CSE481 - Colab 1

Spark Tutorial

In this tutorial you will learn how to use Apache Spark in local mode on a Colab enviroment.

Credits to Tiziano Piccardi for his Spark Tutorial used in the Applied Data Analysis class at EPFL. Adapted From Stanford's CS246

Setup

Let's setup Spark on your Colab environment. Run the cell below!

```
!pip install pyspark
!pip install -U -q PyDrive
!apt update
!apt install openjdk-8-jdk-headless -qq
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
```

```
Collecting pyspark
  Downloading https://files.pythonhosted.org/packages/f0/26/198fc8c0b98580f617cb03cb298c6056587b8f0447e20fa40c5b634ced
     |████████████████████████████████| 204.2MB 26kB/s
Collecting py4j==0.10.9
  Downloading https://files.pythonhosted.org/packages/9e/b6/6a4fb90cd235dc8e265a6a2067f2a2c99f0d91787f06aca4bcf7c23f3f
     |████████████████████████████████| 204kB 46.7MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.0.1-py2.py3-none-any.whl size=204612243 sha256=11dc840c033e07b4adeaf1f
  Stored in directory: /root/.cache/pip/wheels/5e/bd/07/031766ca628adec8435bb40f0bd83bb676ce65ff4007f8e73f
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.0.1
Get:1 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease [15.9 kB]
Hit:2 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Hit:4 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Get:5 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease [3,626 B]
Get:6 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:7 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Ign:8 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease
Get:9 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main Sources [1,685 kB]
Ign:10 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  InRelease
Get:11 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release [697 B]
Get:12 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Release [564 B]
Get:13 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release.gpg [836 B]
Get:14 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Release.gpg [833 B]
Get:15 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main amd64 Packages [863 kB]
Get:16 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ Packages [39.3 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2,165 kB]
Get:18 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [2,115 kB]
Get:19 http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [239 kB]
Get:20 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [45.9 kB]
Ign:21 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Packages
Get:21 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Packages [370 kB]
Get:22 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Packages [57.0 kB]
Get:23 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1,352 kB]
Get:24 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [15.4 kB]
Get:25 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [1,745 kB]
Get:26 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [211 kB]
Fetched 11.2 MB in 3s (4,118 kB/s)
Reading package lists... Done
Building dependency tree
```

```
Reading state information... Done
57 packages can be upgraded. Run 'apt list --upgradable' to see them.
The following additional packages will be installed:
  openjdk-8-jre-headless
Suggested packages:
  openjdk-8-demo openjdk-8-source libnss-mdns fonts-dejavu-extra
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei
  fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  openjdk-8-jdk-headless openjdk-8-jre-headless
0 upgraded, 2 newly installed, 0 to remove and 57 not upgraded.
Need to get 35.8 MB of archives.
After this operation, 140 MB of additional disk space will be used.
Selecting previously unselected package openjdk-8-jre-headless:amd64.
(Reading database ... 144611 files and directories currently installed.)
Preparing to unpack .../openjdk-8-jre-headless_8u265-b01-0ubuntu2~18.04_amd64.deb ...
Unpacking openjdk-8-jre-headless:amd64 (8u265-b01-0ubuntu2~18.04) ...
Selecting previously unselected package openjdk-8-jdk-headless:amd64.
Preparing to unpack .../openjdk-8-jdk-headless_8u265-b01-0ubuntu2~18.04_amd64.deb ...
Unpacking openjdk-8-jdk-headless:amd64 (8u265-b01-0ubuntu2~18.04) ...
Setting up openjdk-8-jre-headless:amd64 (8u265-b01-0ubuntu2~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/orbd to provide /usr/bin/orbd (orbd) in auto mode
```

Now we authenticate a Google Drive client to download the file we will be processing in our Spark job.

Make sure to follow the interactive instructions.

```
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jsadebugd to provide /usr/bin/jsadebugd (jsadebugd) i
```

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


id='1L6pCQkldvdBoaEhRFzL0VnrggEFvqON4'
downloaded = drive.CreateFile({'id': id})
```

```
downloaded.GetContentFile('Bombing_Operations.json.gz')

id='14dyBmcTBA32uXPxDbqr0bFDIzGxMTWwl'
downloaded = drive.CreateFile({'id': id})
downloaded.GetContentFile('Aircraft_Glossary.json.gz')
```

If you executed the cells above, you should be able to see the files *Bombing_Operations.json.gz* and *Aircraft_Glossary.json.gz* under the "Files" tab on the left panel.

```
# Let's import the libraries we will need
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import pyspark
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf
```

Let's initialize the Spark context.

```
# create the session
conf = SparkConf().set("spark.ui.port", "4050")

# create the context
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession.builder.getOrCreate()
```

You can easily check the current version and get the link of the web interface. In the Spark UI, you can monitor the progress of your job and debug the performance bottlenecks (if your Colab is running with a **local runtime**).

```
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
    v3.0.1
Master
    local[*]
AppName
    pyspark-shell

If you are running this Colab on the Google hosted runtime, the cell below will create a *ngrok* tunnel which will allow you to still check the Spark UI.

```
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip ngrok-stable-linux-amd64.zip
get_ipython().system_raw('./ngrok http 4050 &')
!curl http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url']);"
```

--2020-10-26 21:53:52-- https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip

Vietnam War

Pres. Johnson: *What do you think about this Vietnam thing? I'd like to hear you talk a little bit.*

Sen. Russell: *Well, frankly, Mr. President, it's the damn worse mess that I ever saw, and I don't like to brag and I never have been right many times in my life, but I knew that we were going to get into this sort of mess when we went in there.*

May 27, 1964

The Vietnam War, also known as the Second Indochina War, and in Vietnam as the Resistance War Against America or simply the American War, was a conflict that occurred in Vietnam, Laos, and Cambodia from 1 November 1955 to the fall of Saigon on 30 April 1975. It was the second of the Indochina Wars and was officially fought between North Vietnam and the government of South Vietnam.

The dataset describes all the air force operation in during the Vietnam War.

Bombing_Operations Get the dataset here

- AirCraft: *Aircraft model (example: EC-47)*
- ContryFlyingMission: *Country*
- MissionDate: *Date of the mission*
- OperationSupported: *Supported War operation* (example: Operation Rolling Thunder)
- PeriodOfDay: *Day or night*
- TakeoffLocation: *Take off airport*
- TimeOnTarget
- WeaponType
- WeaponsLoadedWeight

Aircraft_Glossary Get the dataset here

- AirCraft: *Aircraft model (example: EC-47)*
- AirCraftName
- AirCraftType

Dataset Information:

THOR is a painstakingly cultivated database of historic aerial bombings from World War I through Vietnam. THOR has already proven useful in finding unexploded ordnance in Southeast Asia and improving Air Force combat tactics: https://www.kaggle.com/usaf/vietnam-war-bombing-operations

Load the datasets:

```
Bombing_Operations = spark.read.json("Bombing_Operations.json.gz")
Aircraft_Glossary = spark.read.json("Aircraft_Glossary.json.gz")
```

Check the schema:

```
Bombing_Operations.printSchema()
```

```
root
 |-- AirCraft: string (nullable = true)
 |-- ContryFlyingMission: string (nullable = true)
 |-- MissionDate: string (nullable = true)
 |-- OperationSupported: string (nullable = true)
 |-- PeriodOfDay: string (nullable = true)
 |-- TakeoffLocation: string (nullable = true)
 |-- TargetCountry: string (nullable = true)
 |-- TimeOnTarget: double (nullable = true)
 |-- WeaponType: string (nullable = true)
```

```
Aircraft_Glossary.printSchema()
```

```
root
 |-- AirCraft: string (nullable = true)
 |-- AirCraftName: string (nullable = true)
 |-- AirCraftType: string (nullable = true)
```

Get a sample with `take()`:

```
Bombing_Operations.take(3)
```

```
[Row(AirCraft='EC-47', ContryFlyingMission='UNITED STATES OF AMERICA', MissionDate='1971-06-05', OperationSupported=No
 Row(AirCraft='EC-47', ContryFlyingMission='UNITED STATES OF AMERICA', MissionDate='1972-12-26', OperationSupported=No
 Row(AirCraft='RF-4', ContryFlyingMission='UNITED STATES OF AMERICA', MissionDate='1973-07-28', OperationSupported=Non
```

Get a formatted sample with `show()`:

```
Aircraft_Glossary.show()
```

```
+--------+--------------------+--------------------+
|AirCraft|        AirCraftName|        AirCraftType|
+--------+--------------------+--------------------+
|     A-1|Douglas A-1 Skyra...|         Fighter Jet|
|    A-26|Douglas A-26 Invader|        Light Bomber|
|    A-37|Cessna A-37 Drago...|Light ground-atta...|
|     A-4|McDonnell Douglas...|         Fighter Jet|
|     A-5|North American A-...|          Bomber Jet|
|     A-6|Grumman A-6 Intruder|     Attack Aircraft|
|     A-7|  LTV A-7 Corsair II|     Attack Aircraft|
|  AC-119|Fairchild AC-119 ...|Military Transpor...|
|  AC-123|Fairchild C-123 P...|Military Transpor...|
|  AC-130|Lockheed AC-130 S...|Fixed wing ground...|
|   AC-47|Douglas AC-47 Spooky|Ground attack air...|
|    AH-1| Bell AH-1 HueyCobra|          Helicopter|
|     B-1| Rockwell B-1 Lancer|Heavy strategic b...|
|    B-52| B-52 Stratofortress|    Strategic bomber|
|    B-57|Martin B-57 Canberra|     Tactical Bomber|
|    B-66|Douglas B-66 Dest...|        Light Bomber|
|     C-1| Grumman C-1A Trader|           Transport|
```

```
print(f"In total there are {Bombing_Operations.count():,d} operations")
```

```
In total there are 4,400,775 operations

only showing top 20 rows
```

Question 1: Which countries are involved and in how many missions?

Keywords: `Dataframe API`, `SQL`, `group by`, `sort`

Let's group the missions by `ContryFlyingMission` and count how many records exist:

```
missions_counts = Bombing_Operations.groupBy("ContryFlyingMission")\
                                    .agg(count("*").alias("MissionsCount"))\
                                    .sort(desc("MissionsCount"))
missions_counts.show()
```

```
+--------------------+-------------+
| ContryFlyingMission|MissionsCount|
+--------------------+-------------+
|UNITED STATES OF ...|      3708997|
|     VIETNAM (SOUTH)|       622013|
|               LAOS|        32777|
|       KOREA (SOUTH)|        24469|
|           AUSTRALIA|        12519|
```

In this case we used the DataFrame API, but we could rewite the `groupBy` using pure SQL:

```
Bombing_Operations.registerTempTable("Bombing_Operations")

query = """
SELECT ContryFlyingMission, count(*) as MissionsCount
FROM Bombing_Operations
GROUP BY ContryFlyingMission
ORDER BY MissionsCount DESC
"""

missions_counts = spark.sql(query)
missions_counts.show()
```

```
+--------------------+-------------+
| ContryFlyingMission|MissionsCount|
+--------------------+-------------+
|UNITED STATES OF ...|      3708997|
|     VIETNAM (SOUTH)|       622013|
|               LAOS|        32777|
|       KOREA (SOUTH)|        24469|
|           AUSTRALIA|        12519|
+--------------------+-------------+
```

The Dataframe is small enough to be moved to Pandas:

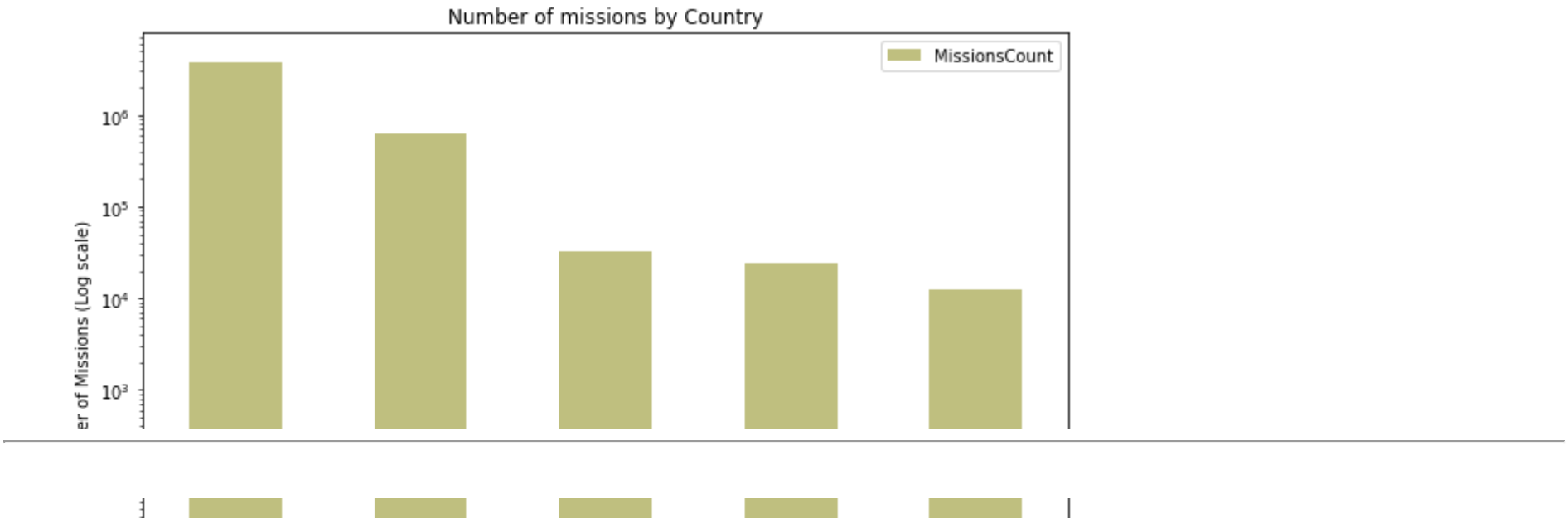
```
missions_count_pd = missions_counts.toPandas()
missions_count_pd.head()
```

	ContryFlyingMission	MissionsCount
0	UNITED STATES OF AMERICA	3708997
1	VIETNAM (SOUTH)	622013
2	LAOS	32777
3	KOREA (SOUTH)	24469
4	AUSTRALIA	12519

Let's plot a barchart with the number of missions by country:

```
pl = missions_count_pd.plot(kind="bar",
                            x="ContryFlyingMission", y="MissionsCount",
                            figsize=(10, 7), log=True, alpha=0.5, color="olive")
pl.set_xlabel("Country")
pl.set_ylabel("Number of Missions (Log scale)")
pl.set_title("Number of missions by Country")
```

```
Text(0.5, 1.0, 'Number of missions by Country')
```



Questions 2: Show the number of missions in time for each of the countries involved.

Keywords: `group by`, `parse date`, `plot`

Let's select the relevant columns:

```
missions_countries = Bombing_Operations.selectExpr(["to_date(MissionDate) as MissionDate", "ContryFlyingMission"])
missions_countries.printSchema()
```

```
root
 |-- MissionDate: date (nullable = true)
 |-- ContryFlyingMission: string (nullable = true)
```

The filed MissionDate is converted to a Spark `data` object.

Now we can group by `MissionDate` and `ContryFlyingMission` to get the count:

```
missions_by_date = missions_countries\
                   .groupBy(["MissionDate", "ContryFlyingMission"])\
                   .agg(count("*").alias("MissionsCount"))\
                   .sort(asc("MissionDate")).toPandas()
missions_by_date.head()
```

	MissionDate	ContryFlyingMission	MissionsCount
0	1965-10-01	UNITED STATES OF AMERICA	447
1	1965-10-02	UNITED STATES OF AMERICA	652
2	1965-10-03	UNITED STATES OF AMERICA	608
3	1965-10-04	UNITED STATES OF AMERICA	532
4	1965-10-05	UNITED STATES OF AMERICA	697

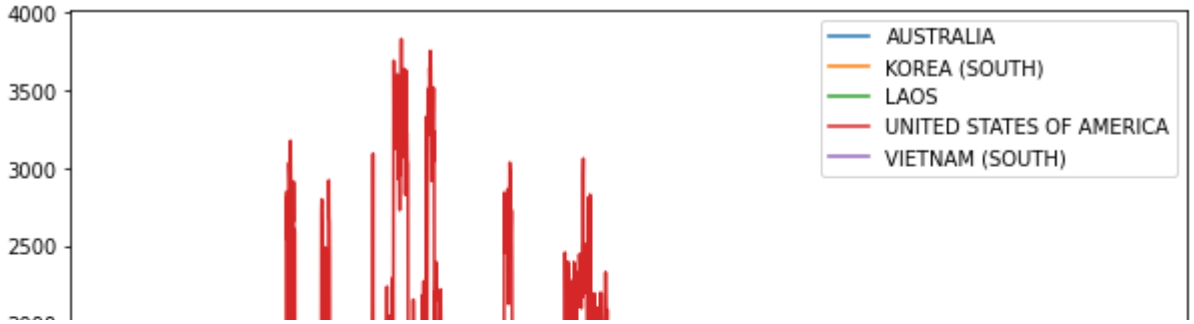
Now we can plot the content with a different series for each country:

```
fig = plt.figure(figsize=(10, 6))

# iterate the different groups to create a different series
for country, missions in missions_by_date.groupby("ContryFlyingMission"):
    plt.plot(missions["MissionDate"], missions["MissionsCount"], label=country)

plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7f25e5d33b00>
```



We can observe how South Vietnam increased its missions starting from 1970. The drop in 1973 is motivated by the [Paris Peace Accords] that took place on January 27th, 1973, to establish peace in Vietnam and end the war.

Question 3: Who bombed this location?

Keywords: `RDD map reduce cache save results`

This picture is the Hanoi POL facility (North Vietnam) burning after it was attacked by the U.S. Air Force on 29 June 1966 in the context of the Rolling Thunder operation.

We are interested in discovering what was the most common take-off location during that day.

```
jun_29_operations = Bombing_Operations.where("MissionDate = '1966-06-29' AND TargetCountry='NORTH VIETNAM'")
```

Which coutries scheduled missions that day?

```
jun_29_operations.groupBy("ContryFlyingMission").agg(count("*").alias("MissionsCount")).toPandas()
```

	ContryFlyingMission	MissionsCount
0	VIETNAM (SOUTH)	6
1	UNITED STATES OF AMERICA	389

Most of the operation that day were performed by USA airplanes.

```
jun_29_operations.take(1)
```

```
[Row(AirCraft='F-105', ContryFlyingMission='UNITED STATES OF AMERICA', MissionDate='1966-06-29', OperationSupported='S
```

You can specify to cache the content in memory:

```
jun_29_operations.cache()
```

```
DataFrame[AirCraft: string, ContryFlyingMission: string, MissionDate: string, OperationSupported: string, PeriodOfDay:
```

Now you can count the number of rows and move the content to the cache:

```
%time jun_29_operations.count()
```

```
CPU times: user 2.81 ms, sys: 1.32 ms, total: 4.13 ms
Wall time: 17.7 s
395
```

The second time the content is cached and the operation is much faster:

```
%time jun_29_operations.count()
```

```
CPU times: user 667 µs, sys: 203 µs, total: 870 µs
Wall time: 80.7 ms
395
```

You can also save the results on a file...

```
jun_29_operations.write.mode('overwrite').json("jun_29_operations.json")
```

... and read from the file:

```
jun_29_operations = spark.read.json("jun_29_operations.json")
```

We can use the simple DataFrame API...

```
TakeoffLocationCounts = jun_29_operations\
                            .groupBy("TakeoffLocation").agg(count("*").alias("MissionsCount"))\
                            .sort(desc("MissionsCount"))
TakeoffLocationCounts.show()
```

```
+----------------+-------------+
| TakeoffLocation|MissionsCount|
+----------------+-------------+
|   CONSTELLATION|           87|
```

... or the explicit Map/Reduce format with RDDs.

First we emit a pair in the format (Location, 1):

```
|          DANANG|           55|
|          RANGER|           25|
```

```
all_locations = jun_29_operations.rdd.map(lambda row: (row.TakeoffLocation, 1))
all_locations.take(3)
```

```
[('TAKHLI', 1), ('DANANG', 1), ('CONSTELLATION', 1)]
```

Then, we sum counters in the reduce step, and we sort by count:

```
locations_counts_rdd = all_locations.reduceByKey(lambda a, b: a+b).sortBy(lambda r: -r[1])
locations_counts_rdd.take(3)
```

```
[('CONSTELLATION', 87), ('TAKHLI', 56), ('KORAT', 55)]
```

Now we can convert the RDD in dataframe by mapping the pairs to objects of type `Row`

```
locations_counts_with_schema = locations_counts_rdd.map(lambda r: Row(TakeoffLocation=r[0], MissionsCount=r[1]))
locations_counts = spark.createDataFrame(locations_counts_with_schema)
locations_counts.show()
```

```
+----------------+-------------+
| TakeoffLocation|MissionsCount|
+----------------+-------------+
|   CONSTELLATION|           87|
|          TAKHLI|           56|
|           KORAT|           55|
|         UBON AB|           44|
```

That day the most common take-off location was the ship USS Constellation (CV-64). We cannot univocally identify one take off location, but we can reduce the possible candidates. Next steps: explore TimeOnTarget feature.

USS Constellation (CV-64), a Kitty Hawk-class supercarrier, was the third ship of the United States Navy to be named in honor of the "new constellation of stars" on the flag of the United States. One of the fastest ships in the Navy, as proven by her victory during a battlegroup race held in 1985, she was nicknamed "Connie" by her crew and officially as "America's Flagship".

Questions 4: What is the most used aircraft type during the Vietnam war (number of missions)?

Keywords: `join group by`

Let's check the content of `Aircraft_Glossary`:

```
Aircraft_Glossary.show(5)
```

```
+--------+--------------------+--------------------+
|AirCraft|        AirCraftName|        AirCraftType|
+--------+--------------------+--------------------+
|     A-1|Douglas A-1 Skyra...|         Fighter Jet|
|    A-26|Douglas A-26 Invader|        Light Bomber|
|    A-37|Cessna A-37 Drago...|Light ground-atta...|
|     A-4|McDonnell Douglas...|         Fighter Jet|
|     A-5|North American A-...|          Bomber Jet|
+--------+--------------------+--------------------+
only showing top 5 rows
```

We are interested in the filed `AirCraftType` .

```
Bombing_Operations.select("AirCraft").show(5)
```

```
+--------+
|AirCraft|
+--------+
|   EC-47|
|   EC-47|
|    RF-4|
|     A-1|
|    A-37|
+--------+
only showing top 5 rows
```

We can join on the column `AirCraft` of both dataframes.

With Dataframe API:

```
missions_joined = Bombing_Operations.join(Aircraft_Glossary,
                                          Bombing_Operations.AirCraft == Aircraft_Glossary.AirCraft)
missions_joined
```

```
DataFrame[AirCraft: string, ContryFlyingMission: string, MissionDate: string, OperationSupported: string, PeriodOfDay:
```

We can select only the field we are interested in:

```
missions_aircrafts = missions_joined.select("AirCraftType")
missions_aircrafts.show(5)
```

```
+--------------------+
|        AirCraftType|
+--------------------+
|Military Transpor...|
|Military Transpor...|
|  Fighter bomber jet|
|         Fighter Jet|
|Light ground-atta...|
+--------------------+
only showing top 5 rows
```

And finally we can group by `AirCraftType` and count:

```
missions_aircrafts.groupBy("AirCraftType").agg(count("*").alias("MissionsCount"))\
                  .sort(desc("MissionsCount"))\
                  .show()
```

```
+--------------------+-------------+
|        AirCraftType|MissionsCount|
+--------------------+-------------+
|  Fighter Jet Bomber|      1073126|
|         Fighter Jet|       882594|
|  Jet Fighter Bomber|       451385|
|     Attack Aircraft|       315246|
|Light ground-atta...|       267457|
|  Fighter bomber jet|       242231|
|Military Transpor...|       228426|
|  Utility Helicopter|       146653|
|    Strategic bomber|        99100|
|     Tactical Bomber|        82219|
|Observation Aircraft|        81820|
|Fixed wing ground...|        75058|
|Ground attack air...|        73843|
|       Light Fighter|        39999|
```

In alternative we can rewrite this in pure SQL:

```
Bombing_Operations.registerTempTable("Bombing_Operations")
Aircraft_Glossary.registerTempTable("Aircraft_Glossary")

query = """
SELECT AirCraftType, count(*) MissionsCount
FROM Bombing_Operations bo
JOIN Aircraft_Glossary ag
ON bo.AirCraft = ag.AirCraft
GROUP BY AirCraftType
ORDER BY MissionsCount DESC
"""

spark.sql(query).show()
```

```
+--------------------+-------------+
|        AirCraftType|MissionsCount|
+--------------------+-------------+
|  Fighter Jet Bomber|      1073126|
|         Fighter Jet|       882594|
|  Jet Fighter Bomber|       451385|
|     Attack Aircraft|       315246|
|Light ground-atta...|       267457|
|  Fighter bomber jet|       242231|
|Military Transpor...|       228426|
|  Utility Helicopter|       146653|
|    Strategic bomber|        99100|
|     Tactical Bomber|        82219|
|Observation Aircraft|        81820|
|Fixed wing ground...|        75058|
|Ground attack air...|        73843|
|Carrier-based Fig...|        58691|
|   Training Aircraft|        48435|
|       Light fighter|        39999|
```

The aircrafts of type `Fighter Jet Bomber` participated in most of the missions in the Vietnam war.

Note: This dataset would require further cleaning and normalization. See `Fighter Jet Bomber`, `Jet Fighter Bomber`, `Fighter bomber jet`