## PROBLEM STATEMENT:-

TO PREDICT THE RAIN FALL BASED ON VARIOUS FEATURES OF THE DATASET

## 1.Data Collection

In [1]:
```python
#importing libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing,svm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
df=pd.read_csv(r"C:\Users\DELL\Downloads\rainfall in india 1901-2015.csv")
df
```

Out[2]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | ANNUAL | Jan-Feb | Mar-May | Jun-Sep | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ANDAMAN & NICOBAR ISLANDS | 1901 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | 332.6 | 388.5 | 558.2 | 33.6 | 3373.2 | 136.3 | 560.3 | 1696.3 | 9 |
| **1** | ANDAMAN & NICOBAR ISLANDS | 1902 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | 666.2 | 197.2 | 359.0 | 160.5 | 3520.7 | 159.8 | 458.3 | 2185.9 | 7 |
| **2** | ANDAMAN & NICOBAR ISLANDS | 1903 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | 339.0 | 181.2 | 284.4 | 225.0 | 2957.4 | 156.7 | 236.1 | 1874.0 | 6 |
| **3** | ANDAMAN & NICOBAR ISLANDS | 1904 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | 820.4 | 222.2 | 308.7 | 40.1 | 3079.6 | 24.1 | 506.9 | 1977.6 | 5 |
| **4** | ANDAMAN & NICOBAR ISLANDS | 1905 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | 297.0 | 260.7 | 25.4 | 344.7 | 2566.7 | 1.3 | 309.7 | 1624.9 | 6 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **4111** | LAKSHADWEEP | 2011 | 5.1 | 2.8 | 3.1 | 85.9 | 107.2 | 153.6 | 350.2 | 254.0 | 255.2 | 117.4 | 184.3 | 14.9 | 1533.7 | 7.9 | 196.2 | 1013.0 | 3 |
| **4112** | LAKSHADWEEP | 2012 | 19.2 | 0.1 | 1.6 | 76.8 | 21.2 | 327.0 | 231.5 | 381.2 | 179.8 | 145.9 | 12.4 | 8.8 | 1405.5 | 19.3 | 99.6 | 1119.5 | 1 |
| **4113** | LAKSHADWEEP | 2013 | 26.2 | 34.4 | 37.5 | 5.3 | 88.3 | 426.2 | 296.4 | 154.4 | 180.0 | 72.8 | 78.1 | 26.7 | 1426.3 | 60.6 | 131.1 | 1057.0 | 1 |
| **4114** | LAKSHADWEEP | 2014 | 53.2 | 16.1 | 4.4 | 14.9 | 57.4 | 244.1 | 116.1 | 466.1 | 132.2 | 169.2 | 59.0 | 62.3 | 1395.0 | 69.3 | 76.7 | 958.5 | 2 |
| **4115** | LAKSHADWEEP | 2015 | 2.2 | 0.5 | 3.7 | 87.1 | 133.1 | 296.6 | 257.5 | 146.4 | 160.4 | 165.4 | 231.0 | 159.0 | 1642.9 | 2.7 | 223.9 | 860.9 | 5 |

4116 rows × 19 columns

## 2.Data Preprocessing

In [3]:
```
df.head()
```

Out[3]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | ANNUAL | Jan-Feb | Mar-May | Jun-Sep | Oct-Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ANDAMAN & NICOBAR ISLANDS | 1901 | 49.2 | 87.1 | 29.2 | 2.3 | 528.8 | 517.5 | 365.1 | 481.1 | 332.6 | 388.5 | 558.2 | 33.6 | 3373.2 | 136.3 | 560.3 | 1696.3 | 980.3 |
| **1** | ANDAMAN & NICOBAR ISLANDS | 1902 | 0.0 | 159.8 | 12.2 | 0.0 | 446.1 | 537.1 | 228.9 | 753.7 | 666.2 | 197.2 | 359.0 | 160.5 | 3520.7 | 159.8 | 458.3 | 2185.9 | 716.7 |
| **2** | ANDAMAN & NICOBAR ISLANDS | 1903 | 12.7 | 144.0 | 0.0 | 1.0 | 235.1 | 479.9 | 728.4 | 326.7 | 339.0 | 181.2 | 284.4 | 225.0 | 2957.4 | 156.7 | 236.1 | 1874.0 | 690.6 |
| **3** | ANDAMAN & NICOBAR ISLANDS | 1904 | 9.4 | 14.7 | 0.0 | 202.4 | 304.5 | 495.1 | 502.0 | 160.1 | 820.4 | 222.2 | 308.7 | 40.1 | 3079.6 | 24.1 | 506.9 | 1977.6 | 571.0 |
| **4** | ANDAMAN & NICOBAR ISLANDS | 1905 | 1.3 | 0.0 | 3.3 | 26.9 | 279.5 | 628.7 | 368.7 | 330.5 | 297.0 | 260.7 | 25.4 | 344.7 | 2566.7 | 1.3 | 309.7 | 1624.9 | 630.8 |

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

Out[4]:

| | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | ANNUAL | Jan-Feb | Mar-May | Jun-Sep | Oc De |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **4111** | LAKSHADWEEP | 2011 | 5.1 | 2.8 | 3.1 | 85.9 | 107.2 | 153.6 | 350.2 | 254.0 | 255.2 | 117.4 | 184.3 | 14.9 | 1533.7 | 7.9 | 196.2 | 1013.0 | 316 |
| **4112** | LAKSHADWEEP | 2012 | 19.2 | 0.1 | 1.6 | 76.8 | 21.2 | 327.0 | 231.5 | 381.2 | 179.8 | 145.9 | 12.4 | 8.8 | 1405.5 | 19.3 | 99.6 | 1119.5 | 167 |
| **4113** | LAKSHADWEEP | 2013 | 26.2 | 34.4 | 37.5 | 5.3 | 88.3 | 426.2 | 296.4 | 154.4 | 180.0 | 72.8 | 78.1 | 26.7 | 1426.3 | 60.6 | 131.1 | 1057.0 | 177 |
| **4114** | LAKSHADWEEP | 2014 | 53.2 | 16.1 | 4.4 | 14.9 | 57.4 | 244.1 | 116.1 | 466.1 | 132.2 | 169.2 | 59.0 | 62.3 | 1395.0 | 69.3 | 76.7 | 958.5 | 290 |
| **4115** | LAKSHADWEEP | 2015 | 2.2 | 0.5 | 3.7 | 87.1 | 133.1 | 296.6 | 257.5 | 146.4 | 160.4 | 165.4 | 231.0 | 159.0 | 1642.9 | 2.7 | 223.9 | 860.9 | 555 |

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

Out[5]:

| | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 4116.000000 | 4112.000000 | 4113.000000 | 4110.000000 | 4112.000000 | 4113.000000 | 4111.000000 | 4109.000000 | 4112.000000 | 4110.000000 | 410 |
| **mean** | 1958.218659 | 18.957320 | 21.805325 | 27.359197 | 43.127432 | 85.745417 | 230.234444 | 347.214334 | 290.263497 | 197.361922 | 9 |
| **std** | 33.140898 | 33.585371 | 35.909488 | 46.959424 | 67.831168 | 123.234904 | 234.710758 | 269.539667 | 188.770477 | 135.408345 | 9 |
| **min** | 1901.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.400000 | 0.000000 | 0.000000 | 0.100000 | |
| **25%** | 1930.000000 | 0.600000 | 0.600000 | 1.000000 | 3.000000 | 8.600000 | 70.350000 | 175.600000 | 155.975000 | 100.525000 | 1 |
| **50%** | 1958.000000 | 6.000000 | 6.700000 | 7.800000 | 15.700000 | 36.600000 | 138.700000 | 284.800000 | 259.400000 | 173.900000 | 6 |
| **75%** | 1987.000000 | 22.200000 | 26.800000 | 31.300000 | 49.950000 | 97.200000 | 305.150000 | 418.400000 | 377.800000 | 265.800000 | 14 |
| **max** | 2015.000000 | 583.700000 | 403.500000 | 605.600000 | 595.100000 | 1168.600000 | 1609.900000 | 2362.800000 | 1664.600000 | 1222.000000 | 94 |

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4116 entries, 0 to 4115
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   SUBDIVISION  4116 non-null   object 
 1   YEAR         4116 non-null   int64  
 2   JAN          4112 non-null   float64
 3   FEB          4113 non-null   float64
 4   MAR          4110 non-null   float64
 5   APR          4112 non-null   float64
 6   MAY          4113 non-null   float64
 7   JUN          4111 non-null   float64
 8   JUL          4109 non-null   float64
 9   AUG          4112 non-null   float64
 10  SEP          4110 non-null   float64
 11  OCT          4109 non-null   float64
 12  NOV          4105 non-null   float64
 13  DEC          4106 non-null   float64
 14  ANNUAL       4090 non-null   float64
 15  Jan-Feb      4110 non-null   float64
 16  Mar-May      4107 non-null   float64
 17  Jun-Sep      4106 non-null   float64
 18  Oct-Dec      4103 non-null   float64
dtypes: float64(17), int64(1), object(1)
memory usage: 611.1+ KB
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
SUBDIVISION     0
YEAR            0
JAN             4
FEB             3
MAR             6
APR             4
MAY             3
JUN             5
JUL             7
AUG             4
SEP             6
OCT             7
NOV            11
DEC            10
ANNUAL         26
Jan-Feb         6
Mar-May         9
Jun-Sep        10
Oct-Dec        13
dtype: int64
```

In [10]:
```
df.fillna(method='ffill',inplace=True)
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
SUBDIVISION    0
YEAR           0
JAN            0
FEB            0
MAR            0
APR            0
MAY            0
JUN            0
JUL            0
AUG            0
SEP            0
OCT            0
NOV            0
DEC            0
ANNUAL         0
Jan-Feb        0
Mar-May        0
Jun-Sep        0
Oct-Dec        0
dtype: int64
```

In [12]:
```
df.columns
```

Out[12]:
```
Index(['SUBDIVISION', 'YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC', 'ANNUAL', 'Jan-Feb', 'Mar-May',
       'Jun-Sep', 'Oct-Dec'],
      dtype='object')
```

In [13]:
```
df.shape
```

Out[13]:
```
(4116, 19)
```

In [14]:
```
df['ANNUAL'].value_counts()
```

Out[14]:
```
ANNUAL
0.0       26
1024.6     4
790.5      4
770.3      4
1114.2     3
          ..
419.8      1
428.9      1
527.8      1
322.9      1
1642.9     1
Name: count, Length: 3713, dtype: int64
```

In [15]:
```
df['Jan-Feb'].value_counts()
```

Out[15]:
```
Jan-Feb
0.0     244
0.1      80
0.2      52
0.3      38
0.4      32
       ...
58.2      1
23.3      1
95.2      1
76.9      1
69.3      1
Name: count, Length: 1220, dtype: int64
```

In [16]:
```
df['Mar-May'].value_counts()
```

Out[16]:
```
Mar-May
0.0      38
0.1      13
0.3      11
8.3      11
2.7      10
         ..
249.5     1
148.8     1
191.9     1
207.0     1
223.9     1
Name: count, Length: 2262, dtype: int64
```

In [17]:
```
df['Jun-Sep'].value_counts()
```

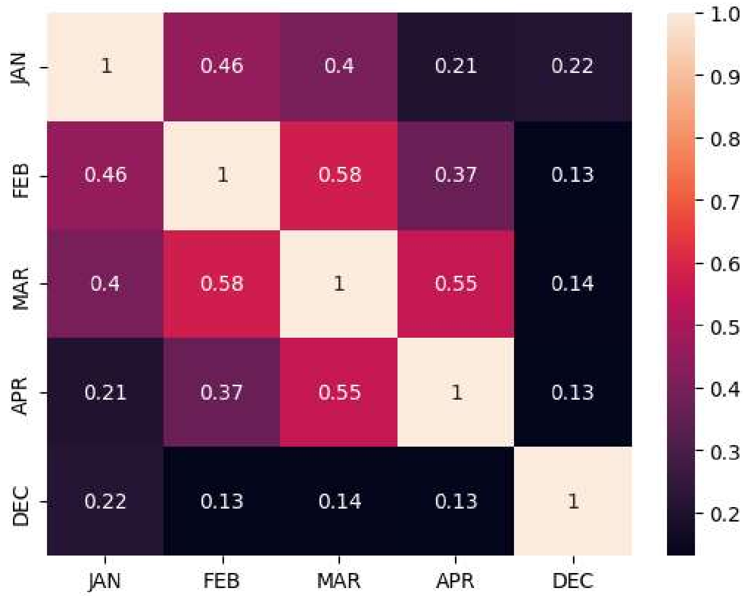
Out[17]:
```
Jun-Sep
0.0       10
573.8      4
613.3      4
434.3      4
334.8      4
          ..
1328.5     1
1073.1     1
1373.0     1
897.7      1
958.5      1
Name: count, Length: 3684, dtype: int64
```

In [18]:
```
df['Oct-Dec'].value_counts()
```

Out[18]:
```
Oct-Dec
0.0      29
0.1      15
0.5      13
0.6      12
0.7      11
         ..
41.5      1
95.4      1
11.7      1
72.9      1
555.4     1
Name: count, Length: 2389, dtype: int64
```

### Exploratory Data Analysis

In [19]:
```
df=df[['JAN','FEB','MAR','APR','DEC']]
sns.heatmap(df.corr(),annot=True)
plt.show()
```




In [20]:
```
df.columns
```

Out[20]:
```
Index(['JAN', 'FEB', 'MAR', 'APR', 'DEC'], dtype='object')
```

In [21]:
```
x=df[["FEB"]]
y=df["JAN"]
```

## 1.Linear Regression

In [22]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=42)
```

In [23]:
```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(X_train,y_train)
print(reg.intercept_)
coeff_=pd.DataFrame(reg.coef_,x.columns,columns=['coefficient'])
coeff_
```

```
9.640361097385627
```

Out[23]:

| | coefficient |
|---|---|
| FEB | 0.442528 |

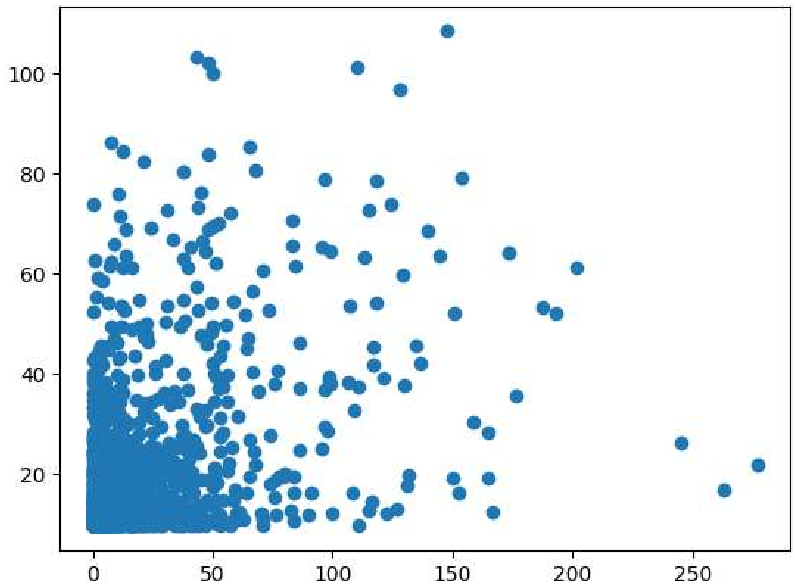
In [24]:
```
score=reg.score(X_test,y_test)
print(score)
```

```
0.17932442210801225
```

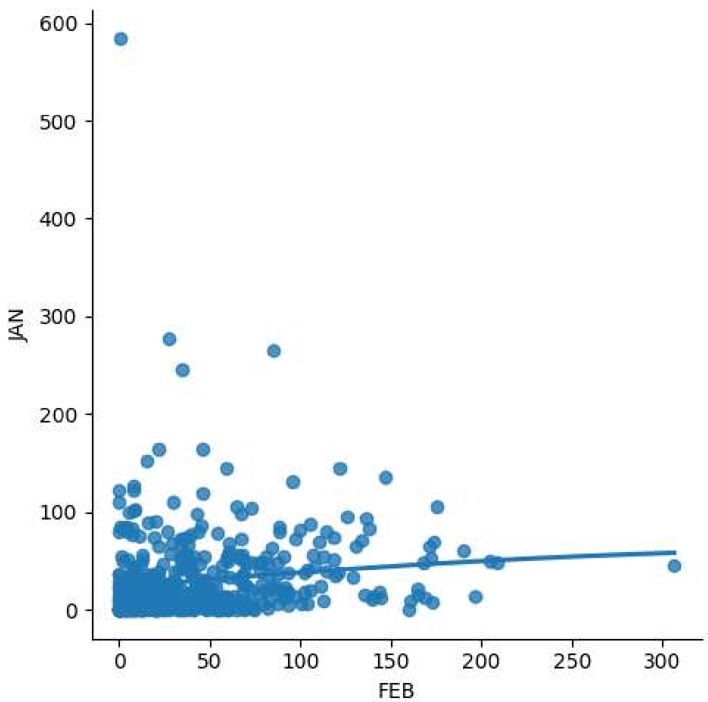
In [25]:
```
predictions=reg.predict(X_test)
```

In [26]:
```python
plt.scatter(y_test,predictions)
```

Out[26]: <matplotlib.collections.PathCollection at 0x172c148e590>




In [27]:
```python
df500=df[:][:500]
sns.lmplot(x="FEB",y="JAN",order=2,ci=None,data=df500)
plt.show()
```

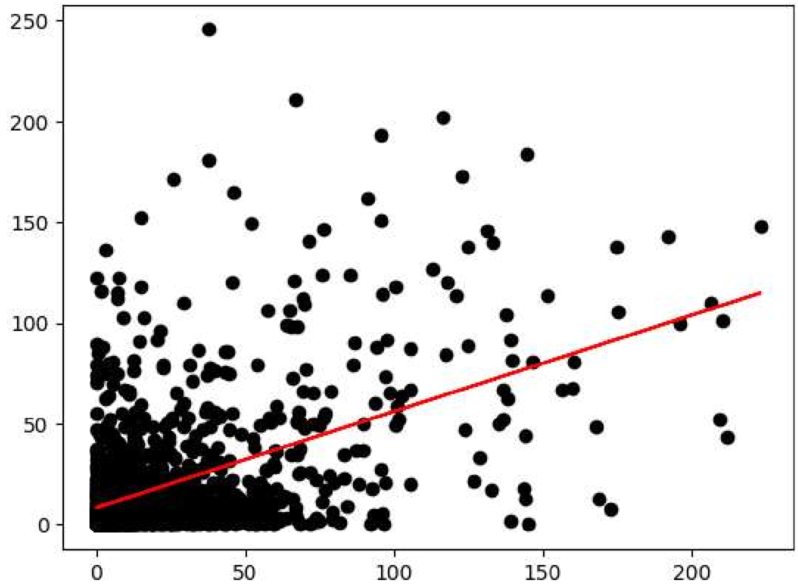



In [28]:
```python
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.33)
reg.fit(X_train,y_train)
reg.fit(X_test,y_test)
```

Out[28]:
▾ LinearRegression

LinearRegression()

In [29]:
```
y_pred=reg.predict(X_test)
plt.scatter(X_test,y_test,color='black')
plt.plot(X_test,y_pred,color='red')
plt.show()
```



In [30]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
model=LinearRegression()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
r2=r2_score(y_test,y_pred)
print("R2 Score:",r2)
```

```
R2 Score: 0.2575267236574985
```

In [31]:
```
#Conclusion:This model has 10% of accuracy
```

## 2.Ridge Regression

In [32]:
```
from sklearn.linear_model import Lasso,Ridge
from sklearn.preprocessing import StandardScaler
```

In [33]:
```
features= df.columns[0:5]
target= df.columns[-5]
```

In [34]:
```
x=np.array(df['JAN']).reshape(-1,1)
y=np.array(df['FEB']).reshape(-1,2)
```

In [35]:
```
x= df[features].values
y= df[target].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=17)
```

In [36]:
```
ridgeReg=Ridge(alpha=10)
ridgeReg.fit(x_train,y_train)
train_score_ridge=ridgeReg.score(x_train,y_train)
test_score_ridge=ridgeReg.score(x_test,y_test)
```

In [37]:
```
print("\n Ridge Model:\n")
print("the train score for ridge model is{}".format(train_score_ridge))
print("the test score for ridge model is{}".format(test_score_ridge))
```

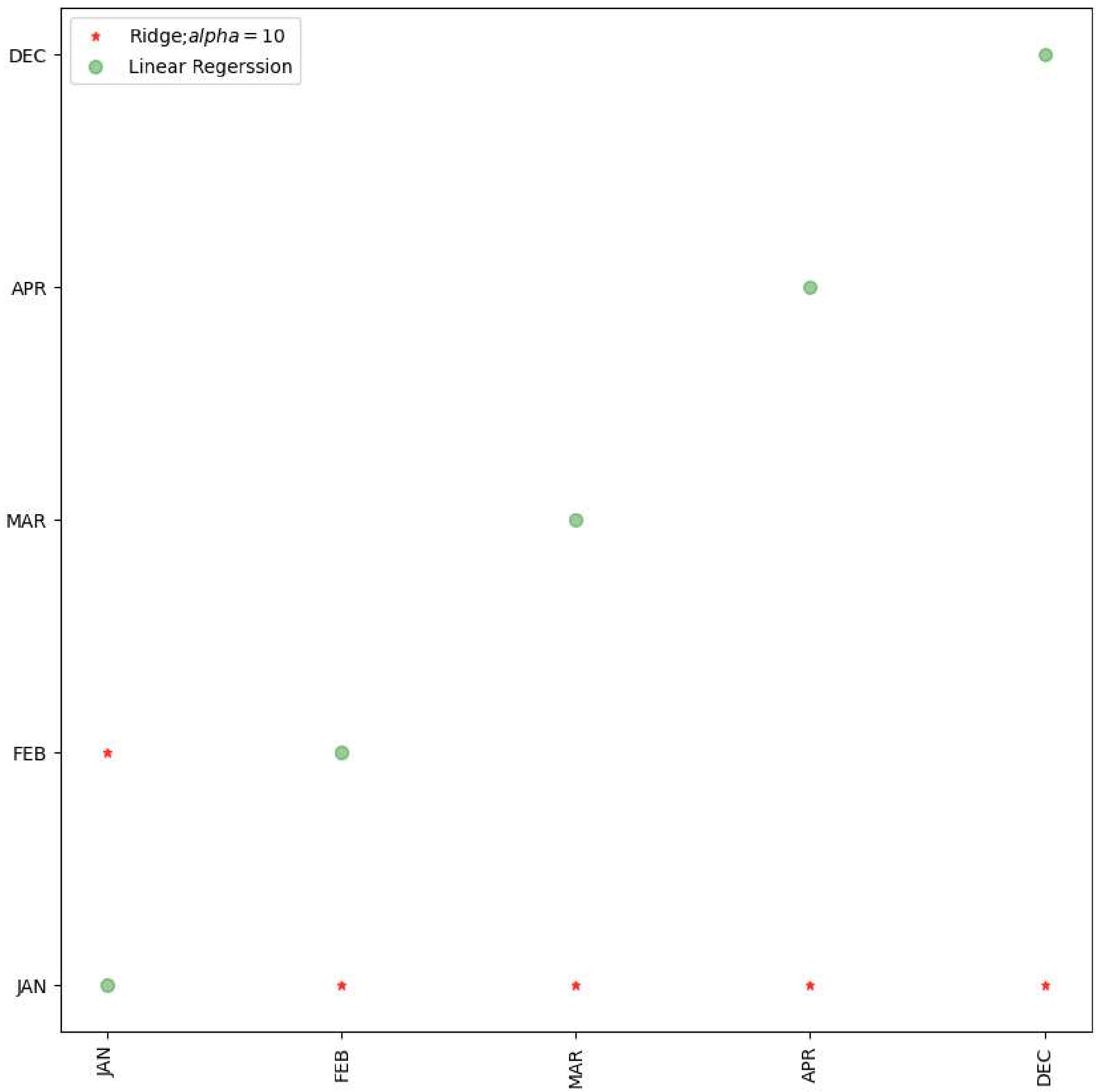
```
 Ridge Model:

the train score for ridge model is0.9999999999874308
the test score for ridge model is0.9999999999883638
```

```
#conclusion:
the train score for ridge model is0.9999999999874192
the test score for ridge model is0.99999999998833
```

In [38]:
```
lr=LinearRegression()
```

In [41]:
```
ze= (10,10))
s,ridgeReg.coef_,alpha=0.7,linestyle='none',marker="*",markersize=5,color="red",label=r'Ridge;$alpha=10$',zorder=7)
s,alpha=0.4,linestyle='none',marker='o',markersize=7,color="green",label='Linear Regerssion')
ion = 90)
```



# 3.Lasso Regression

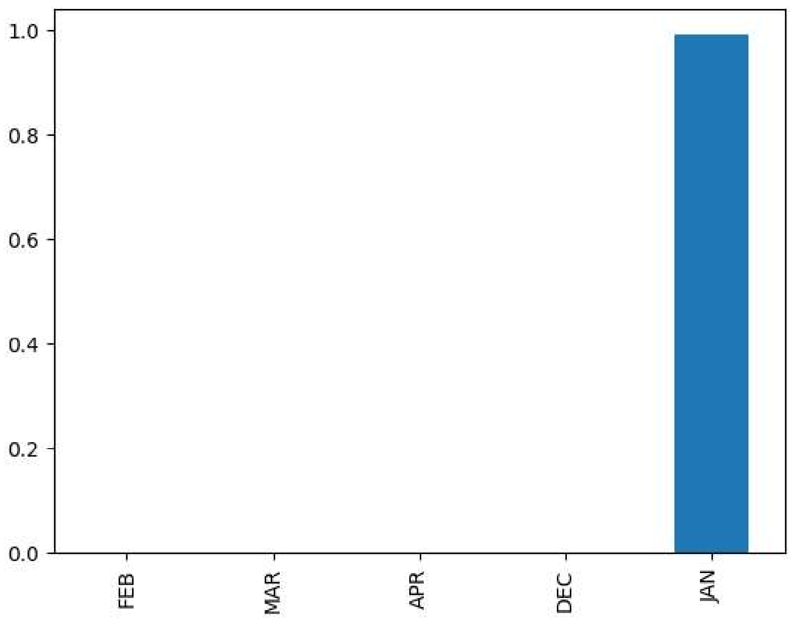
In [43]:
```
print("\n Lasso Model:\n")
lasso=Lasso(alpha=10)
lasso.fit(x_train,y_train)
train_score_ls=lasso.score(x_train,y_train)
test_score_ls=lasso.score(x_test,y_test)
print("The train score for ls model is {}".format(train_score_ls))
print("The test score for ls model is{}".format(test_score_ls))
```

```
 Lasso Model:

The train score for ls model is 0.9999207503194595
The test score for ls model is0.9999206588980594
```

In [44]:
```
pd.Series(lasso.coef_,features).sort_values(ascending=True).plot(kind="bar")
```

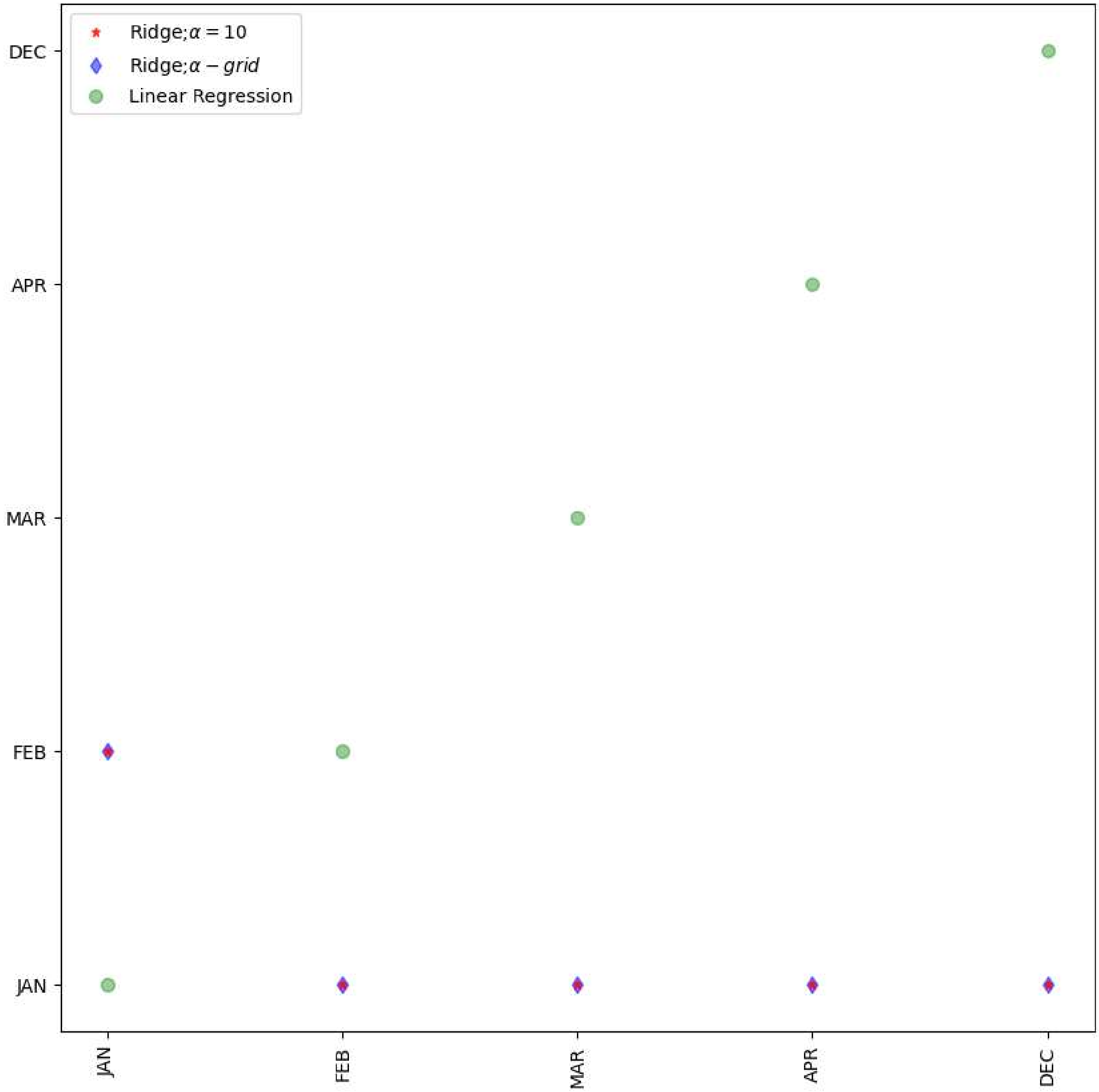
Out[44]: <Axes: >




In [45]:
```
from sklearn.linear_model import LassoCV
lasso_cv=LassoCV(alphas=[0.0001,0.001,0.01,1,10],random_state=0).fit(x_train,y_train)
print(lasso_cv.score(x_train,y_train))
print(lasso_cv.score(x_test,y_test))
```

```
0.9999999999999921
0.9999999999999921
```

In [47]:
```
e= (10,10))
,ridgeReg.coef_,alpha=0.7,linestyle='none',marker="*",markersize=5,color='red',label=r'Ridge;$\alpha=10$',zorder=7)
.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',label=r'Ridge;$\alpha-grid$')
,alpha=0.4,linestyle='none',marker='o',markersize=7,color="green",label='Linear Regression')
on = 90)
```




## 4.Elastic Net

In [48]:
```
from sklearn.linear_model import ElasticNet
eln=ElasticNet()
eln.fit(x,y)
print(eln.coef_)
print(eln.intercept_)
print(eln.score(x,y))
```

```
[9.99098882e-01 0.00000000e+00 2.95025317e-05 0.00000000e+00
 0.00000000e+00]
0.016260183535354855
0.9999992158680019
```

In [50]:
```
y_pred_elastic = eln.predict(x_train)
mean_squared_error=np.mean((y_pred_elastic - y_train)**2)
print(mean_squared_error)
```

```
0.0008817707583048099
```

## Conclusion

```
THE SCORE OF LINEAR REGRESSION IS :- 0.1793580786264921
THE SCORE OF RIDGE MODEL IS :- 0.99999999998833
THE SCORE OF LASSO MODEL IS :- 0.999999999999992
THE SCORE OF ELASTIC NET IS :- 0.9999992160905338
 *AMONG ALL MODELS LASSO YEILD HIGHEST ACCURACY.SO,WE PREFER LASSO MODEL FOR THIS DATA SET*
```

In [ ]: