

Name: Feven Belay Araya

ID: 20027

CS570-Week 4: Homework 2: Introduction to Google Colab and PySpark

✓ Step 2. Introduction to Google Colab and PySpark

Installing Spark

- Install Dependencies:

1. Java 8
2. Apache Spark with hadoop and
3. Findspark (used to locate the spark in the system)

```
!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.tgz
!tar xf spark-3.1.1-bin-hadoop3.2.tgz
!pip install -q findspark
```

Set Environment Variables:

```
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"
```

```
!ls
```

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

```
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 output tables better
spark
```

```
📁 SparkSession - in-memory
```

SparkContext

[Spark UI](#)

Version

v3.1.1

Master

local[*]

AppName

pyspark-shell

Step 2.1: Exploring the Dataset

✓ Loading the Dataset

```
# Downloading and preprocessing Cars Data downloaded origianlly from https://perso.telecom-paristech.fr/eagan/class/igr204/dataset
# Many of these datasets have been cleaned up by Petra Isenberg, Pierre Dragicevic and Yvonne Jansen
!wget https://jacobceles.github.io/knowledge_repo/colab_and_pyspark/cars.csv
```

```
--2024-06-12 05:45:45-- 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]
--2024-06-12 05:45:45-- 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
```

```
2024-06-12 05:45:46 (15.2 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
```

✓ 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 truncation. For example, `df.show(5, False)` or `df.show(5, truncate=False)` will show the entire data without 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)
```

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)
|-- 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)
```

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
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),St
```

```
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

✓ Step 2.2: 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
```

```
# 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
```

```
# 3rd method
# Column name is case insensitive here
from pyspark.sql.functions import col
df.select(col('car')).show(truncate=False)
```

```

+-----+
|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
```

Selecting Multiple Columns

```
# 1st method
# Column name is case sensitive in this usage
print(df.Car, df.Cylinders)
print("*"*40)
df.select(df.Car, df.Cylinders).show(truncate=False)
```

```

Column<'Car'> Column<'Cylinders'>
*****
+-----+-----+
|Car|Cylinders|
+-----+-----+
|Chevrolet Chevelle Malibu|8|
|Buick Skylark 320|8|
|Plymouth Satellite|8|
|AMC Rebel SST|8|
|Ford Torino|8|
|Ford Galaxie 500|8|
|Chevrolet Impala|8|
|Plymouth Fury iii|8|
|Pontiac Catalina|8|
|AMC Ambassador DPL|8|
|Citroen DS-21 Pallas|4|
|Chevrolet Chevelle Concours (sw)|8|
|Ford Torino (sw)|8|
|Plymouth Satellite (sw)|8|
|AMC Rebel SST (sw)|8|
|Dodge Challenger SE|8|
|Plymouth 'Cuda 340|8|
|Ford Mustang Boss 302|8|
|Chevrolet Monte Carlo|8|
|Buick Estate Wagon (sw)|8|
+-----+-----+
only showing top 20 rows
```

```
# 2nd method
# Column name is case insensitive in this usage
print(df['car'],df['cylinders'])
print("*"*40)
df.select(df['car'],df['cylinders']).show(truncate=False)
```

```

Column<'car'> Column<'cylinders'>
*****
+-----+-----+
|car|cylinders|
+-----+-----+
```

Chevrolet Chevelle Malibu	8
Buick Skylark 320	8
Plymouth Satellite	8
AMC Rebel SST	8
Ford Torino	8
Ford Galaxie 500	8
Chevrolet Impala	8
Plymouth Fury iii	8
Pontiac Catalina	8
AMC Ambassador DPL	8
Citroen DS-21 Pallas	4
Chevrolet Chevelle Concours (sw)	8
Ford Torino (sw)	8
Plymouth Satellite (sw)	8
AMC Rebel SST (sw)	8
Dodge Challenger SE	8
Plymouth 'Cuda 340	8
Ford Mustang Boss 302	8
Chevrolet Monte Carlo	8
Buick Estate Wagon (sw)	8

only showing top 20 rows

```
# 3rd method
# Column name is case insensitive in this usage
from pyspark.sql.functions import col
df.select(col('car'),col('cylinders')).show(truncate=False)
```

car	cylinders
Chevrolet Chevelle Malibu	8
Buick Skylark 320	8
Plymouth Satellite	8
AMC Rebel SST	8
Ford Torino	8
Ford Galaxie 500	8
Chevrolet Impala	8
Plymouth Fury iii	8
Pontiac Catalina	8
AMC Ambassador DPL	8
Citroen DS-21 Pallas	4
Chevrolet Chevelle Concours (sw)	8
Ford Torino (sw)	8
Plymouth Satellite (sw)	8
AMC Rebel SST (sw)	8
Dodge Challenger SE	8
Plymouth 'Cuda 340	8
Ford Mustang Boss 302	8
Chevrolet Monte Carlo	8
Buick Estate Wagon (sw)	8

only showing top 20 rows

Adding New Columns

- We will take a look at three cases here:
 1. Adding a new column
 2. Adding multiple columns
 3. Deriving a new column from an existing one

```
# 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|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Chevrolet Chevelle Malibu|18.0|8      |307.0      |130.0      |3504.0|12.0      |70      |US      |1      |2      |
|Buick Skylark 320      |15.0|8      |350.0      |165.0      |3693.0|11.5      |70      |US      |1      |2      |
|Plymouth Satellite      |18.0|8      |318.0      |150.0      |3436.0|11.0      |70      |US      |1      |2      |
|AMC Rebel SST          |16.0|8      |304.0      |150.0      |3433.0|12.0      |70      |US      |1      |2      |
|Ford Torino            |17.0|8      |302.0      |140.0      |3449.0|10.5      |70      |US      |1      |2      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
# CASE 3: Deriving a new column from an existing one
# We will add a new column called 'car_model' which has the value of car and model appended together with a space in between
from pyspark.sql.functions import concat
df = df.withColumn('car_model', concat(col("Car"), lit(" "), col("model")))
# 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|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Chevrolet Chevelle Malibu|18.0|8      |307.0      |130.0      |3504.0|12.0      |70      |US      |1      |2      |
|Buick Skylark 320      |15.0|8      |350.0      |165.0      |3693.0|11.5      |70      |US      |1      |2      |
|Plymouth Satellite      |18.0|8      |318.0      |150.0      |3436.0|11.0      |70      |US      |1      |2      |
|AMC Rebel SST          |16.0|8      |304.0      |150.0      |3433.0|12.0      |70      |US      |1      |2      |
|Ford Torino            |17.0|8      |302.0      |140.0      |3449.0|10.5      |70      |US      |1      |2      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Renaming Columns

- We use the withColumnRenamed function to rename a column in PySpark. Let us see it in action below:

```
#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      |2      |
|Buick Skylark 320      |15.0|8      |350.0      |165.0      |3693.0|11.5      |70      |US      |1      |2      |2      |
|Plymouth Satellite      |18.0|8      |318.0      |150.0      |3436.0|11.0      |70      |US      |1      |2      |2      |
|AMC Rebel SST          |16.0|8      |304.0      |150.0      |3433.0|12.0      |70      |US      |1      |2      |2      |
|Ford Torino            |17.0|8      |302.0      |140.0      |3449.0|10.5      |70      |US      |1      |2      |2      |
|Ford Galaxie 500      |15.0|8      |429.0      |198.0      |4341.0|10.0      |70      |US      |1      |2      |2      |
|Chevrolet Impala      |14.0|8      |454.0      |220.0      |4354.0|9.0      |70      |US      |1      |2      |2      |
|Plymouth Fury iii      |14.0|8      |440.0      |215.0      |4312.0|8.5      |70      |US      |1      |2      |2      |
|Pontiac Catalina      |14.0|8      |455.0      |225.0      |4425.0|10.0      |70      |US      |1      |2      |2      |
|AMC Ambassador DPL      |15.0|8      |390.0      |190.0      |3850.0|8.5      |70      |US      |1      |2      |2      |
|Citroen DS-21 Pallas      |0.0|4      |133.0      |115.0      |3090.0|17.5      |70      |Europe|1      |2      |2      |
|Chevrolet Chevelle Concours (sw)|0.0|8      |350.0      |165.0      |4142.0|11.5      |70      |US      |1      |2      |2      |
|Ford Torino (sw)      |0.0|8      |351.0      |153.0      |4034.0|11.0      |70      |US      |1      |2      |2      |
|Plymouth Satellite (sw)|0.0|8      |383.0      |175.0      |4166.0|10.5      |70      |US      |1      |2      |2      |
|AMC Rebel SST (sw)      |0.0|8      |360.0      |175.0      |3850.0|11.0      |70      |US      |1      |2      |2      |
|Dodge Challenger SE      |15.0|8      |383.0      |170.0      |3563.0|10.0      |70      |US      |1      |2      |2      |
|Plymouth 'Cuda 340      |14.0|8      |340.0      |160.0      |3609.0|8.0      |70      |US      |1      |2      |2      |
|Ford Mustang Boss 302      |0.0|8      |302.0      |140.0      |3353.0|8.0      |70      |US      |1      |2      |2      |
|Chevrolet Monte Carlo      |15.0|8      |400.0      |150.0      |3761.0|9.5      |70      |US      |1      |2      |2      |
|Buick Estate Wagon (sw)|14.0|8      |455.0      |225.0      |3086.0|10.0      |70      |US      |1      |2      |2      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```


only showing top 20 rows

Grouping By Columns

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

1. Group By a single column
2. 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|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|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 columns in one go
df = df.drop('new_column_two') \
        .drop('new_column_three')
df.show(5,truncate=False)
```

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

Step 2.3: DataFrame Operations on Rows

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
EUROPE FILTERED RECORD COUNT: 73
```

Car	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model	Origin	car_model
Citroen DS-21 Pallas	0.0	4	133.0	115.0	3090.0	17.5	70	Europe	Citroen DS-21 Pallas 7
Volkswagen 1131 Deluxe Sedan	26.0	4	97.0	46.0	1835.0	20.5	70	Europe	Volkswagen 1131 Deluxe
Peugeot 504	25.0	4	110.0	87.0	2672.0	17.5	70	Europe	Peugeot 504 70
Audi 100 LS	24.0	4	107.0	90.0	2430.0	14.5	70	Europe	Audi 100 LS 70
Saab 99e	25.0	4	104.0	95.0	2375.0	17.5	70	Europe	Saab 99e 70
BMW 2002	26.0	4	121.0	113.0	2234.0	12.5	70	Europe	BMW 2002 70
Volkswagen Super Beetle 117	0.0	4	97.0	48.0	1978.0	20.0	71	Europe	Volkswagen Super Beetl
Opel 1900	28.0	4	116.0	90.0	2123.0	14.0	71	Europe	Opel 1900 71
Peugeot 304	30.0	4	79.0	70.0	2074.0	19.5	71	Europe	Peugeot 304 71
Fiat 124B	30.0	4	88.0	76.0	2065.0	14.5	71	Europe	Fiat 124B 71
Volkswagen Model 111	27.0	4	97.0	60.0	1834.0	19.0	71	Europe	Volkswagen Model 111 ;
Volkswagen Type 3	23.0	4	97.0	54.0	2254.0	23.5	72	Europe	Volkswagen Type 3 72
Volvo 145e (sw)	18.0	4	121.0	112.0	2933.0	14.5	72	Europe	Volvo 145e (sw) 72
Volkswagen 411 (sw)	22.0	4	121.0	76.0	2511.0	18.0	72	Europe	Volkswagen 411 (sw) 72
Peugeot 504 (sw)	21.0	4	120.0	87.0	2979.0	19.5	72	Europe	Peugeot 504 (sw) 72
Renault 12 (sw)	26.0	4	96.0	69.0	2189.0	18.0	72	Europe	Renault 12 (sw) 72
Volkswagen Super Beetle	26.0	4	97.0	46.0	1950.0	21.0	73	Europe	Volkswagen Super Beetl
Fiat 124 Sport Coupe	26.0	4	98.0	90.0	2265.0	15.5	73	Europe	Fiat 124 Sport Coupe ;
Fiat 128	29.0	4	68.0	49.0	1867.0	19.5	73	Europe	Fiat 128 73
Opel Manta	24.0	4	116.0	75.0	2158.0	15.5	73	Europe	Opel Manta 73

only showing top 20 rows

```
# Filtering rows in PySpark based on Multiple conditions
total_count = df.count()
print("TOTAL RECORD COUNT: " + str(total_count))
europe_filtered_count = df.filter((col('Origin')=='Europe') &
                                   (col('Cylinders')==4)).count() # Two conditions added here
print("EUROPE FILTERED RECORD COUNT: " + str(europe_filtered_count))
df.filter(col('Origin')=='Europe').show(truncate=False)
```

```
↗ TOTAL RECORD COUNT: 406
EUROPE FILTERED RECORD COUNT: 66
```

Car	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model	Origin	car_model
Citroen DS-21 Pallas	0.0	4	133.0	115.0	3090.0	17.5	70	Europe	Citroen DS-21 Pallas ;
Volkswagen 1131 Deluxe Sedan	26.0	4	97.0	46.0	1835.0	20.5	70	Europe	Volkswagen 1131 Deluxe
Peugeot 504	25.0	4	110.0	87.0	2672.0	17.5	70	Europe	Peugeot 504 70
Audi 100 LS	24.0	4	107.0	90.0	2430.0	14.5	70	Europe	Audi 100 LS 70
Saab 99e	25.0	4	104.0	95.0	2375.0	17.5	70	Europe	Saab 99e 70
BMW 2002	26.0	4	121.0	113.0	2234.0	12.5	70	Europe	BMW 2002 70
Volkswagen Super Beetle 117	0.0	4	97.0	48.0	1978.0	20.0	71	Europe	Volkswagen Super Beetl
Opel 1900	28.0	4	116.0	90.0	2123.0	14.0	71	Europe	Opel 1900 71
Peugeot 304	30.0	4	79.0	70.0	2074.0	19.5	71	Europe	Peugeot 304 71
Fiat 124B	30.0	4	88.0	76.0	2065.0	14.5	71	Europe	Fiat 124B 71
Volkswagen Model 111	27.0	4	97.0	60.0	1834.0	19.0	71	Europe	Volkswagen Model 111 ;
Volkswagen Type 3	23.0	4	97.0	54.0	2254.0	23.5	72	Europe	Volkswagen Type 3 72
Volvo 145e (sw)	18.0	4	121.0	112.0	2933.0	14.5	72	Europe	Volvo 145e (sw) 72
Volkswagen 411 (sw)	22.0	4	121.0	76.0	2511.0	18.0	72	Europe	Volkswagen 411 (sw) 72
Peugeot 504 (sw)	21.0	4	120.0	87.0	2979.0	19.5	72	Europe	Peugeot 504 (sw) 72
Renault 12 (sw)	26.0	4	96.0	69.0	2189.0	18.0	72	Europe	Renault 12 (sw) 72
Volkswagen Super Beetle	26.0	4	97.0	46.0	1950.0	21.0	73	Europe	Volkswagen Super Beetl
Fiat 124 Sport Coupe	26.0	4	98.0	90.0	2265.0	15.5	73	Europe	Fiat 124 Sport Coupe ;
Fiat 128	29.0	4	68.0	49.0	1867.0	19.5	73	Europe	Fiat 128 73
Opel Manta	24.0	4	116.0	75.0	2158.0	15.5	73	Europe	Opel Manta 73

only showing top 20 rows

Get Distinct Rows

```
#Get Unique Rows in PySpark
df.select('Origin').distinct().show()
```

```

+-----+
|Origin|
+-----+
|Europe|
|   US  |
|  Japan|
+-----+

```

```
#Get Unique Rows in PySpark based on mutiple columns
df.select('Origin','model').distinct().show()
```

```

+-----+-----+
|Origin|model|
+-----+-----+
|Europe| 71|
|Europe| 80|
|Europe| 79|
|  Japan| 75|
|   US  | 72|
|   US  | 80|
|Europe| 74|
|  Japan| 79|
|Europe| 76|
|   US  | 75|
|  Japan| 77|
|   US  | 82|
|  Japan| 80|
|  Japan| 78|
|   US  | 78|
|Europe| 75|
|   US  | 71|
|   US  | 77|
|  Japan| 70|
|  Japan| 71|
+-----+-----+

```

only showing top 20 rows

Sorting Rows

```
# Sort Rows in PySpark
# By default the data will be sorted in ascending order
df.orderBy('Cylinders').show(truncate=False)
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Car|MPG|Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|car_model|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Mazda RX-4|21.5|3|80.0|110.0|2720.0|13.5|77|Japan|Mazda RX-4 77|
|Mazda RX-7 GS|23.7|3|70.0|100.0|2420.0|12.5|80|Japan|Mazda RX-7 GS 80|
|Mazda RX2 Coupe|19.0|3|70.0|97.0|2330.0|13.5|72|Japan|Mazda RX2 Coupe 72|
|Mazda RX3|18.0|3|70.0|90.0|2124.0|13.5|73|Japan|Mazda RX3 73|
|Datsun 510 (sw)|28.0|4|97.0|92.0|2288.0|17.0|72|Japan|Datsun 510 (sw) 72|
|Opel 1900|28.0|4|116.0|90.0|2123.0|14.0|71|Europe|Opel 1900 71|
|Mercury Capri 2000|23.0|4|122.0|86.0|2220.0|14.0|71|US|Mercury Capri 2000 71|
|Volkswagen 1131 Deluxe Sedan|26.0|4|97.0|46.0|1835.0|20.5|70|Europe|Volkswagen 1131 Deluxe|
|Peugeot 304|30.0|4|79.0|70.0|2074.0|19.5|71|Europe|Peugeot 304 71|
|Fiat 124B|30.0|4|88.0|76.0|2065.0|14.5|71|Europe|Fiat 124B 71|
|Chevrolet Vega (sw)|22.0|4|140.0|72.0|2408.0|19.0|71|US|Chevrolet Vega (sw) 71|
|Datsun 1200|35.0|4|72.0|69.0|1613.0|18.0|71|Japan|Datsun 1200 71|
|Volkswagen Model 111|27.0|4|97.0|60.0|1834.0|19.0|71|Europe|Volkswagen Model 111 71|
|Volkswagen Type 3|23.0|4|97.0|54.0|2254.0|23.5|72|Europe|Volkswagen Type 3 72|
|Audi 100 LS|24.0|4|107.0|90.0|2430.0|14.5|70|Europe|Audi 100 LS 70|
|BMW 2002|26.0|4|121.0|113.0|2234.0|12.5|70|Europe|BMW 2002 70|
|Toyota Corolla 1200|31.0|4|71.0|65.0|1773.0|19.0|71|Japan|Toyota Corolla 1200 71|
|Chevrolet Vega 2300|28.0|4|140.0|90.0|2264.0|15.5|71|US|Chevrolet Vega 2300 71|
|Ford Pinto|25.0|4|98.0|0.0|2046.0|19.0|71|US|Ford Pinto 71|
|Dodge Colt Hardtop|25.0|4|97.5|80.0|2126.0|17.0|72|US|Dodge Colt Hardtop 72|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

only showing top 20 rows

```
# To change the sorting order, you can use the ascending parameter
df.orderBy('Cylinders', ascending=False).show(truncate=False)
```

```

+-----+-----+
|Car|MPG|Cylinders|Displacement|Horsepower|Weight|Acceleration|Model|Origin|car_model|
+-----+-----+
|Plymouth 'Cuda 340|14.0|8|340.0|160.0|3609.0|8.0|70|US|Plymouth 'Cuda 340 70|
|Pontiac Safari (sw)|13.0|8|400.0|175.0|5140.0|12.0|71|US|Pontiac Safari (sw) 71|
|Ford Mustang Boss 302|0.0|8|302.0|140.0|3353.0|8.0|70|US|Ford Mustang Boss 302 70|
|Buick Skylark 320|15.0|8|350.0|165.0|3693.0|11.5|70|US|Buick Skylark 320 70|
|Chevrolet Monte Carlo|15.0|8|400.0|150.0|3761.0|9.5|70|US|Chevrolet Monte Carlo 70|
|AMC Rebel SST|16.0|8|304.0|150.0|3433.0|12.0|70|US|AMC Rebel SST 70|
|Buick Estate Wagon (sw)|14.0|8|455.0|225.0|3086.0|10.0|70|US|Buick Estate Wagon (sw) 70|
|Ford Galaxie 500|15.0|8|429.0|198.0|4341.0|10.0|70|US|Ford Galaxie 500 70|
|Ford F250|10.0|8|360.0|215.0|4615.0|14.0|70|US|Ford F250 70|
|Plymouth Fury iii|14.0|8|440.0|215.0|4312.0|8.5|70|US|Plymouth Fury iii 70|
|Chevy C20|10.0|8|307.0|200.0|4376.0|15.0|70|US|Chevy C20 70|
|AMC Ambassador DPL|15.0|8|390.0|190.0|3850.0|8.5|70|US|AMC Ambassador DPL 70|
|Dodge D200|11.0|8|318.0|210.0|4382.0|13.5|70|US|Dodge D200 70|
|Ford Torino (sw)|0.0|8|351.0|153.0|4034.0|11.0|70|US|Ford Torino (sw) 70|
|Hi 1200D|9.0|8|304.0|193.0|4732.0|18.5|70|US|Hi 1200D 70|
|AMC Rebel SST (sw)|0.0|8|360.0|175.0|3850.0|11.0|70|US|AMC Rebel SST (sw) 70|
|Chevrolet Impala|14.0|8|350.0|165.0|4209.0|12.0|71|US|Chevrolet Impala 71|
|Chevrolet Chevelle Malibu|18.0|8|307.0|130.0|3504.0|12.0|70|US|Chevrolet Chevelle Malibu 70|
|Pontiac Catalina Brougham|14.0|8|400.0|175.0|4464.0|11.5|71|US|Pontiac Catalina Brougham 71|
|Ford Torino|17.0|8|302.0|140.0|3449.0|10.5|70|US|Ford Torino 70|
+-----+-----+
only showing top 20 rows

```

```
# Using groupBy and orderBy together
df.groupBy("Origin").count().orderBy('count', ascending=False).show(10)
```

```

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

```

Union Dataframes You will see three main methods for performing union of dataframes. It is important to know the difference between them and which one is preferred:

1. `union()` – It is used to merge two DataFrames of the same structure/schema. If schemas are not the same, it returns an error
 2. `unionAll()` – This function is deprecated since Spark 2.0.0, and replaced with `union()`
 3. `unionByName()` - This function is used to merge two dataframes based on column name.
- Since `unionAll()` is deprecated, `union()` is the preferred method for merging dataframes. The difference between `unionByName()` and `union()` is that `unionByName()` resolves columns by name, not by position.
 - In other SQLs, Union eliminates the duplicates but `UnionAll` merges two datasets, thereby including duplicate records. But, in PySpark, both behave the same and includes duplicate records. The recommendation is to use `distinct()` or `dropDuplicates()` to remove duplicate records.

```
# CASE 1: Union When columns are in order
df = spark.read.csv('cars.csv', header=True, sep=";", inferSchema=True)
europe_cars = df.filter((col('Origin')=='Europe') & (col('Cylinders')==5))
japan_cars = df.filter((col('Origin')=='Japan') & (col('Cylinders')==3))
print("EUROPE CARS: "+str(europe_cars.count()))
print("JAPAN CARS: "+str(japan_cars.count()))
print("AFTER UNION: "+str(europe_cars.union(japan_cars).count()))
```

```

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

```

```
# CASE 1: Union When columns are not in order
# Creating two dataframes with jumbled columns
df1 = spark.createDataFrame([[1, 2, 3]], ["col0", "col1", "col2"])
df2 = spark.createDataFrame([[4, 5, 6]], ["col1", "col2", "col0"])
df1.unionByName(df2).show()
```

```

+-----+-----+
|col0|col1|col2|
+-----+-----+

```

```
+---+---+---+
| 1 | 2 | 3 |
| 6 | 4 | 5 |
+---+---+---+
```

✓ Step 2.4: Common Data Manipulation Functions

```
# Functions available in PySpark
from pyspark.sql import functions
# Similar to python, we can use the dir function to view the available functions
print(dir(functions))
```

```
['Column', 'DataFrame', 'DataType', 'PandasUDFType', 'PythonEvalType', 'SparkContext', 'StringType', 'UserDefinedFunction',
```

String Functions

```
# Loading the data
from pyspark.sql.functions import col
df = spark.read.csv('cars.csv', header=True, sep=";", inferSchema=True)

# Display the Car column in existing, lower and upper characters, and the first 4 characters of the column

from pyspark.sql.functions import col, lower, upper, substring
# Prints out the details of a function
help(substring)
# alias is used to rename the column name in the output
df.select(col('Car'), lower(col('Car')), upper(col('Car')), substring(col('Car'), 1, 4).alias("concatenated value")).show
```

```
Help on function substring in module pyspark.sql.functions:
```

```
substring(str, pos, len)
Substring starts at `pos` and is of length `len` when str is String type or
returns the slice of byte array that starts at `pos` in byte and is of length
when str is Binary type.
```

```
.. versionadded:: 1.5.0
```

Notes

```
-----
The position is not zero based, but 1 based index.
```

Examples

```
-----
>>> df = spark.createDataFrame([('abcd',)], ['s',])
>>> df.select(substring(df.s, 1, 2).alias('s')).collect()
[Row(s='ab')]
```

pyspark.sql.dataframe.DataFrame.show

```
def show(n=20, truncate=True, vertical=False)
```

```
Number of rows to show.
truncate : bool, optional
    If set to ``True``, truncate strings longer than 20 chars by default.
    If set to a number greater than one, truncates long strings to length ``tr``
    and align cells right.
vertical : bool, optional
    If set to ``True`` print output rows vertically (one line
```

```
#Concatenate the Car column and Model column and add a space between them.
```

```
from pyspark.sql.functions import concat
df.select(col("Car"), col("model"), concat(col("Car"), lit(" "), col("model"))).show(5, False)
```

```
only showing top 5 rows
```

Car	model	concat(Car, , model)
Chevrolet Chevelle Malibu	70	Chevrolet Chevelle Malibu 70
Buick Skylark 320	70	Buick Skylark 320 70
Plymouth Satellite	70	Plymouth Satellite 70
AMC Rebel SST	70	AMC Rebel SST 70
Ford Torino	70	Ford Torino 70

Numeric functions

```
# Show the oldest date and the most recent date
```

```
from pyspark.sql.functions import min, max
df.select(min(col('Weight')), max(col('Weight'))).show()
```

```

+-----+-----+
|min(Weight)|max(Weight)|
+-----+-----+
|         1613|         5140|
+-----+-----+

```

```
# Add 10 to the minimum and maximum weight
```

```
from pyspark.sql.functions import min, max, lit
df.select(min(col('Weight')+lit(10)), max(col('Weight')+lit(10))).show()
```

```

+-----+-----+
|(min(Weight) + 10)|max((Weight + 10))|
+-----+-----+
|             1623|             5150|
+-----+-----+

```

Operations on Date

```
from pyspark.sql.functions import to_date, to_timestamp, lit
df = spark.createDataFrame([('2019-12-25 13:30:00',)], ['DOB'])
df.show()
df.printSchema()
```

```

+-----+
|          DOB|
+-----+
|2019-12-25 13:30:00|
+-----+

```

```

root
 |-- DOB: string (nullable = true)

```

```
df = spark.createDataFrame([('2019-12-25 13:30:00',)], ['DOB'])
df = df.select(to_date(col('DOB'),'yyyy-MM-dd HH:mm:ss'), to_timestamp(col('DOB'),'yyyy-MM-dd HH:mm:ss'))
df.show()
df.printSchema()
```

```

+-----+-----+
|to_date(DOB, yyyy-MM-dd HH:mm:ss)|to_timestamp(DOB, yyyy-MM-dd HH:mm:ss)|
+-----+-----+
|                2019-12-25|                2019-12-25 13:30:00|
+-----+-----+

```

```

root
 |-- to_date(DOB, yyyy-MM-dd HH:mm:ss): date (nullable = true)
 |-- to_timestamp(DOB, yyyy-MM-dd HH:mm:ss): timestamp (nullable = true)

```

```
df = spark.createDataFrame([('25/Dec/2019 13:30:00',)], ['DOB'])
df = df.select(to_date(col('DOB'),'dd/MM/yyyy HH:mm:ss'), to_timestamp(col('DOB'),'dd/MM/yyyy HH:mm:ss'))
df.show()
df.printSchema()
```

```

+-----+-----+
|to_date(DOB, dd/MM/yyyy HH:mm:ss)|to_timestamp(DOB, dd/MM/yyyy HH:mm:ss)|
+-----+-----+
|                2019-12-25|                2019-12-25 13:30:00|
+-----+-----+

```

```

root
 |-- to_date(DOB, dd/MM/yyyy HH:mm:ss): date (nullable = true)
 |-- to_timestamp(DOB, dd/MM/yyyy HH:mm:ss): timestamp (nullable = true)

```

```
# What is 3 days earlier than the oldest date and 3 days later than the most recent date?
```

```
from pyspark.sql.functions import date_add, date_sub
# create a dummy dataframe
df = spark.createDataFrame([('1990-01-01',), ('1995-01-03',), ('2021-03-30',)], ['Date'])
# find out the required dates
df.select(date_add(max(col('Date')), 3), date_sub(min(col('Date')), 3)).show()
```

```
↵ |-----+-----+
  |date_add(max(Date), 3)|date_sub(min(Date), 3)|
  |-----+-----+
  |          2021-04-02|          1989-12-29|
  |-----+-----+
```