Project 10.1:Data Visualization with Accuweather website.

In this part of the project we are going to visualize the data gathered from the **accuweather.com** website. The **first step is to gather the data.** So let's see how we are doing that.

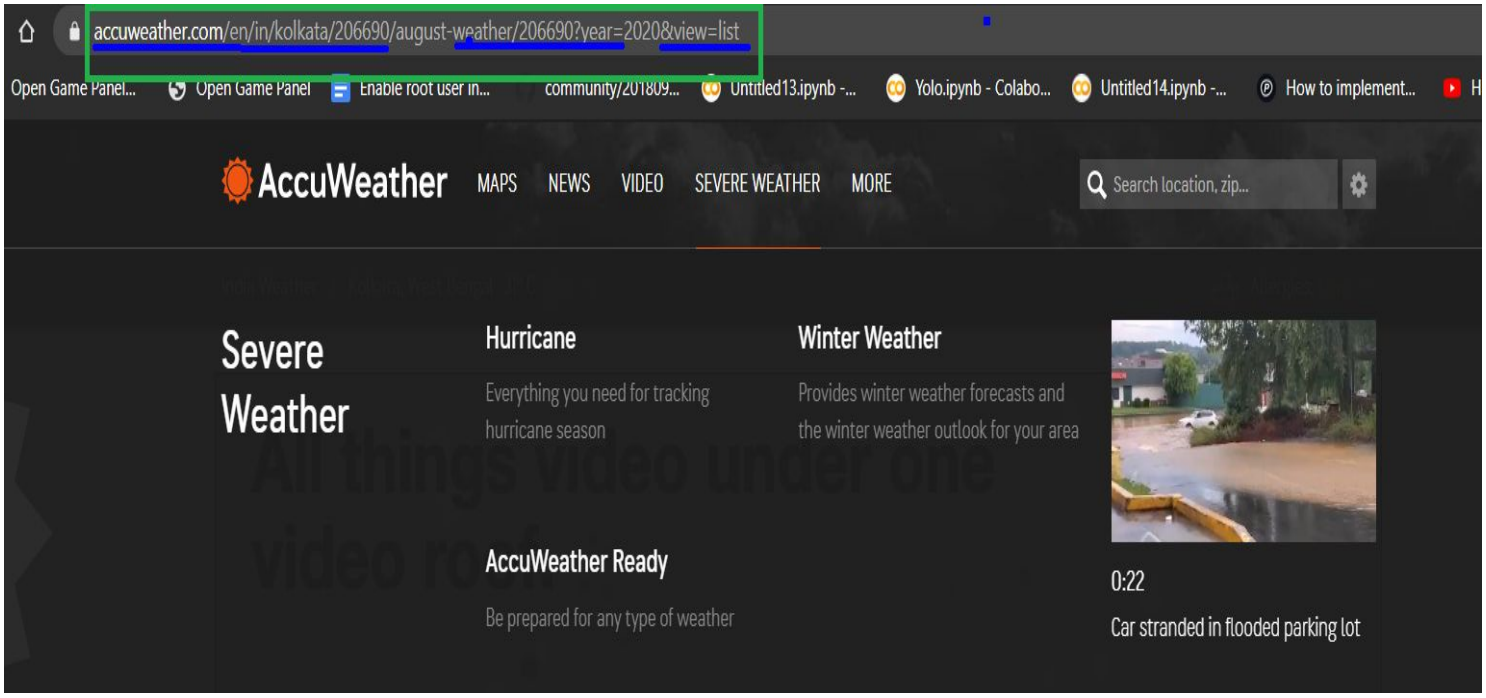
The first thing is to **open the browser**. That can be done by running the following code.

```
from selenium import webdriver
import pandas as pd



cd='C:\\webdrivers\\chromedriver.exe'
chrome_browser = webdriver.Chrome(cd)
```

Here the first 3 lines of code are used to import the important libraries.

The next 2 lines of code are used to open the browser. Here cd specifies the path to my chromedriver.exe file. This is supposed to be changed by your path.

Now once the browser has opened, we need **to open the website**. For this let's look at the link to the page where we are interested.



Here, the part of the url highlighted by blue is going to be constant for kolkata and the month and year will be replaced by the desired month and year.

So to get the url we must have the month and year with us. Let's **take them as input.** To do this, run the following code.

```
month=input('Enter the month in small letters : ')
year=input('Enter the year : ')
```

Here the month and year given by the user as input are stored in the variables month and year respectively.

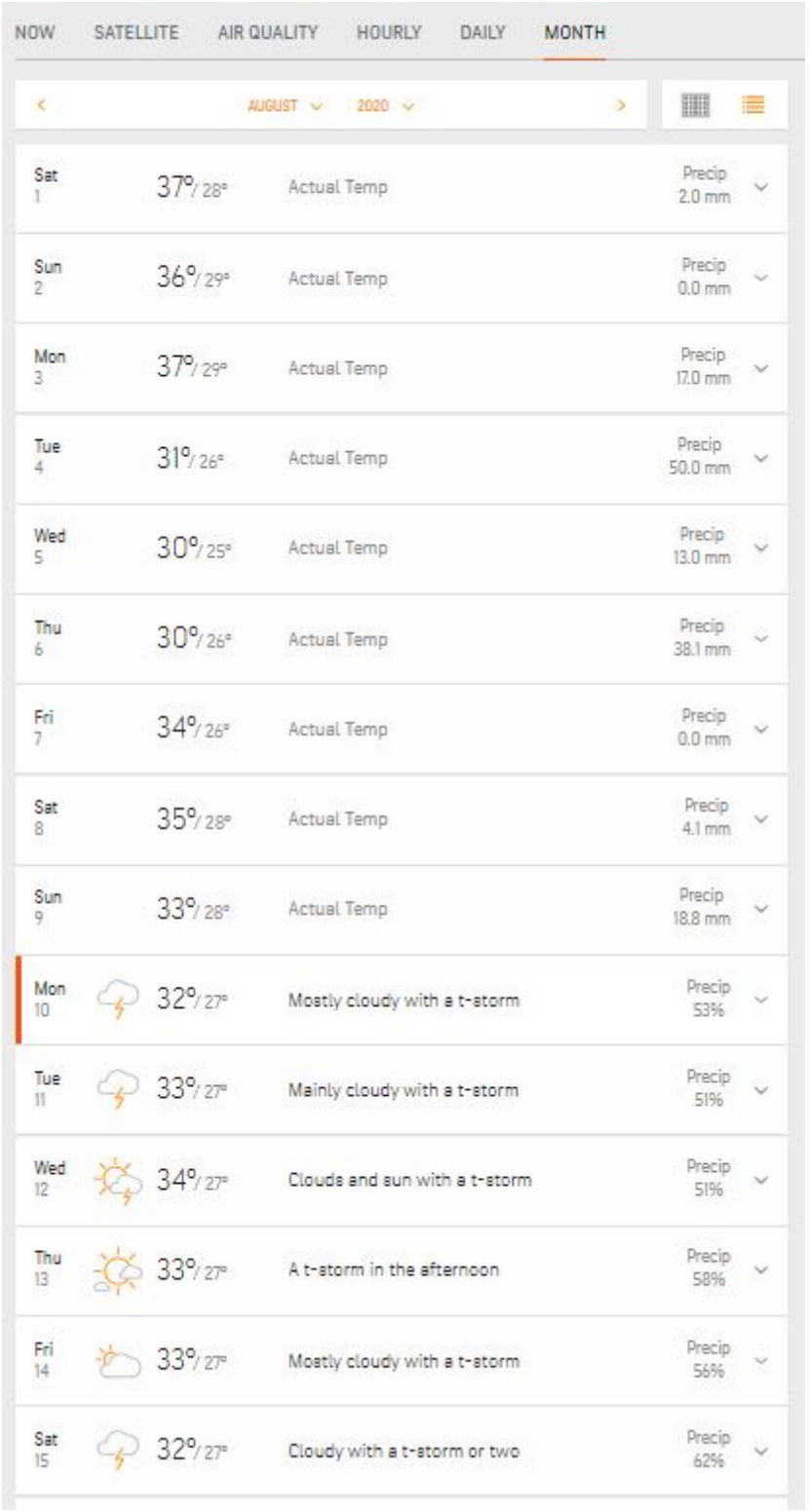
Now we should create the url. We just need to append the static part with these variables. For doing that, run the following code.

```
url='https://www.accuweather.com/en/in/kolkata/206690/'+month+'-weather/206690?year='+year+'&view=list'
```

Here we will have to change the url for any other city as it also has a code for different places which is unknown to us. Once we get the url, we can open the page. To do that run the following code:

```
chrome_browser.get(url)
```

Once you run this code, the page will appear on your browser. Now we are interested in the following table.



NOW SATELLITE AIR QUALITY HOURLY DAILY MONTH

AUGUST 2020

Sat 1	37°/ 28°	Actual Temp	Precip 2.0 mm
Sun 2	36°/ 29°	Actual Temp	Precip 0.0 mm
Mon 3	37°/ 29°	Actual Temp	Precip 17.0 mm
Tue 4	31°/ 26°	Actual Temp	Precip 50.0 mm
Wed 5	30°/ 25°	Actual Temp	Precip 13.0 mm
Thu 6	30°/ 26°	Actual Temp	Precip 38.1 mm
Fri 7	34°/ 26°	Actual Temp	Precip 0.0 mm
Sat 8	35°/ 28°	Actual Temp	Precip 4.1 mm
Sun 9	33°/ 28°	Actual Temp	Precip 18.8 mm
Mon 10	32°/ 27°	Mostly cloudy with a t-storm	Precip 53%
Tue 11	33°/ 27°	Mainly cloudy with a t-storm	Precip 51%
Wed 12	34°/ 27°	Clouds and sun with a t-storm	Precip 51%
Thu 13	33°/ 27°	A t-storm in the afternoon	Precip 58%
Fri 14	33°/ 27°	Mostly cloudy with a t-storm	Precip 56%
Sat 15	32°/ 27°	Cloudy with a t-storm or two	Precip 62%

Now we need to get these elements from our program. For that we need to find how these elements are stored in the source code. If I inspect the page to see how these elements are stored, I get this:

This means the highest temperature is stored as **a span element with class name 'high'.** Now we want to get all the high temperatures present in the page and store them in a list. And to do that, we will run the following code:

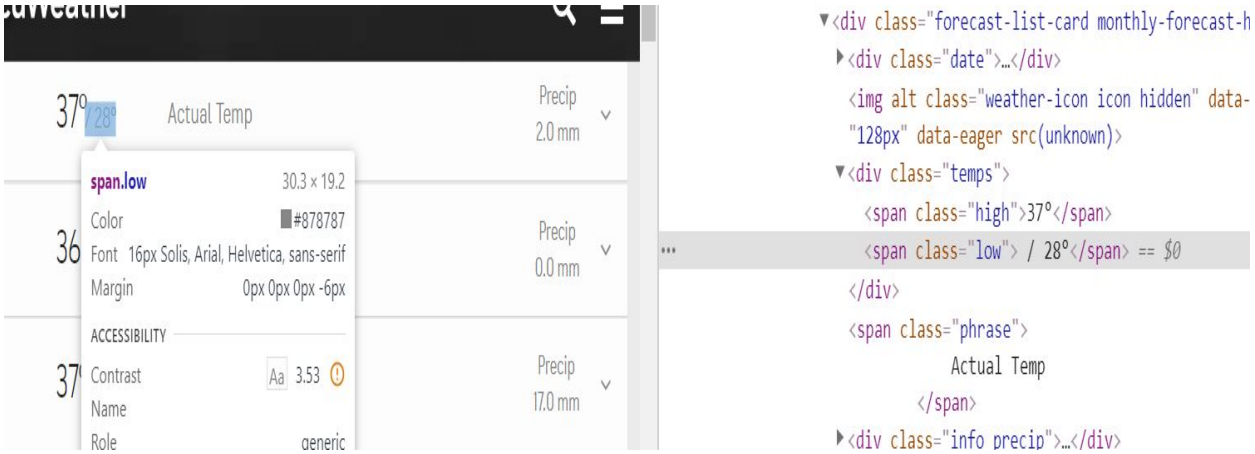
```
data=chrome_browser.find_elements_by_class_name("high")
```

This line of code will return all the elements with the class name high and all the elements are stored inside data. Now I want to extract the text content for all these elements. Now whenever I want to do something for a sequence of that, I need to do that inside a **for loop.** Now to do the same, run the following code:

```
temp_high=[]
for i in data:
    t=i.get_attribute('textContent')
    temp_high.append(int(t[:2]))
```

In the last line we are using slicing so as to include only the numerical characters and then convert them into integers. **For example: 37° will become 37.** With this, temp_high will have all the high temperature values present in the page in integer form.

Now we need to store the low temperatures as well. We will do this in the same manner only the class name will be changed.



Here we are seeing that the lower temperature is stored as **a span element with class name 'low'**. So to get these, we run the following code:

```
data=chrome_browser.find_elements_by_class_name("low")#Temp low
temp_low=[]
for i in data:
    t=i.get_attribute('textContent')
    tt=int(t[3:5])
    temp_low.append(tt)
```

Here everything is the same except the slicing. We are slicing just to get the numerical value out of the text content present there. **For example: ' / 28°' will be converted into 28.**
After running this, each of the low temperatures present there will be stored inside temp_low.

Now we need to store the information about the precipitation as well. So let's look at how this element is stored.

Here the precipitation is stored as **a p element.** But to recognize it, or to find it in the page, I will have to create a nested path starting from the **div element with class name 'info precip'.** So to work with this data, we will have to run the following code.

```
data=chrome_browser.find_elements_by_xpath("//div[@class='info precip']/p[2]")
```

```
precip=[]
for i in data:
    t=i.get_attribute('textContent')
    precip.append(float(t[:2]))
```

Here the x path is given in a way to reach to the **2nd p element inside the div element with class name 'info precip'**

In the last line of the code, from the text only the numerical part is retrieved and converted into float. After running this code, the list **precip** will be holding the information about the precipitation present in the web page.

Now I have temp_high holding the high temperature, temp_low holding the low temperature & precip holding the precipitation information. Now I want to have the dates as well. That will be retrieved using the length and index of any of the above three list using the following code:

```
date=[]
for i in range(len(precip)):
    date.append(i+1)
```

Here I am running a for loop for a sequence whose length is equal to precip, which will be either 28 or 29 or 30 or 31. And inside the loop I am adding 1 with the index because the index starts with 0 and appending with the list date.

Now I want to create a **dictionary** which will be helping me to create the **dataframe.** For this I will be creating different keys which will be converted into the column names and assign them values using the 4 lists which I just created.

To do this, run the following code:

```
d={"Date":date,"High_temp":temp_high, "Low_temp":temp_low, "Precipitation": precip}
```

Now the only things left are creating and storing the dataframe. To create the dataframe, run the following code:

```
df=pd.DataFrame(d)
```

This will create a dataframe with name d, and will have column names Date, High_temp, Low_temp & Precipitation as these are the key names passed to the function pd.DataFrame.

Now the final step, Saving the dataframe to my device memory in csv format. To do this, run the following code:

```
df.to_csv("C:\\Users\\Sweta\\kolkata”+month+”.csv")
print("DONE")
```

Here, I am saving the dataset with the name **kolkataugust.csv** for month=august. And finally there is a print statement to show the completion of the code.

Now we will move to **the second step which is data visualization.**

For this we will move to **Google Colab.** Open google colab and create a new file. By default the file is with .ipynb extension. Once the notebook is created, connect it with google’s cloud. After this upload the dataset here. Once the dataset is uploaded we are ready to start the second part.

Now we need to read the dataset as a dataframe. For this we will have to take the help of pandas which is a python module for dataset analysis and manipulation. So to read the dataset, run the following code:

```
import pandas as pd
df=pd.read_csv("/content/kolkataaugust.csv")
df
```

This will read the dataset which is stored as a csv file and store it as a dataframe whose name is df.

Now for data visualization we are going to use the python module **matplotlib.** This module helps in data visualization and creating multiple (a

lot of) plots.

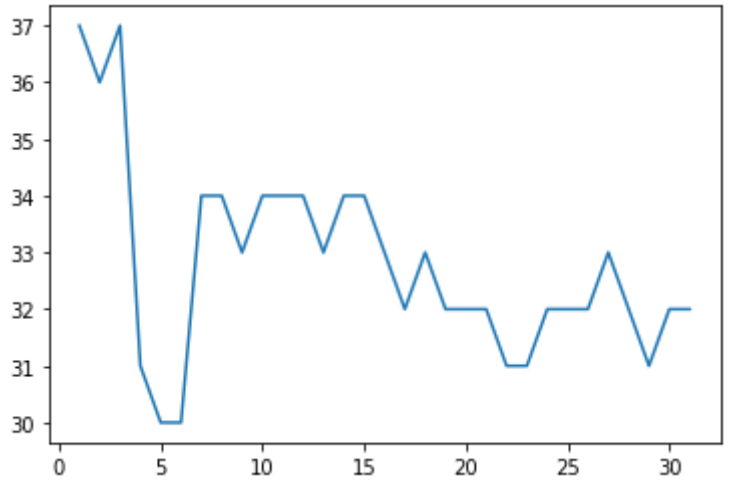
In google colab we don't need to install most of the modules, we can directly import them. So to use matplotlib, we need to **import the pyplot class of matplotlib.** To do this, run the following code:

```
import matplotlib.pyplot as plt
```

Now the first plot that we are going to look at is the line chart. Line charts are used to show resulting **data relative to a continuous variable** - most commonly time or money. Line charts are used to understand trends and fluctuations and to compare with different data or datasets. Here we are going to see how every day the high temperature fluctuated. To do this, we will be plotting a line chart. To do this, run the following code:

```
plt.plot(df['Date'],df['High_Temperature'])
```

This will create a line chart like this:

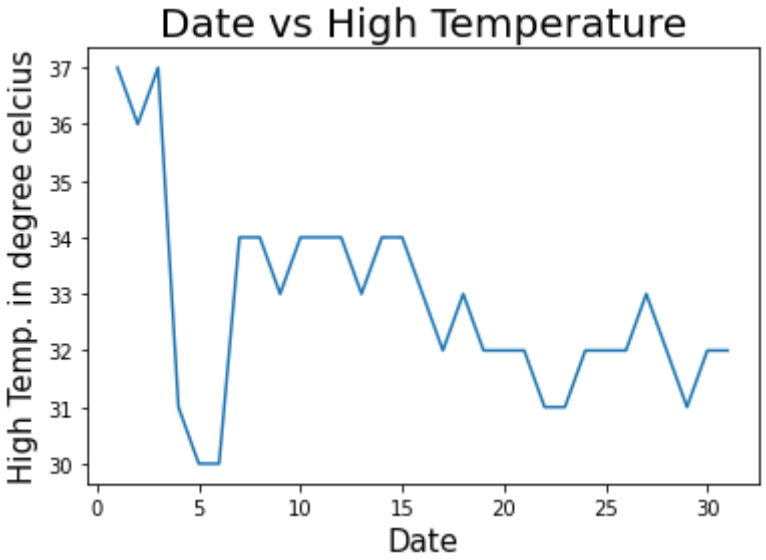


Here we can see that till the 3rd of the month, the temperature was pretty high. But on the 4th, there was a great fluctuation. After the 6th, the temperature rose again. But it was still lower than the temperature at the beginning of the month. From 6th to 16th, the temperature kept on increasing and decreasing but they were in the same range but after that again there was a noticeable difference in the temperature.

Now if you want to put labels and title to this plot, then that can be done by running the following code instead of the code given above:

```
plt.plot(df['Date'],df['High_Temperature'])
plt.title("Date vs High Temperature",fontsize=20)
plt.xlabel("Date",fontsize=15)
plt.ylabel("High Temp. in degree celcius",fontsize=15)
```

The output of this code will look like this:



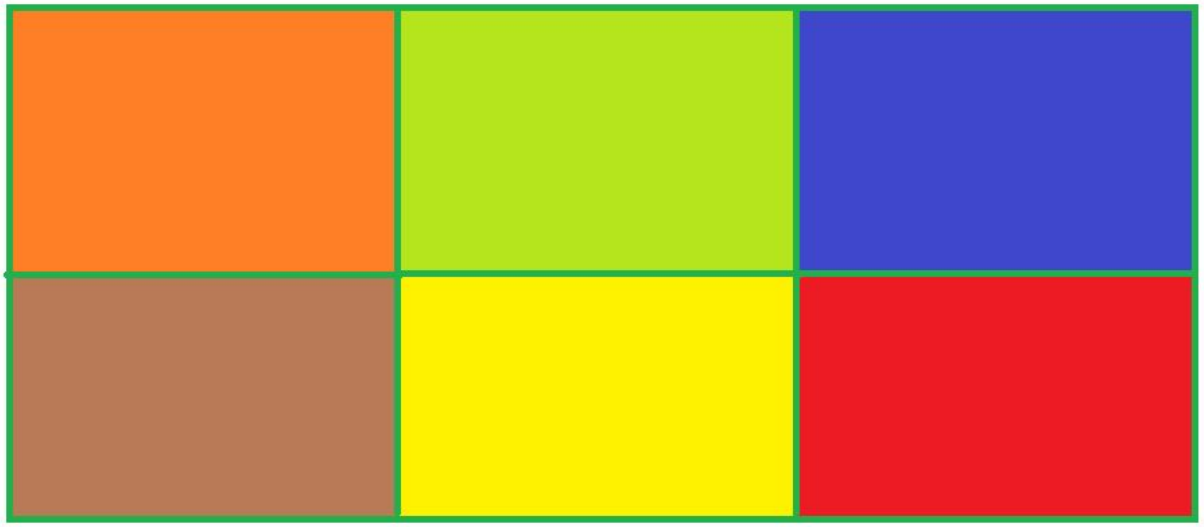
With proper title, and labels. That too in the font size that you wish.

So here using a line chart, we gathered information about the fluctuation of high temperature in the whole month.

We can create different line charts just by changing the arguments of the function **plt.plot.** Check out the code in github repository for reference.

Now what if you want to visualize 2 charts at the same time. Say you want to visualize the high temperature and low temperature both for the month. For such situations, **subplots** can help you.

Subplots let you plot multiple figures at the same time. To create a subplot, you just need to mention the number of columns and rows. For example, let's say you want to create a subplot like this:



Here you have 2 rows and 3 columns. So this thing needs to be mentioned inside **plt.subplot()** which is the function to create subplots using matplotlib. Here the syntax goes like this:

plt.sublot(number_rows,number_of_columns,index)

To understand it in a better way, let's look at an example:

For the figure given above, the subplots will be drawn like this:

plt.subplot(2,3,x)

Where x is the index of each of the plots present in the subplot(shown by different coloured boxes.

*For the box with **this colour** the index is **1**. And the function will be **plt.subplot(2,3,1).***

*For the box with **this colour** the index is **2**. And the function will be **plt.subplot(2,3,2).***

*For the box with **this colour** the index is **3**. And the function will be **plt.subplot(2,3,3).***

*For the box with **this colour** the index is **4**. And the function will be **plt.subplot(2,3,4).***

*For the box with **this colour** the index is **5**. And the function will be **plt.subplot(2,3,5).***

*For the box with **this colour** the index is **6**. And the function will be **plt.subplot(2,3,6).***

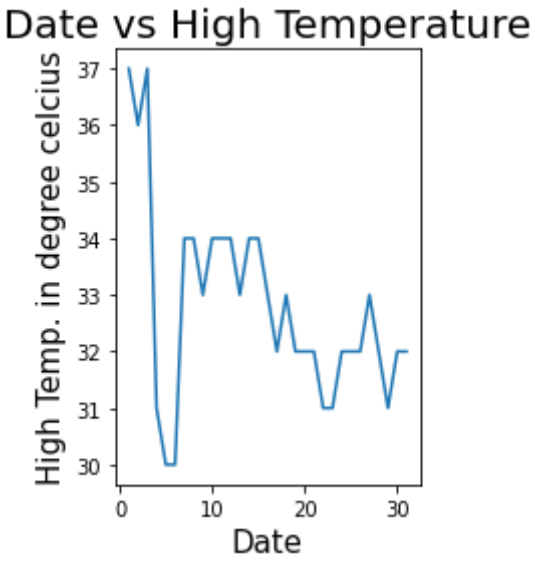
So coming back to our case where we want to visualize the high temperature and low temperature both with respect to the dates. As mentioned we will be using **plt.subplot(2,1,x)** as we want the plots to be one above the other. i.e. 2 rows and 1 columns **.** Here first of all we will be calling subplot for any one of the plots by mentioning the index, then after that we need to write the lines of code which are going to create the plot and then we need to call the function **plt.show()** which shows any created plot. This is important when you are working with multiple plots.

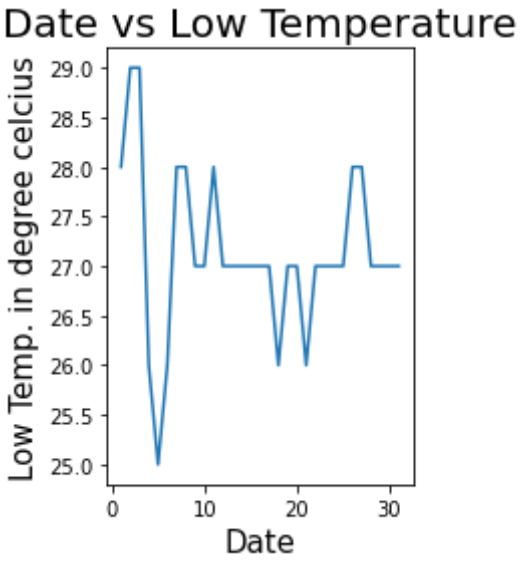
Now let's look at the code to create this subplot.

```
plt.subplot(2,1,1)
plt.plot(df['Date'],df['High_Temperature'])
plt.title("Date vs High Temperature",fontsize=20)
plt.xlabel("Date",fontsize=15)
plt.ylabel("High Temp. in degree celsius",fontsize=15)
plt.show()

plt.subplot(2,1,2)
plt.plot(df['Date'],df['Low_Temperature'])
plt.title("Date vs Low Temperature",fontsize=20)
plt.xlabel("Date",fontsize=15)
plt.ylabel("Low Temp. in degree celsius",fontsize=15)
plt.show()
```

Here **at the top we are plotting the plot for date vs high temperature** & **below that we are plotting the plot for date vs low temperature.**

The result will look like this:





Now let's take another example. Let's say you want to compare the fluctuation of precipitation for two cities, mumbai and kolkata. For this first of all you will have to create a dataset for mumbai. For this only **two things will be changed** in the first step. **1st will be the url and second will be the name of the new file.**

The url will be changed to
'https://www.accuweather.com/en/in/mumbai/204842/'+month+'-weather/204842?year='+year+'&view=list'

And the path will be changed to
"C:\\Users\\Sweta\\mumbai"+august+".csv"

So first of all we will have to create this dataset, save it and then upload it in colab. In colab read it as df1 using pd.read_csv.

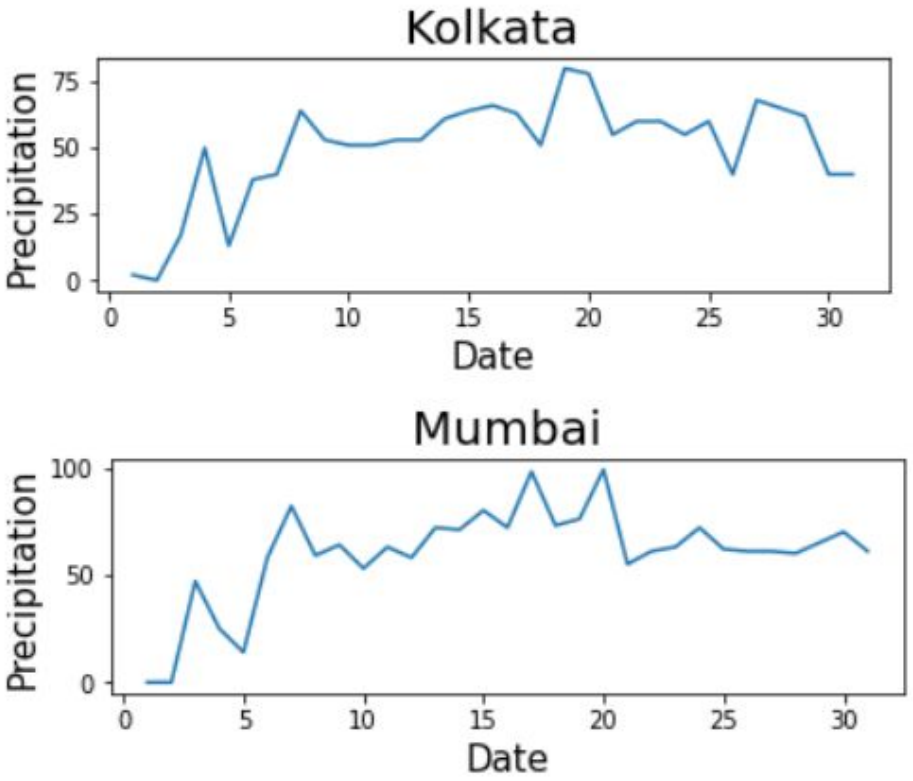
Now let's look at the code for the visualization part.

```
plt.subplot(2,1,1)
plt.plot(df['Date'],df['Precipitation'])
plt.title("Kolkata",fontsize=20)
plt.xlabel("Date",fontsize=15)
plt.ylabel("Precipitation",fontsize=15)
plt.show()
```

```
plt.subplot(2,1,2)
plt.plot(df1['Date'],df1['Precipitation'])
plt.title("Mumbai",fontsize=20)
plt.xlabel("Date",fontsize=15)
plt.ylabel("Precipitation",fontsize=15)
plt.show()
```

Here **at the top we are plotting the plot for date vs precipitation for Kolkata** & **below that we are plotting the plot for date vs precipitation for Mumbai.**

The result will look like this:

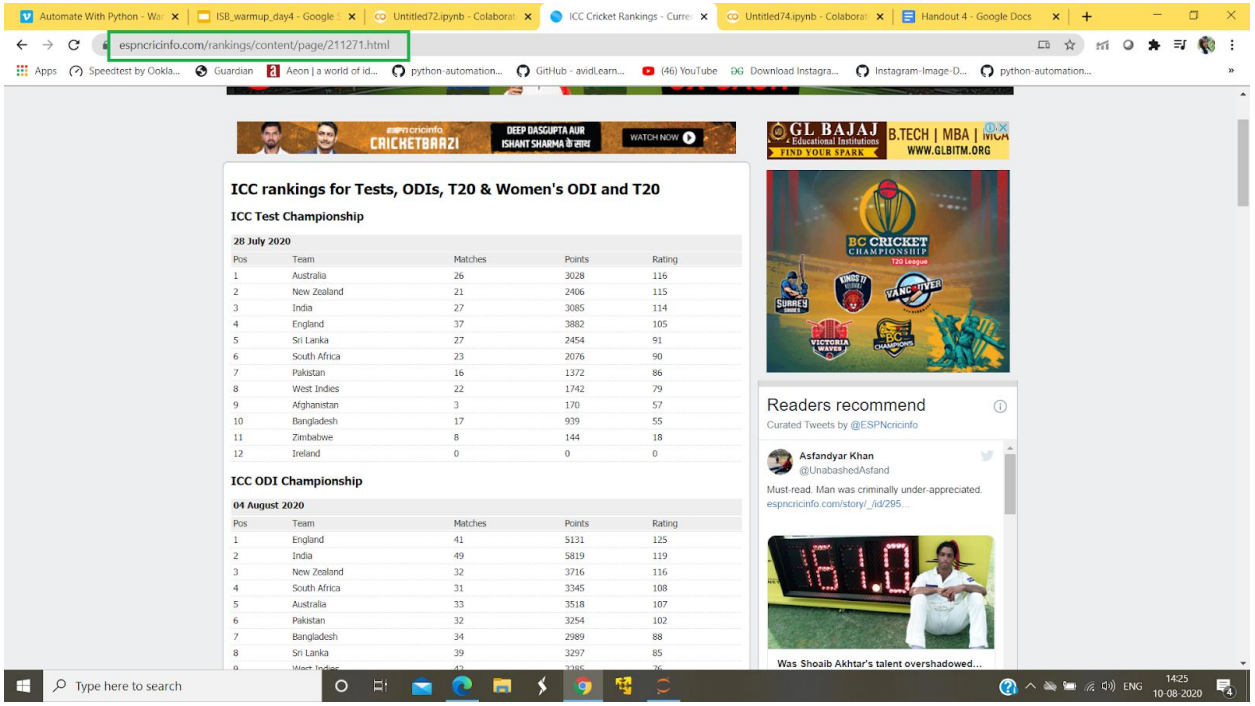


Now using this, we can compare the trend of precipitation in two different cities at the same time. This holds for any other data or dataset as well.

Project 10.2: Data Visualization with ESPN cricinfo website.

So in this project we will be creating datasets that we will use to visualize the data using seaborn library of python The site from where we will be taking the statistics is

https://www.espncricinfo.com/rankings/content/page/211271.html



This basically contains the ICC ranking of the different international teams in different formats like Test, ODIs, Women's ODIs and so one. So let's first find out how to create the relevant datasets

The first step will surely be to import the necessary modules that we will need

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
```

Well you already know what are they used for so let's move forward
Now we will be providing the url of the page from where we will extract the data

```
pg=urlopen('https://www.espncricinfo.com/rankings/content/page/211271.html')
soup=BeautifulSoup(pg,'html.parser')
```

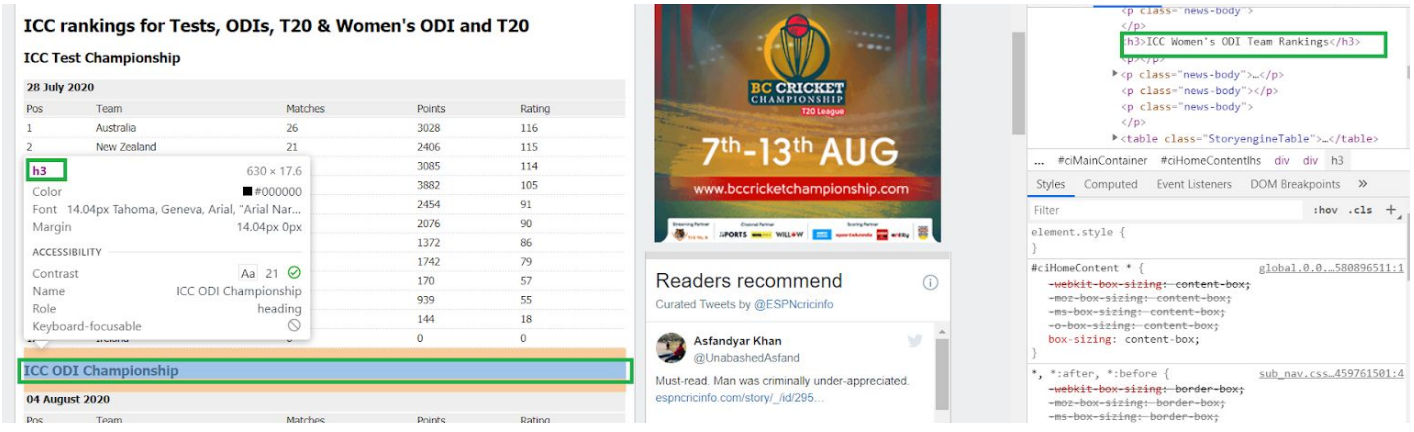
Soup is basically the object of BeautifulSoup which will be parsing that particular html page

Now we need to inspect the page right. So let's inspect and find out which part we need to work with

Well you see that the part that we need is in the class ciPhotoContainer inside div. So we write the following code-

```
body = soup.find('div', {"class": "ciPhotoContainer"})
```

Now you can see that this page contains data for different events. So before we move forward let's create a list of all the headings of the tables. For that Let's inspect and find out where the headings are stored.



You can see the headings are all in h3 tag. So we use the following code and create a list of the headings

```
headings= soup.findAll('h3')
```

Now we need to extract the text from there right?? So we will use these lines of code

```
names=[]
for h in headings:
    names.append(h.text)
```

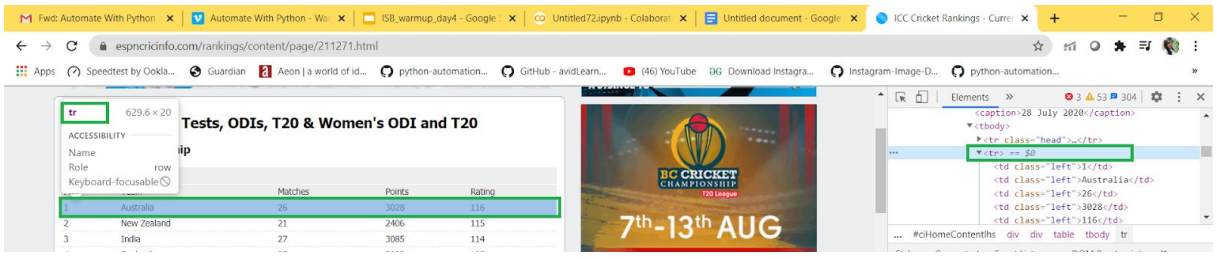
So we will get a list of the headings for the different tables. If we print it out we get the following output -

```
['ICC Test Championship', 'ICC ODI Championship', 'ICC Twenty20 Rankings', "ICC Women's ODI Team Rankings", "ICC Women's T20 Te
am Rankings"]
```

Now to create a dataset what we need is a dataframe. So we need to create a dataframe before we move forward. Well the code to create the dataframe will be

```
import pandas as pd
column_names=['Pos','Team', 'Matches', 'Points', 'Rating']
df=pd.DataFrame(columns= column_names)
```

Now let's inspect the page again and find out how we will actually get the data right?



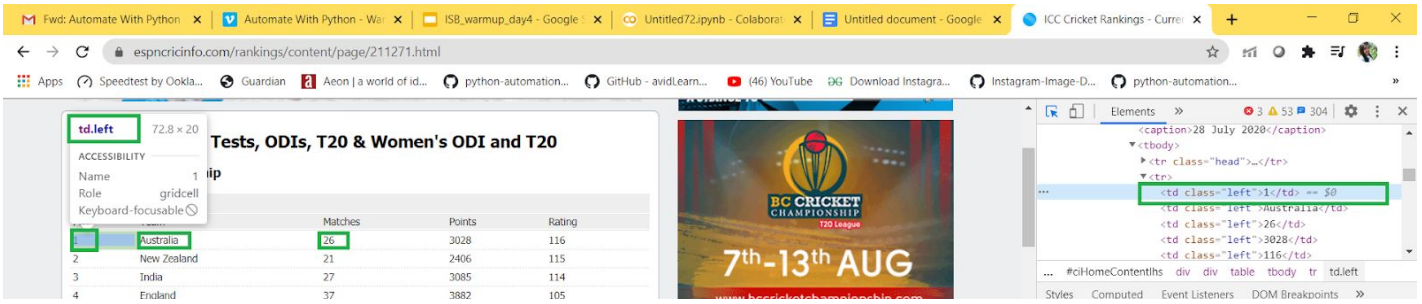
Now you can see that the data is actually present inside tr tag for each row. So let's first create a list.

So we will use the following line of code

```
tr_list=body.findAll('tr')
```

Thus now we have a list of all the tr

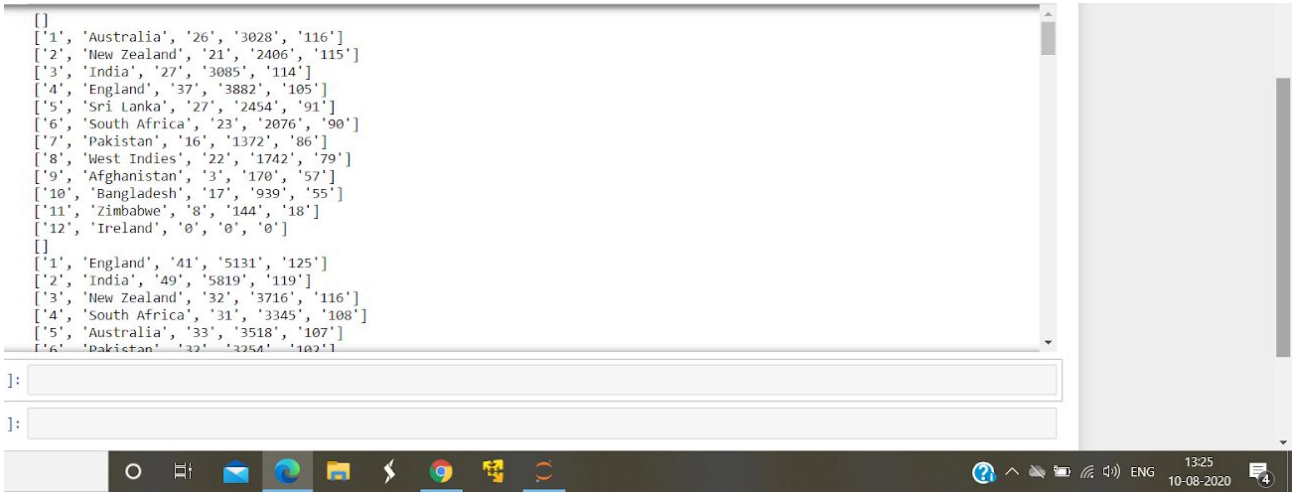
Now look at the following figure



The text is present inside the td actually. So for each tr we need to extract them. So we will need to iterate over all the tr and extract the data
We use the following code

```
for i in tr_list:
    row=[]
    td_list=i.findAll('td')
    for j in td_list:
        row.append(j.text)
```

So basically we are getting into one tr at a time, finding all the td and string them in a list and then using the inner loop for each of the td we are extracting its text. Thus we are getting one complete row of data that we are appending and storing in row. If we print row we get the following output-



Now we need to append this individual row of data to the final dataframe. For that we use the code-

```
data={}
for k in range(len(df.columns)):
            data[df.columns[k]] = row[k]
        df = df.append(data, ignore_index=True)
```

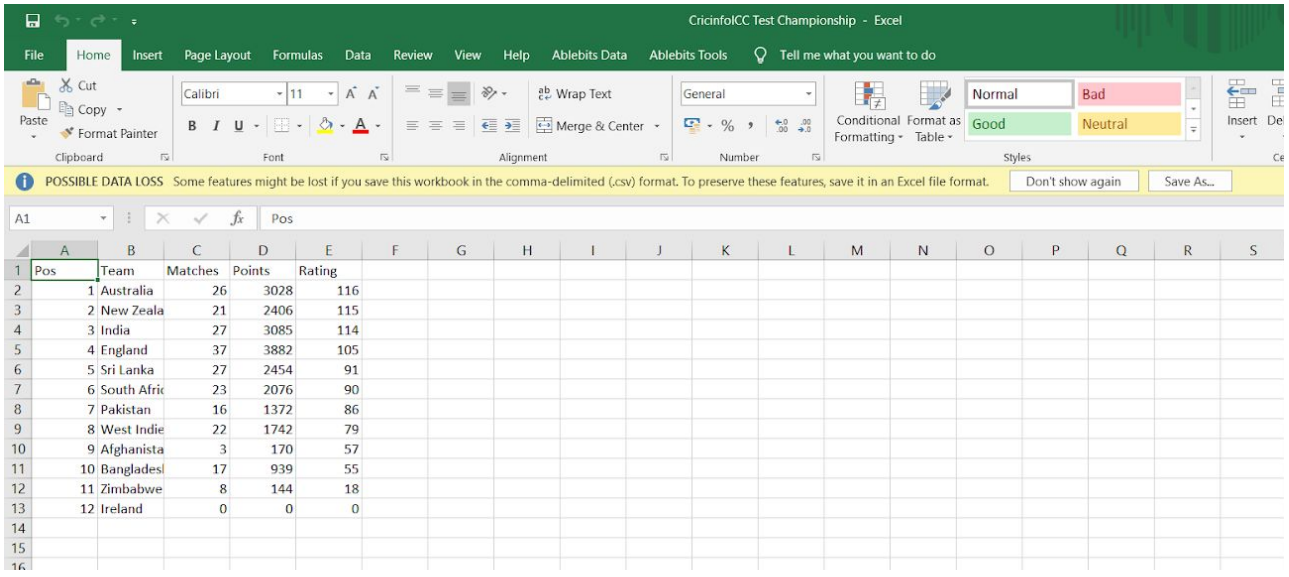
Now the thing is we have different types of tables, one for ODI ranking of the men's team, another for the women's team and so on. So of course we can’t append them to a single dataframe. No if you observe you will see that a [] is returned at the beginning of each table. [] being returned will raise an exception because `data[df.columns[k]] = row[k]` will be then out of range.So let’s use a try and except block i.e. whenever we will encounter a [] we will move to the except block and create a new dataframe. Thus the entire code now looks like this-

```
n=0
for i in tr_list:
    row=[]
    td_list=i.findAll('td')
    for j in td_list:
        row.append(j.text)
    print(row)
```

```
    data={}
    try:
        for k in range(len(df.columns)):
            data[df.columns[k]] = row[k]
        df = df.append(data, ignore_index=True)
    except:
        df=pd.DataFrame(columns= column_names)
        table_name=names[n]
        n=n+1
df.to_csv(os.path.join(path,r'Cricinfo'+table_name+'.csv'), index = False)
```

Thus finally we will provide the path where we wish to save it to and use df.to_csv for this conversion

The final csv file looks like this



Pos	Team	Matches	Points	Rating
1	Australia	26	3028	116
2	New Zeala	21	2406	115
3	India	27	3085	114
4	England	37	3882	105
5	Sri Lanka	27	2454	91
6	South Afric	23	2076	90
7	Pakistan	16	1372	86
8	West Indie	22	1742	79
9	Afghanista	3	170	57
10	Banglades	17	939	55
11	Zimbabwe	8	144	18
12	Ireland	0	0	0

Now let's begin with the visualization part.

In this project for the visualization purpose we will be using Seaborn library

So first find out what is seaborn. Seaborn is a library for making statistical graphics in Python. It provides a high-level interface for drawing attractive and informative statistical graphics. Thus it helps us to visualize and interpret the data quite easily. It is closely related to matplotlib but here in

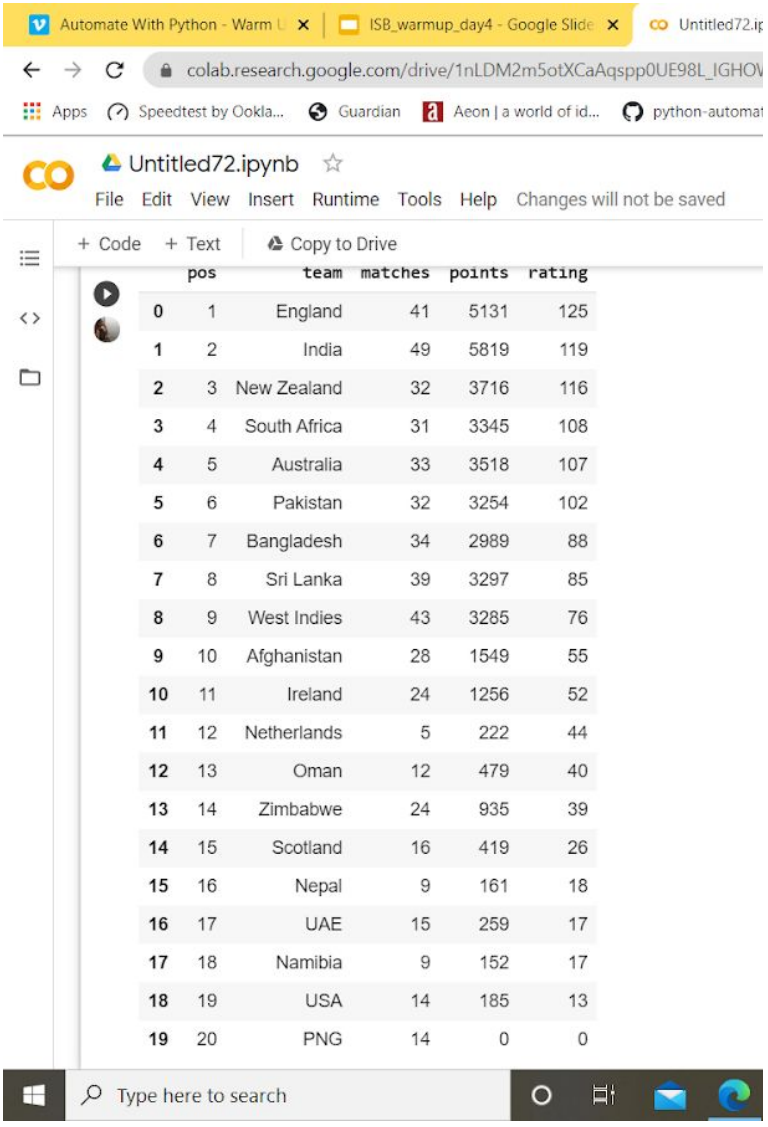
seaborn we need fewer lines of code to get the things done. So at first we will import the library.

```
import seaborn as sns
import pandas as pd
```

We will need pandas also to visualize the dataframe that we will be creating from the csv datasets. So after uploading the file we will store it as a dataframe using the following line

```
odi=pd.read_csv("/content/CricinfoICC ODI Championship.csv")
odi
```

Thus we read the file using pd.read_csv() and now lets see what this dataframe stores



	pos	team	matches	points	rating
0	1	England	41	5131	125
1	2	India	49	5819	119
2	3	New Zealand	32	3716	116
3	4	South Africa	31	3345	108
4	5	Australia	33	3518	107
5	6	Pakistan	32	3254	102
6	7	Bangladesh	34	2989	88
7	8	Sri Lanka	39	3297	85
8	9	West Indies	43	3285	76
9	10	Afghanistan	28	1549	55
10	11	Ireland	24	1256	52
11	12	Netherlands	5	222	44
12	13	Oman	12	479	40
13	14	Zimbabwe	24	935	39
14	15	Scotland	16	419	26
15	16	Nepal	9	161	18
16	17	UAE	15	259	17
17	18	Namibia	9	152	17
18	19	USA	14	185	13
19	20	PNG	14	0	0

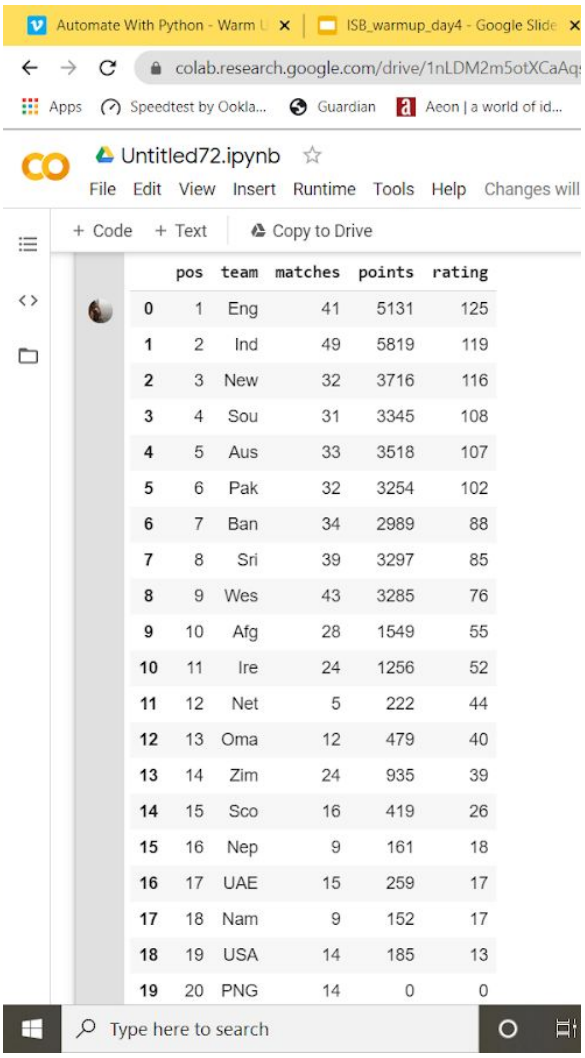
Now since while plotting it will be highly inconvenient to read the name of the teams as they will overlap with each other we will use the following function

```
def trim(x):
  a=x[:3]
  return a
```

This will basically extract the first three characters of the string x and return the trimmed string

```
odi['team']=odi['team'].apply(trim)
odi
```

So we pass the team names from the odi dataset and trim them just to retain the first 3 characters. So now the dataset looks like



	pos	team	matches	points	rating
0	1	Eng	41	5131	125
1	2	Ind	49	5819	119
2	3	New	32	3716	116
3	4	Sou	31	3345	108
4	5	Aus	33	3518	107
5	6	Pak	32	3254	102
6	7	Ban	34	2989	88
7	8	Sri	39	3297	85
8	9	Wes	43	3285	76
9	10	Afg	28	1549	55
10	11	Ire	24	1256	52
11	12	Net	5	222	44
12	13	Oma	12	479	40
13	14	Zim	24	935	39
14	15	Sco	16	419	26
15	16	Nep	9	161	18
16	17	UAE	15	259	17
17	18	Nam	9	152	17
18	19	USA	14	185	13
19	20	PNG	14	0	0

Now we will plot the data. But before that for our convenience to view the data properly we will increase the font size of the data appearing on the graph as well as increase the size of the frame of our graph as well.

So to increase the size of the data we use

```
import matplotlib as mpb
mpb.rcParams['font.size']=25
```

Now for the frame we use

```
import matplotlib.pyplot as plt
f,ax=plt.subplots(figsize=(20,10))
```

Let’s plot the first graph. We will plot a bar graph here.

Well what is a bar graph ?

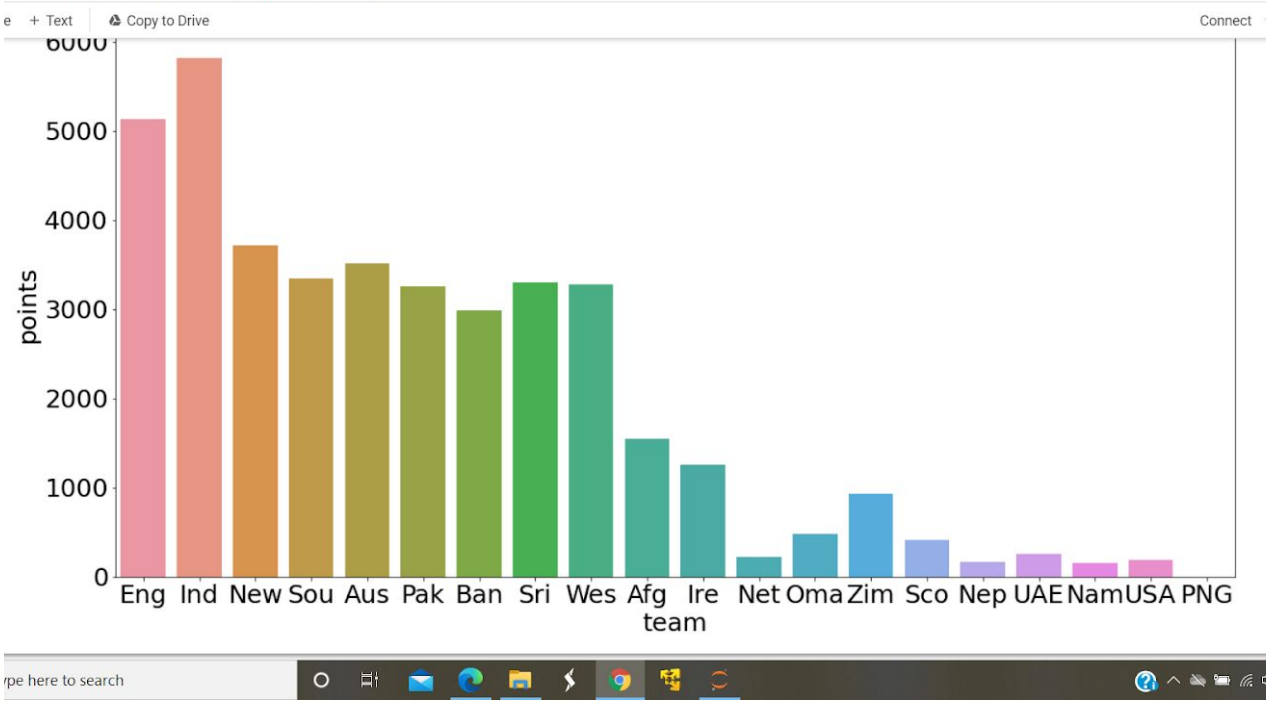
Bar charts are represented with rectangular blocks where the length of the block shows the value .

We will use bar graphs to compare the number of something for different categories.

So we will make use of sns.barplot()

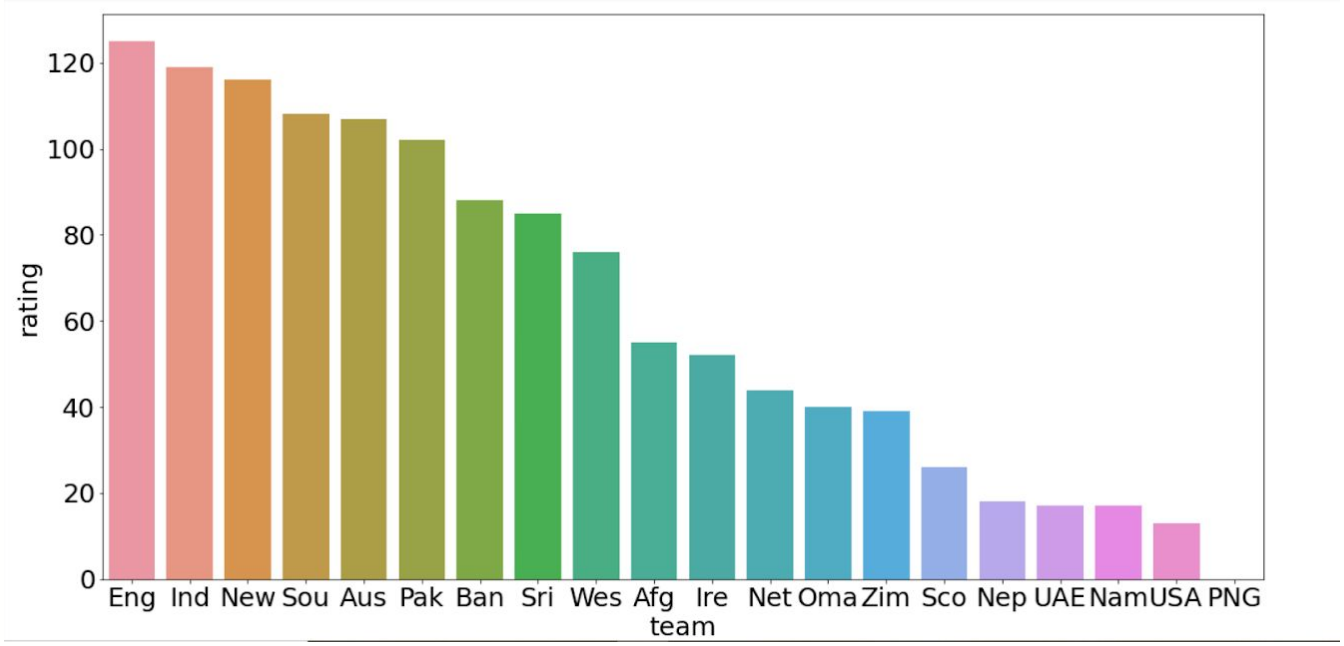
The arguments that we will pass is first the column that we want along the x axis, then the column that we want along the y-axis and then the data that we will plot

```
ax=sns.barplot('team','points',data=odi)
```



Now let’s visualize rating vs team

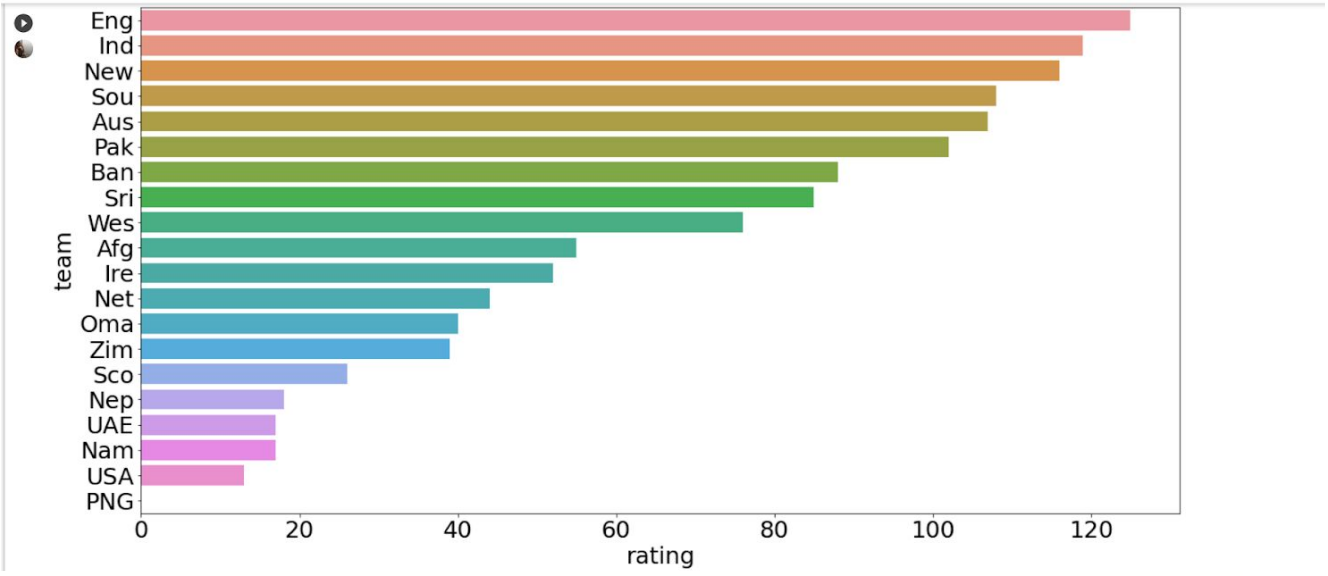
```
f,ax=plt.subplots(figsize=(20,10))
ax=sns.barplot('team','rating',data=odi)
```



Now if we use

```
f,ax=plt.subplots(figsize=(20,10))
ax=sns.barplot('rating','team',data=odi)
```

Team will be plotted along y axis and rating will be along x axis



Now lets load the Women's ICC ODI dataset and visualize it

```
wodi=pd.read_csv("/content/CricinfoICC Women's ODI Team Rankings.csv")
wodi
```

	pos	team	matches	points	rating
0	1	Australia Women	26	3945	152
1	2	India Women	30	3747	125
2	3	England Women	29	3568	123
3	4	New Zealand Women	26	2675	103
4	5	South Africa Women	36	3626	101
5	6	West Indies Women	24	1979	82
6	7	Pakistan Women	25	1835	73
7	8	Sri Lanka Women	22	1208	55
8	9	Bangladesh Women	10	542	54
9	10	Ireland Women	6	110	18

Now we will again apply the trim function on this as well

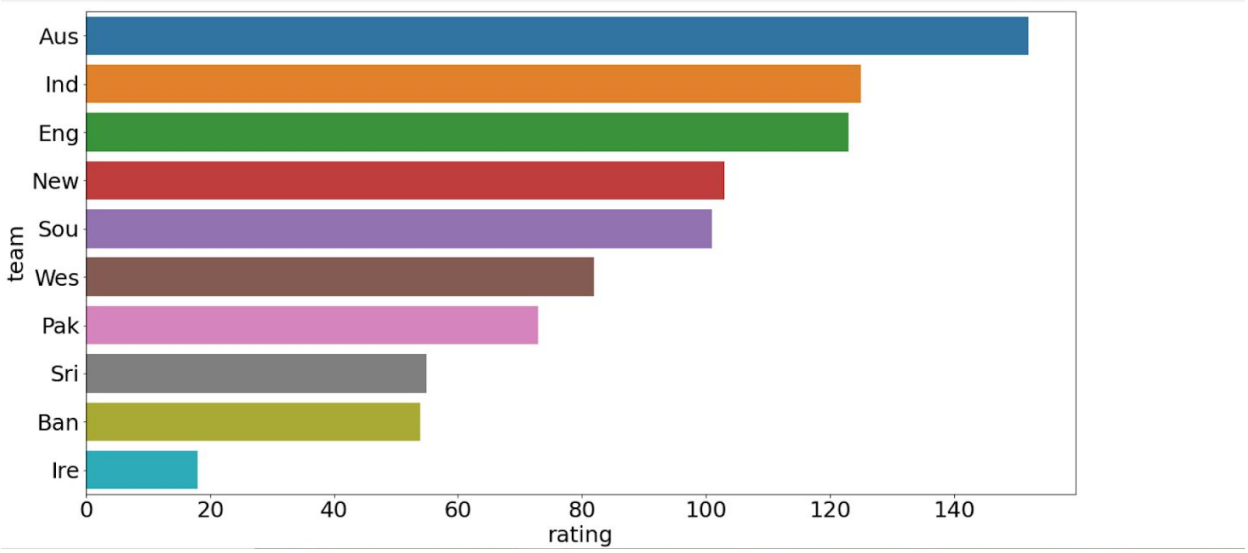
```
wodi['team']=wodi['team'].apply(trim)
wodi
```

We get the following output

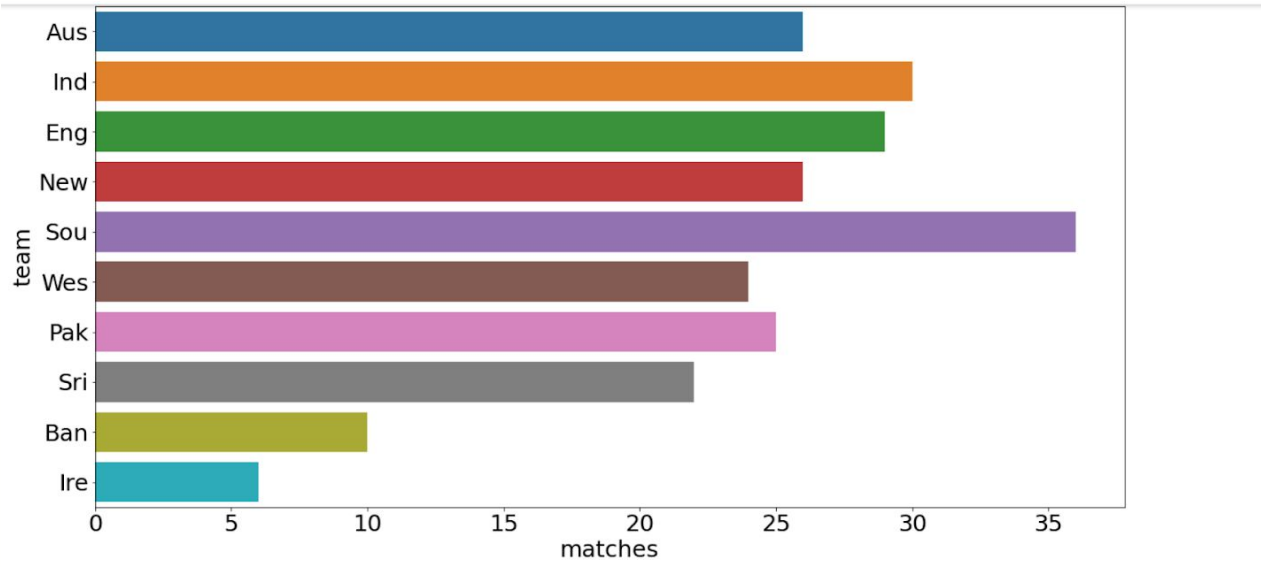
	pos	team	matches	points	rating
0	1	Aus	26	3945	152
1	2	Ind	30	3747	125
2	3	Eng	29	3568	123
3	4	New	26	2675	103
4	5	Sou	36	3626	101
5	6	Wes	24	1979	82
6	7	Pak	25	1835	73
7	8	Sri	22	1208	55
8	9	Ban	10	542	54
9	10	Ire	6	110	18

So let’s visualize this dataset as well

```
f,ax=plt.subplots(figsize=(20,10))
ax=sns.barplot('rating','team',data=wodi)
```




```
f,ax=plt.subplots(figsize=(20,10))
ax=sns.barplot('matches','team',data=wodi)
```



Now we will visualize both these datasets together. So before that let's add a gender column to both the datasets.

```
wodi['gender']='Female'
```

```
odi['gender']='Male'
```

Now one column gets added to both the dataframes like this

+ Code + Text Copy to Drive

[ ]

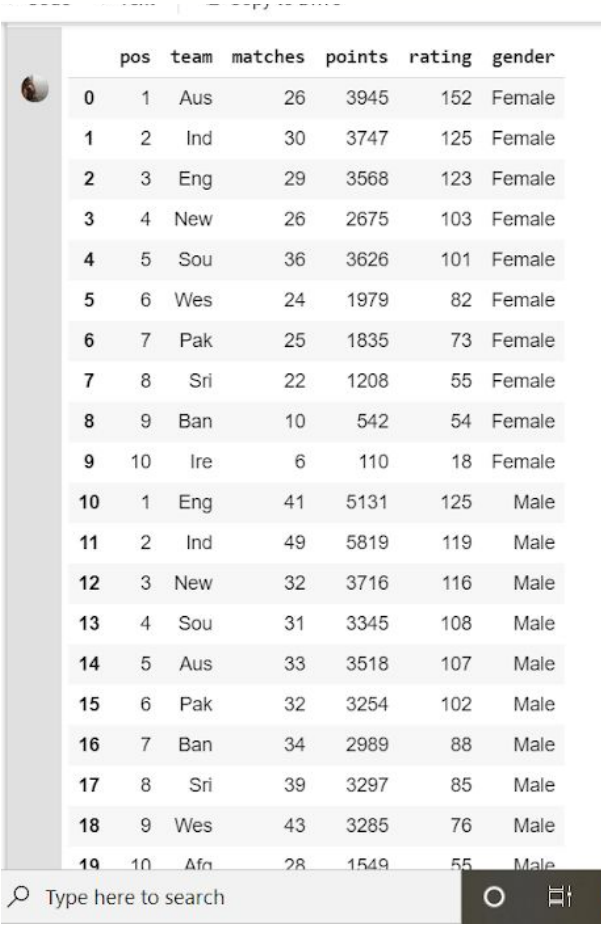
	pos	team	matches	points	rating	gender
0	1	Eng	41	5131	125	Male
1	2	Ind	49	5819	119	Male
2	3	New	32	3716	116	Male
3	4	Sou	31	3345	108	Male
4	5	Aus	33	3518	107	Male
5	6	Pak	32	3254	102	Male
6	7	Ban	34	2989	88	Male
7	8	Sri	39	3297	85	Male
8	9	Wes	43	3285	76	Male
9	10	Afg	28	1549	55	Male
10	11	Ire	24	1256	52	Male
11	12	Net	5	222	44	Male
12	13	Oma	12	479	40	Male
13	14	Zim	24	935	39	Male
14	15	Sco	16	419	26	Male
15	16	Nep	9	161	18	Male
16	17	UAE	15	259	17	Male
17	18	Nam	9	152	17	Male
18	19	USA	14	185	13	Male
19	20	PNG	14	0	0	Male

Type here to search

Now we will combine these two together for visualization.

```
comb=pd.concat([wodi,odi],ignore_index=True)
comb
```

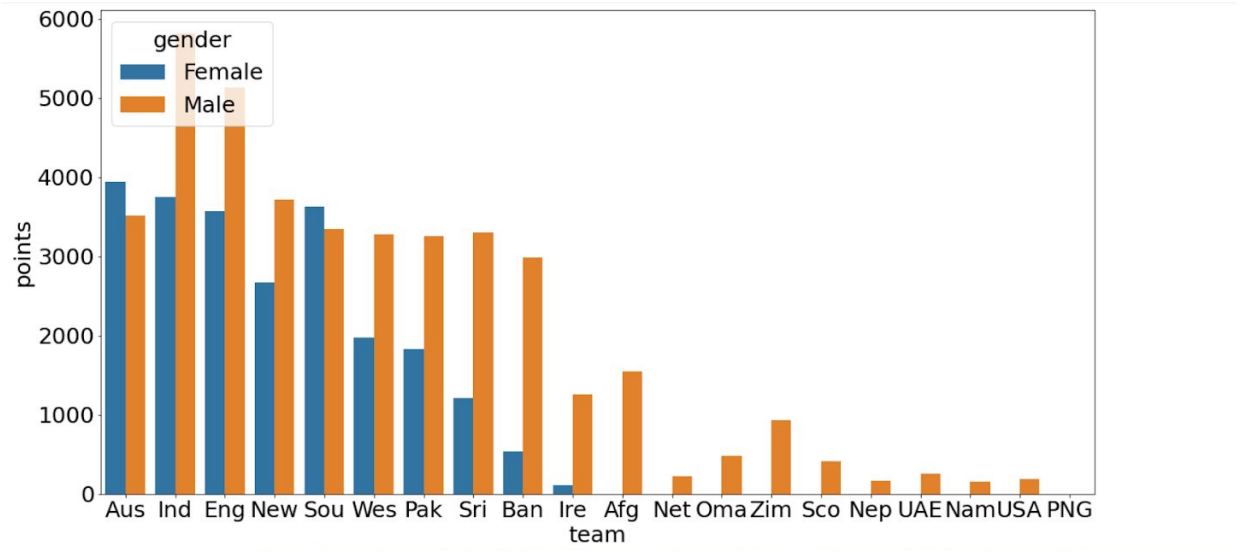
.concat() helps to combine two datasets one after another, we will use ignore_index=True to ensure that the index is continuous when they are concatenated one after another.
Thus comb is the new dataset that contains information about both men's and women's ranking



	pos	team	matches	points	rating	gender
0	1	Aus	26	3945	152	Female
1	2	Ind	30	3747	125	Female
2	3	Eng	29	3568	123	Female
3	4	New	26	2675	103	Female
4	5	Sou	36	3626	101	Female
5	6	Wes	24	1979	82	Female
6	7	Pak	25	1835	73	Female
7	8	Sri	22	1208	55	Female
8	9	Ban	10	542	54	Female
9	10	Ire	6	110	18	Female
10	1	Eng	41	5131	125	Male
11	2	Ind	49	5819	119	Male
12	3	New	32	3716	116	Male
13	4	Sou	31	3345	108	Male
14	5	Aus	33	3518	107	Male
15	6	Pak	32	3254	102	Male
16	7	Ban	34	2989	88	Male
17	8	Sri	39	3297	85	Male
18	9	Wes	43	3285	76	Male
19	10	Afg	28	1549	55	Male


```
f,ax=plt.subplots(figsize=(20,10))
ax=sns.barplot('team','points',hue='gender',data=comb)
```

The hue parameter determines which column in the data frame should be used for colour encoding
So now the output looks like-



Thus you can observe that on the basis of gender we have segregated the data. For those teams whose female team record is not found we got only the data of the male team.

Now we will see another type of plot

Scatter plot - A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables. They are mostly used when

- You have many different data points, and you want to highlight similarities in the data set.
- To find outliers or for understanding the distribution of your data.

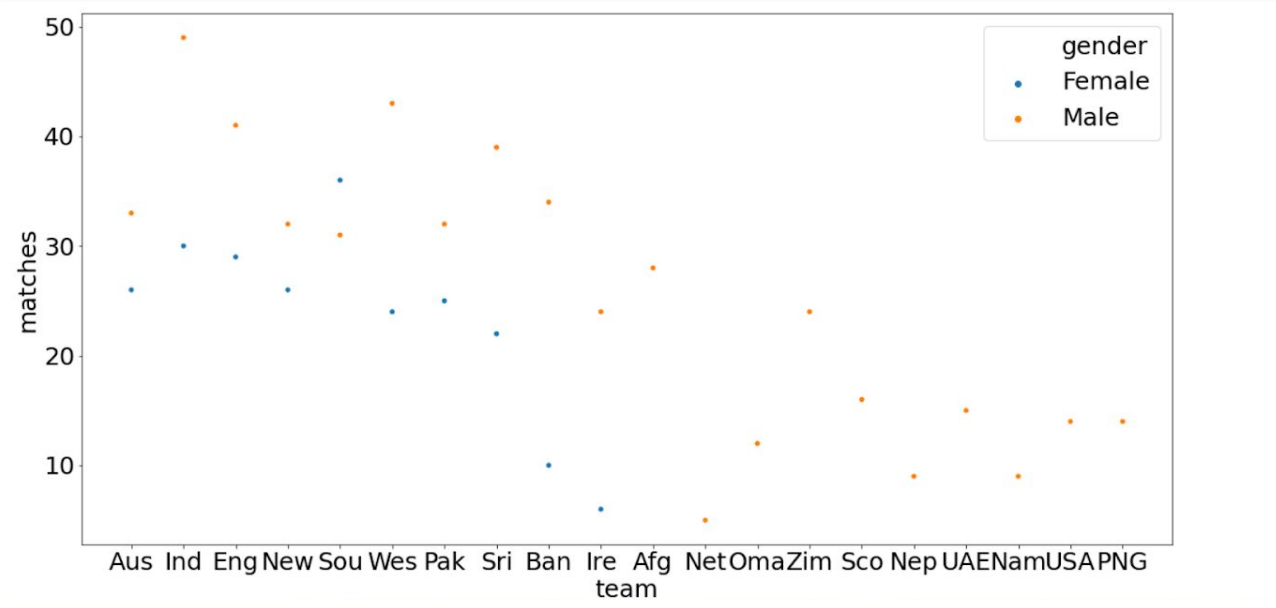
In seaborn we can implement scatter plot using sns.scatterplot()
The arguments that we pass are:

- The column that we wish to plot on x axis
- The column that we wish to plot along y axis
- hue
- The data on which we will work

Thus we use the following code

```
f,ax=plt.subplots(figsize=(20,10))
ax=sns.scatterplot('team','matches',hue='gender',data=comb)
```

Thus the output we get is



Project 10.3: Data Visualization with Mall data.

Now we will start with the next project .

So for this the first thing that you will do is to go to the kaggle website and download the dataset that is already provided there.

The link is -

https://www.kaggle.com/shwetabh123/mall-customers

So just download it and it will be downloaded in a zip format. You need to extract the dataset. For this just right click on it and select extract all and

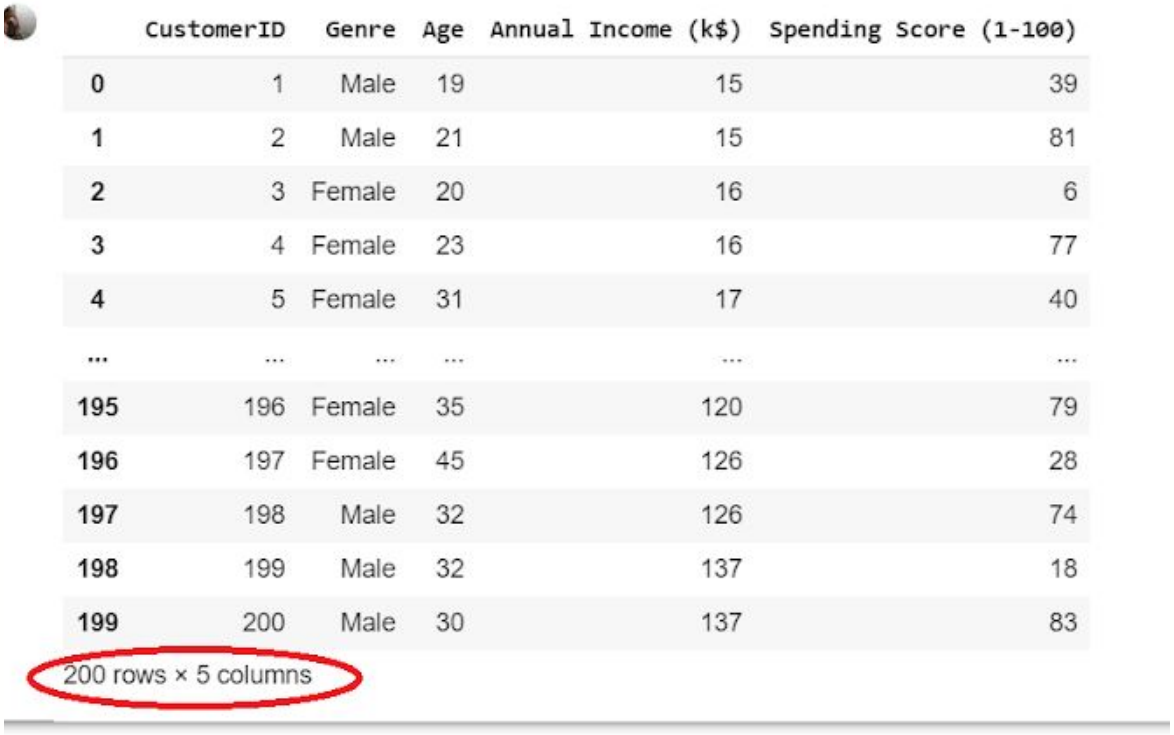
save it to a suitable location. Now open a new notebook in Google Colab and upload the file. Now just type in the following codes

```
import pandas as pd
```

```
m=pd.read_csv("/content/Mall_Customers.csv")
```

```
m
```

Now lets see what the output looks like



	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

So this is the final dataset which is stored as a dataframe and we will be working on this.

You can see the total number of rows and columns are mentioned. There is data in between but all of them are not displayed here at once.

So since we are done with loading the data, let's move forward and start visualizing. So the first type of plot that we will see is a **Count Plot**

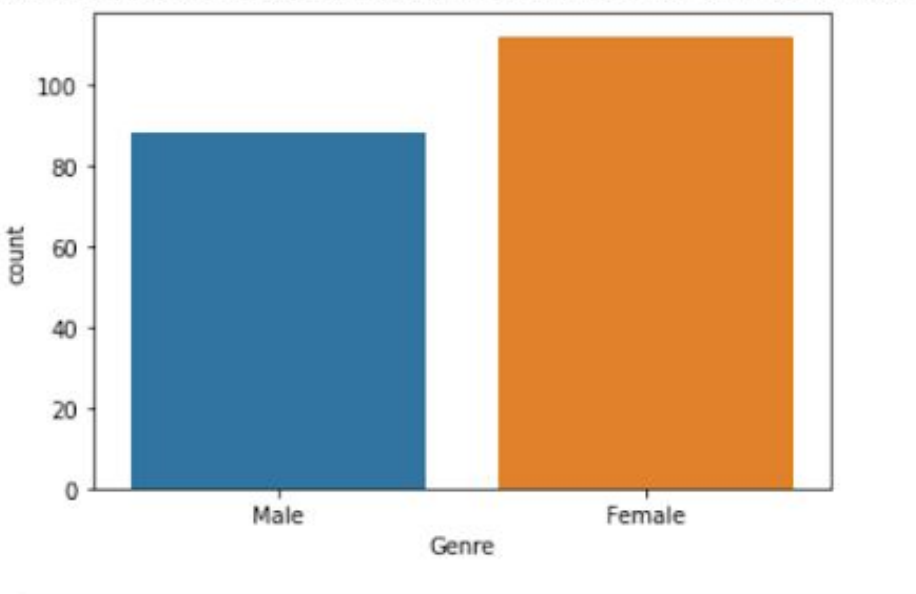
Count Plot - Show the counts of observations in each categorical bin using bars.They are similar to bar graphs but here along the x axis we have the categories and along y the count of each of them.

Just type in the following code

```
import seaborn as sns
```

```
sns.countplot('Genre',data=m)
```

So we have given the column on which we want to get the categorized values and then give the data. Thus we get the following output

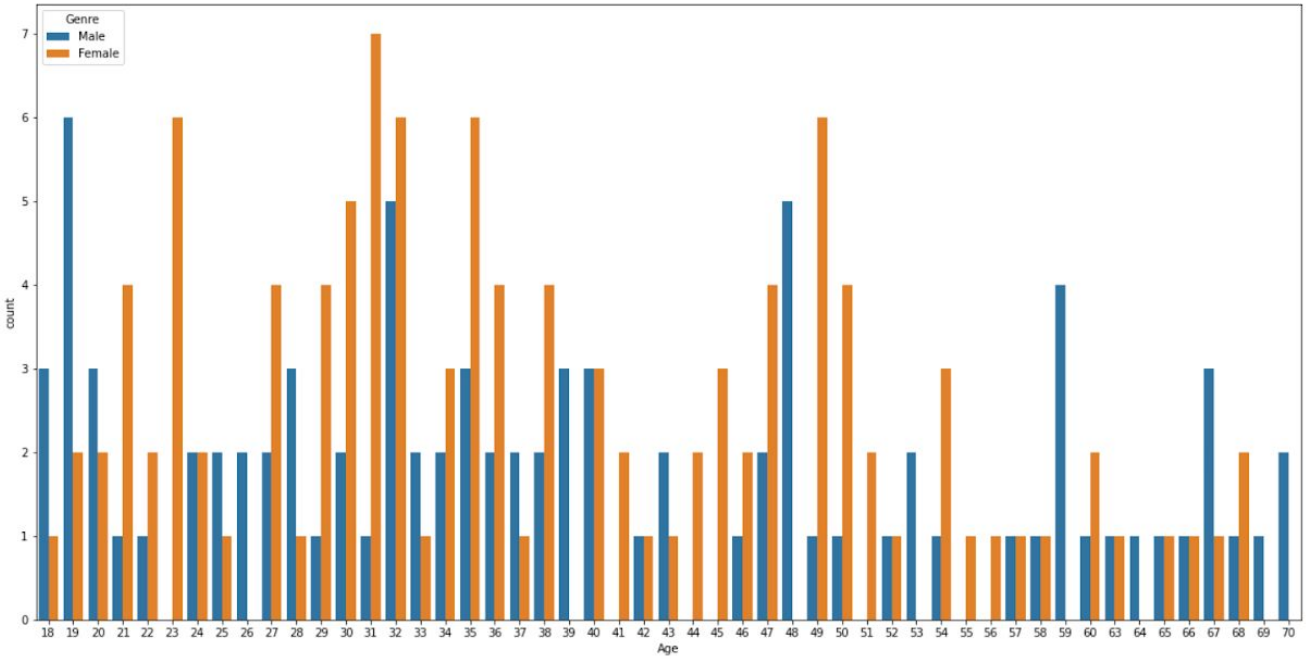


Thus we get the **number of people belonging to different genres i.e. male and female**. Now let's try it out on a different column say age and let's use hue also to segregate on the basis of genres

```
import matplotlib.pyplot as plt
```

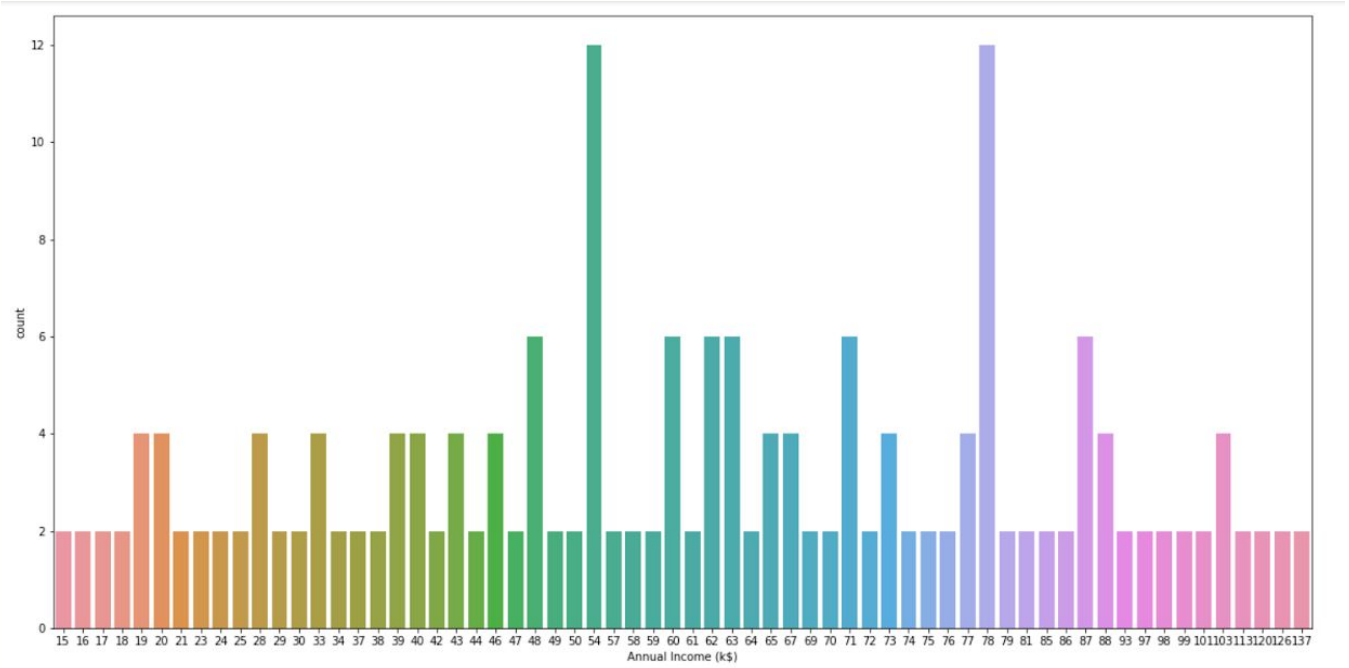
```
f,ax=plt.subplots(figsize=(20,10))
```

```
ax=sns.countplot('Age',hue='Genre',data=m)
```



So we can see that we get an idea of the count of people visiting the mall on the basis of their age and also segregated on the basis of their genre.Now lets visualize for Annual income

```
f,ax=plt.subplots(figsize=(20,10))
ax=sns.countplot('Annual Income (k$)',data=m)
```



You can see that the annual income is somewhat stagnant.
Now let's move onto the next plot type.
Histograms -Histograms represent the data distribution by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin. It is a combination of bar graph and line graph. It is used to show the data distribution.
When are they used ??

- To make comparisons in data sets over an interval or time
- To show the distribution of data

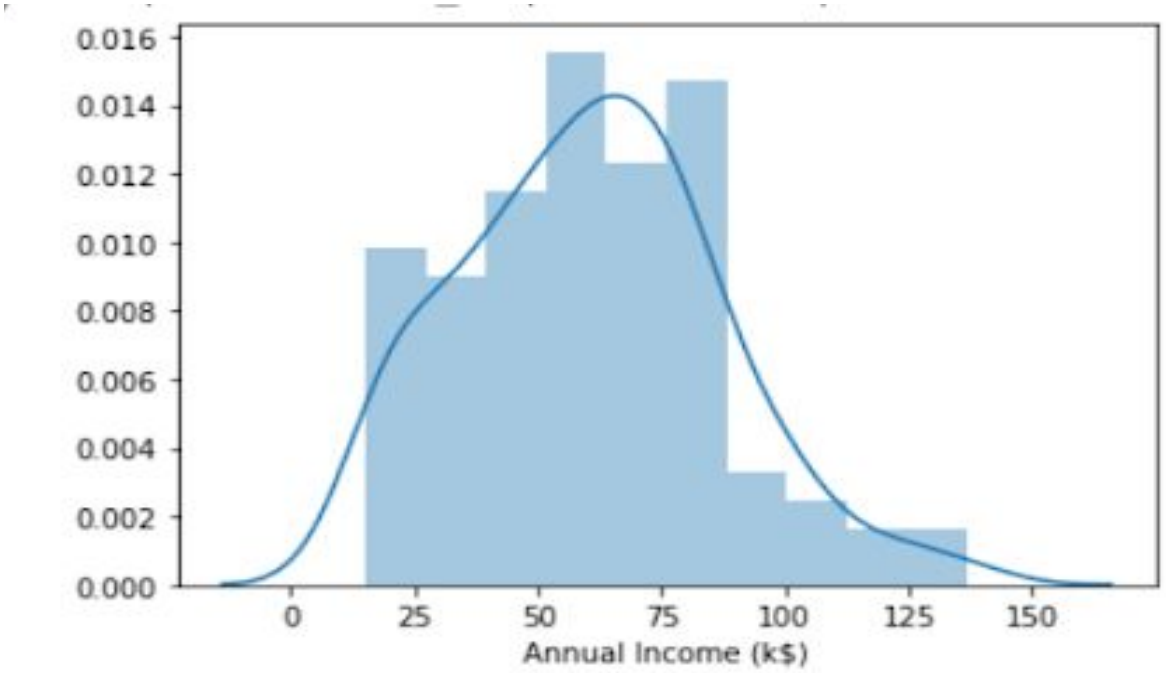
So let's check out histogram using our dataset
For visualizing histogram we will use sns.distplot(). The arguments will be

- The column for which we wish to visualize
- The data on which we will plot

Type in the following code

```
sns.distplot(m['Annual Income (k$)'])
```

The output that we get is



Okay so what does this plot tell us??
Well the peak of this line graph tells us about the average income of the people coming to this place and also the highest income belongs to that range somewhat 52-53 like that.
Now let's move forward to our next plot i.e. **Boxplot.**

Box plot - A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. This shows a distribution of data, usually across groups, based on a five number summary

- the minimum
- first quartile (25%)
- the median (second quartile) (50%)
- third quartile(75%)
- the maximum.

So **when to use a boxplot**

- To display or compare a distribution of data
- To identify the minimum, maximum and median of data
- To identify outliers.

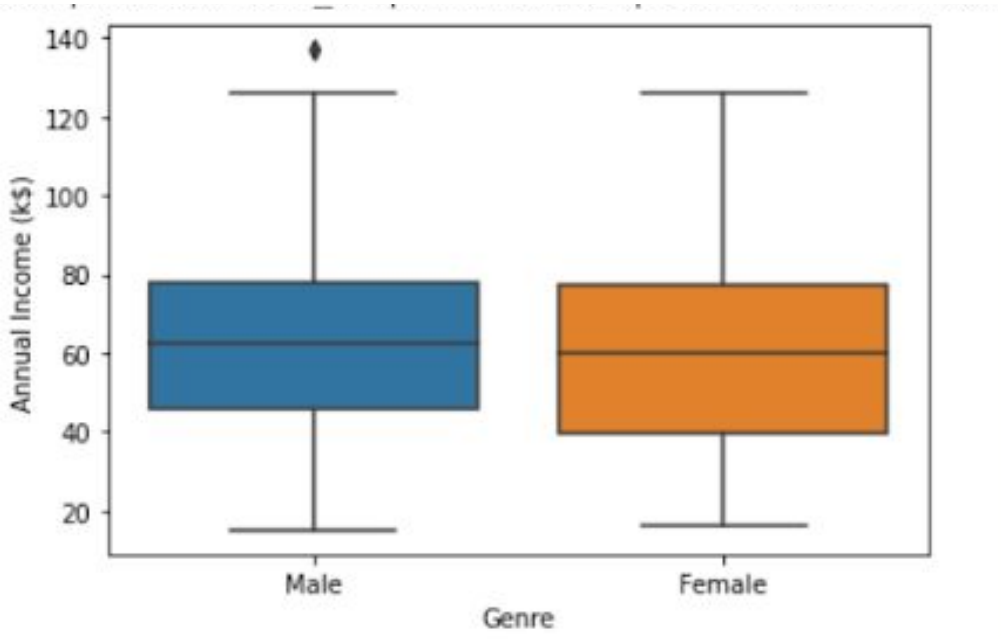
Now lets plot boxplot on our data

Type in the following code

```
sns.boxplot('Genre','Annual Income (k$)',data=m)
```

Here genre is the x axis and Annual income is along y axis.

We get the following output

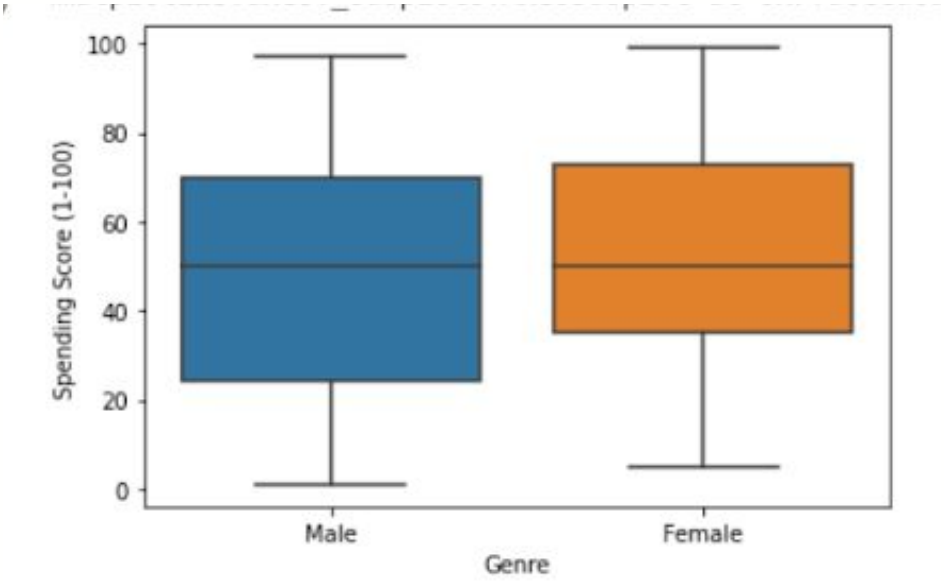


Well you can observe that we get the five point summary of the data based on the two different genres. The black dot on the top represents an outlier. So you can see how these plots help to detect outliers.

Let’s visualize for another column

```
sns.boxplot('Genre','Spending Score (1-100)',data=m)
```

We get the following plot

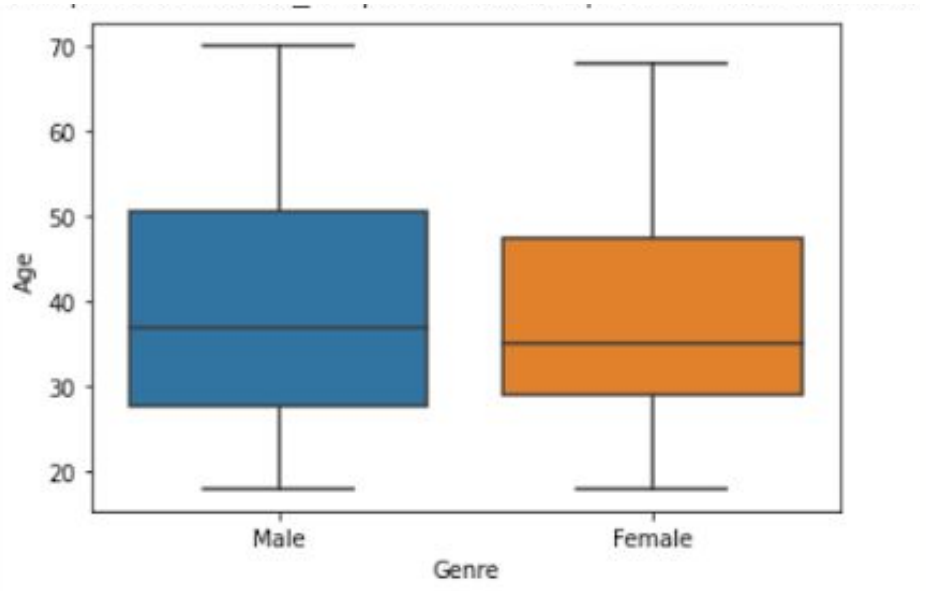


From here you can get an idea about the **minimum spending score of males and females**, their **average spending scores** etc. Like we can conclude that the maximum spending score is greater for females as compared to males.

Similarly lets visualize for

```
sns.boxplot('Genre','Age',data=m)
```

The output we get is



Now let's move onto the next type of plot

Heatmap- Heatmap shows the relationship between two measures. For plotting heatmap we need to find correlation. So **what is correlation**??

Correlation between two quantities means how much related one is to another or how much direct dependency exists between them.The value ranges from **0-1**, **0 means that they are not at all correlated and 1 means that they are highly correlated.** Now we can have **positive or negative correlation.** Positive correlation means with an increment in one, the other also increases and vice versa i.e a direct relationship exists. In negative correlation with an increment in one, the other decreases and vice versa i.e an indirect direct relationship exists. So let's find out the correlation first

```
a=m.corr()
```

Let’s print the correlation values

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
CustomerID	1.000000	-0.026763	0.977548	0.013835
Age	-0.026763	1.000000	-0.012398	-0.327227
Annual Income (k$)	0.977548	-0.012398	1.000000	0.009903
Spending Score (1-100)	0.013835	-0.327227	0.009903	1.000000

Now this is a **4*4 matrix** and this shows the correlation values between all the columns. However since the genre column contains a non numerical value it is not included in this matrix. Although there are ways to do that as well but for the time being we will keep it simple and work with this only.

Now we will be plotting the **heatmap.** So we pass the correlation and if we write annot=True the value of the correlation will also be displayed by the side

```
sns.heatmap(a,annot=True)
```

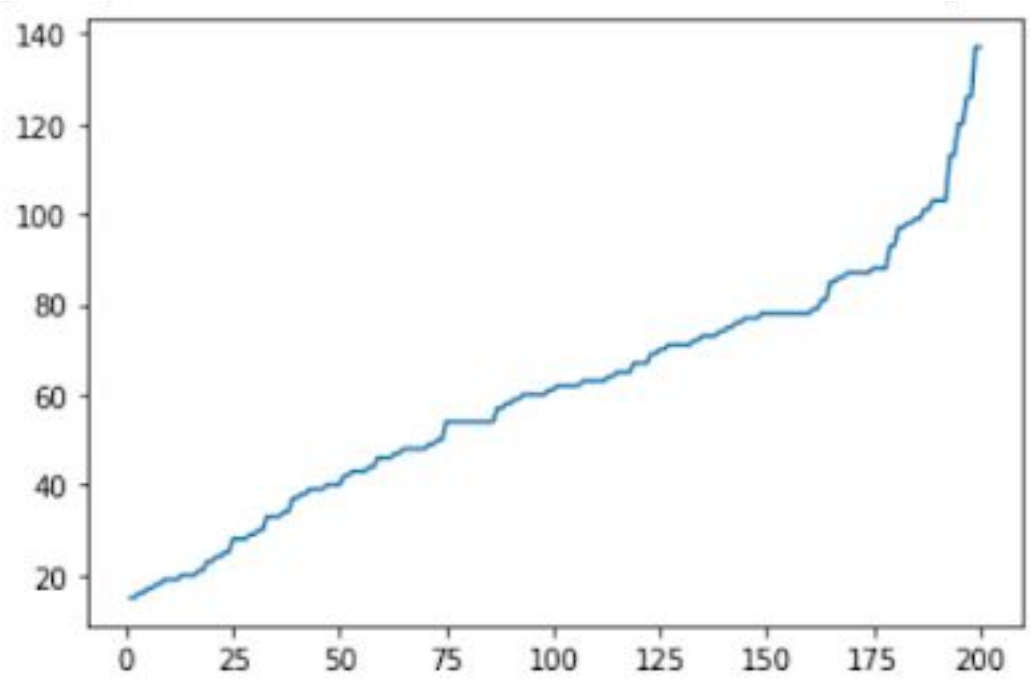
So let's check out the output

The colour scale denotes how the color varies depending upon the value of correlation and the positive and the negative sign denotes if its positive correlation or negative correlation.

You can see that there is really a **strong correlation between annual income and the customer id.** Well let's check that out if such a strong dependency truly exists or not. We will plot a simple graph and check.

```
plt.plot(m['CustomerID'],m['Annual Income (k$)'])
```

Let's check out the output



Well you can see that **such a strong correlation truly exists** and thus heatmap helps us to visualize that.

Project 10.4: Data Visualization with Breast Cancer data.

Now for this project we will download the dataset from **kaggle.**

The link is
https://www.kaggle.com/merishnasuwal/breast-cancer-prediction-dataset

Visit the site, download the file, extract it and save it. Now finally open google colab and upload the file. Now we are ready to visualize the data.

Now we will read the data and store it in a dataframe

```
import pandas as pd
b=pd.read_csv("/content/Breast_cancer_data.csv")
b
```

Lets see the result

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0
565	20.13	28.25	131.20	1261.0	0.09780	0
566	16.60	28.08	108.30	858.1	0.08455	0
567	20.60	29.33	140.10	1265.0	0.11780	0
568	7.76	24.54	47.92	181.0	0.05263	1

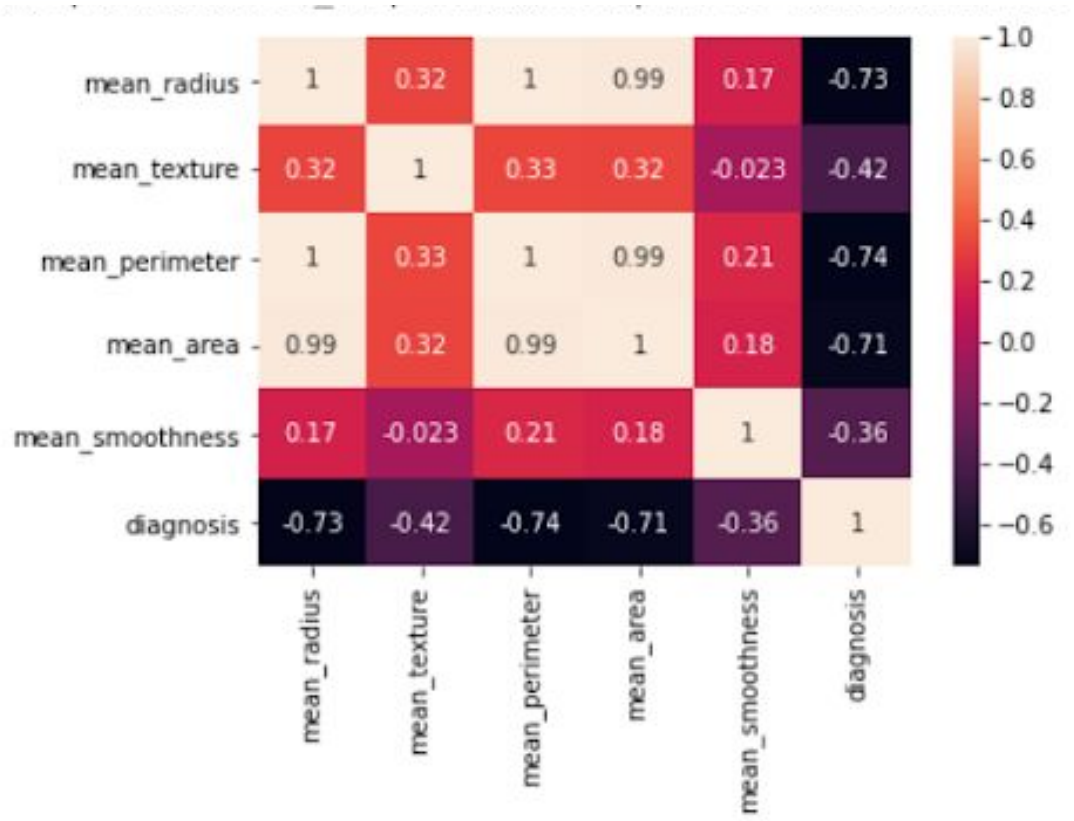
569 rows × 6 columns

This is what we get. There are **569 rows and 6 columns**. In the last column **0 means Positive** for breast cancer and **1 means Negative** for breast cancer.

Well we will be using the same plots here and try to visualize the data. So we will start with Heatmap first

```
import seaborn as sns
sns.heatmap(b.corr(),annot=True)
```

Lets see what it returns

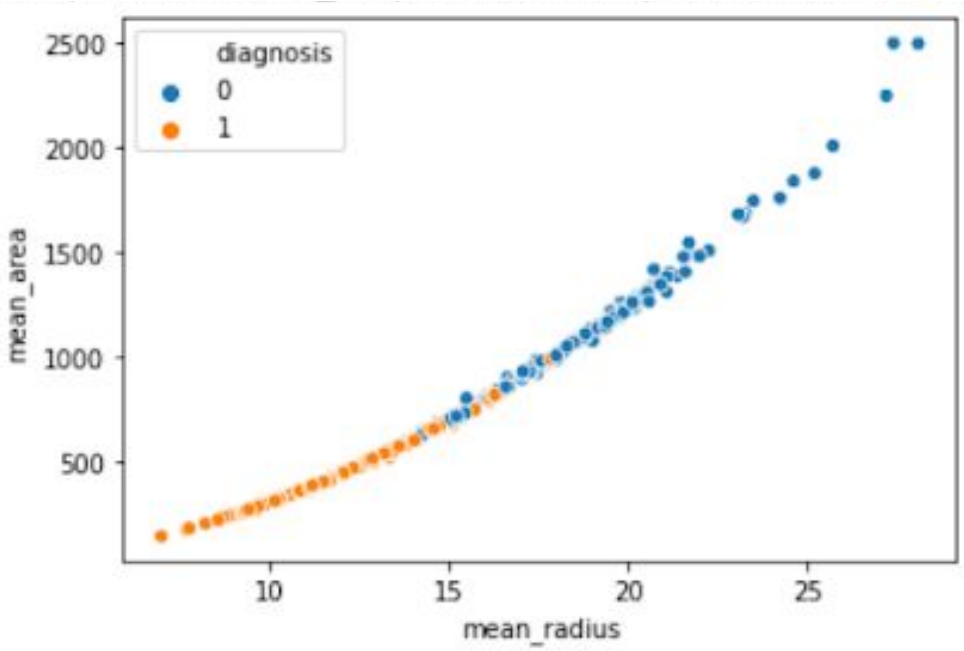


Here our target feature is diagnosis i.e we want to visualize **how the rest of the parameters are correlated with diagnosis.** We see that **mean_radius has a high negative correlation with diagnosis** (remember here 0 means testing positive for cancer and 1 means negative). Thus as the mean_radius of the tumour increases the chance of being diagnosed with cancer too increases(because it approaches 0). A similar kind of **high correlation exists with mean_perimeter and mean_area** as well but **for mean_smoothness or mean_texture the correlation is quite low** though its negative here as well. Apart from this also you can view the correlation between the other factors as well like very **high positive correlation between mean_radius and mean_area.**

Now let's move forward to the next type of visualization. Let’s use scatter plot

```
sns.scatterplot('mean_radius','mean_area',hue='diagnosis',data=b)
```

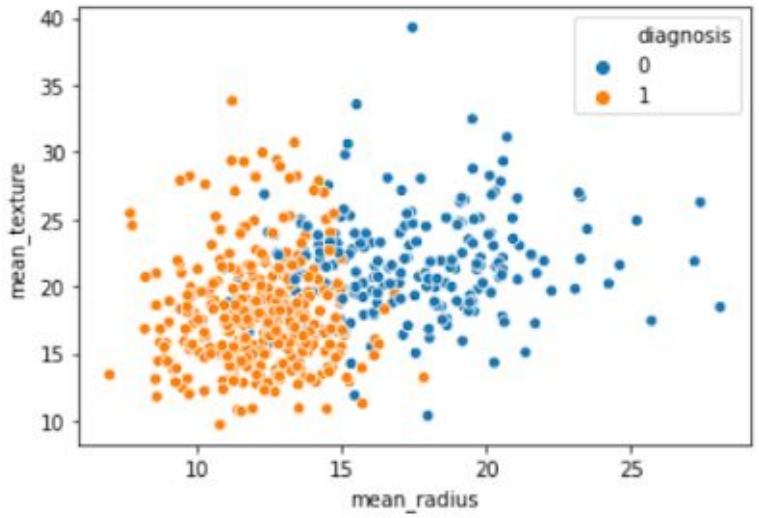
Let's observe the output



Well you get somewhat a **linear trend** and this is obvious as well because we have seen how the mean_radius and mean_area are **highly correlated** with each other and with diagnosis as well. Thus when they increase we observe that there is a higher chance of being diagnosed with breast cancer. Now let's try out with a low correlated pair.

Type in the following code

```
sns.scatterplot('mean_radius','mean_texture',hue='diagnosis',data=b)
```

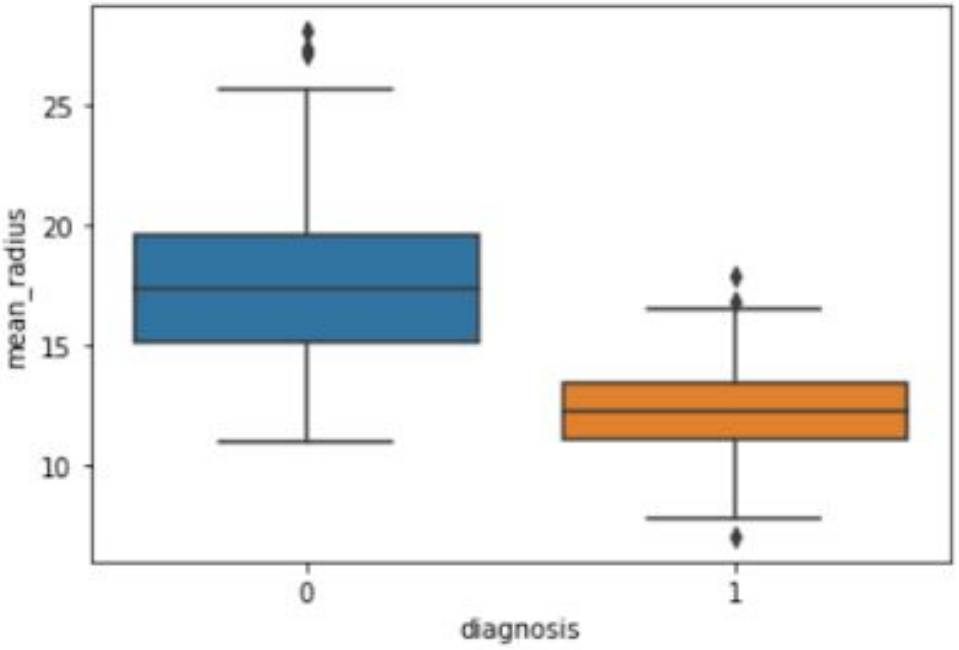


Thus we see that **when mean_radius increases there is a higher chance of having breast cancer.**

Now let's visualize the data using a boxplot

```
sns.boxplot('diagnosis','mean_radius',data=b)
```

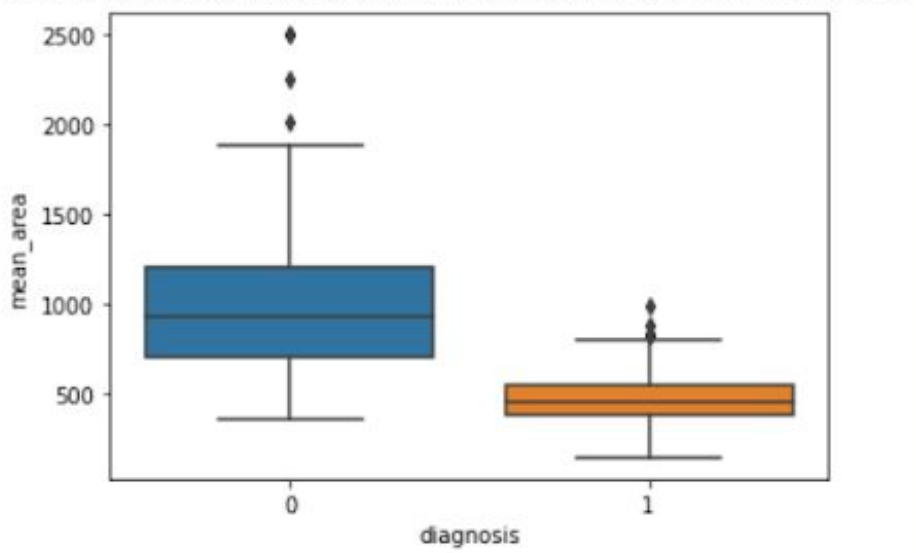
Let's look at the output



Now you can see from this graph that when **the person has breast cancer the mean_radius lies between 15 and 20 but when the patient doesn't have breast cancer then the median is much lower, somewhat between 10-15 (12 around approx).** This should have been the case as well because of the strong correlation existing between mean_radius and being diagnosed with cancer. You can also find the outliers for example those cases where although the mean_radius is quite high the person is not diagnosed with cancer.

```
sns.boxplot('diagnosis','mean_area',data=b)
```

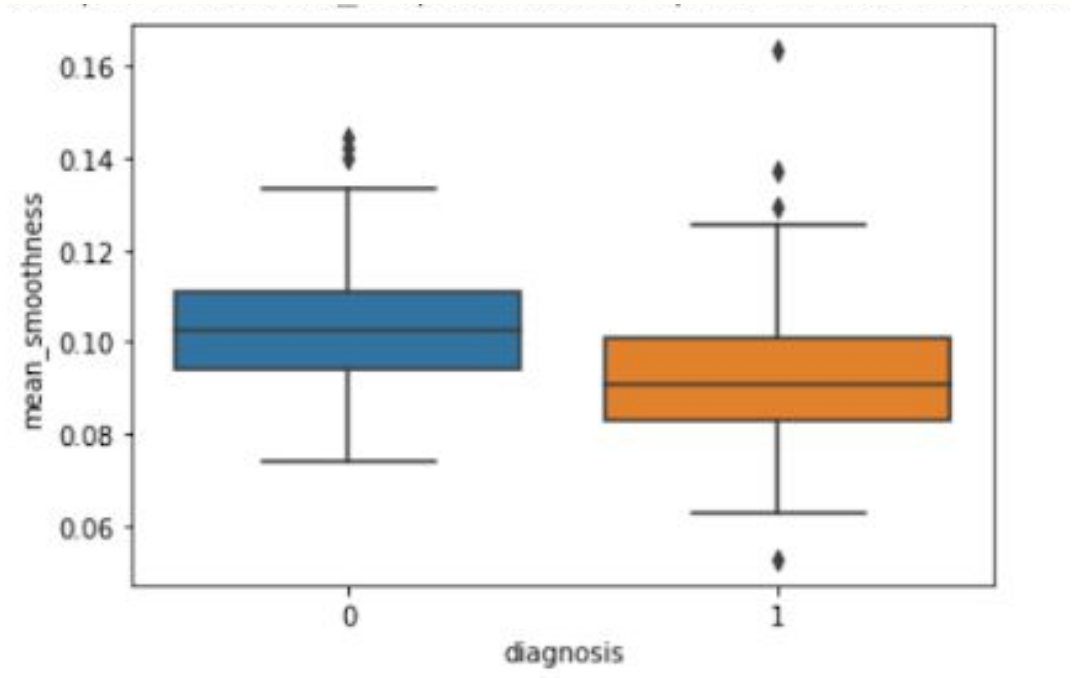
Let's look at the output in this case also



Here also you can see a similar trend like the previous one
Now let's try out with a low correlated pair

```
sns.boxplot('diagnosis','mean_smoothness',data=b)
```

Let's look at the output



Well in this case you will find that there are **a number of outliers and the difference between the data range is not that high and we can't**

deduce that much clearly from this one as compared to the previous ones where a clear noticeable distinction existed.