INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT AKURDI, PUNE

Documentation On

"Indian Railways Train Insight Analysis"

PG-DBDA SEPT-2023

Submitted By:

Group No: 9

Roll No.	**Name:**
239523	Tejas Dnyandeo Kolambe
239536	Purven Nitin Zambre

Mrs. Priti Take
Project Guide

Mr. Rohit Puranik
Centre Coordinator

Abstract

This report presents an analysis of the Indian Railway Catering and Tourism Corporation (IRCTC) web-scrapped dataset using ETL techniques in PySpark. The dataset contains information on train schedules, fares, and related details. The ETL process involved extracting data from the IRCTC website, cleaning and transforming it, and loading it into a format suitable for analysis. The analysis includes insights into fare trends, popular routes, and travel patterns, providing valuable information for decision-making and planning. Future work could explore real-time data integration and advanced analytics for further insights.

Acknowledgement

We would like to express our sincere gratitude to everyone who has contributed to the completion of our project. First and foremost, we would like to thank our project guide **Mrs. Priti Take** mam for their constant guidance and support throughout the project. We extend our sincere thanks to our respected Centre Co-Ordinator, **Mr. Rohit Puranik**, for allowing us to use the facilities available.

We would also like to express our appreciation to the faculty members of our department for their constructive feedback and encouragement. Their insights and suggestions have helped us to refine our ideas and enhance the quality of our work.

Furthermore, we would like to thank our families and friends for their unwavering support and encouragement throughout our academic journey. Their love and support have been a constant source of motivation and inspiration for us.

Thank you all for your valuable contributions to our project.

Tejas Dnyandeo Kolambe (239523)

Purven Nitin Zambre (239536)

Table Of Contents

Table of Figure:

1. Introduction

1.1 Problem Statement

Indian Railways Train Insight Analysis

1.2 Scope

The focus is on providing insights into train schedules, fares, and travel patterns from the IRCTC dataset, with the potential for further analysis and enhancements in the future. Involving steps to do these are Data collection, Data transformation, Data loading, etc.

1.3 Aim & Objective

To analyze and derive insights from IRCTC web-scraped data using PySpark, with the goal of understanding patterns, trends, and dynamics in train schedules, ticket bookings, and passenger traffic.

Through a systematic ETL process, including data collection, transformation, and loading, the raw data extracted from the IRCTC website will be cleansed, standardized, and prepared for analysis. The transformed data will then be subjected to in-depth analysis to identify correlations, anomalies, and insights related to various aspects of train services and passenger behavior. Visualizations such as charts, graphs, and maps will be utilized to illustrate key findings and facilitate interpretation. By summarizing the main findings and implications, the report aims to provide actionable insights and recommendations for stakeholders, including travelers, railway authorities, and policymakers, to enhance service delivery, optimize resource allocation, and improve the overall user experience. Through comprehensive documentation and reporting, the report seeks to provide a valuable resource for decision-making and strategic planning in the context of Indian railway transportation.

2. Tools & Techniques

2.1 Python

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020.

2.2 Pyspark

PySpark is the Python API for Apache Spark, allowing developers to interact with Spark using Python programming language. It provides a familiar Python syntax and enables seamless integration with Python libraries and tools for data analysis and machine learning.

PySpark provides APIs for manipulating distributed data sets called Resilient Distributed Datasets (RDDs) and higher-level abstractions such as DataFrames and Spark SQL for structured data processing.

Distributing: PySpark leverages Spark's distributed computing capabilities, enabling parallel processing of large-scale data across a cluster of machines.

Easy to use : With its Pythonic syntax and high-level APIs, PySpark simplifies the development of complex data processing workflows.

Seamless: PySpark seamlessly integrates with Python libraries and tools such as NumPy, pandas, scikit-learn, TensorFlow, and more, allowing data scientists to leverage their existing Python skills and ecosystem.

Scalable: PySpark scales from a single machine to thousands of nodes, making it suitable for processing petabytes of data across large clusters.

Fault Toleance: Spark's built-in fault-tolerance mechanisms ensure reliable processing of data even in the presence of failures.

Support: PySpark supports various data sources including Hadoop Distributed File System (HDFS), Apache Hive, Apache HBase, Apache Cassandra, JDBC databases, and more.

PySpark is widely used for various data processing tasks such as data cleaning, transformation, aggregation, and analysis.
It's commonly used in industries like finance, healthcare, e-commerce, advertising, and telecommunications for big data analytics, predictive modeling, recommendation systems, and more.

PySpark includes several components such as Spark Core (the foundation of Spark), Spark SQL (for structured data processing and querying), Spark Streaming (for real-time stream processing), MLlib (for machine learning), GraphX (for graph processing), and SparkR (for R programming language).

2.3 HDFS (Hadoop Distributed File System)

In PySpark, HDFS (Hadoop Distributed File System) serves as a primary storage system for distributed data processing. PySpark leverages HDFS to store and manage large volumes of data across a cluster of machines. The integration of PySpark with HDFS allows users to read and write data seamlessly from and to HDFS using familiar Python syntax and APIs. Data stored in HDFS is partitioned into blocks and distributed across multiple nodes in the cluster, ensuring fault tolerance and high availability.

PySpark provides APIs to interact with HDFS, allowing users to perform various data processing operations such as reading files, writing data, and performing transformations directly on HDFS-resident data. This integration enables efficient distributed data processing using PySpark, leveraging the scalability and fault tolerance capabilities of HDFS to handle large-scale datasets. Overall, the combination of PySpark and HDFS empowers users to build scalable and resilient data processing pipelines for big data analytics and machine learning applications.

2.4 Spark Session

In PySpark, a SparkSession serves as the main entry point for interacting with Spark functionality, specifically through the Dataset and DataFrame APIs. It initializes the Spark environment and provides a unified interface for creating, configuring, and managing Spark applications. To create a SparkSession, you typically use the SparkSession.builder object, where you can specify configuration options such as the application name and Spark properties. Once initialized, the SparkSession allows you to read data from various sources, including CSV, JSON, Parquet, and JDBC, and create DataFrame objects to represent the data.

You can then perform a wide range of data processing tasks using DataFrame transformations and actions, such as filtering, grouping, aggregating, and joining. Spark computations are lazily evaluated, meaning transformations are deferred until an action is triggered, ensuring efficient execution. After completing your data processing tasks, it's important to stop the SparkSession to release resources using the spark.stop**()** method. Overall, the SparkSession in PySpark provides a powerful and flexible interface for distributed data processing, enabling developers to efficiently work with large-scale datasets.

2.5 MySQL

MySQL is a robust and widely-used open-source relational database management system (RDBMS) renowned for its reliability, scalability, and ease of use. It organizes data into tables with rows and columns, facilitating the establishment of relationships between them through SQL (Structured Query Language).

MySQL Connectivity (PySpark)

In summary, connecting PySpark with MySQL involves setting up a SparkSession configured to use the MySQL JDBC driver, reading data from a MySQL table into a DataFrame, and performing data processing tasks using

PySpark's DataFrame API. This process begins with downloading and extracting the MySQL Connector/J JAR file, which serves as the JDBC driver for MySQL. Next, a SparkSession is initialized with the appropriate configuration, specifying the path to the MySQL Connector/J JAR file.

Using the jdbc() method of the SparkSession, data is then read from the MySQL table by providing the JDBC connection URL, table name, username, and password. Once the DataFrame is created, various data processing operations can be performed using PySpark's DataFrame API. Finally, after completing the data processing tasks, the SparkSession is stopped to release resources. This approach enables seamless integration between PySpark and MySQL, allowing users to leverage the distributed processing capabilities of Spark for analyzing and processing data stored in MySQL databases.

2.6 pyspark.sql.function

Is a module in PySpark that provides a wide range of functions for working with DataFrame columns. These functions enable users to perform various data manipulation, transformation, and aggregation operations on DataFrame columns efficiently and effectively.

The module includes functions for selecting, filtering, grouping, and sorting DataFrame columns, as well as functions for string manipulation, date and time operations, mathematical calculations, and more. By leveraging pyspark.sql.functions, users can easily perform complex data processing tasks on DataFrame columns within PySpark applications, making it a powerful tool for data manipulation and analysis.

2.7 Pandas

Although not a native PySpark library, the pandas library is commonly used in conjunction with PySpark for data manipulation and analysis.

You can convert PySpark DataFrames to pandas DataFrames using the toPandas() method and perform operations using pandas' rich API, which includes functions for data cleaning, transformation, aggregation, and visualization.

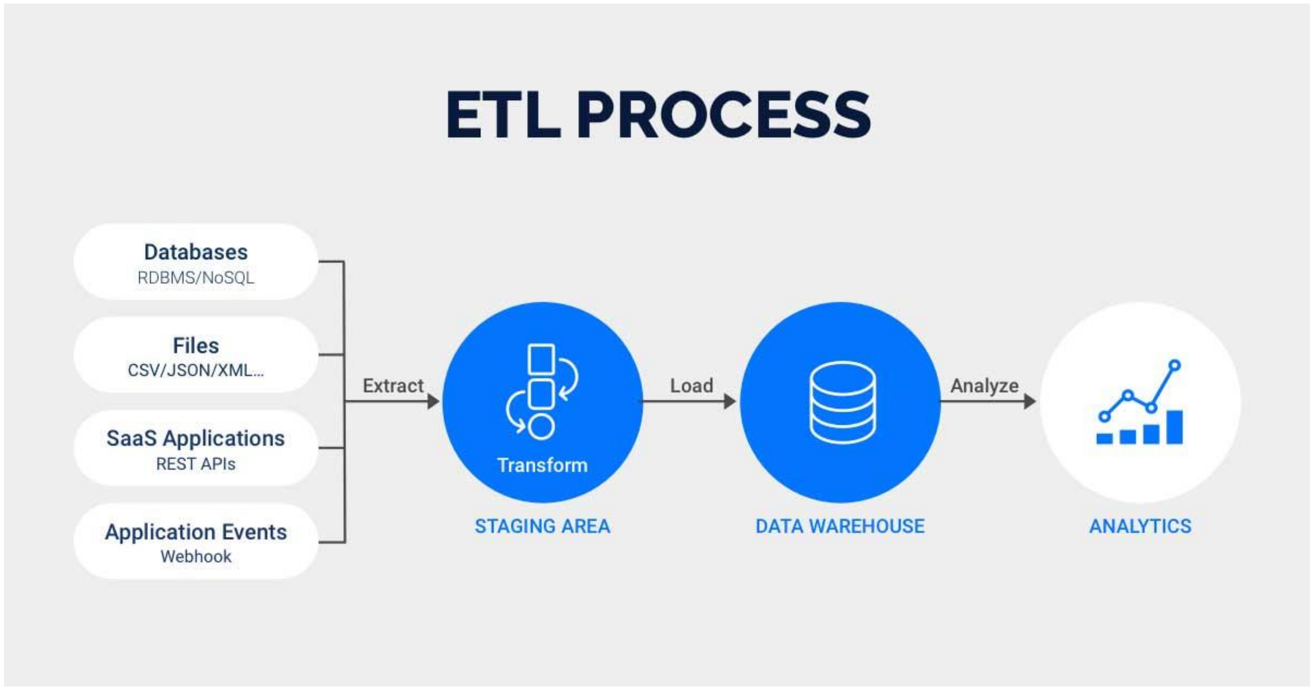
2.8 Ubuntu

Ubuntu is an open-source Linux distribution renowned for its user-friendly interface, stability, and extensive community support. It provides a desktop environment based on the GNOME desktop environment, offering a familiar graphical user interface for users transitioning from other operating systems. With its Advanced Package Tool (APT) for package management, users can easily install, update, and remove software packages from its vast software repository, covering a wide range of categories.

Supported by a large and active community, users can seek assistance from various forums, mailing lists, and IRC channels. Ubuntu comes in different editions tailored for different use cases, including desktop computing, server deployments, cloud infrastructure, and IoT devices. Its Long-Term Support (LTS) releases provide stability and support for enterprise environments, receiving updates and security patches for five years. Widely used in cloud computing and containerized environments, Ubuntu offers flexibility and customization options, allowing users to tailor the system to their specific needs and preferences. Overall, Ubuntu's versatility, reliability, and strong community support make it a popular choice for individuals and organizations worldwide.

3. Workflow

3.1 ETL (Extract Tranform Load)



Extract (E):

The first step in the ETL process is to extract the data from the source, which in this case is a CSV file. This involves reading the CSV file and loading its contents into a DataFrame using PySpark. The DataFrame represents the raw data extracted from the CSV file, with each row representing a record and each column representing a field or attribute.

Data extraction is a crucial step in the Extract, Transform, Load (ETL) pipeline, where raw data is collected from various sources for further processing. To ensure the effectiveness of this phase, it's essential to consider several key factors to extract the best content:

Source Identification:

The first step in data extraction is identifying the sources from which data needs to be collected. Sources can include databases (relational databases like MySQL, PostgreSQL, NoSQL databases like MongoDB), files (CSV, JSON, XML), APIs, streaming data sources (Kafka, Apache Kafka, Amazon Kinesis), or even web scraping.

Data Scope and Relevance:

Define the scope of the data to be extracted based on the project requirements and business objectives. It's essential to focus on extracting data that is relevant to the analysis or processing goals. Prioritize extracting high-value data that directly contributes to achieving the desired outcomes.

Data Quality Assessment:

Conduct an initial assessment of the quality of the data available in the source systems. Evaluate factors such as completeness, accuracy, consistency, and timeliness of the data. Identify any data quality issues upfront to address them during subsequent stages of the ETL process.

Extraction Methodology:

Choose the appropriate extraction method based on the characteristics of the data sources. For relational databases, SQL-based extraction methods (such as querying with SELECT statements) or database connectors (like JDBC for Java-based systems or ODBC for others) can be used. For file-based sources, libraries or tools specific to the file format (e.g., pandas for CSV files in Python) can be employed. Similarly, for APIs, utilize relevant APIs or SDKs provided by the data source. For streaming data sources, implement real-time data ingestion techniques using stream processing frameworks like Apache Kafka or Apache Flink.

Incremental Extraction:

Implement incremental extraction techniques to extract only the new or updated data since the last extraction. This approach minimizes processing time and reduces the risk of data duplication or inconsistency. Common incremental extraction methods include tracking changes using timestamp columns or change data capture (CDC) mechanisms provided by some database systems.

Data Security and Compliance:

Ensure compliance with data security and privacy regulations by implementing appropriate security measures during data extraction. Employ encryption mechanisms to protect data in transit and at rest. Implement access controls to restrict access to sensitive data and ensure that only authorized users have access to the extracted data.

Error Handling and Logging:

Implement robust error handling mechanisms to deal with potential issues encountered during data extraction. Common errors include network failures, data format errors, or source system downtime. Implement logging mechanisms to capture relevant information about extraction failures or warnings for troubleshooting and auditing purposes.

Scalability and Performance:

Design the data extraction process to scale efficiently with growing data volumes and increasing workload demands. Consider factors such as parallelization, distributed processing, and optimized data transfer protocols to improve extraction performance. Utilize cloud-based solutions or distributed computing frameworks like Apache Spark for scalable and high-performance data extraction.

Metadata Management:

Capture and maintain metadata about the extracted data to facilitate data governance, lineage tracking, and impact analysis across the ETL pipeline. Metadata should include source details, extraction timestamps, data lineage, and transformation rules applied during subsequent processing stages. Centralized metadata management solutions or metadata repositories can help manage and maintain metadata effectively.

Transform (T):

After extracting the data, the next step is to transform it to meet the desired format and quality standards. In this phase, various data cleaning and transformation operations are performed on the DataFrame to prepare it for analysis or loading into the target database (MySQL in this case).

Data cleaning operations may include handling missing or null values, removing duplicates, standardizing data formats, and correcting errors or inconsistencies in the data.

Additionally, data transformation operations may involve deriving new columns, aggregating data, applying business rules, and performing calculations or computations on the existing data.

For example, you might convert string columns to lowercase, remove leading and trailing whitespace, parse dates into a standardized format, or calculate new metrics based on existing columns.

Data Cleaning and Standardization:

Data cleaning involves identifying and rectifying errors, inconsistencies, and missing values in the raw data. Standardization ensures uniformity in data formats, units of measurement, and naming conventions. Techniques such as removing duplicates, handling missing values, correcting spelling errors, and normalizing data formats are applied to enhance data quality.

Data Enrichment and Derivation:

Data enrichment involves augmenting the raw data with additional information from external sources to enhance its value and utility. This could include enriching customer records with demographic data, appending geospatial information to location data, or retrieving market insights from third-party APIs. Data derivation involves calculating new derived attributes or metrics based on existing data, such as computing aggregates, calculating ratios, or generating derived features for machine learning models.

Data Transformation Operations:

Various data transformation operations are performed to reshape, aggregate, and manipulate the data according to the analytical requirements. This includes operations like filtering, sorting, grouping, joining, and aggregating data. Advanced transformation techniques such as pivoting, unpivoting, window functions, and custom user-defined functions (UDFs) are applied to prepare the data for analysis and reporting.

Data Quality Checks and Validation:

Data quality checks are conducted to ensure that the transformed data meets the required quality standards and is fit for its intended purpose. This involves validating data against predefined business rules, constraints, or thresholds. Quality checks may include verifying data integrity, accuracy, completeness, consistency, and conformity to regulatory requirements. Any discrepancies or anomalies detected during the validation process are flagged for further investigation or corrective action.

Performance Optimization:

Performance optimization techniques are applied to improve the efficiency and scalability of data transformation processes, especially for large-scale datasets. This includes optimizing query performance, leveraging parallel processing, caching intermediate results, and tuning transformation algorithms for optimal resource utilization. Distributed computing frameworks like Apache Spark are often used to parallelize and accelerate data transformation tasks across distributed computing clusters.

Metadata Management and Lineage Tracking:

Metadata management is crucial for documenting and tracking the lineage of transformed data, including its source, transformation logic, and destination. Metadata repositories or cataloguing systems are used to capture metadata attributes such as data lineage, schema evolution, data provenance, and transformation history. This facilitates data governance, regulatory compliance, and auditability throughout the ETL lifecycle.

Data Security and Privacy:

Data security and privacy considerations are integrated into the data transformation process to safeguard sensitive information and ensure compliance with regulatory requirements. This includes implementing encryption, access controls, anonymization techniques, and data masking to protect confidential data from unauthorized access or disclosure during transformation operations.

Load (L):

Once the data has been extracted and transformed, the final step is to load it into the target database, which in this case is MySQL. This involves creating a connection to the MySQL database and inserting the transformed data from the Data Frame into the appropriate table(s) in the database.

Before loading the data into MySQL, it's important to ensure that the database schema and table structure are compatible with the transformed data. This may involve creating or modifying tables in the database to accommodate the transformed data schema.

Once the database connection is established and the table structure is prepared, the transformed data is inserted into the MySQL database using SQL INSERT statements or other database loading mechanisms provided by PySpark.

Target System Identification:

Identify the target system or destination where the transformed data will be loaded. This could be a relational database (such as MySQL, PostgreSQL), a data warehouse (such as Amazon Redshift, Google BigQuery), a data lake (such as Hadoop Distributed File System), or a cloud storage service (such as Amazon S3, Google Cloud Storage).

Data Loading Strategy:

Choose the appropriate data loading strategy based on the characteristics of the target system and the volume of data to be loaded. Common data loading strategies include bulk loading, incremental loading, and streaming loading. Bulk loading is suitable for initial data loads or large batch updates, incremental loading is used for loading only the new or updated data since the last load, and streaming loading is employed for real-time data ingestion.

Data Loading Methods:

Select the optimal method for loading data into the target system based on its capabilities and integration options. This may involve using native database loading utilities (such as MySQL's LOAD DATA INFILE), database-specific connectors (such as JDBC for relational databases), data warehouse loading tools (such as AWS Data Pipeline, Google Dataflow), or custom scripts and APIs provided by the target system.

Data Validation and Integrity Checks:

Perform data validation and integrity checks before and after loading data into the target system to ensure the accuracy and consistency of the loaded data. Validate data against predefined business rules, constraints, or validation criteria to detect and resolve any anomalies or discrepancies. Implement data quality checks to verify data integrity, completeness, and conformity to expected standards.

Error Handling and Logging:

Implement robust error handling mechanisms to address any issues encountered during the data loading process, such as data format errors, schema mismatches, or connectivity failures. Log relevant information about loading successes, failures, and warnings for auditing, troubleshooting, and monitoring purposes. Provide detailed error messages and notifications to alert stakeholders about any issues that require attention.

Performance Optimization:

Optimize the performance of data loading operations to minimize latency, maximize throughput, and reduce resource consumption. Utilize parallel processing, batching, and partitioning techniques to load data efficiently in parallel across multiple threads or nodes. Tune database configurations, indexing strategies, and query optimizations to improve loading performance and scalability.

Metadata Management and Lineage Tracking:

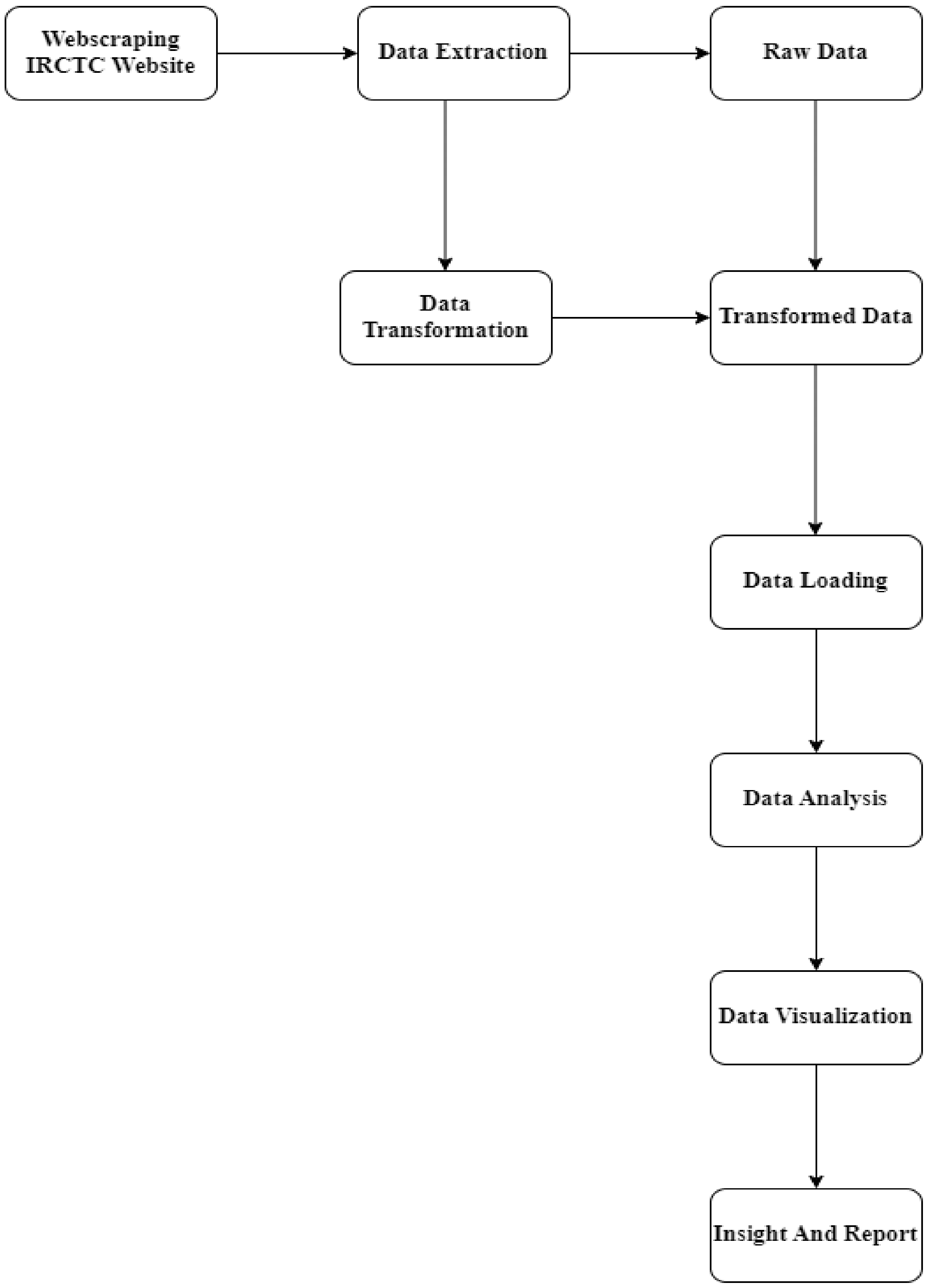
Capture and maintain metadata about the loaded data, including its source, transformation history, and destination. Document metadata attributes such as data lineage, schema mappings, load timestamps, and data provenance to facilitate data governance, lineage tracking, and impact analysis. Ensure that metadata is stored and managed consistently across the ETL pipeline to support traceability and data lineage.

Data Security and Compliance:

Implement security measures to protect the loaded data from unauthorized access, modification, or disclosure. Apply encryption, access controls, and data masking techniques to safeguard sensitive information and ensure compliance with regulatory requirements (such as GDPR, HIPAA, PCI DSS). Regularly monitor and audit data access and usage to detect and mitigate security risks.

By adhering to these best practices for data loading in the ETL pipeline, organizations can ensure that the transformed data is accurately and securely loaded into the target system for analysis and decision-making. Effective data loading lays the foundation for deriving actionable insights and driving business value from the loaded data.

3.2 Project Workflow



Data Collection:

Involves gathering data from the IRCTC website using web scraping techniques. This data may include information such as train schedules, ticket availability, fares, etc.

Data Extraction:

The extracted data is then parsed and transformed into a structured format suitable for further processing. This step may involve cleaning the data, handling missing values, and resolving inconsistencies.

Data Transformation:

In this step, the raw data is transformed to meet the requirements of the analysis. This may include standardizing formats, aggregating data, and deriving new variables.

Data Loading:

Once the data is transformed, it is loaded into a storage system or data warehouse for efficient access and retrieval during the analysis phase.

Data Analysis:

The transformed data is analyzed using PySpark to derive insights and identify patterns, trends, and correlations. Various statistical techniques and machine learning algorithms may be applied to uncover meaningful insights.

Data Visualization:

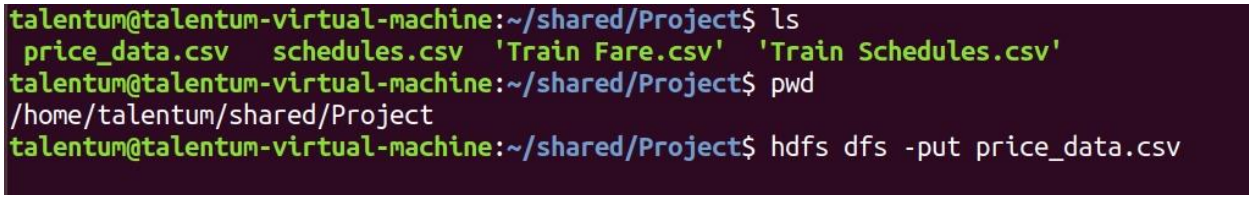
Visualizations such as charts, graphs, and maps are created to present the analysis results in a clear and intuitive manner. This helps stakeholders understand the findings more effectively.

Insights & Reports:

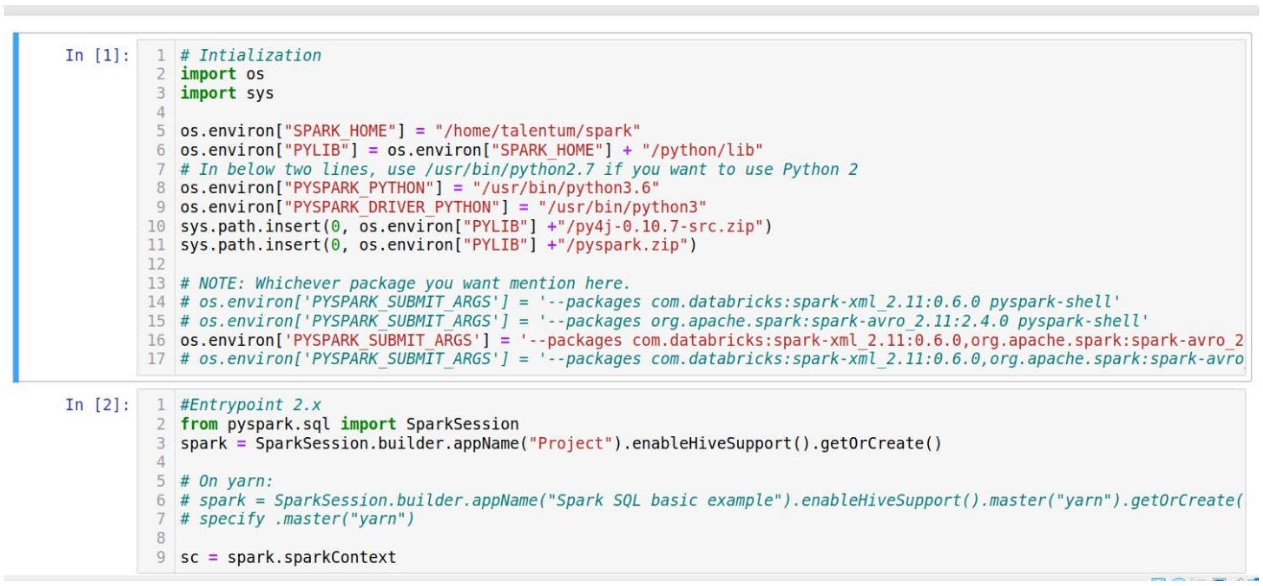
Finally, the insights derived from the analysis are summarized and compiled into a comprehensive report. This report provides actionable recommendations for stakeholders, based on the analysis of IRCTC web-scraped data using PySpark.

4. Figures

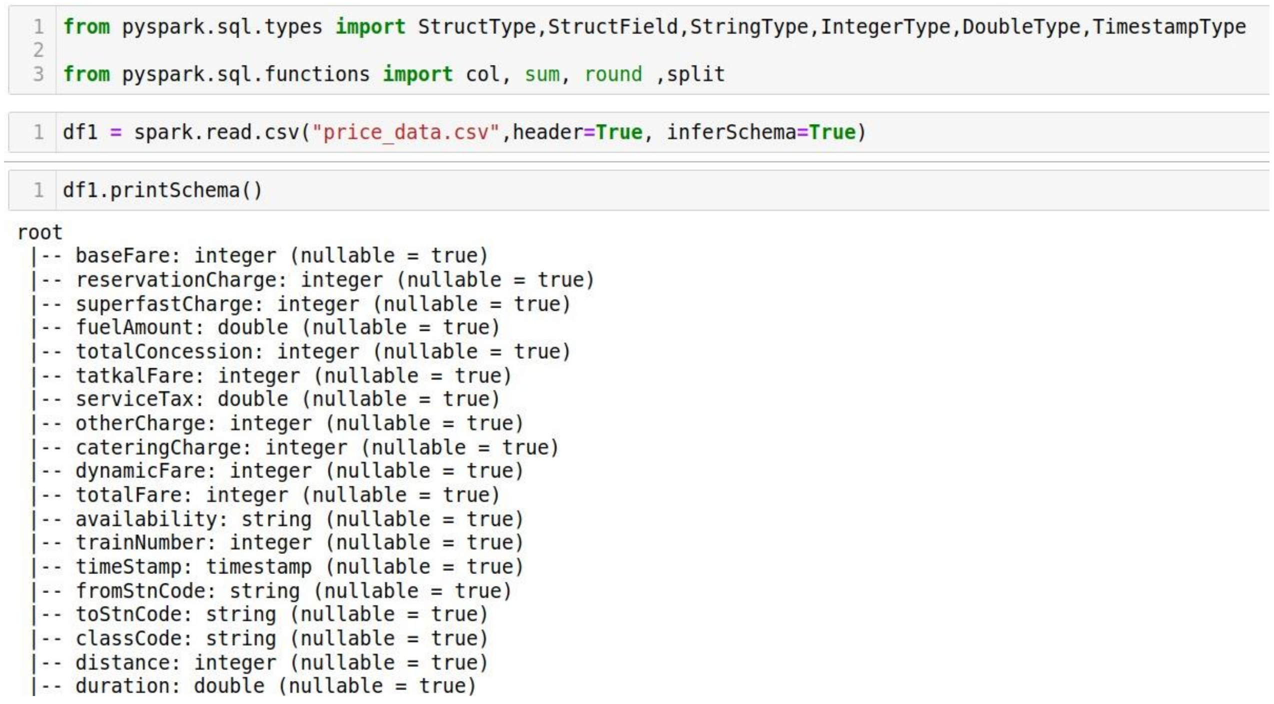
4.1 HDFS Put


```
talentum@talentum-virtual-machine:~/shared/Project$ ls
 price_data.csv   schedules.csv  'Train Fare.csv'  'Train Schedules.csv'
talentum@talentum-virtual-machine:~/shared/Project$ pwd
/home/talentum/shared/Project
talentum@talentum-virtual-machine:~/shared/Project$ hdfs dfs -put price_data.csv
```

4.2 Spark Connection


```
In [1]:
# Intialization
import os
import sys

os.environ["SPARK_HOME"] = "/home/talentum/spark"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
# In below two lines, use /usr/bin/python2.7 if you want to use Python 2
os.environ["PYSPARK_PYTHON"] = "/usr/bin/python3.6"
os.environ["PYSPARK_DRIVER_PYTHON"] = "/usr/bin/python3"
sys.path.insert(0, os.environ["PYLIB"] +"/py4j-0.10.7-src.zip")
sys.path.insert(0, os.environ["PYLIB"] +"/pyspark.zip")

# NOTE: Whichever package you want mention here.
# os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.databricks:spark-xml_2.11:0.6.0 pyspark-shell'
# os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-avro_2.11:2.4.0 pyspark-shell'
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.databricks:spark-xml_2.11:0.6.0,org.apache.spark:spark-avro_2
# os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.databricks:spark-xml_2.11:0.6.0,org.apache.spark:spark-avro

In [2]:
#Entrypoint 2.x
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Project").enableHiveSupport().getOrCreate()

# On yarn:
# spark = SparkSession.builder.appName("Spark SQL basic example").enableHiveSupport().master("yarn").getOrCreate(
# specify .master("yarn")

sc = spark.sparkContext
```

4.3 Import Dataset And Libraries



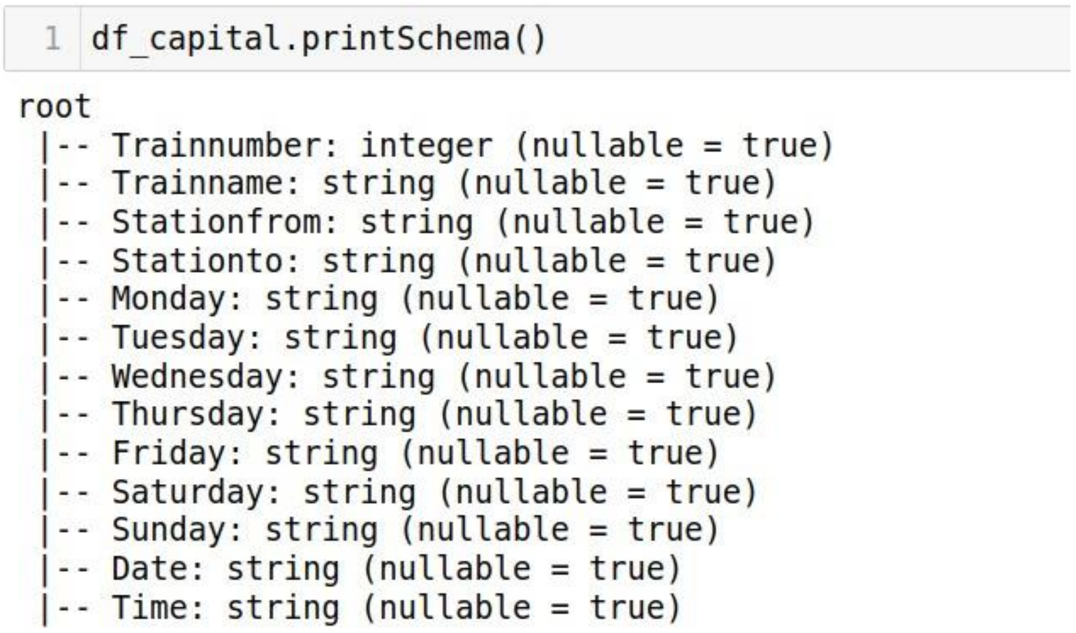
```
from pyspark.sql.types import StructType,StructField,StringType,IntegerType,DoubleType,TimestampType

from pyspark.sql.functions import col, sum, round ,split
```

```
df1 = spark.read.csv("price_data.csv",header=True, inferSchema=True)
```

```
df1.printSchema()
```

```
root
 |-- baseFare: integer (nullable = true)
 |-- reservationCharge: integer (nullable = true)
 |-- superfastCharge: integer (nullable = true)
 |-- fuelAmount: double (nullable = true)
 |-- totalConcession: integer (nullable = true)
 |-- tatkalFare: integer (nullable = true)
 |-- serviceTax: double (nullable = true)
 |-- otherCharge: integer (nullable = true)
 |-- cateringCharge: integer (nullable = true)
 |-- dynamicFare: integer (nullable = true)
 |-- totalFare: integer (nullable = true)
 |-- availability: string (nullable = true)
 |-- trainNumber: integer (nullable = true)
 |-- timeStamp: timestamp (nullable = true)
 |-- fromStnCode: string (nullable = true)
 |-- toStnCode: string (nullable = true)
 |-- classCode: string (nullable = true)
 |-- distance: integer (nullable = true)
 |-- duration: double (nullable = true)
```

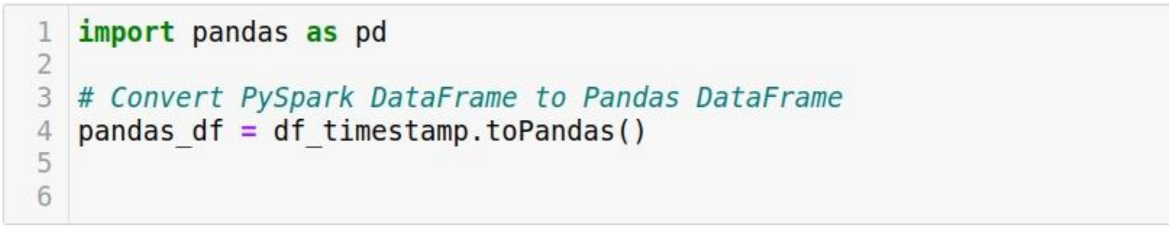
4.4 Schema



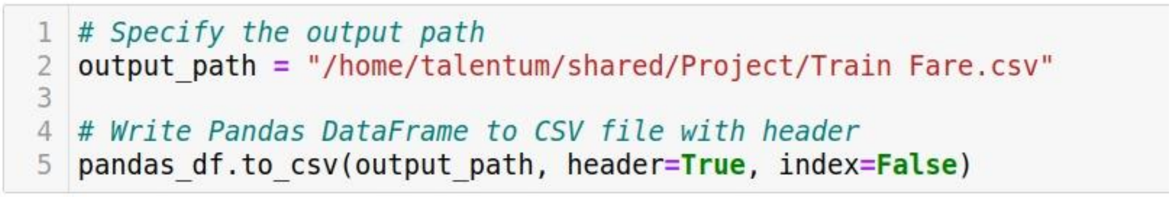
```
df_capital.printSchema()
```

```
root
 |-- Trainnumber: integer (nullable = true)
 |-- Trainname: string (nullable = true)
 |-- Stationfrom: string (nullable = true)
 |-- Stationto: string (nullable = true)
 |-- Monday: string (nullable = true)
 |-- Tuesday: string (nullable = true)
 |-- Wednesday: string (nullable = true)
 |-- Thursday: string (nullable = true)
 |-- Friday: string (nullable = true)
 |-- Saturday: string (nullable = true)
 |-- Sunday: string (nullable = true)
 |-- Date: string (nullable = true)
 |-- Time: string (nullable = true)
```

4.5 Export to CSV

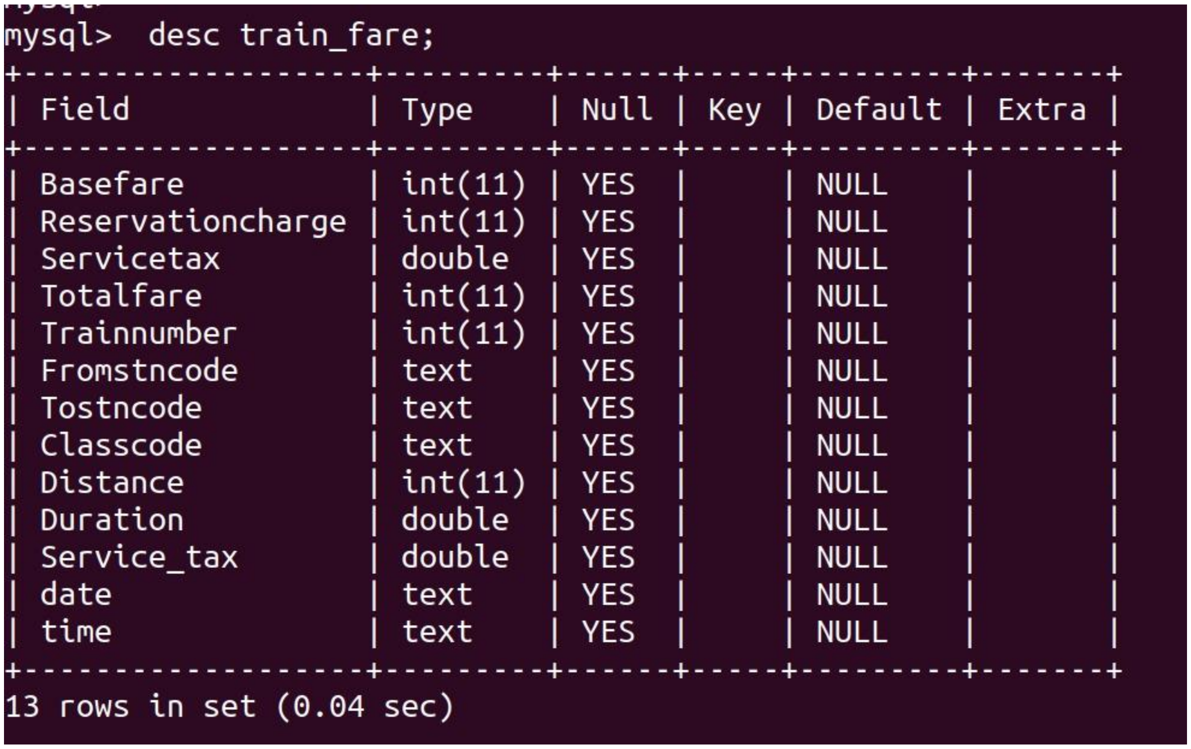


```python
import pandas as pd

# Convert PySpark DataFrame to Pandas DataFrame
pandas_df = df_timestamp.toPandas()


```



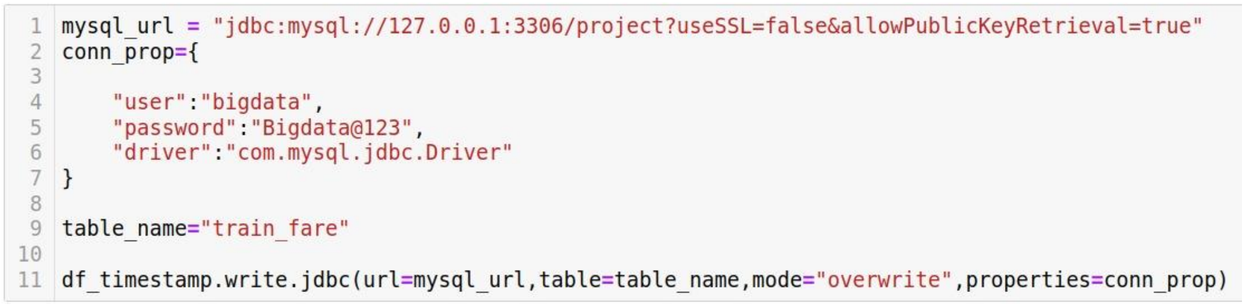
```python
# Specify the output path
output_path = "/home/talentum/shared/Project/Train Fare.csv"

# Write Pandas DataFrame to CSV file with header
pandas_df.to_csv(output_path, header=True, index=False)
```

4.6 MySQL Schema



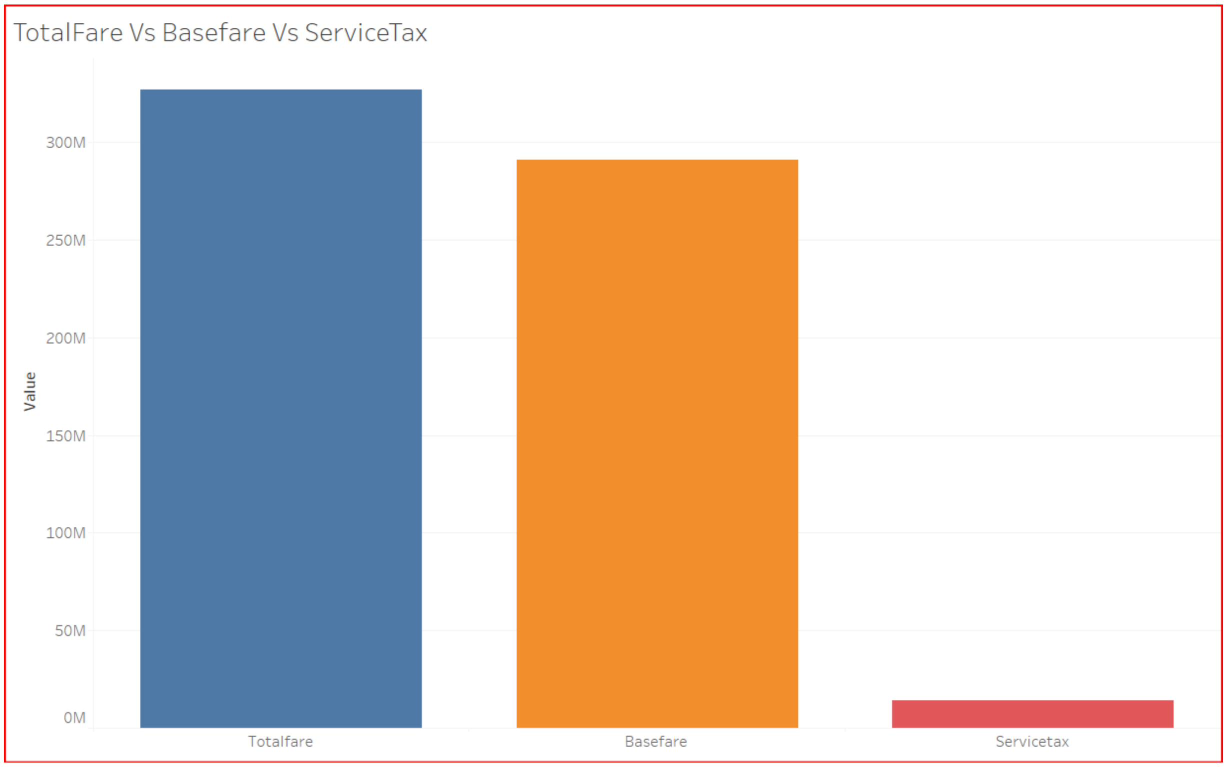
```
mysql>  desc train_fare;
+-------------------+---------+------+-----+---------+-------+
| Field             | Type    | Null | Key | Default | Extra |
+-------------------+---------+------+-----+---------+-------+
| Basefare          | int(11) | YES  |     | NULL    |       |
| Reservationcharge | int(11) | YES  |     | NULL    |       |
| Servicetax        | double  | YES  |     | NULL    |       |
| Totalfare         | int(11) | YES  |     | NULL    |       |
| Trainnumber       | int(11) | YES  |     | NULL    |       |
| Fromstncode       | text    | YES  |     | NULL    |       |
| Tostncode         | text    | YES  |     | NULL    |       |
| Classcode         | text    | YES  |     | NULL    |       |
| Distance          | int(11) | YES  |     | NULL    |       |
| Duration          | double  | YES  |     | NULL    |       |
| Service_tax       | double  | YES  |     | NULL    |       |
| date              | text    | YES  |     | NULL    |       |
| time              | text    | YES  |     | NULL    |       |
+-------------------+---------+------+-----+---------+-------+
13 rows in set (0.04 sec)
```

4.7 Data Loading into Database



```
mysql_url = "jdbc:mysql://127.0.0.1:3306/project?useSSL=false&allowPublicKeyRetrieval=true"
conn_prop={

    "user":"bigdata",
    "password":"Bigdata@123",
    "driver":"com.mysql.jdbc.Driver"
}

table_name="train_fare"

df_timestamp.write.jdbc(url=mysql_url,table=table_name,mode="overwrite",properties=conn_prop)
```

4.8 Fare Classification



5. Requirements Specification

5.1 Hardware Requirement:

- 500 GB hard drive (Minimum requirement)
- 16 GB RAM (Minimum requirement)
- PC x64-bit CPU

5.2 Software Requirement:

- Windows/Mac/Linux
- Python-3.9.1
- VS Code/Anaconda/Spyder/Pyspark
- MySQL

6. Conclusion

Through the implementation of an Extract, Transform, Load (ETL) pipeline using PySpark, we have successfully analyzed the IRCTC web-scraped data to gain valuable insights into various aspects of Indian railway transportation.

The comprehensive ETL process involved data collection from the IRCTC website, extraction of relevant information, transformation to cleanse and preprocess the data, and loading into a suitable storage system for analysis. Leveraging the power of PySpark, we conducted in-depth analysis to uncover patterns, trends, and correlations related to train schedules, ticket bookings, and passenger traffic.

The process of loading the transformed data into MySQL involves establishing a connection between PySpark and MySQL, defining the schema of the MySQL table to match the structure of the processed data, and executing the data insertion operation. Leveraging PySpark's capabilities, we efficiently write the processed data into the MySQL table, maintaining data integrity and consistency throughout the process.

7. Future Scope

The project opens up several avenues for future exploration and enhancement. One potential direction is to incorporate advanced analytics techniques, such as machine learning and predictive modeling, to forecast demand, optimize resource allocation, and improve service reliability.

Additionally, expanding the scope of data collection to include external factors such as weather conditions, socio-economic indicators, and tourism trends could provide deeper insights into the dynamics of train services and passenger behavior. Furthermore, integrating real-time data streams and implementing data governance and security measures will ensure the scalability, reliability, and integrity of the analysis framework. Overall, the project lays a solid foundation for ongoing research and innovation in the field of Indian railway transportation analytics, with opportunities for collaboration and partnership with industry stakeholders and academic institutions.

8. References

1. **https://spark.apache.org/docs/latest/api/python/index.html**
2. **https://dev.mysql.com/doc/**
3. **https://www.geeksforgeeks.org/etl-process-in-data-warehouse/**
4. **https://www.python.org/doc/**
5. **https://spark.apache.org/**
6. **https://hadoop.apache.org/docs/current/**