**Assumptions:**

1. Read the input data and look for ArrDelay or field index 14 only +positive values. Filter out the 0/NA or -ve values
2. Filter out origin and destination (index 16 and 17 if either of them are empty.
3. These are the only 3 field used

**Build Command (maven) :**

1. Download the project and go to folder Swami-UnityChallenge
2. Run ***mvn clean package***
3. ***Using hadoop 2.6, Java 1.8***

**Code:**

1. **Main Driver:** com.unity.challenge.mapreduce.driver.MapreduceDriver
2. Will execute AirlineDelayAverageMR and then TopNAirlineDelayMR (please read forceExecution below)
3. **AirlineDelayAverageMR/ AverageCSVMapper** will filter out the following records from the input in the mapper. The records with ArrDelay if it is ‘NA’ or<= 0 or origin or destination is blank will be filtered out
4. **AirlineDelayAverageMR/ AverageCSVReduce** will have 100 records per reducer file, using PriorityQueue and output. **Important:Only if there are more than 1 reducer file generated will it run the TopNAirlineDelayMR**
5. *Based on the data from 2000 till 2008 did not generate more than 1 reducer file and hence not sure if the TopNAirlineDelayMR is needed*
6. **TopNAirlineDelayMR** will run only if the no of reducer files are more than 1 and the –f is set to true, default is false.

**Run**:

1. Go to HADOOP\_HOME:

2. run command:

***./bin/hadoop jar data-pipeline-challenge/Swami-UnityChallenge /target/unity-mr.jar com.unity.challenge.mapreduce.driver.MapreduceDriver -i input -o output -n 100***

1. Arguments

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Description** | **Default Value /Required** | **Notes** |
| -i | input | Required | Input Local Folder |
| -o | outputfolder | Required | Output Local Folder |
| -n | Number of Top Records | 100 | 100/1000 |
| -f | forceExecution | false | False - Will execute the only when MR1 is successful and only 1 reducer output file is generated from MR1.  True- will execute MR2 even if MR1 generates 1 reducer part file |

**Test:**

1. Create test cases using mrunit and other test cases using TestNG
2. Also tested the output with data from 2000 and 2008. The data I validated was matching by loading into mysql and verifying as well. The <origin-destination>,<average>

|  |
| --- |
| 1. CMI-SPI,575 2. VPS-DHN,506 3. EWR-JFK,383.6 4. BIS-FAR,318 5. SUX-OMA,318 6. ELP-MFE,316 7. MDW-JFK,281 8. PSP-SAN,276 9. BFL-SBA,274 10. ACY-MYR,252 11. LAS-PSC,249 12. GJT-SBA,246 13. MCI-SGF,245 14. OAK-FAT,233.3333 15. ONT-BOS,231 16. LNK-COS,227.5 17. BTR-AUS,223 18. IND-LEX,218 19. ONT-PSP,217.1667 20. JAX-CMH,217 21. PHF-RDU,216 22. LAS-SGU,212.5 23. XNA-MCN,212 24. ATW-DSM,210 25. BHM-VPS,197 26. GSP-MLB,195 27. SAT-JFK,195 28. AVP-BUF,195 29. DCA-SHV,194 30. ACT-SJT,189 31. SHV-TYR,187 32. PFN-GPT,183 33. ORF-BOI,180 34. JFK-GSP,179 35. ACV-RDD,176 36. GSP-RSW,175 37. ORF-BDL,175 38. SJC-FAT,174.5 39. TXK-TYR,170.5 40. GRR-FLL,169 41. TWF-EKO,169 42. TYS-SDF,166.5 43. PSC-MFR,164 44. FMN-ABQ,164 45. PSP-CLD,163 46. RAP-BIS,161 47. SMF-FAT,161 48. OMA-FSD,160 49. BOI-BZN,160 50. BIL-BZN,158.5 51. BFL-FAT,158 52. BTV-LGA,156.5 53. HPN-SWF,150 54. OGD-SLC,148.4 55. FAT-BOI,148 56. ACT-TYR,148 57. VCT-ILE,146 58. FAT-RNO,145 59. LAX-BUR,145 60. RDD-MFR,144 61. OAK-SMF,144 62. MSO-BZN,144 63. RDM-MFR,143 64. FAR-BIS,142 65. CLL-ILE,141 66. BWI-AVP,140 67. TYS-ROA,140 68. SAV-RDU,139 69. MKE-MCN,139 70. ROA-BWI,138 71. MDT-MCN,138 72. TWF-BOI,136.8571 73. ICT-MCI,136.5 74. IDA-JAC,135.6667 75. SBA-MRY,135 76. BFL-SBP,135 77. ORF-PBI,135 78. CHA-BNA,134 79. MDT-JFK,133.5 80. BHM-JFK,133 81. BDL-MHT,133 82. MRY-FAT,132.5 83. LNK-OMA,131.5 84. MSO-FCA,131 85. CKB-LEX,130 86. COD-CPR,130 87. VPS-PFN,130 88. MTJ-GJT,129.5 89. BIL-COD,129.3333 90. RNO-EWR,128.1538 91. LEX-TYS,128 92. LGA-JFK,128 93. LFT-MGM,126 94. LIT-GSO,125 95. RAP-CPR,124 96. JFK-BGM,123.2667 97. SMF-RNO,122.5 98. HTS-MCN,122 99. CYS-DEN,122 100. AVP-ABE,122 |
|  |