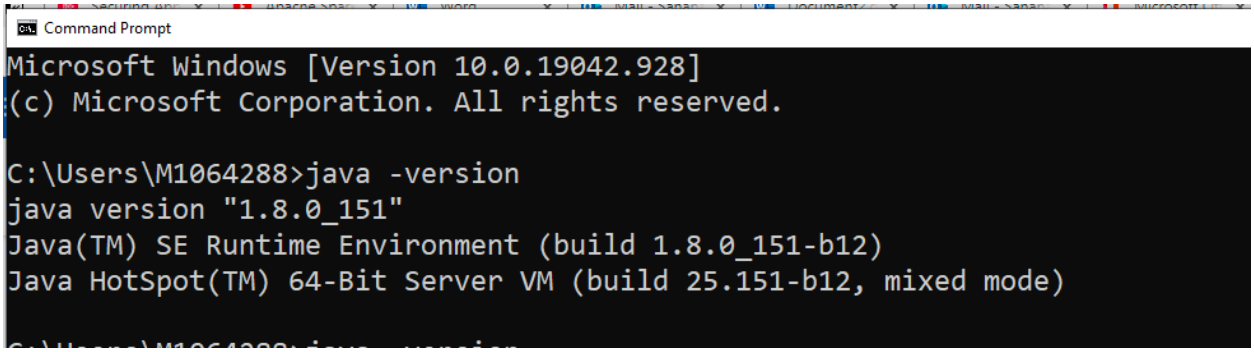
PYSPARK

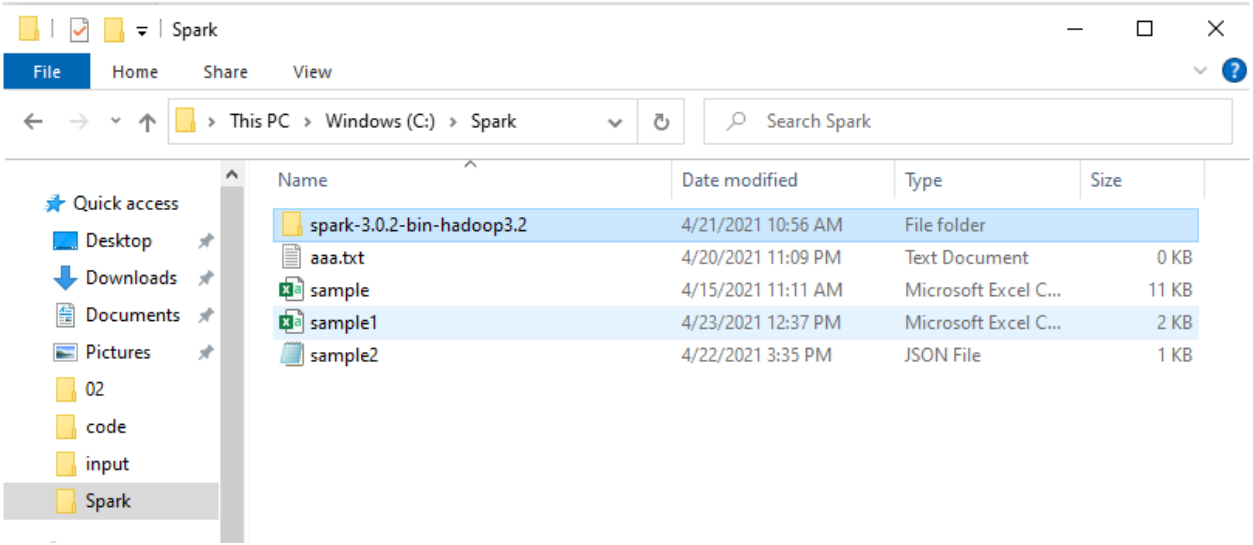
Installation Of Pyspark

Step 1: In order to install pyspark, we need to have java on our system, if not first need to download java. Check if the java is present by giving a command java -version on command prompt.

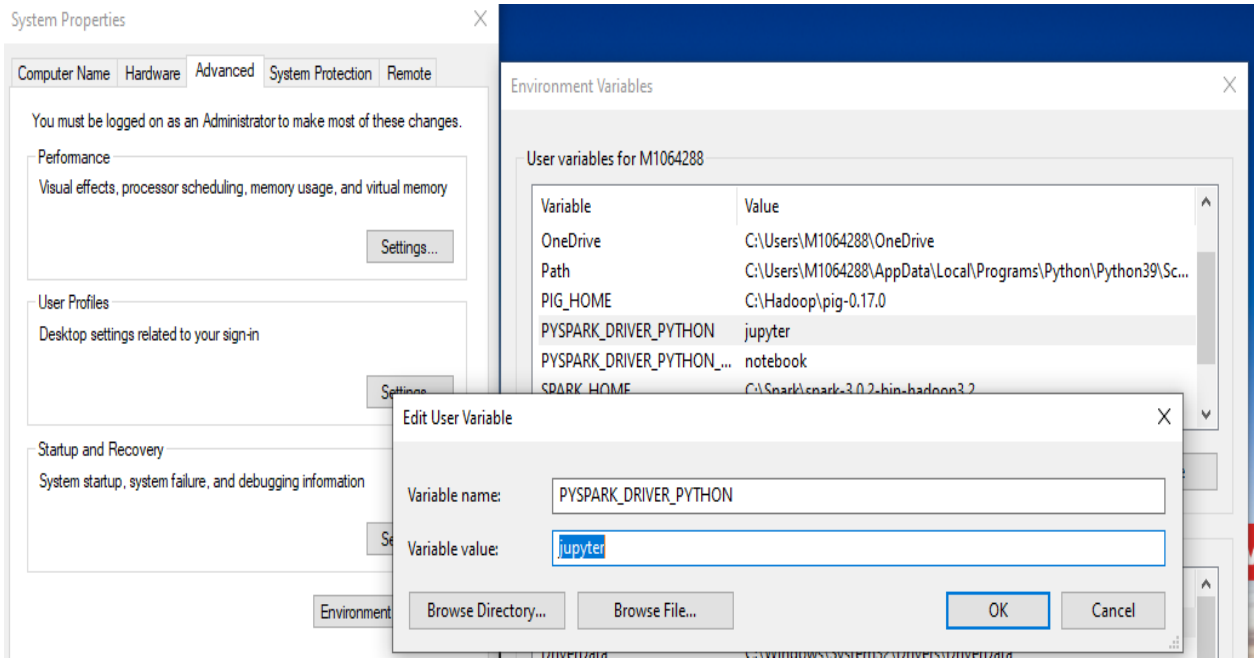


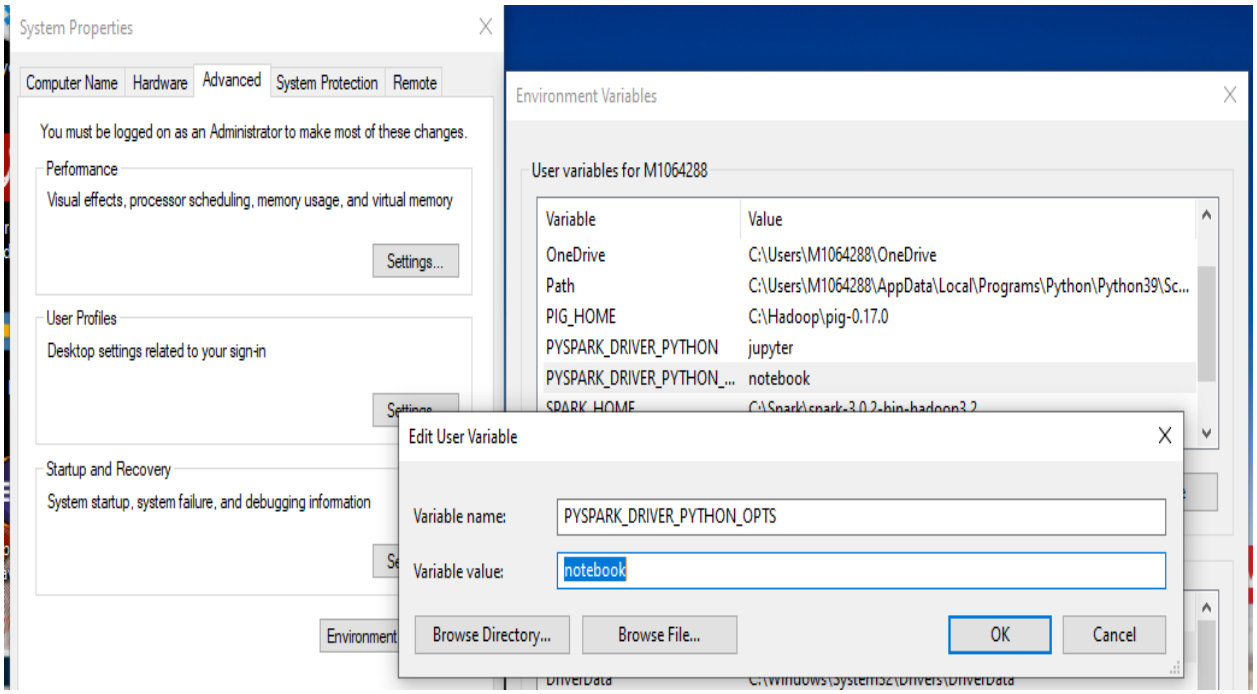
Step2: To download pyspark for the windows then click on the given link https://mirrors.estointernet.in/apache/spark/spark-3.0.2/spark-3.0.2-bin-hadoop3.2.tgz

Step 3: After downloading extract the files to a folder.

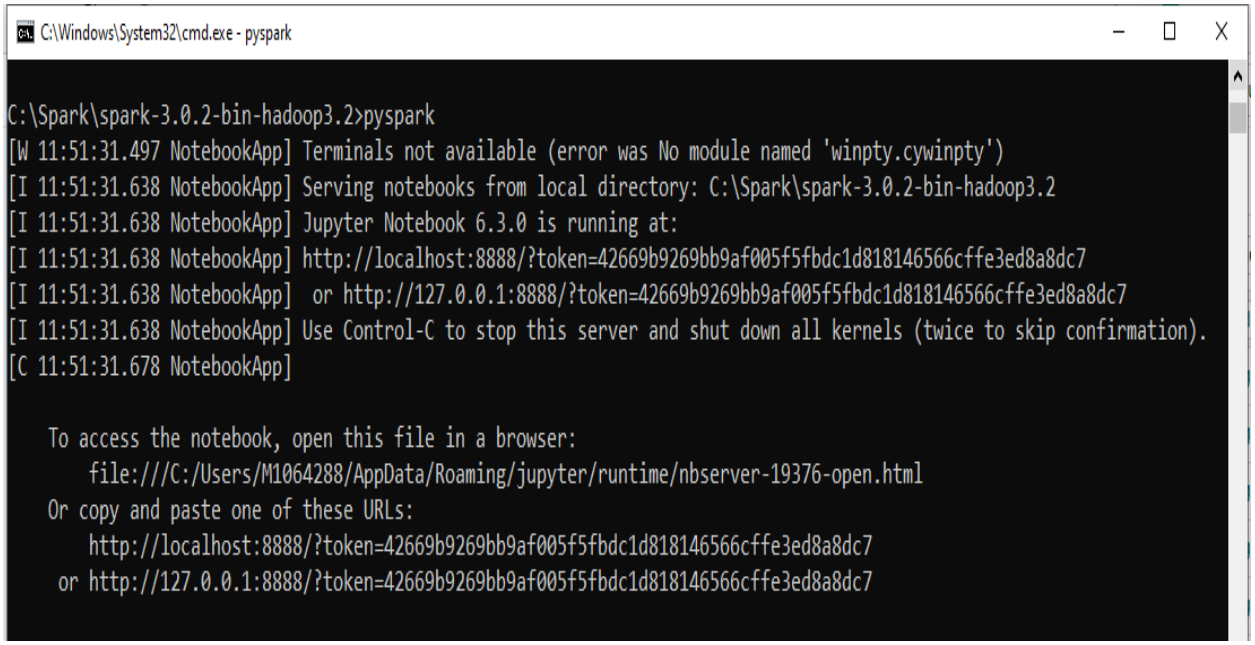


Step 4: Set environment variables for pyspark in order to open it in jupyter platform. If the Hadoop environment variable is not set earlier then need to create it as well because spark runs on top of Hadoop.

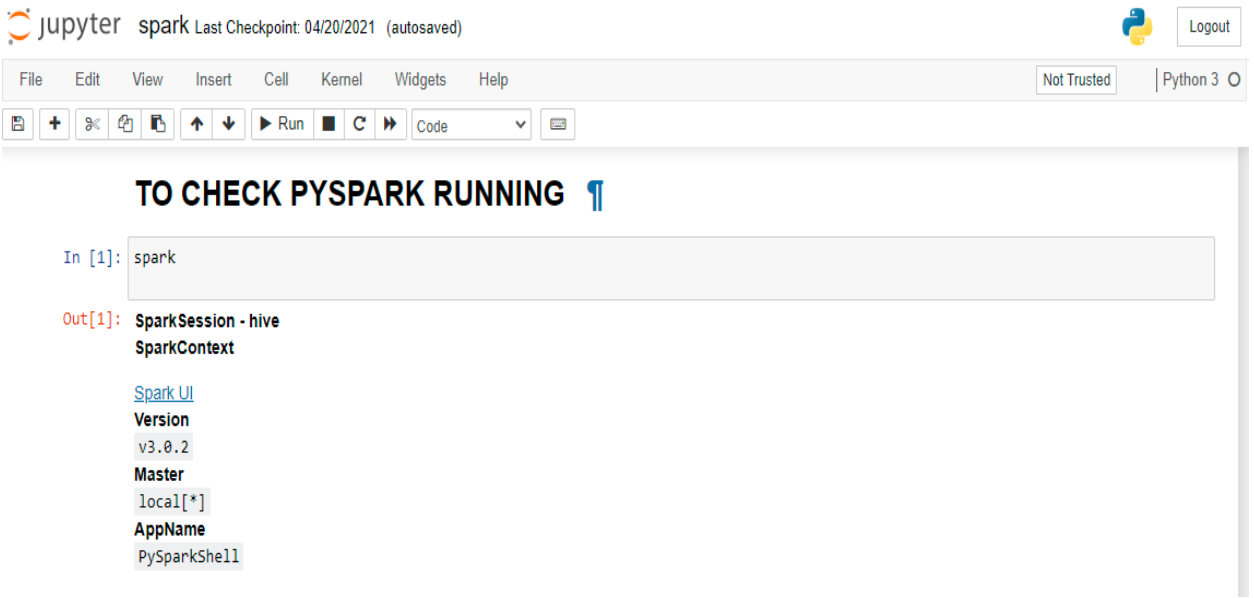




Step 5: Check if pyspark is installed properly and opens in juypter by giving command as pyspark.

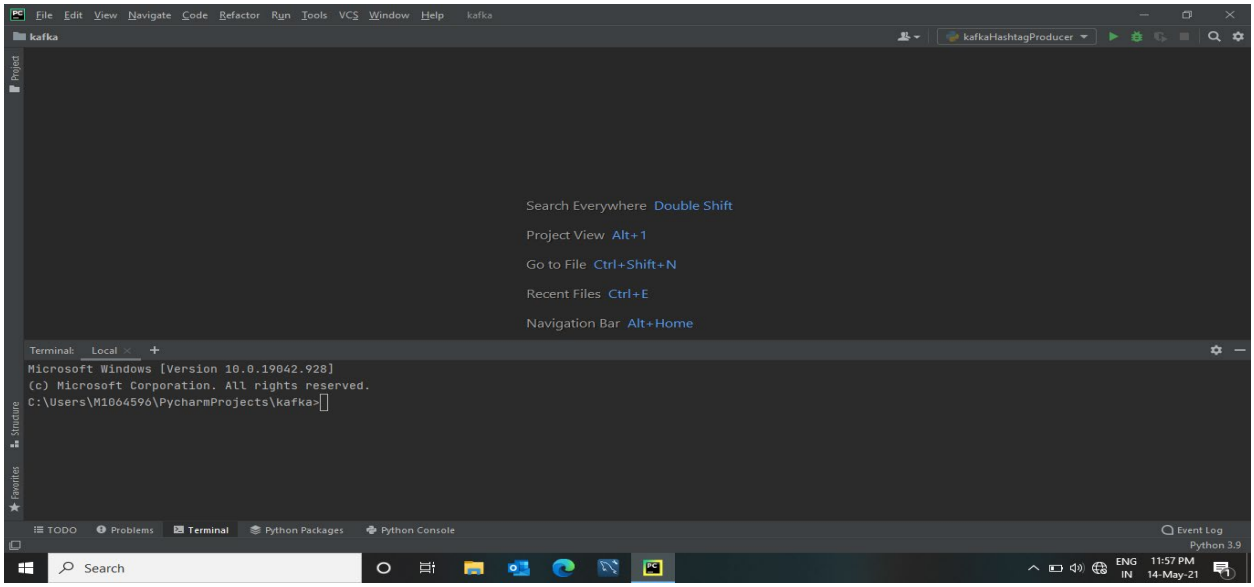


Step 6: To check if the spark is working properly in juypter than use a command spark.If it provides the details

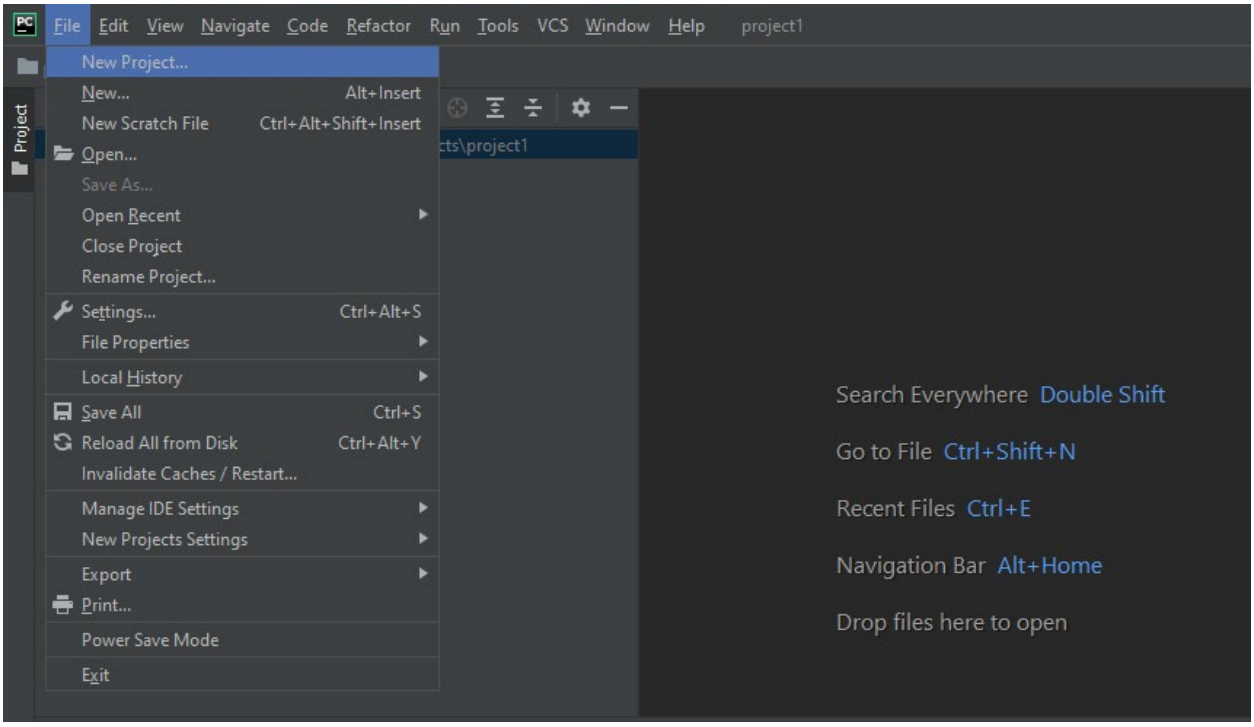


Ability To Set Up Pyspark Project In Pycharm

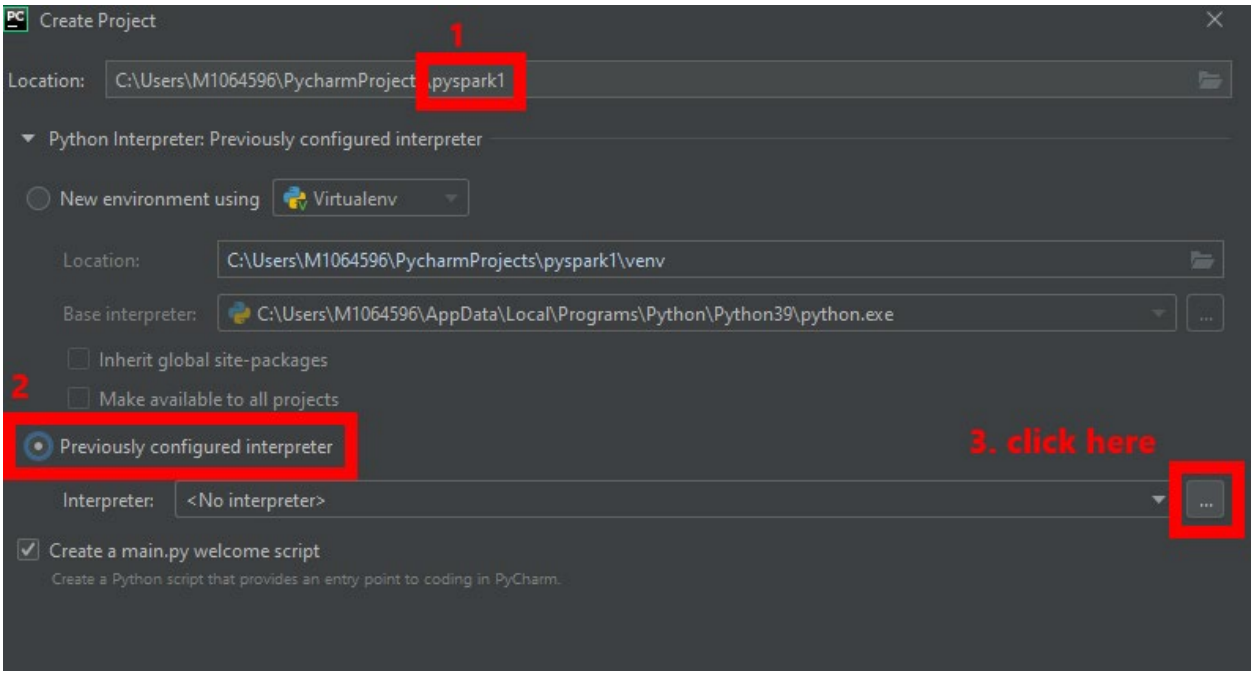
Step 1: Open PyCharm.



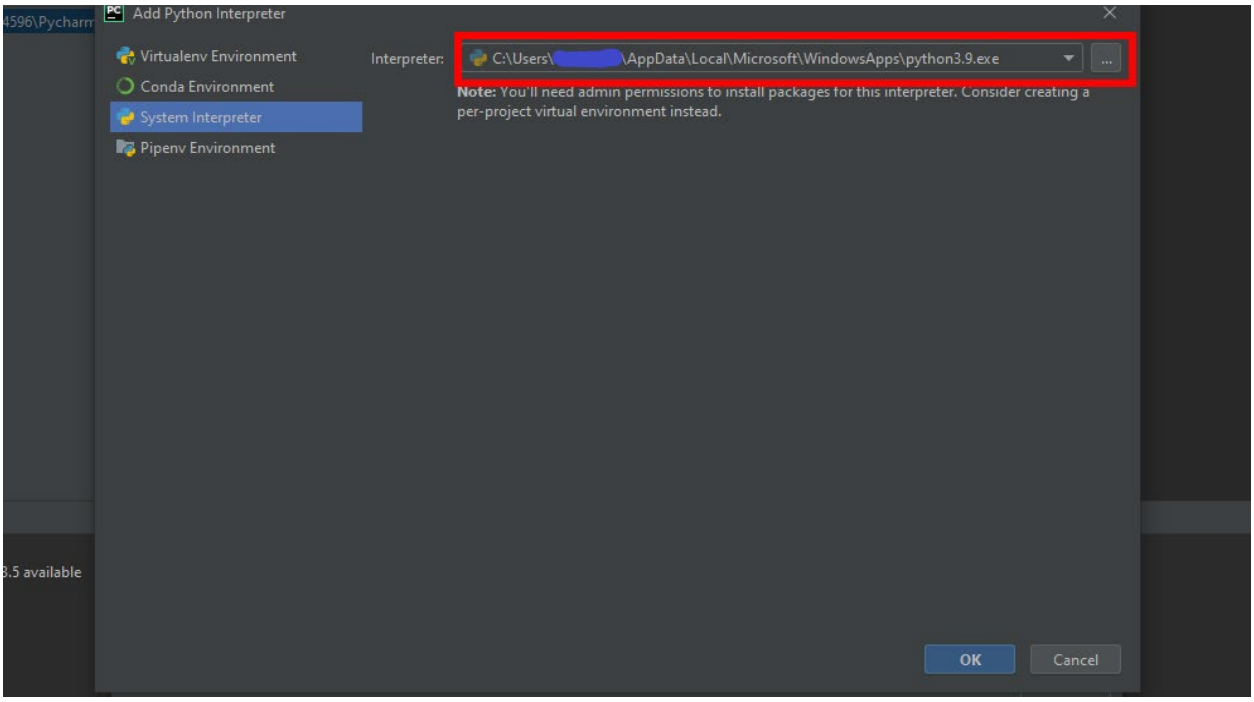
Step 2: Click on File > New Project.



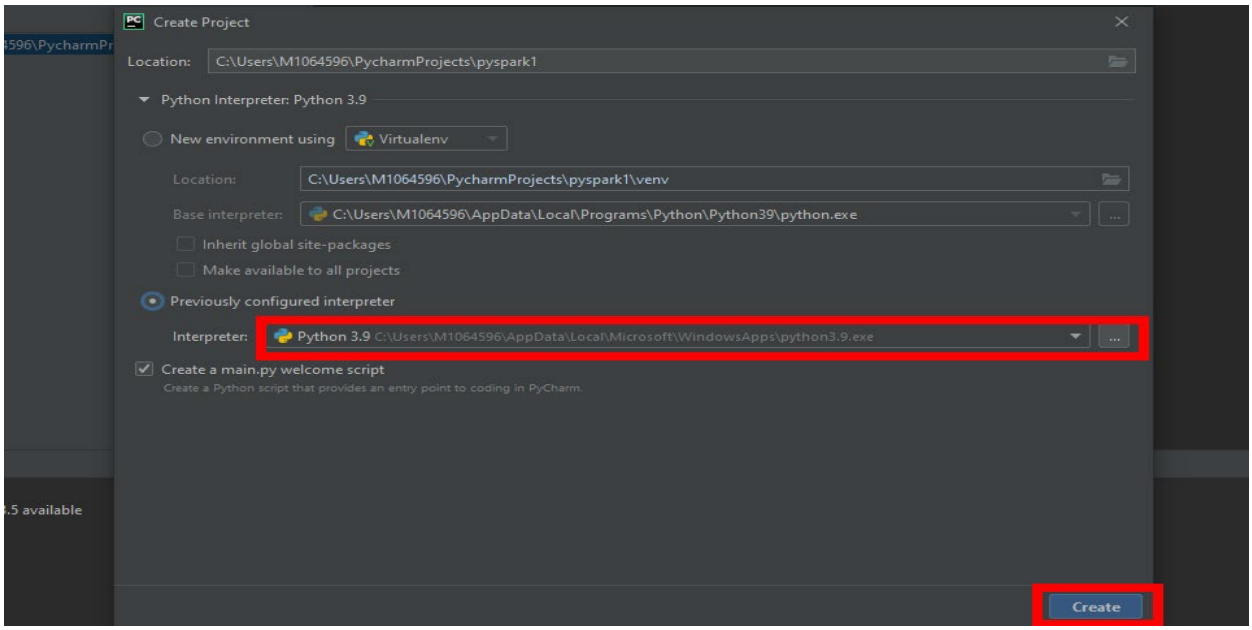
Step 3: Give a name to project, Select ‘previously configured interpreter’ and click on the drop icon.



Step 4: New drop box appears, select System interpreter, select the python.exe file from the location where python installed in your local system. Then click ok.

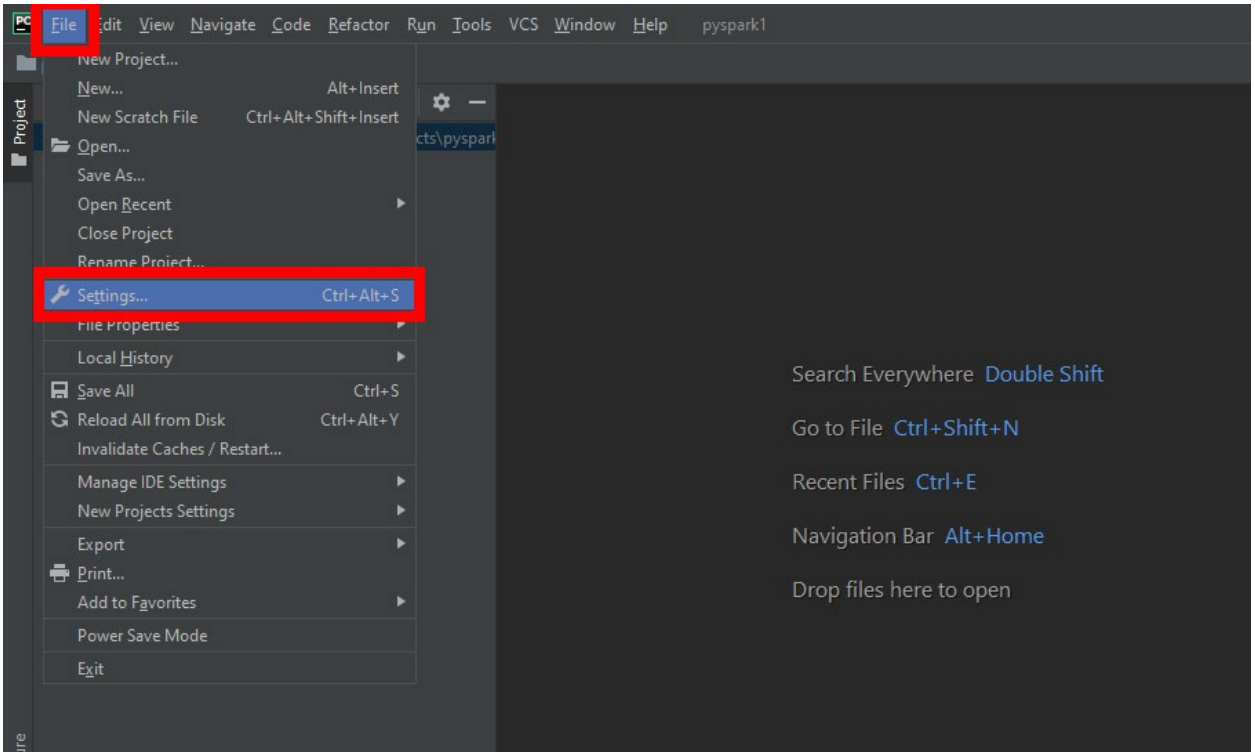


Step 5: Observe the System interpreter added, and click on create.

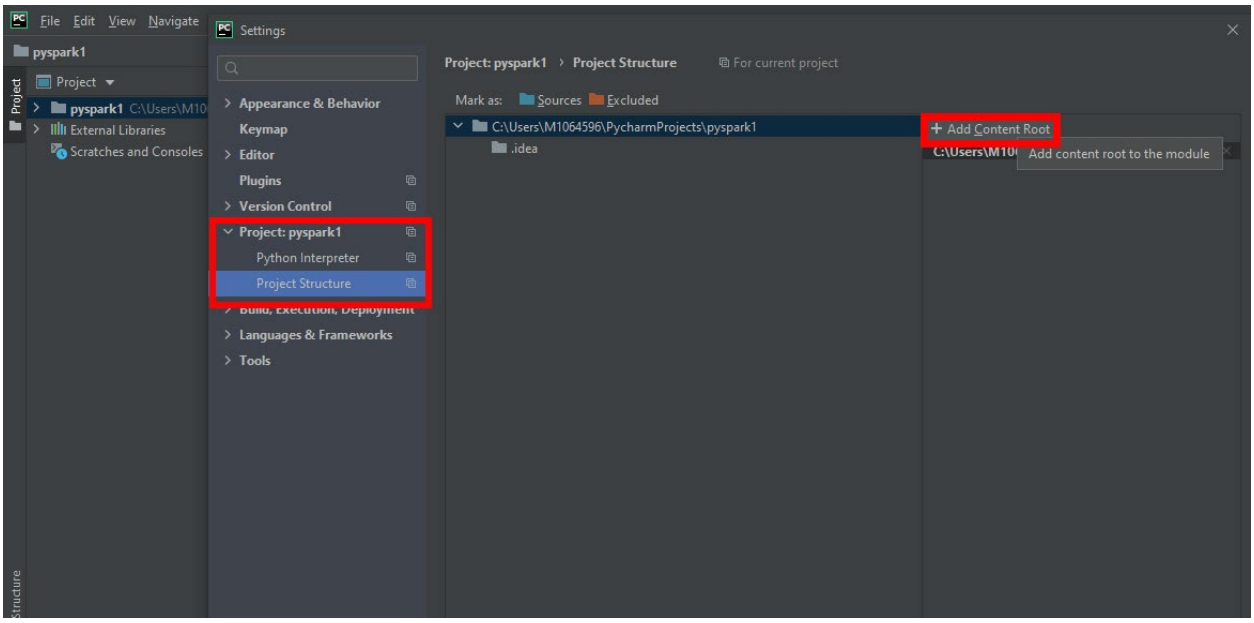


Now project is created.

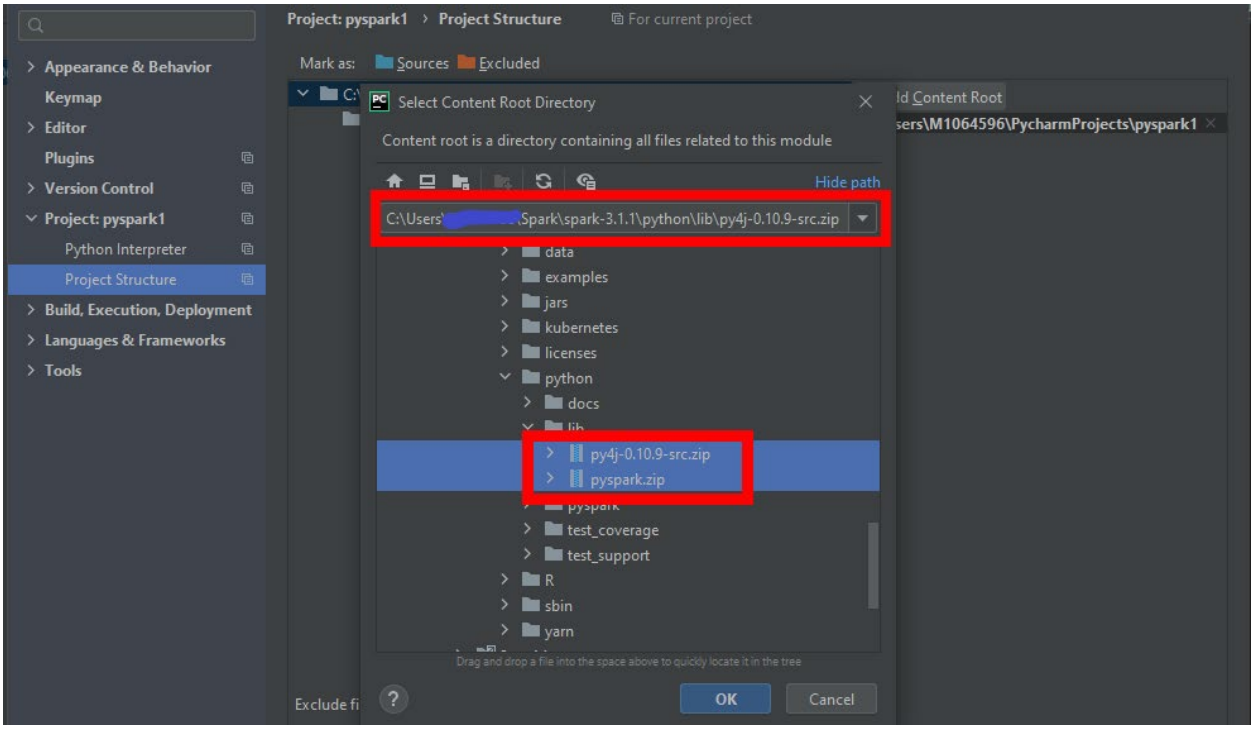
Step 6: Click File > Settings



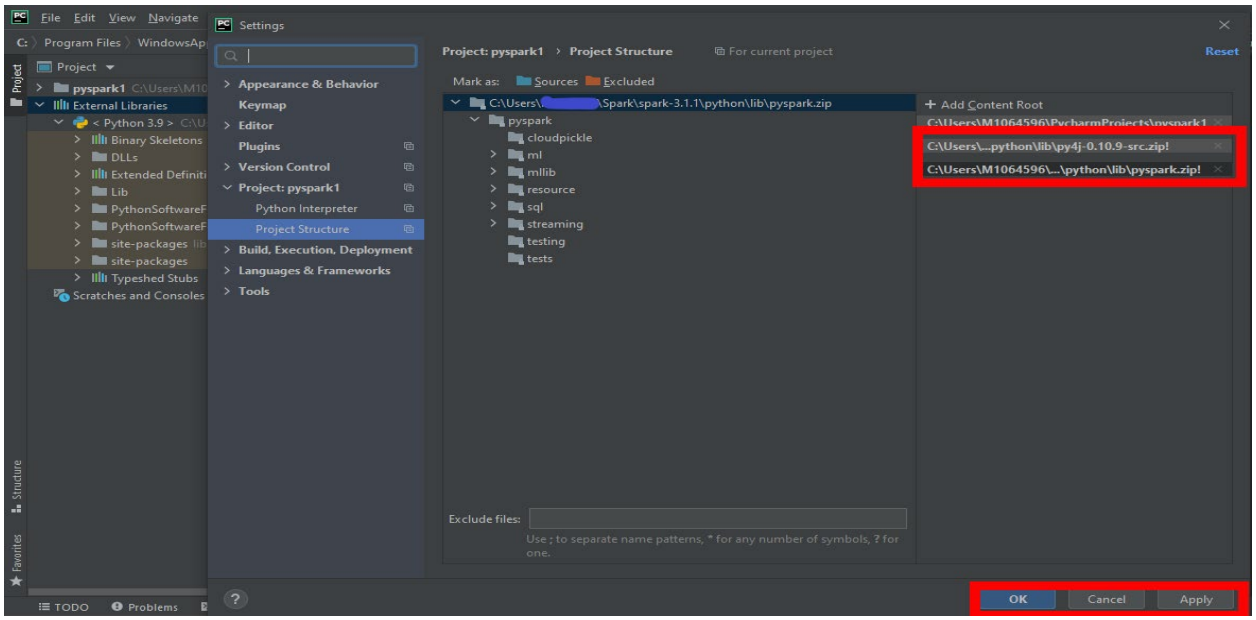
Step 7: Click on project, select Project structure, then click on Add Content Root.



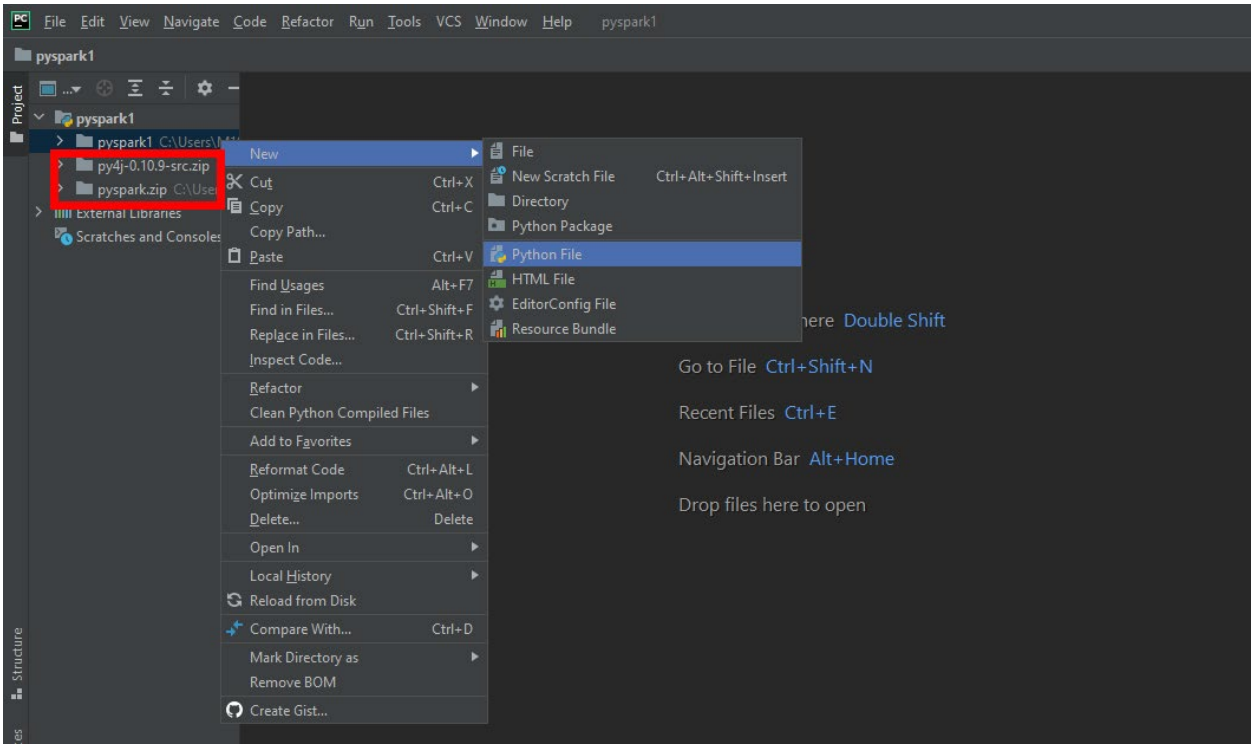
Step 8: Select the Library files py4j-0.10.9-src.zip &pyspark.zip available in the Spark > python > lib directory click on ok.



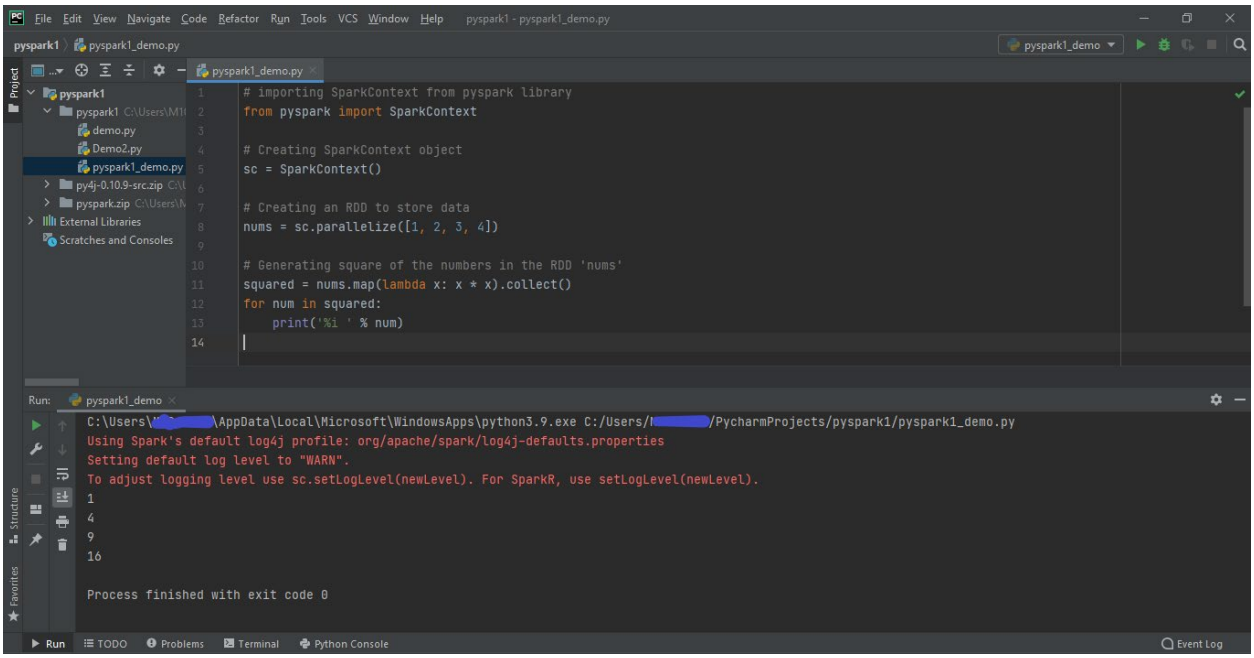
Step 9: Observe the library files root directory added, then click on Apply and then ok.



Step 10: Observe library files added to the project. Then right click on pyspark1 (Project name)> New > Python file.

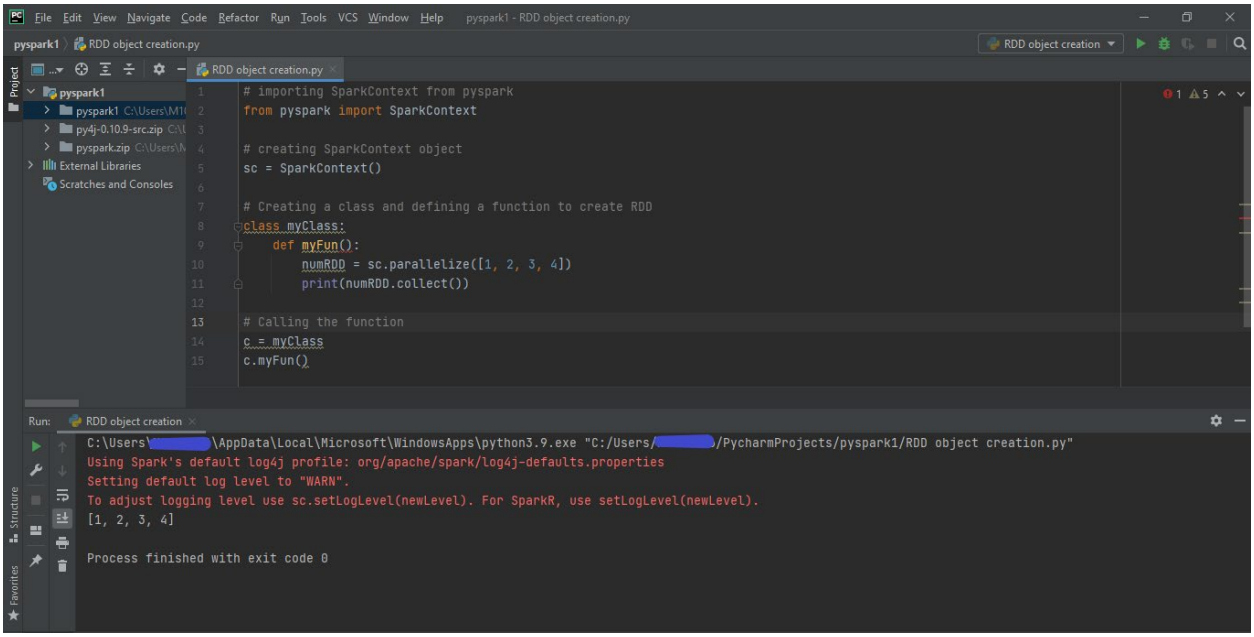


Step 11: Create a python file with name pyspark1_demo, write pyspark code. Below example shows how to create an RDD with integer values and calculate square of them.

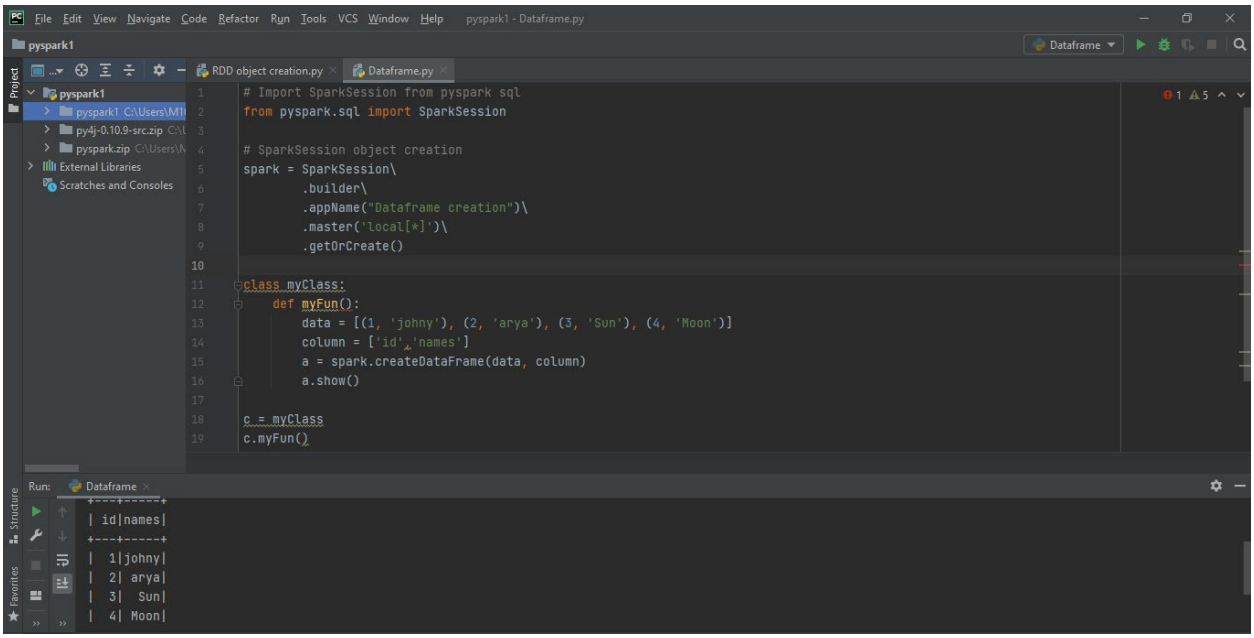


Creating object/class with oops inline

1. Creating RDD in PySpark using class.



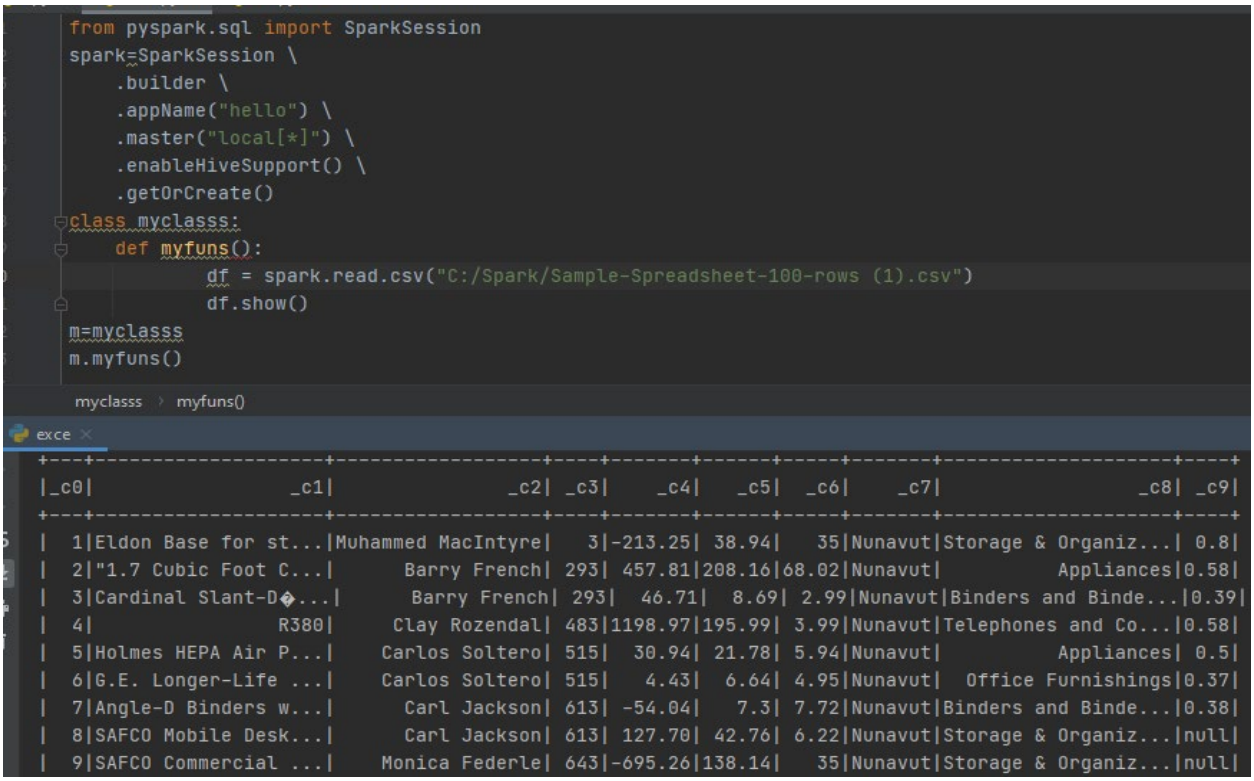
2. Creating Dataframe inside a class



```
# Import SparkSession from pyspark sql
from pyspark.sql import SparkSession

# SparkSession object creation
spark = SparkSession\
        .builder\
        .appName("Dataframe creation")\
        .master('local[*]')\
        .getOrCreate()

class myClass:
    def myFun():
        data = [(1, 'johny'), (2, 'arya'), (3, 'Sun'), (4, 'Moon')]
        column = ['id','names']
        a = spark.createDataFrame(data, column)
        a.show()

c = myClass
c.myFun()
```

```
+---+-----+
| id|names|
+---+-----+
|  1|johny|
|  2| arya|
|  3|  Sun|
|  4| Moon|
+---+-----+
```

3. Reading CSV file in PySpark



```
from pyspark.sql import SparkSession
spark=SparkSession \
    .builder \
    .appName("hello") \
    .master("local[*]") \
    .enableHiveSupport() \
    .getOrCreate()
class myclasss:
    def myfuns():
            df = spark.read.csv("C:/Spark/Sample-Spreadsheet-100-rows (1).csv")
            df.show()
m=myclasss
m.myfuns()
```

myclasss › myfuns()

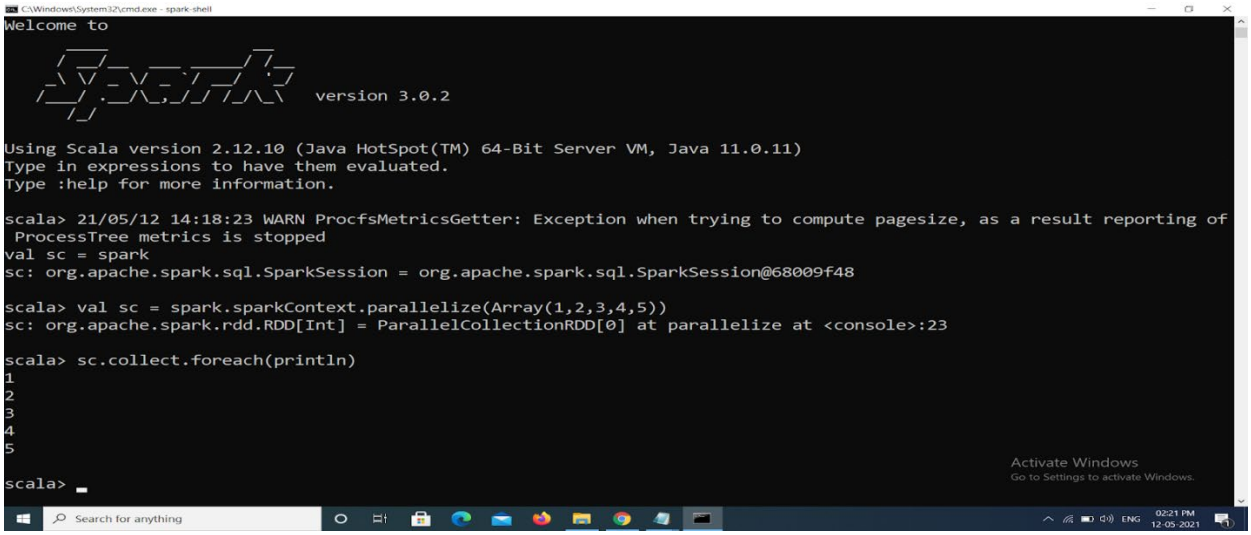
exce

```
+---+--------------------+-----------------+----+-------+------+-----+-------+--------------------+----+
|_c0|                 _c1|              _c2| _c3|    _c4|   _c5|  _c6|    _c7|                 _c8| _c9|
+---+--------------------+-----------------+----+-------+------+-----+-------+--------------------+----+
|  1|Eldon Base for st...|Muhammed MacIntyre|   3|-213.25| 38.94|   35|Nunavut|Storage & Organiz...| 0.8|
|  2|"1.7 Cubic Foot C...|      Barry French| 293| 457.81|208.16|68.02|Nunavut|          Appliances|0.58|
|  3|Cardinal Slant-D...|      Barry French| 293|  46.71|  8.69| 2.99|Nunavut|Binders and Binde...|0.39|
|  4|                R380|     Clay Rozendal| 483|1198.97|195.99| 3.99|Nunavut|Telephones and Co...|0.58|
|  5|Holmes HEPA Air P...|    Carlos Soltero| 515|  30.94| 21.78| 5.94|Nunavut|          Appliances| 0.5|
|  6|G.E. Longer-Life ...|    Carlos Soltero| 515|   4.43|  6.64| 4.95|Nunavut|  Office Furnishings|0.37|
|  7|Angle-D Binders w...|      Carl Jackson| 613| -54.04|   7.3| 7.72|Nunavut|Binders and Binde...|0.38|
|  8|SAFCO Mobile Desk...|      Carl Jackson| 613| 127.70| 42.76| 6.22|Nunavut|Storage & Organiz...|null|
|  9|SAFCO Commercial ...|    Monica Federle| 643|-695.26|138.14|   35|Nunavut|Storage & Organiz...|null|
```

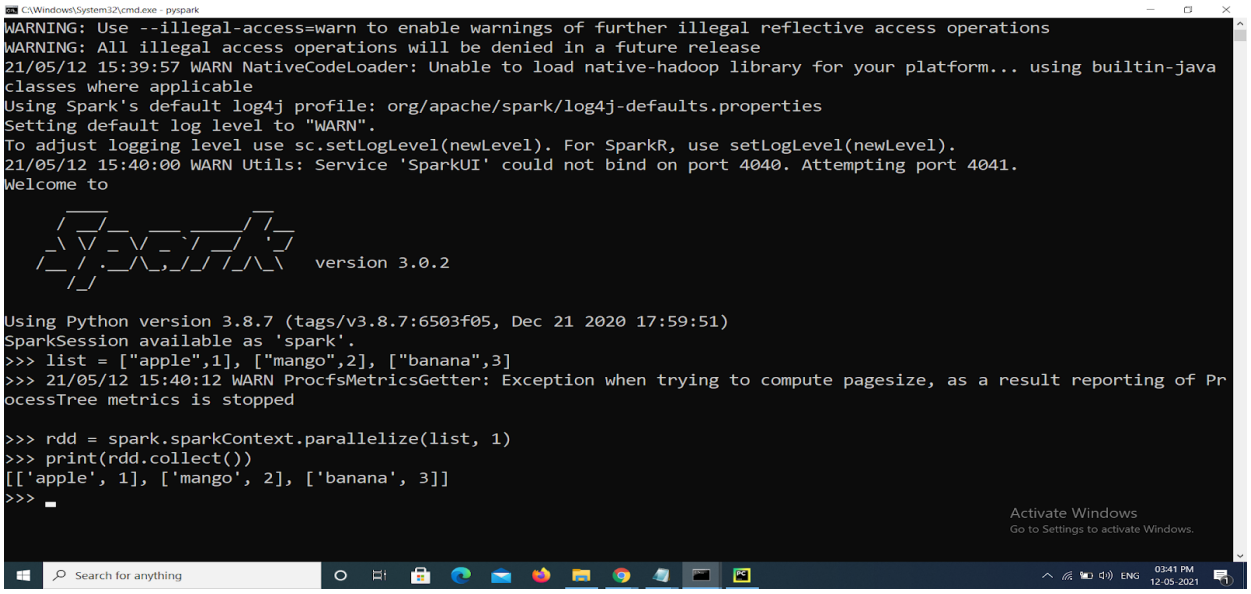
Sparkcontext And Sparksession

SparkContext

SparkContext is the entry gate of Apache Spark functionality. The most important step of any Spark driver application is to generate SparkContext.
It provides a way to interact with various spark's functionality with a lesser number of constructs.
In Scala



In python

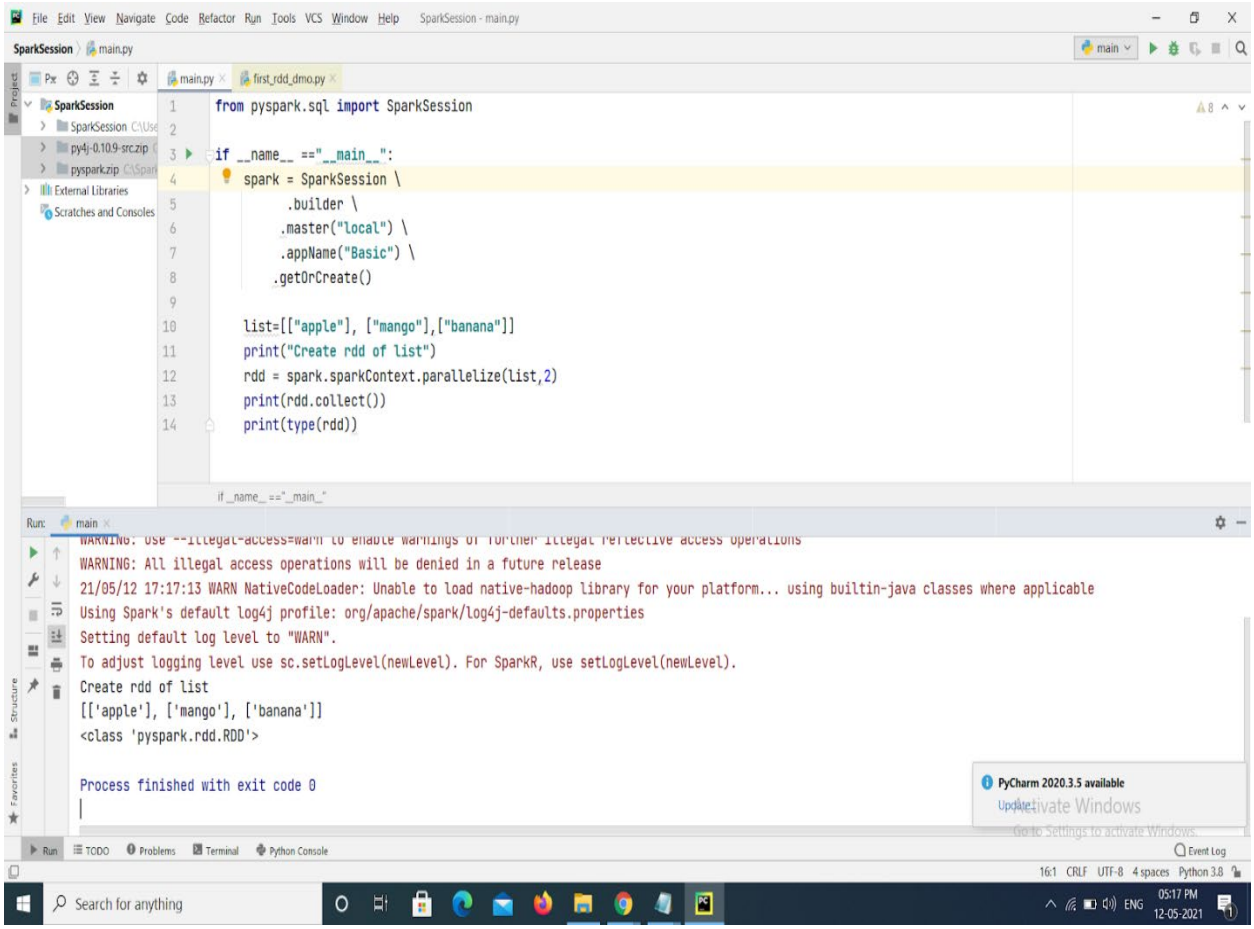


SparkSession

Spark session is a unified entry point of a spark application from Spark 2.0. It provides a way to interact with various spark's functionality with a lesser number of constructs. Instead of having a spark context, hive context, SQL context, now all of it is encapsulated in a Spark session.

Able to Initialize SparkSession: Creating SparkSession using Pycharm IDE.

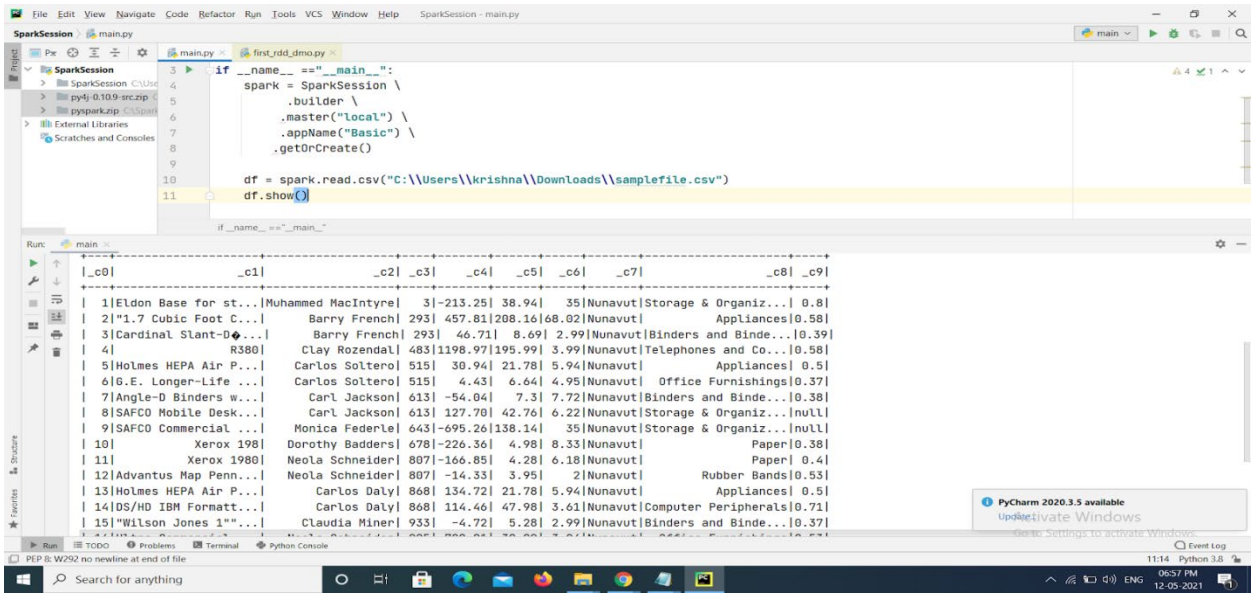
Import sparkSession and write a program for creating sparkSession.



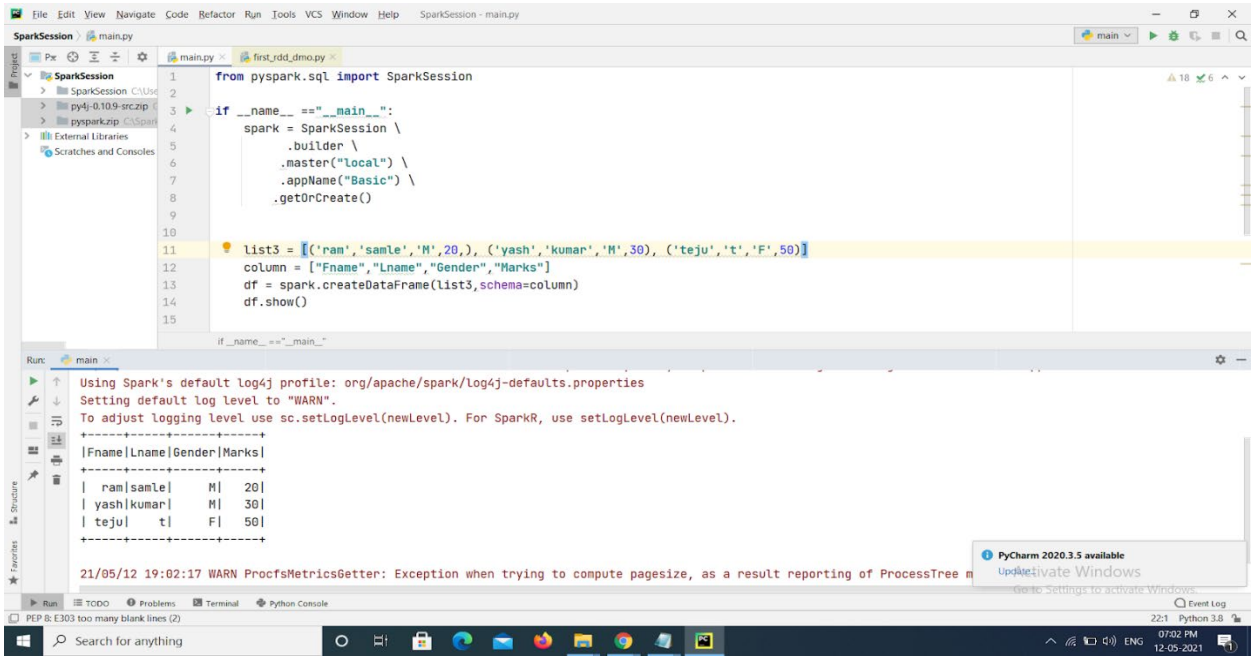
Ability To Ingest Data

Able to Read the input data RDD / Data Frame

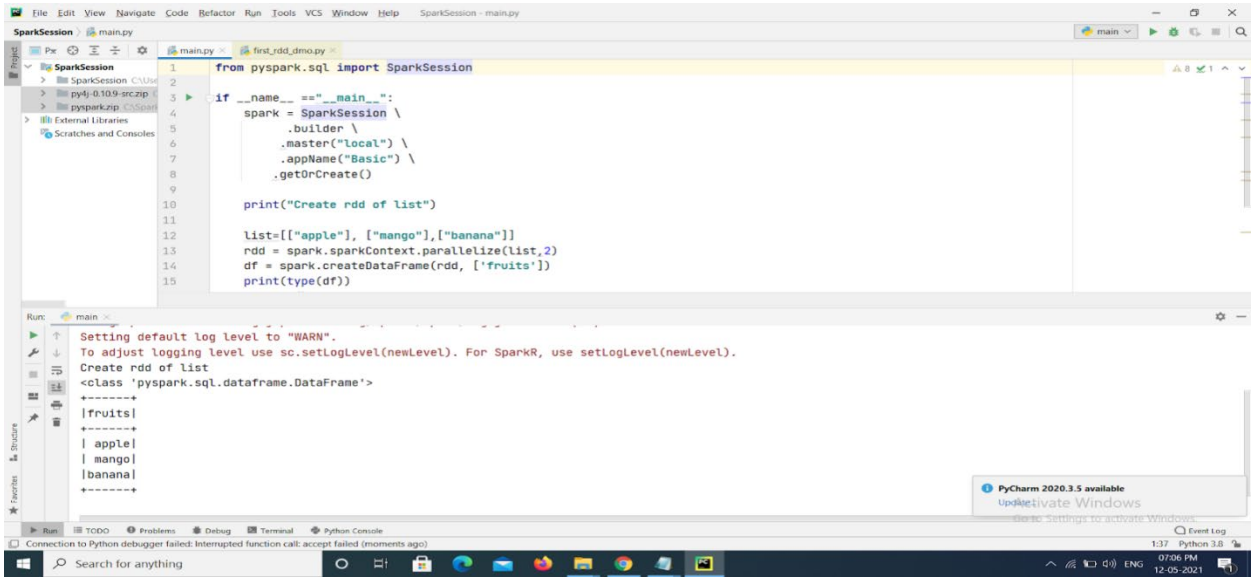
1. Read csv file and show using dataframe.



2. Create dataframe using list



3. Convert RDD to Dataframe

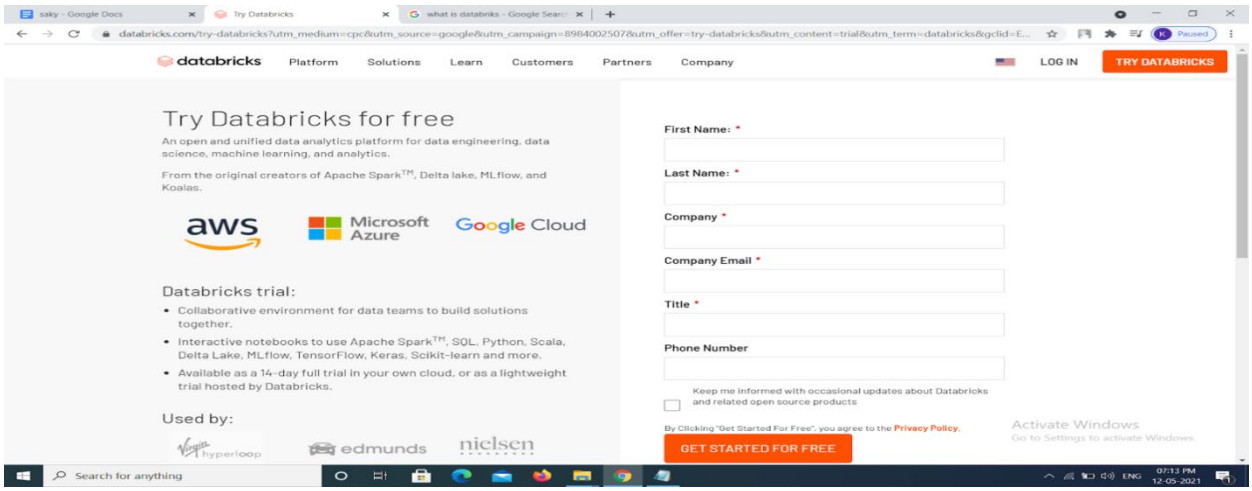


Databricks Platform

Databricks is an enterprise software company founded by the original creators of Apache Spark. The company has also created Delta Lake, MLflow and Koalas, popular open source projects that span data engineering, data science and machine learning. We can also create databricks account in azure.

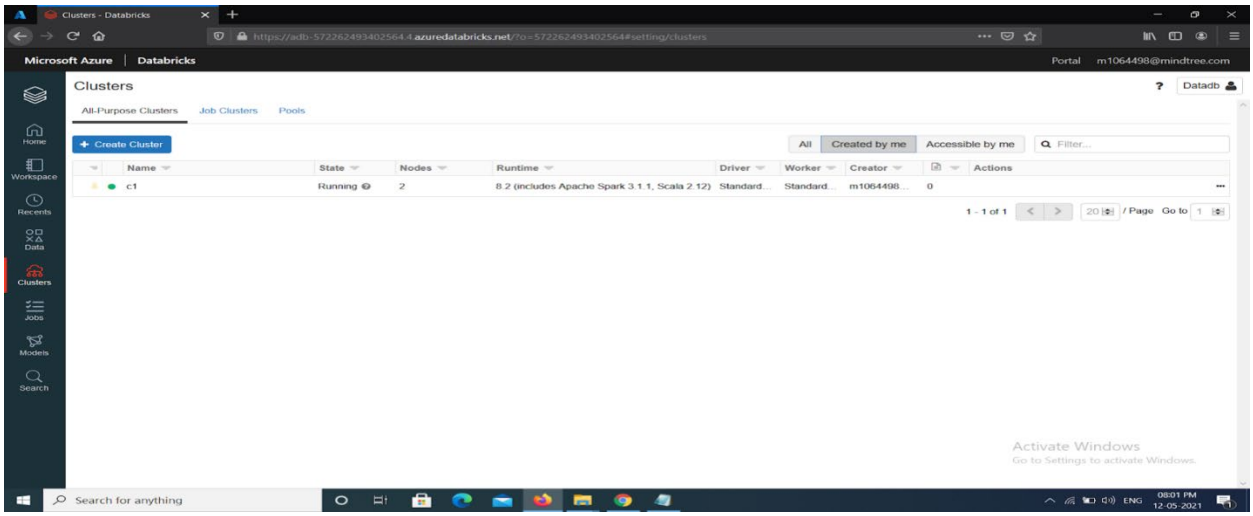
Azure Databricks is based on Apache Spark and provides in memory compute with language support for Scala, R, Python and SQL.

Step 1: Create account and login in databricks https://www.databricks.com/



Step 2: Create clusters and it is use job clusters to run fast and robust automated jobs.

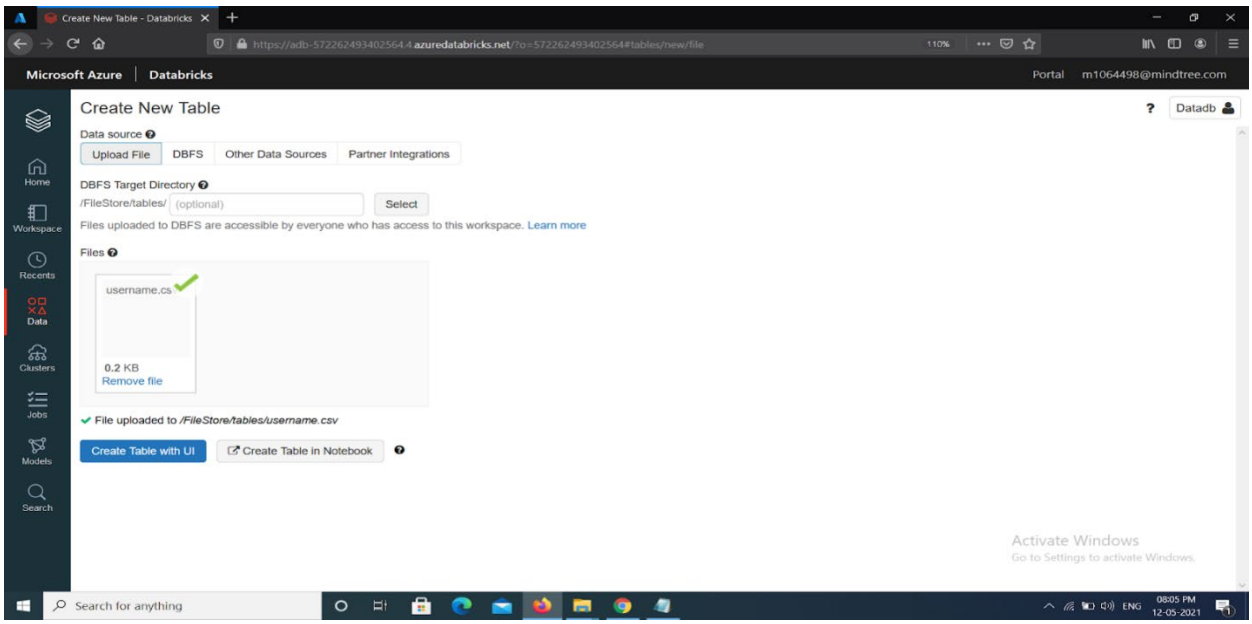
Click Clusters >> Create Cluster >> Cluster Name >> Cluster Mode (Standard) >> Create Cluster



Step 3: Create New Table for upload file in databricks.

Click on Create New Table >> drag and drop file >> Create table with UI >> Select a Cluster to Preview the Table >> preview table >> create table.

If you have small data files on your local machine that you want to analyze with Databricks, you can import them to DBFS using the UI.



Step 4: Open the notebook for getting the automatic generated query from file loaded in the cluster.

```
file_location = "/FileStore/tables/sample.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

▸ (2) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 8 more fields]

	_c0	_c1	_c2	_c3	_c4
1	1	Eldon Base for stackable storage shelf, platinum	Muhammed MacIntyre	3	-213.25
2	2	"1.7 Cubic Foot Compact ""Cube"" Office Refrigerators"	Barry French	293	457.81
3	3	Cardinal Slant-D Ring Binder, Heavy Gauge Vinyl	Barry French	293	46.71
4	4	R380	Clay Rozendal	483	1198.97
5	5	Holmes HEPA Air Purifier	Carlos Soltero	515	30.94
6	6	G.E. Longer-Life Indoor Recessed Floodlight Bulbs	Carlos Soltero	515	4.43

Step 5: We can can run each cell of the notebook for the result or also create new cell for new query.

```
temp_table_name = "sample_csv"

df.createOrReplaceTempView(temp_table_name)
```

nd 4

```
%sql

/* Query the created temp table in a SQL cell */

select * from `sample_csv`
```

▸ (1) Spark Jobs

	_c0	_c1	_c2	_c3	_c4
1	1	Eldon Base for stackable storage shelf, platinum	Muhammed MacIntyre	3	-213.25
2	2	"1.7 Cubic Foot Compact ""Cube"" Office Refrigerators"	Barry French	293	457.81
3	3	Cardinal Slant-D Ring Binder, Heavy Gauge Vinyl	Barry French	293	46.71
4	4	R380	Clay Rozendal	483	1198.97
5	5	Holmes HEPA Air Purifier	Carlos Soltero	515	30.94
6	6	G.E. Longer-Life Indoor Recessed Floodlight Bulbs	Carlos Soltero	515	4.43
7	7	Angle-D Binders with Locking Rings, Label Holders	Carl Jackson	613	-54.04

Showing all 100 rows.

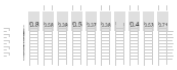
Step 6: There are options for getting result of query in graph formats.

```
# Create a view or table

temp_table_name = "sample_csv"

df.createOrReplaceTempView(temp_table_name)
```

Cmd 4

```
%sql

/* Query the created temp table in a SQL cell */

select * from `sample_csv`
```

▸ (1) Spark Jobs



Quantile

Plot Options...

Metadata

Metadata is a data about data or information that describes the data (summary of data).There are few methods that provide metadata .

1. **show():** This shows all the data in dataframe.

display data ¶

```
In [2]: df=spark.read.csv("C:/Spark/sample.csv")
```

```
In [10]: df.show()
```

```
+---+--------------------+------------------+----+-------+------+-----+-------+--------------------+----+
|_c0|                 _c1|               _c2| _c3|    _c4|   _c5|  _c6|    _c7|                 _c8| _c9|
+---+--------------------+------------------+----+-------+------+-----+-------+--------------------+----+
|  1|Eldon Base for st...|Muhammed MacIntyre|   3|-213.25| 38.94|   35|Nunavut|Storage & Organiz...| 0.8|
|  2|"1.7 Cubic Foot C...|      Barry French| 293| 457.81|208.16|68.02|Nunavut|          Appliances|0.58|
|  3|Cardinal Slant-D...|      Barry French| 293|  46.71|  8.69| 2.99|Nunavut|Binders and Binde...|0.39|
|  4|                R380|     Clay Rozendal| 483|1198.97|195.99| 3.99|Nunavut|Telephones and Co...|0.58|
|  5|Holmes HEPA Air P...|    Carlos Soltero| 515|  30.94| 21.78| 5.94|Nunavut|          Appliances| 0.5|
|  6|G.E. Longer-Life ...|    Carlos Soltero| 515|   4.43|  6.64| 4.95|Nunavut|  Office Furnishings|0.37|
|  7|Angle-D Binders w...|      Carl Jackson| 613| -54.04|   7.3| 7.72|Nunavut|Binders and Binde...|0.38|
|  8|SAFCO Mobile Desk...|      Carl Jackson| 613| 127.70| 42.76| 6.22|Nunavut|Storage & Organiz...|null|
|  9|SAFCO Commercial ...|    Monica Federle| 643|-695.26|138.14|   35|Nunavut|Storage & Organiz...|null|
| 10|           Xerox 198|   Dorothy Badders| 678|-226.36|  4.98| 8.33|Nunavut|               Paper|0.38|
| 11|          Xerox 1980|   Neola Schneider| 807|-166.85|  4.28| 6.18|Nunavut|               Paper| 0.4|
| 12|Advantus Map Penn...|   Neola Schneider| 807| -14.33|  3.95|    2|Nunavut|        Rubber Bands|0.53|
| 13|Holmes HEPA Air P...|       Carlos Daly| 868| 134.72| 21.78| 5.94|Nunavut|          Appliances| 0.5|
| 14|DS/HD IBM Formatt...|       Carlos Daly| 868| 114.46| 47.98| 3.61|Nunavut|Computer Peripherals|0.71|
| 15|"Wilson Jones 1""...|     Claudia Miner| 933|  -4.72|  5.28| 2.99|Nunavut|Binders and Binde...|0.37|
| 16|Ultra Commercial ...|   Neola Schneider| 995| 782.91| 39.89| 3.04|Nunavut|  Office Furnishings|0.53|
| 17|"#10-4 1/8"" x 9 ...|  Allen Rosenblatt| 998|  93.80| 15.74| 1.39|Nunavut|           Envelopes| 0.4|
| 18|Hon 4-Shelf Metal...|   Sylvia Foulston|1154| 440.72|100.98|26.22|Nunavut|           Bookcases| 0.6|
| 19|Lesro Sheffield C...|   Sylvia Foulston|1154|-481.04| 71.37|   69|Nunavut|              Tables|0.68|
```

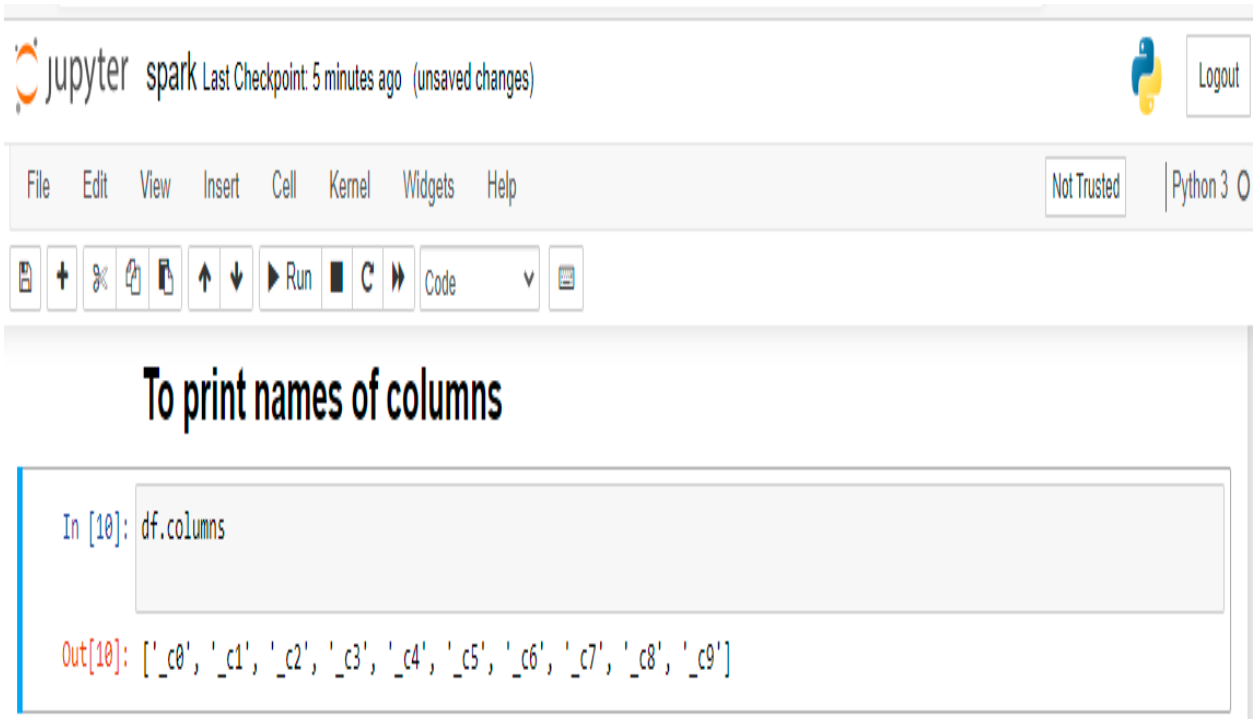
Using show(truncate=False),the truncation of data can be avoided.

To avoid truncation

```
In [25]: df.show(truncate=False)
```

```
+---+----------------------------------------------------------------------------+-----------------+---------+-------+-----+-----+-------+----------+
|_c0|_c1                                                                         |_c2              |_c3      |_c4    |_c5  |_c6  |_c7    |_c8       |_c9       |
+---+----------------------------------------------------------------------------+-----------------+---------+-------+-----+-----+-------+----------+
|1  |Eldon Base for stackable storage shelf, platinum                            |Muhammed MacIntyre|3        |-213.25|38.94|35   |Nunavut|Storage & Organization|0.8       |
|2  |"1.7 Cubic Foot Compact ""Cube"" Office Refrigerators"                      |Barry French     |293      |457.81 |208.16|68.02|Nunavut|Appliances|0.58      |
|3  |Cardinal Slant-D® Ring Binder, Heavy Gauge Vinyl                            |Barry French     |293      |46.71  |8.69 |2.99 |Nunavut|Binders and Binder Accessories|0.39      |
|4  |R380                                                                        |Clay Rozendal    |483      |1198.97|195.99|3.99 |Nunavut|Telephones and Communication|0.58      |
|5  |Holmes HEPA Air Purifier                                                    |Carlos Soltero   |515      |30.94  |21.78|5.94 |Nunavut|Appliances|0.5       |
|6  |G.E. Longer-Life Indoor Recessed Floodlight Bulbs                           |Carlos Soltero   |515      |4.43   |6.64 |4.95 |Nunavut|Office Furnishings|0.37      |
|7  |Angle-D Binders with Locking Rings, Label Holders                           |Carl Jackson     |613      |-54.04 |7.3  |7.72 |Nunavut|Binders and Binder Accessories|0.38      |
|8  |SAFCO Mobile Desk Side File, Wire Frame                                     |Carl Jackson     |613      |127.70 |42.76|6.22 |Nunavut|Storage & Organization|null      |
|9  |SAFCO Commercial Wire Shelving, Black                                       |Monica Federle   |643      |-695.26|138.14|35   |Nunavut|Storage & Organization|null      |
|10 |Xerox 198                                                                   |Dorothy Badders  |678      |-226.36|4.98 |8.33 |Nunavut|Paper     |0.38      |
```

2. columns: This prints the name of columns in dataframe.



3. **describe():** This provides statistical insight on the data present in dataframe.

To show statistical info on data

```
In [12]: df.describe().show()
```

```
+-------+-----------------+------------------+-------------+---------------+----------------+-----------------+------
-----------+---------------+------------------+-----------------+
|summary|              _c0|               _c1|          _c2|            _c3|             _c4|              _c5|
_c6|             _c7|               _c8|              _c9|
+-------+-----------------+------------------+-------------+---------------+----------------+-----------------+------
-----------+---------------+------------------+-----------------+
|  count|              100|               100|          100|            100|             100|              100|
100|            100|               100|               97|
|   mean|             50.5|            6160.0|         null|         5069.0|326.58259999999996| 76.13869999999991|13.875
900000000001|          4.076|              null| 0.5152173913043478|
| stddev|29.011491975882016|               NaN|         null|3453.319445329286|1369.8326563759458|185.10559463986982|18.292
605068494733|1.413552262917788|              null|0.14419521460159157|
|    min|                1|"#10-4 1/8"" x 9 ...|      10/Pack"|          10437|           -0.06|           -144.55|
0.49|           2.27|        Appliances|             0.35|
|    max|               99|               g520|Sylvia Foulston|   Eugene Barchas|           93.80|            95.99|
9.03|        Nunavut|Telephones and Co...|            Paper|
+-------+-----------------+------------------+-------------+---------------+----------------+-----------------+------
-----------+---------------+------------------+-----------------+
```

4. **dtypes:** This provides the datatypes of the columns.

To print datatype of columns

```
In [12]: df.dtypes
```

```
Out[12]: [('_c0', 'string'),
          ('_c1', 'string'),
          ('_c2', 'string'),
          ('_c3', 'string'),
          ('_c4', 'string'),
          ('_c5', 'string'),
          ('_c6', 'string'),
          ('_c7', 'string'),
          ('_c8', 'string'),
          ('_c9', 'string')]
```

5. **printSchema():** This provides the schema of the dataframe in tree format.

To print schema in tree format

```
In [13]: df.printSchema()

root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
```

6. **schema:** This provides the schema of the dataframe in structure format.

To print schema in structure format

```
In [14]: df.schema

Out[14]: StructType(List(StructField(_c0,StringType,true),StructField(_c1,StringType,true),StructField(_c2,StringType,true),StructField
(_c3,StringType,true),StructField(_c4,StringType,true),StructField(_c5,StringType,true),StructField(_c6,StringType,true),Struct
Field(_c7,StringType,true),StructField(_c8,StringType,true),StructField(_c9,StringType,true)))
```

Try And Except

The usage of try and except blocks for reading data is similar to the usage of try and except blocks in python to handle the errors.

Try and Except for reading the data

In [25]:
```
from pyspark.sql import SparkSession
spark=SparkSession \
    .builder \
    .appName("hello") \
    .master("local[*]") \
    .enableHiveSupport() \
    .getOrCreate()
class myclasss:
    def myfuns():
        try:
            df = spark.read.csv("C:/Spark/samples.csv")
            df.show()
        except  Exception as error:
            print("There is error in reading dataframe")

m=myclasss
m.myfuns()
```

```
There is error in reading dataframe
```

Try and Except for reading the data

In [27]:
```
from pyspark.sql import SparkSession
spark=SparkSession \
    .builder \
    .appName("hello") \
    .master("local[*]") \
    .enableHiveSupport() \
    .getOrCreate()
class myclasss:
    def myfuns():
        try:
            df = spark.read.csv("C:/Spark/samples.csv")
            df.show()

        except  Exception as error:
            print(error)

m=myclasss
m.myfuns()
```

```
Path does not exist: file:/C:/Spark/samples.csv;
```

Similar to python any number of excepts blocks along with finally can be used in pyspark.

Multiple Try and Except for reading the data

In [30]:
```
from pyspark.sql import SparkSession
spark=SparkSession \
    .builder \
    .appName("hello") \
    .master("local[*]") \
    .enableHiveSupport() \
    .getOrCreate()
class myclasss:
    def myfuns():
        try:
            df = spark.read.txt("C:/Spark/sahana.csv")
            df.show()
        except IOError as error:
            print(error)
        except FileNotFoundError as error:
            print(error)
        except Exception as error:
            print(error)
        finally:
            print("pyspark running completed")
if __name__=="__main__" :
    m=myclasss
    m.myfuns()
```

```
'DataFrameReader' object has no attribute 'txt'
pyspark running completed
```

Ability To Apply Cleansing Of Data On The Data Provided

Regular Expression

In pyspark, the regular expression is found in pyspark.sql.function. It has two modules

1. regexp_replace
2. regexp_extract

regexp_extract: extract the characters mentioned in regex_string (string that contain regular expression)

syntax:
dataframe.select(regexp_extract(col("columnname"),regex_string,0).alias("cleaned ")). show()

```
+---+--------------------+-----------------+---+-------+------+-----+-------+--------------------+----+
|_c0|                 _c1|              _c2|_c3|    _c4|   _c5|  _c6|    _c7|                 _c8| _c9|
+---+--------------------+-----------------+---+-------+------+-----+-------+--------------------+----+
|  1|Eldon Base for st...|Muhammed MacIntyre|  3|-213.25| 38.94|   35|Nunavut|Storage & Organiz...| 0.8|
|  2|"1.7 Cubic Foot C...|      Barry French|293| 457.81|208.16|68.02|Nunavut|          Appliances|0.58|
|  3|Cardinal Slant-D�...|      Barry French|293|  46.71|  8.69| 2.99|Nunavut|Binders and Binde...|0.39|
|  4|                R380|     Clay Rozendal|483|1198.97|195.99| 3.99|Nunavut|Telephones and Co...|0.58|
|  5|Holmes HEPA Air P...|    Carlos Soltero|515|  30.94| 21.78| 5.94|Nunavut|          Appliances| 0.5|
+---+--------------------+-----------------+---+-------+------+-----+-------+--------------------+----+
only showing top 5 rows
```

Regular exp for cleaning data by fetching only integer part in column _c4

```
In [12]: from pyspark.sql.functions import regexp_extract
         from pyspark.sql.functions import col
         regex_string="\\d+"
         dataframe.select("_c4",regexp_extract(col("_c4"),regex_string,0).alias ("cleaned _c4")).show(5)
```

```
+-------+-----------+
|    _c4|cleaned _c4|
+-------+-----------+
|-213.25|        213|
| 457.81|        457|
|  46.71|         46|
|1198.97|       1198|
|  30.94|         30|
+-------+-----------+
```

regexp_replace: find the matching regex_string (string that contain regular expression) and replaces with the other given value.

syntax:
dataframe.select(regexp_replace(col("columnname"),regex_string,"").alias("cleaned")). show()

```
|_c0|                _c1|              _c2|_c3|   _c4|  _c5| _c6|   _c7|                _c8| _c9|
+---+-------------------+------------------+---+------+-----+----+------+-------------------+----+
|  1|Eldon Base for st...|Muhammed MacIntyre|  3|-213.25| 38.94|  35|Nunavut|Storage & Organiz...| 0.8|
|  2|"1.7 Cubic Foot C...|      Barry French|293| 457.81|208.16|68.02|Nunavut|         Appliances|0.58|
|  3|Cardinal Slant-D...|      Barry French|293|  46.71|  8.69| 2.99|Nunavut|Binders and Binde...|0.39|
|  4|               R380|     Clay Rozendal|483|1198.97|195.99| 3.99|Nunavut|Telephones and Co...|0.58|
|  5|Holmes HEPA Air P...|    Carlos Soltero|515|  30.94| 21.78| 5.94|Nunavut|         Appliances| 0.5|
+---+-------------------+------------------+---+------+-----+----+------+-------------------+----+
only showing top 5 rows
```

Regular exp for cleaning data by replacing with new value in column _c2

```
In [21]: from pyspark.sql.functions import regexp_replace
         from pyspark.sql.functions import col
         regex_string="Barry"
         dataframe.select("_c2",regexp_replace(col("_c2"),regex_string,"Arm").alias ("cleaned")).show(5)
```

```
+------------------+------------------+
|               _c2|           cleaned|
+------------------+------------------+
|Muhammed MacIntyre|Muhammed MacIntyre|
|      Barry French|        Arm French|
|      Barry French|        Arm French|
|     Clay Rozendal|     Clay Rozendal|
|    Carlos Soltero|    Carlos Soltero|
+------------------+------------------+
only showing top 5 rows
```

The cleaned column can also be added as a column to the existing dataframe using withColumn().

Reg Exp cleaned data adding in dataframe

```
In [23]: dataframe=spark.read.csv("C:/Spark/sample.csv")
         dataframe.show(5)
         dataframe=dataframe.withColumn("cleaned _c4",regexp_replace(col("_c4"),"\\-",""))
         dataframe.show(5)
```

```
+---+-------------------+------------------+---+------+------+-----+------+-------------------+----+
|_c0|                _c1|              _c2|_c3|   _c4|   _c5| _c6|   _c7|                _c8| _c9|
+---+-------------------+------------------+---+------+------+-----+------+-------------------+----+
|  1|Eldon Base for st...|Muhammed MacIntyre|  3|-213.25| 38.94|   35|Nunavut|Storage & Organiz...| 0.8|
|  2|"1.7 Cubic Foot C...|      Barry French|293| 457.81|208.16|68.02|Nunavut|         Appliances|0.58|
|  3|Cardinal Slant-D...|      Barry French|293|  46.71|  8.69| 2.99|Nunavut|Binders and Binde...|0.39|
|  4|               R380|     Clay Rozendal|483|1198.97|195.99| 3.99|Nunavut|Telephones and Co...|0.58|
|  5|Holmes HEPA Air P...|    Carlos Soltero|515|  30.94| 21.78| 5.94|Nunavut|         Appliances| 0.5|
+---+-------------------+------------------+---+------+------+-----+------+-------------------+----+
only showing top 5 rows

+---+-------------------+------------------+---+------+------+-----+------+-------------------+----+-----------+
|_c0|                _c1|              _c2|_c3|   _c4|   _c5| _c6|   _c7|                _c8| _c9|cleaned _c4|
+---+-------------------+------------------+---+------+------+-----+------+-------------------+----+-----------+
|  1|Eldon Base for st...|Muhammed MacIntyre|  3|-213.25| 38.94|   35|Nunavut|Storage & Organiz...| 0.8|     213.25|
|  2|"1.7 Cubic Foot C...|      Barry French|293| 457.81|208.16|68.02|Nunavut|         Appliances|0.58|     457.81|
|  3|Cardinal Slant-D...|      Barry French|293|  46.71|  8.69| 2.99|Nunavut|Binders and Binde...|0.39|      46.71|
|  4|               R380|     Clay Rozendal|483|1198.97|195.99| 3.99|Nunavut|Telephones and Co...|0.58|    1198.97|
|  5|Holmes HEPA Air P...|    Carlos Soltero|515|  30.94| 21.78| 5.94|Nunavut|         Appliances| 0.5|      30.94|
+---+-------------------+------------------+---+------+------+-----+------+-------------------+----+-----------+
```

Union To Merge All The Input Data

PySpark UNION is a transformation in PySpark that is used to merge two or more data frames in a PySpark application.
The union operation is applied to spark data frames with the same schema and structure. This is a very important condition for the union operation to be performed in any PySpark application.
The union operation can be carried out with two or more PySpark data frames and can be used to combine the data frame to get the defined result.
It returns a new Spark Data Frame that contains the union of rows of the data frames used.

The syntax for the PYSPARK UNION function is:
Df = df1.union(df2)
where,
Df = DataFrame post union.
Df1 = DataFrame1 used for union operation.
Df2 = DataFrame2 used for union operation.
.union :- The union transformation

The Union is a transformation in Spark that is used to work with multiple data frames in Spark. It takes the data frame as the input and the return type is a new data frame containing the elements that are in data frame1 as well as in data frame2.
This transformation takes out all the elements whether its duplicate or not and appends them making them into a single data frame for further operational purposes.
We can also apply the union operation to more than one data frame in a spark application.
PySpark UNION is a transformation in PySpark that is used to merge two or more data frames in a PySpark application.
The union operation is applied to spark data frames with the same schema and structure.
This is a very important condition for the union operation to be performed in any PySpark application.

The union operation can be carried out with two or more PySpark data frames and can be used to combine the data frame to get the defined result.
It returns a new Spark Data Frame that contains the union of rows of the data frames used.

The syntax for the PYSPARK UNION function is:
Df = df1.union(df2)
where,
Df = DataFrame post union.
Df1 = DataFrame1 used for union operation.
Df2 = DataFrame2 used for union operation.
.union :- The union transformation

The Union is a transformation in Spark that is used to work with multiple data frames in Spark. It takes the data frame as the input and the return type is a new data frame containing the elements that are in data frame1 as well as in data frame2.
This transformation takes out all the elements whether its duplicate or not and appends them making them into a single data frame for further operational purposes.
We can also apply the union operation to more than one data frame in a spark application.
import SparkContext, SparkConf, SparkSession from pyspark

```
In [1]: import findspark
        findspark.init()

In [2]: findspark.find()
        import pyspark
        findspark.find()

Out[2]: 'C:\\bigData\\spark'

In [3]: from pyspark import SparkContext,SparkConf
        from pyspark.sql import SparkSession

In [4]: conf=pyspark.SparkConf().setAppName('appName').setMaster('local')
        sc=pyspark.SparkContext.getOrCreate(conf=conf)
        spark=SparkSession(sc)
```

1. To merge two data frame having having same schema and structure. The output will append both the data frames together and the result will have both the data Frames together.

```
In [7]: a= spark.createDataFrame(["SAM","JOHN","AND","ROBIN","ANAND"], "string").toDF("Name")

In [8]: b= spark.createDataFrame(["DAN","JACK","AND"], "string").toDF("Name")

In [9]: c = a.union(b).show()

+-----+
| Name|
+-----+
|  SAM|
| JOHN|
|  AND|
|ROBIN|
|ANAND|
|  DAN|
| JACK|
|  AND|
+-----+
```

2. To perform multiple union operations over the PySpark Data Frame. The same union operation can be applied to all the data frames to merge more than two data frame having same schema and structure.

```
In [15]: c= spark.createDataFrame(["DAtN","JACKj","AND"], "string").toDF("Name")

In [16]: d=a.union(b).union(c)

In [18]: d.show()

+-----+
| Name|
+-----+
|  SAM|
| JOHN|
|  AND|
|ROBIN|
|ANAND|
|  DAN|
| JACK|
|  AND|
| DAtN|
|JACKj|
|  AND|
+-----+
```

3. The same union operation can be done with Data Frame with Integer type as the data type also.

```
In [20]: c= spark.createDataFrame([23,98,67], "integer").toDF("Number")
         b= spark.createDataFrame([22,12,12], "integer").toDF("Number1")
         d = c.union(b)
         d.show()
```

```
+------+
|Number|
+------+
|    23|
|    98|
|    67|
|    22|
|    12|
|    12|
+------+
```

```
In [21]: d.distinct().show()
```

```
+------+
|Number|
+------+
|    12|
|    22|
|    23|
|    98|
|    67|
```

4. Another way to merge more than two dataframe having the same schema and structure is by unionAll() function in combination with the reduce() function from the functools module.

```
In [5]: # import modules
        from functools import reduce
```

```
In [6]: from pyspark.sql import DataFrame
```

```
In [8]: df1 = sc.parallelize([[1., 'age 18-25']])
        df2 = sc.parallelize([[2., 'age 26-30']])
        df3 = sc.parallelize([[3., 'age 31-35']])
```

```
In [9]: df1=df1.toDF(["f1","age"])
        df1.show()
```

```
+---+---------+
| f1|      age|
+---+---------+
|1.0|age 18-25|
+---+---------+
```

```
In [10]: df2=df2.toDF(["f1", "age"])
         df2.show()
```

```
+---+---------+
| f1|      age|
+---+---------+
|2.0|age 26-30|
+---+---------+
```

```
In [11]: df3=df3.toDF(["f1", "age"])
         df3.show()
```

```
+---+---------+
| f1|      age|
+---+---------+
|3.0|age 31-35|
+---+---------+
```

5. reduce() takes two arguments, a function and the input arguments for the function. Instead of two input arguments, we can provide a list.
   In this case, reduce will apply the function subsequently to the list.

```
In [12]: dfs = [df1, df2, df3]

In [13]: df_complete = reduce(DataFrame.unionAll, dfs)

In [14]: df_complete.collect()

Out[14]: [Row(f1=1.0, age='age 18-25'),
          Row(f1=2.0, age='age 26-30'),
          Row(f1=3.0, age='age 31-35')]
```

There are 3 ways to merge two dataframe having different schema.

1.withColumn,Union
2.Outer Join
3.automate the process

1. withColumn,Union

import lit from pyspark.sql.functions. Dataframe with missing column has to be filled with null. After having two dataframe with same schema use union function.

withColumn,Union

```
In [48]: from pyspark.sql.functions import lit

In [49]: df1 = sc.parallelize([[1., 'age 18-25','mite']])
         df2 = sc.parallelize([[3., 'age 31-35']])

In [51]: df1=df1.toDF(["f1","age","college"])
         df1.show()
+---+---------+-------+
| f1|      age|college|
+---+---------+-------+
|1.0|age 18-25|   mite|
+---+---------+-------+

In [53]: df2=df2.toDF(["f1", "age"])
         df2.show()
+---+---------+
| f1|      age|
+---+---------+
|3.0|age 31-35|
+---+---------+

In [57]: df3=df2.withColumn("college",lit("null")).show()
+---+---------+-------+
| f1|      age|college|
+---+---------+-------+
|3.0|age 31-35|   null|
+---+---------+-------+

In [56]: df3.union(df1).show()
+---+---------+-------+
| f1|      age|college|
+---+---------+-------+
|3.0|age 31-35|   null|
|1.0|age 18-25|   mite|
+---+---------+-------+
```

2. Outer Join

By using the outer join it will automatically fill null values to the missing column.

Outer Join

```
In [97]: df_output=df1.join(df2,on=["f1","age"],how="Outer")

In [98]: df_output.show()
```

```
+---+---------+-------+
| f1|      age|college|
+---+---------+-------+
|3.0|age 31-35|   null|
|1.0|age 18-25|   mite|
+---+---------+-------+
```

3. Automate the process

This is the best way to merge two dataframe having different schema.
It will take set of columns and makes a list of missing columns and each missing column will be filled with null values. After having same schema union function is applied on both dataframe.

automate the process

```
In [100]: listA=list(set(df1.columns)-set(df2.columns))
          listB=list(set(df2.columns)-set(df1.columns))

In [102]: for i in listA:
              df2=df2.withColumn(i,lit("null"))

          for i in listB:
              df1=df1.withColumn(i,lit("null"))

In [104]: df1.union(df2).show()
```

```
+---+---------+-------+
| f1|      age|college|
+---+---------+-------+
|1.0|age 18-25|   mite|
|3.0|age 31-35|   null|
+---+---------+-------+
```

Print The Result In Console And Databrick Platform

1. Printing the result in console

To open scala in command prompt, we use
>>>spark-shell
In scala we use scala programming language to create variable, were variable is prefixed with val keyword. We can perform the same operation with the same command in console also that we have used in jupyter.

```
scala> val x=sc.parallelize(Array(1,2,3))
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24

scala> val y=sc.parallelize(Array(6,3,5))
y: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[1] at parallelize at <console>:24

scala> val z=x.union(y)
z: org.apache.spark.rdd.RDD[Int] = UnionRDD[2] at union at <console>:27

scala> z.collect.foreach(println)
1
2
3
6
3
5

scala> val p=z.distinct()
p: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[5] at distinct at <console>:25

scala> p.collect.foreach(println)
1
2
3
5
6
```

2. Print the result in databrick platform

In databrick platform create a cluster, then in the notebook create a dataframe then perform the action on that dataframe.

```
df1 = sc.parallelize([[1., 'age 18-25','mite']])
df2 = sc.parallelize([[2., 'age 26-30',82]])
df3 = sc.parallelize([[3., 'age 31-35']])
```

Command took 0.21 seconds -- by amithahegde98@gmail.com at 5/14/2021, 10:24:32 AM on demo

Cmd 3

```
df1=df1.toDF(["f1","age","college"])
df1.show()
```

▸ (6) Spark Jobs

▸ df1: pyspark.sql.dataframe.DataFrame = [f1: double, age: string ... 1 more fields]

```
+---+---------+-------+
| f1|      age|college|
+---+---------+-------+
|1.0|age 18-25|   mite|
+---+---------+-------+
```

```
df2=df2.toDF(["f1", "age","marks"])
df2.show()
```

▸ (6) Spark Jobs

▸ df2: pyspark.sql.dataframe.DataFrame = [f1: double, age: string ... 1 more fields]

```
+---+---------+-----+
| f1|      age|marks|
+---+---------+-----+
|2.0|age 26-30|   82|
+---+---------+-----+
```

Command took 1.15 seconds -- by amithahegde98@gmail.com at 4/22/2021, 1:53:56 PM on spark_demo

d 5

```
df3=df3.toDF(["f1", "age"])
df3.show()
```

▸ (6) Spark Jobs

▸ df3: pyspark.sql.dataframe.DataFrame = [f1: double, age: string]

```
+---+---------+
| f1|      age|
+---+---------+
|3.0|age 31-35|
+---+---------+
```

Here in the following screenshot the union function on two dataframes is shown. The databricks not only displays output on the console but it also provides a feature that help in visualizing the result of the query in the graphical manner.

```
1  #from pyspark.sql.functions import lit
2  df3_add=df3.withColumn("college",lit("null"))
```

▸ df3_add: pyspark.sql.dataframe.DataFrame = [f1: double, age: string ... 1 more fields]

Command took 0.10 seconds -- by amithahegde98@gmail.com at 4/22/2021, 1:54:40 PM on spark_demo

Cmd 7

```
1  df3_add.union(df1).show()
```

▸ (3) Spark Jobs

```
+---+---------+-------+
| f1|      age|college|
+---+---------+-------+
|3.0|age 31-35|   null|
|1.0|age 18-25|   mite|
+---+---------+-------+
```

Ability to Aggregate and Summarizing Data into Useful Reports

Aggregate Function

Pyspark provides built-in standard Aggregate functions defines in Dataframe API, these comes in handy when we need to make aggregate operations on Dataframe columns.
Aggregate functions operate on a group of rows and calculate a single return value for every group. Some of the aggregate functions are avg, count, max, mean, min, sum.

create a data frame by taking input from csv file

```
In [1]: from pyspark.sql import SparkSession

In [2]: spark = SparkSession.builder.appName("groupbyagg").getOrCreate()

In [29]: df = spark.read.csv(r'C:\Users\Lenovo\Downloads\sample1.csv', inferSchema=True, header=True)

In [31]: #Showing the data
         df.show()
```

```
+--------------------+----------+---------+---------------+----------+--------------------+-------------+------------+-------
-----+------------+--------------------+-------+-----+--------------------+------+
|           User Name|First Name|Last Name|   Display Name| Job Title|          Department|Office Number|Office Phone|Mobile
Phone|         Fax|             Address|   City|State or Province|ZIP or Postal Code|Country or Region|_c15|salary|
+--------------------+----------+---------+---------------+----------+--------------------+-------------+------------+-------
-----+------------+--------------------+-------+-----+--------------------+------+
|  chris@contoso.com|     Chris|    Green|    Chris Green|IT Manager|Information Techn...|       123451|123-555-1211|123-555
-6641|123-555-9821|1 Microsoft way|Redmond|              Wa|               98052|    United States|null| 10000|
|    ben@contoso.com|       Ben|  Andrews|    Ben Andrews|IT Manager|Information Techn...|       123452|123-555-1212|123-555
-6642|123-555-9822|1 Microsoft way|Redmond|              Wa|               98052|    United States|null| 32000|
|  david@contoso.com|     David| Longmuir| David Longmuir|IT Manager|Information Techn...|       123453|123-555-1213|123-555
-6643|123-555-9823|1 Microsoft way|Redmond|              Wa|               98052|    United States|null| 20000|
|cynthia@contoso.com|   Cynthia|    Carey|  Cynthia Carey|IT Manager|Information Techn...|       123454|123-555-1214|123-555
-6644|123-555-9824|1 Microsoft way|Redmond|              Wa|               98052|    United States|null| 35000|
|melissa@contoso.com|   Melissa|  MacBeth|Melissa MacBeth|IT Manager|Information Techn...|       123455|123-555-1215|123-555
-6645|123-555-9825|1 Microsoft way|Redmond|              Wa|               98052|    United States|null| 60000|
+--------------------+----------+---------+---------------+----------+--------------------+-------------+------------+-------
-----+------------+--------------------+-------+-----+--------------------+------+
```

Usage of group by function on a dataframe and apply aggregate function on it.
Syntax for aggregate function is:-
Dataframe.agg({‘column’:’aggregate function’})

aggregate function

agg: avg, count, max, mean, min, sum

```
In [43]: # Sum by Agg
         group_data = df.groupBy("City")
         group_data.agg({'salary':'sum'}).show()
```

```
+-------+-----------+
|   City|sum(salary)|
+-------+-----------+
|Redmond|     157000|
+-------+-----------+
```

```
In [44]: # Max by agg
         group_data.agg({'salary':'max'}).show()
```

```
+-------+-----------+
|   City|max(salary)|
+-------+-----------+
|Redmond|      60000|
+-------+-----------+
```

Usage of aggregate function directly on dataframe.

In [45]:
```
# Sum
df.agg({'salary':'sum'}).show()
```

```
+-----------+
|sum(salary)|
+-----------+
|     157000|
+-----------+
```

In [46]:
```
df.agg({'salary':'max'}).show()
```

```
+-----------+
|max(salary)|
+-----------+
|      60000|
+-----------+
```

In [47]:
```
df.agg({'salary':'avg'}).show()
```

```
+-----------+
|avg(salary)|
+-----------+
|    31400.0|
+-----------+
```