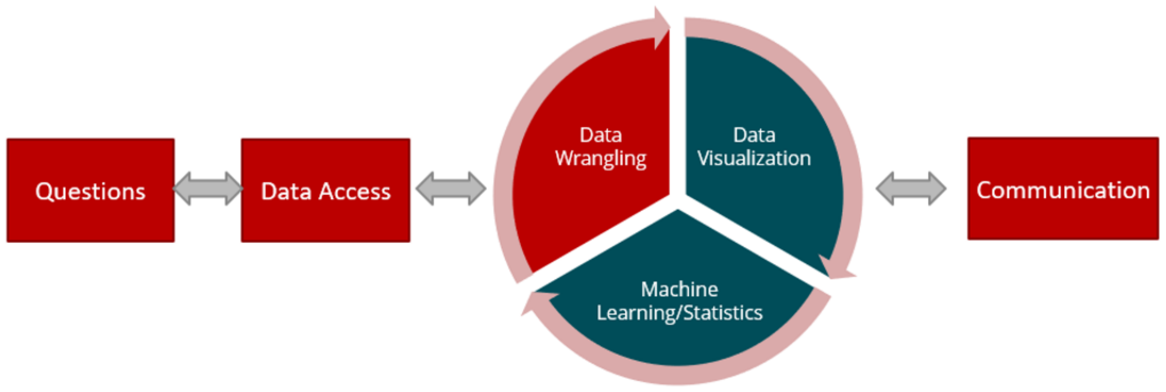
SQL and SparkSQL

Overview

- SQL Basics
- SparkSQL - Context
- JDBC Connectors
- Spark Runtime
- Basic Data Management Operations in SparkSQL
- Take home excercise (not graded)
- Readings

Do you remember the Data Science Process that was introduced in our first lecture?

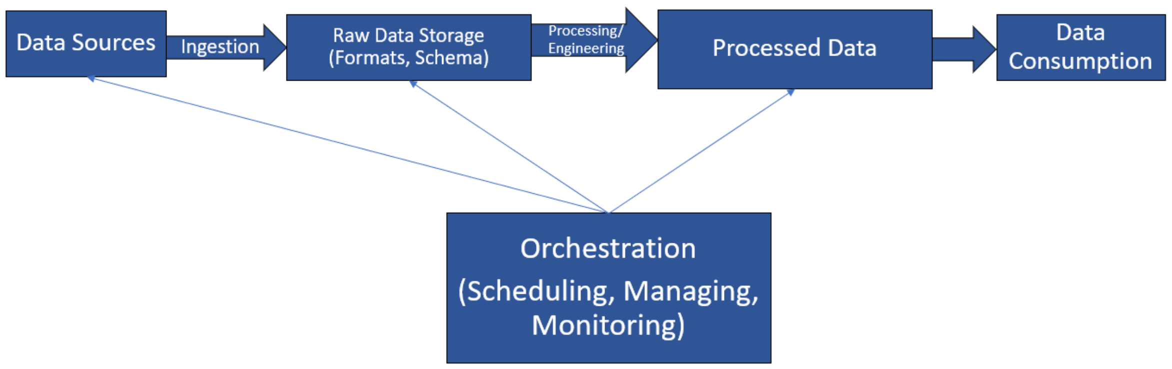
Machine learning has much more to it than just the model.



The Data Science Process

1. The Big Picture - Data Engineering Pipeline

The big picture in machine learning modeling lies in the **Data Engineering pipeline concept**. A data pipeline consists of a series of connected processes that move the data from one point to another, possibly transforming the data along the way.

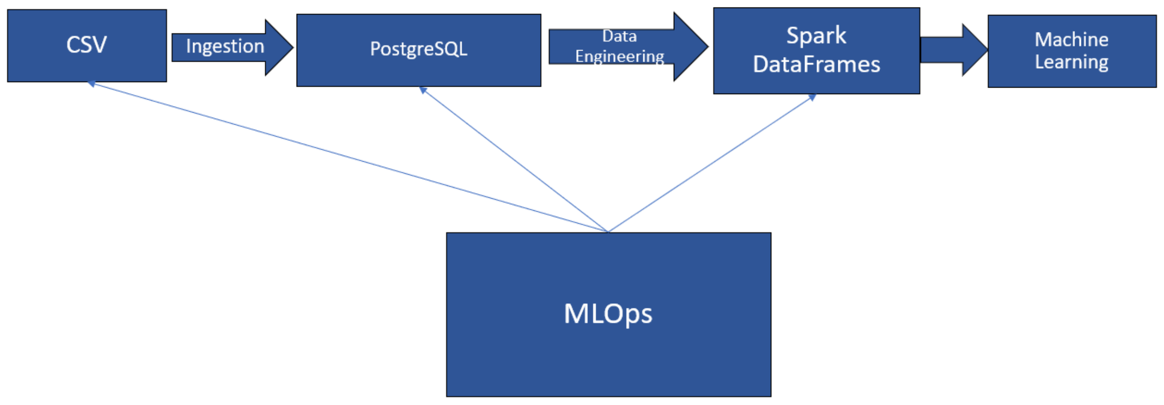

Data Sources
Ingestion
Raw Data Storage (Formats, Schema)
Processing/ Engineering
Processed Data
Data Consumption
Orchestration (Scheduling, Managing, Monitoring)

The General Processes in Data Engineering Pipeline

Ingested data can be stored in a better format by being stored into:

- data warehouses,
- data lakes,
- NoSQL, or
- Relational databases

In our course, we will use the following pipeline:



We will discuss each step in detail throughout the course

Relational Databases - One Option to Store Your Ingested Data

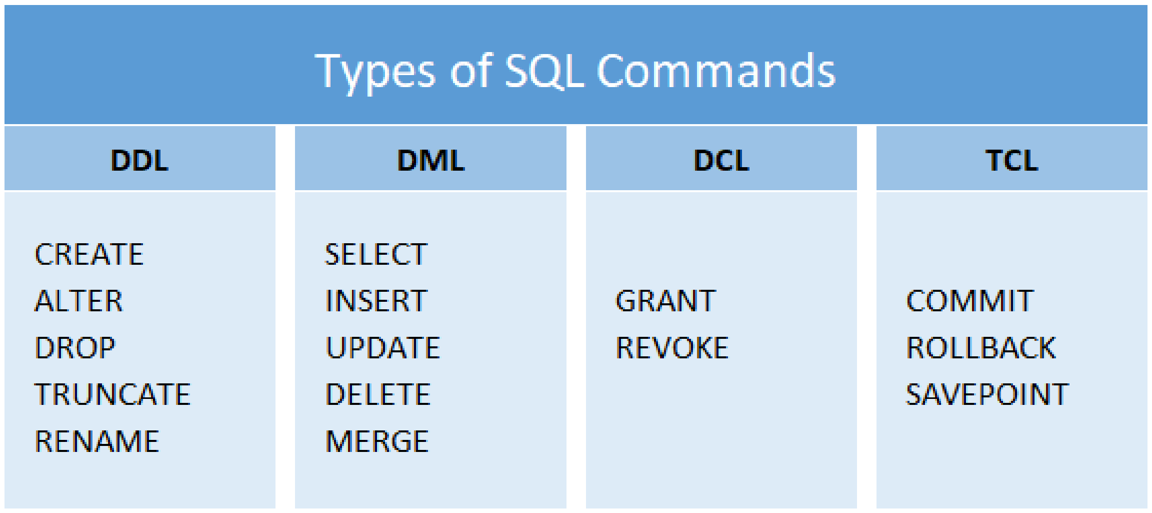
- A relational database (or RDB) is a structure for organizing information that one may think of initially as a supercharged version of a data frame.
- RDBs are commonly manipulated and queried using **SQL: Structured Query Language.**
- SQL is case insensitive

- In an RDB, the basic unit of data storage is the **table (or relation).** Rows are rows, but sometimes dubbed tuples or records, and columns are sometimes dubbed attributes. (Fields are specific cells within a given row.)

Postgres

- There are many implementations of SQL (SEE-kwule): Oracle, MySQL, SQLite, etc.
- In this class, we will make use of PostgreSQL, or more simply Postgres. It is open source, with Mac, Windows, and Linux versions available. You should download and install it!
- Postgres will set up a server on your machine. When you install Postgres, a superuser account "postgres" will be set up, with a password specified by you. Make this a good one, i.e., not just "postgres." (That's because hackers like to look for lazily set up postgres accounts.)

Categories of SQL Commands



Types of SQL Commands			
DDL	**DML**	**DCL**	**TCL**

CREATE
ALTER
DROP
TRUNCATE
RENAME
	SELECT
	INSERT
	UPDATE
	DELETE
	MERGE
		GRANT
		REVOKE
			COMMIT
			ROLLBACK
			SAVEPOINT

SQL Command Categories

Data Definition Language (DDL)

DDL takes care of table structures. We will look into the most popular DDLs for creating, updating or dropping a table.

1. Creating a Table

Assuming you will need to create and populate a table to host your data.

The command is

```
CREATE TABLE IF NOT EXISTS < < name > >
(<< column 1 >>> << type 1 >> << constraint 1 >>, ... , << multi-column constraint(s) >>);
```

Here is a simple example:

```
CREATE TABLE IF NOT EXISTS products(

  product_id SERIAL,
  label TEXT,
  price decimal,
  inventory INTEGER

);
```

This creates a four-column table that contains a label for each product, its price, and the current number of each product available.

ⓘ (Be careful! You may not have the permissions to create tables - depending on your Postgres Installation-). To fix it, grant your user the proper permissions using this command **GRANT CREATE ON SCHEMA public TO postgres;**

Available Data Types

2. Alter a Table

If you wish to add or delete an entire column, or rename an column, or change constraints, etc., you would “update” your table using alter table.

The command is

```
ALTER TABLE < < name > >
<< action >> ;
```

Here are two simple examples:

```
ALTER TABLE products ADD COLUMN rating REAL DEFAULT 0.0;
```

```
ALTER TABLE products DROP COLUMN rating;
```

3. Drop a Table

To remove a table in its entirety:

```
DROP TABLE <<name>>;
```

To check that the table is removed, look for it in the Table list on PgAdmin4.

Data Manipulation Language (DML)

DML takes care of data in the tables. We will look into the most popular DMLs for data management.

1. Insert Values

There are a few ways to insert data into a SQL table. We'll show how to do it row-by-row here, and then utilize select to create bigger tables next week.

To populate the table one row at a time:

```
INSERT INTO << table >> (<< column i >>, << column j >>,...) VALUES
(<< value i >>, << value j >>,...),
...
(<< value i >>, << value j >>,...);
```

If you leave out a column, then the data there will be missing (and can be added later with **UPDATE TABLE**) or will have a default value. Note that any column with data type SERIAL has default behavior: it will auto-increment).

```
INSERT INTO products (label,price,inventory) VALUES
  ('kirk action figure',50,13),
  ('spock action figure',40,22);
```

</div> The *product_id*, being of type SERIAL, will take on the values 1, then 2.

2. Select: Querying a Database

The **SELECT** command is how we query a database. It is a versatile and powerful command!

A shortened definition that highlights elements of the syntax that are important in the context of this class is:

```
SELECT

  <<column1, column2, columnn>>
  FROM <<table>>
  [WHERE <<condition>>]
  [GROUP BY <<expression>>]
  [HAVING <<condition>>]
  [ORDER BY <<expression>>];
```

Example:

```
SELECT product_id, label, price
FROM products
WHERE price >= 10;
```

3. Update Table Values

The **UPDATE** command allows us to modify values in table cells.

```
UPDATE << table >>
  SET <<column1>> = << new value 1 >> ,
  SET <<column2>> = << new value 2 >>,
  ...
  WHERE <<row condition >>;
</div>
```

Think of the **WHERE < row condition >** as being like a call to the **which()** function in R: in it, you set a range of values for one of the table columns, and thereby select which rows to update.

```
UPDATE products
  set price = 100
  where price >= 45;
```

(Note that when you look at an updated table, the serial data may not be displayed in numeric order, i.e., the rows may be rearranged. This is OK.)

4. Delete Rows

To remove one or more entries from a table:

```
DELETE FROM << table >>
  WHERE << condition >>;
```

SparkSQL - Context

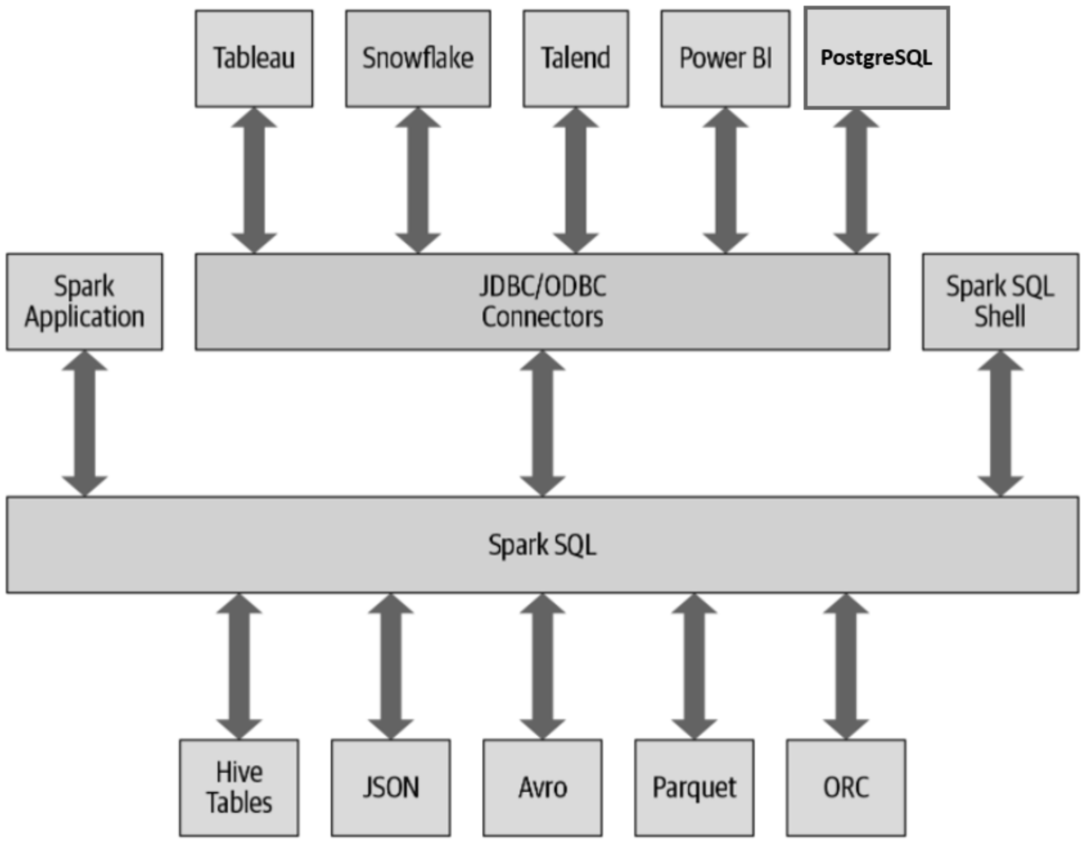
- SQL provides the main knowledge to ingest data into PostgreSQL tables.
- SQL doesn't support big-data processing on its own. So, in order to conduct SQL on big data, one good way to do it is to use Spark SQL

General Rule in SQL and SparkSQL Processing

1. Use plain SQL to conduct DDL and DCL related operations and store them in *.sql files. These operations are not impacted by the data size

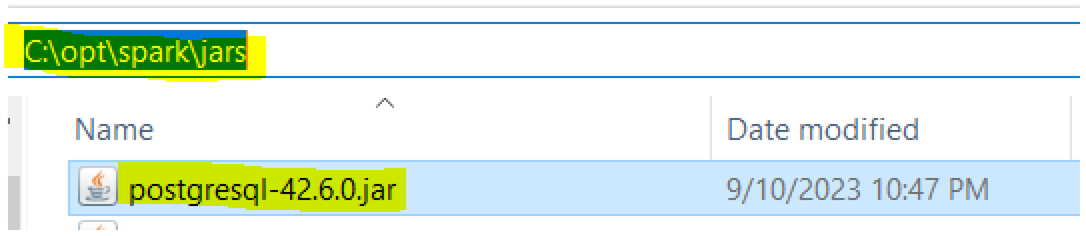
2. Use SparkSQL to conduct DML related operations.

However, Spark uses "connectors" to integrate its operation with external platforms/applications (e.g. PostgreSQL).



Install JDBC Connector

- Install the JDBC driver that matches your Java Version from https://jdbc.postgresql.org/download/
- Place the installed **JAR file** into your **SPARK_HOME directory under jars folder**.
- Close all the terminals and restart Jupyter notebook over in a new terminal (or Restart Your Machine) </ul>



C:\opt\spark\jars

postgresql-42.6.0.jar 9/10/2023 10:47 PM

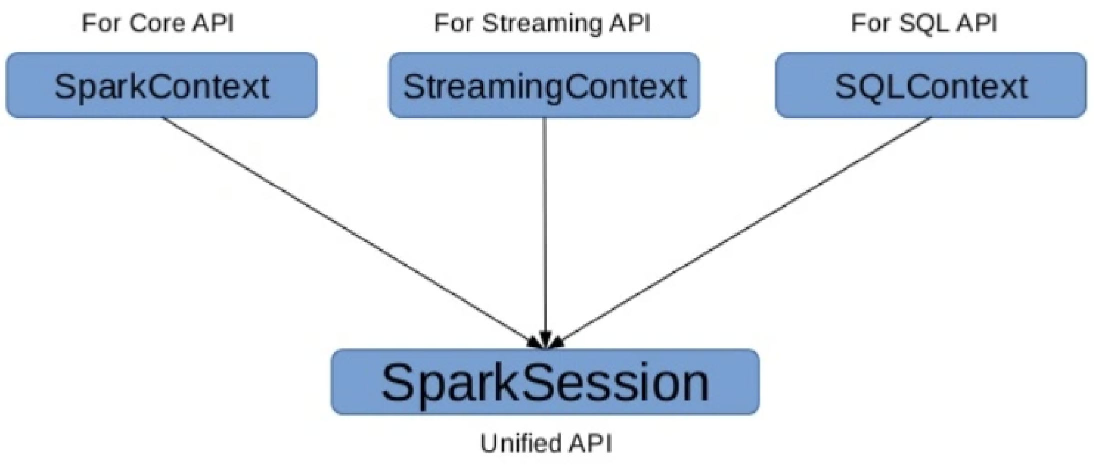
The Non-SparkSQL way to Initialize the Application

Continue to use this approach if you don't need to interface with Database

In [ ]:
```
# Uncomment the following lines if you are using Windows!

#import findspark
#findspark.init()
#findspark.find()

import pyspark
from pyspark.sql import SparkSession


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

Spark Runtime

- SparkContext is used to help you access hardware-level and some software-level configurations for your Spark application.
- Most Spark Components have their own Contexts (e.g. StreamingContext for streaming purposes).
- Starting Spark 2.x, SparkSession was created to run applications in an easier way than SparkContext.
- You may want to access the context object in order to specify the hardware-level or software-level configurations you need to specify



Contexts and SparkSession

Initialize SparkSQL Application - Create SQL Context

In [ ]:
```
# Uncomment the following lines if you are using Windows!

#import findspark
#findspark.init()
#findspark.find()

import pyspark

from pyspark.sql import SparkSession
from pyspark import SparkContext, SQLContext

appName = "Big Data Analytics"
master = "local"

# Create Configuration object for Spark.
conf = pyspark.SparkConf()\
    .set('spark.driver.host','127.0.0.1')\
    .setAppName(appName)\
    .setMaster(master)

# Create Spark Context with the new configurations rather than relying on the default
sc = SparkContext.getOrCreate(conf=conf)

# You need to create SQL Context to conduct some database operations like what we wil
sqlContext = SQLContext(sc)

# If you have SQL context, you create the session from the Spark Context
spark = sqlContext.sparkSession.builder.getOrCreate()
```

Download the Dataset

In [ ]:
```
!python -m wget https://www.andrew.cmu.edu/user/mfarag/763/KDDTrain+.txt
```

Read-in the Dataset

In [ ]:
```
# Load data from csv to a dataframe on a local machine.
# header=False means the first row is not a header
# sep=',' means the column are seperated using ','
col_names = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","classes","difficulty_level"]

df = spark.read.csv("KDDTrain+.txt",header=False, inferSchema= True).toDF(*col_names)
```

Print Schema

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

Write Data to Tables in SparkSQL

What is the SQL command represented in this code snippet?

```
In [ ]: # update the options with your db user and password

        db_properties={}
        db_properties['username']="postgres"
        db_properties['password']="bigdata"
        db_properties['url']= "jdbc:postgresql://localhost:5432/postgres"
        db_properties['table']="intrusion"
        db_properties['driver']="org.postgresql.Driver"

        df.write.format("jdbc")\
        .mode("overwrite")\
        .option("url", db_properties['url'])\
        .option("dbtable", db_properties['table'])\
        .option("user", db_properties['username'])\
        .option("password", db_properties['password'])\
        .option("Driver", db_properties['driver'])\
        .save()
```

Spark Data Write Modes

In Spark, there are several modes for writing your dataframe data to external data source. In the previous example, we used "overwrite" but it's not the only option!

- **append:** Append contents of this dataframe to existing table's data. **Append** simulates the process of **inserting new records.**
- **overwrite:** Overwrite existing data. It recreates the table with all the data. Use it with cautious.
- **ignore:** Silently ignore this operation if data already exists.
- **error** (default case): Throw an exception if data already exists.

Now, read the data back!

```
In [ ]: df_read = sqlContext.read.format("jdbc")\
            .option("url", db_properties['url'])\
            .option("dbtable", db_properties['table'])\
            .option("user", db_properties['username'])\
            .option("password", db_properties['password'])\
            .option("Driver", db_properties['driver'])\
            .load()

        df_read.show(1, vertical=True)
        df_read.printSchema()
```

```
In [ ]: df_read.show(1, vertical=True)
```

Selecting Columns

Keep in mind, you can chain functions here!

In [ ]:
```python
# Selecting a single column
subset_df = df.select("classes")

# Selecting multiple columns and display them on the flyabs
df.select("protocol_type","duration", "flag","classes").show(1,vertical=True)
```

Filtering Data

Filtering allows you to subset a DataFrame based on specific conditions.

In [ ]:
```python
# Filter rows where a condition is met
df_read.filter(df_read["classes"] == 'normal').show(1,vertical=True)
```

In [ ]:
```python
# Filter using SQL-like syntax
df_read.filter("count > 1").show(1,vertical=True)
```

Renaming Columns

To rename columns in a DataFrame, you can use the withColumnRenamed() method.

In [ ]:
```python
# Filter rows where a condition is met
df_read.withColumnRenamed("src_bytes", "source_bytes").printSchema()
```

Transforming Data

Data transformation involves applying functions or expressions to columns in a DataFrame to create new columns or modify existing ones.

In [ ]:
```python
from pyspark.sql.functions import col

# Create a new column
df_with_double_difficulty_level = df_read.withColumn("double_difficulty_level", col("

df_with_double_difficulty_level.show(1,vertical=True)
```

In [ ]:
```python
from pyspark.sql.functions import expr

# Using expressions
df_with_new_count = df_read.withColumn("new_count", expr("count + 1"))

df_with_new_count.show(1,vertical=True)
```

Aggregating Data

Aggregation operations are essential for summarizing data. PySpark provides various aggregation functions like sum(), avg(), min(), and max().

In [ ]:
```python
from pyspark.sql.functions import avg

# Create a new column
df_read.select(avg("count")).show()
```

Group By Example Using SparkSQL Functions

In [ ]:
```python
df_read.groupby("protocol_type").count().show()
```

Select: Order By

Note that one can use an ordinal number (as in **GROUP BY 1**, meaning group by the first column of the selected table). Also note that the output from GroupBy is not necessarily ordered by any column (the output is in essentially random order)...that’s what **ORDER BY** is for.

The ASC means “ascending,” which is the default, which is contrasted with DESC for "descending."

In [ ]:
```python
df_read.orderBy("difficulty_level", ascending=False).show(5,vertical=True)
```

Try at home and enrich Your Knowledge

- Find the most popular 5 difficulty levels.
- Find the total number of records where the user was not logged in
- What is the most popular "attack" for TCP connections? (i.e., records with protocol_type = "tcp").

Readings

- Application of SQL Database, published on Canvas
- Applicationn of PySpark, published on Canvas
- PySpark Dataframe Guide