

Подготовка к упражнениям

Start Hadoop Cluster and Test Spark Shell

(Word Count Example)



Pull and Start Docker Container

1. Pull Docker Image:

```
docker pull marcelmittelstaedt/spark_base:latest
```

2. Start Docker Image:

```
docker network create --driver bridge bigdatanet
docker run -dit --name hadoop \
    -p 8088:8088 -p 9870:9870 -p 9864:9864 -p 10000:10000 \
    -p 8032:8032 -p 8030:8030 -p 8031:8031 -p 9000:9000 \
    -p 8888:8888 --net bigdatanet \
    marcelmittelstaedt/spark_base:latest
```

3. Wait till first Container Initialization finished:

```
docker logs hadoop

[...]  
Stopping nodemanagers  
Stopping resourcemanager  
Container Startup finished.
```

Start Hadoop Cluster

1. Get into Docker container:

```
docker exec -it hadoop bash
```

2. Switch to hadoop user:

```
sudo su hadoop
```

```
cd
```

3. Start Hadoop Cluster:

```
start-all.sh
```

Add Test text file to HDFS (Faust 1)

1. Download test Text File (Faust_1.txt):

```
wget https://raw.githubusercontent.com/marcelmittelstaedt/BigData/master/exercises/winter_semester_2022-2023/01_hadoop/sample_data/Faust_1.txt
```

2. Upload file to HDFS:

```
hadoop fs -put Faust_1.txt /user/hadoop/Faust_1.txt
```

Start Spark (on Yarn)

1. Start Spark Shell:

```
spark-shell --master yarn
```

```
Spark context Web UI available at http://localhost:4040
Spark context available as 'sc' (master = yarn, app id = application_1572177196643_0001).
Spark session available as 'spark'.
Welcome to
```



version 2.3.4

```
Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_222)
Type in expressions to have them evaluated.
Type :help for more information.
```

```
scala>
```

Start Spark – WordCount Example (Scala)

1. Execute Word Count Example in Scala:

```
scala> val text_file = sc.textFile("/user/hadoop/Faust_1.txt")
scala> val words = text_file.flatMap(line => line.split(" "))
scala> val counts = words.map(word => (word, 1))
scala> val reduced_counts = counts.reduceByKey((count1, count2) => count1 + count2)
scala> val sorted_counts = reduced_counts.sortBy(- _._2)
```

```
scala> sorted_counts.take(10)

res0: Array[(String, Int)] = Array(("",1603), (und,509), (die,463), (der,440), (ich,435), (Und,400), (nicht,346), (zu,319), (ist,291), (ein,284))
```

2. Save results to HDFS:

```
scala> sorted_counts.saveAsTextFile("/user/hadoop/Faust_1_WordCounts_Scala.txt")
```

Start Spark – WordCount Example (Scala)

3. Get results from HDFS to local filesystem:

```
hadoop fs -get /user/hadoop/Faust_1_WordCounts_Scala.txt/part-00000 Faust_1_WordCounts_Scala.txt
```


4. Check Result:

```
head -10 Faust_1_WordCounts_Scala.txt
```

```
(,1603)  
(und,509)  
(die,463)  
(der,440)  
(ich,435)  
(Und,400)  
(nicht,346)  
(zu,319)  
(ist,291)  
(ein,284)
```

Start Spark (on Yarn) – WordCount Example

5. See Spark Shell Container Running on Yarn <http://xxx.xxx.xxx.xxx:8088/cluster> :



hadoop

Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED

Scheduler

Tools

RUNNING Applications

Logged in as: dr.who

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
5	0	1	4	3	5 GB	8 GB	0 B	3	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=M), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries

Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1572177196643_0005	hadoop	Spark shell	SPARK	default	0	Sun Oct 27 14:18:28 +0100 2019	N/A	RUNNING	UNDEFINED	3	3	5120	0	0	62.5	62.5	<div></div>	ApplicationMaster	0

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Подготовка к упражнениям

Test PySpark Shell (Word Count Example)



Start PySpark (on Yarn) – Test Install

1. As PySpark is already installed, start PySpark Shell and execute previous example as Python code:

```
pyspark --master yarn
```

```
Python 3.6.8 (default, Oct 7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
```



version 2.3.4

```
Using Python version 3.6.8 (default, Oct 7 2019 12:59:55)
SparkSession available as 'spark'.
>>>
```

PySpark – WordCount Example (Python)

1. Execute Word Count Example in Python:

```
>>> text_file = spark.read.text("/user/hadoop/Faust_1.txt").rdd.map(lambda r: r[0])
>>> words = text_file.flatMap(lambda line: line.split(" "))
>>> counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)
>>> output = counts.collect()
>>> sorted_output = sorted(output,key=lambda x: (-x[1],x[0]))
```

```
>>> sorted_output[:10]

[(',', 1603), ('und', 509), ('die', 463), ('der', 440), ('ich', 435), ('Und', 400), ('nicht', 346), ('zu', 319), ('ist', 291), ('ein', 284)]
```

2. Save results to HDFS:

```
>>> counts.saveAsTextFile("/user/hadoop/Faust_1_WordCounts_Python.txt")
```

PySpark – WordCount Example (Python)

3. Get results from HDFS to local filesystem:

```
hadoop fs -get /user/hadoop/Faust_1_WordCounts_Python.txt/part-00000 Faust_1_WordCounts_Python.txt
```


4. Check Result:

```
head -10 Faust_1_WordCounts_Python.txt
```

```
('Johann', 1)  
(Wolfgang, 1)  
(von, 133)  
(Goethe:, 1)  
(Faust,, 8)  
(Der, 130)  
(Tragödie, 1)  
(erster, 2)  
(Teil, 6)  
(', 1603)
```

PySpark (on Yarn) – WordCount Example

5. See PySpark Shell Container Running on Yarn <http://xxx.xxx.xxx.xxx:8088/cluster> :



hadoop

Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler

Tools

RUNNING Applications

Logged in as: dr.who

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
8	0	1	7	3	5 GB	8 GB	0 B	3	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCoers	Allocated Memory MB	Reserved CPU VCoers	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1572177196643_0009	hadoop	PySparkShell	SPARK	default	0	Sun Oct 27 14:36:36 +0100 2019	N/A	RUNNING	UNDEFINED	3	3	5120	0	0	62.5	62.5	<div></div>	ApplicationMaster	0

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Подготовка к упражнениям

Start and work with Jupyter Notebooks (on PySpark)



Start Jupyter

1. Start Jupyter Notebook

```
jupyter notebook
```

```
[I 14:02:39.790 NotebookApp] Writing notebook server cookie secret to /home/hadoop/.local/share/jupyter/runtime/notebook_cookie_secret
[I 14:02:40.957 NotebookApp] Serving notebooks from local directory: /home/hadoop
[I 14:02:40.957 NotebookApp] The Jupyter Notebook is running at:
[I 14:02:40.957 NotebookApp] http://e0f4472dcb12:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27
[I 14:02:40.957 NotebookApp] or http://127.0.0.1:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27
[I 14:02:40.957 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 14:02:40.979 NotebookApp] No web browser found: could not locate runnable browser.
[C 14:02:40.980 NotebookApp]
```

To access the notebook, open this file in a browser:

file:///home/hadoop/.local/share/jupyter/runtime/nbserver-8624-open.html

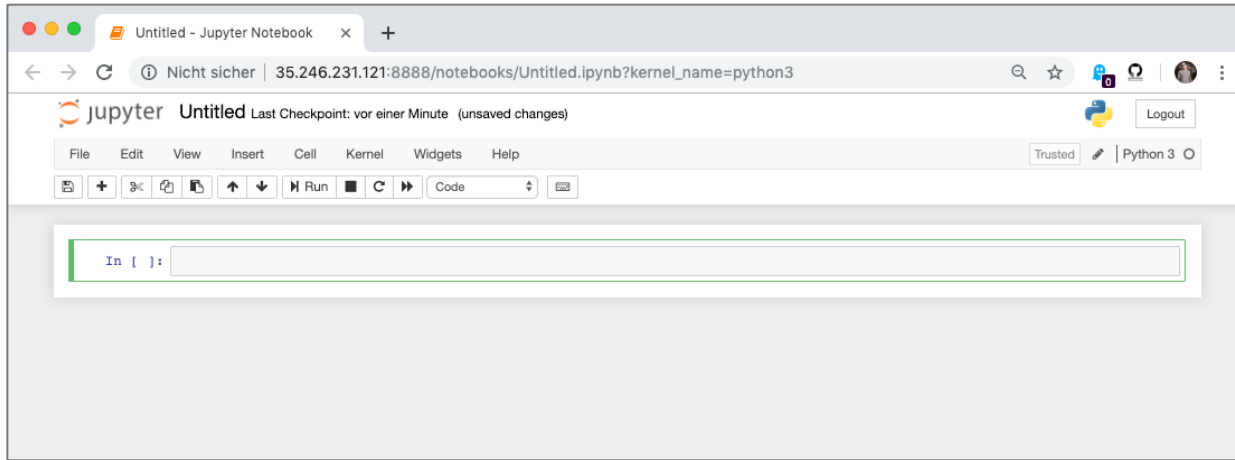
Or copy and paste one of these URLs:

http://e0f4472dcb12:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27

or http://127.0.0.1:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27

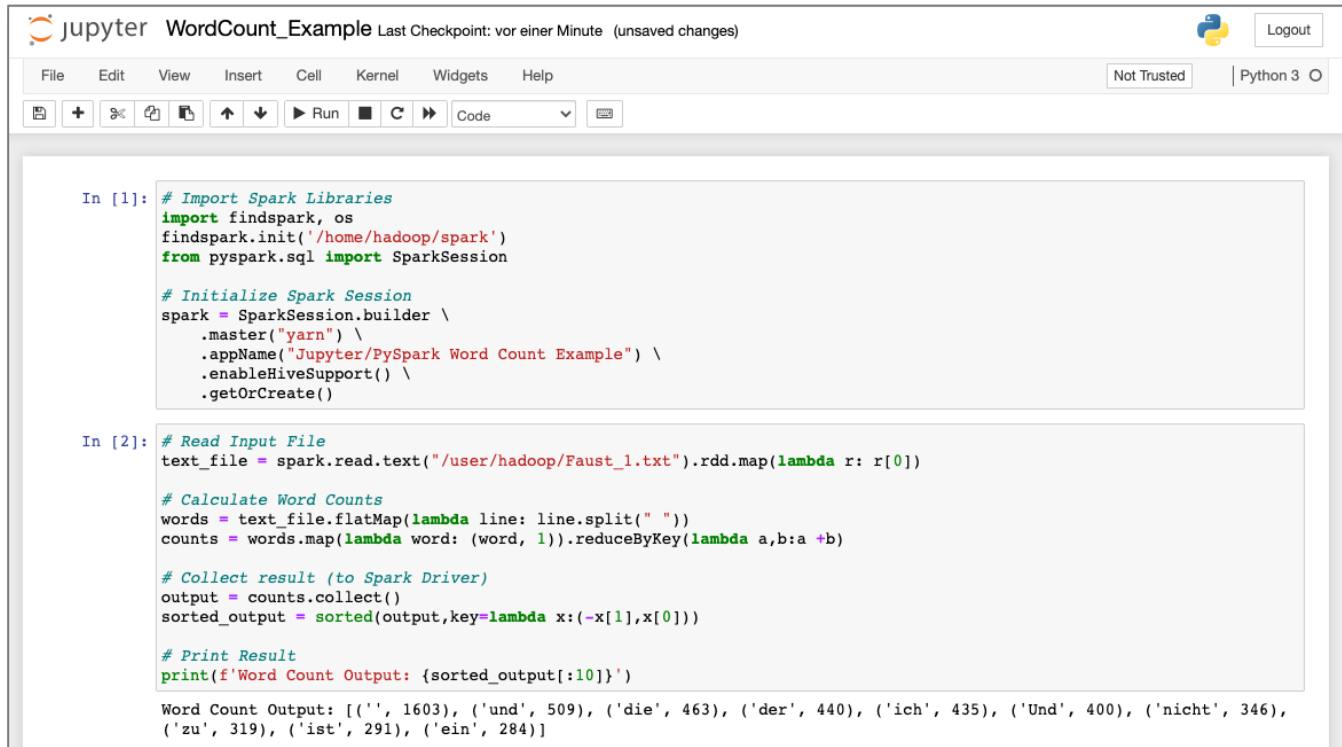
Start Jupyter

2. Open Notebook in Browser: [http://XXX.XXX.XXX.XXX:8888/?token=\[XYZXYZXYZ\]](http://XXX.XXX.XXX.XXX:8888/?token=[XYZXYZXYZ])



Use Jupyter (Word Count Example)

1. Execute previous Word Count example



```
In [1]: # Import Spark Libraries
import findspark, os
findspark.init('/home/hadoop/spark')
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder \
    .master("yarn") \
    .appName("Jupyter/PySpark Word Count Example") \
    .enableHiveSupport() \
    .getOrCreate()

In [2]: # Read Input File
text_file = spark.read.text("/user/hadoop/Faust_1.txt").rdd.map(lambda r: r[0])

# Calculate Word Counts
words = text_file.flatMap(lambda line: line.split(" "))
counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)


# Collect result (to Spark Driver)
output = counts.collect()
sorted_output = sorted(output,key=lambda x:(-x[1],x[0]))

# Print Result
print(f'Word Count Output: {sorted_output[:10]}')

Word Count Output: [(',', 1603), ('und', 509), ('die', 463), ('der', 440), ('ich', 435), ('Und', 400), ('nicht', 346), ('zu', 319), ('ist', 291), ('ein', 284)]
```

Use Jupyter (Word Count Example)

2. See **Jupyter** PySpark Container Running on Yarn <http://xxx.xxx.xxx.xxx:8088/cluster> :



All Applications

Logged in as: dr.who

Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
1	0	1	0	4	7 GB	16 GB	0 B	4	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1614532561077_0001	hadoop	Jupyter/PySpark Word Count Example	SPARK	default	0	Sun Feb 28 18:16:38 +0100 2021	N/A	RUNNING	UNDEFINED	4	4	7168	0	0	43.8	43.8	<div></div>	ApplicationMaster	0

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Get some data...

1. Get some IMDb data:

```
wget https://datasets.imdbws.com/title.basics.tsv.gz && gunzip title.basics.tsv.gz  
wget https://datasets.imdbws.com/title.ratings.tsv.gz && gunzip title.ratings.tsv.gz
```

2. Put them into HDFS:

```
hadoop fs -mkdir /user/hadoop/imdb
```

```
hadoop fs -mkdir /user/hadoop/imdb/title_basics  
hadoop fs -mkdir /user/hadoop/imdb/title_ratings
```


```
hadoop fs -put title.basics.tsv /user/hadoop/imdb/title_basics/title.basics.tsv  
hadoop fs -put title.ratings.tsv /user/hadoop/imdb/title_ratings/title.ratings.tsv
```

PySpark Operations

1. Initialize Spark Session:

```
In [1]: # Import Spark Libraries
import findspark, os
findspark.init('/home/hadoop/spark')
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder \
    .master("yarn") \
    .appName("Jupyter/PySpark Exercises") \
    .enableHiveSupport() \
    .getOrCreate()
```



hadoop

Cluster

- About Nodes
- Node Labels
- Applications
- NEW
- SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler

Tools

RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	V-Cores Used	V-Cores Total	V-Cores Reserved
3	0	1	2	3	5 GB	16 GB	0 B	3	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=M), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 ▾ entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU V-Cores	Allocated Memory MB	Reserved CPU V-Cores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1614532561077_0003	hadoop	Jupyter/PySpark Exercises	SPARK	default	0	Sun Feb 28 19:41:41 +0100 2021	N/A	RUNNING	UNDEFINED	3	3	5120	0	0	31.3	31.3	<div></div>	ApplicationMaster	0

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

PySpark Operations

2. Basic PySpark operations: Read Files from HDFS into DataFrames:

```
In [2]: # Read IMDb title basics CSV file from HDFS
df_title_basics = spark.read \
    .format('csv') \
    .options(header='true', delimiter='\t', nullValue='null', inferSchema='true') \
    .load('/user/hadoop/imdb/title_basics/title.basics.tsv')
```

```
In [3]: # Print Schema of DataFrame
df_title_basics.printSchema()

root
 |-- tconst: string (nullable = true)
 |-- titleType: string (nullable = true)
 |-- primaryTitle: string (nullable = true)
 |-- originalTitle: string (nullable = true)
 |-- isAdult: string (nullable = true)
 |-- startYear: string (nullable = true)
 |-- endYear: string (nullable = true)
 |-- runtimeMinutes: string (nullable = true)
 |-- genres: string (nullable = true)
```

```
In [4]: # Print First 3 Rows of DataFrame Data
df_title_basics.show(3)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| tconst|titleType|primaryTitle|originalTitle|isAdult|startYear|endYear|runtimeMinutes|genres|
+-----+-----+-----+-----+-----+-----+-----+-----+
|tt0000001|short|Carmencita|Carmencita|0|1894|\N|1|Documentar
y,Short|
|tt0000002|short|Le clown et ses c...|Le clown et ses c...|0|1892|\N|5|Animatio
n,Short|
|tt0000003|short|Pauvre Pierrot|Pauvre Pierrot|0|1892|\N|4|Animation,Com
edy,...|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 3 rows
```

PySpark Operations

3. Basic PySpark: Operations on DataFrames (aggregations...):

```
In [5]: # Get Number of Rows of a DataFrame
df_title_basics.count()
```

```
Out[5]: 7656314
```

```
In [6]: # Groups and Counts: Get column titleTypes values with counts and ordered descending
from pyspark.sql.functions import desc
df_title_basics \
    .groupBy("titleType") \
    .count() \
    .orderBy(desc("count")) \
    .show()
```

titleType	count
tvEpisode	5556805
short	796470
movie	569437
video	296405
tvSeries	202321
tvMovie	130185
tvMiniSeries	36080
tvSpecial	31590
videoGame	27416
tvShort	9602
audiobook	1
episode	1
radioSeries	1

```
In [7]: # Calculate average Movie length in minutes
from pyspark.sql.functions import avg, col
df_title_basics \
    .where(col('titleType') == 'movie') \
    .agg(avg('runtimeMinutes')) \
    .show()
```

avg(runtimeMinutes)
89.62651045204976

PySpark Operations

4. Basic PySpark: Save PySpark DataFrame back to HDFS (as partitioned parquet files):

```
In [8]: # Save Dataframe back to HDFS (partitioned) as Parquet files
df_title_basics.repartition('startYear').write \
    .format("parquet") \
    .mode("overwrite") \
    .partitionBy('startYear') \
    .save('/user/hadoop/imdb/title_basics_partitioned_files')
```

/user/hadoop/imdb/title_basics_partitioned_files

Show entries

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:53	0	0 B	startYear=2005	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:53	0	0 B	startYear=2006	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:53	0	0 B	startYear=2007	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:53	0	0 B	startYear=2008	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:53	0	0 B	startYear=2009	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:53	0	0 B	startYear=2010	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:53	0	0 B	startYear=2011	

PySpark Operations



5. Basic PySpark: Save PySpark DataFrame back to HDFS (as table and partitioned parquet files):

```
In [9]: # Save Dataframe back to HDFS (partitioned) as EXTERNAL TABLE and Parquet files
df_title_basics.repartition('startYear').write \
    .format("parquet") \
    .mode("overwrite") \
    .option('path', '/user/hadoop/imdb/title_basics_partitioned_table') \
    .partitionBy('startYear') \
    .saveAsTable('default.title_basics_partitioned')
```

Spark
Table

/user/hadoop/imdb/title_basics_partitioned_table

Show 25 entries

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:54	0	0 B	startYear=2005	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:54	0	0 B	startYear=2006	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:54	0	0 B	startYear=2007	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:54	0	0 B	startYear=2008	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:54	0	0 B	startYear=2009	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:54	0	0 B	startYear=2010	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Feb 28 19:54	0	0 B	startYear=2011	

```
In [10]: spark.sql('SHOW TABLES').show(10, False)
```

database	tableName	isTemporary
default	title_basics_partitioned	false

PySpark Operations

6. Basic PySpark: Interact with Spark Tables (using plain Spark SQL):

```
In [12]: # Read External Spark table using plain Spark SQL
df = spark.sql('SELECT tconst, primaryTitle, startYear FROM default.title_basics_partitioned WHERE startYear = 2020')
# Print Result
df.show(3)
```

tconst	primaryTitle	startYear
tt0060366	A Embalagem de Vidro	2020
tt0062336	El Tango del Viud...	2020
tt0065392	Bucharest Memories	2020

only showing top 3 rows

7. Basic PySpark: Interact with Spark Tables (using Spark programmatically):

```
In [11]: # Read External Spark table in programmatical way
df = spark.table('default.title_basics_partitioned') \
    .where(col('startYear') == '2020') \
    .select('tconst', 'primaryTitle', 'startYear')
# Print Result
df.show(3)
```

tconst	primaryTitle	startYear
tt0060366	A Embalagem de Vidro	2020
tt0062336	El Tango del Viud...	2020
tt0065392	Bucharest Memories	2020

only showing top 3 rows

Same Result!

PySpark Operations

8. PySpark SQL: Join Spark DataFrames:

```
In [13]: # Read title_ratings.tsv into Spark dataframe
df_title_ratings = spark.read \
    .format('csv') \
    .options(header='true', delimiter='\\t', nullValue='null', inferSchema='true') \
    .load('/user/hadoop/imdb/title_ratings/title_ratings.tsv')
```

```
In [14]: # Print Schema of title_ratings dataframe
df_title_ratings.printSchema()
```

```
root
 |-- tconst: string (nullable = true)
 |-- averageRating: double (nullable = true)
 |-- numVotes: integer (nullable = true)
```

```
In [15]: # Show first 3 rows of title_ratings dataframe
df_title_ratings.show(3)
```

tconst	averageRating	numVotes
tt0000001	5.7	1685
tt0000002	6.0	208
tt0000003	6.5	1425

only showing top 3 rows

```
In [16]: # JOIN Data Frames
joined_df = df_title_basics.join(df_title_ratings, df_title_basics.tconst == df_title_ratings.tconst)
```

```
In [17]: # Print Schema of joined DataFrame
joined_df.printSchema()
```

```
root
 |-- tconst: string (nullable = true)
 |-- titleType: string (nullable = true)
 |-- primaryTitle: string (nullable = true)
 |-- originalTitle: string (nullable = true)
 |-- isAdult: string (nullable = true)
 |-- startYear: string (nullable = true)
 |-- endYear: string (nullable = true)
 |-- runtimeMinutes: string (nullable = true)
 |-- genres: string (nullable = true)
 |-- tconst: string (nullable = true)
 |-- averageRating: double (nullable = true)
 |-- numVotes: integer (nullable = true)
```

```
In [18]: # Show First 3 Rows of Joined DataFrame
joined_df.show(3)
```

tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
tt0000001	short	The Puppet's Nigh...	Le cauchemar de F...	0	1908		2	Animation, Sho
tt0000002	short	The Lighthouse Ke...	The Lighthouse Ke...	0	1911			Drama, Sho
tt0000003	short	Home Folks	Home Folks	0	1912		17	Drama, Sho

only showing top 3 rows

PySpark Operations

9. Basic PySpark: Filtering, Ordering and Selecting: Get Top 5 TV Series

```
In [19]: top_tvseries = joined_df \
        .where(col('titleType') == 'tvSeries') \
        .where(col('numVotes') > 200000) \
        .orderBy(desc('averageRating')) \
        .select('originalTitle', 'startYear', 'endYear', 'averageRating', 'numVotes')
```

```
# Print Top 5 TV Series
```

```
top_tvseries.show(5)
```

originalTitle	startYear	endYear	averageRating	numVotes
Breaking Bad	2008	2013	9.5	1470997
The Wire	2002	2008	9.3	286432
Game of Thrones	2011	2019	9.3	1775327
Rick and Morty	2013	\N	9.2	377428
Avatar: The Last ...	2005	2008	9.2	249236

```
only showing top 5 rows
```

PySpark Operations

10. Basic PySpark: Add/Calculate Columns:

```
In [20]: from pyspark.sql.functions import when, lit

# Add a calculated column: classify movies as being either 'good' or 'worse' based on average rating
df_with_classification = joined_df \
    .withColumn('classification',
                when(col('averageRating') > 8, lit('good')) \
                  .otherwise(lit('worse')))) \
    .select('primaryTitle', 'startYear', 'averageRating', 'classification')

# Print Result
df_with_classification.show(3, False)
```

primaryTitle	startYear	averageRating	classification
The Puppet's Nightmare	1908	6.4	worse
The Lighthouse Keeper	1911	7.1	worse
Home Folks	1912	3.7	worse

only showing top 3 rows

PySpark Operations

11. Basic PySpark/Python: Plot Data:

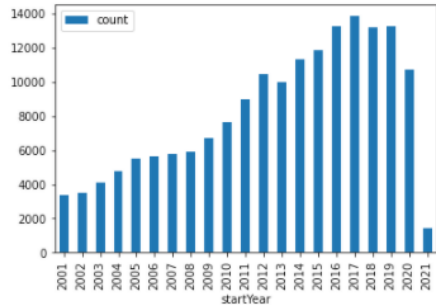
```
In [21]: # Plot data: good movies per year
import matplotlib.pyplot as plt
import pandas

# Create DataFrame to be plotted
good_movies = df_with_classification \
    .select('startYear', 'classification') \
    .where(col('classification') == 'good') \
    .where(col('startYear') > 2000) \
    .groupBy('startYear') \
    .count() \
    .sort(col('startYear').asc())

# Convert Spark DataFrame to Pandas DataFrame
pandas_df = good_movies.toPandas()

# Plot DataFrame
pandas_df.plot.bar(x='startYear', y='count')
```

Out[21]: <AxesSubplot: xlabel='startYear'>



Упражнения

Use PySpark Shell or Jupyter Notebooks on
PySpark to solve exercises



PySpark Exercises - IMDB

1. Execute Tasks of previous HandsOn Slides
2. Create External Spark Table `title_ratings` on HDFS containing data of IMDB file *title_ratings.tsv*
3. Create External Spark Table `name_basics` on HDFS containing data of IMDB file *name_basics.tsv*
4. Use PySpark to answer following questions:
 - a) How many **movies** and how many **TV series** are within the IMDB dataset?
 - b) Who is the **youngest** actor/writer/... within the dataset?

PySpark Exercises - IMDB

4. Use PySpark to answer following questions:

c) Create a list (`tconst`, `original_title`, `start_year`, `average_rating`, `num_votes`) of movies which are:

- equal or newer than year 2010
- have an average rating equal or better than 8,1
- have been voted more than 100.000 times

d) Save result of c) as external Spark Table to HDFS.?

5. Create a Spark Table `name_basics_partitioned`, which:

- contains all columns of table `name_basics`
- is partitioned by column `partition_is_alive`, containing:
 - „*alive*“ in case actor is still alive
 - „*dead*“ in case actor is already dead

PySpark Exercises - IMDB

6. Create a partitioned Spark table `imdb_movies_and_ratings_partitioned`, which:
- contains all columns of the two tables `title_basics_partitioned` and `title_ratings` and
 - is partitioned by start year of movie (create and add column `partition_year`).
7. Create following plot, which visualizes:
- the amount of movies (type!)
 - per year
 - since 2000

