Problem Statement

Implement the below blog at your end and send the complete documentation.
https://drive.google.com/file/d/0B_Qjau8wv1KoUThzZ24tT1NsZGs/view?usp=sharing

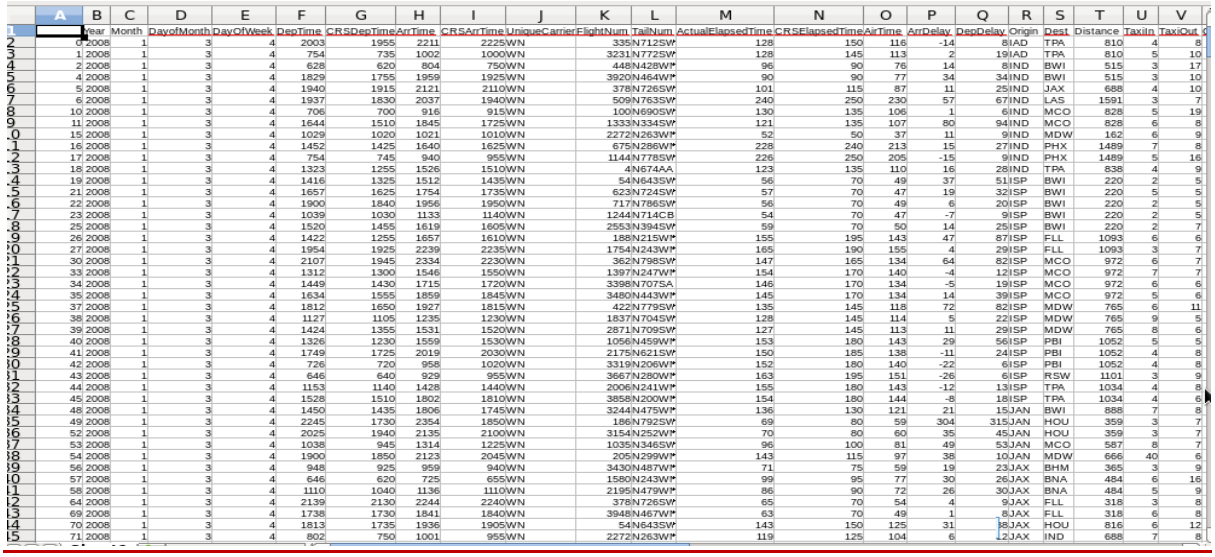
(Airline_data_analysis.odt)

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, canceled, 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.

You can download the datasets from the following links:

Delayed_Flights.csv

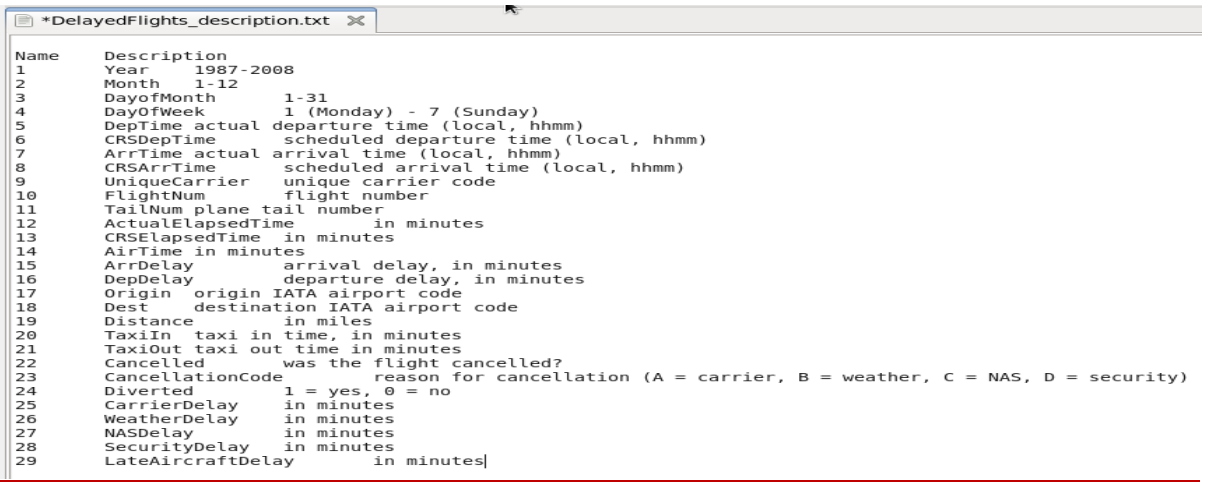
DataSet



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1		Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay	DepDelay	Origin	Dest	Distance	TaxiIn	TaxiOut
2	0	2008	1	3	4	2003	1955	2211	2225	WN	335	N712SW	128	150	116	-14	8	IAD	TPA	810	4	8
3	1	2008	1	3	4	754	735	1002	1000	WN	3231	N772SW	128	145	113	2	19	IAD	TPA	810	5	10
4	2	2008	1	3	4	628	620	804	750	WN	448	N428WN	96	90	76	14	8	IND	BWI	515	3	17
5	4	2008	1	3	4	1829	1755	1959	1925	WN	3920	N464WN	90	90	77	34	34	IND	BWI	515	3	10
6	5	2008	1	3	4	1940	1915	2121	2110	WN	378	N726SW	101	115	87	11	25	IND	JAX	688	4	10
7	6	2008	1	3	4	1937	1830	2037	1940	WN	509	N763SW	240	250	230	57	67	IND	LAS	1591	3	7
8	10	2008	1	3	4	706	700	916	915	WN	100	N690SW	130	135	106	1	6	IND	MCO	828	5	19
9	11	2008	1	3	4	1644	1510	1845	1725	WN	1333	N334SW	121	135	107	80	94	IND	MCO	828	6	8
10	15	2008	1	3	4	1029	1020	1021	1010	WN	2272	N263WN	52	50	37	11	9	IND	MDW	162	6	9
11	16	2008	1	3	4	1452	1425	1640	1625	WN	675	N286WN	228	240	213	15	27	IND	PHX	1489	7	8
12	17	2008	1	3	4	754	745	940	955	WN	1144	N778SW	226	250	205	-15	9	IND	PHX	1489	5	16
13	18	2008	1	3	4	1323	1255	1526	1510	WN	4	N674AA	123	135	110	16	28	IND	TPA	838	4	9
14	19	2008	1	3	4	1416	1325	1512	1435	WN	54	N643SW	56	70	49	37	51	ISP	BWI	220	2	5
15	21	2008	1	3	4	1657	1625	1754	1735	WN	623	N724SW	57	70	47	19	32	ISP	BWI	220	5	5
16	22	2008	1	3	4	1900	1840	1956	1950	WN	717	N786SW	56	70	49	6	20	ISP	BWI	220	2	5
17	23	2008	1	3	4	1039	1030	1133	1140	WN	1244	N714CB	54	70	47	-7	9	ISP	BWI	220	2	5
18	25	2008	1	3	4	1520	1455	1619	1605	WN	2553	N394SW	59	70	50	14	25	ISP	BWI	220	2	7
19	26	2008	1	3	4	1422	1255	1657	1610	WN	188	N215WN	155	195	143	47	87	ISP	FLL	1093	6	6
20	27	2008	1	3	4	1954	1925	2239	2235	WN	1754	N243WN	165	190	155	4	29	ISP	FLL	1093	3	7
21	30	2008	1	3	4	2107	1945	2334	2230	WN	362	N798SW	147	165	134	64	82	ISP	MCO	972	6	7
22	33	2008	1	3	4	1312	1300	1546	1550	WN	1397	N247WN	154	170	140	-4	12	ISP	MCO	972	7	7
23	34	2008	1	3	4	1449	1430	1715	1720	WN	3398	N7075A	146	170	134	-5	19	ISP	MCO	972	6	6
24	35	2008	1	3	4	1634	1555	1859	1845	WN	3480	N443WN	145	170	134	14	39	ISP	MCO	972	5	6
25	37	2008	1	3	4	1812	1650	1927	1815	WN	422	N779SW	135	145	118	72	82	ISP	MDW	765	6	11
26	38	2008	1	3	4	1127	1105	1235	1230	WN	1837	N704SW	128	145	114	5	22	ISP	MDW	765	9	5
27	39	2008	1	3	4	1424	1355	1531	1520	WN	2871	N709SW	127	145	113	11	29	ISP	MDW	765	8	6
28	40	2008	1	3	4	1326	1230	1559	1530	WN	1056	N459WN	153	180	143	29	56	ISP	PBI	1052	5	5
29	41	2008	1	3	4	1749	1725	2019	2030	WN	2175	N621SW	150	185	138	-11	24	ISP	PBI	1052	4	8
30	42	2008	1	3	4	726	720	958	1020	WN	3319	N206WN	152	180	140	-22	6	ISP	PBI	1052	4	8
31	43	2008	1	3	4	646	640	929	955	WN	3667	N280WN	163	195	151	-26	6	ISP	RSW	1101	3	9
32	44	2008	1	3	4	1153	1140	1428	1440	WN	2006	N241WN	155	180	143	-12	13	ISP	TPA	1034	4	8
33	45	2008	1	3	4	1528	1510	1802	1810	WN	3858	N200WN	154	180	144	-8	18	ISP	TPA	1034	4	6
34	48	2008	1	3	4	1450	1435	1806	1745	WN	3244	N475WN	136	130	121	21	15	JAN	BWI	888	7	8
35	49	2008	1	3	4	2245	1730	2354	1850	WN	186	N792SW	69	80	59	304	315	JAN	HOU	359	3	7
36	52	2008	1	3	4	2025	1940	2135	2100	WN	3154	N252WN	70	80	60	35	45	JAN	HOU	359	3	7
37	53	2008	1	3	4	1038	945	1314	1225	WN	1035	N346SW	96	100	81	49	53	JAN	MCO	587	8	7
38	54	2008	1	3	4	1900	1850	2123	2045	WN	205	N299WN	143	115	97	38	10	JAN	MDW	666	40	6
39	56	2008	1	3	4	948	925	959	940	WN	3430	N487WN	71	75	59	19	23	JAX	BHM	365	3	9
40	57	2008	1	3	4	646	620	725	655	WN	1580	N243WN	99	95	77	30	26	JAX	BNA	484	6	16
41	58	2008	1	3	4	1110	1040	1136	1110	WN	2195	N479WN	86	90	72	26	30	JAX	BNA	484	5	9
42	64	2008	1	3	4	2139	2130	2244	2240	WN	378	N726SW	65	70	54	4	9	JAX	FLL	318	3	8
43	69	2008	1	3	4	1738	1730	1841	1840	WN	3948	N467WN	63	70	49	1	8	JAX	FLL	318	6	8
44	70	2008	1	3	4	1813	1735	1936	1905	WN	54	N643SW	143	150	125	31	38	JAX	HOU	816	6	12
45	71	2008	1	3	4	802	750	1001	955	WN	2272	N263WN	119	125	104	6	2	JAX	IND	688	7	8

Delayed_Flights.csv Data Description

There are 29 columns in this dataset



*DelayedFlights_description.txt

```
Name    Description
1       Year    1987-2008
2       Month   1-12
3       DayofMonth      1-31
4       DayOfWeek       1 (Monday) - 7 (Sunday)
5       DepTime actual departure time (local, hhmm)
6       CRSDepTime      scheduled departure time (local, hhmm)
7       ArrTime actual arrival time (local, hhmm)
8       CRSArrTime      scheduled arrival time (local, hhmm)
9       UniqueCarrier   unique carrier code
10      FlightNum       flight number
11      TailNum plane tail number
12      ActualElapsedTime       in minutes
13      CRSElapsedTime  in minutes
14      AirTime in minutes
15      ArrDelay        arrival delay, in minutes
16      DepDelay        departure delay, in minutes
17      Origin  origin IATA airport code
18      Dest    destination IATA airport code
19      Distance        in miles
20      TaxiIn  taxi in time, in minutes
21      TaxiOut taxi out time in minutes
22      Cancelled       was the flight cancelled?
23      CancellationCode        reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
24      Diverted        1 = yes, 0 = no
25      CarrierDelay    in minutes
26      WeatherDelay    in minutes
27      NASDelay        in minutes
28      SecurityDelay   in minutes
29      LateAircraftDelay       in minutes
```

Solution

Importing Spark Sql packages

```
scala> import org.apache.spark.sql._
import org.apache.spark.sql._

scala> import sqlContext.implicits._
import sqlContext.implicits._

scala>
```

Creating base rdd for delayed flights dataset to be used to solve the queries

- Begin by reading the data file as a text file from the local FS using the Spark context object ***sc***. delayedflights
- The 1st line of **delayedflights**, that is the data description, is saved in header.
- **header** is then used in delayed_flights to eliminate data description not to perform analysis on it.

Code

```
scala> val delayedflights = sc.textFile("file:///home/acadgild/Downloads/DelayedFlights.csv")
delayedflights: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[56] at textFile at <console>:39

scala> val header = delayedflights.first()
header: String = ,Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueCarrier,FlightNum,TailNum,Actua
lElapsedTime,CRSElapsedTime,AirTime,ArrDelay,DepDelay,Origin,Dest,Distance,TaxiIn,TaxiOut,Cancelled,CancellationCode,Diverted
,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay

scala> val delayed_flights = delayedflights.filter(row => row != header)
delayed_flights: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[57] at filter at <console>:43

scala>
```

Output

Showing only 5 records from delayed flights dataset

```
scala> delayed_flights.take(5).foreach(println)
0,2008,1,3,4,2003.0,1955,2211.0,2225,WN,335,N712SW,128.0,150.0,116.0,-14.0,8.0,IAD,TPA,810,4.0,8.0,0,N,0,,,,,
1,2008,1,3,4,754.0,735,1002.0,1000,WN,3231,N772SW,128.0,145.0,113.0,2.0,19.0,IAD,TPA,810,5.0,10.0,0,N,0,,,,,
2,2008,1,3,4,628.0,620,804.0,750,WN,448,N428WN,96.0,90.0,76.0,14.0,8.0,IND,BWI,515,3.0,17.0,0,N,0,,,,,
4,2008,1,3,4,1829.0,1755,1959.0,1925,WN,3920,N464WN,90.0,90.0,77.0,34.0,34.0,IND,BWI,515,3.0,10.0,0,N,0,2.0,0.0,0.0,0.0,32.0
5,2008,1,3,4,1940.0,1915,2121.0,2110,WN,378,N726SW,101.0,115.0,87.0,11.0,25.0,IND,JAX,688,4.0,10.0,0,N,0,,,,,

scala>
```

Find out the top 5 most visited destinations

- The data is first split by comma, as it is from a comma separated value (csv) file and mapped to only the 18th column (**Dest**) and 1, i.e. (Dest,1)
- The resultant data is then filtered such that there is no blank values in the Destination field
- Next, the resultant data is reduced by the key element (Dest), such that all the data corresponding to each key element are grouped together and added (i.e. all the 1's)
- Next, the resultant data is mapped to get the key with the summed value i.e. (Dest, sum(Dest 1's)). This gives us the count of how many times the Destination has appeared in the data.
- Next, the resultant data is sorted by key **false**, i.e. in descending order.
- Finally, only select the top 5 values from the resultant data.

Code

```
scala> val most_visited_destn = delayed_flights.map(x => x.split(",")).
     | map(x => (x(18),1)).
     | filter(x => x._1 != null).
     | reduceByKey(_+_).
     | map(x => (x._2, x._1)).
     | sortByKey(false).map(x => (x._2,x._1)).
     | take(5)
most_visited_destn: Array[(String, Int)] = Array((ORD,108984), (ATL,106898), (DFW,70657), (DEN,63003), (LAX,59969))

scala>
```

Output

```
scala> most_visited_destn.foreach(println)
(ORD,108984)
(ATL,106898)
(DFW,70657)
(DEN,63003)
(LAX,59969)

scala>
```

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

- The data is first split by comma, as it is from a comma separated value (csv) file and filtered to data where,
- 22nd column (**Cancelled**) == 1, signifying that a cancellation had happened *and*
- 23rd column (**CancellationCode**) == B, signifying cancellation due to bad weather
- The data is then mapped to the data that satisfies the above criteria's and 1, i.e. (entire_row,1)
- Next, the resultant data is reduced by the key element (the entire row-x(2)), such that all the data corresponding to each key element are grouped together and added (i.e. all the 1's)
- Next, the resultant data is mapped to get the **Month(x._2)** with the summed value i.e. (Month, sum(Cancels in Month)). This gives us the count of how many times cancellation has happened in that month.
- Next, the resultant data is sorted by key **false**, i.e. in descending order and only the 1st row is selected.

Code

```
scala> val most_cancelation_month = delayed_flights.map(x => x.split(",")).
     | filter(x => ((x(22).equals("1"))&&(x(23).equals("B")))).
     | map(x => (x(2),1)).
     | reduceByKey(_+_).
     | map(x => (x._2,x._1)).
     | sortByKey(false).
     | map(x => (x._2,x._1)).
     | take(1)
most_cancelation_month: Array[(String, Int)] = Array((12,250))

scala>
```

Output

```
scala> most_cancelation_month.foreach(println)
(12,250)

scala>
```

Top ten origins with the highest AVG departure delay

- The data is first split by comma, as it is from a comma separated value (csv) file and mapped to only the 17th column (**Origin**) and 16th column (**DepDelay**) respectively
- Both columns are then mapped to (column, 1). i.e. (Origin,1) and (DepDelay,1)
- Next, the resultant data is reduced by the key element for both Origin and DepDelay , such that all the data corresponding to each key element are grouped together and added (i.e. all the 1's for each of them)
- Next, take the Origin as the **sum** and DepDelay as the **count** and get the **avg** for each row and map the results as (Origin, avg)
- Finally, the resultant data is sorted by key **false**, i.e. in Desc order and 1st 10 rows are selected.

Code

```
scala> val avg = delayed_flights.map(x => x.split(",")).
     | map(x => (x(17),x(16).toDouble)).
     | mapValues((_, 1)).
     | reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2)).
     | mapValues{ case (sum, count) => (1.0 * sum)/count}.
     | map(x => (x._2,x._1)).
     | sortByKey(false).
     | map(x => (x._2,x._1)).
     | take(10)
avg: Array[(String, Double)] = Array((CMX,116.1470588235294), (PLN,93.76190476190476), (SPI,83.84873949579831), (ALO,82.2258064516129), (MQT,79.55665024630542), (ACY,79.3103448275862), (MOT,78.66165413533835), (HHH,76.53005464480874), (EGE,74.12891986062718), (BGM,73.15533980582525))

scala>
```

Output

```
scala> avg.foreach(println)
(CMX,116.1470588235294)
(PLN,93.76190476190476)
(SPI,83.84873949579831)
(ALO,82.2258064516129)
(MQT,79.55665024630542)
(ACY,79.3103448275862)
(MOT,78.66165413533835)
(HHH,76.53005464480874)
(EGE,74.12891986062718)
(BGM,73.15533980582525)

scala>
```

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

- The data is first split by comma, as it is from a comma separated value (csv) file and filtered to data where, the 24th column (**Diverted**) == 1, signifying that the flight had been diverted
- The resultant data is then mapped such that the 1st column is a concatenation of the column 17 (**Origin**) and column 18 (**Dest**) and 1, i.e. ("Origin,Dest",1)
- Next, the resultant data is reduced by the key element ("Orgin,Dest"), such that all the data corresponding to each key element are grouped together and added (i.e. all the 1's)
- The data is then mapped to ("Origin,Dest",1)
- Finally, the resultant data is sorted by key **false**, i.e. in Desc order and 1st 10 rows are selected, with the 1st row as the route with the most diversion.

Code

```
scala> val diversion = delayed_flights.map(x => x.split(",")).
     | filter(x => ((x(24).equals("1")))).
     | map(x => ((x(17)+","+x(18)),1)).
     | reduceByKey(_+_).
     | map(x => (x._2,x._1)).
     | sortByKey(false).
     | map(x => (x._2,x._1)).
     | take(10)
diversion: Array[(String, Int)] = Array((ORD,LGA,39), (DAL,HOU,35), (DFW,LGA,33), (ATL,LGA,32), (SLC,SUN,31), (ORD,SNA,31), (
MIA,LGA,31), (BUR,JFK,29), (HRL,HOU,28), (BUR,DFW,25))

scala>
```

Output

```
scala> diversion.foreach(println)
(ORD,LGA,39)
(DAL,HOU,35)
(DFW,LGA,33)
(ATL,LGA,32)
(SLC,SUN,31)
(ORD,SNA,31)
(MIA,LGA,31)
(BUR,JFK,29)
(HRL,HOU,28)
(BUR,DFW,25)

scala>

scala>
```