

End-to-End Big Data Pipeline: Leetcode CSV to Spark Python3 Visual Analytics

Ravi Kumar Rangu		LinkedIn		GitHub
------------------	--	--------------------------	--	------------------------

May 18, 2025

Overview

This document explains the complete process of importing Leetcode problem data from a CSV file into a MySQL database, transferring it to Hive using Sqoop, creating and populating Hive partitions, performing data analysis using Spark (Scala), storing the analytical results in HDFS, and finally conducting visual analytics using Spark with Python (PySpark) and Matplotlib. Each major step is supported with corresponding command outputs and placeholders for screenshots.

Architecture Diagram

End-to-End Big Data Pipeline (Leetcode CSV to Visual Analytics)

Project Workflow

1. [Leetcode.csv](#)
2. [MySQL](#)
3. [Sqoop](#)
4. [Hive](#)
5. [Hive Partitions](#)
6. [Spark Analytics](#)
7. [HDFS](#)
8. [Spark Python3 Visual Analytics](#)

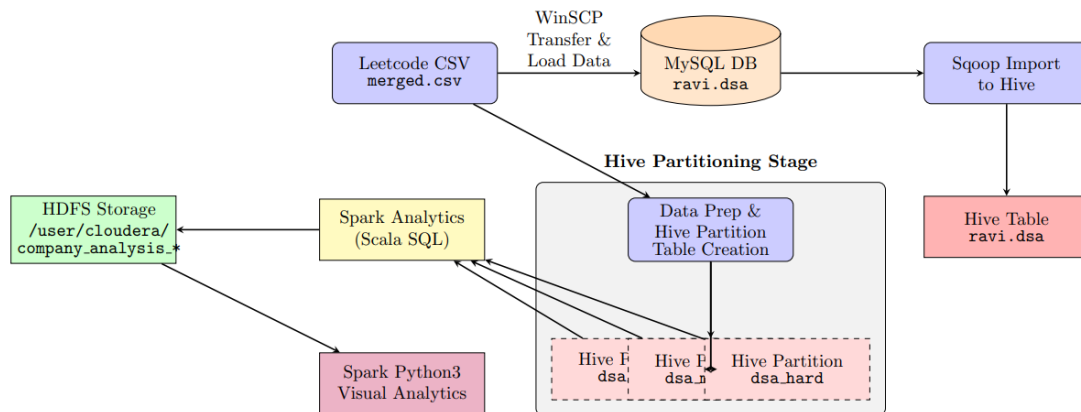


Figure 1: Architecture diagram of the data pipeline from Leetcode CSV to Visual Analytics

Step-by-Step Execution

Step 1: Dataset Preparation and Transfer

The primary dataset used is ‘Leetcode.csv’ (referred to as ‘merged.csv’). This file is transferred to the Cloudera VM’s filesystem under ‘/home/cloudera’ using WinSCP. This step is crucial for making the data accessible to the subsequent processes in the pipeline.

```
-- Leetcode.csv as merged.csv
-- Winscp leetcode.csv into /home/cloudera
```

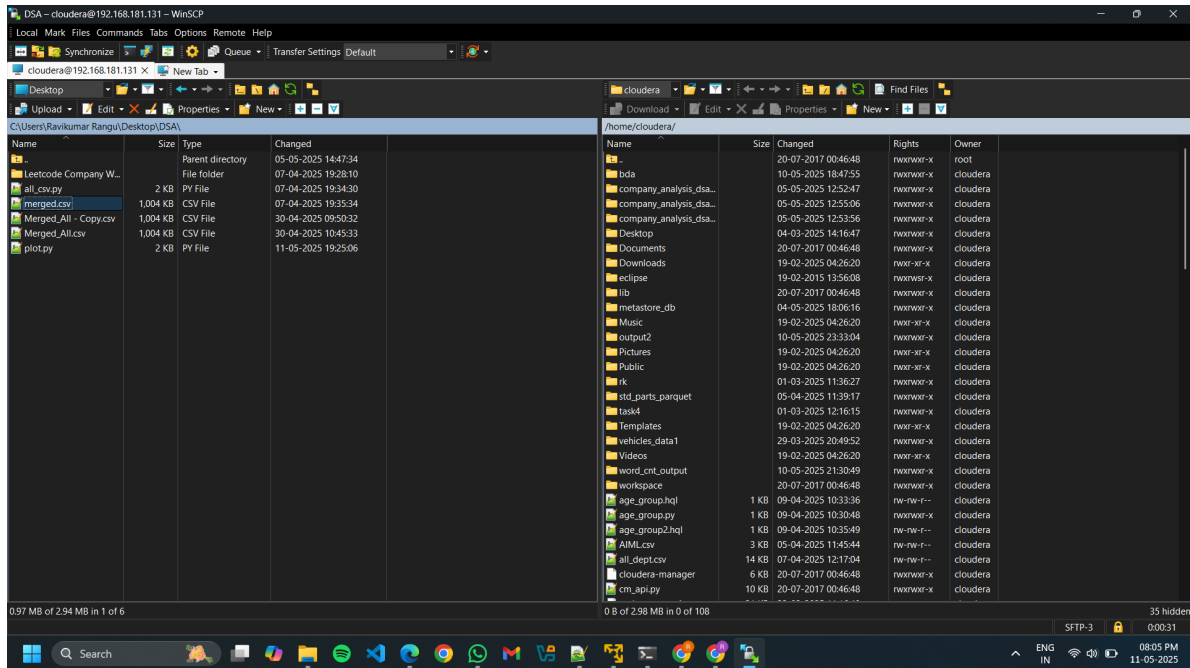


Figure 2: Dataset Transfer to VM (e.g., using WinSCP)

Step 2: Loading Data into MySQL

Once the CSV file is on the VM, it's loaded into a MySQL database. First, a table named 'dsa' is created in the 'ravi' database with an appropriate schema to hold the Leetcode data. Then, the 'LOAD DATA INFILE' command is used to populate this table from 'merged.csv'.

```
mysql> Create table dsa(
    company_name varchar(250),
    difficulty varchar(100),
    title varchar(250),
    frequency float,
    Acc_rate decimal(38,16),
    link varchar(255),
    topics varchar(255)
);

mysql> LOAD DATA INFILE '/home/cloudera/merged.csv'
INTO TABLE dsa
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

```

cloudera@quickstart:~$ mysql -uroot -pcloudera
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 74
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use ravi;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_ravi |
+-----+
| dsa             |
| tab             |
+-----+
2 rows in set (0.00 sec)

mysql> select * from dsa limit 2;
+-----+-----+-----+-----+-----+-----+-----+
| company_name | difficulty | title | frequency | Acc_rate | Link | topics |
+-----+-----+-----+-----+-----+-----+-----+
| Accenture    | MEDIUM   | Bulb Switcher | 100 | 0.5371851376516726 | https://leetcode.com/problems/bulb-switcher | Math, Brainteaser"
| Accenture,EASY,Happy Number 99,1,0.5755148228228275,https://leetcode.com/problems/happy-number,"Hash Table, Math, Two Pointers"
| Accenture,EASY,Two Sum,97,0.6559733212976659,https://leetcode.com/problems/two-sum,"Array, Hash Table"
+-----+-----+-----+-----+-----+-----+-----+
| Accenture    | EASY      | N-th Tribonacci Number | 38.5 | 0.6364687391528628 | https://leetcode.com/problems/n-th-tribonacci-number | Math, Dynamic Programming, Memoization"
| Accenture,MEDIUM,Basic Calculator II,38.5,0.4525733686159466,https://leetcode.com/problems/basic-calculator-ii,"Math, String, Stack"
| Accenture,MEDIUM,Group Anagrams,38.5,0.7833788846418984,https://leetcode.com/problems/group-anagrams"
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

Figure 3: Creating MySQL table and Loading Data

Step 3: Sqoop Import from MySQL to Hive

Apache Sqoop is utilized to import the structured data from the MySQL table 'ravi.dsa' into the Hadoop ecosystem, specifically creating a Hive table named 'ravi.dsa'. This command handles the schema creation in Hive and data transfer.

```

$ sqoop import \
--connect jdbc:mysql://localhost/ravi \
--username root \
--password cloudera \
--table dsa \
--hive-import \
--create-hive-table \
--hive-table ravi.dsa \
--delete-target-dir \
--direct -m 1

```

```

cloudera@quickstart-:~$ sqoop import \
> --connect jdbc:mysql://localhost/ravi \
> --username root \
> --password cloudera \
> --table dsa \
> --hive-import \
> --create-hive-table \
> --hive-table ravi.dsa \
> --delete-target-dir \
> --direct --h
Warning: /usr/lib/sqoop/./accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
25/05/10 14:08:11 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.12.0
25/05/10 14:08:12 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
25/05/10 14:08:12 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
25/05/10 14:08:12 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
25/05/10 14:08:12 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
25/05/10 14:08:12 INFO tool.CodeGenTool: Beginning code generation
25/05/10 14:08:12 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'dsa' AS t LIMIT 1
25/05/10 14:08:12 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM 'dsa' AS t LIMIT 1
25/05/10 14:08:12 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
Note: /tmp/sqoop-cloudera/compile/9cb8f6a69e8f45f19a1345c12e730405/dsa.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
25/05/10 14:08:14 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-cloudera/compile/9cb8f6a69e8f45f19a1345c12e730405/dsa.jar
25/05/10 14:08:15 INFO tool.ImportTool: Destination directory dsa is not present, hence not deleting.
25/05/10 14:08:15 INFO manager.DirectMySQLManager: Beginning mysqldump fast path import
25/05/10 14:08:15 INFO mapreduce.ImportJobBase: Beginning import of dsa
25/05/10 14:08:15 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
25/05/10 14:08:15 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
25/05/10 14:08:15 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
25/05/10 14:08:15 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/05/10 14:08:19 INFO db.DBInputFormat: Using read committed transaction isolation
25/05/10 14:08:19 INFO mapreduce.JobSubmitter: number of splits:1
25/05/10 14:08:20 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1746868259972_0034
25/05/10 14:08:20 INFO impl.VarnClientImpl: Submitted application application_1746868259972_0034
25/05/10 14:08:20 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8080/proxy/application_1746868259972_0034/
25/05/10 14:08:20 INFO mapreduce.Job: Running job: job_1746868259972_0034
25/05/10 14:08:29 INFO mapreduce.Job: Job job_1746868259972_0034 running in uber mode : false
25/05/10 14:08:29 INFO mapreduce.Job: map 0% reduce 0%
25/05/10 14:08:35 INFO mapreduce.Job: map 100% reduce 0%
25/05/10 14:08:35 INFO mapreduce.Job: Job job_1746868259972_0034 completed successfully
25/05/10 14:08:35 INFO mapreduce.Job: Counters: 38
File System Counters
FILE: Number of bytes read=0
FILE: Number of bytes written=151741
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=87
HDFS: Number of bytes written=1017410

```

Figure 4: Sqoop Import to Hive

Step 4: Data Preparation and Hive Table Creation for Partitions

To optimize queries and manage data effectively, the data is partitioned by difficulty. An 'awk' script processes the 'merged.csv' file to split it into separate CSV files based on the 'difficulty' column ('EASY.csv', 'MEDIUM.csv', 'HARD.csv'). Subsequently, partitioned Hive tables ('dsa_{easy}', 'dsa_{medium}', 'dsa_{hard}') are created in the 'parts1' database.

```

$ awk -F',' '
BEGIN {
    OFS = ","
}
{
    difficulty = $2;
    file = difficulty ".csv";
    print >> file;
}' merged.csv

```

```

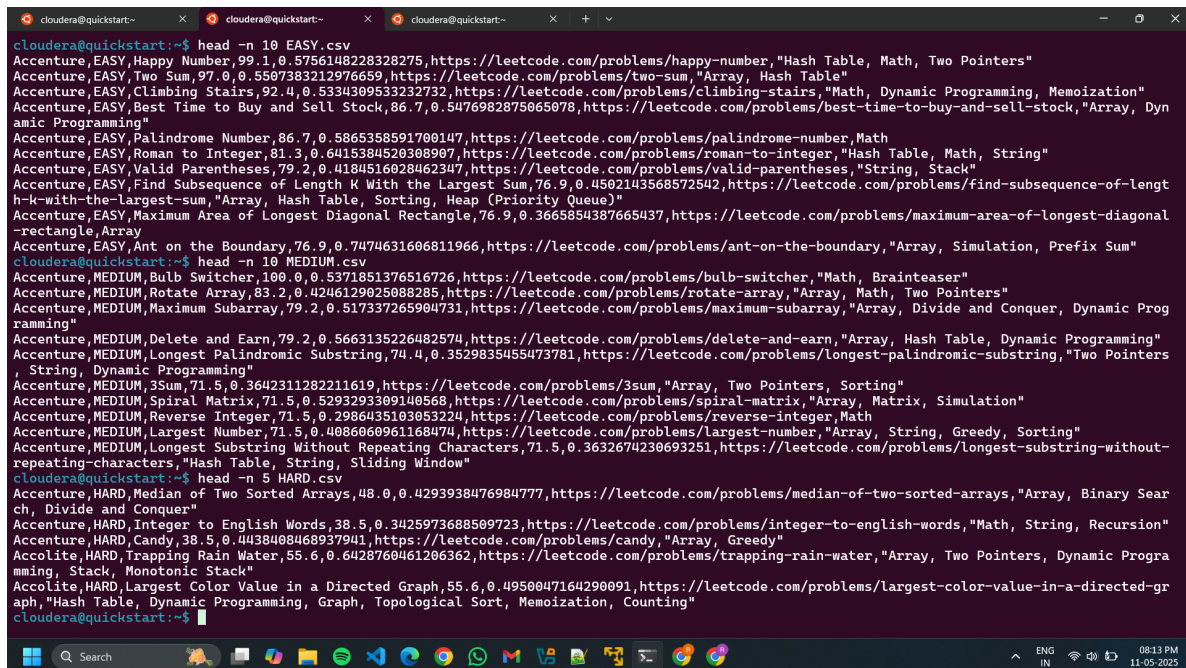
Hive > Create table dsa_easy(
    company_name varchar(250),
    title varchar(250),
    frequency float,
    Acc_rate decimal(38,16),
    link varchar(255),
    topics varchar(255)
) PARTITIONED BY (difficulty string)

```

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

```
Hive > Create table dsa_medium(  
    company_name varchar(250),  
    title varchar(250),  
    frequency float,  
    Acc_rate decimal(38,16),  
    link varchar(255),  
    topics varchar(255)  
) PARTITIONED BY (difficulty string)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

```
Hive > Create table dsa_hard(  
    company_name varchar(250),  
    title varchar(250),  
    frequency float,  
    Acc_rate decimal(38,16),  
    link varchar(255),  
    topics varchar(255)  
) PARTITIONED BY (difficulty string)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```



```
cloudera@quickstart:~$ head -n 10 EASY.csv  
Accenture,EASY,Happy Number,99.1,0.5756148228328275,https://leetcode.com/problems/happy-number,"Hash Table, Math, Two Pointers"  
Accenture,EASY,Two Sum,97.0,0.5587383212976659,https://leetcode.com/problems/two-sum,"Array, Hash Table"  
Accenture,EASY,Climbing Stairs,92.4,0.5334309533232732,https://leetcode.com/problems/climbing-stairs,"Math, Dynamic Programming, Memoization"  
Accenture,EASY,Best Time to Buy and Sell Stock,86.7,0.5476982875065078,https://leetcode.com/problems/best-time-to-buy-and-sell-stock,"Array, Dynamic Programming"  
Accenture,EASY,Palindrome Number,86.7,0.5865358591700147,https://leetcode.com/problems/palindrome-number,"Math"  
Accenture,EASY,Roman to Integer,81.3,0.6415384520308907,https://leetcode.com/problems/roman-to-integer,"Hash Table, Math, String"  
Accenture,EASY,Valid Parentheses,79.2,0.4184516028462347,https://leetcode.com/problems/valid-parentheses,"String, Stack"  
Accenture,EASY,Find Subsequence of Length K With the Largest Sum,76.9,0.4502143568572542,https://leetcode.com/problems/find-subsequence-of-length-k-with-the-largest-sum,"Array, Hash Table, Sorting, Heap (Priority Queue)"  
Accenture,EASY,Maximum Area of Longest Diagonal Rectangle,76.9,0.3665854387665437,https://leetcode.com/problems/maximum-area-of-longest-diagonal-rectangle,"Array"  
Accenture,EASY,Ant on the Boundary,76.9,0.7474631606811966,https://leetcode.com/problems/ant-on-the-boundary,"Array, Simulation, Prefix Sum"  
cloudera@quickstart:~$ head -n 10 MEDIUM.csv  
Accenture,MEDIUM,Bulb Switcher,100.0,0.6371851376516726,https://leetcode.com/problems/bulb-switcher,"Math, Brainteaser"  
Accenture,MEDIUM,Rotate Array,83.2,0.4246129025088285,https://leetcode.com/problems/rotate-array,"Array, Math, Two Pointers"  
Accenture,MEDIUM,Maximum Subarray,79.2,0.617337265904731,https://leetcode.com/problems/maximum-subarray,"Array, Divide and Conquer, Dynamic Programming"  
Accenture,MEDIUM,Delete and Earn,79.2,0.5663135226482574,https://leetcode.com/problems/delete-and-earn,"Array, Hash Table, Dynamic Programming"  
Accenture,MEDIUM,Longest Palindromic Substring,74.4,0.3529835455473781,https://leetcode.com/problems/longest-palindromic-substring,"Two Pointers, String, Dynamic Programming"  
Accenture,MEDIUM,3Sum,71.5,0.3642311282211619,https://leetcode.com/problems/3sum,"Array, Two Pointers, Sorting"  
Accenture,MEDIUM,Spiral Matrix,71.5,0.5293293309140568,https://leetcode.com/problems/spiral-matrix,"Array, Matrix, Simulation"  
Accenture,MEDIUM,Reverse Integer,71.5,0.2986435103053224,https://leetcode.com/problems/reverse-integer,"Math"  
Accenture,MEDIUM,Largest Number,71.5,0.4086060961168474,https://leetcode.com/problems/largest-number,"Array, String, Greedy, Sorting"  
Accenture,MEDIUM,Longest Substring Without Repeating Characters,71.5,0.3632674230693251,https://leetcode.com/problems/longest-substring-without-repeating-characters,"Hash Table, String, Sliding Window"  
cloudera@quickstart:~$ head -n 5 HARD.csv  
Accenture,HARD,Median of Two Sorted Arrays,48.0,0.4293938476984777,https://leetcode.com/problems/median-of-two-sorted-arrays,"Array, Binary Search, Divide and Conquer"  
Accenture,HARD,Integer to English Words,38.5,0.3425973688509723,https://leetcode.com/problems/integer-to-english-words,"Math, String, Recursion"  
Accenture,HARD,Candy,38.5,0.4438408468937941,https://leetcode.com/problems/candy,"Array, Greedy"  
Accolite,HARD,Trapping Rain Water,55.6,0.6428760461206362,https://leetcode.com/problems/trapping-rain-water,"Array, Two Pointers, Dynamic Programming, Stack, Monotonic Stack"  
Accolite,HARD,Largest Color Value in a Directed Graph,55.6,0.4950047164290091,https://leetcode.com/problems/largest-color-value-in-a-directed-graph,"Hash Table, Dynamic Programming, Graph, Topological Sort, Memoization, Counting"
```

Figure 5: AWK processing and Hive Partitioned Table Creation

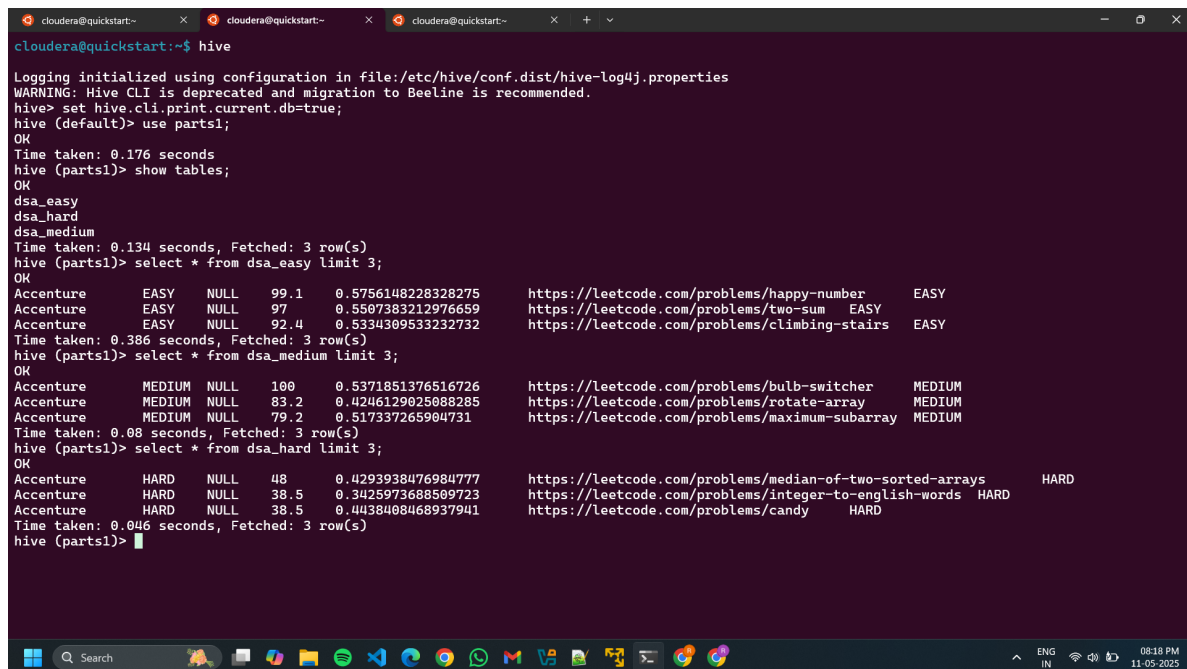
Step 5: Loading Data into Hive Partitions

The data from the difficulty-specific CSV files ('EASY.csv', 'MEDIUM.csv', 'HARD.csv') is loaded into their respective Hive partitioned tables.

```
Hive> LOAD DATA LOCAL INPATH '/home/cloudera/EASY.csv'
INTO TABLE dsa_easy partition (difficulty="EASY");
```

```
Hive> LOAD DATA LOCAL INPATH '/home/cloudera/MEDIUM.csv'
INTO TABLE dsa_medium partition (difficulty="MEDIUM");
```

```
Hive> LOAD DATA LOCAL INPATH '/home/cloudera/HARD.csv'
INTO TABLE dsa_hard partition (difficulty="HARD");
```



```
cloudera@quickstart:~$ hive
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> set hive.cli.print.current.db=true;
hive (default)> use parts1;
OK
Time taken: 0.176 seconds
hive (parts1)> show tables;
OK
dsa_easy
dsa_hard
dsa_medium
Time taken: 0.134 seconds, Fetched: 3 row(s)
hive (parts1)> select * from dsa_easy limit 3;
OK
Accenture      EASY  NULL  99.1  0.5756148228328275  https://leetcode.com/problems/happy-number  EASY
Accenture      EASY  NULL  97    0.5507383212976659  https://leetcode.com/problems/two-sum       EASY
Accenture      EASY  NULL  92.4  0.5334309533232732  https://leetcode.com/problems/climbing-stairs EASY
Time taken: 0.386 seconds, Fetched: 3 row(s)
hive (parts1)> select * from dsa_medium limit 3;
OK
Accenture      MEDIUM NULL  100   0.5371851376516726  https://leetcode.com/problems/bulb-switcher  MEDIUM
Accenture      MEDIUM NULL  83.2  0.4246129025088285  https://leetcode.com/problems/rotate-array   MEDIUM
Accenture      MEDIUM NULL  79.2  0.517337265904731   https://leetcode.com/problems/maximum-subarray MEDIUM
Time taken: 0.08 seconds, Fetched: 3 row(s)
hive (parts1)> select * from dsa_hard limit 3;
OK
Accenture      HARD  NULL  48    0.4293938476984777  https://leetcode.com/problems/median-of-two-sorted-arrays  HARD
Accenture      HARD  NULL  38.5  0.3425973688509723  https://leetcode.com/problems/integer-to-english-words     HARD
Accenture      HARD  NULL  38.5  0.4438408468937941  https://leetcode.com/problems/candy             HARD
Time taken: 0.046 seconds, Fetched: 3 row(s)
hive (parts1)>
```

Figure 6: Loading Data into Hive Partitions

Step 6: Spark Analytics (Scala)

Apache Spark (using Scala and HiveContext) is employed to perform analytics on the partitioned Hive data. Spark SQL queries are executed to calculate average acceptance rates and frequencies for companies, grouped by difficulty. The Spark shell is launched with necessary JARs for CSV handling.

```
// Launch Spark Shell with CSV support
$ spark-shell --jars
spark-csv_2.10-1.5.0.jar,
univocity-parsers-1.5.1.jar,
```

```
commons-csv-1.1.jar

// Inside spark-shell
Scala> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)

Scala> val resultDF1 = sqlContext.sql(
  "select company_name, avg(acc_rate) as Avg_frequency, avg(link) as Avg_acceptancerate
  from parts1.dsa_easy
  group by company_name
  order by Avg_frequency desc, Avg_acceptancerate desc"
)

Scala> val resultDF2 = sqlContext.sql(
  "select company_name, avg(acc_rate) as Avg_frequency, avg(link) as Avg_acceptancerate
  from parts1.dsa_medium
  group by company_name
  order by Avg_frequency desc, Avg_acceptancerate desc"
)

Scala> val resultDF3 = sqlContext.sql(
  "select company_name, avg(acc_rate) as Avg_frequency, avg(link) as Avg_acceptancerate
  from parts1.dsa_hard
  group by company_name
  order by Avg_frequency desc, Avg_acceptancerate desc"
)
```

```
cloudera@quickstart:~$ head -n 3 company_analytics_easy/*
head: cannot open 'company_analytics_easy/*' for reading: No such file or directory
cloudera@quickstart:~$ head -n 3 company_analysis_dsa_easy/*
==> company_analysis_dsa_easy/_SUCCESS <==

==> company_analysis_dsa_easy/part-00000 <==
company_name,Avg_frequency,Avg_acceptancerate
Deliveroo,100.00000000000000000000,0.8865600609175709
Workday,100.00000000000000000000,0.8121764945074768
cloudera@quickstart:~$ head -n 5 company_analysis_dsa_medium/*
==> company_analysis_dsa_medium/_SUCCESS <==

==> company_analysis_dsa_medium/part-00000 <==
company_name,Avg_frequency,Avg_acceptancerate
Millennium,100.00000000000000000000,25.47221651084613
Veeva Systems,100.00000000000000000000,0.6610905626801471
Directi,100.00000000000000000000,0.647493158229546
Hulu,100.00000000000000000000,0.6274542127988242
cloudera@quickstart:~$ head -n 5 company_analysis_dsa_hard/*
==> company_analysis_dsa_hard/_SUCCESS <==

==> company_analysis_dsa_hard/part-00000 <==
company_name,Avg_frequency,Avg_acceptancerate
Nike,100.00000000000000000000,0.7050424603174604
Moloco,100.00000000000000000000,0.686365629870184
OKX,100.00000000000000000000,0.6772211214052375
Intel,100.00000000000000000000,0.6428745738503359
cloudera@quickstart:~$
```

Figure 7: Spark Analytics using Scala and Spark SQL

Step 7: Saving Spark Analytics Results to HDFS

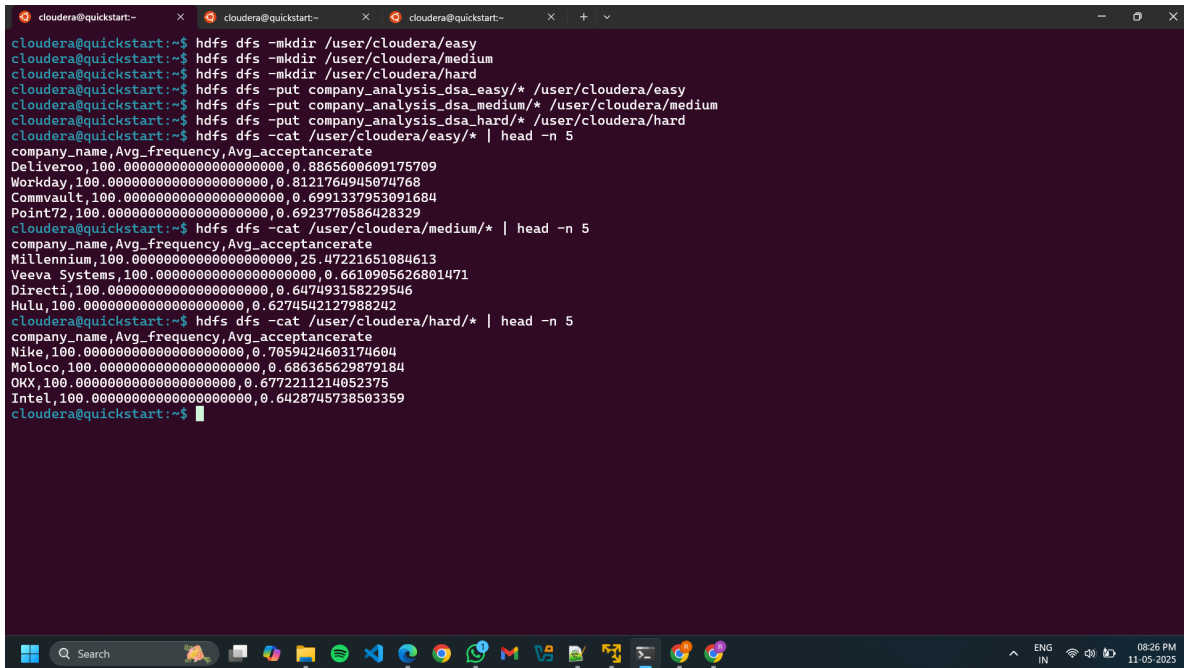
The results from the Spark analytics (DataFrames: 'resultDF1', 'resultDF2', 'resultDF3') are saved as CSV files into the local filesystem ('/home/cloudera/') and then moved to HDFS under '/user/cloudera/'.

```
// Inside spark-shell (Saving to local first)
Scala> (resultDF1.coalesce(1)
      .write
      .format("com.databricks.spark.csv")
      .option("header", "true")
      .save("file:///home/cloudera/company_analysis_dsa_easy"))

Scala> (resultDF2.coalesce(1)
      .write
      .format("com.databricks.spark.csv")
      .option("header", "true")
      .save("file:///home/cloudera/company_analysis_dsa_medium"))

Scala> (resultDF3.coalesce(1)
      .write
      .format("com.databricks.spark.csv")
      .option("header", "true")
      .save("file:///home/cloudera/company_analysis_dsa_hard"))

// Moving results to HDFS
$ hdfs dfs -put company_analysis_dsa_easy/* /user/cloudera
$ hdfs dfs -put company_analysis_dsa_medium/* /user/cloudera
$ hdfs dfs -put company_analysis_dsa_hard/* /user/cloudera
```

A screenshot of a terminal window with a dark purple background. The terminal shows a series of HDFS commands and their outputs. The commands include creating directories for 'easy', 'medium', and 'hard' under '/user/cloudera', uploading data from 'company_analysis_dsa_*.csv' to these directories, and then using 'hdfs dfs -cat' to view the contents of the 'easy' and 'medium' directories. The output shows a list of companies with their average frequency and acceptance rates, such as 'Deliveroo, 100.0000000000000000, 0.3865500609175709'. The terminal window has multiple tabs open, all labeled 'cloudera@quickstart:~'. The Windows taskbar is visible at the bottom with various application icons and a system clock showing 08:26 PM on 11-05-2025.

```
cloudera@quickstart:~$ hdfs dfs -mkdir /user/cloudera/easy
cloudera@quickstart:~$ hdfs dfs -mkdir /user/cloudera/medium
cloudera@quickstart:~$ hdfs dfs -mkdir /user/cloudera/hard
cloudera@quickstart:~$ hdfs dfs -put company_analysis_dsa_easy/* /user/cloudera/easy
cloudera@quickstart:~$ hdfs dfs -put company_analysis_dsa_medium/* /user/cloudera/medium
cloudera@quickstart:~$ hdfs dfs -put company_analysis_dsa_hard/* /user/cloudera/hard
cloudera@quickstart:~$ hdfs dfs -cat /user/cloudera/easy/* | head -n 5
company_name,Avg_frequency,Avg_acceptancerate
Deliveroo,100.0000000000000000,0.3865500609175709
Workday,100.0000000000000000,0.8121764945074768
Commvault,100.0000000000000000,0.6901337953091684
Point72,100.0000000000000000,0.6923770586428379
cloudera@quickstart:~$ hdfs dfs -cat /user/cloudera/medium/* | head -n 5
company_name,Avg_frequency,Avg_acceptancerate
Millennium,100.0000000000000000,25.47221651084613
Veeva Systems,100.0000000000000000,0.6610905626801471
Directi,100.0000000000000000,0.647493158229546
Hulu,100.0000000000000000,0.6274542127988242
cloudera@quickstart:~$ hdfs dfs -cat /user/cloudera/hard/* | head -n 5
company_name,Avg_frequency,Avg_acceptancerate
Nike,100.0000000000000000,0.7059424603174604
Moloco,100.0000000000000000,0.686365629879184
OKX,100.0000000000000000,0.6772211214052375
Intel,100.0000000000000000,0.6428745738503359
cloudera@quickstart:~$
```

Figure 8: Saving Spark Results and Moving to HDFS

Step 8: Spark Python3 Visual Analytics

Finally, PySpark is used to read the processed data from HDFS. The data is converted into Pandas DataFrames for easier manipulation and visualization. Matplotlib is then used to generate plots for average frequency and acceptance rates by company, chunked for better readability.

PySpark Script Excerpts

```
import findspark
findspark.init()
```

```
from pyspark.sql import SparkSession
import pandas as pd
import matplotlib.pyplot as plt
```

```
hdfs_uri = "hdfs://192.168.181.132:8020" # Example HDFS URI
```

```
csv_path_easy = f"{hdfs_uri}/user/cloudera/company_analysis_dsa_easy" # Example path
```

```
spark = SparkSession.builder \
    .appName("Read HDFS and Analyze") \
    .config("spark.hadoop.fs.defaultFS", hdfs_uri) \
    .config("spark.hadoop.dfs.client.use.datanode.hostname", "true") \
    .getOrCreate()
```

```
df_spark_easy = spark.read.option("header", "true").csv(csv_path_easy)
```

```
df_easy = df_spark_easy.toPandas()
```

```
# ... (Data cleaning and type conversion as in GLOB.txt) ...

# Plotting (example for 'easy' data)
df_sorted = df_easy.sort_values(by='Avg_frequency')
chunks = [df_sorted[i:i+55] for i in range(0, len(df_sorted), 55)]

for idx, chunk in enumerate(chunks):
    plt.figure(figsize=(18, 6))
    plt.plot(chunk['company_name'], chunk['Avg_frequency'], marker='o', color='blue')
    plt.title(f"Avg Frequency Rate (Easy - Companies {idx*55+1} to {min((idx+1)*55, len(df_sorted))})")
    plt.xlabel("Company Name")
    plt.ylabel("Avg Frequency Rate")
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.grid(True)
    plt.show()

# ... (Similar plotting for Avg Acceptance Rate and for medium/hard data) ...

spark.stop()
```

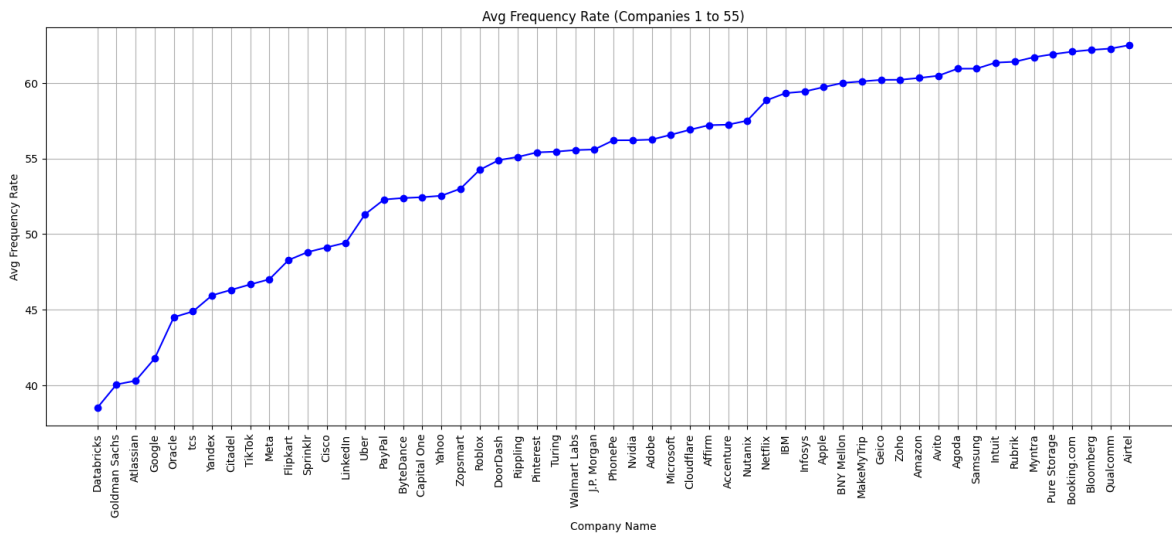


Figure 9: Spark Python3 Visual Analytics

Conclusion

This project successfully demonstrates an end-to-end data engineering pipeline processing Leetcode CSV data. The pipeline involves data ingestion into MySQL, transfer to Hive using Sqoop, creation and population of Hive partitions, advanced analytics using Spark (Scala) on these partitions, storage of results in HDFS, and finally, visual analytics using Spark Python (PySpark) and Matplotlib. Each step from initial data loading to final visualization is detailed, showcasing a comprehensive big data workflow within the Cloudera VM environment.