# Quickstart: Pandas API on Spark

This is a short introduction to pandas API on Spark, geared mainly for new users. This notebook shows you some key differences between pandas and pandas API on Spark. You can run this examples by yourself in 'Live Notebook: pandas API on Spark' at <a href="mailto:the quickstart">the quickstart</a> page.

Customarily, we import pandas API on Spark as follows:

```
[1]: import pandas as pd
import numpy as np
import pyspark.pandas as ps
from pyspark.sql import SparkSession
```

# **Object Creation**

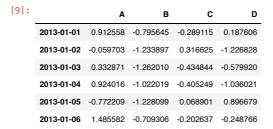
Creating a pandas-on-Spark Series by passing a list of values, letting pandas API on Spark create a default integer index:

Creating a pandas-on-Spark DataFrame by passing a dict of objects that can be converted to series-like.

[5]: psdf

```
30 3 300 three
40 4 400 four
50 5 500 five
60 6 600 six
```

Creating a pandas DataFrame by passing a numpy array, with a datetime index and labeled columns:

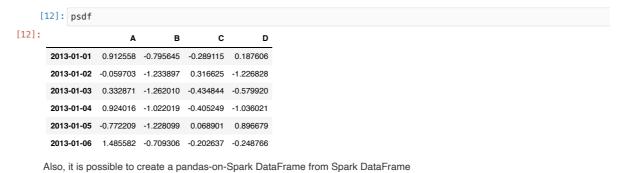


Now, this pandas DataFrame can be converted to a pandas-on-Spark DataFrame

```
[10]: psdf = ps.from_pandas(pdf)
[11]: type(psdf)
```

[11]: pyspark.pandas.frame.DataFrame

It looks and behaves the same as a pandas DataFrame.



easily.

Creating a Spark DataFrame from pandas DataFrame

```
[13]: spark = SparkSession.builder.getOrCreate()
[14]: sdf = spark.createDataFrame(pdf)
[15]: sdf.show()
```

Creating pandas-on-Spark DataFrame from Spark DataFrame.

```
[16]: psdf = sdf.pandas_api()

[17]: psdf

A B C D

0 0.912558 -0.795645 -0.289115 0.187606

1 -0.059703 -1.233897 0.316625 -1.226828

2 0.332871 -1.262010 -0.434844 -0.579920

3 0.924016 -1.022019 -0.405249 -1.036021

4 -0.772209 -1.228099 0.068901 0.896679

5 1.485582 -0.709306 -0.202637 -0.248766

Having specific dtypes . Types that are common to both Spark and pandas are currently supported.
```

[18]: psdf.dtypes

```
[18]: A float64
B float64
C float64
D float64
dtype: object
```

Here is how to show top rows from the frame below.

Note that the data in a Spark dataframe does not preserve the natural order by default.

The natural order can be preserved by setting compute.ordered\_head option but it causes a performance overhead with sorting internally.

```
[19]: psdf.head()
```

# [19]: A B C D 0 0.912558 -0.795645 -0.289115 0.187606 1 -0.059703 -1.233897 0.316625 -1.226828 2 0.332871 -1.262010 -0.434844 -0.579920 3 0.924016 -1.022019 -0.405249 -1.036021 4 -0.772209 -1.228099 0.068901 0.896679

Displaying the index, columns, and the underlying numpy data.

Showing a quick statistic summary of your data

```
[23]: psdf.describe()
```

[23]:		Α	В	С	D
	count	6.000000	6.000000	6.000000	6.000000
	mean	0.470519	-1.041829	-0.157720	-0.334542
	std	0.809428	0.241511	0.294520	0.793014
	min	-0.772209	-1.262010	-0.434844	-1.226828
	25%	-0.059703	-1.233897	-0.405249	-1.036021
	50%	0.332871	-1.228099	-0.289115	-0.579920
	75%	0.924016	-0.795645	0.068901	0.187606
	max	1.485582	-0.709306	0.316625	0.896679

Transposing your data

```
[24]: psdf.T
```

[24]:		0	1	2	3	4	5
	Α	0.912558	-0.059703	0.332871	0.924016	-0.772209	1.485582
	В	-0.795645	-1.233897	-1.262010	-1.022019	-1.228099	-0.709306
	С	-0.289115	0.316625	-0.434844	-0.405249	0.068901	-0.202637
	D	0.187606	-1.226828	-0.579920	-1.036021	0.896679	-0.248766

Sorting by its index

```
[25]: psdf.sort_index(ascending=False)
```



## Missing Data

Pandas API on Spark primarily uses the value np. nan to represent missing data. It is by default not included in computations.

```
[27]: pdf1 = pdf.reindex(index=dates[0:4], columns=list(pdf.columns) + ['E'])
    [28]: pdf1.loc[dates[0]:dates[1], 'E'] = 1
    [29]: psdf1 = ps.from_pandas(pdf1)
    [30]: psdf1
[30]:
      2013-01-01 0.912558 -0.795645 -0.289115 0.187606
      2013-01-02 -0.059703 -1.233897 0.316625 -1.226828
      2013-01-03 0.332871 -1.262010 -0.434844 -0.579920 NaN
      To drop any rows that have missing data.
    [31]: psdf1.dropna(how='any')
[31]:
      2013-01-01 0.912558 -0.795645 -0.289115 0.187606 1.0
      2013-01-02 -0.059703 -1.233897 0.316625 -1.226828 1.0
     Filling missing data.
    [32]: psdf1.fillna(value=5)
[32]:
      2013-01-01 0.912558 -0.795645 -0.289115 0.187606 1.0
      2013-01-02 -0.059703 -1.233897 0.316625 -1.226828 1.0
      2013-01-03 0.332871 -1.262010 -0.434844 -0.579920 5.0
      2013-01-04 0.924016 -1.022019 -0.405249 -1.036021 5.0
```

# **Operations**

#### Stats

Performing a descriptive statistic:

```
[33]: psdf.mean()

[33]: A 0.470519
B -1.041829
C -0.157720
D -0.334542
dtype: float64
```

## Spark Configurations

Various configurations in PySpark could be applied internally in pandas API on Spark. For example, you can enable Arrow optimization to hugely speed up internal pandas conversion. See also PySpark Usage Guide for Pandas with Apache Arrow in PySpark documentation.

```
[34]: prev = spark.conf.get("spark.sql.execution.arrow.pyspark.enabled") # Keep its default value.
    ps.set_option("compute.default_index_type", "distributed") # Use default index prevent overhead.
    import warnings
    warnings.filterwarnings("ignore") # Ignore warnings coming from Arrow optimizations.

[35]: spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", True)
    %timeit ps.range(300000).to_pandas()

[36]: spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", False)
    %timeit ps.range(300000).to_pandas()
[37]: ps.reset_option("compute.default_index_type")
    spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", prev) # Set its default value back.
```

## Grouping

By "group by" we are referring to a process involving one or more of the following steps:

- Splitting the data into groups based on some criteria
- Applying a function to each group independently
- · Combining the results into a data structure

[39]: С **0** foo 1.039632 -0.571950 0.972089 1.085353 1 bar one **2** foo two -1.931621 -2.579164 3 bar three -0.654371 -0.340704 4 foo two -0.157080 0.893736 5 bar 0.882795 0.024978 6 foo one -0.149384 0.201667

7 foo three -1.355136 0.693883

Grouping and then applying the <a href="mailto:sum()">sum()</a> function to the resulting groups.

```
[40]: psdf.groupby('A').sum()
```



Grouping by multiple columns forms a hierarchical index, and again we can apply the sum function.

```
[41]: psdf.groupby(['A', 'B']).sum()

[41]: c D

A B

foo one 0.890248 -0.370283
two -2.088701 -1.685428
bar three -0.654371 -0.340704
foo three -1.355136 0.693883
bar two 0.882795 0.024978
one 0.972089 1.085353
```

## **Plotting**

Q Search the docs ...

Installation

Quickstart: DataFrame

Quickstart: Spark Connect

Quickstart: Pandas API on Spark

Testing PySpark

On a DataFrame, the <u>plot()</u> method is a convenience to plot all of the columns with labels:

```
[49]: psdf.plot()
```

For more details, Plotting documentation.

# Getting data in/out

### **CSV**

CSV is straightforward and easy to use. See  $\underline{\text{here}}$  to write a CSV file and  $\underline{\text{here}}$  to read a CSV file.

```
[50]: psdf.to_csv('foo.csv') head(10)

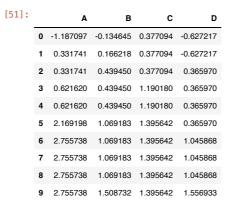
[50]: A B C D

0 -1.187097 -0.134645 0.377094 -0.627217
1 0.331741 0.166218 0.377094 -0.627217
2 0.331741 0.439450 0.377094 0.365970
3 0.621620 0.439450 1.190180 0.365970
4 0.621620 0.439450 1.190180 0.365970
5 2.169198 1.069183 1.395642 0.365970
6 2.755738 1.069183 1.395642 1.045868
7 2.755738 1.069183 1.395642 1.045868
8 2.755738 1.069183 1.395642 1.045868
9 2.755738 1.069183 1.395642 1.045868
```

### Parquet

Parquet is an efficient and compact file format to read and write faster. See <u>here</u> to write a Parquet file and <u>here</u> to read a Parquet file.

```
[51]: psdf.to_parquet('bar.parquet')
ps.read_parquet('bar.parquet').head(10)
```



**Quickstart: Spark Connect** 

## Spark IO

In addition, pandas API on Spark fully supports Spark's various datasources such as ORC and an external datasource. See <a href="here">here</a> to write it to the specified datasource and <a href="here">here</a> to read it from the datasource.

```
[52]: psdf.to_spark_io('zoo.orc', format="orc")
           ps.read_spark_io('zoo.orc', format="orc").head(10)
[52]:
                         В
                                 C
                                          D
               Δ
      0 -1.187097 -0.134645 0.377094 -0.627217
       1 0.331741 0.166218 0.377094 -0.627217
       2 0.331741 0.439450 0.377094 0.365970
       3 0.621620 0.439450 1.190180 0.365970
       4 0.621620 0.439450 1.190180 0.365970
       5 2.169198 1.069183 1.395642 0.365970
         2.755738 1.069183 1.395642 1.045868
       7 2.755738 1.069183 1.395642 1.045868
       8 2.755738 1.069183 1.395642 1.045868
         2.755738 1.508732 1.395642 1.556933
     See the Input/Output documentation for more details.
         Previous
                                                                             Next
```

**Testing PySpark** 

© Copyright .

Created using Sphinx 3.0.4.