Requirements: (by MoSCoW method[[1]](#footnote-1))

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Requirement Type | Id | Requirement | Functionality | Priority | Description |
| User-Type | 1.1 | User defines date range | fromDate, toDate parameters in GetBookingDataFrameStreaming function in “BookingStreaming” Notebook | Must have | String, Mandatory |
| 1.2 | User defines HDFS URI | Uri parameter in GetBookingDataFrameStreaming function in “BookingStreaming” Notebook | Must have | String, Mandatory |
| 1.3 | User defines Flight type |  | Could have | Default value “KL”  (for later change, a new parameter should be added to GetBookingDataFrameStreaming function) |
| 1.4 | User defines Source of Flight |  | Could have | Default value “Netherlands”  (for later change, a new parameter should be added to GetBookingDataFrameStreaming function) |
| 1.5 | User defines the number of top records, desire to receive |  | Could have | (for later change, a paging parameter should be added to GetBookingDataFrameStreaming function) |
|  |  |  |  |  |
| Input-Type | 2.1 | Airports data is read from airports.dat  “coma” delimiter rows | Airport data is located in HDFS in the path: “hdfs:///user/maria\_dev/KLM/airports.dat” | Must have | Necessary data: Country, IATA, Timezone  Only flights that ftype==airport |
| 2.2 | Bookings data is read from bookings.md in Json format | There is a folder in HDFS containing the bookings data   * If data is out of the cluster, kafka will be used to first load the data into the cluster (“write to kafka sample” and “read from kafka” parageraphs in “BookingStreaming” notebook | Must have | Necessary data: nbPassengers, bookingStatus, operatingAirline, originAirport, destinationAirport, departureDate, arrivalDate  Related to 1.1 |
| 2.3 | Data in 2.2 is selected every time by user | Inputs in GetBookingDataFrameStreaming function in “BookingStreaming” Notebook | Must have | Related to 1.2 |
|  |  |  |  |  |
| Output-Type | 3.1 | Add more information to the output |  | Should have | Follow description |
| 3.2 | Passengers count once per flight leg |  | Must have | Assumption: It’s not possible to have duplicate passengers in one flight |
| 3.3 | Only contains confirmed booking | “Merge 2 data sources” in “Analysis notebook, in “GetPopularDestination” function | Must have |  |
| 3.4 | Sorting: descending by the number of bookings, group by season, and day of week | “Merge 2 data sources” in “Analysis notebook, in “GetPopularDestination” function | Must have |  |
|  |  |  |  |  |
| 3.6 | In output, day of week is based on local time zone of the airport | “Merge 2 data sources” in “Analysis notebook, in “GetPopularDestination” function | Must have | “Timezone” field in airport is added to “ArrivalDate” column of booking to get localtime |
|  |  |  |  |  |
| System-Type | 4.1 | Analytical query that span large/small date ranges not that frequent and simultaneous |  | Must have |  |
| 4.2 | Availability |  | Should have | This query is for internal usage of KLM. Availability has less priority in CAP[[2]](#footnote-2) theorem |
| 4.3 | Consistency |  | Should have |  |
| 4.4 | Partition-tolerance |  | Must have |  |
| 4.5 | Security |  | Would have | This query is for internal usage of KLM. |
| 4.6 | Latency |  | Should have |  |
| 4.7 | Streaming | GetBookingDataFrameStreaming function in “BookingStreaming” Notebook  Reading json files | Should have |  |
| 4.8 | Exception handling, data format processing |  | Must have |  |
| 4.8 | Update airports data set daily |  | Could have | Airports dataset must update in the system daily |

# Access Pattern:

Give me the popular destinations, From NL departure airports, With KLM flights, In Date range, located in H folder (HDFS URI)

# Output Schema:

1. The real output is in this format:

|  |  |  |  |
| --- | --- | --- | --- |
| Country | Season | DayofWeek | NumberofBookings |
|  |  |  |  |

1. However, there is a possibility by applying pivot on the output to receive the number of passenger per season in a row. (It’s been explained later)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Country | Spring | Summer | Autumn | Winter | NumberofBookings |
|  |  |  |  |  |  |

1. Moreover, it is possible by applying pivot on the output to receive the number of passenger per week in a row as row. (It’s been explained later)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Country | Sat | Sun | Mon | Tue | Wed | Thu | Fri | NumberofBookings |
|  |  |  |  |  |  |  |  |  |

1. Join of the last 2 schemas leads to this one:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Country | Sat | Sun | Mon | Tue | Wed | Thu | Fri | Spring | Summer | Autumn | Winter | NumberofBookings |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

# Design & Implementation:

(In this part, I didn’t consider the technologies that are currently being used in KLM and just selected Zeppelin since it supports sql, scala, and shell commands in one notebook)

## Used technologies:

* Hortonworks sandbox with:
  + Spark version 2.3.1
  + Zeppelin 0.8.0
  + Zookeeper (in any case that Kafka is used)

## Coding Style

I decided to follow the functional style. Since, this assignment is more inclined toward evolving around a set of things. Adding new operations make more sense than adding new classes here.

### Airport Data

“Airport” notebook is responsible for managing airports data.

1. From Airport data “country”, “iATA”, and “timezone” got selected, imported into a dataframe. and cashed for later usage. (“timezone” is used for finding the local time of the destination airport after merging with bookings data)
   1. If the value of cell is not null, it gets lowercase for easier comparison later on
   2. If the value is null, it is replaced with string.empty for avoiding any null problems
2. Airport data is cashed into a data frame that supports case class of “Airport” as schema.

### Booking Data

Unfortunately, I couldn’t add dependencies to Zeppelin to use a fast JSON parser. I tried a bunch of libraries with even adding new repositories to maven but each time, zeppelin stopped working afterwards. These libraries can be listed as follows:

* + “Play” framework (Play-json)
  + Spray-json
  + Lift-json
  + Jerkson
  + Circe

Already, I spent lots of time for configuration of Hadoop and I couldn’t afford more to find the root cause of it.

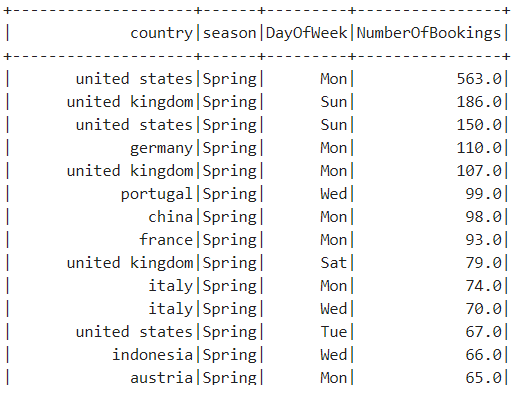
There are two notebooks for managing booking data.

* “Booking” notebook: since it was my first near to real experience with Hadoop, I decided ignore streaming and assumes there is one file for booking and it should be processed.
* “BookingStreaming” notebook: later on, I added the streaming. This notebook uses 2 approaches for streaming (choosing which approach should be done is a manual job):
  + If booking data is **outside** the cluster: reads data by kafka streaming, import it to the cluster and then write it in (HBase, parquet file,…) (actually I didn’t think about that very much. I only think a columnar data base is the best approach). Then data is loaded from the new data source. (“Write to Kafka sample” and “Read from kafka” paragraphs in the notebook)
  + If data is **inside** the cluster: I decided to use structured streaming. (since it’s more inclined towards real streaming than spark streaming, dataFrames are more optimized in processing than RDD. Conversely, I didn’t have any limitation for using newer version of Spark). The process of parsing booking data can be described as follows:
    1. For booking.json format case classes as the schema is defined (“booking schema” paragraph in the note book).
    2. Data is streamed following the schema and necessary columns are formed a new data frame. sample data and structure can be found in data folder, Booking1.csv file
    3. Then data is preprocessed (checking null and split array columns into separate columns), sample data and structure can be found in data folder, Booking2.csv file
    4. At the end, data is flatmapped. Each row is broken down into two separate rows and cell values are lowercased. (to reduce complexity later for joining and filtering) , sample data and structure can be found in data folder, Booking3.csv file
    5. StreamWriting is done in console for debugging purpose but later it should be done in a way that kafka is supposed to write. I put a piece of code here which is commented and I didn’t tested it.

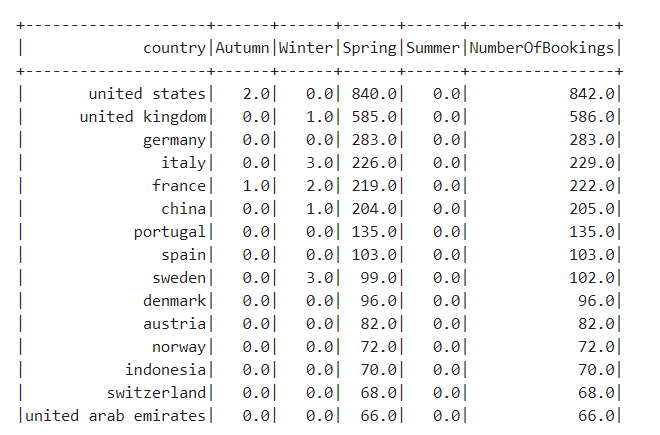
### Merging Data

“Analysis” notebook is responsible for merging and filtering data.

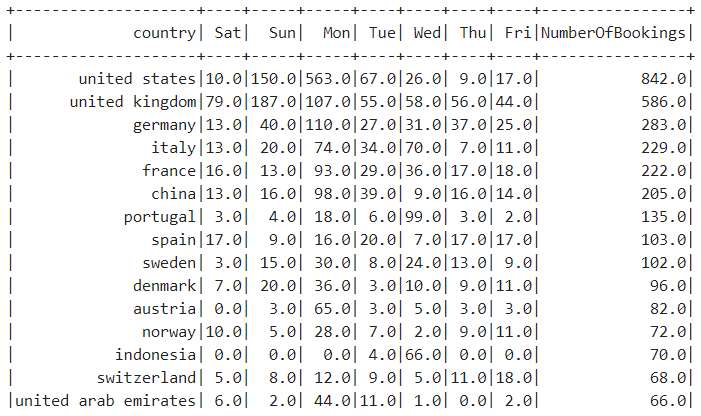
1. Joining is done on related column of “originAirpot” of “booking” and “iATA” of “airport”. two joins should be applied
2. Filter of requirements 1.3, 1.4, 3.3 is done on the first join
3. Then UTC to local time and date filtering is applied
4. Day of week and season are calculated
5. Then schema 1 is calculated. Data is ordered by “NumberOfBookings” descending.



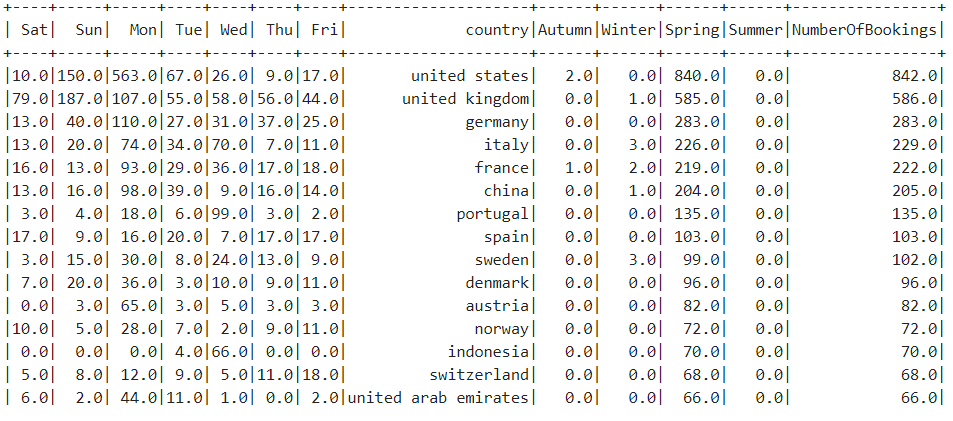
1. Then Schema 2 can be calculated (“pivot result based on season” paragraph). Pivoting is expensive but in spark2+ the performance has been got better. Null values are replaced by zero before calculating “NumberOfBookings”. Data is ordered by “NumberOfBookings” descending.



1. Then Schema 3 can be calculated (“pivot result based on week” paragraph) Data is ordered by “NumberOfBookings” descending.



1. Then Schema 4 can be calculated (“pivot result based on week and season” paragraph) Data is ordered by “NumberOfBookings” descending.



# Execution

These Notebooks are the output of Zeppelin. These should be imported to Zeppelin. Only kafka dependency is added to zeppelin. In case of running kafka, artifact “org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.0 ” must be added to dependencies of spark2 in interpreters.

Order of running Notebooks and their paragraphs:

* Airport
  + Airport Data Processing
  + Create DataFrame for airports
  + Airport table in SparkSQL
* Booking
  + Booking Data Processing
  + Validate Booking HDFS URL
  + In case of loading data without streaming:
    - Create DataFrame from JSON data
* BookingStreaming
  + Booking Schema
  + Streaming from a folder in HDFS
* Analysis
  + Date time analysis
  + Select streaming or not and then:
    - Merge 2 data sources without streaming
    - Merge 2 data sources with streaming
  + Get Final Result (select uir, fromDate, toDate here). Here, is the kind of the start point of program, but all the paragraphs should have been run before executing the function.
  + In case of need for schema 1 and schema 2
    - Pivot result based on a season
    - Pivot result based on a weak

1. <https://en.wikipedia.org/wiki/MoSCoW_method> [↑](#footnote-ref-1)
2. <https://en.wikipedia.org/wiki/CAP_theorem> [↑](#footnote-ref-2)