PySpark Demonstration on Windows 11

Md Mehedi Hassan

ID: 25542607009

Introduction

Apache Spark is an open-source distributed computing engine designed for fast, large-scale data processing where PySpark is the Python API for Apache Spark. Key features of PySpark include – fault tolerance, reliability, extensive libraries for machine learning, graph processing, provides APIs for Python, Java, Scala and R. Prerequisites to learn the framework are to have familiarity with big data or distributed computing concepts, programming fundamentals and obviously, the knowledge of Python.

Installation and Setup

1) Install Java Development Kit (JDK):

Spark runs on Java 8,11 or 17 till date. Download Windows x64 Installer JDK-17. Install and finally keep the file at "C:\jdk17" location as shown in Fig. 1.

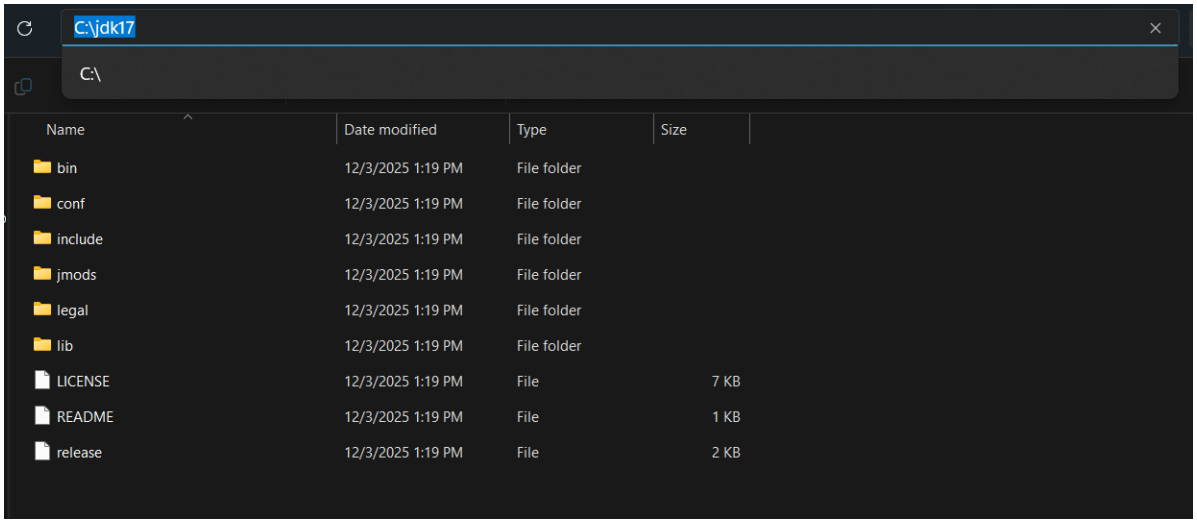

Fig. 1. Jdk17 bin folder location

Make sure the bin folder is directly under this directory as shown in Fig. 1.

Now go to the search bar of windows and type environment variables, click on edit environment variables, then go to the environment variables as shown in Fig. 2. Create Variable name 'JAVA_HOME' and value 'C:\jdk17' in the 'System Variables' section below.

Afterwards, go to the 'Path' variable in the 'System Variables' (if absent then create it manually) and add 'C:\jdk17\bin'. Finally check there is no other jdk related paths or values or anything inside the whole 'System Variables' section.

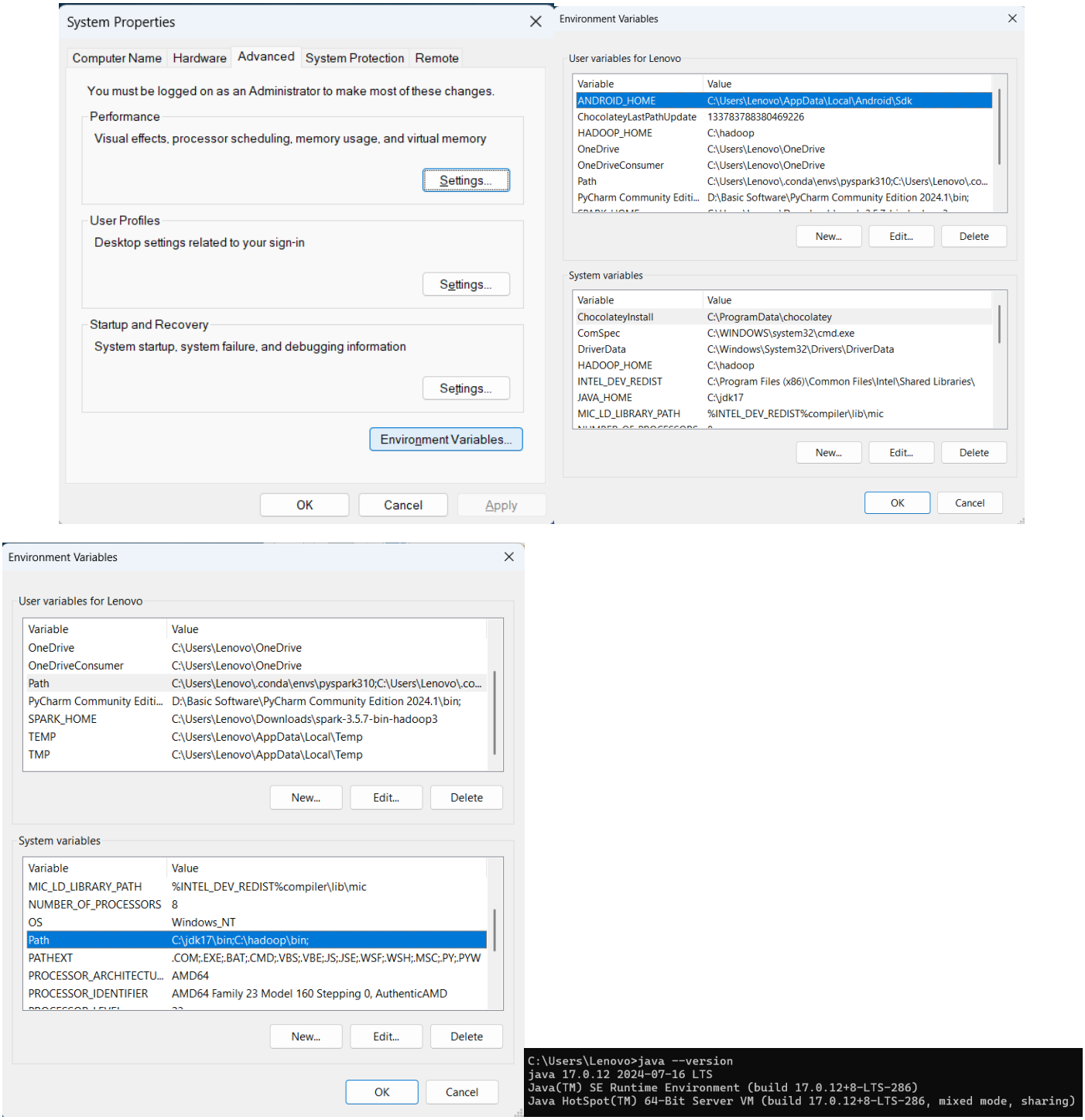

Fig. 2. Jdk17 setup with System Variables

Now go to command prompt and type ‘java --version' to check if the java 17 is ready for use.

2) Install Apache Spark:

Spark-3.x (i.e., 3.5.7) should be installed locally on the PC. Go to the link https://spark.apache.org/downloads.html and select 3.5.7 from option 1. Then select package type Apache Hadoop 3.3 and later as shown in Fig. 3. Finally click on option 3 and download ‘.tgz’ file. Extract the file locally on PC.

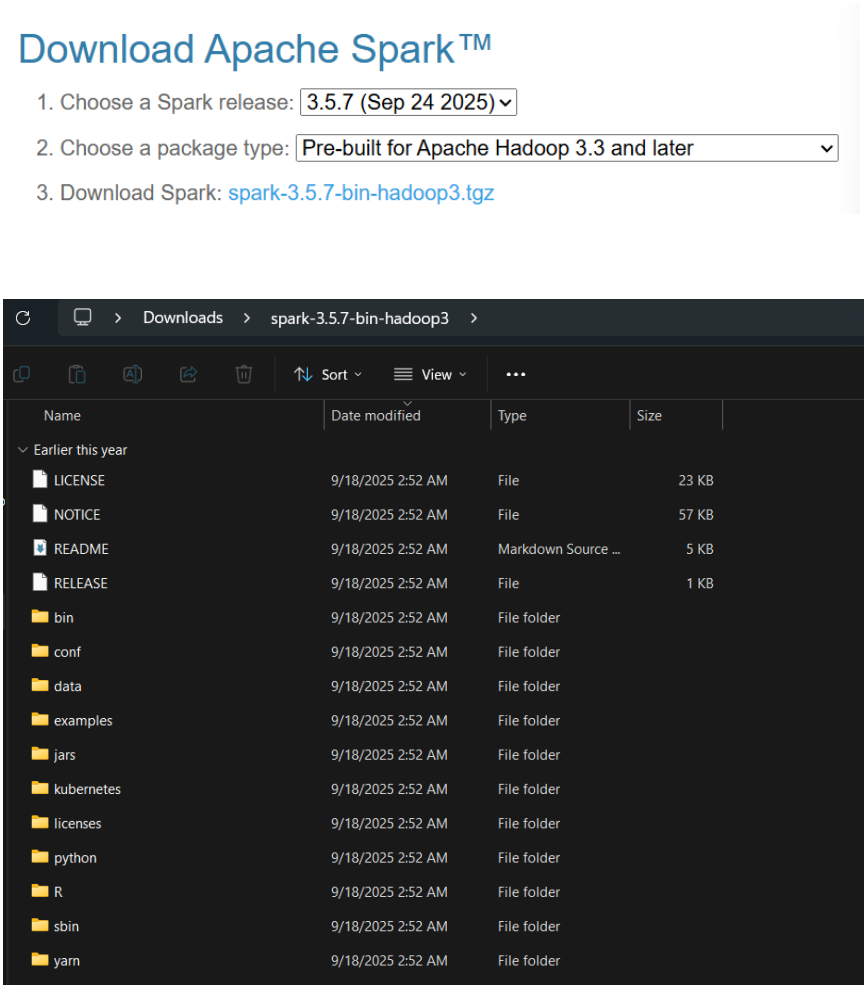

Fig. 3. Spark 3.5.7 download and overview after extraction

Additionally, install pyspark from python terminal with ‘pip install pyspark’. Now, check the version of the spark from command prompt writing ‘pyspark --version' as shown in Fig. 4.

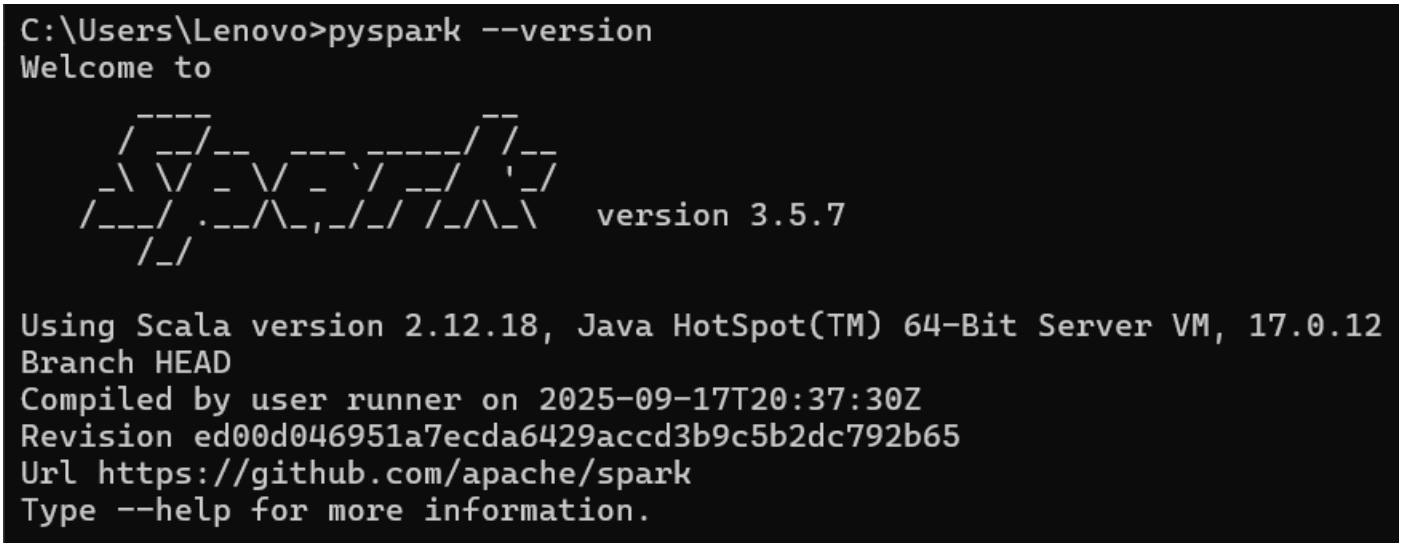

Fig. 4. Pyspark version checking

3) hadoop.dll and winutils.exe setup:

Spark cannot be executed on Windows as it is not completely built compatible. To solve any kind of issues, two files need to be present named hadoop.dll and winutils.exe.

Go to C Drive and manually create a folder named hadoop under which create another folder named bin.

Now, go to https://github.com/cdarlint/winutils/tree/master/hadoop-3.3.6/bin and download hadoop.dll and winutils.exe file. Keep them under the "C:\hadoop\bin" directory that was manually created. Finally, the files seem as Fig. 5.

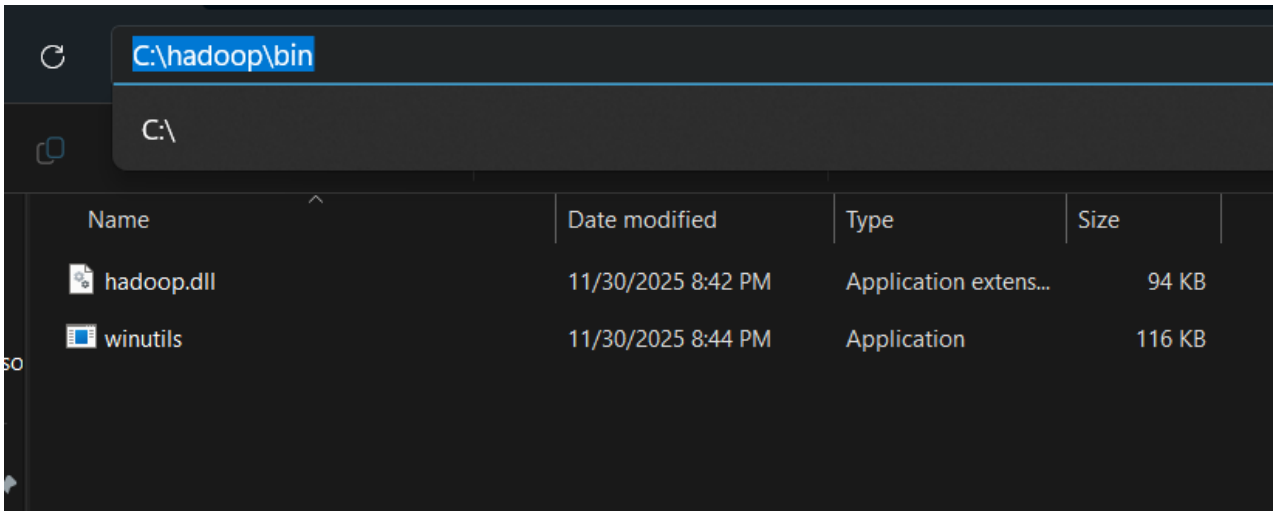

Fig. 5. Location of hadoop.dll and winutils.exe

Again, go to 'System Variables' and create a variable name 'HADOOP_HOME' and value 'C:\hadoop'.

Then go to Path (where jdk is already there) and add 'C:\hadoop\bin'. Ensure there is no other hadoop or anything like this at anywhere else in the system variables.

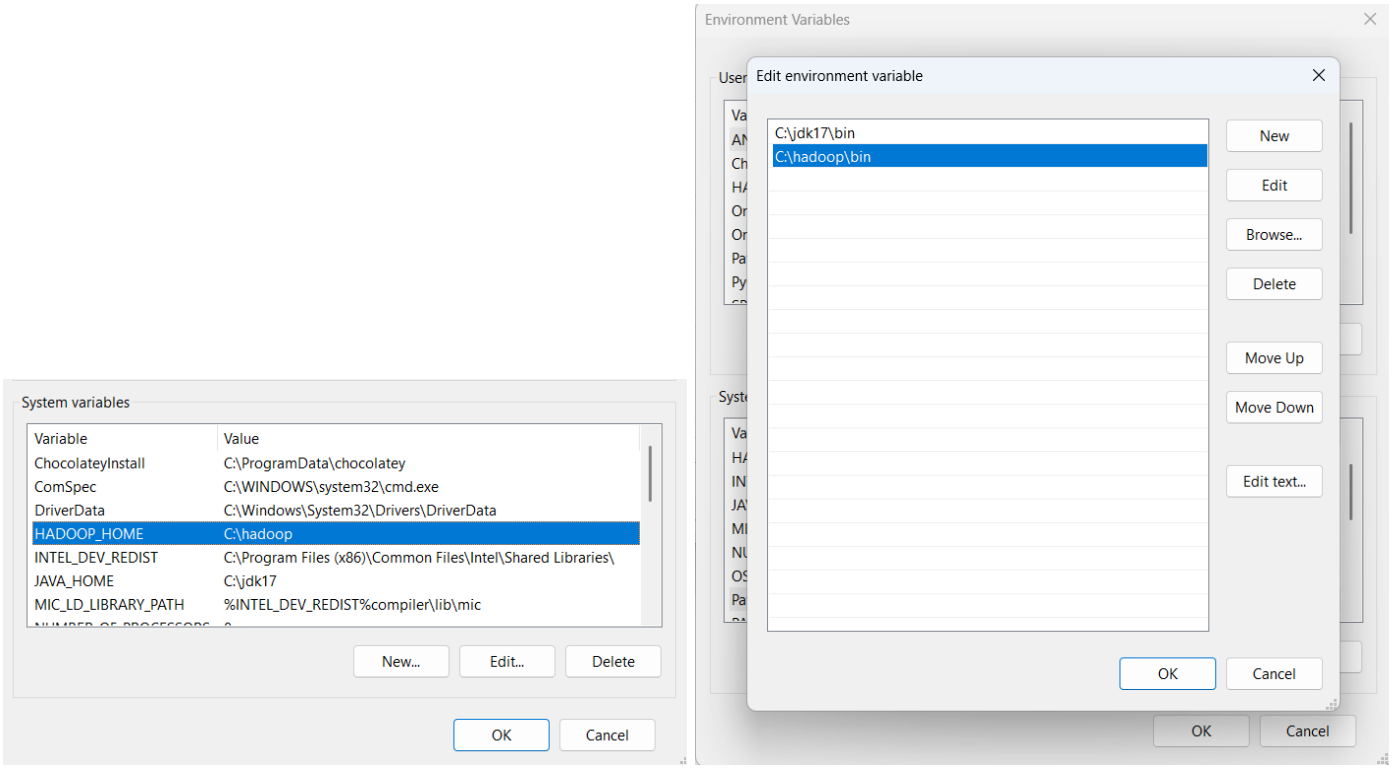

Fig. 6. Addition of hadoop to system variables

4) Python compatibility:

Not all Python versions can be used for spark. To our case, Python 3.10.x version works and it is better to create a virtual environment always. We setup 'Anaconda' and create conda virtual environment named pyspark310.

Go to anaconda_prompt under the anaconda navigator. Inside the prompt write “conda create -n pyspark310 python=3.10” and it will create a virtual conda environment named pyspark310 with Python version 3.10. Remember, the virtual environment (venv) name should be ‘pyspark310’ and pyspark can easily recognize the Python avoiding any missing error.

Now, activate venv from conda prompt with “conda activate pyspark310”

Then install pyspark and findspark from the same conda prompt with “pip install pyspark” and “pip install findspark”.

Close everything and reopen conda prompt and check conda environment pyspark310 exists or not. For this, type ‘conda info –envs’.

```
(base) C:\Users\Lenovo>conda info --envs

# conda environments:
#
pyspark310               C:\Users\Lenovo\.conda\envs\pyspark310
base                  * D:\anaconda
```

Check the python version from inside pyspark310 with ‘conda activate pyspark310’ and ‘python --version'.

```
(base) C:\Users\Lenovo>python --version
Python 3.13.5

(base) C:\Users\Lenovo>conda activate pyspark310

(pyspark310) C:\Users\Lenovo>python --version
Python 3.10.19

(pyspark310) C:\Users\Lenovo>
```

See, we have base python version 3.13.5 but inside pyspark310 virtual environment it is 3.10.19 and this is how the environment removes unwanted incompatibilities.

Code on VsCode:

Initialize:

We can run the pyspark locally now on our PC with VsCode.

```
import os

os.environ["PYSPARK_PYTHON"] = r"C:\Users\Lenovo\.conda\envs\pyspark310\python.exe"
os.environ["PYSPARK_DRIVER_PYTHON"] = r"C:\Users\Lenovo\.conda\envs\pyspark310\python.exe"
```

For pyspark’s easier navigation, we need to set pyspark python and driver both at the conda environment’s python.exe directory as shown.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Test") \
    .master("local[*]") \
    .getOrCreate()
```

.appName() is used to set the name of the spark app.

.master() defines what manager and executor mode are we going to use. So, local[] means run locally on PC, not on any cluster and “*” means to use all available CPU cores.

```
# Print Spark Web UI URL
print("\nSpark Web UI running at:")
print(spark.sparkContext.uiWebUrl, "\n")
```

✓ 0.0s

```
Spark Web UI running at:
http://host.docker.internal:4040
```

Spark provided default web UI to observe the background functionalities and detailed history.

RDD Actions

Resilient Data Distribution has few well known action functions.

```
rdd = spark.sparkContext.parallelize([1,2,3,4,5])
print(rdd.count())
```

✓ 16.5s

```
5
```

The count function returns the number of elements in an rdd.

```
data=[("Alice",27),("Bob",33),("Charlie",24),("Alice",32)]
rdd=spark.sparkContext.parallelize(data)
```

✓ 0.0s

```
#print all elements of data
print(rdd.collect())
```

✓ 0.1s

```
[('Alice', 27), ('Bob', 33), ('Charlie', 24), ('Alice', 32)]
```

The collect function returns all the elements inside data or rdd.

```
first=rdd.first()
print(first)
```

✓ 10.6s

```
('Alice', 27)
```

The first function returns the first element of the rdd.

```
taken_elements=rdd.take(2)
print(taken_elements)
```

✓ 10.3s

```
[('Alice', 27), ('Bob', 33)]
```

This returns first two elements by ‘take’ function.

RDD Transformation

Different transformations on data can be performed by rdd transformation functions.

```
#convert name to uppercase
mapped_rdd=rdd.map(lambda x: (x[0].upper(),x[1]))
result=mapped_rdd.collect()
print(result)
```

✓ 16.0s

```
[('ALICE', 27), ('BOB', 33), ('CHARLIE', 24), ('ALICE', 32)]
```

Map can be used for mapping data in key value pairs, then retrieve it with collect().

```
#filter records where age is greater than 30
filtered_rdd=rdd.filter(lambda x: x[1]> 30)
filtered_rdd.collect()
```

✓ 15.8s

```
[('Bob', 33), ('Alice', 32)]
```

filter function can be utilized to filter data with a certain condition. Here, all data record with age>30 is found.

```
#sorting
sorted_rdd=rdd.sortBy(lambda x: x[1],ascending=False)
sorted_rdd.collect()
```

✓ 1m 3.2s

```
[('Bob', 33), ('Alice', 32), ('Alice', 27), ('Charlie', 24)]
```

sortBy function is used to sort the data as per age in descending order.

DataFrame

```
#read txt file and count the number of occurance of words
#Using RDD
rdd=spark.sparkContext.textFile("long-doc.txt")
result_rdd=rdd.flatMap(lambda line: line.split(" ")) \
.map(lambda word: (word,1)) \
.reduceByKey(lambda a,b: a+b)\
.sortBy(lambda x: x[1],ascending=False)

result_rdd.take(10)
```

[24] ✓ 18.6s

```
[('Lorem', 2000),
 ('ipsum', 2000),
 ('consectetur', 2000),
 ('elit.', 2000),
 ('dolor', 2000),
 ('sit', 2000),
 ('amet,', 2000),
 ('adipiscing', 2000),
 ('', 2),
 ('sample', 2)]
```

A text data was uploaded and counted repeated numbers with help of reduceByKey but this does not provide dataframe structure. Below is the code for dataframe.

```
#using DataFrame
df=spark.read.text("long-doc.txt")

result_df=df.selectExpr("explode(split(value,' ')) as word") \
.groupBy("word").count().orderBy(desc("count"))

result_df.take(10)
```

[26] ✓ 6.1s

```
[Row(word='sit', count=2000),
 Row(word='consectetur', count=2000),
 Row(word='dolor', count=2000),
 Row(word='Lorem', count=2000),
 Row(word='amet,', count=2000),
 Row(word='ipsum', count=2000),
 Row(word='elit.', count=2000),
 Row(word='adipiscing', count=2000),
 Row(word='sample', count=2),
 Row(word='', count=2)]
```

This is the dataframe structure by spark.read.text. However, let us proceed with a real csv file traditionally as we do with numpy or pandas.

```
path="StudentPerformanceFactors.csv"
df=spark.read.csv(path,header=True)
```

[29] ✓ 2.8s

```
df.printSchema()

df.show(5)
```

[30] ✓ 0.5s

```
root
 |-- Hours_Studied: string (nullable = true)
 |-- Attendance: string (nullable = true)
 |-- Parental_Involvement: string (nullable = true)
 |-- Access_to_Resources: string (nullable = true)
 |-- Extracurricular_Activities: string (nullable = true)
 |-- Sleep_Hours: string (nullable = true)
 |-- Previous_Scores: string (nullable = true)
 |-- Motivation_Level: string (nullable = true)
 |-- Internet_Access: string (nullable = true)
 |-- Tutoring_Sessions: string (nullable = true)
 |-- Family_Income: string (nullable = true)
 |-- Teacher_Quality: string (nullable = true)
 |-- School_Type: string (nullable = true)
 |-- Peer_Influence: string (nullable = true)
 |-- Physical_Activity: string (nullable = true)
 |-- Learning_Disabilities: string (nullable = true)
 |-- Parental_Education_Level: string (nullable = true)
 |-- Distance_from_Home: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Exam_Score: string (nullable = true)

+-------------+----------+--------------------+-------------------+--------------------------+-----------+---------------+----------------+----
|Hours_Studied|Attendance|Parental_Involvement|Access_to_Resources|Extracurricular_Activities|Sleep_Hours|Previous_Scores|Motivation_Level|Inte
+-------------+----------+--------------------+-------------------+--------------------------+-----------+---------------+----------------+----
|           23|        84|                 Low|               High|                        No|          7|             73|             Low|
|           19|        64|                 Low|             Medium|                        No|          8|             59|             Low|
|           24|        98|              Medium|             Medium|                       Yes|          7|             91|          Medium|
|           29|        89|                 Low|             Medium|                       Yes|          8|             98|          Medium|
|           19|        92|              Medium|             Medium|                       Yes|          6|             65|          Medium|
+-------------+----------+--------------------+-------------------+--------------------------+-----------+---------------+----------------+----
only showing top 5 rows
```

But, this method sets all the attributes to string. Let us redefine and rectify the typecasting problem.

At first, ‘from pyspark.sql.types import *’ imports all types of sql types of pyspark.

```
schema = StructType([
    StructField("Hours_Studied", IntegerType(), True),
    StructField("Attendance", IntegerType(), True),
    StructField("Parental_Involvement", StringType(), True),
    StructField("Access_to_Resources", StringType(), True),
    StructField("Extracurricular_Activities", StringType(), True),
    StructField("Sleep_Hours", IntegerType(), True),
    StructField("Previous_Scores", IntegerType(), True),
    StructField("Motivation_Level", StringType(), True),
    StructField("Internet_Access", StringType(), True),
    StructField("Tutoring_Sessions", IntegerType(), True),
    StructField("Family_Income", StringType(), True),
    StructField("Teacher_Quality", StringType(), True),
    StructField("School_Type", StringType(), True),
    StructField("Peer_Influence", StringType(), True),
    StructField("Physical_Activity", IntegerType(), True),
    StructField("Learning_Disabilities", StringType(), True),
    StructField("Parental_Education_Level", StringType(), True),
    StructField("Distance_from_Home", StringType(), True),
    StructField("Gender", StringType(), True),
    StructField("Exam_Score", IntegerType(), True)
])
```

[32] ✓ 0.0s

```
[33]
df=spark.read.csv(path,header=True,schema=schema)
✓ 0.0s
```

```
[34]
#again display with proper datatypes
df.printSchema()

df.show(5)
✓ 0.5s
```

```
root
 |-- Hours_Studied: integer (nullable = true)
 |-- Attendance: integer (nullable = true)
 |-- Parental_Involvement: string (nullable = true)
 |-- Access_to_Resources: string (nullable = true)
 |-- Extracurricular_Activities: string (nullable = true)
 |-- Sleep_Hours: integer (nullable = true)
 |-- Previous_Scores: integer (nullable = true)
 |-- Motivation_Level: string (nullable = true)
 |-- Internet_Access: string (nullable = true)
 |-- Tutoring_Sessions: integer (nullable = true)
 |-- Family_Income: string (nullable = true)
 |-- Teacher_Quality: string (nullable = true)
 |-- School_Type: string (nullable = true)
 |-- Peer_Influence: string (nullable = true)
```

Thus, we can manually set the feature types. But more interesting solution is there where inferSchema method automatically guesses the type of the features and cast them with those types.

```
[35]
#automatically guess the data type of the columns
df=spark.read.csv(path,header=True,inferSchema=True)
df.printSchema()
df.show(5)
✓ 0.8s
```

```
root
 |-- Hours_Studied: integer (nullable = true)
 |-- Attendance: integer (nullable = true)
 |-- Parental_Involvement: string (nullable = true)
 |-- Access_to_Resources: string (nullable = true)
 |-- Extracurricular_Activities: string (nullable = true)
 |-- Sleep_Hours: integer (nullable = true)
 |-- Previous_Scores: integer (nullable = true)
 |-- Motivation_Level: string (nullable = true)
 |-- Internet_Access: string (nullable = true)
 |-- Tutoring_Sessions: integer (nullable = true)
 |-- Family_Income: string (nullable = true)
 |-- Teacher_Quality: string (nullable = true)
 |-- School_Type: string (nullable = true)
 |-- Peer_Influence: string (nullable = true)
 |-- Physical_Activity: integer (nullable = true)
 |-- Learning_Disabilities: string (nullable = true)
 |-- Parental_Education_Level: string (nullable = true)
 |-- Distance_from_Home: string (nullable = true)
 |-- Gender: string (nullable = true)
```

Thus, the data types can be automatically casted.

```
[36]
selected_columns=df.select("Sleep_Hours","Attendance")
selected_columns.show(10)
✓ 0.2s
```

```
+-----------+----------+
|Sleep_Hours|Attendance|
+-----------+----------+
|          7|        84|
|          8|        64|
|          7|        98|
|          8|        89|
|          6|        92|
|          8|        88|
|          7|        84|
|          6|        78|
|          6|        94|
|          8|        98|
+-----------+----------+
only showing top 10 rows
```

Individual columns can be viewed with help of pyspark.

```
1 #filtering based on condition on columns
2 filtered_data=df.filter(df.Hours_Studied >20)
3 print(filtered_data.count())
4 filtered_data.show(10)
```

[37] ✓ 0.7s Python

```
3063
+-------------+----------+--------------------+-----------------+-----------------------+-----------+---------------+----------------+---------------+-------
|Hours_Studied|Attendance|Parental_Involvement|Access_to_Resources|Extracurricular_Activities|Sleep_Hours|Previous_Scores|Motivation_Level|Internet_Access|Tutorin
+-------------+----------+--------------------+-----------------+-----------------------+-----------+---------------+----------------+---------------+-------
|           23|        84|                 Low|             High|                     No|          7|             73|             Low|            Yes|
|           24|        98|              Medium|           Medium|                    Yes|          7|             91|          Medium|            Yes|
|           29|        89|                 Low|           Medium|                    Yes|          8|             98|          Medium|            Yes|
|           29|        84|              Medium|              Low|                    Yes|          7|             68|             Low|            Yes|
|           25|        78|                 Low|             High|                    Yes|          6|             50|          Medium|            Yes|
|           23|        98|              Medium|           Medium|                    Yes|          8|             71|          Medium|            Yes|
|           21|        83|              Medium|           Medium|                    Yes|          8|             97|             Low|            Yes|
|           22|        70|                 Low|           Medium|                    Yes|          6|             82|          Medium|            Yes|
|           29|        78|              Medium|           Medium|                     No|          5|             99|            High|            Yes|
|           21|        62|                High|              Low|                    Yes|          6|             54|            High|            Yes|
+-------------+----------+--------------------+-----------------+-----------------------+-----------+---------------+----------------+---------------+-------
only showing top 10 rows
```

Dataframe is filtered and showed records with hours_studied>20

```
1 #sorting
2 sorted_data=df.orderBy("Exam_Score")
3 sorted_data.show(10)
```

[38] ✓ 0.3s Python

```
----+-------------+---------------+----------+-------------+-----------------+--------------------+-----------------------+-----------------+------+----------+
ions|Family_Income|Teacher_Quality|School_Type|Peer_Influence|Physical_Activity|Learning_Disabilities|Parental_Education_Level|Distance_from_Home|Gender|Exam_Score|
----+-------------+---------------+----------+-------------+-----------------+--------------------+-----------------------+-----------------+------+----------+
   1|          Low|         Medium|    Public|     Negative|                3|                  No|            High School|             Near|  Male|        55|
   0|          Low|         Medium|   Private|     Negative|                2|                  No|                College|              Far|  Male|        56|
   0|          Low|         Medium|    Public|     Negative|                2|                 Yes|                College|         Moderate|  Male|        57|
   0|          Low|         Medium|    Public|      Neutral|                4|                  No|            High School|              Far|Female|        57|
   0|       Medium|         Medium|   Private|     Negative|                3|                  No|            High School|             Near|  Male|        57|
   1|          Low|         Medium|    Public|     Positive|                1|                  No|            Postgraduate|             Near|Female|        57|
   3|         High|           NULL|    Public|      Neutral|                2|                  No|            High School|              Far|Female|        58|
   2|       Medium|           High|   Private|     Negative|                3|                  No|            High School|         Moderate|Female|        58|
   0|          Low|         Medium|    Public|      Neutral|                0|                  No|            High School|             Near|Female|        58|
   1|          Low|         Medium|   Private|     Positive|                2|                  No|                College|         Moderate|  Male|        58|
----+-------------+---------------+----------+-------------+-----------------+--------------------+-----------------------+-----------------+------+----------+
```

We can also sort the data basing on single feature. For example, Exam_Score was sorted in ascending order and whole dataframe also shuffled.

```
1 #distinct
2 distnct_rows=df.select("Teacher_Quality").distinct()
3 distnct_rows.show()
```

[39] ✓ 0.5s

```
+---------------+
|Teacher_Quality|
+---------------+
|           High|
|            Low|
|         Medium|
|           NULL|
+---------------+
```

Unique values can be showed of a certain feature with ‘distinct()’ method.

```
1 #creating new column
2 new_col=df.withColumn("NEW_SCORE",df.Previous_Scores+df.Exam_Score)
3 new_col.show(10)
```

[41] ✓ 0.3s Python

```
-------+---------------+----------+-------------+-----------------+--------------------+-----------------------+-----------------+------+----------+---------+
/_Income|Teacher_Quality|School_Type|Peer_Influence|Physical_Activity|Learning_Disabilities|Parental_Education_Level|Distance_from_Home|Gender|Exam_Score|NEW_SCORE|
-------+---------------+----------+-------------+-----------------+--------------------+-----------------------+-----------------+------+----------+---------+
     Low|         Medium|    Public|     Positive|                3|                  No|            High School|             Near|  Male|        67|      140|
  Medium|         Medium|    Public|     Negative|                4|                  No|                College|         Moderate|Female|        61|      120|
  Medium|         Medium|    Public|      Neutral|                4|                  No|            Postgraduate|             Near|  Male|        74|      165|
  Medium|         Medium|    Public|     Negative|                4|                  No|            High School|         Moderate|  Male|        71|      169|
  Medium|           High|    Public|      Neutral|                4|                  No|                College|             Near|Female|        70|      135|
  Medium|         Medium|    Public|     Positive|                3|                  No|            Postgraduate|             Near|  Male|        71|      160|
     Low|         Medium|   Private|      Neutral|                2|                  No|            High School|         Moderate|  Male|        67|      135|
    High|           High|    Public|     Negative|                2|                  No|            High School|              Far|  Male|        66|      116|
  Medium|            Low|   Private|      Neutral|                1|                  No|                College|             Near|  Male|        69|      149|
    High|           High|    Public|     Positive|                5|                  No|            High School|         Moderate|  Male|        72|      143|
-------+---------------+----------+-------------+-----------------+--------------------+-----------------------+-----------------+------+----------+---------+
```

A new column named new_score was made by adding each previous and current exam_score. Thus, columns can be directly manipulated with ‘withColumn’ function.

```
#For SQL, register dataframe as a temporary table
df.createOrReplaceTempView("my_table")
result= spark.sql("SELECT * FROM my_table WHERE Hours_Studied>20")
result.show()
```

[42] ✓ 0.2s

```
+-------------+----------+-------------------+-----------------+------------------------+-----------+--------------+----------------+------
|Hours_Studied|Attendance|Parental_Involvement|Access_to_Resources|Extracurricular_Activities|Sleep_Hours|Previous_Scores|Motivation_Level|Interr
+-------------+----------+-------------------+-----------------+------------------------+-----------+--------------+----------------+------
|           23|        84|                Low|             High|                      No|          7|            73|             Low|
|           24|        98|             Medium|           Medium|                     Yes|          7|            91|          Medium|
|           29|        89|                Low|           Medium|                     Yes|          8|            98|          Medium|
|           29|        84|             Medium|              Low|                     Yes|          7|            68|             Low|
|           25|        78|                Low|             High|                     Yes|          6|            50|          Medium|
|           23|        98|             Medium|           Medium|                     Yes|          8|            71|          Medium|
|           21|        83|             Medium|           Medium|                     Yes|          8|            97|             Low|
|           22|        70|                Low|           Medium|                     Yes|          6|            82|          Medium|
|           29|        78|             Medium|           Medium|                      No|          5|            99|            High|
|           21|        62|               High|              Low|                     Yes|          6|            54|            High|
|           22|        83|               High|             High|                     Yes|          6|            94|          Medium|
|           31|        70|             Medium|             High|                     Yes|          7|            66|          Medium|
|           25|        65|               High|           Medium|                      No|          5|            90|          Medium|
|           21|        65|             Medium|              Low|                     Yes|          7|            91|             Low|
|           21|        65|                Low|           Medium|                     Yes|          6|            98|             Low|
|           24|        68|               High|           Medium|                      No|          8|            56|             Low|
|           21|        84|             Medium|           Medium|                     Yes|          6|            52|             Low|
```

We can use sql query with help of spark.sql. For example, we view from our dataframe that who studied greater than 20 hours.

```
avg_attendance_by_gender=spark.sql("SELECT Gender,AVG(Attendance) as avg_attendance FROM my_table GROUP BY Gender")
avg_attendance_by_gender.show()
```

[43] ✓ 0.5s

```
+------+-----------------+
|Gender|   avg_attendance|
+------+-----------------+
|Female|79.86895810955961|
|  Male|80.05689564761406|
+------+-----------------+
```

Complex query can also be performed like we select the gender and average attendance as per gender and see the ratio is almost equal. Males were less absent with 80.05% average attendance.

```
spark.stop()
```

[51] ✓ 0.9s

Finally, the spark should always be stopped for releasing memory and no further issues while running again.