Question 1

Results

Small	Medium	Large
+--------------------------+-----+ |name |count| +--------------------------+-----+ |Southwest Airlines Co. |12027| |Delta Air Lines Inc. |5897 | |United Airlines |5260 | |Continental Air Lines Inc.|3434 | |AirTran |2212 | +--------------------------+-----+	+--------------------------+------+ |name |count | +--------------------------+------+ |Southwest Airlines Co. |118663| |Delta Air Lines Inc. |58006 | |United Airlines |53298 | |Continental Air Lines Inc.|34145 | |AirTran |21702 | +--------------------------+------+	+--------------------------+-------+ |name |count | +--------------------------+-------+ |Southwest Airlines Co. |1190203| |Delta Air Lines Inc. |579509 | |United Airlines |535073 | |Continental Air Lines Inc.|341940 | |AirTran |219510 | +--------------------------+-------+

Unoptimized Code

Code Description

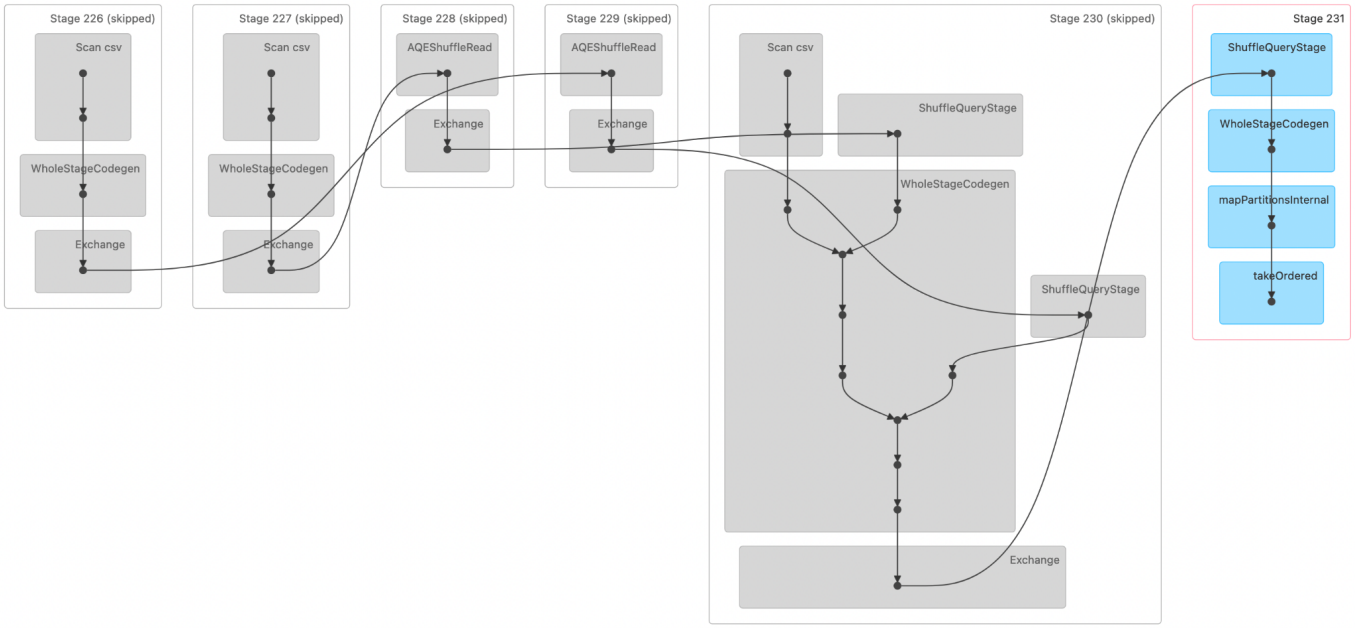
The pyspark code was made by following the given SQL chronologically by joining the relations first, then filtering out data based on the country being the USA and the manufacturer being Boeing.

Explain Output

```
 == Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
 +- TakeOrderedAndProject(limit=5, orderBy=[count#5157L DESC NULLS LAST],
 output=[name#74,count#5157L])
    +- HashAggregate(keys=[name#74], functions=[finalmerge_count(merge count#5161L) AS
 count(1)#5156L], output=[name#74, count#5157L])
       +- Exchange hashpartitioning(name#74, 1), ENSURE_REQUIREMENTS, [plan_id=20991]
          +- HashAggregate(keys=[name#74], functions=[partial_count(1) AS count#5161L],
 output=[name#74, count#5161L])
             +- Project [name#74]
                +- SortMergeJoin [ tail_number#206], [tail_number#46], Inner
                   :- Sort [ tail_number#206 ASC NULLS FIRST], false, 0
                   :  +- Exchange hashpartitioning( tail_number#206, 1),
 ENSURE_REQUIREMENTS, [plan_id=20983]
                   :     +- Project [ tail_number#206, name#74]
                   :        +- SortMergeJoin [carrier_code#201], [carrier_code#73], Inner
                   :           :- Sort [carrier_code#201 ASC NULLS FIRST], false, 0
                   :           :  +- Exchange hashpartitioning(carrier_code#201, 1),
 ENSURE_REQUIREMENTS, [plan_id=20975]
                   :           :     +- Filter (isnotnull(carrier_code#201) AND isnotnull(
 tail_number#206))
                   :           :        +- FileScan csv [carrier_code#201, tail_number#206]
 Batched: false, DataFilters: [isnotnull(carrier_code#201), isnotnull( tail_number#206)],
 Format: CSV, Location: InMemoryFileIndex(1
 paths)[dbfs:/FileStore/tables/ontimeperformance_flights_large.csv], PartitionFilters: [],
 PushedFilters: [IsNotNull(carrier_code), IsNotNull( tail_number)], ReadSchema:
 struct<carrier_code:string, tail_number:string>
```

```
                  :           +- Sort [carrier_code#73 ASC NULLS FIRST], false, 0
                  :              +- Exchange hashpartitioning(carrier_code#73, 1),
ENSURE_REQUIREMENTS, [plan_id=20976]
                  :                 +- Project [carrier_code#73, name#74]
                  :                    +- Filter ((isnotnull(country#75) AND (country#75 =
United States)) AND isnotnull(carrier_code#73))
                  :                       +- FileScan csv
[carrier_code#73,name#74,country#75] Batched: false, DataFilters: [isnotnull(country#75),
(country#75 = United States), isnotnull(carrier_code#73)], Format: CSV, Location:
InMemoryFileIndex(1 paths)[dbfs:/FileStore/tables/ontimeperformance_airlines.csv],
PartitionFilters: [], PushedFilters: [IsNotNull(country), EqualTo(country,United States),
IsNotNull(carrier_code)], ReadSchema:
struct<carrier_code:string,name:string,country:string>
                  +- Sort [tail_number#46 ASC NULLS FIRST], false, 0
                     +- Exchange hashpartitioning(tail_number#46, 1), ENSURE_REQUIREMENTS,
[plan_id=20984]
                        +- Project [tail_number#46]
                           +- Filter ((isnotnull(manufacturer#47) AND (manufacturer#47 =
BOEING)) AND isnotnull(tail_number#46))
                              +- FileScan csv [tail_number#46,manufacturer#47] Batched:
false, DataFilters: [isnotnull(manufacturer#47), (manufacturer#47 = BOEING),
isnotnull(tail_number#46)], Format: CSV, Location: InMemoryFileIndex(1
paths)[dbfs:/FileStore/tables/ontimeperformance_aircrafts.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(manufacturer), EqualTo(manufacturer,BOEING),
IsNotNull(tail_number)], ReadSchema: struct<tail_number:string,manufacturer:string>
```

DAG Visualisation



SparkMetrics

	Small	Medium	Large
Number of Stages	7	7	8
Number of Tasks	11	21	29
Executor Run Time	5765 (6 s)	74468 (1.2 min)	411530 (6.9 min)
Records Read	145437	1430682	14287795
Shuffle Records Read	4785	1433119	4819

Optimized Code

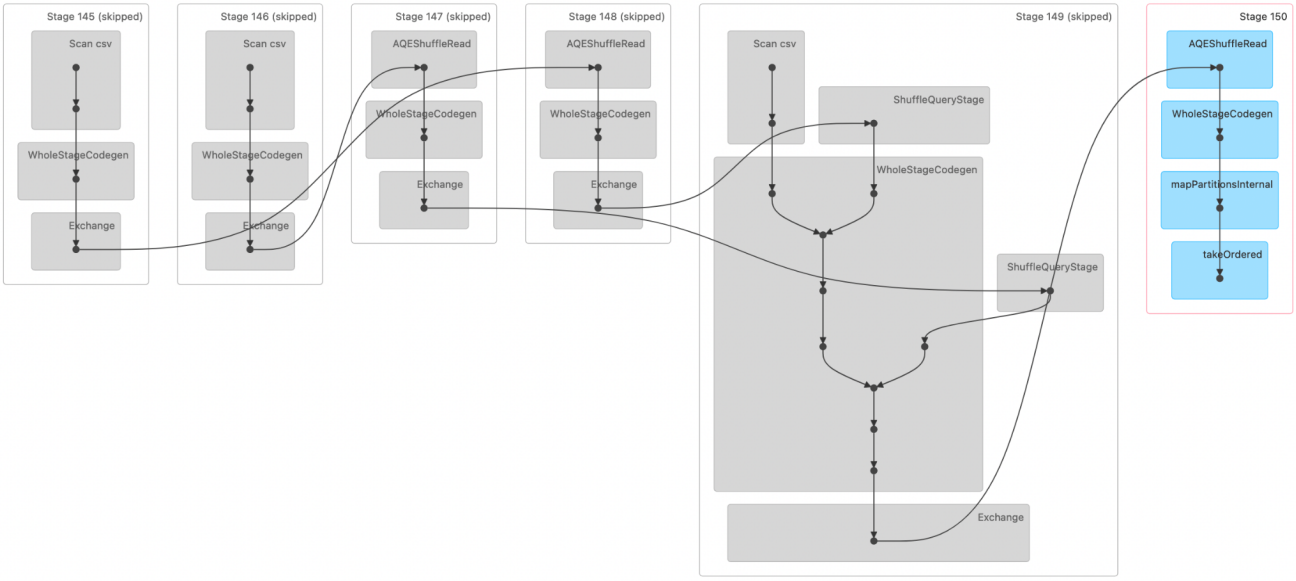
Code Description

The optimized code was made by changing the orders of operations around. Here, the filter and selections were done first in order to reduce the overall number of data needed for joining so that the query is quicker and a broadcast hash join was used so that the filtered datasets is fitted into the worker nodes' memory rather than having data being shuffled around.

Explain Output

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- TakeOrderedAndProject(limit=5, orderBy=[count#2908L DESC NULLS LAST],
output=[name#74,count#2908L])
   +- HashAggregate(keys=[name#74], functions=[finalmerge_count(merge count#2912L) AS
count(1)#2907L])
      +- Exchange hashpartitioning(name#74, 1), ENSURE_REQUIREMENTS, [plan_id=11820]
         +- HashAggregate(keys=[name#74], functions=[partial_count(1) AS count#2912L])
            +- Project [name#74]
               +- BroadcastHashJoin [ tail_number#140], [tail_number#46], Inner,
BuildRight, false
                  :- Project [ tail_number#140, name#74]
                  :  +- BroadcastHashJoin [carrier_code#135], [carrier_code#73], Inner,
BuildRight, false
                  :     :- Filter (isnotnull(carrier_code#135) AND isnotnull(
tail_number#140))
                  :     :  +- FileScan csv [carrier_code#135, tail_number#140] Batched:
false, DataFilters: [isnotnull(carrier_code#135), isnotnull( tail_number#140)], Format:
CSV, Location: InMemoryFileIndex(1
paths)[dbfs:/FileStore/tables/ontimeperformance_flights_medium.csv], PartitionFilters:
[], PushedFilters: [IsNotNull(carrier_code), IsNotNull( tail_number)], ReadSchema:
struct<carrier_code:string, tail_number:string>
                  :     +- Exchange SinglePartition, EXECUTOR_BROADCAST,
[plan_id=11811]
                  :        +- Project [carrier_code#73, name#74]
                  :           +- Exchange SinglePartition, REPARTITION_BY_COL,
[plan_id=11800]
                  :              +- Filter ((isnotnull(country#75) AND (country#75 =
United States)) AND isnotnull(carrier_code#73))
                  :                 +- FileScan csv
[carrier_code#73,name#74,country#75] Batched: false, DataFilters:
[isnotnull(country#75), (country#75 = United States), isnotnull(carrier_code#73)],
Format: CSV, Location: InMemoryFileIndex(1
paths)[dbfs:/FileStore/tables/ontimeperformance_airlines.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(country), EqualTo(country,United States),
IsNotNull(carrier_code)], ReadSchema:
struct<carrier_code:string,name:string,country:string>
                  +- Exchange SinglePartition, EXECUTOR_BROADCAST, [plan_id=11815]
                     +- Project [tail_number#46]
                        +- Exchange SinglePartition, REPARTITION_BY_COL,
[plan_id=11804]
                           +- Filter ((isnotnull(manufacturer#47) AND (manufacturer#47
= BOEING)) AND isnotnull(tail_number#46))
                              +- FileScan csv [tail_number#46,manufacturer#47] Batched:
false, DataFilters: [isnotnull(manufacturer#47), (manufacturer#47 = BOEING),
isnotnull(tail_number#46)], Format: CSV, Location: InMemoryFileIndex(1
paths)[dbfs:/FileStore/tables/ontimeperformance_aircrafts.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(manufacturer), EqualTo(manufacturer,BOEING),
IsNotNull(tail_number)], ReadSchema: struct<tail_number:string,manufacturer:string>
```

DAG Visualisation

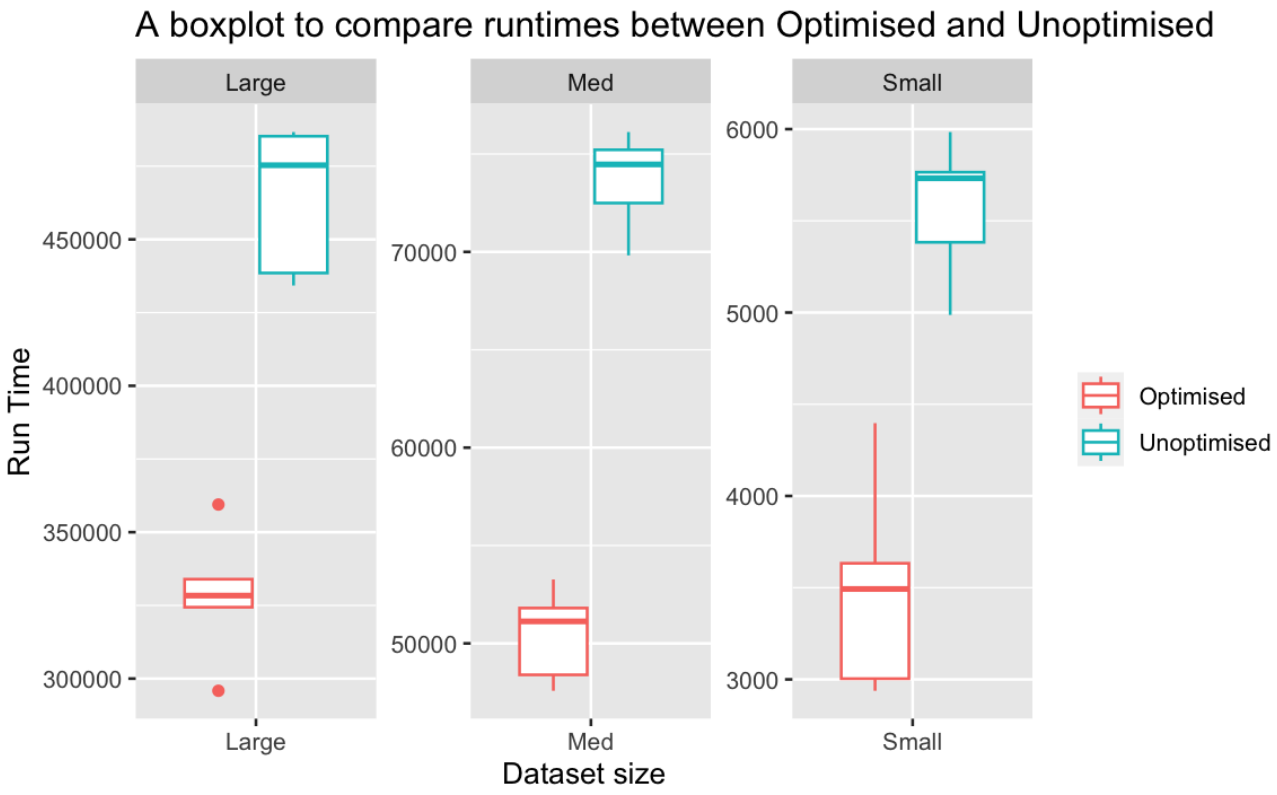


SparkMetrics

	Small	Medium	Large
Number of Stages	6	6	6
Number of Tasks	8	13	13
Executor Run Time	4397 (4 s)	51124 (51 s)	328346 (5.5 min)
Records Read	145437	1430682	14287795
Shuffle Records Read	2405	2437	2439

Comparison

The unoptimised code used SortMergeJoin which has high I/O and computational costs as it requires sorting and shuffling. In contrast, the optimized code uses a broadcast hash join as this helps with efficiency through copying the smaller dataset to each worker node which also minimizes the amount of data being shuffled. Moreover, switching the orders of filtering and joining also help decrease the run time to 5.5 minutes from 6.2 minutes, as reduces the number of stages and tasks in dataset size, with more efficiency in the large dataset as it went from eight stages to six, compared to the smaller and medium seven to six change. Moreover, the number of records had massively reduced especially on the medium dataset the number of shuffled records read was more than the number of records in the unoptimised code. This could be due to some other internal optimization of databricks having a higher threshold and thus could be applied to the large dataset only.
We have ran the two queries over the three dataset each with 5 times. Therefore to make a boxplot to compare the runtime between optimised query and unoptimised one. We can see that there is a large difference between the runtimes in each dataset when the code is optimised compared to the unoptimised version.



Question 2

Results

Small

```
Chicago O'Hare International\t6252.0\t[(United Airlines, 30%),(American Airlines Inc.,
26%),(American Eagle Airlines Inc., 23%),(Independence Air, 11%),(Skywest Airlines Inc., 5%)]

Cincinnati Northern Kentucky Intl\t1349.0\t[(PSA Airlines Inc., 47%),(Delta Air Lines Inc.,
33%),(Independence Air, 16%),(American Eagle Airlines Inc., 2%),(Atlantic Southeast Airlines,
0%)]

Dallas-Fort Worth International\t3477.0\t[(American Airlines Inc., 41%),(American Eagle Airlines
Inc., 26%),(Atlantic Southeast Airlines, 8%),(Air Tanzania, 7%),(United Airlines, 6%)]

Denver Intl\t1357.0\t[(United Airlines, 63%),(Skywest Airlines Inc., 27%),(American Airlines
Inc., 5%),(Delta Air Lines Inc., 1%),(Alaska Airlines Inc., 1%)]

Gen Edw L Logan Intl\t1685.0\t[(American Eagle Airlines Inc., 39%),(Continental Air Lines Inc.,
16%),(PSA Airlines Inc., 13%),(American Airlines Inc., 7%),(Independence Air, 5%)]

George Bush Intercontinental\t1941.0\t[(Continental Air Lines Inc., 54%),(JetSuite Air,
34%),(Delta Air Lines Inc., 9%),(Skywest Airlines Inc., 0%),(American Airlines Inc., 0%)]

Newark Intl\t1865.0\t[(JetSuite Air, 39%),(Continental Air Lines Inc., 32%),(American Airlines
Inc., 7%),(Independence Air, 4%),(United Airlines, 4%)]
```

```
Philadelphia Intl\t1491.0\t[(US Airways, 66%),(AirTran, 11%),(American Airlines Inc.,
6%),(United Airlines, 6%),(Delta Air Lines Inc., 5%)]

Phoenix Sky Harbor International\t1417.0\t[(Southwest Airlines Co., 55%),(Sparrow Aviation,
27%),(Skywest Airlines Inc., 6%),(American Airlines Inc., 4%),(Air Tanzania, 4%)]

William B Hartsfield-Atlanta Intl\t3934.0\t[(Delta Air Lines Inc., 55%),(Atlantic Southeast
Airlines, 22%),(AirTran, 11%),(PSA Airlines Inc., 7%),(United Airlines, 0%)]
```

Medium

```
Chicago O'Hare International \t 44704.0 \t [(United Airlines, 27%),(American Airlines Inc.,
26%),(American Eagle Airlines Inc., 21%),(Independence Air, 12%),(Skywest Airlines Inc., 5%)]

Cincinnati Northern Kentucky Intl \t 15621.0 \t [(PSA Airlines Inc., 46%),(Delta Air Lines Inc.,
32%),(Independence Air, 17%),(JetSuite Air, 2%),(American Eagle Airlines Inc., 0%)]

Dallas-Fort Worth International \t 31131.0 \t [(American Airlines Inc., 48%),(American Eagle
Airlines Inc., 19%),(Atlantic Southeast Airlines, 17%),(Delta Air Lines Inc., 3%),(Skywest
Airlines Inc., 2%)]

George Bush Intercontinental \t 14666.0 \t [(Continental Air Lines Inc., 37%),(JetSuite Air,
32%),(American Airlines Inc., 9%),(Skywest Airlines Inc., 4%),(Delta Air Lines Inc., 4%)]

Los Angeles International \t 17323.0 \t [(Southwest Airlines Co., 25%),(United Airlines,
17%),(American Airlines Inc., 14%),(Skywest Airlines Inc., 13%),(Alaska Airlines Inc., 8%)]

McCarran International \t 16182.0 \t [(Southwest Airlines Co., 50%),(Sparrow Aviation,
15%),(United Airlines, 10%),(American Airlines Inc., 7%),(US Airways, 3%)]

Newark Intl \t 14189.0 \t [(JetSuite Air, 40%),(Continental Air Lines Inc., 30%),(Delta Air
Lines Inc., 7%),(American Airlines Inc., 6%),(US Airways, 3%)]

Philadelphia Intl \t 16067.0 \t [(US Airways, 66%),(American Airlines Inc., 7%),(Southwest
Airlines Co., 5%),(Delta Air Lines Inc., 4%),(United Airlines, 3%)]

Phoenix Sky Harbor International \t 15599.0 \t [(Southwest Airlines Co., 41%),(Sparrow Aviation,
36%),(Alaska Airlines Inc., 7%),(American Airlines Inc., 3%),(United Airlines, 2%)]

William B Hartsfield-Atlanta Intl \t 42558.0 \t [(Delta Air Lines Inc., 45%),(Atlantic Southeast
Airlines, 21%),(AirTran, 18%),(PSA Airlines Inc., 9%),(American Airlines Inc., 1%)]
```

Large

```
Chicago O'Hare International \t 503434.0 \t (United Airlines, 29%),(American Airlines Inc.,
24%),(American Eagle Airlines Inc., 21%),(Independence Air, 11%),(Skywest Airlines Inc., 4%)

Cincinnati Northern Kentucky Intl \t 166842.0 \t (PSA Airlines Inc., 52%),(Delta Air Lines Inc.,
23%),(Independence Air, 15%),(American Eagle Airlines Inc., 3%),(JetSuite Air, 2%)

Dallas-Fort Worth International \t 309314.0 \t (American Airlines Inc., 50%),(American Eagle
Airlines Inc., 21%),(Atlantic Southeast Airlines, 13%),(Delta Air Lines Inc., 4%),(United
Airlines, 1%)

George Bush Intercontinental \t 130282.0 \t (Continental Air Lines Inc., 46%),(JetSuite Air,
31%),(Skywest Airlines Inc., 4%),(American Airlines Inc., 3%),(United Airlines, 2%)

Los Angeles International \t 151066.0 \t (Southwest Airlines Co., 23%),(United Airlines,
16%),(American Airlines Inc., 15%),(Skywest Airlines Inc., 11%),(Delta Air Lines Inc., 7%)
```

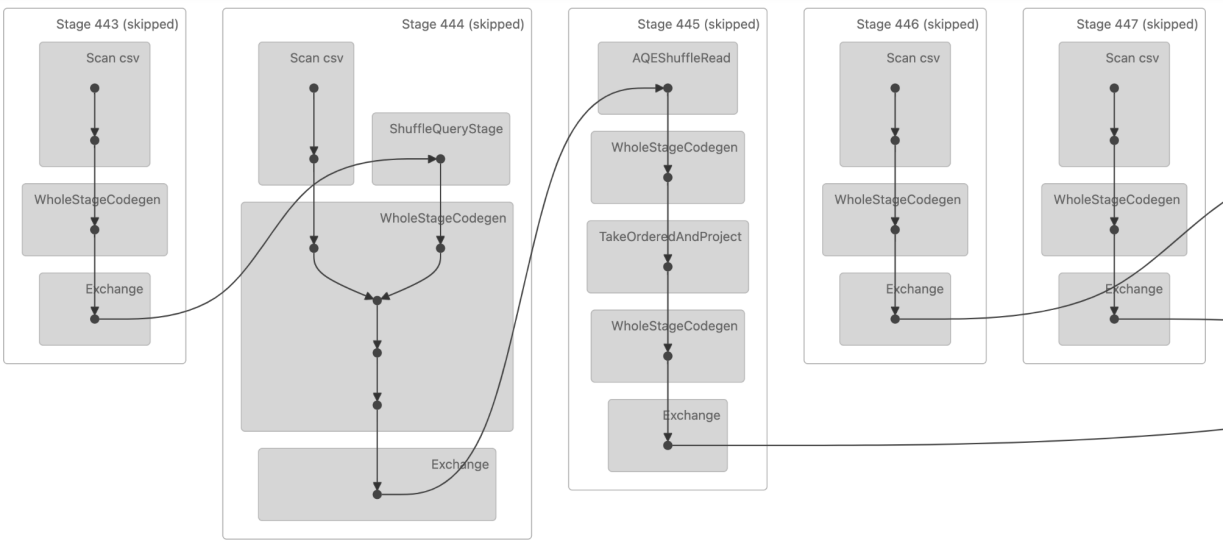
```
McCarran International \t 161953.0 \t (Southwest Airlines Co., 47%),(Sparrow Aviation,
16%),(Continental Air Lines Inc., 12%),(American Airlines Inc., 5%),(United Airlines, 5%)

Newark Intl \t 161077.0 \t (JetSuite Air, 39%),(Continental Air Lines Inc., 29%),(American
Airlines Inc., 4%),(Delta Air Lines Inc., 4%),(United Airlines, 4%)

Philadelphia Intl \t 151407.0 \t (US Airways, 66%),(United Airlines, 5%),(Southwest Airlines
Co., 5%),(Delta Air Lines Inc., 4%),(American Airlines Inc., 4%)

Phoenix Sky Harbor International \t 151805.0 \t (Southwest Airlines Co., 44%),(Sparrow Aviation,
37%),(United Airlines, 3%),(American Airlines Inc., 3%),(Alaska Airlines Inc., 3%)

William B Hartsfield-Atlanta Intl \t 441231.0 \t (Delta Air Lines Inc., 48%),(Atlantic Southeast
Airlines, 21%),(AirTran, 15%),(PSA Airlines Inc., 7%),(American Airlines Inc., 1%)
```

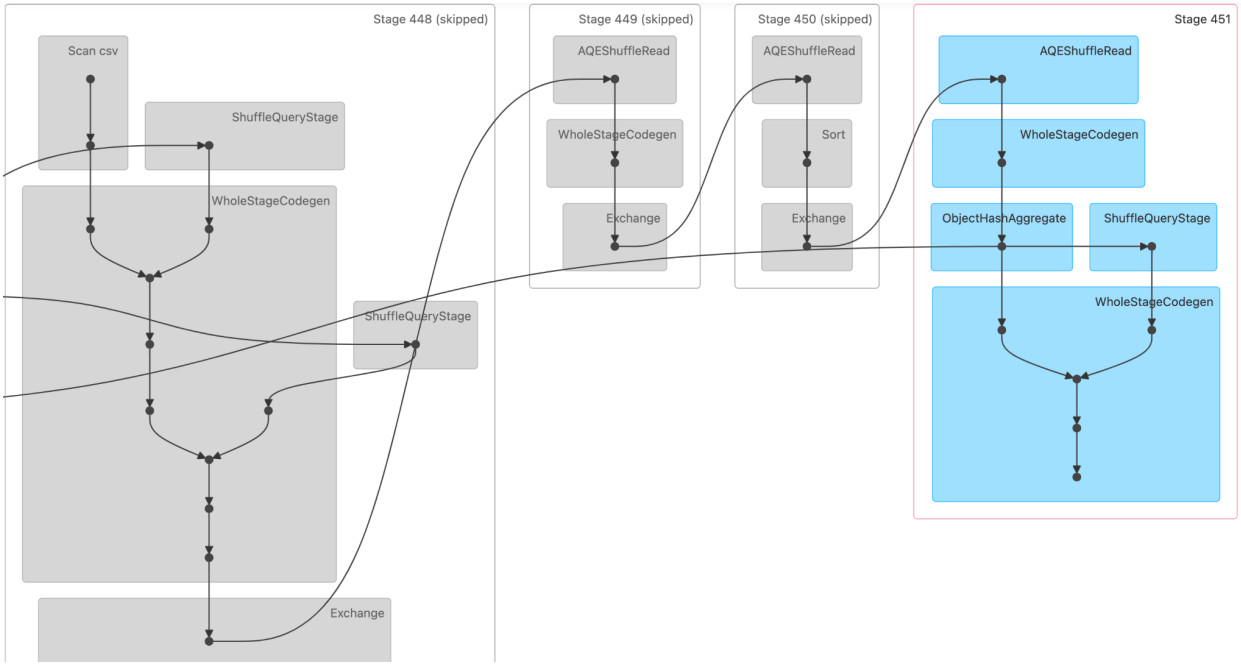
Unoptimized Code

Code Description

The code for this question is a huge function to get the results. The first part is to calculate the departure delay time. Then we join the flights dataframe and airport dataframe to get the top 10 airports with a given year of 2004 and country “USA”. The next step is to join the airline dataframe and airports dataframe together, in order to filter out the airlines in the top 10 airports. After we get the dataframe that contains all data we need, we just group them by airports name, sort by the delay times in descending order, and limit 5 outputs for each airport. Then what we have now is a dataframe containing 10 airports, and each airport contains 5 airlines contribute best to delay times. The next step is to calculate the percentage of the contribution of each airline. We just use the airline delay times divided by the airport's total delay times and convert the floats into percentage format. The final step is to convert the dataframe into the format we want.

DAG Visualisation





SparkMetrics

	Small	Medium	Large
Number of Stages	24	24	24
Number of Tasks	40	80	80

Executor Run Time	58152 (58 s)	682860 (11 min)	5316700 (1.5 h)
Records Read	707200	6963588	69519488
Shuffle Records Read	22230	55273	22578

Explain Output

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [concat(airport_name#114,  , cast(total_delay#536 as string),    , airlines#731) AS output#757]
   +- SortMergeJoin [airport_name#114], [airport_name#748], Inner
      :- Sort [airport_name#114 ASC NULLS FIRST], false, 0
      :  +- ObjectHashAggregate(keys=[airport_name#114], functions=[collect_list(airlines#718, 0, 0)])
      :     +- Project [airport_name#114, airlines#718]
      :        +- Project [airport_name#114, concat((, name#64, , , cast(cast((CASE WHEN (id#698L <= 4) THEN
(total_delay#651 / 6252.0) WHEN ((id#698L > 4) AND (id#698L <= 9)) THEN (total_delay#651 / 1349.0) WHEN ((id#698L > 9) AND
(id#698L <= 14)) THEN (total_delay#651 / 3477.0) WHEN ((id#698L > 14) AND (id#698L <= 19)) THEN (total_delay#651 / 1357.0)
WHEN ((id#698L > 19) AND (id#698L <= 24)) THEN (total_delay#651 / 1685.0) WHEN ((id#698L > 24) AND (id#698L <= 29)) THEN
(total_delay#651 / 1941.0) WHEN ((id#698L > 29) AND (id#698L <= 34)) THEN (total_delay#651 / 1865.0) WHEN ((id#698L > 34)
AND (id#698L <= 39)) THEN (total_delay#651 / 1491.0) WHEN ((id#698L > 39) AND (id#698L <= 44)) THEN (total_delay#651 /
1417.0) WHEN ((id#698L > 44) AND (id#698L <= 49)) THEN (total_delay#651 / 3934.0) END * 100.0) as int) as string), %, )) AS
airlines#718]
      :           +- Filter isnotnull(airport_name#114)
      :              +- Project [airport_name#114, name#64, total_delay#651, monotonically_increasing_id() AS id#698L]
      :                 +- Filter (rank#657 <= 5)
      :                    +- RunningWindowFunction [airport_name#114, name#64, total_delay#651, row_number()
windowspecdefinition(airport_name#114, total_delay#651 DESC NULLS LAST, specifiedwindowframe(RowFrame,
unboundedpreceding$(), currentrow$())) AS rank#657], [airport_name#114], [total_delay#651 DESC NULLS LAST], false
      :                       +- Sort [airport_name#114 ASC NULLS FIRST, total_delay#651 DESC NULLS LAST], false, 0
      :                          +- Exchange hashpartitioning(airport_name#114, 1), ENSURE_REQUIREMENTS, [plan_id=1062]
      :                             +- Sort [airport_name#114 ASC NULLS FIRST, total_delay#651 DESC NULLS LAST], true, 0
      :                                +- Exchange rangepartitioning(airport_name#114 ASC NULLS FIRST, total_delay#651 DESC
NULLS LAST, 1), ENSURE_REQUIREMENTS, [plan_id=1059]
      :                                   +- HashAggregate(keys=[airport_name#114, name#64], functions=[finalmerge_sum(merge
sum#767) AS sum(time_diff_minutes#433)#650])
      :                                      +- Exchange hashpartitioning(airport_name#114, name#64, 1),
ENSURE_REQUIREMENTS, [plan_id=1056]
      :                                         +- HashAggregate(keys=[airport_name#114, name#64],
functions=[partial_sum(time_diff_minutes#433) AS sum#767])
      :                                            +- Project [airport_name#114, name#64, time_diff_minutes#433]
      :                                               +- SortMergeJoin [ origin#262], [airport_code#113], Inner
      :                                                  :- Sort [ origin#262 ASC NULLS FIRST], false, 0
      :                                                  :  +- Exchange hashpartitioning( origin#262, 1),
ENSURE_REQUIREMENTS, [plan_id=1048]
      :                                                  :     +- Project [name#64,  origin#262, time_diff_minutes#433]
      :                                                  :        +- SortMergeJoin [carrier_code#63], [carrier_code#259],
Inner
      :                                                  :           :- Sort [carrier_code#63 ASC NULLS FIRST], false, 0
      :                                                  :           :  +- Exchange hashpartitioning(carrier_code#63, 1),
ENSURE_REQUIREMENTS, [plan_id=1040]
      :                                                  :           :     +- Filter isnotnull(carrier_code#63)
      :                                                  :           :        +- FileScan csv [carrier_code#63,name#64]
Batched: false, DataFilters: [isnotnull(carrier_code#63)], Format: CSV, Location: InMemoryFileIndex(1
paths)[dbfs:/FileStore/tables/ontimeperformance_airlines.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(carrier_code)], ReadSchema: struct<carrier_code:string,name:string>
      :                                                  :           +- Sort [carrier_code#259 ASC NULLS FIRST], false, 0
      :                                                  :              +- Exchange hashpartitioning(carrier_code#259, 1),
ENSURE_REQUIREMENTS, [plan_id=1041]
      :                                                  :                 +- Project [carrier_code#259,  origin#262,
(cast(cast((gettimestamp(format_string(%04d,  actual_departure_time#267), HHmm, TimestampType, Some(Etc/UTC), false) -
gettimestamp(format_string(%04d,  scheduled_depature_time#265), HHmm, TimestampType, Some(Etc/UTC), false)) as int) as
double) / 60.0) AS time_diff_minutes#433]
      :                                                  :                    +- Filter (((((((isnotnull(
actual_departure_time#267) AND isnotnull( scheduled_depature_time#265)) AND isnotnull( flight_date#261)) AND (
actual_departure_time#267 >=  scheduled_depature_time#265)) AND atleastnnonnulls(15, carrier_code#259,  flight_number#260,
flight_date#261,  origin#262,  destination#263,  tail_number#264,  scheduled_depature_time#265,  scheduled_arrival_time#266,
actual_departure_time#267,  actual_arrival_time#268,  distance#269, ephemeralsubstring(cast( flight_date#261 as string), 1,
4), gettimestamp(format_string(%04d,  scheduled_depature_time#265), HHmm, TimestampType, Some(Etc/UTC), false),
gettimestamp(format_string(%04d,  actual_departure_time#267), HHmm, TimestampType, Some(Etc/UTC), false),
(cast(cast((gettimestamp(format_string(%04d,  actual_departure_time#267), HHmm, TimestampType, Some(Etc/UTC), false) -
```

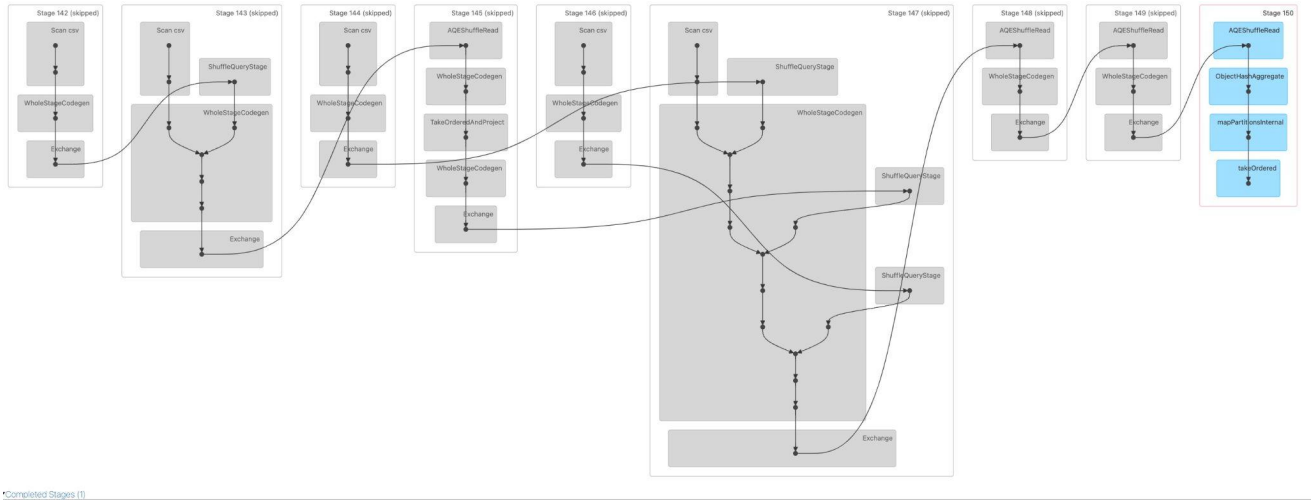
```
gettimestamp(format_string(%04d,  scheduled_depature_time#265), HHmm, TimestampType, Some(Etc/UTC), false)) as int) as
double) / 60.0))) AND (cast(ephemeralsubstring(cast( flight_date#261 as string), 1, 4) as int) = 2004)) AND
isnotnull(carrier_code#259)) AND isnotnull( origin#262))
      :                                                  :                       +- FileScan csv [carrier_code#259,
flight_number#260, flight_date#261, origin#262, destination#263, tail_number#264, scheduled_depature_time#265,
scheduled_arrival_time#266, actual_departure_time#267, actual_arrival_time#268, distance#269] Batched: false, DataFilters:
[isnotnull( actual_departure_time#267), isnotnull( scheduled_depature_time#265), isnotnull( fligh..., Format: CSV, Location:
InMemoryFileIndex(1 paths)[dbfs:/FileStore/tables/ontimeperformance_flights_small.csv], PartitionFilters: [], PushedFilters:
[IsNotNull( actual_departure_time), IsNotNull( scheduled_depature_time), IsNotNull( flight_date),..., ReadSchema:
struct<carrier_code:string, flight_number:int, flight_date:date, origin:string, destination:strin...
      :                                                  +- Sort [airport_code#113 ASC NULLS FIRST], false, 0
      :                                                     +- Exchange hashpartitioning(airport_code#113, 1),
ENSURE_REQUIREMENTS, [plan_id=1049]
      :                                                        +- Project [airport_code#113, airport_name#114]
      :                                                           +- Filter (((isnotnull(country#117) AND airport_name#114
IN (Chicago O'Hare International,William B Hartsfield-Atlanta Intl,Dallas-Fort Worth International,George Bush
Intercontinental,Newark Intl,Gen Edw L Logan Intl,Philadelphia Intl,Phoenix Sky Harbor International,Denver Intl,Cincinnati
Northern Kentucky Intl)) AND (country#117 = USA)) AND isnotnull(airport_code#113))
      :                                                              +- FileScan csv
[airport_code#113,airport_name#114,country#117] Batched: false, DataFilters: [isnotnull(country#117), airport_name#114 IN
(Chicago O'Hare International,William B Hartsfield-A..., Format: CSV, Location: InMemoryFileIndex(1
paths)[dbfs:/FileStore/tables/ontimeperformance_airports.csv], PartitionFilters: [],
PushedFilters: [IsNotNull(country), In(airport_name, [Chicago O'Hare International,Cincinnati Northern Kentucky ...,
ReadSchema: struct<airport_code:string,airport_name:string,country:string>
      +- Sort [airport_name#748 ASC NULLS FIRST], false, 0
         +- Filter (isnotnull(airport_name#748) AND airport_name#748 IN (Chicago O'Hare International,William B
Hartsfield-Atlanta Intl,Dallas-Fort Worth International,George Bush Intercontinental,Newark Intl,Gen Edw L Logan
Intl,Philadelphia Intl,Phoenix Sky Harbor International,Denver Intl,Cincinnati Northern Kentucky Intl))
            +- TakeOrderedAndProject(limit=10, orderBy=[total_delay#536 DESC NULLS LAST],
output=[airport_name#748,total_delay#536])
               +- HashAggregate(keys=[airport_name#748], functions=[finalmerge_sum(merge sum#541) AS
sum(time_diff_minutes#433)#535])
                  +- Exchange hashpartitioning(airport_name#748, 1), ENSURE_REQUIREMENTS, [plan_id=1080]
                     +- HashAggregate(keys=[airport_name#748], functions=[partial_sum(time_diff_minutes#433) AS sum#541])
                        +- Project [time_diff_minutes#433, airport_name#748]
                           +- SortMergeJoin [ origin#739], [airport_code#747], Inner
                              :- Sort [ origin#739 ASC NULLS FIRST], false, 0
                              :  +- Exchange hashpartitioning( origin#739, 1), ENSURE_REQUIREMENTS, [plan_id=1072]
                              :     +- Project [ origin#739, (cast(cast((gettimestamp(format_string(%04d,
actual_departure_time#744), HHmm, TimestampType, Some(Etc/UTC), false) - gettimestamp(format_string(%04d,
scheduled_depature_time#742), HHmm, TimestampType, Some(Etc/UTC), false)) as int) as double) / 60.0) AS
time_diff_minutes#433]
                              :        +- Filter ((((((isnotnull( actual_departure_time#744) AND isnotnull(
scheduled_depature_time#742)) AND isnotnull( flight_date#738)) AND ( actual_departure_time#744 >=
scheduled_depature_time#742)) AND atleastnnonnulls(15, carrier_code#736,  flight_number#737,  flight_date#738,  origin#739,
destination#740,  tail_number#741,  scheduled_depature_time#742,  scheduled_arrival_time#743,  actual_departure_time#744,
actual_arrival_time#745,  distance#746, ephemeralsubstring(cast( flight_date#738 as string), 1, 4),
gettimestamp(format_string(%04d,  scheduled_depature_time#742), HHmm, TimestampType, Some(Etc/UTC), false),
gettimestamp(format_string(%04d,  actual_departure_time#744), HHmm, TimestampType, Some(Etc/UTC), false),
(cast(cast((gettimestamp(format_string(%04d,  actual_departure_time#744), HHmm, TimestampType, Some(Etc/UTC), false) -
gettimestamp(format_string(%04d,  scheduled_depature_time#742), HHmm, TimestampType, Some(Etc/UTC), false)) as int) as
double) / 60.0))) AND (cast(ephemeralsubstring(cast( flight_date#738 as string), 1, 4) as int) = 2004)) AND isnotnull(
origin#739))
                              :           +- FileScan csv [carrier_code#736, flight_number#737, flight_date#738, origin#739,
destination#740, tail_number#741, scheduled_depature_time#742, scheduled_arrival_time#743, actual_departure_time#744,
actual_arrival_time#745, distance#746] Batched: false, DataFilters: [isnotnull( actual_departure_time#744), isnotnull(
scheduled_depature_time#742), isnotnull( fligh..., Format: CSV, Location: InMemoryFileIndex(1
paths)[dbfs:/FileStore/tables/ontimeperformance_flights_small.csv], PartitionFilters: [], PushedFilters: [IsNotNull(
actual_departure_time), IsNotNull( scheduled_depature_time), IsNotNull( flight_date),..., ReadSchema:
struct<carrier_code:string, flight_number:int, flight_date:date, origin:string, destination:strin...
                              +- Sort [airport_code#747 ASC NULLS FIRST], false, 0
                                 +- Exchange hashpartitioning(airport_code#747, 1), ENSURE_REQUIREMENTS, [plan_id=1073]
                                    +- Project [airport_code#747, airport_name#748]
                                       +- Filter ((isnotnull(country#751) AND (country#751 = USA)) AND
isnotnull(airport_code#747))
                                          +- FileScan csv [airport_code#747,airport_name#748,country#751] Batched: false,
DataFilters: [isnotnull(country#751), (country#751 = USA), isnotnull(airport_code#747)], Format: CSV, Location:
InMemoryFileIndex(1 paths)[dbfs:/FileStore/tables/ontimeperformance_airports.csv], PartitionFilters: [], PushedFilters:
[IsNotNull(country), EqualTo(country,USA), IsNotNull(airport_code)], ReadSchema:
struct<airport_code:string,airport_name:string,country:string>
```

Optimised Code

Code Description

This optimised query is built based upon the unoptimised one where the same method is used to calculate the delayed departure time for each flight. In order to minimise the amount of data for the joining operation, we apply filter() method on country of the airports and year of the flights before joining. Then, to calculate the total departure delay for an airport “airport_delay”, we utilised the `Window` function from `pyspark.sql.window` which does a `sum()` operation over a window partitioned by “airport_name. Then we join the airline data frame and use the same method, a window partitioned by “airport_name”, “airport_delay”, and “airline_name”, to calculate the total departure delay for the airlines at each airport “airline_delay”. This is followed by a projection to the four attributes mentioned above with distinct selections and reordering by airport_delay and airline_delay. We then concatenate the airline name with the calculated percentage contributed to airport total delay. To make a list of top 5 contributing airlines, we utilise collect_list() methods and slice to the first five element of each row. Finally, we return the data frame limited by 10.

DAG Visualisation



SparkMetrics

	Small	Medium	Large
Number of Stages	7	7	7
Number of Tasks	9	14	14
Executor Run Time	6599 (7 s)	85267 (1.4 min)	713934 (12 min)
Records Read	178045	1742142	17381117
Shuffle Records Read	9408	49396	434128

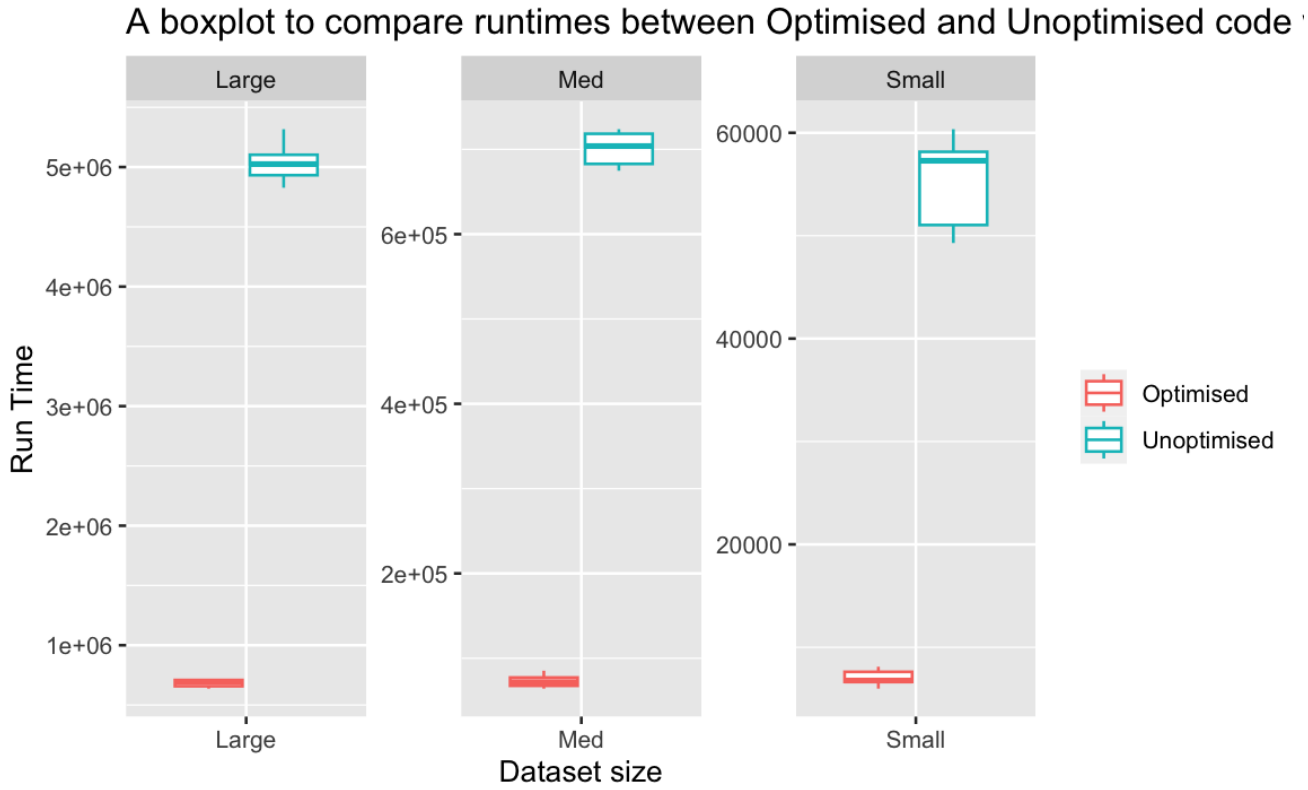
Explain Output

```
== Physical Plan ==
AdaptiveSparkPlan (60)
+- == Final Plan ==
   TakeOrderedAndProject (35)
   +- ObjectHashAggregate (34)
      +- AQEShuffleRead (33)
         +- ShuffleQueryStage (32), Statistics(sizeInBytes=1941.3 KiB, rowCount=2.14E+4,
isRuntime=true)
            +- Exchange (31)
               +- * Project (30)
                  +- * Project (29)
                     +- * Sort (28)
                        +- AQEShuffleRead (27)
                           +- ShuffleQueryStage (26), Statistics(sizeInBytes=1910.5 KiB,
rowCount=2.14E+4, isRuntime=true)
                              +- Exchange (25)
                                 +- Window (24)
                                    +- * Sort (23)
                                       +- * Project (22)
                                          +- * BroadcastHashJoin Inner BuildRight (21)
                                             :- * Filter (16)
                                             :  +- Window (15)
                                             :     +- Sort (14)
                                             :        +- AQEShuffleRead (13)
                                             :           +- ShuffleQueryStage (12),
Statistics(sizeInBytes=1437.7 KiB, rowCount=2.14E+4, isRuntime=true)
                                             :              +- Exchange (11)
                                             :                 +- * Project (10)
                                             :                    +- * BroadcastHashJoin Inner
BuildRight (9)
                                             :                       :- * Project (3)
                                             :                       :  +- * Filter (2)
                                             :                       :     +- Scan csv  (1)
                                             :                       +- ShuffleQueryStage (8),
Statistics(sizeInBytes=169.9 KiB, rowCount=3.37E+3, isRuntime=true)
                                             :                          +- Exchange (7)
                                             :                             +- * Project (6)
                                             :                                +- * Filter (5)
                                             :                                   +- Scan csv  (4)
                                             +- ShuffleQueryStage (20),
Statistics(sizeInBytes=28.4 KiB, rowCount=540, isRuntime=true)
                                                +- Exchange (19)
                                                   +- * Filter (18)
                                                      +- Scan csv  (17)
```

Comparison

The excutor run time of unoptimized code is much longer than optimized code. The time for unoptimized code on flights small is 58s, and the time for optimized is 7s, which is huge improvement. There are some reasons why this happened. First one is SortMergeJoin was utilised in the unoptimized code but optimized code use BroadcastHashJoin. The advantage of BroadcastHashJoin is its speed, it can be much faster than SortMergeJoin for smaller datasets because it avoids the overhead of sorting and shuffling data. The SortMergeJoin can handle larger datasets that wouldn't fit into memory for a BroadcastHashJoin. But it comes with additional overhead because the data needs to be sorted and shuffled, which can be a relatively slow

operation. Also, the optimised method moved the filter operations before joining which minimises the amount of data for BroadcastHashJoinAs we can see from the SparkMetrix above. The number of tasks and stages has been significantly reduced, from 24 to 7 and from 40 to 9. And the records read reduced from 707200 to 178045.
The optimised method utilised the window function instead of `groupby()` and aggregation to calculate the accumulative delay for airports and airlines. Using a window function has been shown 2-time improvement in speed than using `groupby()`. However it has to be followed by a distinct selection for window function because of the replicated rows.
Also, in terms of finding the top five airlines that contribute the most to an airport, the slicing method used in the optimised method is considered to have less time complexity than the unoptimised one.



We have ran three datasets size 5 times, and got the boxplot of their run time. As we can see from the boxplot, the difference of excutor run time of Large dataset between unoptimised and optimised is much higher than others.