Project Report

Airline Dataset

Group 2

Ritika Aggarwal

Aiswarya Baby

Sevim Shafizadegan

Dulanjani Rajapakshe

a. **Problem Definition**:

The airline industry is complex, and effective data analysis is key to understanding flight performance, passenger behavior, and optimizing operational efficiency. With increasing passenger volumes and evolving travel behaviors, airlines and airport authorities face significant challenges in delivering seamless operations, enhancing customer experiences, and responding to dynamic market demands.

The provided dataset, which includes passenger details, flight routes, and flight statuses, offers an opportunity to derive meaningful insights.

The primary objectives of this project are:

- To analyze flight performance, including delays and cancellations.
- To understand the relationship between passenger demographics and travel behavior.
- To identify key factors contributing to flight delays and cancellations.

These insights will help airlines optimize their operations, improve customer satisfaction, and effectively manage resources.

b. **Data Description** :

i. **Attributes Description :**

The Airline dataset can be downloaded from the following open-source link:

https://www.kaggle.com/datasets/iamsouravbanerjee/airline-dataset

Key attributes considered for analysis include:

- **Passenger ID**: Unique identifier for each passenger.
- **Age**, **Gender**, **Nationality**: Used to analyze passenger demographics and travel behavior.
- **Airport Name** and **Country Code**: Details about the boarding airports, used to understand geographic influences on flight performance.
- **Flight Status**: Status of the flight (on-time, delayed, canceled), used to assess operational efficiency.

- **Departure Date** and **Arrival Airport**: Used to identify trends in flight performance, such as seasonal delays and cancellation rates.

ii. **Statistics of the Data:**

- **Apache Nifi** : Apache NiFi was employed to automate the extraction, transformation, and loading (ETL) of data from the source (CSV files) to the storage layer. NiFi enabled us to handle data, ensuring scalability and data consistency while providing a user-friendly interface for designing data flows.
- **HDFS (Hadoop Distributed File System)**: To store the ingested data, we used HDFS, a distributed file system designed for big data applications. HDFS allowed us to manage large datasets effectively by distributing data across multiple nodes, ensuring high fault tolerance and availability.
- **Apache Spark :** Apache Spark was leveraged for data processing. Its distributed computing framework enabled us to clean, transform, and analyze large datasets efficiently. Spark's in-memory processing capability significantly reduced processing time compared to traditional methods.
- **Spark SQL**: Spark SQL was utilized to query the processed data using an SQL-like syntax. It provided an intuitive and flexible way to extract meaningful insights and aggregate results required for analysis.
- **Elasticsearch and Kibana:**
  - **Elasticsearch**: This tool was used to index the processed data, enabling fast searches and analytics.
  - **Kibana**: Kibana, integrated with Elasticsearch, was used for data visualization. It allowed us to create interactive dashboards and charts, presenting data insights in an intuitive and user-friendly manner.

c. **Work Distribution** :

- **Ritika Aggarwal** : HDFS, Analysis with Spark, Data Visualization with Elastic search and Kibana

- **Aiswarya Baby** : HDFS, ETL, Analysis with Spark, Prepared presentation PPT.
- **Sevim Shaffizadegan** : Data Ingestion using Apache Nifi, HDFS, Analysis with Spark.
- **Dulanjani Rajapakshe** : Analysis with Spark, Data Visualization with Kibana.

d. Solution Description :

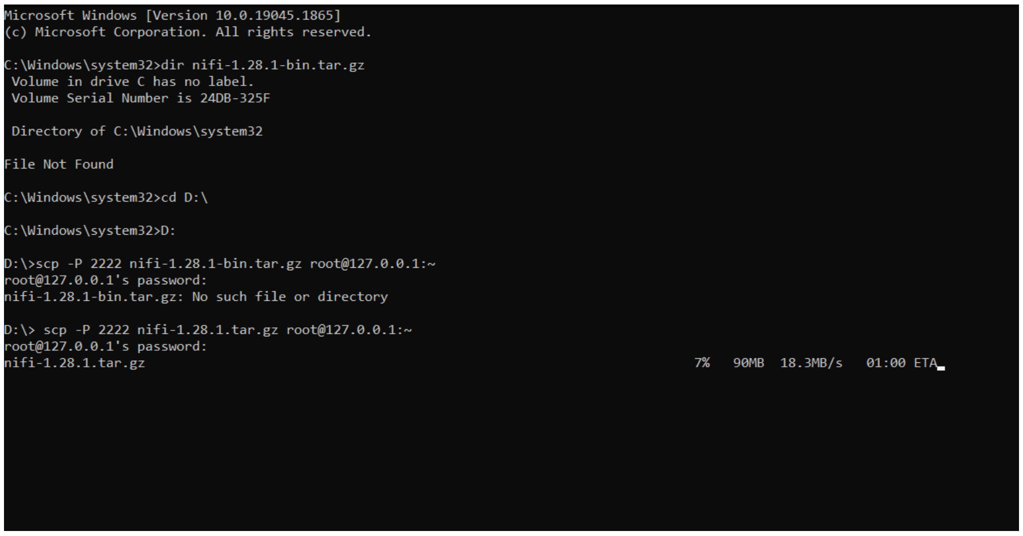
To analyze the airline dataset effectively, we used a big data architecture based on the following tools:

i. **Data Ingestion Layer : Apache Nifi -** One of the most common ways to collect data is through online sources, such as APIs and web servers. We chose NiFi for this task due to its ability to handle live data streams, process large datasets efficiently, and provide a robust UI for managing workflows and error handling.

Implementation Steps :

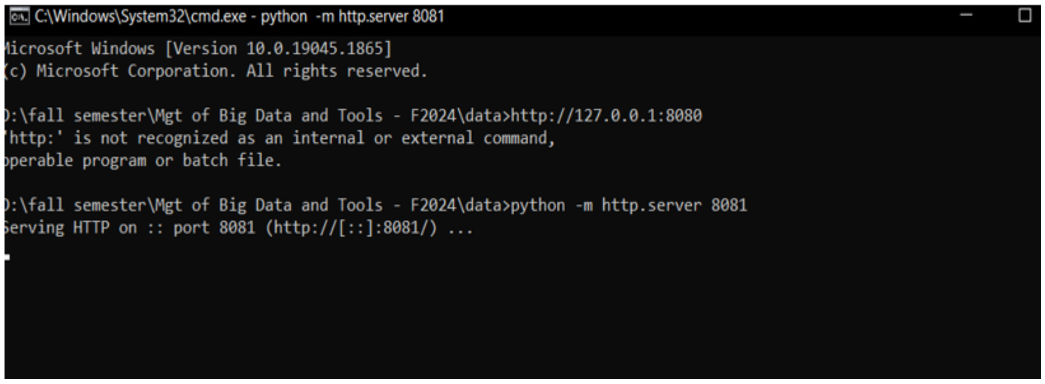
1. **Setting Up the Nifi**
   - We downloaded and installed Apache NiFi into a sandbox environment. Installing NiFi in the sandbox was necessary to avoid potential security errors and ensure seamless integration with the local Hadoop Distributed File System (HDFS).
   - The NiFi installation file was converted into a tar.gz compressed format and imported into the sandbox environment for setup.
2. **Configuring the Dataflow**
   - The objective was to move a file stored locally on my system to a local server and subsequently transfer it to the HDFS environment.
   - Using NiFi, we configured the GetHTTP processor to fetch the data from the local server via HTTPS.
   - The data was then routed through a series of processors, including the PutHDFS processor, to store it in the HDFS environment.

Installing Nifi into sandbox environment



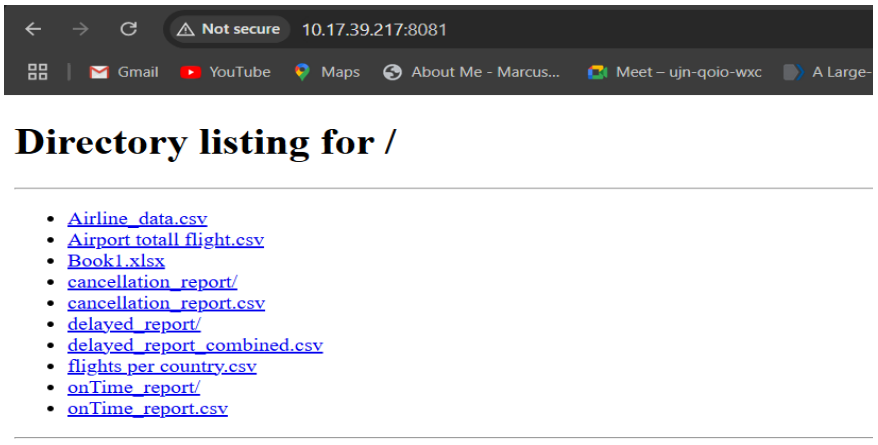
```
Microsoft Windows [Version 10.0.19045.1865]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>dir nifi-1.28.1-bin.tar.gz
 Volume in drive C has no label.
 Volume Serial Number is 24DB-325F

 Directory of C:\Windows\system32

File Not Found

C:\Windows\system32>cd D:\

C:\Windows\system32>D:

D:\>scp -P 2222 nifi-1.28.1-bin.tar.gz root@127.0.0.1:~
root@127.0.0.1's password:
nifi-1.28.1-bin.tar.gz: No such file or directory

D:\> scp -P 2222 nifi-1.28.1.tar.gz root@127.0.0.1:~
root@127.0.0.1's password:
nifi-1.28.1.tar.gz                                                        7%   90MB  18.3MB/s   01:00 ETA
```

Setting up the localhost on the folder



```
C:\Windows\System32\cmd.exe - python  -m http.server 8081
Microsoft Windows [Version 10.0.19045.1865]
(c) Microsoft Corporation. All rights reserved.

D:\fall semester\Mgt of Big Data and Tools - F2024\data>http://127.0.0.1:8080
'http:' is not recognized as an internal or external command,
operable program or batch file.

D:\fall semester\Mgt of Big Data and Tools - F2024\data>python -m http.server 8081
Serving HTTP on :: port 8081 (http://[::]:8081/) ...
```

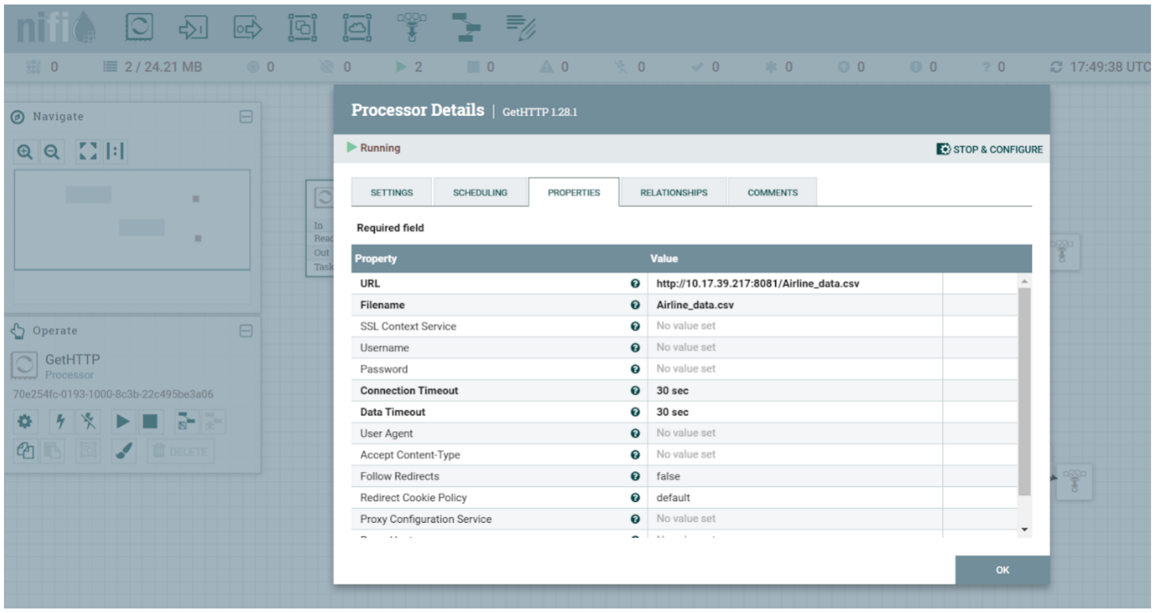
Listing some files in the directory:



Directory listing for /

- Airline_data.csv
- Airport totall flight.csv
- Book1.xlsx
- cancellation_report/
- cancellation_report.csv
- delayed_report/
- delayed_report_combined.csv
- flights per country.csv
- onTime_report/
- onTime_report.csv

Starting NIFI - In the URL we put the url of the running localhost that will lead to downloading the Airline data file.

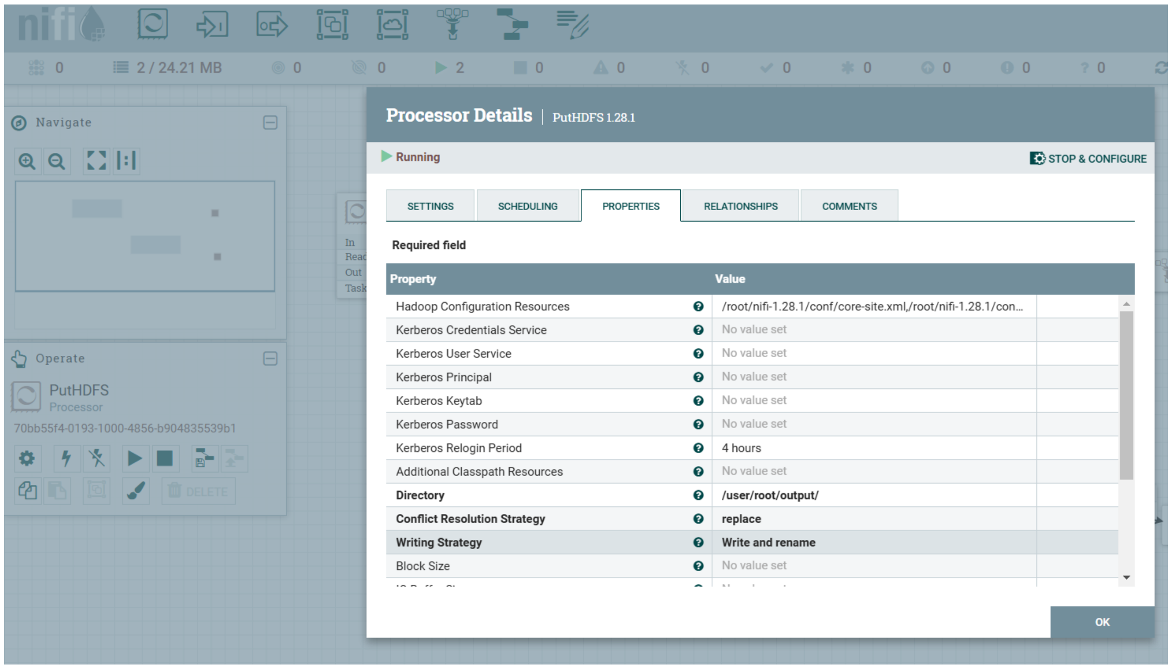


Required field

Property	Value
URL	http://10.17.39.217:8081/Airline_data.csv
Filename	Airline_data.csv
SSL Context Service	No value set
Username	No value set
Password	No value set
Connection Timeout	30 sec
Data Timeout	30 sec
User Agent	No value set
Accept Content-Type	No value set
Follow Redirects	false
Redirect Cookie Policy	default
Proxy Configuration Service	No value set

Downloading the sandbox conf and putting in in PutHDFS configuration

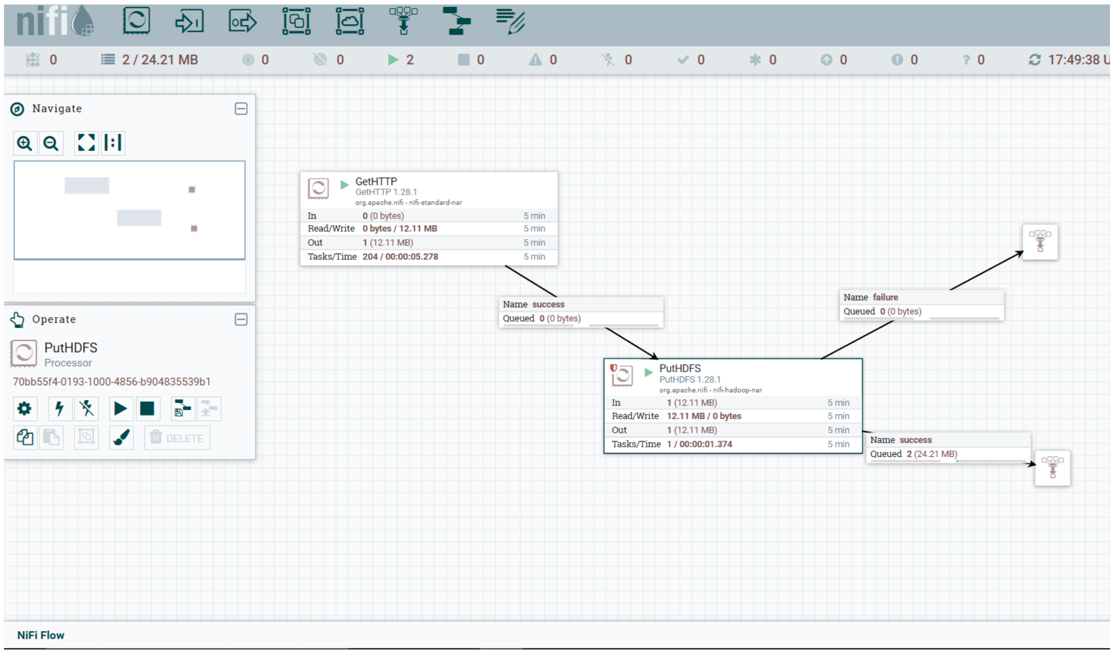
In that way the processor will be linked to out sandbox



Required field

Property	Value
Hadoop Configuration Resources	/root/nifi-1.28.1/conf/core-site.xml,/root/nifi-1.28.1/con...
Kerberos Credentials Service	No value set
Kerberos User Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
Kerberos Password	No value set
Kerberos Relogin Period	4 hours
Additional Classpath Resources	No value set
Directory	/user/root/output/
Conflict Resolution Strategy	replace
Writing Strategy	Write and rename
Block Size	No value set

The Final result



GetHTTP
GetHTTP 1.28.1
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 12.11 MB	5 min
Out	1 (12.11 MB)	5 min
Tasks/Time	204 / 00:00:05.278	5 min

Name success
Queued 0 (0 bytes)

Name failure
Queued 0 (0 bytes)

PutHDFS
PutHDFS 1.28.1
org.apache.nifi - nifi-hadoop-nar

In	1 (12.11 MB)	5 min
Read/Write	12.11 MB / 0 bytes	5 min
Out	1 (12.11 MB)	5 min
Tasks/Time	1 / 00:00:01.374	5 min

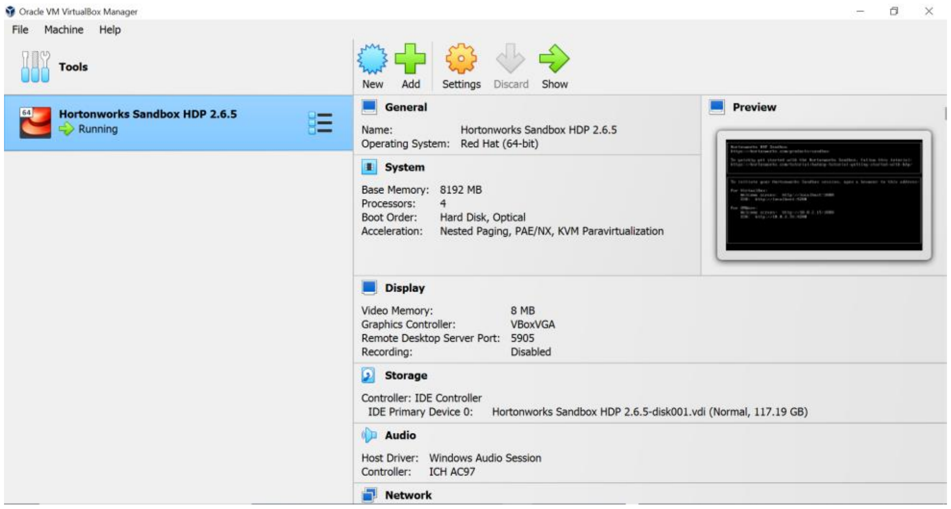
Name success
Queued 2 (24.21 MB)

```
sandbox-hdp login: root
root@sandbox-hdp.hortonworks.com's password:
Last login: Thu Nov 28 17:27:31 2024 from 172.18.0.2
[root@sandbox-hdp ~]# hadoop fs -ls /user/root/output/
Found 5 items
-rw-r--r--   1 root hdfs          1 2024-11-28 17:38 /user/root/output/1dcd48c6-d681-4a77-8940-965a255dbc8e
-rw-r--r--   1 root hdfs          0 2024-11-28 17:37 /user/root/output/47428ad2-4156-45eb-bb6a-da370201fe2a
-rw-r--r--   1 root hdfs          1 2024-11-28 17:39 /user/root/output/94af44f1-f04d-44d5-86a7-5098483bf0c1
-rw-r--r--   1 root hdfs   12687769 2024-11-28 17:40 /user/root/output/95fe6cda-0112-44b7-9f73-97ef96274819
-rw-r--r--   1 root hdfs   12699818 2024-11-28 17:49 /user/root/output/Airline_data.csv
[root@sandbox-hdp ~]#
```

ii. **Data Storage Layer : HDFS (Hadoop Distributed File System) -** Hadoop was a critical component of our project's architecture, particularly in the Data Storage Layer, due to its ability to manage and process large-scale datasets effectively. Below are the primary reasons for selecting Hadoop:

- Distributed Storage with HDFS
- Cost- Effective solution
- Compatibility with spark
- High performance for big data
- Reliable data management
- Flexibility in data storage

Starting the virtual box



Hortonworks Sandbox HDP 2.6.5
Running

General

Name: Hortonworks Sandbox HDP 2.6.5
Operating System: Red Hat (64-bit)

System

Base Memory: 8192 MB
Processors: 4
Boot Order: Hard Disk, Optical
Acceleration: Nested Paging, PAE/NX, KVM Paravirtualization

Display

Video Memory: 8 MB
Graphics Controller: VBoxVGA
Remote Desktop Server Port: 5905
Recording: Disabled

Storage

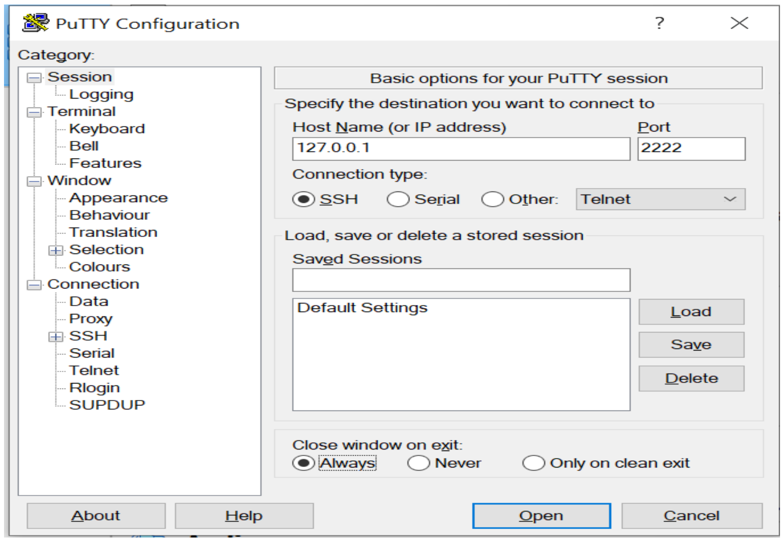
Controller: IDE Controller
IDE Primary Device 0: Hortonworks Sandbox HDP 2.6.5-disk001.vdi (Normal, 117.19 GB)

Audio

Host Driver: Windows Audio Session
Controller: ICH AC97

Network

Configuring putty as our medium of working with Hadoop



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 127.0.0.1
Port: 2222

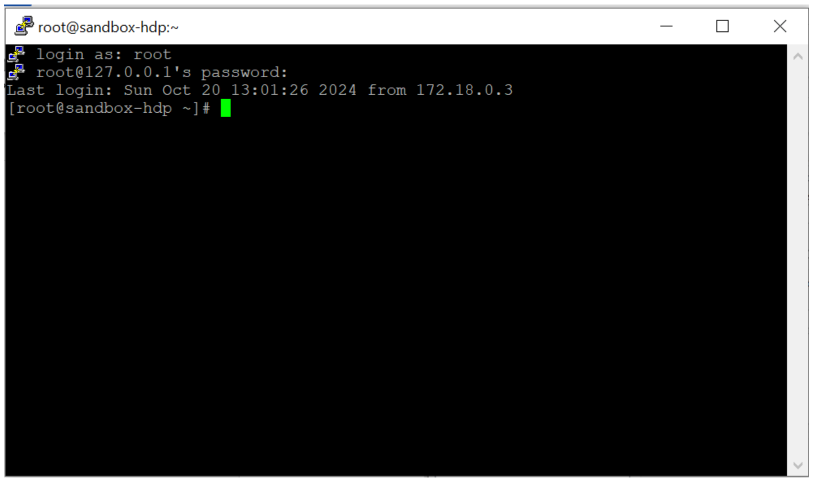
Connection type: [✓ SSH] Serial Other: Telnet

Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: [✓ Always] Never Only on clean exit



```
login as: root
root@127.0.0.1's password:
Last login: Sun Oct 20 13:01:26 2024 from 172.18.0.3
[root@sandbox-hdp ~]#
```

iii. Data Processing and Query Layer : Apache Spark (Spark SQL) –

We chose Spark SQL as the primary tool for querying and analyzing the processed data due to the following advantages:

- **Native support for structured data**

  Since our dataset was stored in CSV files, Spark SQL's ability to work directly with structured and semi-structured data in tabular format was highly advantageous.

- **Simplified Querying with SQL**

  Spark SQL allowed us to leverage our existing knowledge of SQL for data analysis. This eliminated the need for additional coding expertise, enabling the team to focus on querying and deriving insights.

- **High Performance and Scalability**

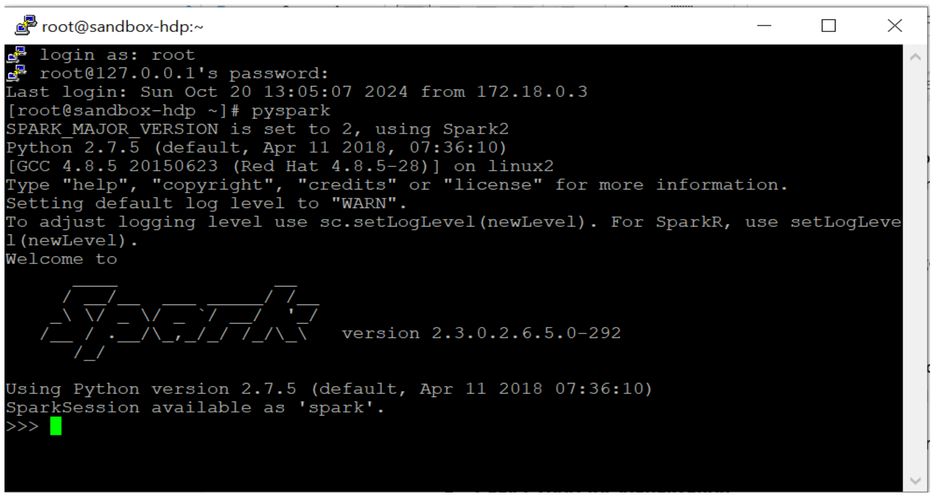
  Its distributed computing framework allowed us to handle and query large datasets effectively, making it ideal for our project's requirements.

- **Integration with Data Pipeline**

  Spark SQL seamlessly integrated with other tools in our pipeline, such as **HDFS** for data storage and **Kibana** for visualization.

- **Easy Export for Visualization**

  After deriving insights through SQL queries, the results were exported in a tabular structure for further visualization. This ensured a smooth workflow from data querying to creating dashboards in **Kibana**.



root@sandbox-hdp:~

```
login as: root
root@127.0.0.1's password:
Last login: Sun Oct 20 13:05:07 2024 from 172.18.0.3
[root@sandbox-hdp ~]# pyspark
SPARK_MAJOR_VERSION is set to 2, using Spark2
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.0.2.6.5.0-292
      /_/

Using Python version 2.7.5 (default, Apr 11 2018 07:36:10)
SparkSession available as 'spark'.
>>>
```

iv. **Data Visualization (Elastic Search and Kibana)** - Kibana was selected as the primary tool for visualizing the processed data in our project. Its powerful visualization capabilities and seamless integration with Elasticsearch made it an ideal choice for creating dynamic and insightful dashboards. Below are the key reasons for choosing Kibana:

- **Seamless Integration with Elasticsearch**

  Kibana integrates directly with **Elasticsearch**, where the processed result tables were indexed after being exported. This ensured real-time access to data and allowed us to create visualizations without additional data preparation.

- **Wide Range of Visualization Options**

  Kibana offers a variety of visualization types, including bar charts, pie charts, line graphs, heatmaps, and data tables. These options allowed us to present insights in a visually appealing and easy-to-understand format.

- **User-Friendly Interface**

  Kibana's intuitive and user-friendly interface made it easy to design and customize visualizations. The drag-and-drop functionality required minimal technical expertise, allowing the team to focus on interpreting the data.

- **Real-Time Visualization**

  Kibana supports real-time data visualization, which was crucial for analyzing time-sensitive trends and patterns in our dataset. As new insights were indexed in Elasticsearch, dashboards updated automatically.

- **Open-Source and Cost-Effective**

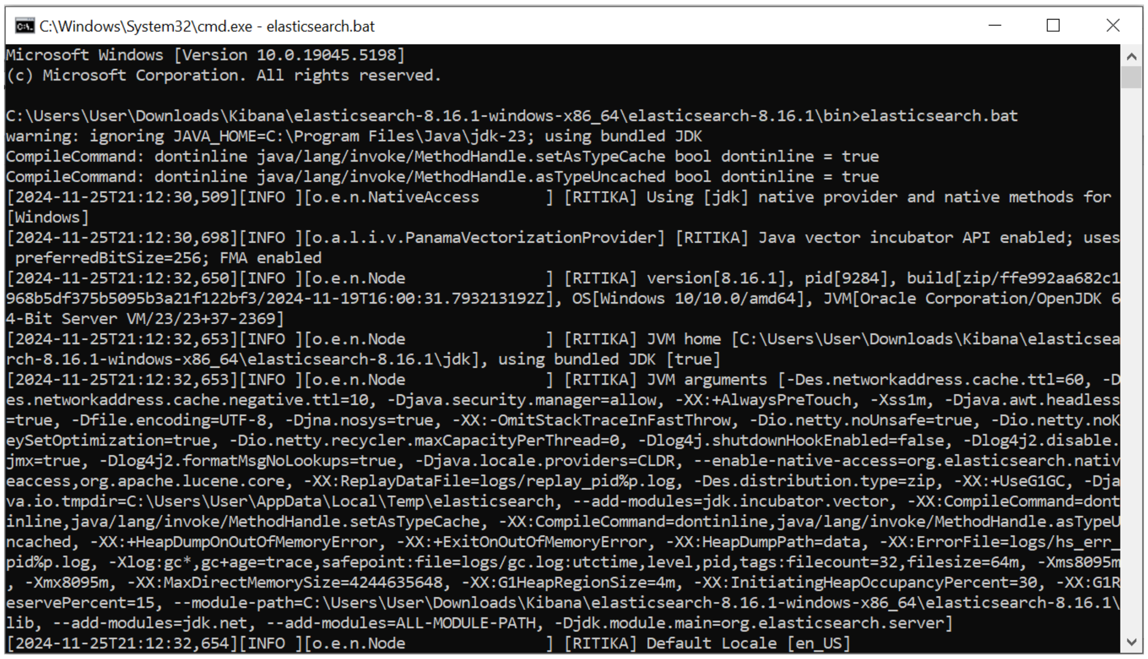
  Kibana is open-source, making it a cost-effective solution for building data visualizations. It provided enterprise-grade features without additional licensing costs.

- **Customizable Dashboards**

Kibana allowed us to create customizable dashboards that consolidated multiple visualizations. This made it easier to present the overall insights from our project in a single, comprehensive view.

Download Elastic search and Kibana:

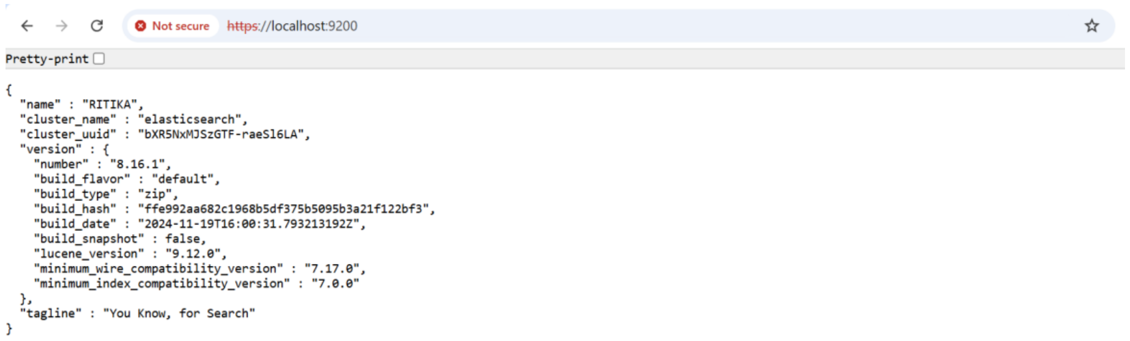
First downlad the Elastic Search Zip file. Then unzip and configure it.



```
Microsoft Windows [Version 10.0.19045.5198]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User\Downloads\Kibana\elasticsearch-8.16.1-windows-x86_64\elasticsearch-8.16.1\bin>elasticsearch.bat
warning: ignoring JAVA_HOME=C:\Program Files\Java\jdk-23; using bundled JDK
CompileCommand: dontinline java/lang/invoke/MethodHandle.setAsTypeCache bool dontinline = true
CompileCommand: dontinline java/lang/invoke/MethodHandle.asTypeUncached bool dontinline = true
[2024-11-25T21:12:30,509][INFO ][o.e.n.NativeAccess       ] [RITIKA] Using [jdk] native provider and native methods for
[Windows]
[2024-11-25T21:12:30,698][INFO ][o.a.l.i.v.PanamaVectorizationProvider] [RITIKA] Java vector incubator API enabled; uses
 preferredBitSize=256; FMA enabled
[2024-11-25T21:12:32,650][INFO ][o.e.n.Node               ] [RITIKA] version[8.16.1], pid[9284], build[zip/ffe992aa682c1
968b5df375b5095b3a21f122bf3/2024-11-19T16:00:31.793213192Z], OS[Windows 10/10.0/amd64], JVM[Oracle Corporation/OpenJDK 6
4-Bit Server VM/23/23+37-2369]
[2024-11-25T21:12:32,653][INFO ][o.e.n.Node               ] [RITIKA] JVM home [C:\Users\User\Downloads\Kibana\elasticsea
rch-8.16.1-windows-x86_64\elasticsearch-8.16.1\jdk], using bundled JDK [true]
[2024-11-25T21:12:32,653][INFO ][o.e.n.Node               ] [RITIKA] JVM arguments [-Des.networkaddress.cache.ttl=60, -D
es.networkaddress.cache.negative.ttl=10, -Djava.security.manager=allow, -XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless
=true, -Dfile.encoding=UTF-8, -Djna.nosys=true, -XX:-OmitStackTraceInFastThrow, -Dio.netty.noUnsafe=true, -Dio.netty.noK
eySetOptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0, -Dlog4j.shutdownHookEnabled=false, -Dlog4j2.disable.
jmx=true, -Dlog4j2.formatMsgNoLookups=true, -Djava.locale.providers=CLDR, --enable-native-access=org.elasticsearch.nativ
eaccess,org.apache.lucene.core, -XX:ReplayDataFile=logs/replay_pid%p.log, -Des.distribution.type=zip, -XX:+UseG1GC, -Dja
va.io.tmpdir=C:\Users\User\AppData\Local\Temp\elasticsearch, --add-modules=jdk.incubator.vector, -XX:CompileCommand=dont
inline,java/lang/invoke/MethodHandle.setAsTypeCache, -XX:CompileCommand=dontinline,java/lang/invoke/MethodHandle.asTypeU
ncached, -XX:+HeapDumpOnOutOfMemoryError, -XX:+ExitOnOutOfMemoryError, -XX:HeapDumpPath=data, -XX:ErrorFile=logs/hs_err_
pid%p.log, -Xlog:gc*,gc+age=trace,safepoint:file=logs/gc.log:utctime,level,pid,tags:filecount=32,filesize=64m, -Xms8095m
, -Xmx8095m, -XX:MaxDirectMemorySize=4244635648, -XX:G1HeapRegionSize=4m, -XX:InitiatingHeapOccupancyPercent=30, -XX:G1R
eservePercent=15, --module-path=C:\Users\User\Downloads\Kibana\elasticsearch-8.16.1-windows-x86_64\elasticsearch-8.16.1\
lib, --add-modules=jdk.net, --add-modules=ALL-MODULE-PATH, -Djdk.module.main=org.elasticsearch.server]
[2024-11-25T21:12:32,654][INFO ][o.e.n.Node               ] [RITIKA] Default Locale [en_US]
```

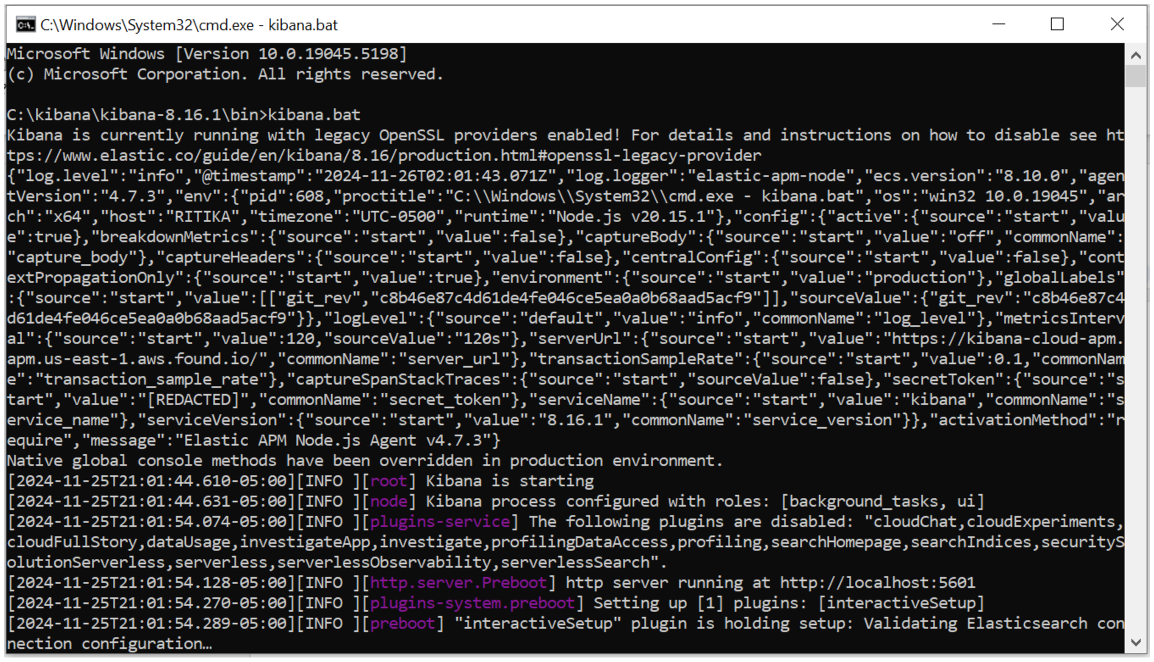
After doing the configuration, check whether Elastic search is installed properly or not.

Run : https://localhost:9200



```
{
  "name" : "RITIKA",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "bXR5NxMJSzGTF-raeSl6LA",
  "version" : {
    "number" : "8.16.1",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "ffe992aa682c1968b5df375b5095b3a21f122bf3",
    "build_date" : "2024-11-19T16:00:31.793213192Z",
    "build_snapshot" : false,
    "lucene_version" : "9.12.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

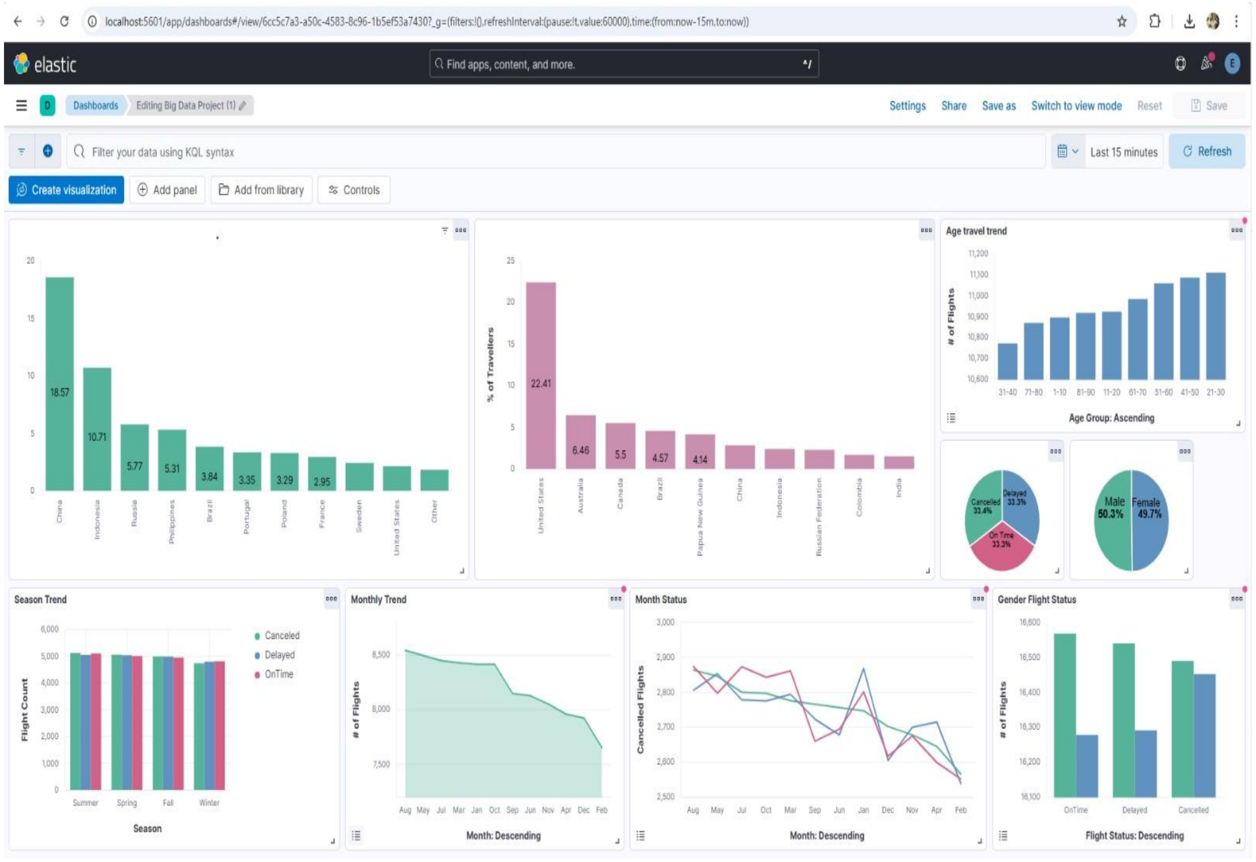
Now, download Kibana's zip file and unzip it and run its bat file



```
C:\Windows\System32\cmd.exe - kibana.bat
Microsoft Windows [Version 10.0.19045.5198]
(c) Microsoft Corporation. All rights reserved.

C:\kibana\kibana-8.16.1\bin>kibana.bat
Kibana is currently running with legacy OpenSSL providers enabled! For details and instructions on how to disable see ht
tps://www.elastic.co/guide/en/kibana/8.16/production.html#openssl-legacy-provider
{"log.level":"info","@timestamp":"2024-11-26T02:01:43.071Z","log.logger":"elastic-apm-node","ecs.version":"8.10.0","agen
tVersion":"4.7.3","env":{"pid":608,"proctitle":"C:\\Windows\\System32\\cmd.exe - kibana.bat","os":"win32 10.0.19045","ar
ch":"x64","host":"RITIKA","timezone":"UTC-0500","runtime":"Node.js v20.15.1"},"config":{"active":{"source":"start","valu
e":true},"breakdownMetrics":{"source":"start","value":false},"captureBody":{"source":"start","value":"off","commonName":
"capture_body"},"captureHeaders":{"source":"start","value":false},"centralConfig":{"source":"start","value":false},"cont
extPropagationOnly":{"source":"start","value":true},"environment":{"source":"start","value":"production"},"globalLabels"
:{"source":"start","value":[["git_rev","c8b46e87c4d61de4fe046ce5ea0a0b68aad5acf9"]],"sourceValue":{"git_rev":"c8b46e87c4
d61de4fe046ce5ea0a0b68aad5acf9"}},"logLevel":{"source":"default","value":"info","commonName":"log_level"},"metricsInterv
al":{"source":"start","value":120,"sourceValue":"120s"},"serverUrl":{"source":"start","value":"https://kibana-cloud-apm.
apm.us-east-1.aws.found.io/","commonName":"server_url"},"transactionSampleRate":{"source":"start","value":0.1,"commonNam
e":"transaction_sample_rate"},"captureSpanStackTraces":{"source":"start","sourceValue":false},"secretToken":{"source":"s
tart","value":"[REDACTED]","commonName":"secret_token"},"serviceName":{"source":"start","value":"kibana","commonName":"s
ervice_name"},"serviceVersion":{"source":"start","value":"8.16.1","commonName":"service_version"}},"activationMethod":"r
equire","message":"Elastic APM Node.js Agent v4.7.3"}
Native global console methods have been overridden in production environment.
[2024-11-25T21:01:44.610-05:00][INFO ][root] Kibana is starting
[2024-11-25T21:01:44.631-05:00][INFO ][node] Kibana process configured with roles: [background_tasks, ui]
[2024-11-25T21:01:54.074-05:00][INFO ][plugins-service] The following plugins are disabled: "cloudChat,cloudExperiments,
cloudFullStory,dataUsage,investigateApp,investigate,profilingDataAccess,profiling,searchHomepage,searchIndices,securityS
olutionServerless,serverless,serverlessObservability,serverlessSearch".
[2024-11-25T21:01:54.128-05:00][INFO ][http.server.Preboot] http server running at http://localhost:5601
[2024-11-25T21:01:54.270-05:00][INFO ][plugins-system.preboot] Setting up [1] plugins: [interactiveSetup]
[2024-11-25T21:01:54.289-05:00][INFO ][preboot] "interactiveSetup" plugin is holding setup: Validating Elasticsearch con
nection configuration…
```

When both elastic search and Kibana are running open : localhost:5601.

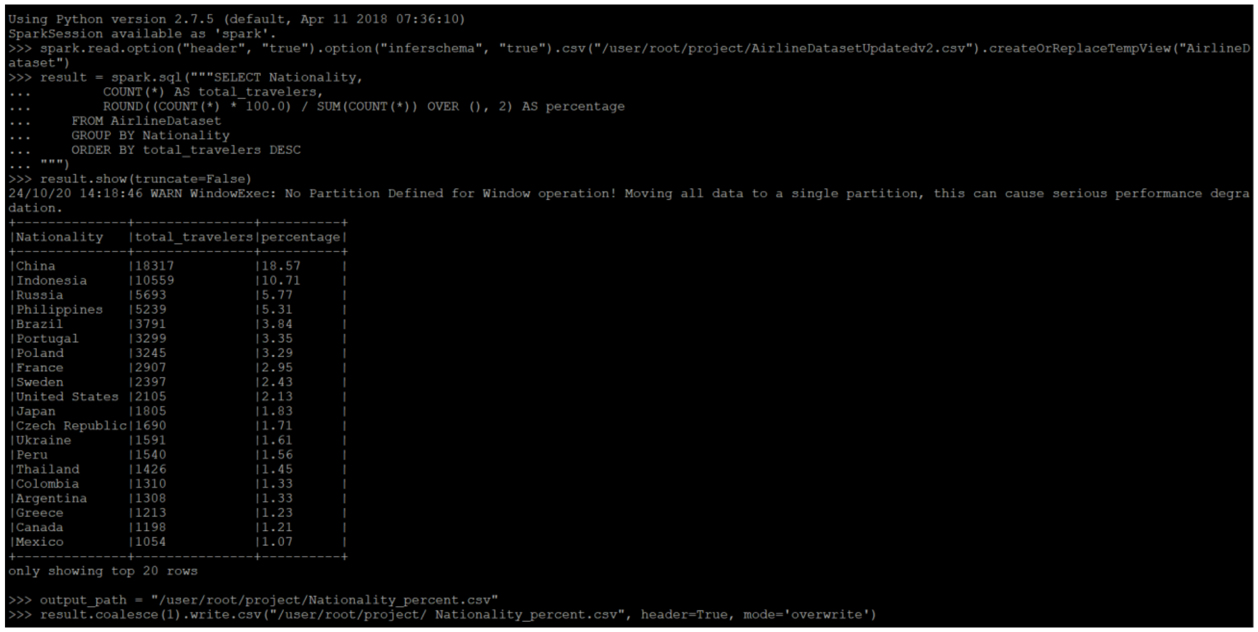
This will open the Kibana elastic search dashboard and then we can start working.



e. Describing 6 insights gathered from data:

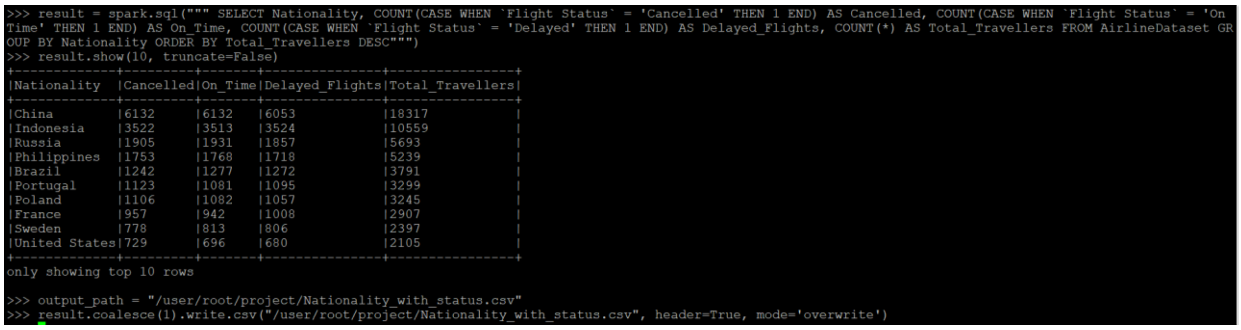
1. Which nationalities contribute the most to travel demand.

The insight shows that China has the highest percentage of airline passengers (18.57%), followed by Indonesia (10.71%) and Russia (5.77%). This highlights key markets for airlines to prioritize, focusing on high-demand routes connecting these countries. Targeting these regions can optimize flight frequencies and enhance revenue opportunities.



```
Using Python version 2.7.5 (default, Apr 11 2018 07:36:10)
SparkSession available as 'spark'.
>>> spark.read.option("header", "true").option("inferschema", "true").csv("/user/root/project/AirlineDatasetUpdatedv2.csv").createOrReplaceTempView("AirlineD
ataset")
>>> result = spark.sql("""SELECT Nationality,
...         COUNT(*) AS total_travelers,
...         ROUND((COUNT(*) * 100.0) / SUM(COUNT(*)) OVER (), 2) AS percentage
...     FROM AirlineDataset
...     GROUP BY Nationality
...     ORDER BY total_travelers DESC
... """)
>>> result.show(truncate=False)
24/10/20 14:18:46 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degra
dation.
+--------------+---------------+----------+
|Nationality   |total_travelers|percentage|
+--------------+---------------+----------+
|China         |18317          |18.57     |
|Indonesia     |10559          |10.71     |
|Russia        |5693           |5.77      |
|Philippines   |5239           |5.31      |
|Brazil        |3791           |3.84      |
|Portugal      |3299           |3.35      |
|Poland        |3245           |3.29      |
|France        |2907           |2.95      |
|Sweden        |2397           |2.43      |
|United States |2105           |2.13      |
|Japan         |1805           |1.83      |
|Czech Republic|1690           |1.71      |
|Ukraine       |1591           |1.61      |
|Peru          |1540           |1.56      |
|Thailand      |1426           |1.45      |
|Colombia      |1310           |1.33      |
|Argentina     |1308           |1.33      |
|Greece        |1213           |1.23      |
|Canada        |1198           |1.21      |
|Mexico        |1054           |1.07      |
+--------------+---------------+----------+
only showing top 20 rows

>>> output_path = "/user/root/project/Nationality_percent.csv"
>>> result.coalesce(1).write.csv("/user/root/project/ Nationality_percent.csv", header=True, mode='overwrite')
```

Nationality with status

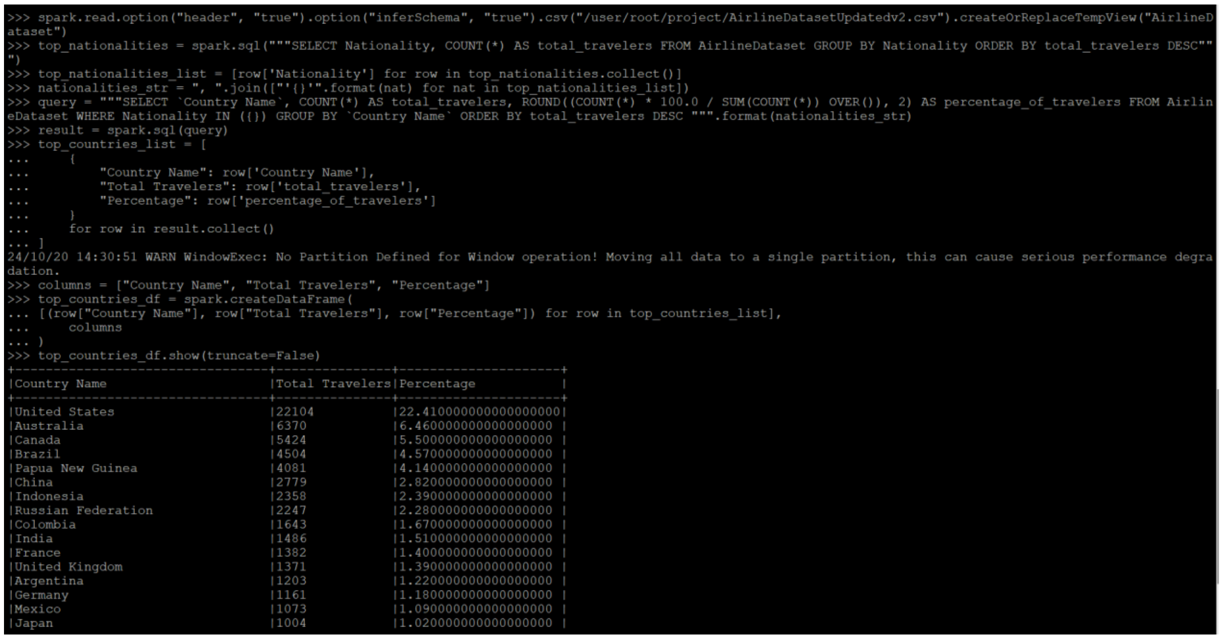


```
>>> result = spark.sql(""" SELECT Nationality, COUNT(CASE WHEN `Flight Status` = 'Cancelled' THEN 1 END) AS Cancelled, COUNT(CASE WHEN `Flight Status` = 'On
Time' THEN 1 END) AS On_Time, COUNT(CASE WHEN `Flight Status` = 'Delayed' THEN 1 END) AS Delayed_Flights, COUNT(*) AS Total_Travellers FROM AirlineDataset GR
OUP BY Nationality ORDER BY Total_Travellers DESC""")
>>> result.show(10, truncate=False)
+-------------+---------+-------+---------------+----------------+
|Nationality  |Cancelled|On_Time|Delayed_Flights|Total_Travellers|
+-------------+---------+-------+---------------+----------------+
|China        |6132     |6132   |6053           |18317           |
|Indonesia    |3522     |3513   |3524           |10559           |
|Russia       |1905     |1931   |1857           |5693            |
|Philippines  |1753     |1768   |1718           |5239            |
|Brazil       |1242     |1277   |1272           |3791            |
|Portugal     |1123     |1081   |1095           |3299            |
|Poland       |1106     |1082   |1057           |3245            |
|France       |957      |942    |1008           |2907            |
|Sweden       |778      |813    |806            |2397            |
|United States|729      |696    |680            |2105            |
+-------------+---------+-------+---------------+----------------+
only showing top 10 rows

>>> output_path = "/user/root/project/Nationality_with_status.csv"
>>> result.coalesce(1).write.csv("/user/root/project/Nationality_with_status.csv", header=True, mode='overwrite')
```

.

2. What are the favorite or most visited destinations for frequent travelers

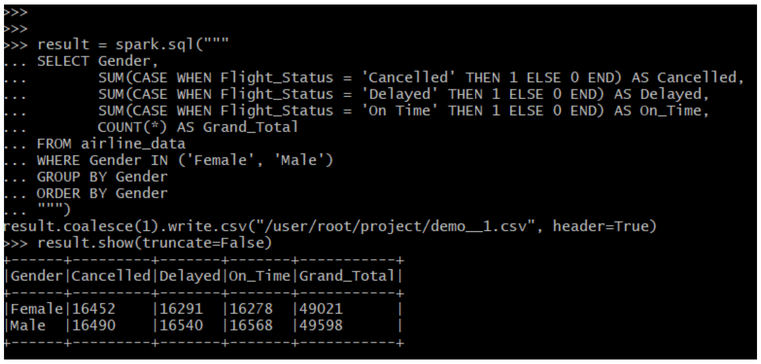
The insights reveal that the United States attracts the highest percentage of air travelers globally (22.41%), followed by Australia (6.46%) and Canada (5.5%). High travel volumes are concentrated in North America, Europe, and Asia, indicating these regions as major hubs for global air traffic. This data helps airlines focus on optimizing their flight networks and increasing operational efficiency in the most popular regions, leading to improved profitability and customer experience.



```
>>> spark.read.option("header", "true").option("inferSchema", "true").csv("/user/root/project/AirlineDatasetUpdatedv2.csv").createOrReplaceTempView("AirlineD
ataset")
>>> top_nationalities = spark.sql("""SELECT Nationality, COUNT(*) AS total_travelers FROM AirlineDataset GROUP BY Nationality ORDER BY total_travelers DESC""
")
>>> top_nationalities_list = [row['Nationality'] for row in top_nationalities.collect()]
>>> nationalities_str = ", ".join(["'{}'".format(nat) for nat in top_nationalities_list])
>>> query = """SELECT `Country Name`, COUNT(*) AS total_travelers, ROUND((COUNT(*) * 100.0 / SUM(COUNT(*)) OVER()), 2) AS percentage_of_travelers FROM Airlin
eDataset WHERE Nationality IN ({}) GROUP BY `Country Name` ORDER BY total_travelers DESC """.format(nationalities_str)
>>> result = spark.sql(query)
>>> top_countries_list = [
...     {
...         "Country Name": row['Country Name'],
...         "Total Travelers": row['total_travelers'],
...         "Percentage": row['percentage_of_travelers']
...     }
...     for row in result.collect()
... ]
24/10/20 14:30:51 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degra
dation.
>>> columns = ["Country Name", "Total Travelers", "Percentage"]
>>> top_countries_df = spark.createDataFrame(
... [(row["Country Name"], row["Total Travelers"], row["Percentage"]) for row in top_countries_list],
...     columns
... )
>>> top_countries_df.show(truncate=False)
+----------------------------------+---------------+---------------------+
|Country Name                      |Total Travelers|Percentage           |
+----------------------------------+---------------+---------------------+
|United States                     |22104          |22.410000000000000000|
|Australia                         |6370           |6.460000000000000000 |
|Canada                            |5424           |5.500000000000000000 |
|Brazil                            |4504           |4.570000000000000000 |
|Papua New Guinea                  |4081           |4.140000000000000000 |
|China                             |2779           |2.820000000000000000 |
|Indonesia                         |2358           |2.390000000000000000 |
|Russian Federation                |2247           |2.280000000000000000 |
|Colombia                          |1643           |1.670000000000000000 |
|India                             |1486           |1.510000000000000000 |
|France                            |1382           |1.400000000000000000 |
|United Kingdom                    |1371           |1.390000000000000000 |
|Argentina                         |1203           |1.220000000000000000 |
|Germany                           |1161           |1.180000000000000000 |
|Mexico                            |1073           |1.090000000000000000 |
|Japan                             |1004           |1.020000000000000000 |
```

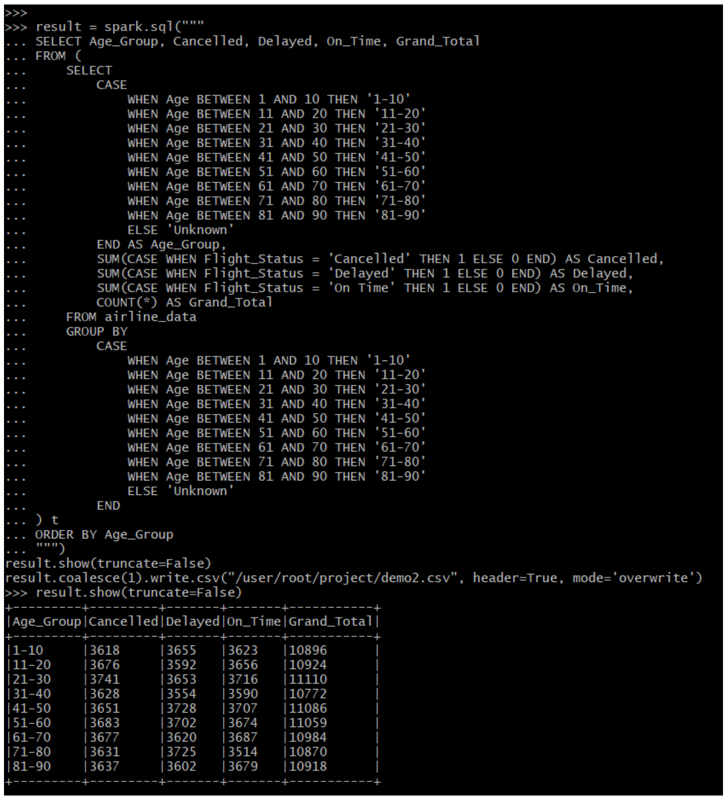
3. Passenger demographics based on Gender and Age

The passenger demographics reveal a nearly equal gender distribution, with males at 50.3% and females at 49.7%. This balance indicates that gender does not significantly impact flight outcomes, reflecting a consistent experience across both groups. There is minimal difference in flight statuses across genders. Airlines can tailor their services and marketing strategies based on gender-specific travel patterns. Understanding gender-based flight trends helps in optimizing flight schedules and enhancing customer satisfaction by targeting the right demographics.



```
>>>
>>>
>>> result = spark.sql("""
... SELECT Gender,
...        SUM(CASE WHEN Flight_Status = 'Cancelled' THEN 1 ELSE 0 END) AS Cancelled,
...        SUM(CASE WHEN Flight_Status = 'Delayed' THEN 1 ELSE 0 END) AS Delayed,
...        SUM(CASE WHEN Flight_Status = 'On Time' THEN 1 ELSE 0 END) AS On_Time,
...        COUNT(*) AS Grand_Total
... FROM airline_data
... WHERE Gender IN ('Female', 'Male')
... GROUP BY Gender
... ORDER BY Gender
... """)
result.coalesce(1).write.csv("/user/root/project/demo__1.csv", header=True)
>>> result.show(truncate=False)
+------+---------+-------+-------+-----------+
|Gender|Cancelled|Delayed|On_Time|Grand_Total|
+------+---------+-------+-------+-----------+
|Female|16452    |16291  |16278  |49021      |
|Male  |16490    |16540  |16568  |49598      |
+------+---------+-------+-------+-----------+
```

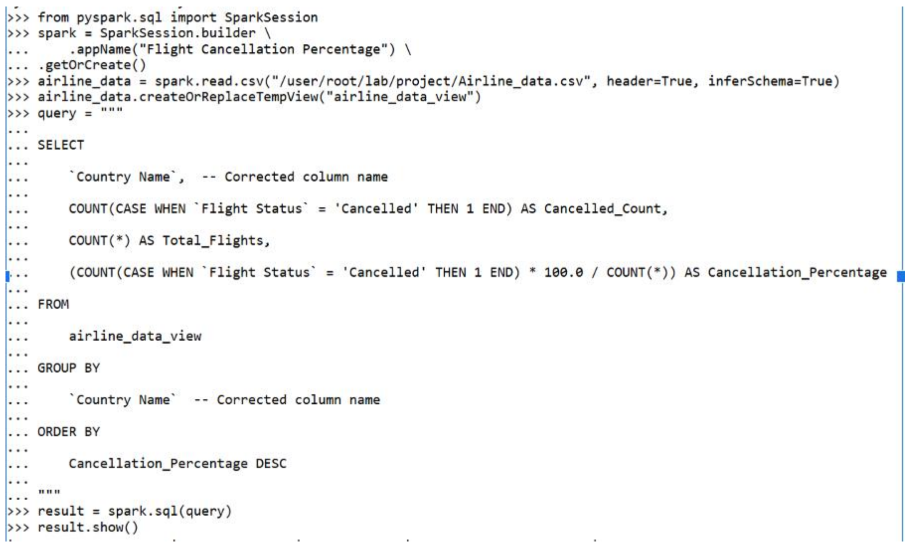
Individuals in the 21–30 age group seem to take the most flights, indicating they may be the most frequent travelers. This could be due to work-related travel, education, or leisure activities typically associated with this age group. 31-40 age groups show fewer flights compared to others, potentially reflecting life stages where travel may be less frequent (e.g., focusing on career). The trend suggests that young adults (21–30) and middle-aged individuals (41–60) are the primary travelers, possibly driven by work and leisure motivations.



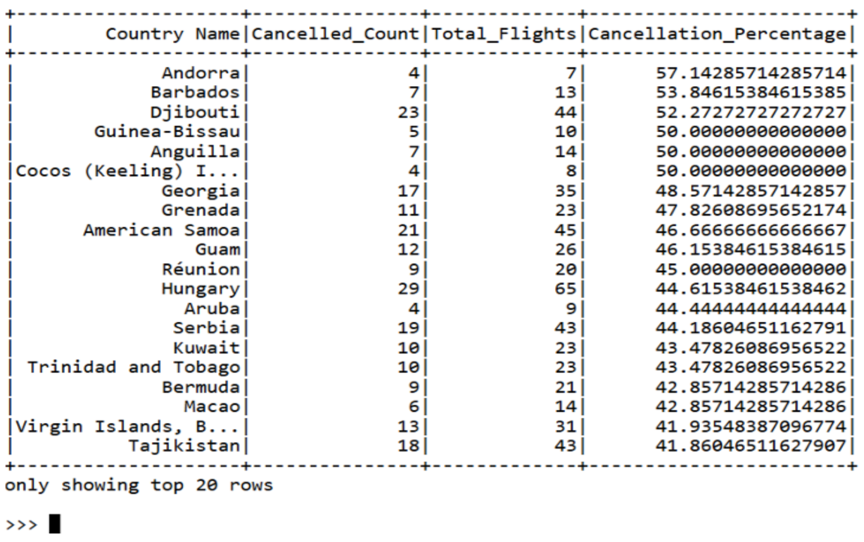
```
>>>
>>> result = spark.sql("""
... SELECT Age_Group, Cancelled, Delayed, On_Time, Grand_Total
... FROM (
...     SELECT
...         CASE
...             WHEN Age BETWEEN 1 AND 10 THEN '1-10'
...             WHEN Age BETWEEN 11 AND 20 THEN '11-20'
...             WHEN Age BETWEEN 21 AND 30 THEN '21-30'
...             WHEN Age BETWEEN 31 AND 40 THEN '31-40'
...             WHEN Age BETWEEN 41 AND 50 THEN '41-50'
...             WHEN Age BETWEEN 51 AND 60 THEN '51-60'
...             WHEN Age BETWEEN 61 AND 70 THEN '61-70'
...             WHEN Age BETWEEN 71 AND 80 THEN '71-80'
...             WHEN Age BETWEEN 81 AND 90 THEN '81-90'
...             ELSE 'Unknown'
...         END AS Age_Group,
...         SUM(CASE WHEN Flight_Status = 'Cancelled' THEN 1 ELSE 0 END) AS Cancelled,
...         SUM(CASE WHEN Flight_Status = 'Delayed' THEN 1 ELSE 0 END) AS Delayed,
...         SUM(CASE WHEN Flight_Status = 'On Time' THEN 1 ELSE 0 END) AS On_Time,
...         COUNT(*) AS Grand_Total
...     FROM airline_data
...     GROUP BY
...         CASE
...             WHEN Age BETWEEN 1 AND 10 THEN '1-10'
...             WHEN Age BETWEEN 11 AND 20 THEN '11-20'
...             WHEN Age BETWEEN 21 AND 30 THEN '21-30'
...             WHEN Age BETWEEN 31 AND 40 THEN '31-40'
...             WHEN Age BETWEEN 41 AND 50 THEN '41-50'
...             WHEN Age BETWEEN 51 AND 60 THEN '51-60'
...             WHEN Age BETWEEN 61 AND 70 THEN '61-70'
...             WHEN Age BETWEEN 71 AND 80 THEN '71-80'
...             WHEN Age BETWEEN 81 AND 90 THEN '81-90'
...             ELSE 'Unknown'
...         END
... ) t
... ORDER BY Age_Group
... """)
result.show(truncate=False)
result.coalesce(1).write.csv("/user/root/project/demo2.csv", header=True, mode='overwrite')
>>> result.show(truncate=False)
+---------+---------+-------+-------+-----------+
|Age_Group|Cancelled|Delayed|On_Time|Grand_Total|
+---------+---------+-------+-------+-----------+
|1-10     |3618     |3655   |3623   |10896      |
|11-20    |3676     |3592   |3656   |10924      |
|21-30    |3741     |3653   |3716   |11110      |
|31-40    |3628     |3554   |3590   |10772      |
|41-50    |3651     |3728   |3707   |11086      |
|51-60    |3683     |3702   |3674   |11059      |
|61-70    |3677     |3620   |3687   |10984      |
|71-80    |3631     |3725   |3514   |10870      |
|81-90    |3637     |3602   |3679   |10918      |
+---------+---------+-------+-------+-----------+
```

4. Regional Flight cancellation

Countries with smaller airports handling fewer flights tend to have higher cancellation rates due to limited resources, inadequate infrastructure, and frequent technical issues. Addressing these challenges can help reduce cancellations and enhance customer satisfaction. For airlines, the insight enables them to optimize operations by prioritizing resource allocation and collaboration with smaller airports to improve infrastructure and reliability. This can reduce flight cancellations, minimize revenue losses, improve on-time performance, enhance customer trust, and foster greater passenger loyalty, ultimately strengthening their market competitiveness.



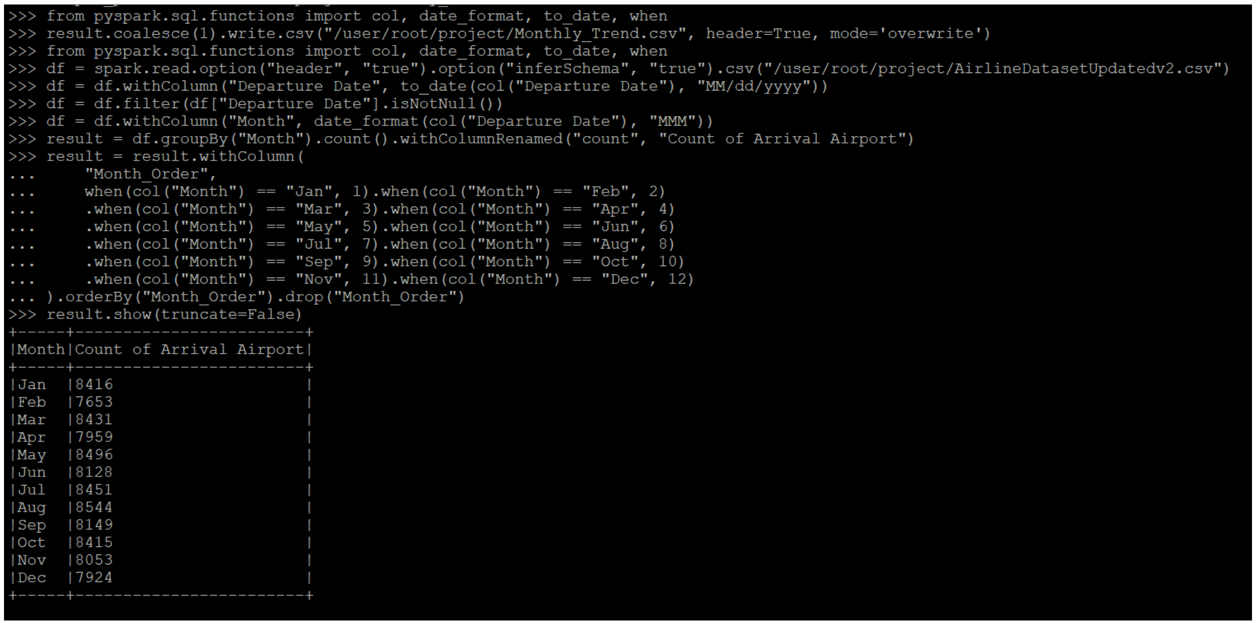
```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder \
...     .appName("Flight Cancellation Percentage") \
... .getOrCreate()
>>> airline_data = spark.read.csv("/user/root/lab/project/Airline_data.csv", header=True, inferSchema=True)
>>> airline_data.createOrReplaceTempView("airline_data_view")
>>> query = """
...
... SELECT
...
...     `Country Name`,  -- Corrected column name
...
...     COUNT(CASE WHEN `Flight Status` = 'Cancelled' THEN 1 END) AS Cancelled_Count,
...
...     COUNT(*) AS Total_Flights,
...
...     (COUNT(CASE WHEN `Flight Status` = 'Cancelled' THEN 1 END) * 100.0 / COUNT(*)) AS Cancellation_Percentage
...
... FROM
...
...     airline_data_view
...
... GROUP BY
...
...     `Country Name`  -- Corrected column name
...
... ORDER BY
...
...     Cancellation_Percentage DESC
...
... """
>>> result = spark.sql(query)
>>> result.show()
```



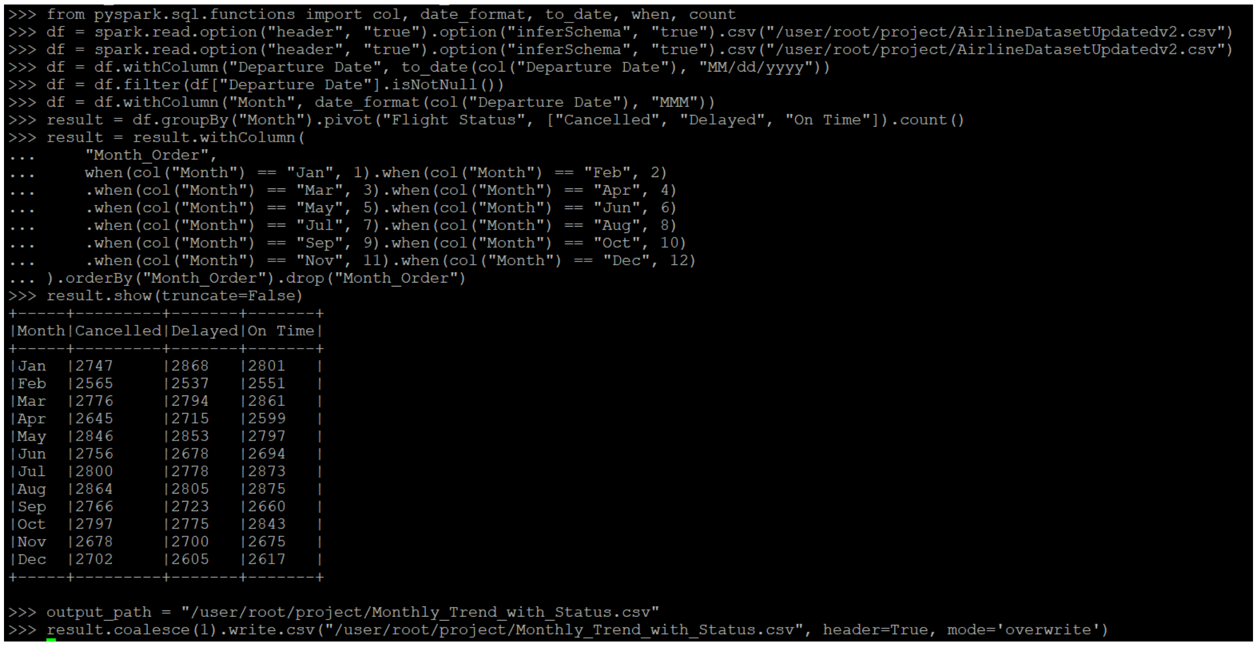
```
+--------------------+---------------+-------------+-----------------------+
|        Country Name|Cancelled_Count|Total_Flights|Cancellation_Percentage|
+--------------------+---------------+-------------+-----------------------+
|             Andorra|              4|            7|      57.14285714285714|
|            Barbados|              7|           13|      53.84615384615385|
|            Djibouti|             23|           44|      52.27272727272727|
|       Guinea-Bissau|              5|           10|      50.00000000000000|
|            Anguilla|              7|           14|      50.00000000000000|
|Cocos (Keeling) I...|              4|            8|      50.00000000000000|
|             Georgia|             17|           35|      48.57142857142857|
|             Grenada|             11|           23|      47.82608695652174|
|      American Samoa|             21|           45|      46.66666666666667|
|                Guam|             12|           26|      46.15384615384615|
|             Réunion|              9|           20|      45.00000000000000|
|             Hungary|             29|           65|      44.61538461538462|
|               Aruba|              4|            9|      44.44444444444444|
|              Serbia|             19|           43|      44.18604651162791|
|              Kuwait|             10|           23|      43.47826086956522|
| Trinidad and Tobago|             10|           23|      43.47826086956522|
|             Bermuda|              9|           21|      42.85714285714286|
|               Macao|              6|           14|      42.85714285714286|
|Virgin Islands, B...|             13|           31|      41.93548387096774|
|          Tajikistan|             18|           43|      41.86046511627907|
+--------------------+---------------+-------------+-----------------------+
only showing top 20 rows

>>> █
```

5. Monthly Trend

The data reveals a steady decline in monthly flights over time, an even distribution of flight statuses (On Time, Delayed, and Cancelled at approximately 33% each), and fluctuating trends in cancelled flights across different months. This insight enables better resource planning, such as adjusting flight schedules, optimizing crew deployment, and managing operational costs during peak months(July, August). It also helps in strategizing marketing campaigns and promotions to boost demand, improving profitability and overall efficiency.



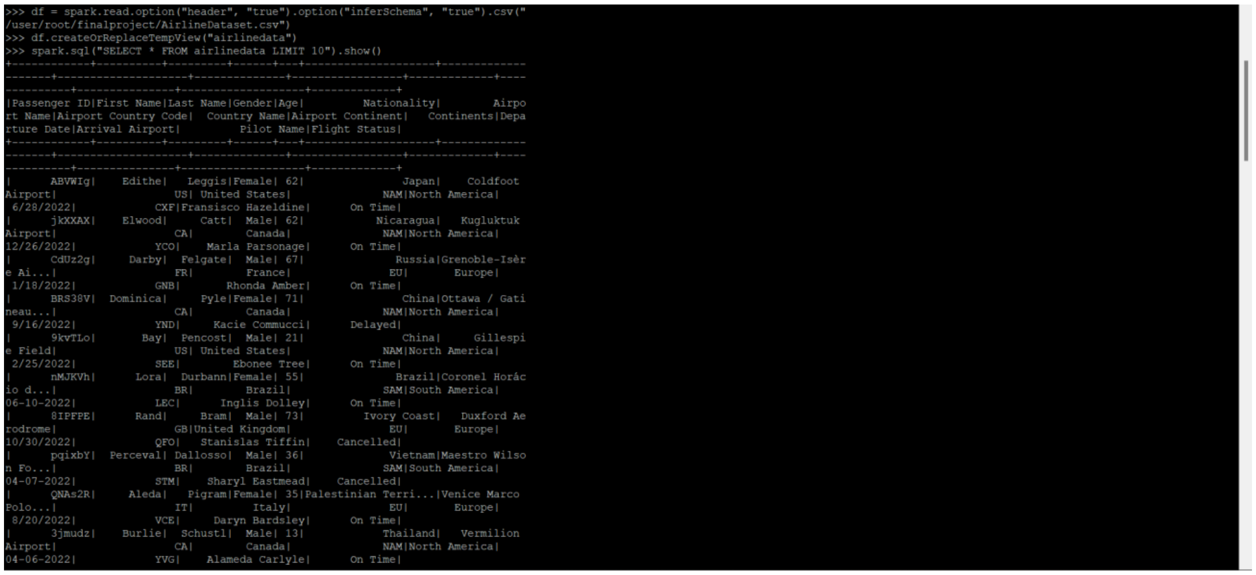
```
>>> from pyspark.sql.functions import col, date_format, to_date, when
>>> result.coalesce(1).write.csv("/user/root/project/Monthly_Trend.csv", header=True, mode='overwrite')
>>> from pyspark.sql.functions import col, date_format, to_date, when
>>> df = spark.read.option("header", "true").option("inferSchema", "true").csv("/user/root/project/AirlineDatasetUpdatedv2.csv")
>>> df = df.withColumn("Departure Date", to_date(col("Departure Date"), "MM/dd/yyyy"))
>>> df = df.filter(df["Departure Date"].isNotNull())
>>> df = df.withColumn("Month", date_format(col("Departure Date"), "MMM"))
>>> result = df.groupBy("Month").count().withColumnRenamed("count", "Count of Arrival Airport")
>>> result = result.withColumn(
...     "Month_Order",
...     when(col("Month") == "Jan", 1).when(col("Month") == "Feb", 2)
...     .when(col("Month") == "Mar", 3).when(col("Month") == "Apr", 4)
...     .when(col("Month") == "May", 5).when(col("Month") == "Jun", 6)
...     .when(col("Month") == "Jul", 7).when(col("Month") == "Aug", 8)
...     .when(col("Month") == "Sep", 9).when(col("Month") == "Oct", 10)
...     .when(col("Month") == "Nov", 11).when(col("Month") == "Dec", 12)
... ).orderBy("Month_Order").drop("Month_Order")
>>> result.show(truncate=False)
+-----+------------------------+
|Month|Count of Arrival Airport|
+-----+------------------------+
|Jan  |8416                    |
|Feb  |7653                    |
|Mar  |8431                    |
|Apr  |7959                    |
|May  |8496                    |
|Jun  |8128                    |
|Jul  |8451                    |
|Aug  |8544                    |
|Sep  |8149                    |
|Oct  |8415                    |
|Nov  |8053                    |
|Dec  |7924                    |
+-----+------------------------+
```



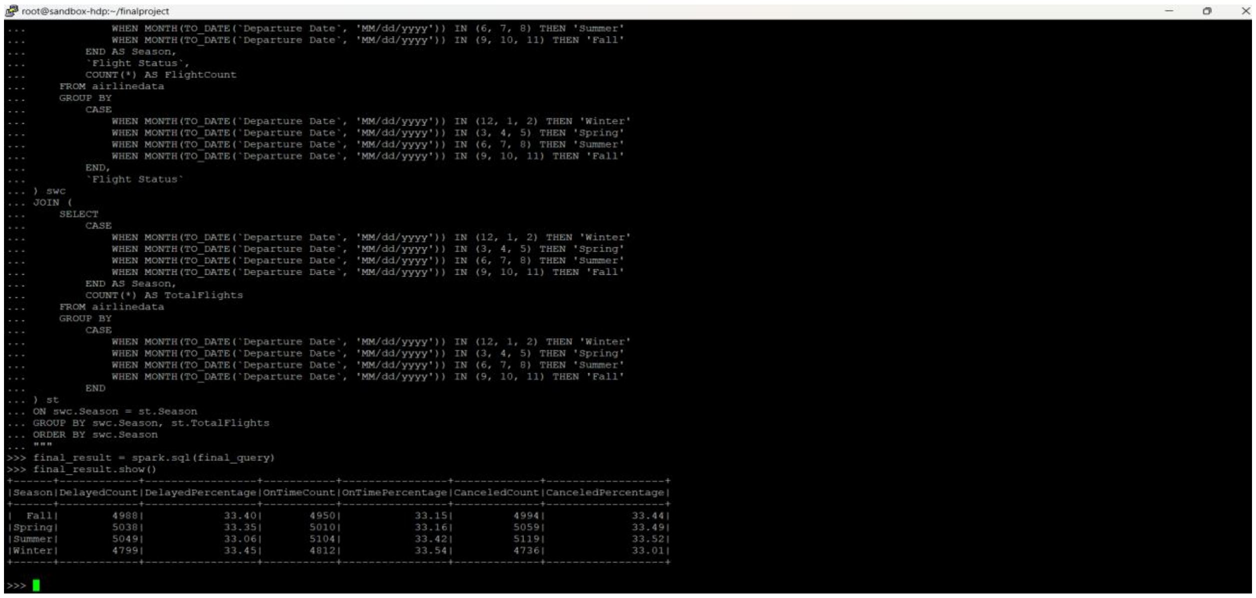
```
>>> from pyspark.sql.functions import col, date_format, to_date, when, count
>>> df = spark.read.option("header", "true").option("inferSchema", "true").csv("/user/root/project/AirlineDatasetUpdatedv2.csv")
>>> df = spark.read.option("header", "true").option("inferSchema", "true").csv("/user/root/project/AirlineDatasetUpdatedv2.csv")
>>> df = df.withColumn("Departure Date", to_date(col("Departure Date"), "MM/dd/yyyy"))
>>> df = df.filter(df["Departure Date"].isNotNull())
>>> df = df.withColumn("Month", date_format(col("Departure Date"), "MMM"))
>>> result = df.groupBy("Month").pivot("Flight Status", ["Cancelled", "Delayed", "On Time"]).count()
>>> result = result.withColumn(
...     "Month_Order",
...     when(col("Month") == "Jan", 1).when(col("Month") == "Feb", 2)
...     .when(col("Month") == "Mar", 3).when(col("Month") == "Apr", 4)
...     .when(col("Month") == "May", 5).when(col("Month") == "Jun", 6)
...     .when(col("Month") == "Jul", 7).when(col("Month") == "Aug", 8)
...     .when(col("Month") == "Sep", 9).when(col("Month") == "Oct", 10)
...     .when(col("Month") == "Nov", 11).when(col("Month") == "Dec", 12)
... ).orderBy("Month_Order").drop("Month_Order")
>>> result.show(truncate=False)
+-----+---------+-------+-------+
|Month|Cancelled|Delayed|On Time|
+-----+---------+-------+-------+
|Jan  |2747     |2868   |2801   |
|Feb  |2565     |2537   |2551   |
|Mar  |2776     |2794   |2861   |
|Apr  |2645     |2715   |2599   |
|May  |2846     |2853   |2797   |
|Jun  |2756     |2678   |2694   |
|Jul  |2800     |2778   |2873   |
|Aug  |2864     |2805   |2875   |
|Sep  |2766     |2723   |2660   |
|Oct  |2797     |2775   |2843   |
|Nov  |2678     |2700   |2675   |
|Dec  |2702     |2605   |2617   |
+-----+---------+-------+-------+

>>> output_path = "/user/root/project/Monthly_Trend_with_Status.csv"
>>> result.coalesce(1).write.csv("/user/root/project/Monthly_Trend_with_Status.csv", header=True, mode='overwrite')
```

6. **Seasonal Report**

The insight highlights that airlines generally maintain strong on-time performance in Spring, Summer, and Fall, with minimal delays and cancellations. However, Winter experiences a higher frequency of delays and cancellations due to challenges like snowstorms and low visibility. To enhance reliability, airlines need to focus on improving Winter operations by investing in weather forecasting, optimizing schedules, and increasing staff to manage disruptions. Additionally, proactive communication with passengers during Winter conditions can help mitigate inconvenience and improve customer satisfaction.



```
>>> df = spark.read.option("header", "true").option("inferSchema", "true").csv("
/user/root/finalproject/AirlineDataset.csv")
>>> df.createOrReplaceTempView("airlinedata")
>>> spark.sql("SELECT * FROM airlinedata LIMIT 10").show()
+------------+----------+---------+------+---+--------------------+------------
-------+--------------------+--------------+-----------------+-------------+----
----------+---------------+-------------------+-------------+
|Passenger ID|First Name|Last Name|Gender|Age|         Nationality|        Airpo
rt Name|Airport Country Code|  Country Name|Airport Continent|   Continents|Depa
rture Date|Arrival Airport|         Pilot Name|Flight Status|
+------------+----------+---------+------+---+--------------------+------------
-------+--------------------+--------------+-----------------+-------------+----
----------+---------------+-------------------+-------------+
|      ABVWIg|    Edithe|   Leggis|Female| 62|               Japan|    Coldfoot
Airport|                  US| United States|              NAM|North America|
 6/28/2022|            CXF|Fransisco Hazeldine|      On Time|
|      jkXXAX|    Elwood|     Catt|  Male| 62|           Nicaragua|   Kugluktuk
Airport|                  CA|        Canada|              NAM|North America|
12/26/2022|            YCO|    Marla Parsonage|      On Time|
|      CdUz2g|     Darby|  Felgate|  Male| 67|              Russia|Grenoble-Isèr
e Ai...|                  FR|        France|               EU|       Europe|
 1/18/2022|            GNB|       Rhonda Amber|      On Time|
|      BRS38V|  Dominica|     Pyle|Female| 71|               China|Ottawa / Gati
neau...|                  CA|        Canada|              NAM|North America|
 9/16/2022|            YND|     Kacie Commucci|      Delayed|
|      9kvTLo|       Bay|  Pencost|  Male| 21|               China|     Gillespi
e Field|                  US| United States|              NAM|North America|
 2/25/2022|            SEE|        Ebonee Tree|      On Time|
|      nMJKVh|      Lora|  Durbann|Female| 55|              Brazil|Coronel Horác
io d...|                  BR|        Brazil|              SAM|South America|
06-10-2022|            LEC|      Inglis Dolley|      On Time|
|      8IPFPE|      Rand|     Bram|  Male| 73|         Ivory Coast|   Duxford Ae
rodrome|                  GB|United Kingdom|               EU|       Europe|
10/30/2022|            QFO|   Stanislas Tiffin|    Cancelled|
|      pqixbY|  Perceval| Dallosso|  Male| 36|             Vietnam|Maestro Wilso
n Fo...|                  BR|        Brazil|              SAM|South America|
04-07-2022|            STM|    Sharyl Eastmead|    Cancelled|
|      QNAs2R|     Aleda|   Pigram|Female| 35|Palestinian Terri...|Venice Marco
Polo...|                  IT|         Italy|               EU|       Europe|
 8/20/2022|            VCE|     Daryn Bardsley|      On Time|
|      3jmudz|    Burlie|  Schustl|  Male| 13|            Thailand|   Vermilion
Airport|                  CA|        Canada|              NAM|North America|
04-06-2022|            YVG|    Alameda Carlyle|      On Time|
```



```
root@sandbox-hdp:~/finalproject
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (6, 7, 8) THEN 'Summer'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (9, 10, 11) THEN 'Fall'
...             END AS Season,
...             `Flight Status`,
...             COUNT(*) AS FlightCount
...         FROM airlinedata
...         GROUP BY
...             CASE
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (12, 1, 2) THEN 'Winter'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (3, 4, 5) THEN 'Spring'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (6, 7, 8) THEN 'Summer'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (9, 10, 11) THEN 'Fall'
...             END,
...             `Flight Status`
... ) swc
... JOIN (
...         SELECT
...             CASE
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (12, 1, 2) THEN 'Winter'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (3, 4, 5) THEN 'Spring'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (6, 7, 8) THEN 'Summer'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (9, 10, 11) THEN 'Fall'
...             END AS Season,
...             COUNT(*) AS TotalFlights
...         FROM airlinedata
...         GROUP BY
...             CASE
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (12, 1, 2) THEN 'Winter'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (3, 4, 5) THEN 'Spring'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (6, 7, 8) THEN 'Summer'
...                 WHEN MONTH(TO_DATE(`Departure Date`, 'MM/dd/yyyy')) IN (9, 10, 11) THEN 'Fall'
...             END
... ) st
... ON swc.Season = st.Season
... GROUP BY swc.Season, st.TotalFlights
... ORDER BY swc.Season
... """
>>> final_result = spark.sql(final_query)
>>> final_result.show()
+------+------------+-----------------+-----------+----------------+-------------+------------------+
|Season|DelayedCount|DelayedPercentage|OnTimeCount|OnTimePercentage|CanceledCount|CanceledPercentage|
+------+------------+-----------------+-----------+----------------+-------------+------------------+
|  Fall|        4988|            33.40|       4950|           33.15|         4994|             33.44|
|Spring|        5038|            33.35|       5010|           33.16|         5059|             33.49|
|Summer|        5049|            33.06|       5104|           33.42|         5119|             33.52|
|Winter|        4799|            33.45|       4812|           33.54|         4736|             33.01|
+------+------------+-----------------+-----------+----------------+-------------+------------------+

>>> 
```

f. Future work :

- Use machine learning to build predictive models that can forecast delays and cancellations based on historical weather patterns, flight data, and seasonal trends.
- Incorporate real-time data streams, such as weather updates and air traffic control data, to enhance operational responsiveness.
- Utilize optimization algorithms to recommend efficient flight routes, considering factors like fuel consumption, weather conditions, and air traffic.
- Integrate social media data and passenger feedback to analyze sentiment trends about airline services and improve customer experience.

g. References:

https://www.hadoopinrealworld.com/building-a-data-pipeline-with-apache-nifi/

https://hadoop.apache.org/

https://spark.apache.org/

https://www.elastic.co/kibana

https://www.youtube.com/watch?v=RH2JbpwgD9s

https://youtu.be/lR0i3xzNKQQ?si=teVsYarJS2g4mulb