RAINFALL PREDICTION IN INDIA
PROJECT REPORT

Team Members :

Name	Roll Number
Kode Dhanya	AM.EN.U4CSE19129
Pulivendula Reddy Gowtham	AM.EN.U4CSE19143
K C Sekhar Madan	AM.EN.U4CSE19167
Thumati Sai Manichandana Devi	AM.EN.U4CSE19169

Problem Definition

Climate is an important aspect of human life. In this project we are trying to deal with the prediction of rainfall which is a major aspect of human life and which provides the major resource of human life that is Fresh Water.

- ➔ **Task (T) :** To predict the rainfall in India
- ➔ **Experience (E) :** Run it through a machine learning algorithm with data about past rainfall patterns.
- ➔ **Performance Measure(P)** : The percentage of accuracy predicted by the model.

Dataset finalization

Dataset -1:

Link: Rainfall_in_india_1901-2015
Link: District_wise_rainfall_normal

Content:

Time Period: 1901 - 2015

Granularity: Monthly

Locations: district wide

Rainfall unit: mm

Dataset -2:

Link: Sub_Division_IMD_2017

Content

Time Period: 1901 - 2017

Granularity: Monthly

Location: 36 meteorological sub-divisions in India

Rainfall unit: mm

Dataset -3:

Link: India_monthly_rainfall_data

Content:

In this dataset, we are given monthly rainfall of cities of states of India.

Time Period: 1901 - 2017

Granularity: Monthly

Location: state wide

Rainfall unit: mm

Dataset -4:

Link: Kerala-Rainfall-Historical

Content:

In this dataset, we are given monthly rainfall of cities and states of India.

Time Period: 1901 - 2017

Granularity: Monthly

Location: Kerala

Rainfall unit: mm

Features in the datasets:

There are 4 attributes in the data set:

- Subdivisions(state,districts)

- Year
- Months(jan -dec)
- Annual

- **Subdivisions:**

In this attribute we have states and districts, which helps us anticipate rainfall in a specific state or district. The datatype of this attribute is string.

- **Year :**

  This feature provides the years, which allows us to forecast rainfall based on previous years data.The data type is integer.

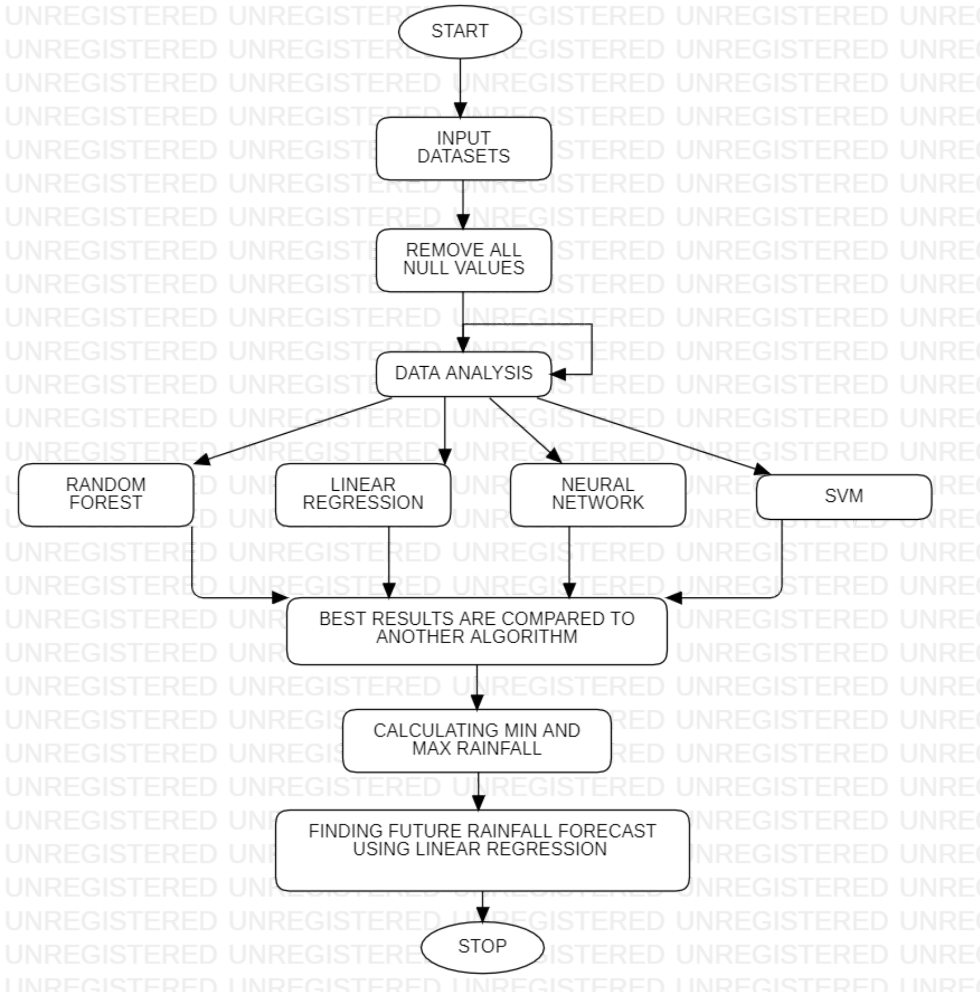
- **Months:**

  This feature has months (from January to December), and the data recorded in it is the monthly rainfall data.The data type was float

- **Annual:**

  This feature provides information about the annual rainfall in that specific year.The data type was float

Prepare Data



Data Exploration and Pre Processing:

Data Exploration or Exploratory data analysis (EDA) provides a simple set of exploration tools that bring out the basic understanding of real-time data into data analytics. Data exploration can use a combination of manual methods and automated tools, such as data visualization, charts, and preliminary reports. The primary purpose of data pre-processing is to modify the input variables so they can better match the predicted output

Summarization:

```
In [2]: #data collection
        data = pd.read_csv("district wise rainfall normal.csv")
        data.head()
```

Out[2]:

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
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	2805.2	165.2	540.7	1207.2	892.1
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	3015.7	69.7	483.5	1757.2	705.3
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	2913.3	48.6	405.6	1884.4	574.7
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	3043.8	123.0	841.3	1848.5	231.0
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	4034.7	112.8	645.4	3008.4	268.1

Dataset used :[Dataset -1]

- *dataframe.info()* function is used to get a concise summary of the dataframe

```
In [3]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 641 entries, 0 to 640
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   STATE_UT_NAME  641 non-null    object
 1   DISTRICT       641 non-null    object
 2   JAN            641 non-null    float64
 3   FEB            641 non-null    float64
 4   MAR            641 non-null    float64
 5   APR            641 non-null    float64
 6   MAY            641 non-null    float64
 7   JUN            641 non-null    float64
 8   JUL            641 non-null    float64
 9   AUG            641 non-null    float64
 10  SEP            641 non-null    float64
 11  OCT            641 non-null    float64
 12  NOV            641 non-null    float64
 13  DEC            641 non-null    float64
 14  ANNUAL         641 non-null    float64
 15  Jan-Feb        641 non-null    float64
 16  Mar-May        641 non-null    float64
 17  Jun-Sep        641 non-null    float64
 18  Oct-Dec        641 non-null    float64
dtypes: float64(17), object(2)
memory usage: 95.3+ KB
```

- *Dataframe.describe()* gives statistical summary of all attributes.

In [12]:
```
data.describe()
```
Out[12]:

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	
count	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	6
mean	18.355070	20.984399	30.034789	45.543214	81.535101	196.007332	326.033697	291.152262	194.609048	90.446334	34.117473	18.150858	13
std	21.082806	27.729596	45.451082	71.556279	111.960390	196.556284	221.364643	152.647325	99.830540	74.990685	59.371274	32.711009	8
min	0.000000	0.000000	0.000000	0.000000	0.900000	3.800000	11.600000	14.100000	8.600000	3.100000	1.200000	0.000000	
25%	6.900000	7.000000	7.000000	5.000000	12.100000	68.800000	206.400000	194.600000	128.800000	34.300000	6.600000	5.300000	8
50%	13.300000	12.300000	12.700000	15.100000	33.900000	131.900000	293.700000	284.800000	181.300000	62.600000	12.900000	7.900000	11
75%	19.200000	24.100000	33.200000	48.300000	91.900000	226.600000	374.800000	358.100000	234.100000	130.200000	32.300000	14.900000	15
max	144.500000	229.600000	367.900000	554.400000	733.700000	1476.200000	1820.900000	1522.100000	826.300000	517.700000	475.100000	297.700000	72

- *dataframe.isnull()* detect missing values which are NA i.e null values

```
data.isnull().sum()
```

```
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
JF              6
MAM             9
JJAS           10
OND            13
dtype: int64
```

- ***dataframe.duplicated()*** method helps in analyzing duplicate values only. It returns a boolean series which is True only for Unique elements.
- *Dataframe.value_counts()* function return a Series containing counts of unique values

```
In [5]: data.duplicated().sum()

Out[5]: 0

In [6]: data['STATE_UT_NAME'].value_counts()

Out[6]: UTTAR PRADESH          71
        MADHYA PRADESH         50
        BIHAR                  38
        MAHARASHTRA            35
        RAJASTHAN              33
        TAMIL NADU             32
        KARNATAKA              30
        ORISSA                 30
        ASSAM                  27
        GUJARAT                26
        JHARKHAND              24
        ANDHRA PRADESH         23
        JAMMU AND KASHMIR      22
        HARYANA                21
        PUNJAB                 20
        WEST BENGAL            19
        CHATISGARH             18
        ARUNACHAL PRADESH      16
        KERALA                 14
        UTTARANCHAL            13
        HIMACHAL               12
        NAGALAND               11
        MIZORAM                 9
        DELHI                   9
        MANIPUR                 9
        MEGHALAYA               7
        SIKKIM                  4
        TRIPURA                 4
        PONDICHERRY             4
```

- The null values present in the dataset can be replaced by finding the mean of each attribute.

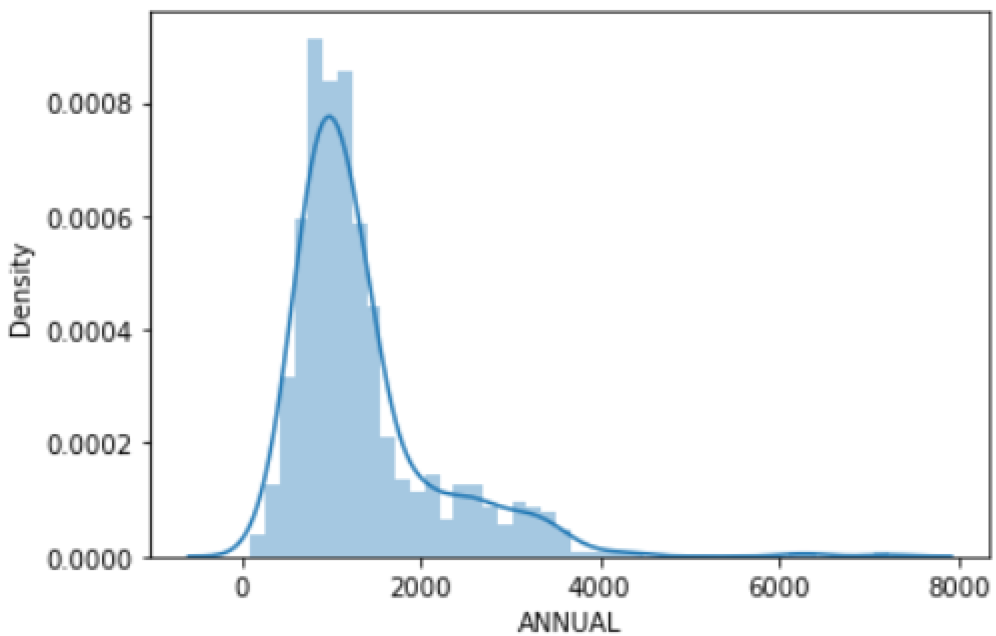
```
In [4]: # date preprocessing
        data.isnull().sum()

Out[4]: STATE_UT_NAME    0
        DISTRICT         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
```

Data Visualization:

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions.

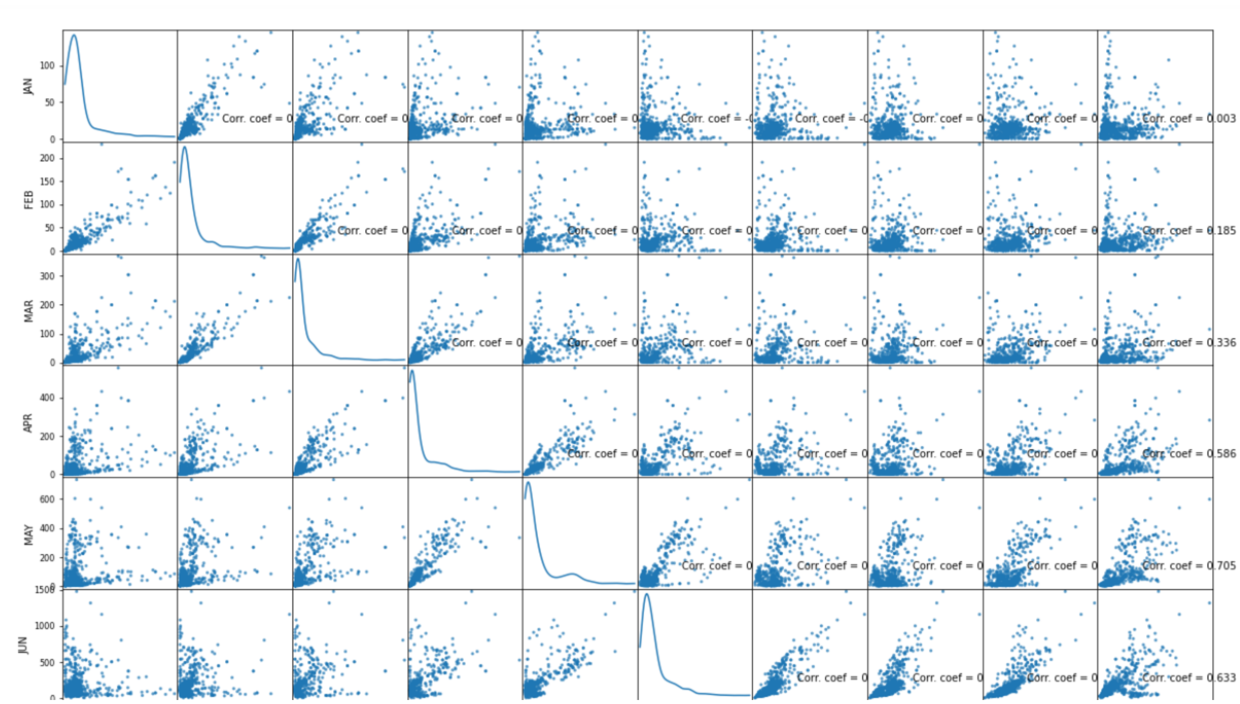
- *Displot* is used basically for a univariant set of observations and visualizes it through a histogram, i.e. only one observation and hence we choose one particular column of the dataset.
  - Here we used an annual attribute to visualize

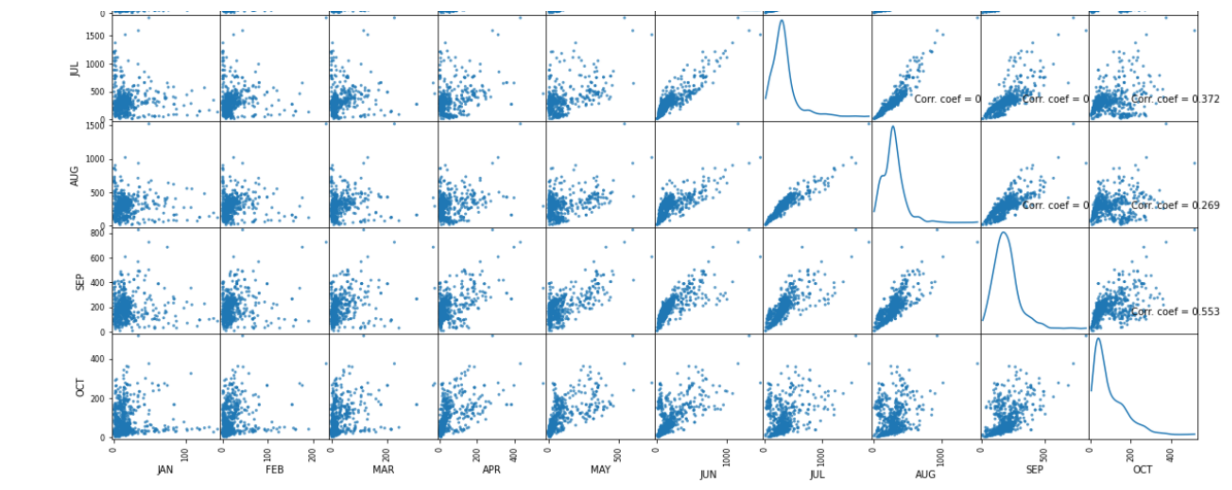


- **Scatter and density plots**
  - A scatter plot is a diagram where each value in the data set is represented by a dot.
  - *Scatter plot of all the attributes*

```
In [15]: # Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center', va='center', size
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

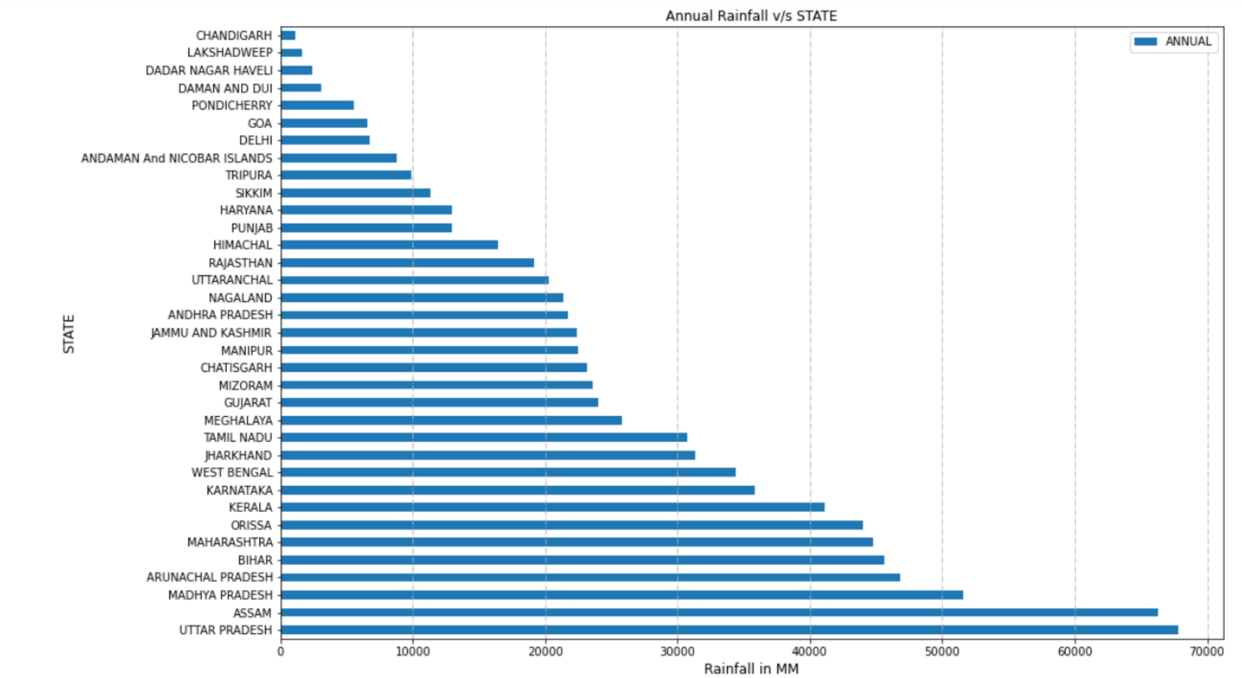
  -

- 

- 

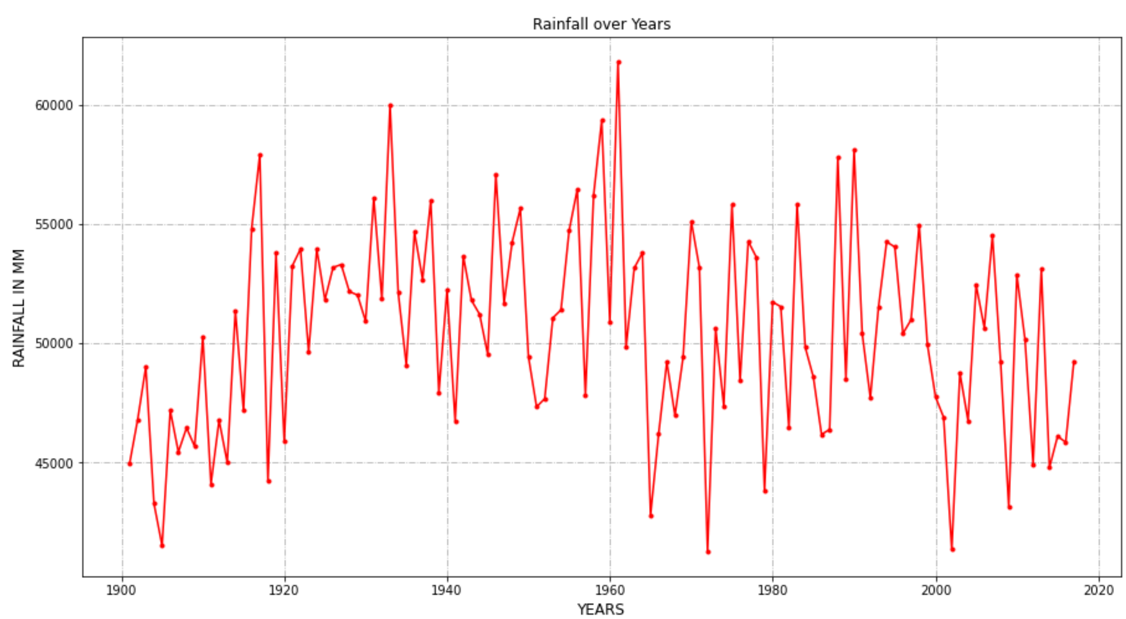
- The below graph shows the distribution of rainfall over states.
  - The graph clearly shows that the amount of rainfall is high in karnataka, goa, kerala

```
In [17]: # data visulization
data[["STATE_UT_NAME","ANNUAL"]].groupby("STATE_UT_NAME").sum().sort_values(by='ANNUAL',ascending=False).plot(kind='barh',stacked
plt.xlabel("Rainfall in MM",size=12)
plt.ylabel("STATE",size=12)
plt.title("Annual Rainfall v/s STATE")
plt.grid(axis="x",linestyle="-.")
plt.show()
```



- The below graph shows the distribution of rainfall over years.
  - Observed high amount of rainfall in 1961

```
plt.figure(figsize=(15,8))
data.groupby("YEAR").sum()['ANNUAL'].plot(kind="line",color="r",marker=".")
plt.xlabel("YEARS",size=12)
plt.ylabel("RAINFALL IN MM",size=12)
plt.grid(axis="both",linestyle="-.")
plt.title("Rainfall over Years")
plt.show()
```

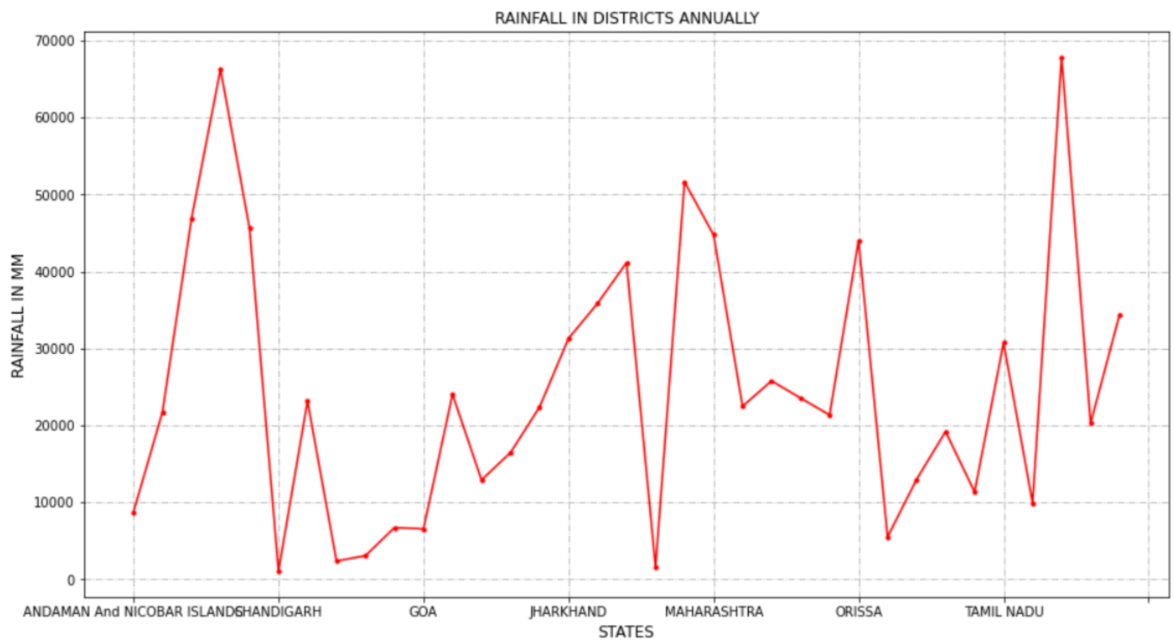


- In the above graph we have observed year wise rainfall, now let us look monthwise in those years

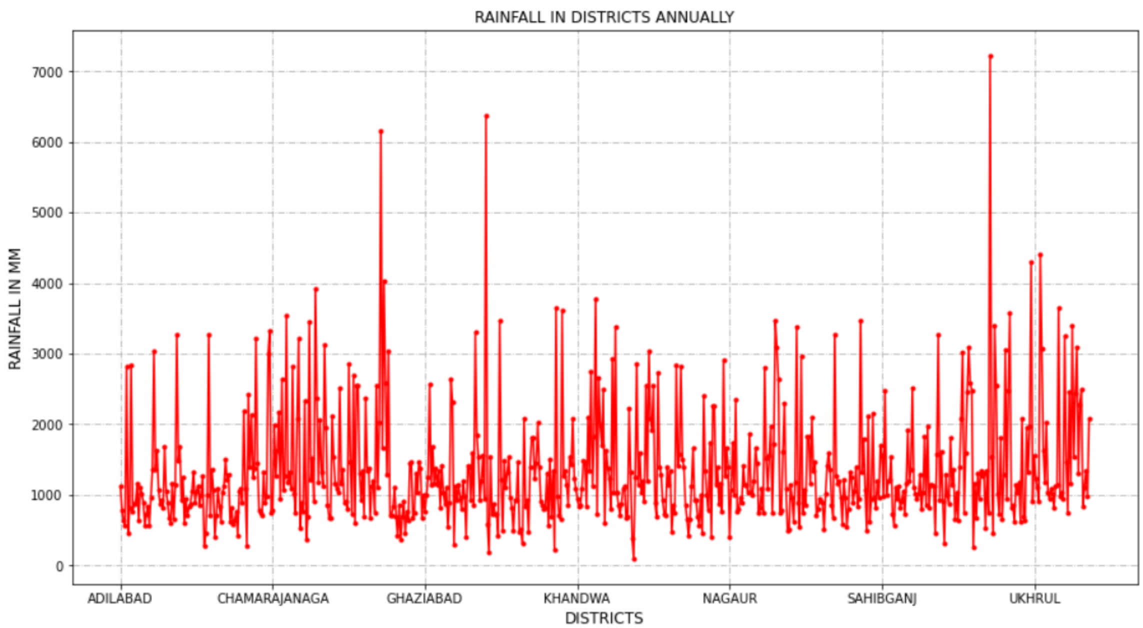
- The below graphs clearly show that the amount of rainfall is high in the months of July, aug, sep which is monsoon season in India.

-

In [18]:
```
plt.figure(figsize=(15,8))
data.groupby("STATE_UT_NAME").sum()['ANNUAL'].plot(kind="line",color="r",marker=".")
plt.xlabel("STATES",size=12)
plt.ylabel("RAINFALL IN MM",size=12)
plt.grid(axis="both",linestyle="-.")
plt.title("RAINFALL IN DISTRICTS ANNUALLY")
plt.show()
```



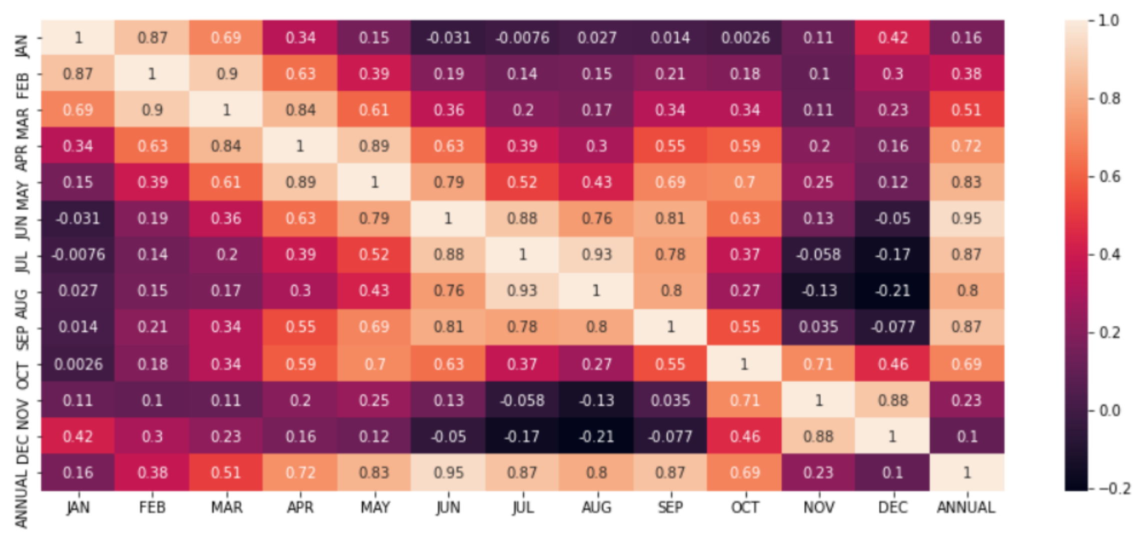
In [19]:
```
plt.figure(figsize=(15,8))
data.groupby("DISTRICT").sum()['ANNUAL'].plot(kind="line",color="r",marker=".")
plt.xlabel("DISTRICTS",size=12)
plt.ylabel("RAINFALL IN MM",size=12)
plt.grid(axis="both",linestyle="-.")
plt.title("RAINFALL IN DISTRICTS ANNUALLY")
plt.show()
```



- ***Heat Map*** shows the correlation(dependency) between the amounts of rainfall over months. 8
  - From below graph it is clear that if the amount of rainfall is high in the months of july, august, september then the amount of rainfall will be high annually.
  - It is also observed that if the amount of rainfall is good in the months of October, November, December then the rainfall is going to be good in the overall year.

DISTRICTS

In [20]:
```
#Correlation between each numeric attribute
plt.figure(figsize=(15,6))
sns.heatmap(data[['JAN','FEB','MAR','APR','MAY','JUN','JUL','AUG','SEP','OCT','NOV','DEC','ANNUAL']].corr(),annot=True)
plt.show()
```



- Subdivisions receiving maximum and minimum rainfall:

```python
# Subdivisions receiving maximum and minimum rainfall
print("Top 10")
print(data.groupby('SUBDIVISION').mean()['ANNUAL'].sort_values(ascending=False).head(10))
print('\n')
print("--------------------------------------------")
print("Tail 10")
print(data.groupby('SUBDIVISION').mean()['ANNUAL'].sort_values(ascending=False).tail(10))
```

```
Top 10
SUBDIVISION
Coastal Karnataka                     3380.644865
Arunachal Pradesh                     3283.079750
Konkan & Goa                          2987.531624
Kerala                                2914.247009
Andaman & Nicobar Islands             2845.109779
Sub Himalayan West Bengal & Sikkim    2750.552991
Assam & Meghalaya                     2579.133333
Naga Mani Mizo Tripura                2432.717949
Lakshadweep                           1570.429666
Gangetic West Bengal                  1490.612821
Name: ANNUAL, dtype: float64


--------------------------------------------
Tail 10
SUBDIVISION
Madhya Maharashtra            881.431624
West Uttar Pradesh            823.899145
Matathwada                    791.745299
Rayalseema                    764.988034
North Interior Karnataka      717.188889
East Rajasthan                656.501709
Punjab                        591.436752
Haryana Delhi & Chandigarh    528.439316
Saurashtra & Kutch            496.398291
West Rajasthan                294.125641
Name: ANNUAL, dtype: float64
```

```python
# STATES receiving maximum and minimum rainfall
print("Top 10")
print(data.groupby('STATE_UT_NAME').mean()['ANNUAL'].sort_values(ascending=False).head(10))
print('\n')
print("------------------------------------------")
print("Tail 10")
print(data.groupby('STATE_UT_NAME').mean()['ANNUAL'].sort_values(ascending=False).tail(10))
```

```
Top 10
STATE_UT_NAME
MEGHALAYA                      3682.842857
GOA                            3278.500000
KERALA                         2937.392857
ARUNACHAL PRADESH              2927.375000
ANDAMAN And NICOBAR ISLANDS    2911.400000
SIKKIM                         2838.350000
MIZORAM                        2616.322222
MANIPUR                        2496.633333
TRIPURA                        2479.125000
ASSAM                          2454.359259
Name: ANNUAL, dtype: float64


--------------------------------------------
Tail 10
STATE_UT_NAME
MADHYA PRADESH       1032.310000
JAMMU AND KASHMIR    1016.618182
TAMIL NADU            960.006250
UTTAR PRADESH         955.445070
ANDHRA PRADESH        945.073913
GUJARAT               924.342308
DELHI                 747.100000
PUNJAB                648.545000
HARYANA               614.557143
RAJASTHAN             581.596970
Name: ANNUAL, dtype: float64
```

```
# DISTRICTS receiving maximum and minimum rainfall
print("Top 10")
print(data.groupby('DISTRICT').mean()['ANNUAL'].sort_values(ascending=False).head(10))
print('\n')
print("--------------------------------------------")
print("Tail 10")
print(data.groupby('DISTRICT').mean()['ANNUAL'].sort_values(ascending=False).tail(10))
```

```
Top 10
```

```
Top 10
DISTRICT
TAMENGLONG        7229.3
JAINTIA HILLS     6379.9
EAST KHASI HI     6166.1
UPPER SIANG       4402.1
UDUPI             4306.0
EAST SIANG        4034.7
DAKSHIN KANDA     3915.8
KOKRAJHAR         3772.2
KARIMGANJ         3650.8
W KHASI HILL      3643.0
Name: ANNUAL, dtype: float64


----------------------------------
Tail 10
DISTRICT
FATEHABAD         364.6
SIRSA             313.5
JODHPUR           308.1
HANUMANGARH       301.6
BIKANER           274.0
BARMER            268.6
SRI GANGANAGA     252.9
KARGIL            223.3
JAISALMER         181.2
LADAKH (LEH)       94.6
Name: ANNUAL, dtype: float64
```

Modelling and Supervised/unsupervised Learning Algorithms

In [23]:

```
#Modelling
data["STATE_UT_NAME"].nunique()
group = data.groupby('STATE_UT_NAME')['DISTRICT','JAN','FEB','MAR','APR','MAY','JUN','JUL','AUG','SEP',
'OCT','NOV','DEC']
data=group.get_group(('KERALA'))
data.head()
```

```
<ipython-input-23-be41abed401d>:3: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.
  group = data.groupby('STATE_UT_NAME')['DISTRICT','JAN','FEB','MAR','APR','MAY','JUN','JUL','AUG','SEP',
```

Out[23]:

	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
626	ALAPPUZHA	17.5	27.9	45.1	134.0	298.7	593.0	533.0	343.1	276.8	332.9	187.6	51.6
627	CANNUR	2.5	2.0	7.6	57.9	235.0	852.4	1055.0	540.9	220.7	229.4	91.6	24.1
628	ERNAKULAM	13.2	18.8	31.2	112.0	300.4	696.4	670.2	401.3	297.1	299.9	145.0	44.4
629	KOTTAYAM	13.0	24.9	42.3	136.1	281.8	649.1	591.4	386.0	270.8	316.6	177.4	41.1
630	KOZHIKODE	2.3	3.9	15.1	80.9	256.6	895.8	955.2	500.9	251.2	262.1	130.1	30.0

In [24]:
```
df=data.melt(['DISTRICT']).reset_index()
df.head()
```

Out[24]:

	index	DISTRICT	variable	value
0	0	ALAPPUZHA	JAN	17.5
1	1	CANNUR	JAN	2.5
2	2	ERNAKULAM	JAN	13.2
3	3	KOTTAYAM	JAN	13.0
4	4	KOZHIKODE	JAN	2.3

In [25]:
```
df= df[['DISTRICT','variable','value']].reset_index().sort_values(by=['DISTRICT','index'])
df.head(24)
```

Out[25]:

	index	DISTRICT	variable	value
0	0	ALAPPUZHA	JAN	17.5
14	14	ALAPPUZHA	FEB	27.9
28	28	ALAPPUZHA	MAR	45.1
42	42	ALAPPUZHA	APR	134.0
56	56	ALAPPUZHA	MAY	298.7
70	70	ALAPPUZHA	JUN	593.0
84	84	ALAPPUZHA	JUL	533.0
98	98	ALAPPUZHA	AUG	343.1
112	112	ALAPPUZHA	SEP	276.8
126	126	ALAPPUZHA	OCT	332.9
140	140	ALAPPUZHA	NOV	187.6
154	154	ALAPPUZHA	DEC	51.6
1	1	CANNUR	JAN	2.5
15	15	CANNUR	FEB	2.0
29	29	CANNUR	MAR	7.6
43	43	CANNUR	APR	57.9
57	57	CANNUR	MAY	235.0
71	71	CANNUR	JUN	852.4
85	85	CANNUR	JUL	1055.0

In [26]:
```
df.DISTRICT.unique()
df.columns=['Index','District','Month','Avg_Rainfall']
df.head()
```

Out[26]:

	Index	District	Month	Avg_Rainfall
0	0	ALAPPUZHA	JAN	17.5
14	14	ALAPPUZHA	FEB	27.9
28	28	ALAPPUZHA	MAR	45.1
42	42	ALAPPUZHA	APR	134.0
56	56	ALAPPUZHA	MAY	298.7

In [27]:
```
Month_map={'JAN':1,'FEB':2,'MAR'  :3,'APR':4,'MAY':5,'JUN':6,'JUL':7,'AUG':8,'SEP':9,  'OCT':10,'NOV':11,'DEC':12}
df['Month']=df['Month'].map(Month_map)
df.head(12)
```

Out[27]:

	Index	District	Month	Avg_Rainfall
0	0	ALAPPUZHA	1	17.5
14	14	ALAPPUZHA	2	27.9
28	28	ALAPPUZHA	3	45.1
42	42	ALAPPUZHA	4	134.0
56	56	ALAPPUZHA	5	298.7
70	70	ALAPPUZHA	6	593.0
84	84	ALAPPUZHA	7	533.0
98	98	ALAPPUZHA	8	343.1
112	112	ALAPPUZHA	9	276.8
126	126	ALAPPUZHA	10	332.9
140	140	ALAPPUZHA	11	187.6
154	154	ALAPPUZHA	12	51.6

In [29]:
```
District_map={'ALAPPUZHA':20,'CANNUR':21,'ERNAKULAM':22,'KOTTAYAM':23,'KOZHIKODE':24,'MALAPPURAM':25,'PALAKKAD':26,'KOLLAM':27,'
df['District']=df['District'].map(District_map)
df.head(24)
```

	Index	District	Month	Avg_Rainfall
0	0	20	1	17.5
14	14	20	2	27.9
28	28	20	3	45.1
42	42	20	4	134.0
56	56	20	5	298.7
70	70	20	6	593.0
84	84	20	7	533.0
98	98	20	8	343.1
112	112	20	9	276.8
126	126	20	10	332.9
140	140	20	11	187.6
154	154	20	12	51.6
1	1	21	1	2.5
15	15	21	2	2.0
29	29	21	3	7.6
43	43	21	4	57.9
57	57	21	5	235.0
71	71	21	6	852.4
85	85	21	7	1055.0
99	99	21	8	540.9
113	113	21	9	220.7
127	127	21	10	229.4

In [30]:
```python
df.drop(columns="Index",inplace=True)
df.head(2)
```

Out[30]:

	District	Month	Avg_Rainfall
0	20	1	17.5
14	20	2	27.9

In [31]:
```python
df.groupby("District").sum().plot()
plt.show()
```

In [32]:
```python
X=np.asanyarray(df[['District','Month']]).astype('int')
y=np.asanyarray(df['Avg_Rainfall']).astype('float')
print(X.shape)
print(y.shape)
```

```
(168, 2)
(168,)
```

- Linear Regression Model

In [34]:
```
#Linear Regression Model
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(X_train,y_train)
## predicting
y_train_predict=LR.predict(X_train)
y_test_predict=LR.predict(X_test)
print(y_train_predict)
print(" ")
print("---------------------------------------------------")
print(" ")
print(y_test_predict)
```

```
[259.4373172  269.05978948 275.71577935 187.30596621 304.58319618
 334.17363807 262.76531214 171.38901659 284.9767391  285.33825163
 320.86165833 184.70099633 216.8964081  171.02750406 266.09330707
 298.28871884 233.17487025 233.53638278 183.97797128 321.22317086
 236.14135266 196.92843849 308.27270365 314.56718099 272.74929695
 197.65146355 256.4708348  353.78009516 327.87916073 181.01148887
 327.5176482  279.04377429 272.02627189 194.32346861 207.27393582
 151.42104698 240.19237265 301.97822631 177.68349393 245.76382493
 354.14160769 197.28995102 318.25668846 330.84564314 317.5336634
 357.10809009 217.25792063 213.56841317 294.96072391 337.50163301
 213.20690064 229.48536279 311.60069858 265.73179455 164.73302672
 161.40503179 190.99547368 223.55239797 206.91242329 262.04228708
 344.15762288 232.81335772 289.02775909 253.14283986 350.45210022
 288.30473404 350.81361275 337.86314554 370.42006984 256.10932227
 190.63396115 219.86289051 327.15613568 188.02899127 324.18965327
 330.48413061 314.20566846 340.82962795 307.91119112 249.09181987
 203.58442836 181.3730014  331.20715567 268.69827695 252.78132733
 242.79734253 252.4198148  193.96195608 291.63272897 285.69976416
 167.69950913 239.46934759 272.38778442 304.94470871 301.25520125
 203.94594089 220.58591557 343.79611035 226.51888038 282.37176922
 291.9942415  229.84687531 216.53489557 282.01025669 246.48684999
 148.09305204 174.355499   278.32074923 305.30622124 207.63544835
 191.3569862  236.86437771 249.4533324  236.50286518 200.25643342
 269.42130201 294.59921138]

--------------------------------------------------

[230.20838784 187.66747874 154.74904191 246.12533746 317.89517593
 337.14012048 259.07580468 275.35426682 334.5351506  168.06102166
 226.88039291 255.74780974 340.46811542 200.61794595 310.87767353
 243.15885506 178.04500646 281.64874417 223.9139105  278.68226176
 311.23918606 295.32223644 321.58468339 164.37151419 297.92720631
 301.61671378 262.40379961 298.65023137 265.37028202 200.97945848
 210.24041823 239.83086012 174.71701153 324.5511658  204.30745342
 367.0920749  333.81212555 220.22440304 158.07703685 180.64997634
```

In [35]:
```
print("-------Test Data ------- ")
print('MAE:', metrics.mean_absolute_error(y_test, y_test_predict))
print('MSE:', metrics.mean_squared_error(y_test, y_test_predict))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_predict)))
print("\n-------Train Data--------")
print('MAE:', metrics.mean_absolute_error(y_train,y_train_predict))
print('MSE:', metrics.mean_squared_error(y_train, y_train_predict))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_predict)))
```

```
-------Test Data -------
MAE: 184.67257922986815
MSE: 51916.00651240563
RMSE: 227.85084268530943

-------Train Data--------
MAE: 192.08061033900566
MSE: 66586.55645267476
RMSE: 258.04371035286783
```

In [36]:
```
print("\n-----Training Accuracy-------")
print(round(LR.score(X_train,y_train),3)*100)
print("-----Testing Accuracy -------")
print(round(LR.score(X_test,y_test),3)*100)
```

```
-----Training Accuracy-------
4.5
-----Testing Accuracy -------
0.8
```

- Lasso Model

In [37]:
```
#Lasso Model
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

In [38]:
```
# create a lasso object
lasso = Lasso(max_iter=100000)
```

In [39]:
```
# check for best alpha value using GridSearch
parameter={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,1e1,1e2,1e3,1e4,1e5,1e6,1e7]}
lasso_regressor=GridSearchCV(
lasso,parameter,
scoring='neg_mean_squared_error',cv=5)
```

In [40]:
```
lasso_regressor.fit(X_train,y_train)
print("Best Parameter for Lasso:",lasso_regressor.best_estimator_)
```

```
C:\Users\capta\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:530: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 2653665.880320302, tolerance: 681.389496795699
  model = cd_fast.enet_coordinate_descent(
C:\Users\capta\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:530: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 3105935.570748394, tolerance: 682.6731038924731
  model = cd_fast.enet_coordinate_descent(
C:\Users\capta\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:530: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 1823284.2732549058, tolerance: 622.2498838297872
  model = cd_fast.enet_coordinate_descent(
C:\Users\capta\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:530: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 3013980.0351236146, tolerance: 674.9222232765958
  model = cd_fast.enet_coordinate_descent(
```

```
Best Parameter for Lasso: Lasso(alpha=10.0, max_iter=100000)
```

In [41]:
```
lasso=Lasso(alpha=100.0,max_iter=100000)
# fit into the object
lasso.fit(X_train,y_train)
```

Out[41]:
```
Lasso(alpha=100.0, max_iter=100000)
```

In [42]:
```
# predicting
y_train_predict=lasso.predict(X_train)
y_test_predict=lasso.predict(X_test)
print(y_train_predict)
print(" ")
print("---------------------------------------------------")
print(" ")
print(y_test_predict)
```

```
[255.13527824 262.43654922 262.43654922 233.23146528 284.34036218
 291.64163317 255.13527824 218.62892331 277.0390912  269.73782021
 291.64163317 218.62892331 240.53273627 225.93019429 255.13527824
 277.0390912  247.83400725 240.53273627 233.23146528 284.34036218
 255.13527824 240.53273627 277.0390912  284.34036218 255.13527824
 225.93019429 247.83400725 298.94290415 284.34036218 225.93019429
 291.64163317 262.43654922 269.73782021 225.93019429 233.23146528
 218.62892331 240.53273627 269.73782021 225.93019429 262.43654922
 291.64163317 233.23146528 277.0390912  291.64163317 291.64163317
 298.94290415 233.23146528 240.53273627 277.0390912  291.64163317
 247.83400725 255.13527824 277.0390912  262.43654922 218.62892331
 218.62892331 225.93019429 240.53273627 240.53273627 269.73782021
 291.64163317 255.13527824 262.43654922 247.83400725 298.94290415
 277.0390912  291.64163317 284.34036218 298.94290415 255.13527824
 233.23146528 247.83400725 298.94290415 218.62892331 291.64163317
 298.94290415 291.64163317 291.64163317 284.34036218 262.43654922
 240.53273627 218.62892331 284.34036218 269.73782021 255.13527824
 255.13527824 262.43654922 233.23146528 277.0390912  262.43654922
 225.93019429 255.13527824 262.43654922 277.0390912  284.34036218
 233.23146528 233.23146528 298.94290415 247.83400725 262.43654922
 269.73782021 247.83400725 247.83400725 269.73782021 247.83400725
 218.62892331 225.93019429 277.0390912  269.73782021 225.93019429
 218.62892331 240.53273627 255.13527824 247.83400725 240.53273627
 255.13527824 284.34036218]

--------------------------------------------------

[240.53273627 225.93019429 218.62892331 255.13527824 284.34036218
 298.94290415 262.43654922 269.73782021 284.34036218 218.62892331
 240.53273627 262.43654922 298.94290415 233.23146528 291.64163317
 247.83400725 218.62892331 277.0390912  233.23146528 269.73782021
 284.34036218 269.73782021 277.0390912  225.93019429 284.34036218
 277.0390912  262.43654922 269.73782021 269.73782021 225.93019429
 240.53273627 247.83400725 218.62892331 284.34036218 225.93019429
 298.94290415 298.94290415 240.53273627 218.62892331 233.23146528
 298.94290415 269.73782021 233.23146528 298.94290415 298.94290415
```

In [43]:
```
#lasso regression
from sklearn import metrics
print("\n-------Train Data--------")
print('MAE:', metrics.mean_absolute_error(y_train,y_train_predict))
print('MSE:', metrics.mean_squared_error(y_train, y_train_predict))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_predict)))
print("-------Test Data ------- ")
print('MAE:', metrics.mean_absolute_error(y_test, y_test_predict))
print('MSE:', metrics.mean_squared_error(y_test, y_test_predict))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_predict)))
```

```
-------Train Data--------
MAE: 194.21982647288868
MSE: 67675.20979534372
RMSE: 260.14459401522015
-------Test Data -------
MAE: 187.8943453001915
MSE: 52003.52658010239
RMSE: 228.04281742712791
```

In [44]:
```
print("\n-----Training Accuracy-------")
print(round(lasso.score(X_train,y_train),3)*100)
print("-----Testing Accuracy -------")
print(round(lasso.score(X_test,y_test),3)*100)
```

```
-----Training Accuracy-------
2.9000000000000004
-----Testing Accuracy -------
0.7000000000000001
```

- Ridge Model

In [45]:
```
#Ridge Model
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
```

In [46]:
```
ridge=Ridge()
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
ridge_regressor=GridSearchCV(ridge,parameters,scoring='neg_mean_squared_error',cv=5)
ridge_regressor.fit(X_train,y_train)
```

Out[46]:
```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.001, 0.01, 1, 5, 10,
                                   20, 30, 35, 40, 45, 50, 55, 100]},
             scoring='neg_mean_squared_error')
```

In [47]:
```
print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
print("Best Parameter for Ridge:",ridge_regressor.best_estimator_)
```

```
{'alpha': 100}
-69449.81274879692
Best Parameter for Ridge: Ridge(alpha=100)
```

In [48]:
```
ridge=Ridge(alpha=100.0)
# fit into the object
ridge.fit(X_train,y_train)
```

Out[48]:
```
Ridge(alpha=100.0)
```

In [49]:
```
# predicting
y_train_predict=ridge.predict(X_train)
y_test_predict=ridge.predict(X_test)
print(y_train_predict)
print(" ")
print("-------------------------------------------------------")
print(" ")
print(y_test_predict)
```

```
[259.59828919 268.37014237 274.71564463 192.03696493 301.03120417
 328.83956414 262.77104032 177.66600971 282.74109759 283.48749781
 316.14855961 190.35701424 219.84532491 176.91960949 265.94379145
 295.43210212 234.96268035 235.70908057 188.8642138  316.89495983
 237.38903126 200.80881811 304.95035552 310.54945756 272.28929372
 202.30161855 257.17193828 347.12967072 323.24046209 186.43786289
 322.49406187 277.88839577 270.79649328 199.12886742 211.07347173
 158.62950291 242.05458283 299.35125347 183.26511175 246.16088444
 347.87607094 201.55521833 314.46860892 325.66681301 312.97580848
 350.30242185 220.59172513 216.67257377 292.25935099 332.01231527
 215.92617355 231.043529   308.12310665 265.19739123 171.32050744
 168.14775631 195.95611628 226.19082717 210.32707151 261.27823988
 338.35781754 234.21628013 287.40664916 253.99918714 343.95691958
 285.91384872 344.7033198  332.75871549 362.99342638 256.42553806
 195.20971606 222.27167582 321.74766165 193.52976537 319.32131074
 324.92041279 309.80305734 335.18506641 304.2039553  249.33363557
 207.15432038 187.18426311 326.41321323 267.62374215 253.25278692
 243.73453353 252.5063867  198.3824672  289.08659986 284.23389803
 173.74685836 240.56178239 271.5428935  301.77760439 297.85845303
 207.9007206  223.76447626 337.61141732 228.61717808 281.0611469
 289.83300008 231.78992922 219.09892469 280.31474668 247.65368488
 155.45675178 180.09236062 276.39559533 302.52400461 211.81987195
 196.7025165  238.8818317  250.08003579 238.13543148 203.98156924
 269.11654259 291.51295077]

--------------------------------------------------

[232.53632944 192.78336515 161.80225405 246.90728466 313.7222087
 331.26591505 258.85188897 273.96924441 329.58596436 174.49325858
 229.3635783  255.67913784 334.43866619 204.72796946 306.63030621
 244.48093375 184.01151197 279.56834646 226.93722739 277.14199555
 307.37670643 293.00575121 317.64136005 170.57410722 294.6857019
 298.60485325 262.0246401  296.17850234 264.45099101 205.47436968
 213.49982264 241.30818261 180.83876084 320.06771096 208.64712082
 359.82067525 328.09316392 223.01807604 164.97500518 185.69146267
 356.64792411 286.66024894 217.41897399 353.47517298 340.78416845
```

```
In [50]: from sklearn import metrics
         print("\n-------Train Data--------")
         print('MAE:', metrics.mean_absolute_error(y_train,y_train_predict))
         print('MSE:', metrics.mean_squared_error(y_train, y_train_predict))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_predict)))

         -------Train Data--------
         MAE: 192.32198600864226
         MSE: 66601.9798823517
         RMSE: 258.0735939269101
```

```
In [51]: print("-------Test Data ------- ")
         print('MAE:', metrics.mean_absolute_error(y_test, y_test_predict))
         print('MSE:', metrics.mean_squared_error(y_test, y_test_predict))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_predict)))

         -------Test Data -------
         MAE: 185.19321627598592
         MSE: 51871.69295897291
         RMSE: 227.75357946467693
```

```
In [52]: print("\n-----Training Accuracy-------")
         print(round(ridge.score(X_train,y_train),3)*100)
         print("-----Testing Accuracy -------")
         print(round(ridge.score(X_test,y_test),3)*100)

         -----Training Accuracy-------
         4.5
         -----Testing Accuracy -------
         0.8999999999999999
```

- Random Forest Model

```
In [53]: #Random Forest Model
         from sklearn.ensemble import RandomForestRegressor
         random_forest_model = RandomForestRegressor(max_depth=100, max_features='sqrt', min_samples_leaf=4,
         min_samples_split=10, n_estimators=800)
         random_forest_model.fit(X_train, y_train)
         y_train_predict=random_forest_model.predict(X_train)
         y_test_predict=random_forest_model.predict(X_test)
         #print(y_train_predict)
         #print(" ")
         #print("--------------------------------------------------")
         #print(" ")
         #print(y_test_predict)

         print("\n-------Train Data--------")
         print('MAE:', metrics.mean_absolute_error(y_train,y_train_predict))
         print('MSE:', metrics.mean_squared_error(y_train, y_train_predict))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_predict)))

         print("-------Test Data ------- ")
         print('MAE:', metrics.mean_absolute_error(y_test, y_test_predict))
         print('MSE:', metrics.mean_squared_error(y_test, y_test_predict))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_predict)))

         print("----------Training Accuracy ---------- ")
         print(round(random_forest_model.score(X_train,y_train),3)*100)
         print("----------Testing Accuracy ---------- ")
         print(round(random_forest_model.score(X_test,y_test),3)*100)
```

```
-------Train Data--------
MAE: 87.43423713905166
MSE: 18138.590862778467
RMSE: 134.6795859170144
-------Test Data -------
MAE: 102.58773358386706
MSE: 18710.08874290393
RMSE: 136.7848264351859
-----------Training Accuracy -----------
74.0
-----------Testing Accuracy -----------
64.3
```

TESTING THE OUTPUT

In [54]:
```
predicted = random_forest_model.predict([[21,10]])
print(predicted)
```

```
[301.41678687]
```

Average accuracy comparison of various algorithms

ALGORITHMS	TRAINING	TESTING
Linear regression	4.5	0.8
Lasso	2.90	0.7
Ridge	4.5	0.8
Svm	Not compatible since data is discreate	--
Random forest	74	64.3

From the above table we can conclude that random forest model was the best model for this dataset

REPORT SUBMITTED BY:

GOWTHAM PULIVENDULA
AM.EN.U4CSE19143(CSE-B)