```
#loading dataset
import pandas as pd
import numpy as np
#visualisation
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#EDA
from collections import Counter
import pandas_profiling as pp
# data preprocessing
from sklearn.preprocessing import StandardScaler
# data splitting
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler # for scaling
from sklearn.pipeline import make_pipeline # for create classifier with preprocessing
from sklearn.tree import DecisionTreeClassifier, plot_tree # for building model and draw (plo
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score

from sklearn.model_selection import cross_val_score, train_test_split # for cross validation,

# data modeling

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
#ensembling
from mlxtend.classifier import StackingCVClassifier


url = 'https://raw.githubusercontent.com/SohaMohajeri/Covid-19-Analysis-Visualization-and-For
df = pd.read_csv(url)
# Dataset is now stored in a Pandas Dataframe


df.head(2)
```

	id	case_in_country	reporting date	summary	location	country	gender	age	sympton
0	765	15.0	02-10-20	new confirmed COVID-19 patient in Vietnam: 3 m...	Vinh Phuc	Vietnam	NaN	0.25	
1	477	27.0	02-05-20	new confirmed COVID-19	Singapore	Singapore	male	0.50	

```
df.shape
```

```
(1085, 20)
```

```
df.dtypes
```

```
id                        int64
case_in_country         float64
reporting date           object
summary                  object
location                 object
country                  object
gender                   object
age                     float64
symptom_onset            object
If_onset_approximated   float64
hosp_visit_date          object
exposure_start           object
exposure_end             object
visiting Wuhan            int64
from Wuhan                int64
death                     int64
recovered                 int64
symptom                  object
source                   object
link                     object
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1085 entries, 0 to 1084
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     1085 non-null   int64  
 1   case_in_country        888 non-null    float64
 2   reporting date         1084 non-null   object 
 3   summary                1080 non-null   object 
 4   location               1085 non-null   object 
```

```
 5   country                1085 non-null   object
 6   gender                 902 non-null    object
 7   age                    843 non-null    float64
 8   symptom_onset          563 non-null    object
 9   If_onset_approximated  560 non-null    float64
 10  hosp_visit_date        507 non-null    object
 11  exposure_start         128 non-null    object
 12  exposure_end           341 non-null    object
 13  visiting Wuhan         1085 non-null   int64
 14  from Wuhan             1085 non-null   int64
 15  death                  1085 non-null   int64
 16  recovered              1085 non-null   int64
 17  symptom                270 non-null    object
 18  source                 1085 non-null   object
 19  link                   1085 non-null   object
dtypes: float64(3), int64(5), object(12)
memory usage: 169.7+ KB
```

```
df.drop(['id','case_in_country','summary','symptom_onset', 'If_onset_approximated', 'hosp_vis
'exposure_end', 'symptom', 'source', 'link'],axis=1,inplace=True)
```

```
100*df.isnull().sum()/df.shape[0]
```

```
reporting date     0.092166
location           0.000000
country            0.000000
gender            16.866359
age               22.304147
visiting Wuhan     0.000000
from Wuhan         0.000000
death              0.000000
recovered          0.000000
dtype: float64
```

```
df['age']= df['age']. fillna(df['age'].mean())
```

```
df_dum=pd.get_dummies(df['gender'].dropna(), drop_first=True)
df_dum['male'].median()
```

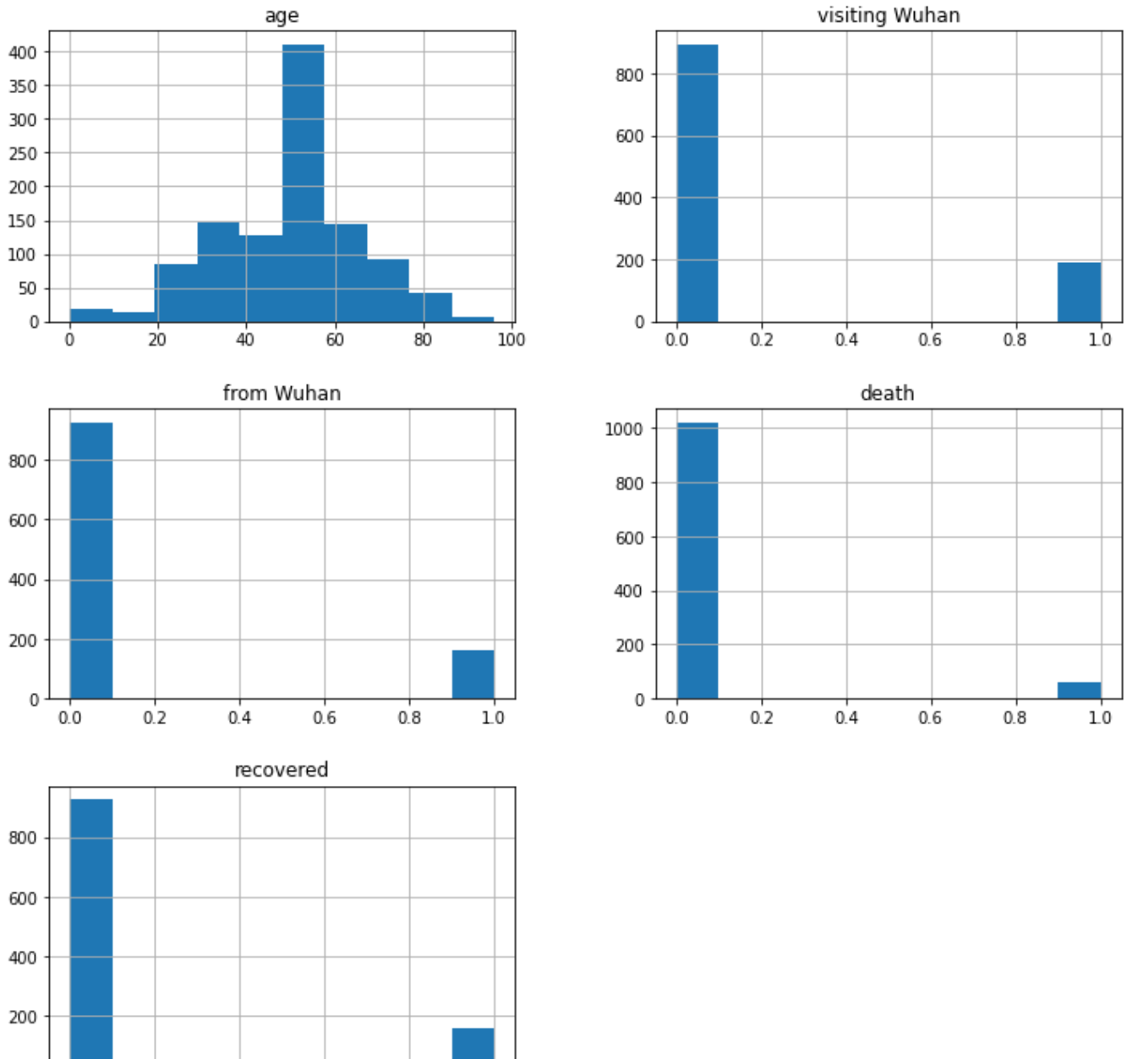
```
1.0
```

```
df['gender']= df['gender']. fillna('male')
df.dropna(inplace=True)
df.isnull().sum()
```

```
reporting date    0
location          0
country           0
gender            0
age               0
visiting Wuhan    0
from Wuhan        0
```

```
death             0
recovered         0
dtype: int64
```

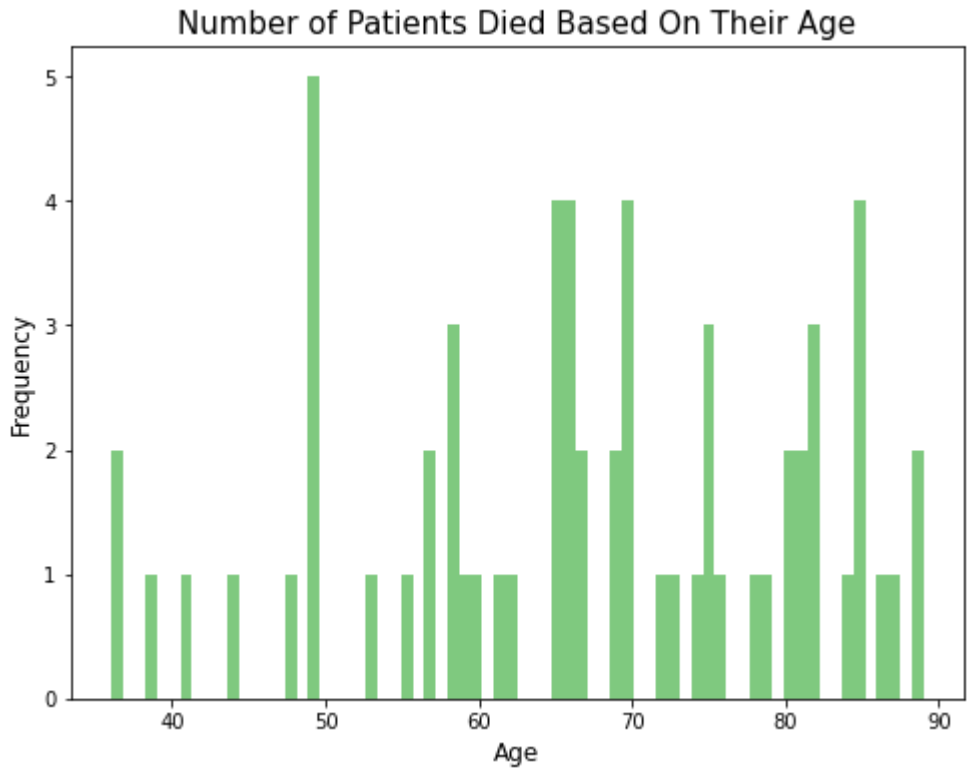
```
# plot histograms for each variable
df.hist(figsize = (12, 12))
plt.show()
```



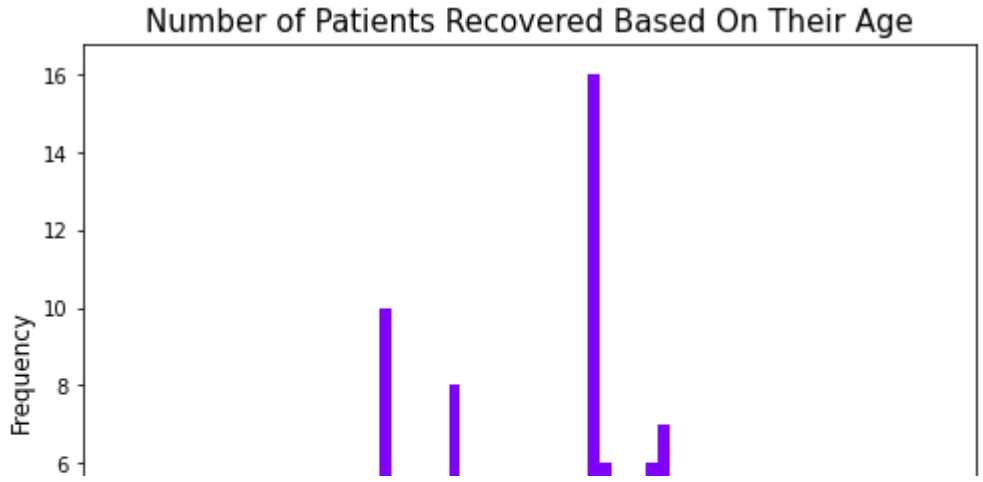

```
df.columns=df.columns.str.lower().str.replace(' ','_')
df['reporting_date']=pd.to_datetime(df['reporting_date'])
df['year']=df['reporting_date'].apply(lambda x:x.year)
df['month']=df['reporting_date'].apply(lambda x:x.month)
df['month'].unique()
df.drop(['reporting_date', 'year'], axis=1, inplace=True)
df.head(2)
```

	location	country	gender	age	visiting_wuhan	from_wuhan	death	recovered	month
0	Vinh Phuc	Vietnam	male	0.25	0	0	0	1	2
1	Singapore	Singapore	male	0.50	0	0	0	1	2

```
plt.figure(figsize=(8,6))
df[df['death']==1]['age'].plot(kind='hist',bins=70,colormap='Accent')
plt.title('Number of Patients Died Based On Their Age',fontsize=15)
plt.xlabel('Age',fontsize=12)
plt.ylabel('Frequency',fontsize=12)
plt.show()
```




```
plt.figure(figsize=(8,6))
df[df['recovered']==1]['age'].plot(kind='hist',bins=70,colormap='rainbow')
plt.title('Number of Patients Recovered Based On Their Age',fontsize=15)
plt.xlabel('Age',fontsize=12)
plt.ylabel('Frequency',fontsize=12)
plt.show()
```



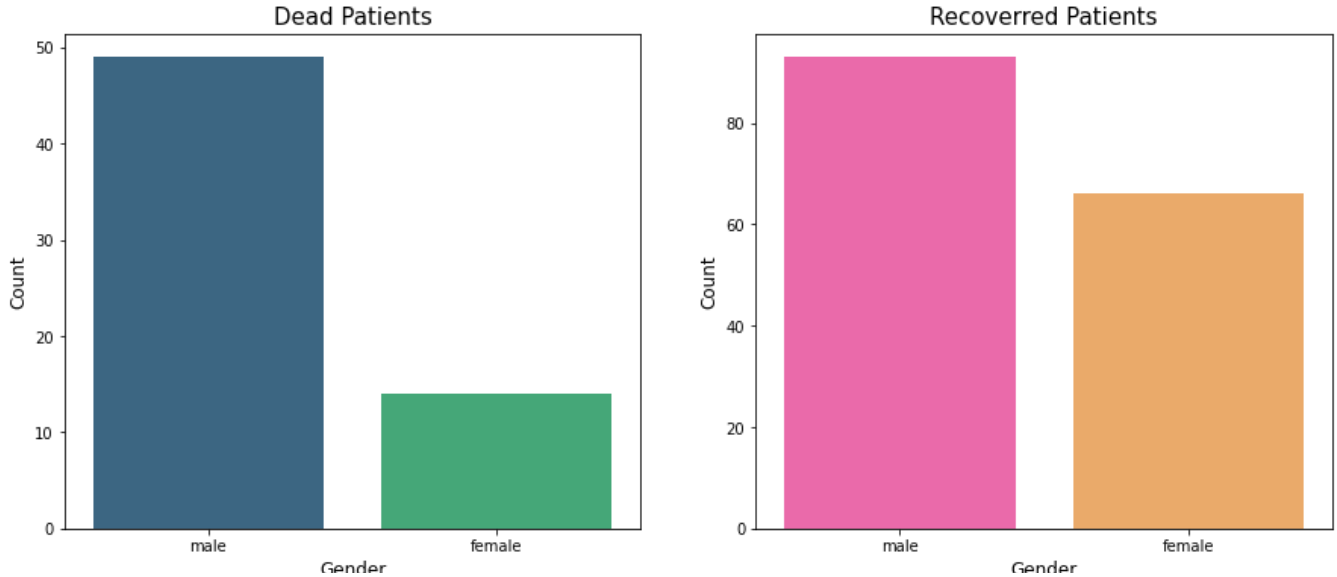

```
print('Current count of patients:',df['death'].count())
print('Number of Dead Patients:', df[df['death']==1]['death'].count())
print('Number of Recovered Patients:',df[df['recovered']==1]['death'].count())
print('Number of Patients Receiving Treatment:',df[(df['death']==0)&(df['recovered']==0)]['de
```

```
Current count of patients: 1084
Number of Dead Patients: 63
Number of Recovered Patients: 159
Number of Patients Receiving Treatment: 862
```

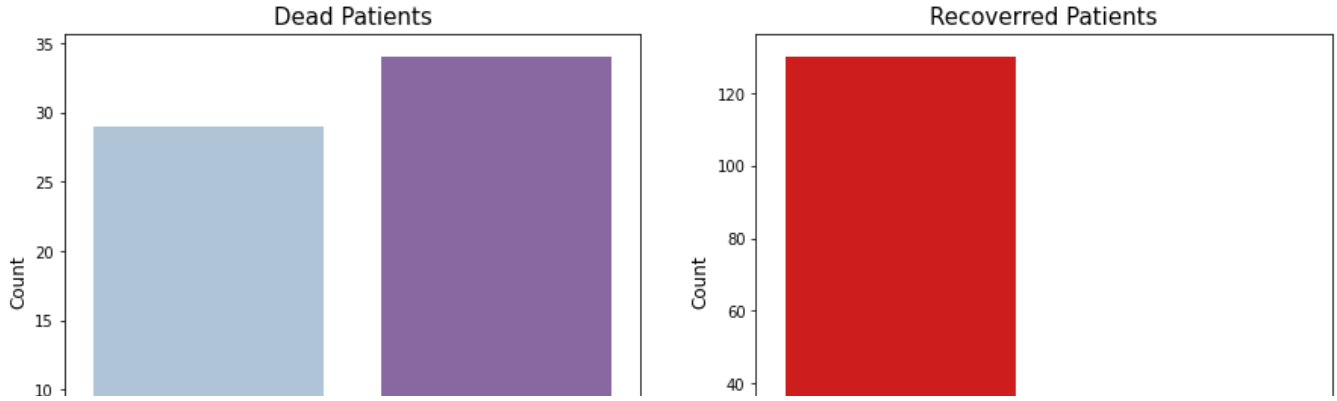
```
plt.figure(figsize=(8,6))
plt.bar(x=['Recovered','Dead'],height=[159,63], color='pink')
plt.title('Patients Status',fontsize=15)
plt.xlabel('Status', fontsize=12)
plt.ylabel('Number',fontsize=12)
plt.show()
```

Patients Status

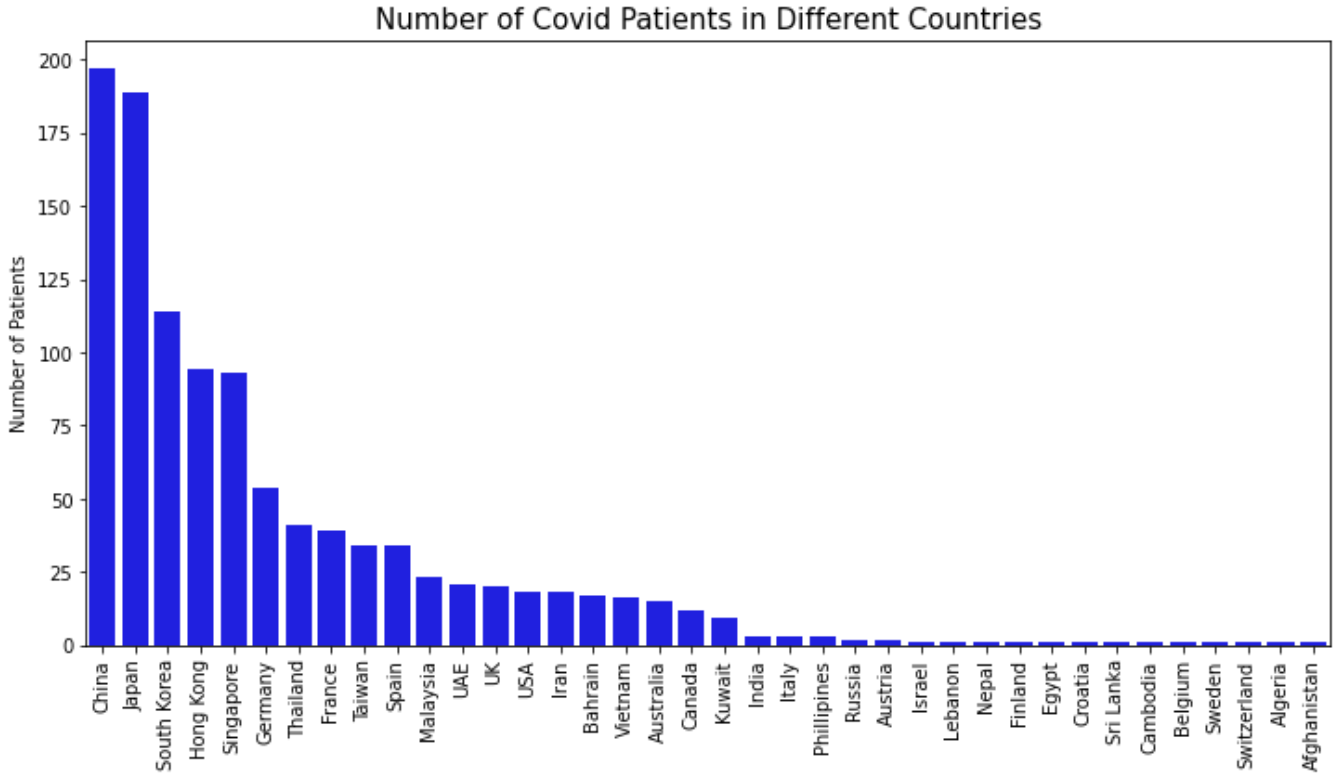
```
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
sns.countplot(x='gender', data=df[df['death']==1], palette='viridis')
plt.xlabel('Gender', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Dead Patients',fontsize=15)
plt.subplot(1,2,2)
sns.countplot(x='gender', data=df[df['recovered']==1], palette='spring')
plt.xlabel('Gender', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Recoverred Patients',fontsize=15)
plt.show()
```




```
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
sns.countplot(x='from_wuhan', data=df[df['death']==1], palette='BuPu')
plt.xticks([0,1], ['Not from Wuhan','from Wuhan'])
plt.xlabel('Origin', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Dead Patients',fontsize=15)
plt.subplot(1,2,2)
sns.countplot(x='from_wuhan', data=df[df['recovered']==1], palette='hot')
plt.xticks([0,1], ['Not from Wuhan','from Wuhan'])
plt.xlabel('Origin', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Recoverred Patients',fontsize=15)
plt.show()
```




```
country_order=list(df.groupby('country').count()['location'].sort_values(ascending=False).ind
plt.figure(figsize=(12,6))
sns.countplot(x='country',data=df,color='blue',order=country_order)
plt.xticks(rotation=90)
plt.ylabel('Number of Patients')
plt.xlabel('Country')
plt.title('Number of Covid Patients in Different Countries',fontsize=15)
plt.show()
```




```
groupby_df=df.groupby('country').sum()


le1=LabelEncoder()
le1.fit(df['location'])
df['location']=le1.transform(df['location'])


le2=LabelEncoder()
le2.fit(df['country'])
```

```
df['country']=le2.transform(df['country'])


le3=LabelEncoder()
le3.fit(df['gender'])
df['gender']=le3.transform(df['gender'])


df.head()
```

	location	country	gender	age	visiting_wuhan	from_wuhan	death	recovered	month
0	141	37	1	0.25	0	0	0	1	2
1	118	26	1	0.50	0	0	0	1	2
2	118	26	1	1.00	0	0	0	1	2
3	42	8	0	2.00	1	0	0	0	1
4	60	22	1	2.00	0	0	0	1	1

```
y=df['recovered']
X=df[['location','country','gender','age','visiting_wuhan','from_wuhan','month']]


from sklearn.model_selection import train_test_split



X_train,X_test, y_train,y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Double-click (or enter) to edit

```
print(y_test.unique())
Counter(y_train)
```

```
[0 1]
Counter({0: 734, 1: 133})
```

logistic regression

```
lr1=LogisticRegression()
lr1.fit(X,y)
predictions_lr1=lr1.predict(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Convergenc

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
print(confusion_matrix(y_test,predictions_lr1))
print('\n')
print(classification_report(y_test,predictions_lr1))
```

```
[[190   1]
 [ 24   2]]


              precision    recall  f1-score   support

           0       0.89      0.99      0.94       191
           1       0.67      0.08      0.14        26

    accuracy                           0.88       217
   macro avg       0.78      0.54      0.54       217
weighted avg       0.86      0.88      0.84       217
```

```
model_acc_LR = accuracy_score(y_test, predictions_lr1)
print('The accuracy of our linear regression classifier model is: %0.3f'% model_acc_LR)
```

```
The accuracy of our linear regression classifier model is: 0.885
```

DEcision Tree Classifier

```
dtc1=DecisionTreeClassifier()
dtc1.fit(X_train,y_train)
predictions_dtc1=dtc1.predict(X_test)
```

```
print(confusion_matrix(y_test,predictions_dtc1))
print("\n")
print(classification_report(y_test,predictions_dtc1))
```

```
[[177  14]
 [ 17   9]]


              precision    recall  f1-score   support

           0       0.91      0.93      0.92       191
           1       0.39      0.35      0.37        26

    accuracy                           0.86       217
```

```
        macro avg       0.65      0.64      0.64       217
     weighted avg       0.85      0.86      0.85       217
```

```
model_acc_DT = accuracy_score(y_test, predictions_dtc1)
print('The accuracy of our Decision Tree classifier model is: %0.3f'% model_acc_DT)
```

```
     The accuracy of our Decision Tree classifier model is: 0.857
```

Random Forest Classifier

```
rfc1=RandomForestClassifier(n_estimators=200)
rfc1.fit(X_train,y_train)
predictions_rfc1=rfc1.predict(X_test)
```

```
print(confusion_matrix(y_test,predictions_rfc1))
print('\n')
print(classification_report(y_test,predictions_rfc1))
```

```
     [[184   7]
      [ 15  11]]


                   precision    recall  f1-score   support

                0       0.92      0.96      0.94       191
                1       0.61      0.42      0.50        26

            accuracy                           0.90       217
           macro avg       0.77      0.69      0.72       217
        weighted avg       0.89      0.90      0.89       217
```

```
model_acc_RF = accuracy_score(y_test, predictions_rfc1)
print('The accuracy of our random forest classifier model is: %0.3f'% model_acc_RF)
```

```
     The accuracy of our random forest classifier model is: 0.899
```

```
svc1=SVC()
svc1.fit(X_train,y_train)
predictions_svc1=svc1.predict(X_test)
```

```
print(confusion_matrix(y_test,predictions_svc1))
print('\n')
print(classification_report(y_test,predictions_svc1))
```

```
     [[191   0]
      [ 26   0]]
```

```
                  precision    recall  f1-score   support

               0       0.88      1.00      0.94       191
               1       0.00      0.00      0.00        26

        accuracy                           0.88       217
       macro avg       0.44      0.50      0.47       217
    weighted avg       0.77      0.88      0.82       217

    /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undefine

    Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
```

```
model_acc_SVC = accuracy_score(y_test, predictions_svc1)
print('The accuracy of our Supprot Vector classifier model is: %0.3f'% model_acc_SVC)
```

```
    The accuracy of our Supprot Vector classifier model is: 0.880
```

PROPOSED ALGORITHM

```
xgbc1=xgb.XGBClassifier(n_estimators=200, learning_rate=0.08, gamma=0, subsample=0.5,colsampl
xgbc1.fit(X_train,y_train)
```

```
    XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=1, gamma=0,
                  learning_rate=0.08, max_delta_step=0, max_depth=8,
                  min_child_weight=1, missing=None, n_estimators=200, n_jobs=1,
                  nthread=None, objective='binary:logistic', random_state=0,
                  reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=None, subsample=0.5, verbosity=1)
```

```
predictions_xgbc1=xgbc1.predict(X_test)
```

```
print(confusion_matrix(y_test,predictions_xgbc1))
print('\n')
print(classification_report(y_test,predictions_xgbc1))
```

```
    [[185   6]
     [ 11  15]]


                  precision    recall  f1-score   support

               0       0.94      0.97      0.96       191
               1       0.71      0.58      0.64        26

        accuracy                           0.92       217
       macro avg       0.83      0.77      0.80       217
```

```
     weighted avg       0.92      0.92      0.92       217
```

```
model_acc_Proposed = accuracy_score(y_test, predictions_xgbc1)
print('The accuracy of our Proposed classifier model is: %0.3f'%  model_acc_Proposed)
```

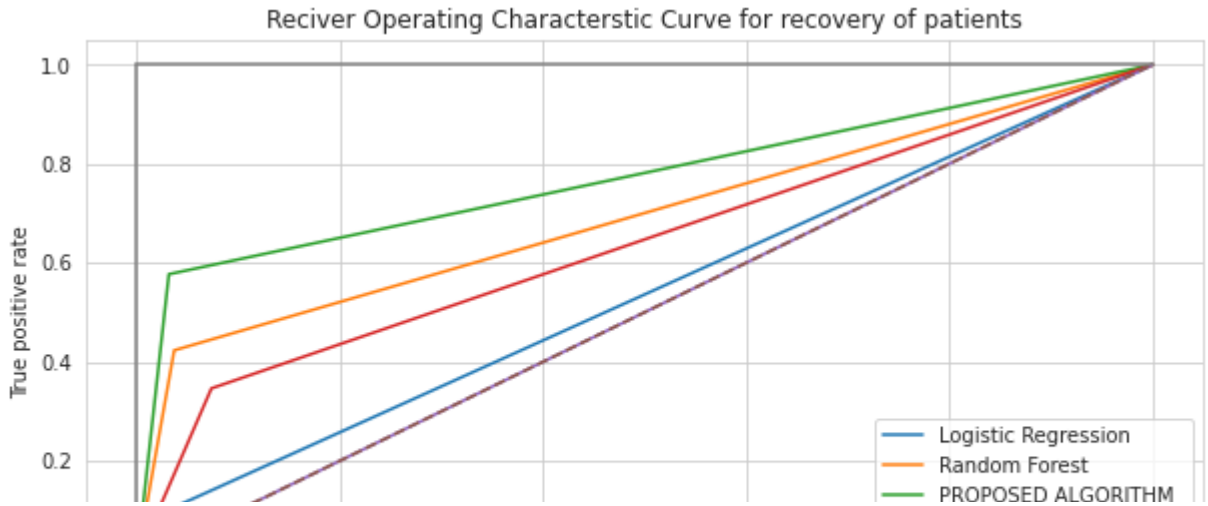
```
     The accuracy of our Proposed classifier model is: 0.922
```

```
print('Accuracy Score, Logistic Regression: ', round(accuracy_score(y_test,predictions_lr1),n
print('Accuracy Score, Decision Tree Classifier: ', round(accuracy_score(y_test,predictions_d
print('Accuracy Score, Random Forest Classifier: ', round(accuracy_score(y_test,predictions_r
print('Accuracy Score, Support Vector Classifier: ', round(accuracy_score(y_test,predictions_
print('Accuracy Score, Proposed Classifier: ', round(accuracy_score(y_test,predictions_xgbc1)
```

```
     Accuracy Score, Logistic Regression:  0.885
     Accuracy Score, Decision Tree Classifier:  0.857
     Accuracy Score, Random Forest Classifier:  0.899
     Accuracy Score, Support Vector Classifier:  0.88
     Accuracy Score, Proposed Classifier:  0.92
```

```
lr_false_positive_rate,lr_true_positive_rate,lr_threshold = roc_curve(y_test,predictions_lr1)
rf_false_positive_rate,rf_true_positive_rate,rf_threshold = roc_curve(y_test,predictions_rfc1
xgb_false_positive_rate,xgb_true_positive_rate,xgb_threshold = roc_curve(y_test,predictions_x
dt_false_positive_rate,dt_true_positive_rate,dt_threshold = roc_curve(y_test,predictions_dtc1
svc_false_positive_rate,svc_true_positive_rate,svc_threshold = roc_curve(y_test,predictions_s
```

```
sns.set_style('whitegrid')
plt.figure(figsize=(10,5))
plt.title('Reciver Operating Characterstic Curve for recovery of patients')
plt.plot(lr_false_positive_rate,lr_true_positive_rate,label='Logistic Regression')
plt.plot(rf_false_positive_rate,rf_true_positive_rate,label='Random Forest')
plt.plot(xgb_false_positive_rate,xgb_true_positive_rate,label='PROPOSED ALGORITHM')
plt.plot(dt_false_positive_rate,dt_true_positive_rate,label='Desion Tree')
plt.plot(svc_false_positive_rate,svc_true_positive_rate,label='Support Vector Classifier')
plt.plot([0,1],ls='--')
plt.plot([0,0],[1,0],c='.5')
plt.plot([1,1],c='.5')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.legend()
plt.show()
```



PREDICTION OF DEATH OF COVID - 19

0.0 0.2 0.4 0.6 0.8 1.0

```
y=df['death']
X=df[['location','country','gender','age','visiting_wuhan','from_wuhan','month']]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

LOGISTIC REGRESSION

```
lr2=LogisticRegression()
lr2.fit(X,y)
predictions_lr2=lr2.predict(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Convergenc

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
print(confusion_matrix(y_test,predictions_lr2))
print('\n')
print(classification_report(y_test,predictions_lr2))
```

```
[[201   1]
 [ 11   4]]


              precision    recall  f1-score   support
```

```
              0       0.95      1.00      0.97       202
              1       0.80      0.27      0.40        15

       accuracy                           0.94       217
      macro avg       0.87      0.63      0.69       217
   weighted avg       0.94      0.94      0.93       217
```

DECISION TREE CLASSIFIER

```
dtc2=DecisionTreeClassifier()
dtc2.fit(X_train,y_train)
predictions_dtc2=dtc2.predict(X_test)
print(confusion_matrix(y_test,predictions_dtc2))
print("\n")
print(classification_report(y_test,predictions_dtc2))
```

```
    [[198   4]
     [  7   8]]


                  precision    recall  f1-score   support

              0       0.97      0.98      0.97       202
              1       0.67      0.53      0.59        15

       accuracy                           0.95       217
      macro avg       0.82      0.76      0.78       217
   weighted avg       0.95      0.95      0.95       217
```

PROPOSED ALGORITHM

```
rfc2=RandomForestClassifier(n_estimators=200)
rfc2.fit(X_train,y_train)
predictions_rfc2=rfc2.predict(X_test)
print(confusion_matrix(y_test,predictions_rfc2))
print('\n')
print(classification_report(y_test,predictions_rfc2))
```

```
    [[202   0]
     [ 10   5]]


                  precision    recall  f1-score   support

              0       0.95      1.00      0.98       202
              1       1.00      0.33      0.50        15

       accuracy                           0.95       217
```

```
        macro avg       0.98      0.67      0.74       217
     weighted avg       0.96      0.95      0.94       217
```

SUPPORT VECTOR CLASSIFIER

```
svc2=SVC()
svc2.fit(X_train,y_train)
predictions_svc2=svc2.predict(X_test)
```

```
print(confusion_matrix(y_test,predictions_svc2))
print('\n')
print(classification_report(y_test,predictions_svc2))
```

```
     [[202   0]
      [ 15   0]]


                   precision    recall  f1-score   support

                0       0.93      1.00      0.96       202
                1       0.00      0.00      0.00        15

         accuracy                           0.93       217
        macro avg       0.47      0.50      0.48       217
     weighted avg       0.87      0.93      0.90       217

     /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undefine

     Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
```

RANDOM BOOST CLASSIFIER

```
xgbc2=xgb.XGBClassifier(n_estimators=200, learning_rate=0.08, gamma=0, subsample=0.5,colsampl
xgbc2.fit(X_train,y_train)
```

```
     XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                   colsample_bynode=1, colsample_bytree=1, gamma=0,
                   learning_rate=0.08, max_delta_step=0, max_depth=8,
                   min_child_weight=1, missing=None, n_estimators=200, n_jobs=1,
                   nthread=None, objective='binary:logistic', random_state=0,
                   reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                   silent=None, subsample=0.5, verbosity=1)
```

```
predictions_xgbc2=xgbc2.predict(X_test)
```

```
print(confusion_matrix(y_test,predictions_xgbc2))
print('\n')
```

```
print( \n )
print(classification_report(y_test,predictions_xgbc2))
```

```
     [[201   1]
      [ 10   5]]


                   precision    recall  f1-score   support

                0       0.95      1.00      0.97       202
                1       0.83      0.33      0.48        15

         accuracy                           0.95       217
        macro avg       0.89      0.66      0.72       217
     weighted avg       0.94      0.95      0.94       217
```

```
print('Accuracy Score, Logistic Regression: ', round(accuracy_score(y_test,predictions_lr2),n
print('Accuracy Score, Decision Tree Classifier: ', round(accuracy_score(y_test,predictions_d
print('Accuracy Score, Proposed Classifier : ', round(accuracy_score(y_test,predictions_rfc2)
print('Accuracy Score, Support Vector Classifier: ', round(accuracy_score(y_test,predictions_
print('Accuracy Score, Random Forest Classifier: ', round(accuracy_score(y_test,predictions_x
```

```
     Accuracy Score, Logistic Regression:  0.945
     Accuracy Score, Decision Tree Classifier:  0.949
     Accuracy Score, Proposed Classifier :  0.954
     Accuracy Score, Support Vector Classifier:  0.931
     Accuracy Score, Random Forest Classifier:  0.949
```

```
lr2_false_positive_rate,lr2_true_positive_rate,lr2_threshold = roc_curve(y_test,predictions_l
rf2_false_positive_rate,rf2_true_positive_rate,rf2_threshold = roc_curve(y_test,predictions_r
xgb2_false_positive_rate,xgb2_true_positive_rate,xgb2_threshold = roc_curve(y_test,prediction
dt2_false_positive_rate,dt2_true_positive_rate,dt2_threshold = roc_curve(y_test,predictions_d
svc2_false_positive_rate,svc2_true_positive_rate,svc2_threshold = roc_curve(y_test,prediction
```

```
sns.set_style('whitegrid')
plt.figure(figsize=(10,5))
plt.title('Reciver Operating Characterstic Curve for death of patients')
plt.plot(lr2_false_positive_rate,lr2_true_positive_rate,label='Logistic Regression')
plt.plot(rf2_false_positive_rate,rf2_true_positive_rate,label='Decision Tree')
plt.plot(xgb2_false_positive_rate,xgb2_true_positive_rate,label='Random Forest')
plt.plot(dt2_false_positive_rate,dt2_true_positive_rate,label='PROPOSED ALGORITHM')
plt.plot(svc2_false_positive_rate,svc2_true_positive_rate,label='Support Vector Classifier')
plt.plot([0,1],ls='--')
plt.plot([0,0],[1,0],c='.5')
plt.plot([1,1],c='.5')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.legend()
plt.show()
```