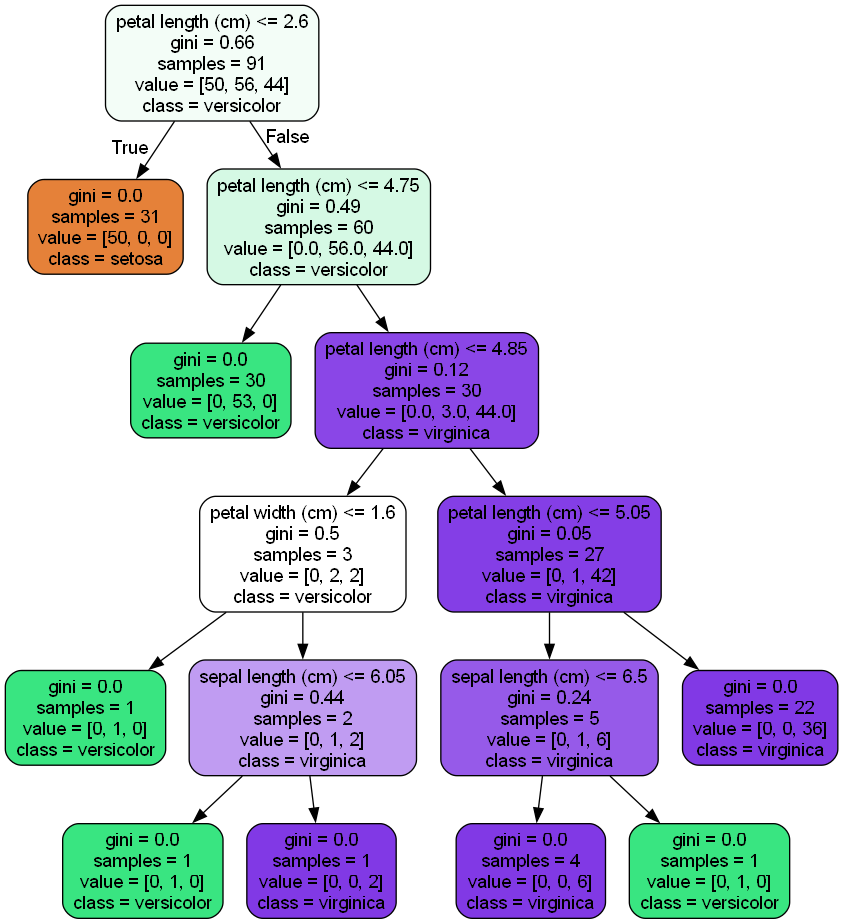
## Random Forest

Random Forest is an ensemble of Decision Trees. With a few exceptions, a RandomForestClassifier has all the hyperparameters of a DecisionTreeClassifier (to control how trees are grown), plus all the hyperparameters of a BaggingClassifier to control the ensemble itself.

The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This results in a greater tree diversity, which (once again) trades a higher bias for a lower variance, generally yielding an overall better model. The following BaggingClassifier is roughly equivalent to the previous RandomForestClassifier. Run the cell below to visualize a single estimator from a random forest model, using the Iris dataset to classify the data into the appropriate species.

from sklearn.tree import plot\_tree  
from sklearn.tree import export\_graphviz  
from sklearn.datasets import load\_iris  
  
iris = load\_iris()  
from sklearn.ensemble import RandomForestClassifier  
  
model = RandomForestClassifier(n\_estimators=10)  
model.fit(iris.data, iris.target)  
estimator = model.estimators\_[5]  
export\_graphviz(estimator, out\_file='tree.dot',  
 feature\_names=iris.feature\_names,  
 class\_names=iris.target\_names,  
 rounded=True, proportion=False,  
 precision=2, filled=True)  
from subprocess import call  
  
call(['C:/Program Files/Graphviz/bin/dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gd pi=600'])  
from IPython.display import Image  
  
Image(filename='tree.png')



Notice how each split seperates the data into buckets of similar observations. This is a single tree and a relatively simple classification dataset, but the same method is used in a more complex dataset with greater depth to the trees.

## Coronavirus

Coronavirus disease (COVID-19) is an infectious disease caused by a new virus. The disease causes respiratory illness (like the flu) with symptoms such as a cough, fever, and in more severe cases, difficulty breathing. You can protect yourself by washing your hands frequently, avoiding touching your face, and avoiding close contact (1 meter or 3 feet) with people who are unwell. An outbreak of COVID-19 started in December 2019 and at the time of the creation of this project was continuing to spread throughout the world. Many governments recommended only essential outings to public places and closed most business that do not serve food or sell essential items. An excellent [spatial dashboard](https://www.arcgis.com/apps/opsdashboard/index.html#/bda7594740fd40299423467b48e9ecf6) built by Johns Hopkins shows the daily confirmed cases by country.

This case study was designed to drive home the important role that data science plays in real-world situations like this pandemic. This case study uses the Random Forest Classifier and a dataset from the South Korean cases of COVID-19 provided on [Kaggle](https://www.kaggle.com/kimjihoo/coronavirusdataset) to encourage research on this important topic. The goal of the case study is to build a Random Forest Classifier to predict the 'state' of the patient.

First, please load the needed packages and modules into Python. Next, load the data into a pandas dataframe for ease of use.

import os  
import pandas as pd  
from datetime import datetime,timedelta  
import seaborn as sns  
import matplotlib.pyplot as plt  
import numpy as np  
%matplotlib inline  
import plotly.graph\_objects as go  
from sklearn.experimental import enable\_iterative\_imputer  
from sklearn.impute import IterativeImputer  
from sklearn.ensemble import ExtraTreesRegressor

url ='SouthKoreacoronavirusdataset/PatientInfo.csv'  
df = pd.read\_csv(url)  
df.head(100)

patient\_id global\_num sex birth\_year age country province \  
0 1000000001 2.0 male 1964.0 50s Korea Seoul   
1 1000000002 5.0 male 1987.0 30s Korea Seoul   
2 1000000003 6.0 male 1964.0 50s Korea Seoul   
3 1000000004 7.0 male 1991.0 20s Korea Seoul   
4 1000000005 9.0 female 1992.0 20s Korea Seoul   
.. ... ... ... ... ... ... ...   
95 1000000096 4645.0 male 1982.0 30s Korea Seoul   
96 1000000097 4709.0 male 1985.0 30s Korea Seoul   
97 1000000098 4743.0 male 1966.0 50s Korea Seoul   
98 1000000099 5377.0 female 1996.0 20s Korea Seoul   
99 1000000100 5592.0 male 1985.0 30s Korea Seoul   
  
 city disease infection\_case infection\_order \  
0 Gangseo-gu NaN overseas inflow 1.0   
1 Jungnang-gu NaN overseas inflow 1.0   
2 Jongno-gu NaN contact with patient 2.0   
3 Mapo-gu NaN overseas inflow 1.0   
4 Seongbuk-gu NaN contact with patient 2.0   
.. ... ... ... ...   
95 Dobong-gu NaN etc NaN   
96 Yangcheon-gu NaN contact with patient NaN   
97 Seocho-gu NaN etc NaN   
98 Dongdaemun-gu NaN etc NaN   
99 Dongdaemun-gu NaN Dongan Church NaN   
  
 infected\_by contact\_number symptom\_onset\_date confirmed\_date \  
0 NaN 75.0 2020-01-22 2020-01-23   
1 NaN 31.0 NaN 2020-01-30   
2 2.002000e+09 17.0 NaN 2020-01-30   
3 NaN 9.0 2020-01-26 2020-01-30   
4 1.000000e+09 2.0 NaN 2020-01-31   
.. ... ... ... ...   
95 NaN NaN 2020-03-01 2020-03-02   
96 NaN NaN NaN 2020-03-01   
97 NaN NaN NaN 2020-03-02   
98 NaN NaN NaN 2020-03-01   
99 NaN NaN NaN 2020-03-04   
  
 released\_date deceased\_date state   
0 2020-02-05 NaN released   
1 2020-03-02 NaN released   
2 2020-02-19 NaN released   
3 2020-02-15 NaN released   
4 2020-02-24 NaN released   
.. ... ... ...   
95 NaN NaN isolated   
96 NaN NaN isolated   
97 NaN NaN isolated   
98 NaN NaN isolated   
99 NaN NaN isolated   
  
[100 rows x 18 columns]

df.shape

(2218, 18)

#Counts of null values   
na\_df=pd.DataFrame(df.isnull().sum().sort\_values(ascending=False)).reset\_index()  
na\_df.columns = ['VarName', 'NullCount']  
na\_df[(na\_df['NullCount']>0)]

VarName NullCount  
0 disease 2199  
1 deceased\_date 2186  
2 infection\_order 2176  
3 symptom\_onset\_date 2025  
4 released\_date 1995  
5 contact\_number 1807  
6 infected\_by 1749  
7 infection\_case 1055  
8 global\_num 904  
9 birth\_year 454  
10 age 261  
11 sex 145  
12 confirmed\_date 141  
13 state 88  
14 city 65

#counts of response variable values  
df.state.value\_counts()

state  
isolated 1791  
released 307  
deceased 32  
Name: count, dtype: int64

**Create a new column named 'n\_age' which is the calculated age based on the birth year column.**

import pandas as pd  
from datetime import datetime  
  
# Create a new column named 'n\_age' which is the calculated age based on the birth year column  
current\_year = datetime.now().year  
df['n\_age'] = current\_year - df['birth\_year']

### Handle Missing Values

**Print the number of missing values by column.**

print(df.isnull().sum())

patient\_id 0  
global\_num 904  
sex 145  
birth\_year 454  
age 261  
country 0  
province 0  
city 65  
disease 2199  
infection\_case 1055  
infection\_order 2176  
infected\_by 1749  
contact\_number 1807  
symptom\_onset\_date 2025  
confirmed\_date 141  
released\_date 1995  
deceased\_date 2186  
state 88  
n\_age 454  
dtype: int64

df.info()# Assuming df is your DataFrame

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 2218 entries, 0 to 2217  
Data columns (total 18 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 patient\_id 2218 non-null int64   
 1 global\_num 1314 non-null float64  
 2 sex 2073 non-null object   
 3 birth\_year 1764 non-null float64  
 4 age 1957 non-null object   
 5 country 2218 non-null object   
 6 province 2218 non-null object   
 7 city 2153 non-null object   
 8 disease 19 non-null object   
 9 infection\_case 1163 non-null object   
 10 infection\_order 42 non-null float64  
 11 infected\_by 469 non-null float64  
 12 contact\_number 411 non-null float64  
 13 symptom\_onset\_date 193 non-null object   
 14 confirmed\_date 2077 non-null object   
 15 released\_date 223 non-null object   
 16 deceased\_date 32 non-null object   
 17 state 2130 non-null object   
dtypes: float64(5), int64(1), object(12)  
memory usage: 312.0+ KB

**Fill the 'disease' missing values with 0 and remap the True values to 1.**

# Fill NaN with 0  
df['disease'].fillna(0, inplace=True)  
  
# Remap True to 1 and False to 0  
df['disease'] = df['disease'].map({True: 1, False: 0})

C:\Users\adlik\AppData\Local\Temp\ipykernel\_30600\4271920480.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df['disease'].fillna(0, inplace=True)

**Fill null values in the following columns with their mean: 'global\_number','birth\_year','infection\_order','infected\_by'and 'contact\_number'**

# List of columns to fill  
columns\_to\_fill = ['global\_num', 'birth\_year', 'infection\_order', 'infected\_by', 'contact\_number']  
  
# Fill NaN with column mean  
for column in columns\_to\_fill:  
 df[column].fillna(df[column].mean(), inplace=True)

C:\Users\adlik\AppData\Local\Temp\ipykernel\_10972\4091713880.py:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df[column].fillna(df[column].mean(), inplace=True)

**Fill the rest of the missing values with any method.**

print(df.isnull().sum())

patient\_id 0  
global\_num 0  
sex 145  
birth\_year 0  
age 261  
country 0  
province 0  
city 65  
disease 0  
infection\_case 1055  
infection\_order 0  
infected\_by 0  
contact\_number 0  
symptom\_onset\_date 2025  
confirmed\_date 141  
released\_date 1995  
deceased\_date 2186  
state 88  
n\_age 454  
dtype: int64

# Find columns with some NaN values  
nan\_columns = df.columns[df.isnull().any()].tolist()  
  
print(nan\_columns)

['age']

# Filling missing values with appropriate methods  
  
# Fill categorical missing values with mode (most frequent value)  
df['sex'].fillna("Unknown", inplace=True)  
df['age'].fillna("Unknown", inplace=True)  
df['city'].fillna("Unknown", inplace=True)  
df['state'].fillna("Unknown", inplace=True)  
  
# # Fill date-related missing values with a placeholder date (e.g., '1900-01-01' as an indicator of missing data)  
# df['symptom\_onset\_date'].fillna(pd.to\_datetime('1900-01-01'), inplace=True)  
# df['confirmed\_date'].fillna(pd.to\_datetime('1900-01-01'), inplace=True)  
# df['released\_date'].fillna(pd.to\_datetime('1900-01-01'), inplace=True)  
# df['deceased\_date'].fillna(pd.to\_datetime('1900-01-01'), inplace=True)  
  
df.head()

C:\Users\adlik\AppData\Local\Temp\ipykernel\_30600\2366854451.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df['sex'].fillna("Unknown", inplace=True)  
C:\Users\adlik\AppData\Local\Temp\ipykernel\_30600\2366854451.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df['age'].fillna("Unknown", inplace=True)  
C:\Users\adlik\AppData\Local\Temp\ipykernel\_30600\2366854451.py:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df['city'].fillna("Unknown", inplace=True)  
C:\Users\adlik\AppData\Local\Temp\ipykernel\_30600\2366854451.py:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df['state'].fillna("Unknown", inplace=True)

patient\_id global\_num sex birth\_year age country province \  
0 1000000001 2.0 male 1964.0 50s Korea Seoul   
1 1000000002 5.0 male 1987.0 30s Korea Seoul   
2 1000000003 6.0 male 1964.0 50s Korea Seoul   
3 1000000004 7.0 male 1991.0 20s Korea Seoul   
4 1000000005 9.0 female 1992.0 20s Korea Seoul   
  
 city disease infection\_case infection\_order infected\_by \  
0 Gangseo-gu 0 overseas inflow 1.0 NaN   
1 Jungnang-gu 0 overseas inflow 1.0 NaN   
2 Jongno-gu 0 contact with patient 2.0 2.002000e+09   
3 Mapo-gu 0 overseas inflow 1.0 NaN   
4 Seongbuk-gu 0 contact with patient 2.0 1.000000e+09   
  
 contact\_number symptom\_onset\_date confirmed\_date released\_date \  
0 75.0 2020-01-22 2020-01-23 2020-02-05   
1 31.0 NaN 2020-01-30 2020-03-02   
2 17.0 NaN 2020-01-30 2020-02-19   
3 9.0 2020-01-26 2020-01-30 2020-02-15   
4 2.0 NaN 2020-01-31 2020-02-24   
  
 deceased\_date state   
0 NaN released   
1 NaN released   
2 NaN released   
3 NaN released   
4 NaN released

**Check for any remaining null values.**

print(df.isnull().sum())

patient\_id 0  
global\_num 904  
sex 0  
birth\_year 454  
age 0  
country 0  
province 0  
city 0  
disease 0  
infection\_case 1055  
infection\_order 2176  
infected\_by 1749  
contact\_number 1807  
symptom\_onset\_date 2025  
confirmed\_date 141  
released\_date 1995  
deceased\_date 2186  
state 0  
dtype: int64

df.head()

patient\_id global\_num sex birth\_year age country province \  
0 1000000001 2.0 male 1964.0 Unknown Korea Seoul   
1 1000000002 5.0 male 1987.0 Unknown Korea Seoul   
2 1000000003 6.0 male 1964.0 Unknown Korea Seoul   
3 1000000004 7.0 male 1991.0 Unknown Korea Seoul   
4 1000000005 9.0 female 1992.0 Unknown Korea Seoul   
  
 city disease infection\_case infection\_order infected\_by \  
0 Gangseo-gu 0 overseas inflow 1.0 2.600789e+09   
1 Jungnang-gu 0 overseas inflow 1.0 2.600789e+09   
2 Jongno-gu 0 contact with patient 2.0 2.002000e+09   
3 Mapo-gu 0 overseas inflow 1.0 2.600789e+09   
4 Seongbuk-gu 0 contact with patient 2.0 1.000000e+09   
  
 contact\_number state n\_age   
0 75.0 released 61.0   
1 31.0 released 38.0   
2 17.0 released 61.0   
3 9.0 released 34.0   
4 2.0 released 33.0

Remove date columns from the data.

df = df.drop(['symptom\_onset\_date','confirmed\_date','released\_date','deceased\_date'],axis =1)

Review the count of unique values by column.

print(df.nunique())

patient\_id 2218  
global\_num 1303  
sex 3  
birth\_year 96  
age 12  
country 4  
province 17  
city 135  
disease 2  
infection\_case 16  
infection\_order 6  
infected\_by 206  
contact\_number 72  
state 4  
dtype: int64

Review the percent of unique values by column.

print(df.nunique()/df.shape[0])

patient\_id 1.000000  
global\_num 0.587917  
sex 0.001353  
birth\_year 0.043733  
age 0.000451  
country 0.001803  
province 0.007665  
city 0.060866  
disease 0.000902  
infection\_case 0.007665  
infection\_order 0.003156  
infected\_by 0.093327  
contact\_number 0.032913  
state 0.001803  
n\_age 0.043733  
dtype: float64

Review the range of values per column.

df.describe().T

count mean std min \  
patient\_id 2218.0 4.014678e+09 2.192419e+09 1.000000e+09   
global\_num 1314.0 4.664817e+03 2.874044e+03 1.000000e+00   
birth\_year 1764.0 1.974989e+03 1.941264e+01 1.916000e+03   
disease 2218.0 8.566276e-03 9.217769e-02 0.000000e+00   
infection\_order 42.0 2.285714e+00 1.254955e+00 1.000000e+00   
infected\_by 469.0 2.600789e+09 1.570638e+09 1.000000e+09   
contact\_number 411.0 2.412895e+01 9.108779e+01 0.000000e+00   
  
 25% 50% 75% max   
patient\_id 1.700000e+09 6.001000e+09 6.004000e+09 7.000000e+09   
global\_num 1.908500e+03 5.210500e+03 7.481500e+03 8.717000e+03   
birth\_year 1.962000e+03 1.974500e+03 1.993000e+03 2.020000e+03   
disease 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00   
infection\_order 1.250000e+00 2.000000e+00 3.000000e+00 6.000000e+00   
infected\_by 1.200000e+09 2.000000e+09 4.100000e+09 6.113000e+09   
contact\_number 2.000000e+00 5.000000e+00 1.600000e+01 1.160000e+03

### Check for duplicated rows

duplicated\_rows = df.duplicated()  
print("Number of duplicate rows: ", duplicated\_rows.sum())

Number of duplicate rows: 0

Print the categorical columns and their associated levels.

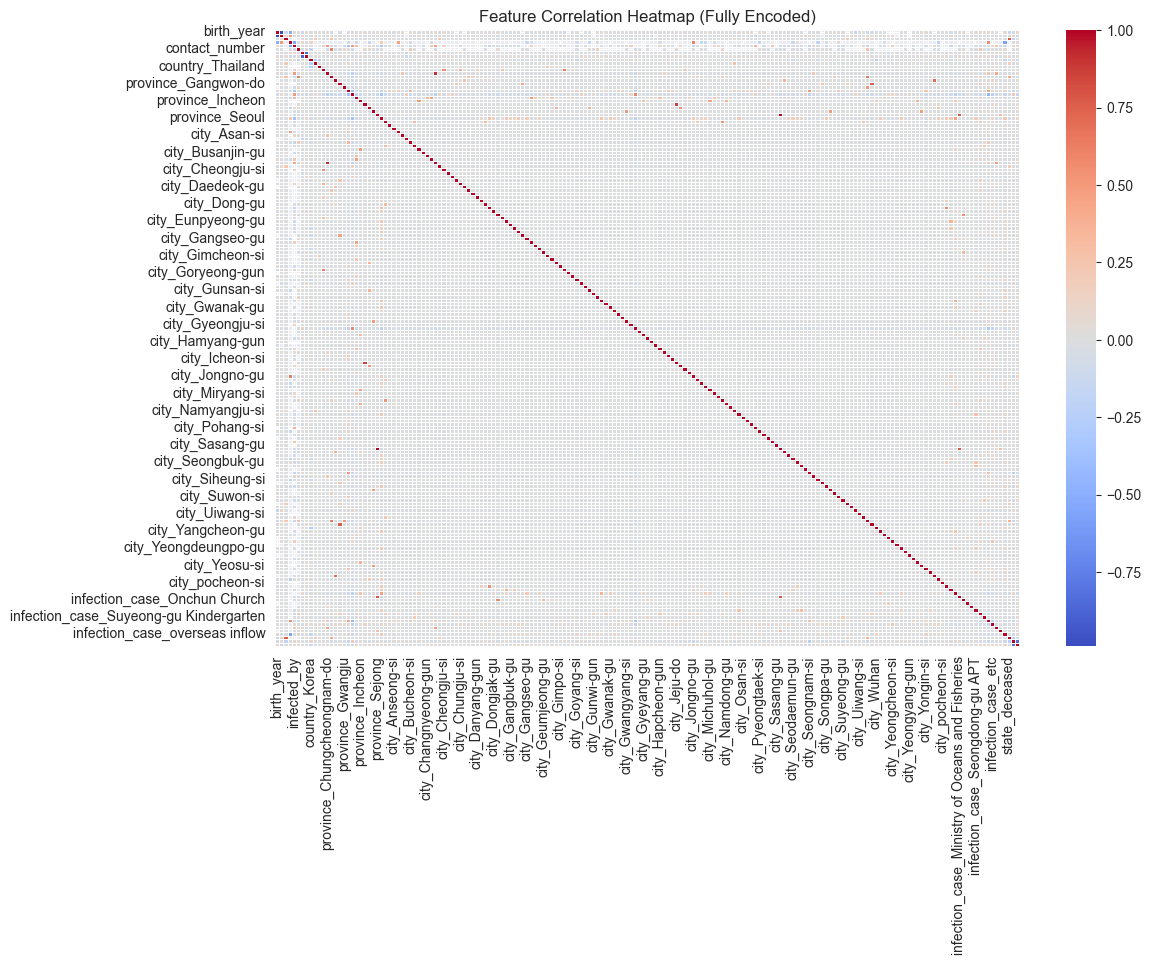
dfo = df.select\_dtypes(include=['object'], exclude=['datetime'])  
dfo.shape  
#get levels for all variables  
vn = pd.DataFrame(dfo.nunique()).reset\_index()  
vn.columns = ['VarName', 'LevelsCount']  
vn.sort\_values(by='LevelsCount', ascending =False)  
vn

VarName LevelsCount  
0 sex 3  
1 age 12  
2 country 4  
3 province 17  
4 city 135  
5 infection\_case 16  
6 state 4

def age\_to\_mean(age\_group):  
 if age\_group == '50s':  
 return 55  
 # Add all other age\_groups here  
 elif age\_group == '0s':  
 return 5  
 elif age\_group == '10s':  
 return 15  
 elif age\_group == '20s':  
 return 25  
 elif age\_group == '30s':  
 return 35  
 elif age\_group == '40s':  
 return 45  
 elif age\_group == '60s':  
 return 65  
 elif age\_group == '70s':  
 return 75  
 elif age\_group == '80s':  
 return 85  
 elif age\_group == '90s':  
 return 95  
 else:  
 return None # or some default value  
  
  
# Assuming 'age\_group' is the column with these values  
# Also assuming df and df['age'] is defined  
df['age'] = df['age'].apply(age\_to\_mean)

from sklearn.preprocessing import LabelEncoder  
  
# Assuming df is a pandas DataFrame and 'gender' is the column with values 'male' and 'female'  
labelencoder = LabelEncoder()  
dfo['sex'] = labelencoder.fit\_transform(dfo['sex'])  
  
# 'male' might be 0 and 'female' might be 1

# Ensure categorical columns are properly one-hot encoded, including 'sex'  
df\_encoded = df.drop(columns=['patient\_id', 'global\_num'])  
  
# Re-identify categorical columns for encoding  
categorical\_cols = ['sex', 'country', 'province', 'city', 'infection\_case', 'state']  
  
# Apply one-hot encoding to categorical columns  
df\_encoded = pd.get\_dummies(df\_encoded, columns=categorical\_cols, drop\_first=True)  
  
# Compute the correlation matrix  
corr\_matrix = df\_encoded.corr()  
  
# Plot the heatmap  
plt.figure(figsize=(12, 8))  
sns.heatmap(corr\_matrix, annot=False, cmap="coolwarm", linewidths=0.5)  
plt.title("Feature Correlation Heatmap (Fully Encoded)")  
plt.show()



df.head()

patient\_id global\_num sex birth\_year age country province \  
0 1000000001 2.0 male 1964.0 55.0 Korea Seoul   
1 1000000002 5.0 male 1987.0 35.0 Korea Seoul   
2 1000000003 6.0 male 1964.0 55.0 Korea Seoul   
3 1000000004 7.0 male 1991.0 25.0 Korea Seoul   
4 1000000005 9.0 female 1992.0 25.0 Korea Seoul   
  
 city disease infection\_case infection\_order infected\_by \  
0 Gangseo-gu 0 overseas inflow 1.0 NaN   
1 Jungnang-gu 0 overseas inflow 1.0 NaN   
2 Jongno-gu 0 contact with patient 2.0 2.002000e+09   
3 Mapo-gu 0 overseas inflow 1.0 NaN   
4 Seongbuk-gu 0 contact with patient 2.0 1.000000e+09   
  
 contact\_number state   
0 75.0 released   
1 31.0 released   
2 17.0 released   
3 9.0 released   
4 2.0 released

df.head()

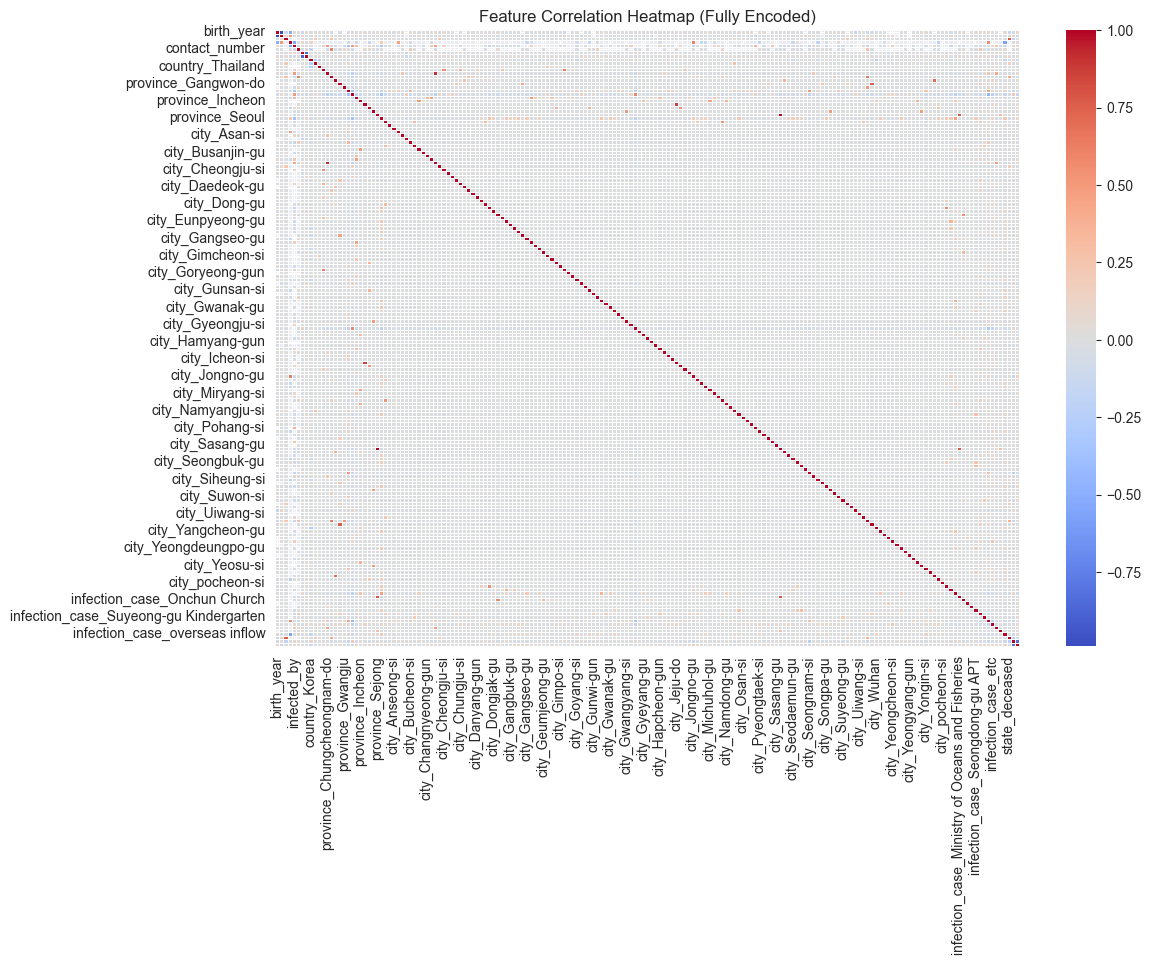
patient\_id global\_num sex birth\_year age country province \  
0 1000000001 2.0 male 1964.0 None Korea Seoul   
1 1000000002 5.0 male 1987.0 None Korea Seoul   
2 1000000003 6.0 male 1964.0 None Korea Seoul   
3 1000000004 7.0 male 1991.0 None Korea Seoul   
4 1000000005 9.0 female 1992.0 None Korea Seoul   
  
 city disease infection\_case infection\_order infected\_by \  
0 Gangseo-gu 0 overseas inflow 1.0 2.600789e+09   
1 Jungnang-gu 0 overseas inflow 1.0 2.600789e+09   
2 Jongno-gu 0 contact with patient 2.0 2.002000e+09   
3 Mapo-gu 0 overseas inflow 1.0 2.600789e+09   
4 Seongbuk-gu 0 contact with patient 2.0 1.000000e+09   
  
 contact\_number state n\_age   
0 75.0 released 61.0   
1 31.0 released 38.0   
2 17.0 released 61.0   
3 9.0 released 34.0   
4 2.0 released 33.0

df.columns

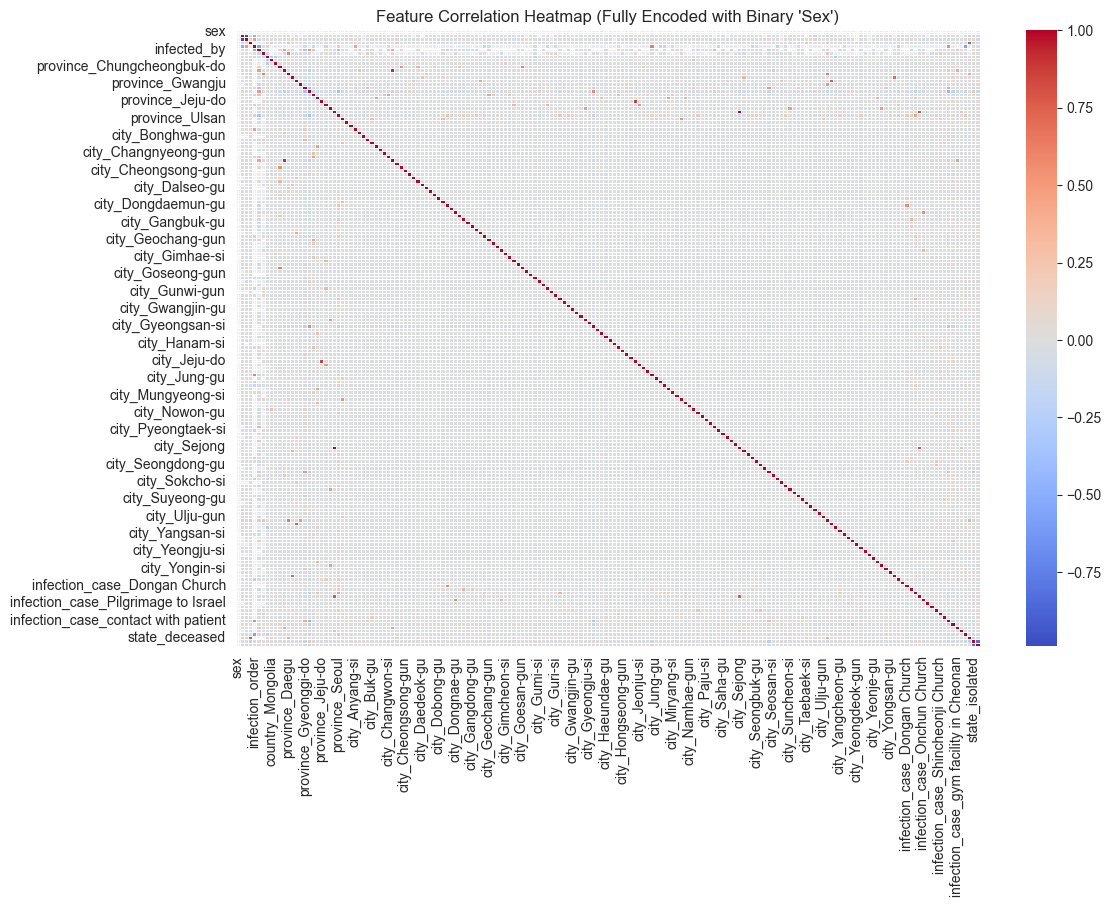
Index(['patient\_id', 'global\_num', 'sex', 'birth\_year', 'age', 'country',  
 'province', 'city', 'disease', 'infection\_case', 'infection\_order',  
 'infected\_by', 'contact\_number', 'state'],  
 dtype='object')

**Plot the correlation heat map for the features.**

# Ensure categorical columns are properly one-hot encoded, including 'sex'  
df\_encoded = df.drop(columns=['patient\_id', 'global\_num'])  
  
# Re-identify categorical columns for encoding  
categorical\_cols = ['sex', 'country', 'province', 'city', 'infection\_case', 'state']  
  
# Apply one-hot encoding to categorical columns  
df\_encoded = pd.get\_dummies(df\_encoded, columns=categorical\_cols, drop\_first=True)  
  
# Compute the correlation matrix  
corr\_matrix = df\_encoded.corr()  
  
# Plot the heatmap  
plt.figure(figsize=(12, 8))  
sns.heatmap(corr\_matrix, annot=False, cmap="coolwarm", linewidths=0.5)  
plt.title("Feature Correlation Heatmap (Fully Encoded)")  
plt.show()

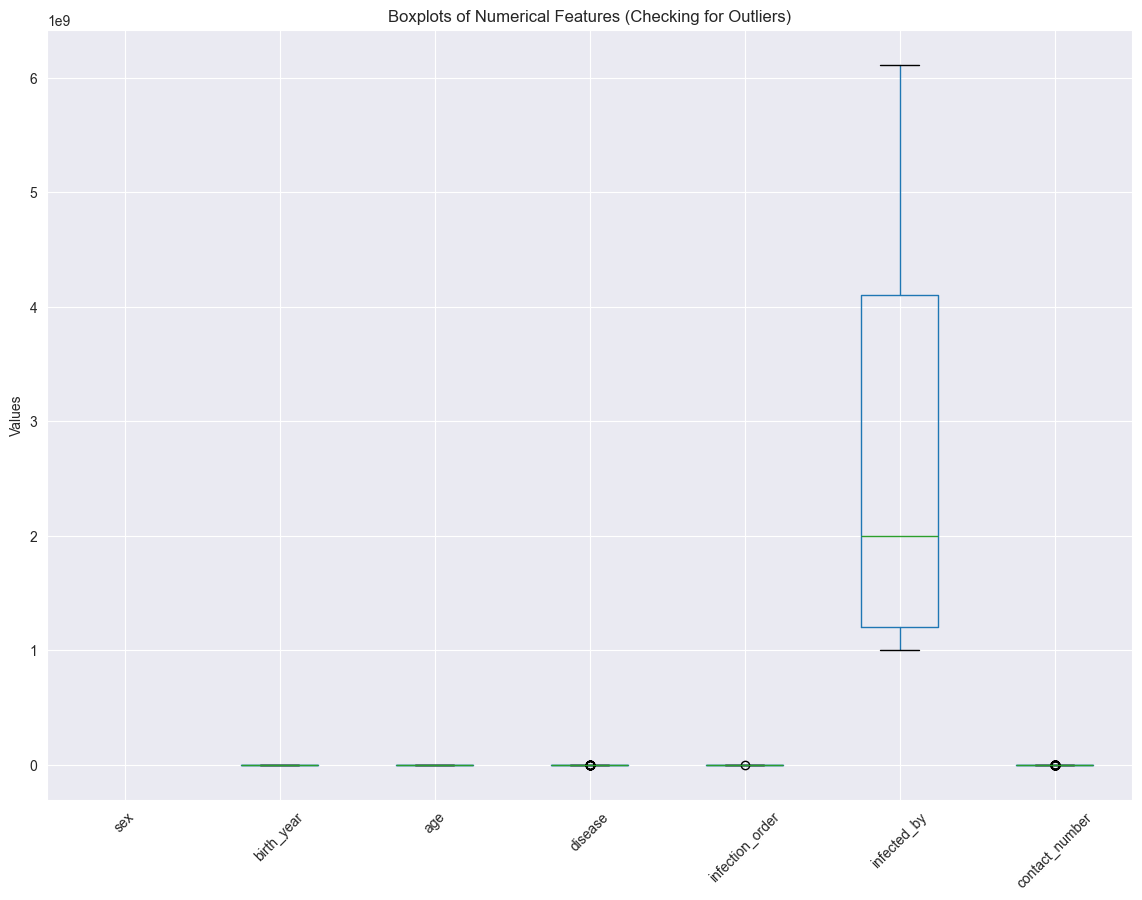


# Ensure 'sex' is properly encoded before correlation computation  
df\_encoded = df.drop(columns=['patient\_id', 'global\_num'])  
  
# Convert 'sex' to binary encoding (M → 1, F → 0) before one-hot encoding other categorical variables  
df\_encoded['sex'] = df\_encoded['sex'].map({'M': 1, 'F': 0})  
  
# Re-identify categorical columns excluding 'sex' (already encoded)  
categorical\_cols = ['country', 'province', 'city', 'infection\_case', 'state']  
  
# Apply one-hot encoding to remaining categorical columns  
df\_encoded = pd.get\_dummies(df\_encoded, columns=categorical\_cols, drop\_first=True)  
  
# Compute the correlation matrix  
corr\_matrix = df\_encoded.corr()  
  
# Plot the heatmap  
plt.figure(figsize=(12, 8))  
sns.heatmap(corr\_matrix, annot=False, cmap="coolwarm", linewidths=0.5)  
plt.title("Feature Correlation Heatmap (Fully Encoded with Binary 'Sex')")  
plt.show()



**Plot the boxplots to check for outliers.**

# Plot boxplots for numerical features to check for outliers  
numeric\_cols = df\_encoded.select\_dtypes(include=['number']).columns  
  
# Create boxplots for each numeric column  
plt.figure(figsize=(14, 10))  
df\_encoded[numeric\_cols].boxplot(rot=90) # Rotate labels for better readability  
  
plt.title("Boxplots of Numerical Features (Checking for Outliers)")  
plt.ylabel("Values")  
plt.xticks(rotation=45)  
plt.show()



**Create dummy features for object type features.**

# Identify object (categorical) columns  
categorical\_cols = df.select\_dtypes(include=['object']).columns.tolist()  
  
# Apply one-hot encoding to all categorical columns  
df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)  
  
# Display the updated dataset with dummy features  
df\_encoded

patient\_id global\_num birth\_year age disease infection\_order \  
0 1000000001 2.0 1964.0 55.0 0 1.0   
1 1000000002 5.0 1987.0 35.0 0 1.0   
2 1000000003 6.0 1964.0 55.0 0 2.0   
3 1000000004 7.0 1991.0 25.0 0 1.0   
4 1000000005 9.0 1992.0 25.0 0 2.0   
... ... ... ... ... ... ...   
2213 6100000085 NaN 1990.0 35.0 0 NaN   
2214 7000000001 139.0 1998.0 25.0 0 NaN   
2215 7000000002 222.0 1998.0 25.0 0 NaN   
2216 7000000003 4345.0 1972.0 45.0 0 NaN   
2217 7000000004 5534.0 1974.0 45.0 0 NaN   
  
 infected\_by contact\_number sex\_female sex\_male ... \  
0 NaN 75.0 False True ...   
1 NaN 31.0 False True ...   
2 2.002000e+09 17.0 False True ...   
3 NaN 9.0 False True ...   
4 1.000000e+09 2.0 True False ...   
... ... ... ... ... ...   
2213 NaN NaN False True ...   
2214 NaN 87.0 False True ...   
2215 NaN 84.0 True False ...   
2216 NaN 21.0 True False ...   
2217 NaN 74.0 False True ...   
  
 infection\_case\_Shincheonji Church \  
0 False   
1 False   
2 False   
3 False   
4 False   
... ...   
2213 False   
2214 False   
2215 False   
2216 False   
2217 False   
  
 infection\_case\_Suyeong-gu Kindergarten \  
0 False   
1 False   
2 False   
3 False   
4 False   
... ...   
2213 False   
2214 False   
2215 False   
2216 False   
2217 False   
  
 infection\_case\_contact with patient infection\_case\_etc \  
0 False False   
1 False False   
2 True False   
3 False False   
4 True False   
... ... ...   
2213 False False   
2214 False True   
2215 False True   
2216 False True   
2217 False True   
  
 infection\_case\_gym facility in Cheonan \  
0 False   
1 False   
2 False   
3 False   
4 False   
... ...   
2213 False   
2214 False   
2215 False   
2216 False   
2217 False   
  
 infection\_case\_gym facility in Sejong infection\_case\_overseas inflow \  
0 False True   
1 False True   
2 False False   
3 False True   
4 False False   
... ... ...   
2213 False False   
2214 False False   
2215 False False   
2216 False False   
2217 False False   
  
 state\_deceased state\_isolated state\_released   
0 False False True   
1 False False True   
2 False False True   
3 False False True   
4 False False True   
... ... ... ...   
2213 False False False   
2214 False True False   
2215 False False True   
2216 False False True   
2217 False True False   
  
[2218 rows x 181 columns]

### Split the data into test and train subsamples

from sklearn.model\_selection import train\_test\_split  
  
# dont forget to define your X and y  
from sklearn.model\_selection import train\_test\_split  
  
# Define features (X) and target variable (y)  
# 'disease' is the target variable  
X = df\_encoded.drop(columns=['disease']) # Features  
y = df\_encoded['disease'] # Target  
  
# Split the data into training (80%) and testing (20%) sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2, random\_state=1, stratify=y)  
  
# Display the shape of the train and test sets  
(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

((1774, 180), (444, 180), (1774,), (444,))

### Scale data to prep for model creation

#scale data  
from sklearn import preprocessing  
import numpy as np  
# build scaler based on training data and apply it to test data to then also scale the test data  
scaler = preprocessing.StandardScaler().fit(X\_train)  
X\_train\_scaled=scaler.transform(X\_train)  
X\_test\_scaled=scaler.transform(X\_test)

from sklearn.metrics import precision\_recall\_curve  
from sklearn.metrics import f1\_score  
from sklearn.metrics import auc  
from sklearn.linear\_model import LogisticRegression  
from matplotlib import pyplot  
from sklearn.metrics import precision\_recall\_curve  
from sklearn.metrics import f1\_score  
from sklearn.metrics import auc  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import classification\_report,confusion\_matrix,roc\_curve,roc\_auc\_score  
from sklearn.metrics import accuracy\_score,log\_loss  
from matplotlib import pyplot

### Fit Random Forest Classifier

The fit model shows an overall accuracy of 80% which is great and indicates our model was effectively able to identify the status of a patients in the South Korea dataset.

from sklearn.ensemble import RandomForestClassifier  
clf = RandomForestClassifier(n\_estimators=300, random\_state = 1,n\_jobs=-1)  
model\_res = clf.fit(X\_train\_scaled, y\_train)  
y\_pred = model\_res.predict(X\_test\_scaled)  
y\_pred\_prob = model\_res.predict\_proba(X\_test\_scaled)  
lr\_probs = y\_pred\_prob[:,1]  
ac = accuracy\_score(y\_test, y\_pred)  
  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
cm = confusion\_matrix(y\_test, y\_pred)  
  
print('Random Forest: Accuracy=%.3f' % (ac))  
  
print('Random Forest: f1-score=%.3f' % (f1))

Random Forest: Accuracy=0.991  
Random Forest: f1-score=0.989

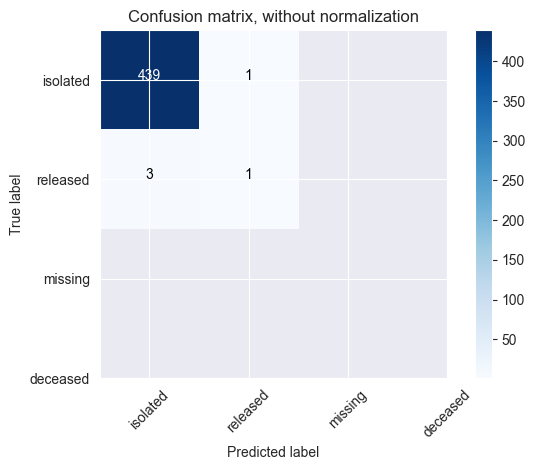
### Create Confusion Matrix Plots

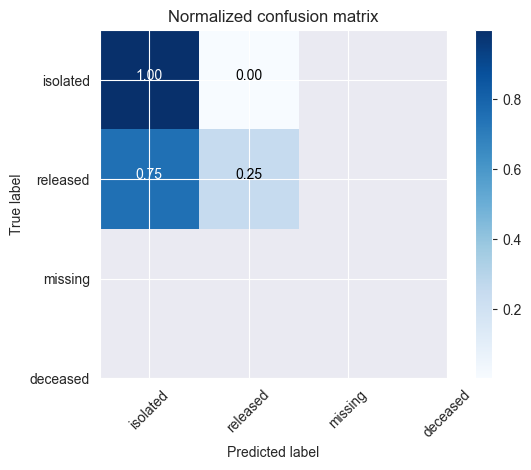
Confusion matrices are great ways to review your model performance for a multi-class classification problem. Being able to identify which class the misclassified observations end up in is a great way to determine if you need to build additional features to improve your overall model. In the example below we plot a regular counts confusion matrix as well as a weighted percent confusion matrix. The percent confusion matrix is particulary helpful when you have unbalanced class sizes.

class\_names=['isolated','released','missing','deceased'] # name of classes

import itertools  
import numpy as np  
import matplotlib.pyplot as plt  
  
from sklearn import svm, datasets  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix  
  
def plot\_confusion\_matrix(cm, classes,  
 normalize=False,  
 title='Confusion matrix',  
 cmap=plt.cm.Blues):  
 """  
 This function prints and plots the confusion matrix.  
 Normalization can be applied by setting `normalize=True`.  
 """  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Normalized confusion matrix")  
 else:  
 print('Confusion matrix, without normalization')  
  
 print(cm)  
  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 fmt = '.2f' if normalize else 'd'  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, format(cm[i, j], fmt),  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.tight\_layout()  
  
  
# Compute confusion matrix  
cnf\_matrix = confusion\_matrix(y\_test, y\_pred)  
np.set\_printoptions(precision=2)  
  
# Plot non-normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=class\_names,  
 title='Confusion matrix, without normalization')  
#plt.savefig('figures/RF\_cm\_multi\_class.png')  
  
# Plot normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=class\_names, normalize=True,  
 title='Normalized confusion matrix')  
#plt.savefig('figures/RF\_cm\_proportion\_multi\_class.png', bbox\_inches="tight")  
plt.show()

Confusion matrix, without normalization  
[[439 1]  
 [ 3 1]]  
Normalized confusion matrix  
[[1. 0. ]  
 [0.75 0.25]]



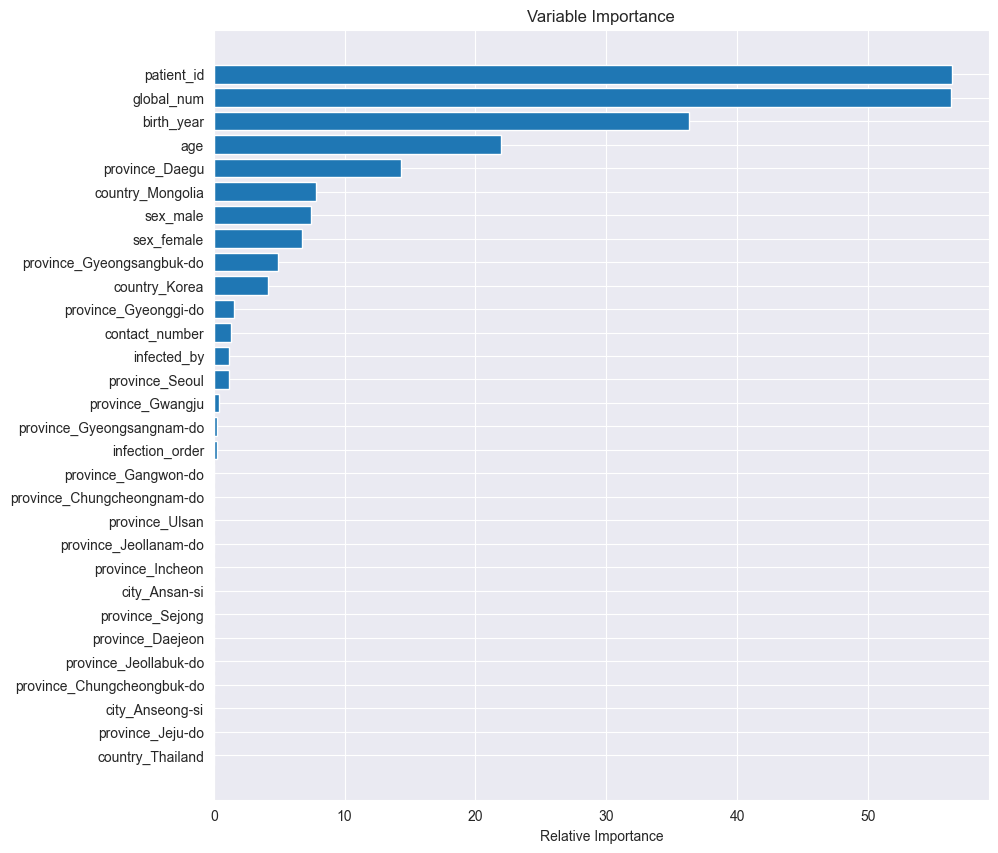


### Plot feature importances

The random forest algorithm can be used as a regression or classification model. In either case it tends to be a bit of a black box, where understanding what's happening under the hood can be difficult. Plotting the feature importances is one way that you can gain a perspective on which features are driving the model predictions.

feature\_importance = clf.feature\_importances\_  
# make importances relative to max importance  
feature\_importance = 100.0 \* (feature\_importance / feature\_importance.max())[:30]  
sorted\_idx = np.argsort(feature\_importance)[:30]  
  
pos = np.arange(sorted\_idx.shape[0]) + .5  
print(pos.size)  
sorted\_idx.size  
plt.figure(figsize=(10,10))  
plt.barh(pos, feature\_importance[sorted\_idx], align='center')  
plt.yticks(pos, X.columns[sorted\_idx])  
plt.xlabel('Relative Importance')  
plt.title('Variable Importance')  
plt.show()

30



The popularity of random forest is primarily due to how well it performs in a multitude of data situations. It tends to handle highly correlated features well, where as a linear regression model would not. In this case study we demonstrate the performance ability even with only a few features and almost all of them being highly correlated with each other. Random Forest is also used as an efficient way to investigate the importance of a set of features with a large data set. Consider random forest to be one of your first choices when building a decision tree, especially for multiclass classifications.