> **Algoritmos para Big Data**
>
> **2023/2024**

Structured API

In this lecture we will overview structured APIs available in Spark, namely the DataFrame.

Notice that whenever we need to carry out data analysis, it is almost certain that we have to put the original data as if it was holding some sort of structure, resembling a table. Structured APIs help to do so.

Disclaimer Some content presented in this notebook is based on references mentioned at the end of the notebook.

Datasets

For this notebook, you need to download two datasets:

- one related to customer churn analysis in the Telcom industry;
- a second one related to retail sales in a particular day, to be used in an additional exercise.

Hence, use the following `wget` commands in the Terminal to download the datasets:

> wget bigdata.iscte.me/abd/telco-customer-churn.csv.zip
>
> wget bigdata.iscte.me/abd/retail-data-2010-12-01.csv.zip

Initial settings

Some settings that are needed or helpful:

In [1]:
```
from IPython.core.display import HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))
```

In [2]:
```
import findspark
findspark.init()
findspark.find()

import pyspark
from pyspark.sql import SparkSession, Row
import pyspark.sql.functions as F

# why F above?

# build our SparkSession

spark = SparkSession\
        .builder\
        .appName("BigData")\
        .config("spark.sql.shuffle.partitions",6)\
        .config("spark.sql.repl.eagereval.enabled",True)\
        .getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/02/27 18:05:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-
```

Structured API - the case of DataFrame

Priori to any explanation, recall that there are both low-level and high-level APIs related to (distributed) collections of data. As mentioned in the previous lecture, we can have collections of Resilient Distributed Dataset (RDD), DataFrame and Dataset.

The two structured APIs available - **DataFrame** and **Dataset** - are distributed table-like collections with well-defined rows and columns. Each column must have the same number of rows as all the other columns. Each column has a type information that is consistent for every row in the collection. Furthermore, they are immutable and lazy evaluated plans specify transformations/operations to apply to data in order to generate output. When we call an action, we instruct Spark to do all those transformations/operations and return the result.

Hereafter we will focus on **DataFrames**.

Columns and rows

- **Columns** represent a simple type like an integer or string, a complex type like an array or map, or a null value
- **Row** is a record of data

In [3]:
```
# Creates two rows and print them

myrows = spark.range(2).collect()
print(type(myrows))
myrows
```

Out[3]:
```
<class 'list'>
[Row(id=0), Row(id=1)]
```

Basic data types

The basic data types available in Spark are as follows:

Data Type	Value in Python	API to access or create a data type
`ByteType`	int or long Ensure number within a range -128 to 127	ByteType()
`ShortType`	int or long	ShortType()
`IntegerType`	int or long	IntegerType()
`LongType`	long	LongType()
`FloatType`	float	FloatType()
`DoubleType`	float	DoubleType()
`DecimalType`	decimal	DecimalType()
`StringType`	string	StringType()
`BinaryType`	bytearray	BinaryType()
`BooleanType`	decimal	BooleanType()
`TimeStampType`	datetime.datetime	TimeStampType()
`DateType`	datetime.date	DateType()
`ArrayType`	list, tuple, or array	ArrayType()
`MapType`	dict	MapType()
`StructType`	list or tuple	StructType()
`ArrayType`	list, tuple, or array	ArrayType()
`StructField`	list, tuple, or array	StructField(name, dataType, [nullable])

Operations

There are plenty of operations we can apply on columns, rows and data frames.

But first, just check the documentation in

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

in particular the functions we are about to use:

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

Columns and expressions

Columns are similar to columns in Pandas dataframes. And we may apply expression operations to columns.

But we cannot manipulate individual columns outside the context of DataFrames. That is, we need transformations to modify the content of a column in a DataFrame.

In [4]:
```
# to create a column (no different to Pandas)

F.col("ColumnName")
```

Out[4]:
```
Column<'ColumnName'>
```

In [5]:
```
F.column("ColumnName")
```

Out[5]:
```
Column<'ColumnName'>
```

In [6]:
```
(((F.col("someCol") + 5) * 200) -6) < F.col("otherCol")
```

Out[6]:
```
Column<'((((someCol + 5) * 200) - 6) < otherCol)'>
```

In [7]:
```
F.expr("(((someCol + 5)* 200) -6 ) < otherCol")
```

Out[7]:
```
Column<'((((someCol + 5) * 200) - 6) < otherCol)'>
```

Records and rows

Creation

In [8]:
```
# Creating a row

a_row =
a_row
```

Out[8]:
```
<Row('555555', '85123B', 'New Product', 2016, 6, 2.1, 141131, 'Lisbon')>
```

In [9]:
```
# Check the type
```

Out[9]:
```
pyspark.sql.types.Row
```

In [10]:
```
# Creating a DataFrame

# create a new row and assign to the columns name numbers like _1, _2 etc..
# If we want a different column name, we need to define the schema

new_df = spark.createDataFrame([a_row])

# checking
new_df.take(1)
```

Out[10]:
```
[Row(_1='555555', _2='85123B', _3='New Product', _4=2016, _5=6, _6=2.1, _7=141131, _8='Lisbon')]
```

In [11]:
```
# check the type
```

Out[11]:
```
pyspark.sql.dataframe.DataFrame
```

In [12]:
```
# Define explicit schema for our data
"""
schema = (StructType([
   StructField("Id", IntegerType(), False),
   StructField("First", StringType(), False),
   StructField("Last", StringType(), False),
   StructField("Url", StringType(), False),
   StructField("Published", StringType(), False),
   StructField("Hits", IntegerType(), False),
   StructField("Campaigns", ArrayType(StringType()), False)]))
   """

our_schema = "`Id` INT,`First` STRING,`Last` STRING,`Url` STRING,`Published` STRING,`Hits` INT,`Campaigns` ARRAY<

# create our data
data = [[1, "Jules", "Damji", "https://tinyurl.1", "1/4/2016", 4535, ["twitter", "LinkedIn"]],
       [2, "Brooke","Wenig","https://tinyurl.2", "5/5/2018", 8908, ["twitter", "LinkedIn"]],
       [3, "Denny", "Lee", "https://tinyurl.3","6/7/2019",7659, ["web", "twitter", "FB", "LinkedIn"]],
       [4, "Tathagata", "Das","https://tinyurl.4", "5/12/2018", 10568, ["twitter", "FB"]],
       [5, "Matei","Zaharia", "https://tinyurl.5", "5/14/2014", 40578, ["web", "twitter", "FB", "LinkedIn"]],
       [6, "Reynold", "Xin", "https://tinyurl.6", "3/2/2015", 25568, ["twitter", "LinkedIn"]]
      ]
```

In [13]:
```
# Create a DataFrame using the schema defined above

blogs_df = spark.createDataFrame(data, our_schema)

# and show the DataFrame; it should reflect our table above
blogs_df.
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|
+---+---------+-------+-----------------+---------+-----+--------------------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
+---+---------+-------+-----------------+---------+-----+--------------------+
```

In [14]:
```
# Show the value of "Hits" * 2

blogs_df.select("Id", F.expr("Hits") * 2).show(5)
```

```
+---+----------+
| Id|(Hits * 2)|
+---+----------+
|  1|      9070|
|  2|     17816|
|  3|     15318|
|  4|     21136|
|  5|     81156|
+---+----------+
only showing top 5 rows
```

In [15]:
```
# Show the value of "Hits" + "Id"

blogs_df.select(               ).show(truncate=False)
```

```
+---+-----------+
|Id |(Hits + Id)|
+---+-----------+
|1  |4536       |
|2  |8910       |
|3  |7662       |
|4  |10572      |
|5  |40583      |
|6  |25574      |
+---+-----------+
```

In [16]:
```
# Creates a column saying whether it is true of false
# that "Hits" are higher than 10000

blogs_df.withColumn("Big Hitters",                  ).show()
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+-----------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|Big Hitters|
+---+---------+-------+-----------------+---------+-----+--------------------+-----------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|      false|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|      false|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|      false|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|       true|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|       true|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|       true|
+---+---------+-------+-----------------+---------+-----+--------------------+-----------+
```

In [17]:

```
# Show the value of "First" + "Last" + "Id"

( blogs_df.withColumn("AuthorsId", (F.concat(F.expr("Id"), F.expr("First"), F.expr("Last"))))
            .select(F.expr("AuthorsId"))
            .show(n=4)
)
```

```
+-------------+
|    AuthorsId|
+-------------+
|  1JulesDamji|
| 2BrookeWenig|
|    3DennyLee|
|4TathagataDas|
+-------------+
only showing top 4 rows
```

Collecting rows to the Driver and showing

Spark maintains the state of the cluster in the Driver. In order to get the results we need we have to call actions:

Action	Description
collect()	Gets all data from the entire DataFrame and returns a list
take()	Selects the first rows
first()	Selects the first row
show()	Prints out a certain number of rows nicely
limit()	Returns a limited number of rows
toLocalIterator()	Similar to collect but returns an iterator

In [18]:
```
# Examples of collecting rows: see available actions above

rows_collected = blogs_df.limit(4)

rows_collected.show()
```

```
+---+---------+-----+-----------------+---------+-----+--------------------+
| Id|    First| Last|              Url|Published| Hits|           Campaigns|
+---+---------+-----+-----------------+---------+-----+--------------------+
|  1|    Jules|Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  2|   Brooke|Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  3|    Denny|  Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  4|Tathagata|  Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
+---+---------+-----+-----------------+---------+-----+--------------------+
```

In [19]:
```
rows_collected.first()
```

Out[19]:
```
Row(Id=1, First='Jules', Last='Damji', Url='https://tinyurl.1', Published='1/4/2016', Hits=4535, Campaigns=['twit
```

In [20]:
```
# Selecting "Published" and "Campaigns"

blogs_df.
```

```
+---------+--------------------------+
|Published|Campaigns                 |
+---------+--------------------------+
|1/4/2016 |[twitter, LinkedIn]       |
|5/5/2018 |[twitter, LinkedIn]       |
|6/7/2019 |[web, twitter, FB, LinkedIn]|
|5/12/2018|[twitter, FB]             |
|5/14/2014|[web, twitter, FB, LinkedIn]|
|3/2/2015 |[twitter, LinkedIn]       |
+---------+--------------------------+
```

Adding/changing columns of a DataFrame

In [21]:
```
# Adding explicit values to compare with (literals)

blogs_df.select(F.expr("*"), F.lit("Portugal").alias("Country")).show()
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+--------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns| Country|
+---+---------+-------+-----------------+---------+-----+--------------------+--------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|Portugal|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|Portugal|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|Portugal|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|Portugal|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|Portugal|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|Portugal|
+---+---------+-------+-----------------+---------+-----+--------------------+--------+
```

In [22]:
```
# Adding/changing a column - "OtherCountry"

df_new_col = blogs_df.withColumn
df_new_col.show()
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+------------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|OtherCountry|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|    Portugal|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|    Portugal|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|    Portugal|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|    Portugal|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|    Portugal|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|    Portugal|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+
```

In [23]:
```
df_new_col = df_new_col.withColumn
df_new_col.show()
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+------------+--------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|OtherCountry| Country|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+--------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|    Portugal|Portugal|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|    Portugal|Portugal|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|    Portugal|Portugal|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|    Portugal|Portugal|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|    Portugal|Portugal|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|    Portugal|Portugal|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+--------+
```

In [24]:
```
# Check if "Country" has the same value of "OtherCountry"

df_new_col.withColumn("SameCountry", F.expr("Country == OtherCountry")).show()
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+------------+--------+-----------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|OtherCountry| Country|SameCountry|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+--------+-----------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|    Portugal|Portugal|       true|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|    Portugal|Portugal|       true|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|    Portugal|Portugal|       true|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|    Portugal|Portugal|       true|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|    Portugal|Portugal|       true|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|    Portugal|Portugal|       true|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+--------+-----------+
```

In [25]:
```
# Rename columns: from "Country" to "OriginCountry"

df_renamed_col = df_new_col.withColumnRenamed("Country", "OriginCountry")
df_renamed_col
```

Out[25]:
```
DataFrame[Id: int, First: string, Last: string, Url: string, Published: string, Hits: int, Campaigns: array<strin
```

In [26]:
```
# Remove columns: "OriginCountry"

df_dropped = df_renamed_col.
df_dropped.show()
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+------------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|OtherCountry|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|    Portugal|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|    Portugal|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|    Portugal|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|    Portugal|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|    Portugal|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|    Portugal|
+---+---------+-------+-----------------+---------+-----+--------------------+------------+
```

In [27]:
```
# Changing a column type: "Id"

df_coltype = df_dropped.withColumn("Id-str",                    )

# Checking
df_coltype.show(truncate=False)
df_dropped.printSchema()
df_coltype.printSchema()
```

```
+---+---------+-------+-----------------+---------+-----+---------------------------+------------+------+
|Id |First    |Last   |Url              |Published|Hits |Campaigns                  |OtherCountry|Id-str|
+---+---------+-------+-----------------+---------+-----+---------------------------+------------+------+
|1  |Jules    |Damji  |https://tinyurl.1|1/4/2016 |4535 |[twitter, LinkedIn]        |Portugal    |1     |
|2  |Brooke   |Wenig  |https://tinyurl.2|5/5/2018 |8908 |[twitter, LinkedIn]        |Portugal    |2     |
|3  |Denny    |Lee    |https://tinyurl.3|6/7/2019 |7659 |[web, twitter, FB, LinkedIn]|Portugal    |3     |
|4  |Tathagata|Das    |https://tinyurl.4|5/12/2018|10568|[twitter, FB]              |Portugal    |4     |
|5  |Matei    |Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB, LinkedIn]|Portugal    |5     |
|6  |Reynold  |Xin    |https://tinyurl.6|3/2/2015 |25568|[twitter, LinkedIn]        |Portugal    |6     |
+---+---------+-------+-----------------+---------+-----+---------------------------+------------+------+

root
 |-- Id: integer (nullable = true)
 |-- First: string (nullable = true)
 |-- Last: string (nullable = true)
 |-- Url: string (nullable = true)
 |-- Published: string (nullable = true)
 |-- Hits: integer (nullable = true)
 |-- Campaigns: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- OtherCountry: string (nullable = false)

root
 |-- Id: integer (nullable = true)
 |-- First: string (nullable = true)
 |-- Last: string (nullable = true)
 |-- Url: string (nullable = true)
 |-- Published: string (nullable = true)
 |-- Hits: integer (nullable = true)
 |-- Campaigns: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- OtherCountry: string (nullable = false)
 |-- Id-str: string (nullable = true)
```

Filtering Rows

To filter rows, we can create an expression that evaluates to true or false then we filter with that expression. To do so we can use `filter` or `where`.

Also, if we want unique rows, we should use `distinct` for that matter.

In [28]:
```
# Filtering "Hits" < 8000

blogs_df.filter(             ).show(truncate=False)
blogs_df.where(              ).show(truncate=False)
```

```
+---+-----+-----+----------------+---------+----+--------------------------+
|Id |First|Last |Url             |Published|Hits|Campaigns                 |
+---+-----+-----+----------------+---------+----+--------------------------+
|1  |Jules|Damji|https://tinyurl.1|1/4/2016 |4535|[twitter, LinkedIn]       |
|3  |Denny|Lee  |https://tinyurl.3|6/7/2019 |7659|[web, twitter, FB, LinkedIn]|
+---+-----+-----+----------------+---------+----+--------------------------+

+---+-----+-----+----------------+---------+----+--------------------------+
|Id |First|Last |Url             |Published|Hits|Campaigns                 |
+---+-----+-----+----------------+---------+----+--------------------------+
|1  |Jules|Damji|https://tinyurl.1|1/4/2016 |4535|[twitter, LinkedIn]       |
|3  |Denny|Lee  |https://tinyurl.3|6/7/2019 |7659|[web, twitter, FB, LinkedIn]|
+---+-----+-----+----------------+---------+----+--------------------------+
```

In [29]:
```
# Use distinct to get the distinct "Country" that appear in the DataFrame,

blogs_df.select("Campaigns").distinct().show(truncate=False)

# and its counting

blogs_df.select("Campaigns").distinct().count()
```

```
+---------------------------+
|Campaigns                  |
+---------------------------+
|[twitter, LinkedIn]        |
|[web, twitter, FB, LinkedIn]|
|[twitter, FB]              |
+---------------------------+
```

Out[29]:
```
3
```

Random Sample

Sometimes we need to get a random sample of our data, like for example to training a model in machine learning. We can do that with the function `sample()`.

Also, we can have random splits, very helpful when we need to break our dataset into random splits of the original. This is often the case of creating both validation and test sets to be used by machine learning algorithms.

In [30]:
```
# Simple random sample

seed = 2
with_replacement = False
# reduce to 50%
fraction = 0.5
df_random = blogs_df.sample(withReplacement=with_replacement, fraction=fraction, seed=seed)
```

In [31]:
```
# and recalling also the original DataFrame
```

```
+---+---------+-------+-----------------+---------+-----+--------------------------+
|Id |First    |Last   |Url              |Published|Hits |Campaigns                 |
+---+---------+-------+-----------------+---------+-----+--------------------------+
|1  |Jules    |Damji  |https://tinyurl.1|1/4/2016 |4535 |[twitter, LinkedIn]       |
|2  |Brooke   |Wenig  |https://tinyurl.2|5/5/2018 |8908 |[twitter, LinkedIn]       |
|3  |Denny    |Lee    |https://tinyurl.3|6/7/2019 |7659 |[web, twitter, FB, LinkedIn]|
|4  |Tathagata|Das    |https://tinyurl.4|5/12/2018|10568|[twitter, FB]             |
|5  |Matei    |Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB, LinkedIn]|
|6  |Reynold  |Xin    |https://tinyurl.6|3/2/2015 |25568|[twitter, LinkedIn]       |
+---+---------+-------+-----------------+---------+-----+--------------------------+

+---+---------+-------+-----------------+---------+-----+--------------------------+
|Id |First    |Last   |Url              |Published|Hits |Campaigns                 |
+---+---------+-------+-----------------+---------+-----+--------------------------+
|1  |Jules    |Damji  |https://tinyurl.1|1/4/2016 |4535 |[twitter, LinkedIn]       |
|3  |Denny    |Lee    |https://tinyurl.3|6/7/2019 |7659 |[web, twitter, FB, LinkedIn]|
|4  |Tathagata|Das    |https://tinyurl.4|5/12/2018|10568|[twitter, FB]             |
|5  |Matei    |Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB, LinkedIn]|
+---+---------+-------+-----------------+---------+-----+--------------------------+

4
6
```

Out[31]:

In [32]:
```
# Split a DataFrame into two but randomly, say 40% - 60%

df_randoms = blogs_df.randomSplit
```

In [33]:
```
# Check the counting of both parts

df_randoms[0].count()
```

Out[33]:
```
2
```

In [34]:
```
df_randoms[1].count()
```

Out[34]:
```
4
```

Sorting Rows

To sort data, we use the functions `sort` and/or `orderBy`. Additionally, we may specify the direction with the keywords:

Keyword	Description
desc	Data is sorted from the biggest value to the lowest value
asc	Data is sorted from the lowest value to the biggest value

In [35]:
```
# Ordering/sorting one column, say "Hits"

blogs_df.orderBy(F.col("Hits")).show()
blogs_df.orderBy("Hits").show()
blogs_df.sort("Hits").show()
```

```
+---+---------+-------+-----------------+---------+-----+--------------------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|
+---+---------+-------+-----------------+---------+-----+--------------------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|
+---+---------+-------+-----------------+---------+-----+--------------------+

+---+---------+-------+-----------------+---------+-----+--------------------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|
+---+---------+-------+-----------------+---------+-----+--------------------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|
+---+---------+-------+-----------------+---------+-----+--------------------+

+---+---------+-------+-----------------+---------+-----+--------------------+
| Id|    First|   Last|              Url|Published| Hits|           Campaigns|
+---+---------+-------+-----------------+---------+-----+--------------------+
|  1|    Jules|  Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  3|    Denny|    Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  2|   Brooke|  Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  4|Tathagata|    Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  6|  Reynold|    Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
|  5|    Matei|Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|
+---+---------+-------+-----------------+---------+-----+--------------------+
```

In [36]:

```
# Sorting the DataFrame by "Hits" but ascending

blogs_df.orderBy(F.expr("Hits asc")).show(5)
blogs_df.orderBy(F.col("Hits").asc()).show(5)
```

```
+---+---------+-----+-----------------+---------+-----+--------------------+
| Id|    First| Last|              Url|Published| Hits|           Campaigns|
+---+---------+-----+-----------------+---------+-----+--------------------+
|  1|    Jules|Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  3|    Denny|  Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  2|   Brooke|Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  4|Tathagata|  Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  6|  Reynold|  Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
+---+---------+-----+-----------------+---------+-----+--------------------+
only showing top 5 rows
```


```
+---+---------+-----+-----------------+---------+-----+--------------------+
| Id|    First| Last|              Url|Published| Hits|           Campaigns|
+---+---------+-----+-----------------+---------+-----+--------------------+
|  1|    Jules|Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  3|    Denny|  Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  2|   Brooke|Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  4|Tathagata|  Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  6|  Reynold|  Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
+---+---------+-----+-----------------+---------+-----+--------------------+
only showing top 5 rows
```


In [37]:
```
# Check with expression

F.expr('Hits asc')
```

Out[37]:
```
Column<'Hits AS asc'>
```

In [38]:
```
# Check using asc

F.col("Hits").asc()
```

Out[38]:
```
Column<'Hits ASC NULLS FIRST'>
```

In [39]:
```
# Sorting the DataFrame by "Hits" but descending

blogs_df.
blogs_df.
```

```
+---+---------+-----+-----------------+---------+-----+--------------------+
| Id|    First| Last|              Url|Published| Hits|           Campaigns|
+---+---------+-----+-----------------+---------+-----+--------------------+
|  1|    Jules|Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  3|    Denny|  Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  2|   Brooke|Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  4|Tathagata|  Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  6|  Reynold|  Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
+---+---------+-----+-----------------+---------+-----+--------------------+
only showing top 5 rows

+---+---------+-----+-----------------+---------+-----+--------------------+
| Id|    First| Last|              Url|Published| Hits|           Campaigns|
+---+---------+-----+-----------------+---------+-----+--------------------+
|  1|    Jules|Damji|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
|  3|    Denny|  Lee|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
|  2|   Brooke|Wenig|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
|  4|Tathagata|  Das|https://tinyurl.4|5/12/2018|10568|       [twitter, FB]|
|  6|  Reynold|  Xin|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
+---+---------+-----+-----------------+---------+-----+--------------------+
only showing top 5 rows
```

In [40]:

```
# Check how sorting will play out

# show differences if any between expr <Colname desc> and <Colname>.desc()

blogs_df.orderBy(F.expr("Hits desc_nulls_last")).explain()
blogs_df.orderBy(F.col("Hits").desc()).explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Sort [Hits#23 ASC NULLS FIRST], true, 0
   +- Exchange rangepartitioning(Hits#23 ASC NULLS FIRST, 6), ENSURE_REQUIREMENTS, [plan_id=583]
      +- Scan ExistingRDD[Id#18,First#19,Last#20,Url#21,Published#22,Hits#23,Campaigns#24]


== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Sort [Hits#23 DESC NULLS LAST], true, 0
   +- Exchange rangepartitioning(Hits#23 DESC NULLS LAST, 6), ENSURE_REQUIREMENTS, [plan_id=593]
      +- Scan ExistingRDD[Id#18,First#19,Last#20,Url#21,Published#22,Hits#23,Campaigns#24]
```

In [41]:
```
# Sorting the DataFrame by "Campaigns" but ascending
# and "Hits" but descending

blogs_df.orderBy(                              ).show(truncate=False)
```

```
+---+---------+-------+-----------------+---------+-----+--------------------------+
|Id |First    |Last   |Url              |Published|Hits |Campaigns                 |
+---+---------+-------+-----------------+---------+-----+--------------------------+
|4  |Tathagata|Das    |https://tinyurl.4|5/12/2018|10568|[twitter, FB]             |
|6  |Reynold  |Xin    |https://tinyurl.6|3/2/2015 |25568|[twitter, LinkedIn]       |
|2  |Brooke   |Wenig  |https://tinyurl.2|5/5/2018 |8908 |[twitter, LinkedIn]       |
|1  |Jules    |Damji  |https://tinyurl.1|1/4/2016 |4535 |[twitter, LinkedIn]       |
|5  |Matei    |Zaharia|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB, LinkedIn]|
|3  |Denny    |Lee    |https://tinyurl.3|6/7/2019 |7659 |[web, twitter, FB, LinkedIn]|
+---+---------+-------+-----------------+---------+-----+--------------------------+
```

API Execution

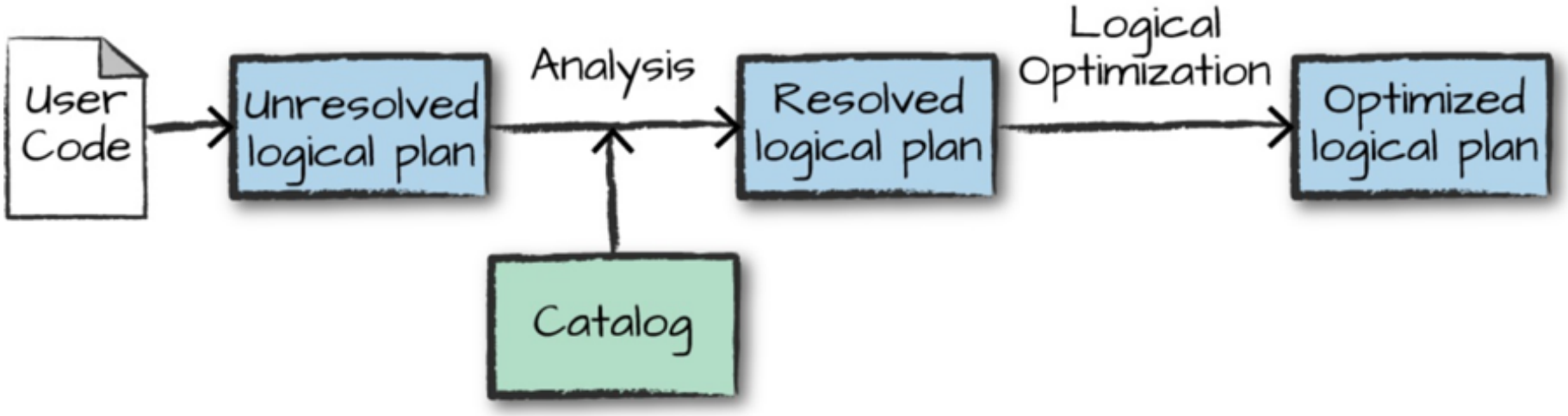
Worth mentioning that any code to be executed by Spark is subject in advance to a procedure of optimization. This process is carried out by the *Catalyst Optimizer*, which will decide how code will be executed and lays out a plan to do so.

There are three levels to be considered:

1. Logical planning

Takes the submitted code and convert it into logical plan, which represents a set set of abstract transformations. It converts the given code into the most optimized version.

The analyser may reject the logical plan if the required table or column name does not exist.
If the analyser can resolve it the result is passed through the Catalyst Optimizer, which is a set of rules that attempt to optimize the logical plan.

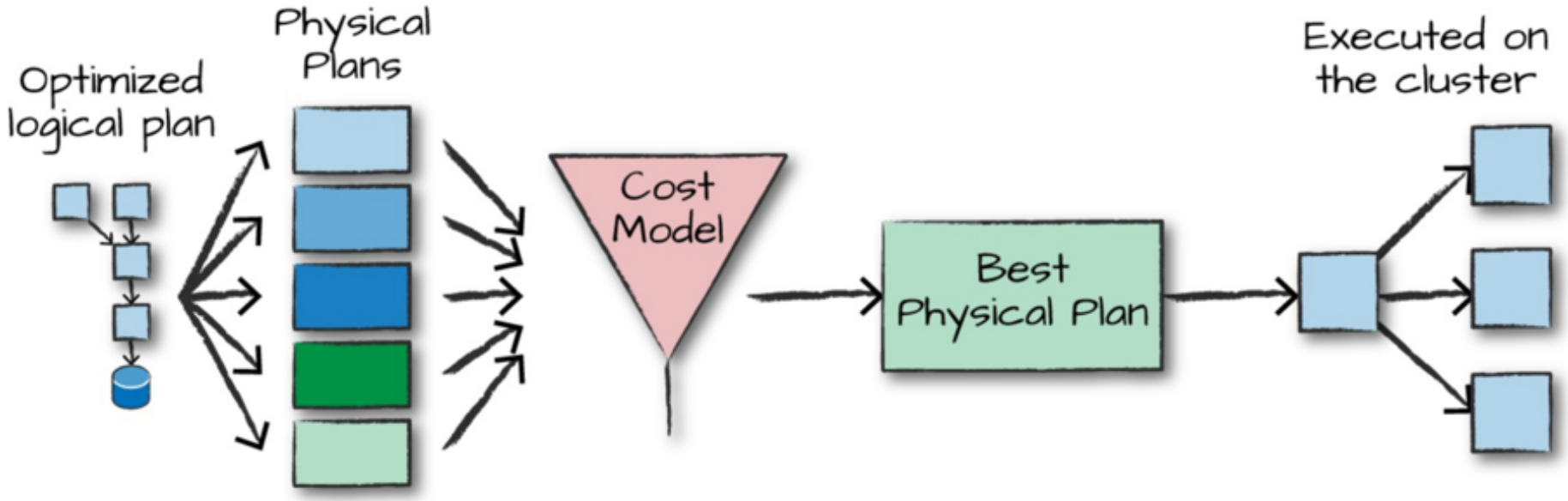


2. Physical planning

After successfully creating the optimized logical plan, Spark begins the physical planning process. This specifies how logical plan will execute in the cluster by generating different physical execution strategies and comparing them using a cost model.

3. Execution

After selecting the physical plan to execute, Spark runs all of his this code over RDDs. Spark performs further optimizations at runtime generate native java bytecode, so it can remove entire tasks or stages if that is the case.


Optimized logical plan
Physical Plans
Cost Model
Best Physical Plan
Executed on the cluster

Exercise

This exercise is about **customer churn** in the Telcom industry. Basically, custom churn - also known as customer attrition, customer turnover, or customer defection - relates to loss of customers.

Our goal is to write down a Spark program that:

a) Reads a file containing the dataset under analysis.

b) Provide answers about such data, as follows:

1. Number of rows and number of columns?
2. Minimum, maximum and average value of the columns "tenure" and "TotalCharges"?
3. Number of unique values in each column, starting with "customerID"?

Note: The dataset we are using that you have already downloaded can also be found in the link
https://www.kaggle.com/blastchar/telco-customer-churn

Context

Companies in the Telcom industry often use customer churn analysis and customer churn rates as one of their key business metrics. This is because the cost of retaining an existing customer is far less than acquiring a new one. Usually, they have customer service branches which attempt to win back defecting clients, because recovered long-term customers can be worth much more to a company than newly recruited clients.

On the other hand, companies usually make a distinction between voluntary churn and involuntary churn. Voluntary churn occurs due to a decision by the customer to switch to another company, whereas involuntary churn occurs due to circumstances such as a customer's relocation to a long-term care facility, death, or the relocation to a distant location. Not surprisingly, data analysis tends to focus on voluntary churn.

Column Description

Column	Type	Description
customerID	String	Customer ID
gender	String	Whether the customer is a male or a female
SeniorCitizen	Integer	Whether the customer is a senior citizen or not (1, 0)
Partner	String	Whether the customer has a partner or not (Yes, No)
Dependents	String	Whether the customer has dependents or not (Yes, No)
tenure	Integer	Number of months the customer has stayed with the company
PhoneService	String	Whether the customer has a phone service or not (Yes, No)
MultipleLines	String	Whether the customer has multiple lines or not (Yes, No, No phone service)
InternetService	String	Customer’s internet service provider (DSL, Fiber optic, No)
OnlineSecurity	String	Whether the customer has online security or not (Yes, No, No internet service)
OnlineBackup	String	Whether the customer has online backup or not (Yes, No, No internet service)
DeviceProtection	String	Whether the customer has device protection or not (Yes, No, No internet service)
TechSupport	String	Whether the customer has tech support or not (Yes, No, No internet service)
StreamingTV	String	Whether the customer has streaming movies or not (Yes, No, No internet service)
StreamingMovies	String	Whether the customer has a partner or not (Yes, No)
Contract	String	The contract term of the customer (Month-to-month, One year, Two year)
PaperlessBilling	String	Whether the customer has paperless billing or not (Yes, No)
PaymentMethod	String	The customer’s payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
MonthlyCharges	Double	The amount charged to the customer monthly
TotalCharges	String	The total amount charged to the customer
Churn	String	Whether the customer churned or not (Yes or No)

Reading the dataset

Some directory and file checking first...

In [ ]:
```
pwd
```

In [ ]:
```
ls -la
```

In [44]:
```
! head -n 3 ../Datasets/telco-customer-churn.csv
```

```
customerID,gender,SeniorCitizen,Partner,Dependents,tenure,PhoneService,MultipleLines,InternetService,OnlineSecuri
7590-VHVEG,Female,0,Yes,No,1,No,No phone service,DSL,No,Yes,No,No,No,No,Month-to-month,Yes,Electronic check,29.85
5575-GNVDE,Male,0,No,No,34,Yes,No,DSL,Yes,No,Yes,No,No,No,One year,No,Mailed check,56.95,1889.5,No
```

In [45]:
```
# Read and create the dataframe

file_path = "../Datasets/telco-customer-churn.csv"

df = ( spark.read.format("csv")
            .option("header","true")
            .option("inferSchema","true")
            .load(file_path)
     )
```

Checking data

Schema and show

In [46]:
```
# Check the schema and some content of the dataframe

df.printSchema()
df.show(5)
```

```
root
 |-- customerID: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- SeniorCitizen: integer (nullable = true)
 |-- Partner: string (nullable = true)
 |-- Dependents: string (nullable = true)
 |-- tenure: integer (nullable = true)
 |-- PhoneService: string (nullable = true)
 |-- MultipleLines: string (nullable = true)
 |-- InternetService: string (nullable = true)
 |-- OnlineSecurity: string (nullable = true)
 |-- OnlineBackup: string (nullable = true)
 |-- DeviceProtection: string (nullable = true)
 |-- TechSupport: string (nullable = true)
 |-- StreamingTV: string (nullable = true)
 |-- StreamingMovies: string (nullable = true)
 |-- Contract: string (nullable = true)
 |-- PaperlessBilling: string (nullable = true)
 |-- PaymentMethod: string (nullable = true)
 |-- MonthlyCharges: double (nullable = true)
 |-- TotalCharges: string (nullable = true)
 |-- Churn: string (nullable = true)

+----------+------+-------------+-------+----------+------+------------+----------------+---------------+--------
|customerID|gender|SeniorCitizen|Partner|Dependents|tenure|PhoneService|   MultipleLines|InternetService|OnlineSe
+----------+------+-------------+-------+----------+------+------------+----------------+---------------+--------
|7590-VHVEG|Female|            0|    Yes|        No|     1|          No|No phone service|            DSL|
|5575-GNVDE|  Male|            0|     No|        No|    34|         Yes|              No|            DSL|
|3668-QPYBK|  Male|            0|     No|        No|     2|         Yes|              No|            DSL|
|7795-CFOCW|  Male|            0|     No|        No|    45|          No|No phone service|            DSL|
|9237-HQITU|Female|            0|     No|        No|     2|         Yes|              No|    Fiber optic|
+----------+------+-------------+-------+----------+------+------------+----------------+---------------+--------
only showing top 5 rows
```

Note on using a pre-defined schema to read a data file:

In [47]:
```
# when reading the dataset, above, we could have set and use a schema with specified data type,
# as follows:

from pyspark.sql.types import StructField, StructType, StringType, IntegerType, TimestampType, DoubleType

schema = StructType([StructField("CustomerID", StringType(), True),
                             StructField("gender", StringType(), True),
                             #... and the other fields

                            ])

# then reading in accordance to the explicit schema above
df_own = spark.read.format("csv").schema(schema).load(file_path)
```

Question: why and when would have been advisable to do set the schema prior to reading the file?

In the case of CSV and JSON, the inference type relies on full-data-read. So, if files are big, then it is problematic as the reading task would be very time consuming.

Another aspect worth considering relates to when we read the same file frequently and therefore the schema does not change. For example, that may be the case of large datasets that are fed into ML models on a daily basis!

Questions to be anwsered

Number of rows and number of columns?

In [48]:
```
# Number of rows



# Number of columns
```

Out[48]:
```
7043
21
```

Minimum, maximum and average value of "tenure" and "TotalCharges"?

In [49]:
```
df.
```

```
+-------+------------------+------------------+
|summary|            tenure|      TotalCharges|
+-------+------------------+------------------+
|  count|              7043|              7043|
|   mean| 32.37114865824223|2283.3004408418697|
| stddev|24.559481023094442| 2266.771361883145|
|    min|                 0|                  |
|    max|                72|             999.9|
+-------+------------------+------------------+
```

Number of unique values in each column, starting with "customerID"?

In [50]:
```
# list with all the distinct numbers of each column

# df.columns

distinct_count = [df.select(column).distinct().count() for column in df.columns]
distinct_count
```

Out[50]:
```
[7043, 2, 2, 2, 2, 73, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 4, 1585, 6531, 2]
```

In [51]:
```
# printing out the result but differently

# for column in df.columns:
#    df.groupBy().agg(countDistinct(col(column))).show()
```

Answers regarding the questions above

Number of rows and number of columns?

answer:

7043

21

Minimum, maximum and average value of "tenure" and "TotalCharges"?

answer:

summary	tenure	TotalCharges
count	7043	7043
mean	32.37114865824223	2283.3004408418697
stddev	24.559481023094442	2266.771361883145
min	0	
max	72	999.9

Number of unique values in each column, starting with "customerID"?

answer:

[7043, 2, 2, 2, 2, 73, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 4, 1585, 6531, 2]

Additional exercise

Using the given dataset `retail-data-2010-12-01.csv` write down code to anwser the following questions:

1. Aggregate the counting of invoices by country and provide the result in descending order.

   `Hint: use in sequence the transformations/actions: .select().groupBy().agg().orderBy()`
2. Find the aggregate count of invoices for the country "United Kingdom" by filtering on "Country".

   `Hint: use in sequence the transformations/actions: .select().where().groupBy().agg().orderBy()`

References

- Learning Spark - Lightning-Fast Data Analytics, 2nd Ed. J. Damji, B. Wenig, T. Das, and D. Lee. O'Reilly, 2020
- Spark: The Definitive Guide - Big Data Processing Made Simple, 1st Ed. B. Chambers and M. Zaharia. O'Reilly, 2018
- https://spark.apache.org/docs/latest
- https://docs.python.org/3/
- https://www.kaggle.com/blastchar/telco-customer-churn

In [ ]: