PROJECT ON RAINFALL DATASET

1.PROBLEM STATEMENT:- TO PREDICT RAINFALL BASED ON VARIOUS FEATURES OF THE DATASET

In [1]:
```
#IMPORT LIBERIES
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn import preprocessing,svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn import metrics
from sklearn.linear_model import Lasso,LassoCV
from sklearn.linear_model import Ridge,RidgeCV
from sklearn.preprocessing import StandardScaler
```

DATA COLLECTION

READ THE DATA

In [2]:
```
pf=pd.read_csv(r"C:\Users\HP\Downloads\district wise rainfall normal.csv")
pf
```

Out[2]:

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	A
0	ANDAMAN And NICOBAR ISLANDS	NICOBAR	107.3	57.9	65.2	117.0	358.5	295.5	285.0	271.9	354.8	326.0	315.2	250.9	
1	ANDAMAN And NICOBAR ISLANDS	SOUTH ANDAMAN	43.7	26.0	18.6	90.5	374.4	457.2	421.3	423.1	455.6	301.2	275.8	128.3	
2	ANDAMAN And NICOBAR ISLANDS	N & M ANDAMAN	32.7	15.9	8.6	53.4	343.6	503.3	465.4	460.9	454.8	276.1	198.6	100.0	
3	ARUNACHAL PRADESH	LOHIT	42.2	80.8	176.4	358.5	306.4	447.0	660.1	427.8	313.6	167.1	34.1	29.8	
4	ARUNACHAL PRADESH	EAST SIANG	33.3	79.5	105.9	216.5	323.0	738.3	990.9	711.2	568.0	206.9	29.5	31.7	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
636	KERALA	IDUKKI	13.4	22.1	43.6	150.4	232.6	651.6	788.9	527.3	308.4	343.2	172.9	48.1	
637	KERALA	KASARGOD	2.3	1.0	8.4	46.9	217.6	999.6	1108.5	636.3	263.1	234.9	84.6	18.4	
638	KERALA	PATHANAMTHITTA	19.8	45.2	73.9	184.9	294.7	556.9	539.9	352.7	266.2	359.4	213.5	51.3	
639	KERALA	WAYANAD	4.8	8.3	17.5	83.3	174.6	698.1	1110.4	592.9	230.7	213.1	93.6	25.8	
640	LAKSHADWEEP	LAKSHADWEEP	20.8	14.7	11.8	48.9	171.7	330.2	287.7	217.5	163.1	157.1	117.7	58.8	

641 rows × 19 columns

In [3]:
```
df=pd.read_csv(r"C:\Users\HP\Downloads\rainfall in india 1901-2015.csv")
df
```

Out[3]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb
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
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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

4116 rows × 19 columns

DATA CLEANING AND PREPOCESSING

In [4]:
```
df.columns
```

Out[4]:
```
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 [5]:
```
df.describe()
```

Out[5]:

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

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

Out[6]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb
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
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
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
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
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

In [7]:
```
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 [8]:
```
df.head()
```

Out[8]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Ma Ma
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
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
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
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
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

In [9]:
```
df.shape
```

Out[9]:
```
(4116, 19)
```

In [10]:
```
df.duplicated().sum()                #
```

Out[10]:
```
0
```

In [11]:
```
df['SUBDIVISION'].unique()
```

Out[11]:
```
array(['ANDAMAN & NICOBAR ISLANDS', 'ARUNACHAL PRADESH',
       'ASSAM & MEGHALAYA', 'NAGA MANI MIZO TRIPURA',
       'SUB HIMALAYAN WEST BENGAL & SIKKIM', 'GANGETIC WEST BENGAL',
       'ORISSA', 'JHARKHAND', 'BIHAR', 'EAST UTTAR PRADESH',
       'WEST UTTAR PRADESH', 'UTTARAKHAND', 'HARYANA DELHI & CHANDIGARH',
       'PUNJAB', 'HIMACHAL PRADESH', 'JAMMU & KASHMIR', 'WEST RAJASTHAN',
       'EAST RAJASTHAN', 'WEST MADHYA PRADESH', 'EAST MADHYA PRADESH',
       'GUJARAT REGION', 'SAURASHTRA & KUTCH', 'KONKAN & GOA',
       'MADHYA MAHARASHTRA', 'MATATHWADA', 'VIDARBHA', 'CHHATTISGARH',
       'COASTAL ANDHRA PRADESH', 'TELANGANA', 'RAYALSEEMA', 'TAMIL NADU',
       'COASTAL KARNATAKA', 'NORTH INTERIOR KARNATAKA',
       'SOUTH INTERIOR KARNATAKA', 'KERALA', 'LAKSHADWEEP'], dtype=object)
```

In [12]:
```
df['YEAR'].value_counts()
```

Out[12]:
```
YEAR
1963    36
2002    36
1976    36
1975    36
1974    36
        ..
1915    35
1918    35
1954    35
1955    35
1909    34
Name: count, Length: 115, dtype: int64
```

In [13]:
```
df.isnull().any()
```

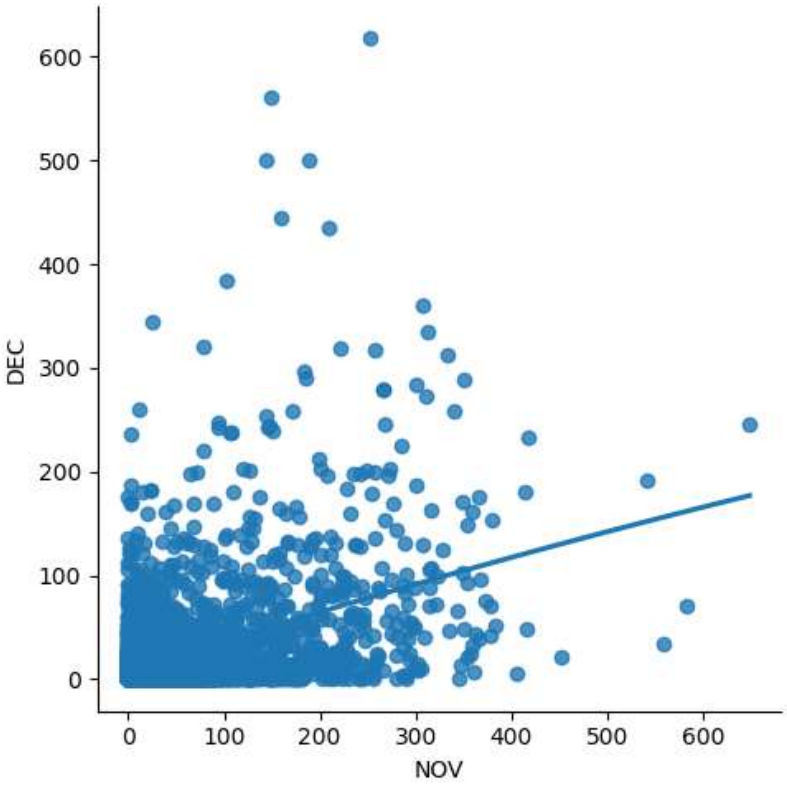
Out[13]:
```
SUBDIVISION    False
YEAR           False
JAN             True
FEB             True
MAR             True
APR             True
MAY             True
JUN             True
JUL             True
AUG             True
SEP             True
OCT             True
NOV             True
DEC             True
ANNUAL          True
Jan-Feb         True
Mar-May         True
Jun-Sep         True
Oct-Dec         True
dtype: bool
```

HERE THERE IS NULL VALUES.SO THIS IS NECESSARY TO ADJUST THE VALUES

In [14]:
```
df.fillna(method="ffill",inplace=True)
```

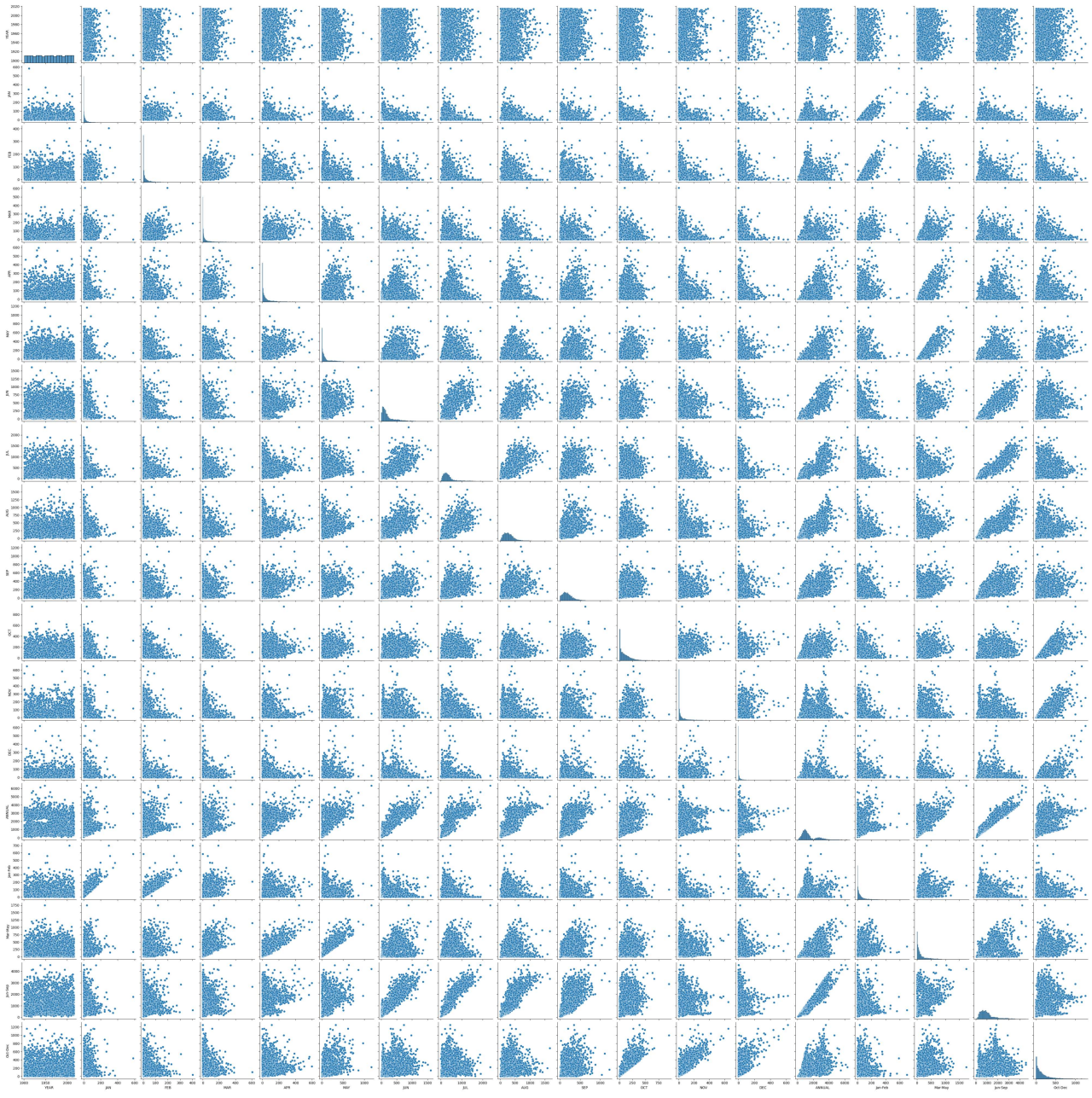
DATA VISUALIZATON

In [15]:
```
sns.lmplot(x='NOV',y='DEC',order=2,data=df,ci=None)
plt.show()
```



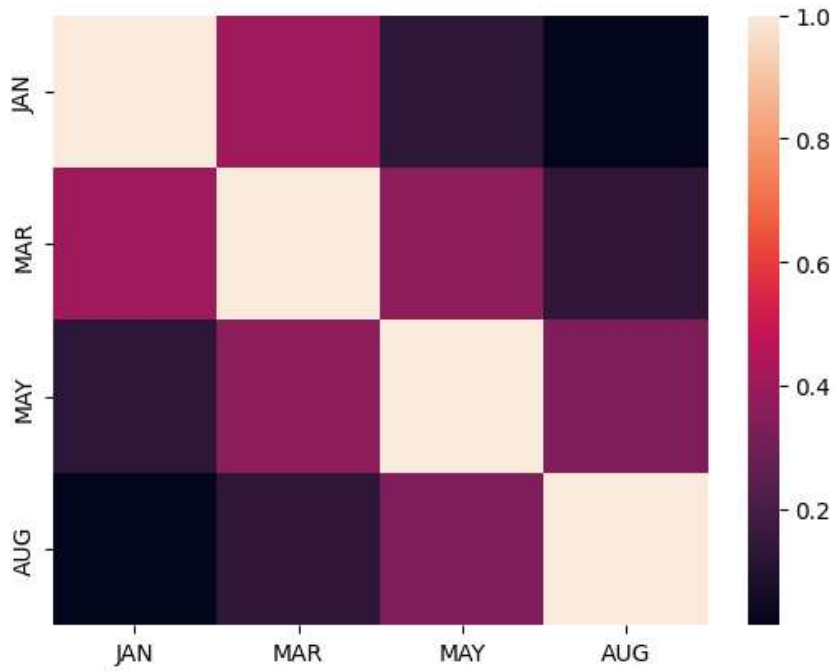
In [18]:
```
sns.pairplot(df)
```

Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x23773b0b0a0>
```



In [19]:
```
Rainfalldf=df[['JAN','MAR', 'MAY','AUG']]
sns.heatmap(Rainfalldf.corr())
```

Out[19]: <Axes: >



Feature Scaling:spliting data into train and test data sets

In [20]:
```
x=np.array(df['JAN']).reshape(-1,1)
y=np.array(df['FEB']).reshape(-1,1)
```

In [21]:
```
from sklearn.linear_model import Ridge,RidgeCV,Lasso
from sklearn.preprocessing import StandardScaler
```

DATA MODELING:

HERE THERE IS CONTINOUS DATASET.SO,WE THROUGH REGRESSION models

In [22]:
```
#1.TO MAKE CONVERSIONS TO HAVE SAME DATA TYPE TO FIND REGRESSION MODEL
```

In [23]:
```
A={"SUBDIVISION":{'ANDAMAN & NICOBAR ISLANDS':0, 'ARUNACHAL PRADESH':1,'ASSAM & MEGHALAYA':3,
        'NAGA MANI MIZO TRIPURA':4,'SUB HIMALAYAN WEST BENGAL & SIKKIM':5, 'GANGETIC WEST BENGAL':6,
       'ORISSA':7, 'JHARKHAND':8, 'BIHAR':9, 'EAST UTTAR PRADESH':10,
       'WEST UTTAR PRADESH':11, 'UTTARAKHAND':12, 'HARYANA DELHI & CHANDIGARH':13,
       'PUNJAB':14, 'HIMACHAL PRADESH':15, 'JAMMU & KASHMIR':16, 'WEST RAJASTHAN':17,
       'EAST RAJASTHAN':18, 'WEST MADHYA PRADESH':19, 'EAST MADHYA PRADESH':20,
       'GUJARAT REGION':21, 'SAURASHTRA & KUTCH':22, 'KONKAN & GOA':23,
       'MADHYA MAHARASHTRA':24, 'MATATHWADA':25, 'VIDARBHA':26, 'CHHATTISGARH':27,
       'COASTAL ANDHRA PRADESH':28, 'TELANGANA':29, 'RAYALSEEMA':30, 'TAMIL NADU':31,
       'COASTAL KARNATAKA':32, 'NORTH INTERIOR KARNATAKA':33,
       'SOUTH INTERIOR KARNATAKA':34, 'KERALA':35, 'LAKSHADWEEP':36}}
df=df.replace(A)
df
```

Out[23]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb
0	0	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
1	0	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
2	0	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
3	0	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
4	0	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4111	36	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
4112	36	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
4113	36	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
4114	36	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
4115	36	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

4116 rows × 19 columns

In [24]:
```
#2.TO FIND DIMENSIONS OF X and y
features = df.columns[0:2]
target = df.columns[-1]
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)
print("The dimension of X_train is {}".format(X_train.shape))
print("The dimension of X_test is {}".format(X_test.shape))
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
The dimension of X_train is (2881, 2)
The dimension of X_test is (1235, 2)
```

In [25]:
```
#3.TO FIND SCORE OF LinearRegression()
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.03,random_state=42)
lir=LinearRegression()
lir.fit(X_train,y_train)
print(lir.score(X_test,y_test))
```

```
0.032476925396737744
```

In [26]:
```
#4.TO FIND TRAIN AND TEST SCORE OF LinearRegression()
train_score_lir = lir.score(X_train, y_train)
test_score_lir = lir.score(X_test, y_test)
print("\nLinear Regression Model:\n")
print("The train score for lir model is {}".format(train_score_lir))
print("The test score for lir model is {}".format(test_score_lir))
```

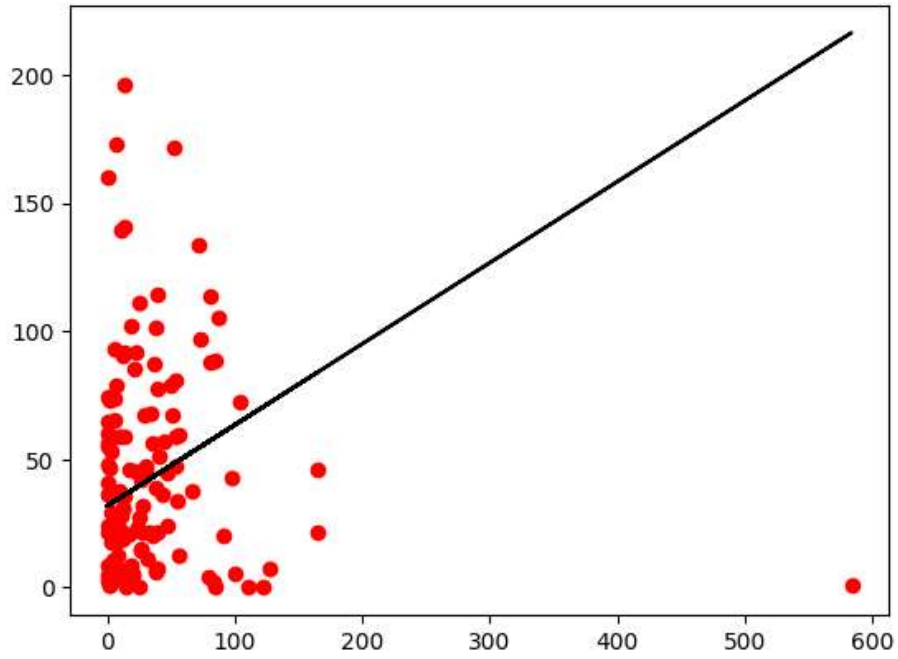
```
Linear Regression Model:

The train score for lir model is 0.007658721435404692
The test score for lir model is 0.032476925396737744
```

In the case of smaller dataset Taken(*not necessary)

In [27]:
```
df500=df[:][:500]
```

In [28]:
```
df500.fillna(method='ffill',inplace=True)
X=np.array(df500['JAN']).reshape(-1,1)
y=np.array(df500['FEB']).reshape(-1,1)
df500.dropna(inplace=True)
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)
regr=LinearRegression()
regr.fit(X_train,y_train)
print("Regression:",regr.score(X_test,y_test))
y_pred=regr.predict(X_test)
plt.scatter(X_test,y_test,color='r')
plt.plot(X_test,y_pred,color='k')
plt.show()
```

```
Regression: -0.28639027858287913
```



RIDGE REGRESSION

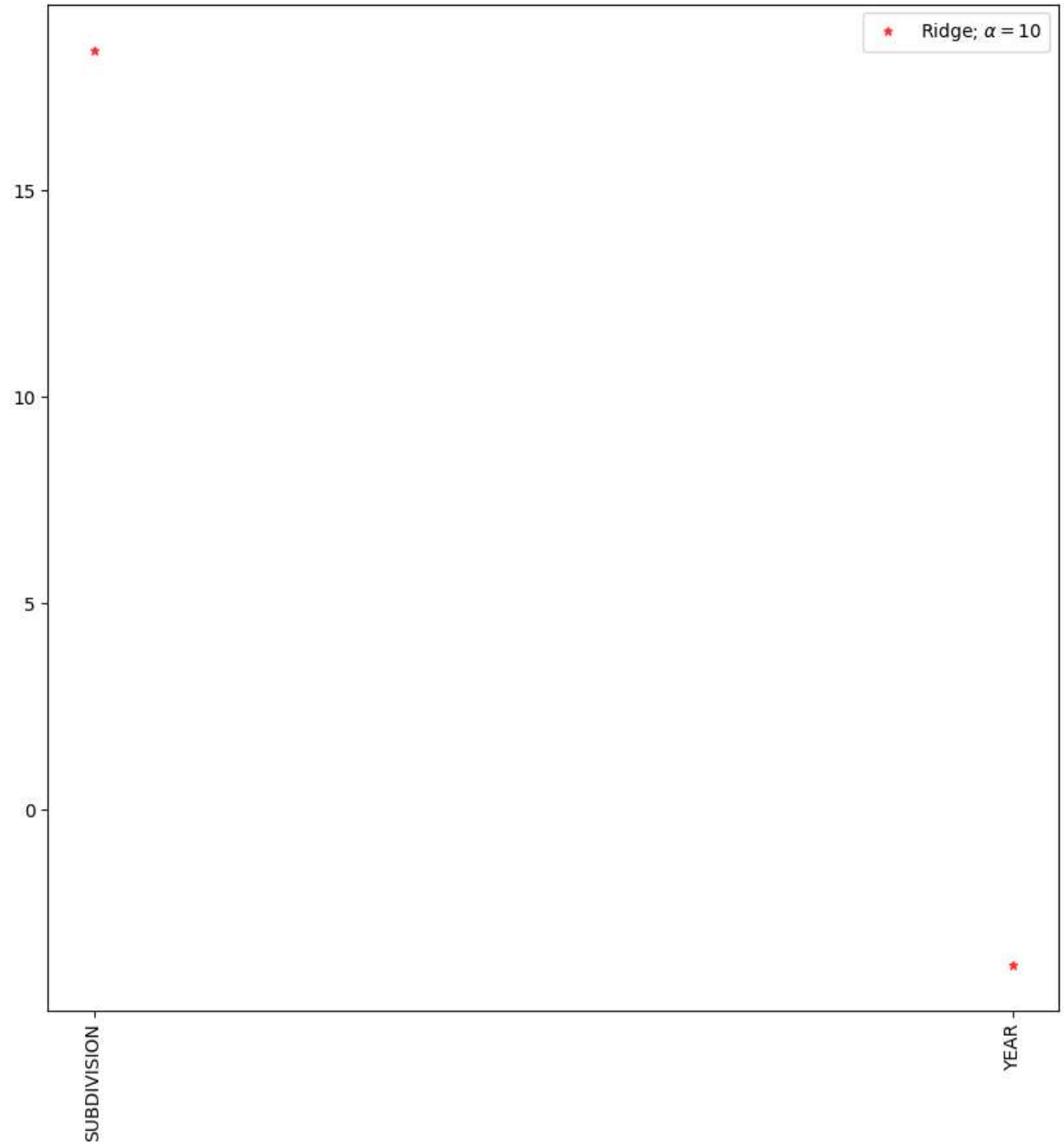
In [29]:
```
#2.TO FIND DIMENSIONS OF X and y
features = df.columns[0:2]
target = df.columns[-1]
#X and y values
x= df[features].values
y = df[target].values
#splot
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=17)
print("The dimension of x_train is {}".format(X_train.shape))
print("The dimension of x_test is {}".format(X_test.shape))
#Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
The dimension of x_train is (2881, 2)
The dimension of x_test is (1235, 2)
```

In [30]:
```
ridgeReg = Ridge(alpha=10)
ridgeReg.fit(X_train,y_train)
print(ridgeReg.score(X_test, y_test))
train_score_ridge = ridgeReg.score(X_train, y_train)
test_score_ridge = ridgeReg.score(X_test, y_test)
print("\nRidge 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))
```

```
0.008372705365307542

Ridge Model:

The train score for ridge model is 0.012965342189359408
The test score for ridge model is 0.008372705365307542
```

In [31]:
```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=
#plt.
#plt.plot(features,lir.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='green',label='
plt.xticks(rotation = 90)
plt.legend()
plt.show()
```



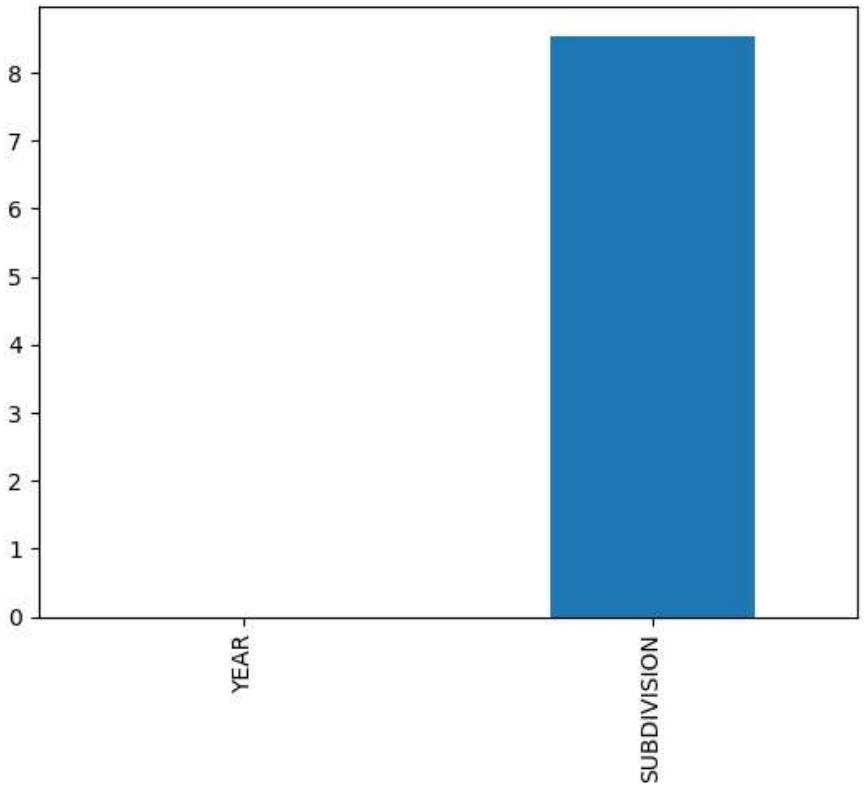
LASSO REGRESSION

In [32]:
```
lr=Lasso(alpha=10)
lr.fit(X_train,y_train)
print(lr.score(X_test,y_test))
```

```
0.006853591543563464
```

In [33]:
```
pd.Series(lr.coef_, features).sort_values(ascending = True).plot(kind = "bar")
```

Out[33]: <Axes: >



In [34]:
```
print("\nLasso Model: \n")
lr = Lasso(alpha = 10)
lr.fit(X_train,y_train)
train_score_ls =lr.score(X_train,y_train)
test_score_ls =lr.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))
```

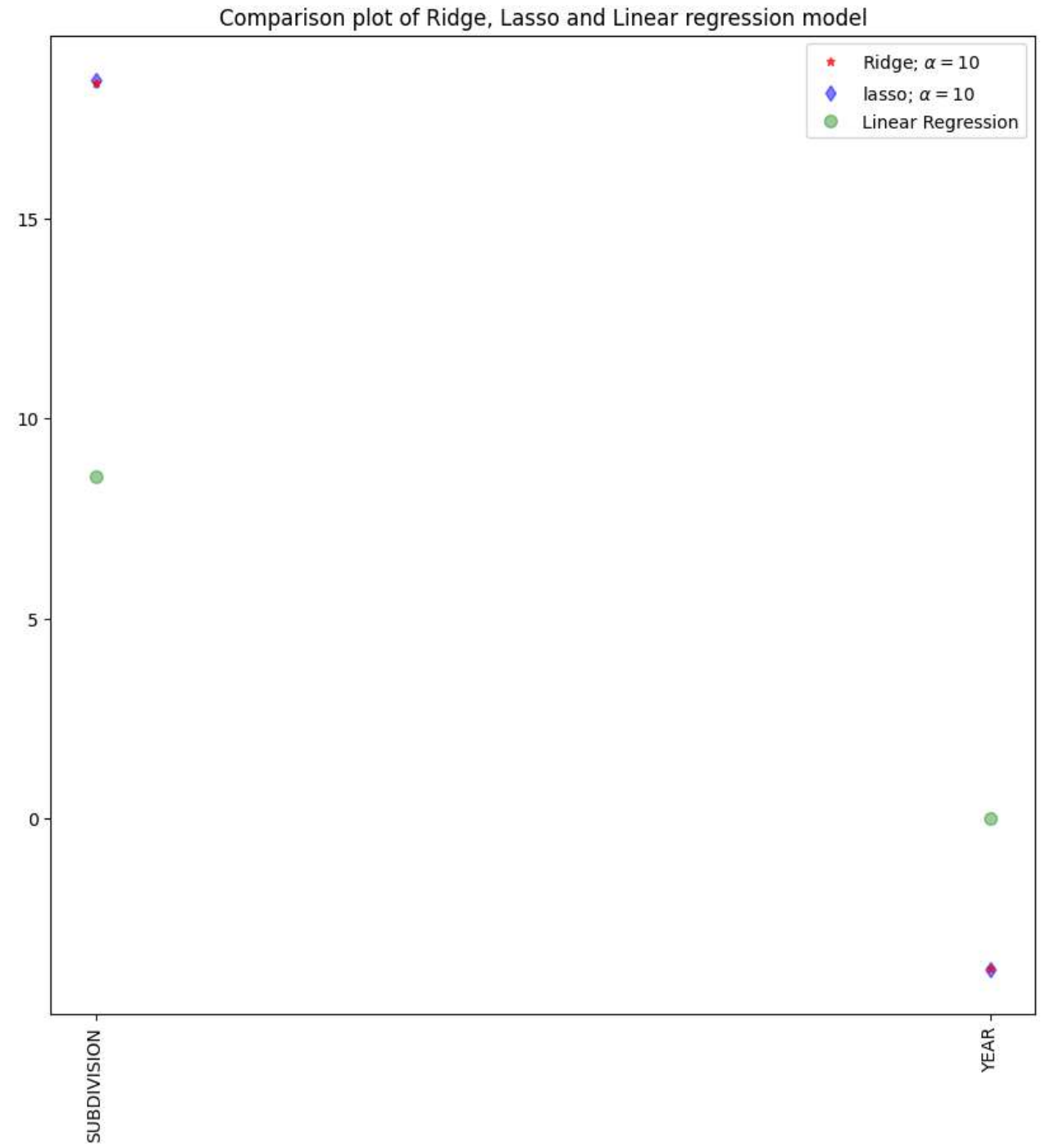
```
Lasso Model:

The train score for ls model is 0.008833354900108392
The test score for ls model is 0.006853591543563464
```

In [35]:
```
from sklearn.linear_model import LassoCV
#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.0001, 0.001,0.01, 0.1, 1, 10], random_state=0).fit(X_train, y_train)
#score
print(lasso_cv.score(X_train, y_train))
print(lasso_cv.score(X_test, y_test))
```

```
0.012965494708651604
0.008362091594152132
```

COMPARISON BETWEEN REGRESSION MODELS

In [36]:
```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=
plt.plot(lasso_cv.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',label=r'lasso
plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='green',label='Li
plt.xticks(rotation = 90)
plt.legend()
plt.title("Comparison plot of Ridge, Lasso and Linear regression model")
plt.show()
```



ELASTIC NET

In [37]:
```
from sklearn.linear_model import ElasticNet
regr=ElasticNet()
regr.fit(x,y)
print(regr.coef_)
print(regr.intercept_)
regr.score(x,y)
```

```
[ 1.7253096  -0.06496651]
250.19040895123723
```

Out[37]:
```
0.011852419739363684
```

In [40]:
```
y_pred_elastic=regr.predict(X_train)
```

In [41]:
```
prediction=lr.predict(X_test)
```

In [42]:
```
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: 0.008362046064003259
```

To find Error

In [43]:
```
print('MAE:',metrics.mean_absolute_error(y_test,prediction))
print('MSE:',metrics.mean_squared_error(y_test,prediction))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE: 122.89171816248202
MSE: 29157.27810826937
RMSE: 170.75502366920094
```

Cross Validation

In [44]:
```
ridge_cv=RidgeCV(alphas=[1,10,100]).fit(X_train,y_train)
print("The train score for ridgecrossvalidation model is : {}".format(ridge_cv.score(X_train,y_train)
print("The test score for ridgcrossvalidation model is :{}".format(ridge_cv.score(X_test,y_test)))
```

```
The train score for ridgecrossvalidation model is : 0.01295114297450084
The test score for ridgcrossvalidation model is :0.008453215142374071
```

In [45]:
```
lasso_cv=LassoCV(alphas=[1,10,100]).fit(X_train,y_train)
print(" The train score for lassocrossvalidation model is : {}".format(lasso_cv.score(X_train,y_train
print("The test score for lassocrossvalidation model is:{}".format(lasso_cv.score(X_test,y_test)))
```

```
 The train score for lassocrossvalidation model is : 0.012894689534923565
The test score for lassocrossvalidation model is:0.00875276826027771
```

CONCLUSION:

In [ ]:
```
From above results,it is concluded that linear regression has best suit to the given rainfall dataset
```