Group 2 A2

July 17, 2021

0.1 [CM1]

[148]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
```

[149]:
```
# load dkmacovid training data
df = pd.read_csv('dkmacovid_train.csv')
# remove comma from dataframe in 'Resident Population 2020 Census' and␣
 ↪'Population Density 2020 Census'
df = df.replace(',','', regex=True)
# convert string to numeric data
df['Resident Population 2020 Census'] = df['Resident Population 2020 Census'].
 ↪astype(float)
df['Population Density 2020 Census'] = df['Population Density 2020 Census'].
 ↪astype(float)
df.head()
```

[149]:

	Day	State ID	State	Lat	Long_	Active	Incident_Rate	\
0	2	1	Alabama	32.3182	-86.9023	162449	7535.061394	
1	2	2	Alaska	61.3707	-152.4044	40421	6534.252848	
2	2	3	Arizona	33.7298	-111.4312	452222	7407.212013	
3	2	4	Arkansas	34.9697	-92.3731	24012	7669.219075	
4	2	5	California	36.1162	-119.6816	2362015	6045.109130	

	Total_Test_Results	Case_Fatality_Ratio	Testing_Rate	\
0	1891468	1.318688	38576.31315	
1	1290349	0.449781	176386.82510	
2	5218721	1.680608	39916.14181	
3	2079788	1.611203	68917.26567	
4	33391442	1.111215	84509.14544	

	Resident Population 2020 Census	Population Density 2020 Census	\
0	5024279.0	99.2	

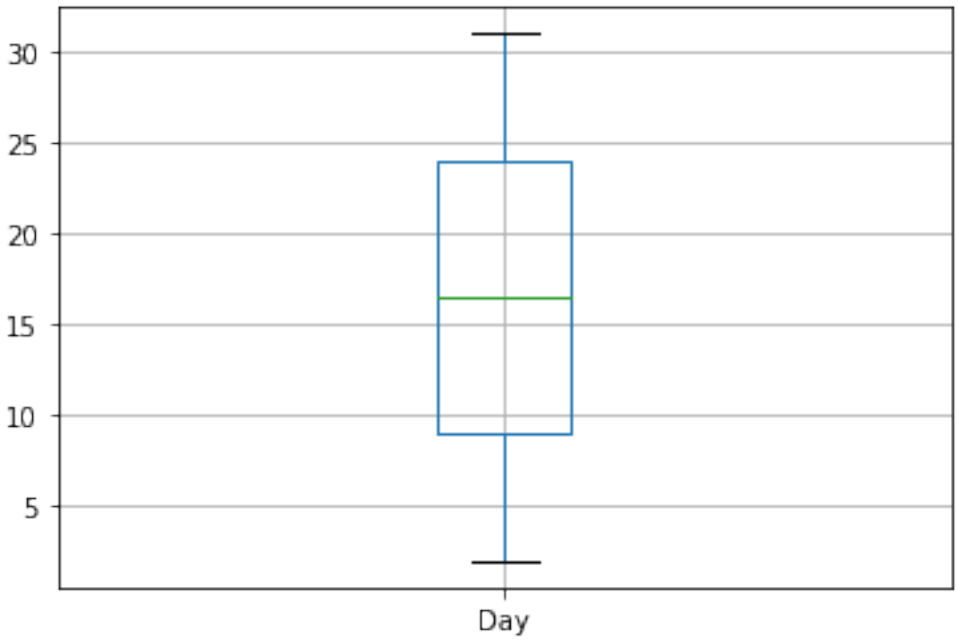
1	733391.0	1.3
2	7151502.0	62.9
3	3011524.0	57.9
4	39538223.0	253.7

	Density Rank 2020 Census	SexRatio	Confirmed	Deaths	Recovered
0	29	94	True	False	False
1	52	109	True	True	False
2	35	99	True	True	True
3	36	96	True	True	True
4	13	99	True	True	False

0.1.1 Data visulization

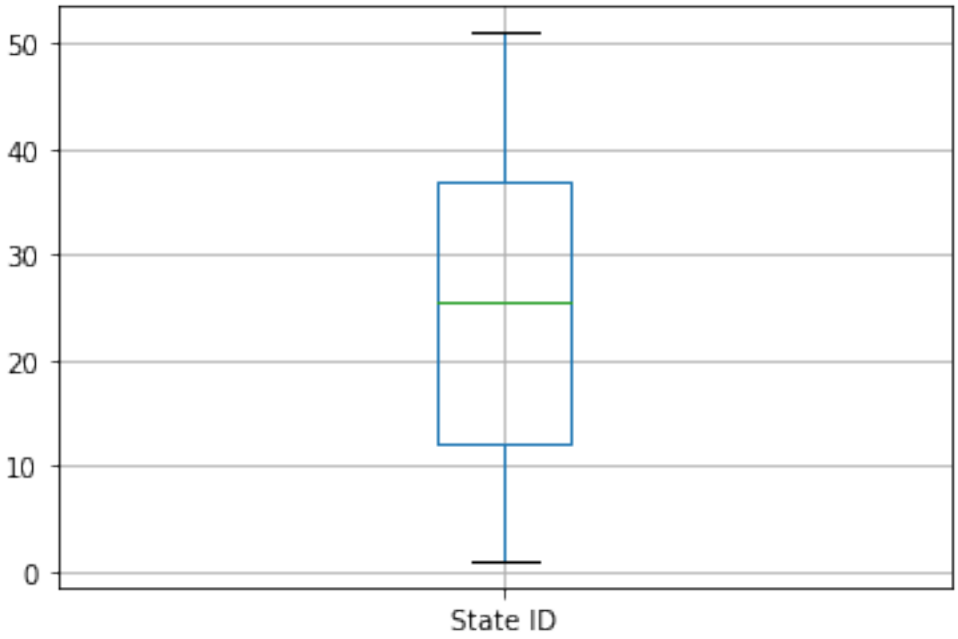
[3]:
```
# df = pd.DataFrame(data = np.random.random(size=(4,4)), columns =␣
↪['A','B','C','D'])
df.boxplot(column=['Day'])
```

[3]: <AxesSubplot:>



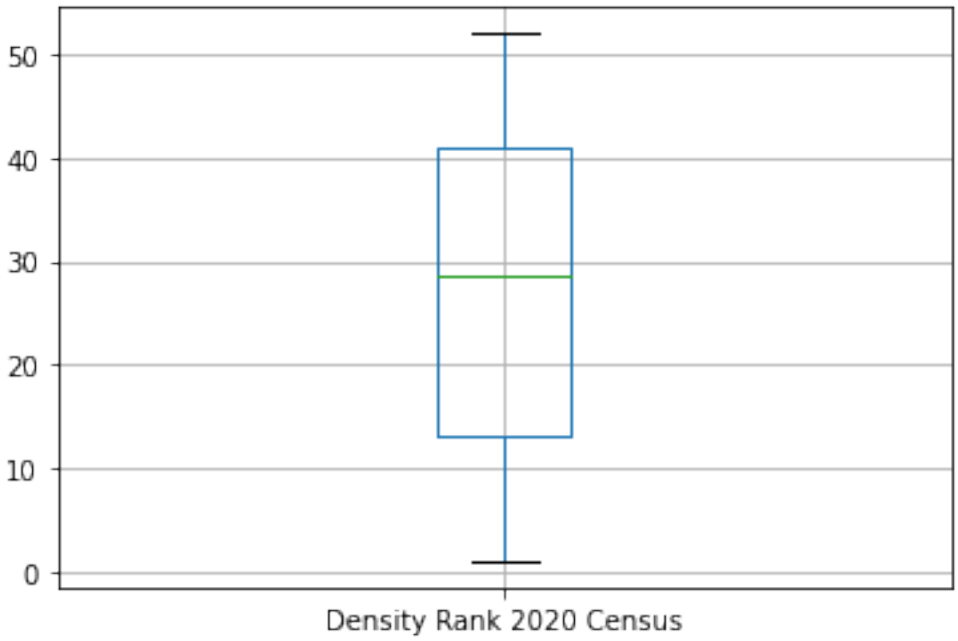
[4]:
```
df.boxplot(column=['State ID'])
```

[4]: <AxesSubplot:>



[5]:
```
df.boxplot(column=['Density Rank 2020 Census'])
```

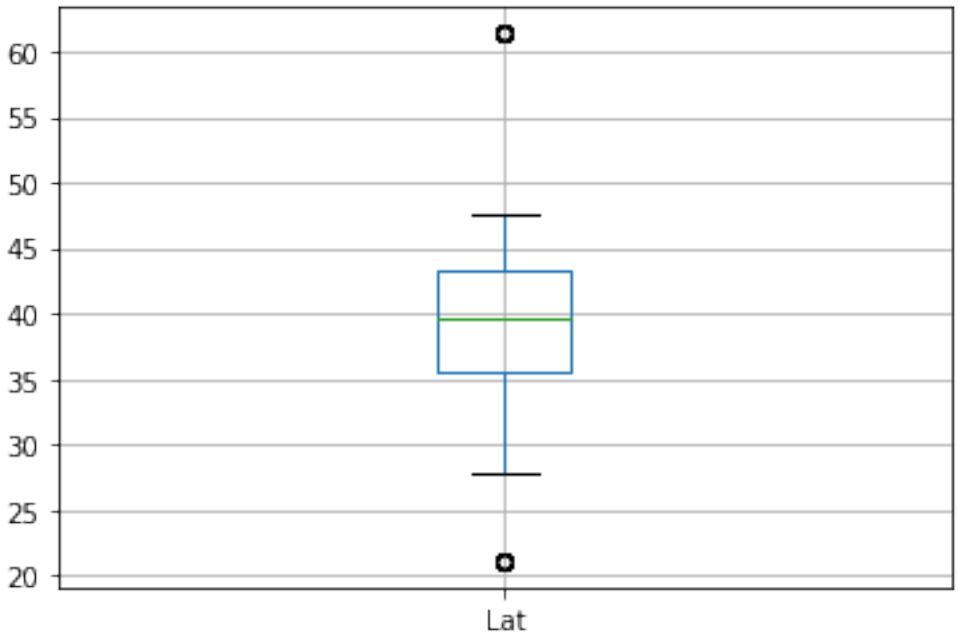
[5]: <AxesSubplot:>



[6]:
```
# print(df.columns.tolist())
```

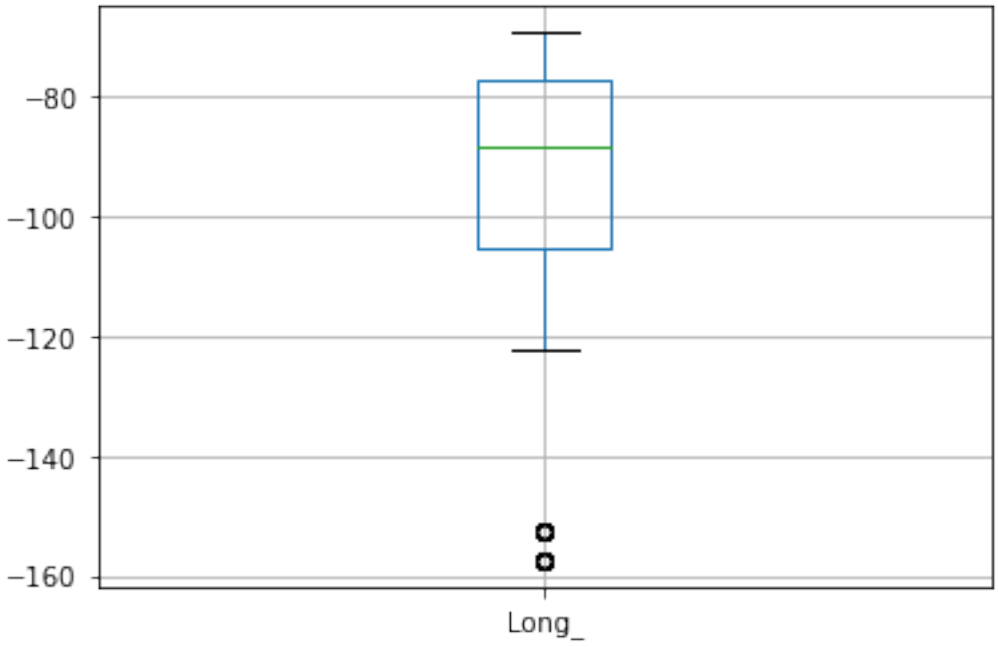
[7]:
```
df.boxplot(column=['Lat'])
```

[7]: <AxesSubplot:>



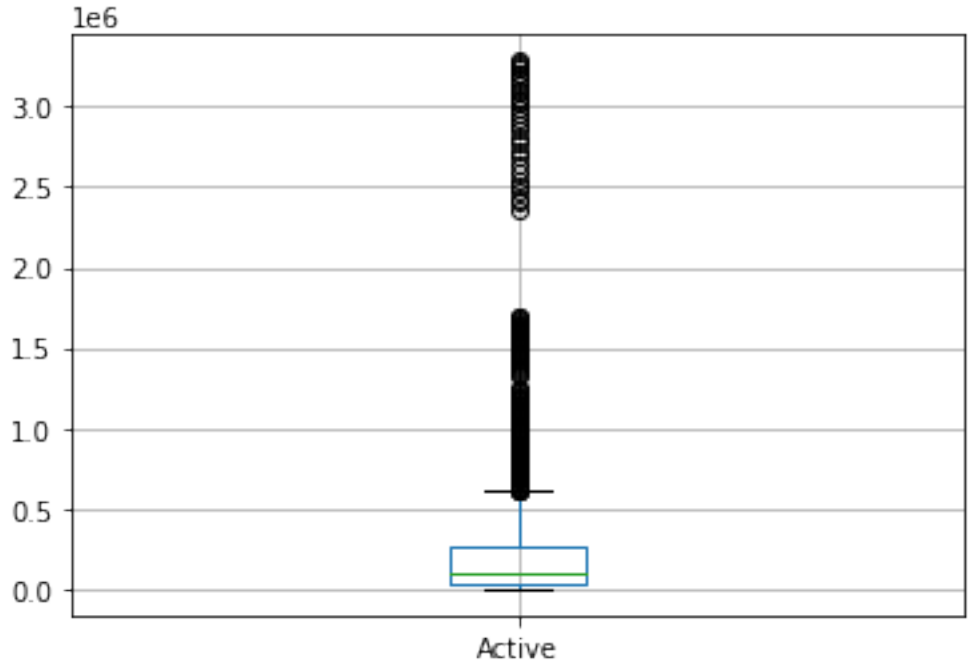
[8]:
```
df.boxplot(column=['Long_'])
```

[8]: <AxesSubplot:>



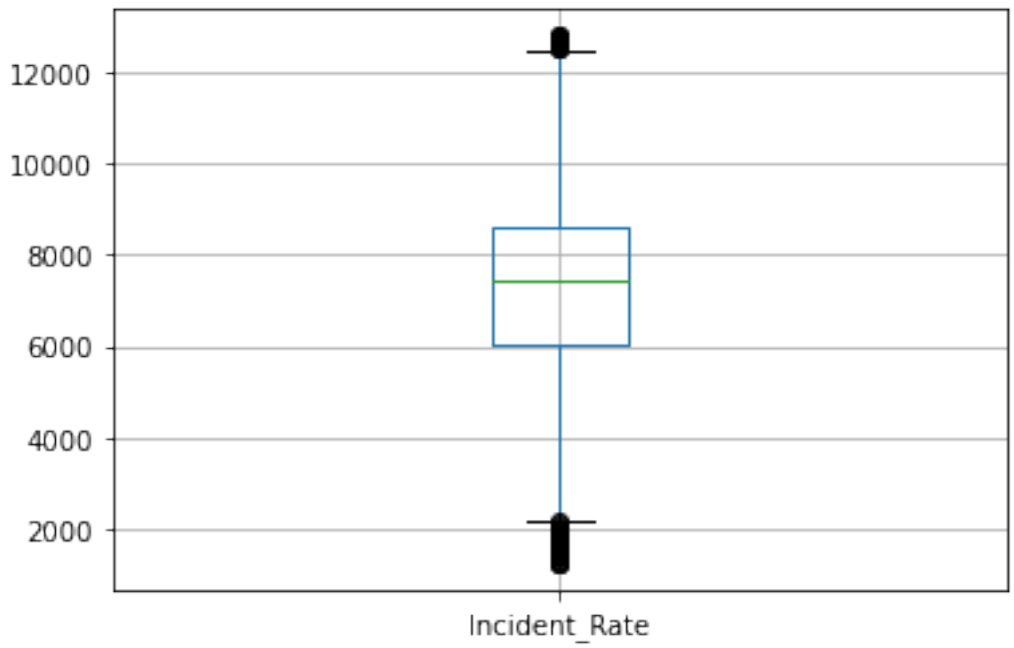
[9]:
```
df.boxplot(column=['Active'])
```

[9]: <AxesSubplot:>



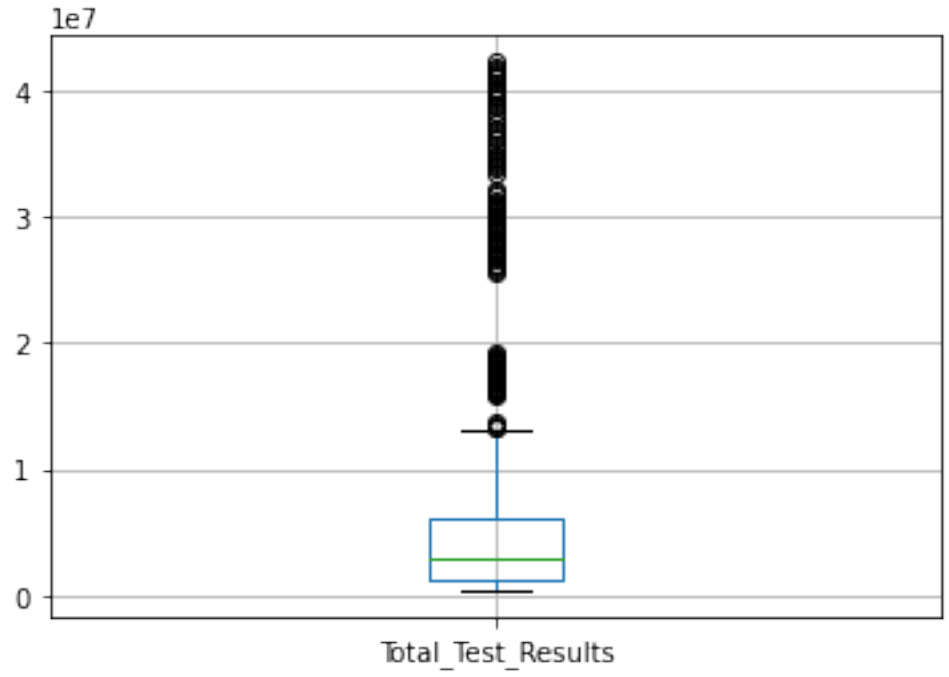
[10]:
```
df.boxplot(column=['Incident_Rate'])
```

[10]: <AxesSubplot:>



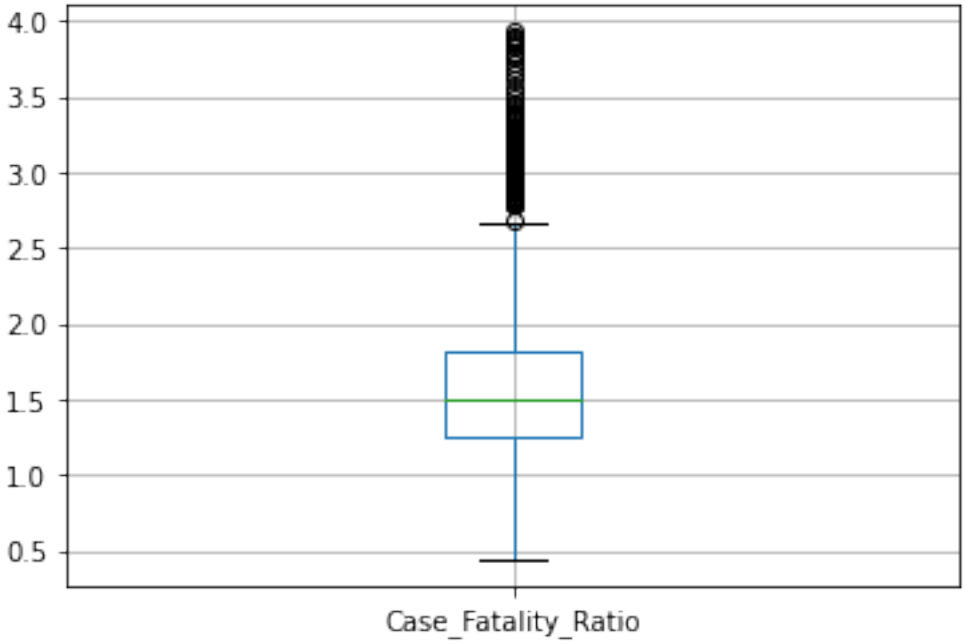
[11]:
```
df.boxplot(column=['Total_Test_Results'])
```

[11]: <AxesSubplot:>



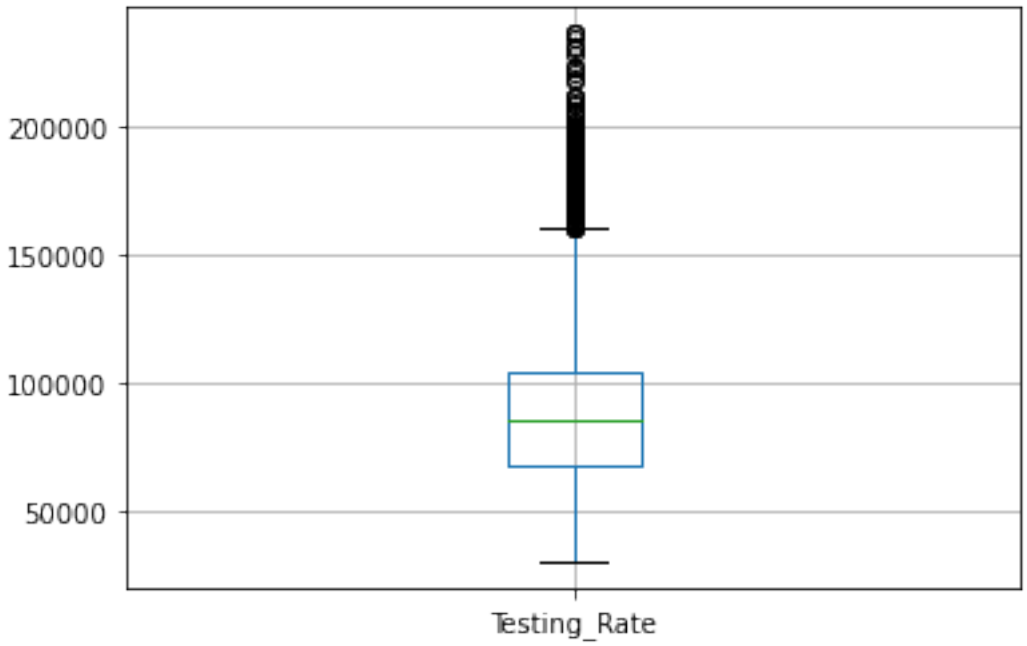
[12]:
```
df.boxplot(column=['Case_Fatality_Ratio'])
```

[12]: <AxesSubplot:>



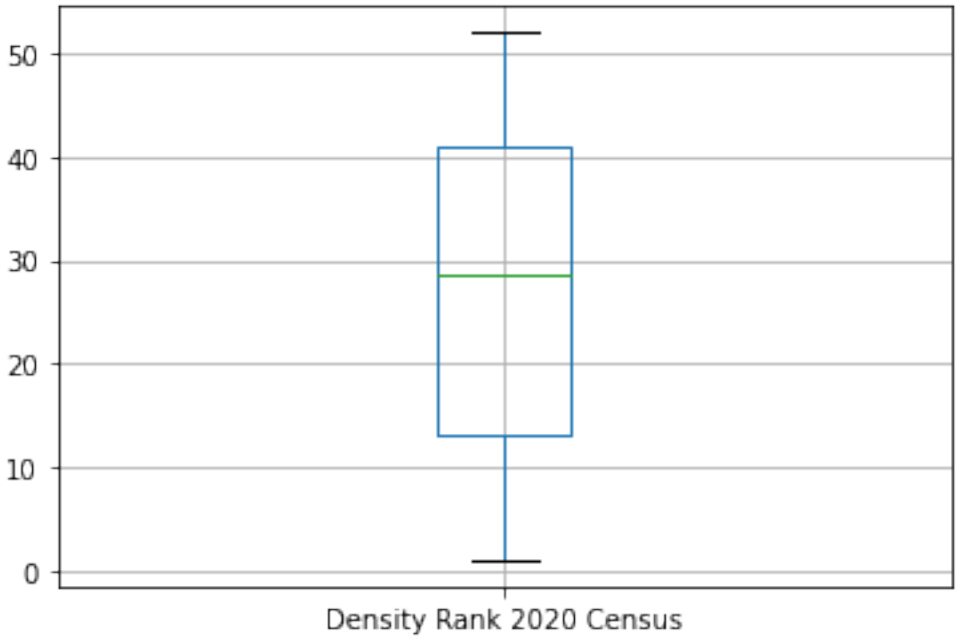
[13]:
```
df.boxplot(column=['Testing_Rate'])
```

[13]: <AxesSubplot:>



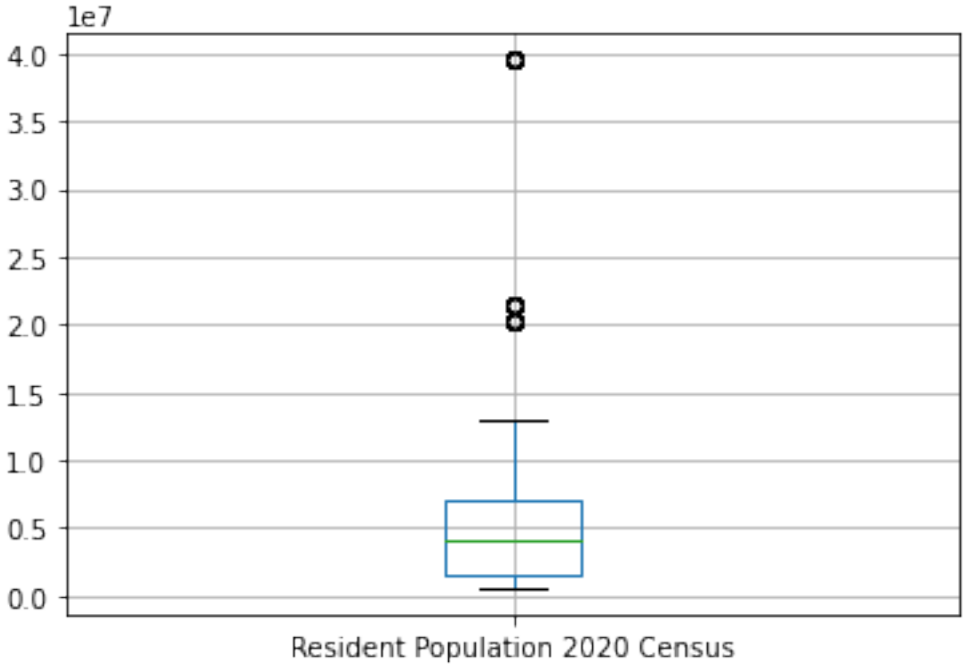
[14]:
```
df.boxplot(column=['Density Rank 2020 Census'])
```

[14]: <AxesSubplot:>



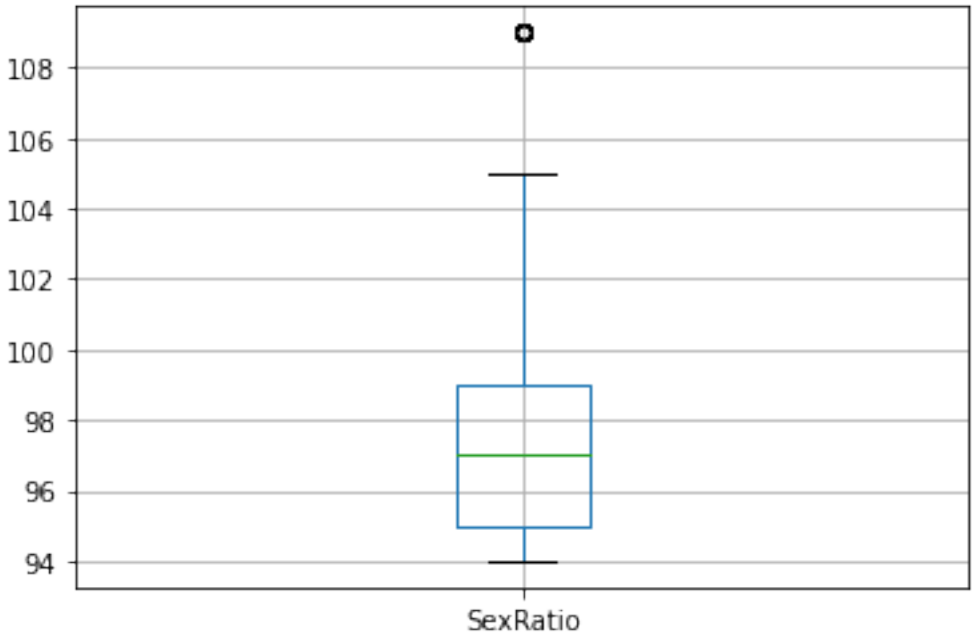
[15]:
```
df.boxplot(column=['Resident Population 2020 Census'])
```

[15]: <AxesSubplot:>



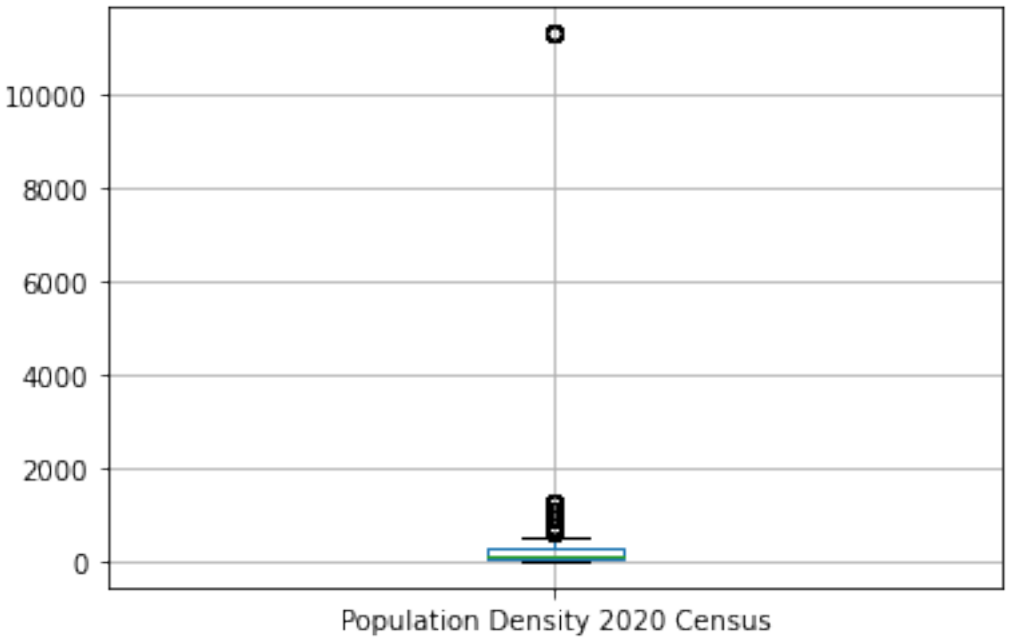
[16]:
```
df.boxplot(column=['SexRatio'])
```

[16]: <AxesSubplot:>



[17]:
```
df.boxplot(column='Population Density 2020 Census')
```

[17]: <AxesSubplot:>



From the above box plots, we can see ‘Lat’, ‘Long_’, ‘Resident Population 2020 Census’, ‘SexRatio’,

'Population Density 2020 Census' contain outliers, but because they are real data corresponding to Alaska and Hawaii, we are going to keep these outliers for our analysis.

0.1.2 Z-score normalization

Because the value differences between features are very large and PCA is affected by scale, we need to scale the data using Z-normalization. StandardScaler will standardize the dataset's features onto unit scale. This will help us reaching optimal performance of machine learning algorithms.

[150]:
```
print (df.columns.tolist())
```

```
['Day', 'State ID', 'State', 'Lat', 'Long_', 'Active', 'Incident_Rate',
'Total_Test_Results', 'Case_Fatality_Ratio', 'Testing_Rate', 'Resident
Population 2020 Census', 'Population Density 2020 Census', 'Density Rank 2020
Census', 'SexRatio', 'Confirmed', 'Deaths', 'Recovered']
```

[151]:
```
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
# select features to normalize
features=['Day','State ID','Lat','Long_','Active', 'Incident_Rate',␣
 ↪'Total_Test_Results', 'Case_Fatality_Ratio', 'Testing_Rate', 'Resident␣
 ↪Population 2020 Census', 'Population Density 2020 Census', 'Density Rank␣
 ↪2020 Census', 'SexRatio']
df_z= df.loc[:,features]
df_z.iloc[:,:]=std_scaler.fit_transform(df_z.iloc[:,:])
df_z.head()
```

[151]:

	Day	State ID	Lat	Long_	Active	Incident_Rate	\
0	-1.675247	-1.670726	-1.178670	0.304586	-0.200701	0.144028	
1	-1.675247	-1.601799	3.608919	-3.033032	-0.449116	-0.290315	
2	-1.675247	-1.532872	-0.946051	-0.945268	0.389194	0.088543	
3	-1.675247	-1.463945	-0.741727	0.025825	-0.482520	0.202252	
4	-1.675247	-1.395018	-0.552794	-1.365662	4.276989	-0.502599	

	Total_Test_Results	Case_Fatality_Ratio	Testing_Rate	\
0	-0.483568	-0.476902	-1.302217	
1	-0.569578	-1.800519	2.071905	
2	-0.007494	0.074416	-1.269412	
3	-0.456622	-0.031311	-0.559356	
4	4.023547	-0.792949	-0.177608	

	Resident Population 2020 Census	Population Density 2020 Census	\
0	-0.128626	-0.217091	
1	-0.754447	-0.276853	
2	0.181627	-0.239250	
3	-0.422184	-0.242302	
4	4.905194	-0.122779	

```
   Density Rank 2020 Census  SexRatio
0                  0.118788 -1.168679
1                  1.614954  3.492526
2                  0.509092  0.385056
3                  0.574143 -0.547185
4                 -0.922023  0.385056
```

0.1.3 Detecting missing values in each columns & data cleaning

[20]:
```
df.isnull().sum()
```

[20]:
```
Day                                0
State ID                           0
State                              0
Lat                                0
Long_                              0
Active                             0
Incident_Rate                      0
Total_Test_Results                 0
Case_Fatality_Ratio                0
Testing_Rate                       0
Resident Population 2020 Census    0
Population Density 2020 Census     0
Density Rank 2020 Census           0
SexRatio                           0
Confirmed                          0
Deaths                             0
Recovered                          0
dtype: int64
```

There is no missing values detected from training data

0.1.4 Detecting negative values in dataframe

[21]:
```
# select only numeric columns
newdf = df.select_dtypes(include=np.number)
```

[22]:
```
# exclude longitude
df1 = newdf.drop(['Long_'], axis=1)
(df1.values < 0).any()
sum(n < 0 for n in df1.values.flatten())
```

[22]:
```
0
```

There is no negative values detected from columns other than 'Long_'. It's normal to have negative longitude values, no further action needed to replace negative values from train data set.

0.2 [CM2] Representation learning

0.2.1 Using PCA

[23]:
```
from sklearn.decomposition import PCA
comp_num =len(features)
pca = PCA(n_components=comp_num)
pca.fit(df_z)
```

[23]:
```
PCA(n_components=13)
```

[24]:
```
pca.explained_variance_
```

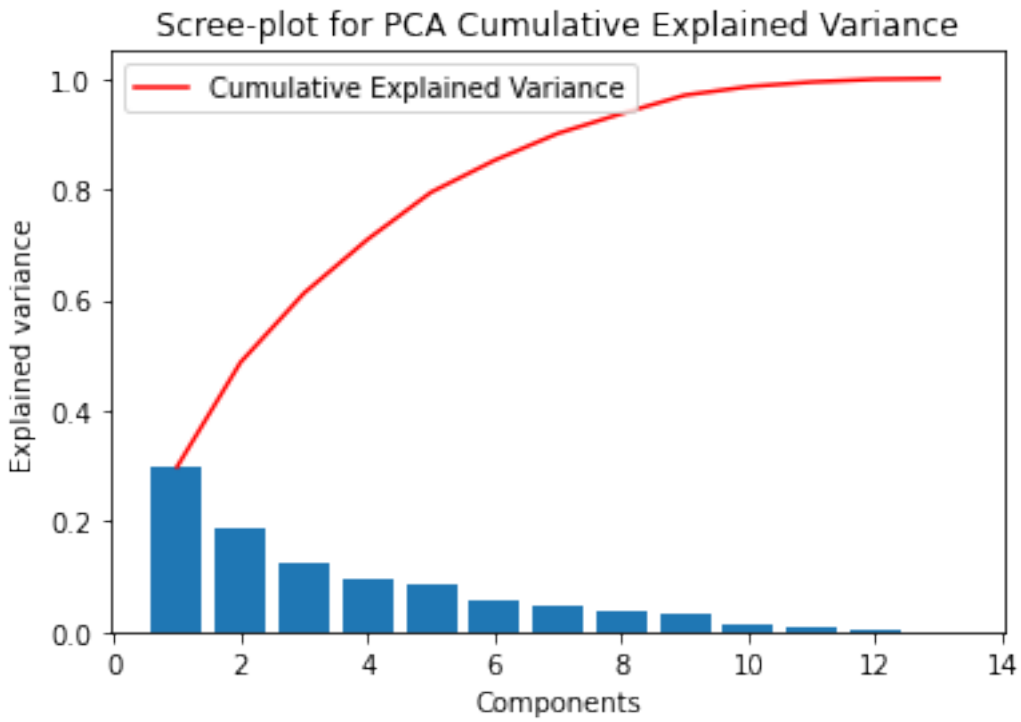
[24]:
```
array([3.87978553, 2.45659942, 1.62576285, 1.26350615, 1.10734858,
       0.75151186, 0.63429167, 0.46324711, 0.43394219, 0.20076863,
       0.10828781, 0.06562137, 0.01875395])
```

[25]:
```
from sklearn.decomposition import PCA
pcamodel = PCA(n_components=comp_num)
pca = pcamodel.fit_transform(df_z)
pca.shape
```

[25]:
```
(1380, 13)
```

[26]:
```
plt.bar(range(1,len(pcamodel.explained_variance_ratio_ )+1),pcamodel.
 ↪explained_variance_ratio_ )
plt.ylabel('Explained variance')
plt.xlabel('Components')
plt.title('Scree-plot for PCA Cumulative Explained Variance')
plt.plot(range(1,len(pcamodel.explained_variance_ratio_ )+1),
         np.cumsum(pcamodel.explained_variance_ratio_),
         c='red',
         label="Cumulative Explained Variance")
plt.legend(loc='upper left')
```

[26]:
```
<matplotlib.legend.Legend at 0x27102b40fd0>
```



From the above graph, the best number of reduced features to use to represent the data is around 6. Because we need to account for 90% of the variance in the original data set.

0.2.2 PCA dataframe

Constructing PCA dataframe for further analysis

[27]:
```
pcamodel2 = PCA(n_components=6)
pca6 = pcamodel2.fit_transform(df_z)
pca6_df=pd.DataFrame(data=pca6, columns=['PC1','PC2','PC3','PC4','PC5','PC6'])
```

[28]:
```
pca6_df.head()
```

[28]:

	PC1	PC2	PC3	PC4	PC5	PC6
0	0.356200	0.000313	-2.464842	0.925023	0.090046	-1.564070
1	-4.536635	1.973629	3.252485	3.144894	2.057861	-0.314367
2	-0.116704	1.338505	-1.693973	1.496746	0.461376	-1.262018
3	-0.297514	-0.161948	-1.555615	1.023983	0.297813	-1.551909
4	4.771793	6.098073	0.668480	1.273762	2.030359	-0.335811

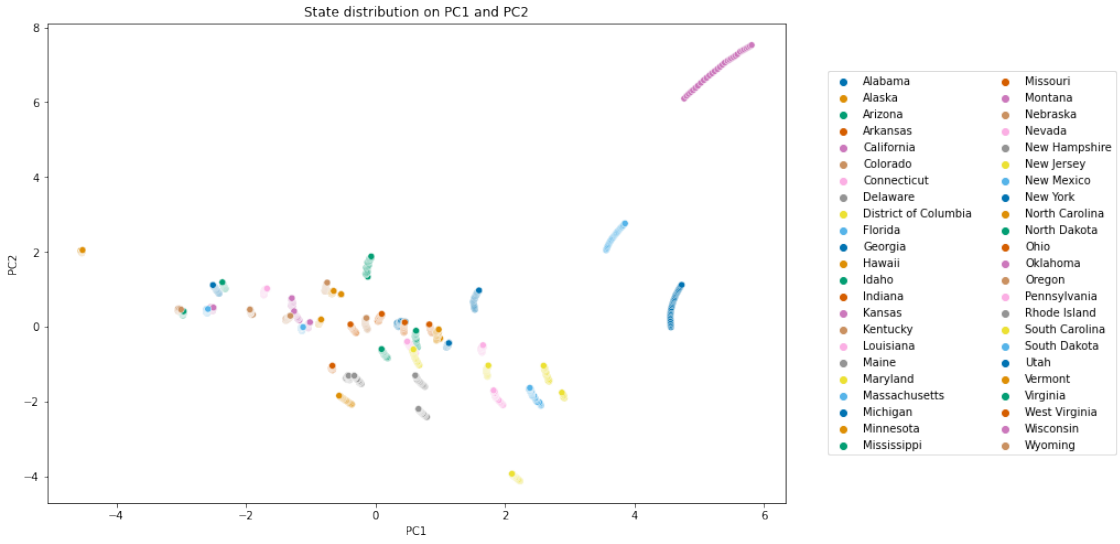
0.2.3 Scatter plot of PCA1 and PCA2

Plot the datapoints, or a useful subset of them (select out one day per state? or use color by State?) on the first two PCA and LDA feature vectors. For this question we

decided to select states and plot them on the first two PCA vectors to visualize the effectiveness of PCA classifiier.

[29]:
```
# Combine first two PCA columns and original data frame
df_combi = pd.concat( [df['State'],pca6_df[['PC1','PC2']]],axis=1)
df_combi.head()

# Change seaborn plot size
import matplotlib.pyplot as plt
fig = plt.gcf()
fig.set_size_inches(12, 8)

plt.title('State distribution on PC1 and PC2')
g=sns.scatterplot(data=df_combi, x="PC1", y="PC2", hue="State",␣
 ↪palette="colorblind")
g.legend(loc='center left', bbox_to_anchor=(1.05,0.5), ncol=2)
```

[29]: <matplotlib.legend.Legend at 0x27102bdb550>



From the above scatter plot, we can observe that the first two components from the PCA classifier is capable for data separation by 'State'. The upper right claster corresponding to data from California and left most claster corresponding to data from Alaska. Data from other states are closer on the graph but there is no obvious mixing between states. This indicates that PCA classifer achieved a good separation between data points with the given data.

0.2.4 Construct LDA dataframe for further analysis

[30]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda= LDA(n_components=1)
features=['Day','State ID','Active','Lat','Long_', 'Incident_Rate',␣
 ↪'Total_Test_Results', 'Case_Fatality_Ratio', 'Testing_Rate', 'Resident␣
 ↪Population 2020 Census', 'Population Density 2020 Census', 'Density Rank␣
 ↪2020 Census', 'SexRatio']
X= df_z.loc[:,features]
```

[31]:
```
# set y1= Confirmed y2= deaths y3= Recovered
y1=df.loc[:,'Confirmed']
y2=df.loc[:,'Deaths']
y3=df.loc[:,'Recovered']
```

[32]:
```
X_r1 = lda.fit(X, y1).transform(X)
X_r2 = lda.fit(X, y2).transform(X)
X_r3 = lda.fit(X, y3).transform(X)
```

[33]:
```
# Constructing LDA dataframe
X_LDA= np.column_stack([X_r1, X_r2,X_r3])
LDA_df=pd.DataFrame(data=X_LDA,␣
 ↪columns=['LDA_Confirmed','LDA_Deaths','LDA_Recovered'])
# LDA_df.head()

# Combine LDA columns and original data frame
df_combi2 = pd.concat([df['State'],LDA_df,y1,y2,y3],axis=1)
df_combi2.head()
```

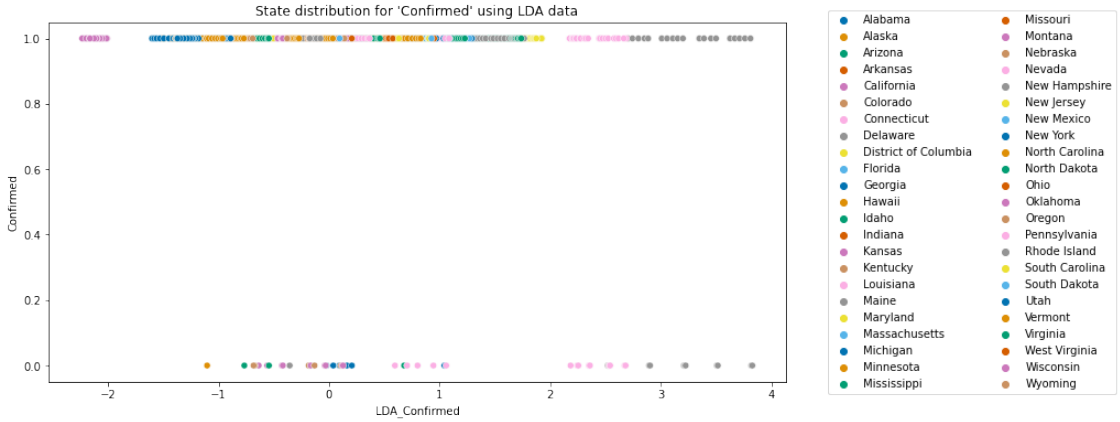
[33]:

	State	LDA_Confirmed	LDA_Deaths	LDA_Recovered	Confirmed	Deaths	Recovered
0	Alabama	0.049435	-1.859826	1.300328	True	False	False
1	Alaska	-0.068422	2.694138	1.868732	True	True	False
2	Arizona	-0.386825	0.182055	1.034923	True	True	True
3	Arkansas	0.182651	-0.844006	0.839964	True	True	True
4	California	-2.189425	-1.674409	3.923431	True	True	False

0.2.5 Plot the datapoints by States on LDA feature vectors

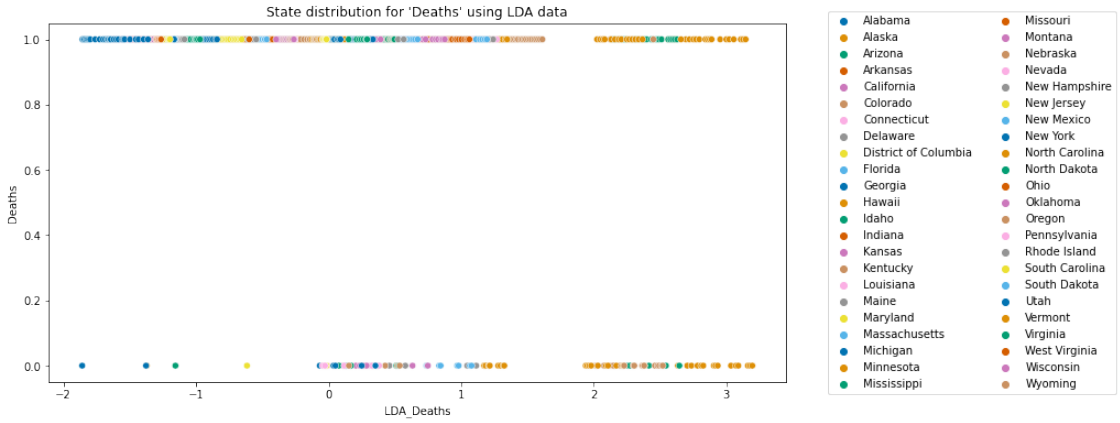
[34]:
```
# Change seaborn plot size
import matplotlib.pyplot as plt
fig = plt.gcf()
fig.set_size_inches(12, 6)

plt.title('State distribution for \'Confirmed\' using LDA data')
g=sns.scatterplot(data=df_combi2, x="LDA_Confirmed", y="Confirmed",␣
↪hue="State", palette="colorblind")
g.legend(loc='center left', bbox_to_anchor=(1.05,0.5), ncol=2)
```

[34]:
```
<matplotlib.legend.Legend at 0x27102df4eb0>
```



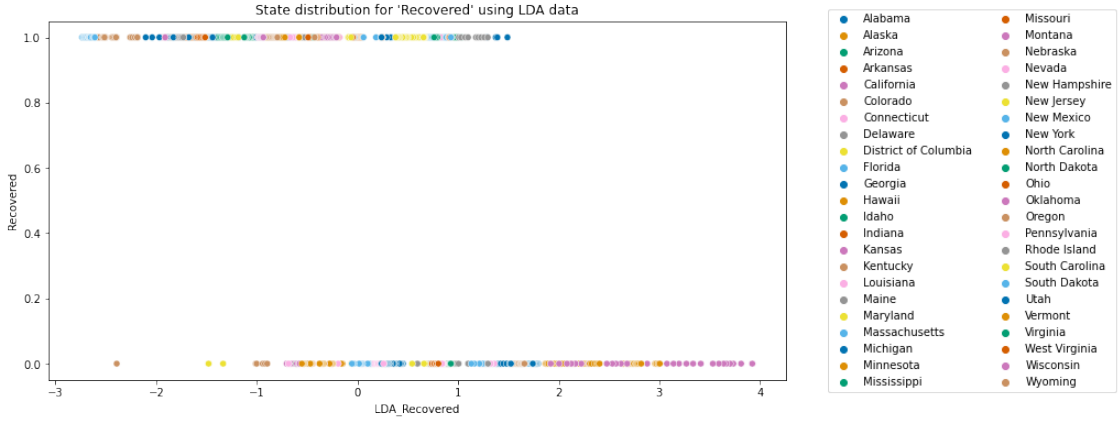
[35]:
```
fig = plt.gcf()
fig.set_size_inches(12, 6)
plt.title('State distribution for \'Deaths\' using LDA data')
g=sns.scatterplot(data=df_combi2, x="LDA_Deaths", y="Deaths", hue="State",␣
↪palette="colorblind")
g.legend(loc='center left', bbox_to_anchor=(1.05,0.5), ncol=2)
```

[35]:
```
<matplotlib.legend.Legend at 0x27102efe970>
```



[36]:
```
fig = plt.gcf()
fig.set_size_inches(12, 6)
plt.title('State distribution for \'Recovered\' using LDA data')
g=sns.scatterplot(data=df_combi2, x="LDA_Recovered", y="Recovered",␣
 ↪hue="State", palette="colorblind")
g.legend(loc='center left', bbox_to_anchor=(1.05,0.5), ncol=2)
```

[36]: <matplotlib.legend.Legend at 0x271033472b0>



Due to limitations of sklearn library and given data. Its not able to produce the LDA with n_components =2. Therefore it's hard to determine if LDA achieved a good separation for the given data or if LDA method provides better results for one label more than the others. It's worth mentioning that 'Recovered' is the most balanced data as shown in the above graph.

1 Creating datasets to be used for label prediction in the following questions

[37]:
```
#forming the three datasets to be used for classification for the three labels␣
 ↪respectively
cleaned_df_1= pd.concat([df_z,df['Confirmed']],axis=1)
cleaned_df_2= pd.concat([df_z,df['Deaths']],axis=1)
cleaned_df_3= pd.concat([df_z,df['Recovered']],axis=1)
```

[38]:
```
df = df.reset_index()
```

[39]:
```
#splitting into X,y to be used for further analysis
X1= cleaned_df_1.loc[ : , cleaned_df_1.columns != 'Confirmed']
y1=cleaned_df_1['Confirmed']
X2= cleaned_df_2.loc[ : , cleaned_df_2.columns != 'Deaths']
y2=cleaned_df_2['Deaths']
X3= cleaned_df_3.loc[ : , cleaned_df_3.columns != 'Recovered']
y3=cleaned_df_3['Recovered']
```

1.1 [CM3] Decision Tree Classifier [1]

[40]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import plot_tree
from sklearn.model_selection import KFold
```

Using Decision tree for all three labels for max_depth=None

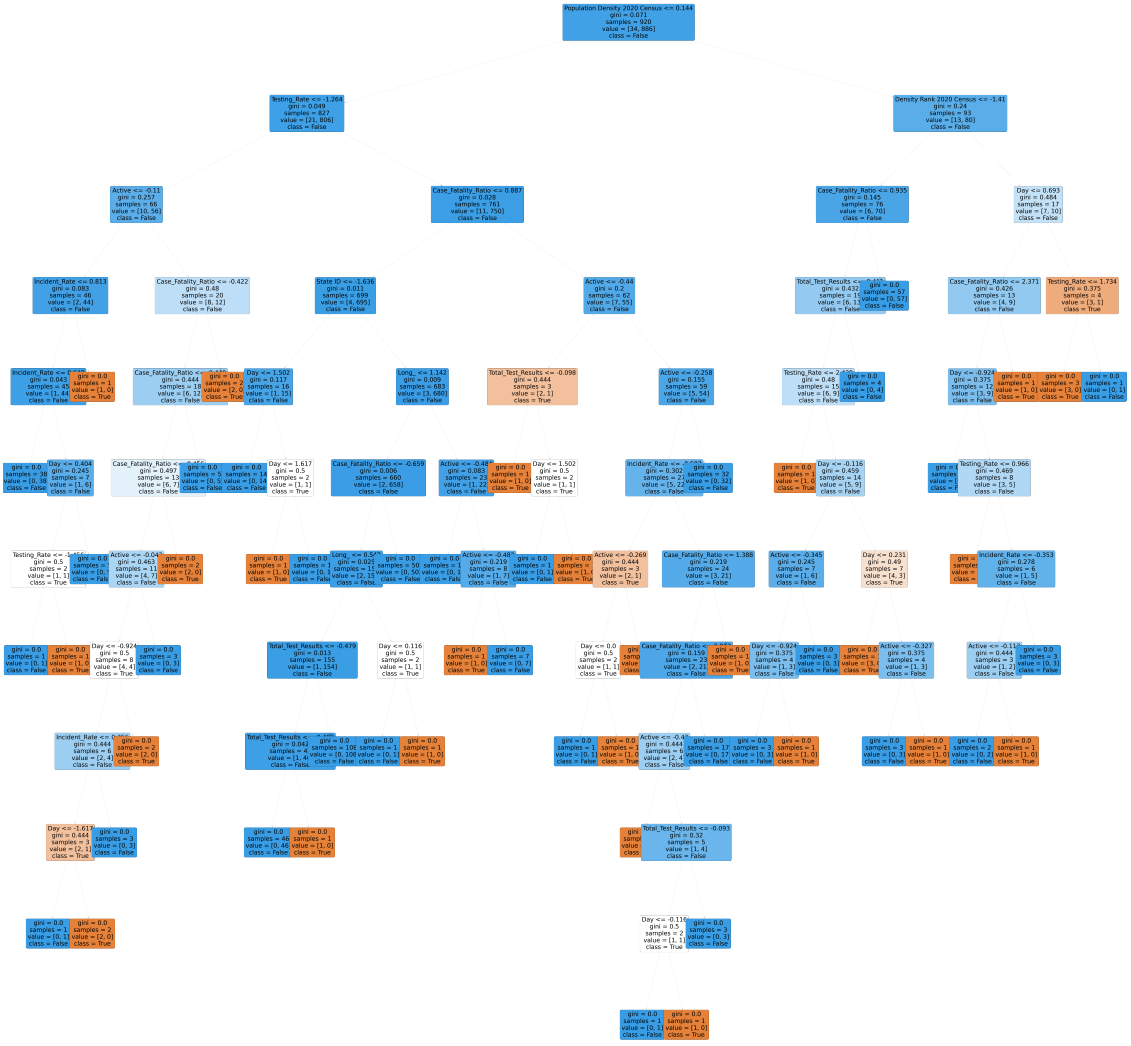
[95]:
```
def DecisionTree_Nonedepth(X,y, nsplits, label_to_predict):

    kf = KFold(n_splits=nsplits, random_state=98, shuffle=True)
    model = DecisionTreeClassifier()
    acc_score = []
    for train_index , test_index in kf.split(X):
        X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]

        model.fit(X_train,y_train)
        y_pred = model.predict(X_test)

        acc = metrics.accuracy_score(y_pred , y_test)
        acc_score.append(acc)

    avg_acc_score = sum(acc_score)/nsplits
    print("Accuracy: ", avg_acc_score)
```

```
    plot1 = plt.figure(2, figsize=(200,200))
    plot_tree(model, feature_names = X.columns, class_names = ['True','False'],␣
↪filled = True, rounded = True, fontsize=60)
    print('Decision tree for '+ '\'' +label_to_predict+ '\'' + ' with splits =␣
↪'+str(nsplits) + ' max_depth =None')

plt.show
```

[95]: <function matplotlib.pyplot.show(close=None, block=None)>

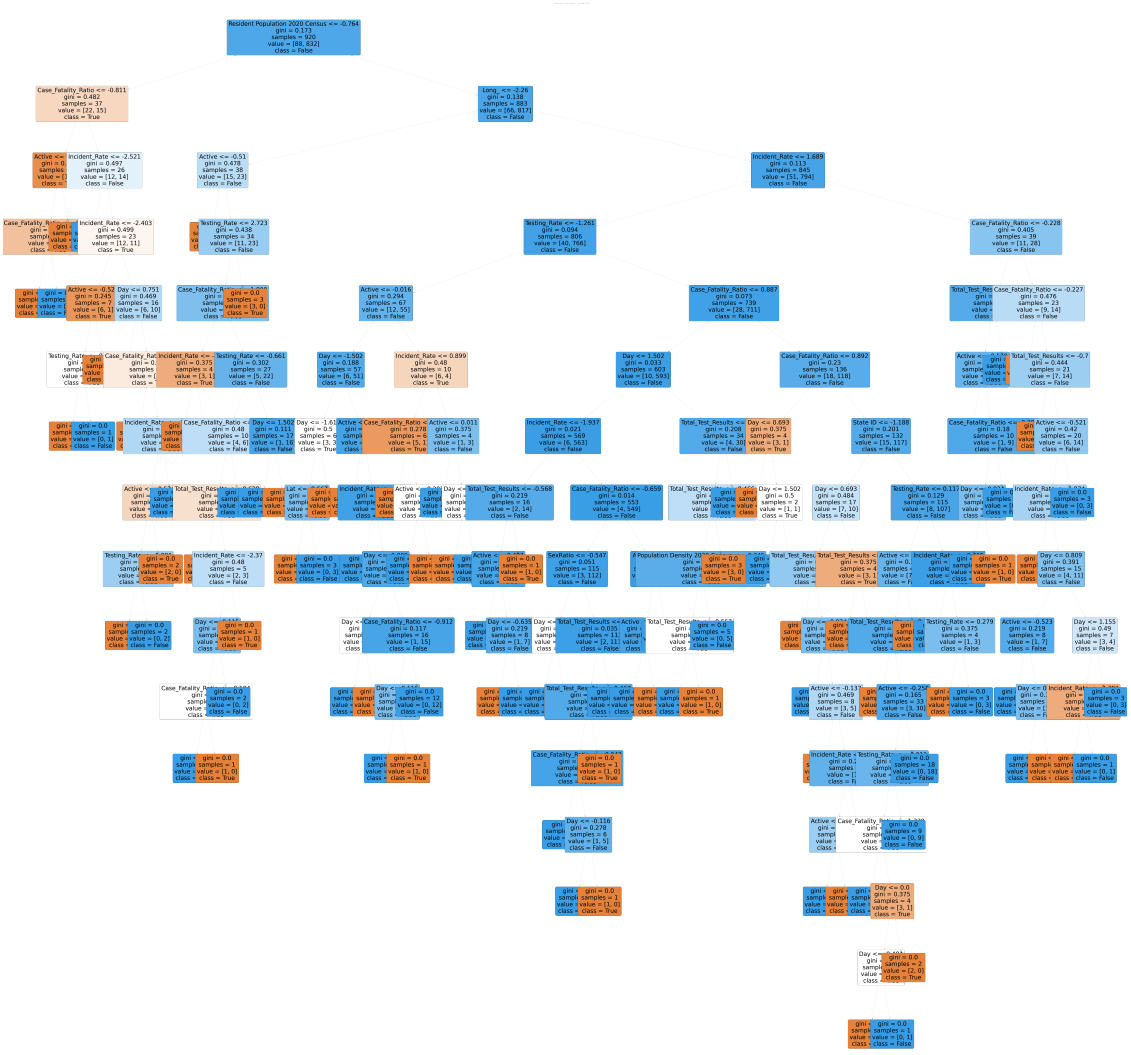
[96]:
```
DecisionTree_Nonedepth(X1,y1,3,'Confirmed')
```

```
Accuracy:  0.9398550724637681
Decision tree for 'Confirmed' with splits = 3 max_depth =None
```



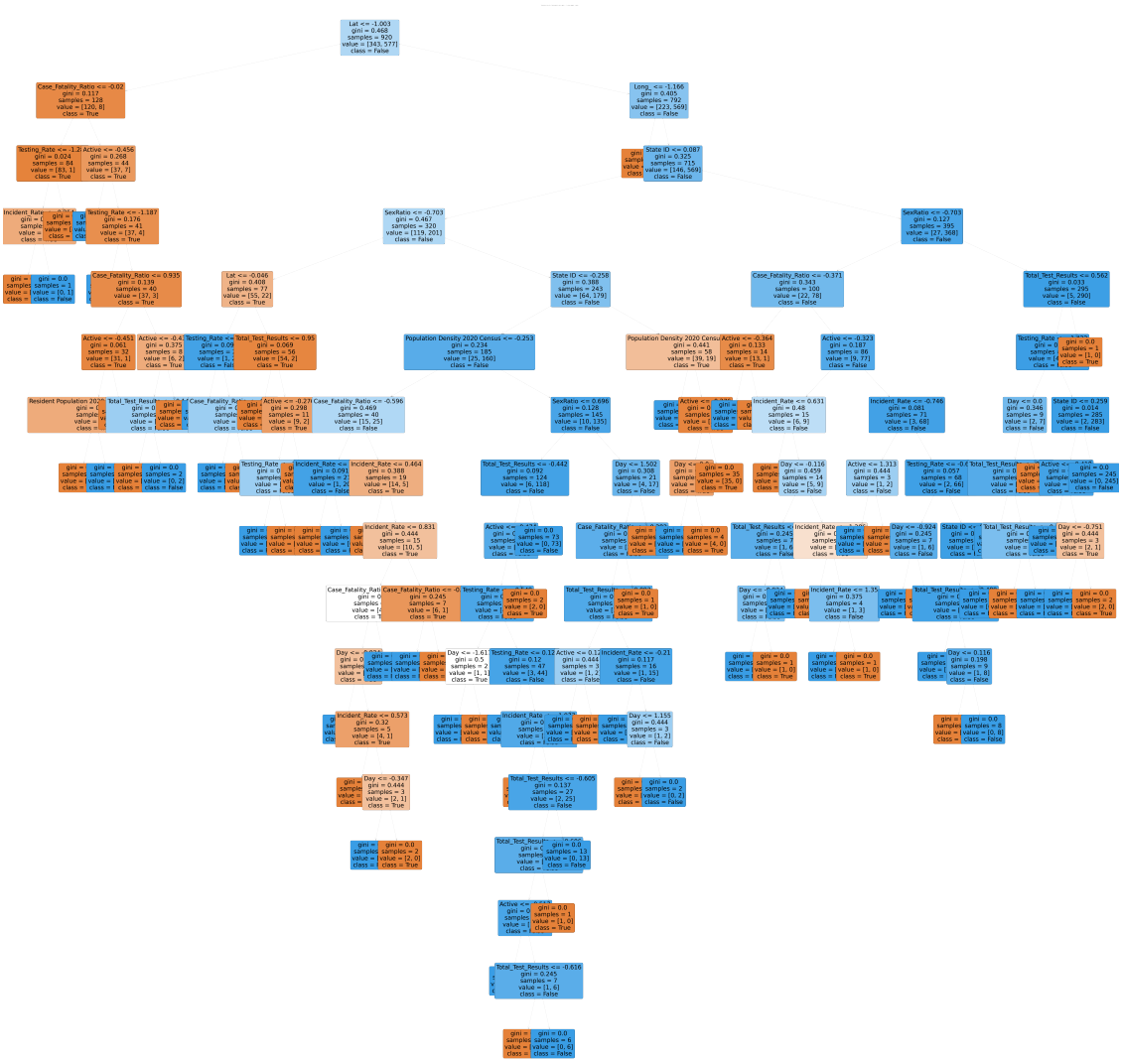
[43]:
```
DecisionTree_Nonedepth(X2,y2,3,'Deaths')
```

```
Accuracy:  0.8724637681159421
```



[44]:
```
DecisionTree_Nonedepth(X3,y3,3,'Recovered')
```

```
Accuracy:  0.9108695652173914
```



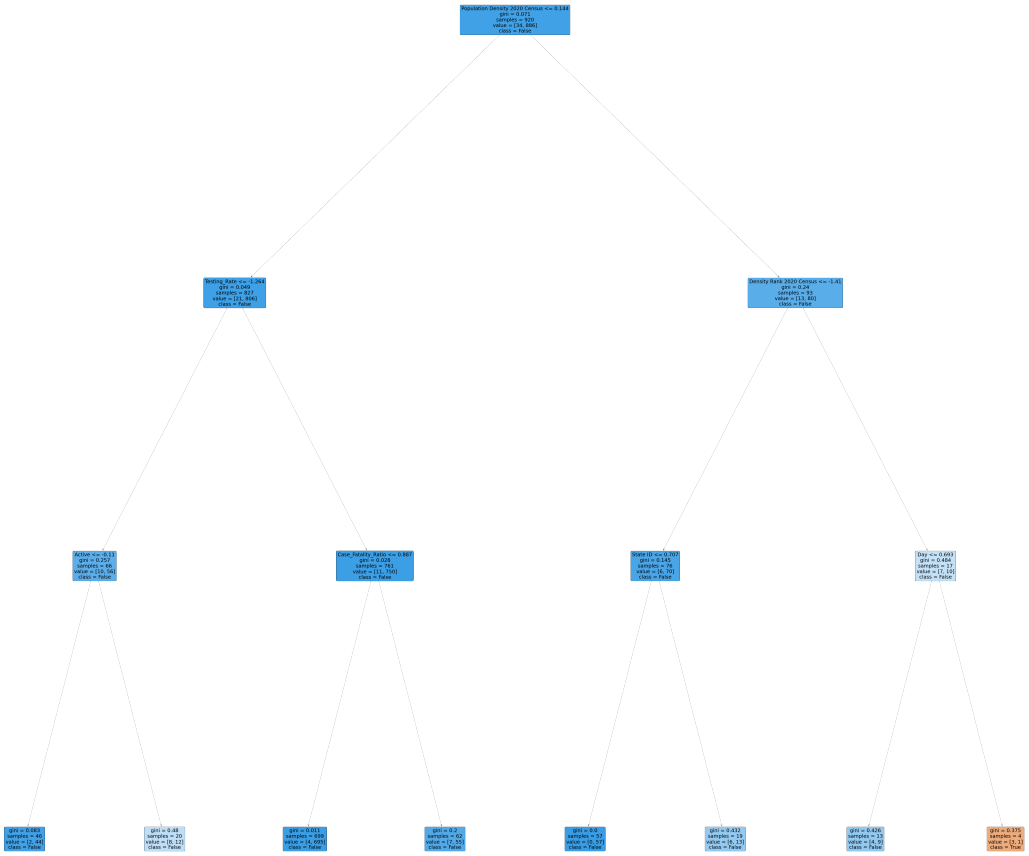
Using Decision tree for all three labels for max_depth=3,5,10

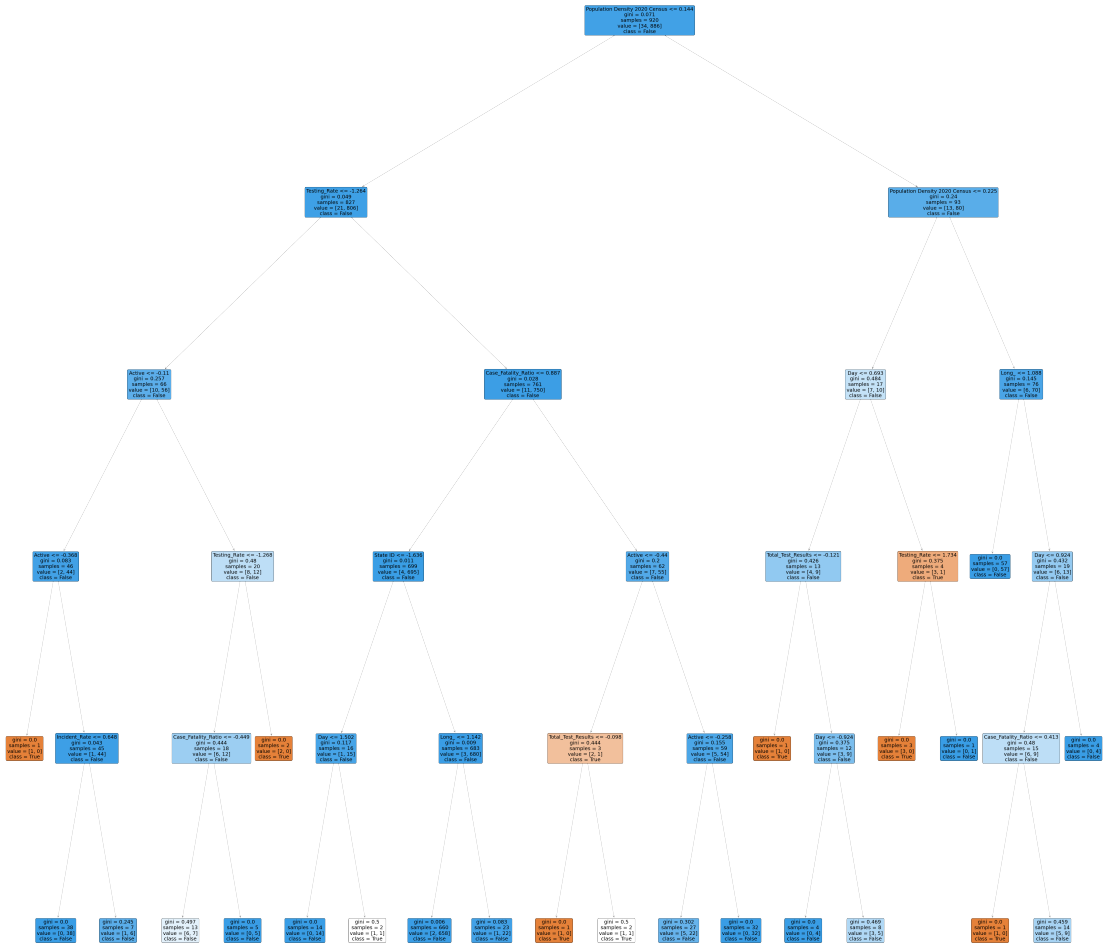
[45]:
```
def DecisionTree_depthspec(X,y, nsplits, label_to_predict):
    depth= [3,5,10]
    acc_array= []
    d_array=[]

    for d in depth:
        kf = KFold(n_splits=nsplits, random_state=98, shuffle=True)
        model = DecisionTreeClassifier(max_depth=d)
        acc_score = []
        for train_index , test_index in kf.split(X):
            X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
            y_train , y_test = y[train_index] , y[test_index]
```

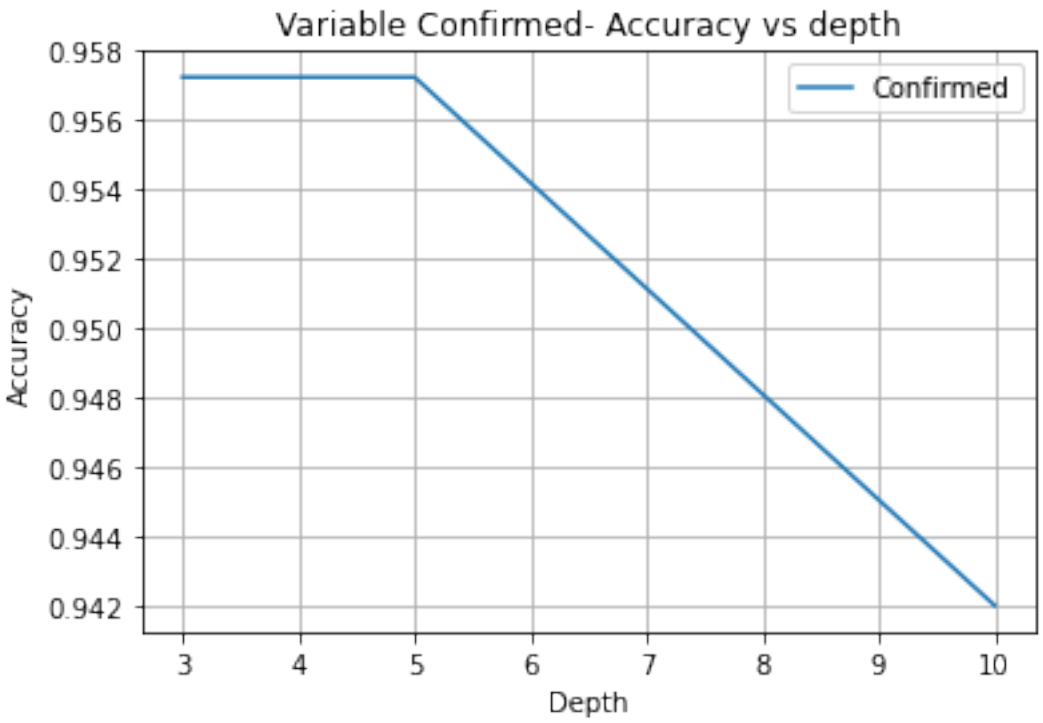
```
            model.fit(X_train,y_train)
            y_pred = model.predict(X_test)

            acc = metrics.accuracy_score(y_pred , y_test)
            acc_score.append(acc)

        avg_acc_score = sum(acc_score)/nsplits
        acc_array.append(avg_acc_score)
        d_array.append(d)

        plot1 = plt.figure(depth.index(d)+1, figsize=(100,100))
        plot_tree(model, feature_names = X.columns, class_names =␣
↪['True','False'], filled = True, rounded = True, fontsize=25)
        figname= "Tree_visualization- "+ label_to_predict + "depth: " + str(d)
        plt.savefig(figname, format='png')



    print("Variable {}- Accuracy for different depth: {}".
↪format(label_to_predict,acc_array))
    print("different d values:",d_array)

    plot2 = plt.figure(len(depth)+1)
    plt.plot(d_array,acc_array,label=label_to_predict)
    plt.title( "Variable {}- Accuracy vs depth".format(label_to_predict))
    plt.xlabel("Depth")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)




    plt.show
```

[46]:
```
DecisionTree_depthspec(X1,y1,3,'Confirmed')
```

```
Variable Confirmed- Accuracy for different depth: [0.9572463768115943,
0.9572463768115943, 0.9420289855072465]
different d values: [3, 5, 10]
```

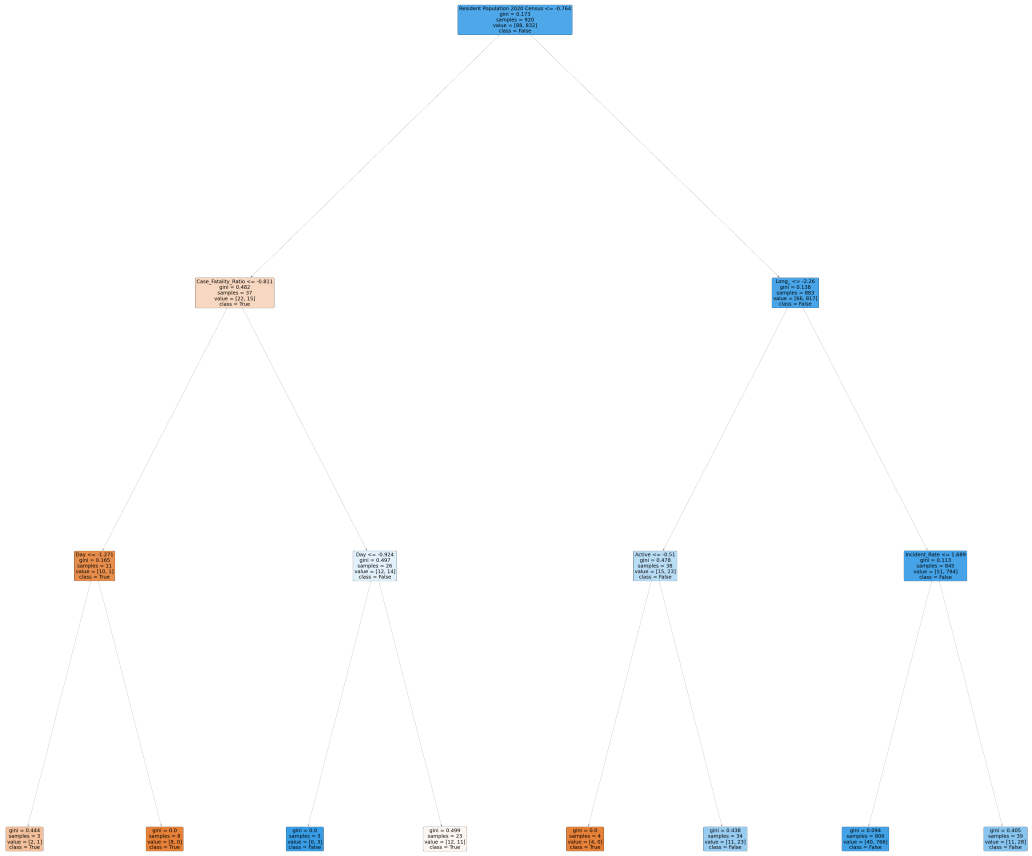

Population Density 2020 Census <= 0.148
gini = 0.071
samples = 100
value = [34, 666]
class = False
Testing Rate <= -1.264
gini = 0.049
samples = 97
value = [33, 651]
class = False
Density Rank 2020 Census <= -1.41
gini = 0.24
samples = 93
value = [33, 15]
class = False
Active <= -0.11
gini = 0.257
samples = 66
value = [29, 96]
class = False
Case_Fatality_Ratio <= 0.887
gini = 0.028
samples = 31
value = [4, 555]
class = False
State ID <= 0.707
gini = 0.145
samples = 76
value = [6, 75]
class = False
Day <= 0.693
gini = 0.484
samples = 17
value = [7, 10]
class = False
gini = 0.469
samples = 46
value = [2, 44]
class = False
gini = 0.48
samples = 20
value = [8, 12]
class = False
gini = 0.011
samples = 556
value = [4, 699]
class = False
gini = 0.2
samples = 62
value = [7, 55]
class = False
gini = 0.1
samples = 57
value = [5, 57]
class = False
gini = 0.432
samples = 19
value = [6, 13]
class = False
gini = 0.426
samples = 15
value = [4, 9]
class = False
gini = 0.375
samples = 4
value = [3, 1]
class = True

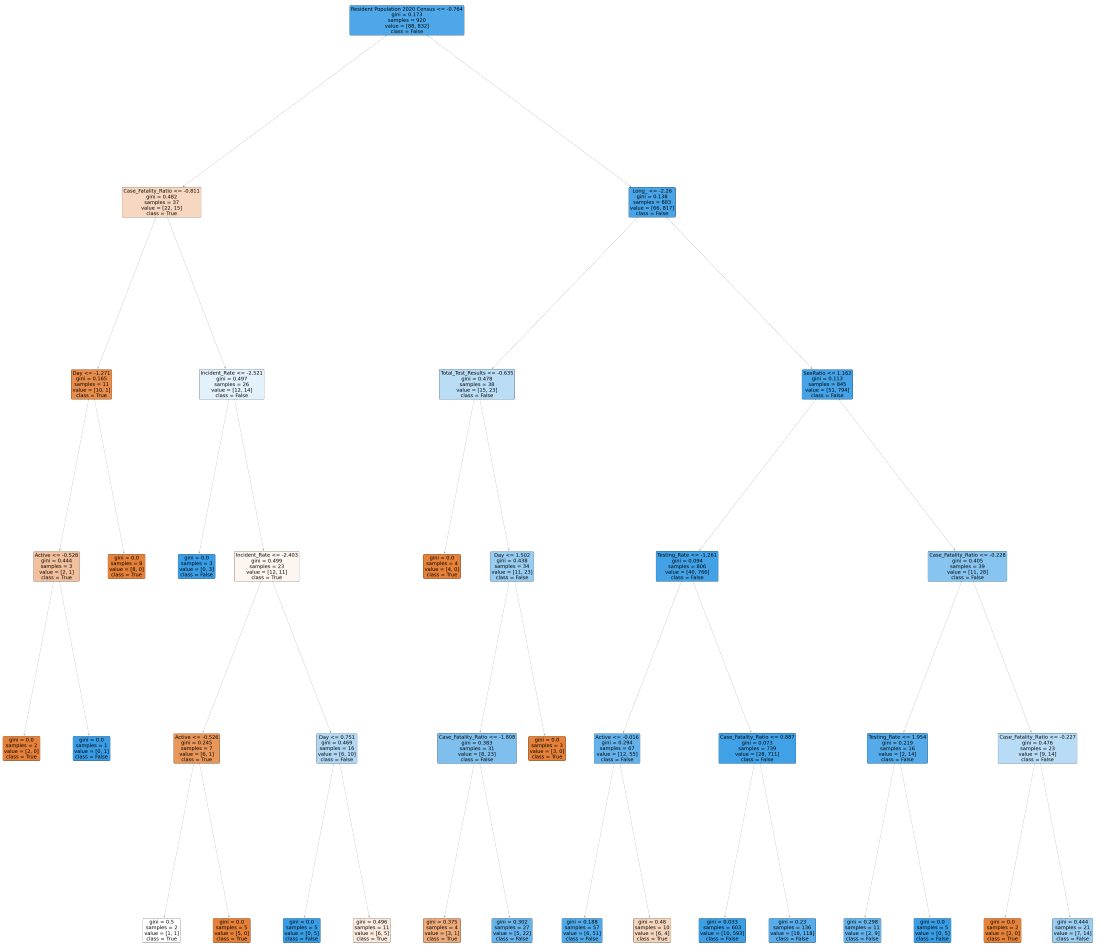

Population Density 2020 Census <= 0.148
gini = 0.071
samples = 920
value = [54, 886]
class = False
Testing_Ratio <= -1.264
gini = 0.049
samples = 827
value = [21, 806]
class = False
Population Density 2020 Census <= 0.225
gini = 0.24
samples = 93
value = [13, 80]
class = False
Active <= -0.11
gini = 0.257
samples = 66
value = [10, 56]
class = False
Case_Fatality_Ratio <= 0.897
gini = 0.029
samples = 761
value = [11, 750]
class = False
Day <= 0.693
gini = 0.486
samples = 17
value = [7, 10]
class = False
Long_ <= 1.099
gini = 0.145
samples = 76
value = [6, 70]
class = False
Active <= -0.369
gini = 0.065
samples = 46
value = [2, 44]
class = False
Testing_Ratio <= -1.260
gini = 0.48
samples = 20
value = [8, 12]
class = False
State ID <= -1.630
gini = 0.013
samples = 699
value = [4, 695]
class = False
Active <= -0.48
gini = 0.2
samples = 62
value = [7, 55]
class = False
Total_Test_Results <= -0.121
gini = 0.426
samples = 13
value = [4, 9]
class = False
Testing_Ratio <= 1.734
gini = 0.375
samples = 4
value = [3, 1]
class = True
gini = 0.0
samples = 57
value = [0, 57]
class = False
Day <= 0.924
gini = 0.411
samples = 19
value = [6, 13]
class = False
gini = 0.0
samples = 8
value = [0, 8]
class = False
Incident_Rate <= 0.648
gini = 0.043
samples = 45
value = [1, 44]
class = False
Case_Fatality_Ratio <= -0.449
gini = 0.444
samples = 18
value = [6, 12]
class = False
gini = 0.0
samples = 2
value = [2, 0]
class = True
Day <= 1.502
gini = 0.117
samples = 16
value = [1, 15]
class = False
Long_ <= 1.142
gini = 0.009
samples = 683
value = [3, 680]
class = False
Total_Test_Results <= -0.098
gini = 0.444
samples = 3
value = [2, 1]
class = True
Active <= -0.258
gini = 0.355
samples = 59
value = [5, 54]
class = False
gini = 0.0
samples = 1
value = [1, 0]
class = True
Day <= -0.924
gini = 0.375
samples = 12
value = [3, 9]
class = False
gini = 0.0
samples = 3
value = [3, 0]
class = True
gini = 0.0
samples = 1
value = [0, 1]
class = False
Case_Fatality_Ratio <= 0.413
gini = 0.48
samples = 15
value = [6, 9]
class = False
gini = 0.0
samples = 4
value = [0, 4]
class = False
gini = 0.0
samples = 38
value = [0, 38]
class = False
gini = 0.245
samples = 7
value = [1, 6]
class = False
gini = 0.497
samples = 13
value = [6, 7]
class = False
gini = 0.0
samples = 5
value = [0, 5]
class = False
gini = 0.0
samples = 14
value = [0, 14]
class = False
gini = 0.5
samples = 2
value = [1, 1]
class = True
gini = 0.066
samples = 660
value = [2, 658]
class = False
gini = 0.063
samples = 23
value = [1, 22]
class = False
gini = 0.0
samples = 1
value = [1, 0]
class = True
gini = 0.5
samples = 2
value = [1, 1]
class = True
gini = 0.302
samples = 27
value = [5, 22]
class = False
gini = 0.0
samples = 32
value = [0, 32]
class = False
gini = 0.0
samples = 4
value = [0, 4]
class = False
gini = 0.469
samples = 8
value = [3, 5]
class = False
gini = 0.0
samples = 1
value = [1, 0]
class = True
gini = 0.459
samples = 14
value = [5, 9]
class = False

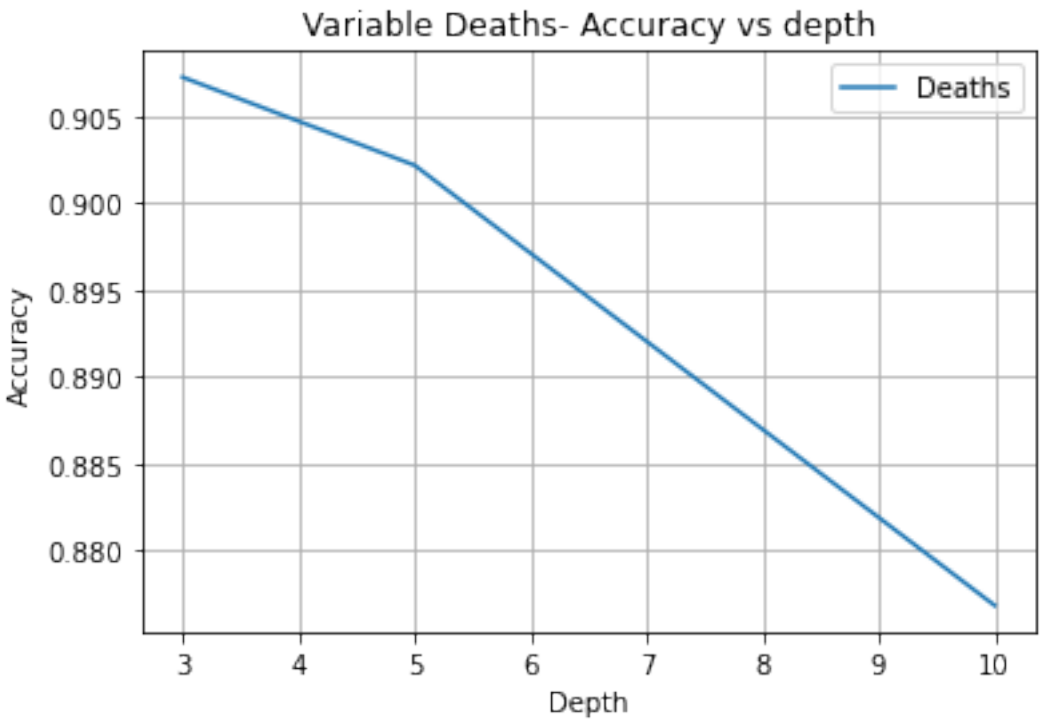


[47]:
```
DecisionTree_depthspec(X2,y2,3,'Deaths')
```

```
Variable Deaths- Accuracy for different depth: [0.9072463768115941,
0.9021739130434782, 0.8768115942028986]
different d values: [3, 5, 10]
```

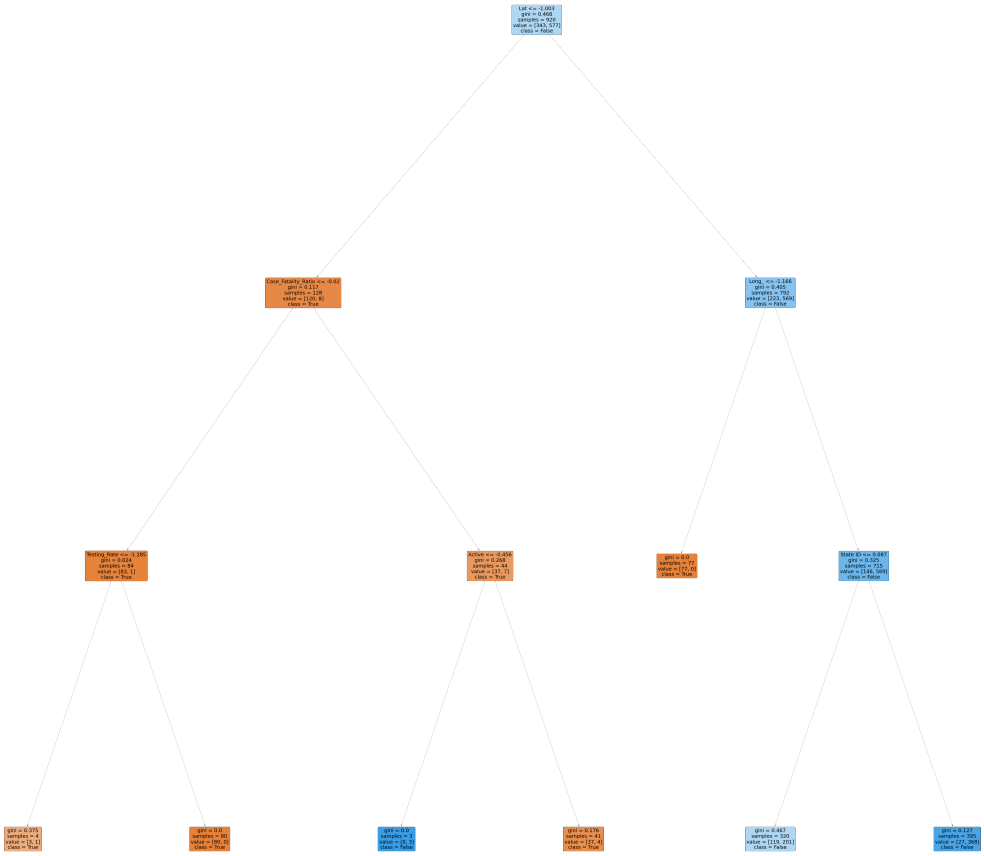

Resident Population 2020 Census <= -0.766
gini = 0.173
samples = 1000
value = [99, 932]
class = False
Case_Fatality_Ratio <= -0.811
gini = 0.482
samples = 37
value = [21, 16]
class = True
Long_ <= -3.26
gini = 0.136
samples = 963
value = [78, 885]
class = False
Day <= -1.277
gini = 0.165
samples = 11
value = [1, 10]
class = True
Day <= -0.924
gini = 0.497
samples = 26
value = [12, 14]
class = False
Active <= -0.51
gini = 0.478
samples = 25
value = [11, 14]
class = False
Incident_Rate <= 1.649
gini = 0.113
samples = 994
value = [77, 794]
class = False
gini = 0.444
samples = 3
value = [2, 1]
class = True
gini = 0.0
samples = 8
value = [8, 0]
class = True
gini = 0.0
samples = 3
value = [0, 3]
class = False
gini = 0.499
samples = 23
value = [12, 11]
class = True
gini = 0.0
samples = 4
value = [4, 0]
class = True
gini = 0.438
samples = 21
value = [7, 14]
class = False
gini = 0.094
samples = 906
value = [45, 766]
class = False
gini = 0.495
samples = 88
value = [32, 56]
class = False

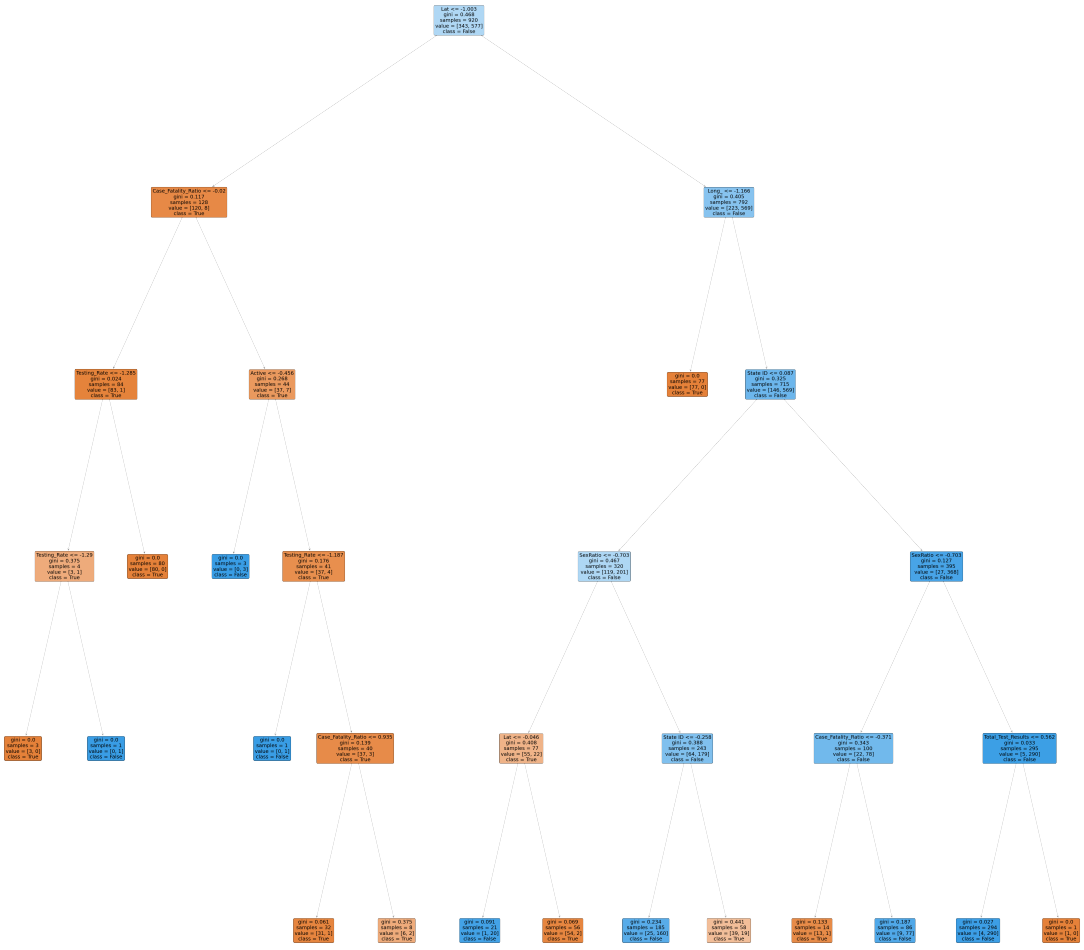

Resident Population 2020 Census <= -0.764
gini = 0.173
samples = 920
value = [95, 832]
class = False
Case_Fatality_Ratio <= -0.811
gini = 0.482
samples = 37
value = [22, 15]
class = True
Long_ <= -2.2
gini = 0.138
samples = 883
value = [69, 817]
class = False
Day <= -1.271
gini = 0.345
samples = 11
value = [10, 1]
class = True
Incident_Rate <= -2.521
gini = 0.497
samples = 26
value = [12, 14]
class = False
Total_Test_Results <= -0.635
gini = 0.476
samples = 38
value = [15, 23]
class = False
SexRatio <= 1.163
gini = 0.113
samples = 845
value = [51, 794]
class = False
Active <= -0.526
gini = 0.444
samples = 3
value = [2, 1]
class = True
gini = 0.0
samples = 8
value = [8, 0]
class = True
gini = 0.0
samples = 3
value = [3, 0]
class = True
Incident_Rate <= -2.403
gini = 0.499
samples = 23
value = [12, 11]
class = True
gini = 0.0
samples = 4
value = [0, 4]
class = False
Day <= 1.502
gini = 0.438
samples = 34
value = [11, 23]
class = False
Testing_Rate <= -1.201
gini = 0.094
samples = 806
value = [40, 766]
class = False
Case_Fatality_Ratio <= -0.228
gini = 0.405
samples = 39
value = [11, 28]
class = False
gini = 0.0
samples = 2
value = [2, 0]
class = True
gini = 0.0
samples = 1
value = [0, 1]
class = False
Active <= -0.526
gini = 0.245
samples = 7
value = [6, 1]
class = True
Day <= 0.751
gini = 0.469
samples = 16
value = [9, 10]
class = False
Case_Fatality_Ratio <= -1.609
gini = 0.353
samples = 31
value = [9, 23]
class = False
gini = 0.0
samples = 3
value = [3, 0]
class = True
Active <= -0.016
gini = 0.294
samples = 67
value = [12, 55]
class = False
Case_Fatality_Ratio <= 0.087
gini = 0.073
samples = 739
value = [28, 711]
class = False
Testing_Rate <= 1.958
gini = 0.219
samples = 16
value = [2, 14]
class = False
Case_Fatality_Ratio <= -0.227
gini = 0.476
samples = 23
value = [9, 14]
class = False
gini = 0.5
samples = 2
value = [1, 1]
class = True
gini = 0.0
samples = 5
value = [5, 0]
class = True
gini = 0.0
samples = 5
value = [0, 5]
class = False
gini = 0.496
samples = 11
value = [5, 6]
class = True
gini = 0.375
samples = 4
value = [1, 3]
class = True
gini = 0.302
samples = 27
value = [7, 22]
class = False
gini = 0.188
samples = 57
value = [5, 52]
class = False
gini = 0.48
samples = 10
value = [6, 4]
class = True
gini = 0.033
samples = 603
value = [10, 593]
class = False
gini = 0.23
samples = 136
value = [18, 118]
class = False
gini = 0.298
samples = 11
value = [2, 9]
class = False
gini = 0.0
samples = 5
value = [0, 5]
class = False
gini = 0.0
samples = 2
value = [2, 0]
class = True
gini = 0.444
samples = 21
value = [7, 14]
class = False

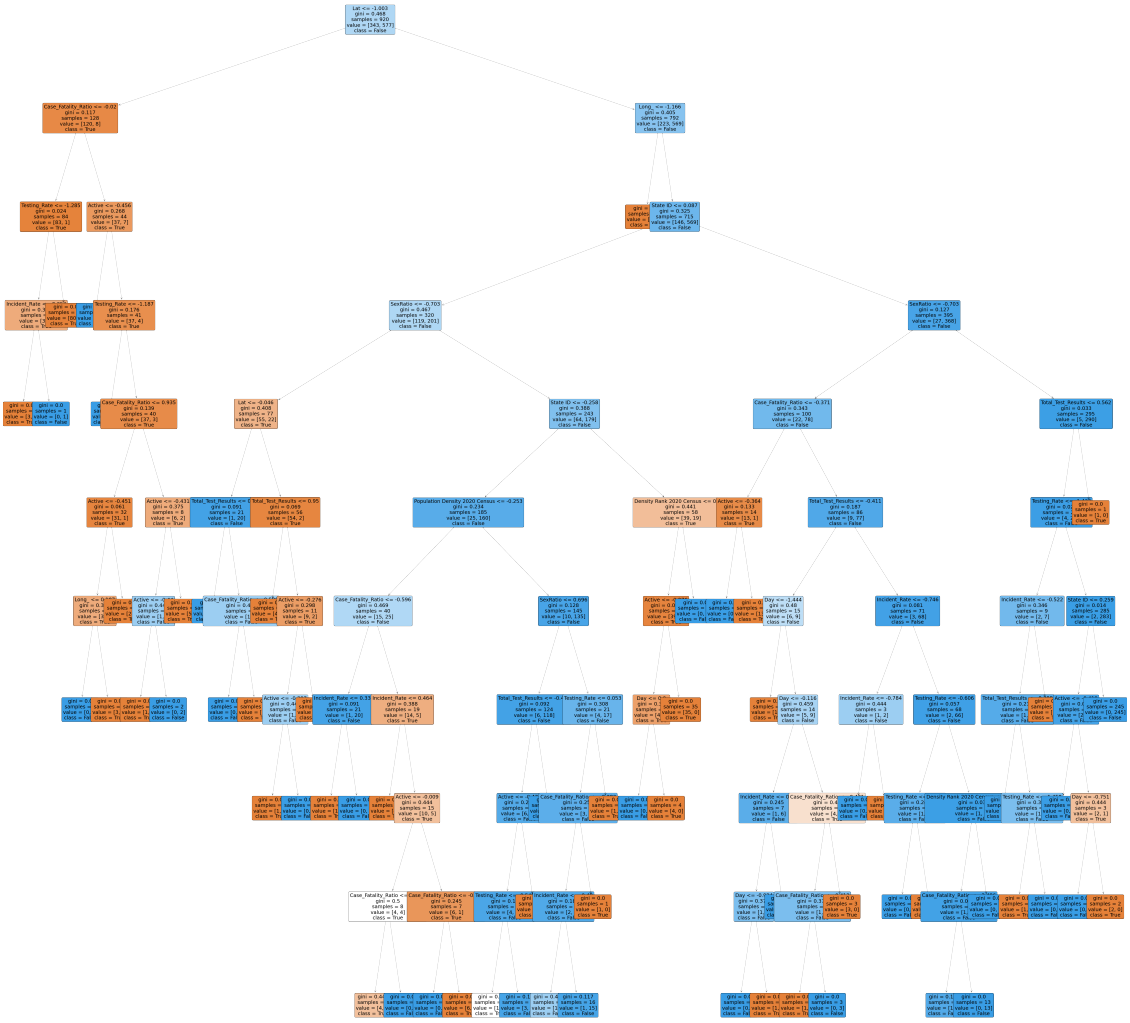


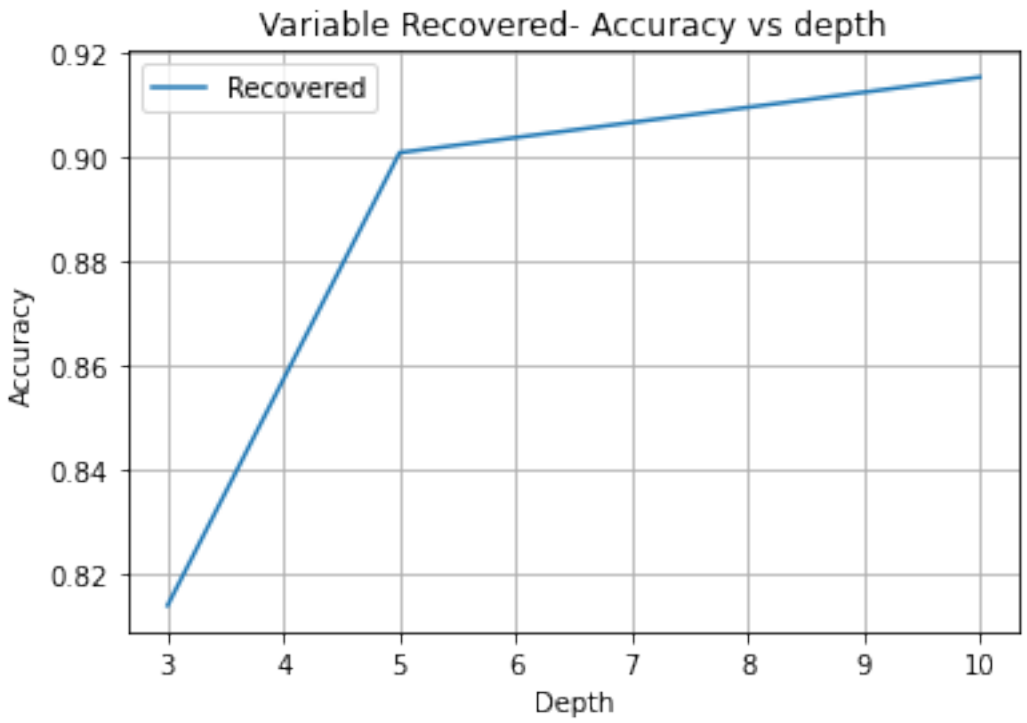
[48]:
```
DecisionTree_depthspec(X3,y3,3,'Recovered')
```

```
Variable Recovered- Accuracy for different depth: [0.8137681159420289,
0.9007246376811594, 0.9152173913043479]
different d values: [3, 5, 10]
```


Lat <= -1.003
gini = 0.468
samples = 1584
value = [1017, 577]
class = False
Case_Fatality_Ratio <= -0.02
gini = 0.117
samples = 328
value = [30, 298]
class = True
Long_ <= -1.166
gini = 0.405
samples = 792
value = [117, 1467]
class = False
Resting_Rate <= -1.240
gini = 0.024
samples = 81
value = [82, 1]
class = True
Active <= -0.456
gini = 0.268
samples = 247
value = [31, 7]
class = True
gini = 0.0
samples = 77
value = [77, 0]
class = True
State ID <= 0.087
gini = 0.325
samples = 715
value = [40, 1467]
class = False
gini = 0.375
samples = 4
value = [3, 1]
class = True
gini = 0.0
samples = 90
value = [80, 0]
class = True
gini = 0.0
samples = 3
value = [0, 3]
class = False
gini = 0.176
samples = 41
value = [31, 4]
class = True
gini = 0.467
samples = 320
value = [119, 201]
class = False
gini = 0.127
samples = 395
value = [27, 369]
class = False


Lat <= -1.003
gini = 0.468
samples = 920
value = [343, 577]
class = False
Case_Fatality_Ratio <= -0.02
gini = 0.117
samples = 128
value = [120, 8]
class = True
Long_ <= -1.166
gini = 0.401
samples = 792
value = [223, 569]
class = False
Testing_Rate <= -1.285
gini = 0.024
samples = 84
value = [83, 1]
class = True
Active <= -0.416
gini = 0.245
samples = 44
value = [37, 7]
class = True
gini = 0.0
samples = 37
value = [37, 0]
class = True
State ID <= 0.087
gini = 0.325
samples = 715
value = [186, 569]
class = False
Testing_Rate <= -1.29
gini = 0.375
samples = 4
value = [1, 3]
class = True
gini = 0.0
samples = 80
value = [82, 0]
class = True
gini = 0.0
samples = 3
value = [0, 3]
class = False
Testing_Rate <= -1.187
gini = 0.176
samples = 41
value = [37, 4]
class = True
SexRatio <= -0.703
gini = 0.467
samples = 320
value = [119, 201]
class = False
SexRatio <= -0.703
gini = 0.127
samples = 395
value = [67, 368]
class = False
gini = 0.0
samples = 1
value = [0, 1]
class = False
gini = 0.0
samples = 1
value = [0, 1]
class = False
Case_Fatality_Ratio <= 0.955
gini = 0.139
samples = 40
value = [37, 3]
class = True
Lat <= -0.040
gini = 0.406
samples = 77
value = [50, 22]
class = True
State ID <= -0.258
gini = 0.366
samples = 243
value = [69, 179]
class = False
Case_Fatality_Ratio <= -0.371
gini = 0.345
samples = 105
value = [27, 78]
class = False
Total_Test_Results <= 0.562
gini = 0.055
samples = 290
value = [5, 290]
class = False
gini = 0.043
samples = 37
value = [1, 36]
class = True
gini = 0.375
samples = 3
value = [0, 3]
class = True
gini = 0.091
samples = 21
value = [1, 20]
class = False
gini = 0.069
samples = 56
value = [49, 7]
class = True
gini = 0.234
samples = 185
value = [20, 165]
class = False
gini = 0.441
samples = 58
value = [39, 19]
class = True
gini = 0.133
samples = 14
value = [1, 13]
class = False
gini = 0.187
samples = 86
value = [6, 77]
class = False
gini = 0.027
samples = 286
value = [4, 282]
class = False
gini = 0.0
samples = 4
value = [1, 3]
class = False





ACCURACY ANALYSIS

For the labels Confirmed and Deaths:- The best value for depth of tree is 3 since accuracy drops after that. It is worth noting that the value of accuracy when depth= None is lower for both the variables.

For the label Recovered:- Although the accuarcy increases with increase in depth, depth=5 looks like an ideal choice since it is quite possible that the increase in accuracy after that is due to overfitting of the model on the given data. Also, in our case, depth= None seems to provide with a similar accuracy but depth=5 would involve lesser computation and therefore would be a preferred choice.

Please note that the accuracy values for depth= None could not be plotted with other numerical values in the scatterplot and was therefore calculated seperately before in the function named 'DecisionTree_Nonedepth'.

SPLITTING RULES ANALYSIS: As we can see the decision trees plotted for each variable, they reveal some interesting features about the data, specifically:

For the label Confirmed:- The places with lower population density i.e., <= 6893872.312 (~0.144 normalized value) have more samples where the value of 'Confirmed' variable is False. We can clearly see that almost 90% of the samples are on the left hand side of the root node and most of them end up in leaves having value of final variable as 'False'. This implies that places with a lower population density do not see a steep rise in daily confirmed cases.

For the label Deaths:- All places with population >= 665992.928 (~ -0.764 normalized value) and longitude >= -137.2494 (~ -2.26 normalized value) have majority samples with value of variable

Deaths= ‘False’ i.e., the number of deaths decrease day by day.

For the label Recovered:- All places with longitude <= -115.7714 (~ -1.166 normalized value) and those with [longitude -115.7714 & latitude <= 33.382 (~ -1.003 normalized value)] have majority samples with value of Recovered= ‘True’ i.e., the number of recovered cases increase day by day.

This indicates that geological location may have influnce on COVID Deaths and Recovered or different States’ strategy for covid may have influnce on deaths and recovery.

1.2 [CM4] Random Forest [1,2,3]

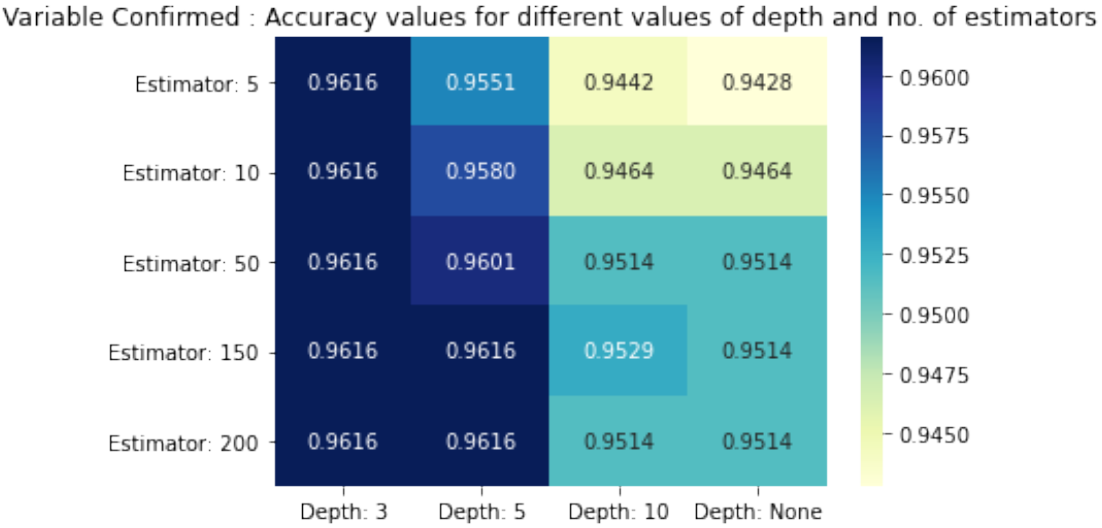
[49]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import seaborn as sns
```

[50]:
```
def RandomForest(X,y, nsplits, label_to_predict):
    depth= [3,5,10, None]
    nestimators= [5,10,50,150,200]
    acc_array= []
    de_array=[]
    i=0
    j=0
    arr = []


    for e in nestimators:
        arr.append([])
        for d in depth:
            kf = KFold(n_splits=nsplits, random_state=98, shuffle=True)
            classifier = RandomForestClassifier(n_estimators=e, max_depth=d,␣
↪random_state=1)
            acc_score = []
            for train_index , test_index in kf.split(X):
                X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
                y_train , y_test = y[train_index] , y[test_index]

                classifier.fit(X_train,y_train)
                y_pred = classifier.predict(X_test)

                acc = metrics.accuracy_score(y_test , y_pred)
                acc_score.append(acc)

            avg_acc_score = sum(acc_score)/nsplits
            acc_array.append(avg_acc_score)
            de_array.append(str(d)+" "+str(e))
            arr[i].append(avg_acc_score)

            j+=1
        i+=1


    print("Variable {}- Accuracy for different values of depth and no. of␣
↪estimators: {}".format(label_to_predict,acc_array))
    print("different depth-estimators values:",de_array)
```

```
    acc_matrix = pd.DataFrame(data=arr, columns=['Depth: 3', 'Depth: 5', 'Depth:
↪ 10', 'Depth: None'],
                              index=['Estimator: 5', 'Estimator: 10',␣
↪'Estimator: 50', 'Estimator: 150', 'Estimator: 200'])

    sns.heatmap(acc_matrix, annot=True, cmap='YlGnBu', fmt='0.4f')
    plt.title('Variable {} : Accuracy values for different values of depth and␣
↪no. of estimators'.format(label_to_predict))

    plt.show
```

[51]:
```
RandomForest(X1,y1,3,'Confirmed')
```

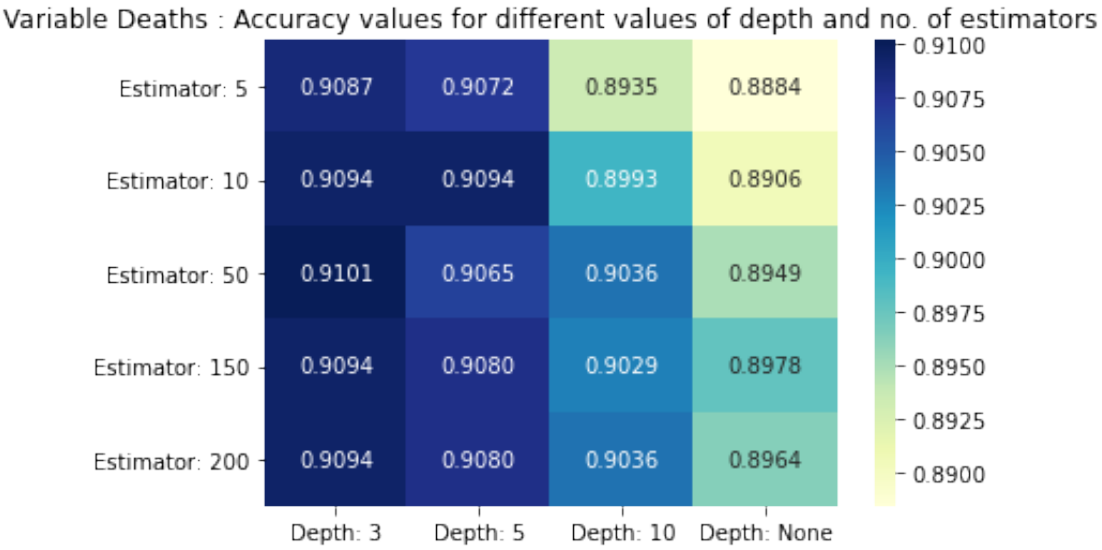
```
Variable Confirmed- Accuracy for different values of depth and no. of
estimators: [0.9615942028985507, 0.9550724637681159, 0.9442028985507246,
0.9427536231884058, 0.9615942028985507, 0.9579710144927537, 0.946376811594203,
0.946376811594203, 0.9615942028985507, 0.9601449275362319, 0.9514492753623189,
0.9514492753623188, 0.9615942028985507, 0.9615942028985508, 0.9528985507246377,
0.9514492753623188, 0.9615942028985507, 0.9615942028985508, 0.9514492753623188,
0.9514492753623189]
different depth-estimators values: ['3 5', '5 5', '10 5', 'None 5', '3 10', '5
10', '10 10', 'None 10', '3 50', '5 50', '10 50', 'None 50', '3 150', '5 150',
'10 150', 'None 150', '3 200', '5 200', '10 200', 'None 200']
```



[52]:
```
RandomForest(X2,y2,3,'Deaths')
```

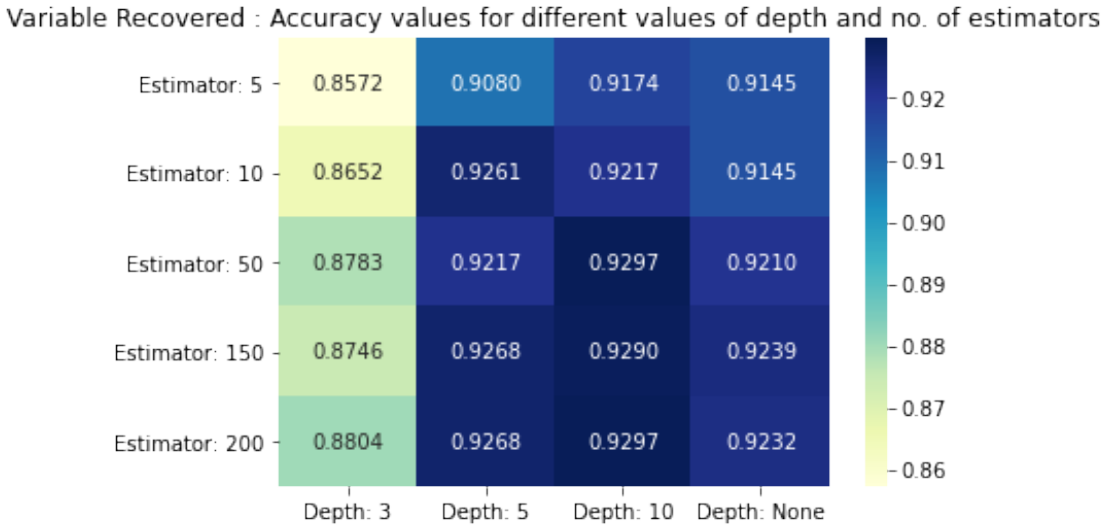
```
Variable Deaths- Accuracy for different values of depth and no. of estimators:
[0.908695652173913, 0.9072463768115941, 0.8934782608695652, 0.8884057971014493,
0.9094202898550724, 0.9094202898550724, 0.8992753623188405, 0.8905797101449275,
0.9101449275362318, 0.9065217391304348, 0.9036231884057971, 0.8949275362318841,
```

```
0.9094202898550724, 0.9079710144927536, 0.9028985507246378, 0.8978260869565217,
0.9094202898550724, 0.9079710144927536, 0.9036231884057971, 0.8963768115942029]
different depth-estimators values: ['3 5', '5 5', '10 5', 'None 5', '3 10', '5
10', '10 10', 'None 10', '3 50', '5 50', '10 50', 'None 50', '3 150', '5 150',
'10 150', 'None 150', '3 200', '5 200', '10 200', 'None 200']
```



[53]:
```
RandomForest(X3,y3,3,'Recovered')
```

```
Variable Recovered- Accuracy for different values of depth and no. of
estimators: [0.8572463768115942, 0.9079710144927536, 0.917391304347826,
0.9144927536231884, 0.8652173913043478, 0.9260869565217392, 0.9217391304347826,
0.9144927536231884, 0.8782608695652173, 0.9217391304347826, 0.9297101449275363,
0.9210144927536232, 0.8746376811594203, 0.9268115942028986, 0.9289855072463769,
0.9239130434782609, 0.8804347826086957, 0.9268115942028986, 0.9297101449275362,
0.9231884057971014]
different depth-estimators values: ['3 5', '5 5', '10 5', 'None 5', '3 10', '5
10', '10 10', 'None 10', '3 50', '5 50', '10 50', 'None 50', '3 150', '5 150',
'10 150', 'None 150', '3 200', '5 200', '10 200', 'None 200']
```



1.2.1 Analysis:

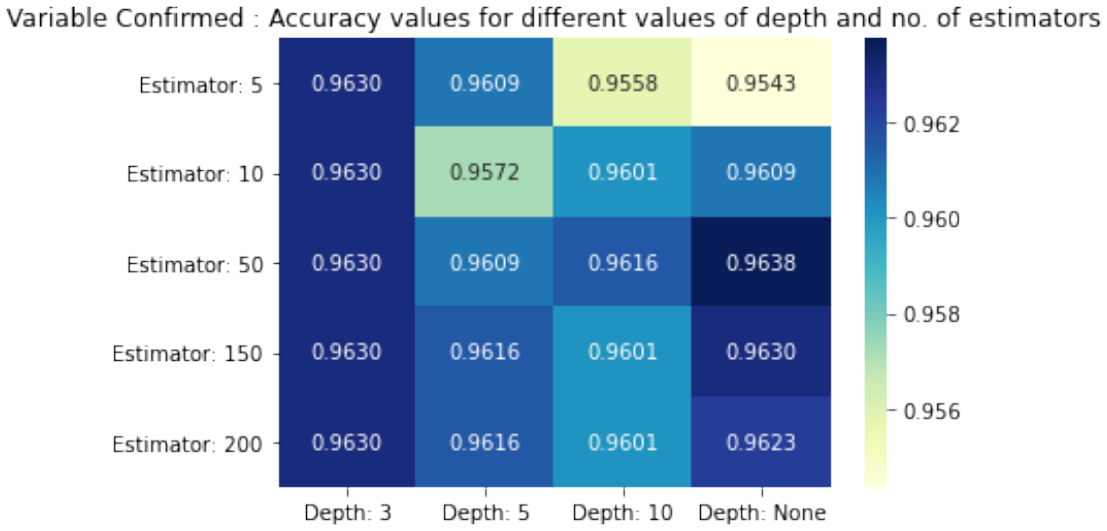
Variable 'Confirmed': As can be seen from the heatmap, many points with depth- (3,5) have the highest accuracy. To avoid overfitting, we would need to choose a high number of estimators (bagging and random feature selection) and the lowest depth without compromising accuracy much. Accordingly, the best metrics in our case would be (depth-3, estimators-50|150|200). Variable 'Deaths': Similarly, the best values for this variable would be with (depth-3, estimators-50) since estimators=50 has the highest accuracy for the lowest depth=3. Variable Recovered: The best value with highest accuracy for this would be (depth-10, estimators-50|150|200) since this gives a very high accuracy.

1.2.2 Report results using using PCA features

Analysis for 'confirmed' label

[54]:
```
RandomForest(pca6_df,y1,3,'Confirmed')
```

```
Variable Confirmed- Accuracy for different values of depth and no. of
estimators: [0.9630434782608696, 0.9608695652173913, 0.9557971014492753,
0.9543478260869565, 0.9630434782608696, 0.9572463768115943, 0.9601449275362319,
0.9608695652173913, 0.9630434782608696, 0.9608695652173913, 0.9615942028985507,
0.9637681159420289, 0.9630434782608696, 0.9615942028985507, 0.9601449275362319,
0.9630434782608696, 0.9630434782608696, 0.9615942028985507, 0.9601449275362319,
0.9623188405797101]
different depth-estimators values: ['3 5', '5 5', '10 5', 'None 5', '3 10', '5
10', '10 10', 'None 10', '3 50', '5 50', '10 50', 'None 50', '3 150', '5 150',
'10 150', 'None 150', '3 200', '5 200', '10 200', 'None 200']
```

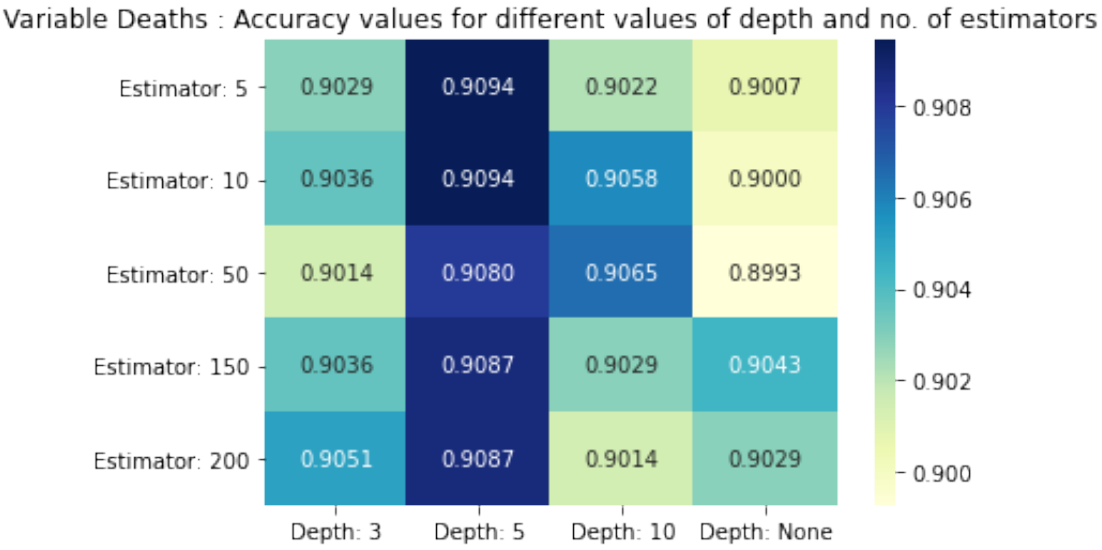


From the above plot, we can determine that with PCA feature datasets, the best accuracy for Random Forest classifer for the variable 'Confirmed' is when depth = 3 as well as depth=None, but since depth= None involves more computation cost, depth=3 with number of estimators = [5,10,50,150,200] would be an ideal choice for large datasets.

Analysis for 'Deaths' label

[55]:
```
RandomForest(pca6_df,y2,3,'Deaths')
```

```
Variable Deaths- Accuracy for different values of depth and no. of estimators:
[0.9028985507246375, 0.9094202898550724, 0.9021739130434782, 0.9007246376811594,
0.9036231884057971, 0.9094202898550724, 0.9057971014492754, 0.9,
0.9014492753623188, 0.9079710144927536, 0.9065217391304348, 0.8992753623188406,
0.9036231884057971, 0.908695652173913, 0.9028985507246375, 0.9043478260869565,
0.905072463768116, 0.908695652173913, 0.9014492753623188, 0.9028985507246378]
different depth-estimators values: ['3 5', '5 5', '10 5', 'None 5', '3 10', '5
10', '10 10', 'None 10', '3 50', '5 50', '10 50', 'None 50', '3 150', '5 150',
'10 150', 'None 150', '3 200', '5 200', '10 200', 'None 200']
```

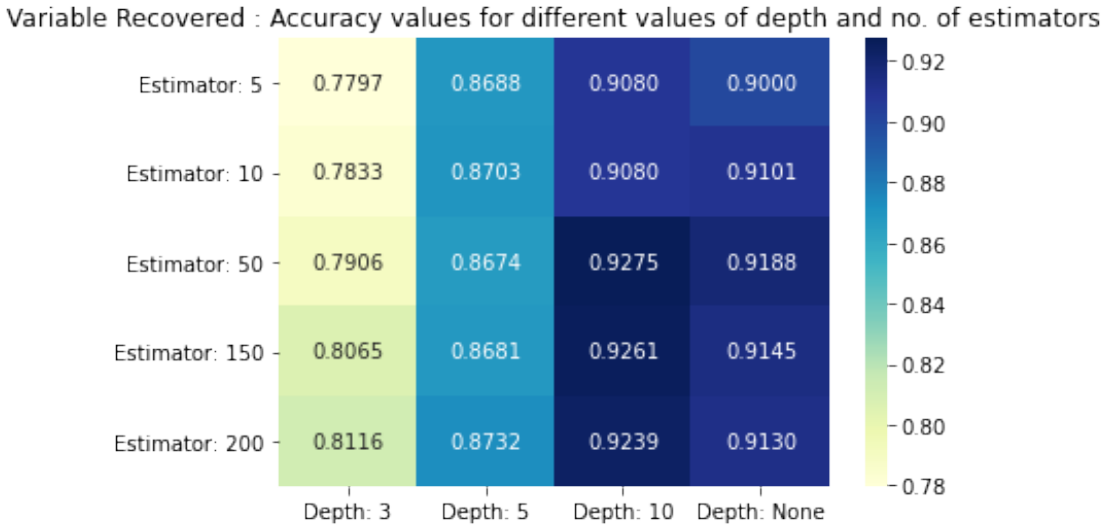


From the above plot, we can determine that with PCA feature datasets, the best accuracy for Random Forest classifer for the variable 'Deaths' is when depth = 5 and number of estimators = [150,200] although no. of estimators=[10, 50] is a good choice too.

Analysis for 'Recovered' label

[56]:
```
RandomForest(pca6_df,y3,3,'Recovered')
```

```
Variable Recovered- Accuracy for different values of depth and no. of
estimators: [0.7797101449275363, 0.868840579710145, 0.9079710144927536, 0.9,
0.7833333333333333, 0.8702898550724637, 0.9079710144927536, 0.910144927536232,
0.7905797101449276, 0.8673913043478261, 0.927536231884058, 0.9188405797101448,
0.8065217391304348, 0.8681159420289855, 0.9260869565217392, 0.9144927536231884,
0.8115942028985508, 0.8731884057971014, 0.9239130434782609, 0.9130434782608696]
different depth-estimators values: ['3 5', '5 5', '10 5', 'None 5', '3 10', '5
10', '10 10', 'None 10', '3 50', '5 50', '10 50', 'None 50', '3 150', '5 150',
'10 150', 'None 150', '3 200', '5 200', '10 200', 'None 200']
```



From the above plot, we can determine that with PCA feature datasets, the best accuracy for Random Forest classifer for the variable 'Recovered' is when depth =10 and number of estimators = [50,150].

We also tried using the automatic best hyperparameter selection algorithm GridSearchCV and it matched our results.

1.3 [CM5] Gradient Tree Boosting

Because the KFold function from sklearn.model_selection defaults to randomly-initialized RandomState at each run, the resulting accuarcy varies each run. Setting random_state to a fixed value will fix the ordering of the indices, which guarantees the same results for each run. During the following parameter tuning for Gradient Tree Boosting, random_state is assigned 98.

[57]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import plot_tree
from sklearn.model_selection import KFold

features=['Day','State ID','Active','Lat','Long_', 'Incident_Rate',␣
 ↪'Total_Test_Results', 'Case_Fatality_Ratio', 'Testing_Rate', 'Resident␣
 ↪Population 2020 Census', 'Population Density 2020 Census', 'Density Rank␣
 ↪2020 Census', 'SexRatio']
X= df_z.loc[:,features]
# set y1= Confirmed y2= deaths y3= Recovered
y1=df.loc[:,'Confirmed']
y2=df.loc[:,'Deaths']
y3=df.loc[:,'Recovered']
```

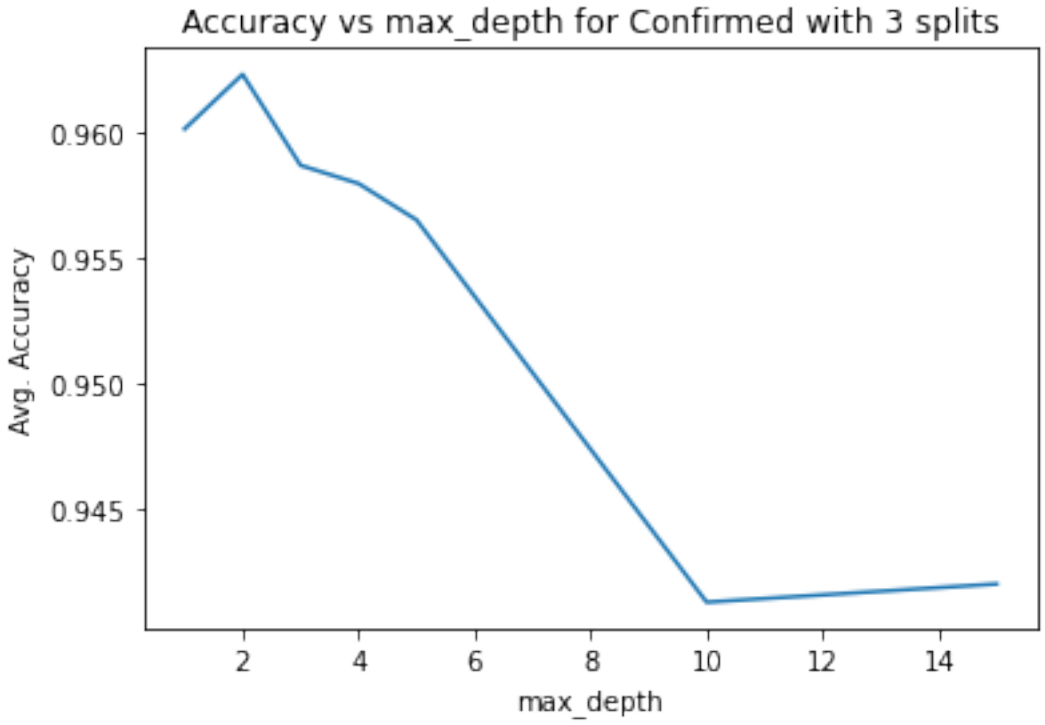
1.3.1 Evaluate the best value for splits and maximum depth

With default number of estimators, we are going to explore the best value for splits an maximum depth for estimator accuarcy. For each label, we train and test Gradient Tree classifier with splits = 3,4 and maximum depth = [1,2,3,4,5,10,15].

[58]:
```
def GradientTree_max_depth(X,y,depth,nsplits):

    kf = KFold(n_splits=nsplits, random_state=98, shuffle=True)
    model = GradientBoostingClassifier(max_depth=depth)
    acc_score = []
    for train_index , test_index in kf.split(X):
        X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]

        model.fit(X_train,y_train)
        y_pred = model.predict(X_test)

        acc = metrics.accuracy_score(y_pred , y_test)
        acc_score.append(acc)

    avg_acc_score = sum(acc_score)/nsplits
    return avg_acc_score
```

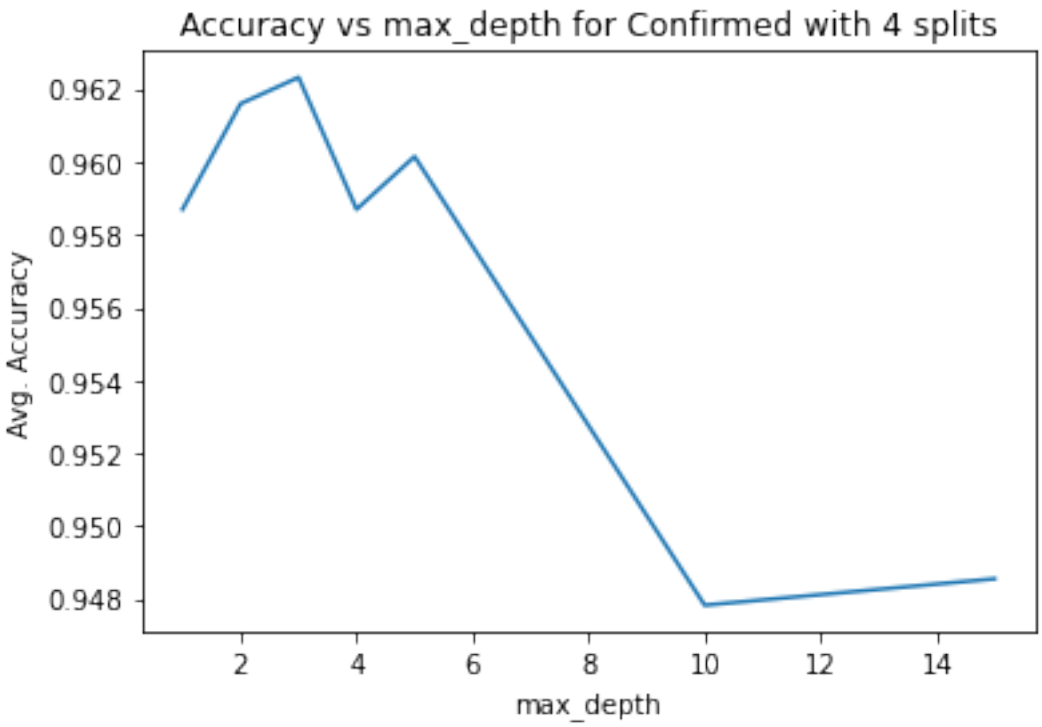
[59]:
```
def plot_GradientTree_max_depth(X,y,nsplits,label):
    depths=[1,2,3,4,5,10,15]
    accuracy_list=[]

    for i in depths:
        accuracy_list.append(GradientTree_max_depth(X,y,i,nsplits))

# plot accuracy vs num of estimators
    plt.plot(depths,accuracy_list)
    plt.xlabel('max_depth')
    plt.ylabel('Avg. Accuracy')
    plt.title('Accuracy vs max_depth for '+label+' with '+str(nsplits)+'␣
 ↪splits')
    plt.show()
    print(accuracy_list)
```

[60]:
```
plot_GradientTree_max_depth(X,y1,3,'Confirmed')
```



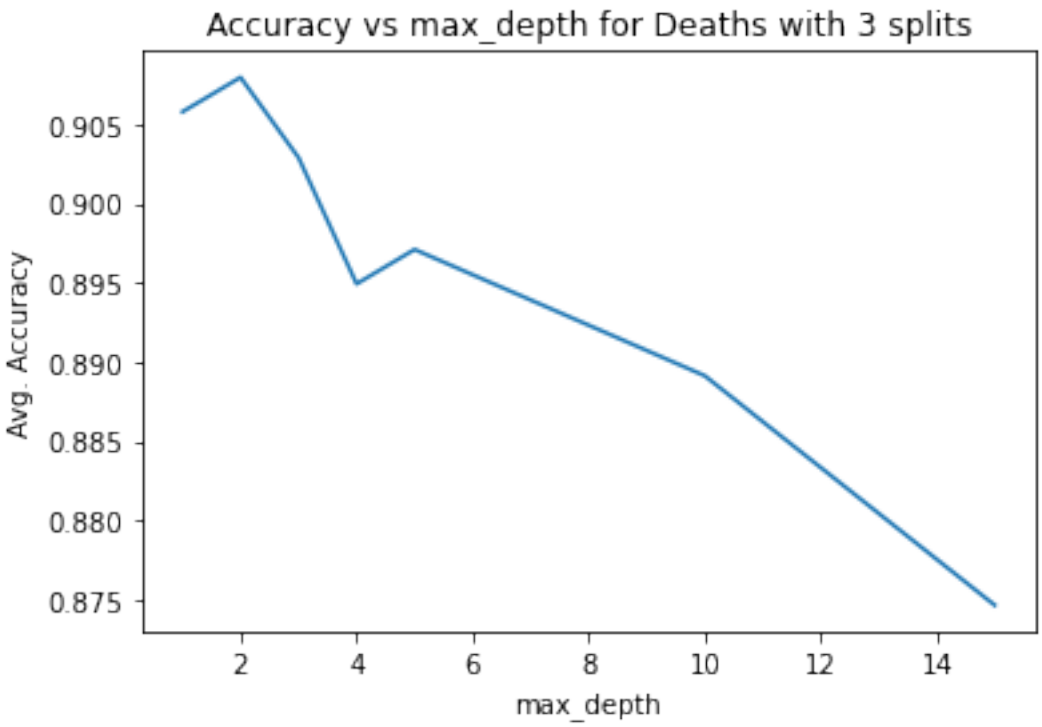

```
[0.9601449275362319, 0.9623188405797101, 0.9586956521739131, 0.9579710144927537,
0.9565217391304349, 0.9413043478260869, 0.9420289855072465]
```

[61]:
```
plot_GradientTree_max_depth(X,y1,4,'Confirmed')
```



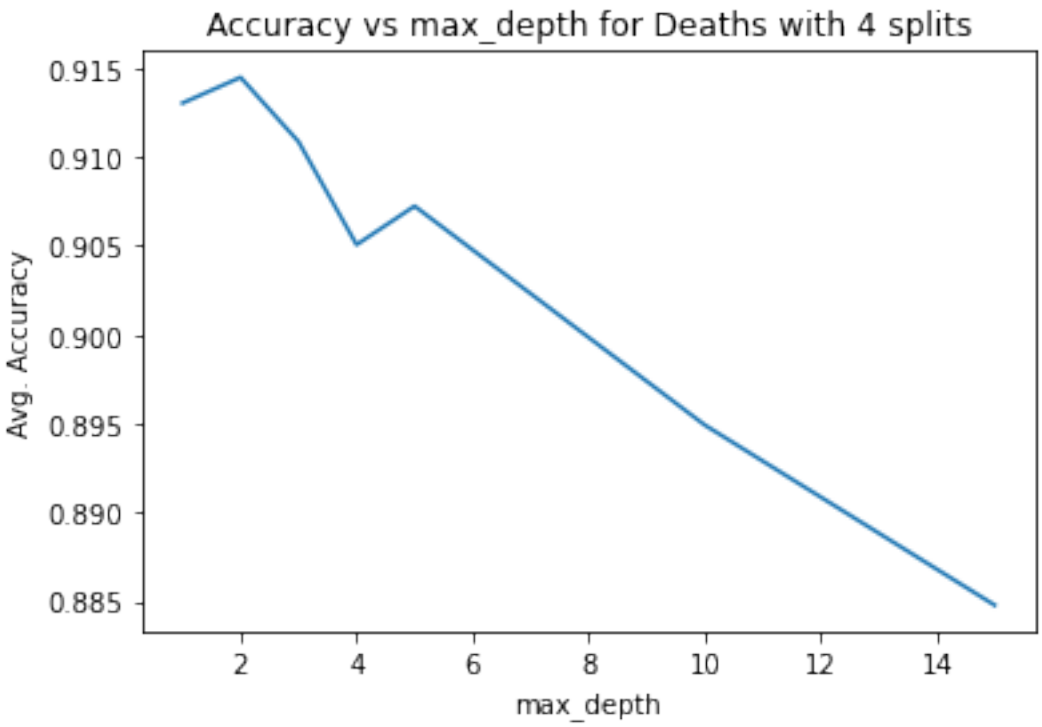

```
[0.9586956521739131, 0.9615942028985507, 0.9623188405797101, 0.9586956521739132,
0.9601449275362319, 0.9478260869565218, 0.9485507246376812]
```

[62]:
```
plot_GradientTree_max_depth(X,y2,3,'Deaths')
```



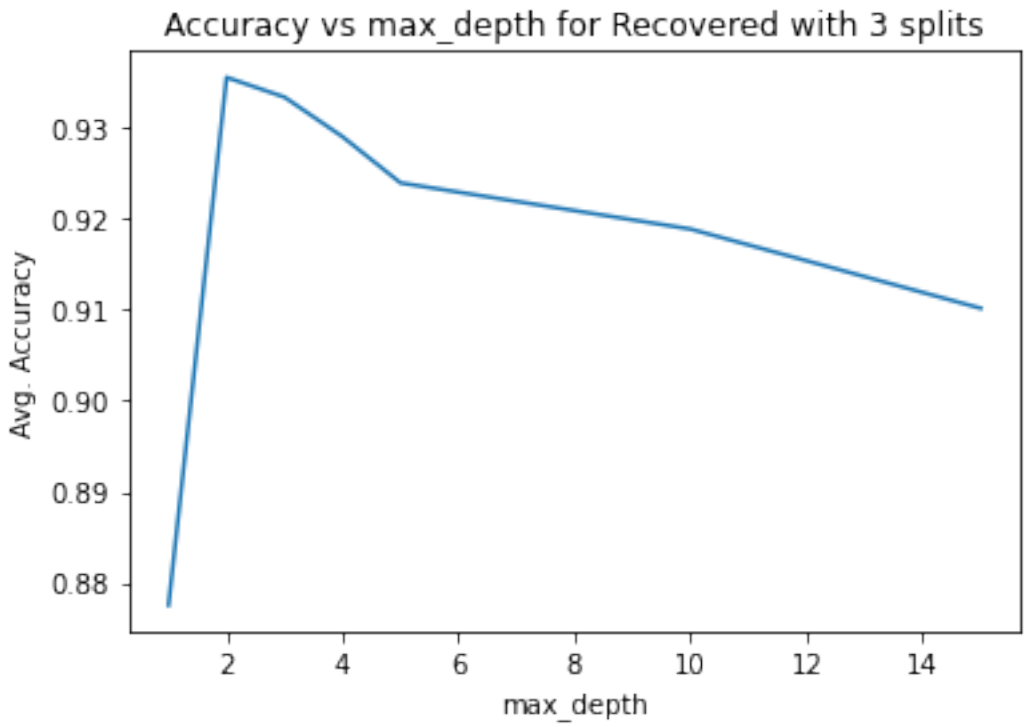

```
[0.9057971014492754, 0.9079710144927535, 0.9028985507246375, 0.8949275362318841,
0.8971014492753623, 0.8891304347826087, 0.8746376811594203]
```

[63]:
```
plot_GradientTree_max_depth(X,y2,4,'Deaths')
```



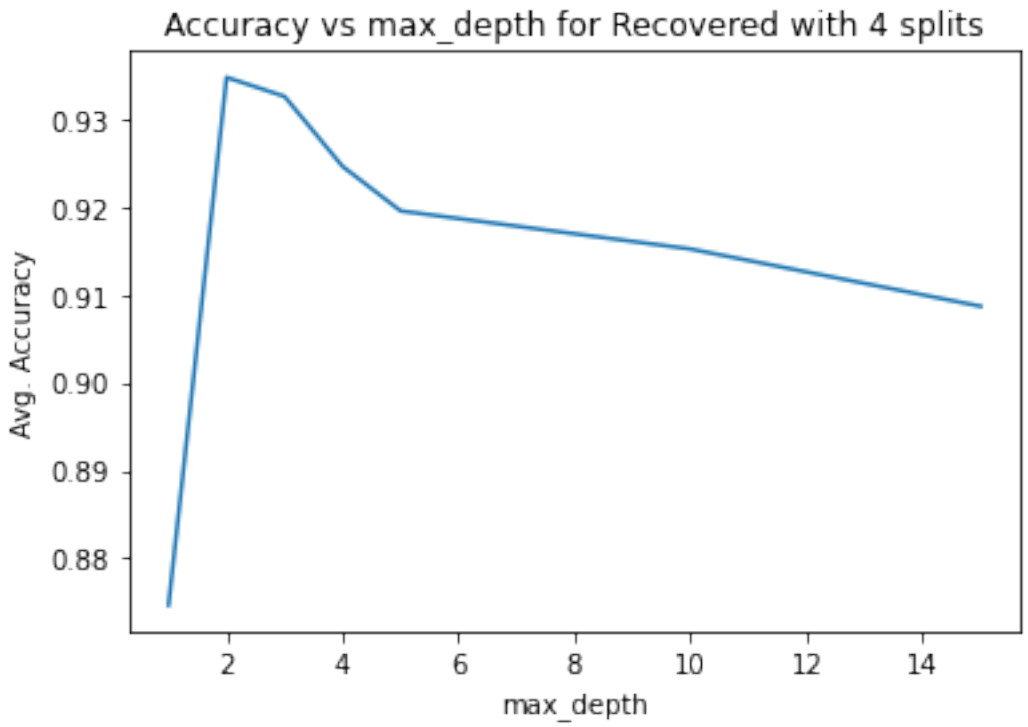

```
[0.9130434782608696, 0.9144927536231884, 0.9108695652173913, 0.905072463768116,
0.9072463768115943, 0.894927536231884, 0.8847826086956521]
```

[64]:
```
plot_GradientTree_max_depth(X,y3,3,'Recovered')
```




```
[0.8775362318840579, 0.9355072463768116, 0.9333333333333332, 0.9289855072463769,
0.9239130434782609, 0.9188405797101448, 0.910144927536232]
```

[65]:
```
plot_GradientTree_max_depth(X,y3,4,'Recovered')
```




```
[0.8746376811594203, 0.9347826086956521, 0.9326086956521739, 0.9246376811594204,
0.9195652173913044, 0.9152173913043478, 0.9086956521739129]
```

Analysis From the above figures a max_depth of 2 gives consistently high accuracy results for all training sets and split numbers. Continue to increase max_depth after 2 significantly decreases the model performance. Thus, the best value for maximum depth of trees for the given dataset is 2. The number of splits only affects model accuarcy by a minimal amount for all labels. In this case, we choose splits = 4 and max_depth = 2 for further analysis.

1.3.2 Evaluate the best value for number of estimators/trees

The following evaluation are based on the following parameter: trees = n_classes x n_estimators, with max_depth =2 and splits =4

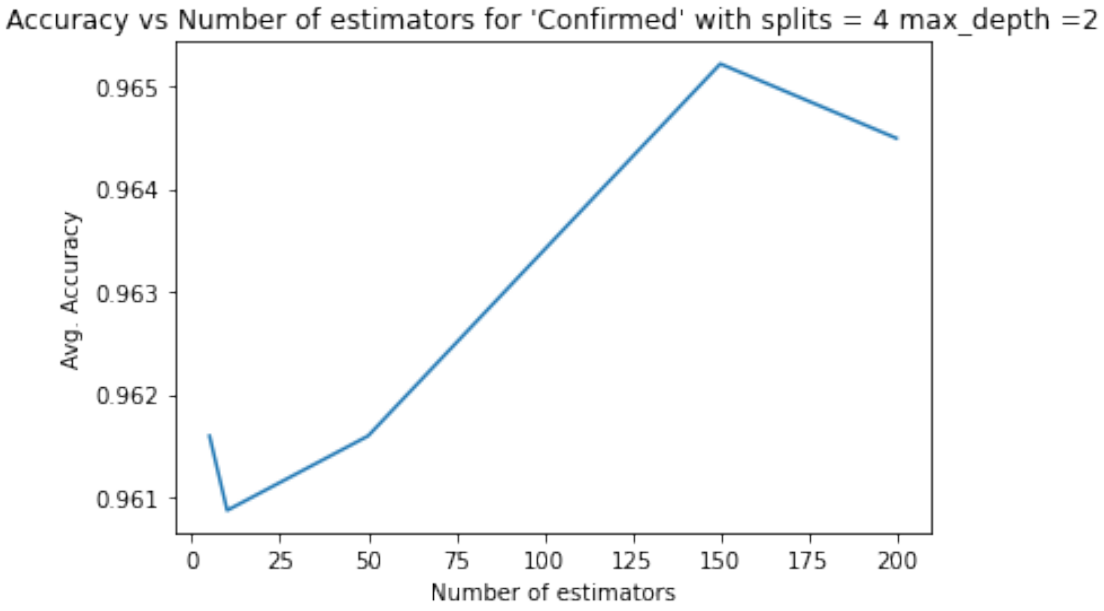
[66]:
```
num_estimators=[5,10,50,150,200]
```

[67]:
```
def GradientTree_n_estimators(X,y, nsplits, nestimators):

    kf = KFold(n_splits=nsplits,random_state=98,shuffle=True)
    model = GradientBoostingClassifier(n_estimators=nestimators, max_depth=2)
    acc_score = []
    for train_index , test_index in kf.split(X):
        X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]
```

```
        model.fit(X_train,y_train)
        y_pred = model.predict(X_test)

        acc = metrics.accuracy_score(y_pred , y_test)
        acc_score.append(acc)

    avg_acc_score = sum(acc_score)/nsplits
    return avg_acc_score
```

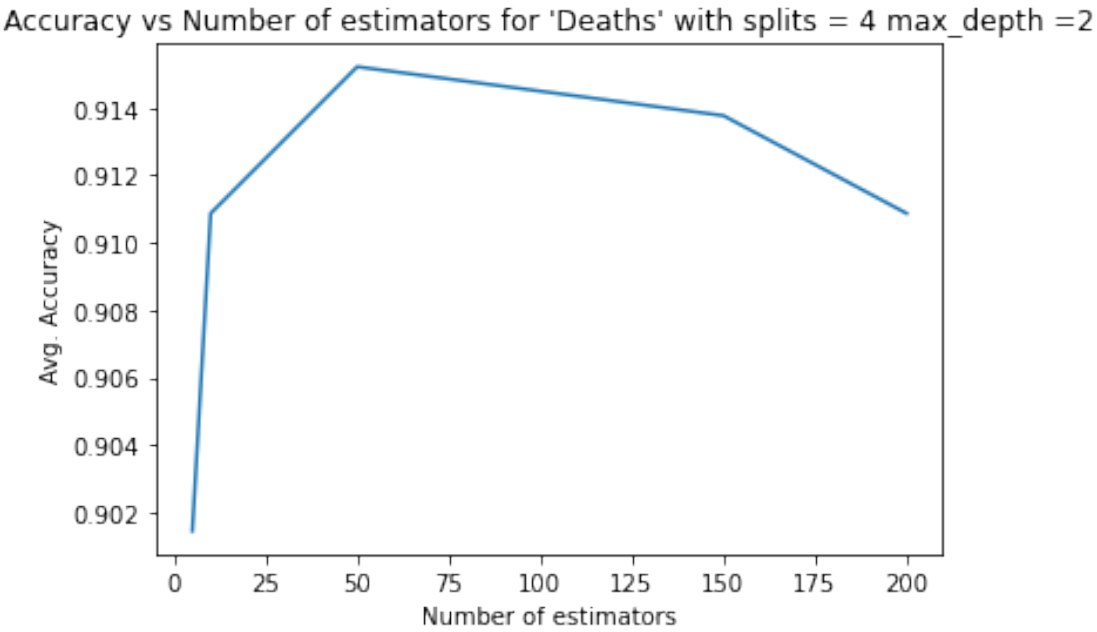
[68]:
```
def plot_GradientTree_n_estimators(X,y,nsplits,label):
    num_estimators=[5,10,50,150,200]
    accuracy_list=[]

    for i in num_estimators:
        accuracy_list.append(GradientTree_n_estimators(X,y,nsplits,i))

# plot accuracy vs num of estimators
    plt.plot(num_estimators,accuracy_list)
    plt.xlabel('Number of estimators')
    plt.ylabel('Avg. Accuracy')
    plt.title('Accuracy vs Number of estimators for '+ '\'' +label+ '\'' + '␣
 ↪with splits = '+str(nsplits) + ' max_depth =2')
    plt.show()
    print(accuracy_list)
```

[69]:
```
plot_GradientTree_n_estimators(X,y1,4,'Confirmed')
```



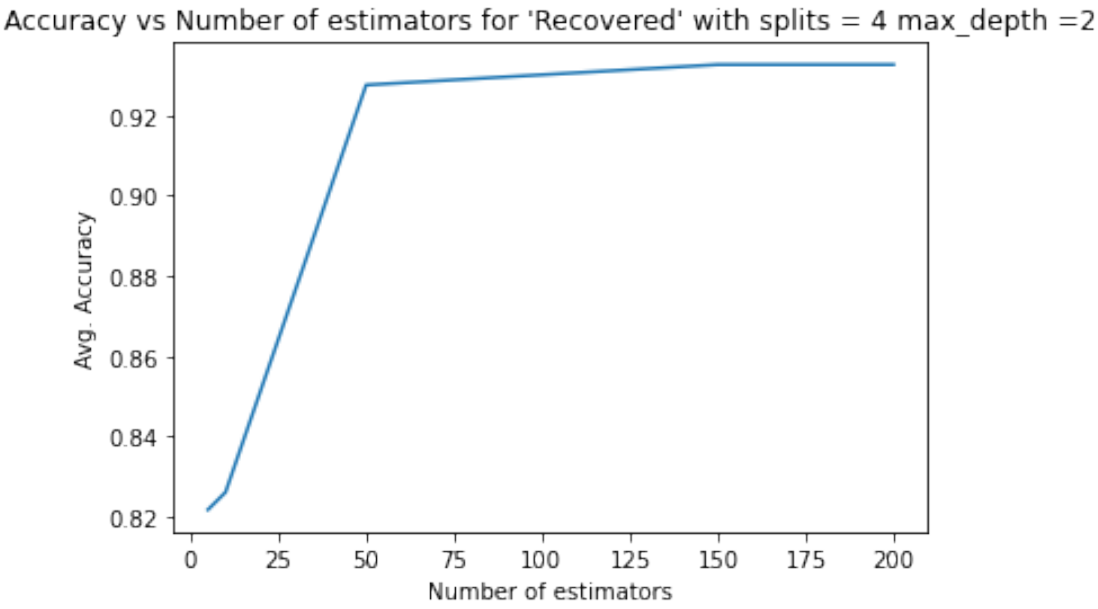
```
[0.9615942028985507, 0.9608695652173913, 0.9615942028985507, 0.9652173913043478,
0.9644927536231884]
```

[70]:
```
plot_GradientTree_n_estimators(X,y2,4,'Deaths')
```




```
[0.9014492753623189, 0.9108695652173913, 0.9152173913043479, 0.913768115942029,
0.9108695652173913]
```

[71]:
```
plot_GradientTree_n_estimators(X,y3,4,'Recovered')
```




```
[0.8217391304347825, 0.826086956521739, 0.927536231884058, 0.9326086956521739,
0.9326086956521739]
```

1.3.3 Observations:

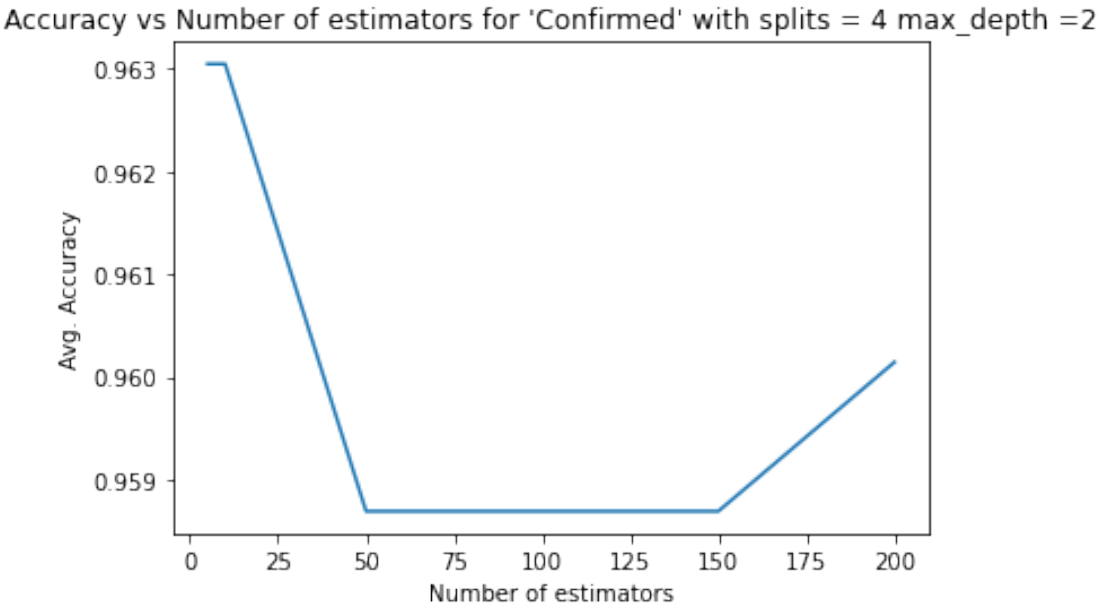
Confirmed From the above plot, the highest accuracy for 'Recovered' is 0.965 with number of estimators = 150, splits = 4 and max_depth =2. #### Deaths The highest accuracy for 'Deaths' is 0.965 with number of estimators = 150, splits = 4 and max_depth =2. #### Recovered The highest accuracy for 'Recovered' is 0.926 with number of estimators = [150,200], splits = 4 and max_depth =2.

For 'Recovered', the accuarcy increases significantly (around 0.1) with increasing number of estimators. For 'Deaths', the accuarcy change with increasing nuber estimators is around 0.014. For 'Confirmed', the accuarcy change with increasing number of estimators is very small (less than 0.003). This might happen because Recovered is the most balanced, so it is easier to train. Deaths is next and Confirmed is the least balanced.

1.3.4 Report results using using PCA features

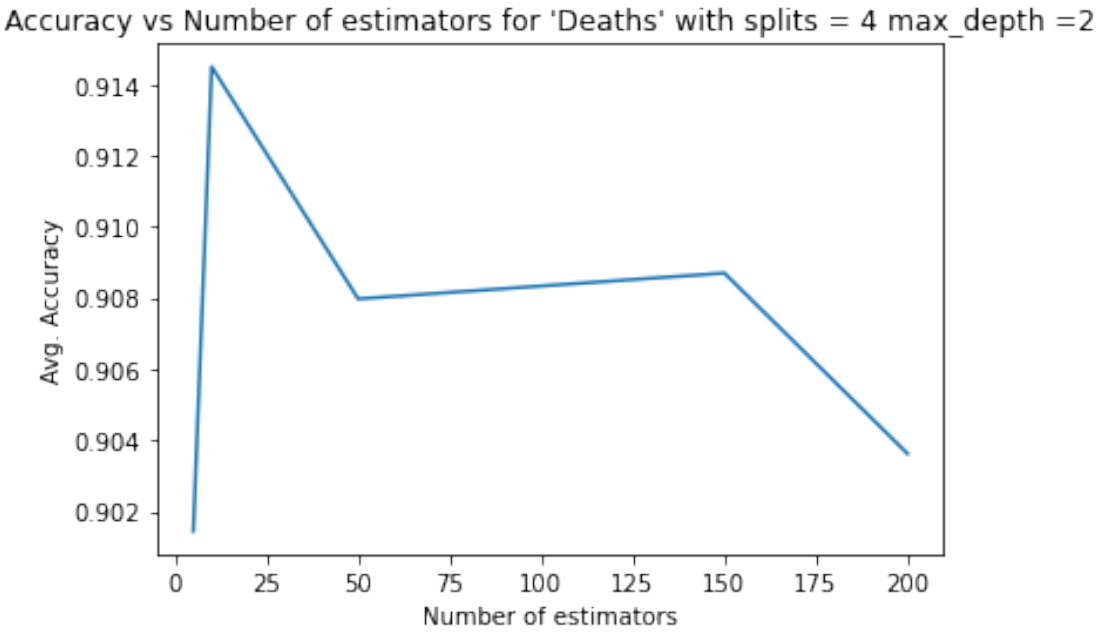
Next, we are going to re train the Gradient Boosing Classifier for each labels using the first 6 major LDA features defined from CM2.

[72]:
```
plot_GradientTree_n_estimators(pca6_df,y1,4,'Confirmed')
```



[0.9630434782608696, 0.9630434782608696, 0.958695652173913, 0.9586956521739131, 0.9601449275362319]

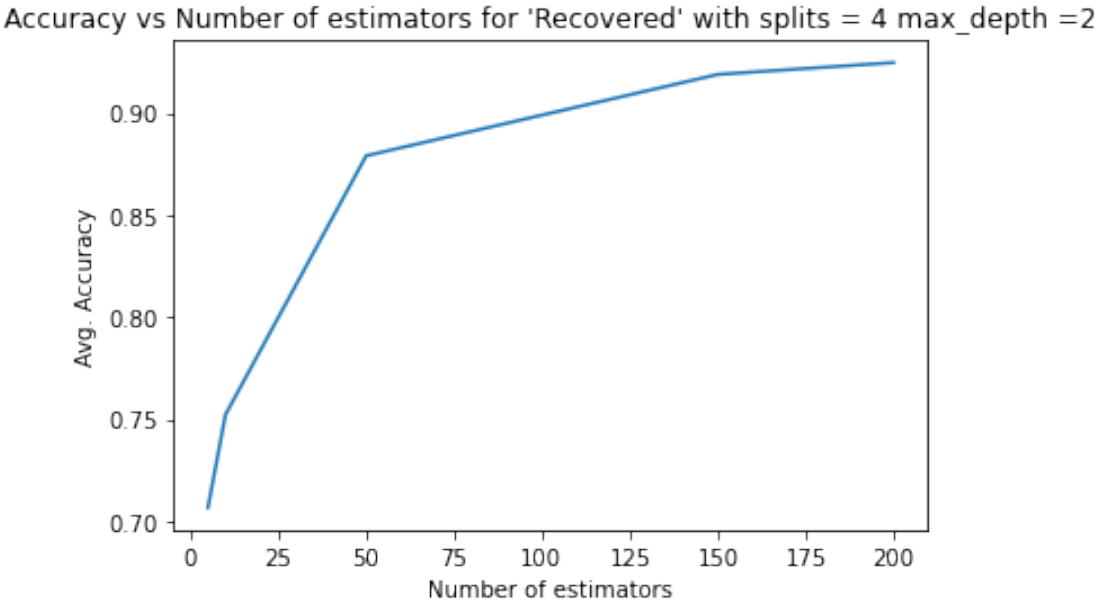
[73]:
```
plot_GradientTree_n_estimators(pca6_df,y2,4,'Deaths')
```



[0.9014492753623189, 0.9144927536231885, 0.9079710144927536, 0.908695652173913,

0.9036231884057973]

[74]:
```
plot_GradientTree_n_estimators(pca6_df,y3,4,'Recovered')
```



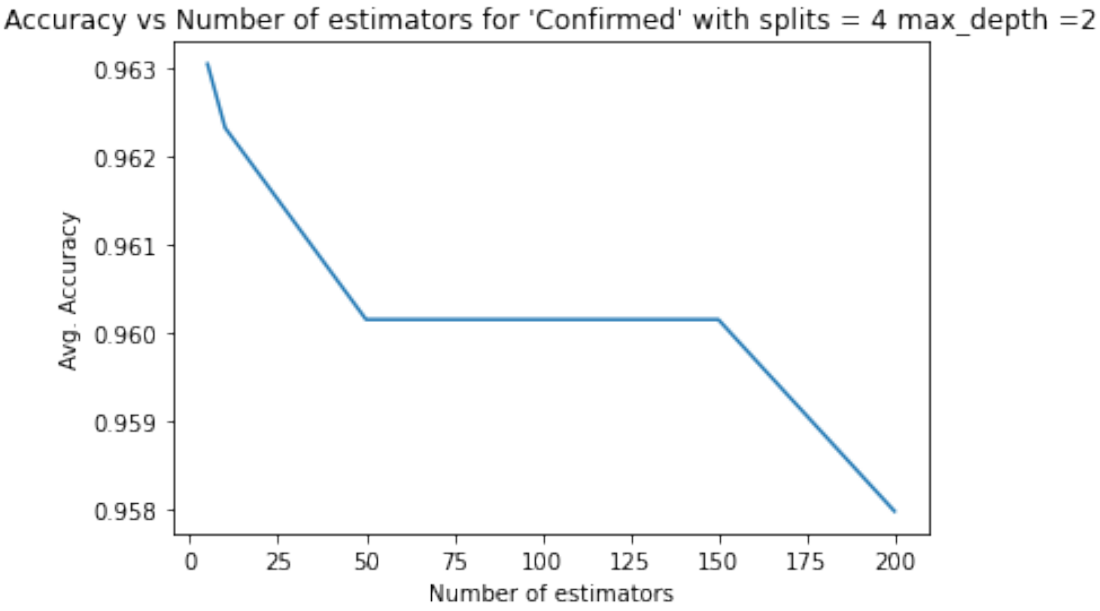
[0.7065217391304348, 0.7521739130434782, 0.8789855072463768, 0.918840579710145, 0.9246376811594202]

From the above plots, we can observe the same trends as those obtained with normalized data. Particularly, increasing number of estimators greatly increases the model accuarcy with balanced data set. However, the max accuarcy dropped slightly compared with the one obtained with original normalized data.

For unbalanced datasets, changing number of estimators slightly changes the model accuarcy (less than 0.015 for 'Deaths' and 'Confirmed'). This is expected since the 6-component PCA data was obtained from original normalized data and accounts 90% of the total cumulative variance.

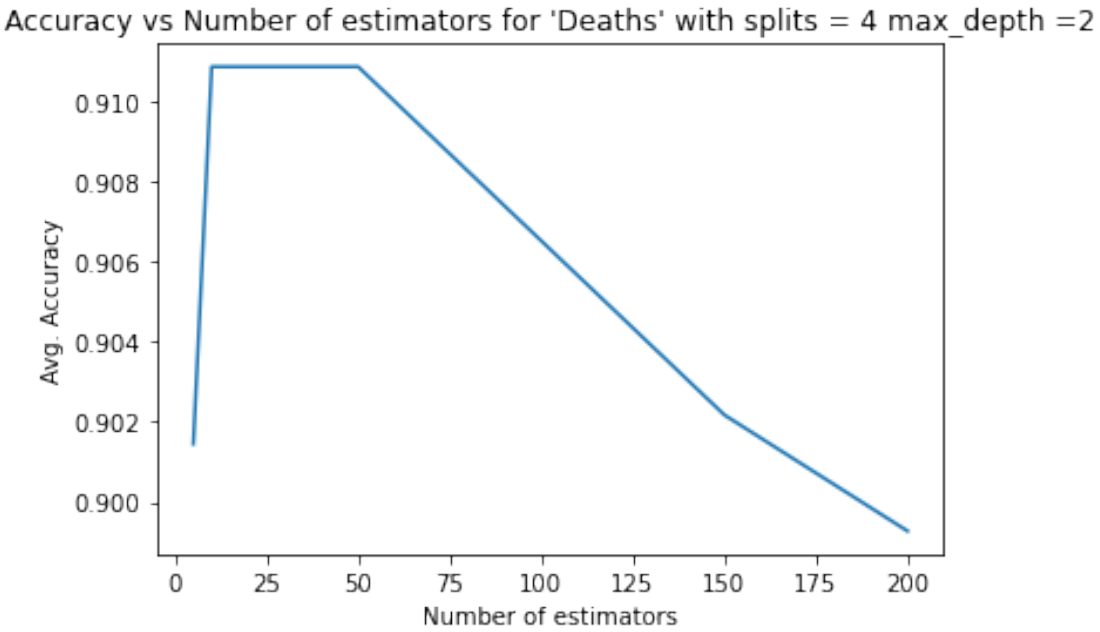
1.3.5 Report results using using LDA features

[75]:
```
plot_GradientTree_n_estimators(LDA_df,y1,4,'Confirmed')
```




```
[0.9630434782608696, 0.9623188405797102, 0.9601449275362318, 0.9601449275362319,
0.9579710144927536]
```

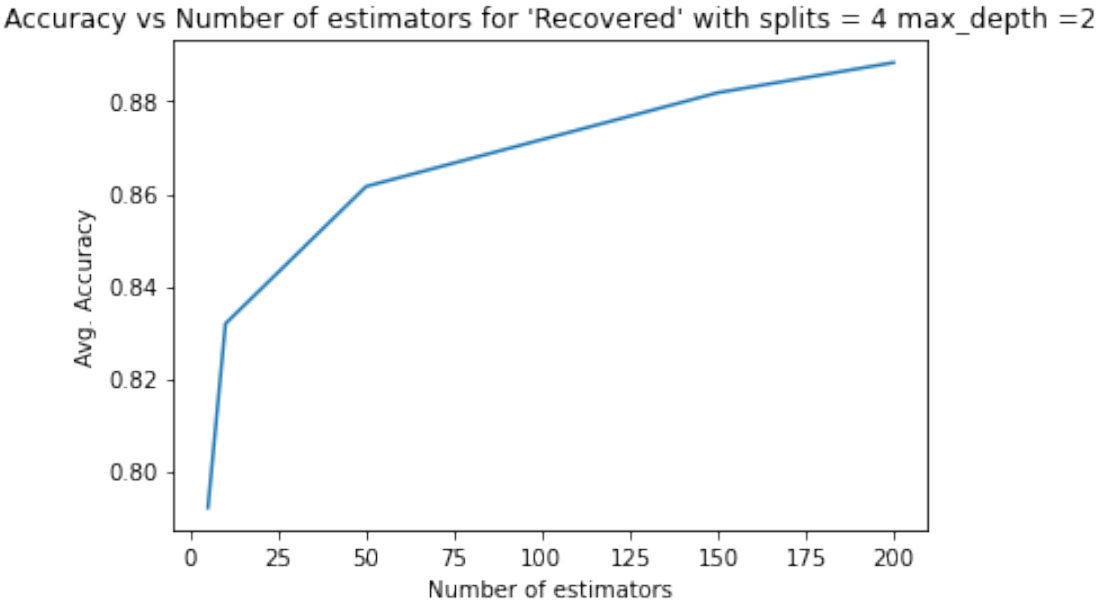
[76]:
```
plot_GradientTree_n_estimators(LDA_df,y2,4,'Deaths')
```




```
[0.9014492753623189, 0.9108695652173913, 0.9108695652173913, 0.9021739130434783,
```

```
0.8992753623188405]
```

[77]:
```
plot_GradientTree_n_estimators(LDA_df,y3,4,'Recovered')
```




```
[0.7920289855072464, 0.8318840579710145, 0.8615942028985507, 0.8818840579710144,
0.8884057971014494]
```

From the above plots, we can observe the same trends as those obtained with normalized data. Particularly, increasing number of estimators greatly increases the model accuarcy with balanced data set. However, the highest accuarcy for 'Recovered' is less than that from PCA data.

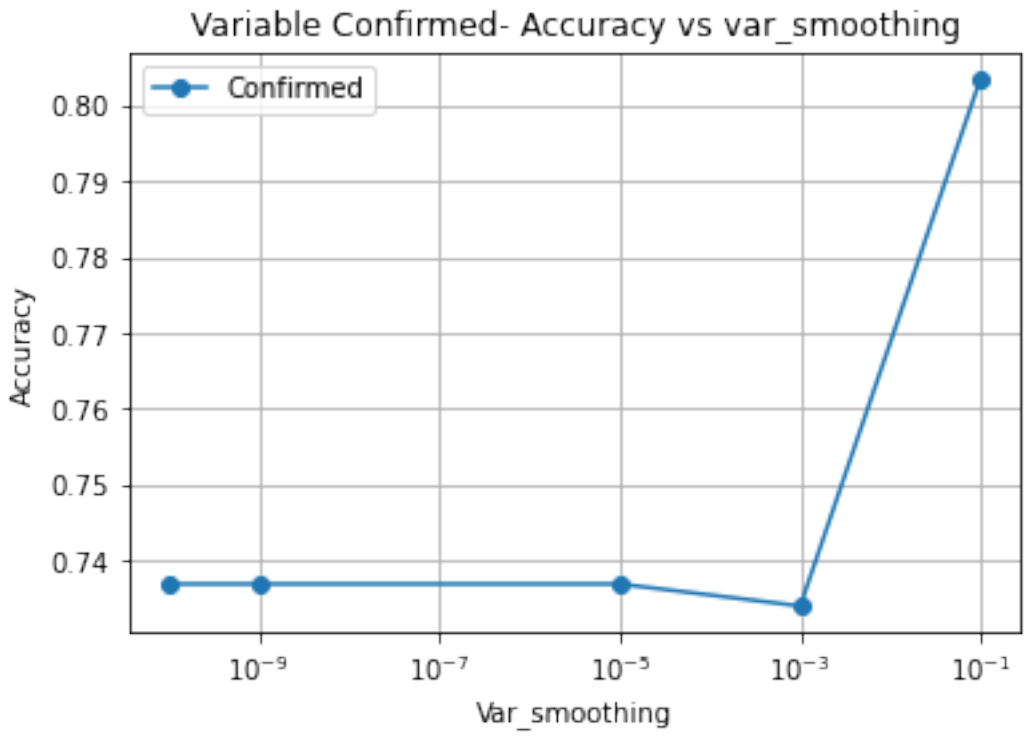
For unbalanced datasets, changing number of estimators slightly changes the model accuarcy (less than 0.015 for 'Deaths' and 'Confirmed'). This is expected since the LDA data was obtained from original normalized data and information loss is expected during LDA transformation.

1.4 [CM6] Naive Bayes

[78]:
```
from sklearn.naive_bayes import GaussianNB

def NaiveBayesClassifier(X,y,label_to_predict, nsplits=10):
    var_smooth= [0.0000000001, 0.000000001, 0.00001, 0.001, 0.1 ]
    acc_array= []
    v_array=[]

    for v in var_smooth:
        kf = KFold(n_splits=nsplits)
        model = GaussianNB(var_smoothing=v)
        acc_score = []
        for train_index , test_index in kf.split(X):
            X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
            y_train , y_test = y[train_index] , y[test_index]

            model.fit(X_train,y_train)
            y_pred = model.predict(X_test)

            acc = metrics.accuracy_score(y_test,y_pred)
            acc_score.append(acc)

        avg_acc_score = sum(acc_score)/nsplits
        acc_array.append(avg_acc_score)
        v_array.append(v)



    print("Variable {}- Accuracy for different var_smoothing: {}".
↪format(label_to_predict,acc_array))
    print("different var_smoothing values:",var_smooth)

    plt.plot(v_array,acc_array,label=label_to_predict, marker='o')
    plt.title( "Variable {}- Accuracy vs var_smoothing".
↪format(label_to_predict))
    plt.xlabel("Var_smoothing")
    plt.ylabel("Accuracy")
    plt.xscale("log")
    plt.legend()
    plt.grid(True)
```

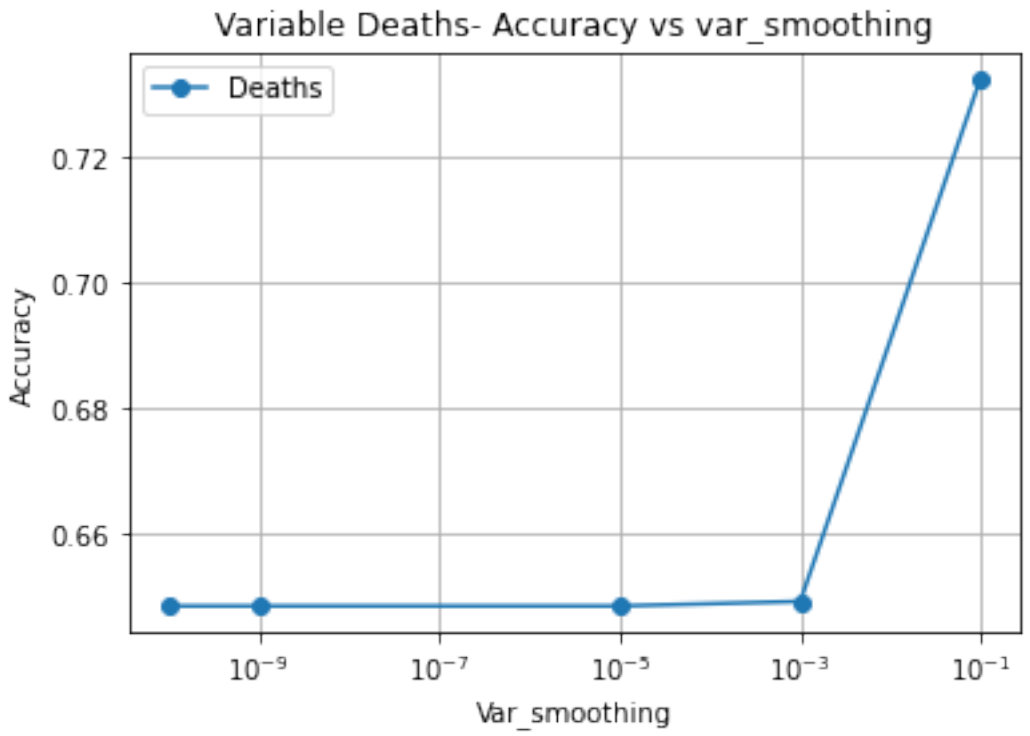
[79]:
```
NaiveBayesClassifier(X1,y1,'Confirmed')
```

```
Variable Confirmed- Accuracy for different var_smoothing: [0.7369565217391304,
0.7369565217391304, 0.7369565217391304, 0.7340579710144927, 0.803623188405797]
different var_smoothing values: [1e-10, 1e-09, 1e-05, 0.001, 0.1]
```



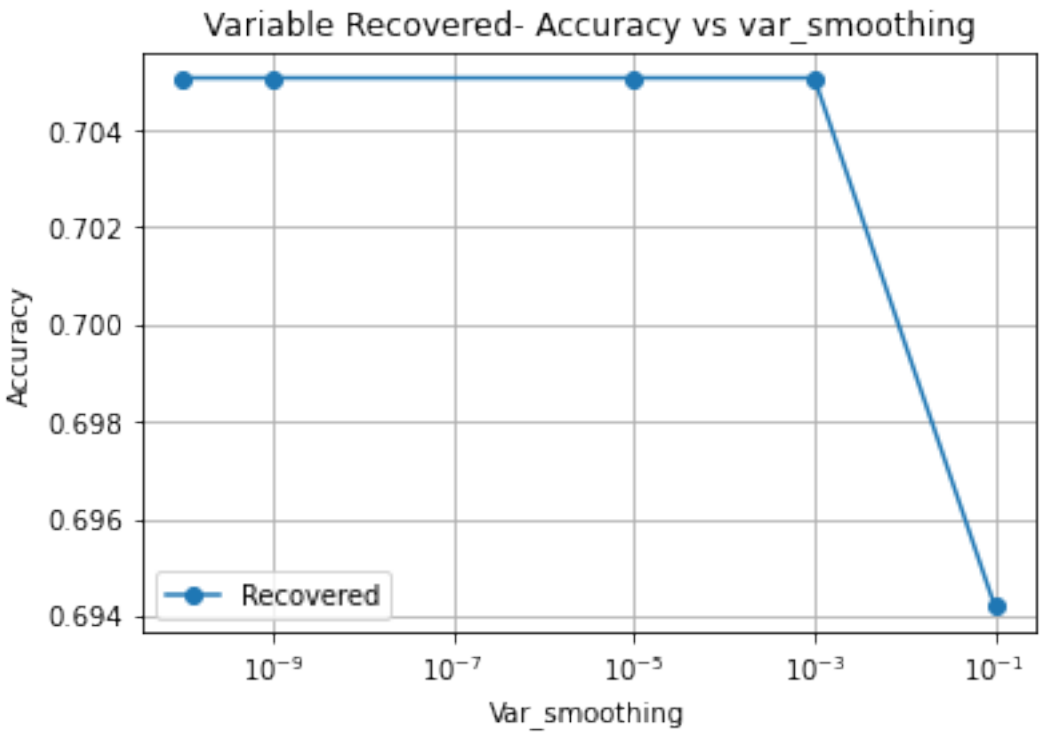
[80]:
```
NaiveBayesClassifier(X2,y2,'Deaths')
```

```
Variable Deaths- Accuracy for different var_smoothing: [0.6485507246376812,
0.6485507246376812, 0.6485507246376812, 0.6492753623188406, 0.732608695652174]
different var_smoothing values: [1e-10, 1e-09, 1e-05, 0.001, 0.1]
```



[81]:
```
NaiveBayesClassifier(X3,y3,'Recovered')
```

```
Variable Recovered- Accuracy for different var_smoothing: [0.7050724637681159,
0.7050724637681159, 0.7050724637681159, 0.7050724637681159, 0.6942028985507246]
different var_smoothing values: [1e-10, 1e-09, 1e-05, 0.001, 0.1]
```



From the above plots, we can see that the var_smoothing in the range of {1e-10, 1e- 9, 1e-5, 1e-3} have little to no impact on the accuarcy of the Naive Bayes classifier predicting all three labels. When the var_smoothing equals 1e-1 the accuarcy for predicting 'Confirmed' increases from 0.734 to 0.804 and the accuarcy for predicting 'Deaths' increased from 0.649 to 0.732. For 'Recovered', the accuarcy slightly decreases from 0.705 to 0.694.

A possible explaination for the above observation is that increasing var_smoothing is able to improve the performance of NB classifier on unbalanced data but not on balanced data. Infact, for balanced data, over smoothing could happen with large var_smoothing value and decrease NB classifier accuarcy. This observation agrees on the given information that 'Recovered' is the most balanced, whereas 'Deaths' and 'Confirmed' are unbalanced.

1.5 [CM7] Performance evaluation

Explain the performance of NB compared to the decision tree approaches in Question 3. Use the original feature space only for this, not the extracted features from PCA or LDA. Can you use the learned parameters of NB to make an interpretation about the data and compare this to the single Decision Tree model?

Comparing the outputs of CM3 (Decision Tree) and CM6 (Naive Bayes), we can notice that Decision Tree performs better with greater accuracy in general. Although this could be a result of overfitting which is a common problem with Decision Trees and was the reason why ensemble methods like Random Forests were introduced. Naive Bayes, on the other hand, is not prone to overfitting but it may not work well for dependent features, this is another possible reason that NB accuarcy is lower since 'Deaths' 'Recovered' and 'Confirmed' are not independent variables. They are highly depends on corresponding State's policies and population distributions. Decision tree gives a better idea of the approach used by observing the splitting rules but naive bayes is harder to interpret.

Boosting is based on weak learners. In terms of decision trees, weak learners are shallow trees, sometimes even as small as decision stumps, that explains why tree depth of 2 gives highest model accuarcy in CM5.

1.6 [CM8] Kaggle

Kaggle Group: Group 2

Kaggle Url: https://www.kaggle.com/c/ece657as21-asg2

Below are the sets of code used to generate results for Kaggle challenge

1.6.1 Test data preparation and normalization

[82]:
```
# load dkmacovid test data
df_test = pd.read_csv('dkmacovid_kaggletest_features.csv')
# remove comma from dataframe in 'Resident Population 2020 Census' and␣
 ↪'Population Density 2020 Census'
df_test = df_test.replace(',','', regex=True)
# convert string to numeric data
df_test['Resident Population 2020 Census'] = df_test['Resident Population 2020␣
 ↪Census'].astype(float)
# df_test['Population Density 2020 Census'] = df_test['Population Density 2020␣
 ↪Census'].astype(float)
df_test.head()
df_test_z= df_test.loc[:,features]
df_test_z.iloc[:,:]=std_scaler.fit_transform(df_test_z.iloc[:,:])
X_kaggle= df_test_z.loc[:,features]
df_test_z.head()
```

[82]:

	Day	State ID	Active	Lat	Long_	Incident_Rate	\
0	-1.675247	-1.283315	1.630365	0.18638	0.68734	-0.034670	
1	-1.559712	-1.283315	1.642290	0.18638	0.68734	-0.019131	
2	-1.444178	-1.283315	1.655878	0.18638	0.68734	-0.001540	
3	-1.328644	-1.283315	1.674170	0.18638	0.68734	0.022241	
4	-1.213110	-1.283315	1.694395	0.18638	0.68734	0.048560	

	Total_Test_Results	Case_Fatality_Ratio	Testing_Rate	\
0	0.874530	1.718143	1.252685	
1	0.882366	1.729474	1.265333	
2	0.890682	1.727361	1.278757	
3	0.905691	1.737463	1.302983	
4	0.919647	1.751645	1.325509	

	Resident Population 2020 Census	Population Density 2020 Census	\
0	0.094176	1.605706	
1	0.094176	1.605706	
2	0.094176	1.605706	
3	0.094176	1.605706	
4	0.094176	1.605706	

	Density Rank 2020 Census	SexRatio
0	-1.392419	-0.559017

```
1                -1.392419 -0.559017
2                -1.392419 -0.559017
3                -1.392419 -0.559017
4                -1.392419 -0.559017
```

1.6.2 Decision Tree

[83]:
```
def DecisionTree_depthspec_Model(X,y,d):

    #kf = KFold(n_splits=nsplits)
    model = DecisionTreeClassifier(max_depth=d)
    model.fit(X,y)

    return model
```

[84]:
```
# set y1= Confirmed y2= deaths y3= Recovered
# fit each label with parameters got from previous training test
y1_model = DecisionTree_depthspec_Model(X,y1,3)
y1_pred = y1_model.predict(X_kaggle)
y2_model = DecisionTree_depthspec_Model(X,y2,3)
y2_pred = y2_model.predict(X_kaggle)
y3_model = DecisionTree_depthspec_Model(X,y3,5)
y3_pred = y3_model.predict(X_kaggle)
```

[85]:
```
# Construct kaggle result data
# Id        Confirmed        Deaths        Recovered
Confirmed= pd.DataFrame(data= y1_pred, columns=['Confirmed']).astype(int)
Deaths= pd.DataFrame(data= y2_pred, columns=['Deaths']).astype(int)
Recovered= pd.DataFrame(data= y3_pred, columns=['Recovered']).astype(int)
ID = pd.DataFrame(df_test['Id'],columns=['Id'])
df_result=pd.concat([ID,Confirmed,Deaths,Recovered], axis=1)
df_result.to_csv('Kaggle_Decision Tree.csv', index=False)
```

1.6.3 Random Forest

[86]:
```
def RandomForest_Model(X,y,e,d):

    #kf = KFold(n_splits=nsplits)
    model = RandomForestClassifier(n_estimators=e, max_depth=d, random_state=1)
    model.fit(X,y)

    return model
```

[87]:
```
# set y1= Confirmed y2= deaths y3= Recovered
# fit each label with parameters got from previous training test
y1_model = RandomForest_Model(X,y1,150,3)
y1_pred = y1_model.predict(X_kaggle)
```

```
y2_model = RandomForest_Model(X,y2,50,3)
y2_pred = y2_model.predict(X_kaggle)
y3_model = RandomForest_Model(X,y3,150,10)
y3_pred = y3_model.predict(X_kaggle)
```

[88]:
```
# Construct kaggle result data
# Id        Confirmed        Deaths        Recovered
Confirmed= pd.DataFrame(data= y1_pred, columns=['Confirmed']).astype(int)
Deaths= pd.DataFrame(data= y2_pred, columns=['Deaths']).astype(int)
Recovered= pd.DataFrame(data= y3_pred, columns=['Recovered']).astype(int)
ID = pd.DataFrame(df_test['Id'],columns=['Id'])
df_result=pd.concat([ID,Confirmed,Deaths,Recovered], axis=1)
df_result.to_csv('Kaggle_RandomForest.csv', index=False)
```

1.6.4 Gradient Tree Boosting

[89]:
```
def GradientTreeModel_n_estimators(X,y, nsplits, nestimators):

    kf = KFold(n_splits=nsplits)
    model = GradientBoostingClassifier(n_estimators=nestimators, max_depth=2)
    model.fit(X,y)
    return model
```

[90]:
```
# set y1= Confirmed y2= deaths y3= Recovered
# fit each label with parameters got from previous training test
y1_model = GradientTreeModel_n_estimators(X,y1,3,50)
y1_pred = y1_model.predict(X_kaggle)
y2_model = GradientTreeModel_n_estimators(X,y2,3,25)
y2_pred = y2_model.predict(X_kaggle)
y3_model = GradientTreeModel_n_estimators(X,y3,3,50)
y3_pred = y3_model.predict(X_kaggle)
```

[91]:
```
# Construct kaggle result data
# Id        Confirmed        Deaths        Recovered
Confirmed= pd.DataFrame(data= y1_pred, columns=['Confirmed']).astype(int)
Deaths= pd.DataFrame(data= y2_pred, columns=['Deaths']).astype(int)
Recovered= pd.DataFrame(data= y3_pred, columns=['Recovered']).astype(int)
ID = pd.DataFrame(df_test['Id'],columns=['Id'])
df_result=pd.concat([ID,Confirmed,Deaths,Recovered], axis=1)
df_result.to_csv('Kaggle_GradientTreeBoosting.csv', index=False)
```

1.6.5 Naive Bayes

[92]:
```
def NaiveBayes_Classifier_Model(X,y,v):

    #kf = KFold(n_splits=nsplits)
    model = GaussianNB(var_smoothing=v)
```

```
    model.fit(X,y)

    return model
```

[93]:
```
# set y1= Confirmed y2= deaths y3= Recovered
# fit each label with parameters got from previous training test
y1_model = NaiveBayes_Classifier_Model(X,y1,0.1)
y1_pred = y1_model.predict(X_kaggle)
y2_model = NaiveBayes_Classifier_Model(X,y2,0.1)
y2_pred = y2_model.predict(X_kaggle)
y3_model = NaiveBayes_Classifier_Model(X,y3,0.00001)
y3_pred = y3_model.predict(X_kaggle)
```

[94]:
```
# Construct kaggle result data
# Id        Confirmed        Deaths        Recovered
Confirmed= pd.DataFrame(data= y1_pred, columns=['Confirmed']).astype(int)
Deaths= pd.DataFrame(data= y2_pred, columns=['Deaths']).astype(int)
Recovered= pd.DataFrame(data= y3_pred, columns=['Recovered']).astype(int)
ID = pd.DataFrame(df_test['Id'],columns=['Id'])
df_result=pd.concat([ID,Confirmed,Deaths,Recovered], axis=1)
df_result.to_csv('Kaggle_NaiveBayes.csv', index=False)
```

2 References

[1] https://www.askpython.com/python/examples/k-fold-cross-validation [2] https://stackabuse.com/ultimate-guide-to-heatmaps-in-seaborn-with-python [3] https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74