

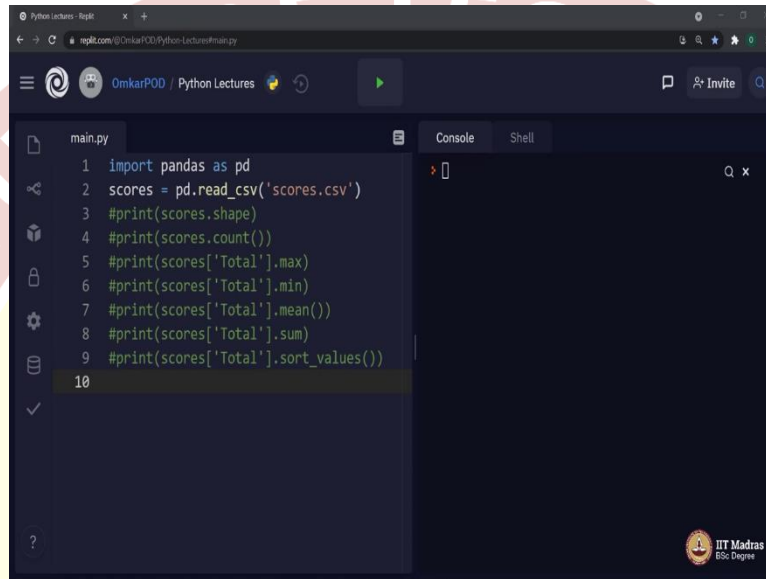


# IIT Madras

ONLINE DEGREE

**Programming in Python**  
**Professor Sudarshan Iyengar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Ropar**  
**Mr. Omkar Joshi**  
**Course Instructor**  
**Indian Institute of Technology, Madras**  
**Online Degree Programme**  
**Pandas Series, DataFrame and more**

(Refer Slide Time: 00:16)

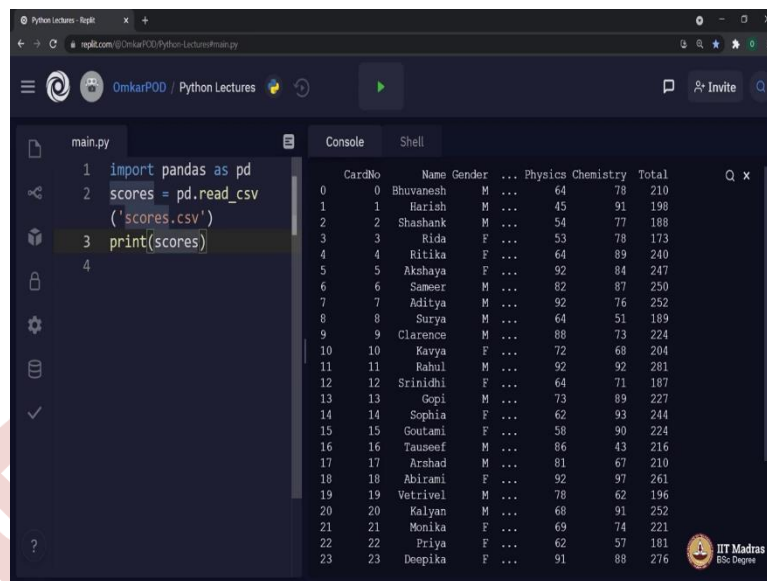


```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 #print(scores.shape)
4 #print(scores.count())
5 #print(scores['Total'].max)
6 #print(scores['Total'].min)
7 #print(scores['Total'].mean())
8 #print(scores['Total'].sum)
9 #print(scores['Total'].sort_values())
10
```

Hello, python students. In last lecture, we started with Pandas. We saw how to input this external library called Pandas. Then we saw how to read a CSV file, and then, we continued our discussion with some Pandas' features like shape, count, maximum, minimum, average sum and even sorting of values of a specific column from the given table.

But during that lecture, we never talked about any specific details related to pandas. In this lecture, we will see some key features of pandas like series, data frame, and then we will continue with some more examples, which are little bit more complicated than what we saw earlier.

(Refer Slide Time: 01:17)



The screenshot shows a Jupyter Notebook interface with a file named 'main.py'. The code in the notebook is as follows:

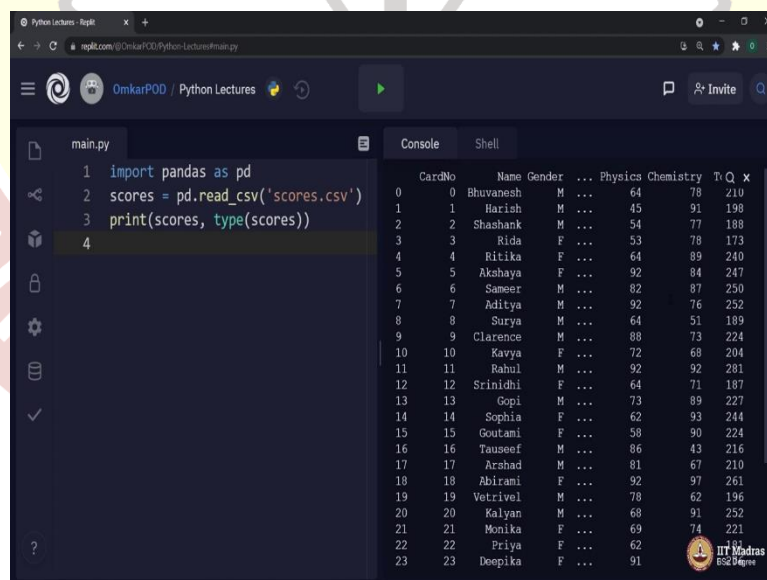
```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores)
```

The console output displays a DataFrame with the following columns: CardNo, Name, Gender, Physics, Chemistry, and Total. The data is as follows:

CardNo	Name	Gender	Physics	Chemistry	Total
0	Bhuvanesh	M	64	78	210
1	Harish	M	45	91	198
2	Shashank	M	54	77	188
3	Rida	F	53	78	173
4	Ritika	F	64	89	240
5	Akshaya	F	92	84	247
6	Sameer	M	82	87	250
7	Aditya	M	92	76	252
8	Surya	M	64	51	189
9	Clarence	M	88	73	224
10	Kavya	F	72	68	204
11	Rahul	M	92	92	281
12	Srinidhi	F	64	71	187
13	Gopi	M	73	89	227
14	Sophia	F	62	93	244
15	Goutami	F	58	90	224
16	Tauseef	M	86	43	216
17	Arshad	M	81	67	210
18	Abirami	F	92	97	261
19	Vetrivel	M	78	62	196
20	Kalyan	M	68	91	252
21	Monika	F	69	74	221
22	Priya	F	62	57	181
23	Deepika	F	91	88	276

Let us start with data frame, print scores. We all know what this particular variable holds with respect to pandas. It stores the entire CSV file in the form of a table. And with respect to Pandas, this particular table is referred as data frame, which means data frame is nothing but a two-dimensional data structure, which is used to store tabular data. We can check the same thing by printing type of scores.

(Refer Slide Time: 01:54)



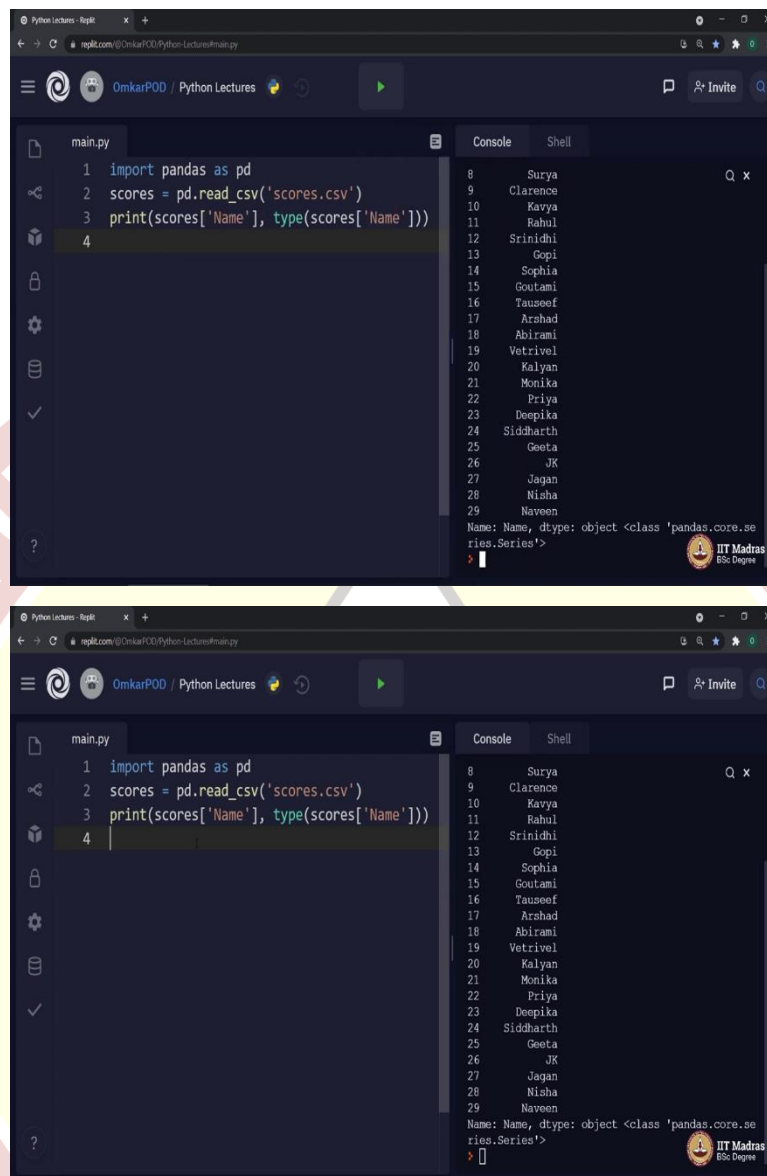
The screenshot shows the same Jupyter Notebook interface, but the code has been updated to print the type of the 'scores' variable:

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores, type(scores))
```

The console output now shows the DataFrame followed by its type: `DataFrame`.

As I explained, it says data frame is, this entire table is referred as data frame, then the question is, what should we call to a specific column in this table. Any specific column from the data frame is referred as series in Pandas. Let us verify that.

(Refer Slide Time: 02:20)



```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores['Name'], type(scores['Name']))
4
```

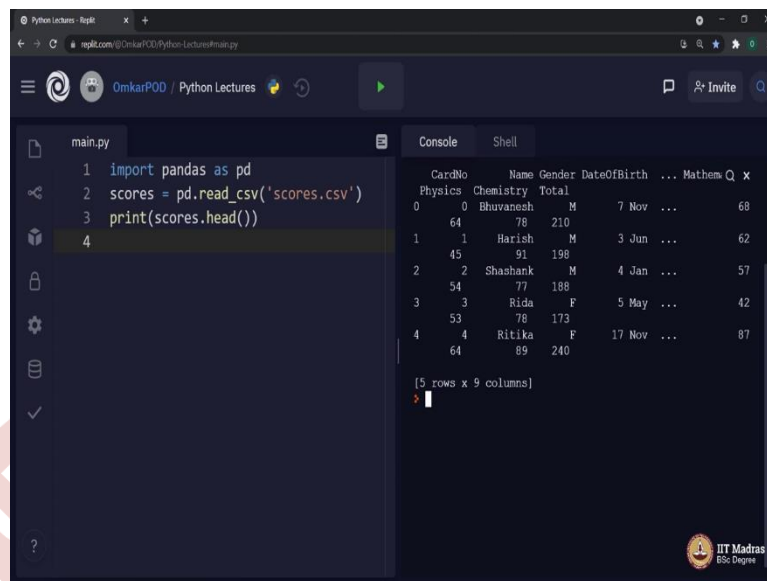
```
8 Surya
9 Clarence
10 Kavya
11 Rahul
12 Srinidhi
13 Gopi
14 Sophia
15 Goutami
16 Tauseef
17 Arshad
18 Abirami
19 Vettrivel
20 Kalyan
21 Monika
22 Priya
23 Deepika
24 Siddharth
25 Geeta
26 JK
27 Jagann
28 Nisha
29 Naveen

Name: Name, dtype: object <class 'pandas.core.series.Series'>
```

Scores of name, type of scores of name, it will print a specific column from the data frame, which is name, and the type is series. This particular variable, which stores the output of read underscore CSV is referred as data frame and it is a two-dimensional entity just like a table. Whereas, any specific column from that table or from the data frame is referred as series, which means series is an one-dimensional entity.

Now, as we know what is series and data frame in Pandas, now, let us try to execute few more examples, which we found very difficult when we were studying these in computational thinking course.

(Refer Slide Time: 03:18)



The screenshot shows a Jupyter Notebook interface with a file explorer on the left, a code editor in the center, and a console on the right. The code in the editor is as follows:

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores.head())
4
```

The console output displays a table with 9 columns: CardNo, Name, Gender, DateOfBirth, ..., Mathem, Q. The first five rows of the data are shown:

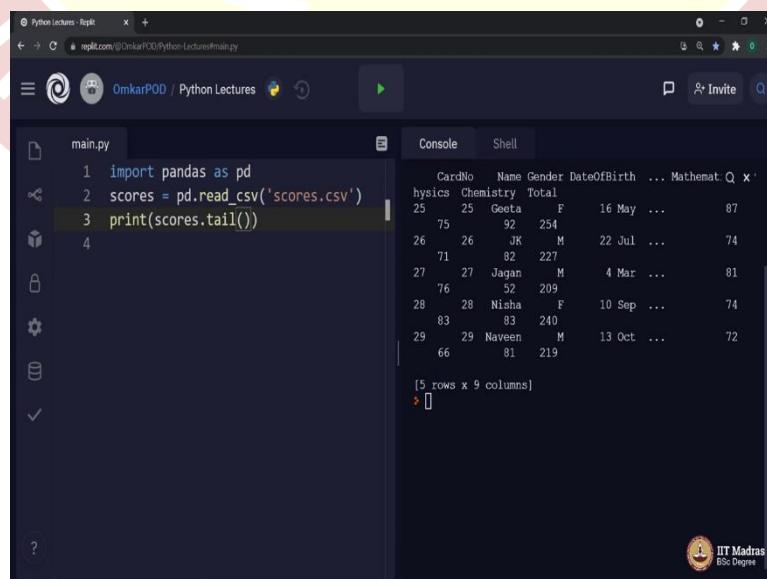
	CardNo	Name	Gender	DateOfBirth	...	Mathem	Q
0	64	Bhuvanech	M	7 Nov	...	68	
1	45	Harish	M	3 Jun	...	62	
2	54	Shashank	M	4 Jan	...	57	
3	53	Rida	F	5 May	...	42	
4	64	Ritika	F	17 Nov	...	87	

Below the table, the console shows the output: [5 rows x 9 columns].

Consider this particular line print scores. Whenever we try to print the entire data frame, it displays all the rows from the given data frame. In the case of scores data set, we have only 30 entries. Therefore, it was easy to print the entire data frame as output. But what if the input data set is much more larger than the score's dataset, then it is not possible to go through all those rows manually.

And sometimes, we may prefer to look at only few sample rows from the given data set. And the way to access these sample rows is using a function called head, scores dot head. This particular function will print only top five rows from the given data set, as in the data frame. As you can see, it is printing index from 0 to 4.

(Refer Slide Time: 04:35)



The screenshot shows a Jupyter Notebook interface with a file explorer on the left, a code editor in the center, and a console on the right. The code in the editor is as follows:

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores.tail())
4
```

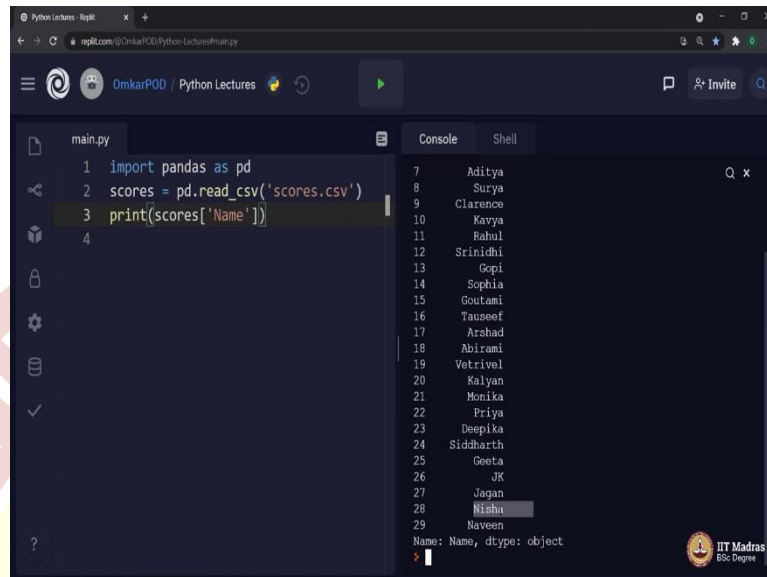
The console output displays a table with 9 columns: CardNo, Name, Gender, DateOfBirth, ..., Mathem, Q. The last five rows of the data are shown:

	CardNo	Name	Gender	DateOfBirth	...	Mathem	Q
25	75	Geeta	F	16 May	...	87	
26	71	JK	M	22 Jul	...	74	
27	76	Jagan	M	4 Mar	...	81	
28	83	Nisha	F	10 Sep	...	74	
29	66	Naveen	M	13 Oct	...	72	

Below the table, the console shows the output: [5 rows x 9 columns].

Similarly, in order to print the last five rows from the data frame, the function is tail. It will print last five rows from the data frame that is index from 25 to 29.

(Refer Slide Time: 04:53)



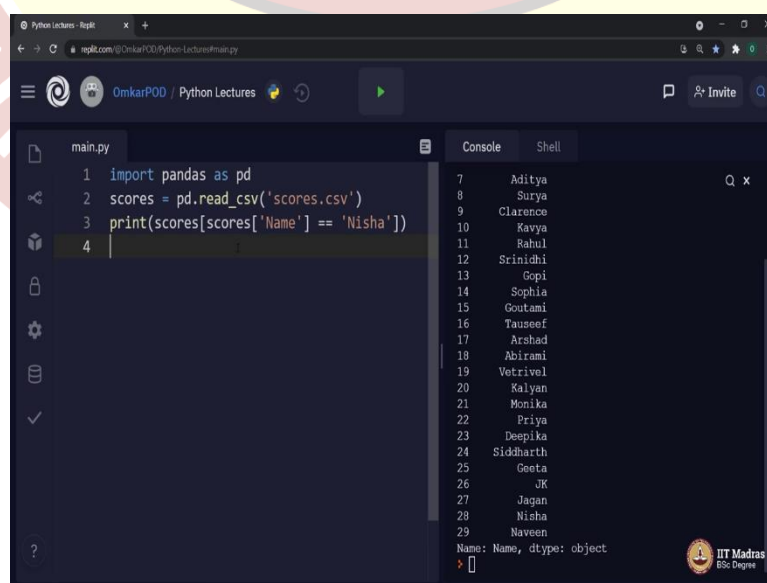
The screenshot shows a Jupyter Notebook interface with a code editor on the left and a console on the right. The code in the editor is:

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores['Name'])
4
```

The console output displays a list of 29 student names, indexed from 7 to 29. The names are: Aditya, Surya, Clarence, Kavya, Rahul, Srinidhi, Gopi, Sophia, Goutami, Tauseef, Arshad, Abirami, Vetrivel, Kalyan, Monika, Priya, Deepika, Siddharth, Geeta, JK, Jagan, Nisha, and Naveen. The name 'Nisha' is highlighted in the list. Below the list, the console shows the output of the print statement: `Name: Name, dtype: object`.

Earlier we saw something like scores of name, and we all know this will print a series from the data frame. This is fine, but what if I want to print all the details of a specific student based on his or her name, which means, what if I want to print the entire details of student Nisha? So, the question is, can we do that using Pandas, because earlier we saw Pandas help us access data column wise. But in this case, I want the entire row which belongs to this student, Nisha.

(Refer Slide Time: 06:13)



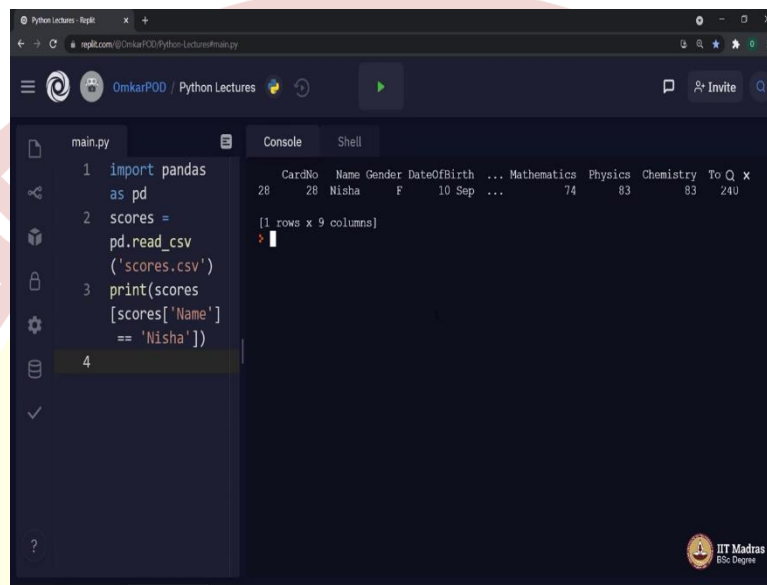
The screenshot shows the same Jupyter Notebook interface as before, but with the code in the editor updated to filter the DataFrame by the name 'Nisha':

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Name'] == 'Nisha'])
4
```

The console output now shows only the row for the student 'Nisha', indexed 28. The output is: `Name: Name, dtype: object`.

Let us say can we do something like that. And the way to do this is scores of name equal to Nisha, and this condition should be executed on the given data frame. Hence, we should have something like this. It says, access the data frame name scores, but make sure, within that data frames name series, the value should be Nisha. Let us execute and see, whether we are getting the expected output or not.

(Refer Slide Time: 06:13)



The screenshot shows a Jupyter Notebook interface with a code editor on the left and a console on the right. The code in the editor reads a CSV file named 'scores.csv' and prints the resulting DataFrame. The console output shows a single row of data for a student named Nisha.

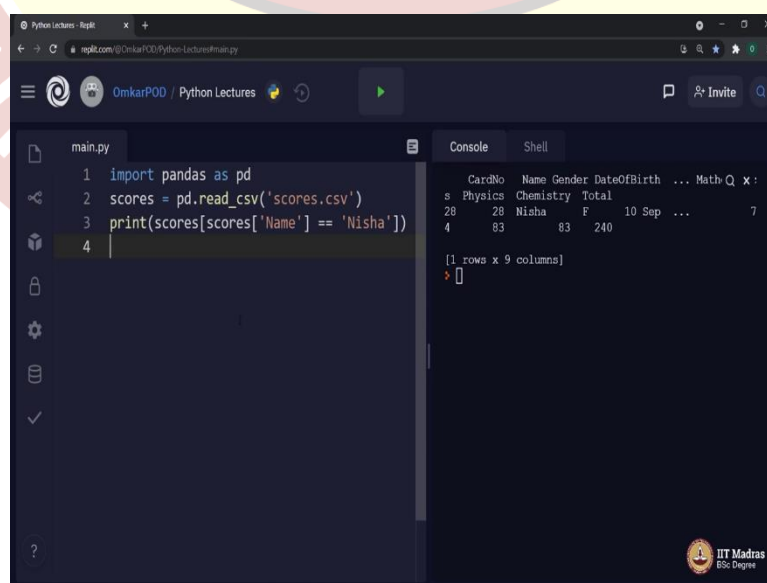
```
1 import pandas
  as pd
2 scores =
  pd.read_csv
  ('scores.csv')
3 print(scores
  [scores['Name']
  == 'Nisha'])
4
```

CardNo	Name	Gender	DateOfBirth	...	Mathematics	Physics	Chemistry	Total
28	Nisha	F	10 Sep	...	74	83	83	240

[1 rows x 9 columns]

As you can see, we are getting all the details of this student Nisha. Card number, name, gender, date of birth, and so on till the total. Pandas does not restrict us from reading data column wise, it is capable of reading data, either by column or even by a row.

(Refer Slide Time: 06:37)



The screenshot shows the same Jupyter Notebook interface, but the code in the editor has been updated to filter the DataFrame for the student named Nisha. The console output now shows an empty list, indicating that the filter condition was not met.

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Name'] == 'Nisha'])
4
```

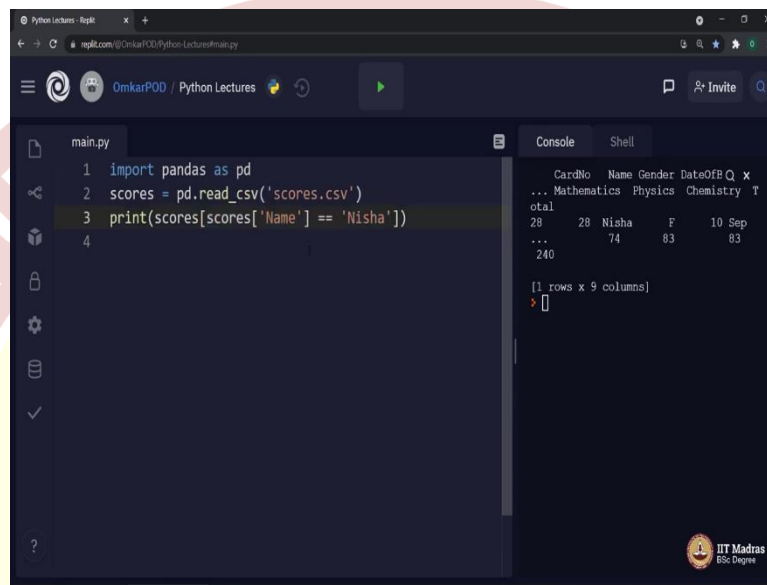
CardNo	Name	Gender	DateOfBirth	...	Math	Q	x
28	Physics	Chemistry	Total	...	7		
28	83	Nisha	F	10 Sep	...		
4	83	83	240				

[1 rows x 9 columns]



Alright, this is related to one particular student. What if I want to find marks of a topper boy, and marks of a topper girl? That means, I want to find topper marks based on their gender. Now, that seems little bit complicated as compared to whatever examples we saw so far. It may appear very complicated, and you might think this may require lot of execution and complicated python code, but actually, it is once again very easy with respect to Pandas.

(Refer Slide Time: 07:21)



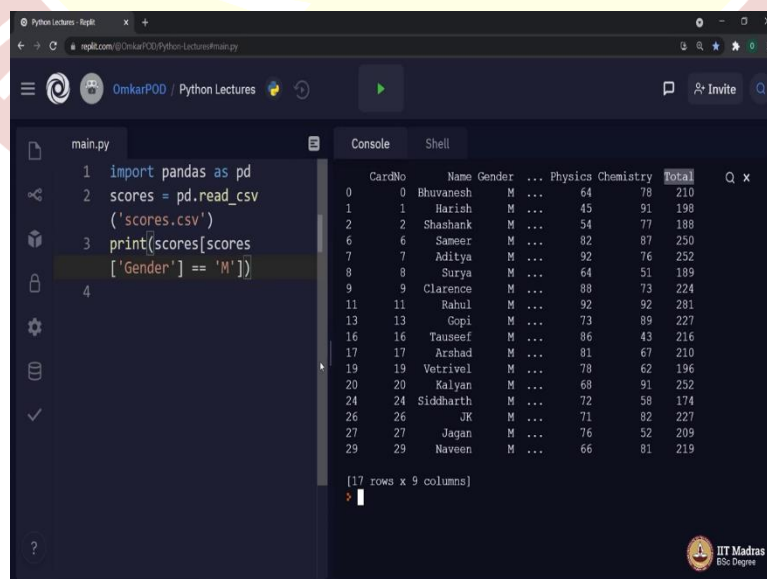
```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Name'] == 'Nisha'])
4
```

CardNo	Name	Gender	DateOfB	Q	X
28	Nisha	F	10 Sep		

[1 rows x 9 columns]

Let us see how that is done. We already know this is how we can write a condition on the data frame. Now the condition is based on gender. Let us say first, I want to find a topper from boys. This will give us all the data where gender is M. Let us see what happens if we try to print something like this.

(Refer Slide Time: 07:50)



```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Gender'] == 'M'])
4
```

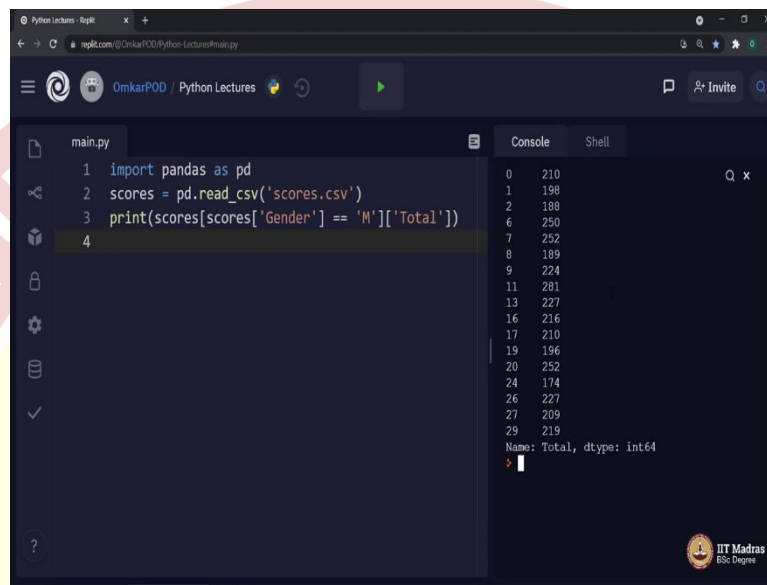
CardNo	Name	Gender	Physics	Chemistry	Total
0	Bhuvanesh	M	64	78	210
1	Harish	M	45	91	198
2	Shashank	M	54	77	188
6	Sameer	M	82	87	250
7	Aditya	M	92	76	252
8	Surya	M	64	51	189
9	Clarence	M	88	73	224
11	Rahul	M	92	92	281
13	Gopi	M	73	89	227
16	Tauseef	M	86	43	216
17	Arshad	M	81	67	210
19	Vetrivel	M	78	62	196
20	Kalyan	M	68	91	252
24	Siddharth	M	72	58	174
26	JK	M	71	82	227
27	Jagan	M	76	52	209
29	Naveen	M	66	81	219

[17 rows x 9 columns]



It prints the entire data where gender is M. Alright, with this, we were able to apply first level of filter. Next, we want only column, which is total. How to do that? Same just like this. So, this particular highlighted part gave us the entire details. And now we are telling computer from all these details, which is nothing but a data frame give us only one specific column, which is a Panda's series total. Let us try that.

(Refer Slide Time: 08:36)



The screenshot shows a Jupyter Notebook interface with a code editor on the left and a console on the right. The code in the editor is as follows:

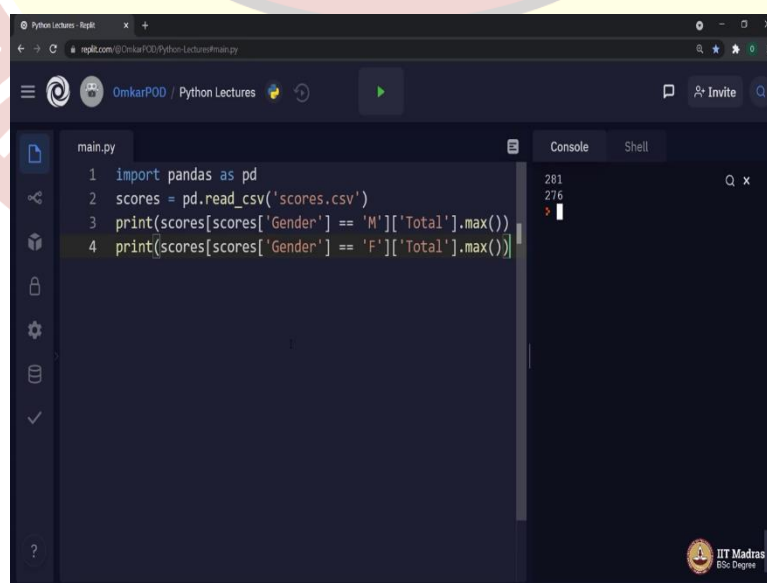
```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Gender'] == 'M']['Total'])
4
```

The console output displays a list of 20 total scores for males, ranging from 210 to 219. Below the list, it shows the data type: `Name: Total, dtype: int64`.

Index	Total
0	210
1	198
2	188
6	250
7	252
8	189
9	224
11	281
13	227
16	216
17	210
19	196
20	252
24	174
26	227
27	209
29	219

Alright, we achieved the next step. Now, as per our original statement, we want a topper from this particular list, and how to get maximum out of something. Correct, we have seen that earlier dot max. Let us execute.

(Refer Slide Time: 09:03)



The screenshot shows a Jupyter Notebook interface with a code editor on the left and a console on the right. The code in the editor is as follows:

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Gender'] == 'M']['Total'].max())
4 print(scores[scores['Gender'] == 'F']['Total'].max())
```

The console output displays the maximum total score for males (281) and females (276).

```
281
276
```

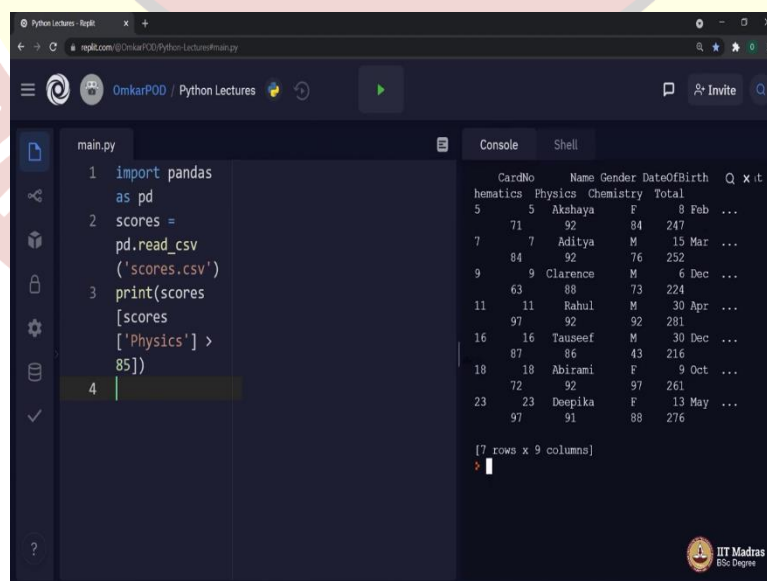
And answer is, 281. Similar line can be executed by modifying a gender parameter over here. Now, we got 281 as a topper for boys and 276 as a topper for girls. Once again, we were able to do this in just four lines of python code. Do you remember how complicated it was earlier when we did this in computational thinking or even when we were reading this data from a dictionary or from a file using file handling operations. Alright, so far, this is going good but now can we try something even more complicated than this?

For example, something like dividing all the students based on their marks into four different categories. If you remember, we did something similar in order to award grades to individual students. Let us say based on physics marks, we want to categorize students into four categories.

Students with physics marks higher than 85 will be the first category. second category, marks between 70 to 85, third category, marks between 60 to 70, and the fourth category, marks less than 60. We will give Grade A to first category students, then Grade B to second category students and so on.

We have done similar exercise earlier, but now, we will not stop there, we will go a step beyond. We will find out how many students are there in each category, which means, how many students got A grade, how many students got B grade, and so on for all four grades. Let us try to write that particular code.

(Refer Slide Time: 11:19)



```
1 import pandas
2 as pd
3 scores =
4 pd.read_csv
5 ('scores.csv')
6 print(scores
7 [scores
8 ['Physics'] >
9 85])
```

CardNo	Name	Gender	DateOfBirth	Q	x	t
hematics	Physics	Chemistry	Total			
5	5	Akshaya	F	8	Feb	...
71	92	84	247			
7	7	Aditya	M	15	Mar	...
84	92	76	252			
9	9	Clarence	M	6	Dec	...
63	88	73	224			
11	11	Rahul	M	30	Apr	...
97	92	92	281			
16	16	Tauseef	M	30	Dec	...
87	86	43	216			
18	18	Abirami	F	9	Oct	...
72	92	97	261			
23	23	Deepika	F	13	May	...
97	91	88	276			

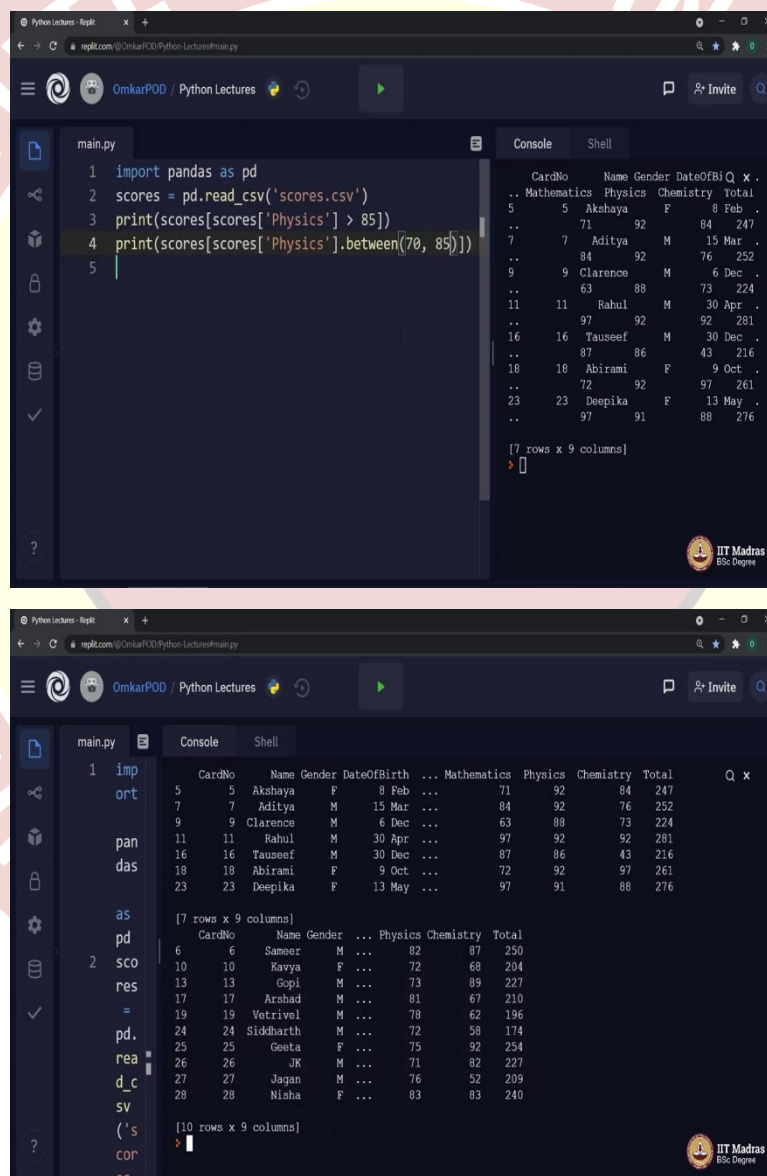
[7 rows x 9 columns]

Now, look at this. This is very simple. On top of scores data frame, we are saying scores of physics greater than 85. That means, now all the students who scored more than 85 marks in

physics should be our output. Let us try it. As you can see, 92, 92, 88, 92, 86, 92, 91 all above 85. All these students fall under A grade. Let us move to second category.

As we said, the second category should be the students between 70 to 85. We can always write something like greater than 70, and then the entire condition with less than 85 or something similar. But with respect to Pandas, we do not even have to do all those things. Panda provides us a method called as between. This particular method will give us the required data between 70 and 85. Let us execute this.

(Refer Slide Time: 12:50)



```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Physics'] > 85])
4 print(scores[scores['Physics'].between(70, 85)])
5
```

CardNo	Name	Gender	DateOfBirth	Mathematics	Physics	Chemistry	Total
5	Akshaya	F	8 Feb	71	92	84	247
7	Aditya	M	15 Mar	84	92	76	252
9	Clarence	M	6 Dec	63	88	73	224
11	Rahul	M	30 Apr	97	92	92	281
16	Tauseef	M	30 Dec	87	86	43	216
18	Abirami	F	9 Oct	72	92	97	261
23	Deepika	F	13 May	97	91	88	276

```
[7 rows x 9 columns]
```

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Physics'] > 85])
4 print(scores[scores['Physics'].between(70, 85)])
5
```

CardNo	Name	Gender	DateOfBirth	Mathematics	Physics	Chemistry	Total
6	Sameer	M	...	82	87	87	256
10	Kavya	F	...	72	68	60	200
13	Gopi	M	...	73	89	60	222
17	Arahad	M	...	81	67	72	220
19	Vetriweli	M	...	78	62	62	202
24	Siddharth	M	...	72	58	60	190
25	Gesta	F	...	75	92	254	254
26	JK	M	...	71	82	227	227
27	Jagan	M	...	76	52	209	209
28	Nisha	F	...	83	83	240	240

```
[10 rows x 9 columns]
```

As you can see, first we got all above 85. And now students who are in between 70 and 85 with respect to physics marks. Now, let us write python code for remaining two categories as

well. Third category, physics marks between 60 to 70 and fourth category physics marks less than 60. Let us execute.

(Refer Slide Time: 13:22)

```

1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Physics'] > 85])
4 print(scores[scores['Physics'] between(70, 85)])
5 print(scores[scores['Physics'] between(60, 70)])
6 print(scores[scores['Physics'] < 60])
7

```

The console output shows a table of student scores with columns: CardNo, Name, Gender, Total, and Physics. The data is filtered to show only students with Physics marks greater than 85.

CardNo	Name	Gender	Total	Physics
6	Sameer	M	250	87
10	Kavya	F	204	72
13	Gopi	M	227	89
17	Arshad	M	210	81
19	Vetrivel	M	196	78
24	Siddharth	M	174	72
25	Geeta	F	254	92
26	JK	M	227	82
27	Jagan	M	209	76
28	Nisha	F	240	83

The console output shows a table of student scores with columns: CardNo, Name, Gender, DateOfBirth, Mathematics, Physics, Chemistry, and Total. The data is filtered to show only students with Physics marks greater than 85.

CardNo	Name	Gender	DateOfBirth	Mathematics	Physics	Chemistry	Total
5	Akshaya	F	8 Feb	71	92	84	247
7	Aditya	M	15 Mar	84	92	76	252
9	Clarence	M	6 Dec	63	88	73	224
11	Rahul	M	30 Apr	97	92	92	281
16	Tauseef	M	30 Dec	87	86	43	216
18	Abirami	F	9 Oct	72	92	97	261
23	Deepika	F	13 May	97	91	88	276

The console also shows a table of student scores with columns: CardNo, Name, Gender, Physics, Chemistry, and Total. The data is filtered to show only students with Physics marks between 70 and 85.

CardNo	Name	Gender	Physics	Chemistry	Total
6	Sameer	M	82	87	250
10	Kavya	F	72	68	204
13	Gopi	M	73	89	227
17	Arshad	M	81	67	210
19	Vetrivel	M	78	62	196
24	Siddharth	M	72	58	174
25	Geeta	F	75	92	254
26	JK	M	71	82	227
27	Jagan	M	76	52	209
28	Nisha	F	83	83	240

The console also shows a table of student scores with columns: CardNo, Name, Gender, Physics, Chemistry, and Total. The data is filtered to show only students with Physics marks less than 60.

CardNo	Name	Gender	Physics	Chemistry	Total
0	Rhuvanesh	M	64	78	210

```
Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures/main.py

OmkarPOD / Python Lectures

ma Console Shell

1 CardNo Name Gender DateOfBirth ... Mathematics Physics Chemistry Total
5 5 Akshaya F 8 Feb ... 71 92 84 247
7 7 Aditya M 15 Mar ... 84 92 76 252
9 9 Clarence M 6 Dec ... 63 88 73 224
11 11 Rahul M 30 Apr ... 97 92 92 281
16 16 Tauseef M 30 Dec ... 87 86 43 216
18 18 Abirani F 9 Oct ... 72 92 97 261
23 23 Deepika F 13 May ... 97 91 88 276

[7 rows x 9 columns]
CardNo Name Gender ... Physics Chemistry Total
6 6 Sameer M ... 82 87 250
10 10 Kavya F ... 72 68 204
13 13 Gopi M ... 73 89 227
17 17 Arshad M ... 81 67 210
19 19 Vetrivel M ... 78 62 196
24 24 Siddharth M ... 72 58 174
25 25 Geeta F ... 75 92 254
26 26 JK M ... 71 82 227
27 27 Jagan M ... 76 52 209
28 28 Nisha F ... 83 83 240

[10 rows x 9 columns]
CardNo Name Gender ... Physics Chemistry Total
0 0 Bhuvanesh M ... 64 78 210
```

```
Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures/main.py

OmkarPOD / Python Lectures

ma Console Shell

1 26 26 JK M ... 71 82 227
27 27 Jagan M ... 76 52 209
28 28 Nisha F ... 83 83 240

[10 rows x 9 columns]
CardNo Name Gender ... Physics Chemistry Total
0 0 Bhuvanesh M ... 64 78 210
4 4 Ritika F ... 64 89 240
8 8 Surya M ... 64 51 189
12 12 Srinidhi F ... 64 71 187
14 14 Sophia F ... 62 93 244
20 20 Kalyan M ... 68 91 252
21 21 Monika F ... 69 74 221
22 22 Priya F ... 62 57 181
29 29 Naveen M ... 66 81 219

[9 rows x 9 columns]
CardNo Name Gender DateOfBirth ... Mathematics Physics Chemistry Total
1 1 Harish M 3 Jun ... 62 45 91 198
2 2 Shashank M 4 Jan ... 57 54 77 188
3 3 Rida F 5 May ... 42 53 78 173
15 15 Goutami F 22 Sep ... 76 58 90 224

[4 rows x 9 columns]
```

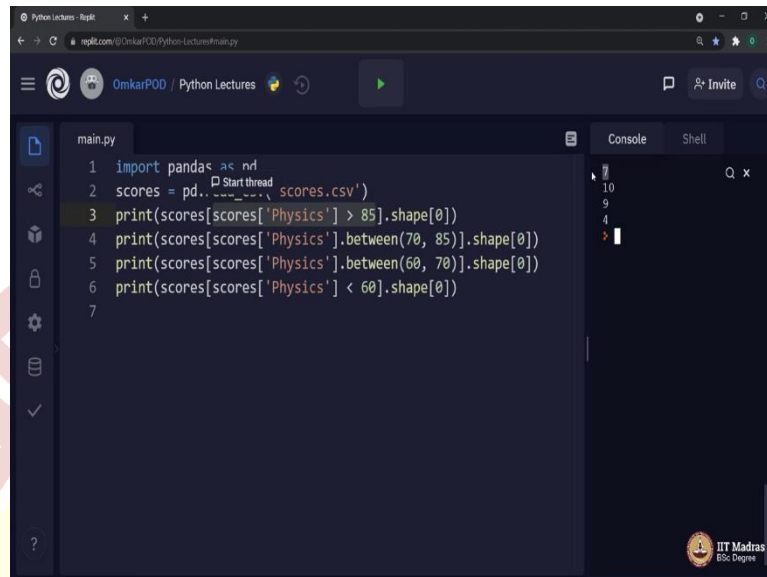
As you can see, first, above 85, 7 students, then 70 to 85, 10 students, then 60 to 70, 9 students, and below 60 we have 4 students. If you add these numbers, we should get 30. This is what we started with. But then we said we will not stop here, we will get the exact counts as in the exact number of students in all these categories, which means the final output should print only these four numbers 7, 10, 9, and 4.

Now, the question is how to do that? How to extract these specific numbers from this entire data, which is being displayed over here. And we already know the answer. We have studied something called as shape in our previous lecture.

If you remember, shape displays a tuple, with two values, number of rows comma number of columns. In this case, we want number of rows, which is at the zeroth index in that tuple.

This should work so, this dot shape of 0 should give us that number. First observe the code, then we will execute it.

(Refer Slide Time: 15:09)



The screenshot shows a Replit Python environment with a file named `main.py`. The code in the file is as follows:

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[scores['Physics'] > 85].shape[0])
4 print(scores[scores['Physics'].between(70, 85)].shape[0])
5 print(scores[scores['Physics'].between(60, 70)].shape[0])
6 print(scores[scores['Physics'] < 60].shape[0])
7
```

The console output on the right shows the results of the execution:

```
7
10
9
4
```

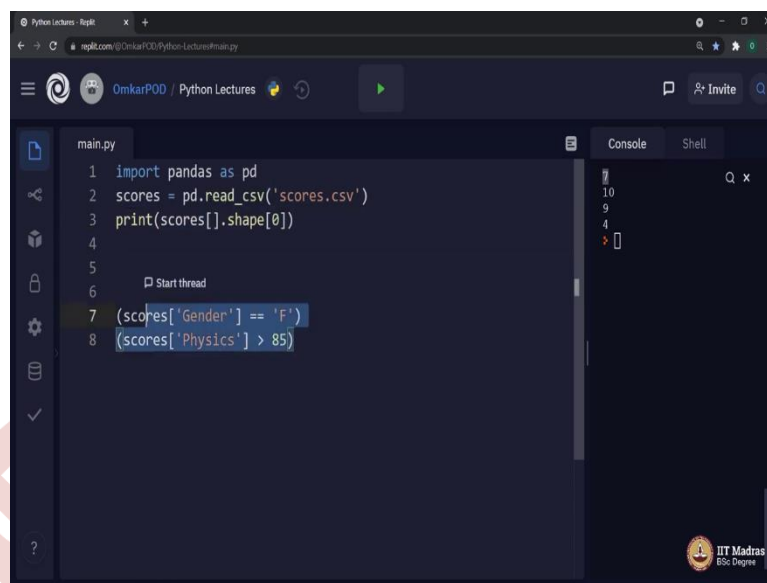
The background of the slide features a large, faint watermark of the IIT Madras logo and the motto "सिद्धिर्भवति कर्मजा" (Siddhirbhavati Karmaja) in Devanagari script.

Says 7, 10, 9, 4, as expected. Once again, very simple. Now, let us increase the complexity by one more step. So far, we are checking only one condition like this. And when we require two conditions, we use this particular way between function. This was possible because both conditions were based on the same series, which is physics marks. What if these conditions are based on two different columns as a two different Panda's series, then what should we do?

Let us consider this example. Scores of physics greater than 85 is fine, but what if I want to know a gender wise split even in this, which means the students above 85 are 7, that is known to us. But now, I want to know how many are male students and how many are female students, which means, now we have to add one more condition, which is based on a different column, which is gender. So how to write these two conditions using two different Panda's series.



(Refer Slide Time: 16:47)



The screenshot shows a Jupyter Notebook interface with a dark theme. The main area displays a Python script in a file named 'main.py'. The script consists of the following lines:

```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[0].shape[0])
4
5
6
7 (scores['Gender'] == 'F')
8 (scores['Physics'] > 85)
```

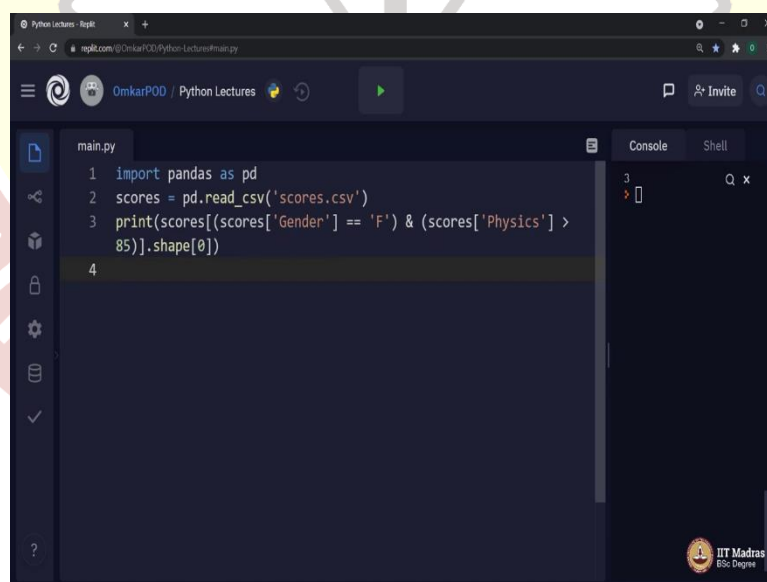
The output of the script is visible in the 'Console' tab on the right, showing the following values:

```
10
9
4
> []
```

The interface includes a sidebar on the left with icons for file management, a top bar with the 'OmkarPOD / Python Lectures' title, and a bottom right corner with the 'IIT Madras BSc Degree' logo.

Let us try that. First, let us remove this. Alright, first, let us extract this condition from here. This was our condition. Now, the second condition which we are trying to write, which says scores of gender is equal equal to, let us say female. Now, somehow, we have to add both these conditions inside this square bracket, then only this particular data frame will be filtered based on both these conditions. Let us see how to do that.

(Refer Slide Time: 17:32)



The screenshot shows a Jupyter Notebook interface with a dark theme. The main area displays a Python script in a file named 'main.py'. The script consists of the following lines:

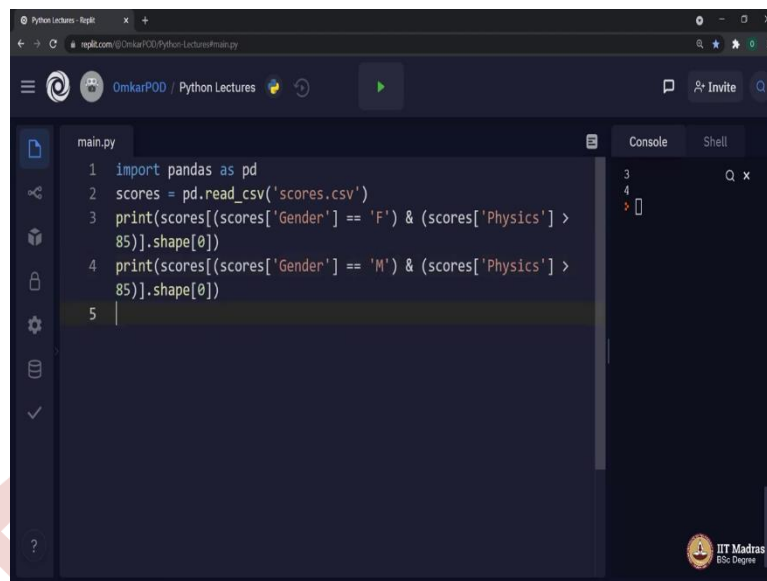
```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[(scores['Gender'] == 'F') & (scores['Physics'] > 85)].shape[0])
4
```

The output of the script is visible in the 'Console' tab on the right, showing the following value:

```
3
> []
```

The interface includes a sidebar on the left with icons for file management, a top bar with the 'OmkarPOD / Python Lectures' title, and a bottom right corner with the 'IIT Madras BSc Degree' logo.





```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores[(scores['Gender'] == 'F') & (scores['Physics'] > 85)].shape[0])
4 print(scores[(scores['Gender'] == 'M') & (scores['Physics'] > 85)].shape[0])
5
```

Console

```
3
4
[]
```

First, let us move this one from there to here and then the second one, as well. Now, we have first condition, followed by a second condition, and we want something like AND operator in between. Before you jump to a conclusion, let me tell you Pandas do not support our regular python AND operator. Instead, it has its own AND operator, and the symbol used is this ampersand sign, like this.

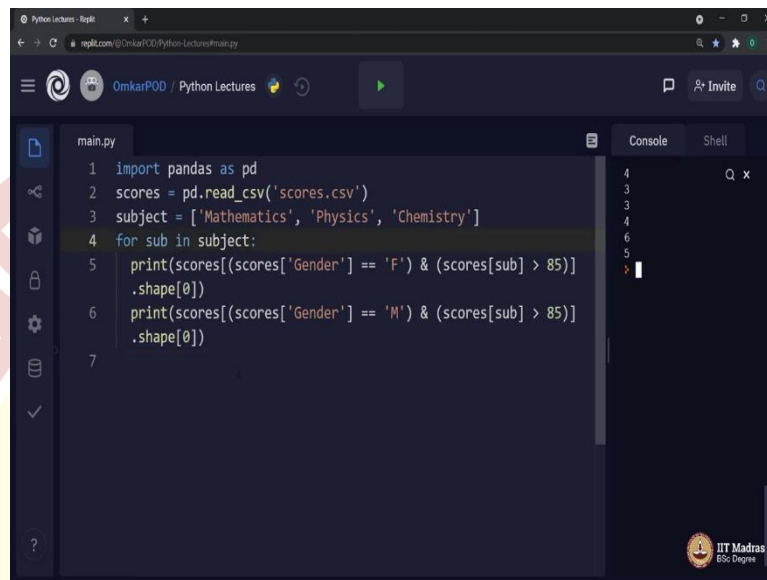
Look at this particular code. First condition scores of gender is equal equal to female and scores of physics greater than 85. When both these conditions are satisfied, on top of the given data frame, then we will get some number of rows, and we are counting the shape of the data frame and extracting the zeroth index from the tuple.

I understand this might be a bit complicated in the first glance. So maybe just pause this video, try to analyze this line number three, and then you can continue. Let us execute this code, the answer is 3. If we can print the same line by modified parameter M, we should get exactly 3 and 4.

If you remember, the original number was 7, which is now split into two, 3 and 4. Are we going to stop here? Of course, not. Let us add one more level of complexity into this code. Why should we do all these things only for physics marks? I want counts of all the students above 85 in all three subjects and based on their genders. That means, we have to execute this particular line of Physics, Chemistry and mathematics, with F. And then once again, with physics, chemistry and mathematics with M. Which means, now we want total, 3 into 2, 6 print statements. But we are programmers, we do not repeat same line six times, what we do, we write a loop.

So, let us declare one list subject is equal to mathematics, physics, and chemistry. And these two lines should be executed for every entry in this list. Hence, for every subject in subject list we should execute these two lines. And now, instead of this subject parameter, we should use this variable sub. Let us try this code.

(Refer Slide Time: 21:21)



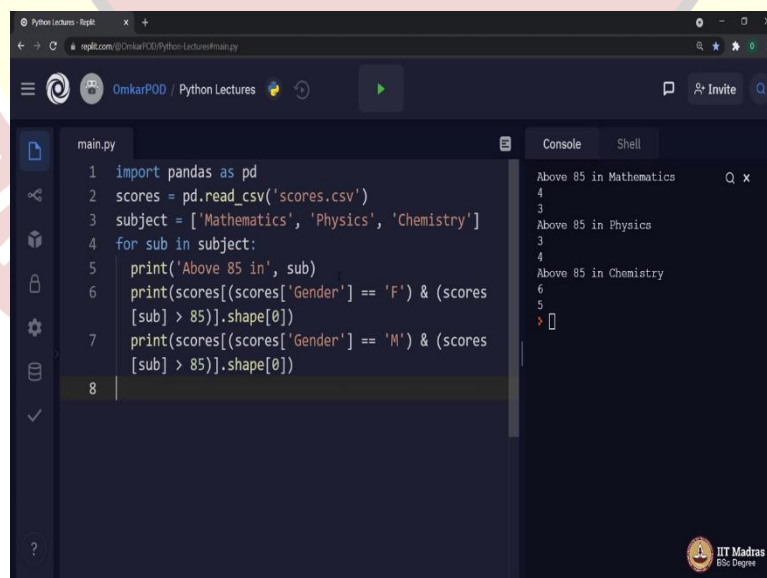
```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 subject = ['Mathematics', 'Physics', 'Chemistry']
4 for sub in subject:
5     print(scores[(scores['Gender'] == 'F') & (scores[sub] > 85)].shape[0])
6     print(scores[(scores['Gender'] == 'M') & (scores[sub] > 85)].shape[0])
7
```

The console output shows the following results:

```
4
3
3
4
6
5
```

This is working without any error, but this looks like some random numbers. Let us write one print statement. Above 85 in sub. Now let us try it.

(Refer Slide Time: 21:36)



```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 subject = ['Mathematics', 'Physics', 'Chemistry']
4 for sub in subject:
5     print('Above 85 in', sub)
6     print(scores[(scores['Gender'] == 'F') & (scores[sub] > 85)].shape[0])
7     print(scores[(scores['Gender'] == 'M') & (scores[sub] > 85)].shape[0])
8
```

The console output shows the following results:

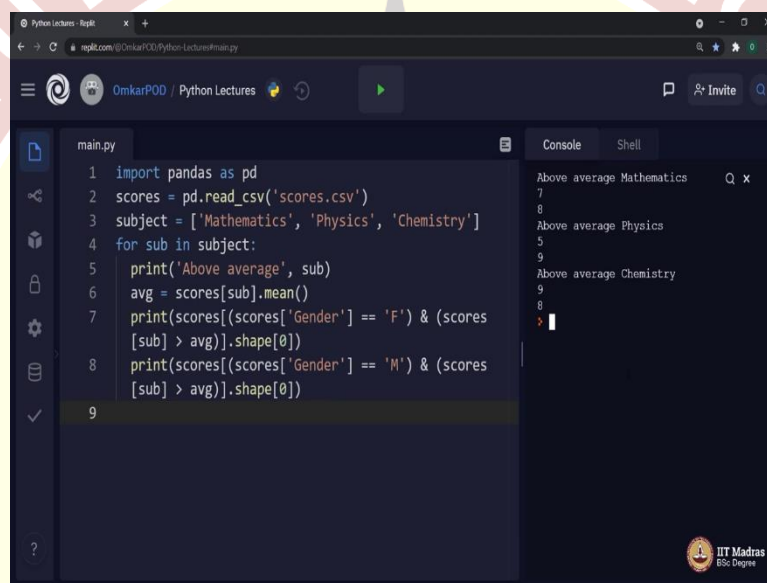
```
Above 85 in Mathematics
4
3
Above 85 in Physics
3
4
Above 85 in Chemistry
6
5
```

Above 85 in mathematics 4, 3. Now we know first is female, second is male. So, 4 girls, 3 boys; above 85 in physics, 3 girls, 4 boys; above 85 in chemistry, 6 girls, 5 boys. This is regarding all the students who are about 85, which means we are focusing only on small

fraction of students. But most of the times, teachers are more interested in knowing how many students are scoring above average or how many students are scoring below average.

Now, how to do that? Is it too complicated? Of course not. We have to make very small change in this particular code. First, we have to find average marks for individual subjects. We already know how to do that. Scores of subject dot mean. This is a mathematical mean, which will give us the average of the given subject. Now, this variable avg will hold the average of a subject. We simply have to replace 85 with average, and then this print statement above average. Let us execute.

(Refer Slide Time: 23:25)



```
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 subject = ['Mathematics', 'Physics', 'Chemistry']
4 for sub in subject:
5     print('Above average', sub)
6     avg = scores[sub].mean()
7     print(scores[(scores['Gender'] == 'F') & (scores
8 [sub] > avg)].shape[0])
9     print(scores[(scores['Gender'] == 'M') & (scores
10 [sub] > avg)].shape[0])
```

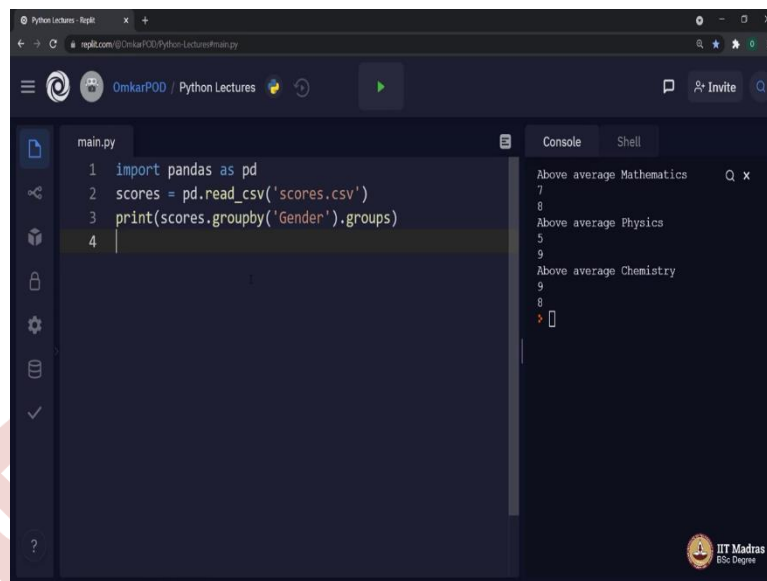
The console output shows the following results:

```
Above average Mathematics 7
8
Above average Physics 5
9
Above average Chemistry 9
8
```

Above Average mathematics, 7, 8; above average physics 5, 9; above average chemistry, 9, 8. I believe all these numbers are accurate, but if you doubt, you can always go back to our scores dataset and check it manually, and then tell me how difficult that was as compared to this code.

Once again, now you must be thinking is that all we can do with Pandas? And once again the answer is of course not. This is only the basic things, which you can do with Pandas. So before closing this lecture, let us introduce one more feature of Pandas called as groupby.

(Refer Slide Time: 24:22)



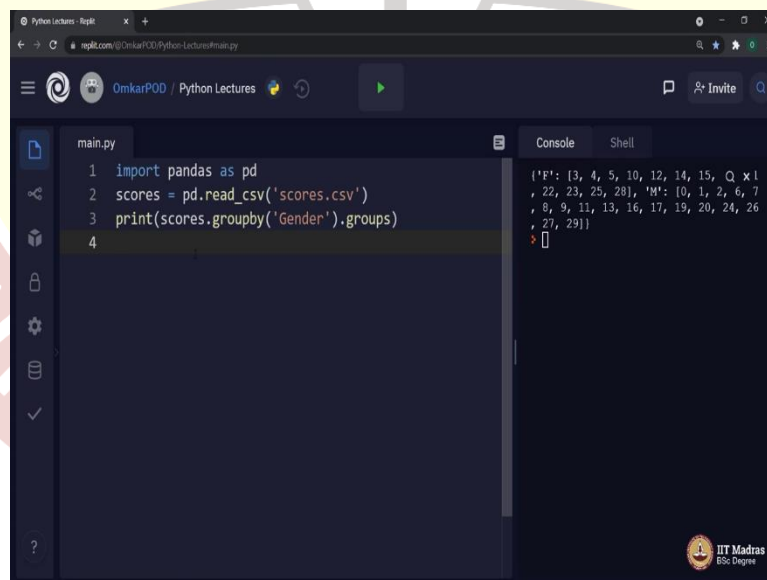
```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores.groupby('Gender').groups)
4
```

Console

```
Above average Mathematics
7
8
Above average Physics
5
9
Above average Chemistry
9
8
>
```

Observe this particular line, print scores, which is this data frame dot groupby is a function to which we have passed one parameter which is nothing but a column heading and then we are saying dot groups. Let us first execute this code and then I will try to explain what is happening over here.

(Refer Slide Time: 24:52)



```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 print(scores.groupby('Gender').groups)
4
```

Console

```
{'F': [3, 4, 5, 10, 12, 14, 15, 22, 23, 25, 28], 'M': [0, 1, 2, 6, 7, 8, 9, 11, 13, 16, 17, 19, 20, 24, 26, 27, 29]}
```

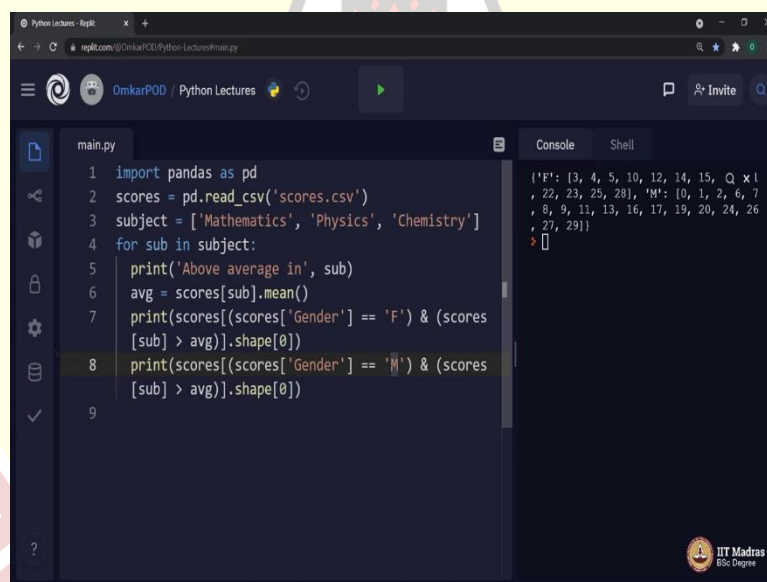
Observe this output it is printing a dictionary of lists. F is a key in the dictionary and as a value we have a list of some numbers. Then second key is M. And again, a list of numbers. Can you guess what these numbers are representing over here? Correct, all these numbers represent the card numbers, where gender is F, and this second list represents all those card

numbers where gender is M, which means, using this particular groupby gender dot groups returns a dictionary where every key is an unique value from this column gender.

As we know, our gender column has only two values hence, we are seeing two keys F and M, and the entire data available in data frame is clubbed together against these keys on the basis of their index values.

You might think this is fine. It is printing a list of card numbers of all male students, all female students and so on, but what is the use of all this? Where can we use this? And the answer is, this groupby feature of Pandas comes very handy when we try to divide our data into bills. I hope you remember billing concept from computational thinking. If you remember, in our previous example, we had to print the same statement twice. Let me quickly go back to the previous code.

(Refer Slide Time: 27:03)

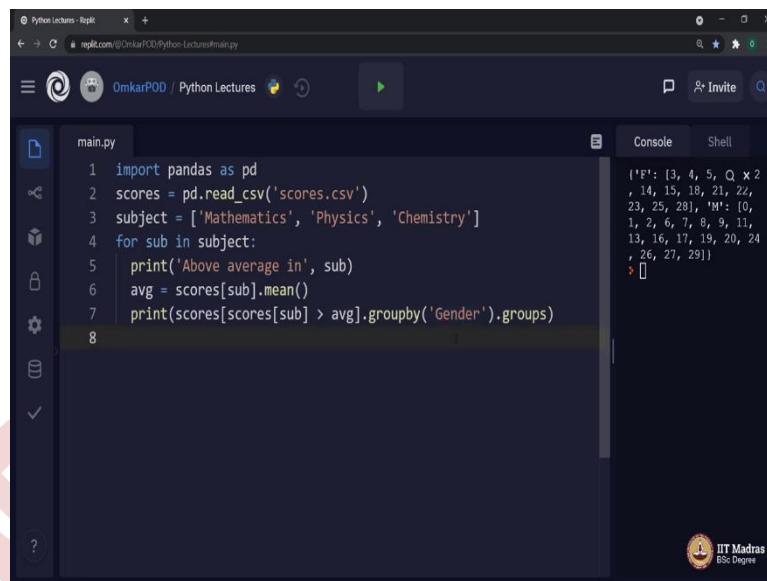


```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 subject = ['Mathematics', 'Physics', 'Chemistry']
4 for sub in subject:
5     print('Above average in', sub)
6     avg = scores[sub].mean()
7     print(scores[(scores['Gender'] == 'F') & (scores
8 [sub] > avg)].shape[0])
9     print(scores[(scores['Gender'] == 'M') & (scores
10 [sub] > avg)].shape[0])
11
```

```
Console
Shell
{'F': [3, 4, 5, 10, 12, 14, 15, 22, 23, 25, 28], 'M': [0, 1, 2, 6, 7, 8, 9, 11, 13, 16, 17, 19, 20, 24, 26, 27, 29]}
```

Here we are printing the same print statement twice, one for F once for M. And if you remember, when I added these subjects, I said we are programmers, we do not repeat same thing twice, we use loops. But writing a loop to iterate over only two values is not a good idea. And this is the place where we can use that particular feature, which we saw just now, which is groupby. Let me modify this code first, then I will try to explain how it works.

(Refer Slide Time: 27:48)



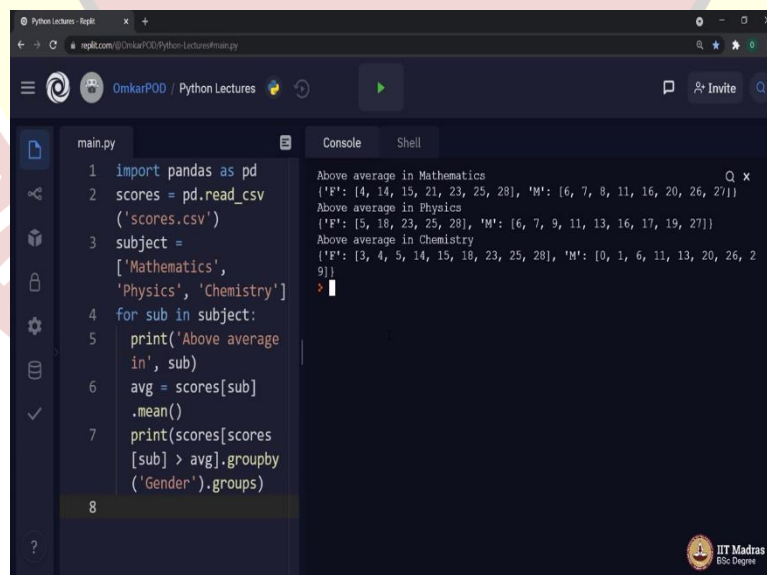
```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 subject = ['Mathematics', 'Physics', 'Chemistry']
4 for sub in subject:
5     print('Above average in', sub)
6     avg = scores[sub].mean()
7     print(scores[scores[sub] > avg].groupby('Gender').groups)
8
```

Console

```
{'F': [3, 4, 5, 14, 15, 18, 21, 22, 23, 25, 28], 'M': [0, 1, 2, 6, 7, 8, 9, 11, 13, 16, 17, 19, 20, 24, 26, 27, 29]}
```

Look at the scores line number 7. First, we are saying scores of subject greater than average, that is the only condition we have. If the value is above average then we are collecting that entire data frame. And then, on top of that entire data frame we are using this groupby function which will be applied using this particular column gender. And at the end dot groups, which will print all the possible groups with respect to this column in form of a dictionary of lists. Let us execute.

(Refer Slide Time: 28:38)



```
main.py
1 import pandas as pd
2 scores = pd.read_csv('scores.csv')
3 subject = ['Mathematics', 'Physics', 'Chemistry']
4 for sub in subject:
5     print('Above average in', sub)
6     avg = scores[sub].mean()
7     print(scores[scores[sub] > avg].groupby('Gender').groups)
8
```

Console

```
Above average in Mathematics
{'F': [4, 14, 15, 21, 23, 25, 28], 'M': [6, 7, 8, 11, 16, 20, 26, 27]}
Above average in Physics
{'F': [5, 18, 23, 25, 28], 'M': [6, 7, 9, 11, 13, 16, 17, 19, 27]}
Above average in Chemistry
{'F': [3, 4, 5, 14, 15, 18, 23, 25, 28], 'M': [0, 1, 6, 11, 13, 20, 26, 29]}
```

Observe this output. Above the average in mathematics, F, these many students; M, these many students, then physics F and M; chemistry F and M. In this case, we are not just getting



the number we are also getting their card numbers or their index values, which are even more useful in order to refer a particular student.

This is the place where we will stop with respect to Pandas in this course. But let me give you one very important instruction. Whatever we have seen so far with respect to python Pandas is just the tip of the iceberg. We have not seen the entire functionality of Pandas, because in this particular course, whatever we have seen so far is more than sufficient.

As you go ahead with the rest of the courses, at a diploma level or at a degree level you will learn many more features of Pandas whenever they are required. So do not break your head over other features, which you might come across while reading Pandas' documentation. Whatever we have covered in these two lectures based on Pandas is more than sufficient for this particular course. Thank you for watching this lecture. Happy learning.

