Importing Libraries

In [1]:

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

import warnings
warnings.filterwarnings('ignore')
```

Reading The Data

In [2]:

```
coviddeathbycountry = pd.read_csv('coviddeathbycountry.csv')
coviddeathbycountry.head()
```

Out[2]:

	Country	Deaths	Cases
0	Peru	213769	3729879
1	Bulgaria	37289	1183877
2	Bosnia and Herzegovina	15817	380749
3	Hungary	46696	1940824
4	Georgia	16847	1667453

Shape Of The Data

In [3]:

```
coviddeathbycountry.shape
```

Out[3]:

```
(217, 3)
```

Using Describe Function On The Continuous Variable

In [4]:

```
coviddeathbycountry.describe()
```

Out[4]:

	Deaths	Cases
count	2.170000e+02	2.170000e+02
mean	3.446119e+04	3.300453e+06
std	1.242084e+05	1.320264e+07
min	0.000000e+00	1.000000e+00
25%	2.330000e+02	2.838900e+04
50%	2.652000e+03	2.348800e+05
75%	1.614300e+04	1.287088e+06
max	1.110232e+06	1.547888e+08

Checking Missing Values In The Data

In [5]:

```
coviddeathbycountry.isnull().sum()
```

Out[5]:

```
Country    0
Deaths     0
Cases      0
dtype: int64
```

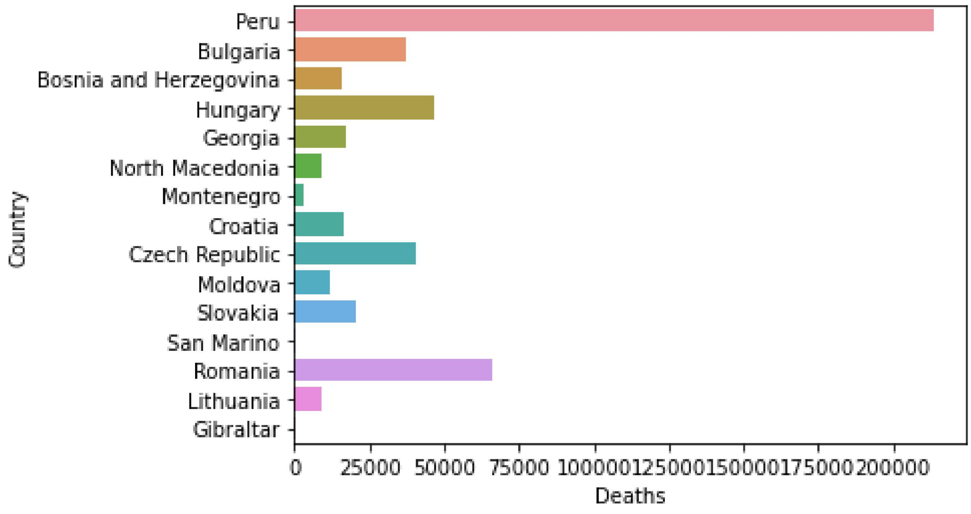
Plotting The Bar Plot

In [6]:

```
sns.barplot(x="Deaths", y="Country", data=coviddeathbycountry[:15])
```

Out[6]:

```
<AxesSubplot:xlabel='Deaths', ylabel='Country'>
```

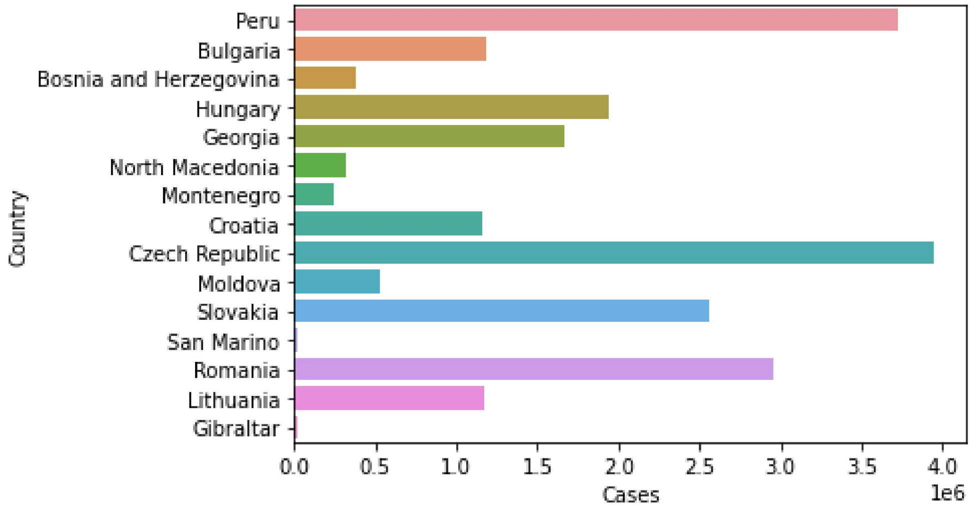


In [8]:

```
sns.barplot(x="Cases", y="Country", data=coviddeathbycountry[:15])
```

Out[8]:

```
<AxesSubplot:xlabel='Cases', ylabel='Country'>
```



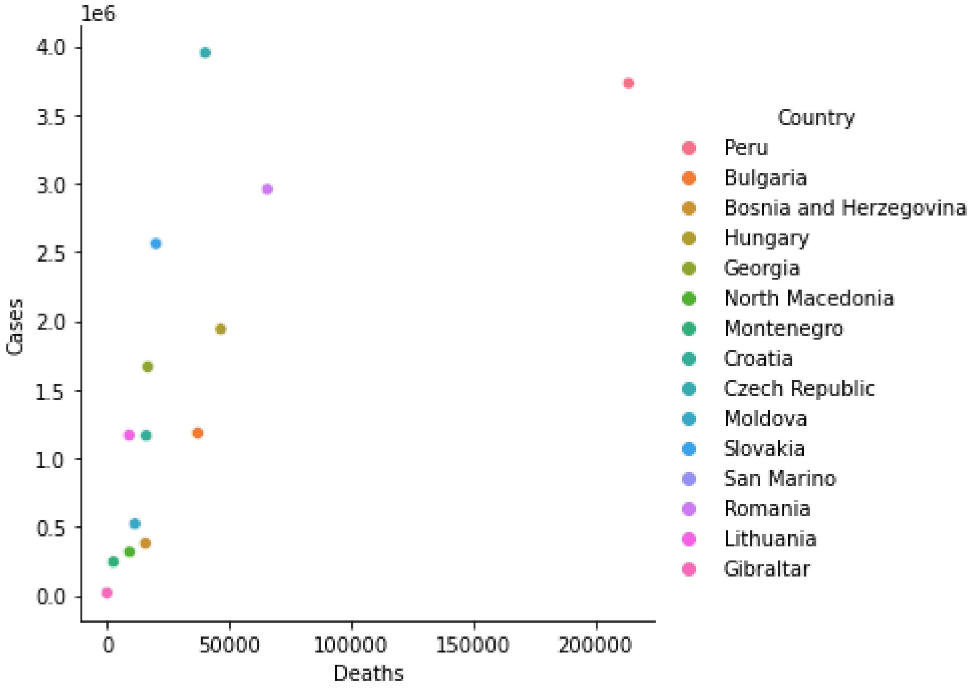
Plotting The Scatter Plot

In [9]:

```
sns.relplot(x="Deaths", y="Cases", hue="Country",data=coviddeathbycountry[:15])
```

Out[9]:

```
<seaborn.axisgrid.FacetGrid at 0x2e2010d6fd0>
```



One Hot Encoding For All The Variables

In [13]:

```
coviddeathbycountry_encoded = pd.get_dummies(coviddeathbycountry)
coviddeathbycountry_encoded.head()
```

Out[13]:

	Deaths	Cases	Country_Afghanistan	Country_Albania	Country_Algeria	Country_Andorra
0	213769	3729879	0	0	0	0
1	37289	1183877	0	0	0	0
2	15817	380749	0	0	0	0
3	46696	1940824	0	0	0	0
4	16847	1667453	0	0	0	0

5 rows × 219 columns

In [14]:

```
coviddeathbycountry.value_counts()
```

Out[14]:

```
Deaths   Cases      Country_Afghanistan  Country_Albania  Country_Algeria
Country_Andorra  Country_Angola  Country_Anguilla  Country_Antigua and Bar
buda  Country_Argentina  Country_Armenia  Country_Aruba  Country_Australia
Country_Austria  Country_Azerbaijan  Country_Bahamas  Country_Bahrain  Cou
ntry_Bangladesh  Country_Barbados  Country_Belarus  Country_Belgium  Count
ry_Belize  Country_Benin  Country_Bermuda  Country_Bhutan  Country_Bolivia
Country_Bosnia and Herzegovina  Country_Botswana  Country_Brazil  Country_
British Virgin Islands  Country_Brunei  Country_Bulgaria  Country_Burkina
Faso  Country_Burundi  Country_Cabo Verde  Country_Cambodia  Country_Camer
oon  Country_Canada  Country_Caribbean Netherlands  Country_Cayman Islands
Country_Central African Republic  Country_Chad  Country_Chile  Country_Chi
na[c]  Country_Colombia  Country_Comoros  Country_Cook Islands  Country_Co
sta Rica  Country_Croatia  Country_Cuba  Country_Curaçao  Country_Cyprus
Country_Czech Republic  Country_Democratic Republic of the Congo  Country_
Denmark  Country_Djibouti  Country_Dominica  Country_Dominican Republic  C
ountry_Ecuador  Country_Egypt  Country_El Salvador  Country_Equatorial Gui
nea  Country_Eritrea  Country_Estonia  Country_Eswatini  Country_Ethiopia
Country_European Union[b]  Country_Falkland Islands  Country_Faroe Islands
```

Segregating Variables: Seperating Independent And Dependent Variables

In [38]:

```
X =coviddeathbycountry.iloc[:,:5]
y = coviddeathbycountry.iloc[:,[10]]

X.shape, y.shape
```

Out[38]:

```
((217, 5), (217, 1))
```

Importing Train Test Split To Create Validation Set

In [39]:

```
from sklearn.model_selection import train_test_split

#creating the train and validation set
X_train, X_valid, y_train, y_valid = train_test_split(X, y, random_state = 101, stratify=No
```

Distribution In Training Set

In [40]:

```
y_train.value_counts(normalize=True)
```

Out[40]:

```
Country_Armenia
0                  0.993827
1                  0.006173
dtype: float64
```

Distribution In Validation Set

In [41]:

```
y_valid.value_counts(normalize=True)
```

Out[41]:

```
Country_Armenia
0                  1.0
dtype: float64
```

Shape Of Training Set

In [43]:

```
X_train.shape, y_train.shape
```

Out[43]:

```
((162, 5), (162, 1))
```

Shape Of Validation Set

In [45]:

```
X_valid.shape, y_valid.shape
```

Out[45]:

```
((55, 5), (55, 1))
```

Import Decision Tree Classifier & Regressor

In [46]:

```
from sklearn.tree import DecisionTreeClassifier

#import decision tree regressor
from sklearn.tree import DecisionTreeRegressor
```

Creating The Decision Tree Function

In [47]:

```
model = DecisionTreeClassifier(random_state=10)

#fitting the model
model.fit(X_train, y_train)
```

Out[47]:

```
DecisionTreeClassifier(random_state=10)
```

Checking The Training Score

In [48]:

```
model.score(X_train, y_train)
```

Out[48]:

```
1.0
```

Checking the validation score

In [49]:

```
model.score(X_valid, y_valid)
```

Out[49]:

```
1.0
```

Predictions On Validation Set

In [50]:

```
model.predict(X_valid)
```

Out[50]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=uint8)
```

In [51]:

```
model.predict_proba(X_valid)
```

Out[51]:

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

In [54]:

```
y_pred = model.predict_proba(X_valid)[:,1]
```

In [55]:

```
y_new = []
for i in range(len(y_pred)):
    if y_pred[i]<=0.7:
        y_new.append(0)
    else:
        y_new.append(1)
```

Checking The Accuracy Score

In [56]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_valid, y_new)
```

Out[56]:

```
1.0
```

Changing The Max Depth

In [57]:

```
train_accuracy = []
validation_accuracy = []
for depth in range(1,10):
    dt_model = DecisionTreeClassifier(max_depth=depth, random_state=10)
    dt_model.fit(X_train, y_train)
    train_accuracy.append(dt_model.score(X_train, y_train))
    validation_accuracy.append(dt_model.score(X_valid, y_valid))
```

In [58]:

```
frame = pd.DataFrame({'max_depth':range(1,10), 'train_acc':train_accuracy, 'valid_acc':vali
frame.head()
```

Out[58]:

	max_depth	train_acc	valid_acc
0	1	0.993827	1.0
1	2	1.000000	1.0
2	3	1.000000	1.0
3	4	1.000000	1.0
4	5	1.000000	1.0

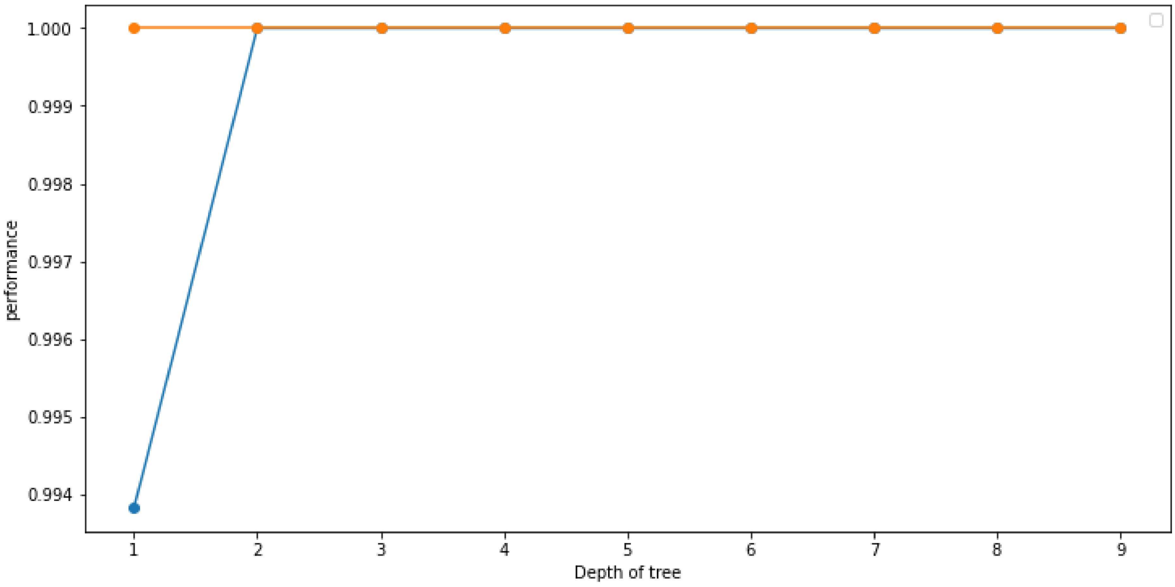
In [61]:

```
plt.figure(figsize=(12,6))
plt.plot(frame['max_depth'], frame['train_acc'], marker='o')
plt.plot(frame['max_depth'], frame['valid_acc'], marker='o')
plt.xlabel('Depth of tree')
plt.ylabel('performance')
plt.legend()
```

```
No artists with labels found to put in legend.  Note that artists whose labe
l start with an underscore are ignored when legend() is called with no argum
ent.
```

Out[61]:

```
<matplotlib.legend.Legend at 0x2e205c956a0>
```



In [63]:

```
model = DecisionTreeClassifier(max_depth=8, max_leaf_nodes=25, random_state=10)
model.fit(X_train, y_train)
```

Out[63]:

```
DecisionTreeClassifier(max_depth=8, max_leaf_nodes=25, random_state=10)
```

Training Score

In [64]:

```
model.score(X_train, y_train)
```

Out[64]:

```
1.0
```

Validation Score

In [66]:

```
model.score(X_valid, y_valid)
```

Out[66]:

```
1.0
```