Title: Use food diet to predict deaths

Introduction: How can food and healthy diet help to improve healthy life and reduce the death rate? The current study indicated that a healthy diet is very important to prevent various infections. Keeping a healthy immune system is so important. However, little knowledge has been developed about which countries' deaths rate are associated with the different kinds of food intaking, neither do we know what is the importance of food with rich nutrition and improve eating habits to combat spreading diseases. This project uses three different machine learning algorithms to analyze the data of Food_Supply_kcal(percentage of energy intake). The model results should indicate how accurately the food diet can predict the countries' death rate. In addition, the ranking of feature importance should summarize the food and diet from high to low importance to predict deaths.

In [5]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn import metrics

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso, ElasticNet, LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error as MSE
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from xgboost.sklearn import XGBRegressor
from sklearn.preprocessing import MinMaxScaler ## feature scaling

import matplotlib.pyplot as plt # visualization
import seaborn as sns
from termcolor import colored as cl # text customization
import itertools # advanced tools
import plotly.graph_objs as go
import plotly.offline as py
import plotly.express as px


plt.rcParams['figure.figsize'] = (6, 4)
plt.rcParams['font.size'] = 14

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

## Loading data

In [6]:
```
data_path = '/Users/Zhang/notebooks/LearningFuze/'
image_path = '/Users/Zhang/notebooks/LearningFuze/output/'
```

In [7]:
```
death = pd.read_csv(data_path + 'data/Food_Supply_Quantity_kg_Data.csv')
```

In [8]:
```
death.head()
```

Out[8]:

| | Country | Alcoholic Beverages | Animal fats | Animal Products | Aquatic Products, Other | Cereals - Excluding Beer | Eggs | Fish, Seafood | Fruits - Excluding Wine | M |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Afghanistan | 0.0014 | 0.1973 | 9.4341 | 0.0 | 24.8097 | 0.2099 | 0.0350 | 5.3495 | 1.2 |
| **1** | Albania | 1.6719 | 0.1357 | 18.7684 | 0.0 | 5.7817 | 0.5815 | 0.2126 | 6.7861 | 1.8 |
| **2** | Algeria | 0.2711 | 0.0282 | 9.6334 | 0.0 | 13.6816 | 0.5277 | 0.2416 | 6.3801 | 1.1 |
| **3** | Angola | 5.8087 | 0.0560 | 4.9278 | 0.0 | 9.1085 | 0.0587 | 1.7707 | 6.0005 | 2.0 |
| **4** | Antigua and Barbuda | 3.5764 | 0.0087 | 16.6613 | 0.0 | 5.9960 | 0.2274 | 4.1489 | 10.7451 | 5.6 |

5 rows × 32 columns

In [9]:
```
death.dtypes
```

Out[9]:
```
Country                              object
Alcoholic Beverages                 float64
Animal fats                         float64
Animal Products                     float64
Aquatic Products, Other             float64
Cereals - Excluding Beer            float64
Eggs                                float64
Fish, Seafood                       float64
Fruits - Excluding Wine             float64
Meat                                float64
Milk - Excluding Butter             float64
Miscellaneous                       float64
Offals                              float64
Oilcrops                            float64
Pulses                              float64
Spices                              float64
Starchy Roots                       float64
Stimulants                          float64
Sugar & Sweeteners                  float64
Sugar Crops                         float64
Treenuts                            float64
Vegetable Oils                      float64
Vegetables                          float64
Vegetal Products                    float64
Obesity                             float64
Undernourished                       object
Confirmed                           float64
Deaths                              float64
Recovered                           float64
Active                              float64
Population                          float64
Unit (all except Population)         object
dtype: object
```

In [10]:
```
death.columns = ['Country', 'Alcoholic_beverages', 'Animal_fats', 'Animal_P
        'Aquatic_Products, Other', 'Cereals_Excluding_Beer', 'Eggs',\
        'Fish_Seafood', 'Fruits_Excluding Wine', 'Meat',\
        'Milk_Excluding Butter', 'Miscellaneous', 'Offals', 'Oilcrops',\
        'Pulses', 'Spices', 'Starchy Roots', 'Stimulants', 'Sugar_Sweeteners
        'Sugar_Crops', 'Treenuts', 'Vegetable Oils', 'Vegetables',\
        'Vegetal_Products', 'Obesity', 'Undernourished', 'Confirmed', 'Death
        'Recovered', 'Active', 'Population', 'Unit']
```

In [11]:
```
death.shape
```

Out[11]:
```
(170, 32)
```

In [12]:
```
death.columns
```

Out[12]:
```
Index(['Country', 'Alcoholic_beverages', 'Animal_fats', 'Animal_Product
s',
       'Aquatic_Products, Other', 'Cereals_Excluding_Beer', 'Eggs',
       'Fish_Seafood', 'Fruits_Excluding Wine', 'Meat',
       'Milk_Excluding Butter', 'Miscellaneous', 'Offals', 'Oilcrops',
       'Pulses', 'Spices', 'Starchy Roots', 'Stimulants', 'Sugar_Sweetene
rs',
       'Sugar_Crops', 'Treenuts', 'Vegetable Oils', 'Vegetables',
       'Vegetal_Products', 'Obesity', 'Undernourished', 'Confirmed', 'Dea
ths',
       'Recovered', 'Active', 'Population', 'Unit'],
      dtype='object')
```

## data cleaning and preprocessing

In [13]:
```
death.isnull().sum()
```

Out[13]:
```
Country                     0
Alcoholic_beverages         0
Animal_fats                 0
Animal_Products             0
Aquatic_Products, Other     0
Cereals_Excluding_Beer      0
Eggs                        0
Fish_Seafood                0
Fruits_Excluding Wine       0
Meat                        0
Milk_Excluding Butter       0
Miscellaneous               0
Offals                      0
Oilcrops                    0
Pulses                      0
Spices                      0
Starchy Roots               0
Stimulants                  0
Sugar_Sweeteners            0
Sugar_Crops                 0
Treenuts                    0
Vegetable Oils              0
Vegetables                  0
Vegetal_Products            0
Obesity                     3
Undernourished              7
Confirmed                   6
Deaths                      6
Recovered                   6
Active                      8
Population                  0
Unit                        0
dtype: int64
```

In [14]:
```
## plot the death to see any imbalanced data
## the provided death is continous variables, so the problems is to be appr
## the output are numerical variables, it CANNOT be the categorical variabl
## Classification of "0,1" cannot be meaningful in this case
## Use MSE, MAE, RMSE for metrics
```

In [15]:
```
### replace null value the mean value
death['Obesity'].fillna((death['Obesity'].mean()), inplace=True)
death['Confirmed'].fillna((death['Confirmed'].mean()), inplace=True)
death['Recovered'].fillna((death['Recovered'].mean()), inplace=True)
death['Deaths'].fillna((death['Deaths'].mean()), inplace=True)
```

In [16]:
```
death.head()
```

Out[16]:

| | Country | Alcoholic_beverages | Animal_fats | Animal_Products | Aquatic_Products, Other | Cereals_Excluc |
|---|---|---|---|---|---|---|
| **0** | Afghanistan | 0.0014 | 0.1973 | 9.4341 | 0.0 | |
| **1** | Albania | 1.6719 | 0.1357 | 18.7684 | 0.0 | |
| **2** | Algeria | 0.2711 | 0.0282 | 9.6334 | 0.0 | |
| **3** | Angola | 5.8087 | 0.0560 | 4.9278 | 0.0 | |
| **4** | Antigua and Barbuda | 3.5764 | 0.0087 | 16.6613 | 0.0 | |

5 rows × 32 columns

In [17]:
```
death.shape
```

Out[17]:
```
(170, 32)
```

In [18]:
```
## find the categorical variables
s = (death.dtypes == 'object')
object_cols = list(s[s].index)

print("Categorical variables:")
print(object_cols)
```

```
Categorical variables:
['Country', 'Undernourished', 'Unit']
```

In [19]:
```
## drop unit
## process the categorical variable "Country","unit" and "Undernourished" u
```

In [20]:
```
## observe the value of undernourished
death['Undernourished'].value_counts()
```

Out[20]:
```
<2.5    44
6.2      3
16.5     3
2.7      3
7.1      3
        ..
37.2     1
7.5      1
6.5      1
9.5      1
17       1
Name: Undernourished, Length: 98, dtype: int64
```

In [21]:
```
## replace the min to "2.5"
death['Undernourished_new'] = death['Undernourished'].str.replace('<2.5','2
```

In [22]:
```
death['Undernourished_new'] = pd.to_numeric(death['Undernourished'], errors
```

In [23]:
```
### use for loop to categorize
## def f(x):
##     if x == 1:
##         return 1
##     if (x >= 2.5) & (x < 10):
##         return 2
##     if (x >= 10) & (x < 20):
##         return 3
##     if x >= 20:
##         return 4
##     else:
##         return 'null'
## death['Undernourished_new'] = death.Undernourished_new.apply(f)
```

In [24]:
```
death['Undernourished_new'].fillna(death['Undernourished_new'].mean(), inpl
```

In [25]:
```
death.Undernourished_new.describe()
```

Out[25]:
```
count    170.000000
mean      14.457143
std       10.414519
min        2.500000
25%        7.275000
50%       14.457143
75%       14.457143
max       59.600000
Name: Undernourished_new, dtype: float64
```

In [26]:
```
death_new = death.drop(['Undernourished','Country','Unit'],axis = 1)
```

In [27]:
```
## use other ways to futher clean the data
```

In [28]:
```
death_new.drop_duplicates(inplace=True)
```

In [29]:
```
len(death_new.dropna())
```

Out[29]:
```
162
```

In [30]:
```
death_new.dropna(inplace=True)
```

In [31]:
```
death_new.isnull().sum()
```

Out[31]:
```
Alcoholic_beverages          0
Animal_fats                  0
Animal_Products              0
Aquatic_Products, Other      0
Cereals_Excluding_Beer       0
Eggs                         0
Fish_Seafood                 0
Fruits_Excluding Wine        0
Meat                         0
Milk_Excluding Butter        0
Miscellaneous                0
Offals                       0
Oilcrops                     0
Pulses                       0
Spices                       0
Starchy Roots                0
Stimulants                   0
Sugar_Sweeteners             0
Sugar_Crops                  0
Treenuts                     0
Vegetable Oils               0
Vegetables                   0
Vegetal_Products             0
Obesity                      0
Confirmed                    0
Deaths                       0
Recovered                    0
Active                       0
Population                   0
Undernourished_new           0
dtype: int64
```

In [32]:
```
death_new.shape ## data ready to use
```

Out[32]:
```
(162, 30)
```

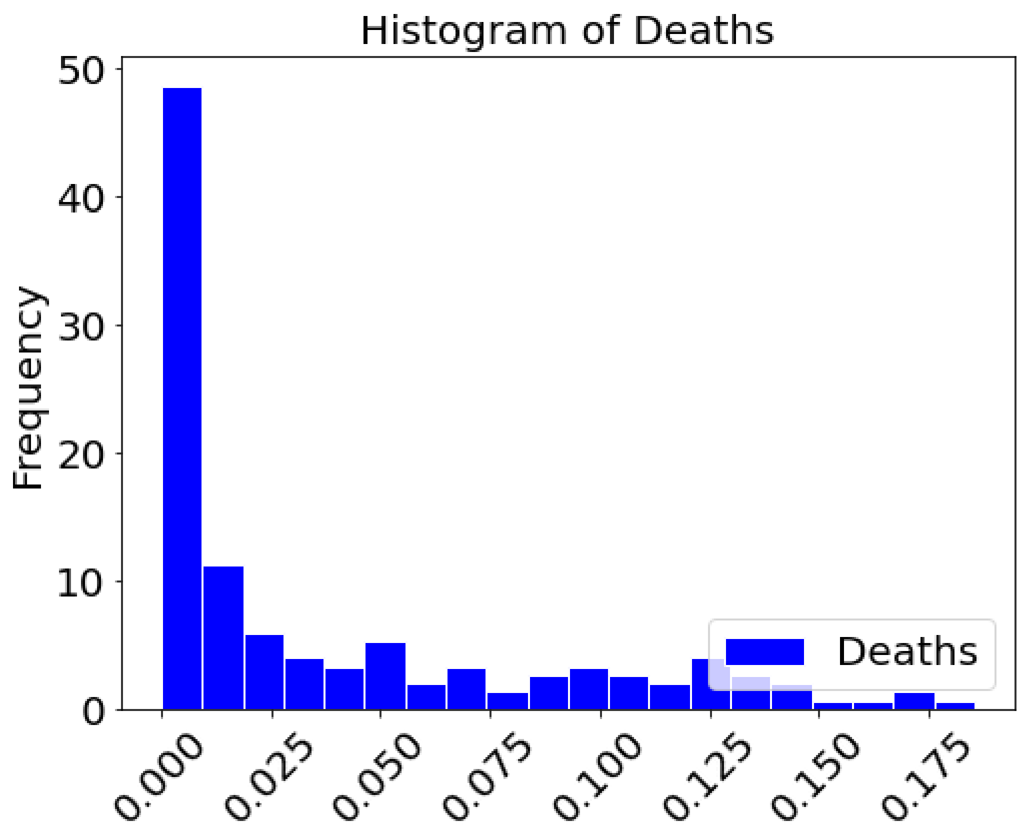
In [33]:
```
death_new.describe
```

Out[33]:
```
<bound method NDFrame.describe of      Alcoholic_beverages  Animal_fats  
Animal_Products  \
0                 0.0014       0.1973           9.4341
1                 1.6719       0.1357          18.7684
2                 0.2711       0.0282           9.6334
3                 5.8087       0.0560           4.9278
4                 3.5764       0.0087          16.6613
..                   ...          ...              ...
165               2.5952       0.0403          14.7565
166               1.4591       0.1640           8.5765
167               0.0364       0.0446           5.7874
168               5.7360       0.0829           6.0197
169               4.0552       0.0755           8.1489

     Aquatic_Products, Other  Cereals_Excluding_Beer    Eggs  Fish_Seafoo
d  \
0                     0.0000                 24.8097  0.2099         0.035
0
1                     0.0000                  5.7817  0.5815         0.212
6
2                     0.0000                 13.6816  0.5277         0.241
6
3                     0.0000                  9.1085  0.0587         1.770
7
4                     0.0000                  5.9960  0.2274         4.148
9
..                       ...                     ...     ...            
...
165                   0.0000                 12.9253  0.3389         0.945
6
166                   0.0042                 16.8740  0.3077         2.639
2
167                   0.0000                 27.2077  0.2579         0.524
0
168                   0.0000                 21.1938  0.3399         1.692
4
169                   0.0000                 22.6240  0.2678         0.551
8

     Fruits_Excluding Wine    Meat  Milk_Excluding Butter  ...   \
0                   5.3495  1.2020                 7.5828  ...
1                   6.7861  1.8845                15.7213  ...
2                   6.3801  1.1305                 7.6189  ...
3                   6.0005  2.0571                 0.8311  ...
4                  10.7451  5.6888                 6.3663  ...
..                     ...     ...                    ...  ...
165                 7.6460  3.8328                 9.3920  ...
166                 5.9029  4.4382                 0.6069  ...
167                 5.1344  2.7871                 1.8911  ...
168                 1.0183  1.8427                 1.7570  ...
169                 2.2000  2.6142                 4.4310  ...

     Vegetable Oils  Vegetables  Vegetal_Products  Obesity  Confirmed  \
0            0.5345      6.7642           40.5645      4.5   0.142134
1            0.3261     11.7753           31.2304     22.3   2.967301
```

```
2              1.0310     11.6484            40.3651     26.6    0.244897
3              0.6463      2.3041            45.0722      6.8    0.061687
4              0.8102      5.4495            33.3233     19.1    0.293878
..                ...         ...                ...      ...         ...
165            1.3734      4.1474            35.2416     25.2    0.452585
166            0.2201     11.9508            41.4232      2.1    0.002063
167            1.0811      3.2135            44.2126     14.1    0.007131
168            0.6657      3.4649            43.9789      6.5    0.334133
169            1.7103      2.3213            41.8526     12.3    0.232033

       Deaths  Recovered    Active  Population  Undernourished_new
0    0.006186   0.123374  0.012574  38928000.0           29.800000
1    0.050951   1.792636  1.123714   2838000.0            6.200000
2    0.006558   0.167572  0.070767  44357000.0            3.900000
3    0.001461   0.056808  0.003419  32522000.0           25.000000
4    0.007143   0.190816  0.095918     98000.0           14.457143
..        ...        ...       ...         ...                 ...
165  0.004287   0.424399  0.023899  28645000.0           21.200000
166  0.000036   0.001526  0.000501  96209000.0            9.300000
167  0.002062   0.004788  0.000282  29826000.0           38.900000
168  0.004564   0.290524  0.039045  18384000.0           46.700000
169  0.008854   0.190964  0.032214  14863000.0           51.300000

[162 rows x 30 columns]>
```

In [34]:
```
## feature scaling varies.. need further feature scaling
```

In [35]:
```
## plot the death rate
death_new.Deaths.plot(kind = 'hist', bins=20,density=True, figsize=(8,6),co
plt.title('Histogram of Death',size = 20);
plt.xlabel('',size = 20);
plt.ylabel('Frequency',size = 20);
plt.title('Histogram of Deaths',size = 20);
plt.xticks(size = 20,rotation=45)
plt.yticks(size = 20)
plt.legend(loc=4, prop={'size': 20})
plt.savefig(r'output/age_hist.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [36]:
```
## the distribution are extremely right skewed, majority of the data are cl
## transform the data to range of '0,1' use minmax.scale
```

## Feature engineering

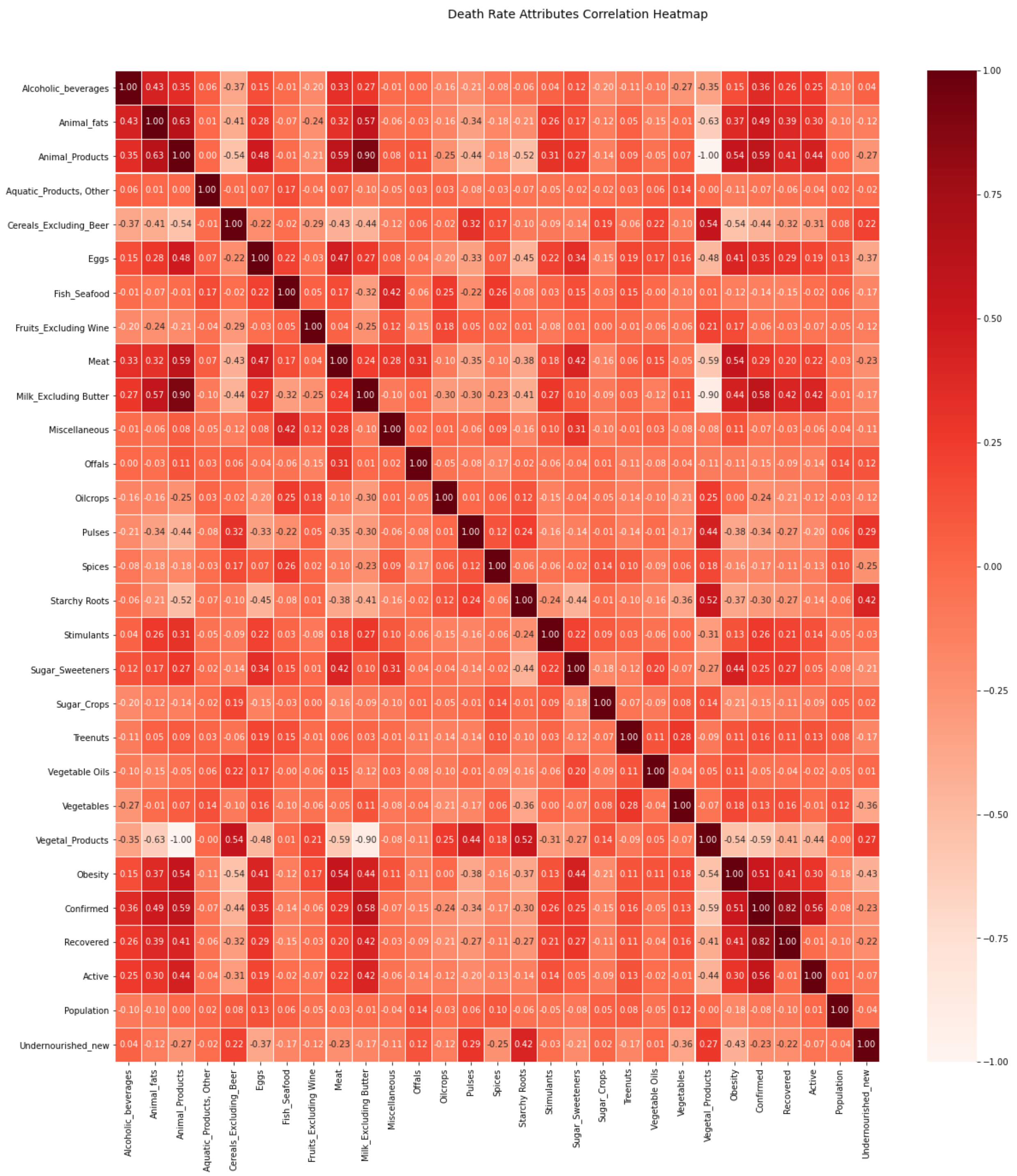
In [37]:
```
## correklation matrix
death_new.corr()
```

Out[37]:

| | Alcoholic_beverages | Animal_fats | Animal_Products | Aquatic_Products, Other | Cere |
|---|---|---|---|---|---|
| **Alcoholic_beverages** | 1.000000 | 0.433924 | 0.351646 | 0.060102 | |
| **Animal_fats** | 0.433924 | 1.000000 | 0.633236 | 0.005328 | |
| **Animal_Products** | 0.351646 | 0.633236 | 1.000000 | 0.000321 | |
| **Aquatic_Products, Other** | 0.060102 | 0.005328 | 0.000321 | 1.000000 | |
| **Cereals_Excluding_Beer** | -0.366784 | -0.410585 | -0.537847 | -0.009767 | |
| **Eggs** | 0.147727 | 0.275366 | 0.483142 | 0.073471 | |
| **Fish_Seafood** | -0.008807 | -0.067951 | -0.006359 | 0.168633 | |
| **Fruits_Excluding Wine** | -0.196503 | -0.241647 | -0.205165 | -0.042335 | |
| **Meat** | 0.328021 | 0.324346 | 0.585222 | 0.071402 | |
| **Milk_Excluding Butter** | 0.265818 | 0.573629 | 0.900917 | -0.096266 | |
| **Miscellaneous** | -0.006880 | -0.062259 | 0.079161 | -0.047654 | |
| **Offals** | 0.000236 | -0.032298 | 0.114030 | 0.025688 | |
| **Oilcrops** | -0.158279 | -0.158425 | -0.253903 | 0.025703 | |
| **Pulses** | -0.207427 | -0.342981 | -0.442456 | -0.076116 | |
| **Spices** | -0.083472 | -0.178545 | -0.184567 | -0.030962 | |
| **Starchy Roots** | -0.064524 | -0.214324 | -0.518404 | -0.067925 | |
| **Stimulants** | 0.043901 | 0.259924 | 0.309943 | -0.045133 | |
| **Sugar_Sweeteners** | 0.120665 | 0.168768 | 0.268206 | -0.016951 | |
| **Sugar_Crops** | -0.197143 | -0.123708 | -0.139790 | -0.021012 | |
| **Treenuts** | -0.105772 | 0.047142 | 0.087807 | 0.025005 | |
| **Vegetable Oils** | -0.097642 | -0.145675 | -0.053824 | 0.058724 | |
| **Vegetables** | -0.270554 | -0.013748 | 0.073080 | 0.135112 | |
| **Vegetal_Products** | -0.351720 | -0.633128 | -1.000000 | -0.000294 | |
| **Obesity** | 0.154923 | 0.374338 | 0.543293 | -0.107291 | |
| **Confirmed** | 0.362384 | 0.494776 | 0.593105 | -0.074362 | |
| **Deaths** | 0.404866 | 0.497139 | 0.537762 | -0.067495 | |
| **Recovered** | 0.262248 | 0.388815 | 0.414246 | -0.064659 | |
| **Active** | 0.250366 | 0.299180 | 0.436103 | -0.036188 | |
| **Population** | -0.099160 | -0.097362 | 0.002157 | 0.024388 | |
| **Undernourished_new** | 0.036051 | -0.115072 | -0.266866 | -0.016677 | |

30 rows × 30 columns

In [38]:
```python
# for visualizing correlations
f, ax = plt.subplots(figsize=(20, 20))
corr = death_new.drop(['Deaths'],axis = 1).corr()
sns.heatmap(round(corr,2), annot=True, ax=ax, cmap="Reds",fmt='.2f',linewid
f.subplots_adjust(top=0.93)
title = f.suptitle('Death Rate Attributes Correlation Heatmap', fontsize=14
```



In [39]:
```
## there are no variables that have a strong correlation to all other varia
## the results show strong correlations such as confirmed, active and veget
## other strong correlations such as cereals beers to Alcoholic, Miscellane
## the results show the poor correlations such as animal products and Misce
```

## Train the data

In [40]:
```
# visualize a minmax scaler transform of the sonar dataset
from pandas import read_csv
from pandas import DataFrame
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot
```

In [41]:
```
# split into inputs and outputs
feature_cols = death_new.columns.drop(['Deaths'])
X = death_new[feature_cols]

# Create response vector (y)
y = death_new.Deaths
```

In [42]:
```
## transform the features
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)
```

In [43]:
```
## use logarithmic to transform the y which is extremely skewed.
y_trans = np.sqrt(y)
```

In [ ]:
```
## from sklearn.preprocessing import MinMaxScaler
## y = MinMaxScaler().fit_transform(y.values.reshape(-1, 1))
```

In [ ]:
```
## transforamtion log square root
## to estimate, exp()
```

In [44]:
```
print(X.shape, y_trans.shape)
```

```
(162, 29) (162,)
```

In [45]:
```
## Create the train/test split.
X_train, X_test, y_train, y_test = train_test_split(X, y_trans, train_size=
```

In [46]:
```
X_train.shape
```

Out[46]:
```
(129, 29)
```

In [47]:
```
X_test.shape
```

Out[47]:
```
(33, 29)
```

```
In [48]: y_train.shape

Out[48]: (129,)

In [49]: y_test.shape

Out[49]: (33,)
```

## Machine Learning

```
In [50]: ## Ridge regression
         ## optimal value of alpha for ridge regression when All coefficients zero,

In [51]: ridge_alphas = np.logspace(0, 5, 200)

         optimal_ridge = RidgeCV(alphas=ridge_alphas, cv=10)
         optimal_ridge.fit(X_train, y_train)

         print(optimal_ridge.alpha_)

         30.36771118035459

In [52]: ## calculate R2 of ridge regression
         ## R2 is high model looks good

         train_preds = optimal_ridge.predict(X_train)
         optimal_ridge.score(X_train, y_train)

Out[52]: 0.7965181812120233
```

## cross-validate score

```
In [53]: ## calculate 10 folds Cross validation score and mean
         ridge_scores = cross_val_score(optimal_ridge, X_train, y_train, cv=10)

         print(ridge_scores)
         print(np.mean(ridge_scores))

         [0.69550047 0.47157677 0.68375667 0.55410032 0.78349861 0.73532483
          0.97805967 0.51176456 0.49372232 0.74108159]
         0.6648385819318452
```

## Model Evaluation

In [54]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sco

# use mean squared error, mean absolute error and R2 to evaluate the model.
def show_scores(model, X_train, X_test, y_train, y_test):
    train_preds = model.predict(X_train)
    test_preds = model.predict(X_test)
    scores = {'Training MAE': mean_absolute_error(y_train, train_preds),
              'Test MAE': mean_absolute_error(y_test, test_preds),
              'Training MSE': mean_squared_error(y_train, train_preds),
              'Test MSE': mean_squared_error(y_test, test_preds),
              'Training R^2': r2_score(y_train, train_preds),
              'Test R^2': r2_score(y_test, test_preds)}
    return scores
```

In [55]:
```
show_scores(optimal_ridge, X_train, X_test , y_train, y_test)
```

Out[55]:
```
{'Training MAE': 0.040523980899040454,
 'Test MAE': 0.04712884384545657,
 'Training MSE': 0.002915215754985201,
 'Test MSE': 0.0035512516031912336,
 'Training R^2': 0.7965181812120233,
 'Test R^2': 0.8019102492715857}
```

In [56]:
```
## use other models
## create pipeline
from sklearn.pipeline import Pipeline

regressor = Pipeline([
    ('scaler', StandardScaler()),
    ('estimator', Ridge(random_state=28))
])
```

## Machine learning

In [57]:
```
## introduced other models of Support vector regressor and random forest
## remove SRV replace KNN or Decision tree

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost.sklearn import XGBRegressor

# First, we create a dict with our desired models
models = {'Ridge':Ridge(random_state=28),
          'Tree_model':DecisionTreeRegressor(max_depth=3, random_state=1),
          'RandomForest':RandomForestRegressor(),
          'XGBoost':XGBRegressor(n_estimators = 1000, learning_rate = 0.05)

# Now to build the function that tests each model
def model_build(model, X_train, y_train, X_test, y_test, scale=True):

    if scale:
        regressor = Pipeline([
            ('scaler', StandardScaler()),
            ('estimator', model)
        ])

    else:
        regressor = Pipeline([
            ('estimator', model)
        ])

    # Training
    regressor.fit(X_train, y_train)

    # Scoring the training set

    train_preds = regressor.predict(X_train)
    print(f"R2 on single split: {regressor.score(X_train, y_train)}")

    # Cross validate
    cv_score = cross_val_score(regressor, X_train, y_train, cv = 10)

    print(f"Cross validate R2 score: {cv_score.mean()}")

    # Scoring the test set
    for k, v in show_scores(regressor, X_train, X_test , y_train, y_test).i
        print("     ", k, v)
```

In [58]:
```
for name, model in models.items():
    print(f"==== Scoring {name} model====")

    if name == 'RandomForest' or name == 'XGBoost' or name == 'Tree_model':
        model_build(model, X_train, y_train, X_test, y_test, scale=False)
    else:
        model_build(model, X_train, y_train, X_test, y_test,)
    print()
    print(40*"=")
```

```
==== Scoring Ridge model====
R2 on single split: 0.8124245223869887
Cross validate R2 score: 0.3919834789170463
      Training MAE 0.03857228641175126
      Test MAE 0.054205418755947664
      Training MSE 0.0026873309411299374
      Test MSE 0.004710726811744006
      Training R^2 0.8124245223869887
      Test R^2 0.7372344164380014

========================================
==== Scoring Tree_model model====
R2 on single split: 0.9131929989271381
Cross validate R2 score: 0.7035976265534025
      Training MAE 0.026019425475613783
      Test MAE 0.04481547960948823
      Training MSE 0.0012436547829086787
      Test MSE 0.004073215006498119
      Training R^2 0.9131929989271381
      Test R^2 0.7727949930172833

========================================
==== Scoring RandomForest model====
R2 on single split: 0.9762874291078611
Cross validate R2 score: 0.7740692735313153
      Training MAE 0.012782556785800333
      Test MAE 0.036550666777351505
      Training MSE 0.00033972204822876976
      Test MSE 0.0026455966958288035
      Training R^2 0.9762874291078611
      Test R^2 0.8524279187839839

========================================
==== Scoring XGBoost model====
R2 on single split: 0.999977856129166
Cross validate R2 score: 0.7442703358353759
      Training MAE 0.00043063677841475 77
      Test MAE 0.03862432091701861
      Training MSE 3.172478087537164e-07
      Test MSE 0.003034460330831356
      Training R^2 0.999977856129166
      Test R^2 0.8307370026980102

========================================
```

In [77]:
```
## calculate the RMSE of all models
Ridge_RMSE = np.sqrt(0.004710726811744006)
Tree_RMSE = np.sqrt(0.004073215006498119)
RF_RMSE = np.sqrt(0.0026455966958288035)
XGBoost_RMSE = np.sqrt(0.003034460330831356)
```

In [78]:
```
print(Ridge_RMSE)
print(Tree_RMSE)
print(RF_RMSE)
print(XGBoost_RMSE)
```

```
0.06863473473208741
0.0638217439944892
0.05143536425290292
0.05508593587143052
```

In [61]:
```
## the results indicate the XGboosting has the lowest test MSE and test MAE
## the next is to hyperparameters tunning for XGBoosting
```

## Hyperparameters Tunning

In [79]:
```
## set up the pamareters for regression

xgb = XGBRegressor()

parameters = {'nthread':[4],
              'objective':['reg:squarederror'],
              'learning_rate': [.03, 0.05, .07], #so called `eta` value
              'max_depth': [5, 6, 7],
              'min_child_weight': [4],
              'subsample': [0.7],
              'colsample_bytree': [0.7],
              'n_estimators': [500, 1000]}
```

In [80]:
```
## use grid serve to find the best grid

from sklearn.model_selection import GridSearchCV

xgb_grid = GridSearchCV(xgb, parameters, cv = 5, n_jobs = 4, verbose = True

xgb_grid.fit(X_train, y_train)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
0.8160060853703651
{'colsample_bytree': 0.7, 'learning_rate': 0.07, 'max_depth': 5, 'min_chi
ld_weight': 4, 'n_estimators': 1000, 'nthread': 4, 'objective': 'reg:squa
rederror', 'subsample': 0.7}
```

In [64]:
```
xgb_best = XGBRegressor(colsample_bytree = 0.7,
                        learning_rate = 0.05,
                        max_depth = 5,
                        min_child_weight = 4,
                        n_estimators = 1000,
                        nthread = 4,
                        objective = 'reg:squarederror',
                        subsample = 0.7)
```

In [81]:
```
model_build(xgb_best, X_train, y_train, X_test, y_test, scale=False)
```

```
R2 on single split: 0.9999892957100863
Cross validate R2 score: 0.7652098308783224
      Training MAE 0.00028740662751354745
      Test MAE 0.03553743053660788
      Training MSE 1.5335677058625871e-07
      Test MSE 0.0026104127632680177
      Training R^2 0.9999892957100863
      Test R^2 0.8543904878186163
```

In [82]:
```
## calculate RMSE
Test_RMSE = np.sqrt(0.0026104127632680177)
print(Test_RMSE)
```

```
0.051092198653688976
```

In [83]:
```
## plot feature importance graph
```

In [84]:
```
## re-define the tree model
Tree_model = DecisionTreeRegressor(max_depth=3, random_state=1)
Tree_model.fit(X_train, y_train)
```

Out[84]:
```
DecisionTreeRegressor(max_depth=3, random_state=1)
```

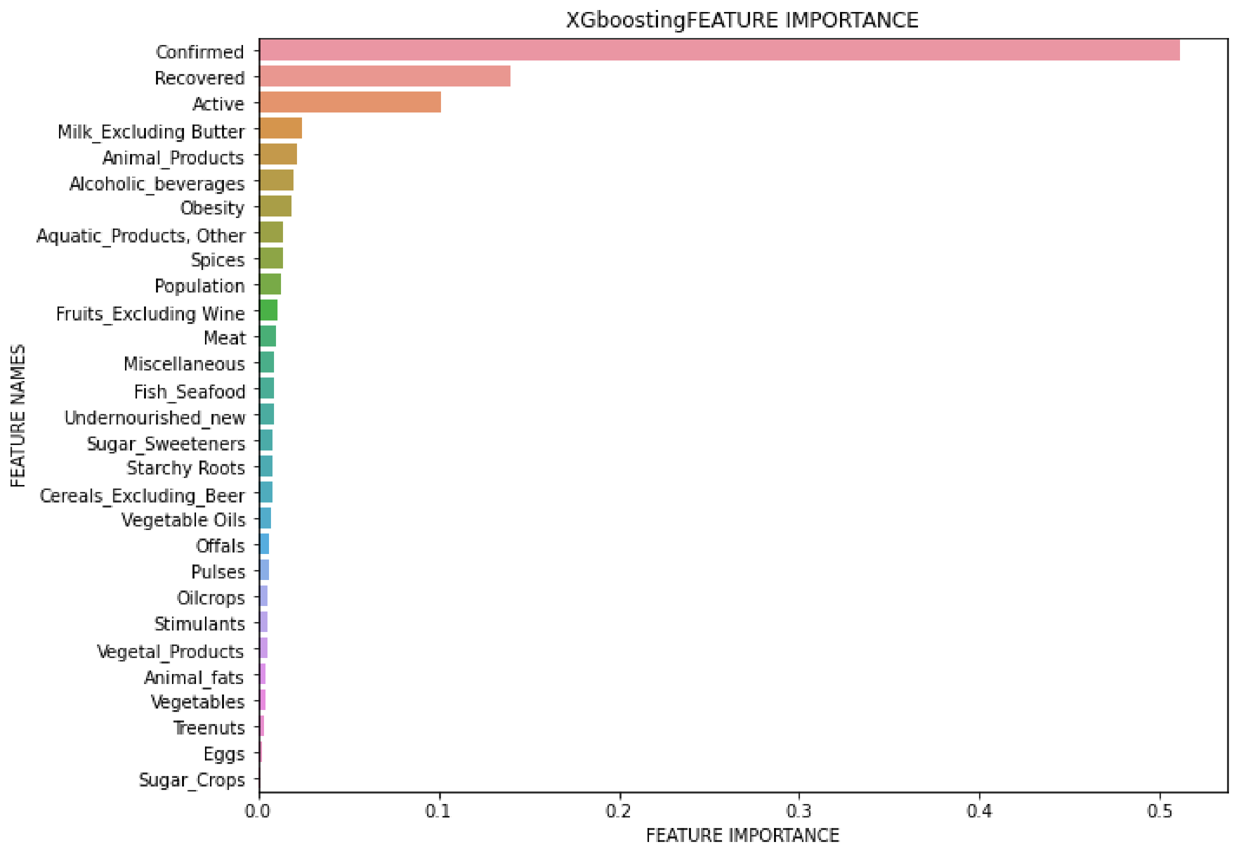
In [85]:
```
def plot_feature_importance(importance,names,model_type):

    feature_importance = np.array(importance)
    feature_names = np.array(names)

    data={'feature_names':feature_names,'feature_importance':feature_import
    df = pd.DataFrame(data)

    df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    #Define size of bar plot
    plt.figure(figsize=(10,8))
    #Plot Searborn bar chart
    sns.barplot(x=df['feature_importance'], y=df['feature_names'])
    #Add chart labels
    plt.title(model_type + 'FEATURE IMPORTANCE')
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
```

In [86]:
```
## feature importance of XGboosting
pd.DataFrame({'feature':feature_cols, 'importance':xgb_best.feature_importa
```

Out[86]:

| | feature | importance |
|---|---|---|
| **24** | Confirmed | 0.511461 |
| **25** | Recovered | 0.140178 |
| **26** | Active | 0.100948 |
| **9** | Milk_Excluding Butter | 0.024808 |
| **2** | Animal_Products | 0.021917 |
| **0** | Alcoholic_beverages | 0.019346 |
| **23** | Obesity | 0.018556 |
| **3** | Aquatic_Products, Other | 0.013735 |
| **14** | Spices | 0.013439 |
| **27** | Population | 0.012495 |
| **7** | Fruits_Excluding Wine | 0.011196 |
| **8** | Meat | 0.010230 |
| **10** | Miscellaneous | 0.009274 |
| **6** | Fish_Seafood | 0.009211 |
| **28** | Undernourished_new | 0.008523 |
| **17** | Sugar_Sweeteners | 0.008042 |
| **15** | Starchy Roots | 0.007918 |
| **4** | Cereals_Excluding_Beer | 0.007822 |
| **20** | Vegetable Oils | 0.007349 |
| **11** | Offals | 0.006425 |
| **13** | Pulses | 0.006043 |
| **12** | Oilcrops | 0.005265 |
| **16** | Stimulants | 0.005263 |
| **22** | Vegetal_Products | 0.004999 |
| **1** | Animal_fats | 0.004296 |
| **21** | Vegetables | 0.004254 |
| **19** | Treenuts | 0.003486 |
| **5** | Eggs | 0.002472 |
| **18** | Sugar_Crops | 0.001051 |

In [87]:
```
plot_feature_importance(xgb_best.feature_importances_,feature_cols,'XGboost
```

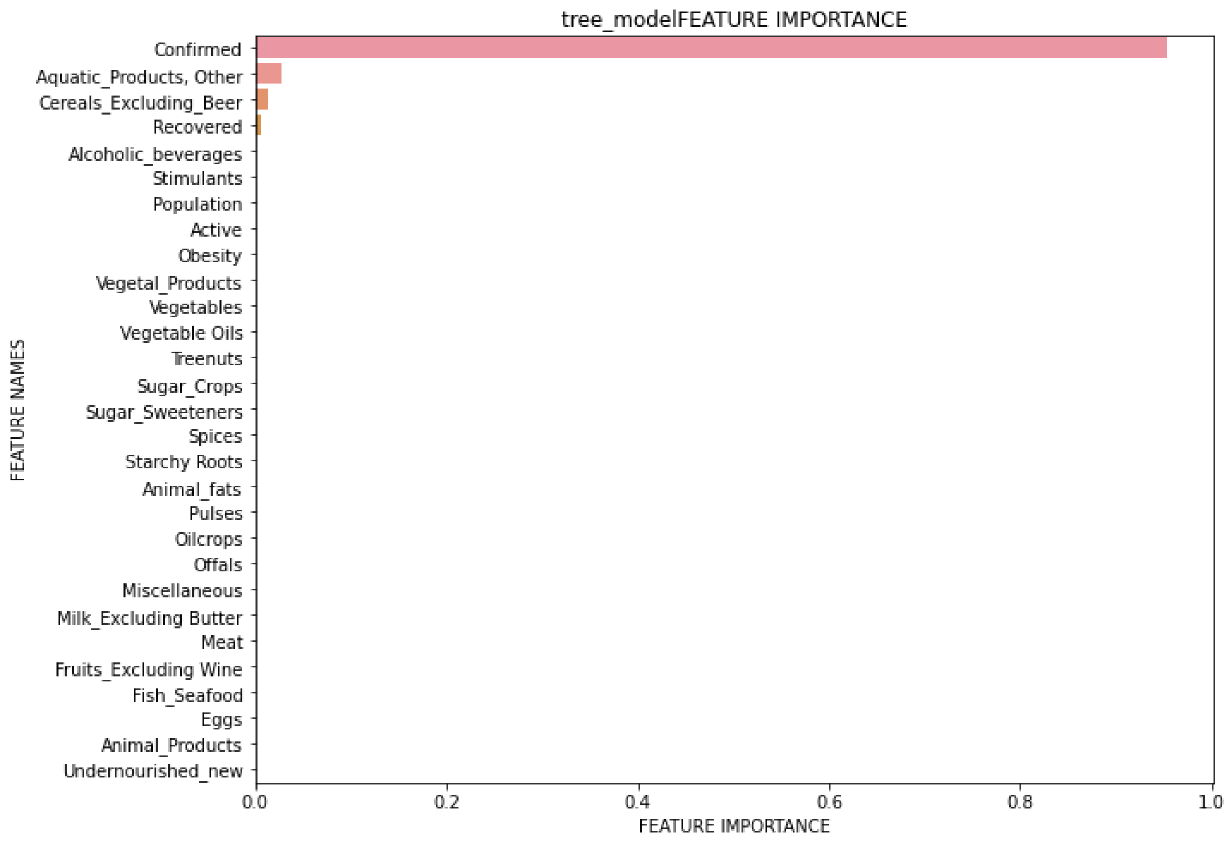


In [88]:
```
## feature importance of tree
pd.DataFrame({'feature':feature_cols, 'importance':Tree_model.feature_impor
```

Out[88]:

| | feature | importance |
|---|---|---|
| 24 | Confirmed | 0.953770 |
| 3 | Aquatic_Products, Other | 0.026582 |
| 4 | Cereals_Excluding_Beer | 0.012880 |
| 25 | Recovered | 0.006769 |
| 0 | Alcoholic_beverages | 0.000000 |
| 16 | Stimulants | 0.000000 |
| 27 | Population | 0.000000 |
| 26 | Active | 0.000000 |
| 23 | Obesity | 0.000000 |
| 22 | Vegetal_Products | 0.000000 |
| 21 | Vegetables | 0.000000 |
| 20 | Vegetable Oils | 0.000000 |
| 19 | Treenuts | 0.000000 |
| 18 | Sugar_Crops | 0.000000 |
| 17 | Sugar_Sweeteners | 0.000000 |
| 14 | Spices | 0.000000 |
| 15 | Starchy Roots | 0.000000 |
| 1 | Animal_fats | 0.000000 |
| 13 | Pulses | 0.000000 |
| 12 | Oilcrops | 0.000000 |
| 11 | Offals | 0.000000 |
| 10 | Miscellaneous | 0.000000 |
| 9 | Milk_Excluding Butter | 0.000000 |
| 8 | Meat | 0.000000 |
| 7 | Fruits_Excluding Wine | 0.000000 |
| 6 | Fish_Seafood | 0.000000 |
| 5 | Eggs | 0.000000 |
| 2 | Animal_Products | 0.000000 |
| 28 | Undernourished_new | 0.000000 |

In [89]:
```
plot_feature_importance(Tree_model.feature_importances_,feature_cols,'tree_
```



In [90]:
```
## create tree graph
```

In [91]:
```
from sklearn.tree import export_graphviz
from sklearn import tree

export_graphviz(Tree_model, out_file='tree.dot', feature_names=feature_cols
```

In [92]:
```
import pydotplus

tree = tree.export_graphviz(Tree_model, out_file=None)
graph = pydotplus.graph_from_dot_data(tree)
graph.write_jpg("output/food_deaths_tree.jpg")
```

Out[92]:
```
True
```

## Conclusion and Next step

The purpose of the study is to use the food diet and nutrition of countries to predict the death rate. The current study used four machine learning algorithms to predict the results. The ridge regression results show the 10 folds cross-validation score is relatively high in the range of .48 ~.98. the average cross-validation score is .66. Test data MAE .047, and test MSE is 0.004. In comparing other three different models including decision tree, random forest, ensembled random forest, and XGboosting, the results of XGboosting seems the best with the lowest RMAE(the root of mean absolute error) and RMSE (root fo mean squared error). Further hyperparameter tunning is applied. The grid search is used to find the best parameter. The results of XGboosting indicated the test RMSE =0.05. The feature importance was the plot for the decision tree and XGboosting. The XGBoosting model shows the ranking of the important features are confirmed, recovered, Active, Fruit_excluding wines and Milk_excluding butter, etc. The decision tree shows the ranking of the important features are confirmed, Acquatic_product, undernourished and recovered, etc. The decision tree shows at the confirmed rate <-.238, the predicted deaths are likely to be true. on the nodes of recovered of value -.09, 79 samples are assigned to two tree leaves, among them 68 cases predicted deaths are more likely to die if the confirmed <.-753, on the other hand, the leftover 11 cases who are predicted deaths are likely to be dead at the point of fruit_excluding wine is <.023

## Next step

From the results of the decision tree and XGboosting results, only a few features are important such as confirmed, Aquatic_products, undernourished, and recovery. Next step I will continue to adjust the model with only useful features and rerun the model for improved model fit.