

```
In [2]: import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

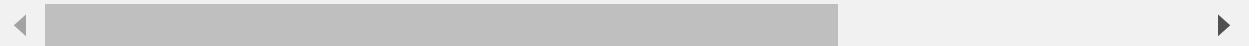
from sklearn import metrics
```

data collection

```
In [67]: #data collection
data = pd.read_csv("rainfall in india 1901-2015.csv")
data.head()
```

Out[67]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV
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
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
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
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
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



In [4]: `data.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
```

data preprocessing

```
In [5]: # data preprocessing  
data.isnull().sum()
```

```
Out[5]: 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 [6]: data.duplicated().sum()
```

```
Out[6]: 0
```

```
In [7]: data['SUBDIVISION'].value_counts()
```

Out[7]:	SOUTH INTERIOR KARNATAKA	115
	RAYALSEEMA	115
	MATATHWADA	115
	JAMMU & KASHMIR	115
	WEST RAJASTHAN	115
	HIMACHAL PRADESH	115
	ORISSA	115
	BIHAR	115
	PUNJAB	115
	COASTAL KARNATAKA	115
	TELANGANA	115
	MADHYA MAHARASHTRA	115
	EAST UTTAR PRADESH	115
	WEST MADHYA PRADESH	115
	VIDARBHA	115
	JHARKHAND	115
	SUB HIMALAYAN WEST BENGAL & SIKKIM	115
	HARYANA DELHI & CHANDIGARH	115
	ASSAM & MEGHALAYA	115
	KERALA	115
	NORTH INTERIOR KARNATAKA	115
	GANGETIC WEST BENGAL	115
	NAGA MANI MIZO TRIPURA	115
	SAURASHTRA & KUTCH	115
	EAST MADHYA PRADESH	115
	KONKAN & GOA	115
	WEST UTTAR PRADESH	115
	CHHATTISGARH	115
	COASTAL ANDHRA PRADESH	115
	UTTARAKHAND	115
	TAMIL NADU	115
	EAST RAJASTHAN	115
	GUJARAT REGION	115
	LAKSHADWEEP	114
	ANDAMAN & NICOBAR ISLANDS	110
	ARUNACHAL PRADESH	97

```
In [8]: data.mean()
```

```
Out[8]: YEAR      1958.218659  
JAN       18.957320  
FEB       21.805325  
MAR       27.359197  
APR       43.127432  
MAY       85.745417  
JUN      230.234444  
JUL      347.214334  
AUG      290.263497  
SEP      197.361922  
OCT      95.507009  
NOV      39.866163  
DEC      18.870580  
ANNUAL    1411.008900  
Jan-Feb   40.747786  
Mar-May   155.901753  
Jun-Sep   1064.724769  
Oct-Dec   154.100487  
dtype: float64
```

```
In [9]: data = data.fillna(data.mean())
```

```
In [10]: data.head(3)
```

```
Out[10]:
```

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV
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
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
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



```
In [11]: data.isnull().any()
```

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

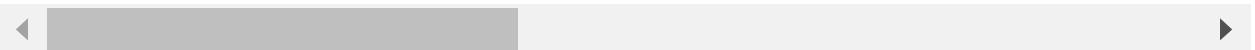
```
In [12]: data.YEAR.unique()
```

```
Out[12]: array([1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1910, 1911, 1912,
 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923,
 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934,
 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1946, 1947, 1949,
 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960,
 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971,
 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982,
 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993,
 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004,
 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
 1943, 1944, 1945, 1948, 1909], dtype=int64)
```

```
In [13]: data.describe()
```

Out[13]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN
count	4116.000000	4116.000000	4116.000000	4116.000000	4116.000000	4116.000000	4116.000000
mean	1958.218659	18.957320	21.805325	27.359197	43.127432	85.745417	230.234444
std	33.140898	33.569044	35.896396	46.925176	67.798192	123.189974	234.568120
min	1901.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.400000
25%	1930.000000	0.600000	0.600000	1.000000	3.000000	8.600000	70.475000
50%	1958.000000	6.000000	6.700000	7.900000	15.700000	36.700000	138.900000
75%	1987.000000	22.125000	26.800000	31.225000	49.825000	96.825000	304.950000
max	2015.000000	583.700000	403.500000	605.600000	595.100000	1168.600000	1609.900000

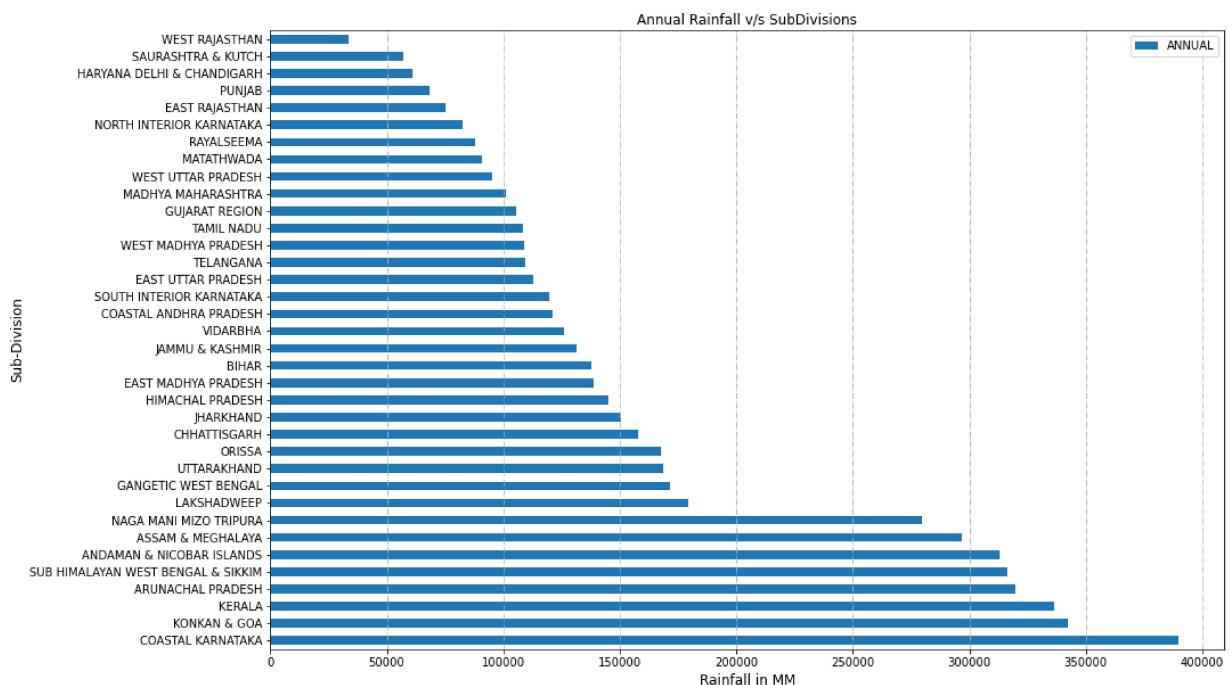


```
In [14]: data.shape
```

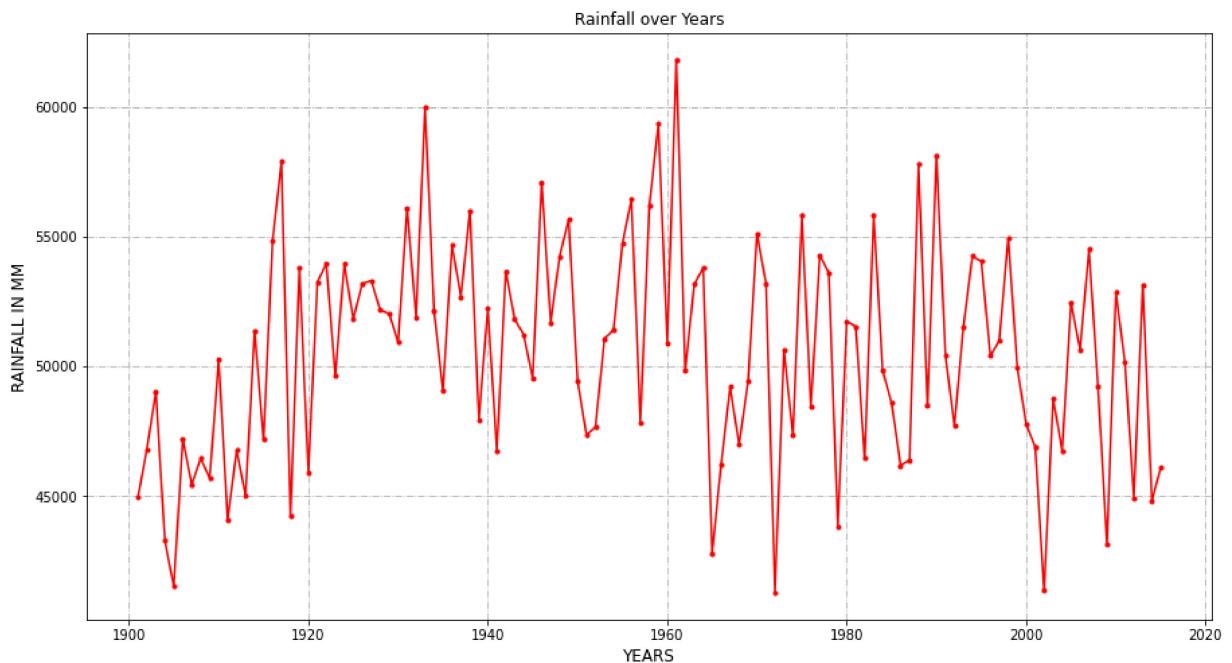
Out[14]: (4116, 19)

```
In [15]: # data visulization
```

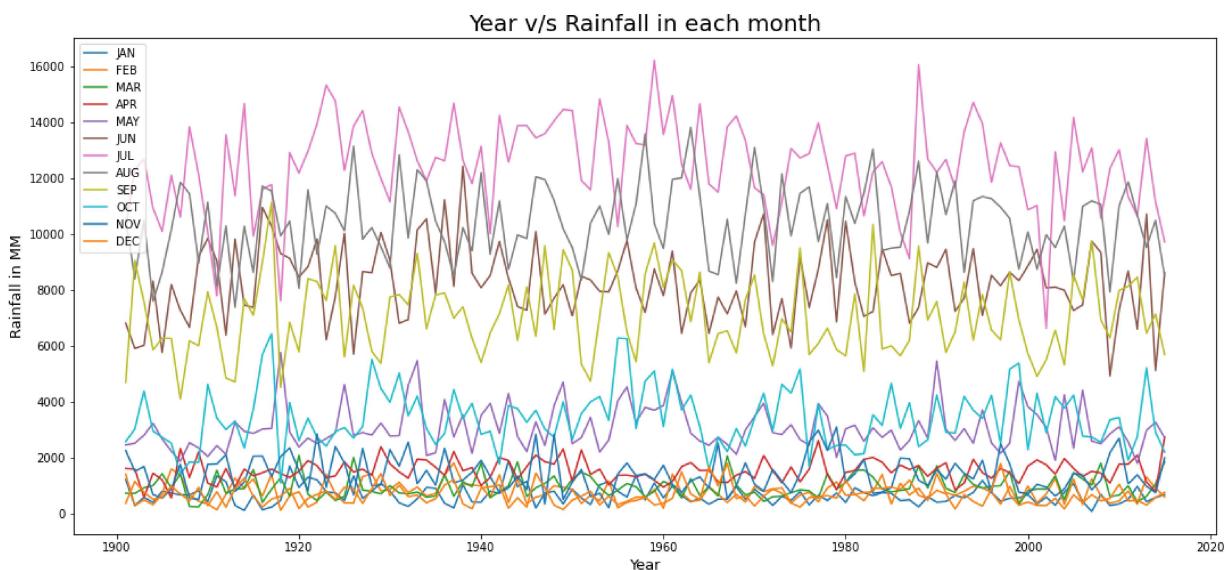
```
data[["SUBDIVISION", "ANNUAL"]].groupby("SUBDIVISION").sum().sort_values(by='ANNUAL')
plt.xlabel("Rainfall in MM", size=12)
plt.ylabel("Sub-Division", size=12)
plt.title("Annual Rainfall v/s SubDivisions")
plt.grid(axis="x", linestyle="- .")
plt.show()
```



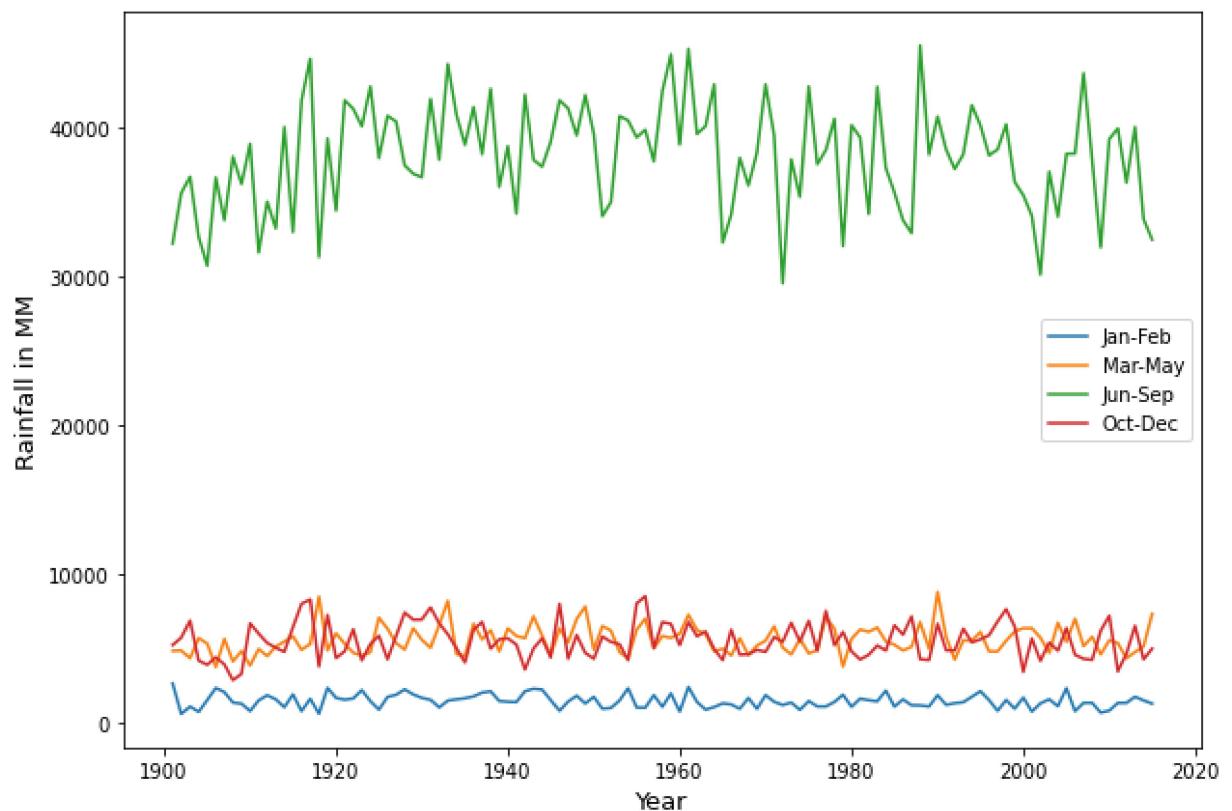
```
In [16]: plt.figure(figsize=(15,8))
data.groupby("YEAR").sum()['ANNUAL'].plot(kind="line",color="r",marker=".")
plt.xlabel("YEARS",size=12)
plt.ylabel("RAINFALL IN MM",size=12)
plt.grid(axis="both",linestyle="-.")
plt.title("Rainfall over Years")
plt.show()
```



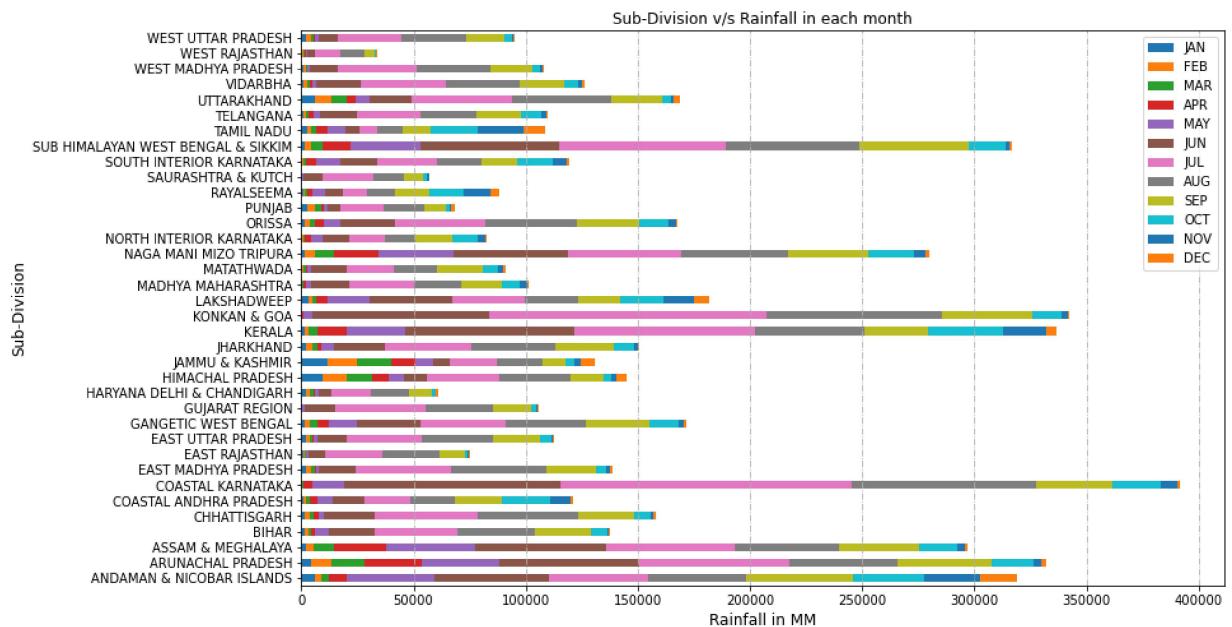
```
In [17]: data[['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP',
          'OCT', 'NOV', 'DEC']].groupby("YEAR").sum().plot(kind="line",figsize=(18,8))
plt.xlabel("Year",size=13)
plt.ylabel("Rainfall in MM",size=13)
plt.title("Year v/s Rainfall in each month",size=20)
plt.show()
```



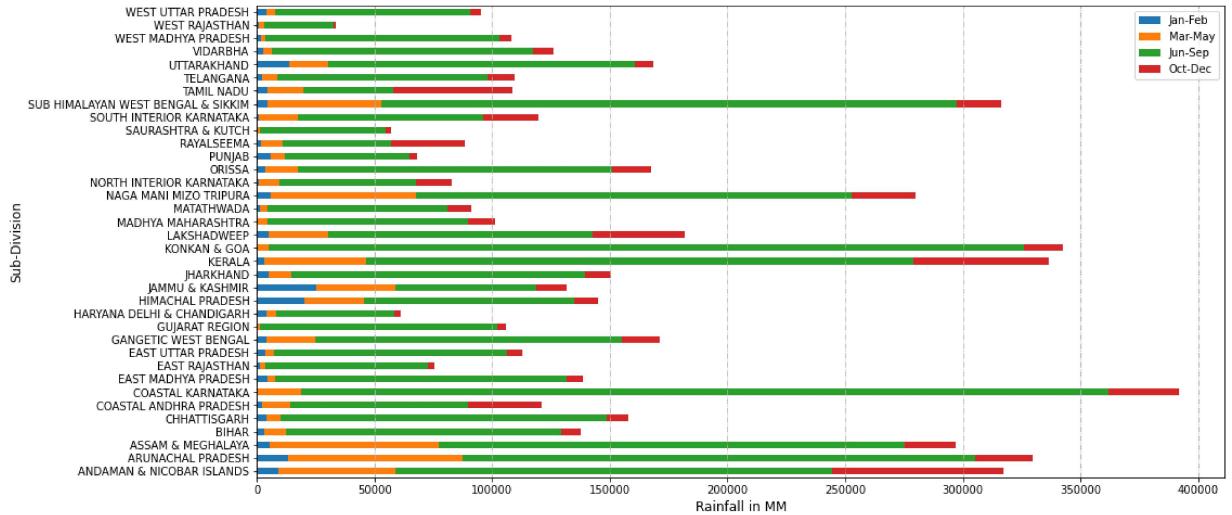
```
In [18]: data[['YEAR', 'Jan-Feb', 'Mar-May',  
           'Jun-Sep', 'Oct-Dec']].groupby("YEAR").sum().plot(figsize=(10,7))  
plt.xlabel("Year",size=13)  
plt.ylabel("Rainfall in MM",size=13)  
plt.show()
```



```
In [19]: data[['SUBDIVISION', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',  
          'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("SUBDIVISION").sum().plot(kind="barh")  
plt.title("Sub-Division v/s Rainfall in each month")  
plt.xlabel("Rainfall in MM",size=12)  
plt.ylabel("Sub-Division",size=12)  
plt.grid(axis="x",linestyle="-.")  
plt.show()
```



```
In [20]: data[['SUBDIVISION', 'Jan-Feb', 'Mar-May',  
          'Jun-Sep', 'Oct-Dec']].groupby("SUBDIVISION").sum().plot(kind="barh",stacked=True)  
plt.xlabel("Rainfall in MM",size=12)  
plt.ylabel("Sub-Division",size=12)  
plt.grid(axis="x",linestyle="-.")  
plt.show()
```

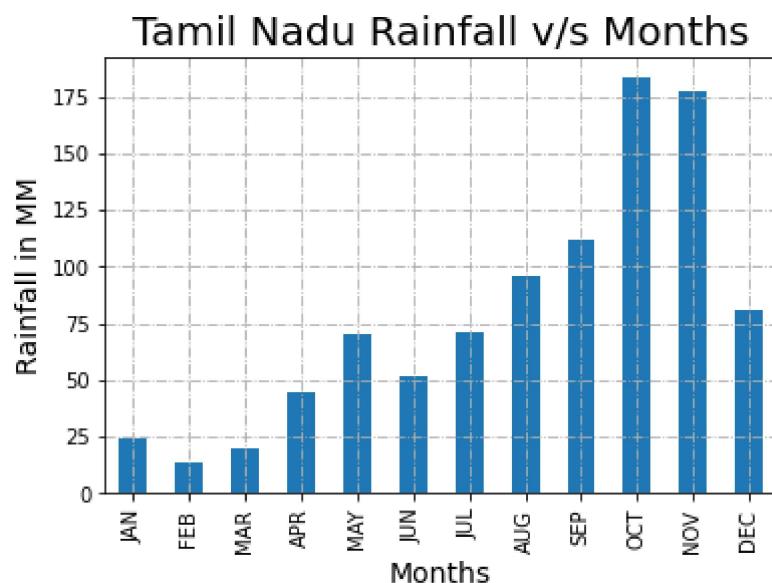


```
In [21]: TN = data.loc[((data['SUBDIVISION'] == 'TAMIL NADU'))]
TN.head(4)
```

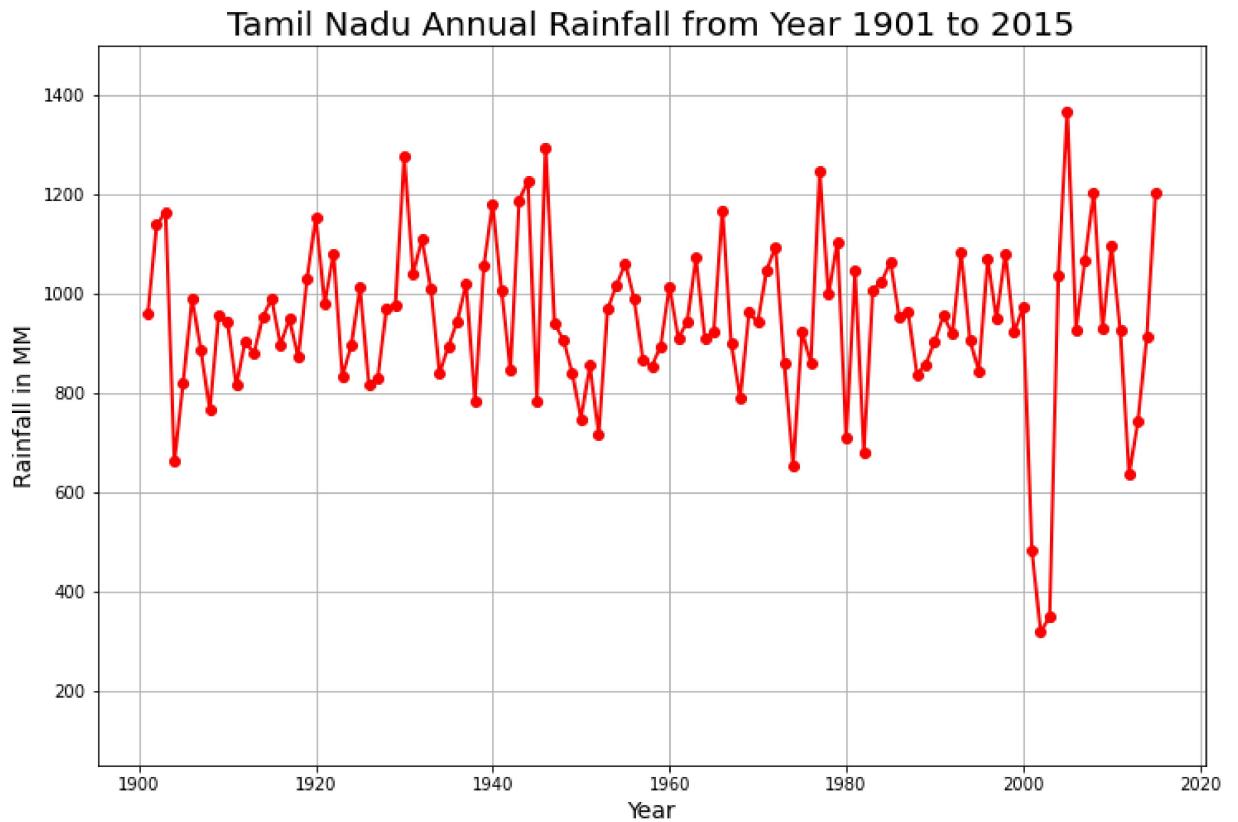
Out[21]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
3427	TAMIL NADU	1901	24.5	39.1	21.7	36.0	74.0	41.8	49.3	67.9	191.1	122.3	212	112.5
3428	TAMIL NADU	1902	67.2	9.8	25.1	21.9	84.7	39.3	55.1	113.8	98.6	282.2	174	100.5
3429	TAMIL NADU	1903	19.3	7.8	1.7	18.2	128.5	58.5	72.6	115.0	210.4	128.1	200	100.5
3430	TAMIL NADU	1904	35.2	0.1	0.7	19.5	121.9	34.9	89.0	40.4	85.7	163.2	23	100.5

```
In [22]: TN[['JAN', 'FEB', 'MAR', 'APR','MAY', 'JUN','JUL','AUG', 'SEP', 'OCT','NOV','DEC']]
plt.title("Tamil Nadu Rainfall v/s Months",size=20)
plt.xlabel("Months",size=14)
plt.ylabel("Rainfall in MM",size=14)
plt.grid(axis="both",linestyle="-.")
plt.show()
```



```
In [23]: TN.groupby("YEAR").sum()['ANNUAL'].plot(ylim=(50,1500),color='r',marker='o',lines  
plt.xlabel('Year',size=14)  
plt.ylabel('Rainfall in MM',size=14)  
plt.title('Tamil Nadu Annual Rainfall from Year 1901 to 2015',size=20)  
plt.grid()  
plt.show()
```



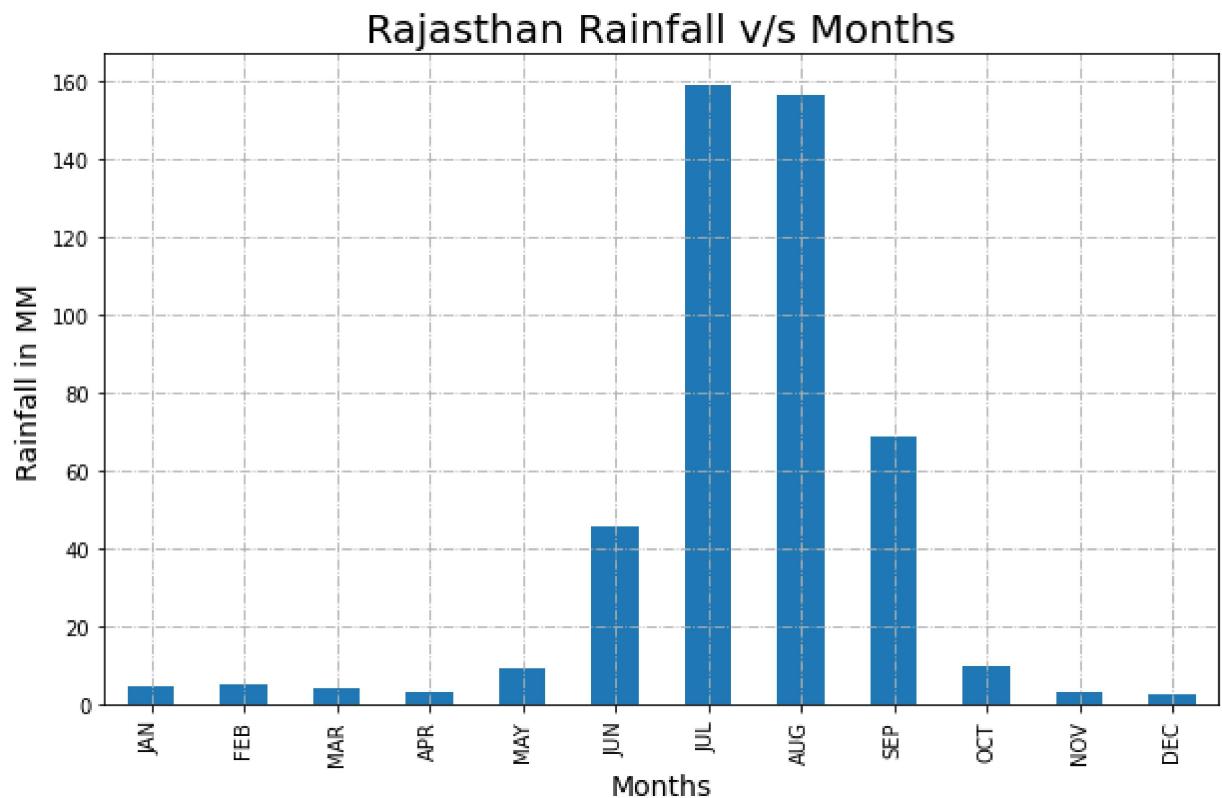
```
In [24]: Rajasthan = data.loc[((data['SUBDIVISION'] == 'WEST RAJASTHAN') | (data['SUBDIVISION'] == 'RAJASTHAN')) & (data['YEAR'] >= 1900) & (data['YEAR'] <= 1905)]  
Rajasthan.head()
```

Out[24]:

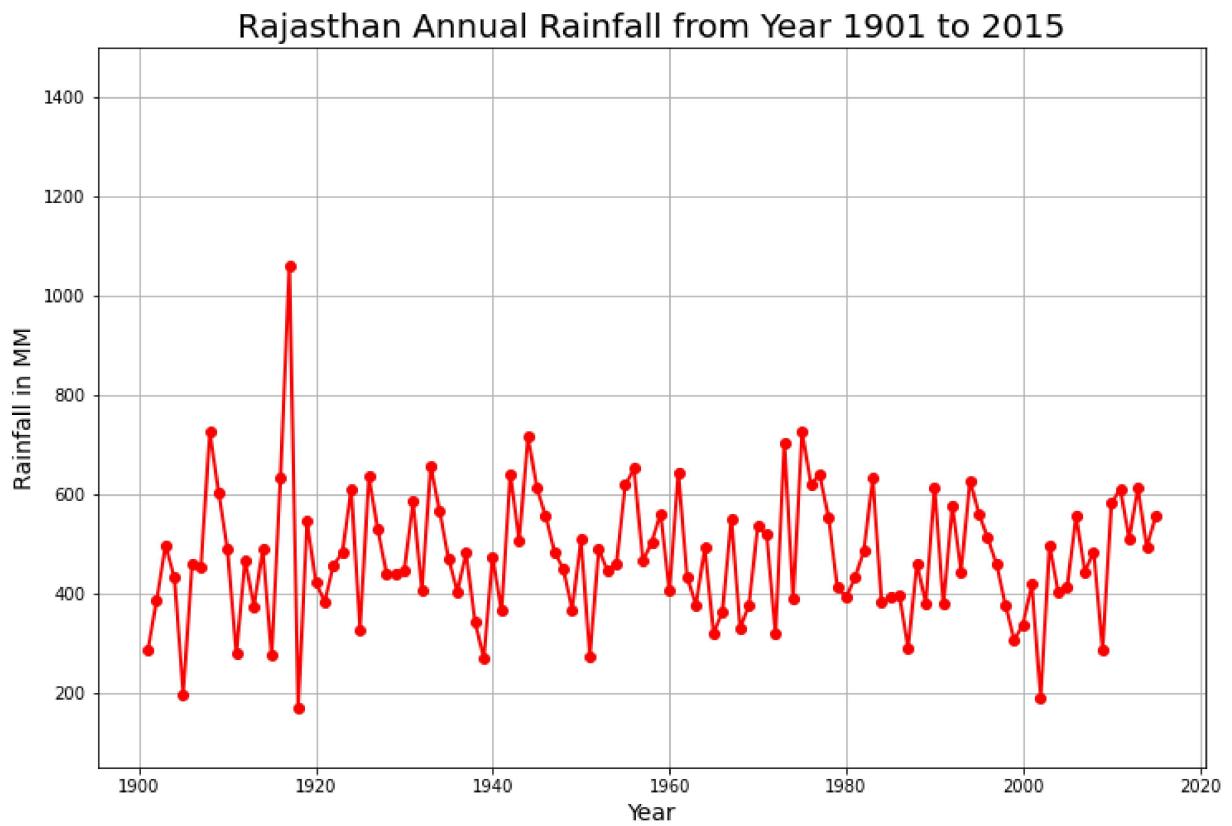
	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
1817	WEST RAJASTHAN	1901	6.7	0.0	1.1	0.0	6.1	3.0	79.0	59.2	1.0	2.1	0.0	0
1818	WEST RAJASTHAN	1902	0.0	0.0	0.0	0.5	4.0	49.1	27.0	71.3	41.8	1.8	0.0	0
1819	WEST RAJASTHAN	1903	1.7	1.3	5.5	0.0	4.2	2.7	154.8	87.1	49.3	0.1	0.0	0
1820	WEST RAJASTHAN	1904	3.8	2.9	16.3	0.7	11.4	14.6	39.8	45.6	21.4	1.4	2.9	7
1821	WEST RAJASTHAN	1905	6.3	4.8	0.7	1.3	0.3	4.9	30.1	0.6	64.5	0.0	0.0	0



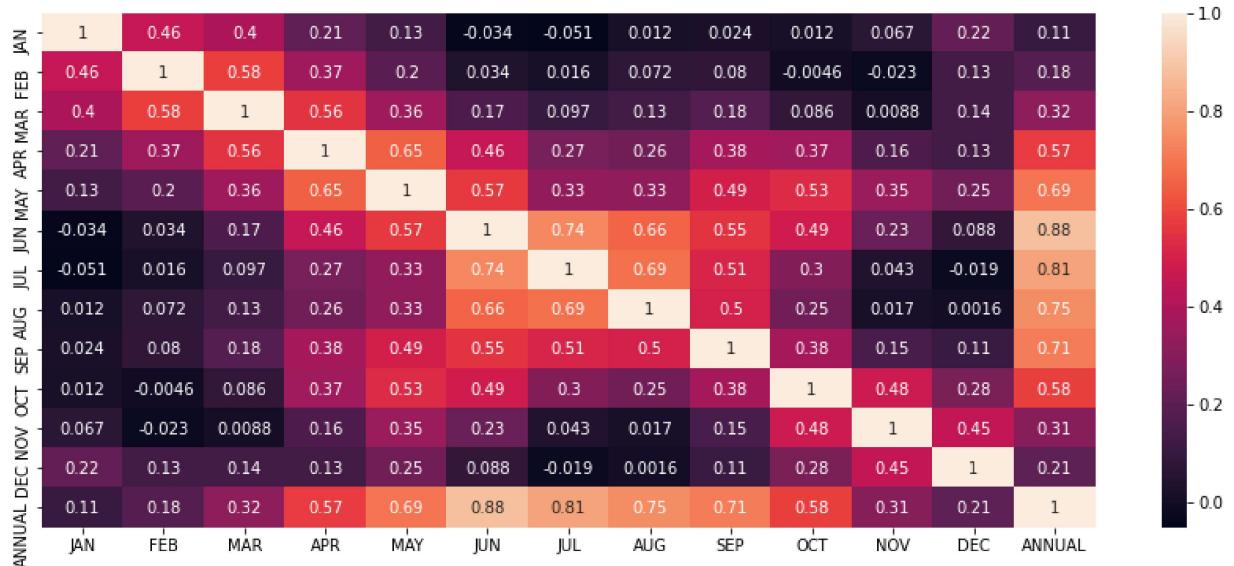
```
In [25]: plt.figure(figsize=(10,6))
Rajasthan[['JAN', 'FEB', 'MAR', 'APR','MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']]
plt.title("Rajasthan Rainfall v/s Months",size=20)
plt.xlabel("Months",size=14)
plt.ylabel("Rainfall in MM",size=14)
plt.grid(axis="both",linestyle="- .")
plt.show()
```



```
In [26]: Rajasthan.groupby("YEAR").mean()['ANNUAL'].plot(ylim=(50,1500),color='r',marker='o',  
plt.xlabel('Year',size=14)  
plt.ylabel('Rainfall in MM',size=14)  
plt.title('Rajasthan Annual Rainfall from Year 1901 to 2015',size=20)  
plt.grid()  
plt.show()
```



```
In [27]: plt.figure(figsize=(15,6))
sns.heatmap(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC', 'ANNUAL']])
plt.show()
```



```
In [28]: data["SUBDIVISION"].nunique()
```

Out[28]: 36

```
In [29]: group = data.groupby('SUBDIVISION')[['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC']]
data=group.get_group(('TAMIL NADU'))
data.head()
```

```
<ipython-input-29-e6f955922236>:1: FutureWarning: Indexing with multiple keys
(implicitly converted to a tuple of keys) will be deprecated, use a list instead.
group = data.groupby(['SUBDIVISION'])[['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC']]
```

Out[29]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
3427	1901	24.5	39.1	21.7	36.0	74.0	41.8	49.3	67.9	191.1	122.3	212.3	80.4
3428	1902	67.2	9.8	25.1	21.9	84.7	39.3	55.1	113.8	98.6	282.2	174.9	165.8
3429	1903	19.3	7.8	1.7	18.2	128.5	58.5	72.6	115.0	210.4	128.1	200.5	203.2
3430	1904	35.2	0.1	0.7	19.5	121.9	34.9	89.0	40.4	85.7	163.2	23.6	49.1
3431	1905	6.5	7.5	17.2	64.8	83.7	49.8	39.0	101.8	73.5	250.4	123.7	3.2

```
In [30]: df=data.melt(['YEAR']).reset_index()
df.head()
```

Out[30]:

	index	YEAR	variable	value
0	0	1901	JAN	24.5
1	1	1902	JAN	67.2
2	2	1903	JAN	19.3
3	3	1904	JAN	35.2
4	4	1905	JAN	6.5

```
In [31]: df=df[['YEAR','variable','value']].reset_index().sort_values(by=['YEAR','index'])
df.head()
```

Out[31]:

	index	YEAR	variable	value
0	0	1901	JAN	24.5
115	115	1901	FEB	39.1
230	230	1901	MAR	21.7
345	345	1901	APR	36.0
460	460	1901	MAY	74.0

```
In [32]: df.YEAR.unique()
```

```
Out[32]: array([1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911,
    1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922,
    1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933,
    1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944,
    1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955,
    1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966,
    1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977,
    1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988,
    1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999,
    2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
    2011, 2012, 2013, 2014, 2015], dtype=int64)
```

```
In [33]: df.columns=['Index','Year','Month','Avg_Rainfall']
```

```
In [34]: df.head()
```

Out[34]:

	Index	Year	Month	Avg_Rainfall
0	0	1901	JAN	24.5
115	115	1901	FEB	39.1
230	230	1901	MAR	21.7
345	345	1901	APR	36.0
460	460	1901	MAY	74.0

```
In [35]: 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[35]:

	Index	Year	Month	Avg_Rainfall
0	0	1901	1	24.5
115	115	1901	2	39.1
230	230	1901	3	21.7
345	345	1901	4	36.0
460	460	1901	5	74.0
575	575	1901	6	41.8
690	690	1901	7	49.3
805	805	1901	8	67.9
920	920	1901	9	191.1
1035	1035	1901	10	122.3
1150	1150	1901	11	212.3
1265	1265	1901	12	80.4

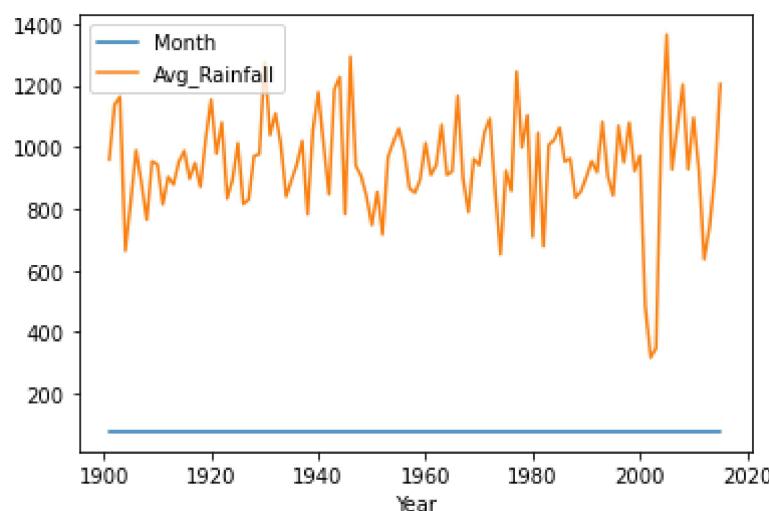
```
In [36]: df.drop(columns="Index",inplace=True)
```

```
In [37]: df.head(2)
```

Out[37]:

	Year	Month	Avg_Rainfall
0	1901	1	24.5
115	1901	2	39.1

```
In [38]: df.groupby("Year").sum().plot()  
plt.show()
```



```
In [39]: X=np.asarray(df[['Year','Month']]).astype('int')  
y=np.asarray(df['Avg_Rainfall']).astype('int')  
print(X.shape)  
print(y.shape)
```

```
(1380, 2)  
(1380,)
```

```
In [40]: 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, random_s
```

Linear Regression Model

```
In [41]: #Linear Regression Model  
from sklearn.linear_model import LinearRegression  
LR = LinearRegression()  
LR.fit(X_train,y_train)
```

```
Out[41]: LinearRegression()
```

```
In [42]: ## predicting  
y_train_predict=LR.predict(X_train)  
y_test_predict=LR.predict(X_test)
```

```
In [69]: #Linear Regression Model
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)))

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)
```

-----Test Data-----

MAE: 34.123460073090925
 MSE: 2342.263355401779
 RMSE: 48.396935392664886

-----Train Data-----

MAE: 25.951357001311507
 MSE: 1461.8265178535378
 RMSE: 38.233839956948316

-----Training Accuracy-----

41.699999999999996

-----Testing Accuracy-----

33.1

Lasso Model

```
In [44]: # Lasso Model
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

# create a Lasso object
lasso = Lasso(max_iter=100000)

# 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 [45]: lasso_regressor.fit(X_train,y_train)
```

```
C:\Users\sampr\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 818339.1530143891, tolerance: 409.5182559585492
    model = cd_fast.enet_coordinate_descent(
C:\Users\sampr\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1037672.1927167852, tolerance: 413.1583712807245
    model = cd_fast.enet_coordinate_descent(
C:\Users\sampr\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 799164.0947291797, tolerance: 391.1758082794308
    model = cd_fast.enet_coordinate_descent(
C:\Users\sampr\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1048996.303884098, tolerance: 430.8688978007762
    model = cd_fast.enet_coordinate_descent(
C:\Users\sampr\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1089589.4842717876, tolerance: 417.9702812419146
    model = cd_fast.enet_coordinate_descent(
C:\Users\sampr\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:529: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1449147.9657562054, tolerance: 516.0125474120083
    model = cd_fast.enet_coordinate_descent(
```

```
Out[45]: GridSearchCV(cv=5, estimator=Lasso(max_iter=100000),
                      param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.001, 0.01, 1, 5, 10,
                                           100.0, 1000.0, 10000.0, 100000.0, 1000000.0,
                                           10000000.0]}, scoring='neg_mean_squared_error')
```

```
In [46]: print("Best Parameter for Lasso:",lasso_regressor.best_estimator_)
```

```
Best Parameter for Lasso: Lasso(alpha=1e-15, max_iter=100000)
```

```
In [48]: lasso=Lasso(alpha=100.0,max_iter=100000)
```

```
# fit into the object
lasso.fit(X_train,y_train)
```

```
Out[48]: Lasso(alpha=100.0, max_iter=100000)
```

```
In [49]: # predicting
y_train_predict=lasso.predict(X_train)
y_test_predict=lasso.predict(X_test)
```

```
In [50]: #Lasso regression
from sklearn import metrics
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)))

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)
```

```
-----Test Data-----
MAE: 41.77463317555058
MSE: 3011.482049035098
RMSE: 54.87697193755408

-----Train Data-----
MAE: 46.66768689446285
MSE: 3948.7608993489275
RMSE: 62.839166921187996

-----Training Accuracy-----
26.1
-----Testing Accuracy-----
25.6
```

Ridge Model

```
In [51]: # Ridge Model
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

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)

print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)

{'alpha': 1e-15}
-3139.0798658992653
```

```
In [52]: print("Best Parameter for Ridge:",ridge_regressor.best_estimator_)
```

```
Best Parameter for Ridge: Ridge(alpha=1e-15)
```

```
In [54]: ridge=Ridge(alpha=100.0)

# fit into the object
ridge.fit(X_train,y_train)
```

```
Out[54]: Ridge(alpha=100.0)
```

```
In [55]: # predicting
y_train_predict=ridge.predict(X_train)
y_test_predict=ridge.predict(X_test)
```

```
In [56]: # Ridge Model
from sklearn import metrics
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)))

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)
```

-----Test Data-----

MAE: 36.694264997117806

MSE: 2700.404122847211

RMSE: 51.96541275547815

-----Train Data-----

MAE: 37.71478463865123

MSE: 3113.4499194422324

RMSE: 55.798296743200254

-----Training Accuracy-----

41.69999999999996

-----Testing Accuracy-----

33.30000000000004

```
In [57]: #SVC
from sklearn import preprocessing
from sklearn import svm

svm_regr = svm.SVC(kernel='rbf')
svm_regr.fit(X_train, y_train)
```

Out[57]: SVC()

```
In [58]: y_test_predict = svm_regr.predict(X_test)
y_train_predict = svm_regr.predict(X_train)
```

In [59]:

```
#SVC
from sklearn import metrics
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)))

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

-----Test Data-----

```
MAE: 76.73671497584542
MSE: 9936.306763285023
RMSE: 99.68102509146374
```

-----Train Data-----

```
MAE: 78.82815734989649
MSE: 11555.623188405798
RMSE: 107.4970845577023
```

-----Training Accuracy-----

```
3.5000000000000004
```

-----Testing Accuracy-----

```
1.7000000000000002
```

Random Forest Model

In [60]:

```
# Random Forest Model
from sklearn.ensemble import RandomForestRegressor
random_forest_model = RandomForestRegressor(max_depth=100, max_features='sqrt',
                                             min_samples_split=10, n_estimators=800)
random_forest_model.fit(X_train, y_train)
```

Out[60]:

```
RandomForestRegressor(max_depth=100, max_features='sqrt', min_samples_leaf=4,
                      min_samples_split=10, n_estimators=800)
```

In [61]:

```
y_train_predict=random_forest_model.predict(X_train)
y_test_predict=random_forest_model.predict(X_test)
```

```
In [62]: # Random Forest Model
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: 34.123460073090925
MSE: 2342.263355401779
RMSE: 48.396935392664886

-----Train Data-----
MAE: 25.951357001311507
MSE: 1461.8265178535378
RMSE: 38.233839956948316
```

```
In [63]: 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)
```

```
-----Training Accuracy-----
72.6
-----Testing Accuracy-----
42.1
```

```
In [64]: predicted = random_forest_model.predict([[2016,11]])
```

```
In [65]: predicted
```

```
Out[65]: array([151.63271319])
```

```
In [ ]:
```