COVID-19 CLASSIFICATION

BY

Nahla Hatem Mohamed 20399128

Supervisor

Marwa Elsayed

CONTENTS

1. ALGORITHMS

1.1 Data Loading

1.1.1 Load data

Load data from CSV file to the work as a Data Frame using pandas.

Import Data

```
[6] data = pd.read_csv("data.csv")
    data.columns = ['Index', 'location', 'country', 'gender', 'age', 'vis_wuhan',
           'from_wuhan', 'symptom1', 'symptom2', 'symptom3', 'symptom4',
           'symptom5', 'symptom6', 'diff_sym_hos', 'result']
    data.head(5)
```

	Index	location	country	gender	age	vis_wuhan	from_wuhan	symptom1	symptom2	symptom3	symptom4	symptom5	symptom6	diff_sym_hos	result
0	0	104	8	1	66.0	1	0	14	31	19	12	3	1	8	1
1	1	101	8	0	56.0	0	1	14	31	19	12	3	1	0	0
2	2	137	8	1	46.0	0	1	14	31	19	12	3	1	13	0
3	3	116	8	0	60.0	1	0	14	31	19	12	3	1	0	0
4	4	116	8	1	58.0	0	0	14	31	19	12	3	1	0	0

1.1.2 View more information about the data

```
[8] data.describe()
```

	Index	location	country	gender	age	vis_wuhan	from_wuhan	symptom1	symptom2	symptom3	symptom4	symptom5	symp
count	863.000000	863.000000	863.000000	863.000000	863.000000	863.000000	863.000000	863.00000	863.000000	863.000000	863.000000	863.000000	863.00
mean	431.000000	76.645423	16.995365	0.849363	49.400000	0.181924	0.107764	12.13905	28.002317	18.298957	11.840093	2.993048	0.99
std	249.270937	39.200264	7.809951	0.726062	15.079203	0.386005	0.310261	3.99787	7.473231	2.864064	1.183771	0.127251	0.03
min	0.000000	0.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00
25%	215.500000	45.000000	11.000000	0.000000	40.000000	0.000000	0.000000	14.00000	31.000000	19.000000	12.000000	3.000000	1.00
50%	431.000000	87.000000	18.000000	1.000000	49.400000	0.000000	0.000000	14.00000	31.000000	19.000000	12.000000	3.000000	1.00
75%	646.500000	110.000000	24.000000	1.000000	57.000000	0.000000	0.000000	14.00000	31.000000	19.000000	12.000000	3.000000	1.00
max	862.000000	138.000000	33.000000	2.000000	96.000000	1.000000	1.000000	24.00000	31.000000	19.000000	12.000000	3.000000	1.00

```
[9]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 863 entries, 0 to 862
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   location      863 non-null    int64
 1   country       863 non-null    int64
 2   gender        863 non-null    int64
 3   age           863 non-null    float64
 4   vis_wuhan     863 non-null    int64
 5   from_wuhan    863 non-null    int64
 6   symptom1      863 non-null    int64
 7   symptom2      863 non-null    int64
 8   symptom3      863 non-null    int64
 9   symptom4      863 non-null    int64
 10  symptom5      863 non-null    int64
 11  symptom6      863 non-null    int64
 12  diff_sym_hos  863 non-null    int64
 13  result        863 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 94.5 KB
```

1.2 Visualize Data

In these steps, we need to view more plots of our data to figure out more information about the data set we work with

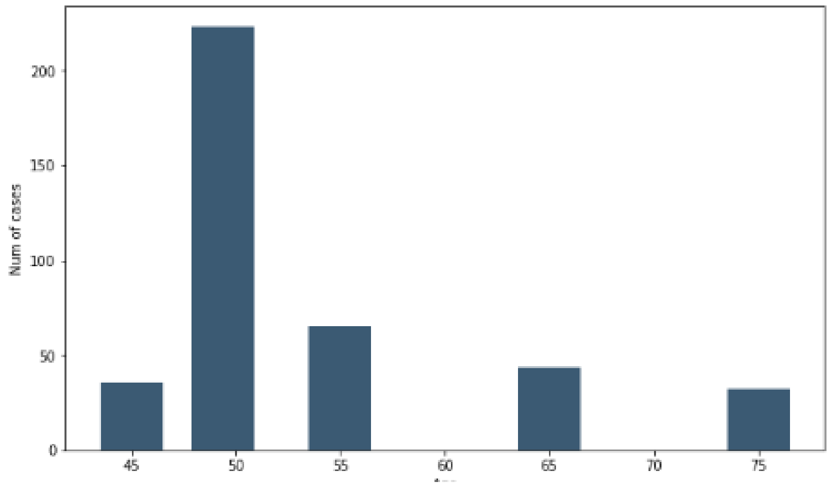
1.2.1 View the most frequent age to get an infection

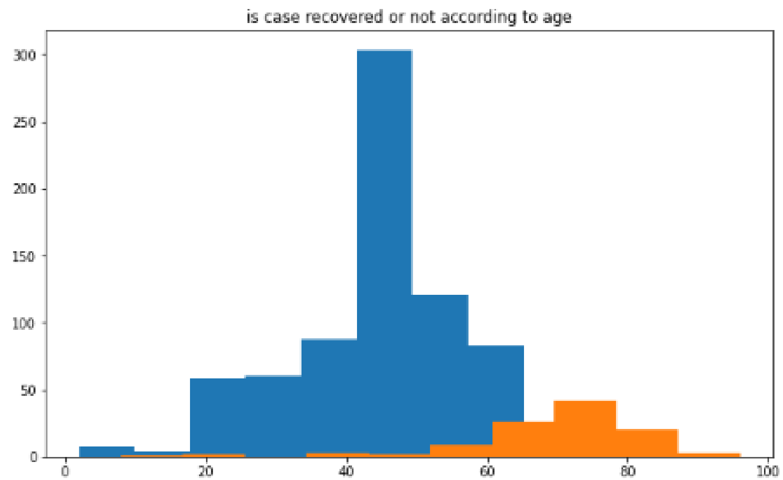
From the visualization, we can notice that the most cases in ages between **40 and 60** but in cases above **50** the greatest possibility of death

what is the most frequent age?

```
df_Age = df['age'].value_counts().reset_index()
df_Age[:5]
```

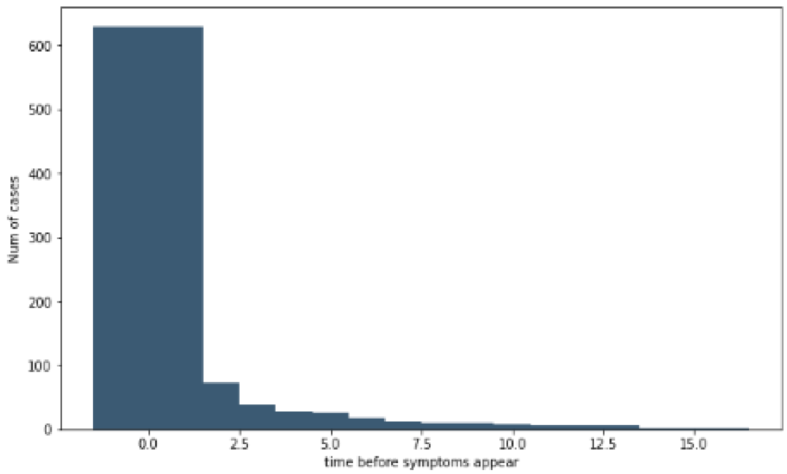
	index	age
0	49.4	223
1	55.0	65
2	65.0	44
3	45.0	36
4	75.0	32



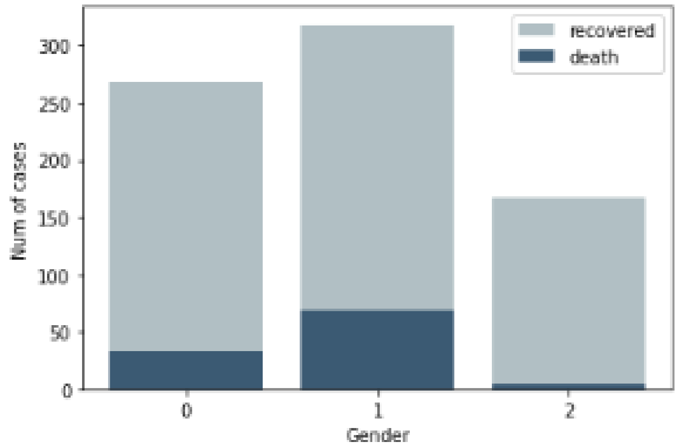


1.2.2 View which time is frequent before symptoms appear

From the chart, we can find that most cases in our data set didn't take time before the symptoms appear.

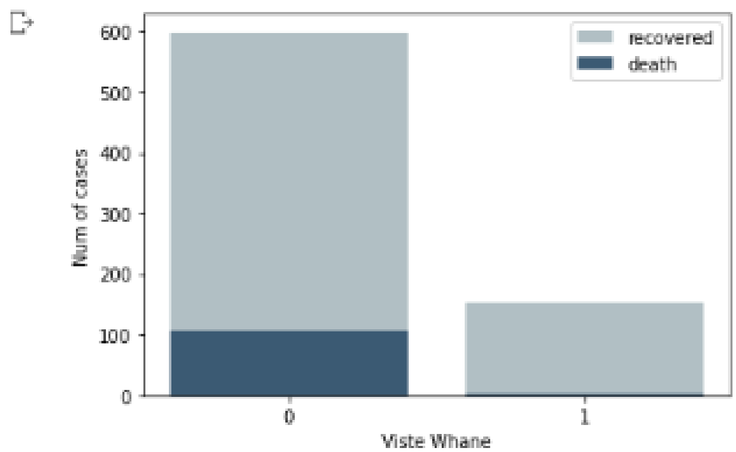


1.2.3 Visualize the distribution of deaths and recovery according to gender



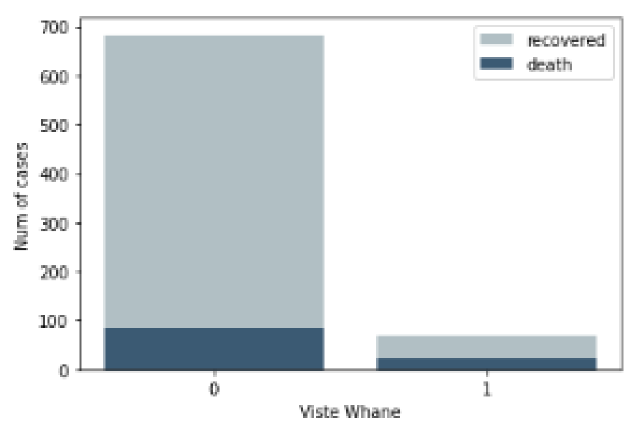
1.2.4 Number of cases Vs. visit chain or not

From this view, we can find that most cases didn't visit the chain



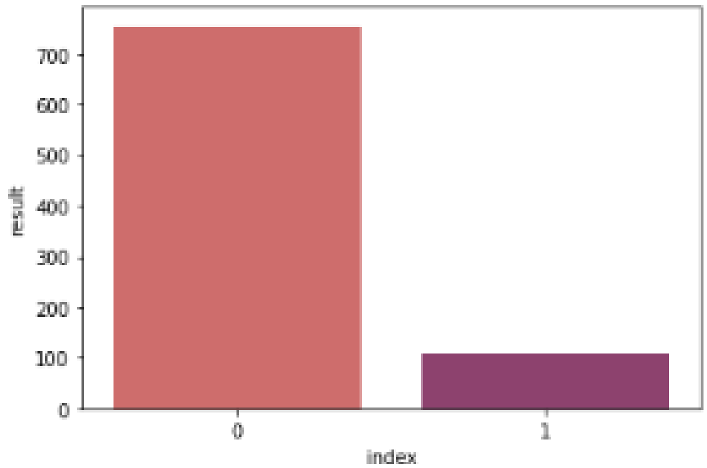
1.2.5 Number of cases Vs. from a chain or not

From this view, we can find that most cases do not from a chain



1.2.6 View if the data is balanced or not

We can figure that the data is not balanced as the number of Recovered is higher than death cases.



1.2.7 View correlation between all features

1.3 Preprocessing Data

1.3.1 Normalization

From the distribution of data, we find that we need to normalize the data

	location	country	gender	age	vis_wuhan	from_wuhan	symptom1	symptom2	symptom3	symptom4	symptom5	symptom6	diff_sym_hos	result
0	0.753623	0.242424	0.5	0.680851	1.0	0.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.533333	1.0
1	0.731884	0.242424	0.0	0.574468	0.0	1.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.000000	0.0
2	0.992754	0.242424	0.5	0.468085	0.0	1.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.866667	0.0
3	0.840580	0.242424	0.0	0.617021	1.0	0.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.000000	0.0
4	0.840580	0.242424	0.5	0.595745	0.0	0.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.000000	0.0
5	0.166667	0.242424	0.0	0.446809	0.0	1.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.000000	0.0
6	0.760870	0.242424	0.5	0.340426	0.0	1.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.000000	0.0
7	0.094203	0.242424	0.5	0.372340	1.0	0.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.400000	0.0
8	0.094203	0.242424	0.5	0.393617	1.0	0.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.333333	0.0
9	0.094203	0.242424	0.5	0.574468	1.0	0.0	0.583333	1.0	1.0	1.0	1.0	1.0	0.266667	0.0

1.3.2 Split data

Now we need to split our data set into train and test but from our visualization, we find that data is imbalanced so we will need to split data with stratify method to be sure that the percentage of each class in the train & test data set is equally

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=23 , stratify=y)
```

Check percentage of each class In train & test data

```
[34] print("Death Percentage in train data : ", y_train[y_train == 1 ].count()/y_train.count())
     print("Death Percentage in test data : ", y_test[y_test == 1 ].count()/y_test.count())

Death Percentage in train data :  0.125
Death Percentage in test data :  0.12716763005780346
```

```
[35] print("Recovered Percentage in train data : ", y_train[y_train == 0 ].count()/y_train.count())
     print("Recovered Percentage in test data : ", y_test[y_test == 0 ].count()/y_test.count())

Recovered Percentage in train data :  0.875
Recovered Percentage in test data :  0.8728323699421965
```

1.4 Create ML Model

Now we will start to create our model we will build four different models and then train them at our data set to find what is the most suitable model for our case also we will use a grid search technique to tune our hyperparameters

1.4.1 K Neighbors Classifier

Create Model

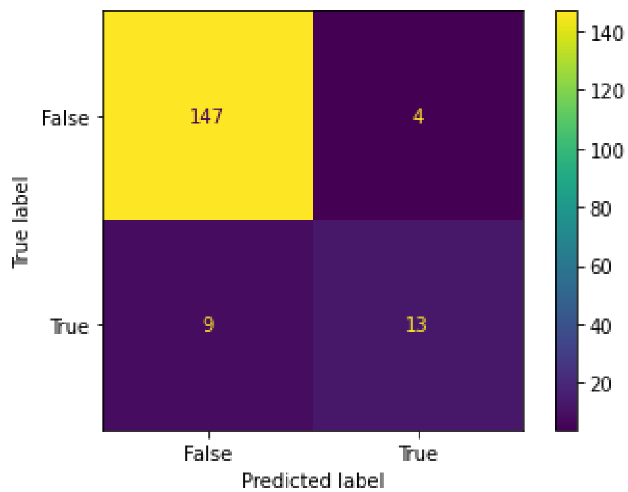
We create a KNN model and we used a grid search technique to tune our hyperparameters after the grid search we find that the best hyperparameters are (leaf_size=1, n_jobs=-1, n_neighbors=4, p=1,weights='distance')

The Model Results

It has been discovered that the accuracy of the KNN model is poor due to declining recall accuracy.

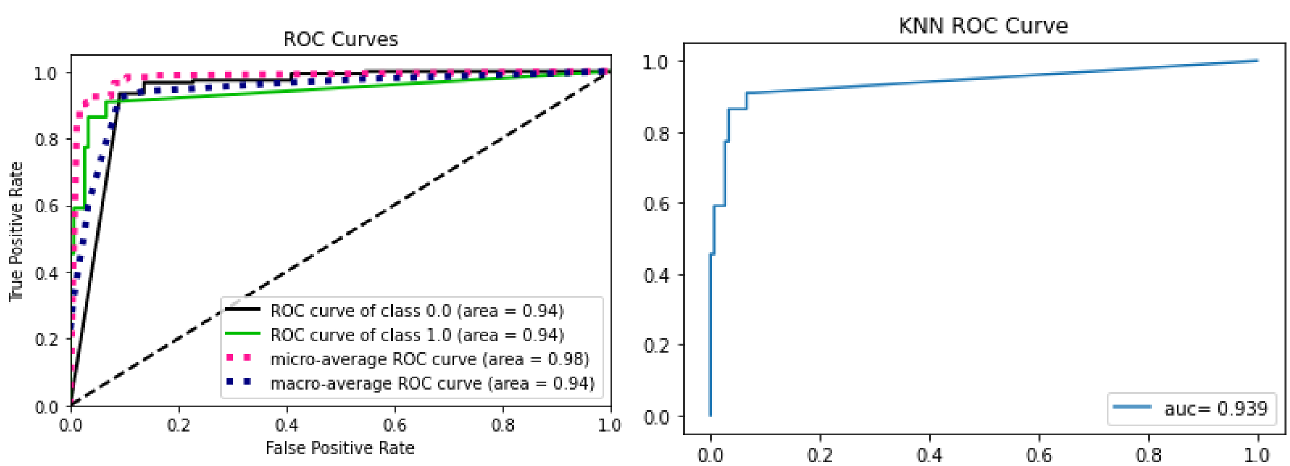
	precision	recall	f1-score	support
0.0	0.94	0.97	0.96	151
1.0	0.76	0.59	0.67	22
accuracy			0.92	173
macro avg	0.85	0.78	0.81	173
weighted avg	0.92	0.92	0.92	173

```
Test Accuracy : 0.9248554913294798
Precision: 0.765
Recall: 0.591
F-Measure: 0.667
```



Calculate AUC & plot ROC Curve

The ROC curve indicates that the AUC accuracy is averagely good.



1.4.2 Logistic Regression

Create Model

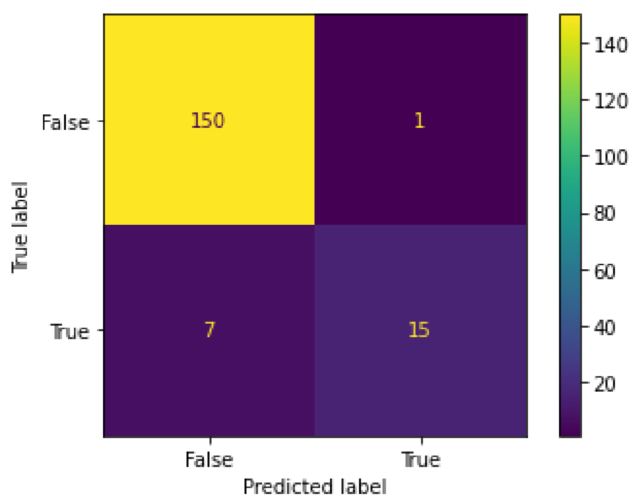
We create a Logistic Regression model and we used a grid search technique to tune our hyperparameters after the grid search we find that the best hyperparameters are **(C=100000, max_iter=10000, random_state=0, solver='liblinear')**

The Model Results

We find that the precision accuracy of the Logistic regression model is acceptable, but the recall accuracy is still poor.

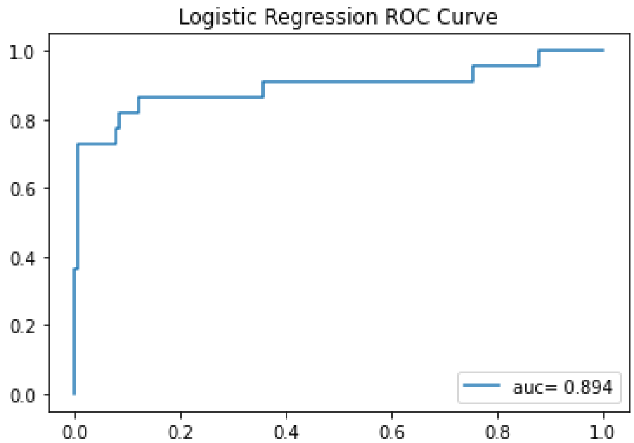
	precision	recall	f1-score	support
0.0	0.96	0.99	0.97	151
1.0	0.94	0.68	0.79	22
accuracy			0.95	173
macro avg	0.95	0.84	0.88	173
weighted avg	0.95	0.95	0.95	173

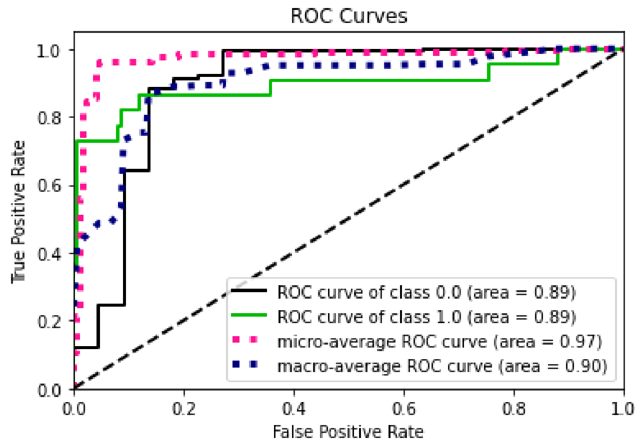
```
Test Accuracy : 0.953757225433526
Precision: 0.938
Recall: 0.682
F-Measure: 0.789
```



Calculate AUC & plot ROC Curve

The ROC curve indicates that the AUC accuracy is averagely good.





1.4.3 Naive Bayes

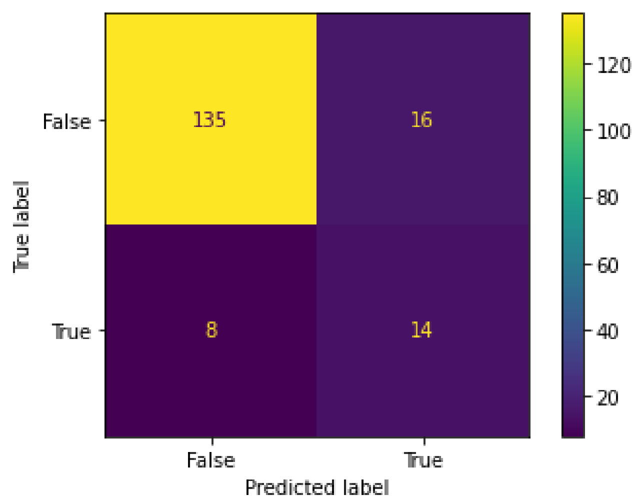
Create Model

We create the Naive Bayes model and we used a grid search technique to tune our hyperparameters after the grid search we find that the best hyperparameters are **(var_smoothing=0.03107)**

The Model Results

It has been discovered that the recall and precision accuracy of the Naive Bayes model is poor.

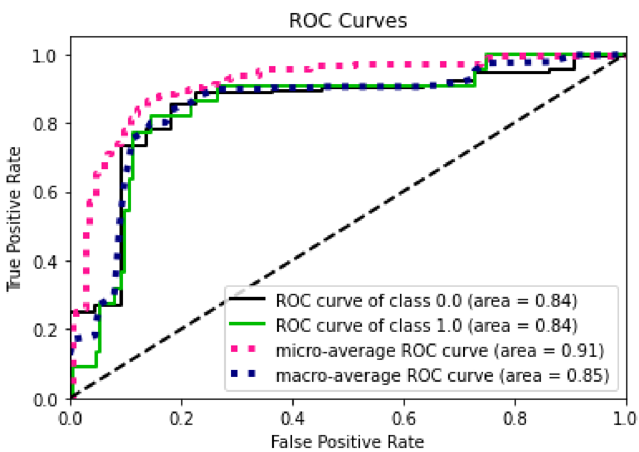
```
Test Accuracy : 0.861271676300578
Precision: 0.467
Recall: 0.636
F-Measure: 0.538
```

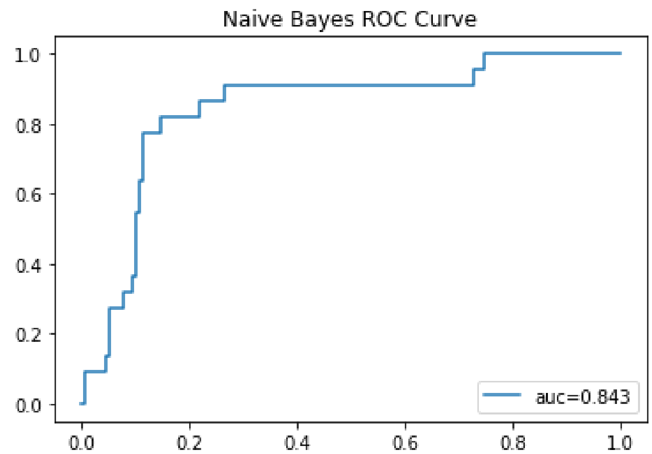


	precision	recall	f1-score	support
0.0	0.94	0.89	0.92	151
1.0	0.47	0.64	0.54	22
accuracy			0.86	173
macro avg	0.71	0.77	0.73	173
weighted avg	0.88	0.86	0.87	173

Calculate AUC & plot ROC Curve

The ROC curve indicates that the AUC accuracy is averagely good.





1.4.4 Decision Tree Classifier

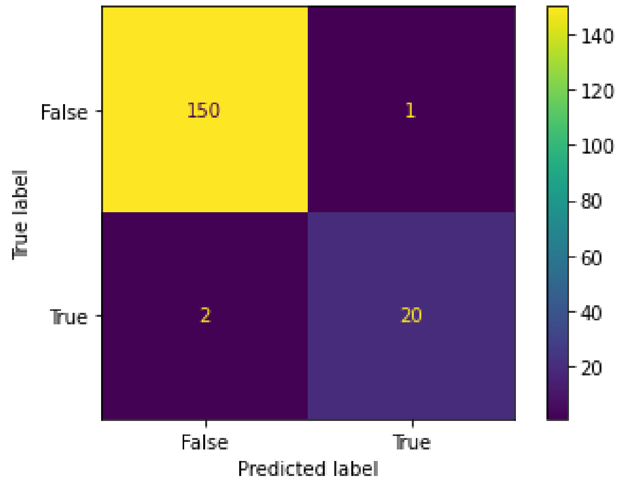
Create Model

We create a Decision Tree model and we used a grid search technique to tune our hyperparameters after the grid search we find that the best hyperparameters are **(max_depth=12, Criterion: Gini)**

The Model Results

It has been discovered that both the recall and precision accuracy of the Decision Tree model are pretty good.

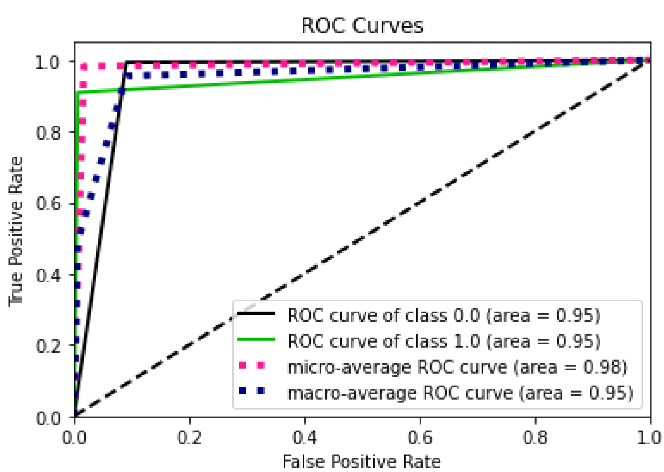
```
Test Accuracy : 0.9826589595375722
Precision: 0.952
Recall: 0.909
F-Measure: 0.930
```

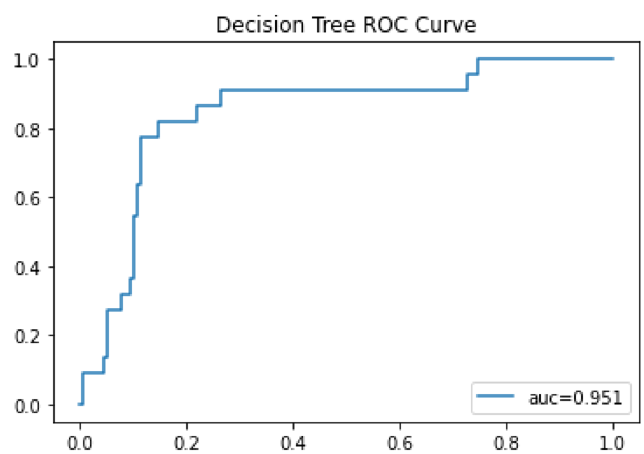



```
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       151
         1.0       0.95      0.91      0.93        22

    accuracy                           0.98       173
   macro avg       0.97      0.95      0.96       173
weighted avg       0.98      0.98      0.98       173
```

Calculate AUC & plot ROC Curve

The ROC curve indicates that the AUC accuracy is good.





1.4.5 Support Vector Machines

Create Model

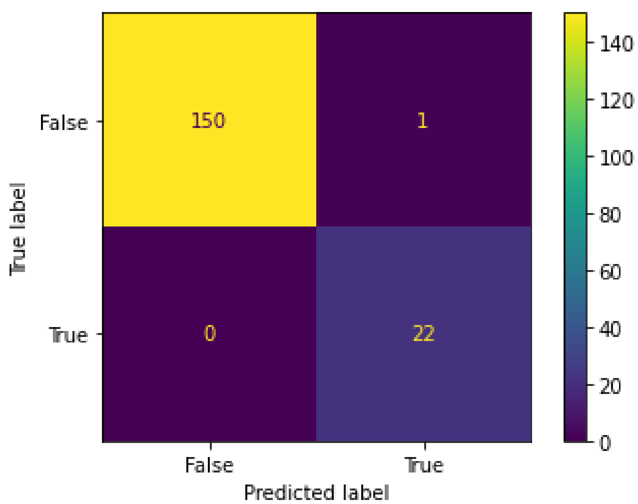
We create a Support vector machine model and used a grid search technique to tune our hyperparameters after the grid search we find that the best hyperparameters are (C=100, gamma=1, 'kernel': 'rbf')

The Model Results

It has been discovered that both the recall and precision accuracy of the Support Vector Machine model are pretty good.

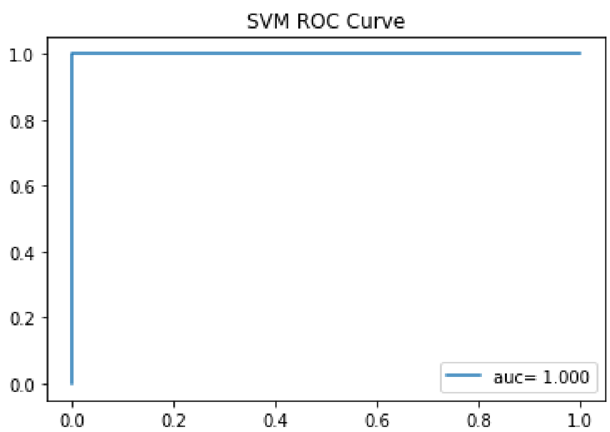
	precision	recall	f1-score	support
0.0	1.00	0.99	1.00	151
1.0	0.96	1.00	0.98	22
accuracy			0.99	173
macro avg	0.98	1.00	0.99	173
weighted avg	0.99	0.99	0.99	173

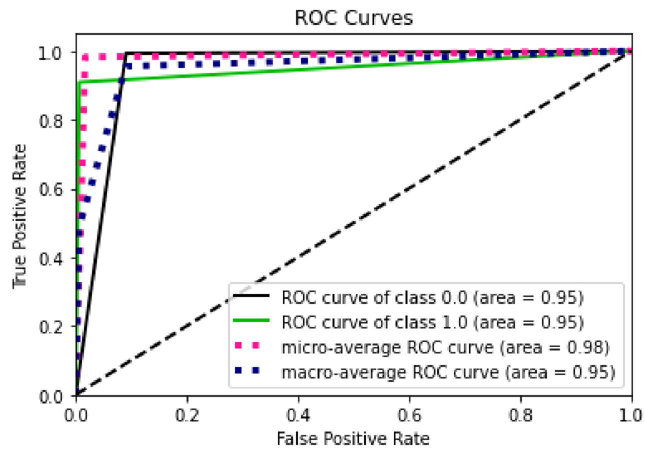
```
Test Accuracy : 0.9942196531791907
Precision: 0.957
Recall: 1.000
F-Measure: 0.978
```



Calculate AUC & plot ROC Curve

The ROC curve indicates that the AUC accuracy is pretty good.





2. Visualize the Result of all models

In the end, when we compare all the models we find that the best model for our data set is the **SVM model** because it has the highest precision and recall accuracy