Problem statement

predicting the house price in USA.To create a model to help him estimate of what the house would sell for.

In [1]:

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

In [2]:

```
df=pd.read_csv("states")
```

To display top 10 rows

In [3]:

```
df.head(10)
```

Out[3]:

	id	name	country_id	country_code	country_name	state_code	type	latitude	longitude
0	3901	Badakhshan	1	AF	Afghanistan	BDS	NaN	36.734772	70.811995
1	3871	Badghis	1	AF	Afghanistan	BDG	NaN	35.167134	63.769538
2	3875	Baghlan	1	AF	Afghanistan	BGL	NaN	36.178903	68.745306
3	3884	Balkh	1	AF	Afghanistan	BAL	NaN	36.755060	66.897537
4	3872	Bamyan	1	AF	Afghanistan	BAM	NaN	34.810007	67.821210
5	3892	Daykundi	1	AF	Afghanistan	DAY	NaN	33.669495	66.046353
6	3899	Farah	1	AF	Afghanistan	FRA	NaN	32.495328	62.262663
7	3889	Faryab	1	AF	Afghanistan	FYB	NaN	36.079561	64.905955
8	3870	Ghazni	1	AF	Afghanistan	GHA	NaN	33.545059	68.417397
9	3888	Ghōr	1	AF	Afghanistan	GHO	NaN	34.099578	64.905955

Data Cleaning And Pre-Processing

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5077 entries, 0 to 5076
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   id            5077 non-null   int64  
 1   name          5077 non-null   object 
 2   country_id    5077 non-null   int64  
 3   country_code  5063 non-null   object 
 4   country_name  5077 non-null   object 
 5   state_code    5072 non-null   object 
 6   type          1597 non-null   object 
 7   latitude      5008 non-null   float64
 8   longitude     5008 non-null   float64
dtypes: float64(2), int64(2), object(5)
memory usage: 357.1+ KB
```

In [5]:

```
# Display the statistical summary
df.describe()
```

Out[5]:

	id	country_id	latitude	longitude
count	5077.000000	5077.000000	5008.000000	5008.000000
mean	2609.765413	133.467599	27.576415	17.178713
std	1503.376799	72.341160	22.208161	61.269334
min	1.000000	1.000000	-54.805400	-178.116500
25%	1324.000000	74.000000	11.399747	-3.943859
50%	2617.000000	132.000000	34.226432	17.501792
75%	3905.000000	201.000000	45.802822	41.919647
max	5220.000000	248.000000	77.874972	179.852222

In [6]:

```
# To display the col headings
df.columns
```

Out[6]:

```
Index(['id', 'name', 'country_id', 'country_code', 'country_name',
       'state_code', 'type', 'latitude', 'longitude'],
      dtype='object')
```

In [7]:

```
cols=df.dropna(axis=1)
```

In [8]:

```
cols.columns
```

Out[8]:

```
Index(['id', 'name', 'country_id', 'country_name'], dtype='object')
```

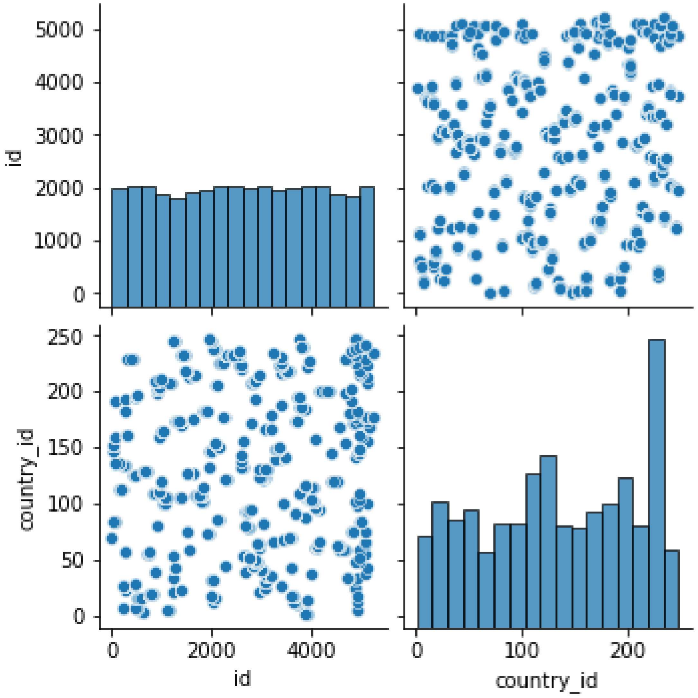
EDA and Visualization

In [9]:

```
sns.pairplot(cols)
```

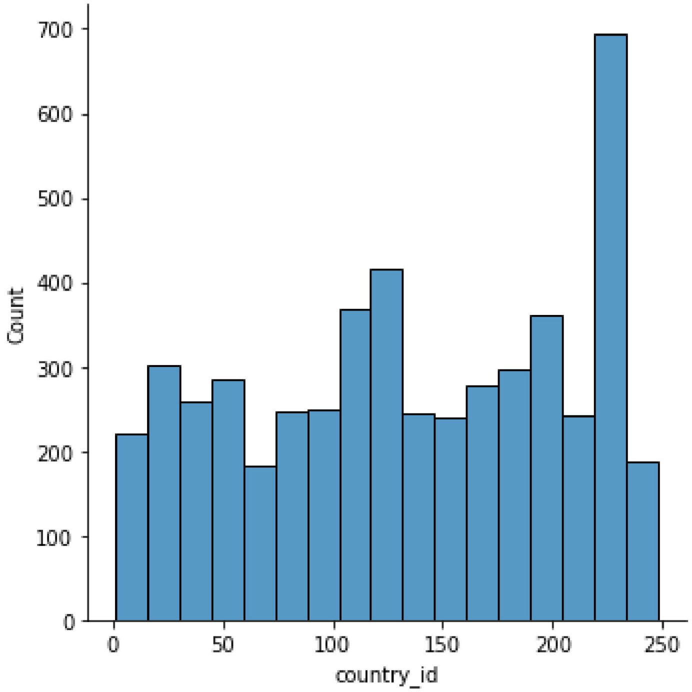
Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x2053e1c3730>
```



In [11]:
```
sns.displot(df['country_id'])
```

Out[11]:
```
<seaborn.axisgrid.FacetGrid at 0x2053942e760>
```



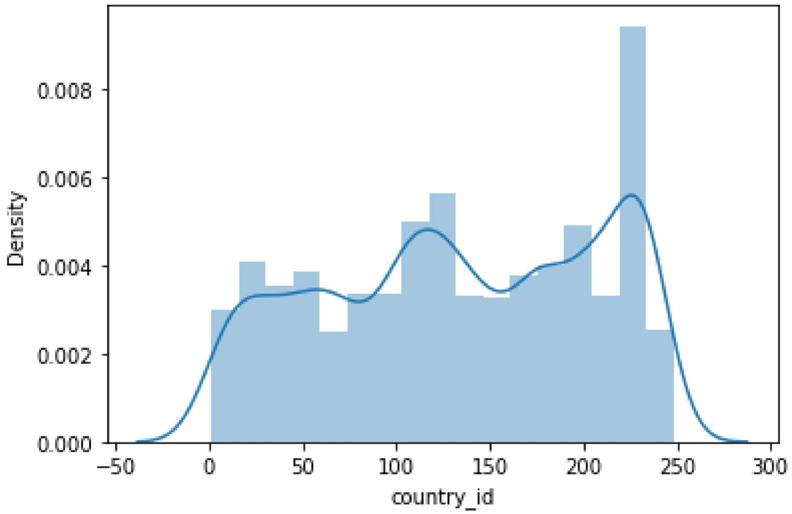
In [12]:

```
# We use displot in older version we get distplot use displot
sns.distplot(df['country_id'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a dep
recated function and will be removed in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[12]:

```
<AxesSubplot:xlabel='country_id', ylabel='Density'>
```



In [15]:

```
df1=cols[['id', 'country_id']]
df1
```

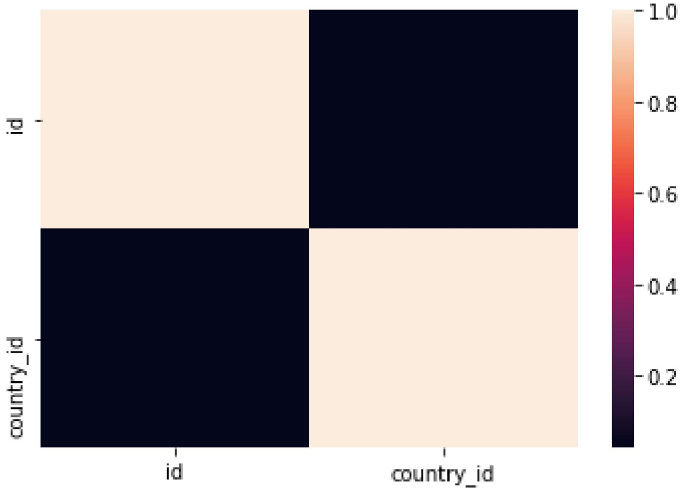
Out[15]:

```
        id    country_id
   0  3901             1
   1  3871             1
   2  3875             1
   3  3884             1
   4  3872             1
 ...   ...           ...
5072  1953           247
5073  1960           247
5074  1954           247
5075  1952           247
5076  1957           247

5077 rows × 2 columns
```

In [16]:
```
sns.heatmap(df1.corr())
```

Out[16]:
```
<AxesSubplot:>
```



To train the model - MODEL BUILD

Going to train linear regression model;We split our data into 2 variables x and y where x is independent var(input) and y is dependent on x(output), we could ignore address col as it is not required for our model

In [18]:
```
x=df1[['id', 'country_id']]
y=df1[['id']]
```

To split the dataset into test data

In [19]:
```
# importing lib for splitting test data
from sklearn.model_selection import train_test_split
```

In [20]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [21]:

```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[21]:

```
LinearRegression()
```

In [22]:

```
print(lr.intercept_)
```

```
[0.]
```

In [23]:

```
print(lr.score(x_test,y_test))
```

```
1.0
```

In [24]:

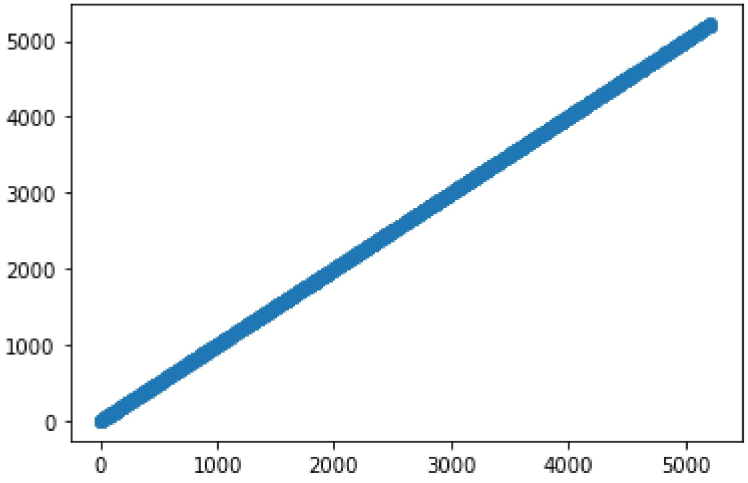
```
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[24]:

	0	1
0	1.0	1.327549e-17

```
In [25]: pred = lr.predict(x_test)
         plt.scatter(y_test,pred)

Out[25]: <matplotlib.collections.PathCollection at 0x2053f6c78b0>
```



```
In [26]: from sklearn.linear_model import Ridge,Lasso

In [27]: rr=Ridge(alpha=10)
         rr.fit(x_train,y_train)

Out[27]: Ridge(alpha=10)

In [28]: rr.score(x_test,y_test)

Out[28]: 1.0

In [29]: la=Lasso(alpha=10)
         la.fit(x_train,y_train)

Out[29]: Lasso(alpha=10)

In [30]: la.score(x_test,y_test)

Out[30]: 0.9999999999806893
```

ELASTIC NET

In [31]:

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[31]:

```
ElasticNet()
```

In [32]:

```
print(en.coef_)
```

```
[0.99999956 0.        ]
```

In [33]:

```
print(en.intercept_)
```

```
[0.00114742]
```

In [34]:

```
prediction=en.predict(x_test)
prediction
```

Out[34]:

```
array([3316.99968981, 3096.99978649, 2767.99993106, ..., 4380.99922225,
         66.00111842, 2944.99985328])
```

In [35]:

```
print(en.score(x_test,y_test))
```

```
0.9999999999998069
```

EVALUATION METRICS

In [36]:

```
from sklearn import metrics
```

In [37]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.000568406353368604
```

In [38]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 4.291842482179795e-07
```

In [39]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 0.0006551215522465885
```

MODEL SAVING

In [40]:

```
import pickle
```

In [43]:

```
filename='prediction2'
pickle.dump(lr,open(filename,'wb'))
```

In [ ]:

```

```