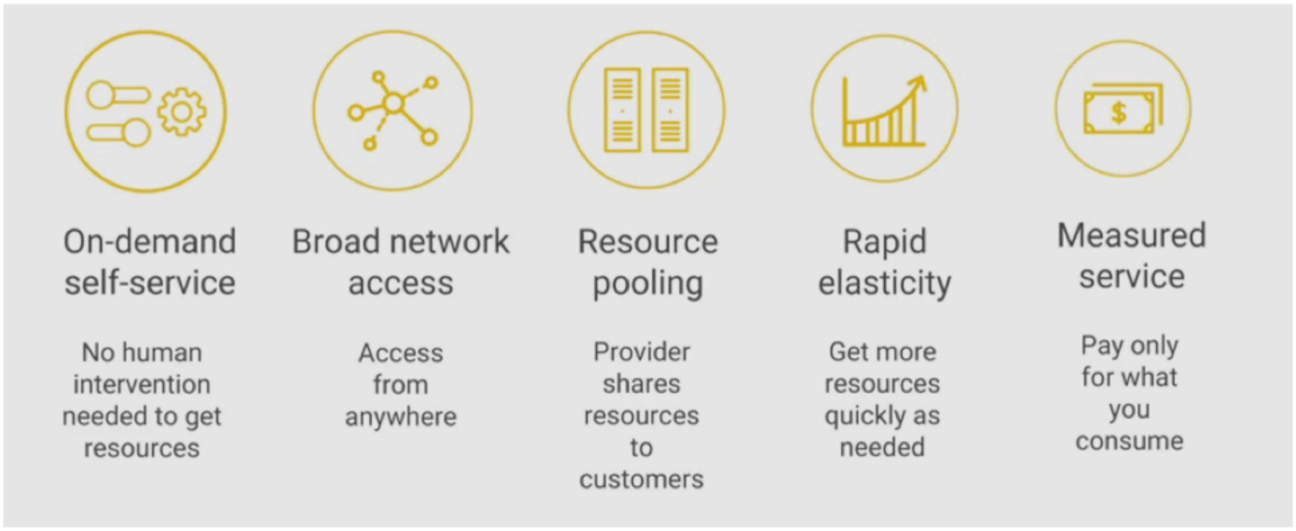
Introduction to the Cloud and Apache Spark

Overview

- Introduction to the Cloud
- Context
- Basics of Apache Spark
- Run Apache Spark on the Cloud

Why would you go with the Cloud?


On-demand self-service
No human intervention needed to get resources
Broad network access
Access from anywhere
Resource pooling
Provider shares resources to customers
Rapid elasticity
Get more resources quickly as needed
Measured service
Pay only for what you consume

What is Cloud in the REAL World?

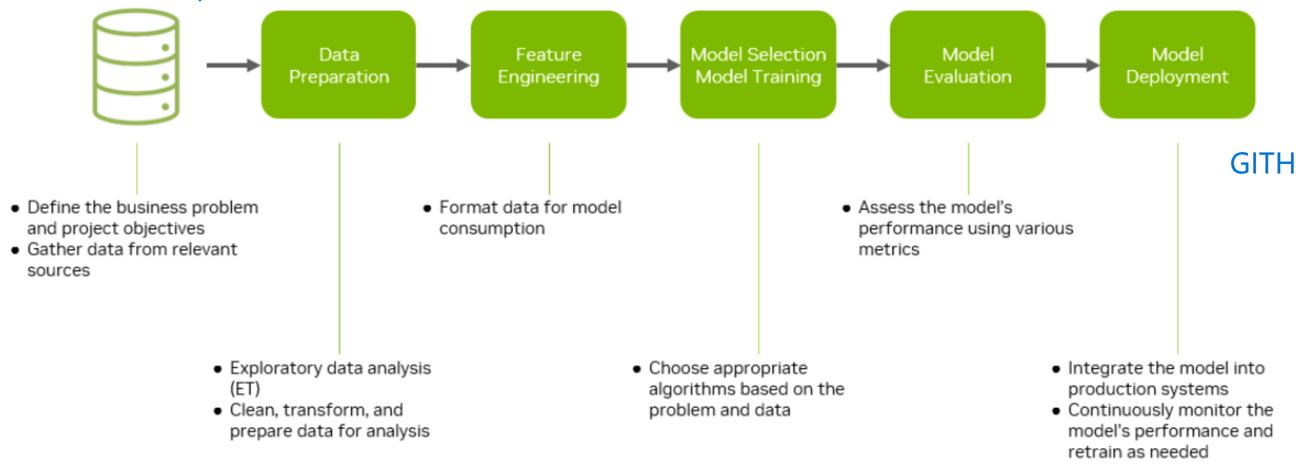
- **“Cloud”** refers to large Internet services running on 10,000s of machines (Amazon, Google, Microsoft, etc)
- **“Cloud computing”** refers to services by these companies that let external customers rent cycles and storage
  - Amazon EC2: virtual machines at 8.5¢/hour (approximated cost)
  - Amazon S3: storage at 21¢/GB/month (approximated cost)
  - Google Cloud AppEngine
  - Windows Azure

Connect the Dots!

SML, TF
pytorch

Spark

Postagres/neo4



- Define the business problem and project objectives
- Gather data from relevant sources

- Format data for model consumption

- Assess the model's performance using various metrics

GITHUB

- Exploratory data analysis (ET)
- Clean, transform, and prepare data for analysis

- Choose appropriate algorithms based on the problem and data

- Integrate the model into production systems
- Continuously monitor the model's performance and retrain as needed

Context

Last lecture, we talked about the machine learning modeling process. Now, let's switch gears and discuss our technical infrastructure that can be deployed later to the cloud. You will need Spark installed on your machine in order to run the code snippets in this lecture.

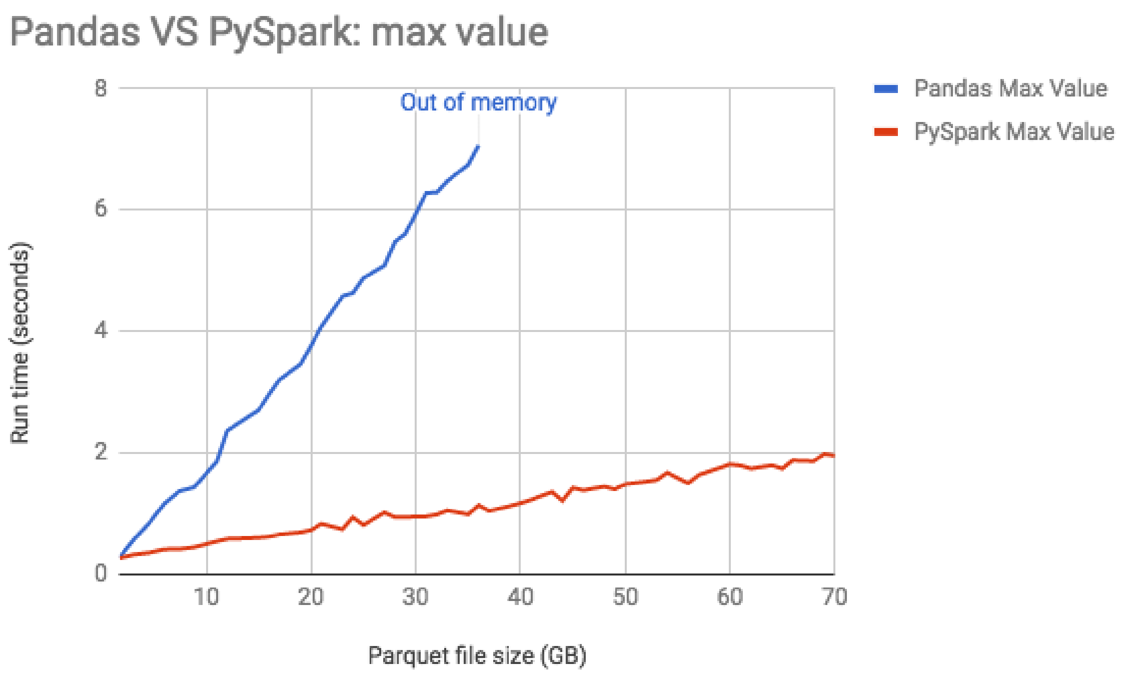
In this class, we will use **Apache Spark** for data preparation, cleaning, and feature engineering. But why Spark?

Why Spark?

- Developed in 2009 at UC Berkeley AMPLab, then open sourced in 2010,
- Spark is the next revolution of the popular Hadoop MapReduce framework.
- Gartner, Advanced Analytics and Data Science (2014) "Organizations that are looking at big data challenges – including collection, ETL, storage, exploration and analytics – should consider Spark for its in-memory performance and the breadth of its model. It supports advanced analytics solutions on Hadoop clusters, including the iterative model required for machine learning and graph analysis."

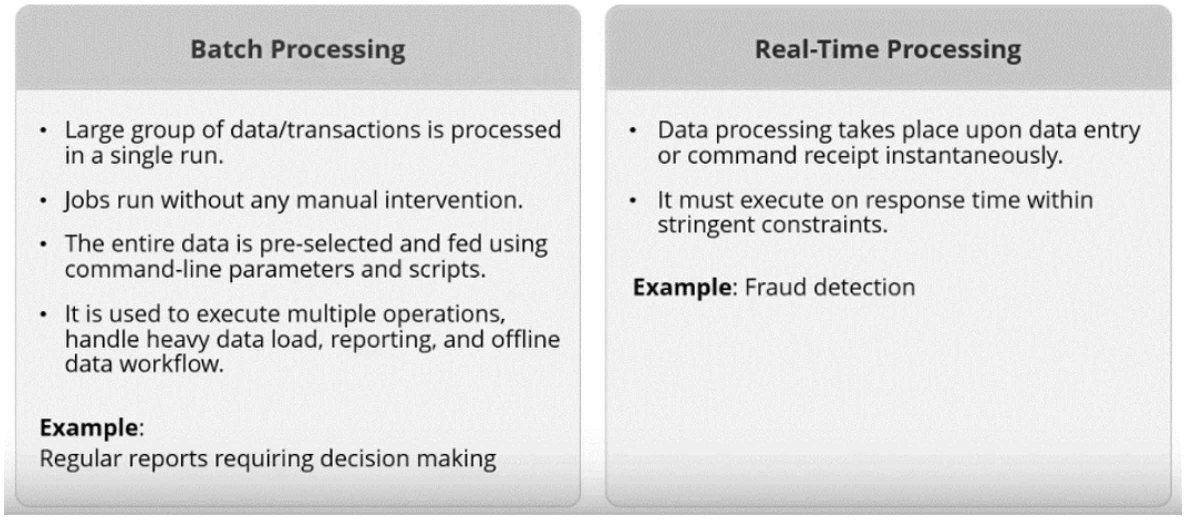
Pandas vs PySpark (Spark developed in Python)

In a research done by Databricks, an industrial leader in the domain of big data storage and processing, PySpark shows superior performance compared to traditional implementations.



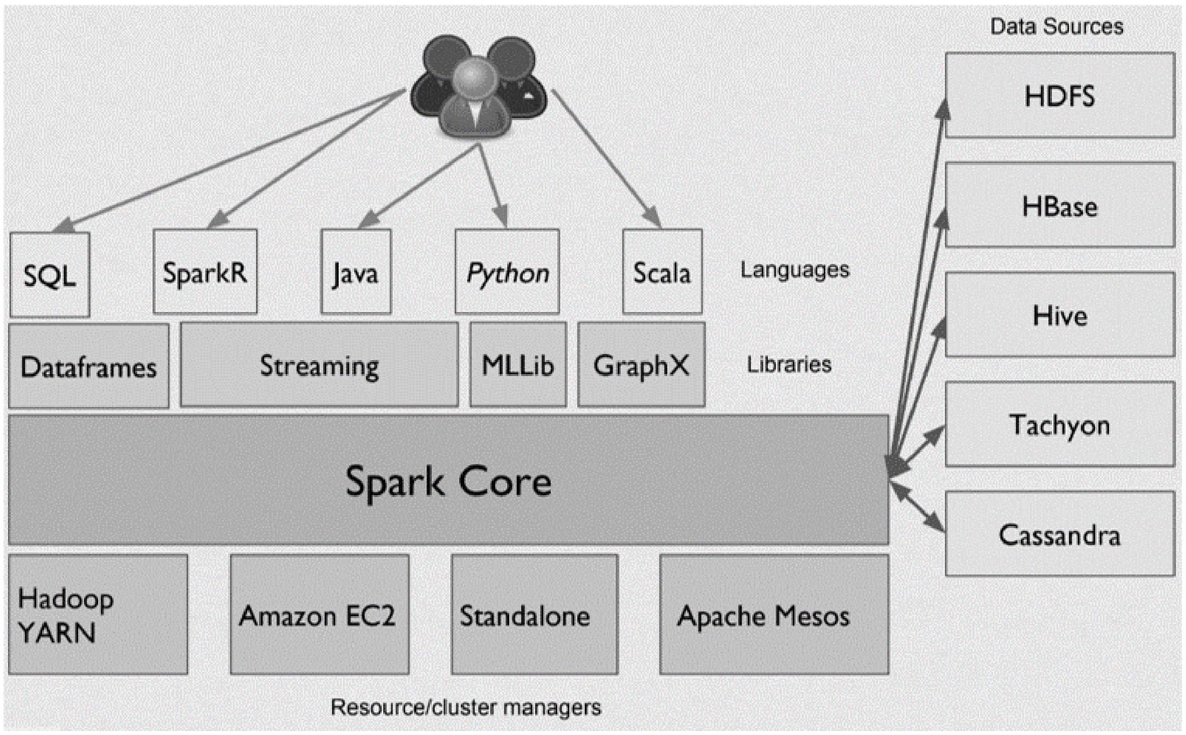
What is Spark?

- Apache Spark is a fast and general-purpose cluster computing system for large scale data processing.
- Spark was originally written in Scala, which allows concise function syntax and interactive use.
- Apache Spark provides High-level APIs in Java, Scala, Python (PySpark) and R.
- Apache Spark combines two different modes of processing:
  - **Batch-based Processing** which can be provided via Apache Hadoop MapReduce
  - **Real-time Processing** which can be provided via Apache Storm.

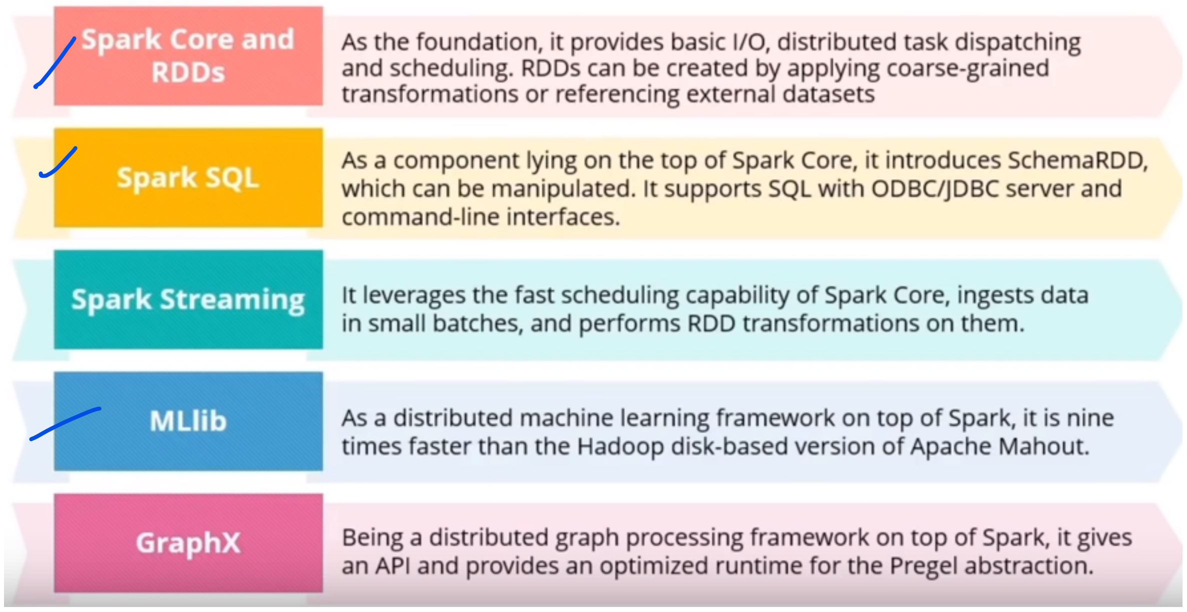


Batch Processing	Real-Time Processing
• Large group of data/transactions is processed in a single run. • Jobs run without any manual intervention. • The entire data is pre-selected and fed using command-line parameters and scripts. • It is used to execute multiple operations, handle heavy data load, reporting, and offline data workflow. **Example**: Regular reports requiring decision making	• Data processing takes place upon data entry or command receipt instantaneously. • It must execute on response time within stringent constraints. **Example**: Fraud detection

Spark Ecosytem



Spark Componenets



Spark Core and RDDs	As the foundation, it provides basic I/O, distributed task dispatching and scheduling. RDDs can be created by applying coarse-grained transformations or referencing external datasets
Spark SQL	As a component lying on the top of Spark Core, it introduces SchemaRDD, which can be manipulated. It supports SQL with ODBC/JDBC server and command-line interfaces.
Spark Streaming	It leverages the fast scheduling capability of Spark Core, ingests data in small batches, and performs RDD transformations on them.
MLlib	As a distributed machine learning framework on top of Spark, it is nine times faster than the Hadoop disk-based version of Apache Mahout.
GraphX	Being a distributed graph processing framework on top of Spark, it gives an API and provides an optimized runtime for the Pregel abstraction.

Spark Core

Spark Core is the general execution engine for the Spark platform that other functionalities are built on top of it. Spark has several advantages:

- Speed: runs programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk
- Ease of Use: Write applications quickly in Java, Scala, Python, R
- Generality: Combine SQL, streaming, and complex analytics
- Runs Everywhere: Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3

Where data is stored if you run data on spark

In [ ]:
```
# if you installed Spark on windows,
# you may need findspark and need to initialize it prior to being able to use pyspark

!pip install findspark
```

In [1]:
```
# Uncomment the following lines if you are using Windows!
import findspark
findspark.init()
findspark.find()
# The above lines are used mostly in Windows only and you don't need them on other platf
import pyspark
from pyspark.sql import SparkSession


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

Download Data Files Remotely

In [ ]:
```
!pip install wget
```

In [5]:
```
# Downloading and preprocessing KDD Train Data
!python -m wget https://www.andrew.cmu.edu/user/mfarag/763/KDDTrain+.txt
```

```
Saved under KDDTrain+.txt
```

Cloud Consideration

Your data need to be moved to a special storage server if you are running Spark on the Cloud. This special storage is called HDFS. The following command is used to move your data from your local storage to the special storage server.

In [ ]:
```
# Uncomment and Execute this line if you are running your notebook on the Cloud
#!hadoop fs -put KDDTrain+.txt /
```

Spark Dataframes

Inspired by pandas DataFrames in structure, format, and a few specific operations, Spark DataFrames are like distributed in-memory tables with named columns and schemas, where each column has a specific data type: integer, string, array, map, real, date, timestamp, etc. To a human’s eye, a Spark DataFrame is like a table

When data are visualized as a structured table, it’s not only easy to digest but also easy to work with when it comes to common operations you might want to execute on rows and columns.
Also, DataFrames are immutable and Spark keeps a lineage of all transformations. You can add or change the names and data types of the columns, creating new DataFrames while the previous versions are preserved. A named column in a DataFrame and its associated Spark data type can be declared in the schema.

Let's examine the generic and structured data types available in Spark before we use them to define a schema. Then we'll illustrate how to create a DataFrame with a schema.

Spark’s Basic Data Types

Spark supports basic internal data types. These data types can be declared in your Spark application or defined in your schema

Data type	Value assigned in Python	API to instantiate
`ByteType`	`int`	`DataTypes.ByteType`
`ShortType`	`int`	`DataTypes.ShortType`
`IntegerType`	`int`	`DataTypes.IntegerType`
`LongType`	`int`	`DataTypes.LongType`
`FloatType`	`float`	`DataTypes.FloatType`
`DoubleType`	`float`	`DataTypes.DoubleType`
`StringType`	`str`	`DataTypes.StringType`
`BooleanType`	`bool`	`DataTypes.BooleanType`
`DecimalType`	`decimal.Decimal`	`DecimalType`

In [2]:

```
# 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 ','
df = spark.read.csv('KDDTrain+.txt', header=False, sep=",")
# on the Cloud, the files will have to be at the root level. So, the cloud version is:
#df = spark.read.csv('/KDDTrain+.txt', header=False, sep=",")

df.show(5, vertical=True)
```

```
-RECORD 0--------
 _c0  | 0
 _c1  | tcp
 _c2  | ftp_data
 _c3  | SF
 _c4  | 491
 _c5  | 0
 _c6  | 0
 _c7  | 0
 _c8  | 0
 _c9  | 0
 _c10 | 0
 _c11 | 0
 _c12 | 0
 _c13 | 0
 _c14 | 0
 _c15 | 0
 _c16 | 0
 _c17 | 0
 _c18 | 0
 _c19 | 0
 _c20 | 0
 _c21 | 0
 _c22 | 2
 _c23 | 2
 _c24 | 0.00
 _c25 | 0.00
 _c26 | 0.00
 _c27 | 0.00
 _c28 | 1.00
 _c29 | 0.00
 _c30 | 0.00
 _c31 | 150
 _c32 | 25
 _c33 | 0.17
 _c34 | 0.03
 _c35 | 0.17
 _c36 | 0.00
 _c37 | 0.00
 _c38 | 0.00
 _c39 | 0.05
 _c40 | 0.00
 _c41 | normal
 _c42 | 20
-RECORD 1--------
 _c0  | 0
 _c1  | udp
 _c2  | other
 _c3  | SF
 _c4  | 146
 _c5  | 0
 _c6  | 0
 _c7  | 0
 _c8  | 0
 _c9  | 0
 _c10 | 0
 _c11 | 0
 _c12 | 0
 _c13 | 0
 _c14 | 0
 _c15 | 0
 _c16 | 0
 _c17 | 0
 _c18 | 0
 _c19 | 0
 _c20 | 0
 _c21 | 0
 _c22 | 13
 _c23 | 1
 _c24 | 0.00
 _c25 | 0.00
 _c26 | 0.00
 _c27 | 0.00
 _c28 | 0.08
 _c29 | 0.15
 _c30 | 0.00
 _c31 | 255
 _c32 | 1
 _c33 | 0.00
 _c34 | 0.60
 _c35 | 0.88
 _c36 | 0.00
 _c37 | 0.00
 _c38 | 0.00
 _c39 | 0.00
 _c40 | 0.00
 _c41 | normal
 _c42 | 15
-RECORD 2--------
 _c0  | 0
 _c1  | tcp
 _c2  | private
 _c3  | S0
 _c4  | 0
 _c5  | 0
 _c6  | 0
 _c7  | 0
 _c8  | 0
 _c9  | 0
 _c10 | 0
 _c11 | 0
 _c12 | 0
 _c13 | 0
 _c14 | 0
 _c15 | 0
 _c16 | 0
 _c17 | 0
 _c18 | 0
 _c19 | 0
 _c20 | 0
 _c21 | 0
 _c22 | 123
 _c23 | 6
 _c24 | 1.00
 _c25 | 1.00
 _c26 | 0.00
 _c27 | 0.00
 _c28 | 0.05
 _c29 | 0.07
 _c30 | 0.00
 _c31 | 255
 _c32 | 26
 _c33 | 0.10
 _c34 | 0.05
 _c35 | 0.00
 _c36 | 0.00
 _c37 | 1.00
 _c38 | 1.00
 _c39 | 0.00
 _c40 | 0.00
 _c41 | neptune
 _c42 | 19
-RECORD 3--------
 _c0  | 0
 _c1  | tcp
 _c2  | http
 _c3  | SF
 _c4  | 232
 _c5  | 8153
 _c6  | 0
 _c7  | 0
 _c8  | 0
 _c9  | 0
 _c10 | 0
 _c11 | 1
 _c12 | 0
 _c13 | 0
 _c14 | 0
 _c15 | 0
 _c16 | 0
 _c17 | 0
 _c18 | 0
 _c19 | 0
 _c20 | 0
 _c21 | 0
 _c22 | 5
 _c23 | 5
 _c24 | 0.20
 _c25 | 0.20
 _c26 | 0.00
 _c27 | 0.00
 _c28 | 1.00
 _c29 | 0.00
 _c30 | 0.00
 _c31 | 30
 _c32 | 255
 _c33 | 1.00
 _c34 | 0.00
 _c35 | 0.03
 _c36 | 0.04
 _c37 | 0.03
 _c38 | 0.01
 _c39 | 0.00
 _c40 | 0.01
 _c41 | normal
 _c42 | 21
-RECORD 4--------
 _c0  | 0
 _c1  | tcp
 _c2  | http
 _c3  | SF
 _c4  | 199
 _c5  | 420
 _c6  | 0
 _c7  | 0
 _c8  | 0
 _c9  | 0
 _c10 | 0
 _c11 | 1
 _c12 | 0
 _c13 | 0
 _c14 | 0
 _c15 | 0
 _c16 | 0
 _c17 | 0
 _c18 | 0
 _c19 | 0
 _c20 | 0
 _c21 | 0
 _c22 | 30
 _c23 | 32
 _c24 | 0.00
 _c25 | 0.00
 _c26 | 0.00
 _c27 | 0.00
 _c28 | 1.00
 _c29 | 0.00
 _c30 | 0.09
 _c31 | 255
 _c32 | 255
 _c33 | 1.00
 _c34 | 0.00
 _c35 | 0.00
 _c36 | 0.00
 _c37 | 0.00
 _c38 | 0.00
 _c39 | 0.00
 _c40 | 0.00
 _c41 | normal
 _c42 | 21
only showing top 5 rows
```

Avoiding Auto-assigned Column Names: Read CSV and Specify the Column Names

In [6]:
```
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)

# on the Cloud, the files will have to be at the root level. So, the cloud version is:
# df = spark.read.csv("/KDDTrain+.txt",header=False, inferSchema= True).toDF(*col_names)

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

```
-RECORD 0--------------------------------
 duration                    | 0
 protocol_type               | tcp
 service                     | ftp_data
 flag                        | SF
 src_bytes                   | 491
 dst_bytes                   | 0
 land                        | 0
 wrong_fragment              | 0
 urgent                      | 0
 hot                         | 0
 num_failed_logins           | 0
 logged_in                   | 0
 num_compromised             | 0
 root_shell                  | 0
 su_attempted                | 0
 num_root                    | 0
 num_file_creations          | 0
 num_shells                  | 0
 num_access_files            | 0
 num_outbound_cmds           | 0
 is_host_login               | 0
 is_guest_login              | 0
 count                       | 2
 srv_count                   | 2
 serror_rate                 | 0.0
 srv_serror_rate             | 0.0
 rerror_rate                 | 0.0
 srv_rerror_rate             | 0.0
 same_srv_rate               | 1.0
 diff_srv_rate               | 0.0
 srv_diff_host_rate          | 0.0
 dst_host_count              | 150
 dst_host_srv_count          | 25
 dst_host_same_srv_rate      | 0.17
 dst_host_diff_srv_rate      | 0.03
 dst_host_same_src_port_rate | 0.17
 dst_host_srv_diff_host_rate | 0.0
 dst_host_serror_rate        | 0.0
 dst_host_srv_serror_rate    | 0.0
 dst_host_rerror_rate        | 0.05
 dst_host_srv_rerror_rate    | 0.0
 classes                       | normal
 difficulty_level              | 20
only showing top 1 row
```

More ways to Display Dataframes

1. `df.take(5)` will return a list of five Row objects.
2. `df.collect()` will get all of the data from the entire DataFrame. Be really careful when using it, because if you have a large data set, you can easily crash the driver node.
3. `df.show()` is the most commonly used method to view a dataframe. There are a few parameters we can pass to this method, like the number of rows and truncaiton. For example, `df.show(5, False)` or `df.show(5, truncate=False)` will show the entire data wihtout any truncation.
4. `df.limit(5)` will **return a new DataFrame** by taking the first n rows. As spark is distributed in nature, there is no guarantee that `df.limit()` will give you the same results each time.

Schemas and Creating DataFrames

You can think about the dataframe as a table. A schema in Spark defines the column names and associated data types for a DataFrame. **Most often, schemas come into play when you are reading structured data from an external data source.** Defining a schema up front as opposed to taking a schema-on-read approach offers three benefits:

- You relieve Spark from the onus of inferring data types.
- You prevent Spark from creating a separate job just to read a large portion of your file to assert the schema, which for a large data file can be expensive and time-consuming.

- You can detect errors early if data don't match the schema.
  </ul> We will explore the creation of schemas and we want you to leverage them. That said, we will automatically infer the schemas when running the code in the lecture for simplicity.

You may also create your own schema using data field name followed by data type.

In [2]:
```
schema = "id INT, firstName STRING, WEBSITE STRING"
data = [[1, "John", "https://tinyurl.1"],
        [2, "Brooke", "https://tinyurl.2"]]

test_df = spark.createDataFrame(data, schema)
test_df.show()
test_df.printSchema()
```

```
+---+---------+-----------------+
| id|firstName|          WEBSITE|
+---+---------+-----------------+
|  1|     John|https://tinyurl.1|
|  2|   Brooke|https://tinyurl.2|
+---+---------+-----------------+

root
 |-- id: integer (nullable = true)
 |-- firstName: string (nullable = true)
 |-- WEBSITE: string (nullable = true)
```

Display Schema Information for Your Dataframe

In [7]:
```
df.printSchema() # or df.dtypes
```

```
root
 |-- duration: integer (nullable = true)
 |-- protocol_type: string (nullable = true)
 |-- service: string (nullable = true)
 |-- flag: string (nullable = true)
 |-- src_bytes: integer (nullable = true)
 |-- dst_bytes: integer (nullable = true)
 |-- land: integer (nullable = true)
 |-- wrong_fragment: integer (nullable = true)
 |-- urgent: integer (nullable = true)
 |-- hot: integer (nullable = true)
 |-- num_failed_logins: integer (nullable = true)
 |-- logged_in: integer (nullable = true)
 |-- num_compromised: integer (nullable = true)
 |-- root_shell: integer (nullable = true)
 |-- su_attempted: integer (nullable = true)
 |-- num_root: integer (nullable = true)
 |-- num_file_creations: integer (nullable = true)
 |-- num_shells: integer (nullable = true)
 |-- num_access_files: integer (nullable = true)
 |-- num_outbound_cmds: integer (nullable = true)
 |-- is_host_login: integer (nullable = true)
 |-- is_guest_login: integer (nullable = true)
 |-- count: integer (nullable = true)
 |-- srv_count: integer (nullable = true)
 |-- serror_rate: double (nullable = true)
 |-- srv_serror_rate: double (nullable = true)
 |-- rerror_rate: double (nullable = true)
 |-- srv_rerror_rate: double (nullable = true)
 |-- same_srv_rate: double (nullable = true)
 |-- diff_srv_rate: double (nullable = true)
 |-- srv_diff_host_rate: double (nullable = true)
 |-- dst_host_count: integer (nullable = true)
 |-- dst_host_srv_count: integer (nullable = true)
 |-- dst_host_same_srv_rate: double (nullable = true)
 |-- dst_host_diff_srv_rate: double (nullable = true)
 |-- dst_host_same_src_port_rate: double (nullable = true)
 |-- dst_host_srv_diff_host_rate: double (nullable = true)
 |-- dst_host_serror_rate: double (nullable = true)
 |-- dst_host_srv_serror_rate: double (nullable = true)
 |-- dst_host_rerror_rate: double (nullable = true)
 |-- dst_host_srv_rerror_rate: double (nullable = true)
 |-- classes: string (nullable = true)
 |-- difficulty_level: integer (nullable = true)
```

Print Column Names in Your Dataframe

In [5]:
```
print(df.columns)
```

```
['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wron
g_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised', 'root
_shell', 'su_attempted', 'num_root', 'num_file_creations', 'num_shells', 'num_access_fil
es', 'num_outbound_cmds', 'is_host_login', 'is_guest_login', 'count', 'srv_count', 'serr
or_rate', 'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_sr
v_rate', 'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_host_same_sr
v_rate', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_hos
t_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate', 'ds
t_host_srv_rerror_rate', 'classes', 'difficulty_level']
```

Print Total Number of Your Records in Your Dataframe

In [7]:
```
print(df.count())
```

```
125973
```

Print Sample Record from Your Dataframe

In [8]:
```
df.show(1, vertical=True)
```

```
-RECORD 0-------------------------------
 duration                      | 0
 protocol_type                 | tcp
 service                       | ftp_data
 flag                          | SF
 src_bytes                     | 491
 dst_bytes                     | 0
 land                        | 0
 wrong_fragment              | 0
 urgent                      | 0
 hot                         | 0
 num_failed_logins           | 0
 logged_in                   | 0
 num_compromised             | 0
 root_shell                  | 0
 su_attempted                | 0
 num_root                    | 0
 num_file_creations          | 0
 num_shells                  | 0
 num_access_files            | 0
 num_outbound_cmds           | 0
 is_host_login               | 0
 is_guest_login              | 0
 count                       | 2
 srv_count                   | 2
 serror_rate                 | 0.0
 srv_serror_rate             | 0.0
 rerror_rate                 | 0.0
 srv_rerror_rate             | 0.0
 same_srv_rate               | 1.0
 diff_srv_rate               | 0.0
 srv_diff_host_rate          | 0.0
 dst_host_count              | 150
 dst_host_srv_count          | 25
 dst_host_same_srv_rate      | 0.17
 dst_host_diff_srv_rate      | 0.03
 dst_host_same_src_port_rate | 0.17
 dst_host_srv_diff_host_rate | 0.0
 dst_host_serror_rate        | 0.0
 dst_host_srv_serror_rate    | 0.0
 dst_host_rerror_rate        | 0.05
 dst_host_srv_rerror_rate    | 0.0
 classes                     | normal
 difficulty_level            | 20
only showing top 1 row
```

DataFrame Operations on Columns

1. Selecting Columns & Creating Subset Dataframes
2. Adding New Columns
3. Renaming Columns
4. Removing Columns

Create a Subset Dataframe from Your Dataframe

In [9]:
```
small_df = df.select("duration","protocol_type","service","classes","difficulty_level")
small_df.show(5)
```

```
+--------+-------------+--------+-------+----------------+
|duration|protocol_type| service|classes|difficulty_level|
+--------+-------------+--------+-------+----------------+
|       0|          tcp|ftp_data| normal|              20|
|       0|          udp|   other| normal|              15|
|       0|          tcp| private|neptune|              19|
|       0|          tcp|    http| normal|              21|
|       0|          tcp|    http| normal|              21|
+--------+-------------+--------+-------+----------------+
only showing top 5 rows
```

Display Summary Statistics in Your Dataframe

In [10]:
```
df.describe().show(vertical=True)
```

```
-RECORD 0-------------------------------------------
 summary                     | count
 duration                    | 125973
 protocol_type               | 125973
 service                     | 125973
 flag                        | 125973
 src_bytes                   | 125973
 dst_bytes                   | 125973
 land                        | 125973
 wrong_fragment              | 125973
 urgent                      | 125973
 hot                         | 125973
 num_failed_logins           | 125973
 logged_in                   | 125973
 num_compromised             | 125973
 root_shell                  | 125973
 su_attempted                | 125973
 num_root                    | 125973
 num_file_creations          | 125973
 num_shells                  | 125973
 num_access_files            | 125973
 num_outbound_cmds           | 125973
 is_host_login               | 125973
 is_guest_login              | 125973
 count                       | 125973
 srv_count                   | 125973
 serror_rate                 | 125973
 srv_serror_rate             | 125973
 rerror_rate                 | 125973
 srv_rerror_rate             | 125973
 same_srv_rate               | 125973
 diff_srv_rate               | 125973
 srv_diff_host_rate          | 125973
 dst_host_count              | 125973
 dst_host_srv_count          | 125973
 dst_host_same_srv_rate      | 125973
 dst_host_diff_srv_rate      | 125973
 dst_host_same_src_port_rate | 125973
 dst_host_srv_diff_host_rate | 125973
 dst_host_serror_rate        | 125973
 dst_host_srv_serror_rate    | 125973
 dst_host_rerror_rate        | 125973
 dst_host_srv_rerror_rate    | 125973
 classes                     | 125973
 difficulty_level            | 125973
-RECORD 1-------------------------------------------
 summary                     | mean
 duration                    | 287.1446500440571
 protocol_type               | null
 service                     | null
 flag                        | null
 src_bytes                   | 45566.74300048423
 dst_bytes                   | 19779.114421344257
 land                        | 1.984552245322410...
 wrong_fragment              | 0.022687401268525795
 urgent                      | 1.111349257380549...
 hot                         | 0.20440888126820828
 num_failed_logins           | 0.001222484183118...
 logged_in                   | 0.3957355941352512
 num_compromised             | 0.279250315543807
 root_shell                  | 0.001341557317837...
 su_attempted                | 0.001103411048399...
 num_root                    | 0.30219173949973405
 num_file_creations          | 0.012669381534138267
 num_shells                  | 4.127868670270613...
 num_access_files            | 0.004096115834345455
 num_outbound_cmds           | 0.0
 is_host_login               | 7.938208981289641E-6
 is_guest_login              | 0.009422654060790804
 count                       | 84.1075547934875
 srv_count                   | 27.737888277646796
 serror_rate                 | 0.28448453239979987
 srv_serror_rate             | 0.282485373849952
 rerror_rate                 | 0.11995848316702792
 srv_rerror_rate             | 0.12118326943075095
 same_srv_rate               | 0.6609276591015695
 diff_srv_rate               | 0.06305263826374652
 srv_diff_host_rate          | 0.09732164828971333
 dst_host_count              | 182.14894461511594
 dst_host_srv_count          | 115.65300500900987
 dst_host_same_srv_rate      | 0.5212416946488765
 dst_host_diff_srv_rate      | 0.08295110857087815
 dst_host_same_src_port_rate | 0.14837885896185457
 dst_host_srv_diff_host_rate | 0.03254244957252493
 dst_host_serror_rate        | 0.28445246203552055
 dst_host_srv_serror_rate    | 0.2784845165233873
 dst_host_rerror_rate        | 0.11883181316631444
 dst_host_srv_rerror_rate    | 0.12023989267541528
 classes                     | null
 difficulty_level            | 19.50406039389393
-RECORD 2-------------------------------------------
 summary                     | stddev
 duration                    | 2604.515309867593
 protocol_type               | null
 service                     | null
 flag                        | null
 src_bytes                   | 5870331.181893545
 dst_bytes                   | 4021269.151441453
 land                        | 0.01408607167151309
 wrong_fragment              | 0.25352998595201326
 urgent                      | 0.014366026620154241
 hot                         | 2.1499684337047613
 num_failed_logins           | 0.04523913898132976
 logged_in                   | 0.48901005300524086
 num_compromised             | 23.942042242794997
 root_shell                  | 0.03660284383979861
 su_attempted                | 0.045154383813865565
 num_root                    | 24.39961808883742
 num_file_creations          | 0.48393506939604286
 num_shells                  | 0.022181128678694186
 num_access_files            | 0.09936955575066152
 num_outbound_cmds           | 0.0
 is_host_login               | 0.002817482738419...
 is_guest_login              | 0.09661232709143097
 count                       | 114.50860735418416
 srv_count                   | 72.63583964723834
 serror_rate                 | 0.4464556243310231
 srv_serror_rate             | 0.44702249836401775
 rerror_rate                 | 0.3204355207495169
 srv_rerror_rate             | 0.3236472280054634
 same_srv_rate               | 0.4396228624074803
 diff_srv_rate               | 0.180314407508575
 srv_diff_host_rate          | 0.2598304981211588
 dst_host_count              | 99.20621303459787
 dst_host_srv_count          | 110.70274078086481
 dst_host_same_srv_rate      | 0.4489493637176793
 dst_host_diff_srv_rate      | 0.18892179990461475
 dst_host_same_src_port_rate | 0.30899713037298815
 dst_host_srv_diff_host_rate | 0.11256380488118997
 dst_host_serror_rate        | 0.444784050316489
 dst_host_srv_serror_rate    | 0.4456691238860298
 dst_host_rerror_rate        | 0.3065574580251692
 dst_host_srv_rerror_rate    | 0.31945939045523153
 classes                     | null
 difficulty_level            | 2.291502939101359
-RECORD 3-------------------------------------------
 summary                     | min
 duration                    | 0
 protocol_type               | icmp
 service                     | IRC
 flag                        | OTH
 src_bytes                   | 0
 dst_bytes                   | 0
 land                        | 0
 wrong_fragment              | 0
 urgent                      | 0
 hot                         | 0
 num_failed_logins           | 0
 logged_in                   | 0
 num_compromised             | 0
 root_shell                  | 0
 su_attempted                | 0
 num_root                    | 0
 num_file_creations          | 0
 num_shells                  | 0
 num_access_files            | 0
 num_outbound_cmds           | 0
 is_host_login               | 0
 is_guest_login              | 0
 count                       | 0
 srv_count                   | 0
 serror_rate                 | 0.0
 srv_serror_rate             | 0.0
 rerror_rate                 | 0.0
 srv_rerror_rate             | 0.0
 same_srv_rate               | 0.0
 diff_srv_rate               | 0.0
 srv_diff_host_rate          | 0.0
 dst_host_count              | 0
 dst_host_srv_count          | 0
 dst_host_same_srv_rate      | 0.0
 dst_host_diff_srv_rate      | 0.0
 dst_host_same_src_port_rate | 0.0
 dst_host_srv_diff_host_rate | 0.0
 dst_host_serror_rate        | 0.0
 dst_host_srv_serror_rate    | 0.0
 dst_host_rerror_rate        | 0.0
 dst_host_srv_rerror_rate    | 0.0
 classes                     | back
 difficulty_level            | 0
-RECORD 4--------------------------------------------
 summary                     | max
 duration                    | 42908
 protocol_type               | udp
 service                     | whois
 flag                        | SH
 src_bytes                   | 1379963888
 dst_bytes                   | 1309937401
 land                        | 1
 wrong_fragment              | 3
 urgent                      | 3
 hot                         | 77
 num_failed_logins           | 5
 logged_in                   | 1
 num_compromised             | 7479
 root_shell                  | 1
 su_attempted                | 2
 num_root                    | 7468
 num_file_creations          | 43
 num_shells                  | 2
 num_access_files            | 9
 num_outbound_cmds           | 0
 is_host_login               | 1
 is_guest_login              | 1
 count                       | 511
 srv_count                   | 511
 serror_rate                 | 1.0
 srv_serror_rate             | 1.0
 rerror_rate                 | 1.0
 srv_rerror_rate             | 1.0
 same_srv_rate               | 1.0
 diff_srv_rate               | 1.0
 srv_diff_host_rate          | 1.0
 dst_host_count              | 255
 dst_host_srv_count          | 255
 dst_host_same_srv_rate      | 1.0
 dst_host_diff_srv_rate      | 1.0
 dst_host_same_src_port_rate | 1.0
 dst_host_srv_diff_host_rate | 1.0
 dst_host_serror_rate        | 1.0
 dst_host_srv_serror_rate    | 1.0
 dst_host_rerror_rate        | 1.0
 dst_host_srv_rerror_rate    | 1.0
 classes                     | warezmaster
 difficulty_level            | 21
```

Display Unique Values from a Column in Your Dataframe

In [11]:
```
df.select("classes").distinct().show(40)
```

```
+----------------+
|         classes|
+----------------+
|         neptune|
|           satan|
|            nmap|
|      portsweep|
|           back|
|    warezclient|
|   guess_passwd|
|         normal|
|        rootkit|
|           perl|
|buffer_overflow|
|       multihop|
|        ipsweep|
|    warezmaster|
|           imap|
|       teardrop|
|            spy|
|           land|
|            pod|
|      ftp_write|
|          smurf|
|     loadmodule|
|            phf|
+---------------+
```

Add a Column to Your Dataframe

In [12]:
```
# We will add a new column called 'first_column' at the end
from pyspark.sql.functions import lit
df = df.withColumn('first_column',lit(1))
# lit means literal. It populates the row with the literal value given.
# When adding static data / constant values, it is a good practice to use it.
df.show(1,vertical=True)
```

```
-RECORD 0-------------------------------
 duration                    | 0
 protocol_type               | tcp
 service                     | ftp_data
 flag                        | SF
 src_bytes                   | 491
 dst_bytes                   | 0
 land                        | 0
 wrong_fragment              | 0
 urgent                      | 0
 hot                         | 0
 num_failed_logins           | 0
 logged_in                   | 0
 num_compromised             | 0
 root_shell                  | 0
 su_attempted                | 0
 num_root                    | 0
 num_file_creations          | 0
 num_shells                  | 0
 num_access_files            | 0
 num_outbound_cmds           | 0
 is_host_login               | 0
 is_guest_login              | 0
 count                       | 2
 srv_count                   | 2
 serror_rate                 | 0.0
 srv_serror_rate             | 0.0
 rerror_rate                 | 0.0
 srv_rerror_rate             | 0.0
 same_srv_rate               | 1.0
 diff_srv_rate               | 0.0
 srv_diff_host_rate          | 0.0
 dst_host_count              | 150
 dst_host_srv_count          | 25
 dst_host_same_srv_rate      | 0.17
 dst_host_diff_srv_rate      | 0.03
 dst_host_same_src_port_rate | 0.17
 dst_host_srv_diff_host_rate | 0.0
 dst_host_serror_rate        | 0.0
 dst_host_srv_serror_rate    | 0.0
 dst_host_rerror_rate        | 0.05
 dst_host_srv_rerror_rate    | 0.0
 classes                     | normal
 difficulty_level            | 20
 first_column                | 1
only showing top 1 row
```

Renaming a Column in Your Dataframe

In [13]:
```
df = df.withColumnRenamed('first_column', 'new_column_one')

df.printSchema()
```

```
root
 |-- duration: integer (nullable = true)
 |-- protocol_type: string (nullable = true)
 |-- service: string (nullable = true)
 |-- flag: string (nullable = true)
 |-- src_bytes: integer (nullable = true)
 |-- dst_bytes: integer (nullable = true)
 |-- land: integer (nullable = true)
 |-- wrong_fragment: integer (nullable = true)
 |-- urgent: integer (nullable = true)
 |-- hot: integer (nullable = true)
 |-- num_failed_logins: integer (nullable = true)
 |-- logged_in: integer (nullable = true)
 |-- num_compromised: integer (nullable = true)
 |-- root_shell: integer (nullable = true)
 |-- su_attempted: integer (nullable = true)
 |-- num_root: integer (nullable = true)
 |-- num_file_creations: integer (nullable = true)
 |-- num_shells: integer (nullable = true)
 |-- num_access_files: integer (nullable = true)
 |-- num_outbound_cmds: integer (nullable = true)
 |-- is_host_login: integer (nullable = true)
 |-- is_guest_login: integer (nullable = true)
 |-- count: integer (nullable = true)
 |-- srv_count: integer (nullable = true)
 |-- serror_rate: double (nullable = true)
 |-- srv_serror_rate: double (nullable = true)
 |-- rerror_rate: double (nullable = true)
 |-- srv_rerror_rate: double (nullable = true)
 |-- same_srv_rate: double (nullable = true)
 |-- diff_srv_rate: double (nullable = true)
 |-- srv_diff_host_rate: double (nullable = true)
 |-- dst_host_count: integer (nullable = true)
 |-- dst_host_srv_count: integer (nullable = true)
 |-- dst_host_same_srv_rate: double (nullable = true)
 |-- dst_host_diff_srv_rate: double (nullable = true)
 |-- dst_host_same_src_port_rate: double (nullable = true)
 |-- dst_host_srv_diff_host_rate: double (nullable = true)
 |-- dst_host_serror_rate: double (nullable = true)
 |-- dst_host_srv_serror_rate: double (nullable = true)
 |-- dst_host_rerror_rate: double (nullable = true)
 |-- dst_host_srv_rerror_rate: double (nullable = true)
 |-- classes: string (nullable = true)
 |-- difficulty_level: integer (nullable = true)
 |-- new_column_one: integer (nullable = false)
```

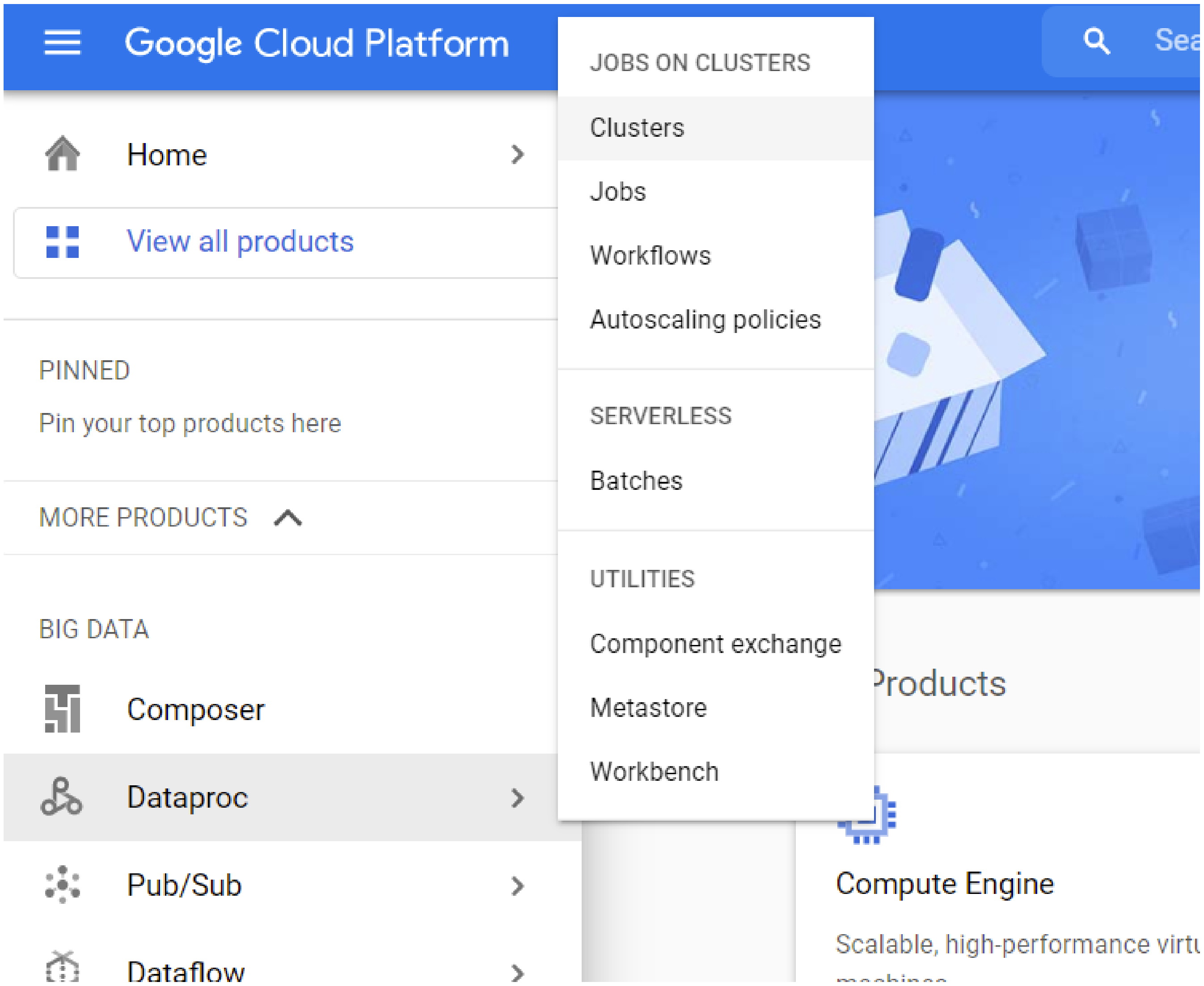
Delete a Column from Your Dataframe

In [ ]:
```
df = df.drop('new_column_one')
df.printSchema()
```

Lab: Run Spark on the Cloud

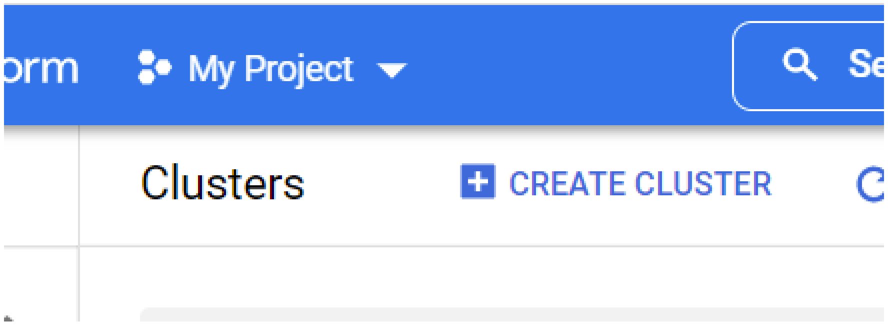
Create Clusters (e.g. Hadoop Clusters)

- A cluster is group of machines, servers, or nodes. It helps providing the sum of the computational power offered by all incorporated machines. It's difficult to build a local machine with 64GB RAM and 20TB of Storage but that is not difficult when you are running on the cloud.
- You may start by navigating to Dataproc and click on the Clusters section



Cluster - Setup

- Next, you need to create your cluster and choose the cluster to use "Google Compute Engine"

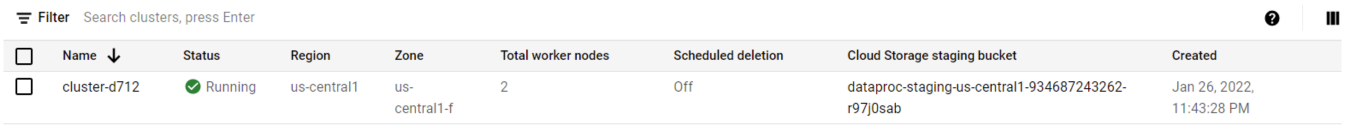


Cluster Configuration

Make sure to follow the cluster creation guide posted on Canvas

Running Cluster

Once you click on the create button, Google Cloud will work on creating your own cluster and if it's successful, you will see your cluster running.



Name	Status	Region	Zone	Total worker nodes	Scheduled deletion	Cloud Storage staging bucket	Created
cluster-d712	Running	us-central1	us-central1-f	2	Off	dataproc-staging-us-central1-934687243262-r97j0sab	Jan 26, 2022, 11:43:28 PM

Connect to Your Cluster

From the Web Interfaces, open Jupyter. Upload your Notebook to `GCS` folder and run the cells.