Minimal Learning Series

From Pandas to PySpark DataFrame

A Quick Reference Guide to Pyspark DataFrame API for Experienced Pandas User

by DataPsycho

Release 2022.2.1

Contents

Preface

Back in mid 2021 after publishing a post about my small *PySpark* project in *LinkedIn*, one of my former colleague from McKinsey & Co. suggested me that I should convert my project into a book. Immediately I start planning to write this book. During the summer of 2021, I remember of taking some holiday and start writing the book. Though the book is quite small, but it took whole 2021 to make a primary draft version. That makes me realize, it is quite difficult to manage a side project while working full-time. I also spent quite a lot of time on rewriting and rearranging some sections and chapters of the book, may be due to not having any experience on writing books.

Throughout the book, I have tried my best to showcase most of my understanding and experience of using *PySpark* during my work at Socialbakers (Emplifi), where I started using *PySpark* with Databricks as my daily data analysis tool. Since 2018 I have been using *PySpark* almost every day. Though, I have not discussed more advanced topics on *PySpark* in the current version of the book, as I wanted this version of the book to be more beginner-friendly. But in the future, I could add more chapters covering advance topics on *PySpark* like machine learning and query optimization.

I will appreciate if readers can share their thoughts and ideas for further improvement of the book. Readers can contact with me via Twitter handle `@MrDataPsycho`, share their thoughts with hashtag, *#pandastopyspark* or send me emails at *mr.data.psycho@gmail.com*.

– DataPsycho; May 7, 2022

Cover Illustration

The background cover is created by *Dave Hoefler*, published in the Unsplash platform. The cover is published under Free to use Unsplash Licence. A slightly modified version or original graphics is used as background cover.

Platform Compatibility

The current version of the book is tested and published only in *PDF* friendly format. Converting the book into *EPUB* or *Kindle* friendly format or trying to read the book in *Kindle* like platform may break the format of the book. It is suggested that, readers should only use *PDF* friendly platform or software to read the book.

Chapter 1

Introduction

The ultimate reason to use any *Apache Spark* based API is being able to manipulate large amount of data distributed across multiple files (preferably) or in a single file which won't fit in the memory or even if it fits in the memory, It will take a while to transform the data using framework like *Pandas*. As you may know, *Pandas* always utilize a single core (or physical thread) of your computer, whereas *Spark* can take advantages of all or some of the available cores of your computer while processing the data. You can interact with *Spark* API using *Python* which is called *PySpark*, but it is possible to use few other languages too. In simple words, *there are many flavours of Spark, and PySpark is one of them*.

1.1 About The Book

The main focus of this book is to introduce basic *PySpark* to the audience who already have good experience of using *Pandas*. I had never been able to finish a programming book with more than five hundred pages. So, throughout the book I have tried my best to keep the contents as short as possible but at the same time informative and easy to understand. From that point of view, this book is all about how to use *PySpark DataFrame* API with respect to another popular *DataFrame* API *Pandas*.

In the book, I did not distinguish between *Spark* and *PySpark*. But in reality, the two word does not mean the same thing. *Spark* being the main technology and *PySpark* being one of the way to use the technology. So use your own judgement when you find one of the term in the book. You will find some place where *Spark* would be more appropriate, but I have used *PySpark* instead may be and vice versa.

1.2 Target Audience

This book is intended toward data enthusiastic (scientist, analyst or engineers) who already have working experience with *Pandas* at least for one year and also started learning *PySpark* or wish to learn *PySpark*.

It is very easy to forget the code you have written yesterday for a particular project to solve some use case. But when you encounter similar use case in other projects, what you do? Either you write the code again in case it is easy to write or google the solution again if it is something you have borrowed from places such as *StackOverflow* or search through your git commit to find the solution. I have faced this kind of situations many times in my 5+ years of data science and analytics career.

At first, I was thinking about writing some sort of *PySpark* reference for myself, which I can put on my office desk and search through quickly whenever needed. But later I though why not publishing a more organized version as a book which might help other fellow data analytics and data science developers.

> *"If you do not have any experience on Pandas, it will be difficult to follow the book and I would suggest you to be comfortable with Pandas' syntax first."*

1.3 Content Summary

I tried my best to not have interconnection in between the chapters. So that I can make code snippets as independent as possible and easy to follow at any part of the book. But sometimes it was not just possible to avoid inter-dependency among chapters. Though the data sets are constant throughout the book, inside a chapter any arbitrary section might depend on the previous section, but they should be less dependent from chapter to chapter. The main focus of the book is to show how to use *Pyspark* instead of *Pandas* where ever possible.

I am going to use a subset of Amazon Review data set to demonstrate the Modules of *PySpark DataFrame* API. But in each part I will first show you the solution in *Pandas* and then try to accomplish the same task in *PySpark*.

There always will be multiple ways to accomplish a particular use case, but most of the time I will show you one possible way. When you will gain more experience, you will be able to choose the way you think best, I believe.

The content flow of the book is followed by a short analytics project life-cycle, where you follow an almost predetermined set of actions to get some valuable information out of the data, as follows:

1. **Load/Read** the data (Ex. CSV, JSON, parquet) in the tabular form with *Pandas* or *PySpark*
2. **Select** fields only needed based on project requirements (sub setting)
3. **Explore** a bit, if the data is new to you
4. **Filter** or **Impute** the invalid data
5. Introduce new **Calculated Columns** based on existing columns by aggregating the data with framework (*Pandas* or *PySpark*) provided methods (group by, order by, limit etc.)
6. Finally, calculate some **Metrics** or produce **Visualization** which can easily be reviewed by business partners as a support document when taking some data driven decision

I suggest you to do one more step, which you will also see throughout the book, always make a snapshot of your working *DataFrame* when ever it makes sense to you, which will reduce extra overhead of querying the whole *DataFrame* and will make your query much faster. Also, you will be able to get rid of redundant fields from the data which you are not going to use. If you use *Pandas* then it does not help much but In case of *PySpark* use of *Cache* to create a subset of data in the memory or save a subset of the data locally and use the subset for further task will increase query speed significantly.

There is no Machine Learning content in the book, though *PySpark* is good enough for ML task as well. But I have considered it a more advanced use case and might add a separate chapter later on it. One very important note:

> *"When reading through the book you should never focus on why I am doing something but rather focus on What I am doing (Selecting, Grouping, Pivoting the data etc.) and How I am doing it using Pandas and PySpark DataFrame API. This book is not meant for teaching about data analysis but to show you, how to translate your Pandas code into PySpark code".*

1.4 Data Set

I am using Amazon's review data set on Toys & Games [1]. The data set is quite larger. If you do not have enough memory to load the data, then you can use Home & Kitchen Appliance sample data set as a proxy for the Toys & Games data set. You can download the data from an unofficial website[1].

So if you can not load 4-6 GB of data in your machine with *Pandas* try with the smaller sample of Home & Kitchen Appliance. For Toys & Games data set, I could not find metadata of the reviews. The both data is provided in a single JSON file, where each review is a JSON object. Later I will distribute the data into multiple files for better performance while using *PySpark*.

You can download the data directly from the website. But I prefer to download the data using *wget* CLI tool. The CLI tool is available for all major platforms, at least for Mac, Windows and Linux.

```
wget http://deepyeti.ucsd.edu/jianmo/amazon/
    ↪ categoryFilesSmall/Toys_and_Games_5.json.gz
wget http://deepyeti.ucsd.edu/jianmo/amazon/sample/
    ↪ sample_meta_Home_and_Kitchen.json
wget http://deepyeti.ucsd.edu/jianmo/amazon/
    ↪ categoryFilesSmall/Home_and_Kitchen_5.json.gz
```

Be aware of the line break when copying the code to the terminal. After you unzip the data by using bash: `gzip -d Home_and_Kitchen_5.json.gz` it will produce a JSON file where you will have all the data. The smaller files are not zipped (Home & Kitchen sample), so you will get them as JSON directly. Metadata includes descriptions, price, sales-rank, brand info, and

```
{
"image": ["https://..."],
"overall": 5.0,
"vote": "2",
"verified": True,
"reviewTime": "01 1, 2018",
"reviewerID": "AUI6WTTT0QZYS",
"asin": "5120053084",
"style": {
    "Size:": "Large",
    "Color:": "Charcoal"
    },
"reviewerName": "Abbey",
"reviewText": "I now have 4 ... ",
"summary": "Comfy, flattering, ...!",
"unixReviewTime": 1514764800
}
```

Listing 1: A single sample of a review

co-purchasing links. You will find an exercise at the final chapter where you will have to use the Home & Kitchen data sets and answer a set of analytical question.

> *You might not be able to download the data under enterprise VPN service as the URL is http but not https, which most of the company/enterprise computer will consider as unsecure. So you should be disconnecting from VPN for some moment while downloading the data.*

The interpretation of the field names are explained more detailed on the website. Feel free to check out the website if you wish to know more details about the data and metadata. But do not worry for now. By following the examples in the book, you will be more familiar with the data.

```json
{
  "asin": "0000031852",
  "title": "Girls Ballet Tutu Zebra Hot Pink",
  "feature": ["Botiquecutie Trademark exclusive Brand",
              "Hot Pink Layered Zebra Print Tutu"
              ],
  "description": "This tutu is great ...",
  "price": 3.17,
  "imageURL": "http://...",
  "imageURLHighRes": "http://...",
  "also_buy": ["B00JHONN1S", "B002BZX8Z6", "..."],
  "salesRank": {"Toys & Games": 211836},
  "brand": "Coxlures",
  "categories": [
    [
        "Sports & Outdoors",
        "Other Sports",
        "Dance"
    ]
  ]
}
```

Listing 2: A single sample of metadata

1.5 Hardware and OS

All the codes of this book are executed on a workstation with Intel Core i7 8^{th} Gen Processor with 8 cores and 16 GB of RAM. I use Linux-Mint as my primary OS, and I am using *Linux-Mint 20, Uliana* at the time of writing the book.

1.6 Setting Up Pandas and PySpark

Installing *PySpark* in local machine is rather complex. I suggest you to use a free online platform such as *Databricks* community. I was never able to set up *PySpark* successfully in Windows machine, so can not suggest any solution for Windows user. May be look for some solution on the internet. But if you decide to set up *PySpark* in your Linux or Mac, there is numerous tutorial available online. I am using *PySpark* with *PyCharm* community in my local machine. You also will need to install Open JDK 8 as *Spark* is based on JVM means any process written using *Spark* API runs through JVM. I have added an *environtment.yml* file into the *GitHub* repository of the project (see the Source Code section) which you can use to regenerate the same environment.

I am using 1.2.4 version of *Pandas* and 3.1.1 version of *Spark* (with *PySpark*) throughout this book. If certain code does not work for you, please check if you are trying with the exact same version or not.

1.7 Source Code

The source code of the book is available in *GitHub*. You can have a look or download the source code from the pandas-to-pyspark-ed2022 repository. The main codes for each chapter can be found in *dev* directory. As only chapter 2 has a section on writing production code, the corresponding code can be found on *prod* directory.

Chapter 2

Data IO

The most common task you will encounter in any of your data analysis project is reading the data and writing back a new form of the same data, either as final format or an updated version of the raw data. I assume you already have a good knowledge of writing Python code and also have hands-on experience on *Pandas DataFrame* API.

2.1 Summary

By the end of this chapter you should be able to:

- **Read** data into *PySpark DataFrame*
- **Rename** the columns of the DataFrame using *withColumnRenamed* method
- **Select** a subset of columns only relevant to your analysis
- **Write** the data back into disk as distributed dataset across multiple files using *Partition* and *Sorting* for better performance

Reading a dataset depends on the data source is given to you. You might need some pre-processing before you read the data into *PySpark DataFrame*. You might get CSV, JSON or Parquet format as an input source. The data set I am using is in JSON format, So I will elaborate more on reading JSON data, but I will share some references to look for other formats. You can also read data from databases to *PySpark DataFrame* using database specific *JDBC/ODBC* drivers, try to look for that solution too.

2.2 Setup

If you are not using a commercial third party managed service such as *Databricks* or *Sagemaker* then you need to start with some configuration code to set up the *PySpark* cluster. For this project, I have created a project folder Called *ExperiMind* in my local machine and then created a *.env* file in the folder and added an Environment variable called *SPARK_HOME* as follows:

```
mkdir PycharmProjects/ExperiMind
cd PycharmProjects/ExperiMind/
touch .env
echo "SPARK_HOME=/home/datapsycho/spark" >> .env
```

Listing 2.1: Setup a Spark Session

```python
from pyspark.sql import SparkSession
from dotenv import load_dotenv

def create_spark_session():
    """Create a Spark Session"""
    _ = load_dotenv()
    return (
        SparkSession
        .builder
        .appName("ExperiMind")
        .master("local[5]")
        .getOrCreate()
    )
spark = create_spark_session()
```

Due to a bug in the Debian version of *PyCharm*, if you use *PyCharm* launcher icon to start *PyCharm* then it will not load the environment variable by default from *.bashrc* file. So I am using *load_dotenv* package to load the environment variable from env file as *SparkSession* look for that variable, otherwise it will fail to create a session. If you are using managed services you do not need to use that *create_spark_session* function, there will always be a default spark session available for you with the same keyword *spark*. Which means, you should never use *spark* when assigning some variable or function, otherwise it will override the default *SparkSession* which is a common source of bug for beginners (my spark is not working I do not know why?).

In short, this function creates a Spark executor with five follower nodes and one leader node (no master or slave please), which will most probably be using 5 thread of my computer to accomplish any spark task (please do some research on your own to validate my claim). For *Pandas*, you just need to `pip install pandas`.

2.3 Read Data into DataFrame

Let's first set up a directory to save the data locally. Running the following code will first create a directory and then download the data in it. I assumed that, you have followed the previous setup to create a Project directory.

```
cd PycharmProjects/ExperiMind
mkdir -p data/raw
cd data/raw
wget http://deepyeti.ucsd.edu/jianmo/amazon/
    ↪ categoryFilesSmall/Toys_and_Games_5.json.gz
gzip -d Toys_and_Games_5.json.gz
cd ..
```

In the previous bash instructions, I have created a folder to store the raw data, then I have used *wget* to download the large compressed JSON file and extract the compressed file in to the following same directory. Finally, I have moved one step back to our project root, where I have my *Python* files. It is possible to read JSON data directly into *Pandas* if the format is known to *Pandas*. Let's read the data into *Pandas* first:

Listing 2.2: Read Json Data into Pandas with Built-In Method

```python
import pandas as pd
PATH_BIGDATA = 'data/raw/Toys_and_Games_5.json'
# This code fails with run time error
raw_pdf = pd.read_json(PATH_BIGDATA, orient='records')
```

But using the following code, I was not able to read the data. I also tried without *orient* argument, but no luck with that. Then I checked documentation and by following the document here is one of the way to read the data:

Listing 2.3: Read Json Data into Pandas with Custom Method

```python
import pandas as pd
from tqdm import tqdm
import json

PATH_BIGDATA = 'data/raw/Toys_and_Games_5.json'

def read_json_to_pdf(path: str) -> pd.DataFrame:
    data = []
    with open(path, 'r') as f:
        for line in tqdm(f):
            data.append(json.loads(line))
    df = pd.DataFrame(data)
    return df

raw_pdf = read_json_to_pdf(PATH_BIGDATA)
```

As each line in the file is a JSON object of review, we can iterate through each line and convert it to dictionary and append the result into a python *list*. Then we can use *DataFrame* class to convert the list of python dictionaries into a *DataFrame*. Here *tqdm* package is used just to show the progress of the task, nothing special. It will take quite a while to load the data. The code is partially taken from the maintainer's documentation.

To load the data into Spark *DataFrame* I tried to use `spark.read.json()`. But It failed to load the data. Then I found a stack overflow suggestion which works fine as follows:

Listing 2.4: Read Json Data into PySpark

```python
from pyspark.sql import SparkSession
from dotenv import load_dotenv

def create_spark_session():
    """Create a Spark Session"""
    # code from listing 2.1
    # ...
    return
```

```

spark = create_spark_session()

spark.conf.set("spark.sql.caseSensitive", "true")
PATH_BIGDATA = 'data/raw/Toys_and_Games_5.json'
raw_sdf = spark.read.json(PATH_BIGDATA)
```

I had to change the default setting of Case Sensitivity of Column Naming to be able to read the data. Here, the *spark* object is the *SparkSession* I have created in the previous section.

So what is the Difference between the two Framework when reading data files? As you can see, *PySpark* and *Pandas* can read a JSON file directly into the *DataFrame* if each of the records is a JSON object in the file. But behind the scene, *Spark* does not read the whole content of the file in the memory. But *Pandas* does. *PySpark* just created a query plan and metadata schema to identify the column type (I am explaining in general, it is much more complex than that actually). I will show you a proof of that at the end of this chapter.

Reading data of course depends on the structure of the data and the file type. But whether it is a CSV or JSON if the format inside the file is known to *PySpark*, it will be able to read the data much easily like *Pandas*. Spark by Example is a fantastic resource about *Spark*, and they have a chapter on reading data into *Spark*, please check it here.

> *"As Spark does not load all the data in to memory, there is a caveat in case of distributed data. You might encounter with run time error when PySpark try to manipulate certain part of the data and find a miss match. It happens mostly when data is distributed across multiple file and one of the file is corrupted or malformed, which Spark did not know about when inferring the Schema."*

2.4 Rename Attributes

In an Organization, data is generated from different sources, systems and processes before it reaches to you. So the column naming might surprise you. You will encounter different way of column naming such as *camelCase*,