# Big Data: Basic Data Analysis using Spark

Azhar Chowdhury
Statistical Programmer|Data Scientist[SAS, Python, R]

The following meta summary is helpful to follow the problem at hand and it's solution.

**Dataset** A JSON dataset qualifies for big data operations

**Domain** Standard database - JSON

**Operations** Standard big data analysis

**Components** Python→Spark

**Problem Description** Import a dataset into the Spark environment, create tables for the data imported, perform basic data analysis using transformation and Spark SQL

[1]:
```
pip install pyspark
```

```
Requirement already satisfied: pyspark in c:\users\azhar\anaconda3\lib\site-
packages (3.5.5)
Requirement already satisfied: py4j==0.10.9.7 in
c:\users\azhar\anaconda3\lib\site-packages (from pyspark) (0.10.9.7)
Note: you may need to restart the kernel to use updated packages.
```

### Some Essential Spark definitions

**RDD** is an abbreviation stands for Resilient Distributed Dataset. Key features: It is immutable - that is once created can not be changed. Distributed – that is data is spread across multiple nodes. Fault-tolerant – that is if a partition is lost it can be recomputed using its lineage. Lazy Evaluation – that is transformations are not executed until an action is called. Typed Objects – that is they can hold objects of any type including custom ones. Limitations: No schema enforcement that is they are unstructured data. It is of low-level API, that makes operations verbose. Performance overhead due to Java Serialization.

**DataFrame** DataFrame is a distributed collection of data organized into named columns which is similar to a table in a relational database. DataFrame is built on top of RDDs and optimized for performance using Spark's Catalyst Optimizer. Key Features: It is Schema-based – that is it stores data in named column. It optimizes execution – that is it uses Catalyst Optimizer for query optimization. It supports Interoperability – that is it works well with SQL, JSON, Parquet, and Hive. It has High-Level API – that means it is easier to use than RDDs, it supports SQL-like queries. Limitations: It has Less Type Safety – that is it has runtime errors instead compile-time errors. It has limited compile-time optimization compared to datasets.

**Dataset** A dataset is a distributed collection of strongly-typed objects combining the benefits of RDDs and DataFrame. Where, RDDs contribute strong typing and functional transformations and DataFrames contribute optimized execution. Though it is not available in PySpark. Key Features: Type Safety – that is the type is checked during compile time. Optimized Execution – that is it uses Catalyst optimizer like the DataFrames. It hat Encapsulated Objects – that is it stores data in user-defined case classes. Limitations: It is not supported in Python yet. Compared to DataFrames it is slightly more verbose.

**When to use**

- RDD If we need fine-grained control over data and transformations, and when schema enforcement is not necessary.
- DataFrames: when working with structured data, where performance and ease of use are important.
- Datasets: when working in Scala or Java and need type safety along with optimization.

[2]:
```
# Creating a Spark object

from pyspark.sql import SparkSession
spark = (
    SparkSession.builder.config("spark.driver.host", "localhost").
 ↪appName("Sample").getOrCreate()
)
spark.sparkContext.setLogLevel("DEBUG")
```

[3]:
```
# read success from local file [Tusher 2025-Mar-10] - iotDF is the dataset␣
 ↪converted from JSON

path = "C://Users//azhar//Downloads//iot_devices.json"
iotDF=spark.read.json(path)
```

[4]:
```
# display the json data in terms of pd.DataFrame:

iotDF.show(truncate=False)
import pandas as pd

pd_df = iotDF.toPandas()
display(pd_df)
iotDF.printSchema()
```

```
+-------------+---------+----+----+-------------+---------+--------------------
---+--------+---------------+--------+------+---------+-------+----+-----------
-+
|battery_level|c02_level|cca2|cca3|cn           |device_id|device_name
|humidity|ip             |latitude|lcd   |longitude|scale  |temp|timestamp    |
+-------------+---------+----+----+-------------+---------+--------------------
---+--------+---------------+--------+------+---------+-------+----+-----------
```

```
-+
|8            |868      |US  |USA |United States|1        |meter-gauge-1xbYRYcj
|51      |68.161.225.1   |38.0    |green |-97.0    |Celsius|34  |1458444054093|
|7            |1473     |NO  |NOR |Norway       |2        |sensor-pad-2n2Pea
|70      |213.161.254.1  |62.47   |red   |6.15     |Celsius|11  |1458444054119|
|2            |1556     |IT  |ITA |Italy        |3        |device-mac-36TWSKiT
|44      |88.36.5.1      |42.83   |red   |12.83    |Celsius|19  |1458444054120|
|6            |1080     |US  |USA |United States|4        |sensor-pad-4mzWkz
|32      |66.39.173.154  |44.06   |yellow|-121.32  |Celsius|28  |1458444054121|
|4            |931      |PH  |PHL |Philippines  |5        |therm-stick-5gimpUrBB
|62      |203.82.41.9    |14.58   |green |120.97   |Celsius|25  |1458444054122|
|3            |1210     |US  |USA |United States|6        |sensor-pad-6al7RTAobR
|51      |204.116.105.67 |35.93   |yellow|-85.46   |Celsius|27  |1458444054122|
|3            |1129     |CN  |CHN |China        |7        |meter-gauge-7GeDoanM
|26      |220.173.179.1  |22.82   |yellow|108.32   |Celsius|18  |1458444054123|
|0            |1536     |JP  |JPN |Japan        |8        |sensor-pad-8xUD6pzsQI
|35      |210.173.177.1  |35.69   |red   |139.69   |Celsius|27  |1458444054123|
|3            |807      |JP  |JPN |Japan        |9        |device-mac-9GcjZ2pw
|85      |118.23.68.227  |35.69   |green |139.69   |Celsius|13  |1458444054124|
|7            |1470     |US  |USA |United States|10       |sensor-pad-10BsywSYUF
|56      |208.109.163.218|33.61   |red   |-111.89  |Celsius|26  |1458444054125|
|3            |1544     |IT  |ITA |Italy        |11       |meter-gauge-11dlMTZty
|85      |88.213.191.34  |42.83   |red   |12.83    |Celsius|16  |1458444054125|
|0            |1260     |US  |USA |United States|12       |sensor-pad-12Y2kIm0o
|92      |68.28.91.22    |38.0    |yellow|-97.0    |Celsius|12  |1458444054126|
|6            |1007     |IN  |IND |India        |13       |meter-
gauge-13GrojanSGBz|92      |59.144.114.250 |28.6    |yellow|77.2     |Celsius|13
|1458444054127|
|1            |1346     |NO  |NOR |Norway       |14       |sensor-
pad-14QL93sBR0j  |90      |193.156.90.200 |59.95   |yellow|10.75    |Celsius|16
|1458444054127|
|9            |1259     |US  |USA |United States|15       |device-mac-15se6mZ
|70      |67.185.72.1    |47.41   |yellow|-122.0   |Celsius|13  |1458444054128|
|4            |1425     |US  |USA |United States|16       |sensor-
pad-16aXmIJZtdO  |53      |68.85.85.106   |38.0    |red   |-97.0    |Celsius|15
|1458444054128|
|0            |1466     |US  |USA |United States|17       |meter-
gauge-17zb8Fghhl  |98      |161.188.212.254|39.95   |red   |-75.16   |Celsius|31
|1458444054129|
|4            |1096     |CN  |CHN |China        |18       |sensor-pad-18XULN9Xv
|25      |221.3.128.242  |25.04   |yellow|102.72   |Celsius|31  |1458444054130|
|9            |1531     |US  |USA |United States|19       |meter-
gauge-19eg1BpfCO  |75      |64.124.180.215 |38.0    |red   |-97.0    |Celsius|29
|1458444054130|
|7            |1155     |US  |USA |United States|20       |sensor-pad-20gFNfBgqr
|33      |66.153.162.66  |33.94   |yellow|-78.92   |Celsius|10  |1458444054131|
+-------------+---------+----+----+-------------+---------+--------------------
---+--------+---------------+--------+------+---------+-------+----+------------
```

```
-+
only showing top 20 rows


        battery_level  c02_level cca2 cca3             cn  device_id  \
0                   8        868   US  USA  United States          1
1                   7       1473   NO  NOR         Norway          2
2                   2       1556   IT  ITA          Italy          3
3                   6       1080   US  USA  United States          4
4                   4        931   PH  PHL    Philippines          5
...               ...        ...  ...  ...            ...        ...
198159              5       1594   IT  ITA          Italy     198160
198160              4       1051   CA  CAN         Canada     198161
198161              3       1455   NL  NLD    Netherlands     198162
198162              4       1358   DE  DEU        Germany     198163
198163              0       1291   PL  POL         Poland     198164

                        device_name  humidity              ip  latitude  \
0                meter-gauge-1xbYRYcj        51    68.161.225.1     38.00
1                   sensor-pad-2n2Pea        70   213.161.254.1     62.47
2                 device-mac-36TWSKiT        44       88.36.5.1     42.83
3                   sensor-pad-4mzWkz        32   66.39.173.154     44.06
4                therm-stick-5gimpUrBB        62     203.82.41.9     14.58
...                                ...       ...             ...       ...
198159   sensor-pad-198160wdsQqMjV8b        49   195.94.190.34     42.83
198160    meter-gauge-198161t6pF0Xhg        63   204.19.147.19     45.50
198161       sensor-pad-198162778iVI        51  212.204.219.81     52.37
198162  meter-gauge-198163jphYrUReMD        56   93.135.125.41     48.19
198163   sensor-pad-198164NdlyvsOr9C        98     80.55.20.25     53.08

           lcd  longitude    scale  temp      timestamp
0        green     -97.00  Celsius    34  1458444054093
1          red       6.15  Celsius    11  1458444054119
2          red      12.83  Celsius    19  1458444054120
3       yellow    -121.32  Celsius    28  1458444054121
4        green     120.97  Celsius    25  1458444054122
...        ...        ...      ...   ...            ...
198159     red      12.83  Celsius    24  1458444061098
198160  yellow     -73.55  Celsius    27  1458444061098
198161     red       4.89  Celsius    17  1458444061098
198162  yellow      11.38  Celsius    30  1458444061098
198163  yellow      18.62  Celsius    12  1458444061098

[198164 rows x 15 columns]
```

```
root
 |-- battery_level: long (nullable = true)
 |-- c02_level: long (nullable = true)
 |-- cca2: string (nullable = true)
```

```
 |-- cca3: string (nullable = true)
 |-- cn: string (nullable = true)
 |-- device_id: long (nullable = true)
 |-- device_name: string (nullable = true)
 |-- humidity: long (nullable = true)
 |-- ip: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- lcd: string (nullable = true)
 |-- longitude: double (nullable = true)
 |-- scale: string (nullable = true)
 |-- temp: long (nullable = true)
 |-- timestamp: long (nullable = true)
```

[5]:
```
## Query (i): How many sensor pads are reported to be from Poland?

x=iotDF.filter((iotDF.cn.isin('Poland')) & iotDF.device_name.
 ↪like("sensor-pad%")).count()
print("Number of Sensor Pads from Poland:", x)
```

```
Number of Sensor Pads from Poland: 1413
```

[6]:
```
## Query (ii): How many different LCDs (distinct colors) are present in the␣
 ↪dataset?

lcd_grp = iotDF.select("lcd").distinct().count()
print(f"Number of distinct colours in 'LCD' Column: {lcd_grp}")
```

```
Number of distinct colours in 'LCD' Column: 3
```

[7]:
```
## Query (iii): Find 5 countries that have the largest number of MAC devices␣
 ↪used

# Collect Mac Devices: achieved by filtering using where clause
macDevices= iotDF.select("cn" , "device_name")\
.where(iotDF.device_name.like("device-mac%"))
macDevices.show(3) # This shows only first 3 rows to confirm the device_name␣
 ↪values are Mac_Devices (device-mac)
```

```
+-------------+-------------------+
|           cn|        device_name|
+-------------+-------------------+
|        Italy|device-mac-36TWSKiT|
|        Japan|device-mac-9GcjZ2pw|
|United States| device-mac-15se6mZ|
+-------------+-------------------+
only showing top 3 rows
```

[8]:
```
## Query (iii)  continued...

# counting number of mac devices by country and displaying top five countries.
# we're achieving this by soring the number of mac devices in descending order␣
 ↪and \
# displaying only the first five  countries

from pyspark.sql import functions as saf  # sql aggrigate function for spark
cnMacCount = ( # counting mac devices by country using countDistinct() function
    macDevices.groupBy("cn")
    .agg(saf.countDistinct("device_name").alias("mac_device"))
    .orderBy(saf.col("mac_device").desc())
)
cnMacCount.show(5)
```

```
+-----------------+----------+
|               cn|mac_device|
+-----------------+----------+
|    United States|     11508|
|            China|      2300|
|            Japan|      2001|
|Republic of Korea|      1999|
|          Germany|      1314|
+-----------------+----------+
only showing top 5 rows
```

'## Query (iv) Hypothesis Testing:

Based on the column 'lcd' it has only 3 distinct values - green, yellow and red. Green is more darker and can block $CO_2$ (just assuming), while, yellow is lighter than green and can block less $CO_2$ than green lcds. Our data does not reveal much information about the features (columns), hence we'll assume certain things to carry out the research and computation.

Based on the given data we want to find out whether a country like Canada which is more environmentally cautious uses more green lcds comparing to a country which is less environmentally cautious like Canada, for example India.

Here, we'll compute the sameple mean, sample variance, sample count for green lcds. We'll apply a test of the differences of means to find out.

Our test of hypothesis is as follows:

$H_0$: $\mu_1 = \mu_2$, That is the difference is merely by chance (no difference).

$H_A$: $\mu_1 \neq \mu_2$, There is a significant difference between them.

From the data we'll compute sample means ($\bar{X}_1, \bar{X}_2$ - for green lcds Canada and India. sample variances $\sigma_1, \sigma_2$, and, Sample count $(n_1, n_2)$.

[9]:
```
# iv-a1 Calculating sample counts for Canada

Can = iotDF.select("cn" , "lcd", "c02_level")\
.where(iotDF.cn == "Canada")
# Can.show() # This shows only first 3 rows to confirm the device_name values␣
 ↪are Mac_Devices (device-mac)

from pyspark.sql import functions as saf  # sql aggrigate function for spark
clcdc = ( # counting mac devices by country using countDistinct() function
    Can.groupBy("lcd")
    .agg(saf.count("c02_level").alias("co2"))
)
clcdc.show()
```

```
+------+----+
|   lcd| co2|
+------+----+
| green|1472|
|yellow|3005|
|   red|1564|
+------+----+
```

[10]:
```
# iv-a2 Calculating sample variance for Canada

from pyspark.sql import functions as saf  # sql aggrigate function for spark
clcdv = ( # counting mac devices by country using countDistinct() function
    Can.groupBy("lcd")
    .agg(saf.variance("c02_level").alias("co2"))
)
clcdv.show()
```

```
+------+------------------+
|   lcd|               co2|
+------+------------------+
| green|3379.2623603434513|
|yellow|13335.087493325596|
|   red|3346.2973248049143|
+------+------------------+
```

[11]:
```
# iv-3 Calculating sample mean for Canada

from pyspark.sql import functions as saf  # sql aggrigate function for spark
clcdm = ( # counting mac devices by country using countDistinct() function
    Can.groupBy("lcd")
    .agg(saf.avg("c02_level").alias("co2"))
)
```

```
clcdm.show()
```

```
+------+------------------+
|   lcd|               co2|
+------+------------------+
| green| 900.4307065217391|
|yellow| 1197.944758735441|
|   red|1498.2544757033247|
+------+------------------+
```

[12]:
```
# iv-b1 Calculating sample count for India

Ind = iotDF.select("cn" , "lcd", "c02_level")\
.where(iotDF.cn == "India")

from pyspark.sql import functions as saf  # sql aggrigate function for spark
ilcdc = ( # counting mac devices by country using countDistinct() function
    Ind.groupBy("lcd")
    .agg(saf.count("c02_level").alias("co2"))
)
ilcdc.show()
```

```
+------+---+
|   lcd|co2|
+------+---+
| green|498|
|yellow|923|
|   red|446|
+------+---+
```

[13]:
```
# iv-b2 Calculating variance for India

from pyspark.sql import functions as saf  # sql aggrigate function for spark
ilcdv = ( # counting mac devices by country using countDistinct() function
    Ind.groupBy("lcd")
    .agg(saf.variance("c02_level").alias("co2"))
)
ilcdv.show()
```

```
+------+------------------+
|   lcd|               co2|
+------+------------------+
| green|3200.2285843575514|
|yellow|13632.465369221833|
|   red| 3156.623595505618|
+------+------------------+
```

[14]:
```
# iv-b3 Calculating variance for India

from pyspark.sql import functions as saf  # sql aggrigate function for spark
ilcdm = ( # counting mac devices by country using countDistinct() function
    Ind.groupBy("lcd")
    .agg(saf.avg("c02_level").alias("co2"))
)
ilcdm.show()
```

```
+------+-----------------+
|   lcd|              co2|
+------+-----------------+
| green|899.3052208835342|
|yellow|1198.943661971831|
|   red|           1493.5|
+------+-----------------+
```

[15]:
```
v1=3379.2623603434513 # Canada's green lcd variance
v2= 3200.2285843575514 # India's green lcd variance
n1 = 1325434 # Canada's green lcd count
n2 =  498 # India's green lcd count
x1_b = 900.4307065217391
x2_b = 899.3052208835342
```

The computation formula we're going to use is:

$\sigma_{\bar{X}_1-\bar{X}_2} = \sqrt{\frac{\sigma_1^2}{n_1}\frac{\sigma_2^2}{n_2}}$

The expressions are calculated from the above cell as: $\bar{X}_1$: x1_b, $\bar{X}_2$: x2_b, $\sigma_{\bar{X}_1-\bar{X}_2}$: x_diff, $\sigma_1^2$: v1, $\sigma_2^2$: v2, $n_1$: n1, $n_2$: n2.

[16]:
```
# computing the hypothesis result
import numpy as np
sigma = np.sqrt(v1/n1 + v2/n2)
x_diff = x1_b - x2_b

z = x_diff/sigma
print(z)
```

```
0.4438926906578166
```

**Conclusion** :

At 5% level of significance the test statistic 0.44 lies in the interval $[-1.96, 1.96]$. Hence we fail to reject the null hypothesis. Hence we conclude that there is no difference in using a dark lcd between these two countries.