Author : Sharmo and Sarita

-----Implementation-

The input is all CSV files from the Flight data.
These are then split every row based on "," delimiter.
The data is cleansed based on the general rules for the Flight Dataset (as from Assignmnet 01)
Then we then create 2 RDDs one for arrival flight details and other for departure flight details.

Departure Data :-

Key: Carrier Code, Origin Airport Code
Value: Scheduled Departure time, Actual Departure Time, Cancelled, Flight Date

Arrival Data :-

Key: Carrier Code, Destination Airport Code,
Value: Scheduled Arrival time, Actual Arrival Time, Cancelled, Flight Date

Then we join both the RDDs which groups the data from both the RDDs sharing the same key.

Now all the resulting output of the join are potential connections.
We use the following logic for checking whether the flights are truely connected or not, and if connected then whether there was a missed connection or not.
  A connection is any pair of flight F and G of the same carrier such as F.Destination = G.Origin and the scheduled departure of G is <= 6 hours and >= 30   minutes after the scheduled arrival of F.
  A connection is missed when the actual arrival of F < 30 minutes before the actual departure of G

The output from the program is in the form of :::
AirLineCarrier Year Connections Missed_Connections Percentage_Missed_connections

------- Output ---------------
The output from the dataset is provided in part-00000

-------Time Comparison----------

Map reduce on local: 1.5 hrs approx
Map reduce on EMR: 1 hrs aprox

Spark(Scala) on local: 45 mins approx
Spark(Scala) on EMR: 40 mins approx

Spark(Python) on local : 1.25 hrs approx

----------Conclusion-----------

We can conclude that Spark implementation runs faster than Map Reduce implementation. This is due to the fact that Spark uses RDDs and the data thats loaded into the Spark program is in these RDDs. As opposed to using ArrayLists in the MapReduce Implementation (to separate values/entries corresponding to Origin or destination flight details) we simply used RDDs. The transformtion operations on these RDDs go a long way to speed up the process. The direct join operation amongst the RDDs made this Spark implementation both readable and more faster in execution. Amongst the two Spark implementations, the Scala implementation runs faster than the Python implementation. This is due to the fact that Scala is a complied language while Python is an interpreted language. Besides this, the Spark RDDs are most affected when it comes to Python executors. Every data that comes to and goes from the Python executor has to pass through the socket and the JVM worker. This creates some overheads. Python is a process based executor while Scala is a thread based executor. Each Python executor runs its own process which results in potentially higher memory usage than the thread based Scala. These are a few reasons why the Python implementation using Spark is shown to be so slow in comparison to Scala.