Spark fundamentals
Exercise 1

Today you'll learn what Spark is.
How to manipulate the data with SQL and DataFrames APIs.
Sources: https://spark.apache.org/;
Spark: The Definitive Guide by by Bill Chambers; Matei Zaharia
Published by O'Reilly Media, Inc., 2018

Introductory text and images are directly copied from the sources.

1) What is Spark?

Apache Spark is a general-purpose cluster computing system. That is, a computing engine with a set of libraries for parallel data processing on computer clusters.

Spark provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. Figure below shows the libraries Spark offers for users.

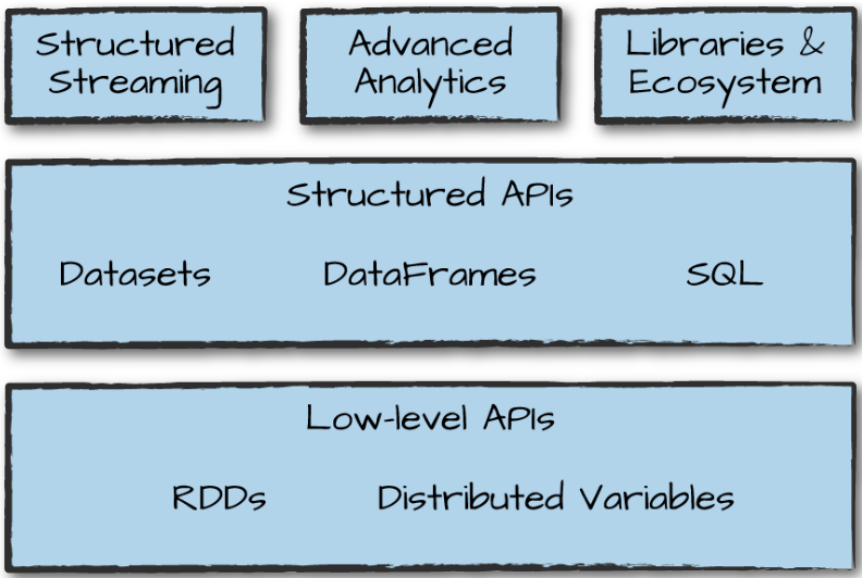


Figure source: Spark: The Definitive Guide by by Bill Chambers; Matei Zaharia
Published by O'Reilly Media, Inc., 2018

Spark applications:
Spark Applications consist of a driver process (Spark Session) and a set of executor processes. The driver process sits on a node in the cluster and runs the main() application function. The driver process is responsible for: maintaining information about the Spark Application;

responding to a user's program or input; and analyzing, distributing, and scheduling work across the executors.

The executors carry out the work that the driver assigns them. This means that each executor is responsible for only two things: executing code assigned to it, and reporting the state of the computation on that executor back to the driver node.
Cluster manager is responsible for resource allocation in Spark.

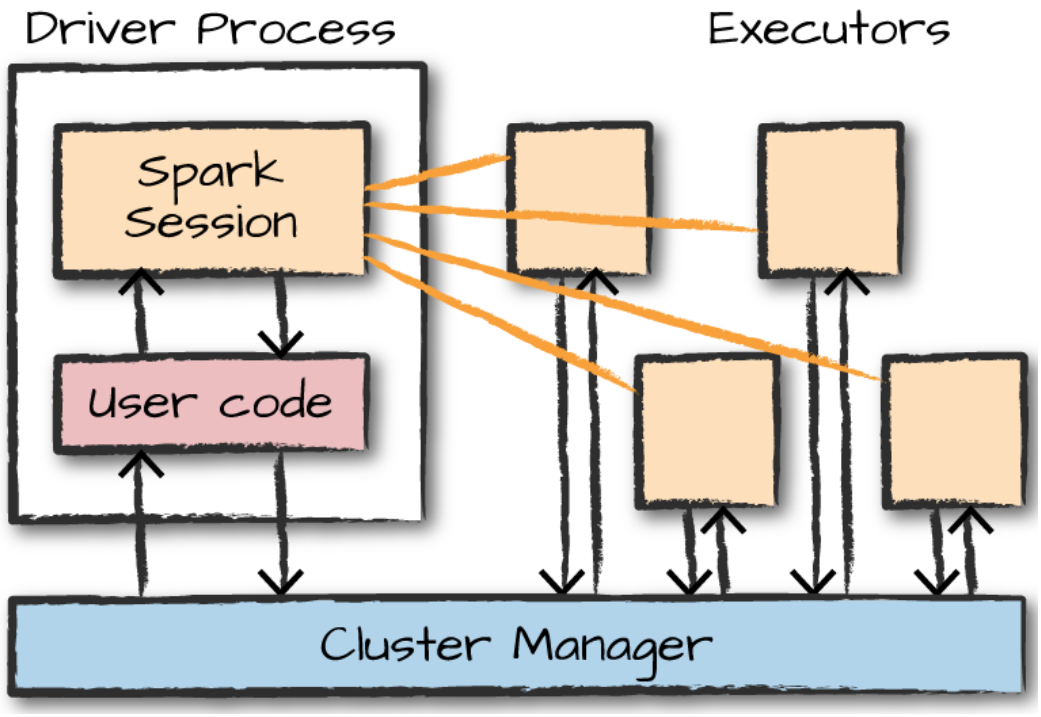


Figure source: Spark: The Definitive Guide by by Bill Chambers; Matei Zaharia
Published by O'Reilly Media, Inc., 2018.

Transformations and Actions
In Spark, the core data structures are immutable, meaning they cannot be changed after they're created. Transformations are the core of how the business logic is expressed in Spark and represent the way of data manipulation in Spark.

Transformations allow to build a logical transformation plan. To trigger the computation, an action is run. An action instructs Spark to compute a result from a series of transformations. There are different kinds of actions, e.g. to view data, collect data to objects, write to output.

In this exercise, we'll explore DataFrames and Spark SQL structured APIs.

2) Accessing CSC PySpark Notebook

1. Navigate to https://notebooks.csc.fi/
2. Select access type (HAKA)
3. Select university (University of Oulu)
4. Look for Pyspark
5. Click and launch on Jupyter Pyspark
6. Start coding :)

3) DataFrames

A DataFrame represents a table of data with rows and columns. The list that defines the columns and the types within those columns is called the *schema*.

So, as we've learned from previous section, we should start from creating a SparkSession:

EXAMPLE:

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import sys
from pyspark.sql import SparkSession
```

Let's create sparkSession now to access functionalities of Spark

```
#parameter for master can be changed depending on the environment

spark = SparkSession.builder \
    .master("local") \
    .appName("Exercise1") \
    .getOrCreate()
```

Let's load the data to DataFrame. In this example, we will try to access data from working directory.

1- Download example data file from below mentioned URL:

https://drive.google.com/open?id=1YLo060ccO-JNBJ7Cn_DT_y0LTkps-0Lg

2- Upload into your jupyter work folder

Now that we have the data uploaded successfully, we can read it. To read the data from the data source to the DataFrame, we use the read method.

1- To find out working directory path, use the following code:

```
pwd
```

'/home/jovyan'

2- A path will be returned, which can be further used to read the file

```
df = spark.read.csv("PATH_TO_FILE", header=True, inferSchema=True)
df.show(vertical=True)
```

```
-RECORD 0-------------------------------
 _c0                | 14
 dateTime           | 01-jan-1990 00:00
 indicator_rain     | 0
 precipitation      | 0.3
 indicator_temp     | 0
 air_temperature    | 9.1
 indicator_wetb     | 0
 wetb               | 9.0
 dewpt              | 8.9
 vappr              | 11.4
 relative_humidity  | 99
 msl                | 1006.7
 indicator_wdsp     | 2
 wind_speed         | 7
 indicator_wddir    | 2
 wind_from_direction | 190
-RECORD 1-------------------------------
```

Let’s read the schema of our dataframe using **printSchema** operation.

```
df.printSchema()
```

```
|-- _c0: integer (nullable = true)
|-- dateTime: string (nullable = true)
|-- indicator_rain: integer (nullable = true)
|-- precipitation: string (nullable = true)
|-- indicator_temp: integer (nullable = true)
|-- air_temperature: string (nullable = true)
|-- indicator_wetb: integer (nullable = true)
|-- wetb: string (nullable = true)
|-- dewpt: string (nullable = true)
|-- vappr: string (nullable = true)
|-- relative_humidity: string (nullable = true)
|-- msl: string (nullable = true)
|-- indicator_wdsp: integer (nullable = true)
|-- wind_speed: string (nullable = true)
|-- indicator_wddir: integer (nullable = true)
|-- wind_from_direction: integer (nullable = true)
```

What do you see here?
Q: Find out method to directly read from URL using Pyspark dataframe API
Sample URL: `https://cli.fusio.net/cli/climate_data/webdata/hly1075.csv`

To write the DataFrame to a file, e.g., we use **write** method.

```
df.write.csv("PATH_TO_FILE", header=True)
```

You can use **col("colName")** function to refer to column of a DataFrame or you can refer by attribute or by indexing:

If we want to access air_temperature column, we can use select like below:

```
from pyspark.sql import functions as F - we load the libraries required here
df.select(F.col("wetb"))
```

You can use then count() or show() the data as a prefix to the code.

There are different ways to access the row in the data frame, we will go for the simplest one by identifying row number:

```
df.collect()[3]
```

However, if data is not indexed just like in our case, we cannot really track the rows. In that case we can use monotonically_increasing_id() and access particular rows.

Q. This is for you to try.

Now, let's do some transformations:

select and selectExpr functions allow you to perform SQL-like queries on data frame. We can also use aggregate functions within the text, like avg or count:

These allow us to perform different operations and even customise the dataframe.

```
df.select("wetb").count()
df.select("wetb").distinct()  #Allows  you  to  select  only  distinct
values
df.select("wetb").distinct().count() #count of distinct values
```

Q: Try to view the results with and without distinct operation, and check how it operates, you may use distinct operation on the whole dataframe as well?

```
df.selectExpr("wetb  *  5  as  newColumn",  "round(air_temperature)  as
roundedTemper").show()
```

SelectExpr can be also used for renaming columns, like:

```
df0 = df.selectExpr("wetb as wetbulb")
```

```
+-------+
|wetbulb|
+-------+
|    9.0|
|    7.4|
|    7.4|
|    7.5|
|    7.3|
|    7.0|
|    6.8|
|    6.0|
|    6.2|
|    6.1|
|    6.1|
|    6.6|
|    7.8|
```

SQL equivalent here is "select wetb as wetbulb from …."

To add column to DataFrame, method **withColumn** can be used and **drop** to remove the column:

Following example removes the column _c0 from our data frame.

```
df.drop("_c0")
```

```
DataFrame[dateTime: string, indicator_rain: string, precipitation: string, indicator_temp: string, air_temperature: string, ind
icator_wetb: string, wetb: string, dewpt: string, vappr: string, relative_humidity: string, msl: string, indicator_wdsp: strin
g, wind_speed: string, indicator_wddir: string, wind_from_direction: string]
```

Let's suppose, we want to get rounded values for air_temperature in a separate column, we will use withColumn for this

```
df.withColumn("air_temperature",F.round(df["air_temperature"],
1)).show()
```

If you noticed the output from **drop** operation, most of the values are of **string type**. Lets see, how can we change the type of the columns

```
df1 = df.selectExpr("cast(air_temperature as float) air_temperature")
df1.dtypes
```

To filter the rows you have to create an expression that evaluates to true or false. Then filter out the rows with an expression that results in false. You can use **filter** or **where** operations:

Let's find cases where the temperature was less than 5 degrees.

```
df_new.filter((F.col('air_temperature')<5)).show()
```

Q: Use where operation to find also the cases where precipitation is greater than 1 mm.

To append to DataFrame, you have to use **union.** However, note that union is made by location, not by schema:

EXAMPLE

```
df_u = df.select("wetb")
df_i = df.select("air_temperature")
unionDF= df_u.union(df_i)
```

#this is just an example, it can include additional operations e.g. you can use filter operation along union operation etc.

Some other useful functions to check: describe - calculates the count, mean, standard deviation, min, and max for numeric columns; String functions (lower, upper,ltrim,rtrim,trim), regular expressions (regexp_replace, regex_extract); date and time functions (current_date, current_timestamp,date_sub, date_add, datediff, to_date)
Aggregation functions include count, countDistinct, etc.

Please check manual for more information about available functions http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.Dataset

https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#

groupBy function allows to perform calculations based on groups of data:

EXAMPLE

```
df.groupBy('precipitation').count().orderBy('count', ascending=False).show()
```

Some additional methods for grouping include windows, rollups, cubes

3) SQL

Structured Query Language is a domain-specific language that is designed for expressing relational operations over data.

Developers can write SQL queries in Spark, via the sql method of SparkSession object. DataFrame is returned.

Therefore, we can register existing DataFrame as a view and start querying it in SQL:

```
df.registerTempTable("example")
```

Below, we can see the example of SQL query, displaying frequency of wetb and air_temperature along with their distinct/unique values in data.

```
pr = spark.sql("Select air_temperature, count(air_temperature) AS total_freq, count(distinct air_temperature) AS unique_air_temp, count(wetb) AS
```

```
wetb_freq,count(distinct wetb) AS unique_wetb FROM example GROUP BY air_temperature")

pr.show()
```

```
+---------------+----------+---------------+---------+-----------+
|air_temperature|total_freq|unique_air_temp|wetb_freq|unique_wetb|
+---------------+----------+---------------+---------+-----------+
|           10.7|      1926|              1|     1926|         48|
|            8.5|      1761|              1|     1761|         44|
|           20.5|        70|              1|       70|         33|
|           -1.2|        24|              1|       24|         12|
|            1.0|        93|              1|       93|         20|
|            8.2|      1560|              1|     1560|         45|
|            2.6|       342|              1|      342|         29|
|            7.3|      1230|              1|     1230|         43|
|            3.1|       464|              1|      464|         31|
|           16.6|       994|              1|      994|         51|
|           12.8|      1623|              1|     1623|         48|
|           14.2|      1655|              1|     1655|         51|
|           17.1|       713|              1|      713|         56|
|           -2.4|        11|              1|       11|          6|
|            8.3|      1639|              1|     1639|         45|
|           22.4|        16|              1|       16|         16|
|            4.2|       649|              1|      649|         31|
|           -0.1|        45|              1|       45|         15|
|            9.2|      1989|              1|     1989|         47|
|           18.1|       365|              1|      365|         56|
+---------------+----------+---------------+---------+-----------+
only showing top 20 rows
```

Q: Try to bypass this step by reading the file directly using SQL context.

Note, there is a difference between temporary and global temporary views. Temporary view is session-scoped and will disappear if the session that has created it terminates.

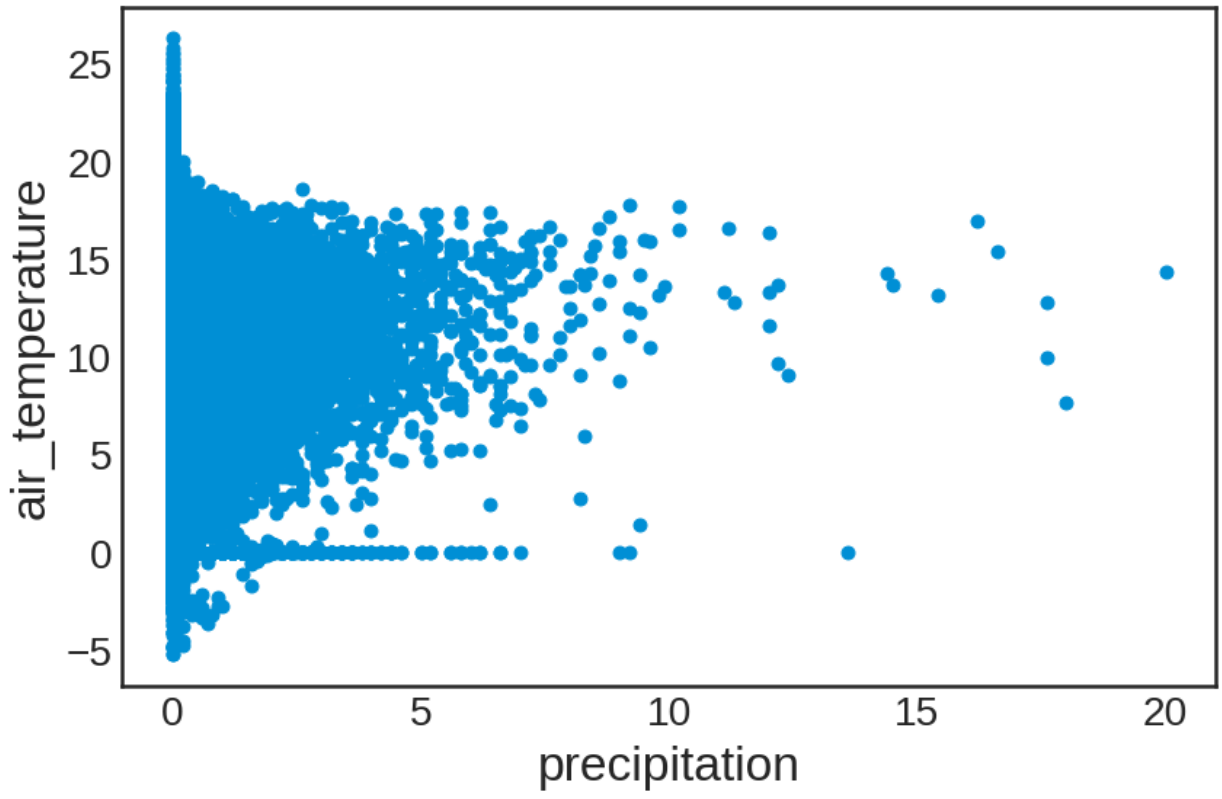
It is also possible to write the actual table into Hive metastore (saveAsTable). That way, the table will materialize in the metastore. To load the persistent table from metasore to DataFrame, table method on SparkSession should be used.

4) Using matplotlib to visualize data

Here, we'll use what we have learned so far and learn how we can visualize the data. We'll use here matplotlib to visualize data.
Visualizing data with matplotlib requires conversion of spark dataframe to pandas dataframe

```
import pandas as pd
import matplotlib.pyplot as plt

plt.style.use('seaborn-white')
```

```
plt.style.use('seaborn-white')

dff = df.toPandas()

params = {'legend.fontsize': 'large',
          'figure.figsize': (15, 9),
         'axes.labelsize': 'x-large',
         'axes.titlesize':'x-large',
         'xtick.labelsize':'large',
         'ytick.labelsize':'large'}
dff.plot(kind='scatter',x='precipitation',y='air_temperature')

#This will most likely give a type error because of dtype object, you can use the following code to convert object data to numerical type. This code should be called before the plot line above.
dff['precipitation'] = pd.to_numeric(dff.precipitation,errors='coerce')
dff['air_temperature'] = pd.to_numeric(dff.air_temperature,errors='coerce')
```



Q: Try to plot histograms and other kinds of plots on different data.

Questions for the Exercise 1 report:

Please answer the following questions. Compile your answers into a 1 page report, put there your student ID number and your name.

1. What have you learned today?
2. What was difficult? At which step have you encountered problems?
3. Provide some figures by using matplotlib and comment what is visualized there.