2_df_operations

February 22, 2022

1 Basic Operations

How to perform basic operations on data frames?

We will play around with some stock data from Apple.

[1]:
```
# Import SparkContext and SparkConf
#from pyspark import SparkContext, SparkConf
# Initialize spark
#conf = SparkConf().setAppName("createRDD").setMaster("local[4]")
#sc = SparkContext(conf=conf)

from pyspark.sql import SparkSession
```

[2]:
```
spark = SparkSession.builder.appName("Operations").getOrCreate()
```

```
22/02/22 10:41:17 WARN Utils: Your hostname, ThinkCentre resolves to a loopback
address: 127.0.1.1; using 10.180.5.223 instead (on interface eno1)
22/02/22 10:41:17 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
22/02/22 10:41:17 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
22/02/22 10:41:19 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
```

[3]:
```
# Let Spark know about the header and infer the Schema types!
df = spark.read.csv('appl_stock.csv',inferSchema=True,header=True)
```

[4]:
```
# The column can contain null values if the nullable property
# is set to true.

df.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)
```

[5]:
```
df.show(5)
```

```
+----------+----------+----------+------------------+------------------+--------
-+------------------+
|      Date|      Open|      High|               Low|             Close|
Volume|         Adj Close|
+----------+----------+----------+------------------+------------------+--------
-+------------------+
|2010-01-04|213.429998|214.499996|212.38000099999996|
214.009998|123432400|         27.727039|
|2010-01-05|214.599998|215.589994|        213.249994|
214.379993|150476200|27.774976000000002|
|2010-01-06|214.379993|    215.23|        210.750004|
210.969995|138040000|27.333178000000004|
|2010-01-07|    211.75|212.000006|        209.050005|
210.58|119282800|          27.28265|
|2010-01-08|210.299994|212.000006|209.06000500000002|211.98000499999998|11190270
0|         27.464034|
+----------+----------+----------+------------------+------------------+--------
-+------------------+
only showing top 5 rows
```

[6]:
```
df.head()
```

[6]:
```
Row(Date='2010-01-04', Open=213.429998, High=214.499996, Low=212.38000099999996,
Close=214.009998, Volume=123432400, Adj Close=27.727039)
```

[7]:
```
print((df.count(), len(df.columns)))
```

```
(1762, 7)
```

2 Filtering Data

- A large part of working with DataFrames is the ability to quickly filter out data based on conditions.
- Spark DataFrames are built on top of the Spark SQL platform, which means that is you already know SQL, you can quickly and easily grab that data using SQL commands, or using

the DataFrame methods.

[10]:
```python
# Filter the rows which are having Close index less than 200

# Using SQL -- all give the same result
df.filter("Close<500").show(5)
```

```
+----------+----------+----------+------------------+------------------+--------
-+-------------------+
|      Date|      Open|      High|               Low|             Close|
Volume|          Adj Close|
+----------+----------+----------+------------------+------------------+--------
-+-------------------+
|2010-01-04|213.429998|214.499996|212.38000099999996|
214.009998|123432400|          27.727039|
|2010-01-05|214.599998|215.589994|        213.249994|
214.379993|150476200|27.774976000000002|
|2010-01-06|214.379993|    215.23|        210.750004|
210.969995|138040000|27.333178000000004|
|2010-01-07|    211.75|212.000006|        209.050005|
210.58|119282800|          27.28265|
|2010-01-08|210.299994|212.000006|209.06000500000002|211.98000499999998|11190270
0|          27.464034|
+----------+----------+----------+------------------+------------------+--------
-+-------------------+
only showing top 5 rows
```

[11]:
```python
df.filter("Close<500").show(5)
df.filter(df["Close"]<500).show(5)
df.filter(df.Close<500).show(5)
```

```
+----------+----------+----------+------------------+------------------+--------
-+-------------------+
|      Date|      Open|      High|               Low|             Close|
Volume|          Adj Close|
+----------+----------+----------+------------------+------------------+--------
-+-------------------+
|2010-01-04|213.429998|214.499996|212.38000099999996|
214.009998|123432400|          27.727039|
|2010-01-05|214.599998|215.589994|        213.249994|
214.379993|150476200|27.774976000000002|
|2010-01-06|214.379993|    215.23|        210.750004|
210.969995|138040000|27.333178000000004|
|2010-01-07|    211.75|212.000006|        209.050005|
210.58|119282800|          27.28265|
|2010-01-08|210.299994|212.000006|209.06000500000002|211.98000499999998|11190270
0|          27.464034|
+----------+----------+----------+------------------+------------------+--------
-+------------------+
only showing top 5 rows
```

[12]:
```
df.filter(df.Close<500).show(5)
```

```
+----------+----------+----------+------------------+------------------+--------
-+------------------+
|      Date|      Open|      High|               Low|             Close|
Volume|         Adj Close|
+----------+----------+----------+------------------+------------------+--------
-+------------------+
|2010-01-04|213.429998|214.499996|212.38000099999996|
214.009998|123432400|         27.727039|
|2010-01-05|214.599998|215.589994|        213.249994|
214.379993|150476200|27.774976000000002|
|2010-01-06|214.379993|    215.23|        210.750004|
210.969995|138040000|27.333178000000004|
|2010-01-07|    211.75|212.000006|        209.050005|
210.58|119282800|         27.28265|
|2010-01-08|210.299994|212.000006|209.06000500000002|211.98000499999998|11190270
0|         27.464034|
+----------+----------+----------+------------------+------------------+--------
-+------------------+
only showing top 5 rows
```

[13]:
```
# Filter the rows which are having Open index less than 200,
# print only that column

# Using SQL with .select()
# select() is to select a column
df.filter("Close<200").select('Open').show(5)
```

```
+------------------+
|              Open|
+------------------+
|206.78000600000001|
|        204.930004|
|        201.079996|
|192.36999699999998|
|        195.909998|
+------------------+
only showing top 5 rows
```

```
[14]: # Using SQL with .select(), multiple columns
df.filter("Close<200").select(['Open','Close']).show(5)
```

```
+------------------+----------+
|              Open|     Close|
+------------------+----------+
|206.78000600000001|    197.75|
|        204.930004|199.289995|
|        201.079996|192.060003|
|192.36999699999998|194.729998|
|        195.909998|195.859997|
+------------------+----------+
only showing top 5 rows
```

Using normal python comparison operators is another way to do this, they will look very similar to SQL operators, except you need to make sure you are calling the entire column within the dataframe, using the format: df["column name"]

Let's see some examples:

```
[15]: df.filter(df["Close"] < 200).show(5)
```

```
+----------+------------------+----------+------------------+----------+--------
-+------------------+
|      Date|              Open|      High|               Low|     Close|
Volume|         Adj Close|
+----------+------------------+----------+------------------+----------+--------
-+------------------+
|2010-01-22|206.78000600000001|207.499996|            197.16|
197.75|220441900|         25.620401|
|2010-01-28|        204.930004|205.500004|
198.699995|199.289995|293375600|25.819922000000002|
|2010-01-29|        201.079996|202.199995|
190.250002|192.060003|311488100|         24.883208|
|2010-02-01|192.36999699999998|
196.0|191.29999899999999|194.729998|187469100|         25.229131|
|2010-02-02|        195.909998|196.319994|193.37999299999998|195.859997|17458560
0|25.375532999999997|
+----------+------------------+----------+------------------+----------+--------
-+------------------+
only showing top 5 rows
```

```
[ ]: # Will produce an error, make sure to read the error!
# df.filter(df["Close"] < 200 and df['Open'] > 200).show()
```

```
[ ]: # Make sure to add in the parenthesis separating the statements!
     df.filter( (df["Close"] < 200) & (df['Open'] > 200) ).show()
```

```
[16]: # Make sure to add in the parenthesis separating the statements!
      df.filter( (df["Close"] < 200) | (df['Open'] > 200) ).show(5)
```

```
+----------+----------+----------+------------------+------------------+--------
-+------------------+
|      Date|      Open|      High|               Low|             Close|
Volume|         Adj Close|
+----------+----------+----------+------------------+------------------+--------
-+------------------+
|2010-01-04|213.429998|214.499996|212.38000099999996|
214.009998|123432400|         27.727039|
|2010-01-05|214.599998|215.589994|        213.249994|
214.379993|150476200|27.774976000000002|
|2010-01-06|214.379993|    215.23|        210.750004|
210.969995|138040000|27.333178000000004|
|2010-01-07|    211.75|212.000006|        209.050005|
210.58|119282800|          27.28265|
|2010-01-08|210.299994|212.000006|209.06000500000002|211.98000499999998|11190270
0|         27.464034|
+----------+----------+----------+------------------+------------------+--------
-+------------------+
only showing top 5 rows
```

```
[17]: # Make sure to add in the parenthesis separating the statements!
      df.filter( (df["Close"] < 200) & ~(df['Open'] < 200) ).show()
```

```
+----------+------------------+----------+----------+----------+---------+------
------------+
|      Date|              Open|      High|       Low|     Close|   Volume|
Adj Close|
+----------+------------------+----------+----------+----------+---------+------
------------+
|2010-01-22|206.78000600000001|207.499996|    197.16|    197.75|220441900|
25.620401|
|2010-01-28|
204.930004|205.500004|198.699995|199.289995|293375600|25.819922000000002|
|2010-01-29|        201.079996|202.199995|190.250002|192.060003|311488100|
24.883208|
+----------+------------------+----------+----------+----------+---------+------
------------+
```

```
[18]: df.filter(df["Low"] == 197.16).show()
```

```
+----------+------------------+----------+------+------+---------+---------+
|      Date|              Open|      High|   Low| Close|   Volume|Adj Close|
+----------+------------------+----------+------+------+---------+---------+
|2010-01-22|206.78000600000001|207.499996|197.16|197.75|220441900|25.620401|
+----------+------------------+----------+------+------+---------+---------+
```

[19]:
```
# Collecting results as Python objects
df.filter(df["Low"] == 197.16).collect()
```

[19]:
```
[Row(Date='2010-01-22', Open=206.78000600000001, High=207.499996, Low=197.16,
Close=197.75, Volume=220441900, Adj Close=25.620401)]
```

[21]:
```
result = df.filter(df["Low"] == 197.16).collect()
```

[22]:
```
# Note the nested structure returns a nested row object
type(result[0])
```

[22]:
```
pyspark.sql.types.Row
```

[23]:
```
row = result[0]
```

[ ]:
```
type(result[0])
```

Rows can be called to turn into dictionaries

[24]:
```
row
```

[24]:
```
Row(Date='2010-01-22', Open=206.78000600000001, High=207.499996, Low=197.16,
Close=197.75, Volume=220441900, Adj Close=25.620401)
```

[25]:
```
row.count(197.75)
```

[25]:
```
1
```

[26]:
```
row.index(197.75)
```

[26]:
```
4
```

[27]:
```
row.asDict()
```

[27]:
```
{'Date': '2010-01-22',
 'Open': 206.78000600000001,
 'High': 207.499996,
 'Low': 197.16,
 'Close': 197.75,
 'Volume': 220441900,
 'Adj Close': 25.620401}
```

[28]:
```
for item in row.asDict().keys():
#     print(type(item))
    print(item)
```

```
Date
Open
High
Low
Close
Volume
Adj Close
```

[29]:
```
for item in row.asDict().values():
#     print(type(item))
    print(item)
```

```
2010-01-22
206.78000600000001
207.499996
197.16
197.75
220441900
25.620401
```