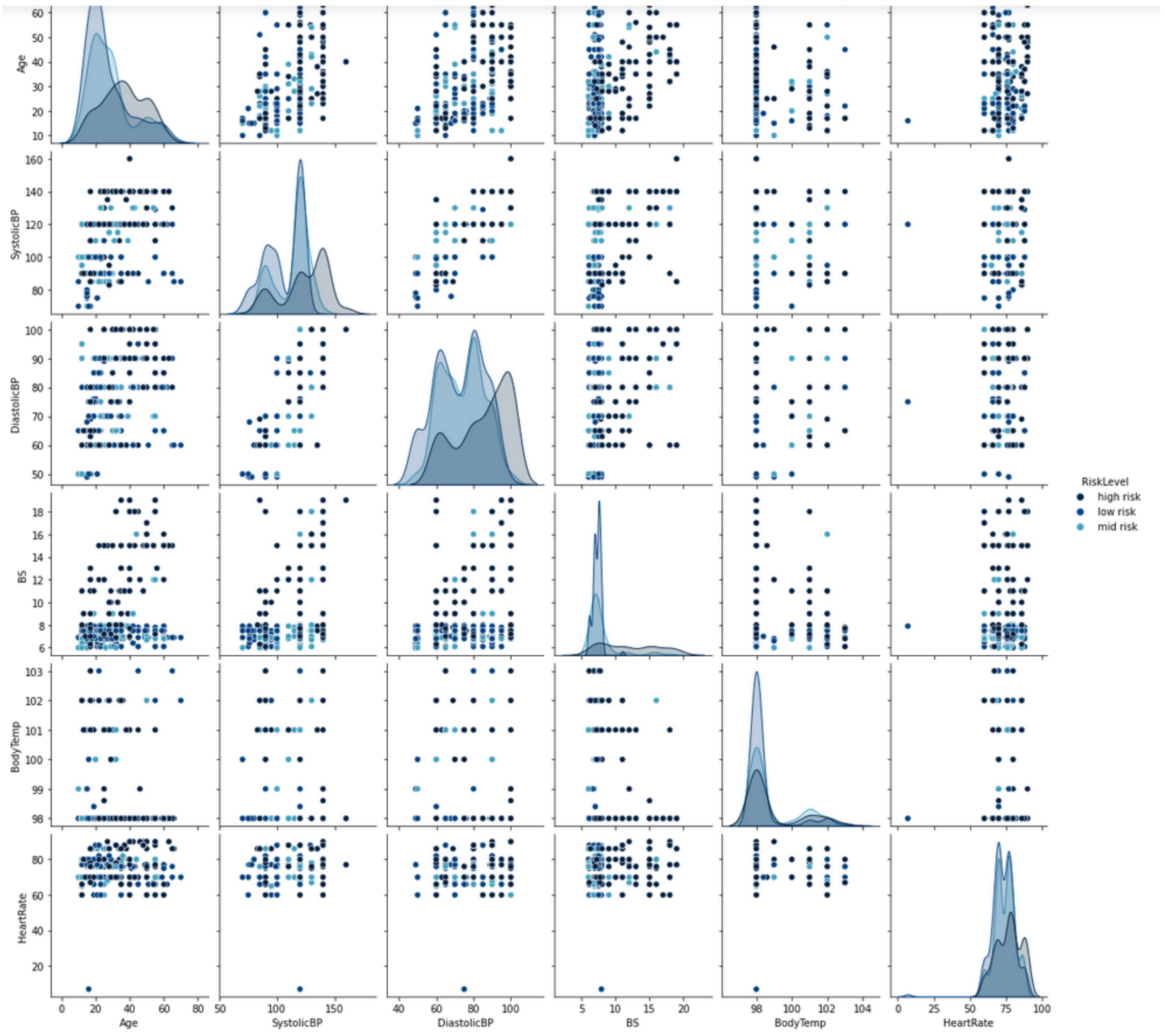
It shows us where the data is most dense and where it is sparse, in a continuous way. For example, we can see that the records having the age of around 20 years are the most abundant, generally decreasing in density as we move away from the peak.

Pair Plot

we use **seaborn.pairplot()** to pairwise plot all our attributes. The diagonal plots are univariate and the rest are bivariate plots. It shows the (n,2) combination relationship of attributes of the dataset.



```
plt.figure(figsize=(10,5))
sns.pairplot(df, hue="RiskLevel", palette="ocean",diag_kind="kde")
```



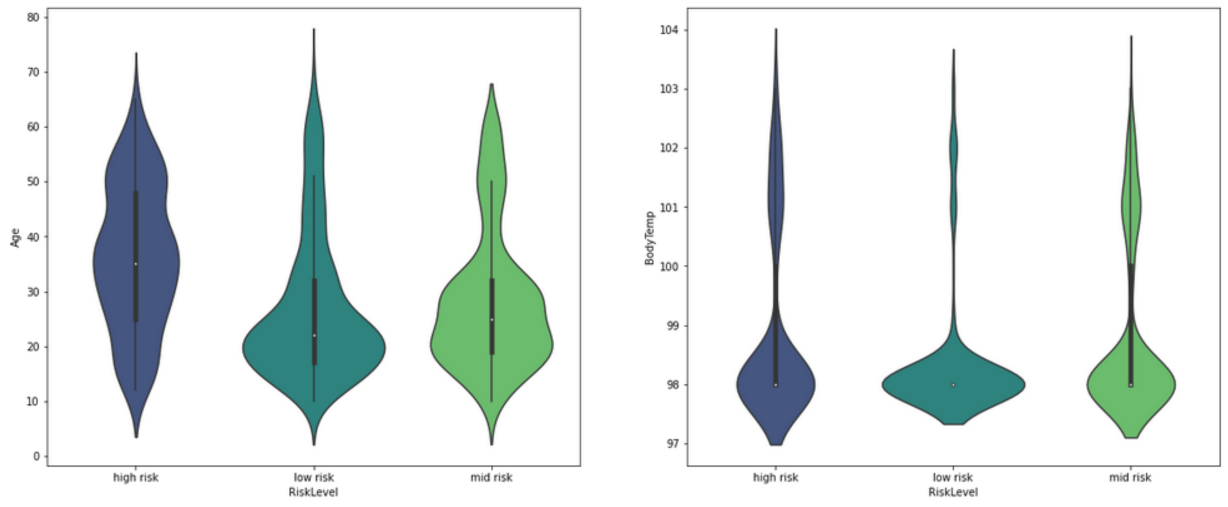
- The scatter plots give us the relationship between any two attributes of our dataset.
- One on the X-axis and other on the Y-axis.
- Each Scatter plot can be studied individually to extract and infer even greater insight in the relationships between the two attributes.
- The dark blue circles represent the cases with the highest risk, the slightly lighter blue represent the medium risk cases, while the low risk cases are represented by the light blue circles.

Violin Plot

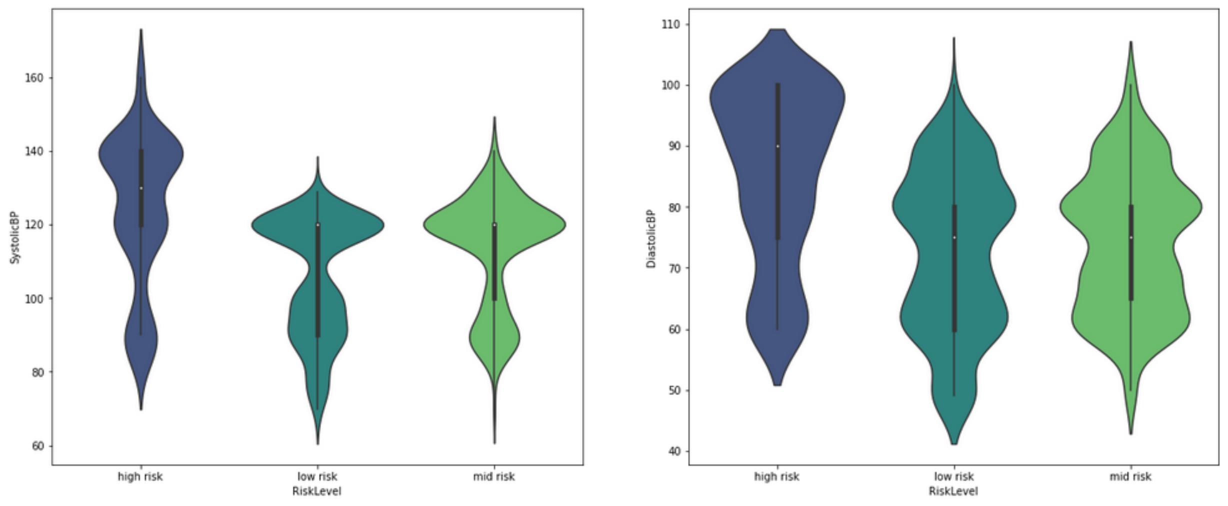
We implement a Violin Plot by using **Seaborn.violinplot()** as shown.

```
def violin(att1,att2):
    plt.figure(figsize=(20,8))

    plt.subplot(1,2,1)
    sns.violinplot(data=df,y=att1,x="RiskLevel",palette='viridis')

    plt.subplot(1,2,2)
    sns.violinplot(data=df,y=att2,x="RiskLevel",palette='viridis')
    plt.show()
```

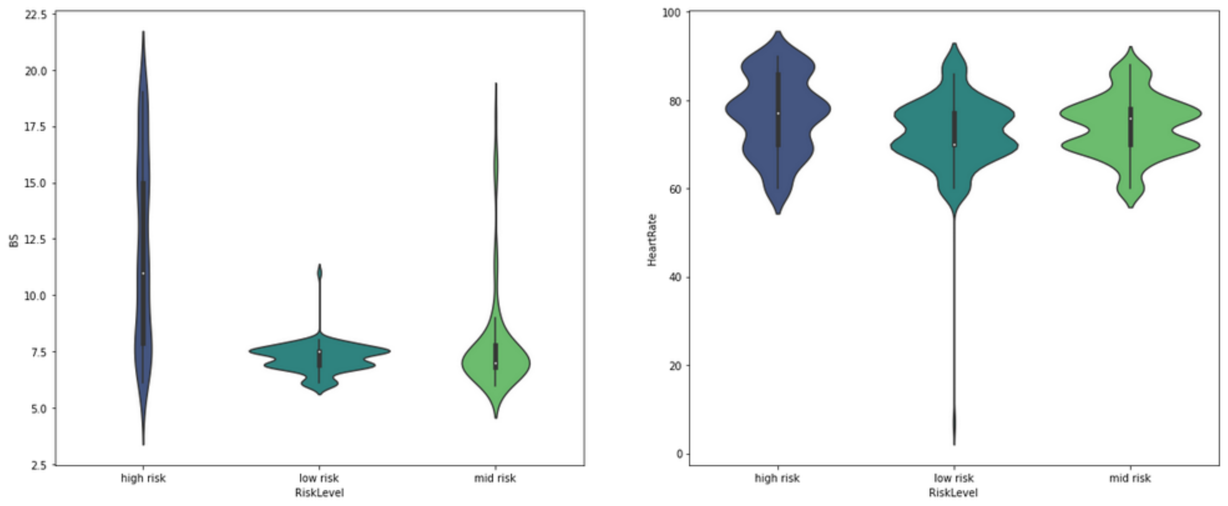
```
violin("Age", "BodyTemp")
```



```
violin("SystolicBP", "DiastolicBP")
```



```
violin("BS", "HeartRate")
```



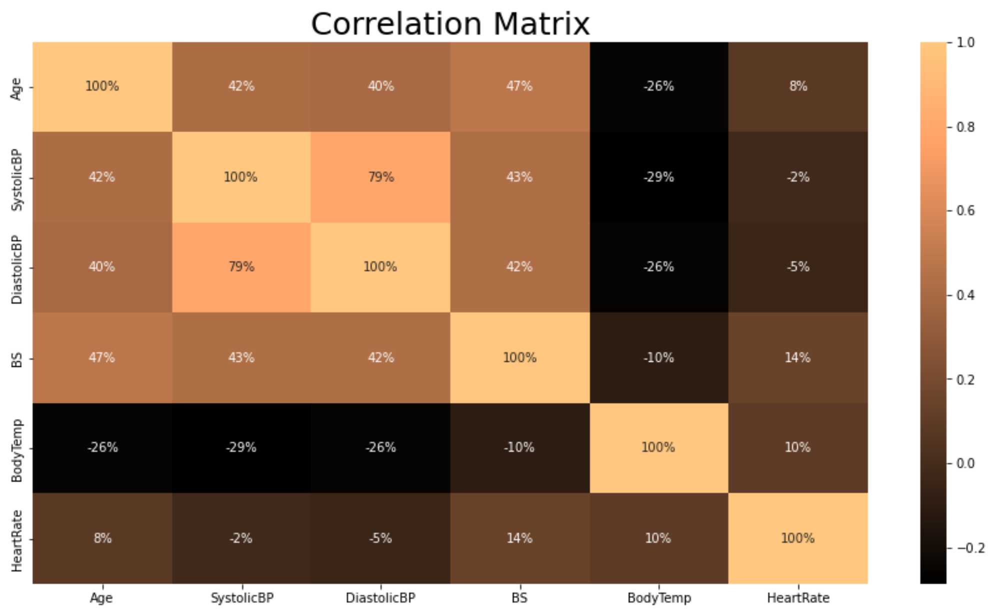
- A violin plot is a hybrid of a box plot and a kernel density plot, which shows peaks in the data.
- It is used to visualize the distribution of numerical data.
- Unlike a box plot that can only show summary statistics, violin plots depict summary statistics and the density of each variable.

Correlation Matrix

We then move towards the Correlation Matrix.

The Correlation Matrix is implemented by passing the **DataFrame.corr()** function inside the **Seaborn.heatmap()** function

```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),annot=True,fmt=".0%",cmap='copper')
plt.title('Correlation Matrix',size=25)
plt.show()
```



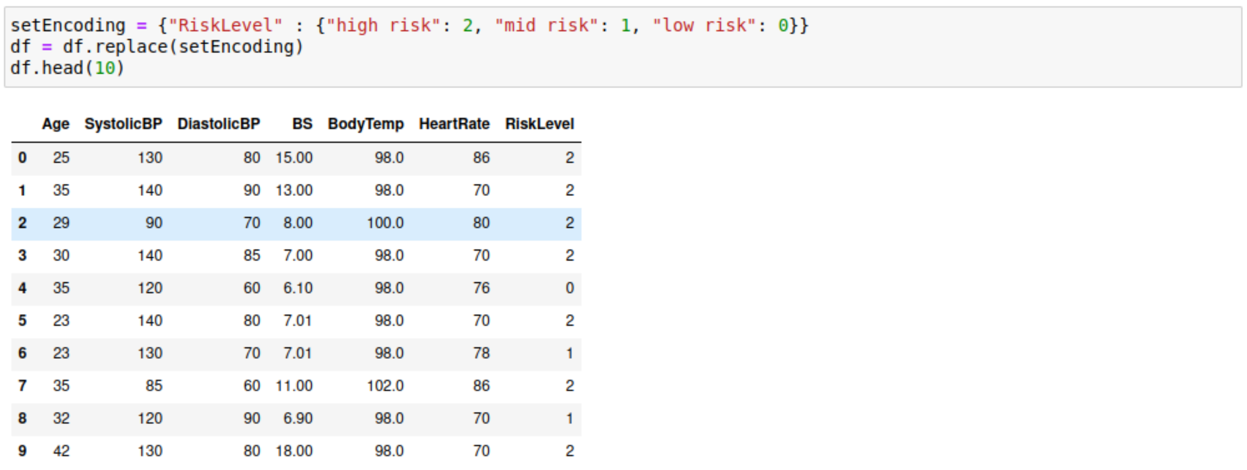
- A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data.
- Hence, from this correlation matrix we can see that attributes such as Age and BodyTemp have negative correlation value, which we can also confirm from our logic as Age is in no way determinant to the Body temperature of a person.
- While we can see that the attributes SystolicBP and DiastolicBP have a high positive correlation value at around 80%, which is obvious.

Data Preprocessing

We now arrive at the Data Preprocessing step after analyzing and visualizing our dataset thoroughly.

Since the last column of our Dataset (*RiskLevel)* is not in an integer format, we have to encode it.

We can do it by simply making a dictionary and using it to replace the occurrences of *high risk* with the number 2, *mid risk* with the number 1, and *low risk* with the number 0.



```
setEncoding = {"RiskLevel" : {"high risk": 2, "mid risk": 1, "low risk": 0}}
df = df.replace(setEncoding)
df.head(10)
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.00	98.0	86	2
1	35	140	90	13.00	98.0	70	2
2	29	90	70	8.00	100.0	80	2
3	30	140	85	7.00	98.0	70	2
4	35	120	60	6.10	98.0	76	0
5	23	140	80	7.01	98.0	70	2
6	23	130	70	7.01	98.0	78	1
7	35	85	60	11.00	102.0	86	2
8	32	120	90	6.90	98.0	70	1
9	42	130	80	18.00	98.0	70	2

We might have used the LabelEncoder() function, but it does not care for the priority and would have possibly resulted in, although technically unique, but confusing values for the *RiskLevel* attribute. Hence, we end up using a dictionary.

Now, before we can proceed onto the classification algorithms we first have to isolate the Input and Output attributes, normalize the dataset and split it into two sets: Training and Testing. We choose the ratio of 3 parts Test and 7 parts Training.

```
X = df.drop(['RiskLevel'], axis=1)
Y = df.loc[:, ['RiskLevel']]
```

```
for attr in X.columns:
    max_val = df[attr].max()
    min_val = df[attr].min()
    X[attr] = (X[attr] - min_val) / (max_val - min_val)

X.head(10)
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate
0	0.250000	0.666667	0.607843	0.692308	0.0	0.951807
1	0.416667	0.777778	0.803922	0.538462	0.0	0.759036
2	0.316667	0.222222	0.411765	0.153846	0.4	0.879518
3	0.333333	0.777778	0.705882	0.076923	0.0	0.759036
4	0.416667	0.555556	0.215686	0.007692	0.0	0.831325
5	0.216667	0.777778	0.607843	0.077692	0.0	0.759036
6	0.216667	0.666667	0.411765	0.077692	0.0	0.855422
7	0.416667	0.166667	0.215686	0.384615	0.8	0.951807
8	0.366667	0.555556	0.803922	0.069231	0.0	0.759036
9	0.533333	0.666667	0.607843	0.923077	0.0	0.759036

```
print(Y)
```

```
      RiskLevel
0             2
1             2
2             2
3             2
4             0
...         ...
1009          2
1010          2
1011          2
1012          2
1013          1

[1014 rows x 1 columns]
```

We separate the RiskLevel and rest of the attributes into Y and X, for Output and Input.

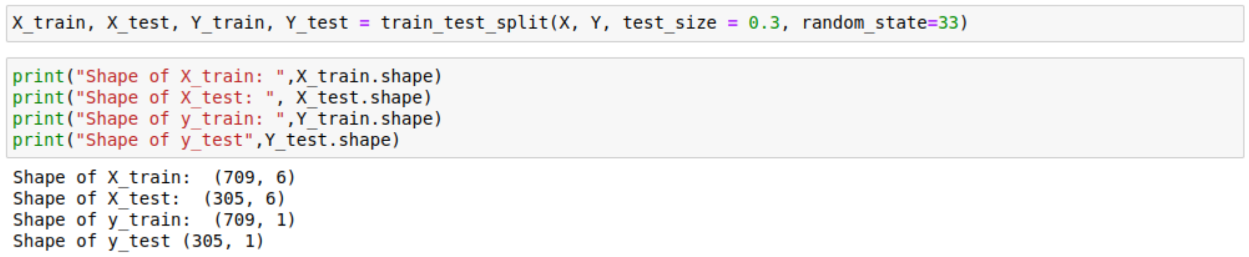
To normalize X, we use the method discussed during the class.

$$x' = (x - x_{min})/(x_{max} - x_{min})$$

This method of normalization is also known as **Scaling to a range** and is useful when we know the Maximum and Minimum values for the attributes.

Now, all of our attributes have values between 0 and 1.

To form the Training and Testing sets, we use the **train_test_split()** function as shown below.



```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state=33)
```

```
print("Shape of X_train: ",X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ",Y_train.shape)
print("Shape of y_test",Y_test.shape)
```

```
Shape of X_train:  (709, 6)
Shape of X_test:  (305, 6)
Shape of y_train:  (709, 1)
Shape of y_test (305, 1)
```

Now can see, the dataset has been divided into 2 sets, with 709 records in the Training set, and the rest 305 records in the Testing set.

We now move onto the classification algorithms.

Classification Algorithms

A classification algorithm, in general, is a function that weighs the input features so that the output separates one class into positive values and the other into negative values.

The classification in this project is a type of multi-class classification since there are more than two classes possible (3.)

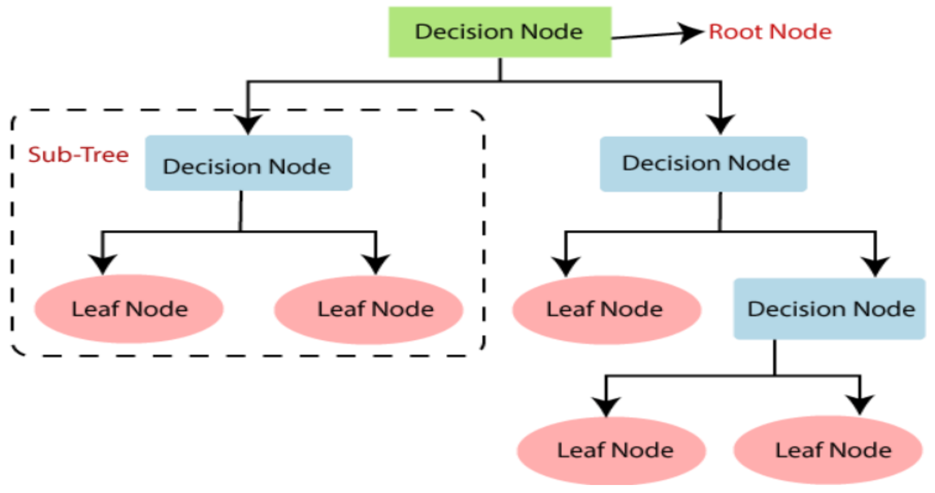
We have tried four types of Classification Algorithms, and based on the accuracy results we will choose the final model.

The models are implemented using the sk_learn library from python.

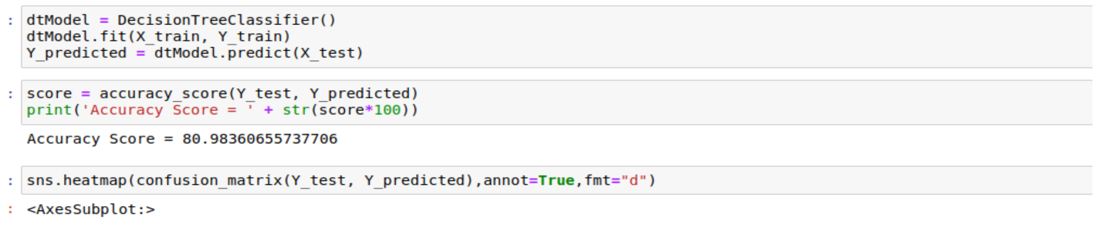
The first one is the Decision Tree Classifier.

Decision Tree Classifier

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.



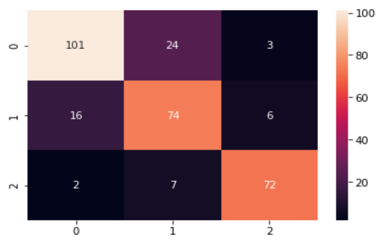
The Decision Tree Classifier implementation is done as follows.



```
: dtModel = DecisionTreeClassifier()
  dtModel.fit(X_train, Y_train)
  Y_predicted = dtModel.predict(X_test)
```

```
: score = accuracy_score(Y_test, Y_predicted)
  print('Accuracy Score = ' + str(score*100))

  Accuracy Score = 80.98360655737706
```

```
: sns.heatmap(confusion_matrix(Y_test, Y_predicted),annot=True,fmt="d")
: <AxesSubplot:>
```



	0	1	2
0	101	24	3
1	16	74	6
2	2	7	72

As we can see, it performs at an accuracy of around 81. The confusion matrix is also created to further infer the correctly and incorrectly identified classes.

Gaussian Naive Bayes

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. We have explored the idea behind Gaussian Naive Bayes along with an example.

Naive Bayes are a group of supervised machine learning classification algorithms based on the Bayes theorem. It is a simple classification technique, but has high functionality. They find use when the dimensionality of the inputs is high. Complex classification problems can also be implemented by using Naive Bayes Classifier.

The Naïve Bayes classifier assumes that the value of one feature is independent of the value of any other feature. Naïve Bayes classifiers need training data to estimate the parameters required for classification. Due to simple design and application, Naïve Bayes classifiers can be suitable in many real-life scenarios.

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

In the above formulae, sigma and mu is the variance and mean of the continuous variable X computed for a given class c of Y.

The Implementation of Gaussian Naive Bayes and its Confusion Matrix is as given below.

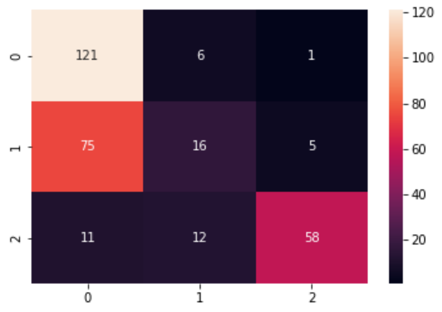
```
nbModel = GaussianNB()
nbModel.fit(X_train, Y_train.values.ravel())
Y_predicted = nbModel.predict(X_test)
```

```
score = accuracy_score(Y_test, Y_predicted)
print('Accuracy Score = ' + str(score*100))
```

```
Accuracy Score = 63.934426229508205
```

```
sns.heatmap(confusion_matrix(Y_test, Y_predicted),annot=True,fmt="d")
```

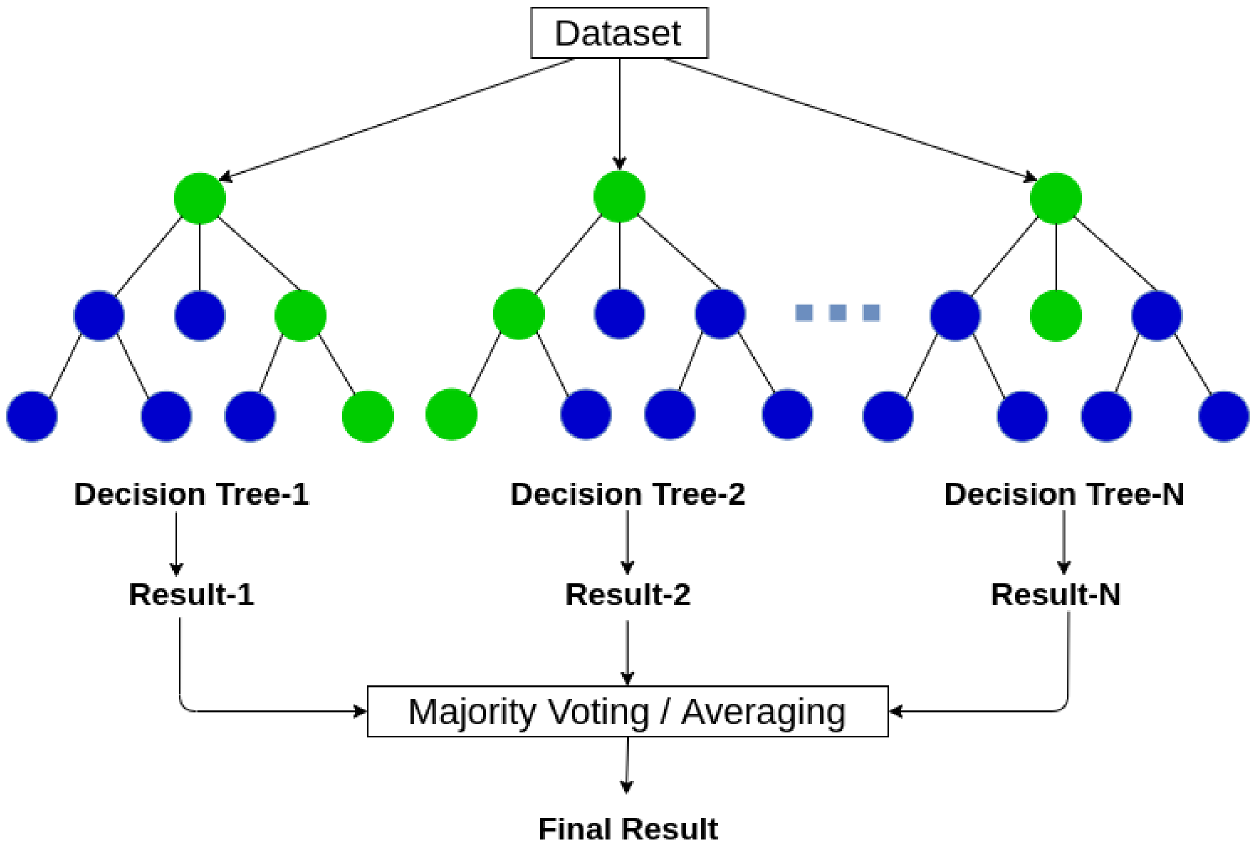
```
<AxesSubplot:>
```



As we can see, the Gaussian Naive Bayes Classifier works at an accuracy of around 64, which isn’t very good.

Random Forest Classifier

The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction.



The implementation of Random Forest and its Confusion matrix in python is done as shown below.

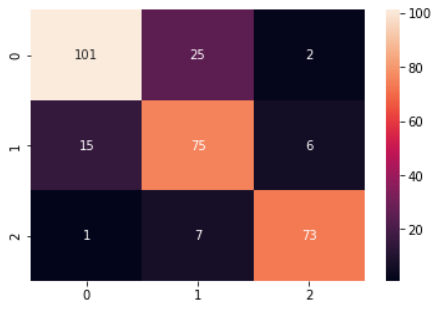
```
rfModel = RandomForestClassifier()
rfModel.fit(X_train, Y_train.values.ravel())
Y_predicted = rfModel.predict(X_test)
```

```
score = accuracy_score(Y_test, Y_predicted)
print('Accuracy Score = ' + str(score*100))
```

```
Accuracy Score = 81.63934426229508
```

```
sns.heatmap(confusion_matrix(Y_test, Y_predicted),annot=True,fmt="d")
```

```
<AxesSubplot:>
```

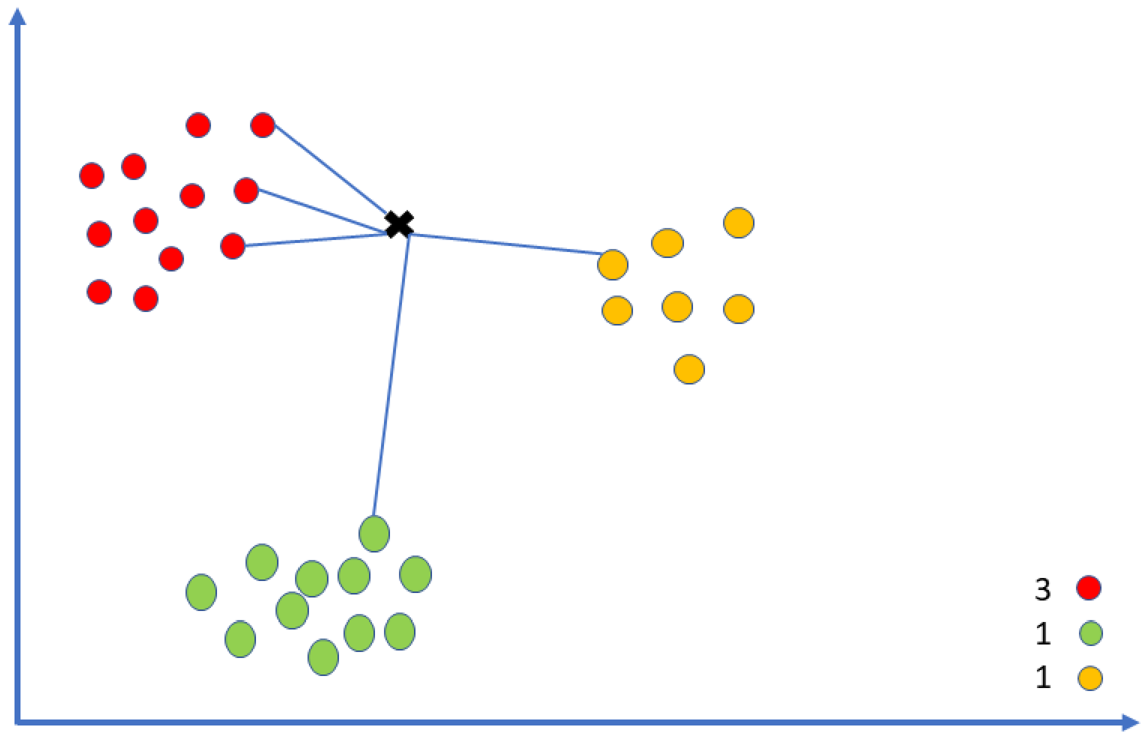


As we can see, the Random Forest classifier works at an accuracy score of around 82 here, which is moderately good.

K Nearest Neighbours Classifier

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

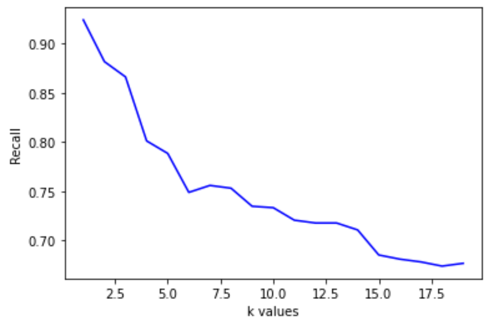
For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used. While this is technically considered “plurality voting”, the term, “majority vote” is more commonly used in literature.



First, to choose a value of K, we try the classifier for the K in range of 1 to 20, and then look at the Recall values.

```
knn = KNeighborsClassifier()
k_range = list(range(1,20))
k_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, Y_train.values.ravel())
    k_scores.append(knn.score(X_train, Y_train))

print(np.round(k_scores, 4))

plt.plot(k_range, k_scores, color="Blue")
plt.xlabel('k values')
plt.ylabel('Recall')
plt.show()
```

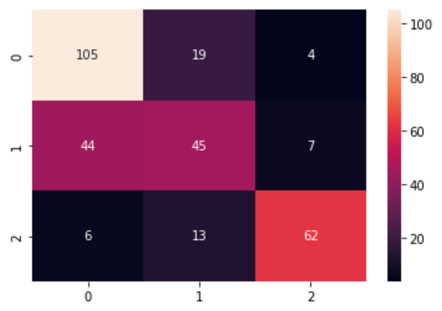
```
[0.9238 0.8815 0.866  0.8011 0.7884 0.7489 0.756  0.7532 0.7348 0.7334
 0.7207 0.7179 0.7179 0.7109 0.6855 0.6812 0.6784 0.6742 0.677 ]
```



We need to choose a value for K. K=1 is not plausible, even though its recall is maximum. The value of k needs not be very large or very small in such a way that recall is not compromised. Hence, we choose the value of K = 10

```
classifier = KNeighborsClassifier(n_neighbors=10)
classifier.fit(X_train, Y_train.values.ravel())

Y_predicted = classifier.predict(X_test)

print ("Accuracy score: ", accuracy_score(Y_test,Y_predicted), '\n')
sns.heatmap(confusion_matrix(Y_test, Y_predicted),annot=True,fmt="d")
```

```
Accuracy score:  0.6950819672131148


<AxesSubplot:>
```

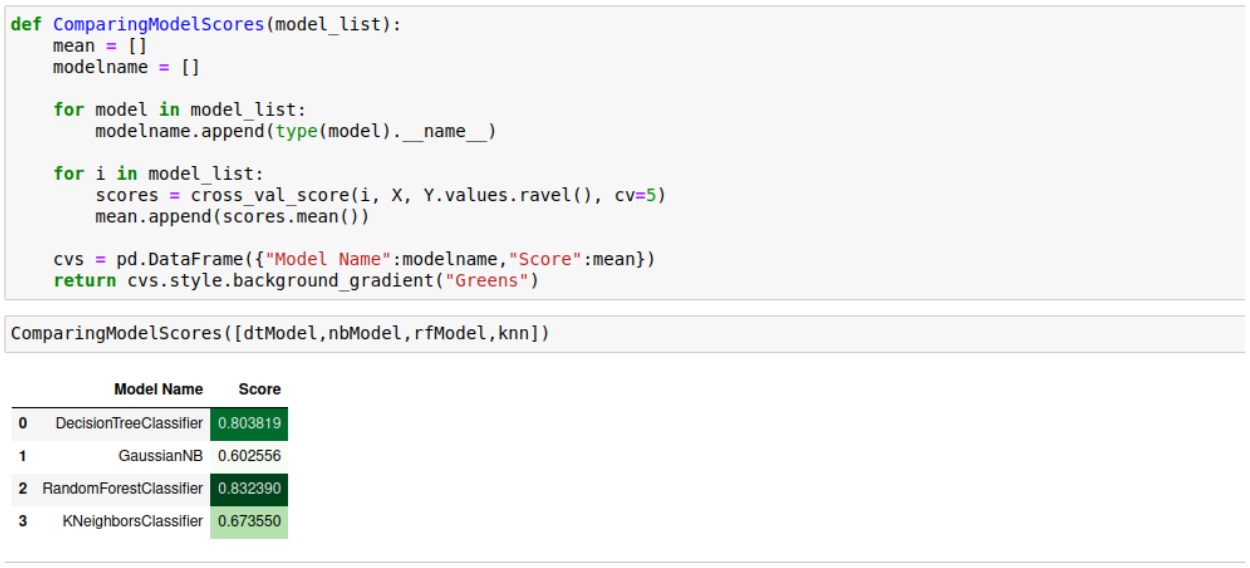


The accuracy score of KNN classifier is around 70, and its confusion matrix also shows a lot of false positives for category 1, almost as much as its True Positives!

Hence, it is not plausible to use KNN here.

Conclusion

We now compare the different classifiers and their scores to choose a final model.



```python
def ComparingModelScores(model_list):
    mean = []
    modelname = []

    for model in model_list:
        modelname.append(type(model).__name__)

    for i in model_list:
        scores = cross_val_score(i, X, Y.values.ravel(), cv=5)
        mean.append(scores.mean())

    cvs = pd.DataFrame({"Model Name":modelname,"Score":mean})
    return cvs.style.background_gradient("Greens")
```

```python
ComparingModelScores([dtModel,nbModel,rfModel,knn])
```

	Model Name	Score
0	DecisionTreeClassifier	0.803819
1	GaussianNB	0.602556
2	RandomForestClassifier	0.832390
3	KNeighborsClassifier	0.673550

As we can see, the scores of both Decision Tree Classifier and Random Forest Classifier are the highest and around the same, with Random Forest in the lead. Also, in general, Random Forest Classifiers are more robust in comparison to Decision Tree Classifiers.

Hence, in conclusion, we should use the Random Forest Classifiers for this classification problem.

References

- https://archive.ics.uci.edu/ml/datasets/Maternal+Health+Risk+Data+Set
- https://matplotlib.org/3.1.1/tutorials/index.html
- https://seaborn.pydata.org/introduction.html

GitHub Link:

https://github.com/Shreyashsh/Predicting-Maternal-Health-Risk-Levels