Importing Libraries

In [1]:

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

Importing Datasets

In [2]:

```
df=pd.read_csv("stations.csv")
df
```

Out[2]:

	id	name	address	lon	lat	elevation
0	28079004	Pza. de España	Plaza de España	-3.712247	40.423853	635
1	28079008	Escuelas Aguirre	Entre C/ Alcalá y C/ O' Donell	-3.682319	40.421564	670
2	28079011	Avda. Ramón y Cajal	Avda. Ramón y Cajal esq. C/ Príncipe de Vergara	-3.677356	40.451475	708
3	28079016	Arturo Soria	C/ Arturo Soria esq. C/ Vizconde de los Asilos	-3.639233	40.440047	693
4	28079017	Villaverde	C/. Juan Peñalver	-3.713322	40.347139	604
5	28079018	Farolillo	Calle Farolillo - C/Ervigio	-3.731853	40.394781	630
6	28079024	Casa de Campo	Casa de Campo (Terminal del Teleférico)	-3.747347	40.419356	642
7	28079027	Barajas Pueblo	C/. Júpiter, 21 (Barajas)	-3.580031	40.476928	621
8	28079035	Pza. del Carmen	Plaza del Carmen esq. Tres Cruces.	-3.703172	40.419208	659
9	28079036	Moratalaz	Avd. Moratalaz esq. Camino de los Vinateros	-3.645306	40.407947	685
10	28079038	Cuatro Caminos	Avda. Pablo Iglesias esq. C/ Marqués de Lema	-3.707128	40.445544	698
11	28079039	Barrio del Pilar	Avd. Betanzos esq. C/ Monforte de Lemos	-3.711542	40.478228	674
12	28079040	Vallecas	C/ Arroyo del Olivar esq. C/ Río Grande.	-3.651522	40.388153	677
13	28079047	Mendez Alvaro	C/ Juan de Mariana / Pza. Amanecer Mendez Alvaro	-3.686825	40.398114	599
14	28079048	Castellana	C/ Jose Gutierrez Abascal	-3.690367	40.439897	676
15	28079049	Parque del Retiro	Paseo Venezuela- Casa de Vacas	-3.682583	40.414444	662
16	28079050	Plaza Castilla	Plaza Castilla (Canal)	-3.688769	40.465572	728
17	28079054	Ensanche de Vallecas	Avda La Gavia / Avda. Las Suertes	-3.612117	40.372933	627
18	28079055	Urb. Embajada	C/ Riaño (Barajas)	-3.580747	40.462531	618
19	28079056	Pza. Fernández	Pza. Fernández Ladreda - Avda.	-3.718728	40.384964	604

	id	name	address	lon	lat	elevation
		Ladreda	Oporto			
20	28079057	Sanchinarro	C/ Princesa de Eboli esq C/ Maria Tudor	-3.660503	40.494208	700
21	28079058	El Pardo	Avda. La Guardia	-3.774611	40.518058	615
22	28079059	Juan Carlos I	Parque Juan Carlos I (frente oficinas mantenim...	-3.609072	40.465250	660
23	28079060	Tres Olivos	Plaza Tres Olivos	-3.689761	40.500589	715

Data Cleaning and Data Preprocessing

In [3]:
```
df=df.dropna()
```

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

Out[8]:
```
Index(['id', 'name', 'address', 'lon', 'lat', 'elevation'], dtype='object')
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24 entries, 0 to 23
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   id         24 non-null     int64  
 1   name       24 non-null     object 
 2   address    24 non-null     object 
 3   lon        24 non-null     float64
 4   lat        24 non-null     float64
 5   elevation  24 non-null     int64  
dtypes: float64(2), int64(2), object(2)
memory usage: 1.3+ KB
```

In [10]:
```
data=df[['id' ,'lon']]
data
```

Out[10]:

	id	lon
0	28079004	-3.712247
1	28079008	-3.682319
2	28079011	-3.677356
3	28079016	-3.639233
4	28079017	-3.713322
5	28079018	-3.731853
6	28079024	-3.747347
7	28079027	-3.580031
8	28079035	-3.703172
9	28079036	-3.645306

	id	lon
10	28079038	-3.707128
11	28079039	-3.711542
12	28079040	-3.651522
13	28079047	-3.686825
14	28079048	-3.690367
15	28079049	-3.682583
16	28079050	-3.688769
17	28079054	-3.612117
18	28079055	-3.580747
19	28079056	-3.718728
20	28079057	-3.660503
21	28079058	-3.774611
22	28079059	-3.609072
23	28079060	-3.689761

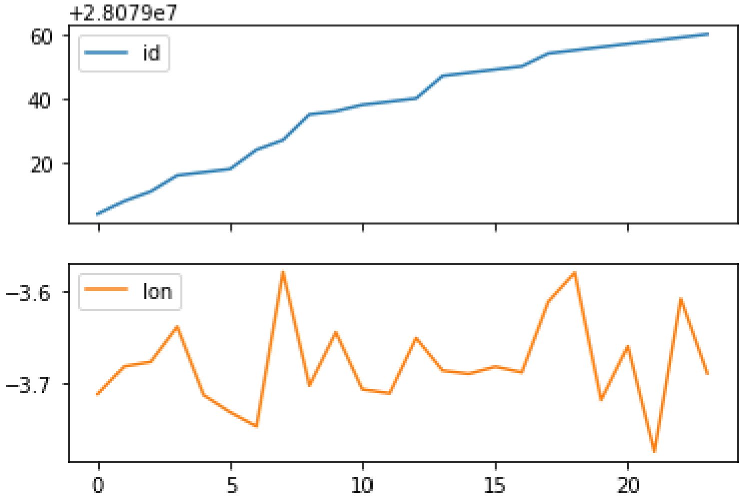
Line chart

In [11]:

```
data.plot.line(subplots=True)
```

Out[11]:

```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



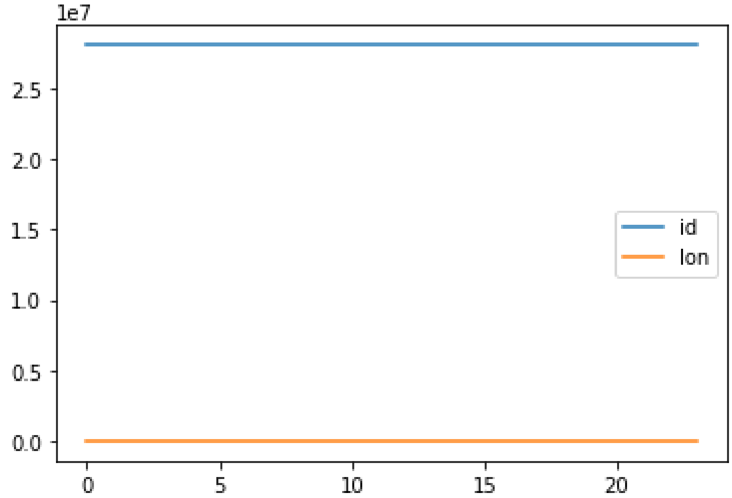
Line chart

In [12]:

```
data.plot.line()
```

Out[12]:

```
<AxesSubplot:>
```

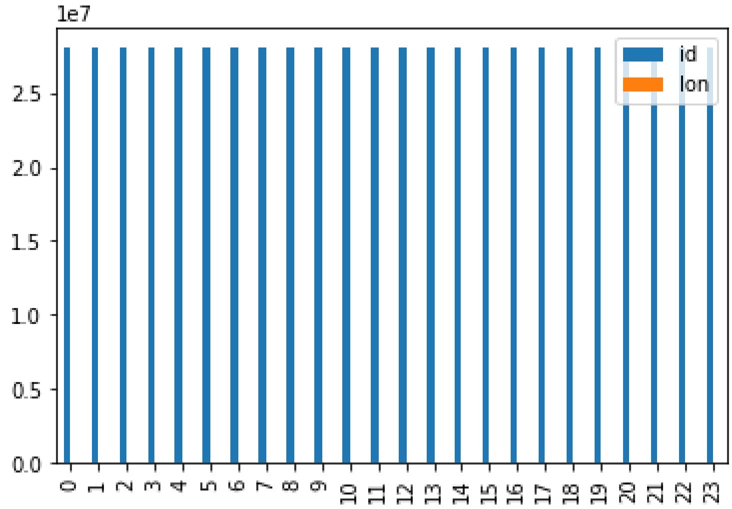


Bar chart

In [13]:
```
b=data[0:50]
```

In [14]:
```
b.plot.bar()
```

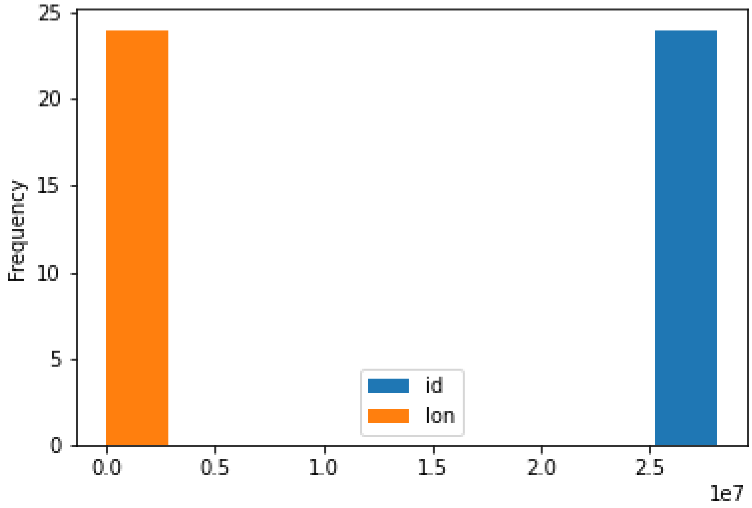
Out[14]: <AxesSubplot:>



Histogram

In [15]:
```
data.plot.hist()
```

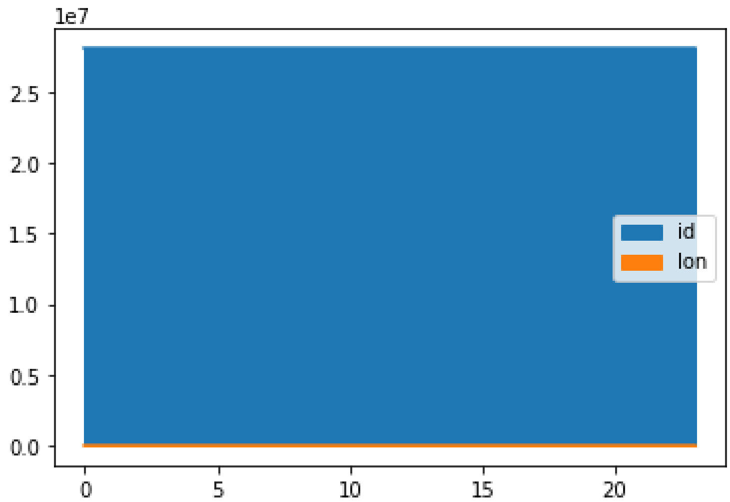
Out[15]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [16]:
```
data.plot.area()
```

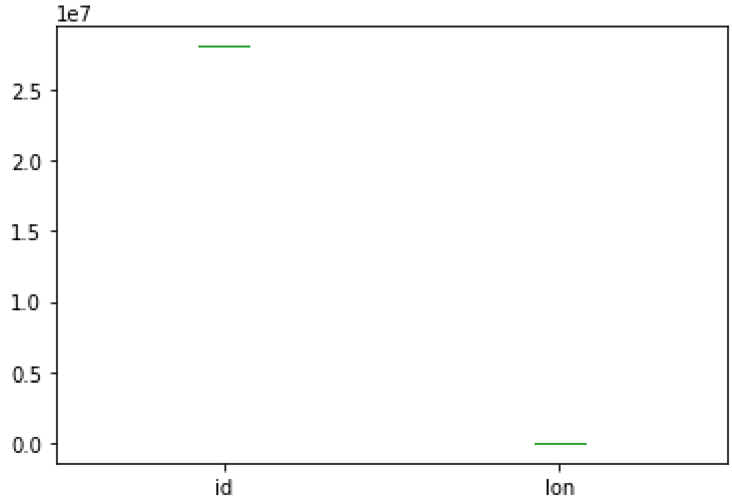
Out[16]: <AxesSubplot:>



Box chart

In [17]:
```
data.plot.box()
```

Out[17]: <AxesSubplot:>

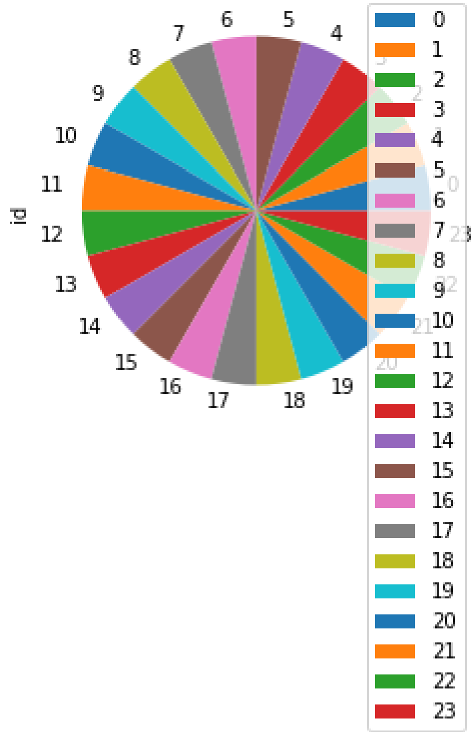


Pie chart

In [19]:

```
b.plot.pie(y='id')
```

Out[19]: <AxesSubplot:ylabel='id'>

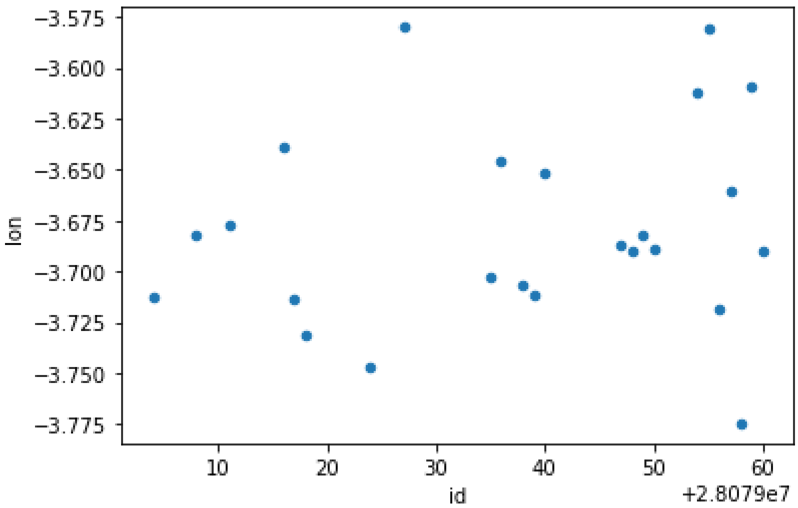


Scatter chart

In [106…

```
data.plot.scatter(x='id' ,y='lon')
```

Out[106… <AxesSubplot:xlabel='id', ylabel='lon'>



In [22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24 entries, 0 to 23
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   id         24 non-null     int64  
 1   name       24 non-null     object 
 2   address    24 non-null     object 
 3   lon        24 non-null     float64
 4   lat        24 non-null     float64
 5   elevation  24 non-null     int64  
dtypes: float64(2), int64(2), object(2)
memory usage: 1.3+ KB
```

In [19]:
```
df.columns
```

Out[19]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [17]:
```
df.describe()
```

Out[17]:

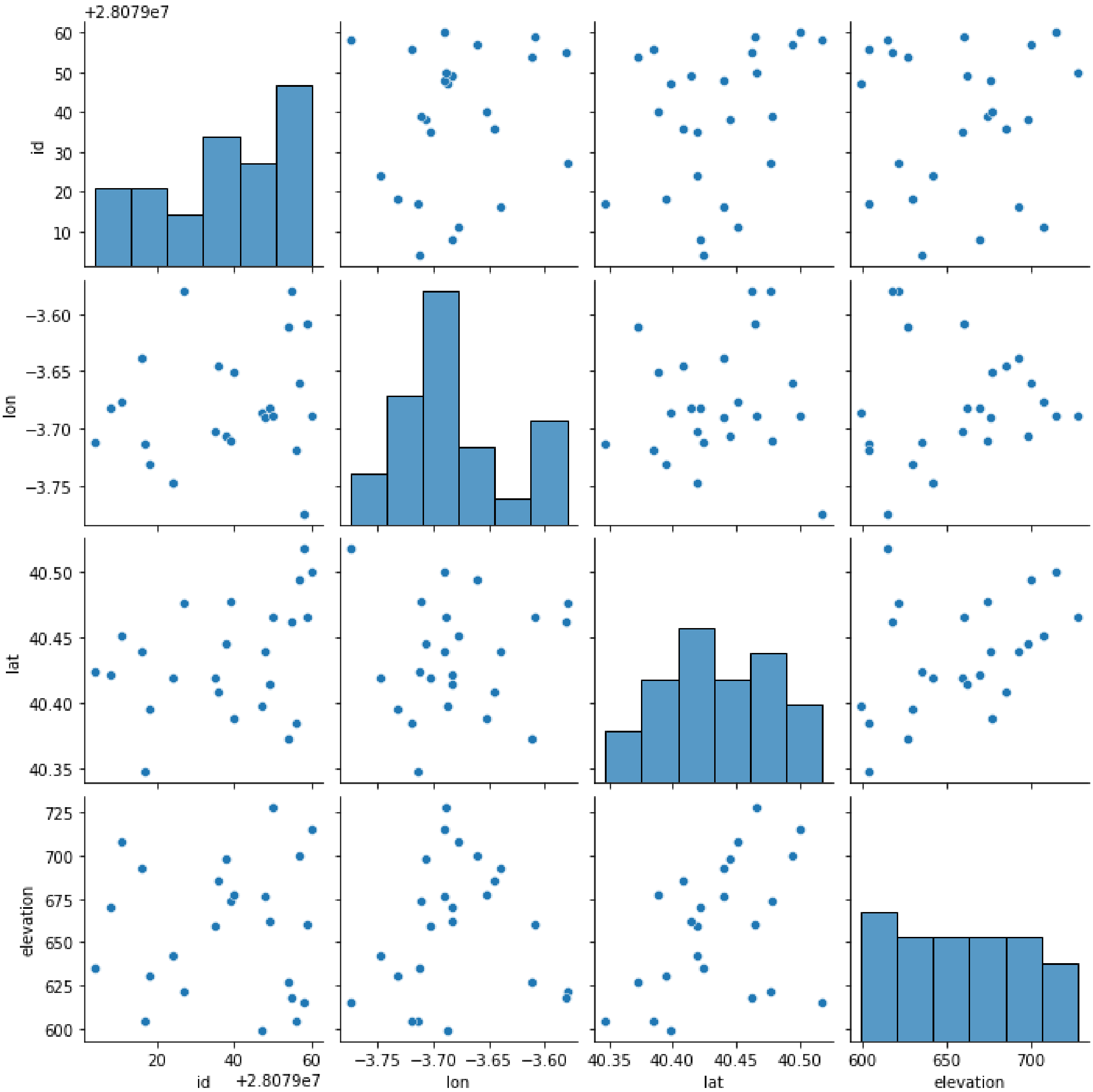
	BEN	CO	EBE	NMHC	NO	NO_2	
count	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.0C
mean	0.784014	0.279333	0.992213	0.215755	18.795529	31.262642	44.23
std	0.632755	0.167922	0.804554	0.075169	40.038872	27.234732	29.53
min	0.100000	0.100000	0.100000	0.050000	0.000000	1.000000	1.0C
25%	0.400000	0.200000	0.500000	0.160000	1.000000	9.000000	18.0C
50%	0.600000	0.200000	0.800000	0.220000	3.000000	24.000000	44.0C
75%	0.900000	0.300000	1.200000	0.250000	18.000000	47.000000	65.0C
max	7.000000	2.500000	9.700000	0.670000	525.000000	225.000000	157.0C

In [23]:
```
df1=df[['id', 'name', 'address', 'lon', 'lat', 'elevation']]
```

EDA AND VISUALIZATION

In [24]:
```
sns.pairplot(df1[0:50])
```

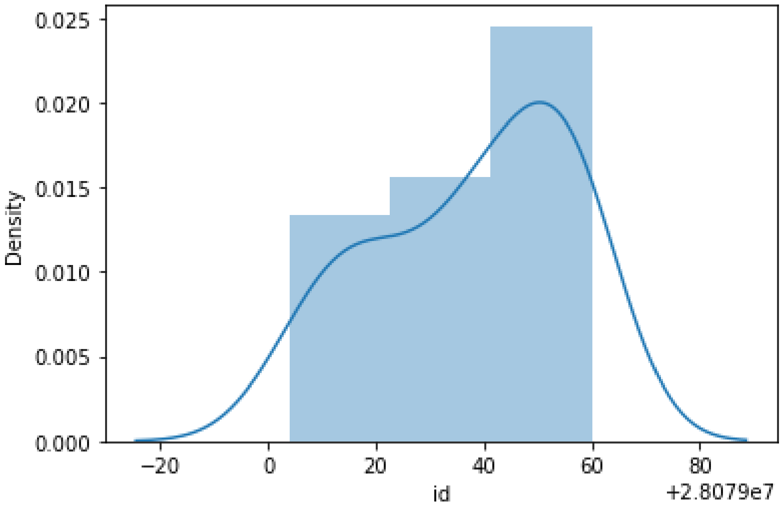
Out[24]: <seaborn.axisgrid.PairGrid at 0x190514fe400>



In [26]:
```
sns.distplot(df1['id'])
```

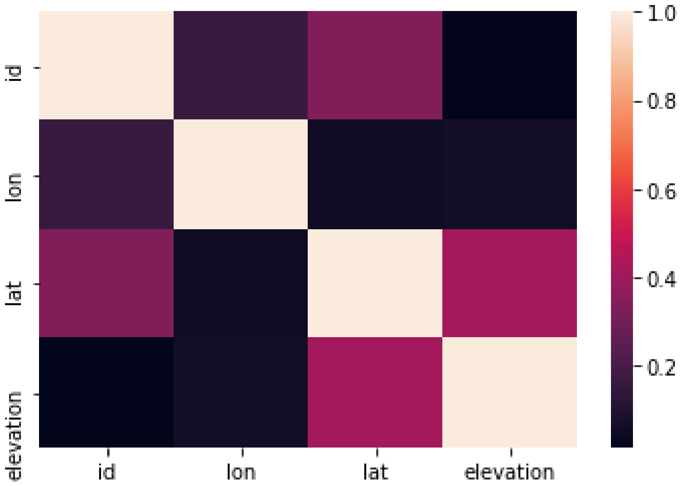
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function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

Out[26]: <AxesSubplot:xlabel='id', ylabel='Density'>



In [27]:
```
sns.heatmap(df1.corr())
```

Out[27]: <AxesSubplot:>



In [29]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24 entries, 0 to 23
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   id         24 non-null     int64  
 1   name       24 non-null     object 
 2   address    24 non-null     object 
 3   lon        24 non-null     float64
 4   lat        24 non-null     float64
 5   elevation  24 non-null     int64  
dtypes: float64(2), int64(2), object(2)
memory usage: 1.3+ KB
```

TO TRAIN THE MODEL AND MODEL BULDING

In [30]:
```
x=df[['id']]
y=df['elevation']
```

In [31]:
```
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)
```

Linear Regression

In [32]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[32]: LinearRegression()

In [33]:
```
lr.intercept_
```

Out[33]: 4206047.767358211

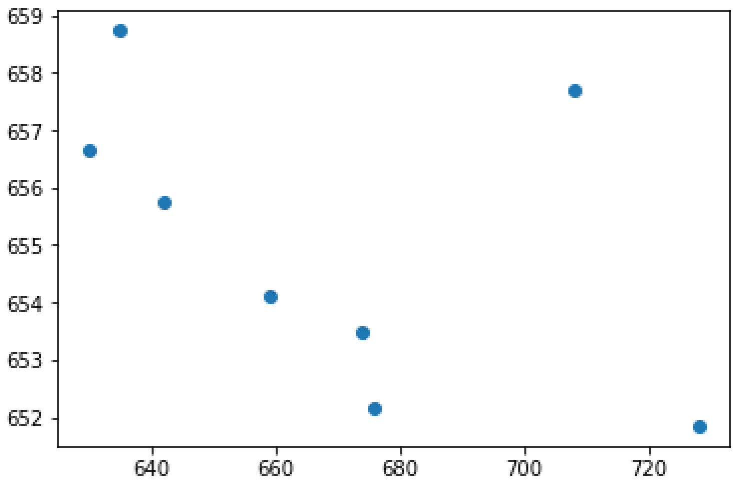
In [34]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[34]:

	Co-efficient
id	-0.14977

In [35]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[35]: <matplotlib.collections.PathCollection at 0x190528c32e0>



ACCURACY

In [36]:
```
lr.score(x_test,y_test)
```

Out[36]: -0.25910603716409164

In [37]:
```
lr.score(x_train,y_train)
```

Out[37]: 0.004162373438242106

Ridge and Lasso

In [38]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [39]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[39]: Ridge(alpha=10)

Accuracy(Ridge)

In [40]:
```
rr.score(x_test,y_test)
```

Out[40]: -0.2590389528358372

In [41]:
```
rr.score(x_train,y_train)
```

Out[41]: 0.004162352167841776

In [42]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[42]: Lasso(alpha=10)

In [43]:
```
la.score(x_train,y_train)
```

Out[43]: 0.003918492372358862

Accuracy(Lasso)

In [44]:
```
la.score(x_test,y_test)
```

Out[44]: -0.2524587869827213

Elastic Net

In [45]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[45]: ElasticNet()

In [46]:
```
en.coef_
```

Out[46]:
```
array([-0.14768952])
```

In [47]:
```
en.intercept_
```

Out[47]:
```
4147633.394964873
```

In [48]:
```
prediction=en.predict(x_test)
```

In [49]:
```
en.score(x_test,y_test)
```

Out[49]:
```
-0.2586953411650055
```

Evaluation Metrics

In [50]:
```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test,prediction))
print(metrics.mean_squared_error(y_test,prediction))
print(np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
29.966904852713924
1350.2654272347597
36.74595797138455
```

Logistic Regression

In [58]:
```
from sklearn.linear_model import LogisticRegression
```

In [59]:
```
feature_matrix=df[['id']]
target_vector=df['elevation']
```

In [60]:
```
feature_matrix.shape
```

Out[60]:
```
(24, 1)
```

In [61]:
```
target_vector.shape
```

Out[61]:
```
(24,)
```

In [62]:
```
from sklearn.preprocessing import StandardScaler
```

In [63]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [64]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[64]: LogisticRegression(max_iter=10000)

In [70]:
```
observation=[[1]]
```

In [71]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[604]
```

In [72]:
```
logr.classes_
```

Out[72]:
```
array([599, 604, 615, 618, 621, 627, 630, 635, 642, 659, 660, 662, 670,
       674, 676, 677, 685, 693, 698, 700, 708, 715, 728], dtype=int64)
```

In [73]:
```
logr.score(fs,target_vector)
```

Out[73]: 0.16666666666666666

In [74]:
```
logr.predict_proba(observation)[0][0]
```

Out[74]: 0.05149080255479361

In [75]:
```
logr.predict_proba(observation)
```

Out[75]:
```
array([[0.0514908 , 0.07628281, 0.06459573, 0.06111161, 0.02846647,
        0.05992926, 0.01982391, 0.00955813, 0.02542849, 0.03721458,
        0.06573175, 0.05391838, 0.01206135, 0.04186834, 0.05270507,
        0.04305319, 0.03836362, 0.0181069 , 0.04069126, 0.06344616,
        0.01416966, 0.06685293, 0.05512959]])
```

Random Forest

In [76]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [77]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[77]: RandomForestClassifier()

In [95]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [96]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="accuracy
grid_search.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: Us
erWarning: The least populated class in y has only 1 members, which is less than n_s
```

```
plits=2.
  warnings.warn(("The least populated class in y has only %d"
```

Out[96]:
```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1], 'min_samples_leaf': [5],
                         'n_estimators': [10]},
             scoring='accuracy')
```

In [97]:
```
grid_search.best_score_
```

Out[97]:
```
0.0
```

In [98]:
```
rfc_best=grid_search.best_estimator_
```

Conclusion

Scores

Linear Regression

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

In [ ]:
```
lr.score(x_train,y_train)
```

Lasso

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

Ridge

In [83]:
```
rr.score(x_test,y_test)
```

Out[83]:
```
-0.2590389528358372
```

In [84]:
```
rr.score(x_train,y_train)
```

Out[84]:
```
0.004162352167841776
```

Elastic Net

In [85]:
```
en.score(x_test,y_test)
```

Out[85]:
```
-0.2586953411650055
```

Logistic Regression

In [86]:
```
logr.score(fs,target_vector)
```

Out[86]: 0.16666666666666666

Random Forest

In [87]:
```
grid_search.best_score_
```

Out[87]: 0.125

From the above data, we can conclude that logistic regression is preferrable to other regression types

In [ ]:
```
```