In [1]:

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

In [2]:

```
vac=pd.read_csv(r"C:\Users\DELL\Downloads\country_vaccinations.csv")
```

In [3]:

```
vac.head()
```

Out[3]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated
0	Afghanistan	AFG	22-02-2021	0.0	0.0	NaN
1	Afghanistan	AFG	23-02-2021	NaN	NaN	NaN
2	Afghanistan	AFG	24-02-2021	NaN	NaN	NaN
3	Afghanistan	AFG	25-02-2021	NaN	NaN	NaN
4	Afghanistan	AFG	26-02-2021	NaN	NaN	NaN

In [4]:

```
vac.tail()
```

Out[4]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated
23463	Zimbabwe	ZWE	03-06-2021	1048504.0	684164.0	364340.0
23464	Zimbabwe	ZWE	04-06-2021	1056238.0	685564.0	370674.0
23465	Zimbabwe	ZWE	05-06-2021	1061951.0	686636.0	375315.0
23466	Zimbabwe	ZWE	06-06-2021	1068107.0	687321.0	380786.0
23467	Zimbabwe	ZWE	07-06-2021	1073971.0	688696.0	385275.0

In [5]:

```
vac
```

Out[5]:

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinate
0	Afghanistan	AFG	22-02-2021	0.0	0.0	Na
1	Afghanistan	AFG	23-02-2021	NaN	NaN	Na
2	Afghanistan	AFG	24-02-2021	NaN	NaN	Na
3	Afghanistan	AFG	25-02-2021	NaN	NaN	Na
4	Afghanistan	AFG	26-02-2021	NaN	NaN	Na
...	...	...	...	...	...	
23463	Zimbabwe	ZWE	03-06-2021	1048504.0	684164.0	364340
23464	Zimbabwe	ZWE	04-06-2021	1056238.0	685564.0	370674
23465	Zimbabwe	ZWE	05-06-2021	1061951.0	686636.0	375315
23466	Zimbabwe	ZWE	06-06-2021	1068107.0	687321.0	380786
23467	Zimbabwe	ZWE	07-06-2021	1073971.0	688696.0	385275

23468 rows × 15 columns

In [6]:

```
vac.shape
```

Out[6]:

```
(23468, 15)
```

In [7]:

```
vac.isnull().sum()
```

Out[7]:

```
country                                    0
iso_code                                   0
date                                       0
total_vaccinations                     10280
people_vaccinated                      11072
people_fully_vaccinated                13697
daily_vaccinations_raw                 12538
daily_vaccinations                       229
total_vaccinations_per_hundred         10280
people_vaccinated_per_hundred          11072
people_fully_vaccinated_per_hundred    13697
daily_vaccinations_per_million           229
vaccines                                   0
source_name                                0
source_website                             0
dtype: int64
```

In [8]:

```
vac.columns
```

Out[8]:

```
Index(['country', 'iso_code', 'date', 'total_vaccinations',
       'people_vaccinated', 'people_fully_vaccinated',
       'daily_vaccinations_raw', 'daily_vaccinations',
       'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred',
       'people_fully_vaccinated_per_hundred', 'daily_vaccinations_per_millio
n',
       'vaccines', 'source_name', 'source_website'],
      dtype='object')
```

In [9]:

```
vac.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23468 entries, 0 to 23467
Data columns (total 15 columns):
 #   Column                               Non-Null Count  Dtype  
---  ------                               --------------  -----  
 0   country                              23468 non-null  object 
 1   iso_code                             23468 non-null  object 
 2   date                                 23468 non-null  object 
 3   total_vaccinations                   13188 non-null  float64
 4   people_vaccinated                    12396 non-null  float64
 5   people_fully_vaccinated              9771 non-null   float64
 6   daily_vaccinations_raw               10930 non-null  float64
 7   daily_vaccinations                   23239 non-null  float64
 8   total_vaccinations_per_hundred       13188 non-null  float64
 9   people_vaccinated_per_hundred        12396 non-null  float64
 10  people_fully_vaccinated_per_hundred  9771 non-null   float64
 11  daily_vaccinations_per_million       23239 non-null  float64
 12  vaccines                             23468 non-null  object 
 13  source_name                          23468 non-null  object 
 14  source_website                       23468 non-null  object 
dtypes: float64(9), object(6)
memory usage: 2.7+ MB
```

In [10]:

```
vac.describe()
```

Out[10]:

	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	da
count	1.318800e+04	1.239600e+04	9.771000e+03	1.093000e+04	
mean	7.881621e+06	4.273327e+06	2.379615e+06	1.829314e+05	
std	3.736120e+07	1.578617e+07	9.902365e+06	9.953903e+05	
min	0.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00	
25%	8.706825e+04	7.094775e+04	3.377300e+04	3.670000e+03	
50%	6.232595e+05	4.740495e+05	2.531460e+05	1.916850e+04	
75%	2.869333e+06	1.937770e+06	1.043344e+06	7.532625e+04	
max	8.089620e+08	1.883639e+08	1.404420e+08	2.291800e+07	

In [11]:

```
vac.total_vaccinations.describe()
```

Out[11]:

```
count    1.318800e+04
mean     7.881621e+06
std      3.736120e+07
min      0.000000e+00
25%      8.706825e+04
50%      6.232595e+05
75%      2.869333e+06
max      8.089620e+08
Name: total_vaccinations, dtype: float64
```

In [12]:

```
vac.total_vaccinations=vac.total_vaccinations.fillna(vac.total_vaccinations.mean())
```

In [13]:

```
vac.people_vaccinated.describe()
```

Out[13]:

```
count    1.239600e+04
mean     4.273327e+06
std      1.578617e+07
min      0.000000e+00
25%      7.094775e+04
50%      4.740495e+05
75%      1.937770e+06
max      1.883639e+08
Name: people_vaccinated, dtype: float64
```

In [14]:

```
vac.people_vaccinated=vac.people_vaccinated.fillna(vac.people_vaccinated.mean())
```

In [15]:

```
vac.people_fully_vaccinated.describe()
```

Out[15]:

```
count    9.771000e+03
mean     2.379615e+06
std      9.902365e+06
min      1.000000e+00
25%      3.377300e+04
50%      2.531460e+05
75%      1.043344e+06
max      1.404420e+08
Name: people_fully_vaccinated, dtype: float64
```

In [16]:

```
vac.people_fully_vaccinated=vac.people_fully_vaccinated.fillna(vac.people_fully_vaccinated.
```

In [17]:

```
vac.daily_vaccinations_raw.describe()
```

Out[17]:

```
count    1.093000e+04
mean     1.829314e+05
std      9.953903e+05
min      0.000000e+00
25%      3.670000e+03
50%      1.916850e+04
75%      7.532625e+04
max      2.291800e+07
Name: daily_vaccinations_raw, dtype: float64
```

In [18]:

```
vac.daily_vaccinations_raw=vac.daily_vaccinations_raw.fillna(vac.daily_vaccinations_raw.mea
```

In [19]:

```
vac.daily_vaccinations.describe()
```

Out[19]:

```
count    2.323900e+04
mean     9.414671e+04
std      6.527215e+05
min      0.000000e+00
25%      8.230000e+02
50%      5.974000e+03
75%      3.207400e+04
max      2.029871e+07
Name: daily_vaccinations, dtype: float64
```

In [20]:

```
vac.daily_vaccinations=vac.daily_vaccinations.fillna(vac.daily_vaccinations.mean())
```

In [21]:

```
vac.total_vaccinations_per_hundred.describe()
```

Out[21]:

```
count    13188.000000
mean        22.030738
std         28.918313
min          0.000000
25%          2.070000
50%          9.940000
75%         30.782500
max        230.780000
Name: total_vaccinations_per_hundred, dtype: float64
```

In [22]:

```
vac.total_vaccinations_per_hundred=vac.total_vaccinations_per_hundred.fillna(vac.total_vacc
```

In [23]:

```
vac.people_vaccinated_per_hundred.describe()
```

Out[23]:

```
count    12396.000000
mean        15.135511
std         18.073016
min          0.000000
25%          1.857500
50%          7.600000
75%         22.482500
max        116.110000
Name: people_vaccinated_per_hundred, dtype: float64
```

In [24]:

```
vac.people_vaccinated_per_hundred=vac.people_vaccinated_per_hundred.fillna(vac.people_vacci
```

In [25]:

```
vac.people_fully_vaccinated_per_hundred.describe()
```

Out[25]:

```
count    9771.000000
mean        8.645363
std        12.581196
min         0.000000
25%         0.910000
50%         3.630000
75%        10.950000
max       114.670000
Name: people_fully_vaccinated_per_hundred, dtype: float64
```

In [26]:

```
vac.people_fully_vaccinated_per_hundred=vac.people_fully_vaccinated_per_hundred.fillna(vac.
```

In [27]:

```
vac.daily_vaccinations_per_million.describe()
```

Out[27]:

```
count     23239.000000
mean       3247.743061
std        4584.199398
min           0.000000
25%         359.000000
50%        1658.000000
75%        4653.000000
max      118759.000000
Name: daily_vaccinations_per_million, dtype: float64
```

In [28]:

```
vac.daily_vaccinations_per_million=vac.daily_vaccinations_per_million.fillna(vac.daily_vacc
```

In [29]:

```
from sklearn.preprocessing import LabelEncoder
```

In [30]:

```
le=LabelEncoder()
```

In [31]:

```
vac[vac.select_dtypes(include=['object']).columns]=vac[vac.select_dtypes(include=['object']
```

In [73]:

```
from sklearn.model_selection import train_test_split
```

In [74]:

```
vac_train,vac_test=train_test_split(vac,test_size=.2)
```

In [75]:

```
vac_train_x=vac_train.iloc[:,0:-1]
```

In [76]:

```
vac_train_y=vac_train.iloc[:,-1]
```

In [77]:

```
vac_test_x=vac_test.iloc[:,0:-1]
```

In [78]:

```
vac_test_y=vac_test.iloc[:,-1]
```

boxplots

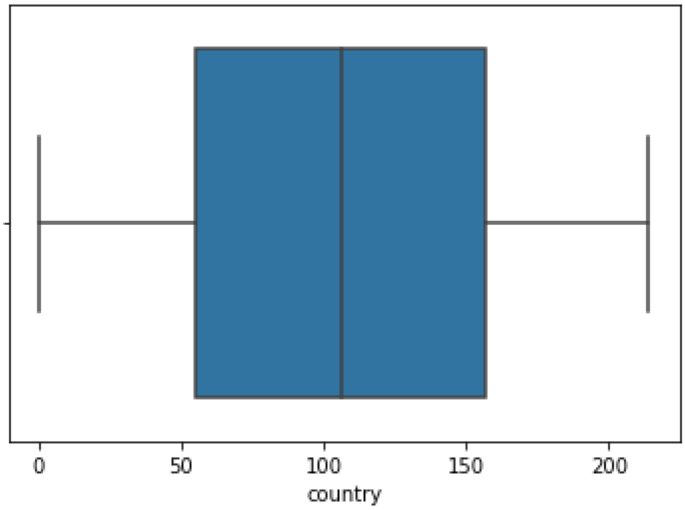
In [38]:

```
sns.boxplot(vac.country)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[38]:

```
<AxesSubplot:xlabel='country'>
```



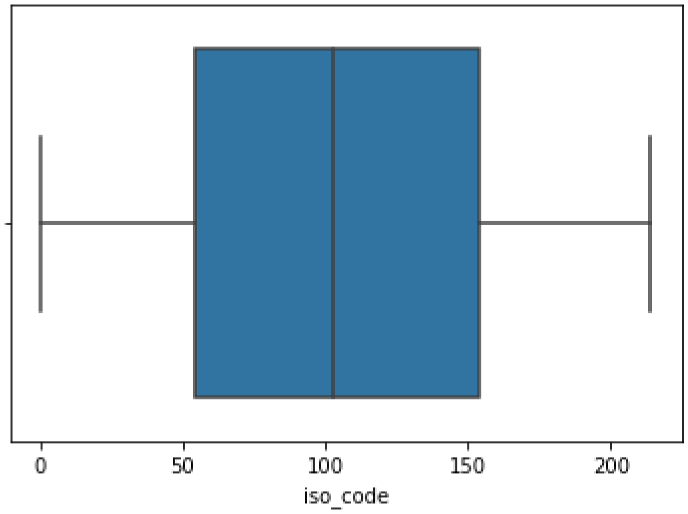
In [39]:

```
sns.boxplot(vac.iso_code)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[39]:

```
<AxesSubplot:xlabel='iso_code'>
```



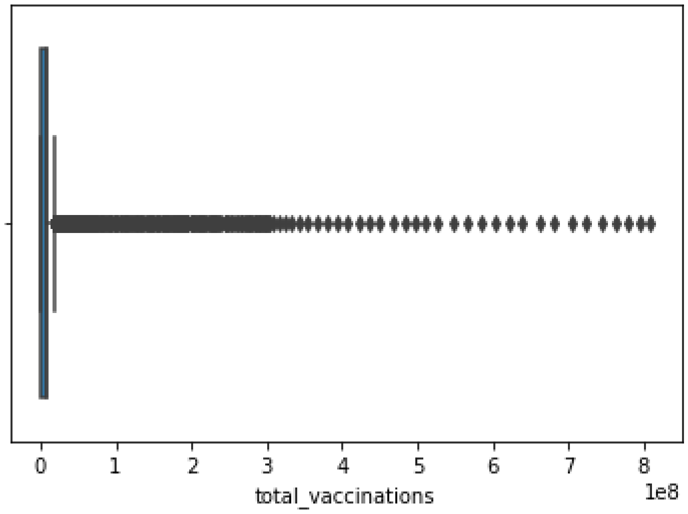
In [40]:

```
sns.boxplot(vac.total_vaccinations)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[40]:

```
<AxesSubplot:xlabel='total_vaccinations'>
```



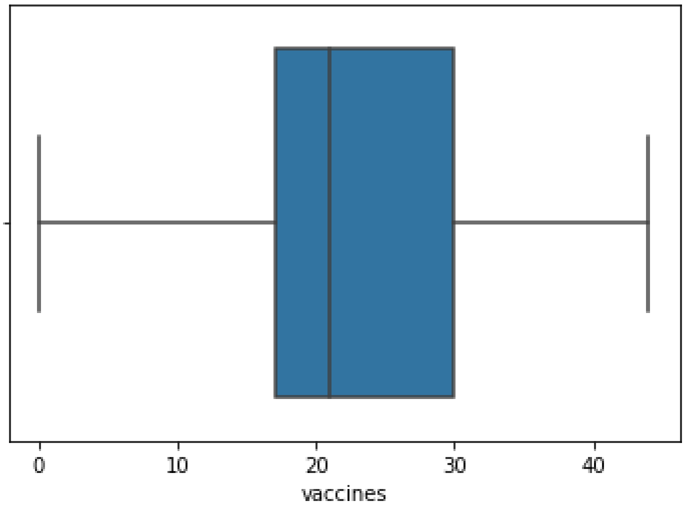
In [41]:

```
sns.boxplot(vac.vaccines)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[41]:

```
<AxesSubplot:xlabel='vaccines'>
```



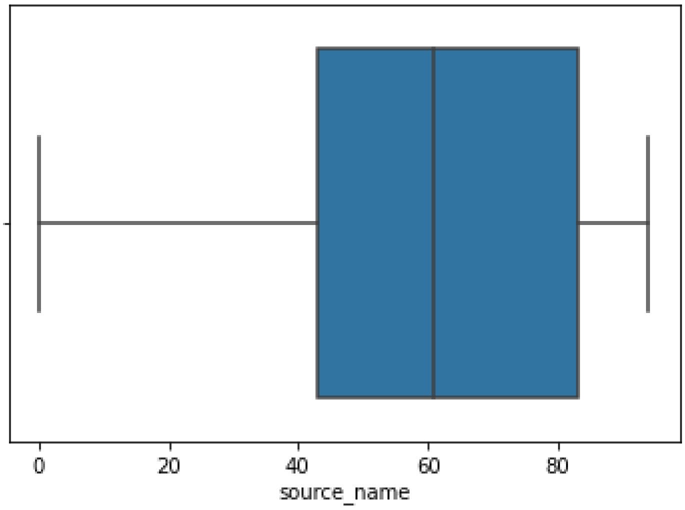
In [42]:

```
sns.boxplot(vac.source_name)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[42]:

```
<AxesSubplot:xlabel='source_name'>
```



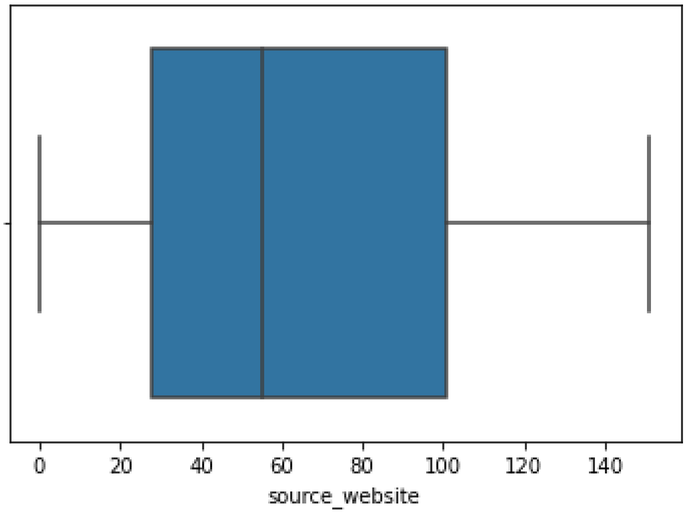
In [43]:

```
sns.boxplot(vac.source_website)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[43]:

```
<AxesSubplot:xlabel='source_website'>
```

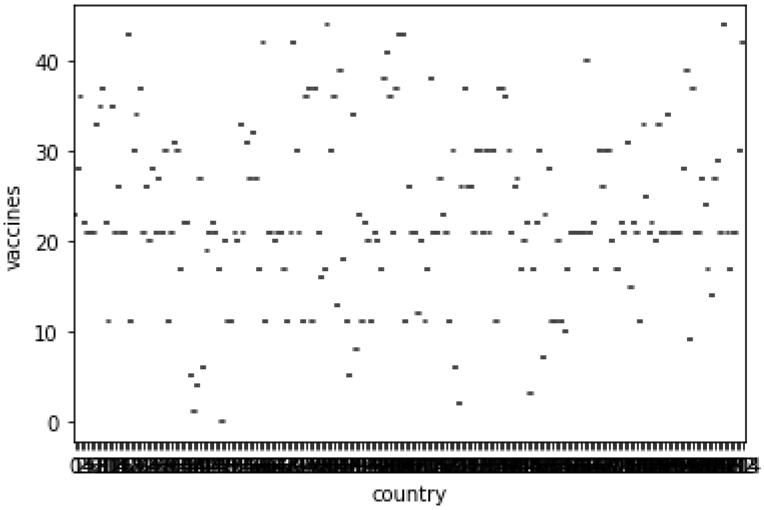


In [44]:

```
sns.boxplot(x='country',y='vaccines',data=vac)
```

Out[44]:

```
<AxesSubplot:xlabel='country', ylabel='vaccines'>
```

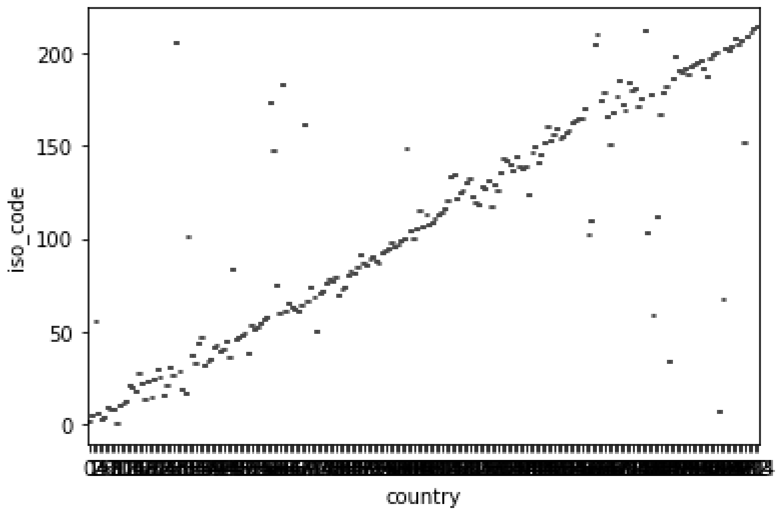


In [45]:

```
sns.boxplot(x='country',y='iso_code',data=vac)
```

Out[45]:

```
<AxesSubplot:xlabel='country', ylabel='iso_code'>
```

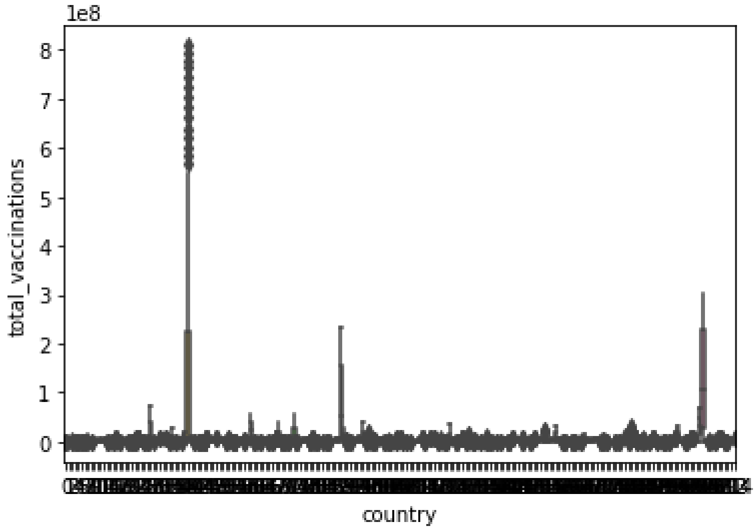


In [46]:

```
sns.boxplot(x='country',y='total_vaccinations',data=vac)
```

Out[46]:

```
<AxesSubplot:xlabel='country', ylabel='total_vaccinations'>
```



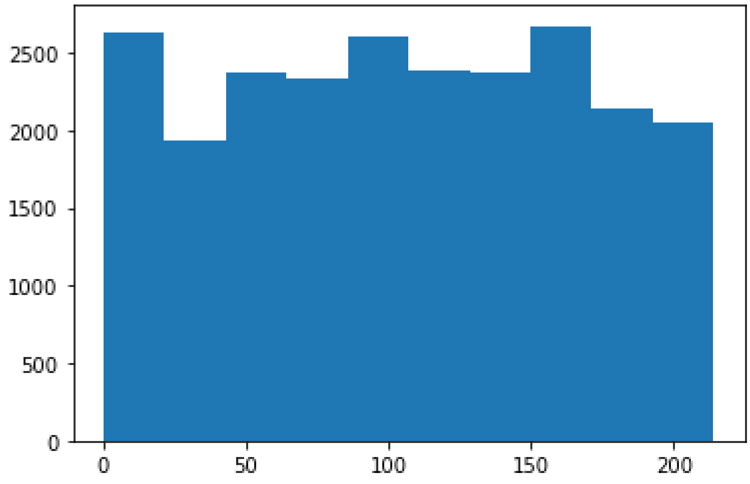
histograms

In [47]:

```
plt.hist(vac.country)
```

Out[47]:

```
(array([2634., 1933., 2366., 2330., 2603., 2386., 2367., 2671., 2135.,
        2043.]),
 array([  0. ,  21.4,  42.8,  64.2,  85.6, 107. , 128.4, 149.8, 171.2,
        192.6, 214. ]),
 <BarContainer object of 10 artists>)
```

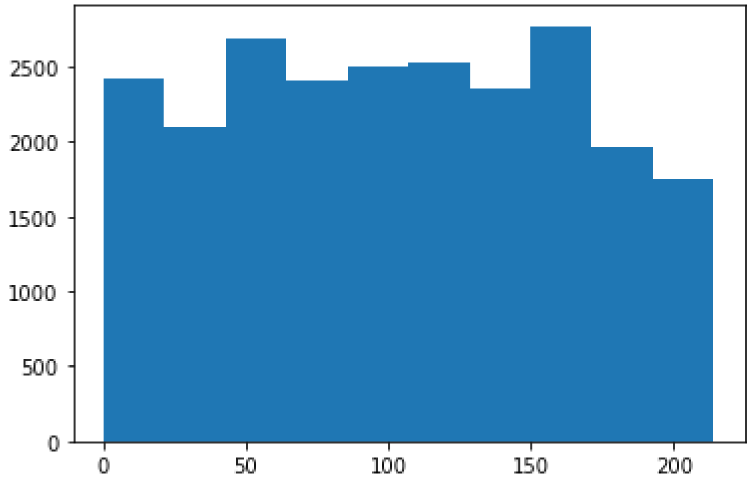


In [48]:

```
plt.hist(vac.iso_code)
```

Out[48]:

```
(array([2422., 2095., 2682., 2400., 2504., 2520., 2356., 2771., 1965.,
        1753.]),
 array([  0. ,  21.4,  42.8,  64.2,  85.6, 107. , 128.4, 149.8, 171.2,
        192.6, 214. ]),
 <BarContainer object of 10 artists>)
```

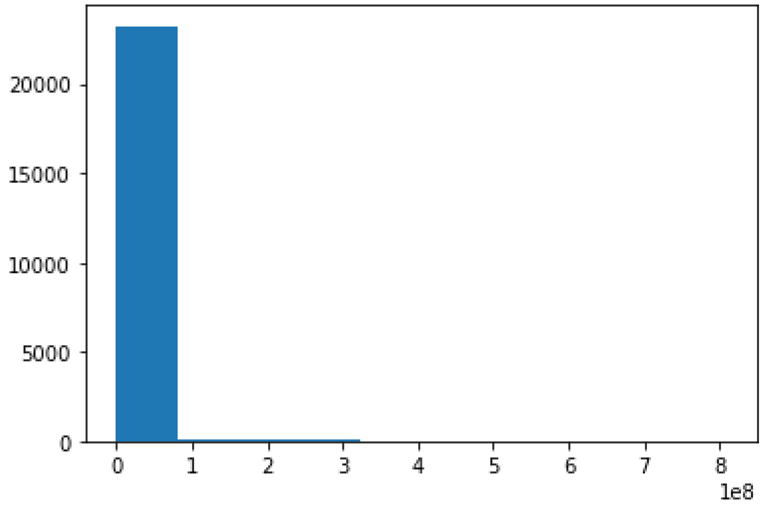


In [49]:

```
plt.hist(vac.total_vaccinations)
```

Out[49]:

```
(array([2.3233e+04, 7.7000e+01, 7.8000e+01, 4.9000e+01, 7.0000e+00,
        6.0000e+00, 4.0000e+00, 5.0000e+00, 4.0000e+00, 5.0000e+00]),
 array([0.000000e+00, 8.089620e+07, 1.617924e+08, 2.426886e+08,
        3.235848e+08, 4.044810e+08, 4.853772e+08, 5.662734e+08,
        6.471696e+08, 7.280658e+08, 8.089620e+08]),
 <BarContainer object of 10 artists>)
```

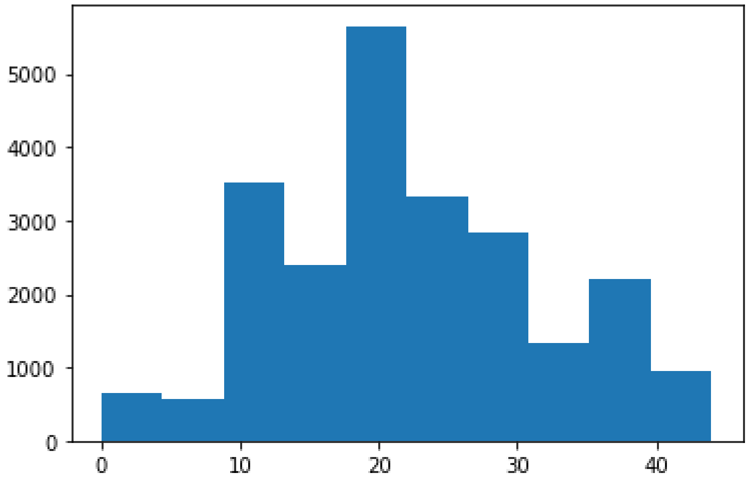


In [50]:

```
plt.hist(vac.vaccines)
```

Out[50]:

```
(array([ 651.,  579., 3518., 2407., 5651., 3326., 2833., 1335., 2205.,
         963.]),
 array([ 0. ,  4.4,  8.8, 13.2, 17.6, 22. , 26.4, 30.8, 35.2, 39.6, 44. ]),
 <BarContainer object of 10 artists>)
```

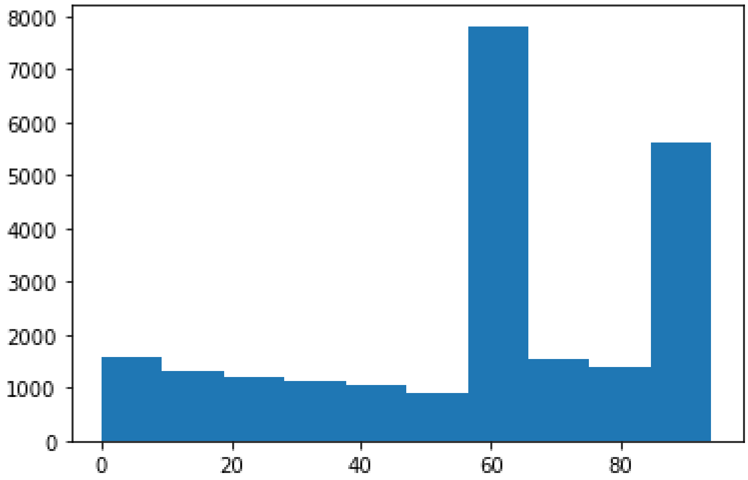


In [51]:

```
plt.hist(vac.source_name)
```

Out[51]:

```
(array([1559., 1295., 1213., 1112., 1054.,  900., 7810., 1525., 1374.,
        5626.]),
 array([ 0. ,  9.4, 18.8, 28.2, 37.6, 47. , 56.4, 65.8, 75.2, 84.6, 94. ]),
 <BarContainer object of 10 artists>)
```

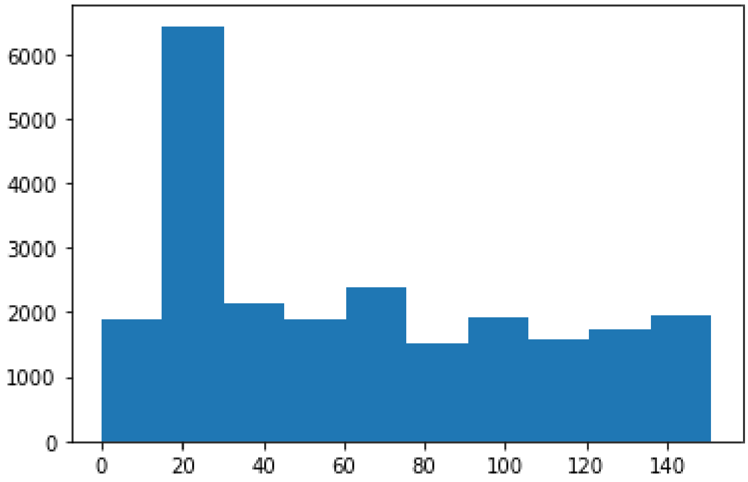


In [52]:

```
plt.hist(vac.source_website)
```

Out[52]:

```
(array([1895., 6437., 2152., 1892., 2372., 1530., 1908., 1582., 1741.,
        1959.]),
 array([  0. ,  15.1,  30.2,  45.3,  60.4,  75.5,  90.6, 105.7, 120.8,
        135.9, 151. ]),
 <BarContainer object of 10 artists>)
```



In [53]:

```
vac=vac.drop(['iso_code','source_website'],axis=1)
```

In [54]:

```
vac.head()
```

Out[54]:

	country	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinatic
0	0	134	0.000000e+00	0.000000e+00	2.379615e+06	182931
1	0	140	7.881621e+06	4.273327e+06	2.379615e+06	182931
2	0	146	7.881621e+06	4.273327e+06	2.379615e+06	182931
3	0	152	7.881621e+06	4.273327e+06	2.379615e+06	182931
4	0	158	7.881621e+06	4.273327e+06	2.379615e+06	182931

Linear Regression

In [79]:

```
from sklearn.linear_model import LinearRegression
```

In [80]:

```
linreg=LinearRegression()
```

In [81]:

```
linreg.fit(vac_train_x,vac_train_y)
```

Out[81]:

```
LinearRegression()
```

In [82]:

```
linreg.score(vac_train_x,vac_train_y)
```

Out[82]:

```
0.08648069828282634
```

In [83]:

```
pred_reg=linreg.predict(vac_train_x)
pred_reg
```

Out[83]:

```
array([57.21543729, 51.65104312, 54.26420346, ..., 61.10345202,
       66.4481081 , 60.5550023 ])
```

In [84]:

```
linreg.coef_
```

Out[84]:

```
array([ 4.64854327e-02,  4.62337434e-03,  5.37131254e-07, -3.61256950e-07,
       -7.71434316e-07,  1.16449874e-05, -3.00490963e-05,  2.84206732e-02,
        1.26867497e-01, -4.72635404e-01, -9.57630636e-04, -1.47241708e-01])
```

In [85]:

```
linreg.intercept_
```

Out[85]:

```
61.62195222548637
```

In [86]:

```
from sklearn.metrics import confusion_matrix
```

In [90]:

```
tab1=confusion_matrix(pred_reg,vac_test_y)
tab1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-90-210a1fced809> in <module>
----> 1 tab1=confusion_matrix(pred_reg,vac_test_y)
      2 tab1

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args,
 **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args
)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in confusio
n_matrix(y_true, y_pred, labels, sample_weight, normalize)
    274 
    275     """
--> 276     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    277     if y_type not in ("binary", "multiclass"):
    278         raise ValueError("%s is not supported" % y_type)

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in _check_t
argets(y_true, y_pred)
     79     y_pred : array or indicator matrix
     80     """
---> 81     check_consistent_length(y_true, y_pred)
     82     type_true = type_of_target(y_true)
     83     type_pred = type_of_target(y_pred)

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_consisten
t_length(*arrays)
    253     uniques = np.unique(lengths)
    254     if len(uniques) > 1:
--> 255         raise ValueError("Found input variables with inconsistent nu
mbers of"
    256                          " samples: %r" % [int(l) for l in lengths])
    257 

ValueError: Found input variables with inconsistent numbers of samples: [187
74, 4694]
```

In [88]:

```
from sklearn.metrics import accuracy_score
```

In [89]:

```
accuracy_score(pred_reg,vac_train_y)*100
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-89-df8e584ebe5e> in <module>
----> 1 accuracy_score(pred_reg,vac_train_y)*100

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args,
 **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args
)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in accuracy
_score(y_true, y_pred, normalize, sample_weight)
    185 
    186     # Compute accuracy for each possible representation
--> 187     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    188     check_consistent_length(y_true, y_pred, sample_weight)
    189     if y_type.startswith('multilabel'):

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in _check_t
argets(y_true, y_pred)
     88 
     89     if len(y_type) > 1:
---> 90         raise ValueError("Classification metrics can't handle a mix
 of {0} "
     91                          "and {1} targets".format(type_true, type_pr
ed))
     92 

ValueError: Classification metrics can't handle a mix of continuous and mult
iclass targets
```

Logistic Regression

In [91]:

```
from sklearn.linear_model import LogisticRegression
```

In [92]:

```
logreg=LogisticRegression()
```

In [93]:

```
logreg.fit(vac_train_x,vac_train_y)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[93]:

```
LogisticRegression()
```

In [94]:

```
logreg.score(vac_train_x,vac_train_y)
```

Out[94]:

```
0.2524768296580377
```

In [95]:

```
logreg.coef_
```

Out[95]:

```
array([[ 4.06205360e-11,  2.67418730e-11, -4.26910748e-07, ...,
         8.86229168e-12,  4.70564427e-09, -1.77243492e-12],
       [-7.66062140e-11, -1.40991083e-10, -2.43783295e-07, ...,
        -1.22354511e-11, -4.83170633e-09, -2.00966182e-11],
       [-8.64320702e-11, -2.66013768e-11,  2.18699838e-07, ...,
         3.83887698e-12,  3.23481425e-09, -6.23476781e-12],
       ...,
       [-1.34919397e-11,  3.49650509e-11, -1.43128530e-08, ...,
         2.51601010e-12,  1.34142194e-09,  5.97445206e-12],
       [ 2.40191378e-11, -7.79212505e-12, -1.36147607e-07, ...,
        -5.77619716e-13, -1.69667412e-09, -2.99000923e-12],
       [ 8.14128604e-10,  7.35646534e-10,  9.25318933e-07, ...,
         3.99221204e-11, -1.48968490e-08,  1.83297097e-10]])
```

In [96]:

```
logreg.intercept_
```

Out[96]:

```
array([ 2.85237505e-13, -1.43514612e-12, -2.50344640e-13, -8.87399148e-12,
       -3.20756517e-13, -6.34746003e-13,  1.82861039e-13, -4.41051162e-13,
       -1.83162729e-13, -1.59434372e-13, -1.13123284e-12,  9.36213027e-13,
        9.09143178e-14, -3.89030706e-13, -1.15600347e-13, -1.18569614e-13,
       -1.26751016e-13, -3.52890826e-13, -7.61771778e-14, -3.32948764e-13,
       -1.22515205e-13, -3.85091623e-13, -1.47185058e-13,  9.86163834e-14,
       -2.84662731e-13, -3.89944787e-13,  8.20462978e-14, -1.75553683e-14,
       -2.62063011e-12, -5.34771333e-13, -3.91112854e-13, -3.78204725e-13,
       -3.40589897e-13, -3.37107627e-13,  5.76075814e-14,  4.86249980e-14,
        5.58700171e-14,  1.56329767e-14,  6.91621925e-14, -3.55108311e-13,
       -2.41290518e-13, -2.16128985e-13,  1.02027765e-13,  4.93405940e-14,
       -4.32634351e-13, -4.00663563e-13, -1.31032282e-13, -2.72659196e-13,
       -1.75957423e-13, -3.91325633e-13, -4.34392568e-13, -9.03595109e-14,
       -2.41661256e-13, -4.24655560e-13, -1.94243534e-13, -4.23084804e-13,
       -2.56973165e-13,  6.69103906e-13, -1.72179395e-14,  1.11240193e-13,
       -1.37730213e-13,  2.34818969e-11, -5.50876114e-14, -7.61975442e-14,
       -3.83020477e-13, -1.66080811e-13,  1.02067402e-14,  1.31743377e-14,
       -3.63792182e-14, -4.26271738e-13, -4.37544261e-13, -4.67642430e-13,
        4.08206885e-13, -7.29798928e-13, -2.37883286e-13,  4.36484431e-13,
        3.93375752e-13,  4.29468470e-13, -1.20159870e-12, -4.80726759e-13,
        1.95621983e-13, -3.17935157e-13, -3.96632212e-13, -1.13258409e-12,
       -4.45431751e-13, -4.36969597e-13, -1.40518025e-12,  1.32996659e-13,
       -4.47650447e-13,  1.63722443e-13, -8.20057122e-13, -1.81419808e-13,
        3.56415578e-13, -1.17397688e-13,  7.25174537e-12])
```

In [97]:

```
pred_reg=logreg.predict(vac_train_x)
pred_reg
```

Out[97]:

```
array([61, 94, 94, ..., 61, 94, 57])
```

In [98]:

```
from sklearn.metrics import confusion_matrix
```

In [99]:

```
tab1=confusion_matrix(pred_reg,vac_train_y)
tab1
```

Out[99]:

```
array([[   0,    0,    0, ...,    0,    0,    0],
       [   0,    1,    0, ...,    0,    0,    0],
       [   0,    0,    0, ...,    0,    0,    0],
       ...,
       [   0,    0,    0, ...,    0,    0,    0],
       [   0,    0,    0, ...,    0,    0,    0],
       [   1,   28,   96, ...,    3,   14, 2593]], dtype=int64)
```

In [100]:

```
from sklearn.metrics import accuracy_score
```

In [101]:

```
accuracy_score(pred_reg,vac_train_y)*100
```

Out[101]:

```
25.24768296580377
```

Decision trees

In [103]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [104]:

```
dt=DecisionTreeClassifier()
```

In [105]:

```
dt.fit(vac_train_x,vac_train_y)
```

Out[105]:

```
DecisionTreeClassifier()
```

In [106]:

```
dt.score(vac_train_x,vac_train_y)
```

Out[106]:

```
1.0
```

In [107]:

```
pred_dt=dt.predict(vac_train_x)
pred_dt
```

Out[107]:

```
array([61, 61, 46, ..., 61, 41, 57])
```

In [108]:

```
from sklearn.metrics import confusion_matrix
```

In [109]:

```
tab1=confusion_matrix(pred_dt,vac_train_y)
tab1
```

Out[109]:

```
array([[ 109,    0,    0, ...,    0,    0,    0],
       [   0,  119,    0, ...,    0,    0,    0],
       [   0,    0,  128, ...,    0,    0,    0],
       ...,
       [   0,    0,    0, ...,  140,    0,    0],
       [   0,    0,    0, ...,    0,   66,    0],
       [   0,    0,    0, ...,    0,    0, 3027]], dtype=int64)
```

In [110]:

```
from sklearn.metrics import accuracy_score
```

In [111]:

```
accuracy_score(pred_dt,vac_train_y)
```

Out[111]:

```
1.0
```

Random forest

In [112]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [113]:

```
rf=RandomForestClassifier()
```

In [114]:

```
rf.fit(vac_train_x,vac_train_y)
```

Out[114]:

```
RandomForestClassifier()
```

In [115]:

```
rf.score(vac_train_x,vac_train_y)
```

Out[115]:

```
1.0
```

In [116]:

```
pred_rf=rf.predict(vac_train_x)
pred_rf
```

Out[116]:

```
array([61, 61, 46, ..., 61, 41, 57])
```

In [117]:

```
from sklearn.metrics import confusion_matrix
```

In [118]:

```
tab1=confusion_matrix(pred_rf,vac_train_y)
tab1
```

Out[118]:

```
array([[ 109,    0,    0, ...,    0,    0,    0],
       [   0,  119,    0, ...,    0,    0,    0],
       [   0,    0,  128, ...,    0,    0,    0],
       ...,
       [   0,    0,    0, ...,  140,    0,    0],
       [   0,    0,    0, ...,    0,   66,    0],
       [   0,    0,    0, ...,    0,    0, 3027]], dtype=int64)
```

In [119]:

```
from sklearn.metrics import accuracy_score
```

In [120]:

```
accuracy_score(pred_rf,vac_train_y)
```

Out[120]:

```
1.0
```

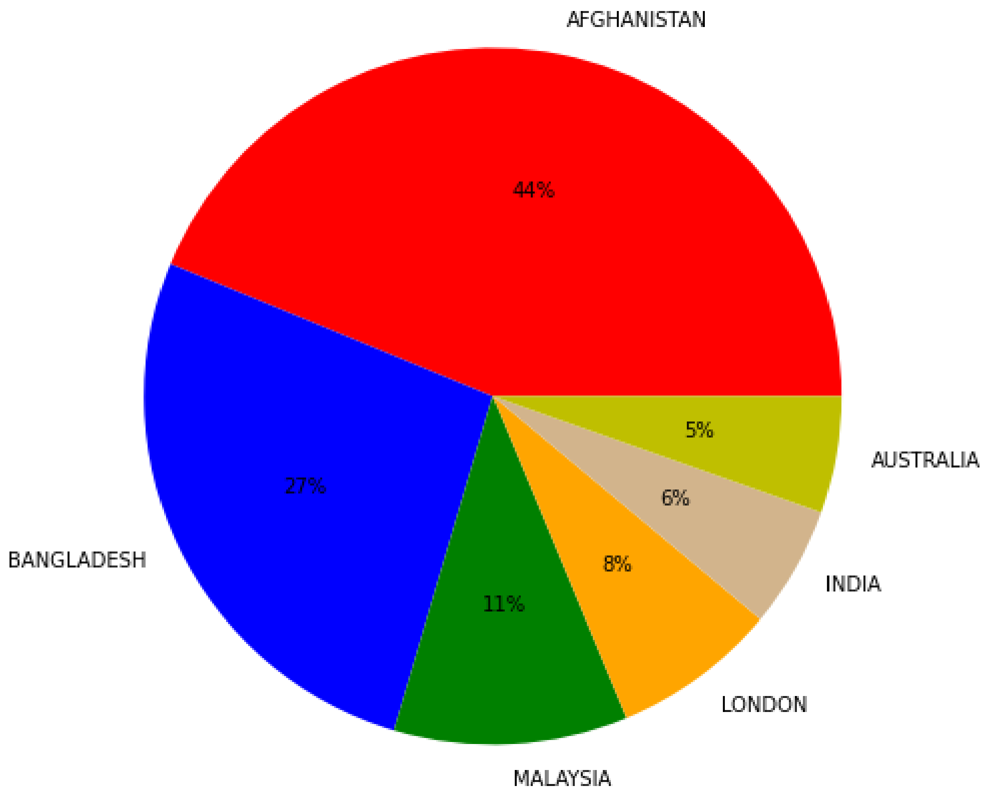
Pie chart

In [122]:

```
fig=plt.figure(figsize=(10,8))
GDP=(19.4,11.8,4.8,3.4,2.5,2.4)
countries=('AFGHANISTAN','BANGLADESH','MALAYSIA','LONDON','INDIA','AUSTRALIA')
colors_list=['r','b','g','orange','tan','y']
explode_list=[0,.2,0,0,0,.2]
plt.pie(GDP,labels=countries,colors=colors_list,autopct='%1.0f%%')
```

Out[122]:

```
([<matplotlib.patches.Wedge at 0x1ffd09c5fa0>,
  <matplotlib.patches.Wedge at 0x1ffd09d16d0>,
  <matplotlib.patches.Wedge at 0x1ffd09d1d60>,
  <matplotlib.patches.Wedge at 0x1ffd09dd430>,
  <matplotlib.patches.Wedge at 0x1ffd09ddac0>,
  <matplotlib.patches.Wedge at 0x1ffd09e9190>],
 [Text(0.21316471456361924, 1.0791481846646507, 'AFGHANISTAN'),
  Text(-0.9920308589942705, -0.47526284811995345, 'BANGLADESH'),
  Text(0.05847809043212525, -1.098444496977163, 'MALAYSIA'),
  Text(0.6522301842354419, -0.8857741172399437, 'LONDON'),
  Text(0.9558614433705955, -0.544361002531851, 'INDIA'),
  Text(1.084106096082776, -0.1863168603110388, 'AUSTRALIA')],
 [Text(0.11627166248924685, 0.5886262825443549, '44%'),
  Text(-0.541107741269602, -0.2592342807927019, '27%'),
  Text(0.031897140235704675, -0.5991515438057251, '11%'),
  Text(0.35576191867387735, -0.4831495184945147, '8%'),
  Text(0.5213789691112339, -0.29692418319919145, '6%'),
  Text(0.5913305978633322, -0.10162737835147571, '5%')])
```



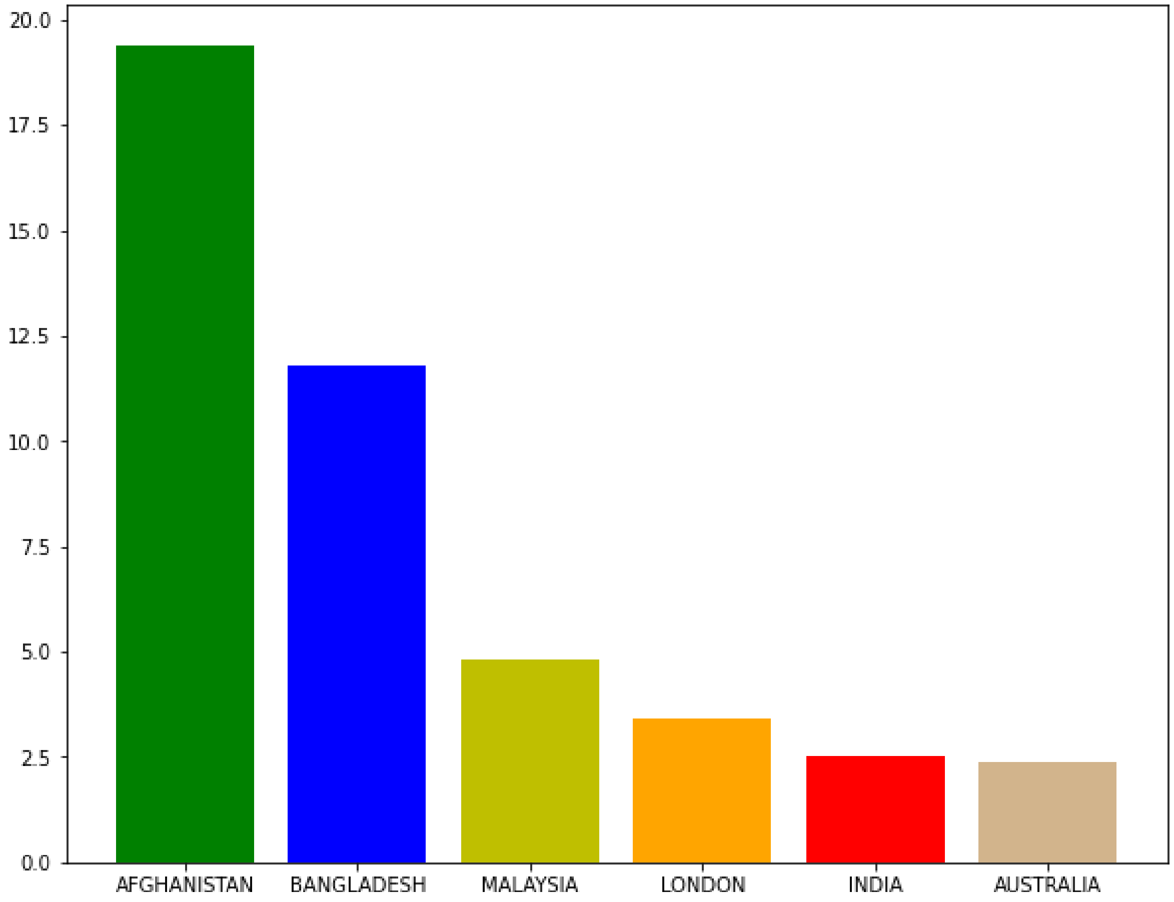
Bar plot

In [124]:

```
fig=plt.figure(figsize=(10,8))
plt.bar(countries,GDP)
color_list=['g','b','y','orange','r','tan']
plt.bar(countries,GDP,color=color_list)
```

Out[124]:

```
<BarContainer object of 6 artists>
```



In [ ]: