CLR204 Assessment 2 - *COVID-19 Problem Analysis*

David Lawler - A00075945

Imports

```
import numpy as np # Numerical Python - used for arrays, linear algebra, fourier transform, matrices
import pandas as pd # Panel Data - used for data manipulation and analysis
import datetime as dt # Used for manipulating dates and times
import sklearn # Machine learning library
from scipy import stats # scipy is a scientific library dependant on numpy - stats is a statistic function sub-package
from sklearn import preprocessing # Common utility functions and transformer classes to change raw feature vectors
from sklearn.model_selection import GridSearchCV # Automates tuning of hyperparameters
from sklearn.ensemble import RandomForestClassifier # Meta Estimators
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split # Model validation simulator
from sklearn.metrics import recall_score as rs # Performance Metrics
from sklearn.metrics import precision_score as ps
from sklearn.metrics import f1_score as fs
from sklearn.metrics import log_loss # Lower log loss = better predictions, based off likelyhood function

encoder = preprocessing.LabelEncoder() # Transformer used to encode target values
```

Data Preprocessing

```
data = pd.read_csv('data.csv') # Retrieves data from data.csv
data = data.drop('id',axis=1) # Drops id column
data = data.fillna(np.nan,axis=0) # Fills NA/NAN values
data['location'] = encoder.fit_transform(data['location'].astype(str)) # Label encoding
```

```
data['location'] = encoder.fit_transform(data['location'].astype(str)) # Label encoding
data['country'] = encoder.fit_transform(data['country'].astype(str))
data['gender'] = encoder.fit_transform(data['gender'].astype(str))
data['symptom1'] = encoder.fit_transform(data['symptom1'].astype(str))
data['symptom2'] = encoder.fit_transform(data['symptom2'].astype(str))
data['symptom3'] = encoder.fit_transform(data['symptom3'].astype(str))
data['symptom4'] = encoder.fit_transform(data['symptom4'].astype(str))
data['symptom5'] = encoder.fit_transform(data['symptom5'].astype(str))
data['symptom6'] = encoder.fit_transform(data['symptom6'].astype(str))


data['sym_on'] = pd.to_datetime(data['sym_on']) # Convert sym-on to datetime object
data['hosp_vis'] = pd.to_datetime(data['hosp_vis'])
data['sym_on']= data['sym_on'].map(dt.datetime.toordinal) # Datetime columns
data['hosp_vis']= data['hosp_vis'].map(dt.datetime.toordinal)
data['diff_sym_hos']= data['hosp_vis'] - data['sym_on'] # New Feature


data = data.drop(['sym_on','hosp_vis'],axis=1) # Drops two indirectly used variables


print(data.dtypes)
```

```
location          int64
country           int64
gender            int64
age             float64
vis_wuhan         int64
from_wuhan      float64
death             int64
recov             int64
symptom1          int64
symptom2          int64
symptom3          int64
symptom4          int64
symptom5          int64
symptom6          int64
diff_sym_hos      int64
dtype: object
```

The dataset used in this study has been compiled from various sources including the World Health Organization and John Hopkins University. Accessed from Kaggle as “Novel Corona Virus 2019 Dataset” and has been pre-processed further to meet the needs of the study - *COVID-19 Patient Health Prediction Using Boosted Random Forest Algorithm*(Iwendi et al,. 2019).

The feature of the data are presented below.

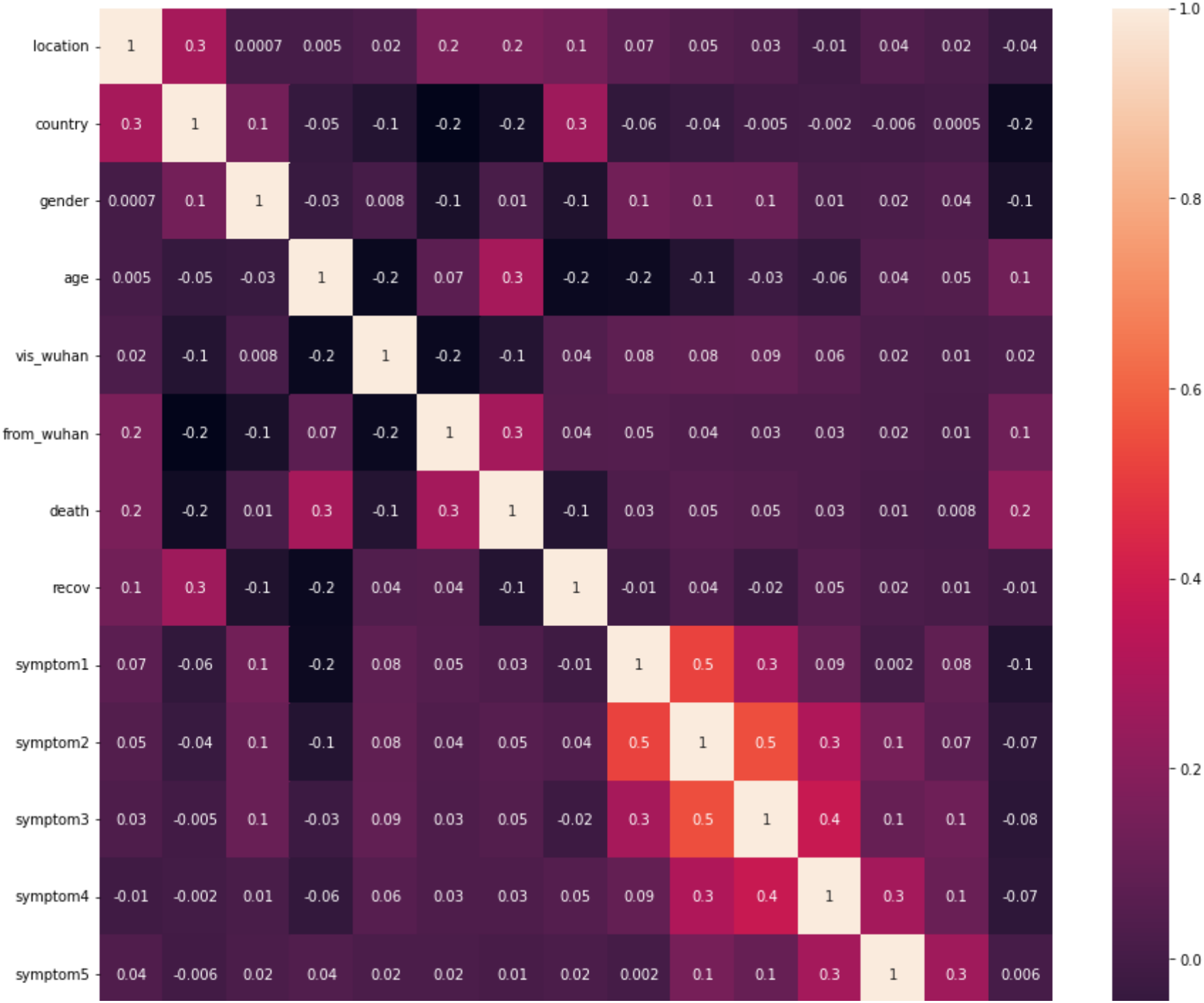
Column	Description	Values (for categorical variables)	Type
id	Patient Id	NA	Numeric
location	The location where the patient belongs to	Multiple cities located throughout the world	String, Categorical
country	Patient’s native country	Multiple countries	String, Categorical
gender	Patient’s gender	Male, Female	String, Categorical
age	Patient’s age	NA	Numeric
sym_on	The date patient started noticing the symptoms	NA	Date
hosp_vis	Date when the patient visited the hospital	NA	Date
vis_wuhan	Whether the patient visited Wuhan, China	Yes (1), No (0)	Numeric, Categorical
from_wuhan	Whether the patient belonged to Wuhan, China	Yes (1), No (0)	Numeric, Categorical
death	Whether the patient passed away due to	Yes (1), No (0)	Numeric, Categorical

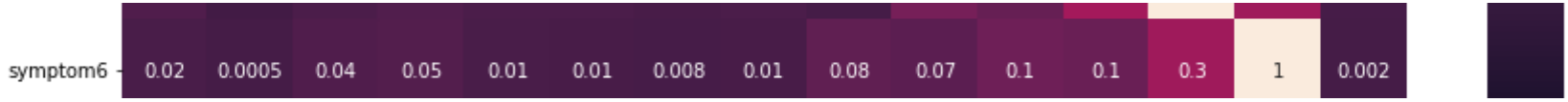
	COVID-19		
Recov	Whether the patient recovered	Yes (1), No (0)	Numeric, Categorical
symptom1, symptom2, symptom3, symptom4, symptom5, symptom6	Symptoms noticed by the patients	Multiple symptoms noticed by the patients	String, Categorical

Model Reproduction

Visualisation

```
# Figure 2. Heat map of Correlation between data features
import seaborn as sb # Used for making statistical graphics
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(15,15))         # Sample figsize in inches
dataplot=sb.heatmap(data.corr(),annot=True, fmt='.01g', ax=ax)
```


location 1 0.3 0.0007 0.005 0.02 0.2 0.2 0.1 0.07 0.05 0.03 -0.01 0.04 0.02 -0.04
country 0.3 1 0.1 -0.05 -0.1 -0.2 -0.2 0.3 -0.06 -0.04 -0.005 -0.002 -0.006 0.0005 -0.2
gender 0.0007 0.1 1 -0.03 0.008 -0.1 0.01 -0.1 0.1 0.1 0.1 0.01 0.02 0.04 -0.1
age 0.005 -0.05 -0.03 1 -0.2 0.07 0.3 -0.2 -0.2 -0.1 -0.03 -0.06 0.04 0.05 0.1
vis_wuhan 0.02 -0.1 0.008 -0.2 1 -0.2 -0.1 0.04 0.08 0.08 0.09 0.06 0.02 0.01 0.02
from_wuhan 0.2 -0.2 -0.1 0.07 -0.2 1 0.3 0.04 0.05 0.04 0.03 0.03 0.02 0.01 0.1
death 0.2 -0.2 0.01 0.3 -0.1 0.3 1 -0.1 0.03 0.05 0.05 0.03 0.01 0.008 0.2
recov 0.1 0.3 -0.1 -0.2 0.04 0.04 -0.1 1 -0.01 0.04 -0.02 0.05 0.02 0.01 -0.01
symptom1 0.07 -0.06 0.1 -0.2 0.08 0.05 0.03 -0.01 1 0.5 0.3 0.09 0.002 0.08 -0.1
symptom2 0.05 -0.04 0.1 -0.1 0.08 0.04 0.05 0.04 0.5 1 0.5 0.3 0.1 0.07 -0.07
symptom3 0.03 -0.005 0.1 -0.03 0.09 0.03 0.05 -0.02 0.3 0.5 1 0.4 0.1 0.1 -0.08
symptom4 -0.01 -0.002 0.01 -0.06 0.06 0.03 0.03 0.05 0.09 0.3 0.4 1 0.3 0.1 -0.07
symptom5 0.04 -0.006 0.02 0.04 0.02 0.02 0.01 0.02 0.002 0.1 0.1 0.3 1 0.3 0.006
1.0
0.8
0.6
0.4
0.2
0.0

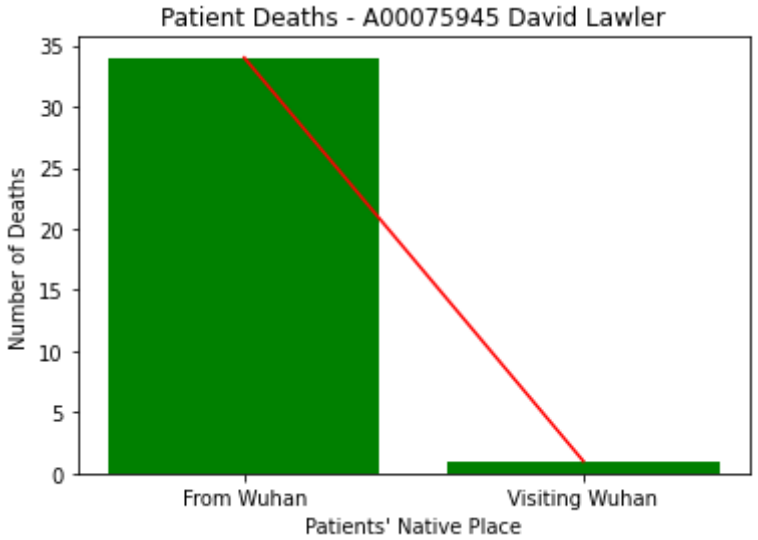



```
import matplotlib.pyplot as plt

def counter2(colname1,colname2): # for loop for 2 column logic
  colname1 = pd.Series(colname1)
  colname2 = pd.Series(colname2)
  count1 = 0
  for i in range(min([colname1.size,colname2.size])):
    if(colname1[i]==1 and colname2[i]==1):
      count1 = count1+1
  return count1

def counter1(colname): # for loop for 1 column logic
  colname1 = pd.Series(colname)
  count = 0
  for i in range(colname1.size):
    if(colname1[i]==1):
      count = count+1
  return count


fwuh = counter1(data['from_wuhan'])
vwuh = counter1(data['vis_wuhan'])


print(counter1(data['death'])) # Nunber of deaths
print(counter2(data['from_wuhan'],data['death'])) # Number of deaths people from Wuhan
print(counter2(data['vis_wuhan'],data['death'])) # Number of deaths people visiting Wuhan
```

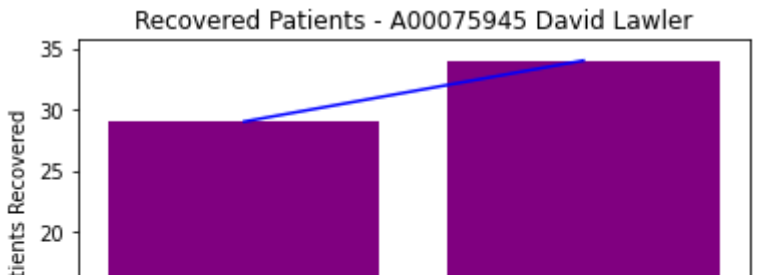
```
63
34
1
```

```
import matplotlib.pyplot as plt
```

```
plt.bar(['From Wuhan','Visiting Wuhan'],[counter2(data['death'],data['from_wuhan']),counter2(data['death'],data['vis_wuhan'])],color=
plt.title('Patient Deaths - A00075945 David Lawler')
plt.xlabel('Patients\' Native Place')
plt.ylabel('Number of Deaths')
plt.plot([counter2(data['death'],data['from_wuhan']),counter2(data['death'],data['vis_wuhan'])],color='red')
plt.show()
```




```
plt.bar(['From Wuhan','Visiting Wuhan'],[counter2(data['recov'],data['from_wuhan']),counter2(data['recov'],data['vis_wuhan'])],color=
plt.title('Recovered Patients - A00075945 David Lawler')
plt.xlabel('Patients\' Native Place')
plt.ylabel('Number of Patients Recovered')
plt.plot([counter2(data['recov'],data['from_wuhan']),counter2(data['recov'],data['vis_wuhan'])],color='blue')
plt.show()
```



Training


```
tdata = pd.read_csv('train.csv') # Reads train.csv
print(tdata.head())
```

```
   id      location country gender   age      sym_on    hosp_vis  vis_wuhan  \
0  49  Wuhan, Hubei   China   male  61.0  12/20/2019  12/27/2019          0
1  50  Wuhan, Hubei   China   male  69.0  12/30/2019    1/3/2020          0
2  51  Wuhan, Hubei   China   male  89.0         NaN         NaN          0
3  52  Wuhan, Hubei   China   male  89.0   1/13/2020   1/13/2020          0
4  53  Wuhan, Hubei   China   male  66.0   1/10/2020   1/16/2020          0

   from_wuhan  death symptom1 symptom2 symptom3  symptom4  symptom5  symptom6
0           1      1      NaN      NaN      NaN       NaN       NaN       NaN
1           1      1      NaN      NaN      NaN       NaN       NaN       NaN
2           1      1      NaN      NaN      NaN       NaN       NaN       NaN
3           1      1      NaN      NaN      NaN       NaN       NaN       NaN
4           1      1      NaN      NaN      NaN       NaN       NaN       NaN
```

```
tdata = pd.read_csv('train.csv') # Same preprocessing as with data.csv
tdata = tdata.drop('id',axis=1)
tdata = tdata.fillna(np.nan,axis=0)
tdata['age'] = tdata['age'].fillna(value=tdata['age'].mean())
tdata['location'] = encoder.fit_transform(tdata['location'].astype(str))
tdata['country'] = encoder.fit_transform(tdata['country'].astype(str))
tdata['gender'] = encoder.fit_transform(tdata['gender'].astype(str))
tdata['symptom1'] = encoder.fit_transform(tdata['symptom1'].astype(str))
```

```
tdata['symptom2'] = encoder.fit_transform(tdata['symptom2'].astype(str))
tdata['symptom3'] = encoder.fit_transform(tdata['symptom3'].astype(str))
tdata['symptom4'] = encoder.fit_transform(tdata['symptom4'].astype(str))
tdata['symptom5'] = encoder.fit_transform(tdata['symptom5'].astype(str))
tdata['symptom6'] = encoder.fit_transform(tdata['symptom6'].astype(str))

tdata['sym_on'] = pd.to_datetime(tdata['sym_on'])
tdata['hosp_vis'] = pd.to_datetime(tdata['hosp_vis'])
tdata['sym_on']= tdata['sym_on'].map(dt.datetime.toordinal)
tdata['hosp_vis']= tdata['hosp_vis'].map(dt.datetime.toordinal)
tdata['diff_sym_hos']= tdata['hosp_vis'] - tdata['sym_on']


tdata = tdata.drop(['sym_on','hosp_vis'],axis=1)
print(tdata)
```

```
     location  country  gender        age  vis_wuhan  from_wuhan  death  \
0          38        2       1  61.000000          0           1      1
1          38        2       1  69.000000          0           1      1
2          38        2       1  89.000000          0           1      1
3          38        2       1  89.000000          0           1      1
4          38        2       1  66.000000          0           1      1
..        ...      ...     ...        ...        ...         ...    ...
217         2        3       2  49.747537          0           0      0
218        22        6       2  49.747537          0           0      1
219        22        6       2  49.747537          0           0      1
220        22        6       2  49.747537          0           0      1
221        22        6       2  49.747537          0           0      1

     symptom1  symptom2  symptom3  symptom4  symptom5  symptom6  diff_sym_hos
0           9         9        11         0         0         0             7
1           9         9        11         0         0         0             4
2           9         9        11         0         0         0             0
3           9         9        11         0         0         0             0
4           9         9        11         0         0         0             6
..        ...       ...       ...       ...       ...       ...           ...
217         9         9        11         0         0         0             0
218         9         9        11         0         0         0             0
```

```
219         9         9        11         0         0         0             0
220         9         9        11         0         0         0             0
221         9         9        11         0         0         0             0

[222 rows x 14 columns]
```

```
print(tdata.isna().sum())
```

```
location        0
country         0
gender          0
age             0
vis_wuhan       0
from_wuhan      0
death           0
symptom1        0
symptom2        0
symptom3        0
symptom4        0
symptom5        0
symptom6        0
diff_sym_hos    0
dtype: int64
```

```
from sklearn.metrics import recall_score as rs # Importing metrics
from sklearn.metrics import precision_score as ps
from sklearn.metrics import f1_score as fs
from sklearn.metrics import balanced_accuracy_score as bas
from sklearn.metrics import confusion_matrix as cm
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression as lr
```

```
X = tdata[['location','country','gender','age','vis_wuhan','from_wuhan','symptom1','symptom2','symptom3','symptom4','symptom5','sympt
```

```
Y = tdata['death'].values.ravel() # Creates flattened array


X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=10) # split X and Y into training and testing sets
classifier = lr(solver='lbfgs', max_iter=500 )
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1).ravel())
```

```
LogisticRegression(max_iter=500)
```

Evaluation of model

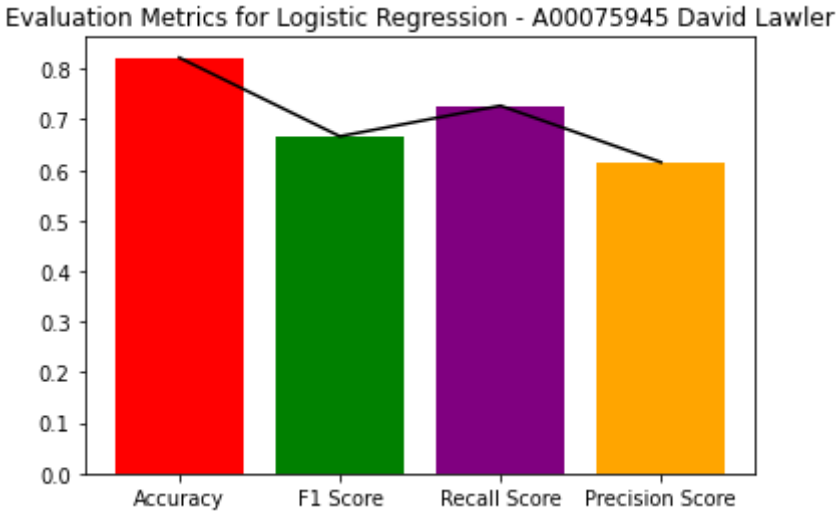
```
pred = np.array(classifier.predict(X_test)) # Evaluates test dataset using Logistic Regression

recall_lr = rs(Y_test,pred)
precision_lr = ps(Y_test,pred)
f1_lr = fs(Y_test,pred)
ma_lr = classifier.score(X_test,Y_test)


print('*** Evaluation metrics for test dataset ***\n')
print('Recall Score: ',recall_lr)
print('Precision Score: ',precision_lr)
print('F1 Score: ',f1_lr)
print('Accuracy: ',ma_lr)
a = pd.DataFrame(Y_test)
a['pred']= classifier.predict(X_test)
print('\n\tTable 3\n')
print(a.head())
```

```
*** Evaluation metrics for test dataset ***

Recall Score:  0.7272727272727273
Precision Score:  0.6153846153846154
F1 Score:  0.6666666666666667
Accuracy:  0.8222222222222222
```

```
Table 3

   0  pred
0  0     0
1  1     0
2  0     0
3  0     0
4  1     1
```

```
import matplotlib.pyplot as plt

plt.bar(['Accuracy','F1 Score','Recall Score','Precision Score'],[ma_lr,f1_lr,recall_lr,precision_lr],color=['red','green','purple',
plt.plot([ma_lr,f1_lr,recall_lr,precision_lr],color='black')
plt.title('Evaluation Metrics for Logistic Regression - A00075945 David Lawler')
```

```
Text(0.5, 1.0, 'Evaluation Metrics for Logistic Regression - A00075945 David Lawler')
```




```
print(pd.DataFrame({'Val':Y_test,'Pred':classifier.predict(X_test)}))
```

```
     Val  Pred
0      0     0
1      1     0
2      0     0
```

3	0	0
4	1	1
5	0	0
6	1	0
7	0	0
8	1	1
9	0	0
10	1	1
11	0	0
12	0	0
13	1	1
14	0	0
15	0	0
16	0	0
17	1	1
18	0	0
19	0	0
20	0	0
21	1	1
22	0	1
23	0	0
24	1	0
25	0	0
26	0	0
27	0	0
28	0	0
29	0	1
30	0	0
31	0	0
32	0	0
33	0	0
34	0	1
35	0	0
36	0	1
37	0	0
38	1	1
39	0	0
40	1	1
41	0	0
42	0	0

```
43    0     1
44    0     0
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier as dtc # Decision Tree setup
classifier = dtc(max_depth=2)
```

```
X = tdata[['location','country','gender','age','vis_wuhan','from_wuhan','symptom1','symptom2','symptom3','symptom4','symptom5','sympt
Y = tdata['death']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=10)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1))
```

```
DecisionTreeClassifier(max_depth=2)
```

Evaluation of the model

```
pred = np.array(classifier.predict(X_test))

recall_dtc = rs(Y_test,pred)
precision_dtc = ps(Y_test,pred)
f1_dtc = fs(Y_test,pred)
ma_dtc = classifier.score(X_test,Y_test)
```

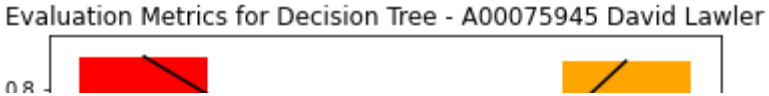
```
print('*** Evaluation metrics for test dataset ***\n')
print('Recall Score: ',recall_dtc)
print('Precision Score: ',precision_dtc)
print('F1 Score: ',f1_dtc)
print('Accuracy: ',ma_dtc)
a = pd.DataFrame(Y_test)
```

```
a['pred']= classifier.predict(X_test)
print('\n\tTable 3\n')
print(a.head())
```

```
*** Evaluation metrics for test dataset ***

Recall Score:  0.5454545454545454
Precision Score:  0.8571428571428571
F1 Score:  0.6666666666666665
Accuracy:  0.8666666666666667

        Table 3

     death  pred
184      0     0
170      1     0
142      0     0
182      0     0
49       1     0
```

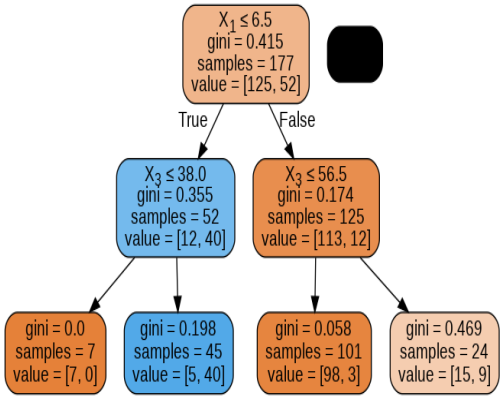
```
import matplotlib.pyplot as plt

plt.bar(['Accuracy','F1 Score','Recall Score','Precision Score'],[ma_dtc,f1_dtc,recall_dtc,precision_dtc],color=['red','green','purp
plt.plot([ma_dtc,f1_dtc,recall_dtc,precision_dtc],color='black')
plt.title('Evaluation Metrics for Decision Tree - A00075945 David Lawler')
```

```
Text(0.5, 1.0, 'Evaluation Metrics for Decision Tree - A00075945 David Lawler')
```



Visualizing the decision trees from random forest

```
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1)) # Decision tree visualised depth 2
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

estimator = classifier
dot_data = StringIO()
export_graphviz(estimator, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png(),width=250,height=200)
```



SVM

```
from sklearn import svm # Support Vector Machine setup
classifier = svm.SVC()
```

```
X = tdata[['location','country','gender','age','vis_wuhan','from_wuhan','symptom1','symptom2','symptom3','symptom4','symptom5','sympt
Y = tdata['death']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=10)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1).ravel())
```

```
SVC()
```

Evaluation of the model

```
pred = np.array(classifier.predict(X_test))

recall_svm = rs(Y_test,pred)
precision_svm = ps(Y_test,pred)
f1_svm = fs(Y_test,pred)
ma_svm = classifier.score(X_test,Y_test)
```

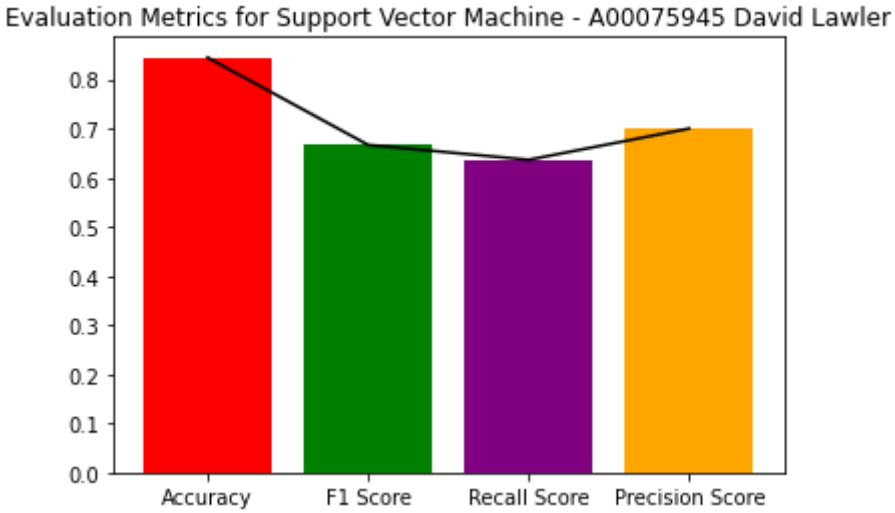
```
print('*** Evaluation metrics for test dataset ***\n')
print('Recall Score: ',recall_svm)
print('Precision Score: ',precision_svm)
print('F1 Score: ',f1_svm)
print('Accuracy: ',ma_svm)
a = pd.DataFrame(Y_test)
a['pred']= classifier.predict(X_test)
print('\n\tTable 3\n')
print(a.head())
```

```
*** Evaluation metrics for test dataset ***
```

```
Recall Score:  0.6363636363636364
Precision Score:  0.7
F1 Score:  0.6666666666666666
Accuracy:  0.8444444444444444

        Table 3

     death  pred
184      0     0
170      1     0
142      0     0
182      0     0
49       1     1
```

```
import matplotlib.pyplot as plt

plt.bar(['Accuracy','F1 Score','Recall Score','Precision Score'],[ma_svm,f1_svm,recall_svm,precision_svm],color=['red','green','purp
plt.plot([ma_svm,f1_svm,recall_svm,precision_svm],color='black')
plt.title('Evaluation Metrics for Support Vector Machine - A00075945 David Lawler')
```

```
Text(0.5, 1.0, 'Evaluation Metrics for Support Vector Machine - A00075945 David Lawler')
```



Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB # GNB setup
classifier = GaussianNB()
```

```
X = tdata[['location','country','gender','age','vis_wuhan','from_wuhan','symptom1','symptom2','symptom3','symptom4','symptom5','sympt
Y = tdata['death']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=10)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1).ravel())
```

```
GaussianNB()
```

Evaluation of the model

```
pred = np.array(classifier.predict(X_test))

recall_gnb = rs(Y_test,pred)
precision_gnb = ps(Y_test,pred)
f1_gnb = fs(Y_test,pred)
ma_gnb = classifier.score(X_test,Y_test)
```

```
print('*** Evaluation metrics for test dataset ***\n')
print('Recall Score: ',recall_gnb)
print('Precision Score: ',precision_gnb)
print('F1 Score: ',f1_gnb)
print('Accuracy: ',ma_gnb)
a = pd.DataFrame(Y_test)
a['pred']= classifier.predict(X_test)
print('\n\tTable 3\n')
```

```
print(a.head())
```

```
*** Evaluation metrics for test dataset ***

Recall Score:  0.7272727272727273
Precision Score:  0.5333333333333333
F1 Score:  0.6153846153846153
Accuracy:  0.7777777777777778

        Table 3

     death  pred
184      0     0
170      1     0
142      0     0
182      0     0
49       1     1
```

```
import matplotlib.pyplot as plt

plt.bar(['Accuracy','F1 Score','Recall Score','Precision Score'],[ma_gnb,f1_gnb,recall_gnb,precision_gnb],color=['red','green','purp
plt.plot([ma_gnb,f1_gnb,recall_gnb,precision_gnb],color='black')
plt.title('Evaluation Metrics for Gaussian Naive Bayes - A00075945 David Lawler')
```

```
Text(0.5, 1.0, 'Evaluation Metrics for Gaussian Naive Bayes - A00075945 David Lawler')
```

Boosted Random Forest

```
from sklearn.metrics import recall_score as rs # BRF Setup
from sklearn.metrics import precision_score as ps
from sklearn.metrics import f1_score as fs
from sklearn.metrics import balanced_accuracy_score as bas
from sklearn.metrics import confusion_matrix as cm
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifierrf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=2, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
classifier = AdaBoostClassifier(rf)
```

```
X = tdata[['location','country','gender','age','vis_wuhan','from_wuhan','symptom1','symptom2','symptom3','symptom4','symptom5','sympt
Y = tdata['death'].values.ravel()
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=0)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1).ravel())
```

```
AdaBoostClassifier(base_estimator=RandomForestClassifier(max_depth=2,
                                                         min_samples_leaf=2))
```

Evaluation of the model

```
pred = np.array(classifier.predict(X_test))

recall = rs(Y_test,pred)
precision = ps(Y_test,pred)
f1 = fs(Y_test,pred)
ma = classifier.score(X_test,Y_test)


print('*** Evaluation metrics for test dataset ***\n')
print('Recall Score: ',recall)
print('Precision Score: ',precision)
print('F1 Score: ',f1)
print('Accuracy: ',ma)
a = pd.DataFrame(Y_test)
a['pred']= classifier.predict(X_test)
print('\n\tTable 3\n')
print(a.head())
```

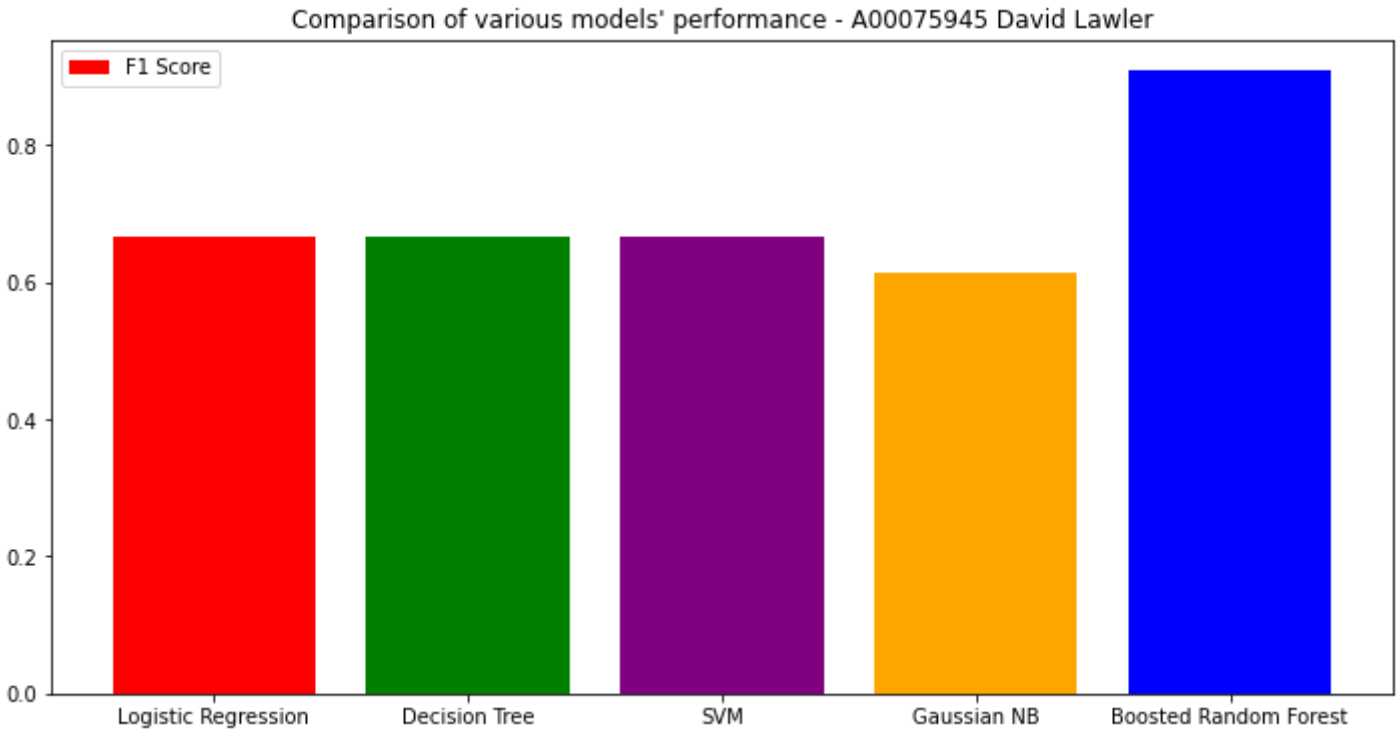
```
*** Evaluation metrics for test dataset ***

Recall Score:  0.8333333333333334
Precision Score:  1.0
F1 Score:  0.9090909090909091
Accuracy:  0.9555555555555556

        Table 3

   0  pred
0  0     0
1  0     0
2  1     1
3  0     0
4  0     0
```

Comparison of Evaluation Metrics

```
import matplotlib.pyplot as plt # Evaluation of Evaluation Metrics
fig = plt.figure(figsize=(8,6))
plt.plot(['Logistic Regression','Decision Tree','Boosted Random Forest','SVM','Gaussian NB'],[f1_lr,f1_dtc,f1,f1_svm,f1_gnb],color='
plt.plot(['Logistic Regression','Decision Tree','Boosted Random Forest','SVM','Gaussian NB'],[ma_lr,ma_dtc,ma,ma_svm,ma_gnb],color='
plt.plot(['Logistic Regression','Decision Tree','Boosted Random Forest','SVM','Gaussian NB'],[precision_lr,precision_dtc,precision,p
plt.plot(['Logistic Regression','Decision Tree','Boosted Random Forest','SVM','Gaussian NB'],[recall_lr,recall_dtc,recall,recall_svm
plt.legend(('F1 Score','Accuracy','Precision','Recall'))
plt.title('Comparison of various models\' performance - A00075945 David Lawler')

plt.show(fig)
```


```
fig = plt.figure(figsize=(12,6)) # visualisation of evaluation of classifiers using f1
plt.bar(['Logistic Regression','Decision Tree','SVM','Gaussian NB','Boosted Random Forest'],[f1_lr,f1_dtc,f1_svm,f1_gnb,f1],color=['
plt.legend(('F1 Score','Accuracy','Precision','Recall')) # Not sure why this won't print properly
plt.title('Comparison of various models\' performance - A00075945 David Lawler')
```

```
plt.show(fig)
```



Takes 10 minutes - Meta Estimator

```
X1 = tdata[['location','country','gender','age','vis_wuhan','from_wuhan','symptom1','symptom2','symptom3','symptom4','symptom5','sym
Y1 = tdata['death'].values.ravel()
classifier1 = RandomForestClassifier()

n_estimators = [100,200,300,400,500] # number of trees in forest
max_depth = [1,2,5,6] # depth of tree
min_samples_split = [1.0,2,6,7] # min number of samples to split an internal node
```

```
min_samples_leaf = [2,3,4,5] # min number of samples required to be at a leaf node

# parameter grid to explore
params_grid = {'n_estimators':n_estimators,'max_depth':max_depth,'min_samples_split':min_samples_split,'min_samples_leaf':min_sample

gridder = GridSearchCV(estimator=classifier1,param_grid=params_grid,n_jobs=-1,cv=5,verbose=5 ) # Finds best parameters
gridder.fit(X1,np.array(Y1).reshape(Y1.shape[0],1).ravel())
```

```
Fitting 5 folds for each of 320 candidates, totalling 1600 fits
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [1, 2, 5, 6],
                         'min_samples_leaf': [2, 3, 4, 5],
                         'min_samples_split': [1.0, 2, 6, 7],
                         'n_estimators': [100, 200, 300, 400, 500]},
             verbose=5)
```

```
print(gridder.best_estimator_) # prints best estimator
```

```
RandomForestClassifier(max_depth=6, min_samples_leaf=2, n_estimators=300)
```

```
!ls # print files
```

```
data.csv  sample_data  test.xlsx  train.csv
```

```
udata = pd.read_excel('test.xlsx')
udata = udata.drop('id',axis=1)
```

```
print(udata.columns)
```

```
Index(['location', 'country', 'gender', 'age', 'sym_on', 'hosp_vis',
       'vis_wuhan', 'from_wuhan', 'symptom1', 'symptom2', 'symptom3',
       'symptom4', 'symptom5', 'symptom6'],
      dtype='object')
```

```
udata = udata.fillna(np.nan,axis=0)
udata['age'] = udata['age'].fillna(value=udata['age'].mean())
```

```
udata['from_wuhan'] = udata['from_wuhan'].fillna(value=0)
udata['from_wuhan'] = udata['from_wuhan'].astype(int)
udata['location'] = encoder.fit_transform(udata['location'].astype(str))
udata['country'] = encoder.fit_transform(udata['country'].astype(str))
udata['gender'] = encoder.fit_transform(udata['gender'].astype(str))
udata['symptom1'] = encoder.fit_transform(udata['symptom1'].astype(str))
udata['symptom2'] = encoder.fit_transform(udata['symptom2'].astype(str))
udata['symptom3'] = encoder.fit_transform(udata['symptom3'].astype(str))
udata['symptom4'] = encoder.fit_transform(udata['symptom4'].astype(str))
udata['symptom5'] = encoder.fit_transform(udata['symptom5'].astype(str))
udata['symptom6'] = encoder.fit_transform(udata['symptom6'].astype(str))
```

```
print(udata['from_wuhan'].mode())
```

```
0    0
dtype: int64
```

```
udata['sym_on'] = pd.to_datetime(udata['sym_on'])
udata['hosp_vis'] = pd.to_datetime(udata['hosp_vis'])
udata['sym_on']= udata['sym_on'].map(dt.datetime.toordinal)
udata['hosp_vis']= udata['hosp_vis'].map(dt.datetime.toordinal)
udata['diff_sym_hos']= udata['hosp_vis'] - udata['sym_on']
```

```
print(udata['from_wuhan'].unique())
```

```
[0 1]
```

```
print(udata.dtypes)
```

```
location          int64
country           int64
gender            int64
age             float64
sym_on            int64
hosp_vis          int64
vis_wuhan         int64
```

```
from_wuhan        int64
symptom1          int64
symptom2          int64
symptom3          int64
symptom4          int64
symptom5          int64
symptom6          int64
diff_sym_hos      int64
dtype: object
```

```
udata = udata[['location','country','gender','age','vis_wuhan','from_wuhan','symptom1','symptom2','symptom3','symptom4','symptom5','s
udata['result'] = classifier.predict(udata)
```

```
print(udata['result'])
```

```
0      0
1      0
2      1
3      0
4      1
      ..
858    1
859    0
860    0
861    0
862    0
Name: result, Length: 863, dtype: int64
```

✓ 0s completed at 2:41 PM