Problem Statement

A real estate agent want help to predict the house price for regions in USA.He gave us the dataset to work on to use linear regression model.Create a model that helps him to estimate of what the house would sell for

Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# To import dataset
df=pd.read_csv('22_countries.csv')
df
```

Out[2]:

	id	name	iso3	iso2	numeric_code	phone_code	capital	currency	currency_na
0	1	Afghanistan	AFG	AF	4	93	Kabul	AFN	Afghan afgh
1	2	Aland Islands	ALA	AX	248	+358-18	Mariehamn	EUR	E
2	3	Albania	ALB	AL	8	355	Tirana	ALL	Albanian
3	4	Algeria	DZA	DZ	12	213	Algiers	DZD	Algerian di
4	5	American Samoa	ASM	AS	16	+1-684	Pago Pago	USD	US Do
...	...	...	...	...	...	...	...	...	
245	243	Wallis And Futuna Islands	WLF	WF	876	681	Mata Utu	XPF	CFP fr
246	244	Western Sahara	ESH	EH	732	212	El-Aaiun	MAD	Moroc Dirh
247	245	Yemen	YEM	YE	887	967	Sanaa	YER	Yemeni
248	246	Zambia	ZMB	ZM	894	260	Lusaka	ZMW	Zamb kwa
249	247	Zimbabwe	ZWE	ZW	716	263	Harare	ZWL	Zimbab Do

250 rows × 19 columns

Loading [MathJax]/extensions/Safe.js

In [3]:
```
# To display top 10 rows
df.head(10)
```

Out[3]:

	id	name	iso3	iso2	numeric_code	phone_code	capital	currency	currency_name
0	1	Afghanistan	AFG	AF	4	93	Kabul	AFN	Afghan afghani
1	2	Aland Islands	ALA	AX	248	+358-18	Mariehamn	EUR	Euro
2	3	Albania	ALB	AL	8	355	Tirana	ALL	Albanian lek
3	4	Algeria	DZA	DZ	12	213	Algiers	DZD	Algerian dinar
4	5	American Samoa	ASM	AS	16	+1-684	Pago Pago	USD	US Dollar
5	6	Andorra	AND	AD	20	376	Andorra la Vella	EUR	Euro
6	7	Angola	AGO	AO	24	244	Luanda	AOA	Angolan kwanza
7	8	Anguilla	AIA	AI	660	+1-264	The Valley	XCD	East Caribbean dollar
8	9	Antarctica	ATA	AQ	10	672	NaN	AAD	Antarctican dollar
9	10	Antigua And Barbuda	ATG	AG	28	+1-268	St. John's	XCD	Eastern Caribbean dollar

Data Cleaning and Pre-Processing

Loading [MathJax]/extensions/Safe.js

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               250 non-null    int64  
 1   name             250 non-null    object 
 2   iso3             250 non-null    object 
 3   iso2             249 non-null    object 
 4   numeric_code     250 non-null    int64  
 5   phone_code       250 non-null    object 
 6   capital          245 non-null    object 
 7   currency         250 non-null    object 
 8   currency_name    250 non-null    object 
 9   currency_symbol  250 non-null    object 
 10  tld              250 non-null    object 
 11  native           249 non-null    object 
 12  region           248 non-null    object 
 13  subregion        247 non-null    object 
 14  timezones        250 non-null    object 
 15  latitude         250 non-null    float64
 16  longitude        250 non-null    float64
 17  emoji            250 non-null    object 
 18  emojiU           250 non-null    object 
dtypes: float64(2), int64(2), object(15)
memory usage: 37.2+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	id	numeric_code	latitude	longitude
count	250.000000	250.00000	250.000000	250.00000
mean	125.500000	435.80400	16.402597	13.52387
std	72.312977	254.38354	26.757204	73.45152
min	1.000000	4.00000	-74.650000	-176.20000
25%	63.250000	219.00000	1.000000	-49.75000
50%	125.500000	436.00000	16.083333	17.00000
75%	187.750000	653.50000	39.000000	48.75000
max	250.000000	926.00000	78.000000	178.00000

In [6]:
```
df.columns
```

Out[6]:
```
Index(['id', 'name', 'iso3', 'iso2', 'numeric_code', 'phone_code', 'capital',
       'currency', 'currency_name', 'currency_symbol', 'tld', 'native',
       'region', 'subregion', 'timezones', 'latitude', 'longitude', 'emoji',
       'emojiU'],
      dtype='object')
```

Loading [MathJax]/extensions/Safe.js

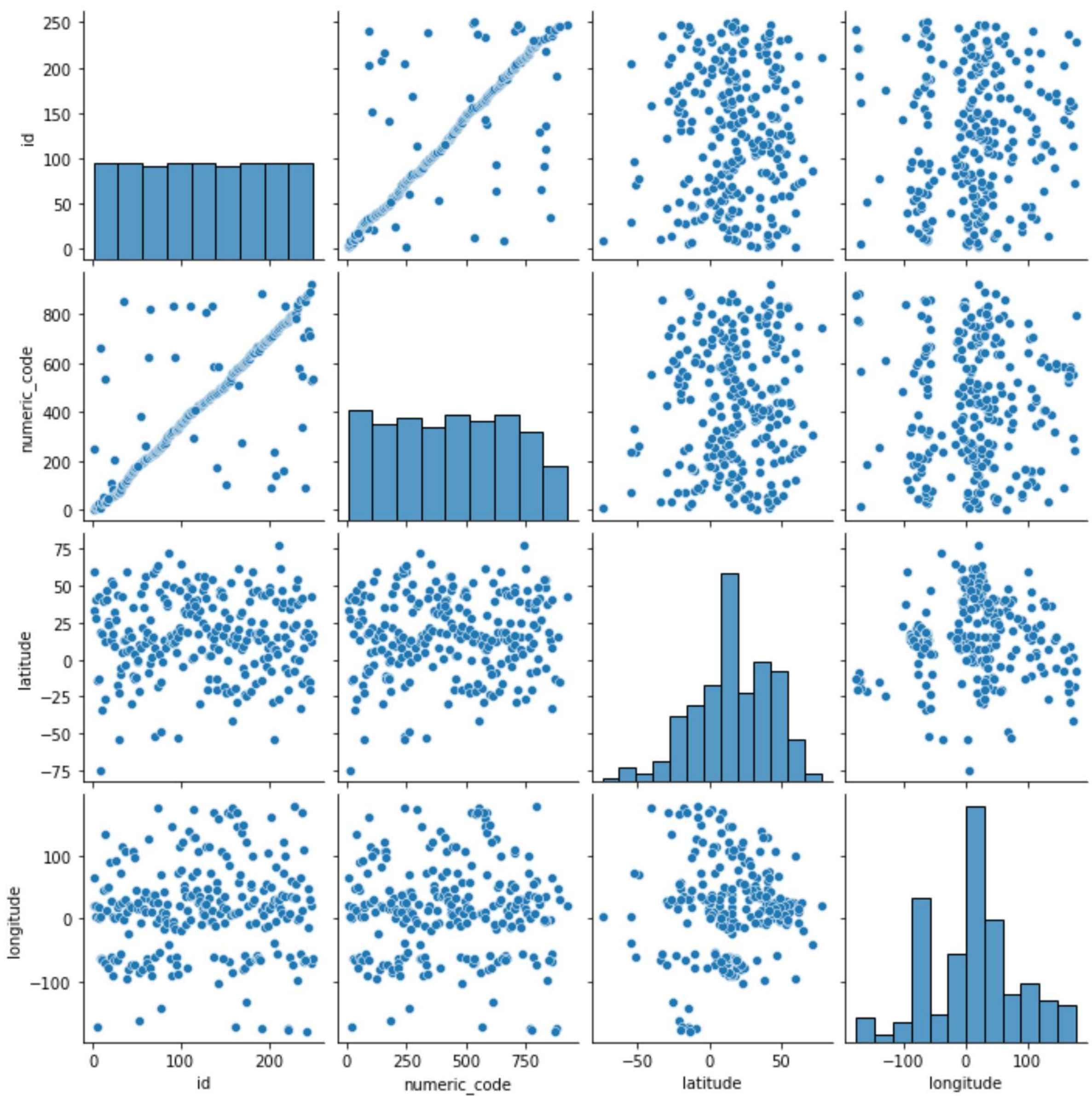
In [7]:
```
a = df.dropna(axis='columns')
a.columns
```

Out[7]:
```
Index(['id', 'name', 'iso3', 'numeric_code', 'phone_code', 'currency',
       'currency_name', 'currency_symbol', 'tld', 'timezones', 'latitude',
       'longitude', 'emoji', 'emojiU'],
      dtype='object')
```

EDA and Visualization

In [8]:
```
sns.pairplot(a)
```

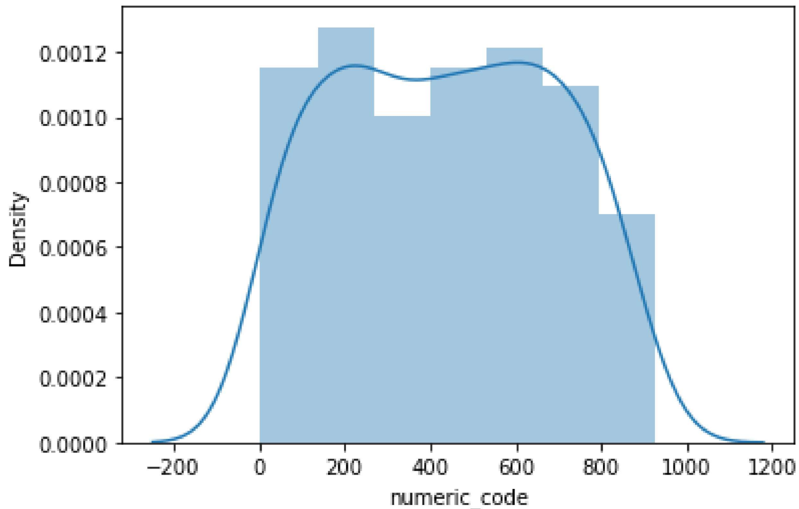
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x12edf5a3ca0>
```



In [9]:
```
sns.distplot(a['numeric_code'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated 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)
```

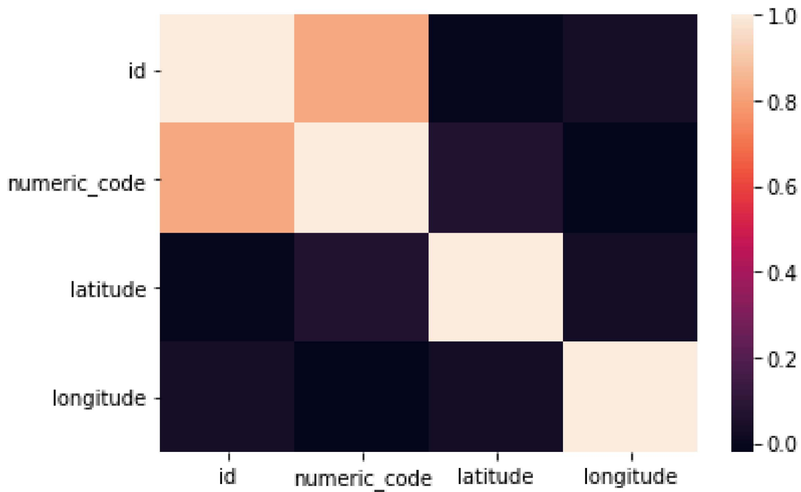
Out[9]:
```
<AxesSubplot:xlabel='numeric_code', ylabel='Density'>
```



In [10]:
```
a1=a[['id', 'numeric_code', 'latitude', 'longitude']]
```

In [11]:
```
sns.heatmap(a1.corr())
```

Out[11]:
```
<AxesSubplot:>
```



Loading [MathJax]/extensions/Safe.js

To Train the Model - Model Building

We are going to train Linear Regression model;We need to split out data into two variables x and y where x is independent variable (input) and y is dependent on x(output). We could ignore address column as it is not required for our model.

In [12]:
```
x=a1[['id', 'latitude', 'longitude']]
y=a1['numeric_code']
```

To split my dataset into training and test data

In [13]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [14]:
```
from sklearn.linear_model import LinearRegression

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

Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
74.39869766275831
```

In [16]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

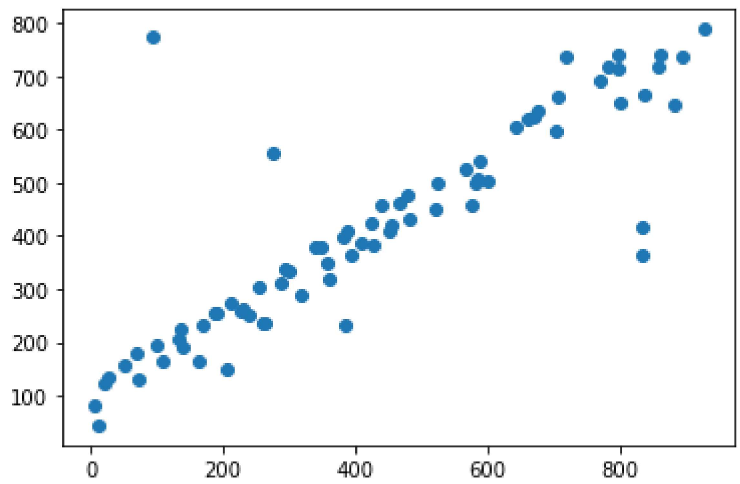
Out[16]:

	Co-efficient
id	2.780843
latitude	0.726757
longitude	-0.288837

Loading [MathJax]/extensions/Safe.js

In [17]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[17]: <matplotlib.collections.PathCollection at 0x12ee10d9d00>



In [18]:
```
print(lr.score(x_test,y_test))
```

0.7365062237836579

In [19]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [20]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[20]: Ridge(alpha=10)

In [21]:
```

rr.score(x_train,y_train)
```

Out[21]: 0.6586767670226825

In [22]:
```
rr.score(x_test,y_test)
```

Out[22]: 0.7365048972940147

In [23]:
```
rr.score(x_test,y_test)
```

Out[23]: 0.7365048972940147

In [24]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[24]: Lasso(alpha=10)

In [25]:
```
la.score(x_test,y_test)
```

Out[25]: 0.7365929117743792

Loading [MathJax]/extensions/Safe.js

In [26]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[26]:
```
ElasticNet()
```

In [27]:
```
print(en.coef_)
```

```
[ 2.78047604  0.72551035 -0.28868996]
```

In [28]:
```
print(en.intercept_)
```

```
74.46648557167816
```

In [29]:
```
print(en.predict(x_test))
```

```
[540.36299798  44.03809115 691.63849276 272.96679298 407.8497359
 132.48223705 478.92714203 603.88887643 503.13740253 738.9202379
 789.03115981 190.26256265 364.12416657 164.24561902 363.81753234
 598.80673093 235.64441355 380.02699853 557.34883036 738.07316218
 253.33677601 147.44070652 156.81523502  82.42395581 399.34859436
 204.22225195 263.93367286 319.80469489 742.07978701 347.57564595
 410.371934   231.85972448 776.58939669 121.55049685 225.85299698
 714.55308761 662.52286624 192.98236714 458.50648758 378.05880111
 419.09899872 237.22474887 259.32250151 651.22412747 332.75067378
 636.48648844 624.47681927 666.15305208 163.36363805 432.40325598
 498.72412929 527.06871223 741.92859854 451.30521532 311.62745982
 416.67572998 619.54536119 179.04585096 460.53871107 386.57801956
 255.68685775 458.13461775 645.43346388 720.10334447 508.19898866
 720.10315169 335.7860718  425.1057967  231.61705416 499.41575215
 384.05434348 251.36783152 303.9852741  132.21050401 289.90156466]
```

In [30]:
```
print(en.score(x_test,y_test))
```

```
0.736499008436385
```

Evaluation Metrics

In [31]:
```
from sklearn import metrics
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,pre
```

```
Mean Absolytre Error: 82.93096298133341
Mean Squared Error: 18015.666732470734
Root Mean Squared Error: 134.2224524156474
```

In [32]:
```

import pickle
```

Loading [MathJax]/extensions/Safe.js

In [33]:
```
filename='prediction4'
pickle.dump(lr,open(filename,'wb'))
```

In [ ]:

Loading [MathJax]/extensions/Safe.js