PYSPARK.ipynb

File Edit View Insert Runtime Tools Help Last saved at February 16

Comment Share m

+ Code + Text Connect Colab AI

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://archive.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-hadoop3.2.
!tar xf spark-3.1.1-bin-hadoop3.2.tgz
!pip install -q findspark
```

```
!ls
```

```
sample_data  spark-3.1.1-bin-hadoop3.2  spark-3.1.1-bin-hadoop3.2.tgz
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.1-bin-hadoop3.2"
```

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark.conf.set("spark.sql.repl.eagerEval.enabled", True) # Property used to format o
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
v3.1.1
Master
local[*]
AppName
pyspark-shell

```
# Downloading and preprocessing Cars Data downloaded origianlly from https://perso.te
!wget https://jacobceles.github.io/knowledge_repo/colab_and_pyspark/cars.csv
```

```
--2024-02-16 10:15:13--  https://jacobceles.github.io/knowledge_repo/colab_and_pyspark/cars.csv
Resolving jacobceles.github.io (jacobceles.github.io)... 185.199.108.153, 185.199.109.153, 185.199.110.153, ...
Connecting to jacobceles.github.io (jacobceles.github.io)|185.199.108.153|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://jacobcelestine.com/knowledge_repo/colab_and_pyspark/cars.csv [following]
--2024-02-16 10:15:14--  https://jacobcelestine.com/knowledge_repo/colab_and_pyspark/cars.csv
Resolving jacobcelestine.com (jacobcelestine.com)... 185.199.108.153, 185.199.109.153, 185.199.110.153, ...
Connecting to jacobcelestine.com (jacobcelestine.com)|185.199.108.153|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 22608 (22K) [text/csv]
Saving to: ‘cars.csv’

cars.csv            100%[===================>]  22.08K  --.-KB/s    in 0.001s

2024-02-16 10:15:14 (22.7 MB/s) - ‘cars.csv’ saved [22608/22608]
```

```
!ls
```

```
cars.csv  sample_data  spark-3.1.1-bin-hadoop3.2  spark-3.1.1-bin-hadoop3.2.tgz
```

```
# Load data from csv to a dataframe.
# header=True means the first row is a header
# sep=';' means the column are seperated using ''
df = spark.read.csv('cars.csv', header=True, sep=";")
df.show(5)
```

```
+--------------------+----+---------+------------+----------+------+------------+-----+------+
|                 Car| MPG|Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|
+--------------------+----+---------+------------+----------+------+------------+-----+------+
|Chevrolet Chevell...|18.0|        8|       307.0|     130.0| 3504.|        12.0|   70|    US|
|   Buick Skylark 320|15.0|        8|       350.0|     165.0| 3693.|        11.5|   70|    US|
|  Plymouth Satellite|18.0|        8|       318.0|     150.0| 3436.|        11.0|   70|    US|
|       AMC Rebel SST|16.0|        8|       304.0|     150.0| 3433.|        12.0|   70|    US|
|         Ford Torino|17.0|        8|       302.0|     140.0| 3449.|        10.5|   70|    US|
+--------------------+----+---------+------------+----------+------+------------+-----+------+
only showing top 5 rows
```

The above command loads our data from into a dataframe (DF). A dataframe is a 2-dimensional labeled data structure with columns of potentially different types.

Viewing the Dataframe

There are a couple of ways to view your dataframe(DF) in PySpark:

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.

Let us see some of them in action below:

[ ]
```
df.show(5, truncate=False)
```

```
+-------------------------+----+---------+------------+----------+------+------------+-----+------+
|Car                      |MPG |Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|
+-------------------------+----+---------+------------+----------+------+------------+-----+------+
|Chevrolet Chevelle Malibu|18.0|8        |307.0       |130.0     |3504. |12.0        |70   |US    |
|Buick Skylark 320        |15.0|8        |350.0       |165.0     |3693. |11.5        |70   |US    |
|Plymouth Satellite       |18.0|8        |318.0       |150.0     |3436. |11.0        |70   |US    |
|AMC Rebel SST            |16.0|8        |304.0       |150.0     |3433. |12.0        |70   |US    |
|Ford Torino              |17.0|8        |302.0       |140.0     |3449. |10.5        |70   |US    |
+-------------------------+----+---------+------------+----------+------+------------+-----+------+
only showing top 5 rows
```

[ ]
```
df.limit(5)
```

Car	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model	Origin
Chevrolet Chevell...	18.0	8	307.0	130.0	3504.	12.0	70	US
Buick Skylark 320	15.0	8	350.0	165.0	3693.	11.5	70	US
Plymouth Satellite	18.0	8	318.0	150.0	3436.	11.0	70	US
AMC Rebel SST	16.0	8	304.0	150.0	3433.	12.0	70	US
Ford Torino	17.0	8	302.0	140.0	3449.	10.5	70	US

Viewing Dataframe Columns

[ ]
```
df.columns
```

```
['Car',
 'MPG',
 'Cylinders',
 'Displacement',
 'Horsepower',
 'Weight',
 'Acceleration',
 'Model',
 'Origin']
```

Dataframe Schema

There are two methods commonly used to view the data types of a dataframe:

[ ]
```
df.dtypes
```

```
[('Car', 'string'),
 ('MPG', 'string'),
 ('Cylinders', 'string'),
 ('Displacement', 'string'),
 ('Horsepower', 'string'),
 ('Weight', 'string'),
 ('Acceleration', 'string'),
 ('Model', 'string'),
 ('Origin', 'string')]
```

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

```
root
 |-- Car: string (nullable = true)
 |-- MPG: string (nullable = true)
 |-- Cylinders: string (nullable = true)
 |-- Displacement: string (nullable = true)
 |-- Horsepower: string (nullable = true)
 |-- Weight: string (nullable = true)
```

```
|-- Weight: string (nullable = true)
|-- Acceleration: string (nullable = true)
|-- Model: string (nullable = true)
|-- Origin: string (nullable = true)
```

Inferring Schema Implicitly

We can use the parameter inferschema=true to infer the input schema automatically while loading the data. An example is shown below:

```
[ ] df = spark.read.csv('cars.csv', header=True, sep=";", inferSchema=True)
    df.printSchema()
```

```
root
 |-- Car: string (nullable = true)
 |-- MPG: double (nullable = true)
 |-- Cylinders: integer (nullable = true)
 |-- Displacement: double (nullable = true)
 |-- Horsepower: double (nullable = true)
 |-- Weight: decimal(4,0) (nullable = true)
 |-- Acceleration: double (nullable = true)
 |-- Model: integer (nullable = true)
 |-- Origin: string (nullable = true)
```

As you can see, the datatype has been infered automatically spark with even the correct precison for decimal type. A problem that might arise here is that sometimes, when you have to read multiple files with different schemas in different files, there might be an issue with implicit inferring leading to null values in some columns. Therefore, let us also see how to define schemas explicitly.

Defining Schema Explicitly

```
[ ] from pyspark.sql.types import *
    df.columns
```

```
['Car',
 'MPG',
 'Cylinders',
 'Displacement',
 'Horsepower',
 'Weight',
 'Acceleration',
 'Model',
 'Origin']
```

Creating a list of the schema in the format column_name, data_type

```
[ ] # Creating a list of the schema in the format column_name, data_type
    labels = [
         ('Car',StringType()),
         ('MPG',DoubleType()),
         ('Cylinders',IntegerType()),
         ('Displacement',DoubleType()),
         ('Horsepower',DoubleType()),
         ('Weight',DoubleType()),
         ('Acceleration',DoubleType()),
         ('Model',IntegerType()),
         ('Origin',StringType())
    ]
```

Creating the schema that will be passed when reading the csv

```
[ ] schema = StructType([StructField (x[0], x[1], True) for x in labels])
```

```
schema
```

```
StructType(List(StructField(Car,StringType,true),StructField(MPG,DoubleType,true),StructField(Cylinders,IntegerType,true),StructField(Displacement,DoubleType,true),StructF
```

```
[ ] df = spark.read.csv('cars.csv', header=True, sep=";", schema=schema)
    df.printSchema()
    # The schema comes as we gave!
```

```
root
 |-- Car: string (nullable = true)
 |-- MPG: double (nullable = true)
 |-- Cylinders: integer (nullable = true)
 |-- Displacement: double (nullable = true)
 |-- Horsepower: double (nullable = true)
 |-- Weight: double (nullable = true)
 |-- Acceleration: double (nullable = true)
 |-- Model: integer (nullable = true)
 |-- Origin: string (nullable = true)
```

```
[ ] df.show(truncate=False)
```

```
+------------------------------+----+---------+------------+----------+------+------------+-----+------+
|Car                           |MPG |Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|
+------------------------------+----+---------+------------+----------+------+------------+-----+------+
|Chevrolet Chevelle Malibu     |18.0|8        |307.0       |130.0     |3504.0|12.0        |70   |US    |
|Buick Skylark 320             |15.0|8        |350.0       |165.0     |3693.0|11.5        |70   |US    |
|Plymouth Satellite            |18.0|8        |318.0       |150.0     |3436.0|11.0        |70   |US    |
|AMC Rebel SST                 |16.0|8        |304.0       |150.0     |3433.0|12.0        |70   |US    |
|Ford Torino                   |17.0|8        |302.0       |140.0     |3449.0|10.5        |70   |US    |
|Ford Galaxie 500              |15.0|8        |429.0       |198.0     |4341.0|10.0        |70   |US    |
|Chevrolet Impala              |14.0|8        |454.0       |220.0     |4354.0|9.0         |70   |US    |
|Plymouth Fury iii             |14.0|8        |440.0       |215.0     |4312.0|8.5         |70   |US    |
|Pontiac Catalina              |14.0|8        |455.0       |225.0     |4425.0|10.0        |70   |US    |
|AMC Ambassador DPL            |15.0|8        |390.0       |190.0     |3850.0|8.5         |70   |US    |
|Citroen DS-21 Pallas          |0.0 |4        |133.0       |115.0     |3090.0|17.5        |70   |Europe|
|Chevrolet Chevelle Concours (sw)|0.0 |8        |350.0       |165.0     |4142.0|11.5        |70   |US    |
|Ford Torino (sw)              |0.0 |8        |351.0       |153.0     |4034.0|11.0        |70   |US    |
|Plymouth Satellite (sw)       |0.0 |8        |383.0       |175.0     |4166.0|10.5        |70   |US    |
|AMC Rebel SST (sw)            |0.0 |8        |360.0       |175.0     |3850.0|11.0        |70   |US    |
|Dodge Challenger SE           |15.0|8        |383.0       |170.0     |3563.0|10.0        |70   |US    |
|Plymouth 'Cuda 340            |14.0|8        |340.0       |160.0     |3609.0|8.0         |70   |US    |
|Ford Mustang Boss 302         |0.0 |8        |302.0       |140.0     |3353.0|8.0         |70   |US    |
|Chevrolet Monte Carlo         |15.0|8        |400.0       |150.0     |3761.0|9.5         |70   |US    |
|Buick Estate Wagon (sw)       |14.0|8        |455.0       |225.0     |3086.0|10.0        |70   |US    |
+------------------------------+----+---------+------------+----------+------+------------+-----+------+
only showing top 20 rows
```

As we can see here, the data has been successully loaded with the specified datatypes.

DataFrame Operations on Columns

We will go over the following in this section:

1. Selecting Columns
2. Selecting Multiple Columns
3. Adding New Columns
4. Renaming Columns
5. Grouping By Columns
6. Removing Columns

Selecting Columns

There are multiple ways to do a select in PySpark. You can find how they differ and how each below:

```
[ ] # 1st method
    # Column name is case sensitive in this usage
    print(df.Car)
    print("*"*20)
    df.select(df.Car).show(truncate=False)
```

```
Column<'Car'>
********************
+------------------------------+
|Car                           |
```

```
+-------------------------------+
|Chevrolet Chevelle Malibu      |
|Buick Skylark 320              |
|Plymouth Satellite             |
|AMC Rebel SST                  |
|Ford Torino                    |
|Ford Galaxie 500               |
|Chevrolet Impala               |
|Plymouth Fury iii              |
|Pontiac Catalina               |
|AMC Ambassador DPL             |
|Citroen DS-21 Pallas           |
|Chevrolet Chevelle Concours (sw)|
|Ford Torino (sw)               |
|Plymouth Satellite (sw)        |
|AMC Rebel SST (sw)             |
|Dodge Challenger SE            |
|Plymouth 'Cuda 340             |
|Ford Mustang Boss 302          |
|Chevrolet Monte Carlo          |
|Buick Estate Wagon (sw)        |
+-------------------------------+
only showing top 20 rows
```

NOTE:

> **We can't always use the dot notation because this will break when the column names have reserved names or attributes to the data frame class. Additionally, the column names are case sensitive in nature so we need to always make sure the column names have been changed to a paticular case before using it.**

[ ]
```
# 2nd method
# Column name is case insensitive here
print(df['car'])
print("*"*20)
df.select(df['car']).show(truncate=False)
```

```
Column<'car'>
********************
+-------------------------------+
|car                            |
+-------------------------------+
|Chevrolet Chevelle Malibu      |
|Buick Skylark 320              |
|Plymouth Satellite             |
|AMC Rebel SST                  |
|Ford Torino                    |
|Ford Galaxie 500               |
|Chevrolet Impala               |
|Plymouth Fury iii              |
|Pontiac Catalina               |
|AMC Ambassador DPL             |
|Citroen DS-21 Pallas           |
|Chevrolet Chevelle Concours (sw)|
|Ford Torino (sw)               |
|Plymouth Satellite (sw)        |
|AMC Rebel SST (sw)             |
|Dodge Challenger SE            |
|Plymouth 'Cuda 340             |
|Ford Mustang Boss 302          |
|Chevrolet Monte Carlo          |
|Buick Estate Wagon (sw)        |
+-------------------------------+
only showing top 20 rows
```

We will take a look at three cases here:

Adding a new column

Adding multiple columns

[ ]
```
# CASE 1: Adding a new column
# 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(5,truncate=False)
```

```
+-------------------------+----+---------+------------+----------+------+------------+-----+------+------------+
|Car                      |MPG |Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|first_column|
+-------------------------+----+---------+------------+----------+------+------------+-----+------+------------+
|Chevrolet Chevelle Malibu|18.0|8        |307.0       |130.0     |3504.0|12.0        |70   |US    |1           |
|Buick Skylark 320        |15.0|8        |350.0       |165.0     |3693.0|11.5        |70   |US    |1           |
|Plymouth Satellite       |18.0|8        |318.0       |150.0     |3436.0|11.0        |70   |US    |1           |
```

```
|AMC Rebel SST              |16.0|8        |304.0       |150.0     |3433.0|12.0        |70   |US    |1           |
|Ford Torino                |17.0|8        |302.0       |140.0     |3449.0|10.5        |70   |US    |1           |
+-------------------------+----+---------+------------+----------+------+------------+-----+------+------------+
only showing top 5 rows
```

```
[ ] # CASE 2: Adding multiple columns
    # We will add two new columns called 'second_column' and 'third_column' at the end
    df = df.withColumn('second_column', lit(2)) \
           .withColumn('third_column', lit('Third Column'))
    # 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(5,truncate=False)
```

```
+-------------------------+----+---------+------------+----------+------+------------+-----+------+------------+-------------+------------+
|Car                      |MPG |Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|first_column|second_column|third_column|
+-------------------------+----+---------+------------+----------+------+------------+-----+------+------------+-------------+------------+
|Chevrolet Chevelle Malibu|18.0|8        |307.0       |130.0     |3504.0|12.0        |70   |US    |1           |2            |Third Column|
|Buick Skylark 320        |15.0|8        |350.0       |165.0     |3693.0|11.5        |70   |US    |1           |2            |Third Column|
|Plymouth Satellite       |18.0|8        |318.0       |150.0     |3436.0|11.0        |70   |US    |1           |2            |Third Column|
|AMC Rebel SST            |16.0|8        |304.0       |150.0     |3433.0|12.0        |70   |US    |1           |2            |Third Column|
|Ford Torino              |17.0|8        |302.0       |140.0     |3449.0|10.5        |70   |US    |1           |2            |Third Column|
+-------------------------+----+---------+------------+----------+------+------------+-----+------+------------+-------------+------------+
only showing top 5 rows
```

Renaming Columns

```
[ ] #Renaming a column in PySpark
    df = df.withColumnRenamed('first_column', 'new_column_one') \
           .withColumnRenamed('second_column', 'new_column_two') \
           .withColumnRenamed('third_column', 'new_column_three')
    df.show(truncate=False)
```

```
+-------------------------------+----+---------+------------+----------+------+------------+-----+------+--------------+--------------+----------------+
|Car                            |MPG |Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|new_column_one|new_column_two|new_column_three|
+-------------------------------+----+---------+------------+----------+------+------------+-----+------+--------------+--------------+----------------+
|Chevrolet Chevelle Malibu      |18.0|8        |307.0       |130.0     |3504.0|12.0        |70   |US    |1             |2             |Third Column    |
|Buick Skylark 320              |15.0|8        |350.0       |165.0     |3693.0|11.5        |70   |US    |1             |2             |Third Column    |
|Plymouth Satellite             |18.0|8        |318.0       |150.0     |3436.0|11.0        |70   |US    |1             |2             |Third Column    |
|AMC Rebel SST                  |16.0|8        |304.0       |150.0     |3433.0|12.0        |70   |US    |1             |2             |Third Column    |
|Ford Torino                    |17.0|8        |302.0       |140.0     |3449.0|10.5        |70   |US    |1             |2             |Third Column    |
|Ford Galaxie 500               |15.0|8        |429.0       |198.0     |4341.0|10.0        |70   |US    |1             |2             |Third Column    |
|Chevrolet Impala               |14.0|8        |454.0       |220.0     |4354.0|9.0         |70   |US    |1             |2             |Third Column    |
|Plymouth Fury iii              |14.0|8        |440.0       |215.0     |4312.0|8.5         |70   |US    |1             |2             |Third Column    |
|Pontiac Catalina               |14.0|8        |455.0       |225.0     |4425.0|10.0        |70   |US    |1             |2             |Third Column    |
|AMC Ambassador DPL             |15.0|8        |390.0       |190.0     |3850.0|8.5         |70   |US    |1             |2             |Third Column    |
|Citroen DS-21 Pallas           |0.0 |4        |133.0       |115.0     |3090.0|17.5        |70   |Europe|1             |2             |Third Column    |
|Chevrolet Chevelle Concours (sw)|0.0 |8        |350.0       |165.0     |4142.0|11.5        |70   |US    |1             |2             |Third Column    |
|Ford Torino (sw)               |0.0 |8        |351.0       |153.0     |4034.0|11.0        |70   |US    |1             |2             |Third Column    |
|Plymouth Satellite (sw)        |0.0 |8        |383.0       |175.0     |4166.0|10.5        |70   |US    |1             |2             |Third Column    |
|AMC Rebel SST (sw)             |0.0 |8        |360.0       |175.0     |3850.0|11.0        |70   |US    |1             |2             |Third Column    |
|Dodge Challenger SE            |15.0|8        |383.0       |170.0     |3563.0|10.0        |70   |US    |1             |2             |Third Column    |
|Plymouth 'Cuda 340             |14.0|8        |340.0       |160.0     |3609.0|8.0         |70   |US    |1             |2             |Third Column    |
|Ford Mustang Boss 302          |0.0 |8        |302.0       |140.0     |3353.0|8.0         |70   |US    |1             |2             |Third Column    |
|Chevrolet Monte Carlo          |15.0|8        |400.0       |150.0     |3761.0|9.5         |70   |US    |1             |2             |Third Column    |
|Buick Estate Wagon (sw)        |14.0|8        |455.0       |225.0     |3086.0|10.0        |70   |US    |1             |2             |Third Column    |
+-------------------------------+----+---------+------------+----------+------+------------+-----+------+--------------+--------------+----------------+
only showing top 20 rows
```

Grouping By Columns

Here, we see the Dataframe API way of grouping values. We will discuss how to:

Group By a single column

Group By multiple columns

```
[ ] # Group By a column in PySpark
    df.groupBy('Origin').count().show(5)
```

```
+------+-----+
|Origin|count|
+------+-----+
|Europe|   73|
|    US|  254|
| Japan|   79|
+------+-----+
```

```
[ ] # Group By multiple columns in PySpark
    df.groupBy('Origin', 'Model').count().show(5)
```

```
+------+-----+-----+
|Origin|Model|count|
+------+-----+-----+
|Europe|   71|    5|
|Europe|   80|    9|
|Europe|   79|    4|
| Japan|   75|    4|
|    US|   72|   18|
+------+-----+-----+
only showing top 5 rows
```

Removing Columns

```
[ ] #Remove columns in PySpark
    df = df.drop('new_column_one')
    df.show(5,truncate=False)
```

```
+-------------------------+----+---------+------------+----------+------+------------+-----+------+--------------+----------------+
|Car                      |MPG |Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|new_column_two|new_column_three|
+-------------------------+----+---------+------------+----------+------+------------+-----+------+--------------+----------------+
|Chevrolet Chevelle Malibu|18.0|8        |307.0       |130.0     |3504.0|12.0        |70   |US    |2             |Third Column    |
|Buick Skylark 320        |15.0|8        |350.0       |165.0     |3693.0|11.5        |70   |US    |2             |Third Column    |
|Plymouth Satellite       |18.0|8        |318.0       |150.0     |3436.0|11.0        |70   |US    |2             |Third Column    |
|AMC Rebel SST            |16.0|8        |304.0       |150.0     |3433.0|12.0        |70   |US    |2             |Third Column    |
|Ford Torino              |17.0|8        |302.0       |140.0     |3449.0|10.5        |70   |US    |2             |Third Column    |
+-------------------------+----+---------+------------+----------+------+------------+-----+------+--------------+----------------+
only showing top 5 rows
```

Remove multiple columnss in one go

Double-click (or enter) to edit

```
[ ]
    df = df.drop('new_column_two') \
           .drop('new_column_three')
    df.show(5,truncate=False)
```

```
+-------------------------+----+---------+------------+----------+------+------------+-----+------+
|Car                      |MPG |Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|
+-------------------------+----+---------+------------+----------+------+------------+-----+------+
|Chevrolet Chevelle Malibu|18.0|8        |307.0       |130.0     |3504.0|12.0        |70   |US    |
|Buick Skylark 320        |15.0|8        |350.0       |165.0     |3693.0|11.5        |70   |US    |
|Plymouth Satellite       |18.0|8        |318.0       |150.0     |3436.0|11.0        |70   |US    |
|AMC Rebel SST            |16.0|8        |304.0       |150.0     |3433.0|12.0        |70   |US    |
|Ford Torino              |17.0|8        |302.0       |140.0     |3449.0|10.5        |70   |US    |
+-------------------------+----+---------+------------+----------+------+------------+-----+------+
only showing top 5 rows
```

DataFrame Operations on Rows

We will discuss the follwoing in this section:

Filtering Rows

Get Distinct Rows

Sorting Rows

Union Dataframes

Filtering Rows

```
[ ] # Filtering rows in PySpark
    total_count = df.count()
    print("TOTAL RECORD COUNT: " + str(total_count))
    europe_filtered_count = df.filter(col('Origin')=='Europe').count()
```

```
print("EUROPE FILTERED RECORD COUNT: " + str(europe_filtered_count))
df.filter(col('Origin')=='Europe').show(truncate=False)
```

```
TOTAL RECORD COUNT: 406
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-04d62147d69b> in <cell line: 4>()
      2 total_count = df.count()
      3 print("TOTAL RECORD COUNT: " + str(total_count))
----> 4 europe_filtered_count = df.filter(col('Origin')=='Europe').count()
      5 print("EUROPE FILTERED RECORD COUNT: " + str(europe_filtered_count))
      6 df.filter(col('Origin')=='Europe').show(truncate=False)

NameError: name 'col' is not defined
```