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(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs(Dataset)\station
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 Ladreda	Pza. Fernández Ladreda - Avda. Oporto	-3.718728	40.384964	604
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 [4]:
```
df.columns
```

Out[4]:
```
Index(['id', 'name', 'address', 'lon', 'lat', 'elevation'], dtype='object')
```

In [5]:
```
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 [6]:
```
data=df[['id', 'lon']]
data
```

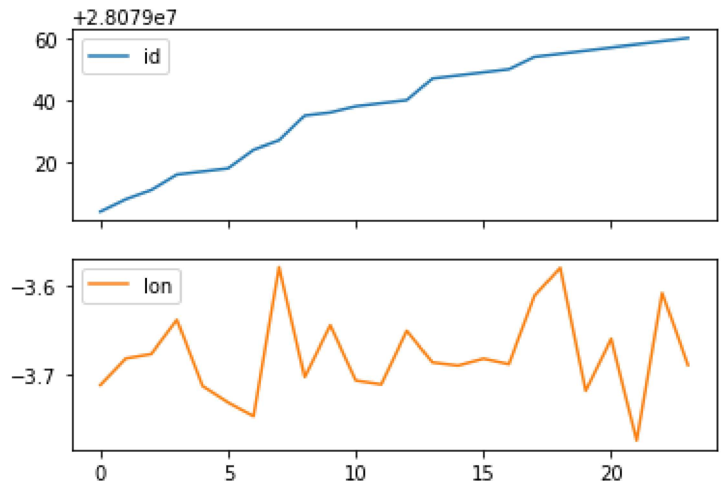
Out[6]:

	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
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 [7]:
```
data.plot.line(subplots=True)
```

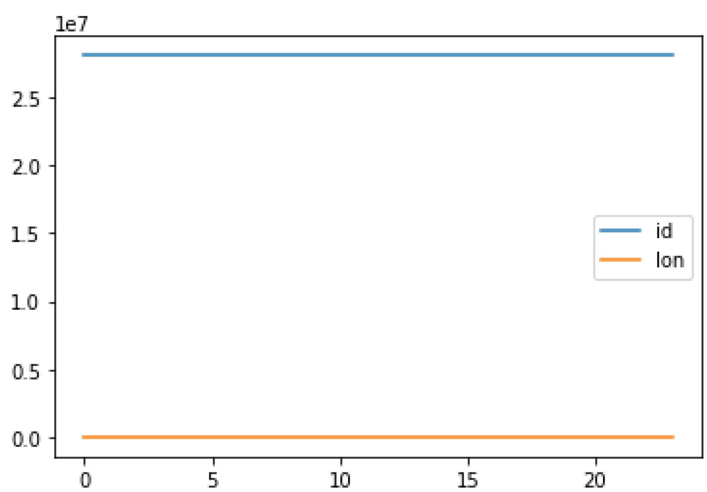
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

In [8]:
```
data.plot.line()
```

Out[8]: <AxesSubplot:>

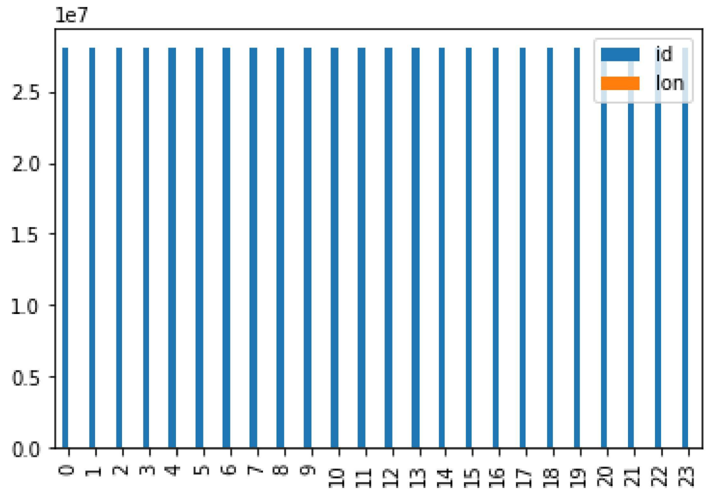


Bar chart

In [9]:
```
b=data[0:50]
```

In [10]:
```
b.plot.bar()
```

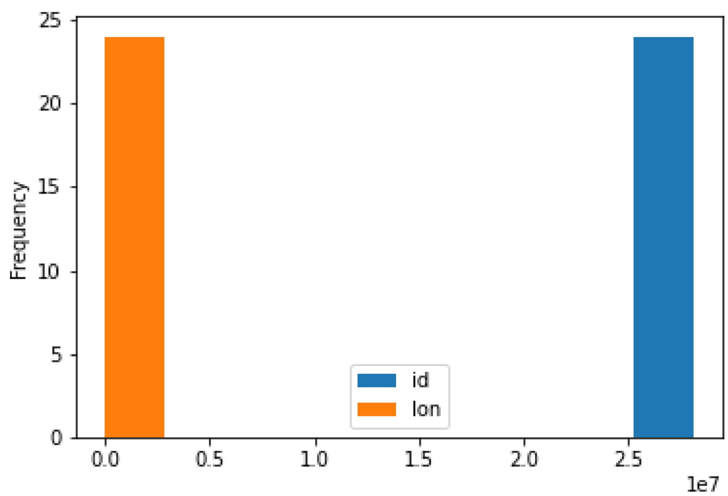
Out[10]: <AxesSubplot:>



Histogram

In [11]:
```
data.plot.hist()
```

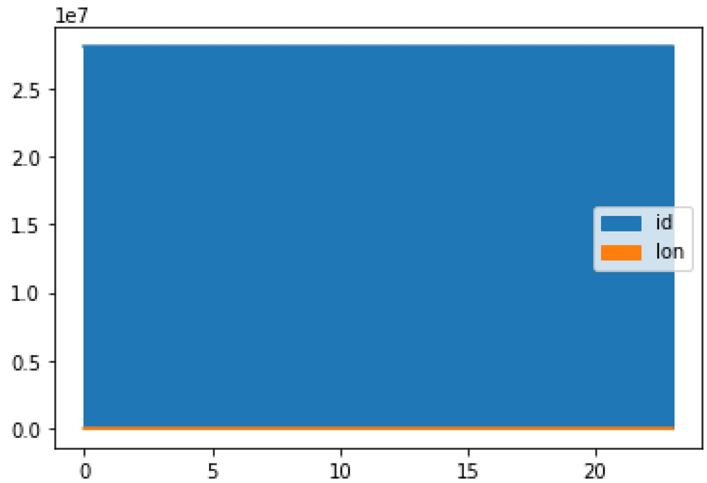
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [12]:
```
data.plot.area()
```

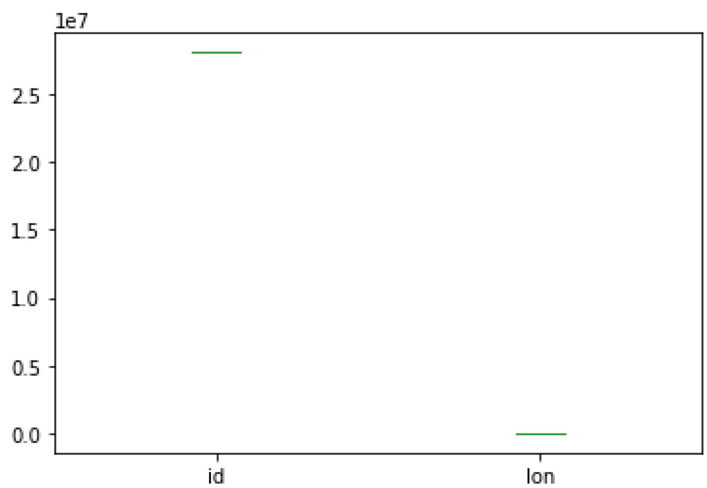
Out[12]: <AxesSubplot:>



Box chart

In [13]:
```
data.plot.box()
```

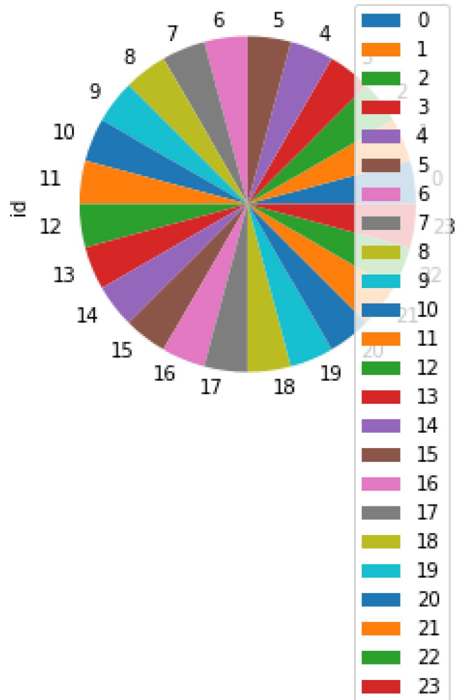
Out[13]: <AxesSubplot:>



Pie chart

In [16]:
```
b.plot.pie(y='id' )
```

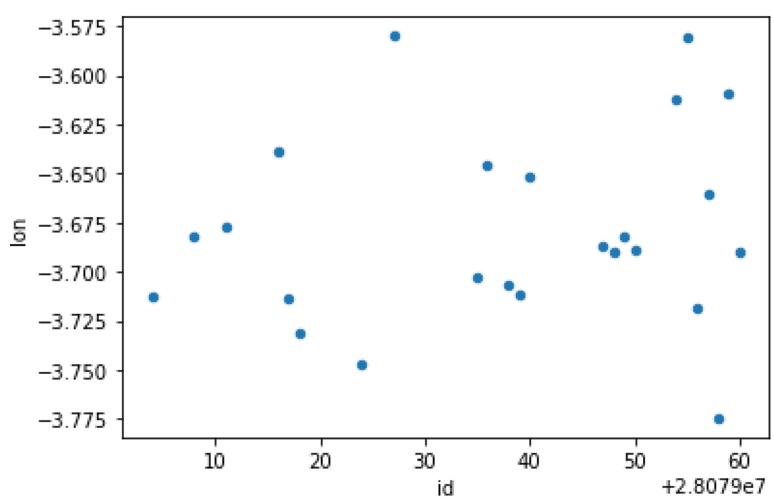
Out[16]: <AxesSubplot:ylabel='id'>



Scatter chart

In [17]:
```
data.plot.scatter(x='id' ,y='lon')
```

Out[17]: <AxesSubplot:xlabel='id', ylabel='lon'>



In [18]:
```
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.describe()
```

Out[19]:

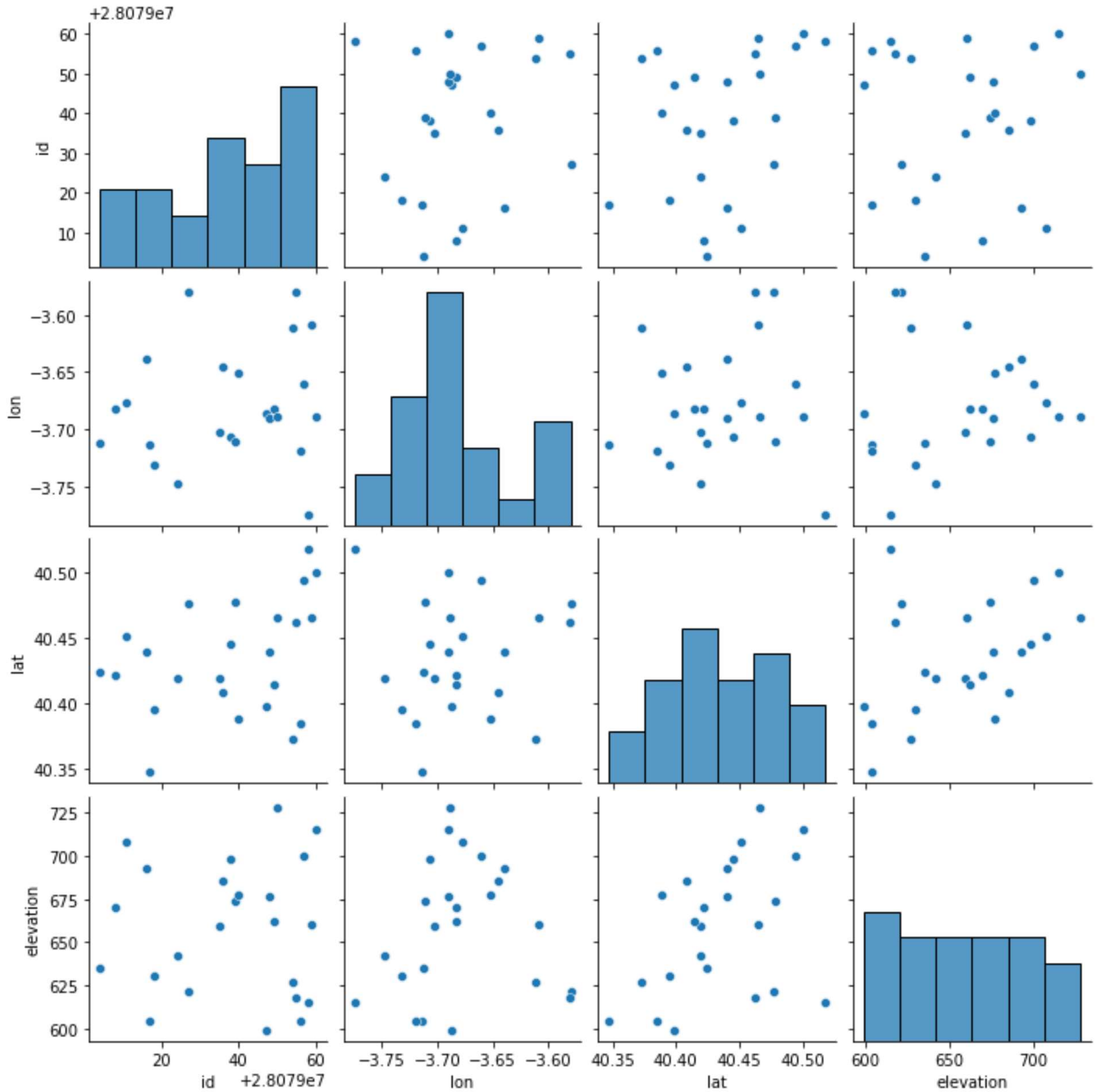
	id	lon	lat	elevation
count	2.400000e+01	24.000000	24.000000	24.000000
mean	2.807904e+07	-3.679019	40.434616	658.333333
std	1.799094e+01	0.049324	0.043022	38.295949
min	2.807900e+07	-3.774611	40.347139	599.000000
25%	2.807902e+07	-3.711718	40.405489	625.500000
50%	2.807904e+07	-3.687797	40.431875	661.000000
75%	2.807905e+07	-3.649968	40.465331	687.000000
max	2.807906e+07	-3.580031	40.518058	728.000000

In [20]:
```
df1=df[['id', 'name', 'address', 'lon', 'lat', 'elevation']]
```

EDA AND VISUALIZATION

In [21]:
```
sns.pairplot(df1[0:50])
```

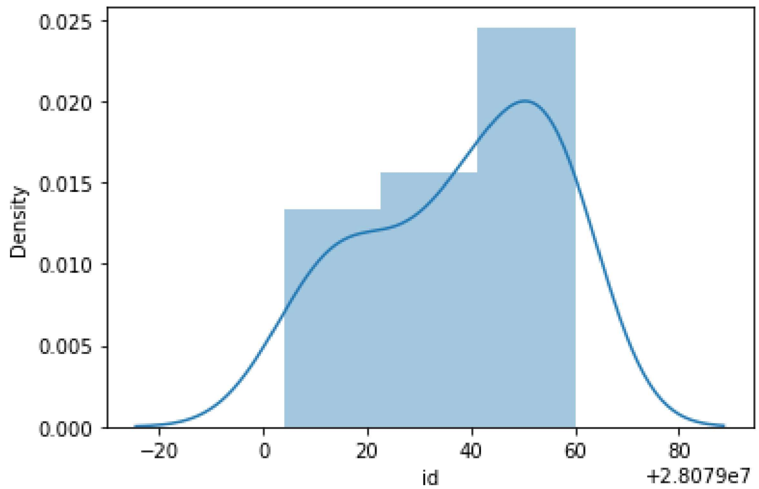
Out[21]: `<seaborn.axisgrid.PairGrid at 0x1814e1f27f0>`



In [24]:
```
sns.distplot(df1['id'])
```

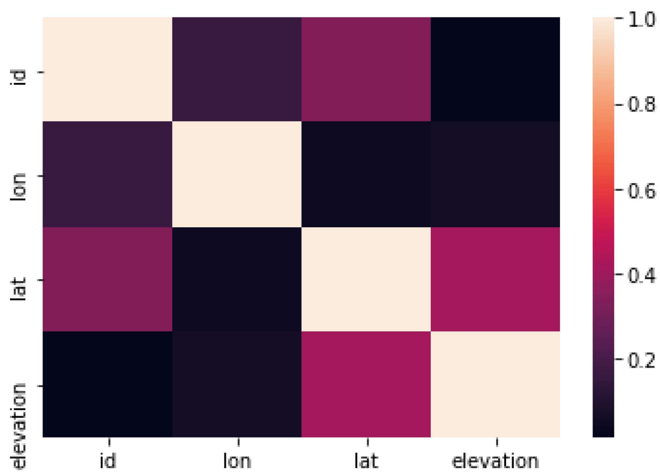
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWar
ning: `distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function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

Out[24]: <AxesSubplot:xlabel='id', ylabel='Density'>



In [25]:
```
sns.heatmap(df1.corr())
```

Out[25]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [26]:
```
x=df[['id']]
y=df['elevation']
```

In [27]:
```
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 [28]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

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

In [29]:
```
lr.intercept_
```

Out[29]:
```
-10043641.837007152
```

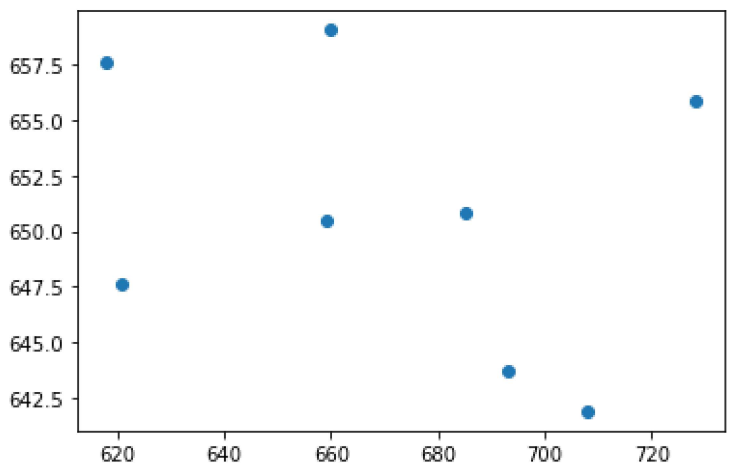
In [30]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[30]:

	Co-efficient
id	0.357715

In [31]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[31]:
```
<matplotlib.collections.PathCollection at 0x18150ae9bb0>
```



ACCURACY

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

Out[32]:
```
-0.4286690092552836
```

In [33]:
```
lr.score(x_train,y_train)
```

Out[33]:
```
0.03214202174913028
```

Ridge and Lasso

In [34]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [35]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[35]: Ridge(alpha=10)

Accuracy(Ridge)

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

Out[36]: -0.4283479389212217

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

Out[37]: 0.03214190424205121

In [38]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[38]: Lasso(alpha=10)

In [39]:
```
la.score(x_test,y_test)
```

Out[39]: -0.4144698759213483

Accuracy(Lasso)

In [40]:
```
la.score(x_train,y_train)
```

Out[40]: 0.0319060238163732

Accuracy(Elastic Net)

In [41]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[41]: ElasticNet()

In [42]:
```
en.coef_
```

Out[42]: array([0.35563738])

In [43]:
```
en.intercept_
```

Out[43]: -9985303.982509833

In [44]:
```
prediction=en.predict(x_test)
```

In [45]:
```
en.score(x_test,y_test)
```

Out[45]:
```
-0.42769432950583486
```

Evaluation Metrics

In [46]:
```
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)))
```

```
37.17529917322099
1939.8796702160532
44.0440650964015
```

Logistic Regression

In [47]:
```
from sklearn.linear_model import LogisticRegression
```

In [48]:
```
feature_matrix=df[['id']]
target_vector=df[ 'elevation']
```

In [49]:
```
feature_matrix.shape
```

Out[49]:
```
(24, 1)
```

In [50]:
```
target_vector.shape
```

Out[50]:
```
(24,)
```

In [51]:
```
from sklearn.preprocessing import StandardScaler
```

In [52]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [53]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[53]:
```
LogisticRegression(max_iter=10000)
```

In [54]:
```
observation=[[1]]
```

In [55]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[604]
```

In [56]:
```
logr.classes_
```

Out[56]:
```
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 [57]:
```
logr.score(fs,target_vector)
```

Out[57]:
```
0.16666666666666666
```

In [58]:
```
logr.predict_proba(observation)[0][0]
```

Out[58]:
```
0.05149080255479361
```

In [59]:
```
logr.predict_proba(observation)
```

Out[59]:
```
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 [60]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [61]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[61]:
```
RandomForestClassifier()
```

In [62]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [63]:
```
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: U
serWarning: The least populated class in y has only 1 members, which is less than n
_splits=2.
  warnings.warn(("The least populated class in y has only %d"
```

Out[63]:
```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [64]:
```
grid_search.best_score_
```

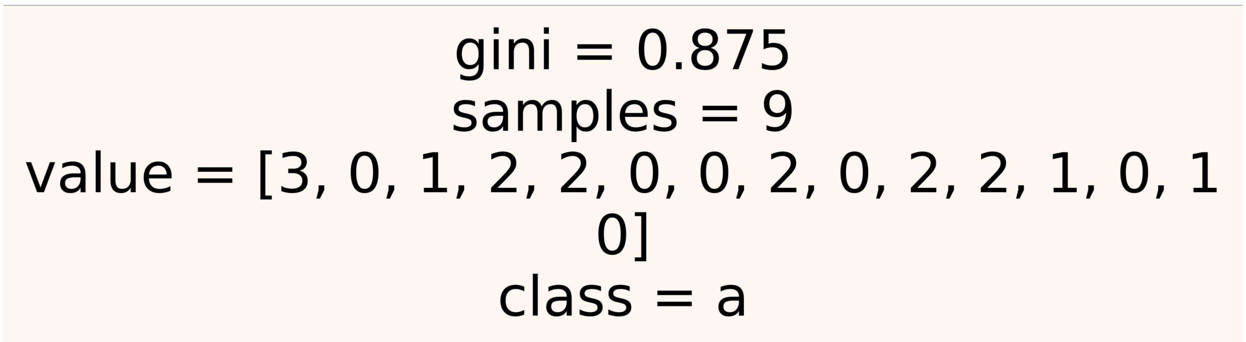
Out[64]:
```
0.125
```

In [65]:
```
rfc_best=grid_search.best_estimator_
```

In [66]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b','c','
```

Out[66]:
```
[Text(2232.0, 1087.2, 'gini = 0.875\nsamples = 9\nvalue = [3, 0, 1, 2, 2, 0, 0, 2,
0, 2, 2, 1, 0, 1\n0]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.03214202174913028

Ridge Regression:0.03214190424205121

Lasso Regression:0.0319060238163732

ElasticNet Regression:-0.42769432950583486

Logistic Regression:0.16666666666666666

Random Forest:0.125

Logistic Regression is suitable for this dataset