Aviation Data Analysis Using Apache Pig

The U.S. Department of Transportation’s (DOT) Bureau of Transportation Statistics (BTS) tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, cancelled, and diverted flights appears in DOT’s monthly Air Travel Consumer Report, published about 30 days after the month’s end, as well as in summary tables posted on this website. Summary statistics and raw data are made available to the public at the time the Air Travel Consumer Report is released.

These are 2 different datasets, i.e., Delayed\_Flights.csv and Airports.csv. Let us understand one at a time.

<https://drive.google.com/file/d/0B_Qjau8wv1KoWTVDUVFOdzlJNWM/view?usp=sharing>

**Delayed\_Flights.csv Datasets**

There are 29 columns in this dataset. Some of them have been mentioned below:

* Year 1987-2008
* Month 1-12
* DayofMonth 1-31
* DayOfWeek 1 (Monday) - 7 (Sunday)
* DepTime actual departure time (local, hhmm)
* CRSDepTime scheduled departure time (local, hhmm)
* ArrTime actual arrival time (local, hhmm)
* CRSArrTime scheduled arrival time (local, hhmm)
* UniqueCarrier unique carrier code
* FlightNum flight number
* TailNum plane tail number
* ActualElapsedTime in minutes
* CRSElapsedTime in minutes
* AirTime in minutes
* ArrDelay arrival delay, in minutes
* DepDelay departure delay, in minutes
* Origin origin IATA airport code
* Dest destination IATA airport code
* Distance in miles
* TaxiIn taxi in time, in minutes
* TaxiOut taxi out time in minutes
* Cancelled was the flight cancelled?
* CancellationCode reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
* Diverted 1 = yes, 0 = no
* CarrierDelay in minutes
* WeatherDelay in minutes
* NASDelay in minutes
* SecurityDelay in minutes
* LateAircraftDelay in minutes

**Airports.csv Datasets**

<https://drive.google.com/file/d/0B_Qjau8wv1KocDR3djk1Qm96Mmc/view?usp=sharing>

* iata: the international airport abbreviation code
* name of the airport
* city and country in which airport is located.
* lat and long: the latitude and longitude of the airport

Now, using Apache pig, we will try to gain more insights from these datasets.

**Problem Statement 1**

Find out the top 5 most visited destinations.

***Source Code***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | REGISTER /usr/lib/pig/piggybank.jar;    A = load '/user/cloudera/pig\_input/DelayedFlights.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO\_MULTILINE','UNIX','SKIP\_INPUT\_HEADER');    B = foreach A generate (int)$1 as year, (int)$10 as flight\_num, (chararray)$17 as origin,(chararray) $18 as dest;    C = filter B by dest is not null;    D = group C by dest;    E = foreach D generate group, COUNT(C.dest);    F = order E by $1 DESC;    Result = LIMIT F 5;    A1 = load '/user/cloudera/pig\_input/airports.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO\_MULTILINE','UNIX','SKIP\_INPUT\_HEADER');    A2 = foreach A1 generate (chararray)$0 as dest, (chararray)$2 as city, (chararray)$4 as country;    joined\_table = join Result by $0, A2 by dest;    dump joined\_table; |

**In Line 1**: We are registering the *piggybank* jar in order to use the CSVExcelStorage class.

In relation **A**, we are loading the dataset using CSVExcelStorage because of its effective technique to handle double quotes and headers.

In relation **B**, we are generating the columns that are required for processing and explicitly typecasting each of them.

In relation **C**, we are filtering the null values from the “dest” column.

In relation **D**, we are grouping relation C by “dest.”

In relation **E**, we are generating the grouped column and the count of each.

Relation **F** and **Result** is used to order and limit the result to top 5.

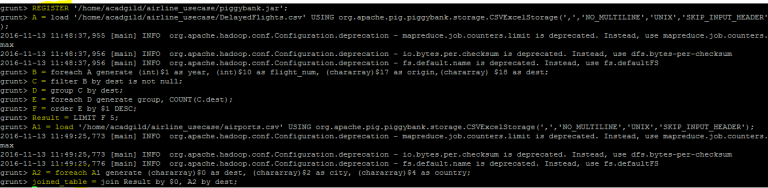
These are the steps to find the top 5 most visited destinations. However, adding few more steps in this process, we will be using another table to find the city name and country as well.

In relation **A1**, we are loading another table to which we will look-up and find the city as well as the country.

In relation **A2**, we are generating dest, city, and country from the previous relation.

In relation **joined\_table**, we are joining Result and A2 based on a common column, i.e., “dest”

Finally, using dump, we are printing the result.





**STORE Result INTO '/user/cloudera/pig\_avaitation\_output/result.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO\_MULTILINE','UNIX');**

**Problem Statement 2**

Which month has seen the most number of cancellations due to bad weather?

***Source code***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | REGISTER '/home/acadgild/airline\_usecase/piggybank.jar';    A = load '/home/acadgild/airline\_usecase/DelayedFlights.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO\_MULTILINE','UNIX','SKIP\_INPUT\_HEADER');    B = foreach A generate (int)$2 as month,(int)$10 as flight\_num,(int)$22 as cancelled,(chararray)$23 as cancel\_code;    C = filter B by cancelled == 1 AND cancel\_code =='B';    D = group C by month;    E = foreach D generate group, COUNT(C.cancelled);    F= order E by $1 DESC;    Result = limit F 1;    dump Result; |

**In Line 1**: We are registering *piggybank* jar in order to use the CSVExcelStorage class.

In relation **A**, we are loading the dataset using CSVExcelStorage because of its effective technique to handle double quotes and header.

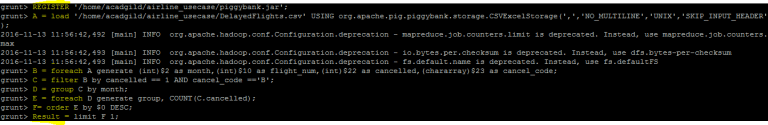
In relation **B**, we are generating the columns which are required for processing and explicitly typecasting each of them.

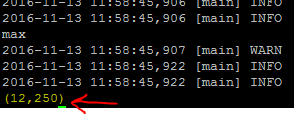
In relation **C**, we are filtering the data based on cancellation and cancellation code, i.e.,  canceled = 1 means flight have been canceled and cancel\_code = ‘B’ means the reason for cancellation is “weather.” So relation C will point to the data which consists of canceled flights due to bad weather.

In relation**D**, we are grouping the relation C based on every month.

In relation **E**, we are finding the count of canceled flights every month.

Relation **F** and **Result** is for ordering and finding the top month based on cancellation.





**Problem Statement 3**

Top ten origins with the highest AVG departure delay

***Source code***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | REGISTER '/home/acadgild/airline\_usecase/piggybank.jar';    A = load '/home/acadgild/airline\_usecase/DelayedFlights.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO\_MULTILINE','UNIX','SKIP\_INPUT\_HEADER');    B1 = foreach A generate (int)$16 as dep\_delay, (chararray)$17 as origin;    C1 = filter B1 by (dep\_delay is not null) AND (origin is not null);    D1 = group C1 by origin;    E1 = foreach D1 generate group, AVG(C1.dep\_delay);    Result = order E1 by $1 DESC;    Top\_ten = limit Result 10;    Lookup = load '/home/acadgild/airline\_usecase/airports.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO\_MULTILINE','UNIX','SKIP\_INPUT\_HEADER');    Lookup1 = foreach Lookup generate (chararray)$0 as origin, (chararray)$2 as city, (chararray)$4 as country;    Joined = join Lookup1 by origin, Top\_ten by $0;    Final = foreach Joined generate $0,$1,$2,$4;    Final\_Result = ORDER Final by $3 DESC;    dump Final\_Result; |

Explanation of first 3 lines are the same as explained in the previous 2 problem statements.

In relation**C1**, we are removing the null values fields present if any.

In relation **D1**, we are grouping the data based on column “origin.”

In relation **E1**, we are finding average delay from each unique origin.

Relations named **Result** and **Top\_ten** are ordering the results in descending order and printing the top ten values.

These steps are good enough to find the top ten origins with the highest average departure delay.

However, rather than generating just the code of origin, we will be following a few more steps to find some more details like country and city.

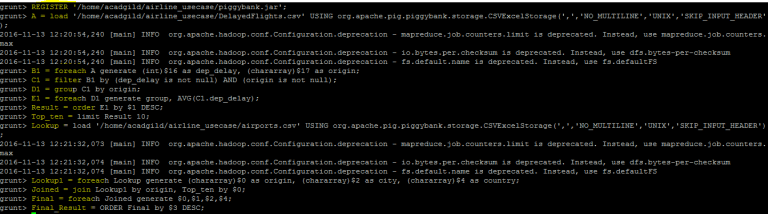
In the relation **Lookup**, we are loading another table to which we will look up and find the city as well as the country.

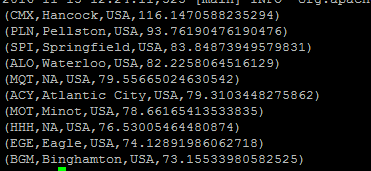
In the relation L**ookup1,** we are generating the destination, city, and country from the previous relation.

In the relation **Joined**, we are joining relation Top\_ten and Lookup1 based on common a column, i.e., “origin.”

In the relation**Final,**we are generating required columns from the Joined table.

Finally, we are ordering and printing the results.





**Problem Statement 4**

Which route (origin & destination) has seen the maximum diversion?

***Source code***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | REGISTER '/home/acadgild/airline\_usecase/piggybank.jar';    A = load '/home/acadgild/airline\_usecase/DelayedFlights.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO\_MULTILINE','UNIX','SKIP\_INPUT\_HEADER');    B = FOREACH A GENERATE (chararray)$17 as origin, (chararray)$18 as dest, (int)$24 as diversion;    C = FILTER B BY (origin is not null) AND (dest is not null) AND (diversion == 1);    D = GROUP C by (origin,dest);    E = FOREACH D generate group, COUNT(C.diversion);    F = ORDER E BY $1 DESC;    Result = limit F 10;    dump Result; |

**In Line 1**: We are registering *piggybank* jar in order to use CSVExcelStorage class.

In relation **A**, we are loading the dataset using CSVExcelStorage because of its effective technique to handle double quotes and headers.

In relation **B**, we are generating the columns which are required for processing and explicitly type-casting each of them.

In relation **C**, we are filtering the data based on “not null” and diversion =1. This will remove the null records, if any, and give the data corresponding to the diversion taken.

In relation **D**, we are grouping the data based on origin and destination.

Relation **D** finds the count of diversion taken per unique origin and destination.

Relations **F** and **Result** orders the result and produces top 10 results.

