Spark Azure Databricks

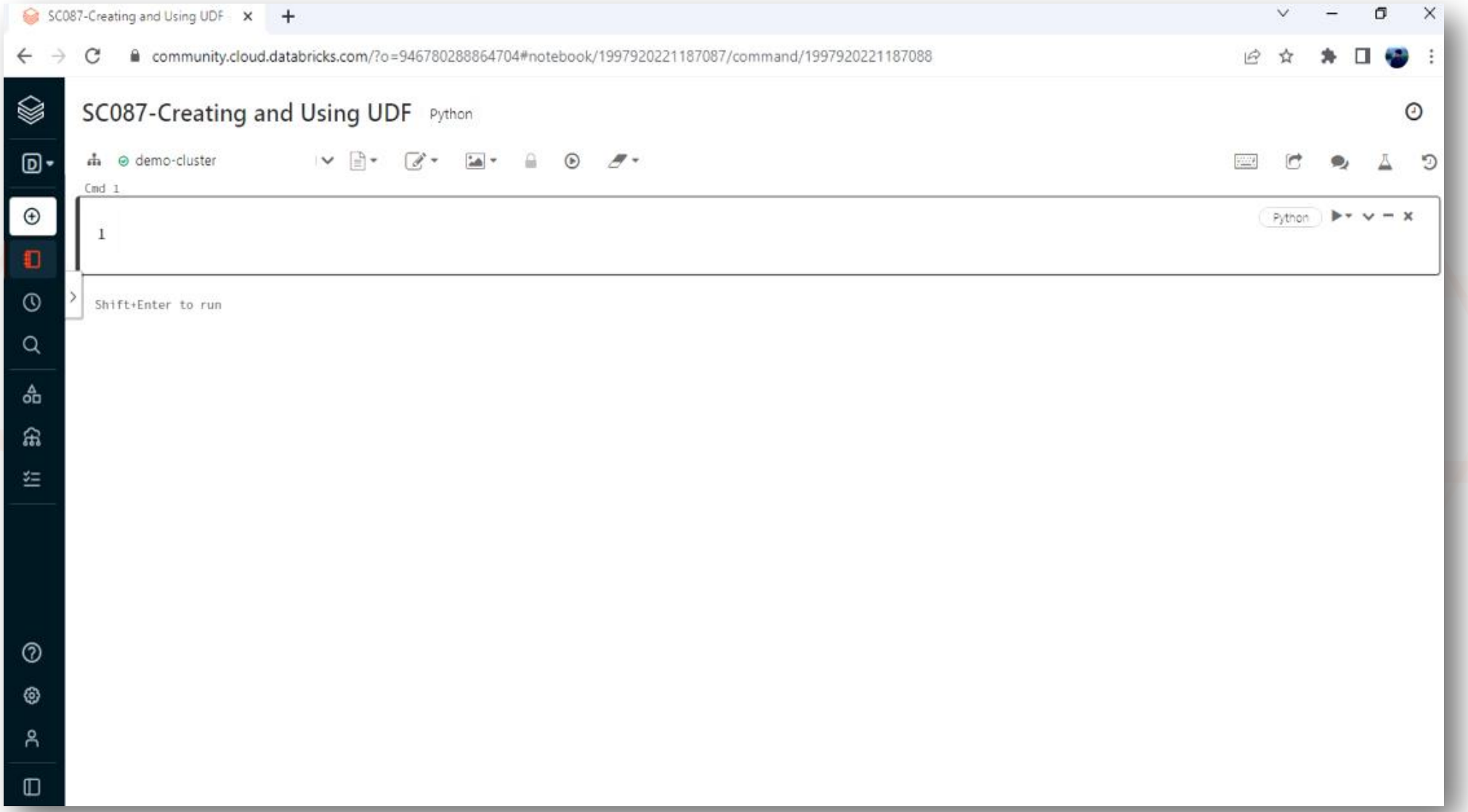
Databricks Spark Certification and Beyond

Module:
Miscellaneous Topics

Lecture:
Creating and Using UDFs

Creating and Using UDFs

Let me go to the Databricks workspace and create a new notebook
(Reference: SC087-Creating and using UDF)



Here I have a CSV data file, and I want to load it and create a Dataframe.
We have Male and Female values in this column. However, the values are not standardized.
Scroll down to see more values from this column, and you will see Female, M, Male, Male-ish, Trans-female, Cis-female, and a bunch of other strings. **(Reference: /data/survey.csv)**

Cmd 1

```
1  survey_df = spark.read \
2                  .option("header", "true") \
3                  .option("inferSchema", "true") \
4                  .csv("/FileStore/survey.csv")
5
6  display(survey_df)
```

▸ (3) Spark Jobs

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_
1	2014-08-27T11:29:31.000+0000	37	Female	United States	IL	NA	No	Yes	Often
2	2014-08-27T11:29:37.000+0000	44	M	United States	IN	NA	No	No	Rarely
3	2014-08-27T11:29:44.000+0000	32	Male	Canada	NA	NA	No	No	Rarely
4	2014-08-27T11:29:46.000+0000	31	Male	United Kingdom	NA	NA	Yes	Yes	Often
5	2014-08-27T11:30:22.000+0000	31	Male	United States	TX	NA	No	No	Never
6	2014-08-27T11:31:22.000+0000	33	Male	United States	TN	NA	Yes	No	Somet
7	2014-08-27T11:31:50.000+0000	35	Female	United States	MI	NA	Yes	Yes	Somet

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 11.05 seconds -- by prashant@scholarnest.com at 8/8/2022, 11:29:13 PM on demo-cluster

We have a gender column, and the values in this column are not standard.
I want to standardize it.
We want three values: Male, Female, or Unknown.

Cmd 1

```
1  survey_df = spark.read \
2                  .option("header", "true") \
3                  .option("inferSchema", "true") \
4                  .csv("/FileStore/survey.csv")
5
6  display(survey_df)
```

▸ (3) Spark Jobs

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_
1	2014-08-27T11:29:31.000+0000	37	Female	United States	IL	NA	No	Yes	Often
2	2014-08-27T11:29:37.000+0000	44	M	United States	IN	NA	No	No	Rarely
3	2014-08-27T11:29:44.000+0000	32	Male	Canada	NA	NA	No	No	Rarely
4	2014-08-27T11:29:46.000+0000	31	Male	United Kingdom	NA	NA	Yes	Yes	Often
5	2014-08-27T11:30:22.000+0000	31	Male	United States	TX	NA	No	No	Never
6	2014-08-27T11:31:22.000+0000	33	Male	United States	TN	NA	Yes	No	Somet
7	2014-08-27T11:31:50.000+0000	35	Female	United States	MI	NA	Yes	Yes	Somet

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 11.05 seconds -- by prashant@scholarnest.com at 8/8/2022, 11:29:13 PM on demo-cluster

Well, you can transform the gender column using the withColumn() method. And you can use some string pattern matching techniques or regular expression functions to do this.
Spark offers you some regular expression functions, which you can easily do.
But let us assume spark does not offer you a function to do this.
Can I create my own function and use it in Spark Dataframe expression?
Yes. You can do that.
You can define your own function in Python and use it to transform a Spark Dataframe.

Here is a user defined python function.
So I defined a parse_gender() function.
This function takes a string value and applies some regular expression techniques to determine if the given value is a Male or a Female.

Cmd 2

```
1  def parse_gender(gender):
2      import re
3      female_pattern = r"^f$|f.m|w.m"
4      male_pattern = r"^m$|ma|m.l"
5      if re.search(female_pattern, gender.lower()):
6          return "Female"
7      elif re.search(male_pattern, gender.lower()):
8          return "Male"
9      else:
10         return "Unknown"
```


I defined two regular expression patterns.
The first one is the female_pattern and then a male_pattern.

Cmd 2

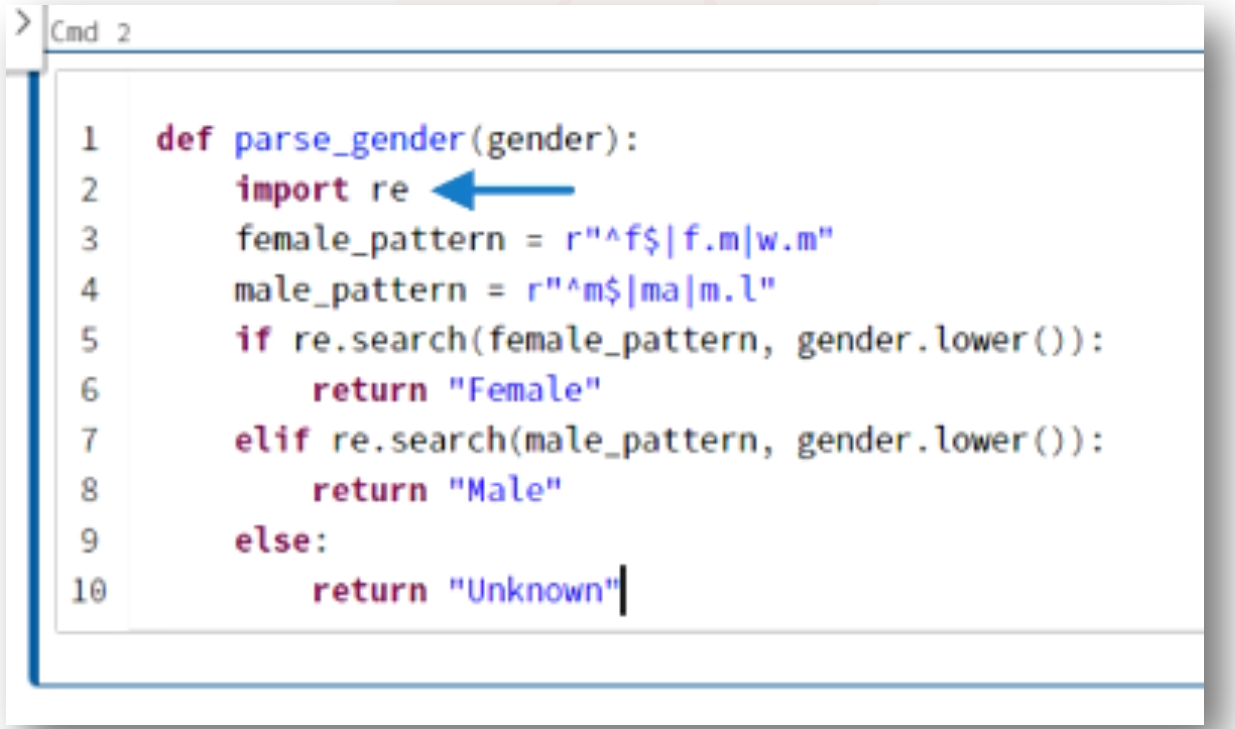
```
1   def parse_gender(gender):
2       import re
3       female_pattern = r"^f$|f.m|w.m"
4       male_pattern = r"^m$|ma|m.l"
5       if re.search(female_pattern, gender.lower()):
6           return "Female"
7       elif re.search(male_pattern, gender.lower()):
8           return "Male"
9       else:
10          return "Unknown"
```

Then I match the gender with the female_pattern.
And return Female when it matches with the female_pattern.
Otherwise, I will match it with the male_pattern and return Male.
If nothing matches, we will return Unknown.

Cmd 2

```
1   def parse_gender(gender):
2       import re
3       female_pattern = r"^f$|f.m|w.m"
4       male_pattern = r"^m$|ma|m.l"
5       if re.search(female_pattern, gender.lower()):
6           return "Female"
7       elif re.search(male_pattern, gender.lower()):
8           return "Male"
9       else:
10          return "Unknown"
```


This function requires a Python "re" package.
So I am importing the re-package inside the function. And this is important to import the "re" package inside the function. I cannot import it outside the function and expect it to work. Why? Spark will send this function to the executors, and they will run it. If I import the re package outside the function body, it remains at the driver. Because Spark driver will only send the function body. But if I import the ‘re’ package inside the function, the import statement also goes to the executors.



Here I am testing my UDF.
If I pass some malish string, it returns Male. And that's what I wanted.
But this is a Python function.
Spark driver doesn't know we want to send this function to the executors.
You must inform the Spark driver, and you can do it using the udf() function.

The udf() function takes two arguments.
The first argument is the Python function name. And the second argument is the return type. My function returns a string, so the return type is StringType(). The default value of the return type is a string. So you can skip the second argument. But let me keep it here as a best practice. he udf() function will inform the Spark driver to send this function to the executors. And it will give back a new function handle. Now you can use the new handle instead of using your Python function. The new handle parse_gender_udf is your user-defined function.

Now we are using our UDF.
I am using parse_gender_udf function in the withColumn() to transform the Gender column.
And I am using it the same way I use any built-in Dataframe function.
If you check the Gender column, and you will see standard values.

Cmd 5

```
1  survey_df2 = survey_df.withColumn("Gender", parse_gender_udf("Gender"))
2  display(survey_df2)
```

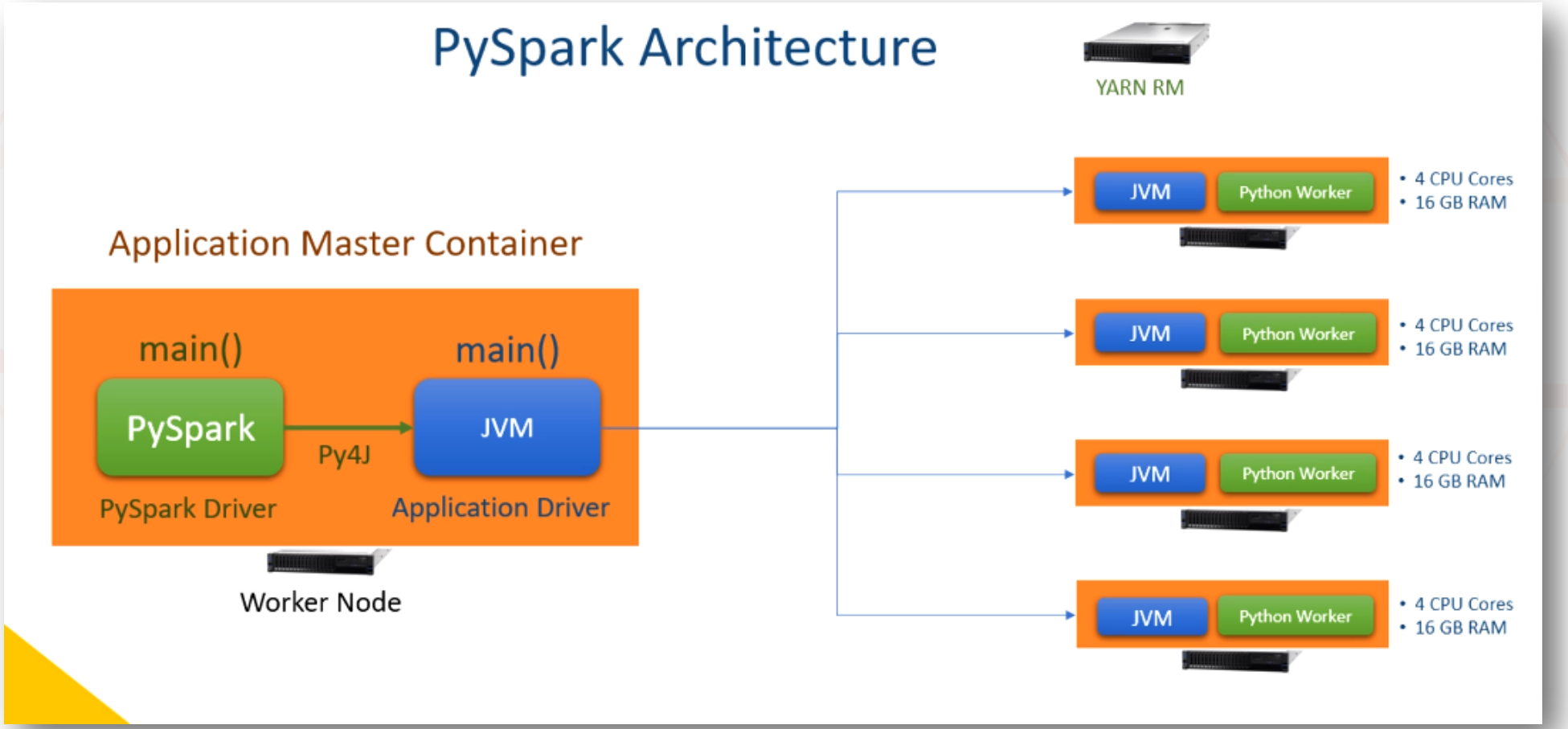
▸ (1) Spark Jobs

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere
1	2014-08-27T11:29:31.000+0000	37	Female	United States	IL	NA	No	Yes	Often
2	2014-08-27T11:29:37.000+0000	44	Male	United States	IN	NA	No	No	Rarely
3	2014-08-27T11:29:44.000+0000	32	Male	Canada	NA	NA	No	No	Rarely
4	2014-08-27T11:29:46.000+0000	31	Male	United Kingdom	NA	NA	Yes	Yes	Often
5	2014-08-27T11:30:22.000+0000	31	Male	United States	TX	NA	No	No	Never
6	2014-08-27T11:31:22.000+0000	33	Male	United States	TN	NA	Yes	No	Sometimes
7	2014-08-27T11:31:50.000+0000	35	Female	United States	MI	NA	Yes	Yes	Sometimes

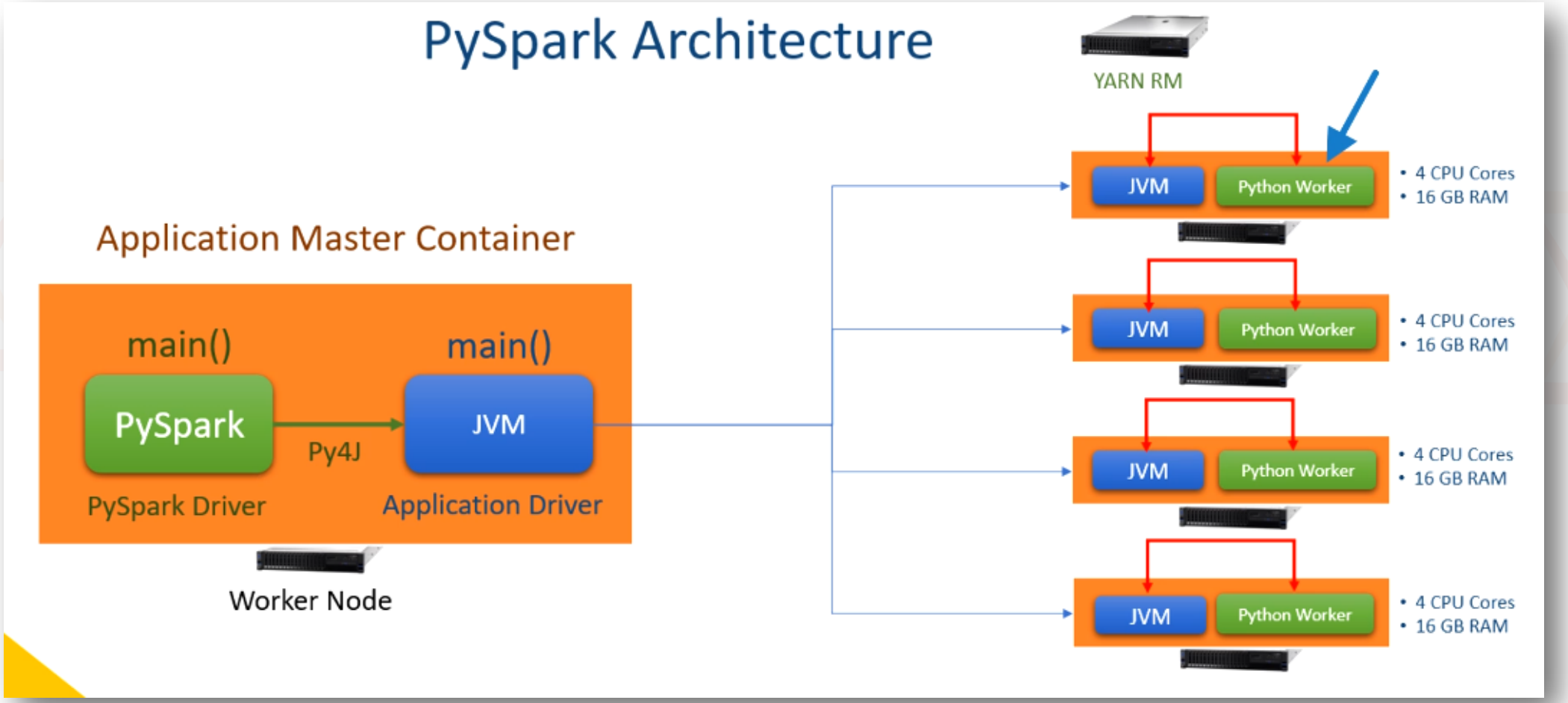
Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 2.56 seconds -- by prashant@scholarnest.com at 8/8/2022, 11:46:42 PM on demo-cluster

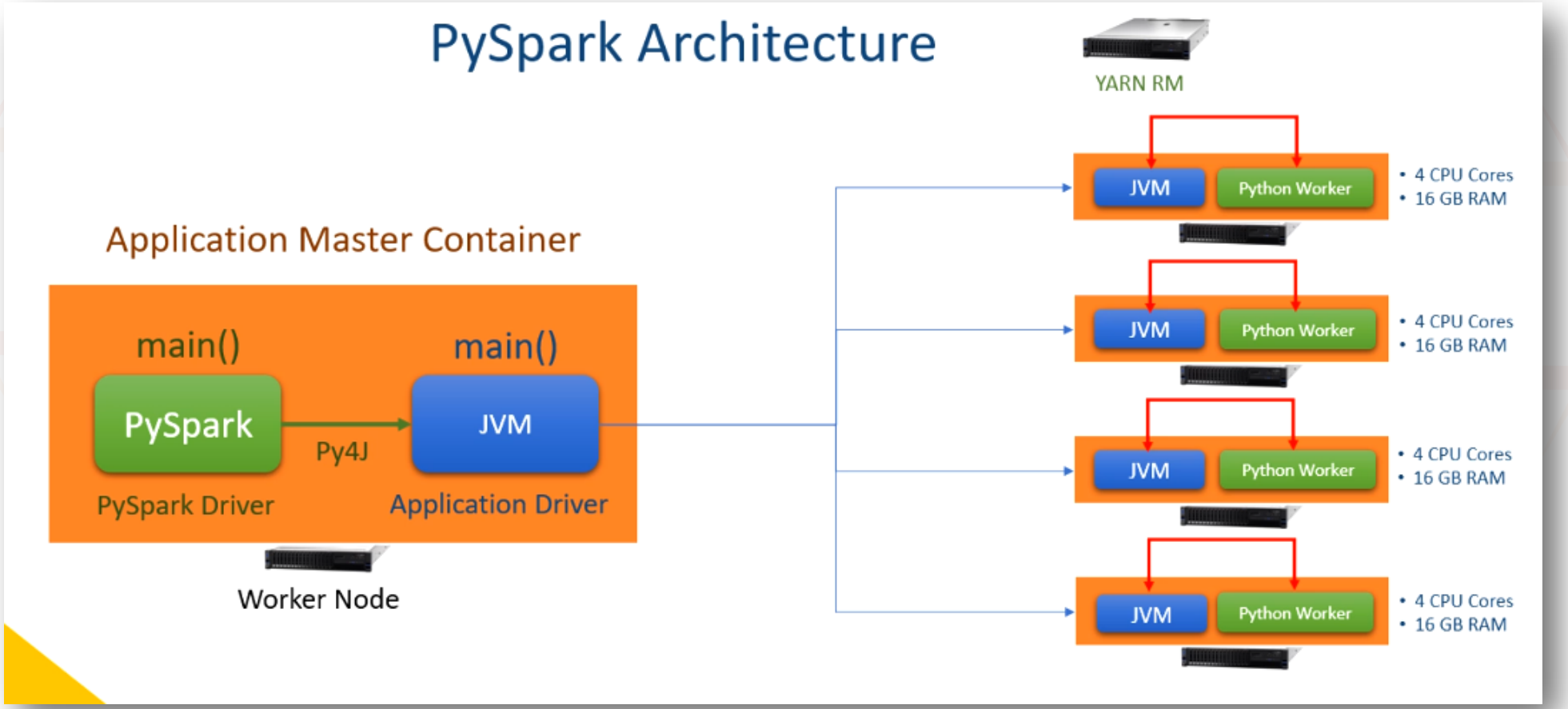
User-defined functions in PySpark are an easy way to develop custom logic and use it to transform your dataframe columns.
However, they execute inside the Python worker at the executors.
So you will need some extra memory to run Python workers.



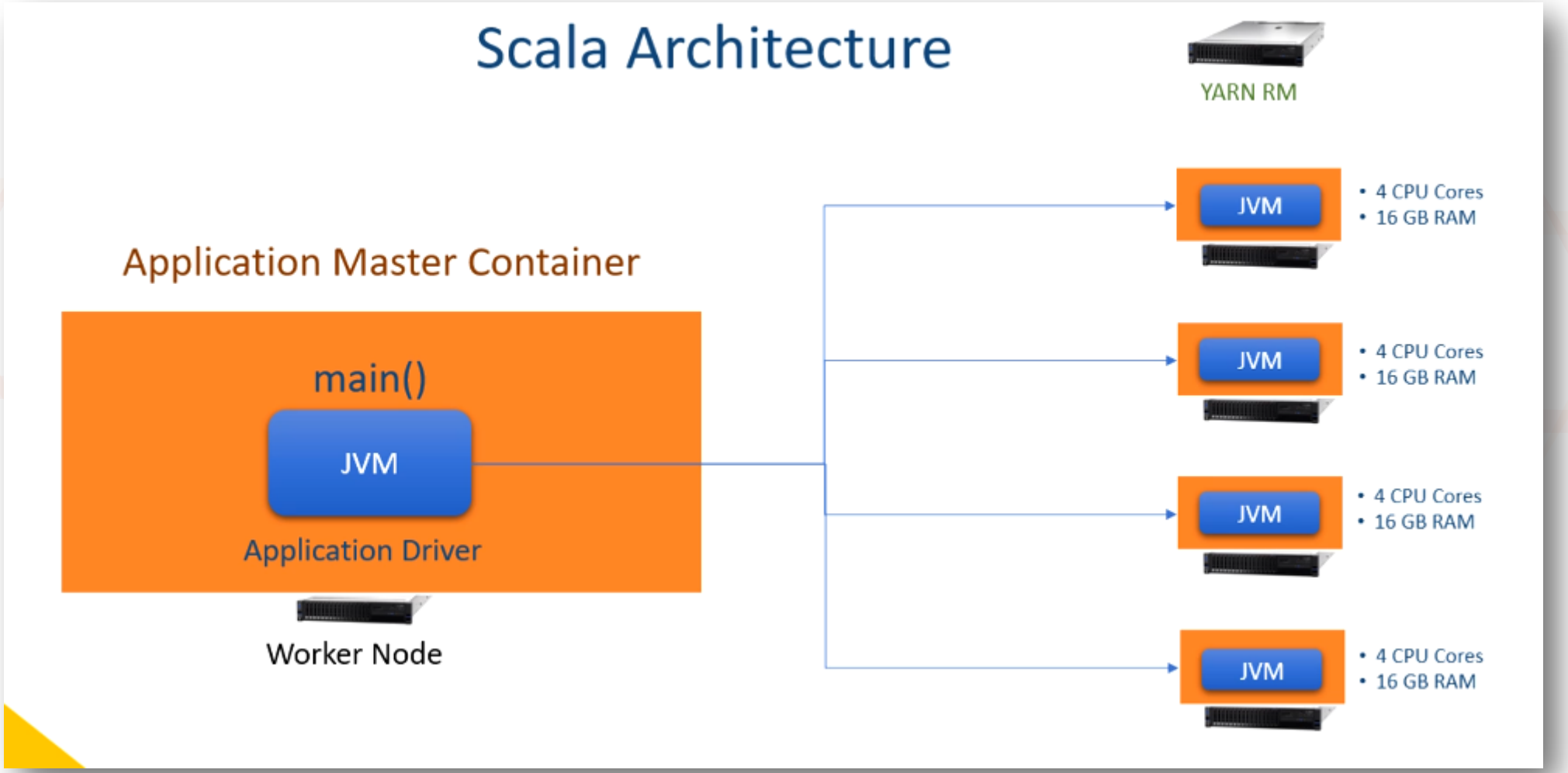
If you use a Python UDF, your UDF code runs inside the Python worker. And data will move from the JVM executor to the Python worker. Then the result will also come back from the Python worker to the JVM. And that's why UDF is not recommended in PySpark.



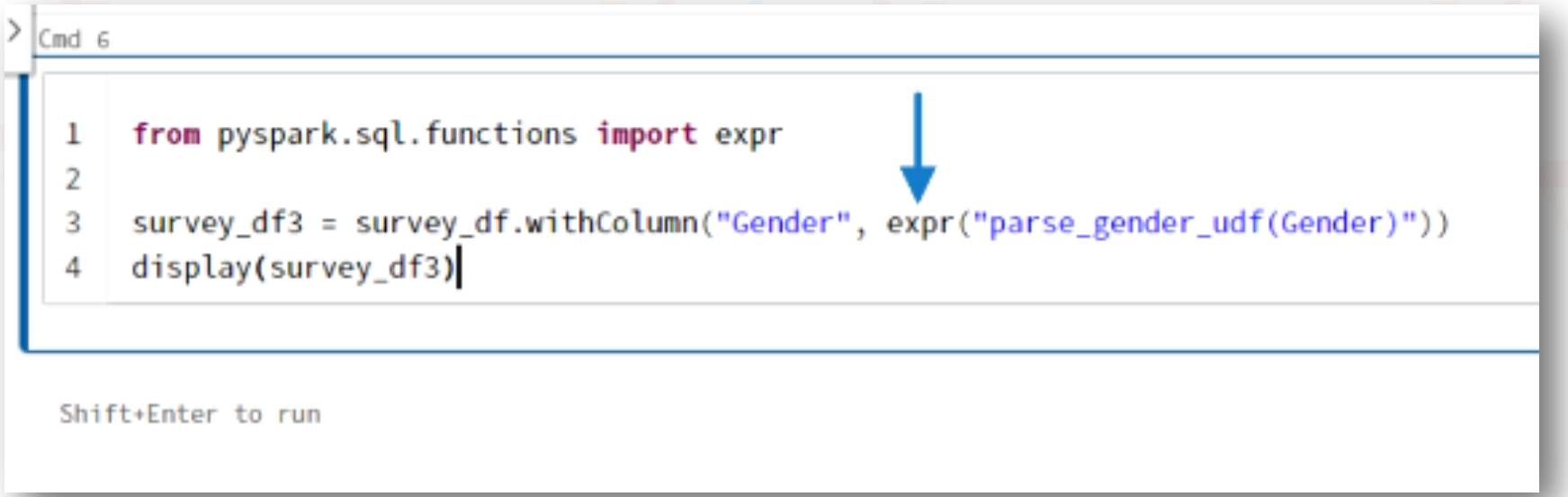
Spark offers you a lot of functions to do most things.
And they keep on adding new functions with every new release.
So try managing your requirement with Spark's built-in functions and avoid creating UDF.
Because UDF could cost you massive performance problems.



And this is the only place where Scala is better than PySpark.
Scala UDF runs inside the JVM and doesn't transfer data between JVM and Python worker.
So the Scala code does not suffer from this problem.



So we learned to create and use the UDF. However, this UDF is only available as a Dataframe expression. You cannot use this UDF in a SQL-like expression as shown below.
We are doing the same thing here.
I am using withColumn() method and transforming the Gender column using the parse_gender_udf() function.
But this time, I am using the expr() function evaluate is an SQL-like expression.



Look at the previous expression and compare it with the current one.
Do you see the difference?
The old one uses Dataframe expression. The new one uses SQL-like string expression.
The new one will not work. Because the udf() function does not register your function in the Spark catalog. As a result, you can use it as a Dataframe expression. But it is not a SQL function, and you cannot use it in the SQL-like expression.

Cmd 5

```
1  survey_df2 = survey_df.withColumn("Gender", parse_gender_udf("Gender"))
2  display(survey_df2)
```

Show result

Cmd 6

```
1  from pyspark.sql.functions import expr
2
3  survey_df3 = survey_df.withColumn("Gender", expr("parse_gender_udf(Gender)"))
4  display(survey_df3)
```


You will see the following error when using SQL-like expressions on your UDF.
How to handle it?
Well! that's not difficult.

Cmd 6

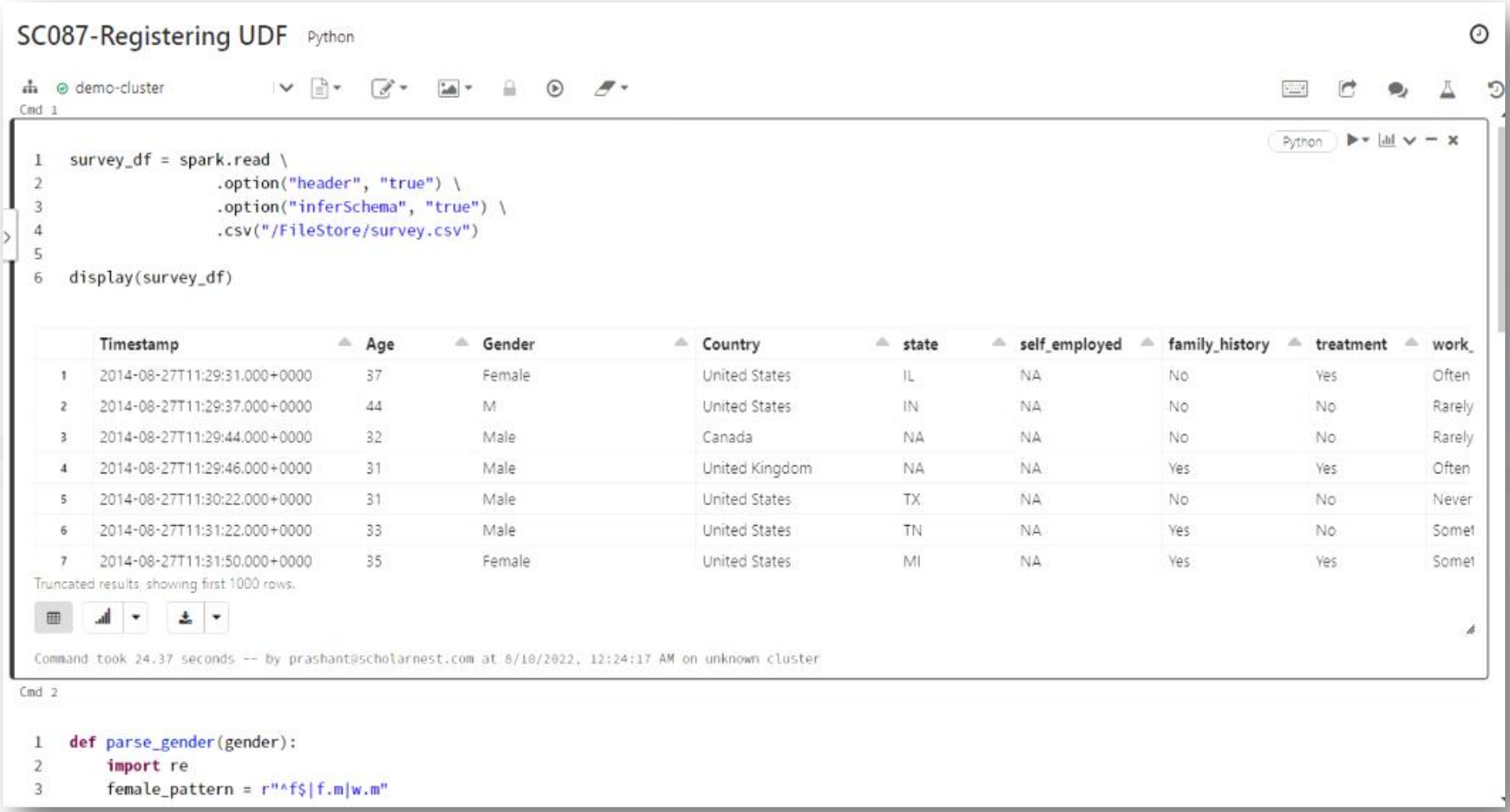
```
1  from pyspark.sql.functions import expr
2
3  survey_df3 = survey_df.withColumn("Gender", expr("parse_gender_udf(Gender)"))
4  display(survey_df3)
```

```
AnalysisException: Undefined function: parse_gender_udf. This function is neither a built-in/temporary function, nor a persistent function that is qualifie
  d as spark_catalog.default.parse_gender_udf.; line 1 pos 0
```

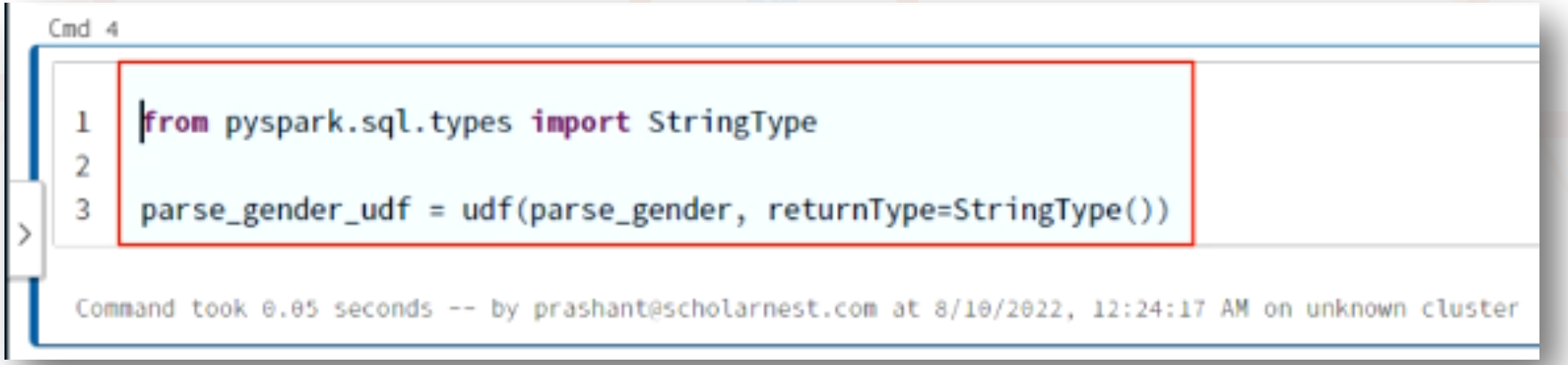
Command took 0.21 seconds -- by prashant@scholarnest.com at 8/8/2022, 11:54:35 PM on demo-cluster

I have cloned my previous notebook and renamed the cloned notebook.

(Reference: SC087-Registering UDF)



I will keep the first three cells and run it to create the Dataframe, then create the Python function and finally test if the Python function is working as expected.
Now the next cell is to inform the Spark driver about the UDF.
This approach allows me to create a new UDF handle that I can use in the dataframe expression.
But I want to use my UDF as a SQL function.
So I will delete this cell because the udf() function does not allow me to do that.



The following cell uses the UDF in a Dataframe expression.
So let me delete this also.

Add a new cell above this and write code to register the UDF as a SQL function.
This code is similar to the udf() function. But I am using *spark.udf.register()* function. The register function will register my Python function as a SQL function. The register function takes three arguments. The first argument is the new name of your UDF. The second argument is the Python function. The last one is the return type of your Python function. I am giving a new name to my UDF. However, you can use the same name. But I give a new name to my UDF as a best practice.

If you want to use your Python function in a Dataframe expression, you will use the udf() function we learned earlier.
But if you want to use your Python function in the SQL expression, you can use the register() function.

Cmd 4

```
1  from pyspark.sql.functions import StringType
2
3  spark.udf.register("parse_gender_udf", parse_gender, StringType())
```

```
Out[4]: <function __main__.parse_gender(gender)>
```

```
Command took 0.12 seconds -- by prashant@scholarnest.com at 8/10/2022, 12:30:27 AM on demo-cluster
```


Now if you try running the next cell, you will see that it works.
So, you can even register your Python function to work as a SQL function and use it.

Cmd 5

```
1  from pyspark.sql.functions import expr
2
3  survey_df3 = survey_df.withColumn("Gender", expr("parse_gender_udf(Gender)"))
4  display(survey_df3)
```

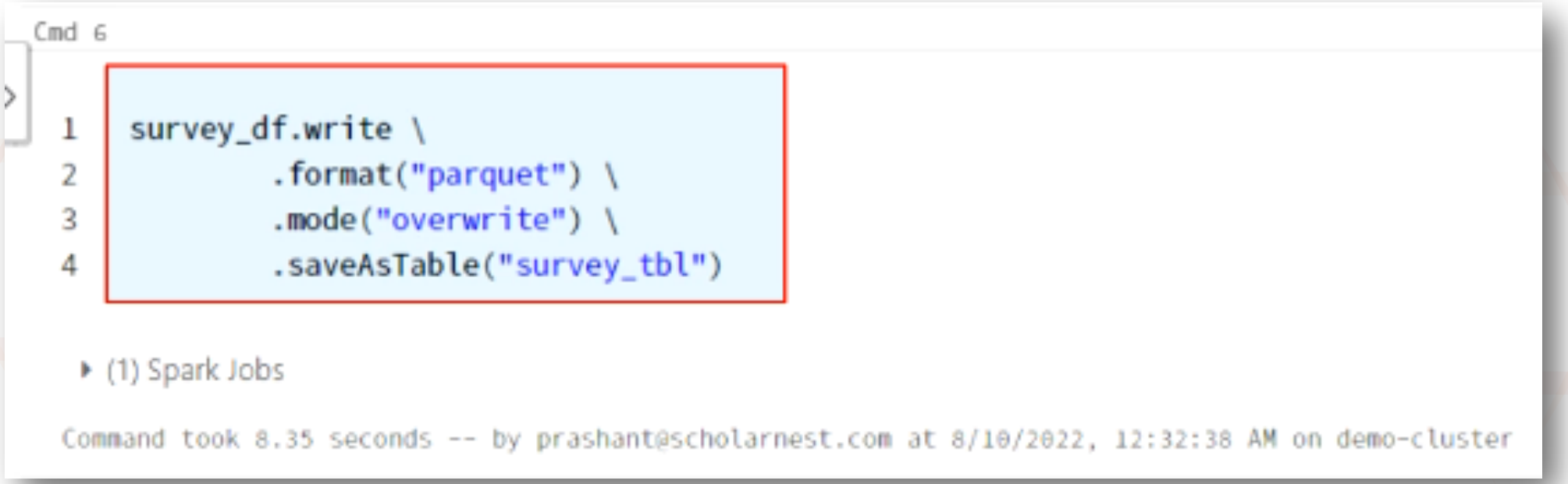
▸ (1) Spark Jobs

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere
1	2014-08-27T11:29:31.000+0000	37	Female	United States	IL	NA	No	Yes	Often
2	2014-08-27T11:29:37.000+0000	44	Male	United States	IN	NA	No	No	Rarely
3	2014-08-27T11:29:44.000+0000	32	Male	Canada	NA	NA	No	No	Rarely
4	2014-08-27T11:29:46.000+0000	31	Male	United Kingdom	NA	NA	Yes	Yes	Often
5	2014-08-27T11:30:22.000+0000	31	Male	United States	TX	NA	No	No	Never
6	2014-08-27T11:31:22.000+0000	33	Male	United States	TN	NA	Yes	No	Sometimes
7	2014-08-27T11:31:50.000+0000	35	Female	United States	MI	NA	Yes	Yes	Sometimes

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 1.50 seconds -- by prashant@scholarnest.com at 8/10/2022, 12:30:33 AM on demo-cluster

So here I am taking the survey_df and writing it as a Spark table.



Next I am trying the Spark SQL and using the parse_gender function. And you can see that it works! And I am not surprised. Because I registered my Python function as a SQL function. So it works in the SQL expressions.

Cmd 7

```
1  %sql
2  SELECT Gender, parse_gender_udf(Gender) as new_gender FROM survey_tbl
```

▸ (1) Spark Jobs

	Gender	new_gender
1	Female	Female
2	M	Male
3	Male	Male
4	Male	Male
5	Male	Male
6	Male	Male
7	Female	Female

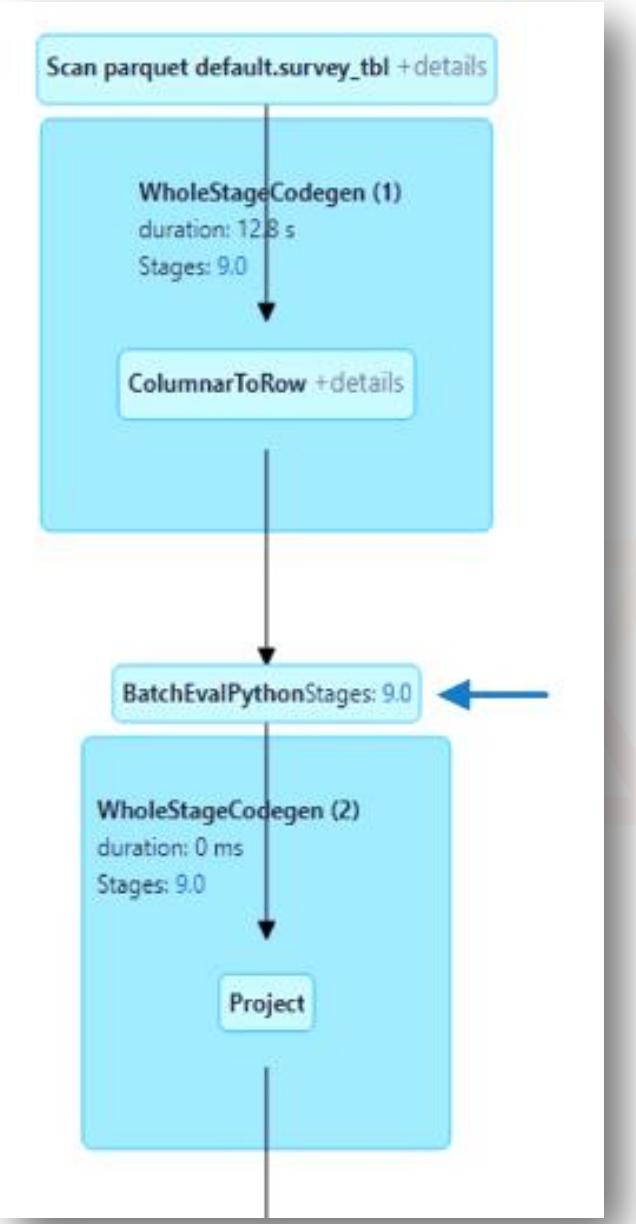
Truncated results, showing first 1000 rows.

Click to re-execute with maximum result limits.

Command took 8.61 seconds -- by prashant@scholarnest.com at 8/10/2022, 12:34:09 AM on demo-cluster

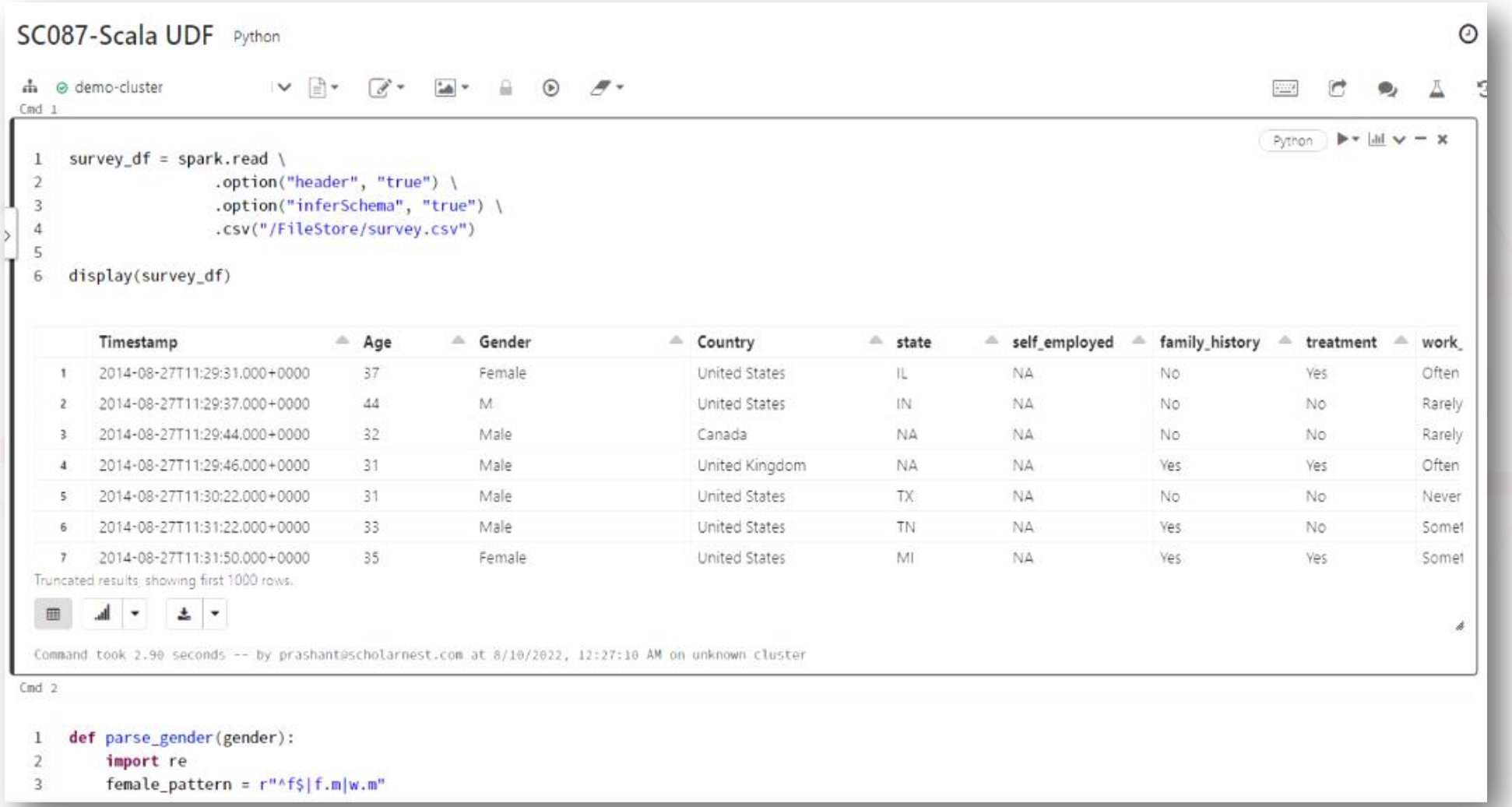
You can also go to Spark UI and see the execution plan for our last SQL.
Do you see this BatchEvalPython? You will see this in the plan using a Python UDF. This BatchEvalPython indicates your work being done at the Python worker.
And we know it is a significant performance problem.
So, how do we handle it?
Well, we have two options:

1. Do not create Python UDF and manage your work with the built-in functions. But that's impossible in some cases. What If I have a requirement and I don't have a built-in function that doesn't fulfill my purpose.
2. Create a Scala function and use a Scala UDF.



Let us see how to create a Scala UDF.

Clone the previous notebook once again **(Reference: SC087-Scala UDF)**



I will keep the first cell and create a Dataframe.
The second cell defines a Python function.
Let me replace this Python function with a Scala function.

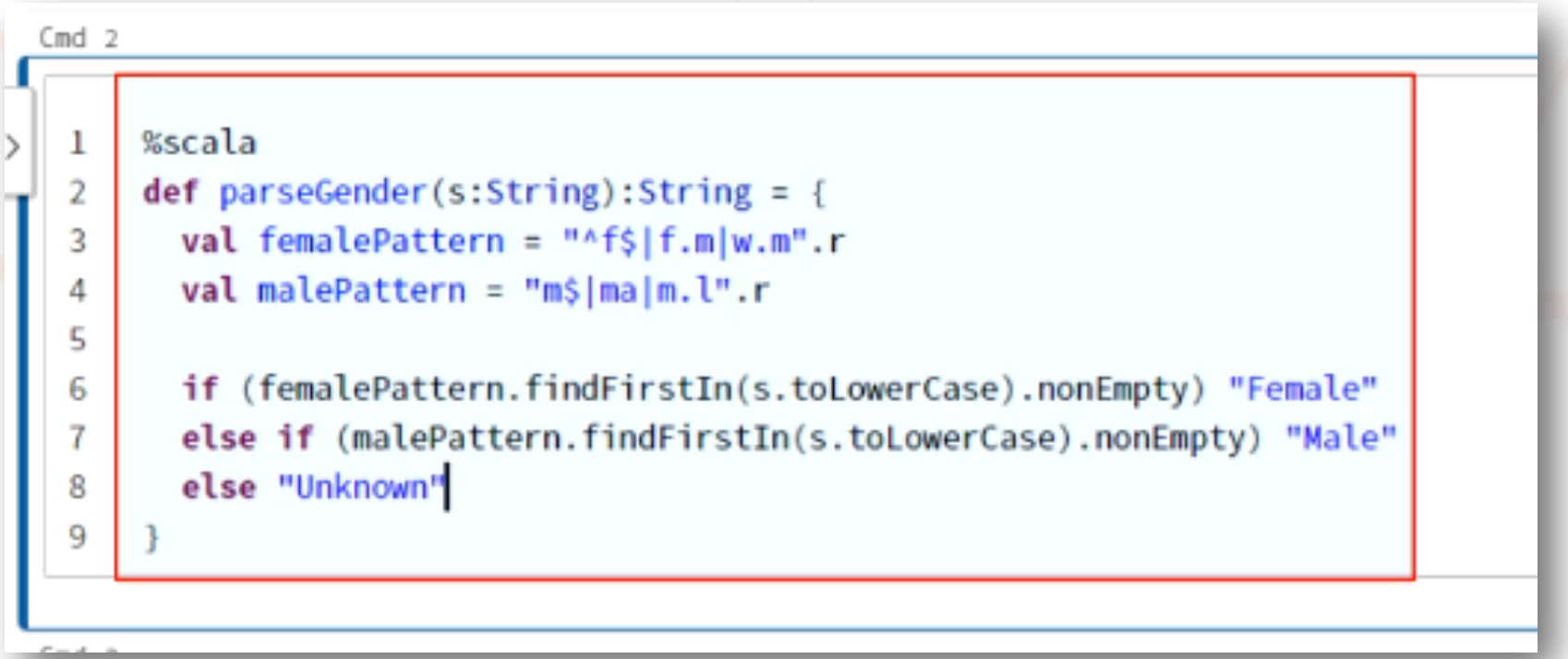
Cmd 2

```
1  def parse_gender(gender):
2      import re
3      female_pattern = r"^f$|f.m|w.m"
4      male_pattern = r"^m$|ma|m.l"
5      if re.search(female_pattern, gender.lower()):
6          return "Female"
7      elif re.search(male_pattern, gender.lower()):
8          return "Male"
9      else:
10         return "Unknown"
```

Command took 0.04 seconds -- by prashant@scholarnest.com at 8/10/2022, 12:27:21 AM on unknown cluster

Cmd 3

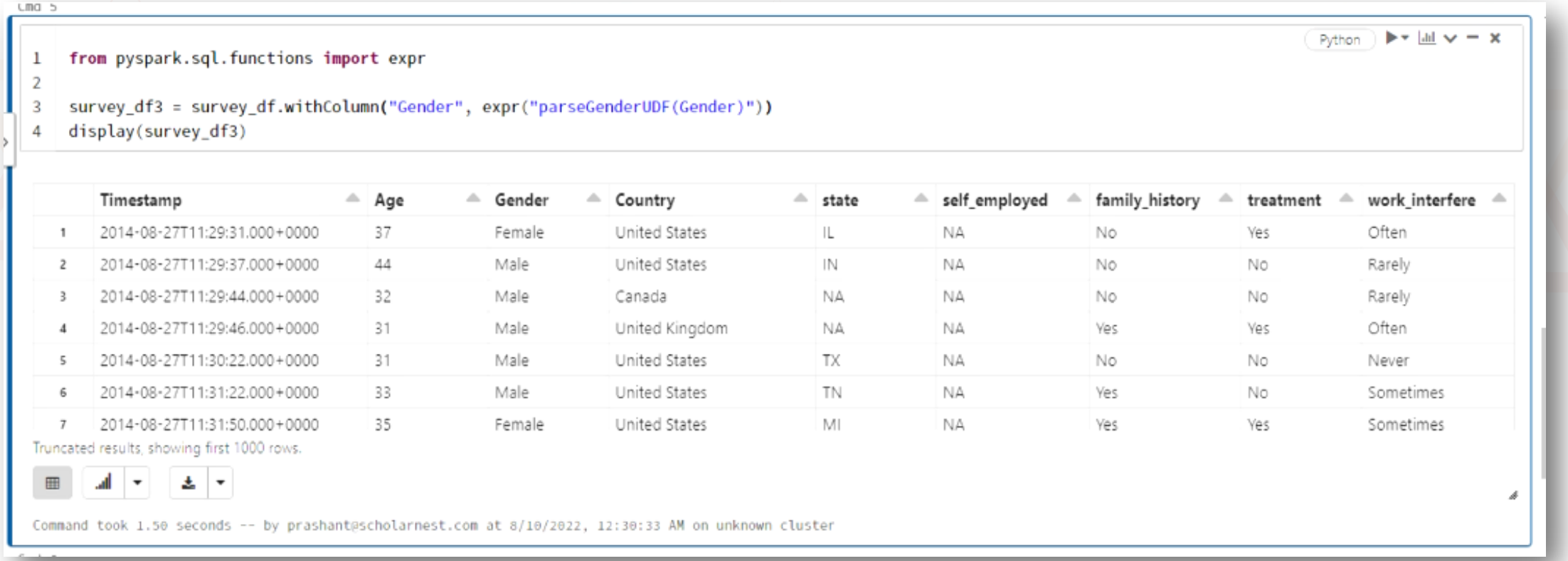
Here is the corresponding Scala function.
It is equivalent to the same Python code. You can write Scala code in the same notebook.
But mark the cell with the %scale, so the notebook knows you are running Scala code in this cell.
So I created a Scala function. The name of the function is parseGender(). I kept a different name so we know we are using the Scala function.



Then I am replacing the Python code for testing as per the Scala UDF.
And you can see that it works, and we get the desired output.

Next, I am replacing the code to register the function as well.
This one is also Scala code. Why?
Because my function is a Scala function, I need a Scala code to register it.

My function is now, a registered Spark SQL function. So it should work from PySpark and from Spark SQL. This is PySpark code shown below, I am simply replacing the UDF function name with the Scala UDF. But I am using it from the PySpark. And you can see that it worked and we got the correct output.



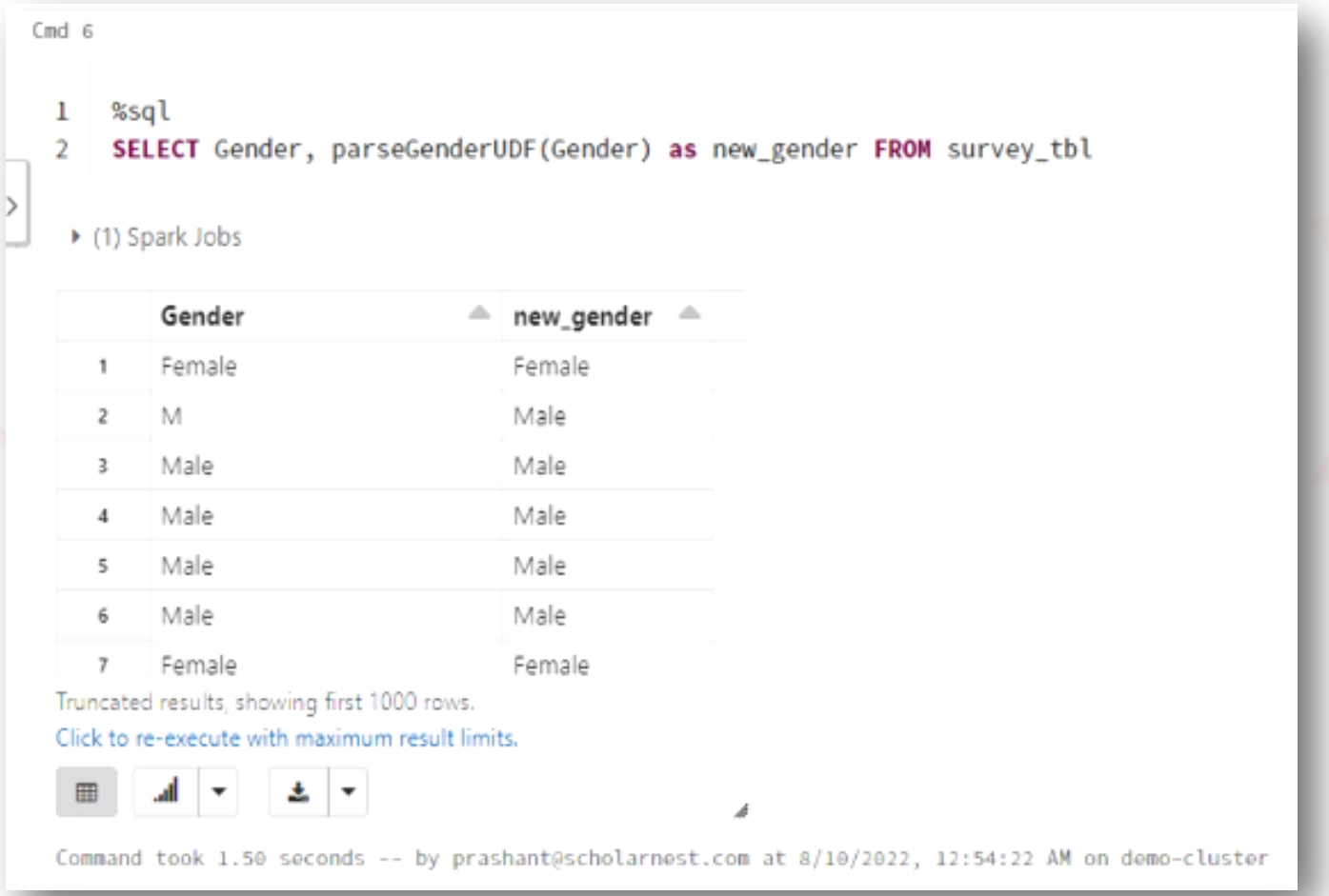
```
from pyspark.sql.functions import expr

survey_df3 = survey_df.withColumn("Gender", expr("parseGenderUDF(Gender)"))
display(survey_df3)
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere
1	2014-08-27T11:29:31.000+0000	37	Female	United States	IL	NA	No	Yes	Often
2	2014-08-27T11:29:37.000+0000	44	Male	United States	IN	NA	No	No	Rarely
3	2014-08-27T11:29:44.000+0000	32	Male	Canada	NA	NA	No	No	Rarely
4	2014-08-27T11:29:46.000+0000	31	Male	United Kingdom	NA	NA	Yes	Yes	Often
5	2014-08-27T11:30:22.000+0000	31	Male	United States	TX	NA	No	No	Never
6	2014-08-27T11:31:22.000+0000	33	Male	United States	TN	NA	Yes	No	Sometimes
7	2014-08-27T11:31:50.000+0000	35	Female	United States	MI	NA	Yes	Yes	Sometimes

Truncated results, showing first 1000 rows.

Command took 1.50 seconds -- by prashant@scholarnest.com at 8/10/2022, 12:30:33 AM on unknown cluster

Will it also work from Spark SQL?
I have changed the UDF name ran the cell, and you can see that this worked as well.
So, If you need a UDF, don't do it in Python. Do it in Scala. You need Scala code only for the function definition and registering your UDF. Once registered, you can use it from PySpark and SQL.

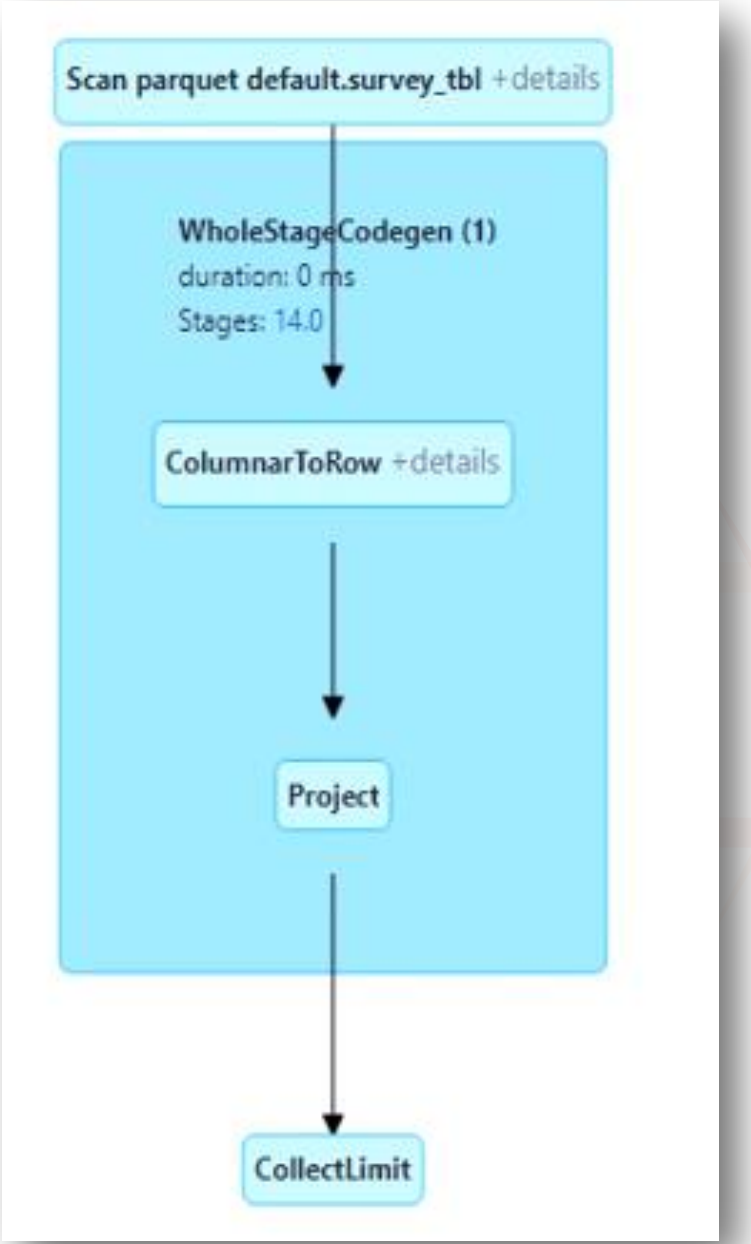


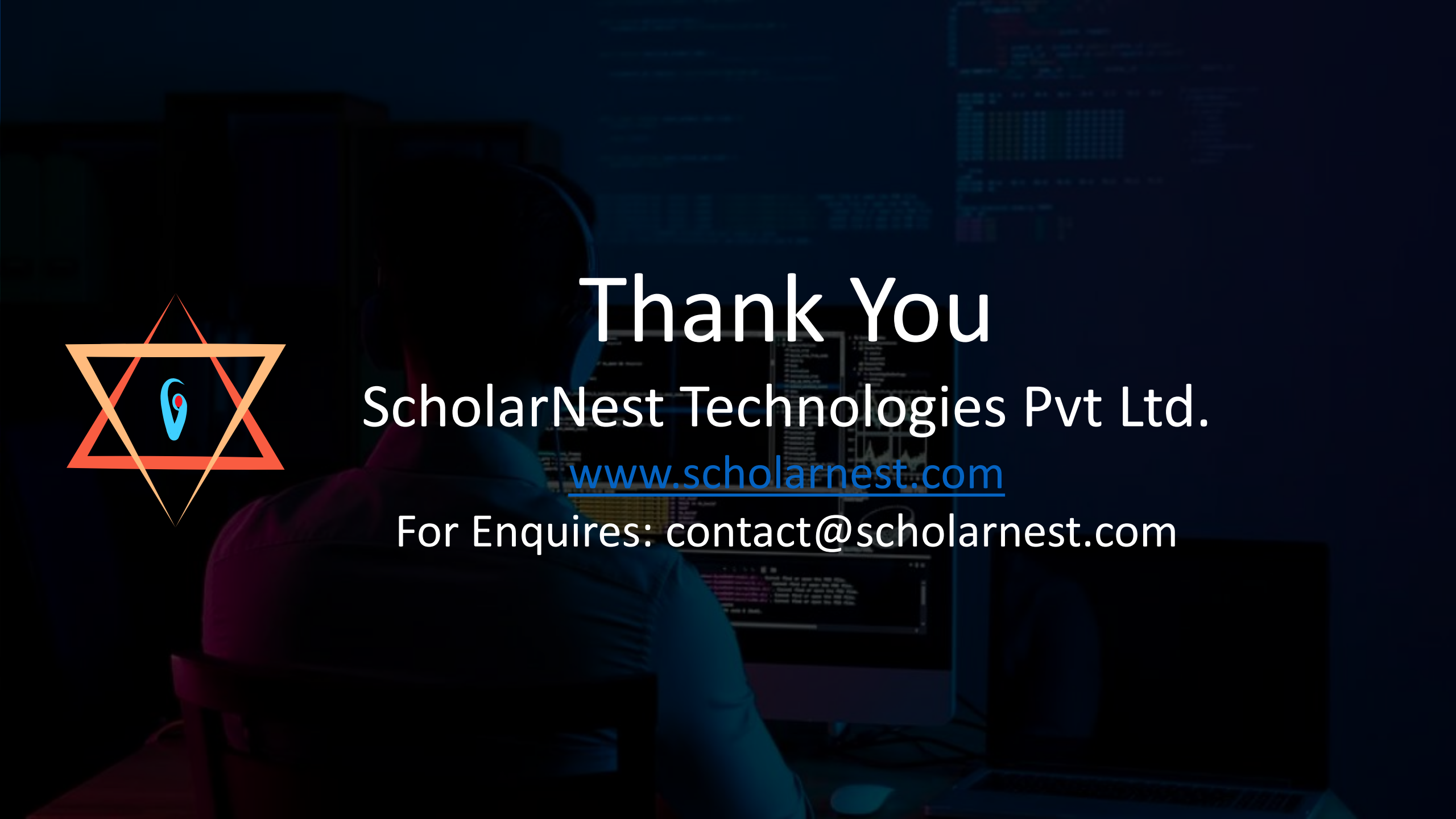
But remember, the UDF is Session specific.
You cannot use this UDF from another Spark session.

You must register the UDF in the session and use it.
If you want to use it in a different session, you must register it again in the new session.

Now, go to Spark UI and check the execution plan for the most recent SQL.

Now you don't see that extra BatchEvalPython step.
That's gone. Isn't it?
And using the Scala UDF, you just learned a technique to improve your performance.





Thank You

ScholarNest Technologies Pvt Ltd.

www.scholarnest.com

For Enquires: contact@scholarnest.com

Spark Azure Databricks

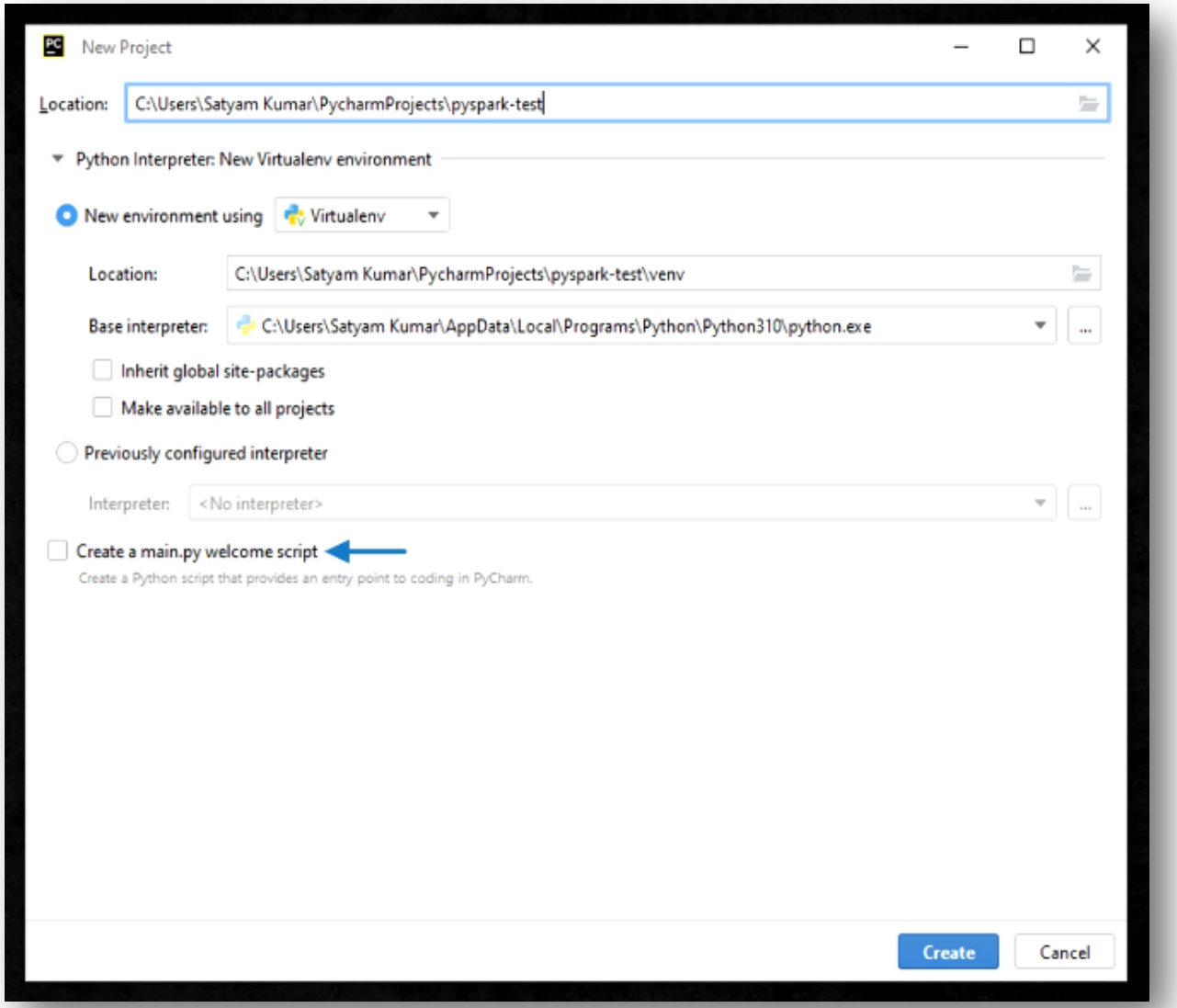
Databricks Spark Certification and Beyond

Module:
Miscellaneous Topics

Lecture:
Unit Testing

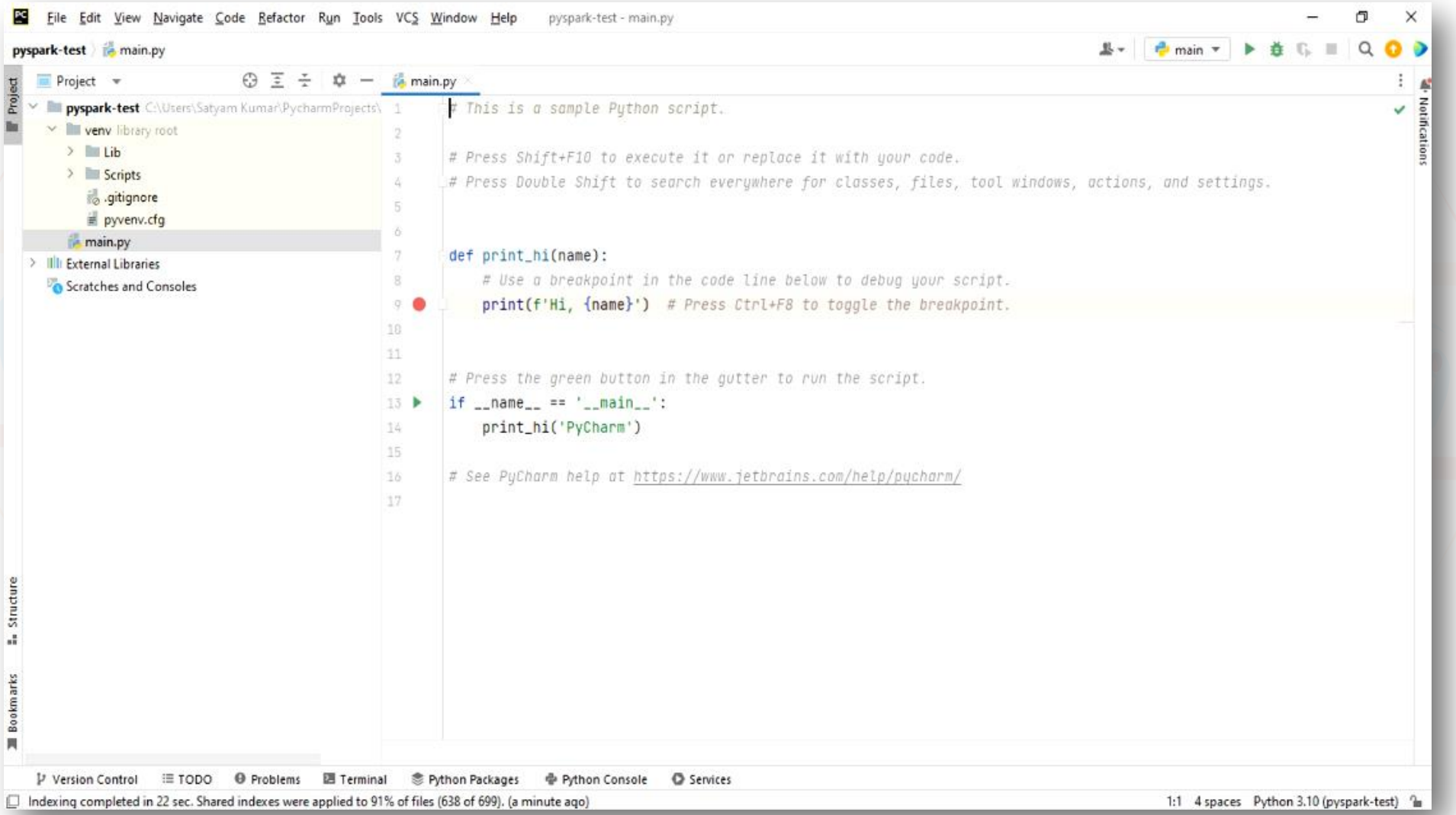
Unit Testing in Spark

Start your PyCharm IDE and create a new Spark project. I will also choose the option to create a main.py welcome script. **(Reference: /pyspark-test/)**

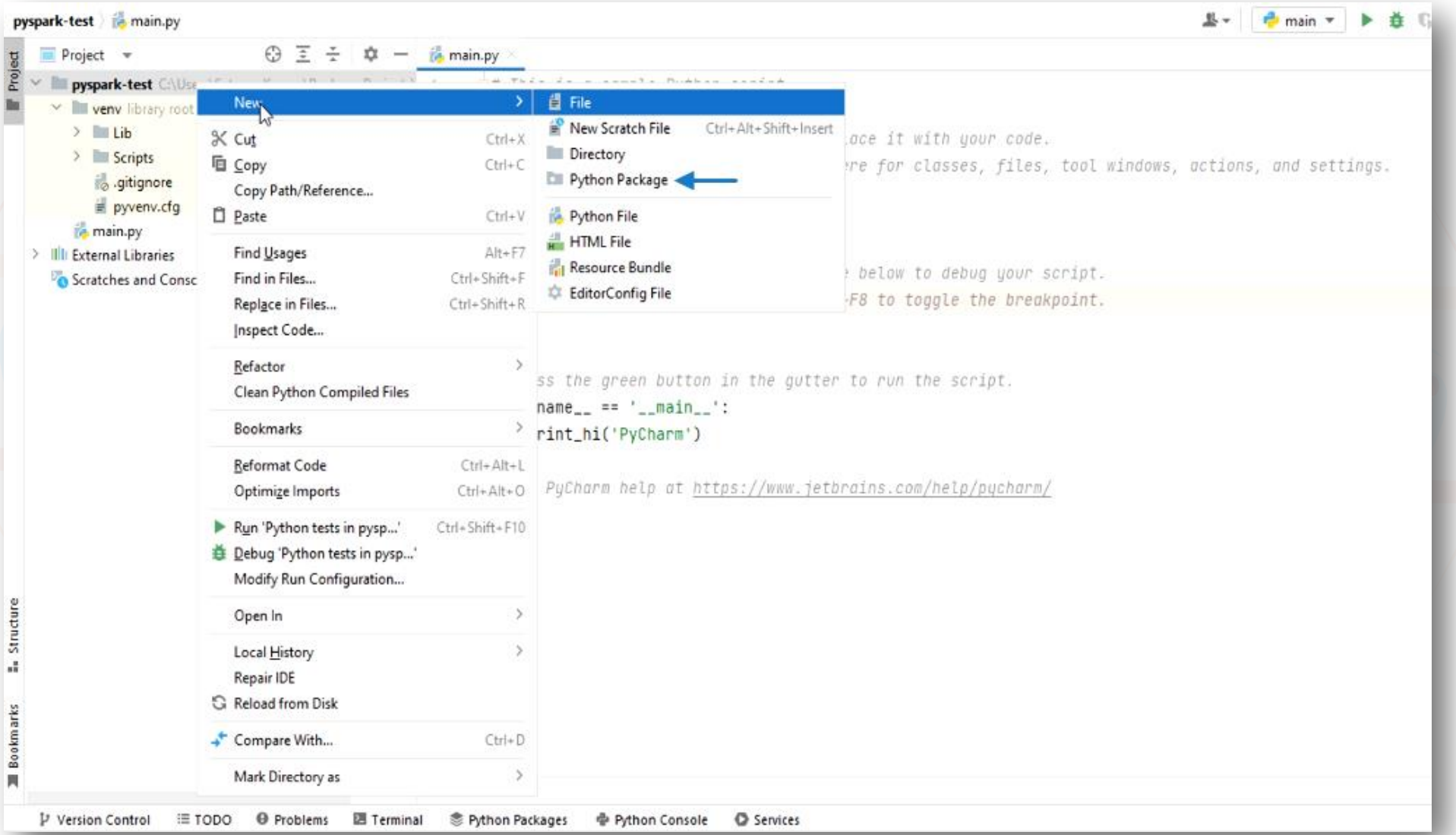


I will create a small Spark project and then implement unit testing in PyCharm.
Why am I using PyCharm to create a Spark project? Why not use a notebook environment? Can we not do unit testing on the notebook environment?
We can do that. But doing unit testing on Databricks notebook is a little tricky.
I will show you that also in the end.
But writing unit test cases for your application makes more sense in the local environment.
So let's learn how to do it in your local environment, and then we will take the same to the notebook environment.

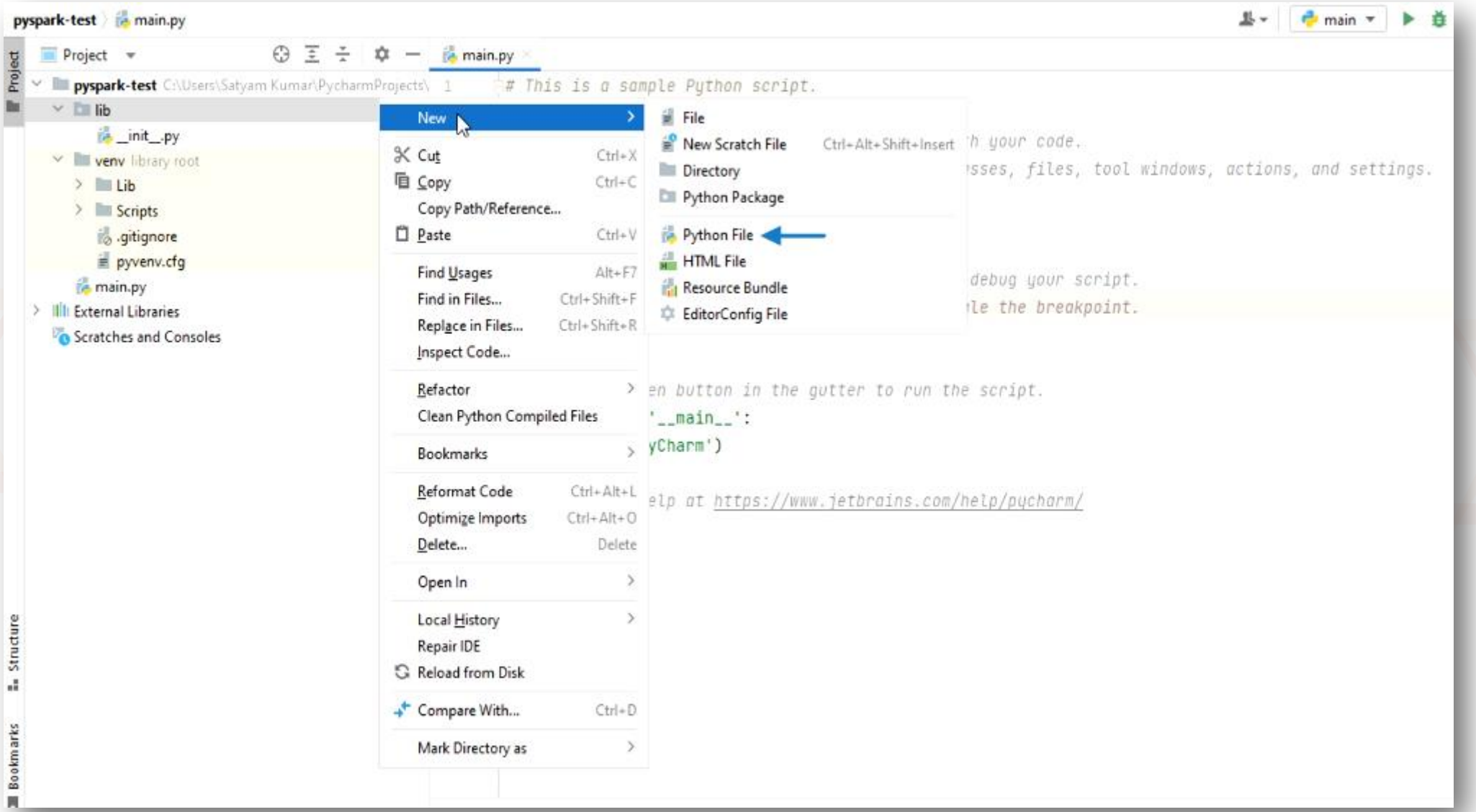
Here we are with a brand new project. Now let's create a Spark application, and then we will implement unit testing.



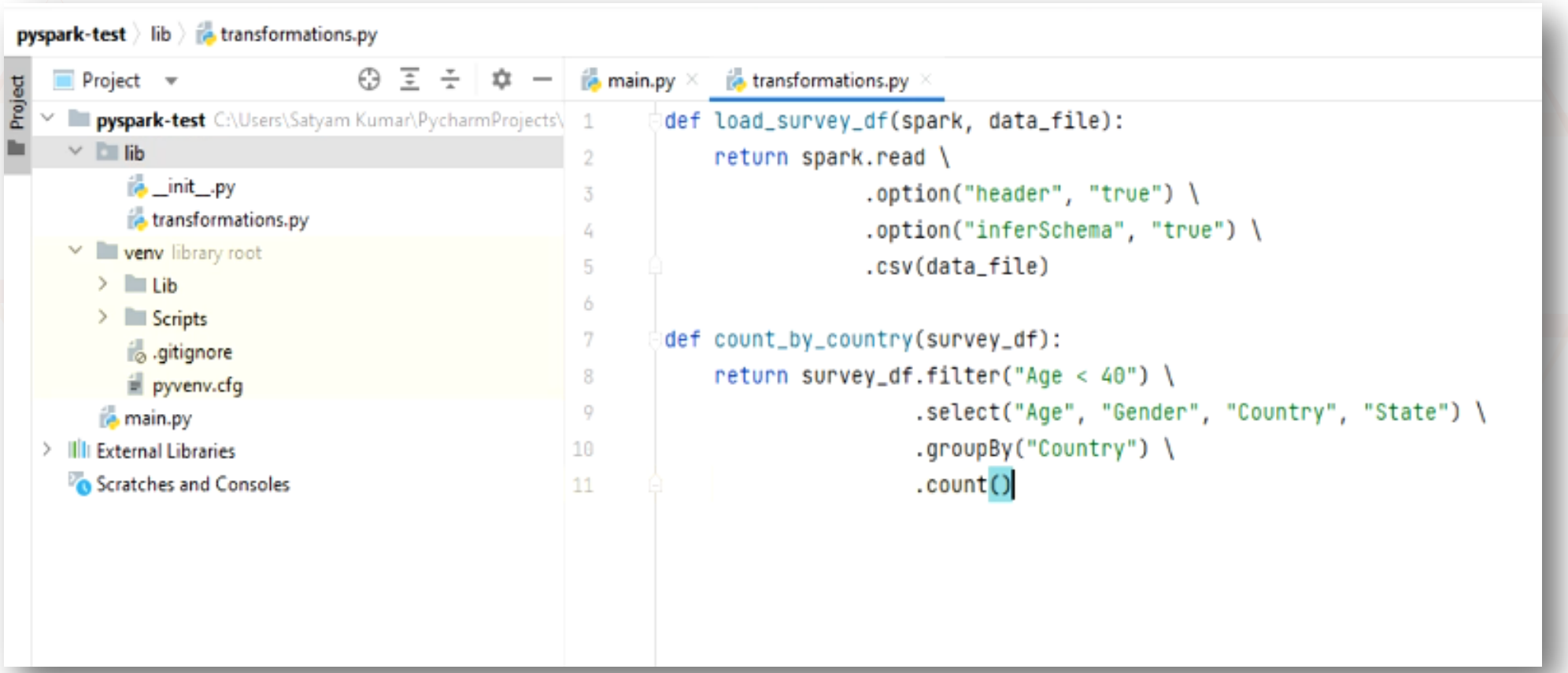
Right-click on your project name, go to new, and create a python package.
Name it lib.



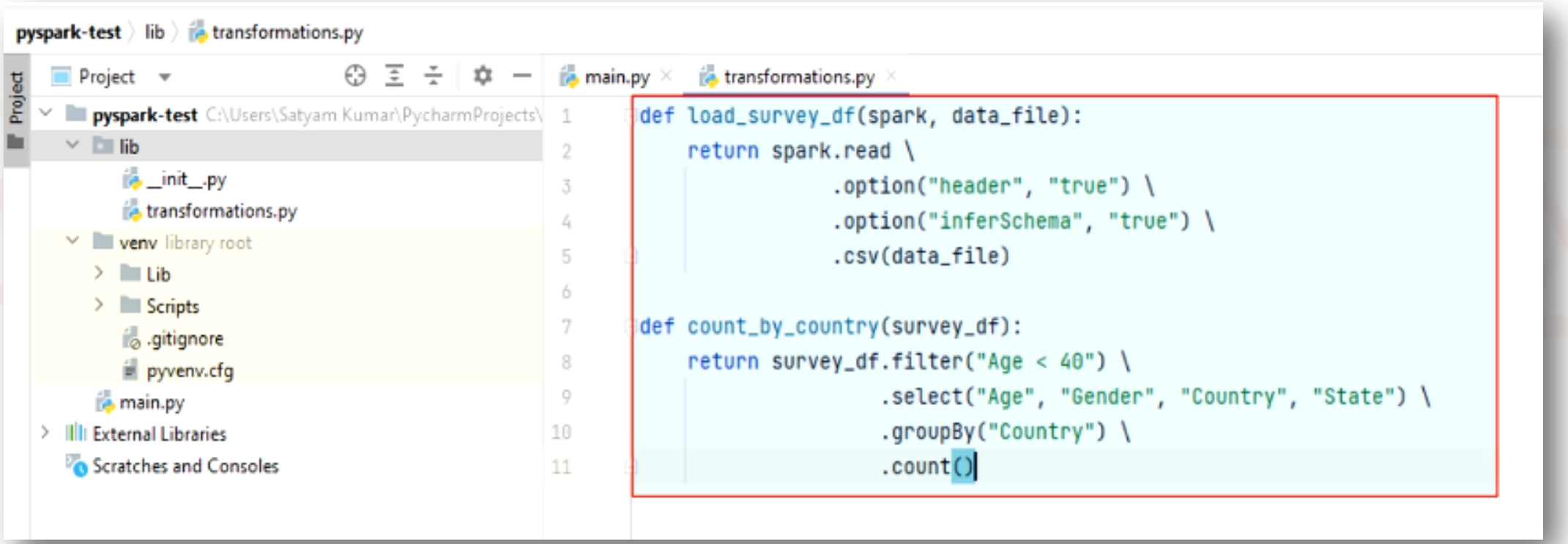
Now, right-click the lib folder, again go to new, and create a python file.



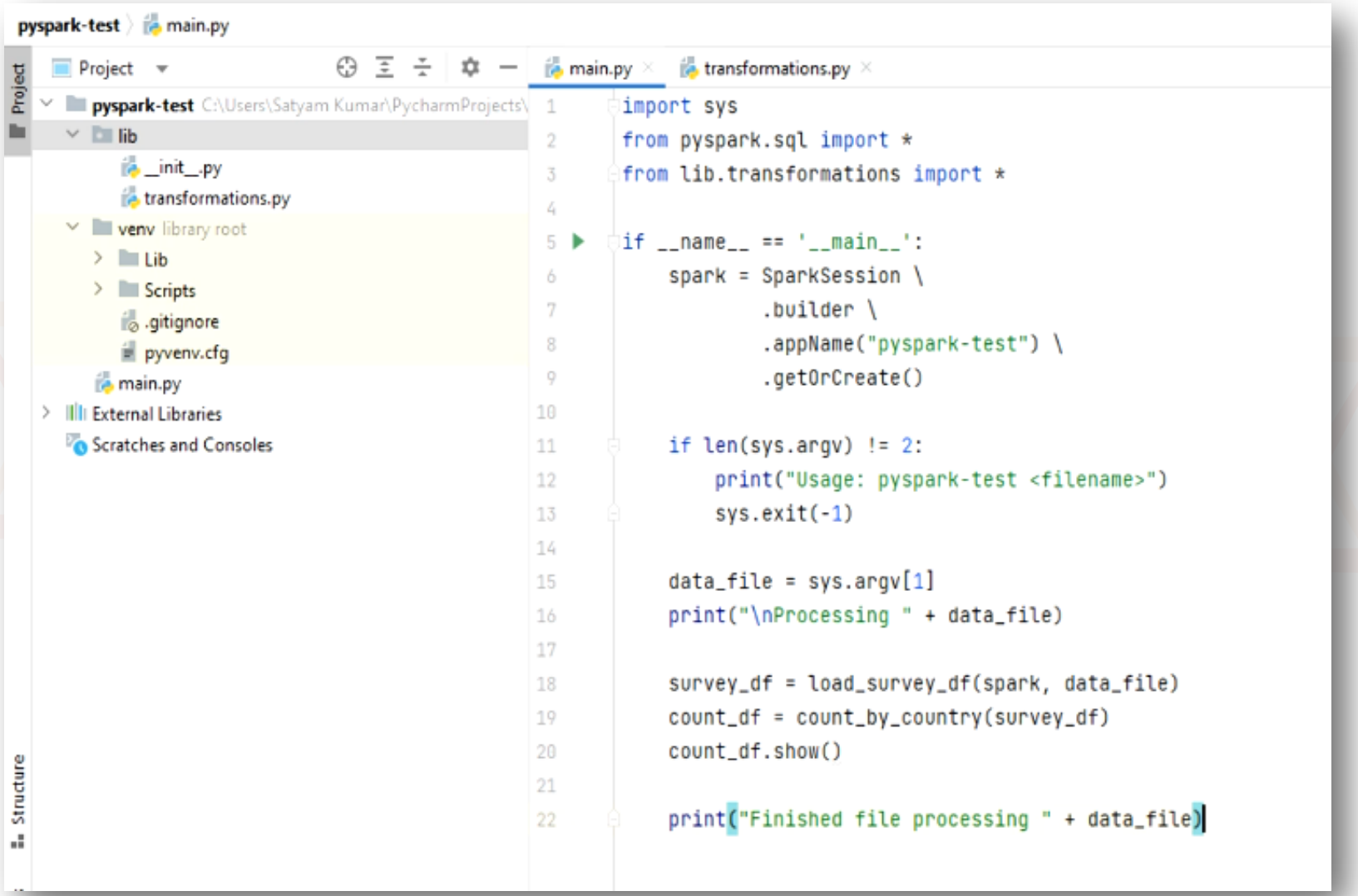
I created a Python library. Then I will create two functions in the library.
So go to your transformations file and create two functions as shown below.
The first function will read a data file, create a Dataframe and return it. The second function takes a Dataframe as input. Then it will apply some transformations to the Dataframe and return the result.



I am developing my application using a modular structure. I have this transformations library which defines two business functions. My main application will use these functions to implement the requirement. So let me go to the main.py file.



Here is my main.py file. I deleted everything and re-created the application main.



I am importing those two functions I created in the lib/transformations.

main.py transformations.py

```
1  import sys
2  from pyspark.sql import *
3  from lib.transformations import *
4
5  if __name__ == '__main__':
6      spark = SparkSession \
7              .builder \
8              .appName("pyspark-test") \
9              .getOrCreate()
10
11     if len(sys.argv) != 2:
12         print("Usage: pyspark-test <filename>")
13         sys.exit(-1)
14
15     data_file = sys.argv[1]
16     print("\nProcessing " + data_file)
17
18     survey_df = load_survey_df(spark, data_file)
19     count_df = count_by_country(survey_df)
20     count_df.show()
21
22     print("Finished file processing " + data_file)
```


Then I create a Spark session.

main.py transformations.py

```
import sys
from pyspark.sql import *
from lib.transformations import *

if __name__ == '__main__':
    spark = SparkSession \
            .builder \
            .appName("pyspark-test") \
            .getOrCreate()

    if len(sys.argv) != 2:
        print("Usage: pyspark-test <filename>")
        sys.exit(-1)

    data_file = sys.argv[1]
    print("\nProcessing " + data_file)

    survey_df = load_survey_df(spark, data_file)
    count_df = count_by_country(survey_df)
    count_df.show()

    print("Finished file processing " + data_file)
```


I am taking the data file name from the command line argument.

main.py transformations.py

```
import sys
from pyspark.sql import *
from lib.transformations import *

if __name__ == '__main__':
    spark = SparkSession \
            .builder \
            .appName("pyspark-test") \
            .getOrCreate()

    if len(sys.argv) != 2:
        print("Usage: pyspark-test <filename>")
        sys.exit(-1)

    data_file = sys.argv[1]
    print("\nProcessing " + data_file)

    survey_df = load_survey_df(spark, data_file)
    count_df = count_by_country(survey_df)
    count_df.show()

    print("Finished file processing " + data_file)
```


Once I have the data file name, I call the load_survey_df() function to load the data file and create a Dataframe. Then I call count_by_country to transform the data and get the results. Finally, I am showing the results.

main.py transformations.py

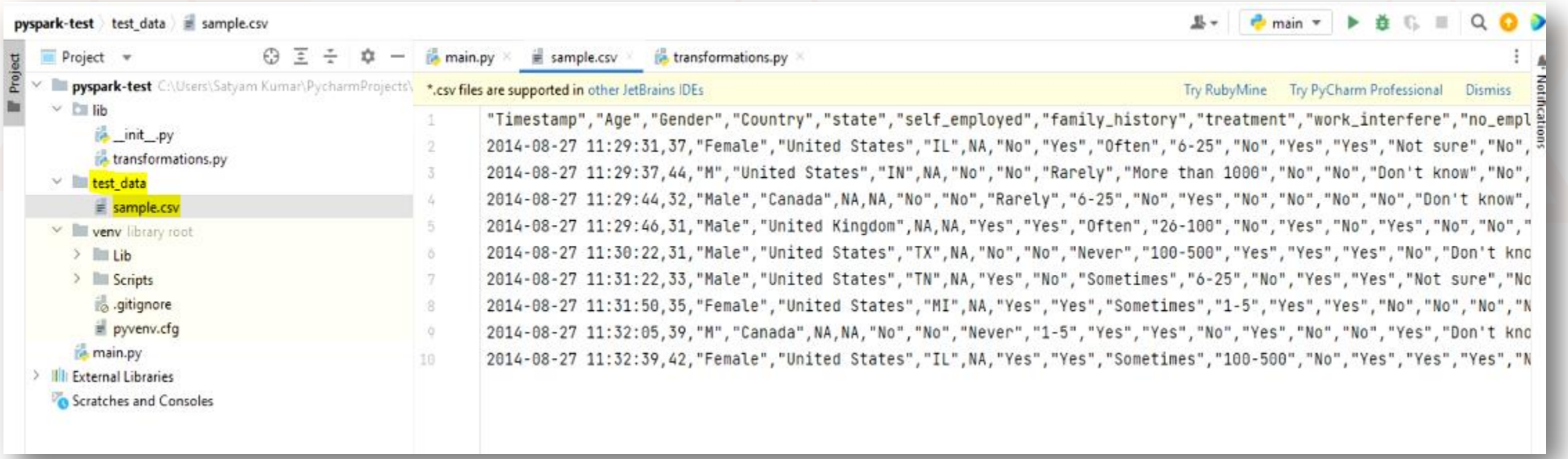
```
1  import sys
2  from pyspark.sql import *
3  from lib.transformations import *
4
5  if __name__ == '__main__':
6      spark = SparkSession \
7              .builder \
8              .appName("pyspark-test") \
9              .getOrCreate()
10
11     if len(sys.argv) != 2:
12         print("Usage: pyspark-test <filename>")
13         sys.exit(-1)
14
15     data_file = sys.argv[1]
16     print("\nProcessing " + data_file)
17
18     survey_df = load_survey_df(spark, data_file)
19     count_df = count_by_country(survey_df)
20     count_df.show()
21
22     print("Finished file processing " + data_file)
```


I could have written all the code in the main method itself. But I decided to take a modular approach and implement everything as functions.
Then call those functions from the main.
And this is how we develop real applications.
We implement functions and then use them.
This approach also allows us to do unit testing.
So in this example, I have two functions:
load_survey_df and *count_by_country*.
These two guys are my two units.
So I can test these two guys and implement unit testing.

main.py transformations.py

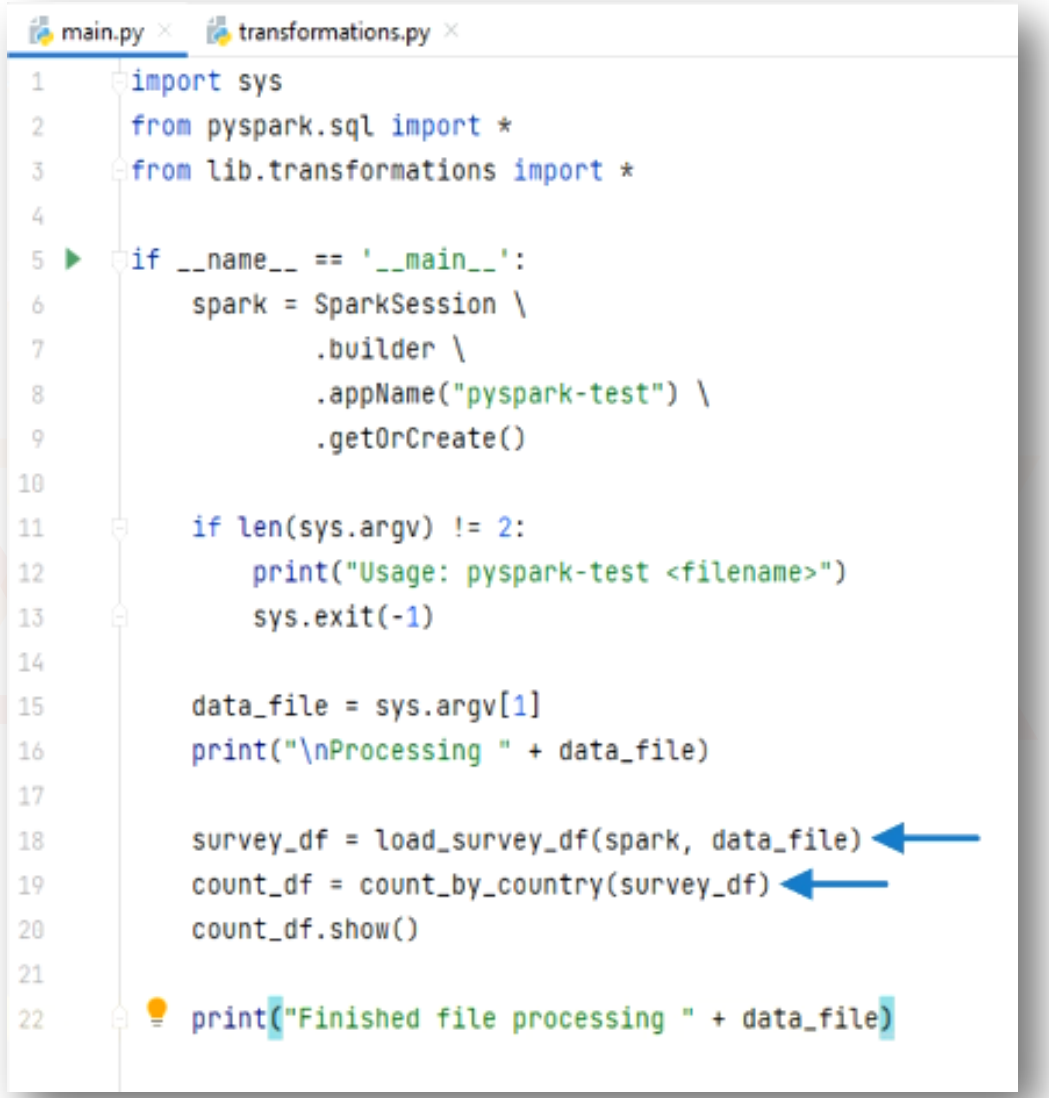
```
import sys
from pyspark.sql import *
from lib.transformations import *

if __name__ == '__main__':
    spark = SparkSession \
            .builder \
            .appName("pyspark-test") \
            .getOrCreate()

    if len(sys.argv) != 2:
        print("Usage: pyspark-test <filename>")
        sys.exit(-1)

    data_file = sys.argv[1]
    print("\nProcessing " + data_file)

    survey_df = load_survey_df(spark, data_file)
    count_df = count_by_country(survey_df)
    count_df.show()

    print("Finished file processing " + data_file)
```


For unit testing, I need sample data. Testing is not done on large volumes of data.
We always perform unit testing on test data. So, I have added some test data to our project.
(Reference: /pyspark-test/test_data/sample.csv)
I have created a directory and inserted a sample.csv file into this directory.
You can see the file contents.
We have ten lines in the file; the first line is a header row.

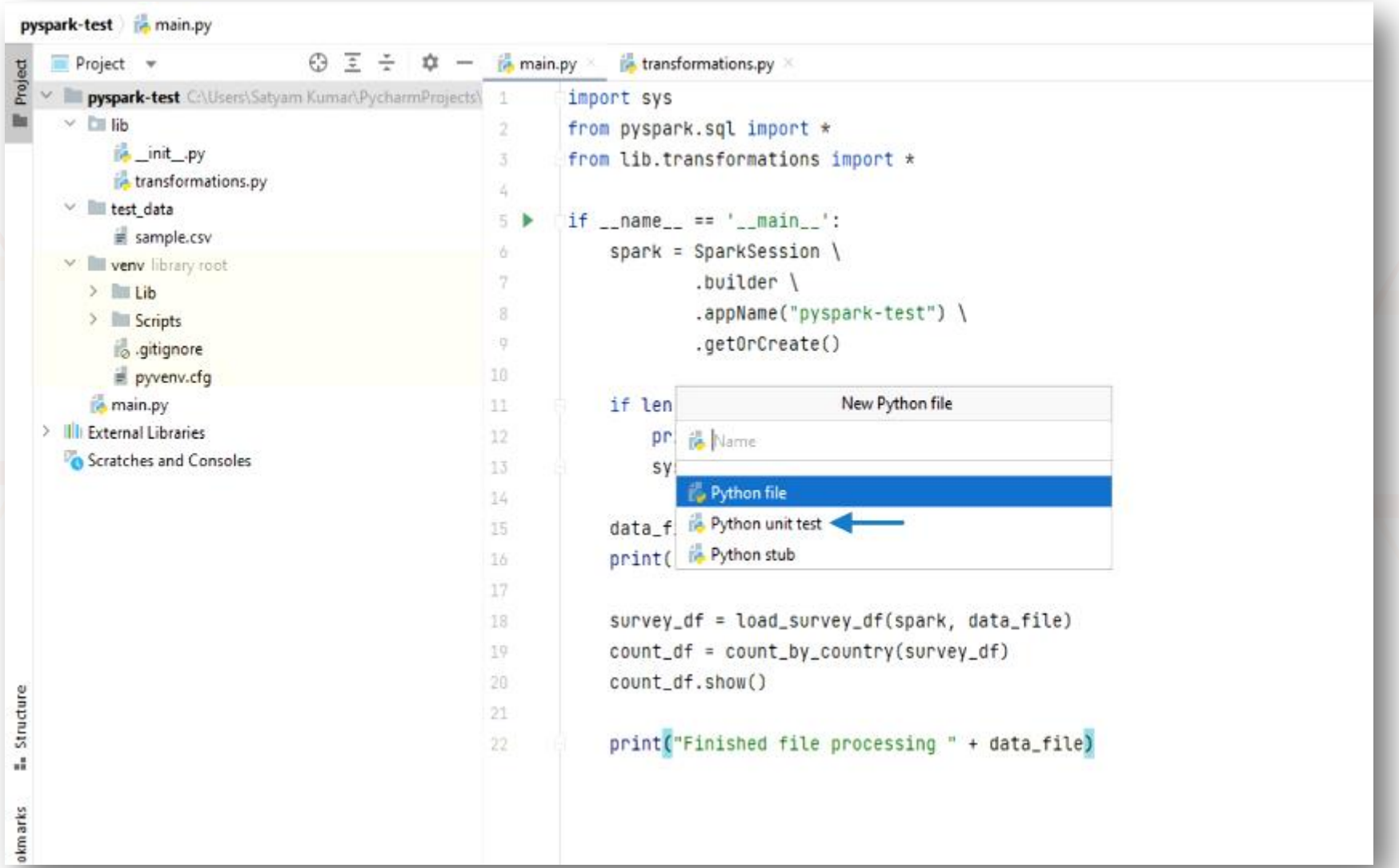


So we have a Spark application.
I can package it and deploy it on the cluster.
When I want to run it on large volumes of data, I will pass the data file location from the command line.
The application will read the data file and run on large volumes of data.

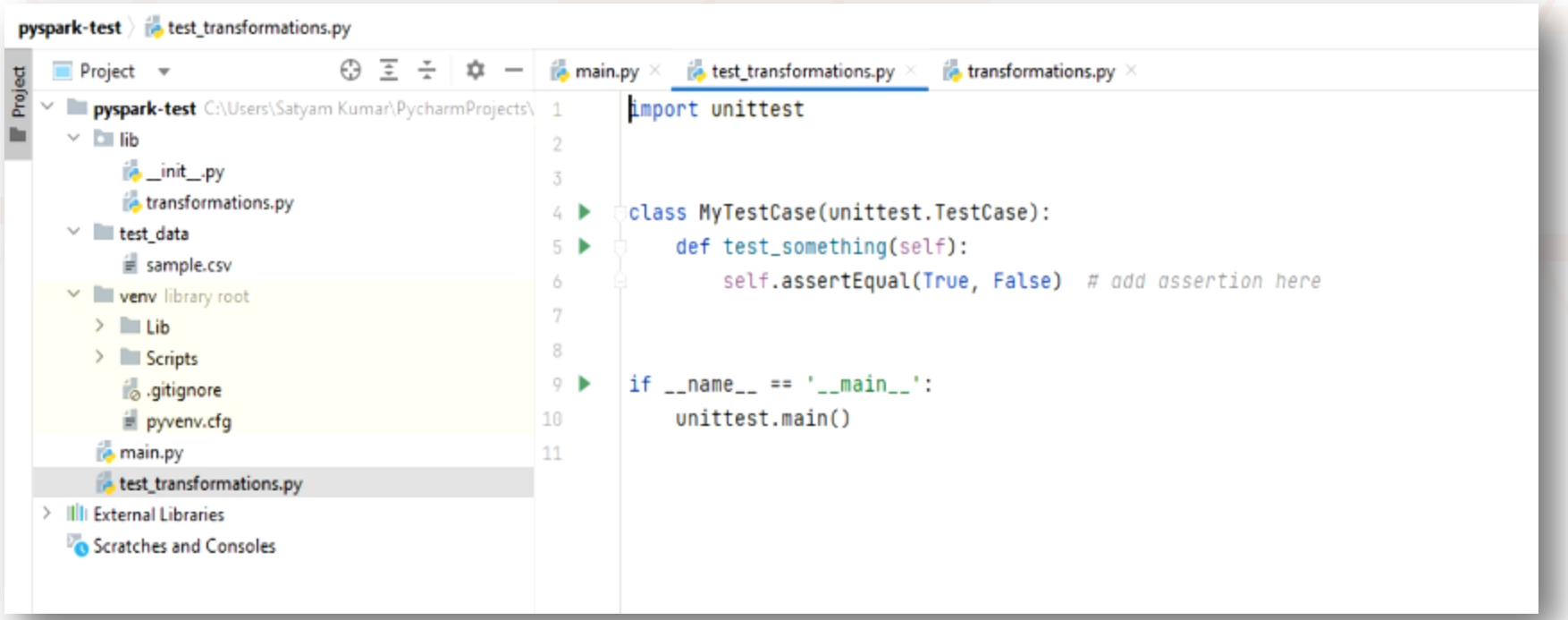
But what about unit testing?
I have two functions in this application.
So as a best practice, I must implement unit test cases for those two functions.



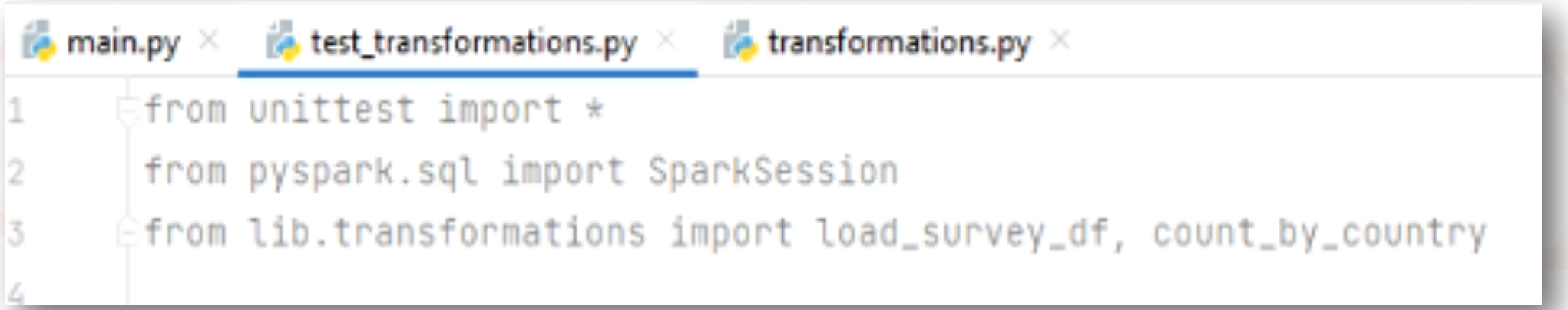
Again, go to your project and create a python file. But this time, we will choose the Python unit test.



So we have a unit test starter file. By default, Python will use the unit test framework.
There are many other unit testing frameworks. For example, PyTest is another unit testing framework. You can choose a suitable unit testing framework as per your project mandate.
However, the default unittest framework is good enough, and we will use the same.
We already have some starter codes here.
But let me delete everything and create everything from scratch.



So I imported everything from the unittest package.
I am also importing SparkSession because we need it.
Then I import the two functions that I want to test.



Then I will create a test class.
I assume you already know Python and how to create unit tests in Python.
The next step is to create a Spark session.
And the best place to create the Spark session is the setUpClass() method.

main.py | test_transformations.py | transformations.py

```
1  from unittest import *
2  from pyspark.sql import SparkSession
3  from lib.transformations import load_survey_df, count_by_country
4
5  class TransformationTestCase(TestCase):
6      spark = None
```


Here I created a spark session so we can run some spark code.
What is next? I want to test two functions.

main.py | test_transformations.py | transformations.py

```
1   from unittest import *
2   from pyspark.sql import SparkSession
3   from lib.transformations import load_survey_df, count_by_country
4
5   class TransformationTestCase(TestCase):
6       spark = None
7
8       @classmethod
9       def setUpClass(cls) -> None:
10          cls.spark = SparkSession.builder \
11                      .appName("test-case-pyspark") \
12                      .getOrCreate()
```


So I created test_datafile_loading() method. This method will call the load_survey_df() function and create the sample_df. Then I want to validate if the load_survey_df is working as expected. How Do I validate? One approach is to count the number of rows in the sample_df and check if I have 9 records in the result. Why nine? Because I know that the test data has nine records. And that's what I am doing here. Count the records, and assert if we have nine records. My first test case is done.

main.py | test_transformations.py | transformations.py

```
1  from unittest import *
2  from pyspark.sql import SparkSession
3  from lib.transformations import load_survey_df, count_by_country
4
5  class TransformationTestCase(TestCase):
6      spark = None
7
8      @classmethod
9      def setUpClass(cls) -> None:
10         cls.spark = SparkSession.builder \
11                     .appName("test-case-pyspark") \
12                     .getOrCreate()
13
14     def test_datafile_loading(self):
15         sample_df = load_survey_df(self.spark, "test_data/sample.csv")
16         result_count = sample_df.count()
17         self.assertEqual(result_count, 9, "Record count should be 9")
```


Here is another test case for the second function.
I will call the load_survey_df() and create the sample_df.

```
def test_country_count(self):
    sample_df = load_survey_df(self.spark, "test_data/sample.csv")
    count_list = count_by_country(sample_df).collect()

    count_dict = dict()
    for row in count_list:
        count_dict[row["Country"]] = row["count"]

    self.assertEqual(count_dict["United States"], 4, "Count for United States should be 4")
    self.assertEqual(count_dict["Canada"], 2, "Count for Canada should be 2")
    self.assertEqual(count_dict["United Kingdom"], 1, "Count for United Kingdom should be 1")
```


Then I will call the count_by_country() and collect the records in the count_list. Why? Because I want to test if the count_by_country() is working as expected. So I will call the count_by_country() and collect the result.

```
def test_country_count(self):
    sample_df = load_survey_df(self.spark, "test_data/sample.csv")
    count_list = count_by_country(sample_df).collect()

    count_dict = dict()
    for row in count_list:
        count_dict[row["Country"]] = row["count"]

    self.assertEqual(count_dict["United States"], 4, "Count for United States should be 4")
    self.assertEqual(count_dict["Canada"], 2, "Count for Canada should be 2")
    self.assertEqual(count_dict["United Kingdom"], 1, "Count for United Kingdom should be 1")
```

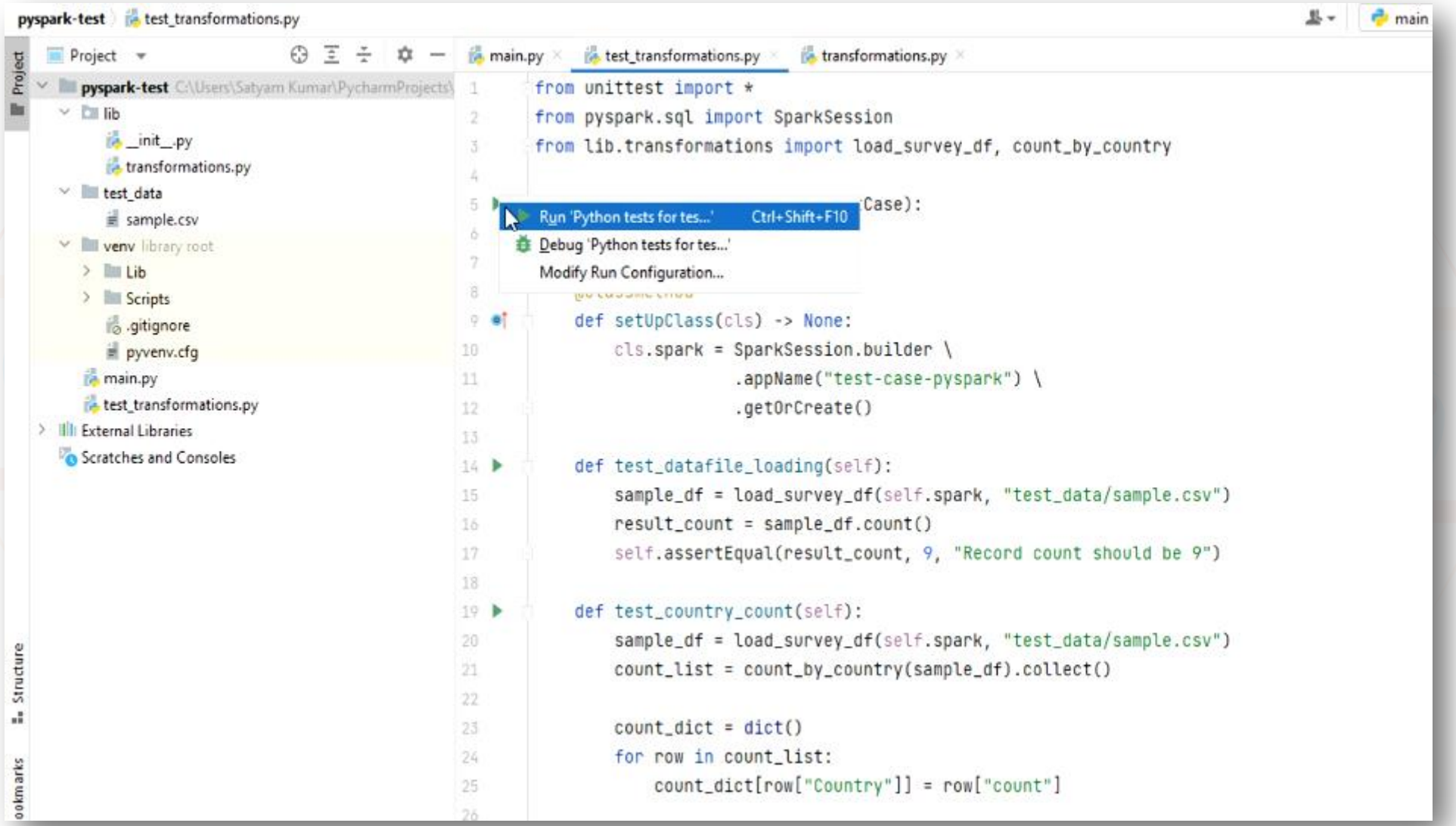

Then I will convert the count_list into the count dictionary.
Converting a list into a dictionary is a common practice for unit testing.
Why? Because a dictionary is a key/value pair.
I can easily refer to a key and get the value for validation.
And that's what I am doing in the following three lines.

```
def test_country_count(self):
    sample_df = load_survey_df(self.spark, "test_data/sample.csv")
    count_list = count_by_country(sample_df).collect()

    count_dict = dict()
    for row in count_list:
        count_dict[row["Country"]] = row["count"]

    self.assertEqual(count_dict["United States"], 4, "Count for United States should be 4")
    self.assertEqual(count_dict["Canada"], 2, "Count for Canada should be 2")
    self.assertEqual(count_dict["United Kingdom"], 1, "Count for United Kingdom should be 1")
```

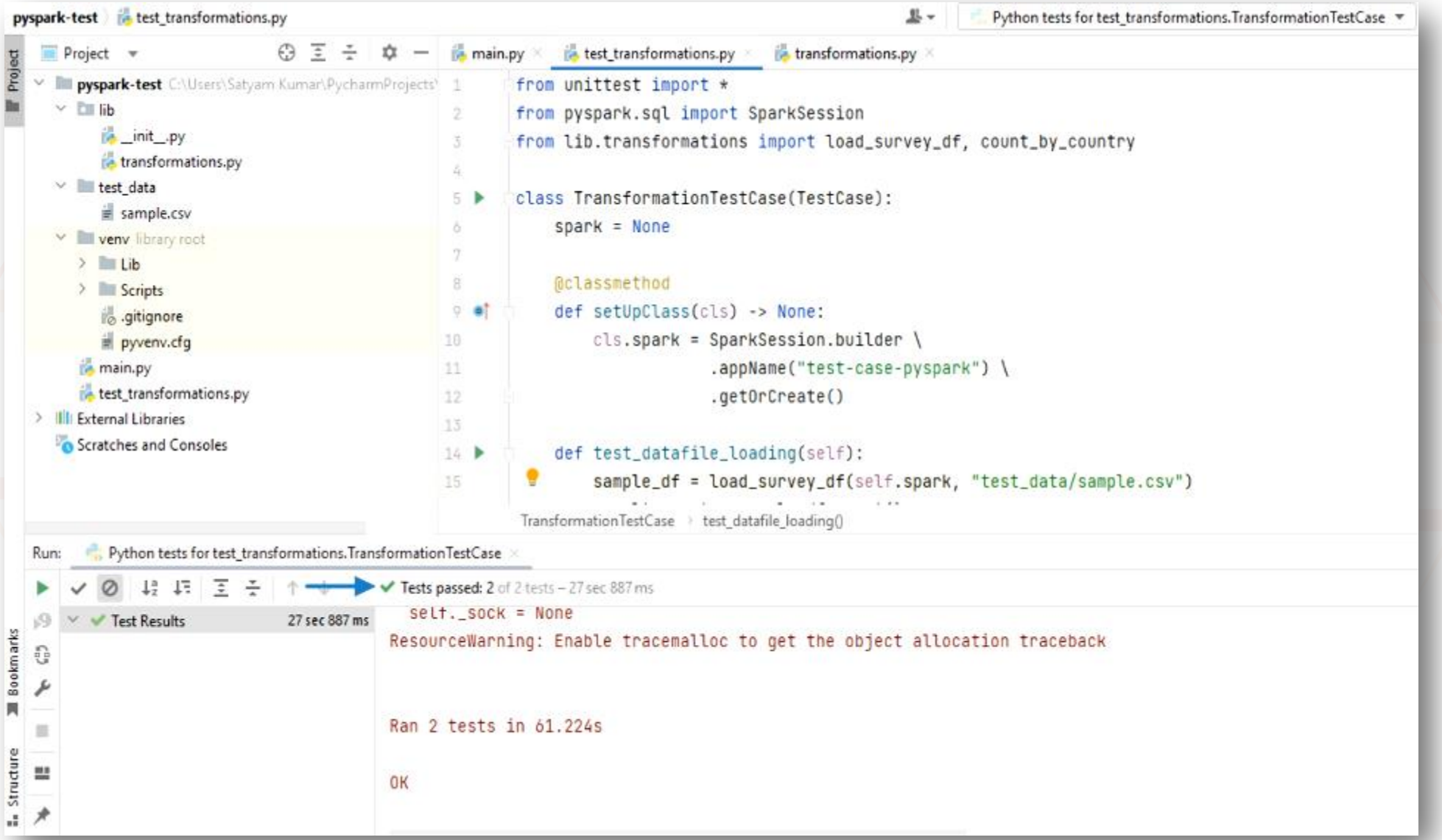

I know that if the count_by_country() worked as expected, it should give 4 for the United States, Canada should be 2, and the United Kingdom count is 1.
And that's what I am asserting in the last three lines.

```
def test_country_count(self):
    sample_df = load_survey_df(self.spark, "test_data/sample.csv")
    count_list = count_by_country(sample_df).collect()

    count_dict = dict()
    for row in count_list:
        count_dict[row["Country"]] = row["count"]

    self.assertEqual(count_dict["United States"], 4, "Count for United States should be 4")
    self.assertEqual(count_dict["Canada"], 2, "Count for Canada should be 2")
    self.assertEqual(count_dict["United Kingdom"], 1, "Count for United Kingdom should be 1")
```

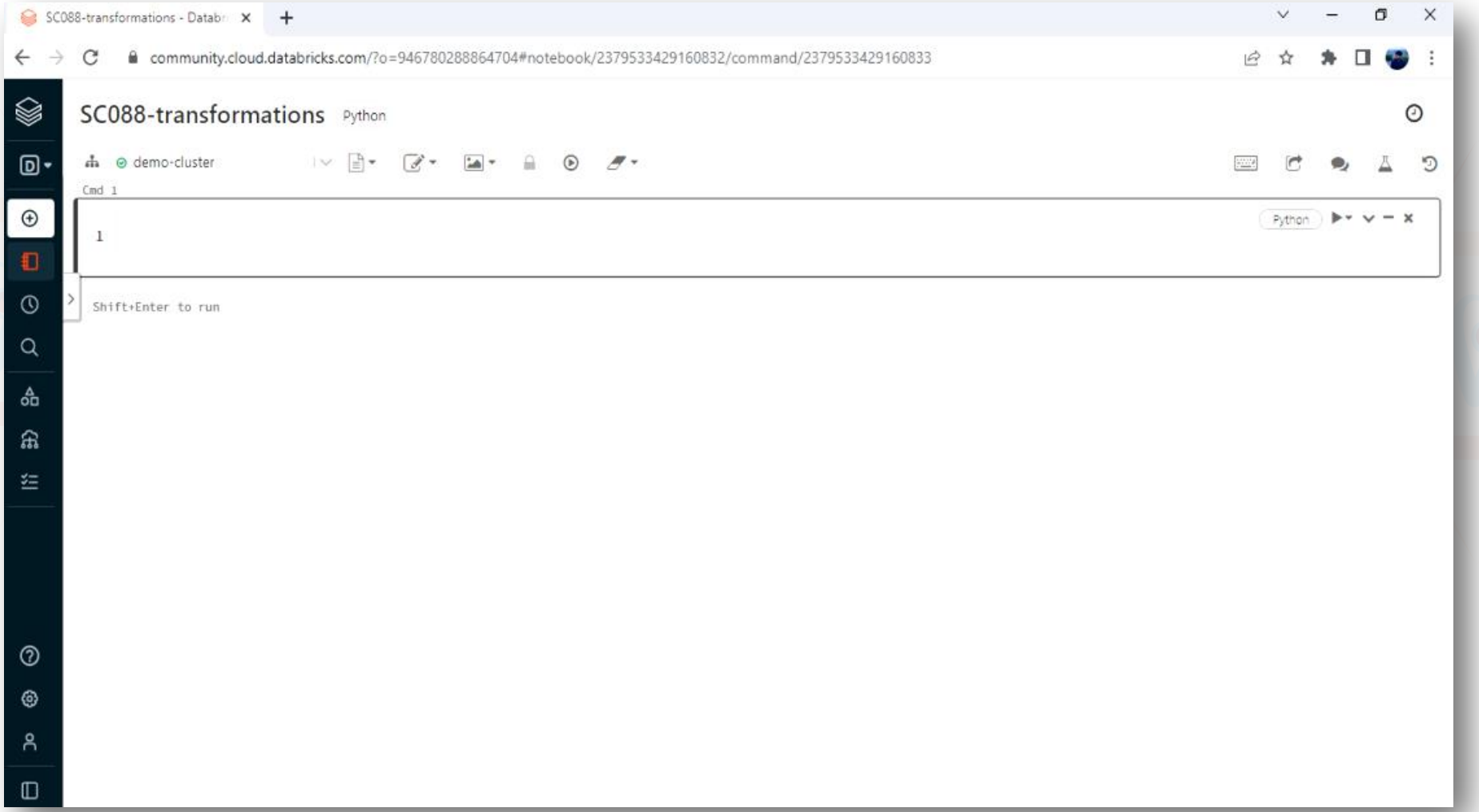

Now, in order to test this go to the run button at the top and run your test cases.



So, it worked. You can see the message: Tests passed 2 of 2 tests.

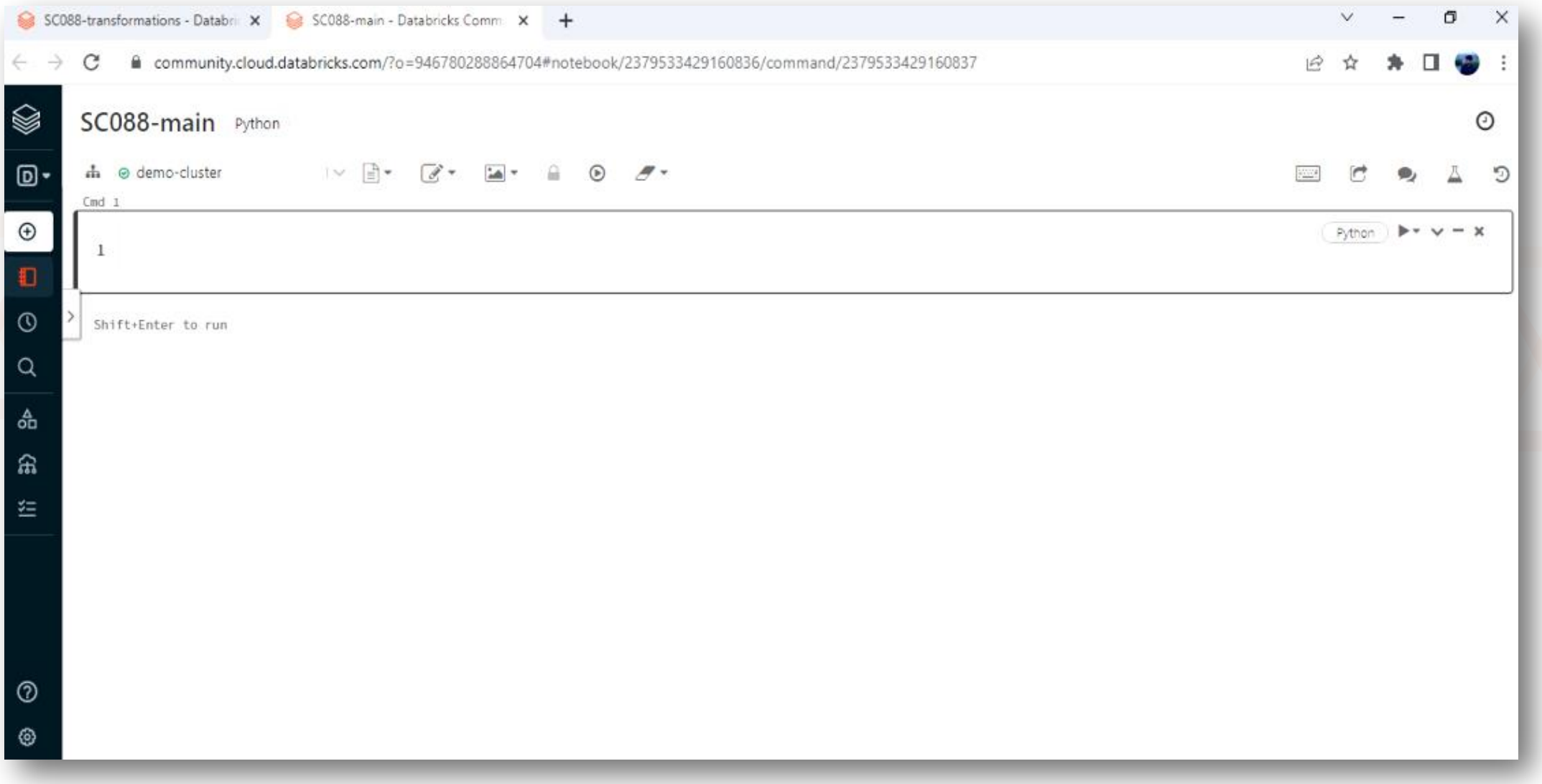


Now let's see how you can do a similar thing on the notebook environment.
Go to your data bricks workspace and create a new notebook.
(Reference: SC088-transformations)



Then I will paste two of my functions into this notebook as shown below. I created a library in the earlier setup. And this notebook represents my library.

Now create a new notebook. **(Reference: SC088-main)**
This one is my main.



So I will import my functions from the transformations notebook.
The %run is the same as the import.
When we want to import our functions from the notebook, we will be using %run.
This command will not run the notebook. It will simply import all the functions from the given notebook.
Next, I can implement the main logic.

Here is you main logic and it’s corresponding output.
But what about the unit testing?

Cmd 2

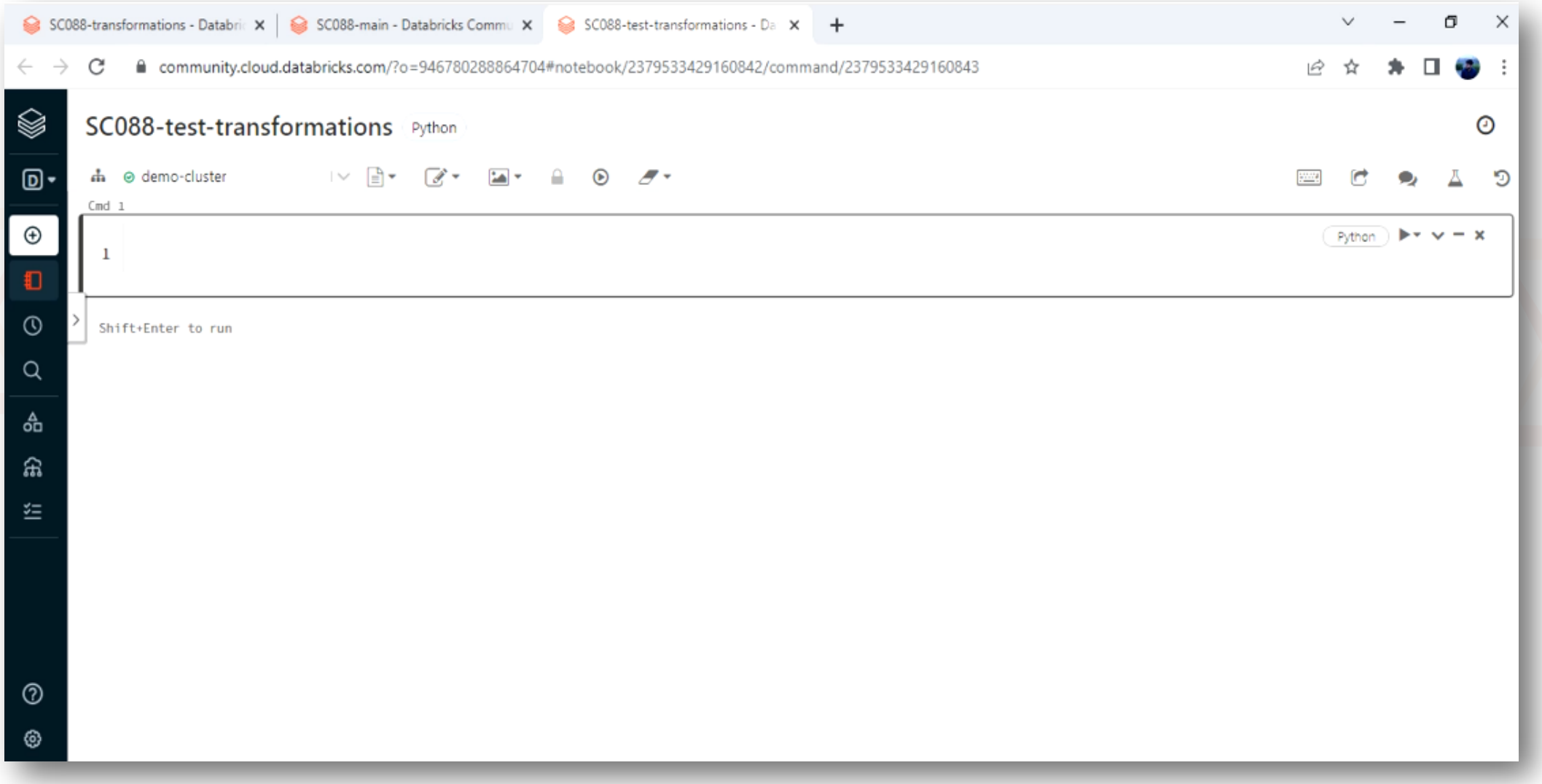
```
1  survey_df = load_survey_df(spark, "/FileStore/sample.csv")
2  count_df = count_by_country(survey_df)
3  count_df.show()
```

▸ (4) Spark Jobs

```
+--------------+-----+
|       Country|count|
+--------------+-----+
| United States|    4|
|        Canada|    2|
|United Kingdom|    1|
+--------------+-----+
```

Command took 12.77 seconds -- by prashant@scholarnest.com at 8/10/2022, 11:28:05 PM on demo-cluster

Let's create a new notebook and implement test cases in this separate notebook.
(Reference: SC088-test_transformations)



The first step is to import the functions from another notebook.
But I have some other imports as well that I have done in the second cell.
I need the unittest package, and I also need Spark Session.
So I am importing both.

Cmd 1

```
1  %run ./SC088-transformations
```

Command took 0.42 seconds -- by prashant@scholarnest.com at 8/10/2022, 11:29:48 PM on demo-cluster

Cmd 2

```
1  from unittest import *
2  from pyspark.sql import SparkSession
```

Command took 0.09 seconds -- by prashant@scholarnest.com at 8/10/2022, 11:30:29 PM on demo-cluster

Then, we can copy-paste our test class from PyCharm. I have made one change here.
I changed the test data file location. The local environment can read from the project file.
However, the notebook will read data from the DBFS location. So I copied the sample.csv in a DBFS location and changed the path accordingly. The rest of the code is the same as earlier.

Cmd 3

Python

```
class TransformationTestCase(TestCase):
    spark = None

    @classmethod
    def setUpClass(cls) -> None:
        cls.spark = SparkSession.builder \
                    .appName("test-case-pyspark") \
                    .getOrCreate()

    def test_datafile_loading(self):
        sample_df = load_survey_df(self.spark, "/FileStore/sample.csv")
        result_count = sample_df.count()
        self.assertEqual(result_count, 9, "Record count should be 9")

    def test_country_count(self):
        sample_df = load_survey_df(self.spark, "/FileStore/sample.csv")
        count_list = count_by_country(sample_df).collect()

        count_dict = dict()
        for row in count_list:
            count_dict[row["Country"]] = row["count"]

        self.assertEqual(count_dict["United States"], 4, "Count for United States should be 4")
        self.assertEqual(count_dict["Canada"], 2, "Count for Canada should be 2")
        self.assertEqual(count_dict["United Kingdom"], 1, "Count for United Kingdom should be 1")
```


Now we are ready to perform unit testing. But this is a notebook environment. If you run this cell, it will define the test class. It will not run the test cases. The PyCharm IDE allows us to run the test cases using a run button. But the notebook will not do the same. We have to write some extra code for running the test cases.

SC088-test-transformations Python

demo-cluster

```
3
4       @classmethod
5       def setUpClass(cls) -> None:
6           cls.spark = SparkSession.builder \
7                       .appName("test-case-pyspark") \
8                       .getOrCreate()
9
10      def test_datafile_loading(self):
11          sample_df = load_survey_df(self.spark, "/FileStore/sample.csv")
12          result_count = sample_df.count()
13          self.assertEqual(result_count, 9, "Record count should be 9")
14
15      def test_country_count(self):
16          sample_df = load_survey_df(self.spark, "/FileStore/sample.csv")
17          count_list = count_by_country(sample_df).collect()
18
19          count_dict = dict()
20          for row in count_list:
21              count_dict[row["Country"]] = row["count"]
22
23          self.assertEqual(count_dict["United States"], 4, "Count for United States should be 4")
24          self.assertEqual(count_dict["Canada"], 2, "Count for Canada should be 2")
25          self.assertEqual(count_dict["United Kingdom"], 1, "Count for United Kingdom should be 1")
```

Command took 0.05 seconds -- by prashant@scholarnest.com at 8/10/2022, 11:35:13 PM on demo-cluster

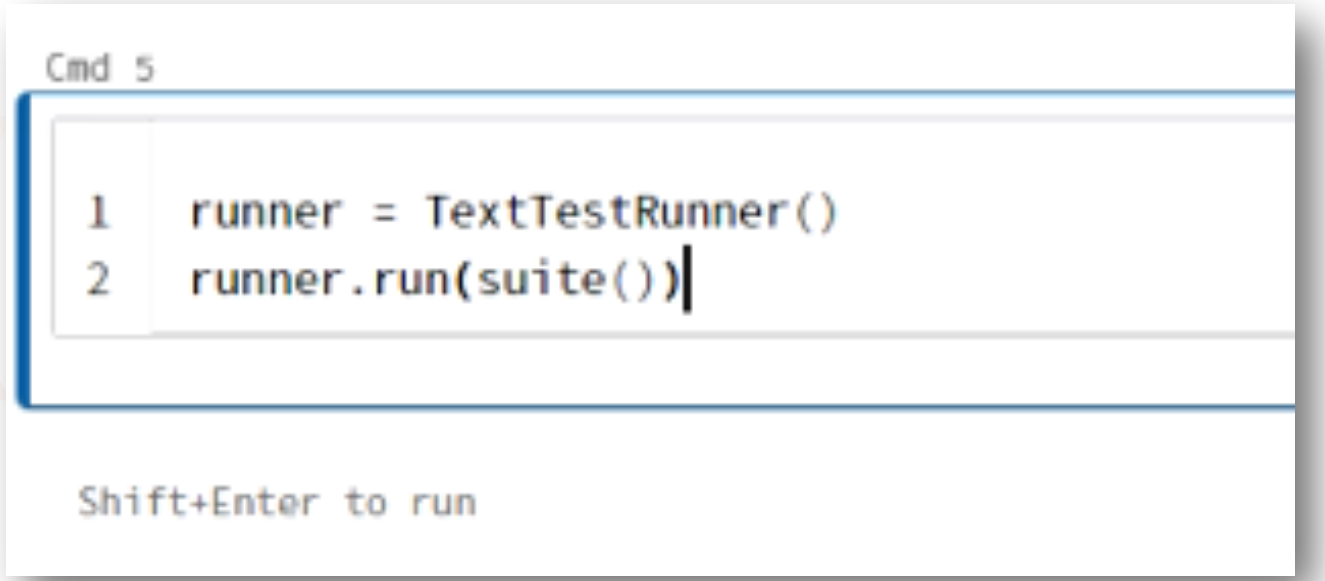
I defined another function here.
This function will create a test suite. The TestSuite is also part of the unit test framework.
The code is simple.
I am creating an empty test suite.
Then I am adding my two test cases to the TestSuite.
These are nothing but the test functions in my test class.

Cmd 4

```
1  def suite():
2      suite = TestSuite()
3      suite.addTest(TransformationTestCase('test_datafile_loading'))
4      suite.addTest(TransformationTestCase('test_country_count'))
5      return suite
```

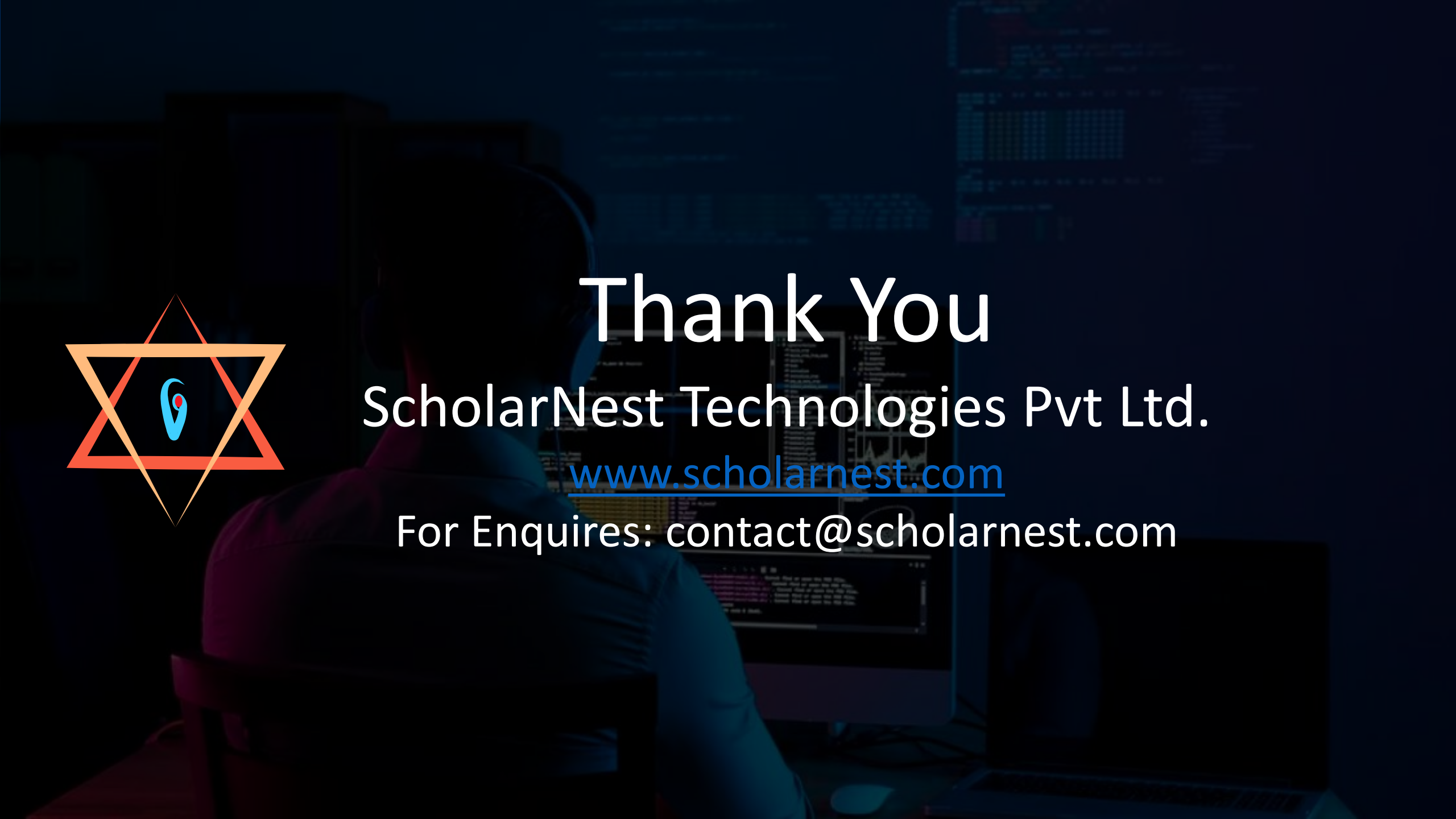
Command took 0.03 seconds -- by prashant@scholarnest.com at 8/10/2022, 11:41:21 PM on demo-cluster

Now we can run our test suit. Here is the code to run it.
I create a TextTestRunner and use it to run the TestSuite.



You can see that it worked, both the test cases have passed.

```
Cmd 5

1  runner = TextTestRunner()
2  runner.run(suite())

▸ (8) Spark Jobs

..
----------------------------------------------------------------------
Ran 2 tests in 6.058s

OK
Out[10]: <unittest.runner.TextTestResult run=2 errors=0 failures=0>

Command took 6.29 seconds -- by prashant@scholarnest.com at 8/10/2022, 11:41:48 PM on demo-cluster
```

Thank You

ScholarNest Technologies Pvt Ltd.

www.scholarnest.com

For Enquires: contact@scholarnest.com