```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import datetime

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import jaccard_score
from sklearn.metrics import balanced_accuracy_score

from sklearn.preprocessing import StandardScaler

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

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from math import sqrt

```

```
url_country = 'https://raw.githubusercontent.com/Advait177013/CSE4020_ML_JComp/main/clear
url_income = 'https://raw.githubusercontent.com/Advait177013/CSE4020_ML_JComp/main/cleane
url_region = 'https://raw.githubusercontent.com/Advait177013/CSE4020_ML_JComp/main/cleane
df_country = pd.read_csv(url_country)
df_income = pd.read_csv(url_income)
df_region = pd.read_csv(url_region)
```

Team Members :

Advait Deochakke

Harish T R

Shah Siddh

```
#Code by Advait

#creating a copy to facilitate operations
df=df_country.copy()

#changing the date format to a format which we can easily process
#from complex string which COULD be converted to categories
# → to easily processable integers
```

```
df['date'] = pd.to_datetime(df['date'])
df['date'] = df['date'].dt.strftime('%Y.%m.%d')
df['year'] = pd.DatetimeIndex(df['date']).year
df['month'] = pd.DatetimeIndex(df['date']).month
df['day'] = pd.DatetimeIndex(df['date']).day

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 151164 entries, 0 to 151163
Data columns (total 22 columns):
 #   Column                                Non-Null Count   Dtype  
---  ------                                --------------   -----  
 0   Unnamed: 0                            151164 non-null  int64  
 1   iso_code                              151164 non-null  object 
 2   location                              151164 non-null  object 
 3   date                                  151164 non-null  object 
 4   total_cases                           151164 non-null  float64
 5   total_deaths                          151164 non-null  int64  
 6   total_cases_per_million               151164 non-null  float64
 7   total_deaths_per_million              151164 non-null  float64
 8   total_tests                           151164 non-null  float64
 9   total_tests_per_thousand              151164 non-null  float64
 10  positive_rate                         151164 non-null  float64
 11  people_vaccinated                     151164 non-null  float64
 12  people_fully_vaccinated               151164 non-null  float64
 13  people_fully_vaccinated_per_hundred   151164 non-null  float64
 14  population                            151164 non-null  int64  
 15  population_density                    151164 non-null  float64
 16  median_age                            151164 non-null  float64
 17  gdp_per_capita                        151164 non-null  float64
 18  human_development_index               151164 non-null  float64
 19  year                                  151164 non-null  int64  
 20  month                                 151164 non-null  int64  
 21  day                                   151164 non-null  int64  
dtypes: float64(13), int64(6), object(3)
memory usage: 25.4+ MB
None
```

```
#Defining our X and y to predict country of origin
#taking in X
#total cases per million
#total deaths per million
#positive rate
#people fully vaccinated per hundred
#year
#month

X = df.iloc[:, [6, 7, 10, 13, 19, 20]]

#taking in y
#location to predict
y = df.iloc[:, [2]]
```

```
#train test split , 60:40

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
#Decision Tree Algorithm to predict which country the given data is from
#provided an unlabelled dataset.

country_cdvpym_clf = DecisionTreeClassifier()

country_cdvpym_clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
y_pred = country_cdvpym_clf.predict(X_test)

treeclf_acc = accuracy_score(y_test,y_pred)
treeclf_balacc = balanced_accuracy_score(y_test, y_pred)
treeclf_prec = precision_score(y_test, y_pred, average='macro')
treeclf_jacc = jaccard_score(y_test, y_pred, average='macro')

print("Decision Tree \naccuracy = ", treeclf_acc, "\nbalanced accuracy = ", treeclf_balac
```

```
Decision Tree
accuracy =  0.9743326828300202
balanced accuracy =  0.9744314682819577
precision =  0.9744823265342877
jaccard =  0.9508690618074491
```

```
#Defining our X and y to predict how well vaccinations will go
#taking in X
#total cases per million
#total deaths per million
#positive rate
#year
#month

X = df.iloc[:, [6, 7, 10, 19, 20]]

#taking in y
#location to predict
y = df.iloc[:, [13]]

#train test split , 60:40

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
#Decision Tree and Random Forest Regression to predict how well vaccinations will go
#provided a date and some data at that date

country_cdpym_dtreg = DecisionTreeRegressor()
country_cdpym_rfreg = RandomForestRegressor()

country_cdpym_dtreg.fit(X_train_scaled, y_train)
country_cdpym_rfreg.fit(X_train_scaled, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: DataConversionWarning: A

RandomForestRegressor()
```

```
y_pred = country_cdpym_dtreg.predict(X_test_scaled)

tree_mse = mean_squared_error(y_test, y_pred)
tree_mae = mean_absolute_error(y_test, y_pred)

y_pred = country_cdpym_rfreg.predict(X_test_scaled)

rf_mse = mean_squared_error(y_test, y_pred)
rf_mae = mean_absolute_error(y_test, y_pred)

print("Decision Tree \nmse = ",tree_mse,"\nmae = ",tree_mae,"\nrmse = ", sqrt(tree_mse))
print("\nRandom Forest \nmse = ",rf_mse," \nmae = ",rf_mae,"\nrmse = ", sqrt(rf_mse))
```

```
Decision Tree
mse =  6.104453299225523
mae =  0.32586605787972966
rmse =  2.470719186638887

Random Forest
mse =  2.9265676986387823
mae =  0.42199349009762854
rmse =  1.7107213971417972
```

```
#-------------------------------------------------------------------------------
```

```
#Code by Harish

# creating a copy to perform various opeations
df=df_income.copy()
```

```
# changing the date format to the int form string to prcocess it easily
df['date'] = pd.to_datetime(df['date'])
df['date'] = df['date'].dt.strftime('%Y.%m.%d')
```

```
df['year'] = pd.DatetimeIndex(df['date']).year
df['month'] = pd.DatetimeIndex(df['date']).month
df['day'] = pd.DatetimeIndex(df['date']).day

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3864 entries, 0 to 3863
Data columns (total 15 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Unnamed: 0                          3864 non-null   int64  
 1   iso_code                            3864 non-null   object 
 2   location                            3864 non-null   object 
 3   date                                3864 non-null   object 
 4   total_cases                         3864 non-null   int64  
 5   total_deaths                        3864 non-null   int64  
 6   total_cases_per_million             3864 non-null   float64
 7   total_deaths_per_million            3864 non-null   float64
 8   people_vaccinated                   3864 non-null   int64  
 9   people_fully_vaccinated             3864 non-null   int64  
 10  people_fully_vaccinated_per_hundred  3864 non-null   float64
 11  population                          3864 non-null   int64  
 12  year                                3864 non-null   int64  
 13  month                               3864 non-null   int64  
 14  day                                 3864 non-null   int64  
dtypes: float64(3), int64(9), object(3)
memory usage: 452.9+ KB
None
```

```
# X (labels) -->
# x1 = total cases per million
# x2 = total deaths per million
# x3 = people fully vaccinated per hundred
# x4 = year
# x5 = month

X = df.iloc[:, [6, 7, 10, 12, 13]]

# predicting y -> regions affected
y = df.iloc[:, [2]]

# splitting the train(60) test(40) data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
# Decision Tree Algorithm to predict which country the given data is from
country_cdvpym_clf = DecisionTreeClassifier()
country_cdvpym_clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
y_pred = country_cdvpym_clf.predict(X_test)
```

```
print(y_pred)

acc = accuracy_score(y_test,y_pred)
bal_acc = balanced_accuracy_score(y_test, y_pred)
pre = precision_score(y_test, y_pred, average='macro')
jacc = jaccard_score(y_test, y_pred, average='macro')

print("Decision Tree \naccuracy = ", acc, "\nbalanced accuracy = ", bal_acc, "\nprecisior
```

```
['High income' 'High income' 'High income' ... 'High income'
 'Lower middle income' 'High income']
Decision Tree
accuracy =  0.9844760672703752
balanced accuracy =  0.9847029487974239
precision =  0.9846308971009321
jaccard =  0.9698604184896511
```

```
# X (labels) -->
# x1 = total cases per million
# x2 = total deaths per million
# x4 = year
# x5 = month

X = df.iloc[:, [6, 7, 12, 13]]

# y = people fully vaccinated per hundred (predcited)
y = df.iloc[:, [10]]

# splitting the train(60) test(40) data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Desicion tree algorithm to predict the y (no. of people gets vaccinated)
reigion_dt = DecisionTreeRegressor()
reigion_dt.fit(X_train_scaled, y_train)

y_pred = reigion_dt.predict(X_test_scaled)
print(y_pred)

dt_mse = mean_squared_error(y_test, y_pred)
dt_mae = mean_absolute_error(y_test, y_pred)
```

```
[7.656e+01 7.839e+01 2.946e+01 ... 1.820e+00 2.000e-02 5.930e+00]
```

```
# Random Forest algorithm to predict the y (no. of people gets vaccinated)
reigion_rf = RandomForestRegressor()
reigion_rf.fit(X_train_scaled, y_train)
```

```
4
5 y_pred = reigion_rf.predict(X_test_scaled)
6 print(y_pred)
7
8 rf_mse = mean_squared_error(y_test, y_pred)
9 rf_mae = mean_absolute_error(y_test, y_pred)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A
  This is separate from the ipykernel package so we can avoid doing imports until
[7.65607e+01 7.84550e+01 2.98320e+01 ... 1.80680e+00 1.75000e-02
 6.14680e+00]
```

```
1 # comparing the errors in algorithm
2
3 print("Decision Tree \nmse = ",dt_mse,"\nmae = ",dt_mae,"\nrmse = ", sqrt(dt_mse))
4 print("\nRandom Forest \nmse = ",rf_mse," \nmae = ",rf_mae,"\nrmse = ", sqrt(rf_mse))
```

```
Decision Tree
mse =  0.0303601552393273
mae =  0.05817593790426913
rmse =  0.1742416575888995

Random Forest
mse =  0.06462492855109962
mae =  0.05394320827943078
rmse =  0.2542143358489045
```

```
1 #-------------------------------------------------------------------------------------
```

```
1 # Code by Siddh
2
3 # creating a copy to perform various opeations
4 df=df_region.copy()
```

```
1 # changing the date format to the int form string to prcocess it easily
2 df['date'] = pd.to_datetime(df['date'])
3 df['date'] = df['date'].dt.strftime('%Y.%m.%d')
4 df['year'] = pd.DatetimeIndex(df['date']).year
5 df['month'] = pd.DatetimeIndex(df['date']).month
6 df['day'] = pd.DatetimeIndex(df['date']).day
7
8 print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7734 entries, 0 to 7733
Data columns (total 15 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Unnamed: 0                          7734 non-null   int64
 1   iso_code                            7734 non-null   object
```

```
 2   location                             7734 non-null   object
 3   date                                 7734 non-null   object
 4   total_cases                          7734 non-null   int64
 5   total_deaths                         7734 non-null   int64
 6   total_cases_per_million              7734 non-null   float64
 7   total_deaths_per_million             7734 non-null   float64
 8   people_vaccinated                    7734 non-null   int64
 9   people_fully_vaccinated              7734 non-null   int64
 10  people_fully_vaccinated_per_hundred  7734 non-null   float64
 11  population                           7734 non-null   int64
 12  year                                 7734 non-null   int64
 13  month                                7734 non-null   int64
 14  day                                  7734 non-null   int64
dtypes: float64(3), int64(9), object(3)
memory usage: 906.5+ KB
None
```

```
# X (labels) -->
# x1 = total cases per million
# x2 = total deaths per million
# x3 = people fully vaccinated per hundred
# x4 = year
# x5 = month

X = df.iloc[:, [6, 7, 10, 12, 13]]

# predicting y -> regions affected
y = df.iloc[:, [2]]

# splitting the train(60) test(40) data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
# Decision Tree Algorithm to predict which country the given data is from
country_cdvpym_clf = DecisionTreeClassifier()
country_cdvpym_clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
y_pred = country_cdvpym_clf.predict(X_test)
print(y_pred)

acc = accuracy_score(y_test,y_pred)
bal_acc = balanced_accuracy_score(y_test, y_pred)
pre = precision_score(y_test, y_pred, average='macro')
jacc = jaccard_score(y_test, y_pred, average='macro')

print("Decision Tree \naccuracy = ", acc, "\nbalanced accuracy = ", bal_acc, "\nprecisior
```

```
['Europe' 'Asia' 'Africa' ... 'South America' 'World' 'Asia']
Decision Tree
accuracy =  0.9091790562378798
```

```
balanced accuracy =  0.9096047809120531
precision =  0.9098267920680132
jaccard =  0.8379172876900025
```

```
# X (labels) -->
# x1 = total cases per million
# x2 = total deaths per million
# x4 = year
# x5 = month

X = df.iloc[:, [6, 7, 12, 13]]

# y = people fully vaccinated per hundred (predcited)
y = df.iloc[:, [10]]

# splitting the train(60) test(40) data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Desicion tree algorithm to predict the y (no. of people gets vaccinated)
reigion_dt = DecisionTreeRegressor()
reigion_dt.fit(X_train_scaled, y_train)

y_pred = reigion_dt.predict(X_test_scaled)
print(y_pred)

dt_mse = mean_squared_error(y_test, y_pred)
dt_mae = mean_absolute_error(y_test, y_pred)
```

```
[15.73  0.06 44.22 ... 41.32 13.15 48.05]
```

```
# Random Forest algorithm to predict the y (no. of people gets vaccinated)
reigion_rf = RandomForestRegressor()
reigion_rf.fit(X_train_scaled, y_train)

y_pred = reigion_rf.predict(X_test_scaled)
print(y_pred)

rf_mse = mean_squared_error(y_test, y_pred)
rf_mae = mean_absolute_error(y_test, y_pred)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A
  This is separate from the ipykernel package so we can avoid doing imports until
[15.7029  0.0565 44.5292 ... 42.7917 13.0369 51.1516]
```

```
# comparing the errors in algorithm

print("Decision Tree \nmse = ",dt_mse,"\nmae = ",dt_mae,"\nrmse = ", sqrt(dt_mse))
print("\nRandom Forest \nmse = ",rf_mse," \nmae = ",rf_mae,"\nrmse = ", sqrt(rf_mse))
```

```
Decision Tree
mse =  1.8603172268907564
mae =  0.2360730446024564
rmse =  1.3639344657610044

Random Forest
mse =  1.1992298690723981
mae =  0.2672010019392378
rmse =  1.0950935435260305
```


✓ 0s completed at 11:46 PM