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]:

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

In [2]:

```python
df=pd.read_csv("cities")
```

To display top 10 rows

In [3]:

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

Out[3]:

	id	name	state_id	state_code	state_name	country_id	country_code	country_name	latitude	longitude	wikiDataId
0	52	Ashkāsham	3901	BDS	Badakhshan	1	AF	Afghanistan	36.68333	71.53333	Q4805192
1	68	Fayzabad	3901	BDS	Badakhshan	1	AF	Afghanistan	37.11664	70.58002	Q156558
2	78	Jurm	3901	BDS	Badakhshan	1	AF	Afghanistan	36.86477	70.83421	Q10308323
3	84	Khandūd	3901	BDS	Badakhshan	1	AF	Afghanistan	36.95127	72.31800	Q3290334
4	115	Rāghistān	3901	BDS	Badakhshan	1	AF	Afghanistan	37.66079	70.67346	Q2670909
5	131	Wākhān	3901	BDS	Badakhshan	1	AF	Afghanistan	37.05710	73.34928	Q2509959
6	72	Ghormach	3871	BDG	Badghis	1	AF	Afghanistan	35.73062	63.78264	Q5556982
7	108	Qala i Naw	3871	BDG	Badghis	1	AF	Afghanistan	34.98735	63.12891	Q26396
8	54	Baghlān	3875	BGL	Baghlan	1	AF	Afghanistan	36.13068	68.70829	Q732879
9	140	Ḩukūmatī Dahanah-ye Ghōrī	3875	BGL	Baghlan	1	AF	Afghanistan	35.90617	68.48869	Q5194960

Data Cleaning And Pre-Processing

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150454 entries, 0 to 150453
Data columns (total 11 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            150454 non-null  int64  
 1   name          150454 non-null  object 
 2   state_id      150454 non-null  int64  
 3   state_code    150129 non-null  object 
 4   state_name    150454 non-null  object 
 5   country_id    150454 non-null  int64  
 6   country_code  150406 non-null  object 
 7   country_name  150454 non-null  object 
 8   latitude      150454 non-null  float64
 9   longitude     150454 non-null  float64
 10  wikiDataId    147198 non-null  object 
dtypes: float64(2), int64(3), object(6)
memory usage: 12.6+ MB
```

In [5]:
```
# Display the statistical summary
df.describe()
```

Out[5]:

	id	state_id	country_id	latitude	longitude
count	150454.000000	150454.000000	150454.000000	150454.000000	150454.000000
mean	76407.091689	2678.377677	140.658460	31.556175	2.369557
std	44357.755335	1363.513591	70.666123	22.813220	68.012770
min	1.000000	1.000000	1.000000	-75.000000	-179.121980
25%	38160.250000	1451.000000	82.000000	19.000000	-58.468150
50%	75975.500000	2174.000000	142.000000	40.684720	8.669980
75%	115204.750000	3905.000000	207.000000	47.239220	27.750000
max	153528.000000	5116.000000	247.000000	73.508190	179.466000

In [6]:
```
# To display the col headings
df.columns
```

Out[6]:
```
Index(['id', 'name', 'state_id', 'state_code', 'state_name', 'country_id',
       'country_code', 'country_name', 'latitude', 'longitude', 'wikiDataId'],
      dtype='object')
```

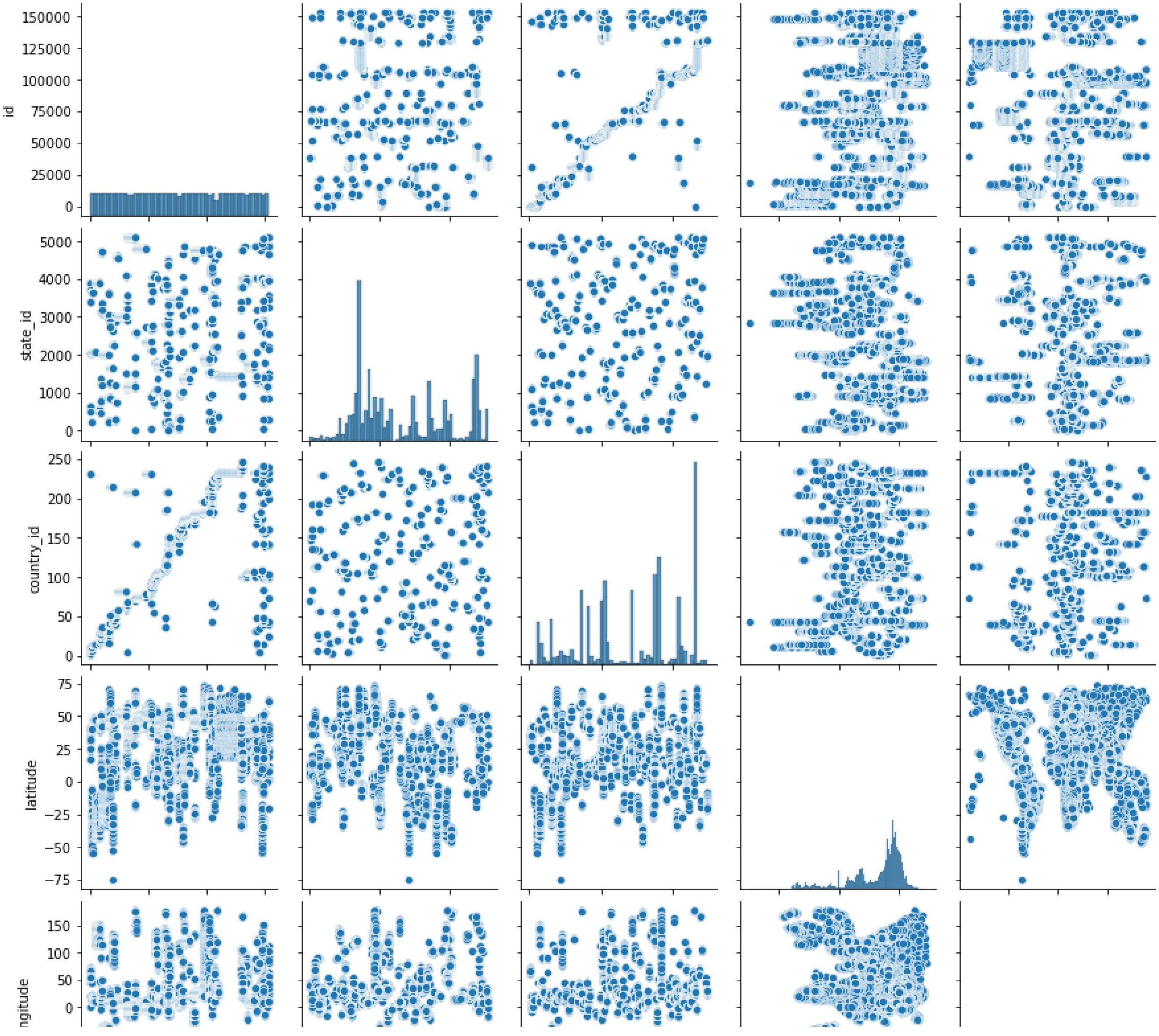
In [7]:
```
cols=df.dropna(axis=1)
```

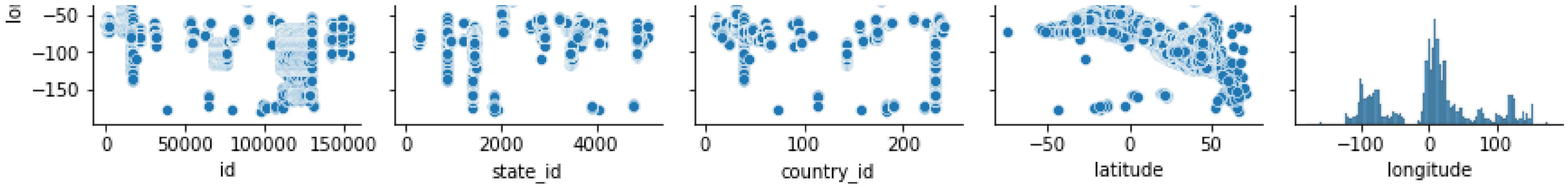
In [8]:
```
cols.columns
```

Out[8]:
```
Index(['id', 'name', 'state_id', 'state_name', 'country_id', 'country_name',
       'latitude', 'longitude'],
      dtype='object')
```

EDA and Visualization

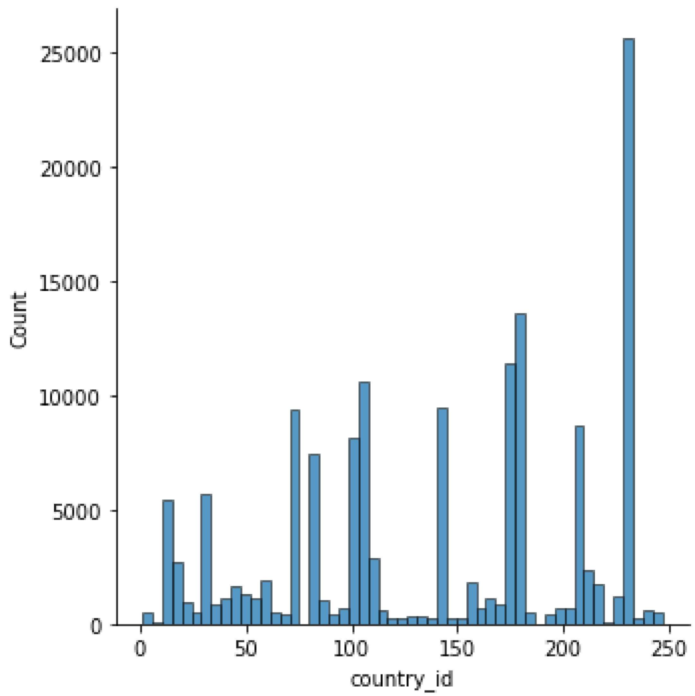
In [9]:
```
sns.pairplot(cols)
```
Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1f79079bfa0>
```


id
state_id
country_id
latitude
ngitude
150000
125000
100000
75000
50000
25000
0
5000
4000
3000
2000
1000
0
250
200
150
100
50
0
75
50
25
0
−25
−50
−75
150
100
50
0



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

Out[11]: <seaborn.axisgrid.FacetGrid at 0x1f790793bb0>



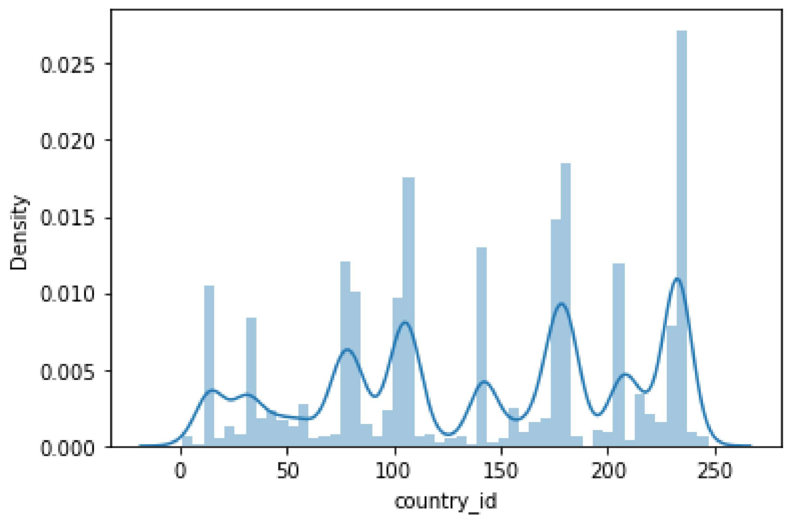
In [12]:

```python
# 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 [13]:

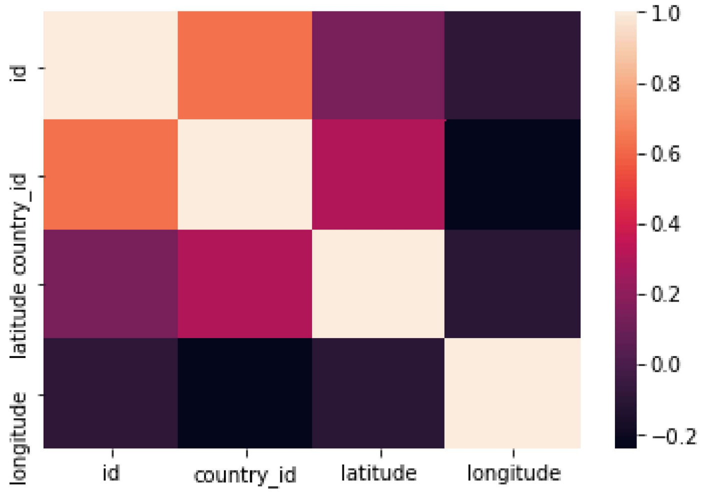
```
df1=cols[['id', 'country_id',
        'latitude', 'longitude']]
df1
```

Out[13]:

	id	country_id	latitude	longitude
0	52	1	36.68333	71.53333
1	68	1	37.11664	70.58002
2	78	1	36.86477	70.83421
3	84	1	36.95127	72.31800
4	115	1	37.66079	70.67346
...	...	...	...	...
150449	131496	247	-19.03333	29.78333
150450	131502	247	-19.78333	29.36667
150451	131503	247	-19.67016	30.00589
150452	131504	247	-19.75000	30.16667
150453	131508	247	-20.30345	30.07514

150454 rows × 4 columns

In [14]:
```
sns.heatmap(df1.corr())
```
Out[14]: <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 [15]:
```
x=df1[['id', 'country_id',
        'latitude', 'longitude']]
y=df1[['id']]
```

To split the dataset into test data

In [16]:
```
# importing lib for splitting test data
from sklearn.model_selection import train_test_split
```

In [17]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [18]:
```
from sklearn.linear_model import LinearRegression

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

Out[18]:
```
LinearRegression()
```

In [19]:
```
print(lr.intercept_)
```

```
[-1.45519152e-10]
```

In [20]:
```
print(lr.score(x_test,y_test))
```

```
1.0
```

In [21]:
```
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[21]:

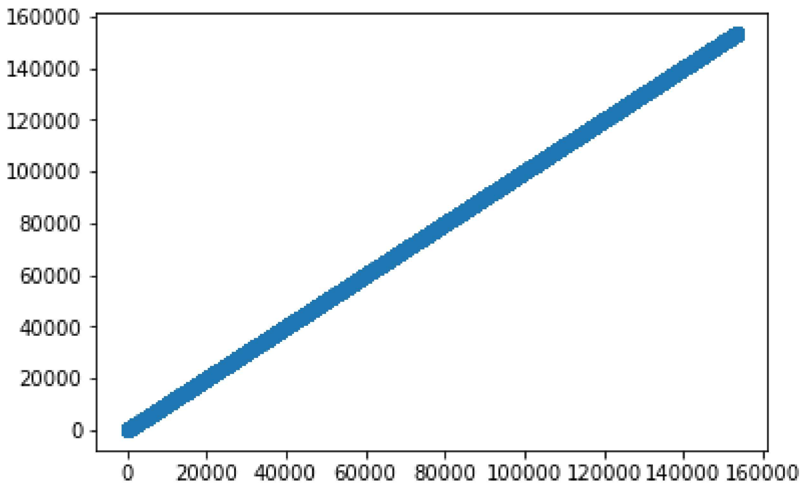
	0	1	2	3
0	1.0	4.991850e-16	1.945751e-14	1.134941e-15

In [22]:

```
pred = lr.predict(x_test)
plt.scatter(y_test,pred)
```

Out[22]:

```
<matplotlib.collections.PathCollection at 0x1f79b7c9f10>
```



In [23]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [24]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[24]:

```
Ridge(alpha=10)
```

In [25]:

```
rr.score(x_test,y_test)
```

Out[25]:

```
1.0
```

In [26]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[26]:

```
Lasso(alpha=10)
```

In [27]:

```
la.score(x_test,y_test)
```

Out[27]:

```
1.0
```

ELASTIC NET

In [28]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[28]:
```
ElasticNet()
```

In [29]:
```
print(en.coef_)
```

```
[ 1.  0.  0. -0.]
```

In [30]:
```
print(en.intercept_)
```

```
[3.87143809e-05]
```

In [31]:
```
prediction=en.predict(x_test)
prediction
```

Out[31]:
```
array([103822.99998611, 138295.99996865,  77237.99999958, ...,
       146839.99996432, 150597.99996241,  67971.00000428])
```

In [32]:
```
print(en.score(x_test,y_test))
```

```
1.0
```

EVALUATION METRICS

In [33]:
```
from sklearn import metrics
```

In [34]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.938745408377021e-05
```

In [35]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 5.014330437380037e-10
```

In [36]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 2.2392700679864493e-05
```

MODEL SAVING

In [37]:
```
import pickle
```

In [38]:
```
filename='prediction3'
pickle.dump(lr,open(filename,'wb'))
```

In [ ]:
```

```