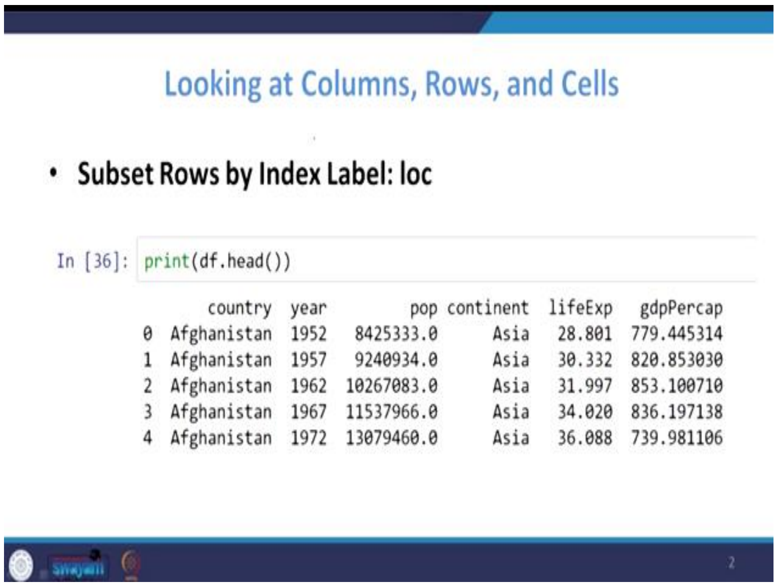
Data Analytics with Python

Prof. Ramesh Anbanandam
Department of management studies
Indian Institute of Technology, Roorkee

Lecture No 3
Python fundamentals

Okay? We will continue our lecture. How to access different rows and columns, because, it is very important applications.

(Refer Slide Time 00:33)



Looking at Columns, Rows, and Cells

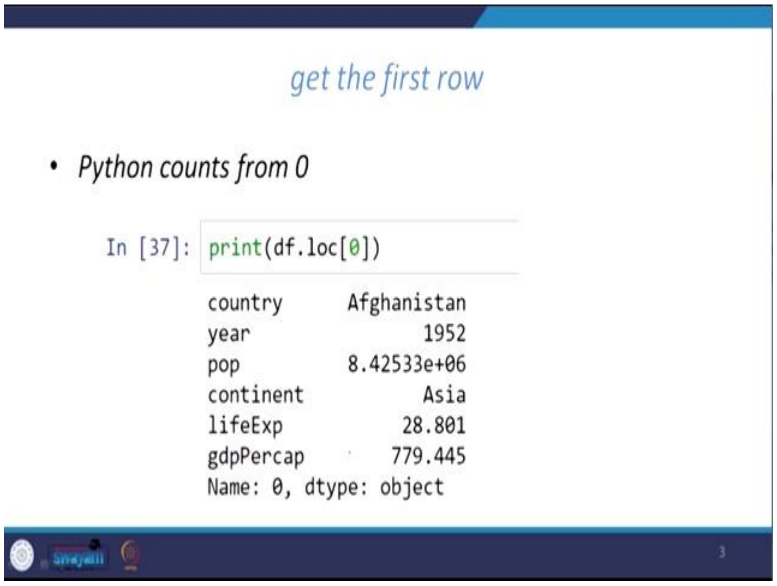
- **Subset Rows by Index Label: loc**

```
In [36]: print(df.head())
```

```
       country  year         pop continent  lifeExp   gdpPercap
0  Afghanistan  1952   8425333.0      Asia   28.801  779.445314
1  Afghanistan  1957   9240934.0      Asia   30.332  820.853030
2  Afghanistan  1962  10267083.0      Asia   31.997  853.100710
3  Afghanistan  1967  11537966.0      Asia   34.020  836.197138
4  Afghanistan  1972  13079460.0      Asia   36.088  739.981106
```

When the data file is very big sometimes you need to access only some rows or some columns for your calculation purpose. That we will learn how to access a particular rows or particular columns there is looking at columns, rows and cells. When you look at this see print df.head when I use this command and getting there are different. There for example; the first column says 0, 1, 2, 3, 4, country, year, population, continent, life expectations, gdppercapita.

(Refer Slide Time: 01:04)


get the first row

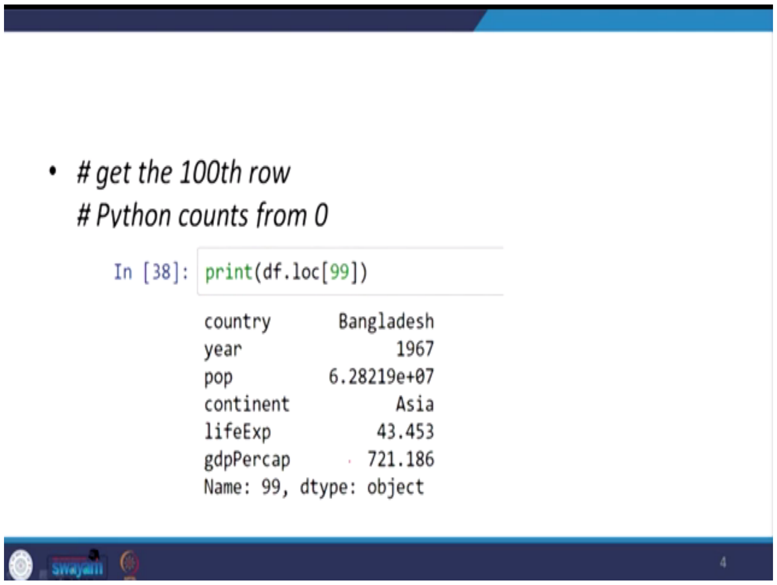
- Python counts from 0

```
In [37]: print(df.loc[0])

country       Afghanistan
year                 1952
pop           8.42533e+06
continent            Asia
lifeExp            28.801
gdpPercap         779.445
Name: 0, dtype: object
```

Suppose I want to get the first row as we know that the Python counts from 0. If you want to know the first row you type a print df.loc, it is a location in square bracket 0. Will do that you will get the details which are there in the first row.

(Refer Slide Time: 01:26)

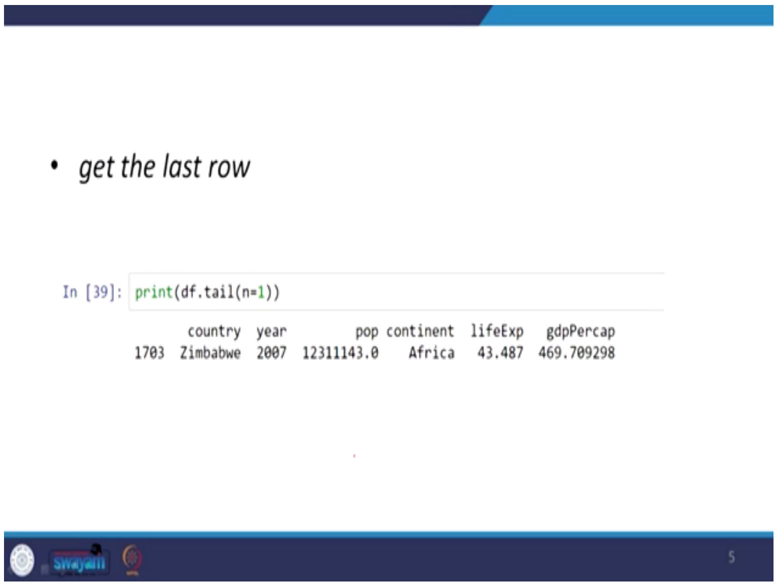

- # get the 100th row
  # Python counts from 0

```
In [38]: print(df.loc[99])

country       Bangladesh
year                1967
pop          6.28219e+07
continent           Asia
lifeExp           43.453
gdpPercap        721.186
Name: 99, dtype: object
```

So first if I want to know hundredth row so printed df.loc 99. We knew that python count from zero. If I want to know 100th row you have to type 99. So it should be in Square bracket you can see the details in the 100th row.

(Refer Slide Time: 01:42)



- *get the last row*

```
In [39]: print(df.tail(n=1))

         country  year         pop continent  lifeExp   gdpPercap
1703  Zimbabwe  2007  12311143.0    Africa   43.487  469.709298
```

Suppose we want to know the last row in the data set. So print df.tail n equal to 1. If you type n equal to – 1, it will not work, that we will see why if you want to know the last row simply type to df.tail n equal to 1, you will get to know that what is the last two, So we will see that.

(Video Starts: 02:01)

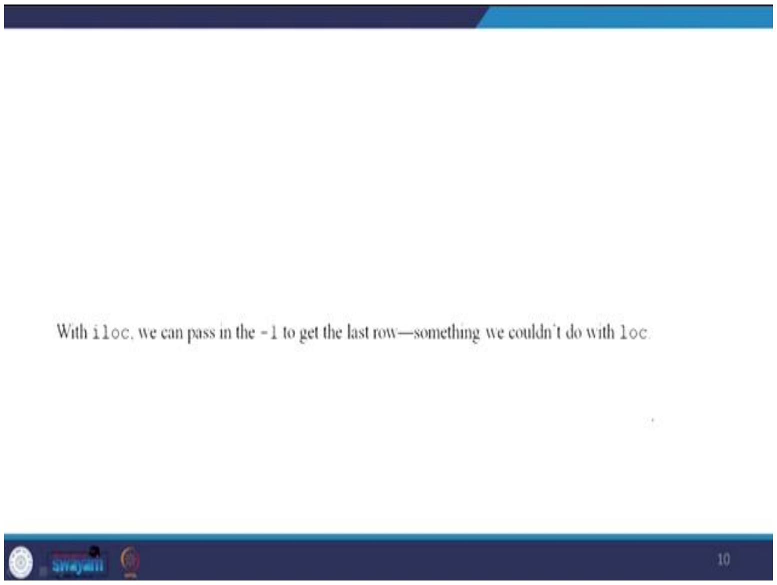
Now we are going to use this command to see the last row that is a detail about the last rows. Now we can subset a multiple rows at a time. For example; there will be requirement we have to select 100th row, 1st row 100th rows and 1000th rows. For that purpose you type this command print df.loc. You see there are two square brackets 0, 99, 999, you will see what output where getting. So type print df.loc. Yes, so we are able to see the 1st row, 100th row, 1000th row.

There is another way we can subset rows by row number by using this command iloc. Previously loc, now we are going to use iloc. Suppose for type I want to get the 2nd row, if I type print df.iloc 1. I will get the details about the 2nd row. Okay? Yeah, this is a detail about the 2nd row. Suppose I want to know 100th row by using iloc command so go there. Yes? That is the details about the 100th row. You see that if I want to access the last row by using iloc command.

So you can directly type print df.iloc in squared bracket - 1. So that will be the details of the last row. So what you can do we can open our Excel file you can verify what was the title, the last row and soon.

(Video Ends: 04:27)

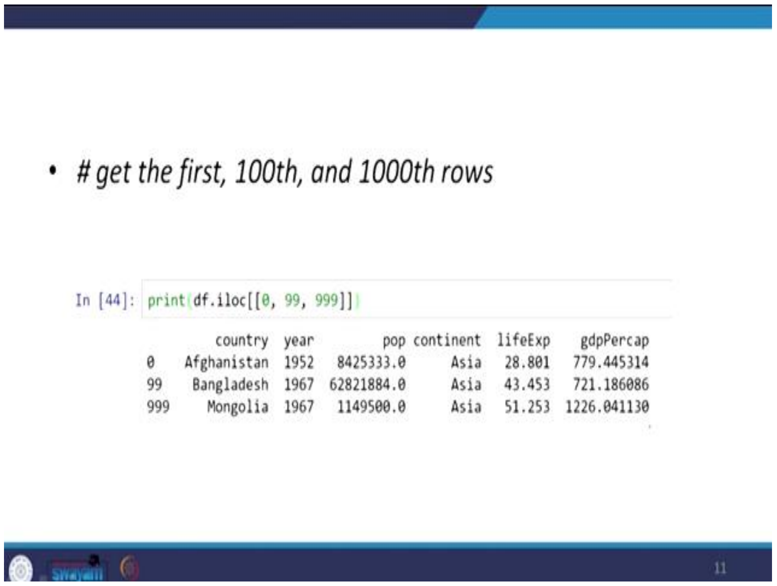
(Refer Slide Time: 04:27)



With iloc, we can pass in the -1 to get the last row—something we couldn't do with loc.

See then important note here with iloc command. We can pass in the - 1 to get the last row, but same thing that we could not do with loc. That is the difference between loc command and iloc command.

(Refer Slide Time: 04:42)



- *# get the first, 100th, and 1000th rows*

```
In [44]: print(df.iloc[[0, 99, 999]])

         country  year         pop continent  lifeExp    gdpPercap
0    Afghanistan  1952   8425333.0      Asia   28.801   779.445314
99    Bangladesh  1967  62821884.0      Asia   43.453   721.186086
999     Mongolia  1967   1149500.0      Asia   51.253  1226.041130
```

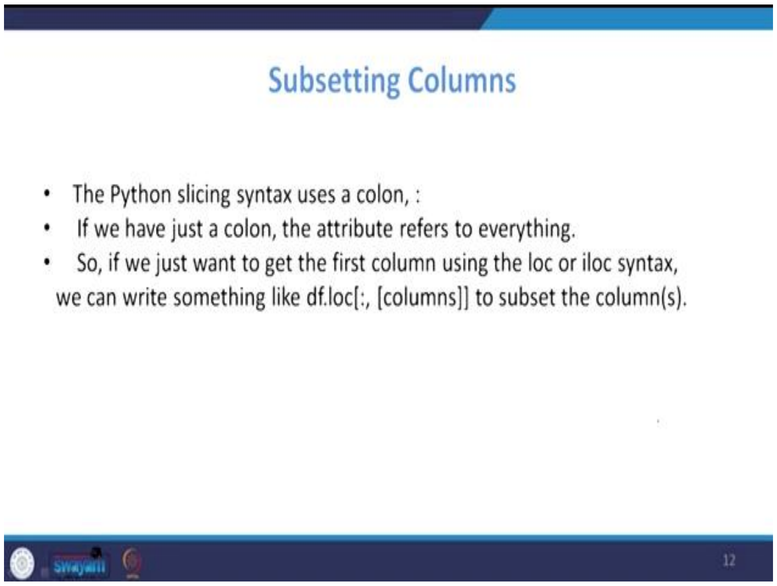
Suppose we want to get the first 100th and 1000th rows, using iloc command. So we are going to type this print df.iloc 0, 99, 999. Let us see what answer we are getting.

(Video Starts: 04:58)

Yes? See, we have getting 1st, 100th and 1000th row.

(Video Ends: 05:21)

(Refer Slide Time: 05:21)



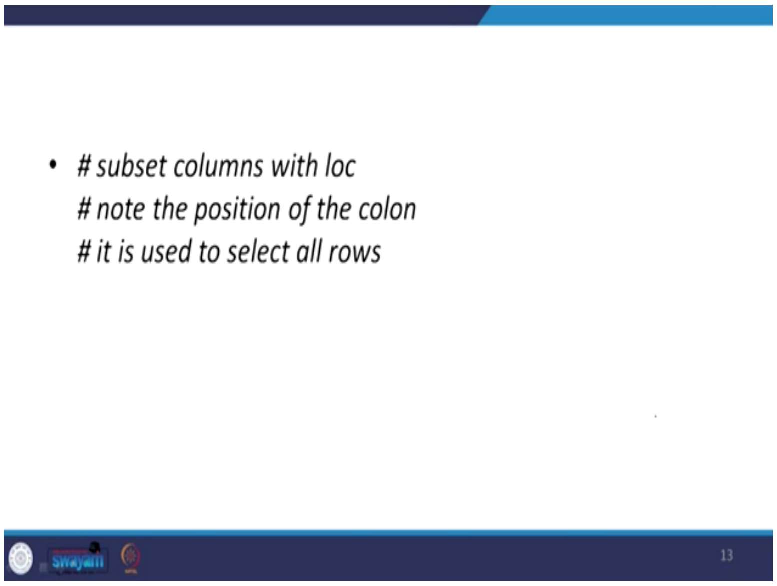
Subsetting Columns

- The Python slicing syntax uses a colon, :
- If we have just a colon, the attribute refers to everything.
- So, if we just want to get the first column using the loc or iloc syntax, we can write something like df.loc[:, [columns]] to subset the column(s).

12

So far we are seeing subsetting rows. Now we will see subsetting columns, the Python slicing syntax used a colon, colon represents all the rows. If you have just a colon that attribute refers to everything. So if you just want to get the first columns using a loc, or iloc syntax. We can write something like df.loc[: , which column we need to refer, to subset the columns.

(Refer Slide Time: 05:49)

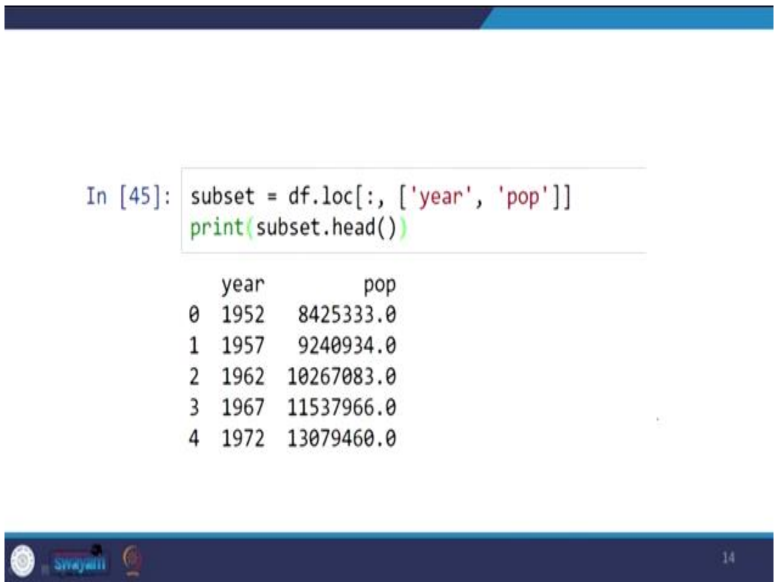


- *# subset columns with loc*
  # note the position of the colon
  # it is used to select all rows

13

The next slide I need to show that we are going to subset the columns with loc, not the position of the colon. It is used to select all rows.

(Refer Slide Time: 05:58)



```
In [45]: subset = df.loc[:, ['year', 'pop']]
         print(subset.head())

            year         pop
         0  1952   8425333.0
         1  1957   9240934.0
         2  1962  10267083.0
         3  1967  11537966.0
         4  1972  13079460.0
```

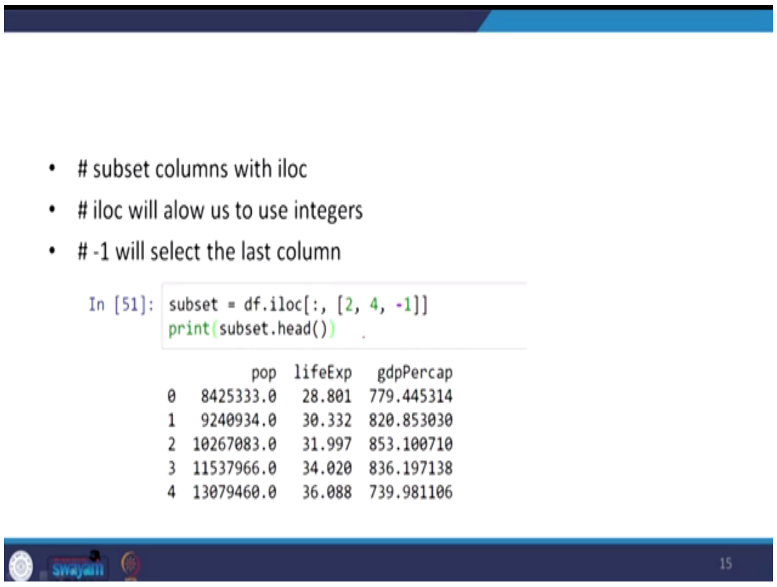
You see that, subset equal df.loc: , I want to see only two columns that is year and population. So when you type this way you will get all the rows only two columns details that is year and population. You will type this so when you type print subset.head. You can get the first 5 rows. So you will see how it appearing.

(Video Starts: 06:26)

Subsets equal to Subset is object because from the df is the initial object which has all the details. Now I am going to fetch only few columns from the df object that I am going to saved in the name subset, subset is the object. So all the rows but I need only year column and population column so I am going to type I want to see the first 5 rows, see that I am able to see 1st 5 rows, only for 2 cells. That is year and population. This is the way to get only 2 cells from the 2 columns from the Big Files.

(Video Ends: 07:21)

(Refer Slide Time: 07:21)



- # subset columns with iloc
- # iloc will alow us to use integers
- # -1 will select the last column

```
In [51]: subset = df.iloc[:, [2, 4, -1]]
         print(subset.head())

                pop  lifeExp   gdpPercap
         0   8425333.0   28.801  779.445314
         1   9240934.0   30.332  820.853030
         2  10267083.0   31.997  853.100710
         3  11537966.0   34.020  836.197138
         4  13079460.0   36.088  739.981106
```

15

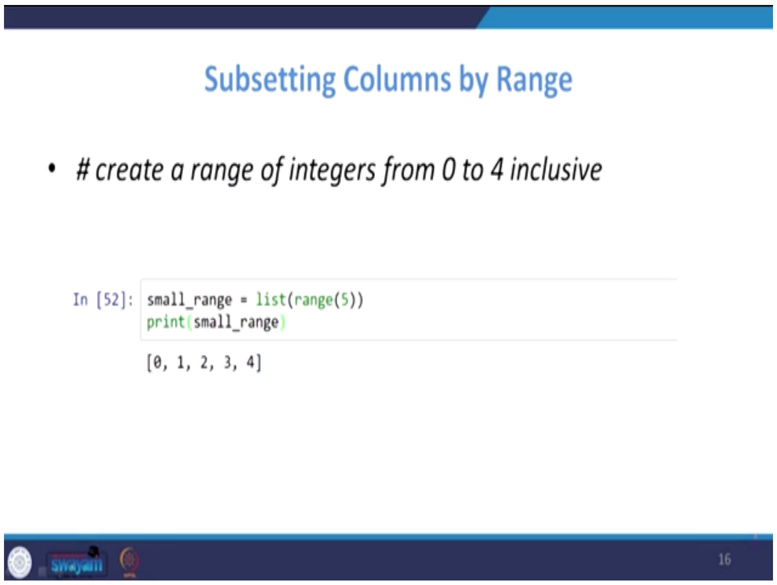
There is another example subset column with iloc, iloc will allow us to use integers - 1 will select the last column. The same thing whatever we have seen in the previously so subset equal to df.iloc:, represents all the rows. Then [2 , 4 , - 1, then we can see by using this command print subset.head 1st 5 rows.

(Video Starts: 07:46)

See that we are able to see the last column and the population column, life expectancy column. You can open our Excel sheet you can verify whether we are getting the right answer or not.

(Video Ends: 08:26)

(Refer Slide Time: 08:26)



Subsetting Columns by Range

- *# create a range of integers from 0 to 4 inclusive*

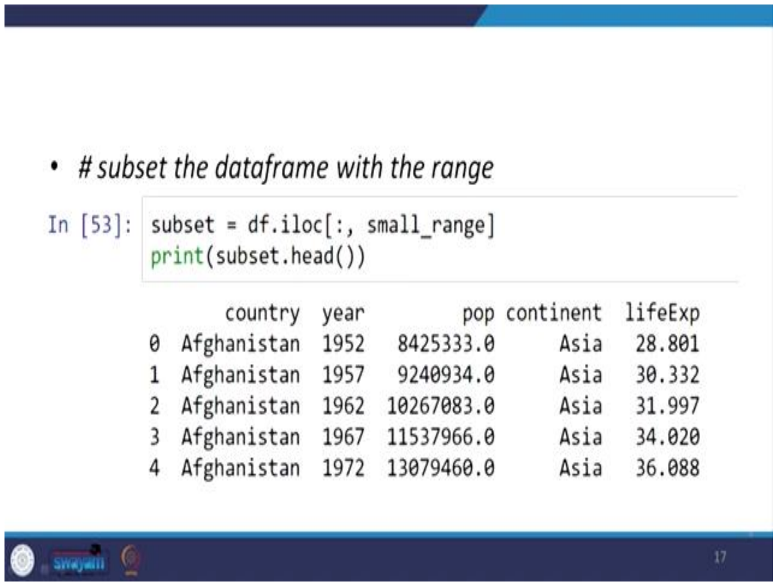
```
In [52]: small_range = list(range(5))
         print(small_range)

         [0, 1, 2, 3, 4]
```

16

Sometime there is another way for subsetting columns by using the command called range. First will make range of numbers we are going to save that range of a number in object called small _ range, so small _ range equal to list range 5. Print small range will get 0, 1, 2, 3, and 4. Now this small _ range, object can be used to access the corresponding columns.

(Refer Slide Time: 08:57)



- *# subset the dataframe with the range*

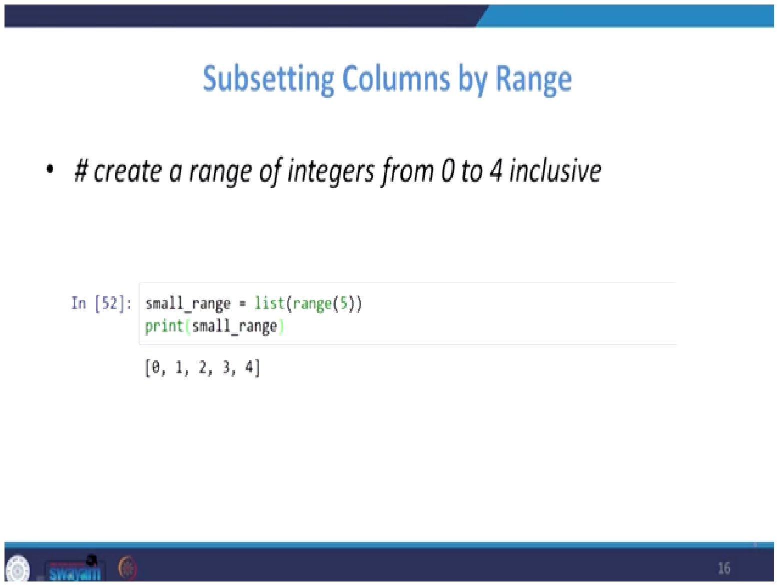
```
In [53]: subset = df.iloc[:, small_range]
         print(subset.head())

        country  year         pop continent  lifeExp
0   Afghanistan  1952   8425333.0      Asia   28.801
1   Afghanistan  1957   9240934.0      Asia   30.332
2   Afghanistan  1962  10267083.0      Asia   31.997
3   Afghanistan  1967  11537966.0      Asia   34.020
4   Afghanistan  1972  13079460.0      Asia   36.088
```

17

So if I type a subset equal to df.iloc:, small _ range I can get.

(Refer Slide Time: 09:04)



Subsetting Columns by Range

- *# create a range of integers from 0 to 4 inclusive*

```
In [52]: small_range = list(range(5))
         print(small_range)

         [0, 1, 2, 3, 4]
```

16

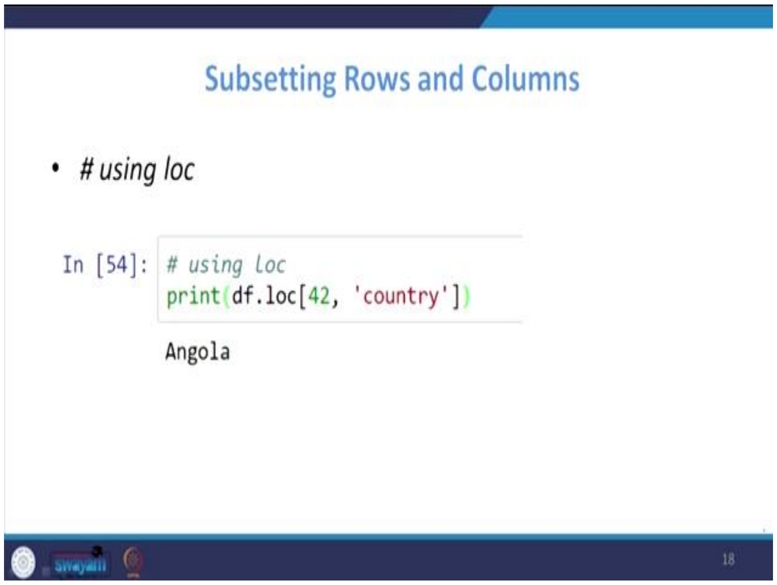
1st column, 2nd column, 3rd column, 4th column and 5th column, so we will try this.

(Video Starts: 09:09)

Small _ range is an object, we are going to create a range. Suppose we want to see what small _range is. So it is up to 0 to 4, that means 1 to 5. Now we are going to subset using that object called small _ range using ilocation command. df.ilocation:small_ range we see that here we are able to see 5 column that is a country, year, population, continent and life expectancy.

(Video Ends: 10:21)

(Refer Slide Time: 10:22)



Subsetting Rows and Columns

- *# using loc*

```
In [54]: # using Loc
         print(df.loc[42, 'country'])

         Angola
```

18

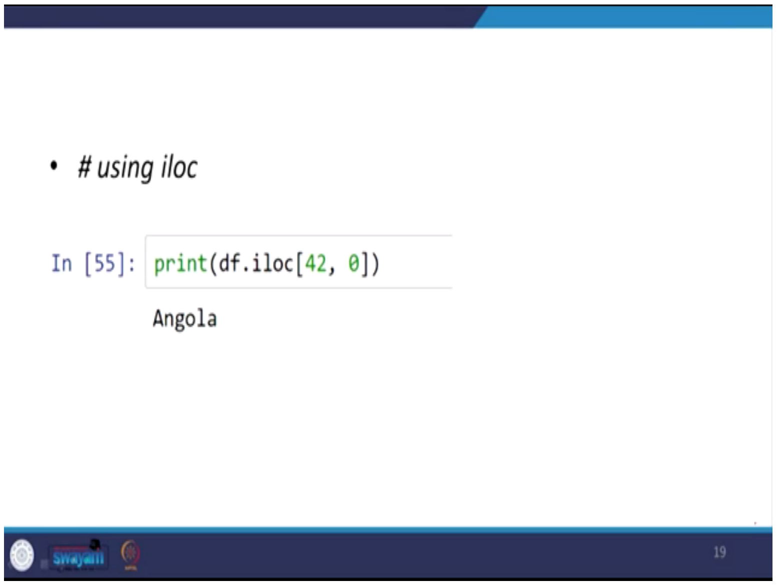
So far we have seen subsetting only rows and columns. Now we are going to subset rows and columns simultaneously. For example; using loc command so if you type print df.loc 42 countries. We can check in the 42 label in country columns. What is the cell name, there cell name is Angola. Will try this.

(Video Starts: 10:47)

Going to see in that file in 42nd label in country column what value is there so that is an Angola, Yes?

(Video Ends: 11:09)

(Refer Slide Time: 11:09)



- *# using iloc*

```
In [55]: print(df.iloc[42, 0])

         Angola
```

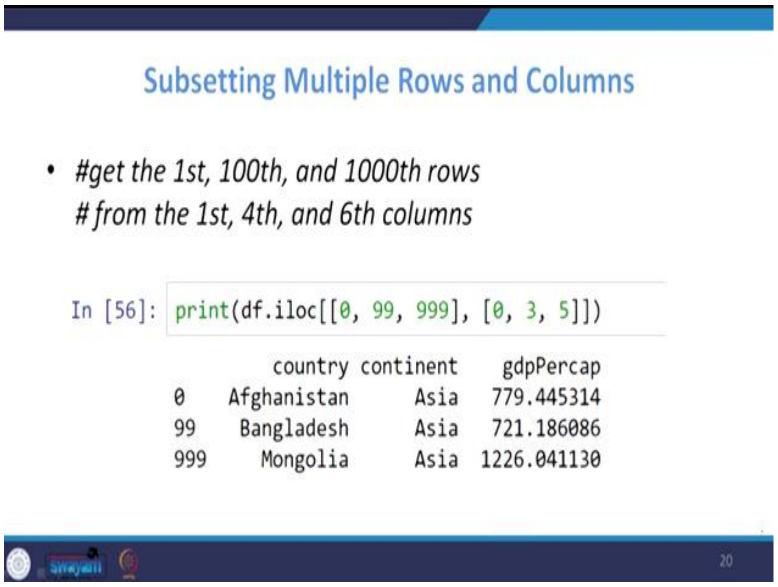
Yes, we can see what is in the using the same ilocation we can see in 42nd label in 0th column. Now we can represent column also with 0 columns, what value it is, you will see that. You can verify you have to get to the answer. You can open the Excel file. You can verify we are correctly accessing the cell or not.

(Video Starts: 11:29)

Print df.iloc in 42nd label 0th column what is the value it is Angola.

(Video Ends: 11:46)

(Refer Slide Time: 11:46)



Subsetting Multiple Rows and Columns

- *#get the 1st, 100th, and 1000th rows*
  # from the 1st, 4th, and 6th columns

```
In [56]: print(df.iloc[[0, 99, 999], [0, 3, 5]])

                country continent    gdpPercap
         0    Afghanistan      Asia   779.445314
         99    Bangladesh      Asia   721.186086
         999     Mongolia      Asia  1226.041130
```

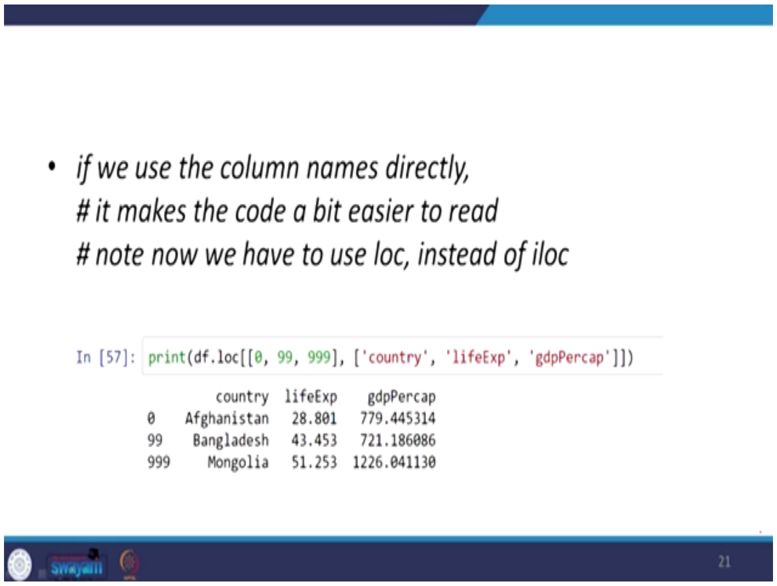
Next we can subset multiple rows and columns. For example; get the 1st, 100th and 100th rows from the 1st, 4th and 6th column. So now we are going too simultaneously we are going to fetch rows and columns and corresponding cells. So print to df.iloc 0, 99, 999. Similarly column labels is 0, 3, 5. Let us see what answer.

(Video Starts: 12:13)

This accessing rows and columns are very important functions because nowadays data file comes with a lot of rows and lot of columns. We need not use all the columns, all the rows for further analysis. Sometimes we need only specific rows or specific columns. So these basic commands will help you, how to access a particular rows and columns, that will be very useful when we do further analysis using Python. Yeah? This is the value so that means 1st row, 100th row 1000th row, 1st column and soon.

(Video Starts: 13:08)

(Refer Slide Time: 13:08)



- *if we use the column names directly,*
  # it makes the code a bit easier to read
  # note now we have to use loc, instead of iloc

```
In [57]: print(df.loc[[0, 99, 999], ['country', 'lifeExp', 'gdpPercap']])

         country  lifeExp    gdpPercap
0    Afghanistan   28.801   779.445314
99    Bangladesh   43.453   721.186086
999     Mongolia   51.253  1226.041130
```

21

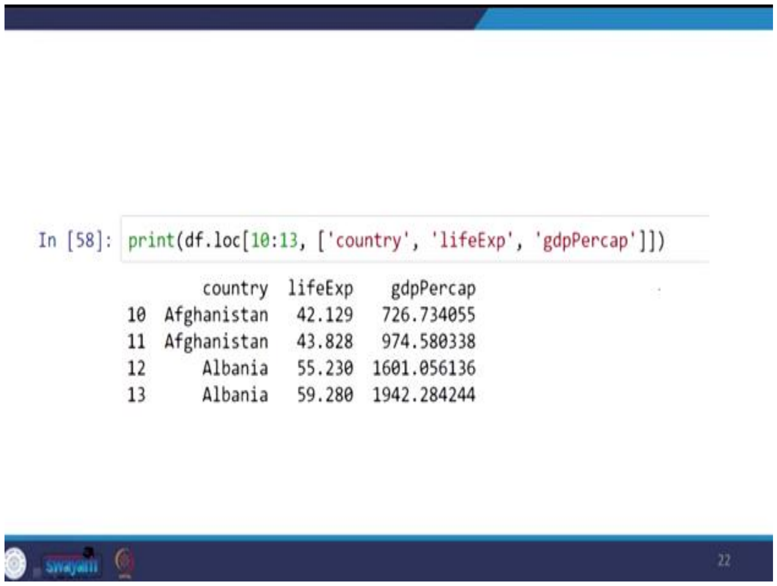
And there is another way if you use the column names directly it makes the code a bit easier to read. In terms of number and so you see number column. If you use for representing column, if you use column name we can see what is there, so simply type the column name. So we use this command, print df.loc 0, 99, 999. Then directly will type the column name country, life expectancy, gdpPercap you see there is a square bracket here.

(Video Starts: 13:36)

That you have to do as the same that Life capital Exp, Yes? This is because country, life expectation this is the easy way to because we cannot remember column name.

(Video Ends 14:48)

(Refer Slide Time 14:49)



```
In [58]: print(df.loc[10:13, ['country', 'lifeExp', 'gdpPercap']])

        country  lifeExp    gdpPercap
10  Afghanistan   42.129   726.734055
11  Afghanistan   43.828   974.580338
12      Albania   55.230  1601.056136
13      Albania   59.280  1942.284244
```

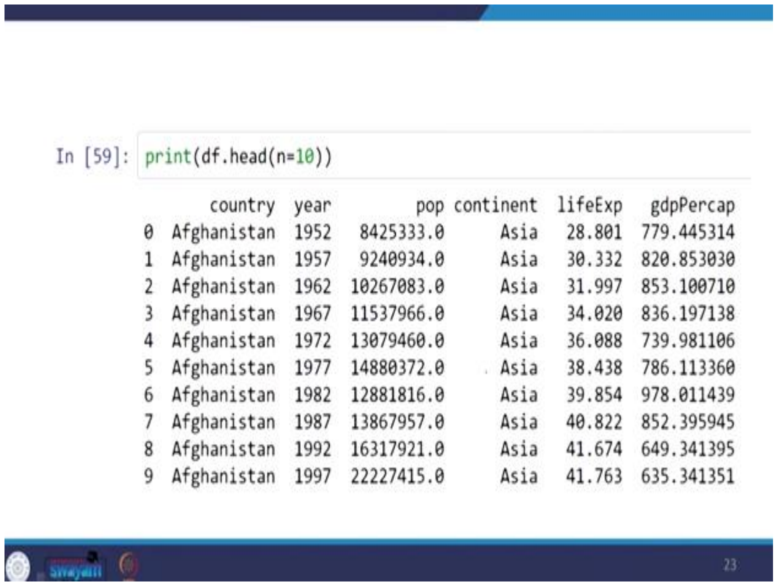
This was not only that instead of see suppose if you put a 10 column 13 that corresponding rows will be displayed. So print df.loc 10 to 13, the 10th row 11th, row 12th, row 13th, row will be shown and in columns country and life expectancy and gdpPercap so we will try this command.

(Video Starts: 15:11)

That means we can see the range of rows at a time, gdpPercap. You are able to see the 10th row, 11th, 12th and 13th.

(Video Ends: 16:17)

(Refer Slide Time: 16:17)

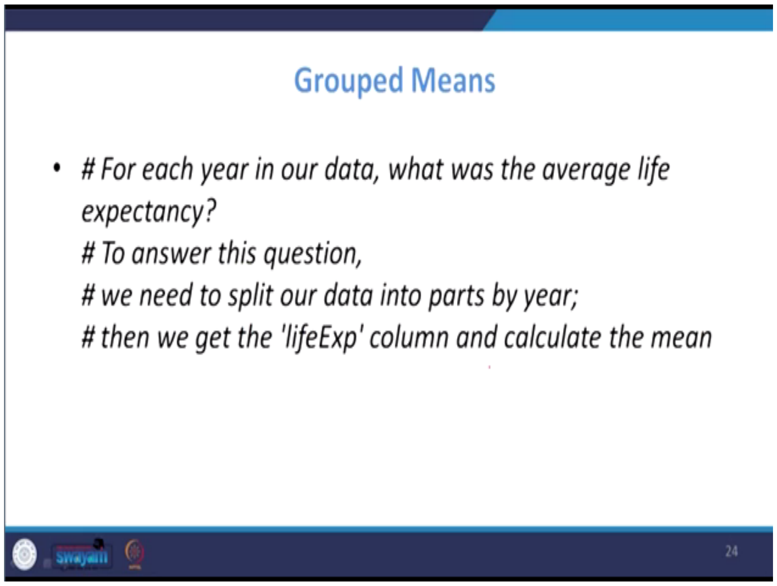


```
In [59]: print(df.head(n=10))

       country  year         pop continent  lifeExp   gdpPercap
0  Afghanistan  1952   8425333.0      Asia   28.801  779.445314
1  Afghanistan  1957   9240934.0      Asia   30.332  820.853030
2  Afghanistan  1962  10267083.0      Asia   31.997  853.100710
3  Afghanistan  1967  11537966.0      Asia   34.020  836.197138
4  Afghanistan  1972  13079460.0      Asia   36.088  739.981106
5  Afghanistan  1977  14880372.0      Asia   38.438  786.113360
6  Afghanistan  1982  12881816.0      Asia   39.854  978.011439
7  Afghanistan  1987  13867957.0      Asia   40.822  852.395945
8  Afghanistan  1992  16317921.0      Asia   41.674  649.341395
9  Afghanistan  1997  22227415.0      Asia   41.763  635.341351
```

Okay? Next see print df. head we can see we can able to see 1st, 10 rows.

(Refer Slide Time: 16:23)



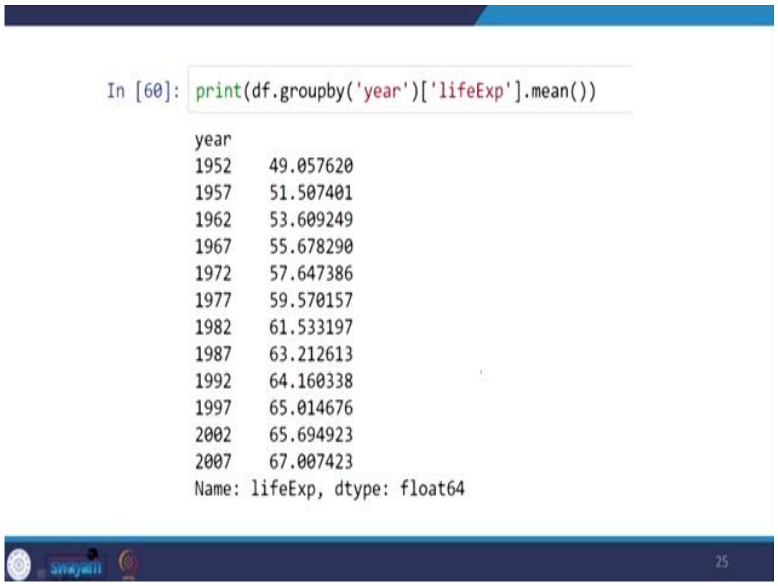
Grouped Means

- # For each year in our data, what was the average life expectancy?
  # To answer this question,
  # we need to split our data into parts by year;
  # then we get the 'lifeExp' column and calculate the mean

24

The 10th row some time for each year in our data what was the average life expectancy. To answer this question we need to split our data into parts per year and then we can get the life expectancy column and calculate the mean.

(Refer Slide Time: 16:38)



```
In [60]: print(df.groupby('year')['lifeExp'].mean())

year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

25

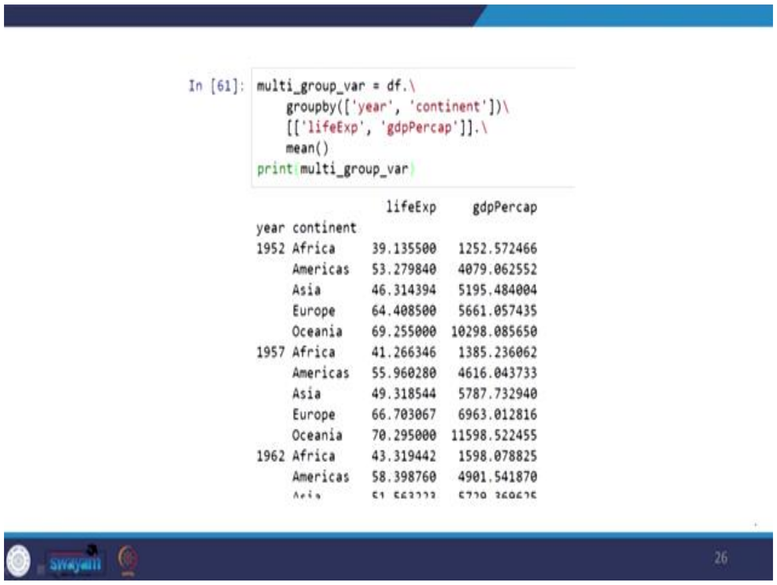
So what is happening there is a command which I go to use called groupby, we look at the data it is not grouped. So when you use this command print df.groupby year,and life expectancy and corresponding mean. The mean of the on the in the year 1952, the mean of the life expectancy variable is 49.05. In 57, 51.09. We look at the data; it is not in this order. So the groupby by year this command is grouping all the values, with respect to year. So we will see what is the answer for this, we will verify this.

(Video Starts: 17:15)

When you open that Excel file you will see that the Excel file will be in some other form it is not grouped by year, different years are appearing at different places. So this command that is a group by will help you to group the data in year wise groupby. Yes, you see that you are able to get 1952 the life expectancy was 49 years you see that when you look at this data. When year increases the life expectancy year also increases due to advancement of medical facility available and the standard of life is also increasing.

(Video Ends: 18:42)

(Refer Slide Time: 18:42)



```
In [61]: multi_group_var = df.\
             groupby(['year', 'continent'])\
             [['lifeExp', 'gdpPercap']].\
             mean()
         print(multi_group_var)
```

```
                  lifeExp     gdpPercap
year continent
1952 Africa     39.135500   1252.572466
     Americas   53.279840   4079.062552
     Asia       46.314394   5195.484004
     Europe     64.408500   5661.057435
     Oceania    69.255000  10298.085650
1957 Africa     41.266346   1385.236062
     Americas   55.960280   4616.043733
     Asia       49.318544   5787.732940
     Europe     66.703067   6963.012816
     Oceania    70.295000  11598.522455
1962 Africa     43.319442   1598.078825
     Americas   58.398760   4901.541870
```

Now, we can form a stacked table. Stacked table is using the group by command. So you type this multi _ group _ variable = df . \ . See the \ represents to breaking the command we can use \. Otherwise you can write straightaway also no problem. df.group by year, continent, life expectancy,gdp per capita, then we can find the mean. Then we will get this output for that means in 1952, in Africa, the life expectancies 39, in America 53, in Asia 46 in Europe 64 will try this command.

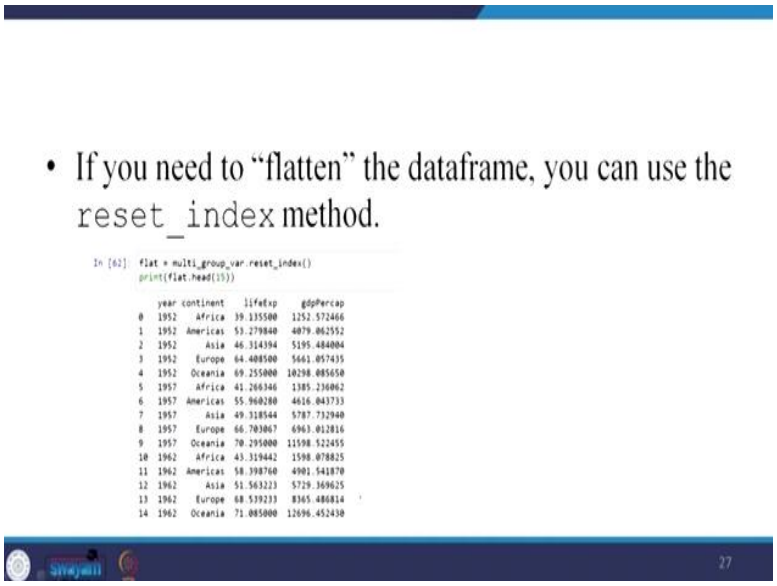
(Video Starts: 19:28)

When we takes these command you will get an output, that is a stacked table. That is very useful for interpreting the whole dataset, is kind of a way of summarizing the data in the form of table.

Multi_group. You see that now year wise. It is very, very useful command it is year by 1952, some country Africa. What was the average year 1957 Africa. We see that if you look at only the Africa data. 52 to 39 in 57 41, in 62 43, in 67 45, see that we can interpret this way, by looking at the, this table. Suppose you have to flatten this.

(Video Ends 21:24)

(Refer Slide Time: 21:24)



- If you need to "flatten" the dataframe, you can use the reset_index method.

```
In [62]: flat = multi_group_var.reset_index()
         print(flat.head(15))

    year continent    lifeExp     gdpPercap
0   1952    Africa  39.135500   1252.572466
1   1952  Americas  53.279840   4079.062552
2   1952      Asia  46.314394   5195.484004
3   1952    Europe  64.408500   5661.057435
4   1952   Oceania  69.255000  10298.085650
5   1957    Africa  41.266346   1385.236062
6   1957  Americas  55.960280   4616.043733
7   1957      Asia  49.318544   5787.732940
8   1957    Europe  66.703067   6963.012816
9   1957   Oceania  70.295000  11598.522455
10  1962    Africa  43.319442   1598.078825
11  1962  Americas  58.398760   4901.541870
12  1962      Asia  51.563223   5729.369625
13  1962    Europe  68.539233   8365.486814
14  1962   Oceania  71.085000  12696.452430
```

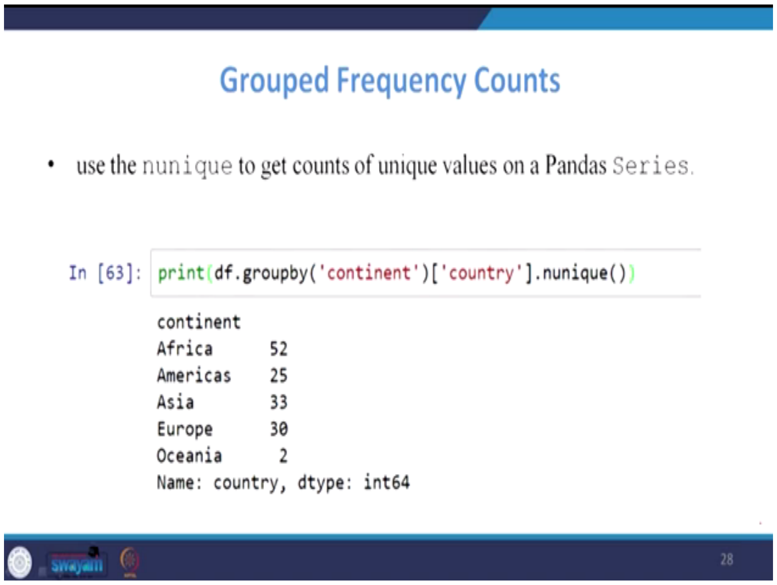
If, you need to flatten the data frame. You can use this reset underscore index method, just to type flat = multi _ group _ var . reset _ index. Then you see now the data is again. Now it is flattened. The same data set, which was it in the table form now it in the simple learned form. So we will try this comment.

(Video Starts 21:48)

This is what you are doing the data manipulation, because from the big data file, we have to learn this kind of fundamental data manipulation methods that will be very useful, in coming classes. So able to use reset _ index command to flatten the, that stacked table. See that now we can see first 15 rows. Now it is data is flattened into the normal form.

(Video Ends 22:41)

(Refer Slide Time: 22:41)



Grouped Frequency Counts

- use the `nunique` to get counts of unique values on a Pandas `Series`.

```
In [63]: print(df.groupby('continent')['country'].nunique())

continent
Africa      52
Americas    25
Asia        33
Europe      30
Oceania      2
Name: country, dtype: int64
```

28

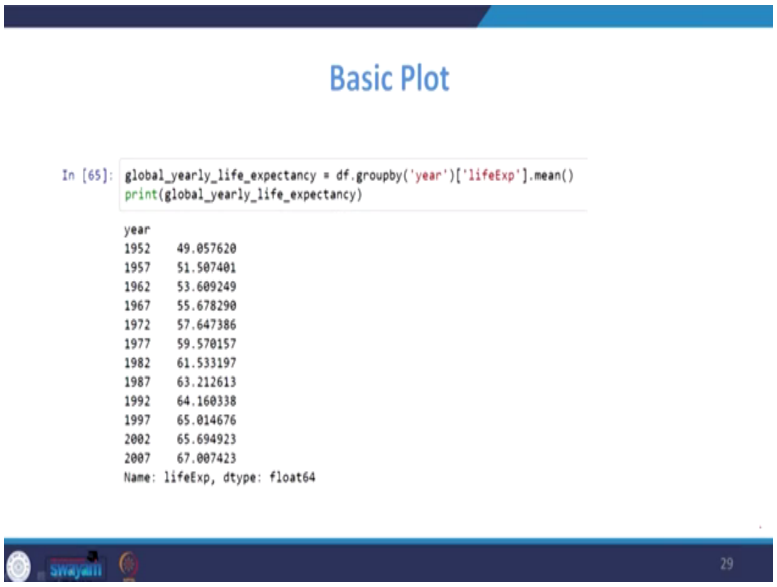
The next one is grouped frequency counts. By using nunique command, we can get a count of unique values on the panda series. So when you type print df. groupby continent, country. nunique, you can get unique values the new frequency. Okay, will try this command.

(Video Starts: 23:04)

Print, See Africa 52, America is 25, Asia 33. When you look at the data, again, you go to excel,Excel data you can interpret what is the 52 means, what is the America 25 and soon.

(Video Ends: 23:49)

(Refer Slide Time: 23:49)



Basic Plot

```
In [65]: global_yearly_life_expectancy = df.groupby('year')['lifeExp'].mean()
         print(global_yearly_life_expectancy)

year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

29

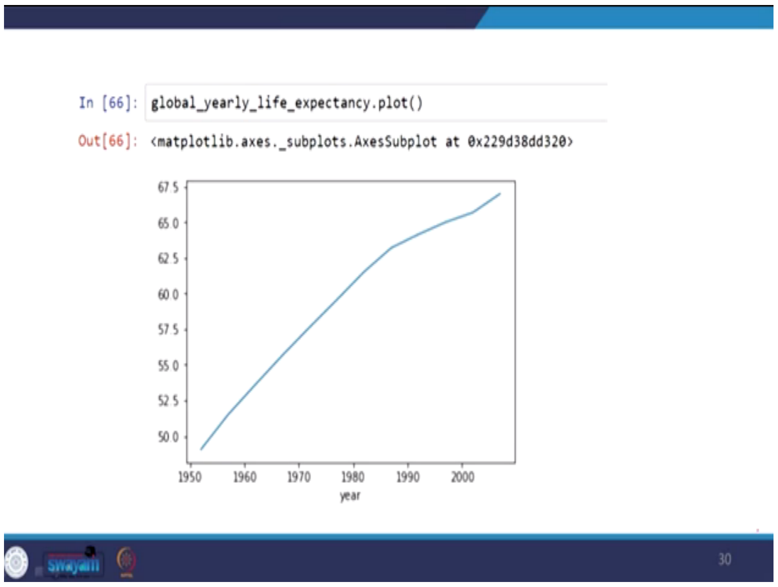
Now, some basic plot a way to construct two things one is year and life expectancy. So we are going to create a new object that is called Global _ yearly_ life _expectancy. By grouping year and life expectancy, with respect to its mean. Then we are going to print it. So you are going to get two values one is year. Next one is life expectancy. That is a mean life expectancy, you will see this.

(Video Starts: 24:17)

There is a new object. The object name is called Global _ yearly _ life expectancy. Yes, see that year, and supposed we want to plot it. We will see we are going to plot this data, how we are going to plot it.

(Video Ends: 25:28)

(Refer Slide Time: 25:28)



```
In [66]: global_yearly_life_expectancy.plot()
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x229d38dd320>
```

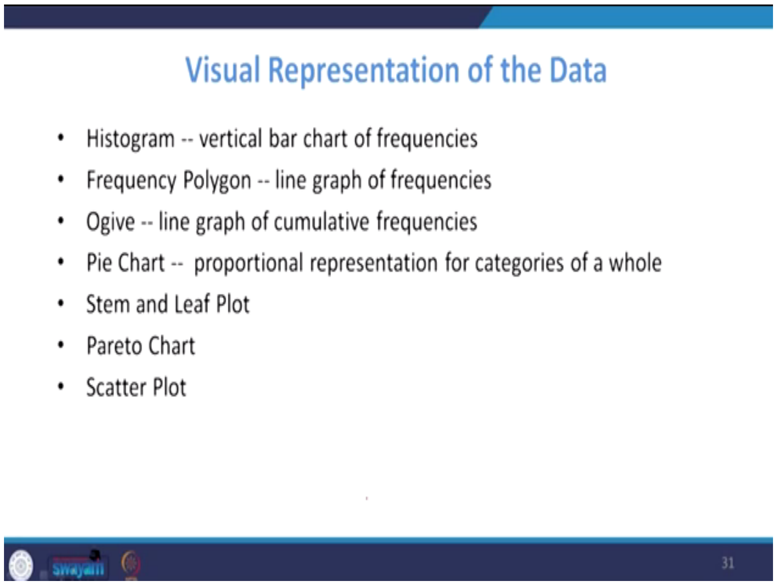
Simply, just that object name. plot. That automatically takes this was output, which I got it, in x axis in a year, in y axis, average life expectancy. We will run this.

(Video Starts 25:40)

So, what this data says that, when the year 1950 - 2000 you see when the year increases, the life expectancy also increases.

(Video Ends: 26:07)

(Refer Slide Time 26:07)



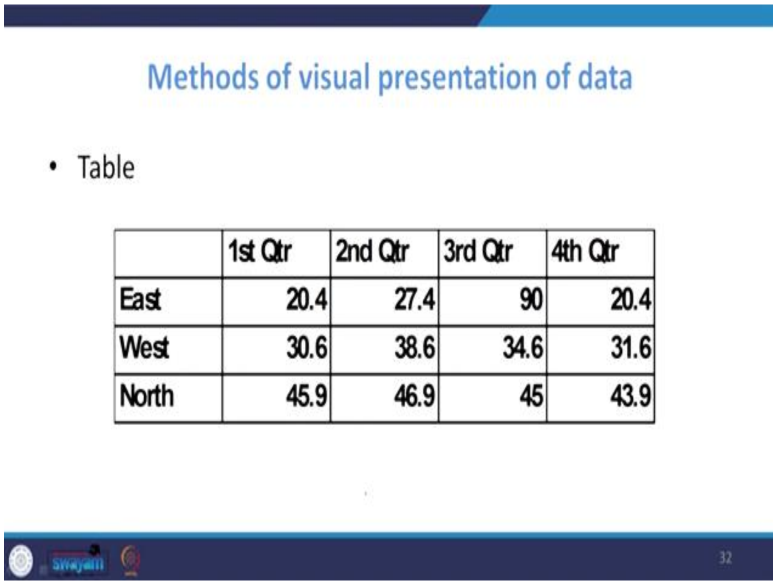
Visual Representation of the Data

- Histogram -- vertical bar chart of frequencies
- Frequency Polygon -- line graph of frequencies
- Ogive -- line graph of cumulative frequencies
- Pie Chart -- proportional representation for categories of a whole
- Stem and Leaf Plot
- Pareto Chart
- Scatter Plot

31

Just we have seen only the simple plot, in coming classes, we will see some of the visual representation of the data. We are going to see a histogram, frequency polygon, ogive curves,pie chart, stem and leaf plot and pareto chart and scatter plot .

(Refer Slide Time: 26:21)



Methods of visual presentation of data

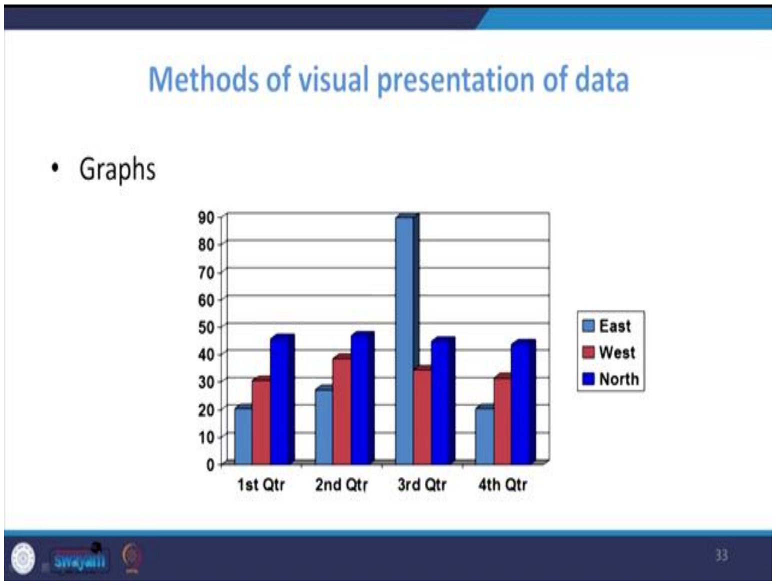
- Table

	1st Qtr	2nd Qtr	3rd Qtr	4th Qtr
East	20.4	27.4	90	20.4
West	30.6	38.6	34.6	31.6
North	45.9	46.9	45	43.9

32

Suppose, this is the data, see what is there in East, west, north. In column first quarter, second quarter, third quarter, fourth quarter.

(Refer Slide Time: 26:30)

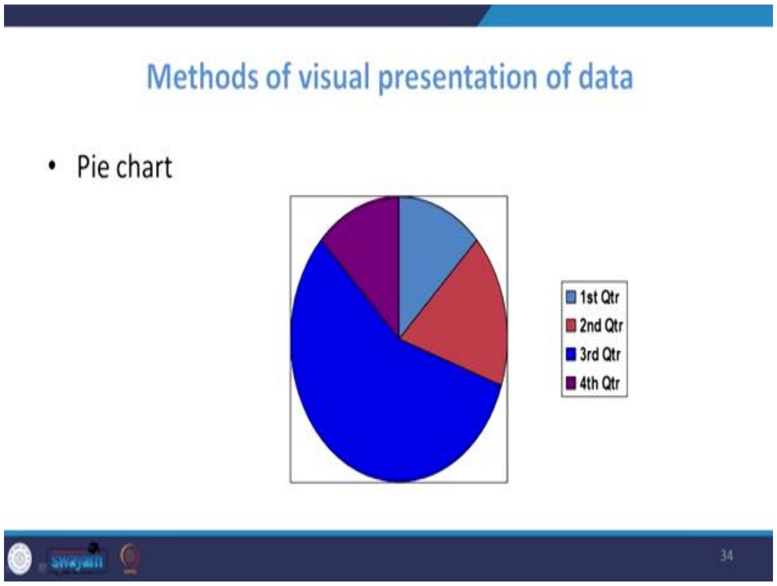


Methods of visual presentation of data

- Graphs

Suppose the very easiest way is the graph. By using this is called bar graph, bar chart. Bar chart is different regions are labeled as different colors. This is a method of visual representation of the data. If you look at this, the eastern side in third quarter, there are more sales. Okay.

(Refer Slide Time: 26:53)

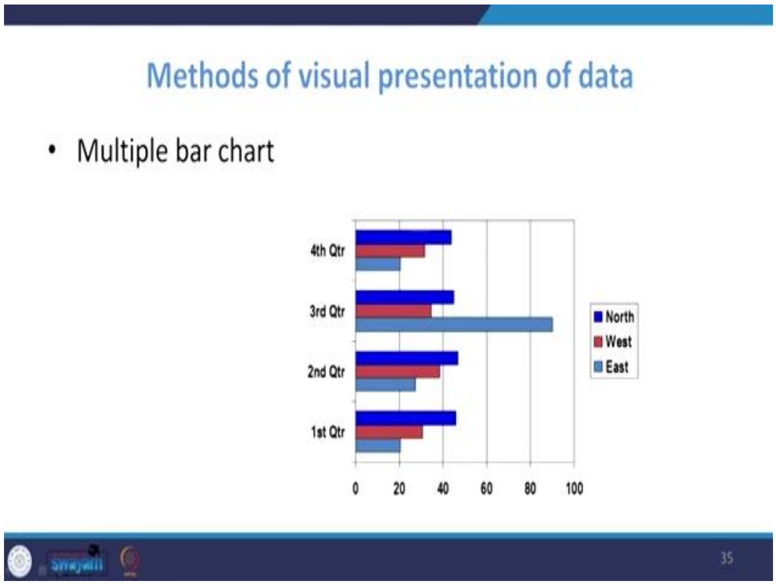


Methods of visual presentation of data

- Pie chart

The another way to represent visually, the data is pie chat, is the first quarter, third quarter. You look at this, third quarter, which is in blue in color. There are more sales. And most importantly the pie chart, we can get pie chart only for categorical variable. The variable is continuous, you cannot use bar chart, you cannot use pie chart. So the pie chart is used only for categorical variable. That is for only count data.

(Refer Slide Time: 27:31)

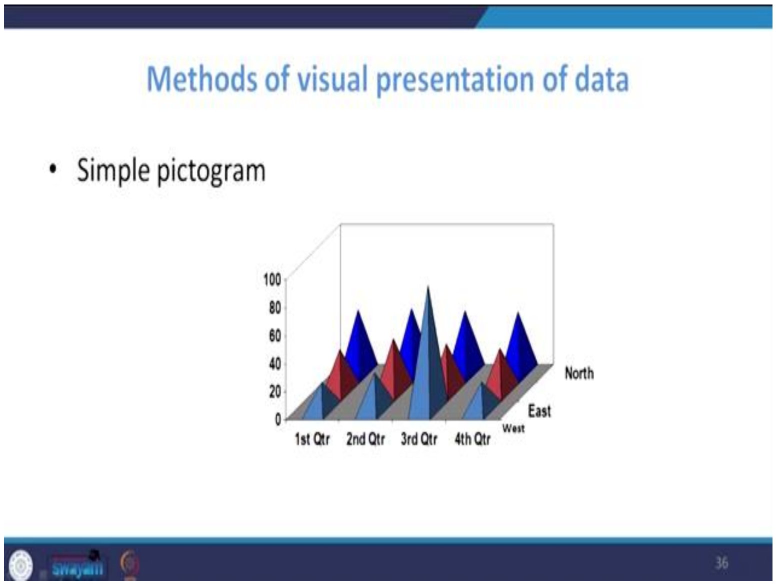


Methods of visual presentation of data

- Multiple bar chart

The another one is the Multiple bar chart. This is another way to represent the data visually.

(Refer Slide Time: 27:39)

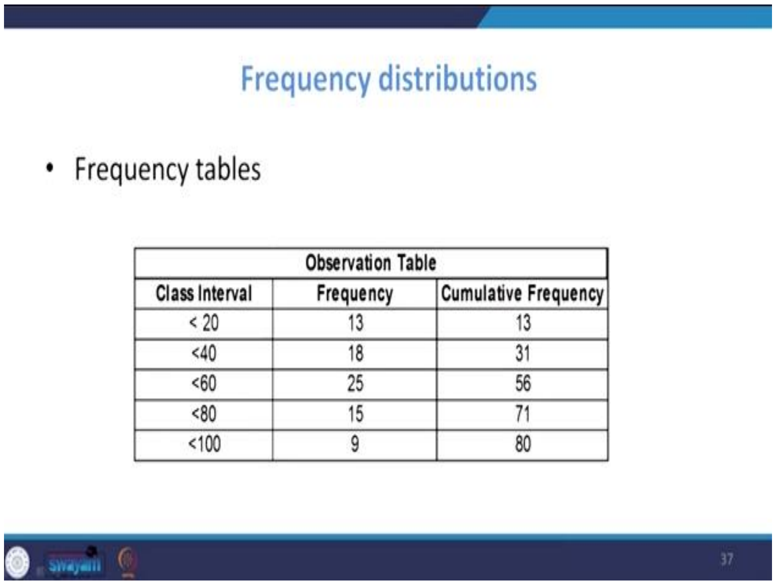


Methods of visual presentation of data

- Simple pictogram

Another one is a simple pictogram.

(Refer Slide Time: 27:43)



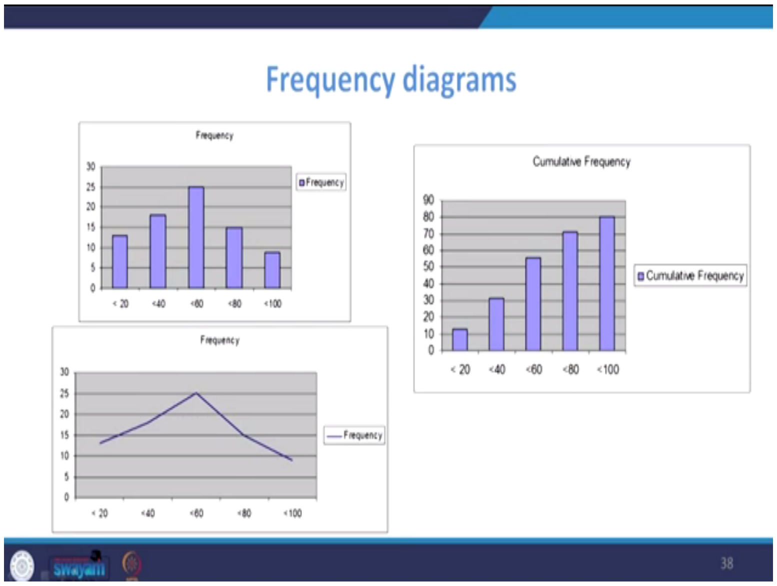
Frequency distributions

- Frequency tables

Observation Table		
Class Interval	Frequency	Cumulative Frequency
< 20	13	13
<40	18	31
<60	25	56
<80	15	71
<100	9	80

See, this is the frequency table.

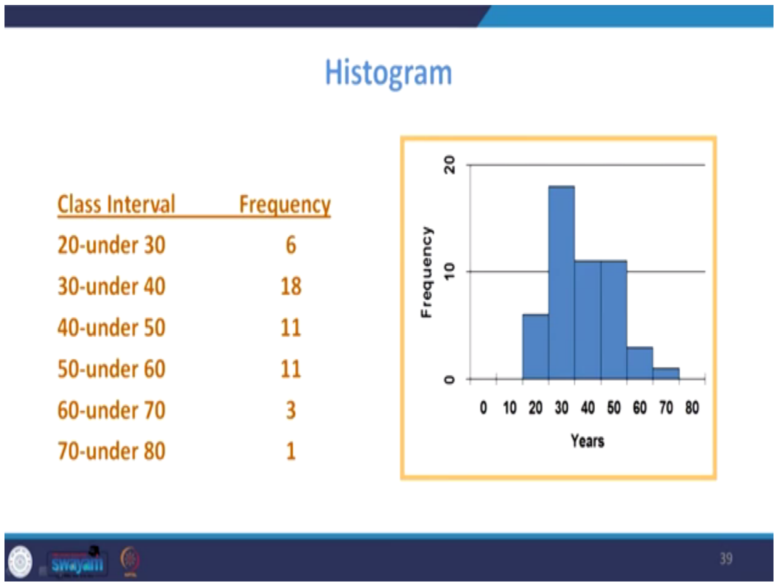
(Refer Slide Time: 27:25)



Frequency diagrams

See, next one is frequency polygon. This figure is drawn from the previous table, which was shown in the previous slide. So below 20 around 13,14. This represents frequency polygon. When you connect the midpoint, you see that this is the. This is called frequency polygon. Then the, this one is the cumulative frequency. It is not always, you cannot connect the midpoint, you have to be very careful with the data is continuous, then only you can connect one this bar. The data is not continuous, you cannot connect it.

(Refer Slide Time: 28:24)

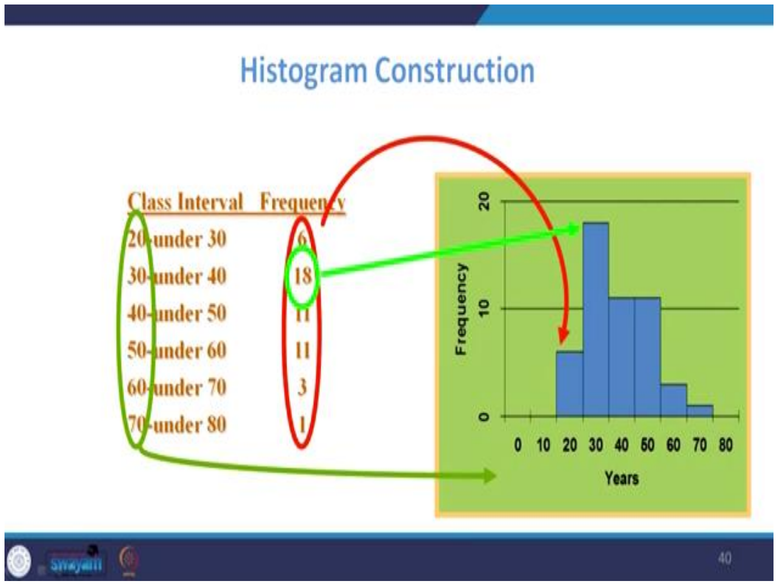


Histogram

Class Interval	Frequency
20-under 30	6
30-under 40	18
40-under 50	11
50-under 60	11
60-under 70	3
70-under 80	1

Next one is a histogram .The histogram was constructed from the given table. You see.

(Refer Slide Time: 28:30)

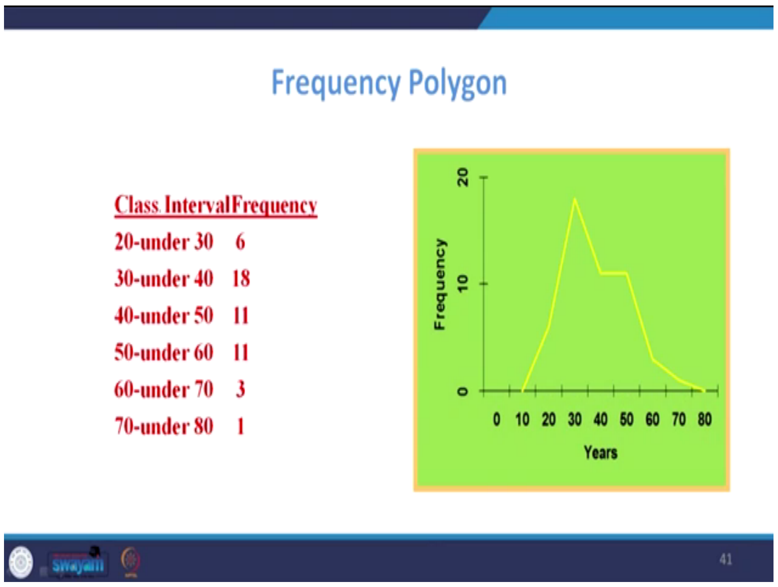


Histogram Construction

Class Interval	Frequency
20-under 30	6
30-under 40	18
40-under 50	11
50-under 60	11
60-under 70	3
70-under 80	1

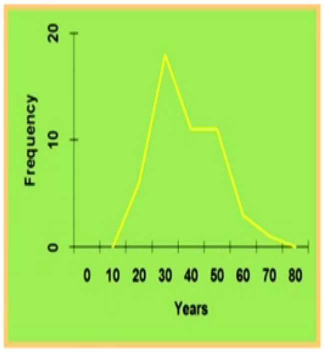
The lower limit of the table values is going to in x axis. The frequency is shown in the y axis. You see that this is data in continuous data. Okay, that was histogram. The purpose of histogram is, the histogram will give you a rough idea what is the nature of the data whether, what kind of distribution it follows. Whether it is following bell shaped curve, whether the data is skewed right or skewed left.

(Refer Slide Time: 29:03)



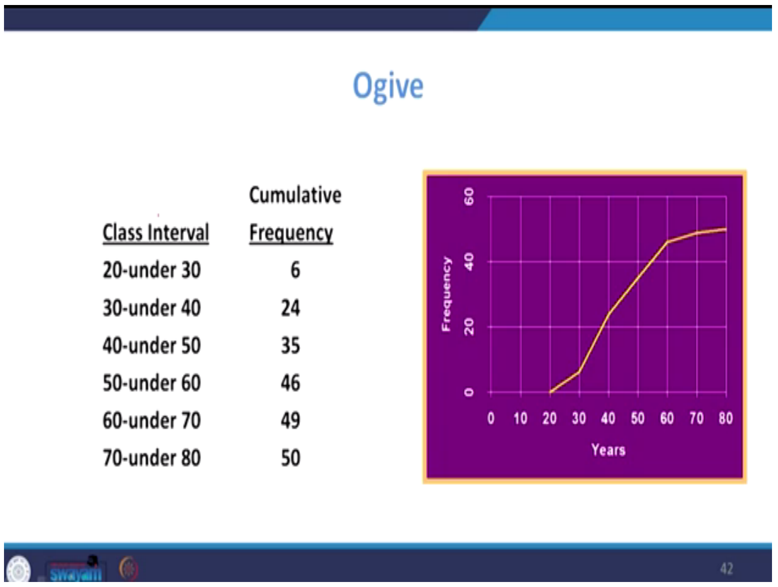
Frequency Polygon

Class Interval	Frequency
20-under 30	6
30-under 40	18
40-under 50	11
50-under 60	11
60-under 70	3
70-under 80	1



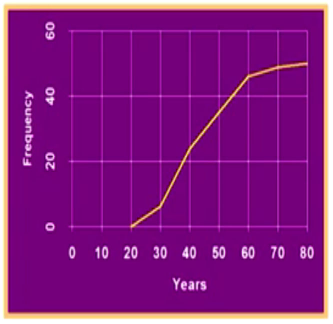
Next one is the frequency polygon which I have shown you. If, the midpoint of histogram are connected then there is called frequency polygon. Because, the frequency polygon is used to know the trend.

(Refer Slide Time: 29:20)



Ogive

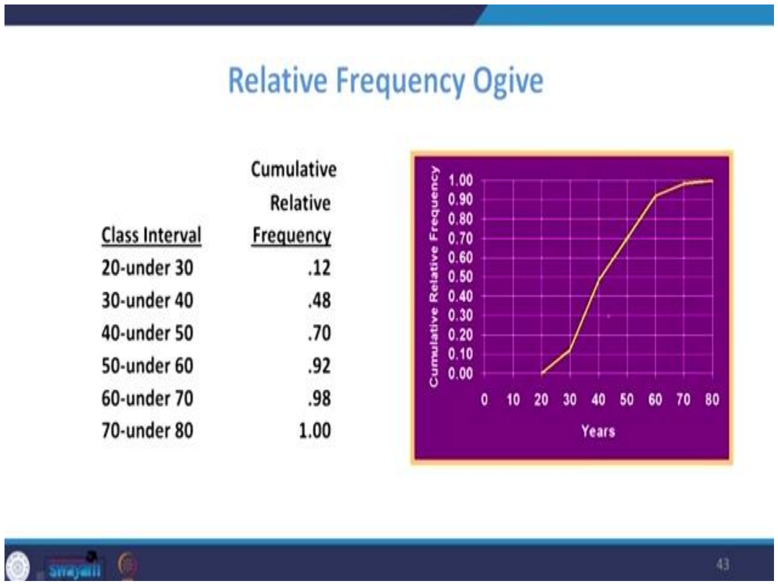
Class Interval	Cumulative Frequency
20-under 30	6
30-under 40	24
40-under 50	35
50-under 60	46
60-under 70	49
70-under 80	50



Trend of the data. The next one is ogive curve. This is cumulative frequency curve .So what is happening in the, for example 20- under 30, the upper limit of the interval is taken the x axis, the cumulative frequency is taken in the y axis. For example, the first interval.20 - 30.So 30 the upper interval is 6. For 40, upper interval is to 24, that is marked.

Because the advantage of this ogive curve is, supposed if we want to know below 16, how many numbers are there, that can be read directly from the ogive curve. That is the purpose of ogive curve.

(Refer Slide Time: 29:56)

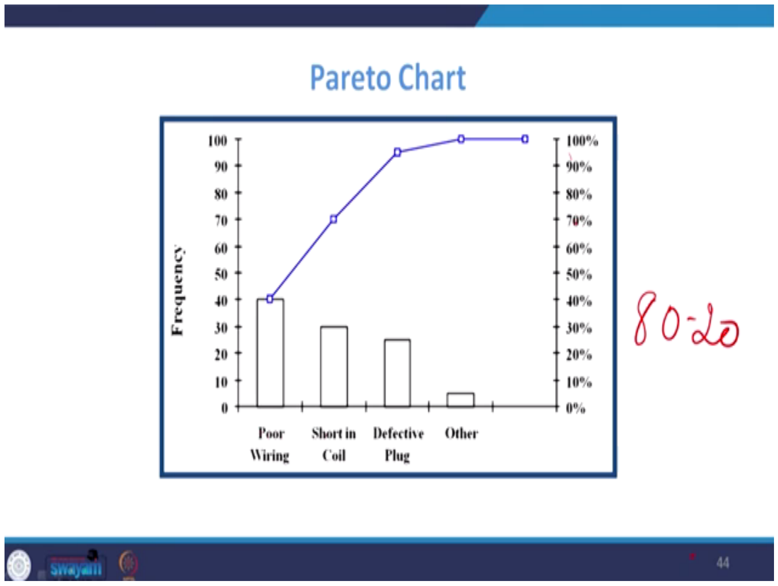


Relative Frequency Ogive

Class Interval	Cumulative Relative Frequency
20-under 30	.12
30-under 40	.48
40-under 50	.70
50-under 60	.92
60-under 70	.98
70-under 80	1.00

Next one is the relative frequency curve. Exactly similar to that now actual frequency that relative frequency was taken.

(Refer Slide Time: 30:08)



Pareto Chart

Okay. The next way to represent the data using pareto chart. The Pareto chart is having some applications in quality control also. This is to identify which is more important, important variable. Assume that, if you look at this Pareto chart. There are 3 axes one is frequency. In x axis, different name is given poor wiring, short in coil, defective plug, other. You see there is one more variable in terms of percentage.

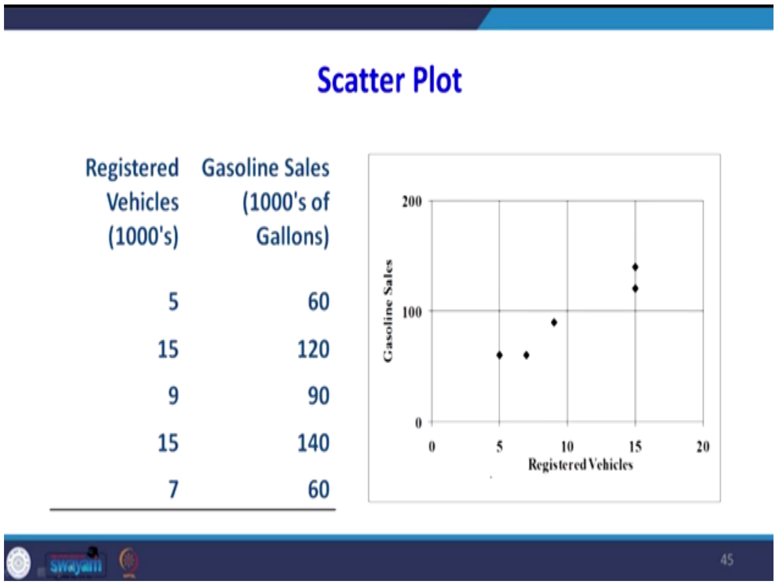
For example, I am a quality control engineer, suppose my motor is failing so often. I want to know there are different reason for failing of the motor. I want to know what are the main reasons, due to which the motor fails. So what I have done. First I have go to frequency table, that is due to poor wiring, the motor was falling for failing 40 times, frequencies 40. Due to short in coil, the motor was failed 30 times.

Due to defective in plug, the motor was failed 25 times. Due to some other reasons the motor was failed by say below 10 times. So the first technique is for drawing this one, we have to arrange in the descending order of their frequency. So in x axis that values are taken. Then the cumulative frequencies plotted on the, this axis. For example, how to interpret this table is. You see, here this value corresponding this only 70.

So 70 % of the failure is due to only two reasons, that is poor wiring and short circuiting. So what is the meaning of this one is, if you are able to address these 2 problems, 70% of the failures can be eliminated. So the purpose of a Pareto chart is, to identify which is critical for us. Generally it is called 80-20 principle. This is called the Pareto principle .That is 80% of the problems are due to 20% of the reasons.

To similarly here, when you look at this, the cell here, need not always 80, see the 70% the failures, only due to 2 factors that is due to poor wiring and short coil. So this is the pareto chart.

(Refer Slide Time: 32:33)

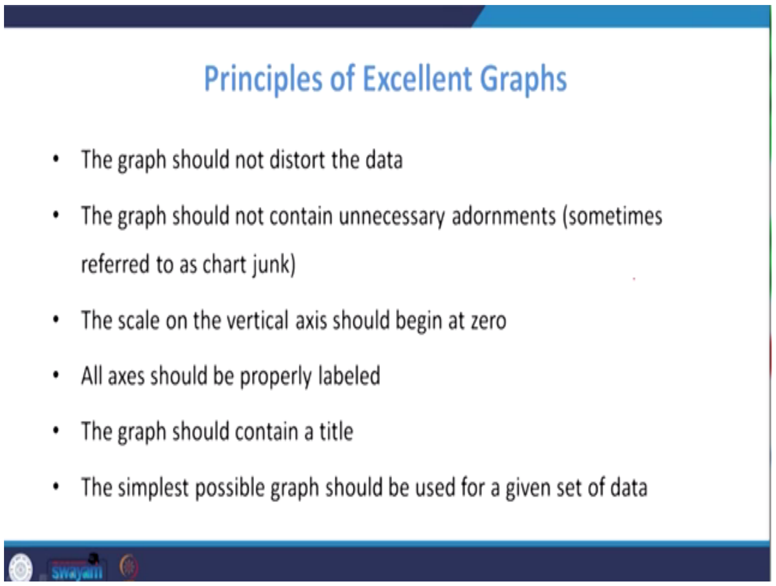


Scatter Plot

Registered Vehicles (1000's)	Gasoline Sales (1000's of Gallons)
5	60
15	120
9	90
15	140
7	60

The next one is scatter plot. The scatter plot is so far what ever seen only for one variable, the scatter plot is used for two variable. In x axis registered vehicle, y axis the gasoline sales. So this says the scatter plot says, when the number of registered vehicle is increasing the gasoline sales is also increasing. So the scatter plot is used to know the trend out the data.

(Refer Slide Time: 32:59)

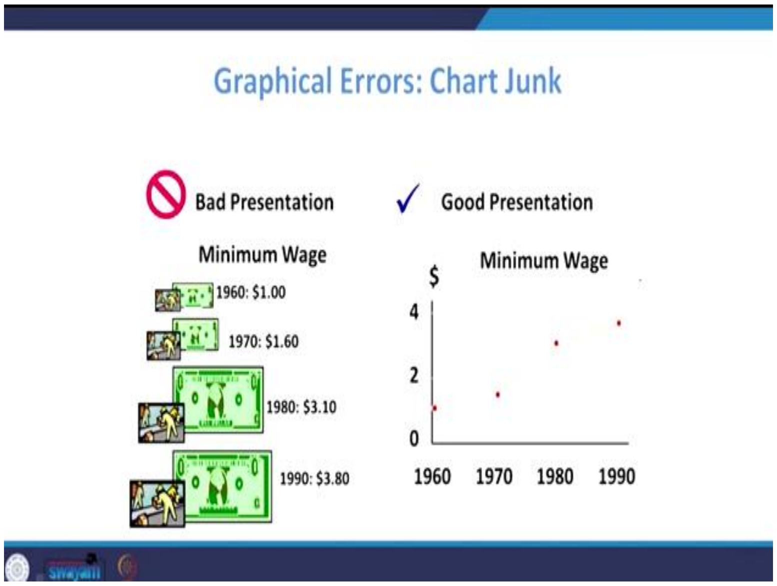


Principles of Excellent Graphs

- The graph should not distort the data
- The graph should not contain unnecessary adornments (sometimes referred to as chart junk)
- The scale on the vertical axis should begin at zero
- All axes should be properly labeled
- The graph should contain a title
- The simplest possible graph should be used for a given set of data

Some of the basic principle for excellent graph. One is the graph should not distort the data. The graph should be very simple. It should not contain unnecessary adornments. So, so much decoration in the graph is not required, the scale on the vertical axis should begin at 0. All axes should be properly labeled. Weather should be x axis or y axis, it has to be properly labeled. The graph should contain a title. It the simplest possible graph should be used for given set of data. These are the basic principle of excellent graph.

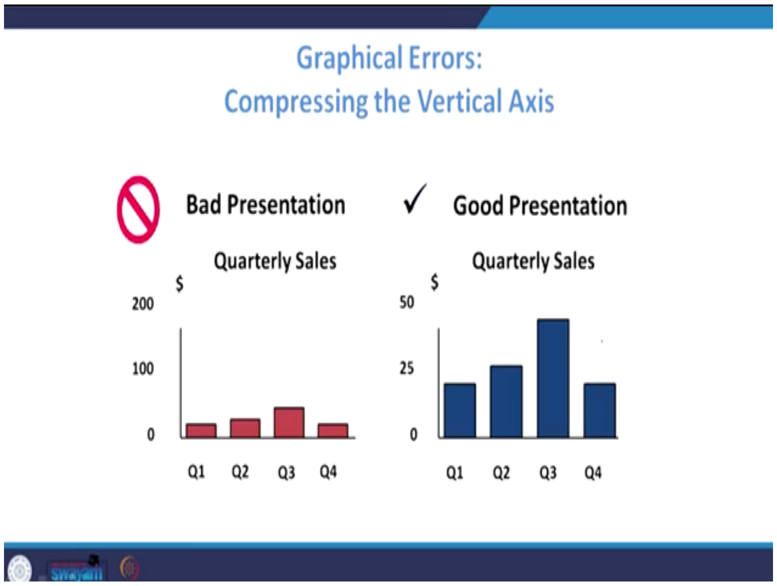
(Refer Slide Time: 33:39)



Graphical Errors: Chart Junk

See when you look at this one. The left hand side it is a bad representation of the graph. What is happening lot of animations, unnecessary pictures. The right hand side, it is a simple graph x axis is taken as were year, in y axis it has taken the wage. So it is showing some trend. But when you look on the left hand side it is not giving any idea. What is happening year with respect to wage.

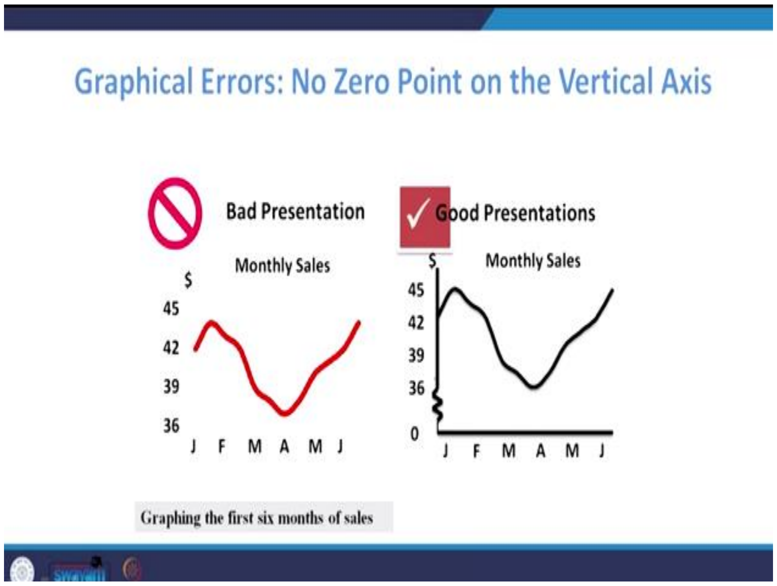
(Refer Slide Time: 34:04)



Graphical Errors: Compressing the Vertical Axis

Another one you look at the left side picture and right side picture. Both are the same data. But what is happening. When here in th e left side picture the scale is 0 to 100, here it is 0 to 25 just by changing the scale, we are able to get different interpretation. You see that when the when the scale is increased. It looked like flat. If you are drawing in smaller scale. You see that look like there's a lot of variations. So what is the learning is that we are to use proper scale to draw the picture.

(Refer Slide Time: 34:40)



The next one is the graphical error, no 0 point on the vertical axis. When you look at the left side of the figure January, February, March, April, May, June, the month is given in x axis. Monthly sales is given y axis. But the problem on the left hand side is it did not start from 0. The right side is you see that the small Brake is given. So, that, even though, 0 to 36 there is no data, you have to make a small break like this. So that, we can come to know it start from 0.

So this is the right hand side is the right way of drawing the graph. This is the basic requirement. In this lecture, what you have seen, how to access particular rows and columns by using basic commands. Then we have seen the different visualization techniques, different theories of the visualization technique. The next class will take in some sample data. By using the sample data with the help of the sample data will try to visualize the data.

By having different tools like a pie chart, bar chart, pictogram, Pareto chart, simple graph. Thank you, we will see you next class.