```
import numpy as np
import pandas as pd
df = pd.read_csv('/content/kerala.csv')
```

```
df.head()
```

```
   SUBDIVISION  YEAR   JAN   FEB   MAR    APR    MAY     JUN     JUL    AUG    SEP    OCT    NOV    DEC  ANNUAL RAINFALL  FLOODS
0       KERALA  1901  28.7  44.7  51.6  160.0  174.7   824.6   743.0  357.5  197.7  266.9  350.8   48.4           3248.6     YES
1       KERALA  1902   6.7   2.6  57.3   83.9  134.5   390.9  1205.0  315.8  491.6  358.4  158.3  121.5           3326.6     YES
2       KERALA  1903   3.2  18.6   3.1   83.6  249.7   558.6  1022.5  420.2  341.8  354.1  157.0   59.0           3271.2     YES
3       KERALA  1904  23.7   3.0  32.2   71.5  235.7  1098.2   725.5  351.8  222.7  328.1   33.9    3.3           3129.7     YES
4       KERALA  1905   1.2  22.3   9.4  105.9  263.3   850.2   520.5  293.6  217.2  383.5   74.4    0.2           2741.6      NO
```

```
df.shape
```

```
(118, 16)
```

```
# Calculating the average rainfall for each month
cols = ['JAN','FEB','MAR','APR','MAY','JUN','JUL','AUG','SEP','OCT','NOV','DEC']
monthly_avg_rainfall=df[cols].mean()
monthly_avg_rainfall
```

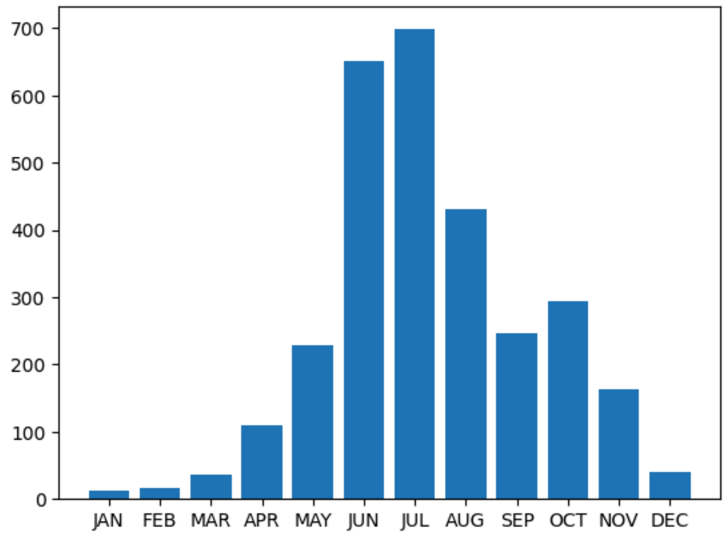
```
              0
JAN   12.218644
FEB   15.633898
MAR   36.670339
APR  110.330508
MAY  228.644915
JUN  651.617797
JUL  698.220339
AUG  430.369492
SEP  246.207627
OCT  293.207627
NOV  162.311017
DEC   40.009322

dtype: float64
```

```
import matplotlib.pyplot as plt
import seaborn as sns

x = monthly_avg_rainfall.index
y = monthly_avg_rainfall
plt.bar(x,y)
```

```
<BarContainer object of 12 artists>
```



We can make few **conclusions** here:

- The data reveals significant seasonal variation in rainfall.
    - **June and July** have the **highest average rainfall**, while **January and February** are the driest months
    - The rainfall in **August and September** is still relatively high but begins to decline
    - Surprisingly, **October** has a **higher average rainfall than September**, which may seem counterintuitive.

There are two monsoon seasons in Kerala, **one during Jun-Aug, Other during Oct**.

the important features in this dataset are "JUN", "JUL", "OCT" , "ANNAUL_RAINFALL", "FLOODS"

because in these months only we have seen the peak of the rainfall which can be one of the major source of causing the flood

```
df.columns
```

```
Index(['SUBDIVISION', 'YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC', ' ANNUAL RAINFALL', 'FLOODS'],
      dtype='object')
```

```
df.columns = [c.replace(' ANNUAL RAINFALL', 'ANNUAL_RAINFALL') for c in df.columns]
```

```
df.head()
```

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL_RAINFALL	FLOODS
0	KERALA	1901	28.7	44.7	51.6	160.0	174.7	824.6	743.0	357.5	197.7	266.9	350.8	48.4	3248.6	YES
1	KERALA	1902	6.7	2.6	57.3	83.9	134.5	390.9	1205.0	315.8	491.6	358.4	158.3	121.5	3326.6	YES
2	KERALA	1903	3.2	18.6	3.1	83.6	249.7	558.6	1022.5	420.2	341.8	354.1	157.0	59.0	3271.2	YES
3	KERALA	1904	23.7	3.0	32.2	71.5	235.7	1098.2	725.5	351.8	222.7	328.1	33.9	3.3	3129.7	YES
4	KERALA	1905	1.2	22.3	9.4	105.9	263.3	850.2	520.5	293.6	217.2	383.5	74.4	0.2	2741.6	NO

```
impactful_columns = ['YEAR', 'JUN', 'JUL', 'OCT', 'ANNUAL_RAINFALL', 'FLOODS']
df[impactful_columns]
```

	YEAR	JUN	JUL	OCT	ANNUAL_RAINFALL	FLOODS
0	1901	824.6	743.0	266.9	3248.6	YES
1	1902	390.9	1205.0	358.4	3326.6	YES
2	1903	558.6	1022.5	354.1	3271.2	YES
3	1904	1098.2	725.5	328.1	3129.7	YES
4	1905	850.2	520.5	383.5	2741.6	NO
...	...	...	...	...	...	...
113	2014	454.4	677.8	355.5	3046.4	YES
114	2015	563.6	406.0	308.1	2600.6	NO
115	2016	522.2	412.3	225.9	2176.6	NO
116	2017	498.5	319.6	192.4	2117.1	NO
117	2018	625.4	1048.5	356.1	4473.0	YES

118 rows × 6 columns

> **Q. But how much rainfall index is considered as a heavy ranifall?**

One of the parameter is using the **Median** values of these columns.

If their individual **rainfall index value > median value** then it'll we considered as **heavy rainfall** and vice a versa

```
data = df[impactful_columns]

data.head()
```

	YEAR	JUN	JUL	OCT	ANNUAL_RAINFALL	FLOODS
0	1901	824.6	743.0	266.9	3248.6	YES
1	1902	390.9	1205.0	358.4	3326.6	YES
2	1903	558.6	1022.5	354.1	3271.2	YES
3	1904	1098.2	725.5	328.1	3129.7	YES
4	1905	850.2	520.5	383.5	2741.6	NO

```
threshold_jun = data['JUN'].median().astype(int)
threshold_jul = data['JUL'].median().astype(int)
threshold_oct = data['OCT'].median().astype(int)
threshold_ar = data['ANNUAL_RAINFALL'].median().astype(int)

threshold_jun, threshold_jul, threshold_oct, threshold_ar
```

```
(625, 691, 284, 2934)
```

```
thresholds = {
    'JUN': 625,
    'JUL': 691,
    'OCT': 284,
    'ANNUAL_RAINFALL': 2934
}
```

```
for col, threshold in thresholds.items():
    data[col] = (data[col] > threshold).astype(int)
```

```
data.head()
```

```
<ipython-input-33-b28e5c1ba0b8>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data[col] = (data[col] > threshold).astype(int)
<ipython-input-33-b28e5c1ba0b8>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data[col] = (data[col] > threshold).astype(int)
<ipython-input-33-b28e5c1ba0b8>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data[col] = (data[col] > threshold).astype(int)
<ipython-input-33-b28e5c1ba0b8>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data[col] = (data[col] > threshold).astype(int)
```

	YEAR	JUN	JUL	OCT	ANNUAL_RAINFALL	FLOODS
0	1901	1	1	0	1	YES
1	1902	0	1	1	1	YES
2	1903	0	1	1	1	YES
3	1904	1	1	1	1	YES
4	1905	1	0	1	0	NO

```
data['FLOODS'].unique()
```

```
array(['YES', 'NO'], dtype=object)
```

Q1. Calculate the Probability of flood given that rainfall in June is greater than the median june rainfall value (threshold for heavy rainfall)

Question Explanation:

Let A represents : Flood
B represents: heavy rain in June

We need to calculate $P(A|B)$ i.e. $\frac{P(A \cap B)}{P(B)}$

```
pd.crosstab(data['JUN'],
            data['FLOODS'],
            margins=True,
            margins_name='Total')
```

FLOODS	NO	YES	Total
JUN			
0	42	16	58
1	16	44	60
Total	58	60	118

Now, $P(A \cap B)$ = Probability of Flood occuring `AND` heavy rainfall in JUNE

As we know in the contingency table, FLOODS = YES represents that flood has occured and JUN = 1 means heavy rainfall.

We need to check value where FLOODS = YES and JUN = 1 which is **44**

Then by the formula of condititonal probability we can feed this data

```
# probability of high rainfall in June P(J)
# P(J) = possible outcomes in june having heavy rainfall / total outcomes
```

```
P_J = (16+44)/(42+16+16+44)

# now, P(A and B) (Flood = YES and Jun = 1)

P_F_and_J = 44/(42+16+16+44)

#, so our probability of flood occurring given that the high rainfall occured in June will be

P_F_J = P_F_and_J / P_J

print(f'P(J) : {P_J}')
print(f'P(F AND J) : {P_F_and_J}')
print(f'P(F|J): {P_F_J}')
```

```
P(J) : 0.5084745762711864
P(F AND J) : 0.3728813559322034
P(F|J): 0.7333333333333334
```

Approach 2: using normalize attribute

Explanation of Normalize attribute:

Rather putting all the values in the formula and then calculate the probability

We can just pass one more attribute in pd.crosstab() function which will divide all values by the sum of values.

- This is the probability only, as in probability we divide `possible outcome / total outcome (sum of all values)`

Parameter is : **normalize = ' '**

- Without this attribute, the contingency table will show the raw counts of occurrences for each combination of variables.
- It will not be normalized, and the values in the table will represent counts.

Here we can pass these strings in this attribute:

normalize='index' or **normalize='columns'** :

- The normalize attribute specifies how the values in the contingency table should be normalized.
  - When set to **'index'**, it calculates conditional probabilities based on rows, treating each row as a separate condition.
  - When set to **'columns'**, it calculates conditional probabilities based on columns, treating each column as the condition we are focusing on.
- This means that each row in the table is divided by the sum of its row, making each row's values sum up to 1, representing conditional probabilities.

Same with the column

In this case:

By setting **normalize='index'**,

- the code calculates conditional probabilities within each row.
- Each value in the table represents the probability of the corresponding event (FLOODS) given the value of 'JUN' in that row.

The row sums up to 1, ensuring that it reflects the conditional probabilities.

In summary,

setting normalize='index' in pd.crosstab allows you to calculate and visualize conditional probabilities based on the specified row variable ('JUN' in this case),

making it easier to assess the impact of one variable on another.

```
pd.crosstab(index = data['JUN'],
            columns = data['FLOODS'],
            margins=True,
            normalize='index')
```

FLOODS	NO	YES
JUN		
0	0.724138	0.275862
1	0.266667	0.733333
All	0.491525	0.508475

The values in the table represent the conditional probabilities, where each cell contains the probability of the corresponding outcome (FLOODS) given the condition in June (JUN).

> Then the probability of flood occurring given that the heavy rainfall occured in June will be:

- In the cell at row 1, column 1, the value **0.73333** represents the conditional probability of flooding (FLOODS = YES) given that high rainfall occurred in June (JUN = 1).

Conclusion:

```
So, there is 73.33% chance of Floods when there is a heavy rainfall in June
```

As we can see by calculating using formula also, we are getting the same answer as using directly conditional probability using normalize = 'index'

Q2. Given that there is a flooding, calculate the probability that heavy rainfall has occurred in July (more than threshold value)?

Here we want to find $P(July = 1|Flood = YES)$

We are already aware of using formula based approach, so We will solve this using contingency table

Before proceeding,

> **Q. In this question, which string will be passed inside normalize=' ' attribute? 'index' or 'columns'**

In this question, we should normalize the contingency table along the columns

- As we want to find the conditional probability of **high rainfall in July (JUL = 1) given that there was flooding (FLOODS = YES)**,

We want to see how the 'JUL' column behaves when there is flooding.

```
pd.crosstab(index = data['JUL'],
            columns = data['FLOODS'],
            margins=True,
            normalize='columns')
```

FLOODS	NO	YES	All
JUL			
0	0.655172	0.35	0.5
1	0.344828	0.65	0.5

Conclusion:

The probability that high rainfall occurred in July (JUL = 1) given flooding (FLOODS = YES) is **0.65**.

- **This means that when there is flooding, there is a 65% chance of heavy rainfall in July.**

Q3.Calculate the probability of flood given that june and july rainfall was greater than their median rainfall value?

Solution:

We want to find $P(Flood = Yes|\ june = 1\ and\ Jul = 1)$

Here, we can pass multiple columns in the **pd.crosstab()**

```
pd.crosstab(index = [data['JUN'], data['JUL']],
            columns = data['FLOODS'],
            margins=True,
            normalize='index')
```

	FLOODS	NO	YES
JUN	JUL		
0	**0**	0.862069	0.137931
	1	0.586207	0.413793
1	**0**	0.433333	0.566667