Using Docker and PySpark

Bryant Crocker
Jan 10, 2019 · 7 min read

Recently, I have been playing with PySpark a bit and decided I would write a blog post about using PySpark and Spark SQL. Spark is a great open source tool for munging data and machine learning across distributed computing clusters. PySpark is the python API to Spark.

PySpark can be a bit difficult to get up and running on your machine. Docker is a quick and easy way to get a Spark environment working on your local machine and is how I run PySpark on my local machine.

What is Docker?

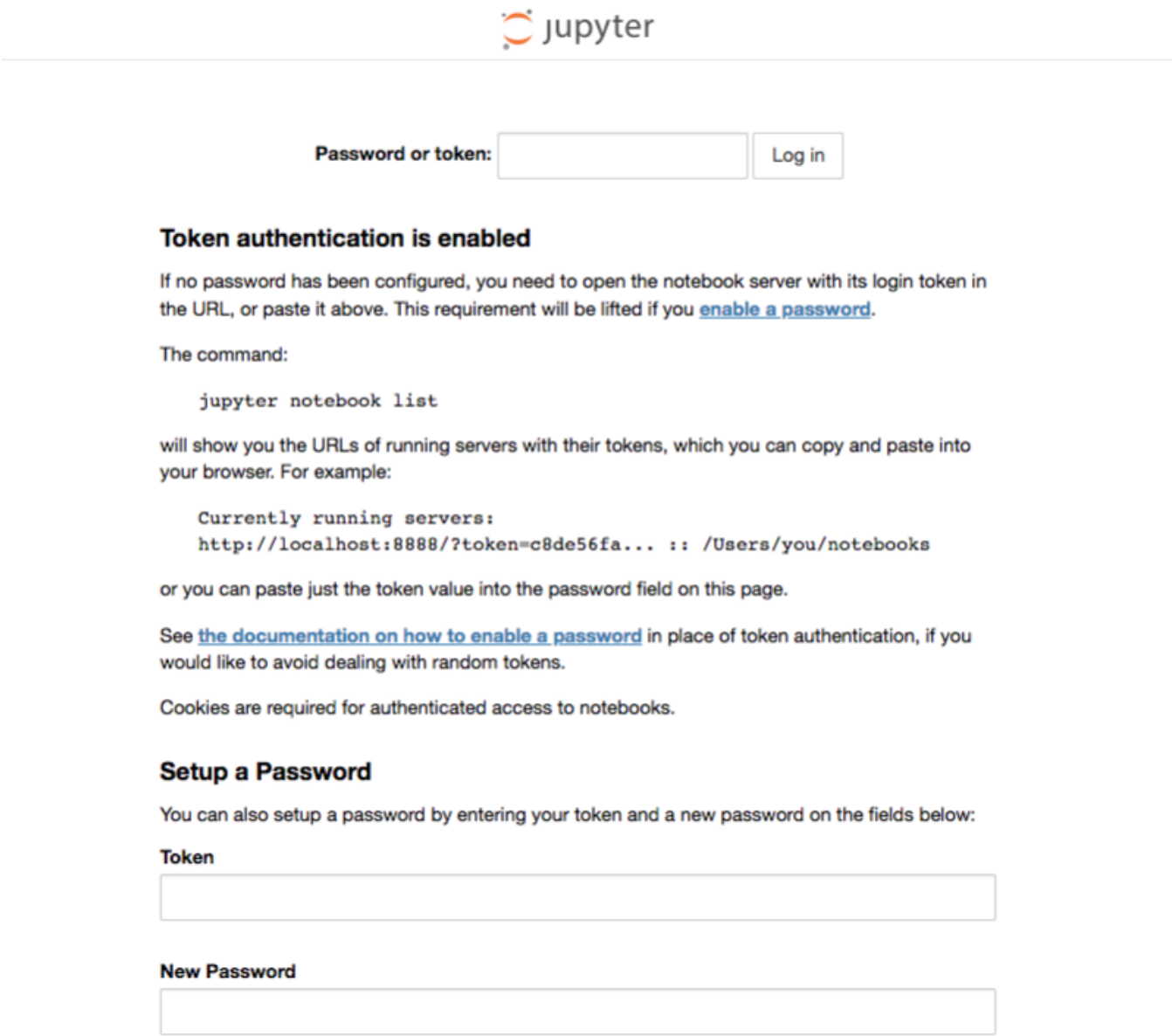
I'll start by giving an introduction to Docker. According to wikipedia "Docker is a computer program that performs operating-system-level virtualization, also known as 'containerization' ". To greatly simplify, Docker creates a walled off linux operating system to run software on top of your machine's OS called a container. For those familiar with virtual machines, a container is basically a vm without a hypervisor. These containers can be preconfigured with scripts to install specific software and provide customized functionality. Dockerhub is a website that contains various preconfigured docker containers that can be quickly run on your computer. One of these is the jupyter/pysparknotebook. This is the docker image we will be using today.

Starting up the Docker container:

Setting up a Docker container on your local machine is pretty simple. Simply download docker from the docker website and run the following command in the terminal:

```
docker run -it -p 8888:8888 jupyter/pyspark-notebook
```

navigate to http://localhost:8888 in your browser and you will see the following screen:



In your terminal you should see a token:

```
    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(9e955fe93eda or 127.0.0.1):8888/?token=230d9e459b7509e61e62bebeb
79eed910252cee23259399e
```

copy and paste this token, the numbers following "/?token=", into the token textbook and set a password for the Jupyter notebook server in the New Password box.

With that done, you are all set to go! Spark is already installed in the container. You are all ready to open up a notebook and start writing some Spark code. I will include a copy

of the notebook but I would recommend entering the code from this article into a new Jupyter notebook on your local computer. This helps you to learn.

To stop the docker container and Jupyter notebook server, simply enter control + c in the terminal that is running it.

PySpark Basics

Spark is an open source cluster computing framework written in mostly scala with APIs in R, python, scala and java. It is made mostly for large scale data analysis and machine learning that cannot fit into local memory. In this brief tutorial, I will not use a dataset that is too big to fit into memory. This tutorial borrows from the official getting starting guide: https://spark.apache.org/docs/latest/sql-getting-started.html.

Spark Datatypes:

There are two main datatypes in the spark ecosystem, Resilient Distributed Datasets or RDDs (which are kind of like a cross between a python list and dictionary) and dataframes (dataframes much like in R and python). Both data types in spark are partitioned and immutable (which means you cannot change the object, a new one is returned instead). In this tutorial I am going to focus on the dataframe datatype.

The Dataset:

The dataset that I will be using is a somewhat large Vermont vendor data dataset from the Vermont open data Socrata portal. It can be downloaded easily by following the link.

Setting up a Spark session:

This code snippet starts up the PySpark enviroment in the docker container and imports basic libraries for numerical computing.

```
# import necessary libraries
import pandas as pd
import numpy
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession

# create sparksession
spark = SparkSession \
    .builder \
    .appName("Pysparkexample") \
```

```
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Reading in a CSV:

I wanted to start by comparing reading in a CSV with pandas vs Spark. Spark ends up reading in the CSV much faster than pandas. This demonstrates how Spark dataframes are much faster when compared to their pandas equivalent.

```
In [28]: %%timeit
         df = spark.read.csv('Vermont_Vendor_Payments (1).csv', header='true')

         167 ms ± 6.34 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [29]: %%timeit
         df_pandas = pd.read_csv('Vermont_Vendor_Payments (1).csv', low_memory = False)

         4.53 s ± 109 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

For this analysis I will read in the data using the inferSchema option and cast the Amount column to a double.

```
df = spark.read.csv('Vermont_Vendor_Payments (1).csv',
header='true', inferSchema = True)
df = df.withColumn("Amount", df["Amount"].cast("double"))
```

Basic Spark Methods:

like with pandas, we access column names with the .columns attribute of the dataframe.

```
#we can use the columns attribute just like with pandas
columns = df.columns
print('The column Names are:')
for i in columns:
    print(i)
```

```
The column Names are:
Quarter Ending
Department
UnitNo
Vendor Number
Vendor
City
```

```
State
DeptID Description
DeptID
Amount
Account
AcctNo
Fund Description
Fund
```

We can get the number of rows using the .count() method and we can get the number of columns by taking the length of the column names.

```
print('The total number of rows is:', df.count(), '\nThe total
number of columns is:', len(df.columns))
```

```
The total number of rows is: 1484734
The total number of columns is: 14
```

The .show() method prints the first 20 rows of the dataframe by default. I chose to only print 5 in this article.

```
#show first 5 rows
df.show(5)
```

```
+--------------+--------------------+------+-------------+--------------------+----+-----+--------------------+------
----+------+--------------------+------+--------------------+-----+
|Quarter Ending|          Department|UnitNo|Vendor Number|              Vendor|City|State|  DeptID Description|    De
ptID|Amount|             Account|AcctNo|    Fund Description| Fund|
+--------------+--------------------+------+-------------+--------------------+----+-----+--------------------+------
----+------+--------------------+------+--------------------+-----+
|    09/30/2009|Environmental Con...| 06140|   0000276016|1st Run Computer ...|null|   NY|     WQD - Waterbury|614004
0206|930.00|Rep&Maint-Info Te...|513000|Environmental Per...|21295|
|    09/30/2009|Environmental Con...| 06140|   0000276016|1st Run Computer ...|null|   NY|Water Supply Divi...|614004
0406|930.00|Rep&Maint-Info Te...|513000|Environmental Per...|21295|
|    09/30/2009|Vermont Veterans'...| 03300|   0000284121| 210 Innovations LLC|null|   CT|         MAINTENANCE|330001
0300| 24.00|Freight & Express...|517300|    Vermont Medicaid|21782|
|    09/30/2009|Vermont Veterans'...| 03300|   0000284121| 210 Innovations LLC|null|   CT|         MAINTENANCE|330001
0300|420.00|Building Maintena...|520200|    Vermont Medicaid|21782|
|    09/30/2009|         Corrections| 03480|   0000207719|21st Century Cell...|null|   PA|     Brattleboro P&P|348000
4630|270.80|Telecom-Wireless ...|516659|        General Fund|10000|
+--------------+--------------------+------+-------------+--------------------+----+-----+--------------------+------
----+------+--------------------+------+--------------------+-----+
only showing top 5 rows
```

The .head() method can also be used to display the first row. This prints much nicer in the notebook.

```
#show first row
df.head()
```

```
Row(Quarter Ending='09/30/2009', Department='Environmental Conservation', UnitNo='06140', Vendor Number='0000276016',
Vendor='1st Run Computer Services Inc', City=None, State='NY', DeptID Description='WQD - Waterbury', DeptID='61400402
06', Amount='930.00', Account='Rep&Maint-Info Tech Hardware', AcctNo='513000', Fund Description='Environmental Permit
Fund', Fund='21295')
```

Like in pandas, we can call the describe method to get basic numerical summaries of the data. We need to use the show method to print it to the notebook.This does not print very nicely in the notebook.

```
df.describe().show()
```

```
+-------+--------------+--------------------+------------------+------------------+--------------------+--------+----
--------------+------------------+--------------------+--------------------+------------------+------------------+--
------------------+--------------------+
|summary|Quarter Ending|          Department|            UnitNo|     Vendor Number|              Vendor|    City|
         State|DeptID Description|              DeptID|              Amount|           Account|            AcctNo|
  Fund Description|                Fund|
+-------+--------------+--------------------+------------------+------------------+--------------------+--------+----
--------------+------------------+--------------------+--------------------+------------------+------------------+--
------------------+--------------------+
|  count|       1484734|             1484734|           1484734|           1484734|             1484734|  742411|
       1484686|           1484197|             1484734|             1484734|           1484734|           1484734|
          1484732|             1484733|
|   mean|          null|                null|4055.2432354886464|101513.27828033261|                null|     0.0|2.23
88059701492535|              null|4.0563299528571525E9|  195691.78465843684|7.177366097966365E8| 532239.5738273305|
517532.55924170616|  25816.550023812364|
| stddev|          null|                null| 2321.533688620958| 118133.2225931531|                null|     0.0|12.8
59484829021829|              null| 2.321553861897507E9|1.4713605127641063E7| 5.64268575671878E8|30379.608056930345|
4509.844363278799|   19200.86011688401|
|    min|    03/31/2010|AOT Proprietary F...|             01100|        0000000002|"Jewett,Martin A ...|       0|
            0|         """Admin.|             CCV"""|               -0.01|            -294.00|           -294.00|
           507200|               10000|
|    max|    12/31/2017|  Women's Commission|             09150|            SINGLE|           xAd, Inc.|w Berlin|
           ZZ|     Youth at Risk|                Seg|                Din"|  Youth Development|       Water/Sewer|Yo
uth Substance A...|Facilities Operat...|
+-------+--------------+--------------------+------------------+------------------+--------------------+--------+----
--------------+------------------+--------------------+--------------------+------------------+------------------+--
------------------+--------------------+
```

Querying the data:

One of the strengths of Spark is that it can be queried with each language's respective Spark library or with Spark SQL. I will demonstrate a few queries using both the pythonic and SQL options.

The following code registers temporary table and selects a few columns using SQL syntax:

```
# I will start by creating a temporary table query with SQL
df.createOrReplaceTempView('VermontVendor')
spark.sql(
'''
SELECT `Quarter Ending`, Department, Amount, State FROM
VermontVendor
LIMIT 10
'''
).show()
```

```
+--------------+--------------------+------+-----+
|Quarter Ending|          Department|Amount|State|
+--------------+--------------------+------+-----+
|    09/30/2009|Environmental Con...|930.00|   NY|
|    09/30/2009|Environmental Con...|930.00|   NY|
|    09/30/2009|Vermont Veterans'...| 24.00|   CT|
|    09/30/2009|Vermont Veterans'...|420.00|   CT|
|    09/30/2009|         Corrections|270.80|   PA|
|    09/30/2009|         Corrections| 35.00|   PA|
|    09/30/2009|       Public Safety|971.40|   PA|
|    09/30/2009|Agriculture, Food...| 60.59|   TX|
|    09/30/2009|Agriculture, Food...|541.62|   TX|
|    09/30/2009|              Health|283.98|   PA|
+--------------+--------------------+------+-----+
```

This code performs pretty much the same operation using pythonic syntax:

```
df.select('Quarter Ending', 'Department', 'Amount',
'State').show(10)
```

```
+--------------+----------+-------+-----+
|Quarter Ending|Department| Amount|State|
+--------------+----------+-------+-----+
|    09/30/2009| Education|9423.36|   VT|
|    09/30/2009| Education| 110.03|   IL|
|    09/30/2009| Education| 332.58|   IL|
|    09/30/2009| Education| 145.86|   IL|
|    09/30/2009| Education|  60.08|   IL|
|    09/30/2009| Education| 284.83|   IL|
|    09/30/2009| Education| 377.15|   IL|
|    09/30/2009| Education| 114.74|   IL|
|    09/30/2009| Education|  74.46|   IL|
|    09/30/2009| Education| 129.72|   IL|
+--------------+----------+-------+-----+
only showing top 10 rows
```

One thing to note is that the pythonic solution is significantly less code. I like SQL and it's syntax, so I prefer the SQL interface over the pythonic one.

I can filter the columns selected in my query using the SQL WHERE clause

```
spark.sql(
'''

SELECT `Quarter Ending`, Department, Amount, State FROM
VermontVendor
WHERE Department = 'Education'
LIMIT 10

'''
).show()
```

```
+--------------+----------+-------+-----+
|Quarter Ending|Department| Amount|State|
+--------------+----------+-------+-----+
|    09/30/2009| Education|9423.36|   VT|
|    09/30/2009| Education| 110.03|   IL|
|    09/30/2009| Education| 332.58|   IL|
|    09/30/2009| Education| 145.86|   IL|
|    09/30/2009| Education|  60.08|   IL|
|    09/30/2009| Education| 284.83|   IL|
|    09/30/2009| Education| 377.15|   IL|
|    09/30/2009| Education| 114.74|   IL|
|    09/30/2009| Education|  74.46|   IL|
|    09/30/2009| Education| 129.72|   IL|
+--------------+----------+-------+-----+
```

A similar result can be achieved with the .filter() method in the python API.

```
df.select('Quarter Ending', 'Department', 'Amount',
'State').filter(df['Department'] == 'Education').show(10)
```

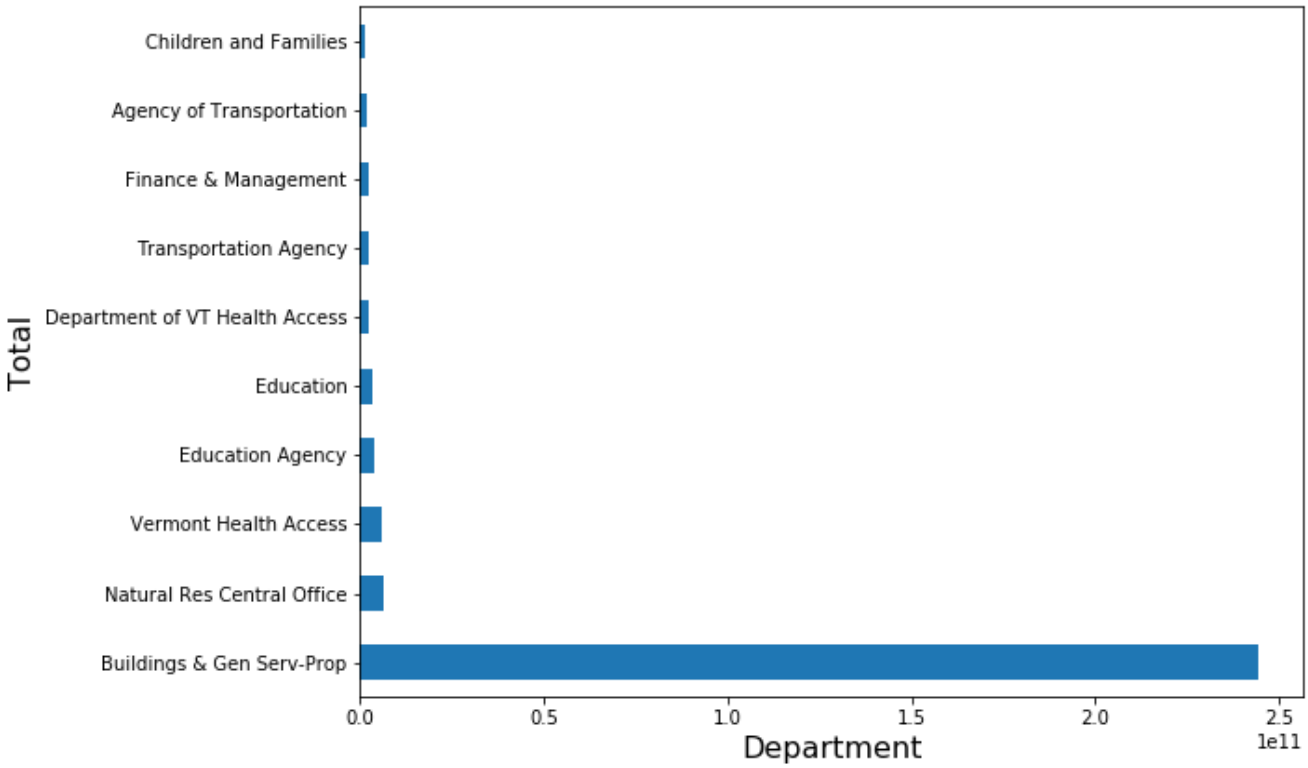
```
+--------------+----------+-------+-----+
|Quarter Ending|Department| Amount|State|
+--------------+----------+-------+-----+
|    09/30/2009| Education|9423.36|   VT|
|    09/30/2009| Education| 110.03|   IL|
|    09/30/2009| Education| 332.58|   IL|
|    09/30/2009| Education| 145.86|   IL|
|    09/30/2009| Education|  60.08|   IL|
|    09/30/2009| Education| 284.83|   IL|
|    09/30/2009| Education| 377.15|   IL|
|    09/30/2009| Education| 114.74|   IL|
|    09/30/2009| Education|  74.46|   IL|
|    09/30/2009| Education| 129.72|   IL|
+--------------+----------+-------+-----+
only showing top 10 rows
```

Plotting

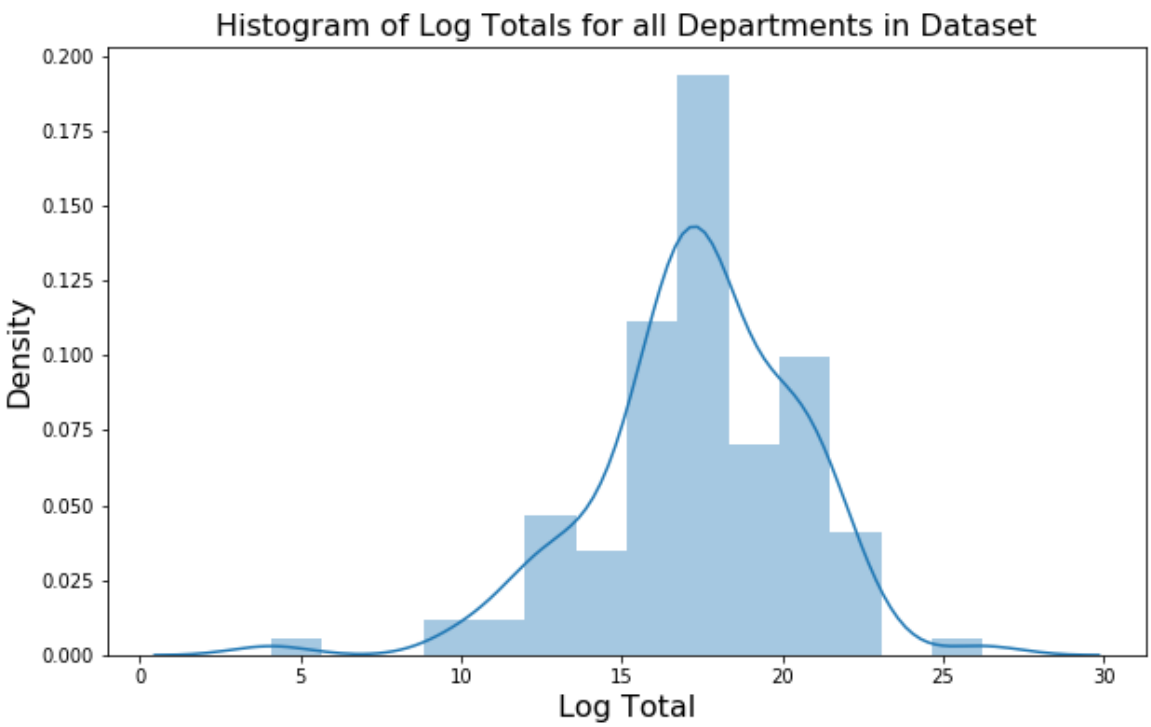
Unfortunately, one cannot directly create plots with a Spark dataframe. The simplest solution is to simply use the .toPandas() method to convert the result of Spark computations to a pandas dataframe. I give a couple examples below.

```
plot_df = spark.sql(
'''

SELECT Department, SUM(Amount) as Total FROM VermontVendor
GROUP BY Department
ORDER BY Total DESC
LIMIT 10

'''
).toPandas()
```

```
fig,ax = plt.subplots(1,1,figsize=(10,6))
plot_df.plot(x = 'Department', y = 'Total', kind = 'barh', color =
'C0', ax = ax, legend = False)
ax.set_xlabel('Department', size = 16)
ax.set_ylabel('Total', size = 16)
plt.savefig('barplot.png')
plt.show()
```



```
import numpy as np
import seaborn as sns
plot_df2 = spark.sql(
'''
SELECT Department, SUM(Amount) as Total FROM VermontVendor
GROUP BY Department
'''
).toPandas()
plt.figure(figsize = (10,6))
sns.distplot(np.log(plot_df2['Total']))
plt.title('Histogram of Log Totals for all Departments in Dataset',
size = 16)
plt.ylabel('Density', size = 16)
plt.xlabel('Log Total', size = 16)
plt.savefig('distplot.png')
plt.show()
```



Starting up you docker container again:

Once you have started and exited out of your docker container the first time, you will start it differently for future uses since the container has already been run.

Pass the following command to return all container names:

```
docker ps -a
```

Get the container id from the terminal:

```
   CREATED              STATUS                      PORTS
                  NAMES
903f152e92c5          jupyter/pyspark-notebook            "tini -g -- start-no…"
   2 hours ago           Up 2 hours                 0.0.0.0:8888->8888/tcp
                  quirky_shirley
cfdd2616074c          jupyter/minimal-notebook            "tini -g -- start-no…"
```

Then run docker start with the container id to start the container:

```
docker start 903f152e92c5
```

Your Jupyter notebook server will then again be running on http://localhost:8888.

The full code with a few more examples can be found on my github:

https://github.com/crocker456/PlayingWithPyspark

Sources:

PySpark 2.0 The size or shape of a DataFrame

Thanks for contributing an answer to Stack Overflow! Some of your past answers have not been well-received, and you're...

stackoverflow.com

Getting Started - Spark 2.4.0 Documentation

Edit description

spark.apache.org

. . .

Learn Python - Best Python Tutorials (2019) | gitconnected

The top 77 Python tutorials. Courses are submitted and voted on by developers, enabling you to find the best Python...

gitconnected.com