Spark Exercise

# 

[**You can find solutions in the github repository**](#_may9mjxsae05) **2**

[**Exercise 1: Launch Single Node Spark Cluster**](#_p3ro03d1hf6q) **3**

[**Exercise 2: Analyze movielens dataset using RDD**](#_k110hkvylv1y) **3**

[**Exercise 3: Analyze Stock Data using RDD operations**](#_hkmfaw69g7hw) **4**

[**Exercise 4: Understanding Spark Internals**](#_w5au3e9guqwf) **5**

[**Exercise 5 - Movielens dataset analysis using DataFrame**](#_xa7xgcfc5lcj) **6**

[**Exercise 6: Stocks analysis using DataFrame**](#_8v7elu6oynst) **6**

[**Exercise 7: Read data from mysql using DataFrame reader**](#_9du9xdczsnei) **8**

[**Exercise 8: Use SFPD data analysis using DataFrame**](#_xhgxr1oj7veq) **10**

[**Exercise 9: Hive tables and Table partitions**](#_r3f3uieli21c) **10**

[**Exercise 10: Convert RDD to Dataframe**](#_d6j5ffx2gnqr) **14**

[**Exercise 11: Convert CSV into compressed CSV, Avro, Parquet using DataFrame**](#_e6e6tae85g0j) **14**

[**Check the size of the file/directory**](#_h0dv2icrucb3) **14**

[**Exercise 12: Size of DataFrame in cache**](#_pu5lz6bmjp60) **14**

[**Exercise 13: Find word distance with a string**](#_xjj9hcryj22m) **15**

[**Exercise 14: Airlines data analysis (DataFrame)**](#_saguo7iuof72) **16**

[**Exercise 15: Submit Spark Application**](#_b4pvdjbzcoie) **19**

[**Exercise 16: Structured Streaming (Kafka)**](#_d8vryr68gko1) **21**

# You can find solutions in the github repository

<https://github.com/abulbasar/pyspark-examples>

<https://github.com/abulbasar/pyspark-examples/blob/master/Dataframe_examples.ipynb>

# 

# **Exercise 1: Launch Single Node Spark Cluster**

<https://blog.einext.com/apache-spark/setup-spark-cluster>

# **Exercise 2: Analyze movielens dataset using RDD**

Dataset: <https://drive.google.com/file/d/1wRPuq9CJisdneG2uUkziV8TU-uEi0V-W/view?usp=share_link> (Download the file manually using the browser and unzip the content)

1. Data files are in CSV format. Some values contain commas in them. Such values are quoted with double quotes. If you are using Java/scana use the Univocity/Open CSV library for parsing the CSV and if you are using Python, use the built-in csv module. Create a class for Movie and Rating with relevant fields. You can find the field names in the file header. Excluding the headers convert each line movies.csv to Movie class and each line in ratings.csv file Rating class object. How many records are there in movies and ratings [movies: 9125, ratings: 100004]
2. Find the movies, for which title contains Godfather?
3. How many ratings the movie "Godfather, The (1972)" received?[Answer: 200]
4. Using ratings dataset, calculate average rating and number of ratings for each movie. What is the average rating for "Godfather, The (1972)" [Answer: 4.487, 200]
5. Select all movie Ids that have received more than 100 ratings. How many movies are there having more than 100 ratings? [Answer: 149]
6. From the output of step 4 and 5, take top 10 records with the highest avg rating. Output columns: movieId, title, average rating, rating count.
7. Merge the output of 6 with movies dataframe to show title
8. Find how many movies are there for each genre. What is the frequency for the Adventure genre?
9. For each genre find top 3 movies [Hint: use flatMap function]
10. Most of the movie titles contain year of release in bracket, for example: Godfather, The (1972) was released in 1972. Extract the year from the movie titles and find which year most number of movies were released. You can use regular expressions for extraction of year from title.
11. What is the min, max, and avg length of the movie title
12. Which words are most common in movie titles

Pyspark solution

<https://github.com/abulbasar/pyspark-examples/blob/master/RDD%2BBasics.ipynb>

# 

# **Exercise 3: Analyze Stock Data using RDD operations**

Dataset: <https://drive.google.com/open?id=1Bp_jnJ7ZVwvisaJkt6vJLm3Ow82V3WOO>

Data looks like

date,open,high,low,close,volume,adjclose,symbol

2000-07-17,95.4375,97.5,92.75,96.625,3508100.0,74.269199,XLNX

2000-07-17,22.625,22.75,22.4375,22.5625,201600.0,13.48614,ES

2000-07-17,6.750002,6.937503,6.375,6.5,1235700.0,5.241649,CHK

2000-07-17,19.812501,20.1875,19.500001,20.1875,1434100.0,3.806147,NI

2000-07-17,30.5,30.6875,30.0,30.03125,254600.0,19.81183,SNA

2000-07-17,44.749996,45.062498,44.500004,45.000009,535200.0,17.400773,FOXA

2000-07-17,19.625,19.625,19.25,19.375,309500.0,13.768835,R

2000-07-17,16.6562,16.6875,16.125,16.25,5507200.0,1.755466,ROST

2000-07-17,56.25,57.25,56.0625,56.125,7941200.0,18.31076,PG

1. Load the stocks.csv file in a new Rdd and convert each record to a Stock class object. Exclude the header. How many records are there in the output rdd? [Note: If you are using pyspark consider using collections.namedtuple or dict instead of class. If you want to use class, save the class definition in a file called models.py and pass this file while launching spark session. You can use spark.submit.pyFiles property to pass the file] [Answer: 1857092]
2. How many records are there in 2016? [Answer: 78041]
3. How many records are there in 2016 and for stock INTC [Answer: 156]
4. What percentage of days INTC stocks ended the day in positive gain in 2016 (Note: positive gain means - the close price is higher than the open price). Show the percentage value up to 2 decimal points. [Answer: 61.54% ]
5. Find top 3 records based with highest trading volume in a day in 2016. Find the stock name and the trading volume.

[Answer:

BAC -> 375088700

PFE -> 284468100

MRO -> 273996600

]

1. Find average daily return [daily return = (close - open)/open] and number of records per stock symbol traded in 2016. What is the average daily return and count for stock INTC? [Hint: use groupByKey or reduceByKey function on pairRdd]

[Answer:

Top 3

StockStats(symbol=SWN, count=156, avgReturn=0.774805421857114)

StockStats(symbol=OKE, count=156, avgReturn=0.5241872213902886)

StockStats(symbol=FCX, count=156, avgReturn=0.47976417772228247)

INTC

StockStats(symbol=INTC, count=156, avgReturn=0.13338813423580165)

]

1. In 2016, which stock has given the highest return based on adjclose. [Hint: find the first and the last value for adjclose in 2016 and compute the percent change].

[Answer:

NEM -> 144.00

NVDA -> 95.78

OKE -> 95.35

]

# **Exercise 4: Understanding Spark Internals**

Download the following stocks.csv.zip file. Unzip it locally. Create a copy of the file using gzip compress one of them, so you should have two files - stocks.csv and stocks.csv.gz.

Dataset: <https://drive.google.com/open?id=1Bp_jnJ7ZVwvisaJkt6vJLm3Ow82V3WOO>

1. Print the url for the spark web UI. Which information is not available in Spark Web UI - A. number of executors, B. executor and driver memory C. Jobs in the current session D. jobs in the previous spark sessions E. Cached RDDs?
2. Which of these two RDD operations is a wide operation - map and groupByKey? How can you determine?
3. Set the number of partitions to 5 while creating RDD using sc.textFile function. What is the number of partitions for the base RDD if you use stocks.csv and what is number of partitions if you use stocks.csv.gz as input?[Answer: 5, 1]
4. Reduce the number partitions = 1 using the coalesce function of RDD. Has the coalesce function increased the number of stages?
5. Increase number of partitions = 10 using repartition function. Has the repartition function increased the number of stages?
6. For each above case, run an action (.count function) to verify that the number of tasks equals the number of partitions. Use spark web UI to inspect the jobs, stages and tasks. What is total input read size, median task durations?
7. Calculate sum, min and max of total volume by each symbol. Compare the size of shuffle data for groupByKey function and reduceByKey functions.
8. Set caching level to MEMORY\_ONLY and MEMORY\_ONLY\_SER to compare differences in size of cache in memory. Which caching level requires less data?

# 

# **Exercise 5 - Movielens dataset analysis using DataFrame**

Dataset: <https://drive.google.com/file/d/1wRPuq9CJisdneG2uUkziV8TU-uEi0V-W/view?usp=share_link>

1. Load movies.csv as movies dataframe. Cache the dataframe
2. Load ratings.csv as ratings dataframe. Cache the dataframe
3. Show temporary views for current Spark session
4. Register movies dataframe and ratings dataframe as movies and ratings temporary view respectively. Verify that you can see the new temporary views you just created. These view definitions are required if you want to perform the analysis using SQL commands.
5. Find the number of records in movies dataframe
6. Find the number of records in ratings dataframe
7. Validate the userId and movieId combination is unique in ratings dataframe
8. Find the movieId that contains "Godfather" in the title
9. Find the number of ratings and avg rating by the movie "Godfather, The (1972)"
10. Find average rating and count of rating per movieId using ratings dataframe
11. Find top 10 movies based on the highest average ratings. Consider the movies that have at least 100 ratings. Show movieId, title, average rating and rating count columns.
12. Find top 3 movies based on the highest average ratings for each genre. Consider only those movies that have at least 100 ratings. Show movieId, title, average rating and rating count columns.

[Hint: WindowSpec windowSpec = Window

.*partitionBy*("genre")

.orderBy(*desc*("rating\_avg"));

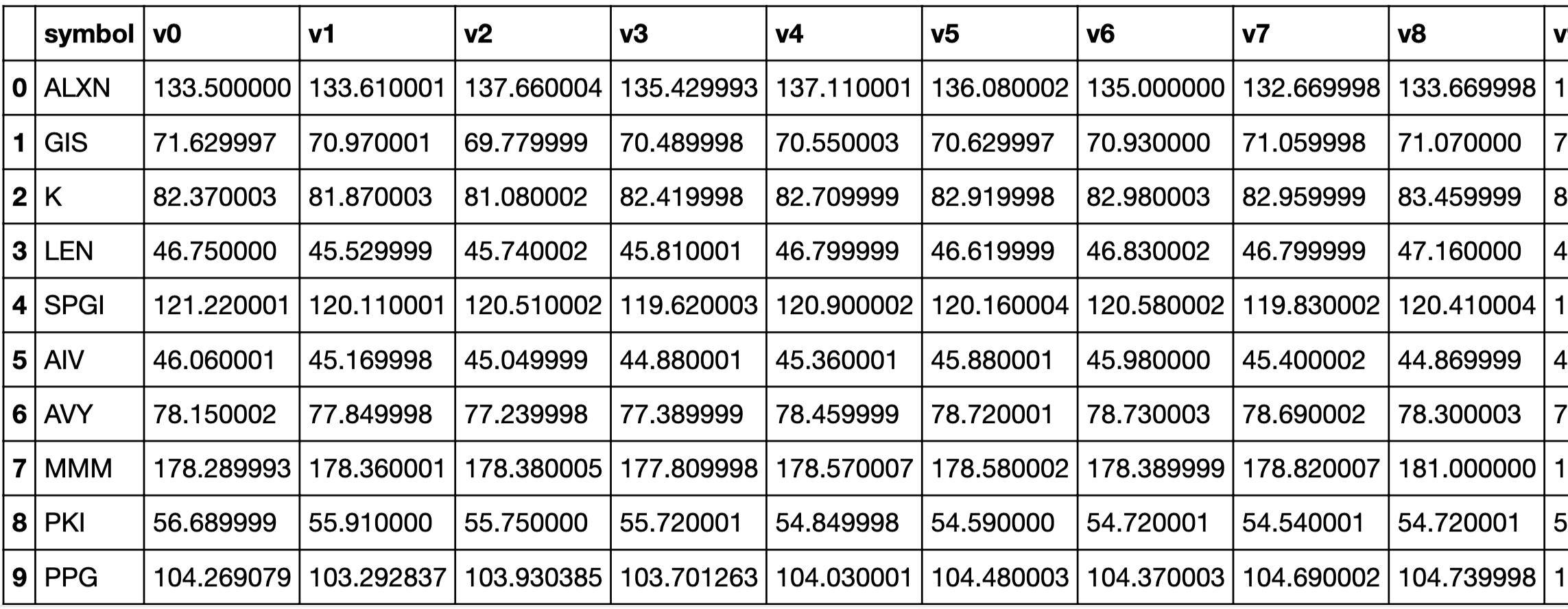
]

# **Exercise 6: Stocks analysis using DataFrame**

Dataset: <https://drive.google.com/open?id=1Bp_jnJ7ZVwvisaJkt6vJLm3Ow82V3WOO>

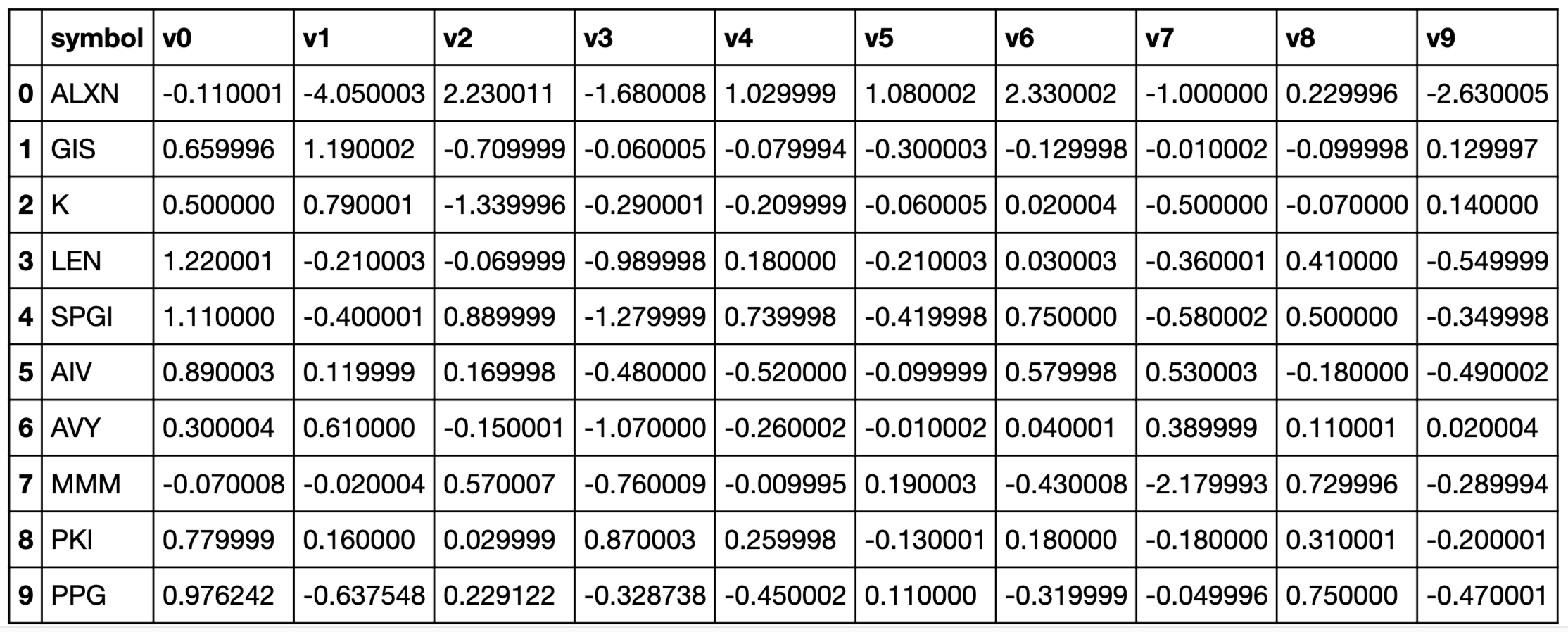
Load stocks.csv as stocks dataframe with scheme inferencing enabled. Cache the dataframe.

1. How many rows and columns are present?
2. Find data type for each column. How many columns are numeric?
3. Cast the date field of stocks dataframe as date type. What is the minimum and maximum value (range) in the date column.
4. What avg return for APPLE stocks? What percent of days AAPL stock ended in positive daily return in 2016.
5. Create a new dataframe (stocks\_last10, output columns: date, symbol and adj close) 10 consecutive lagged adj close prices for each stock. [Hint: use window partition by clause]
6. Create a new dataframe stocks\_pivot, by pivoting the stocks\_last10 dataframe. For each stock, we want to find the previous 10 days' actual values.



[The number are calculated based on end date 2016-08-15]

1. Find percentage changes between two consecutives for each stock on the above output so that row-wise values are comparable.



[The number are calculated based on date 2016-08-15]

1. Find percent change in adj close between two consecutive days for each stock. In 2016, what is the correlation coefficient between the daily changes in price between AAPL and INTC stocks. [Hint: use df.stat.corr]

Hint:

w = Window.partitionBy('location\_point').orderBy('timestamp').rangeBetween(-60\*5,0)

df = df.withColumn('occurrences\_in\_5\_min',F.count('timestamp').over(w))

df.show()

# **Exercise 7: Read data from mysql using DataFrame reader**

**High Level Steps:**

1. Create a user in mysql
2. Test the mysql access under the new user
3. Download a sample mysql database (e.g. retail\_db). Import the database to mysql.
4. Query mysql from Spark SQL
5. Login to mysql with root privileges on demo1 and create a user for spark. In the training cluster mysql is installed on demo1.

| $ ssh ubuntun@sa01  $ sudo mysql  mysql> SELECT user, host FROM mysql.user; CREATE USER 'spark'@'%' IDENTIFIED BY 'spark'; REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'spark'@'%'; GRANT ALL PRIVILEGES ON \*.\* TO 'spark'@'%'; FLUSH PRIVILEGES; |
| --- |

1. Test the mysql connection using spark user. Run the command in the demo1 machine. It should take you to the mysql prompt. If you can login to mysql and see the existing databases, you are good to proceed.

| $ mysql -h sa01 -uspark -pspark  mysql>  select user();  show databases; |
| --- |

1. Sample Mysql Database

Download and unzip the retail\_db.zip file. It contains a .sql file dump containing schema and data. Copy this file to demo1 where mysql is running. Use scp command to copy the file to demo1.

Retail db <https://drive.google.com/file/d/1rY4usOm3g7Pb824rmG0SacIH5q7UGIYk/view?usp=sharing>

Start the import

| $ ls -l retail\_db.sql  $ mysql -h demo1 -u spark -pspark -ne "DROP DATABASE IF EXISTS retail\_db" $ mysql -h demo1 -u spark -pspark -ne "CREATE DATABASE IF NOT EXISTS retail\_db"  $ mysql -h demo1 -u spark -pspark retail\_db < retail\_db.sql |
| --- |

1. Use the following code snippet to query a table in mysql from pyspark. Launch pyspark in Jupyter or in a terminal application and execute the code.

| host = "sa01"  database = "retail\_db"  db\_type = "mysql"  driver = "com.mysql.cj.jdbc.Driver"  table\_name = "customers"  user = "spark"  password = "spark"  customers = (spark .read.format("jdbc") .option("url", f"jdbc:{db\_type}://{host}/{database}") .option("driver", driver) .option("dbtable", table\_name) .option("user", user) .option("password", password) .load())  customers.show() |
| --- |

Host: localhost:3306

Username: <password>

Password: <password>

1. Create a dataframe - orders in Spark based on the orders table in retail\_db in mysql
2. Save the orders as parquet file in HDFS
3. Write a query joining customers table in mysql and "orders" parquet file in HDFS to find customers with the most number of completed orders.
4. Save the orders dataframe as a hive table. Verify that the orders table is accessible in hive as well.
5. Delete the orders table from hive.

**Maven dependency for mysql JDBC connector**

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

</dependency>

# **Exercise 8: Use SFPD data analysis using DataFrame**

Filename: [Crime\_Incidents.csv.gz](https://drive.google.com/open?id=1VYfoDBYcwcD3HLXqTtcGbjInmNsEeDiF) (Download the file from this link using browser)

1. Create a dataframe with crime incident data and show the first 10 values of a dataframe
2. Check number of partitions of the dataframe [Hint: df.rdd().getNumPartitions()] [Answer: 8]
3. Find the number of incident records in the dataframe [Answer: 1888567]
4. Find the distinct count of categories of incidents [Answer: 39]
5. What are the number of incidents in each category? What are the count for ASSAULT and VANDALISM [Answer: 165324 (8.75%), 96350 (5.1.0%)]
6. What is the minimum incident date and maximum incident date?[Answer: 01/01/2003, 12/31/2015]
7. Is there a duplicate in the Incident Number? [Answer: yes]
8. How many null values are there per column? [Answer: PdDistrict=1, other have 0 null values]
9. Create a UDF to parse the date field to convert it into a timestamp type field. Find out on which day each of the incidents have occurred most. [Anwer: FRAUD -> Fri, SUICIDE -> Mon etc.]
10. [Optional] Plot the frequency for each category of events (Skip this one)
11. Which district is the most prone to crime than the other? [Anser: SOUTHERN (339071, ~18%)]

SQL functions in spark <https://spark.apache.org/docs/latest/api/sql/index.html>

Hint: to parse date string using built-in UDF

to\_timestamp(substr(Date, 0, 10), 'MM/dd/yyyy')

# 

# **Exercise 9: Hive tables and Table partitions**

Inside the cloudera sandbox, open a terminal and launch hive

| $ hive |
| --- |

For more hive query examples view this link <https://blog.einext.com/hadoop/hive-join-example>

Check existing tables in hive

| hive> show tables; |
| --- |

In hive terminal, create a table for movies

| hive> CREATE EXTERNAL TABLE movies(  movieId string,  title string,  genres string) ROW FORMAT SERDE  'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES (  'separatorChar' = ',',  'quoteChar' = '"',  'escapeChar' = '\\') STORED AS INPUTFORMAT  'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat' LOCATION  '/user/cloudera/movielens/movies' TBLPROPERTIES (  'serialization.null.format' = '',  'skip.header.line.count' = '1'); |
| --- |

Upload the movies.csv to the /user/cloudera/movielens/movies

Now, query the movies table in hive

| hive> select \* from movies limit 10; |
| --- |

Before launching pyspark, link the hive-site.xml inside spark conf directory.

| $ sudo ln -s /etc/hive/conf/hive-site.xml $SPARK\_HOME/conf |
| --- |

Launch pyspark in another terminal

| $ cd $SPARK\_HOME $ bin/pyspark |
| --- |

Check the current database and set current database

| pyspark> spark.catalog.getCurrentDatabase(); pyspark> spark.catalog.setCurrentDatabase("default"); |
| --- |

Check the available tables in Spark session

| pyspark> sql("show tables").show() |
| --- |

Back in the pyspark session,

| pyspark> sql("desc formatted movies").show() pyspark> sql("select \* from movies").show() |
| --- |

Next, create a table in a hive from spark.

Define a dataframe

| pyspark> df = (spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/user/cloudera/movielens/ratings.csv")) |
| --- |

Save the table as a hive table. This will create a managed table in hive. The data is stored in snappy compressed Parquet format. Optionally, we can partition the table and set bucket and sort field.

| pyspark> df.write.saveAsTable("ratings") |
| --- |

In hive terminal, you can check the table

| hive> show tables hive> describe formatted ratings hive> select \* from ratings limit 10; |
| --- |

Create a table in hive using sql statement

| pyspark> sql("""CREATE EXTERNAL TABLE ratings(  userId string,  movieId string,  rating double,  timestamp bigint) ROW FORMAT SERDE  'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES (  'separatorChar' = ',',  'quoteChar' = '"',  'escapeChar' = '\\') STORED AS INPUTFORMAT  'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat' LOCATION  '/user/cloudera/movielens/ratings' TBLPROPERTIES (  'serialization.null.format' = '',  'skip.header.line.count' = '1'); """) |
| --- |

This exercise is to explore data partitioning for files. Data partitions are different from RDD partitions. For this exercise use weblogs dataset

1. Create a hive table - weblogs using weblogs dataset. Follow the steps mentioned in this doc. <http://blog.einext.com/hadoop/hive-table-using-regex-serde>
2. Create a dataframe in Spark that refers to Hive table weblogs.
3. Find the total number of rows.
4. Parse the time column as date time
5. Save the weblogs data with partitioned by year and month based on the time field that you parsed in step #4
6. Reload the partitioned dataset and verify the number of record matches with the original.

# **Exercise 10: Convert RDD to Dataframe**

There are a couple of ways to convert the RDD into a dataframe.

1. Supply the schema defined using StructType object
2. Infer the schema

# **Exercise 11: Convert CSV into compressed CSV, Avro, Parquet using DataFrame**

Download the stocks.csv.zip and uncompress the file.

https://drive.google.com/file/d/1Bp\_jnJ7ZVwvisaJkt6vJLm3Ow82V3WOO/view?usp=sharing

Save stocks.csv dataset in the following formats and compare the size on disk

df.write.format("json").options(compression = "gzip").mode("overwrite").save("/user/admin/stocks\_json\_gz")

| Format | Size |
| --- | --- |
| Csv | 122 MB |
| Csv with GZip compression | 43 MB |
| Json, no compression | 246 MB |
| Json with gzip compression | 54 MB |
| Parquet with Snappy | 41 MB |
| Parquery with GZip | 35 MB |

# Check the size of the file/directory

hadoop fs -du -h /user/admin/stocks\_json\_gz

# **Exercise 12: Size of DataFrame in cache**

Cache the stocks.csv in RDD format and do the same in Dataframe format and compare the memory utilization.

**Exercise 13: Submitting Application using spark-submit**

1. Create project and test locally
2. Build jar file (applicable for scala and java project. For pyspark, there is no build process)
3. Submit the jar file (.py file for pyspark) to the cluster with yarn as master and 2 executors. Assign 1 CPU and 1 GB RAM for driver and each executor.
4. Find spark session details in YARN resource manager UI (<http://localhost:8088>).

Pyspark solutions <https://github.com/abulbasar/pyspark-examples/blob/master/Dataframe_examples.ipynb>

Submission:

* Submit pom.xml, and src folder from the project.
* Code should run using maven exec:java command. For example,

mvn exec:java -Dexec.mainClass="com.example.SparkExcercise" \

-Dexec.args="/Users/abasar/data/stocks.csv.gz"

* Upload output.csv
* Put descriptive comment the code

# **Exercise 13: Find word distance with a string**

Define a UDF that finds the shortest distance (number of words in between) between two words within a given string. A word is a consecutive set of non-whitespace characters. Whitespaces are " ", "\t", "\n", "\n\r". You can use regular expression patterns to find words "\W+" or use "\s+" regular expression pattern to split a string into words.

word\_distance(col\_name, word1, word2)

Parameters

col\_name: string column

word1, word2: string values to find the distance between. Each of these can be another string column or a string literal.

Returns

* number of words in between
* null if col\_name value, word1 or word2 is null
* 0 is word1 and word2 are equal

For example: Input string is "Fast text searching for regular expressions or automaton searching on tries", word1: "text"

word2: "expression"

returns: 3

Hint:

Create a test dataframe as below

| final String[] testStrings = new String[]{  "Fast text searching for regular expressions or automaton searching on tries"  };  final Dataset<Row> dataset = sparkSession  .createDataset(Arrays.asList(testStrings), Encoders.STRING())  .toDF("value")  ; |
| --- |

# **Exercise 14: Airlines data analysis (DataFrame)**

Data:

airports.csv: Data relating to Airports. It contains name, location, altitude, type of airport details.

passengers\_by\_year\_month.csv: this file contains passenger count (domestic and international) by year and month. This data is given for the US. Source: <https://www.transtats.bts.gov/>

Folder: <https://drive.google.com/drive/folders/1mmE-Ylfs1rnqz-c7S6oveezR33rHBlPc?usp=sharing>

Use any python core modules or any other python packages to answer the following questions. While reading files, ignore any blank lines.

Use airports.csv to answer the following questions.

1. How many records are there in airports.csv excluding the header.
2. How many airports are of the type large\_airport?
3. What is the global rank of India in terms of the most number of airports? Starting rank is 1.
4. How many airports in India are closed (type = closed)?
5. In the airports.csv file, the city name is not directly mentioned. Name of the airport always does not contain the name of the city. name, keywords, municipality or iata\_code contains information about the city. Find the id for Mumbai and Kolkata airports. For example, Trivandrum airport has id 26629.
6. Define a function that returns latitude and longitude for a given airport id. What is the latitude and longitude of Mumbai airport? Round the lat and long to 2 decimal points.
7. Aerial distance between two geographical locations (latitude and longitude) is calculated using Haversine formula (<https://en.wikipedia.org/wiki/Haversine_formula>). Write down a function that returns distance between two geo locations. Consider the radius of Earth 6367.5 KM. What is the distance in KM between the international airport at Mumbai and that in Kolkata? Round the answer to the nearest integer.
8. Which airport [specify the id] is farthest international from the international airport at Mumbai? Assume that an international airport contains international in its name.
9. In the above question, what is the distance in KM between the two airports? Round the answer to the nearest integer.

***Use passengers\_by\_year\_month.csv to answer the following.***

1. In 2019, which month shows the highest number of passenger traffic? Specify the month number starting with Jan = 1. Consider both the domestic and international. [Difficulty: 1]
2. Is the above pattern the same in 2017 and 2018 - Yes/No?
3. How many months' data is given for the year 2020?
4. Due to Covid, the passenger traffic has been significantly reduced. Which month in 2020 shows the highest drop in passenger traffic in comparison to the same month last year.
5. In the above question, what is the percentage of drop? Round off your answer to the nearest integer.
6. As per 2019 data, what percentage of passenger traffic is domestic? Round off your answer to the nearest integer.
7. As per 2019 data, calculate percentage of passenger traffic per financial quarter. Consider Q1 starts with Jan. What is the percentage of Q1 to Q4? Use single digit precision for your answers.

# 

# **Exercise 15: Submit Spark Application**

Submit spark application to run on cluster.

Check the spark version. Version of the spark inside the cluster should be compatible with the version of spark used in the project. Minor-version can be different, but major-version must match.

| $ spark-submit --version |
| --- |

Check the java version. Expected version should be 1.8

| $ java -version |
| --- |

Build the jar file

In the IntelliJ project open the terminal and run the following command to build the application and create a jar file.

| $ mvn clean $ mvn package |
| --- |

You can find the output jar file inside the target directory.

View the classes inside the jar

| $ jar tf <target/jar file name> |
| --- |

Transfer the file to cloudera sandbox. Inside the sandbox you will submit the jar to run using spark-submit command.

| $ scp -P 2222 <jar file path> cloudera@localhost:~/ |
| --- |

Download the dependent jar files. In our project we need argparser4j.

| $ wget https://repo1.maven.org/maven2/net/sourceforge/argparse4j/argparse4j/0.9.0/argparse4j-0.9.0.jar |
| --- |

You can find the link in the maven repository.

Download stocks from [google drive](https://drive.google.com/file/d/15_BF38a6IIFVbp27kypxlq1TaHf4eBlO/view?usp=sharing) and upload to HDFS location /user/cloudera/stocks

Submit the spark application using spark-submit

| $ spark-submit --master yarn --**class** <classname> --**jars** <dependent **jar** **files**> **<project jar file>** --**input** <input **HDFS** **path**> --**output** <output **hdfs** **path**> |
| --- |

Text marked in red will vary depending on the project and input and output path. The

For example:

| $ spark-submit --**class** **com**.**example**.**dataframe**.**StockApp** --**jars** **argparse4j**-0.9.0.**jar** **SparkDemo**-1.0-**SNAPSHOT**.**jar** --**input** /**user**/**cloudera**/**stocks** --**output** /**user**/**cloudera**/**stocks\_agg**/ |
| --- |

Find complete list of submit options

<https://spark.apache.org/docs/latest/submitting-applications.html>

Find the spark configuration

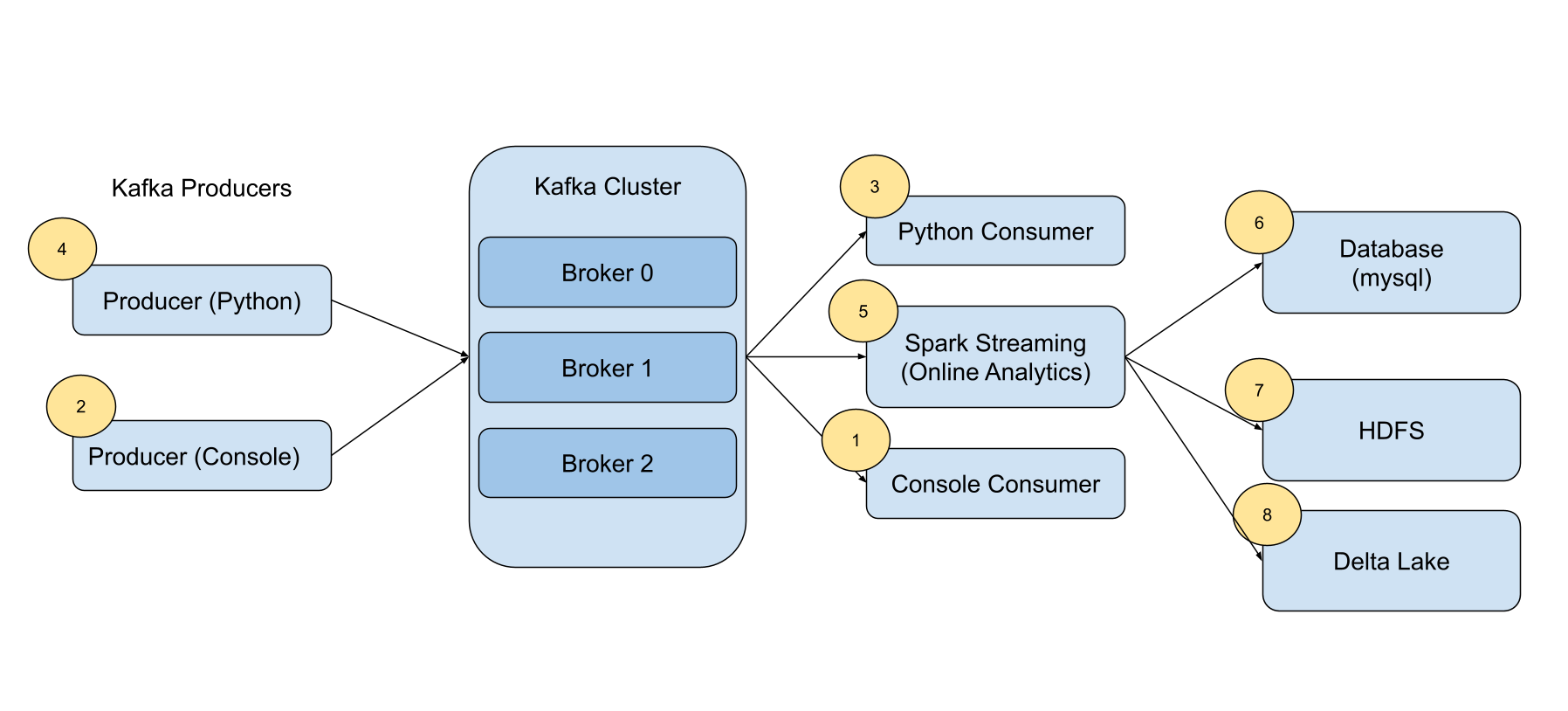
<https://spark.apache.org/docs/latest/configuration.html>

If you want to update default configuration

update the following file

$SPARK\_HOME/conf/spark-defaults.conf

# **Exercise 16: Structured Streaming (Kafka)**



Login to any of the cluster nodes.

Check kafka service status on each node

| sudo systemctl status kafka-server.service |
| --- |

Check all running brokers

| /app/kafka/bin/zookeeper-shell.sh $(hostname):2181 ls \ /kafka/brokers/ids |
| --- |

Get details of a kafka broker

| /app/kafka/bin/zookeeper-shell.sh $(hostname):2181 get \ /kafka/brokers/ids/**1** |
| --- |

Show topics

| /app/kafka/bin/kafka-topics.sh --zookeeper $(hostname):2181/kafka --list |
| --- |

Create topic

| /app/kafka/bin/kafka-topics.sh --zookeeper $(hostname):2181/kafka --topic T1 --create --partitions 1 --replication-factor 1 |
| --- |

Describe topic

| /app/kafka/bin/kafka-topics.sh --zookeeper $(hostname):2181/kafka --topic T1 --describe |
| --- |

Test Kafka cluster using kafka-console consumer and producer that are packaged as a part of a Kafka binary.

Start a new terminal and start the consumer. DO NOT close this terminal. Streaming applications are real time in nature. Keep them running.

| /app/kafka/bin/kafka-console-consumer.sh --bootstrap-server $(hostname):9092 --topic T1 |
| --- |

In another terminal start Producer and type in some messages and press enter to send to Kafka. Messages sent from the producer will appear on the consumer terminal.

| /app/kafka/bin/kafka-console-producer.sh --bootstrap-server $(hostname):9092 --topic T1 |
| --- |

You can find the location of the kafka logs (basically where Kafka stores the data) in the $KAFKA\_HOME/config/server.properties. Look for "log.dirs". You can find a directory for each topic-partition combination (e.g T1-0). Under this directory you can find multiple file-sets. Each set contains an offset index file, a time index file and data file. These files are called segment files. Kafka keeps writing to the active segment file as new messages arrive to the topic. When the messages expire based on the TTL, Kafka automatically deletes the segment files. At a given time only one segment file will be active for writing. How long Kafka writes to a segment before creating a new segment file (a.k.a rolling) depends on a few configurable parameters such as log.roll.hours, log.roll.ms.

-rw-r--r-- 1 ubuntu ubuntu 10485760 Mar 23 15:01 00000000000000000000.index

-rw-r--r-- 1 ubuntu ubuntu 400168092 Mar 23 15:01 00000000000000000000.log

-rw-r--r-- 1 ubuntu ubuntu 10485756 Mar 23 15:01 00000000000000000000.timeindex

These are binary files. You can see the binary content.

| hexdump -C "00000000000000000000.log" |
| --- |

Using the producer you can send content of a file as well. For example,

| cat /etc/passwd | /app/kafka/bin/kafka-console-producer.sh --bootstrap-server $(hostname):9092 --topic T1 |
| --- |

When you want to close the terminal producer or consumer, press CTRL + C.

We send and receive messages from Kafka very easily. For that we can use the kafka-python package.

Install kafka-python

| pip install kafka-python |
| --- |

Download the consumer.py script.

| wget https://raw.githubusercontent.com/abulbasar/pyspark-examples/master/kafka-clients/json\_consumer.py |
| --- |

Update the server name in the hostname field. Create the topic "events" as before. Start python consumer. It is a single node consumer application. Keep it running. It keeps checking if any new message arrives at topic "events" and shows them on the console.

| python json\_consumer.py |
| --- |

Open another terminal and start the producer. First download the python file as below and update the hostname.

| wget https://raw.githubusercontent.com/abulbasar/pyspark-examples/master/kafka-clients/json\_producer.py |
| --- |

Start the producer. The producer randomly generates a message in json format and sends to the topic "events" at 0.1 sec interval.

| python json\_producer.py |
| --- |

For big data, a single consumer is not suitable. We can use spark applications as a consumer group. Each executor of the spark application acts as a kafka consumer. By specifying the number of executors, it is easy to consume a high velocity kafka topic with Spark's distributed computation framework. Start spark structured streaming example

| wget https://raw.githubusercontent.com/abulbasar/pyspark-examples/master/structured\_streaming\_kafka.py |
| --- |

In the above file update the host name and topic name as required and start the streaming application.

| export SPARK\_HOME=/app/spark $SPARK\_HOME/bin/spark-submit --master yarn --verbose structured\_streaming\_kafka.py |
| --- |