U.S. Domestic Passenger Aviation Industry Investigation Using Apache Spark and GeoSpark

Tom Lee, *University of Wisconsin, Madison*
August 16, 2019

Introduction

This study uses Apache Spark and flight summary data from the U.S. Department of Transportation to investigate market conditions in the U.S. domestic passenger aviation industry. Database design and implementation steps are discussed and detailed for Apache Spark with the Geospark package, using Scala and the data frames API. The database includes geospatial geometry fields and results are shown on maps created with QGIS.

Apache Spark

Apache Spark is an open source distributed general purpose cluster computing framework. It was originally developed by the University of California, Berkeley's AMP Lab and is now maintained by the Apache Software Foundation. GeoSpark is a package that extends Spark to store and manipulate geospatial data. This includes having a data format to be able to store and query geometries as well as tools for running calculations and joins on geometries.

Source data

The US Department of Transportation Air Carrier Statistics Database (T-100) includes monthly data on domestic and international flights. This dataset can be used to get the monthly counts of flights and passengers by carrier between airports and also includes information on aircraft type, seats and route distance. Additional tables include information for looking up the aircraft type name and manufacturer, as well as airport names and latitude, longitude coordinates.

Database design

The data model for this study was set up to have four entities: flight statistics, airports, carriers, and aircraft. A flight statistics is a monthly count of flights and passengers between two airports, by a specific carrier, using a specific aircraft type. Each flight statistic has an origin and a destination airport. Additional attributes for those airports are available in the airport entity table including the latitude/longitude coordinates and the market name. A market is ht major city that an airport serves. For example, New York, NY has three major airports — JFK, Newark (EWR), and LaGuardia (LGA). These three airport each have their own airport code and name, but they share the same market name of "New York, NY".

Each flight statistic has one aircraft type. The aircraft entity includes additional attributes for each aircraft type including the short name, long name, and manufacturer. Each flight statistic also has one carrier and the full carrier attributes are available in the carrier entity table.

The raw tables from the T-100 database are not normalized and include many values duplicated between records. To normalize the data and reduce data redundancy, I first loaded the raw tables and then transformed the raw data to new tables with redundant fields moved to other tables.

For example, the flight statistics table includes the counts of flights and passengers by route and carrier. This includes the airport code and airport name. The airport code and airport name have a one-to-one relationship where an airport code always has the same matching airport name. So to normalize the flight statistics table I removed the airport name column from the flight statistics table and added it to a table of airport data with one record per airport code. This reduces the size of the flight statistics table but still makes the data available by using a join to the airports table.

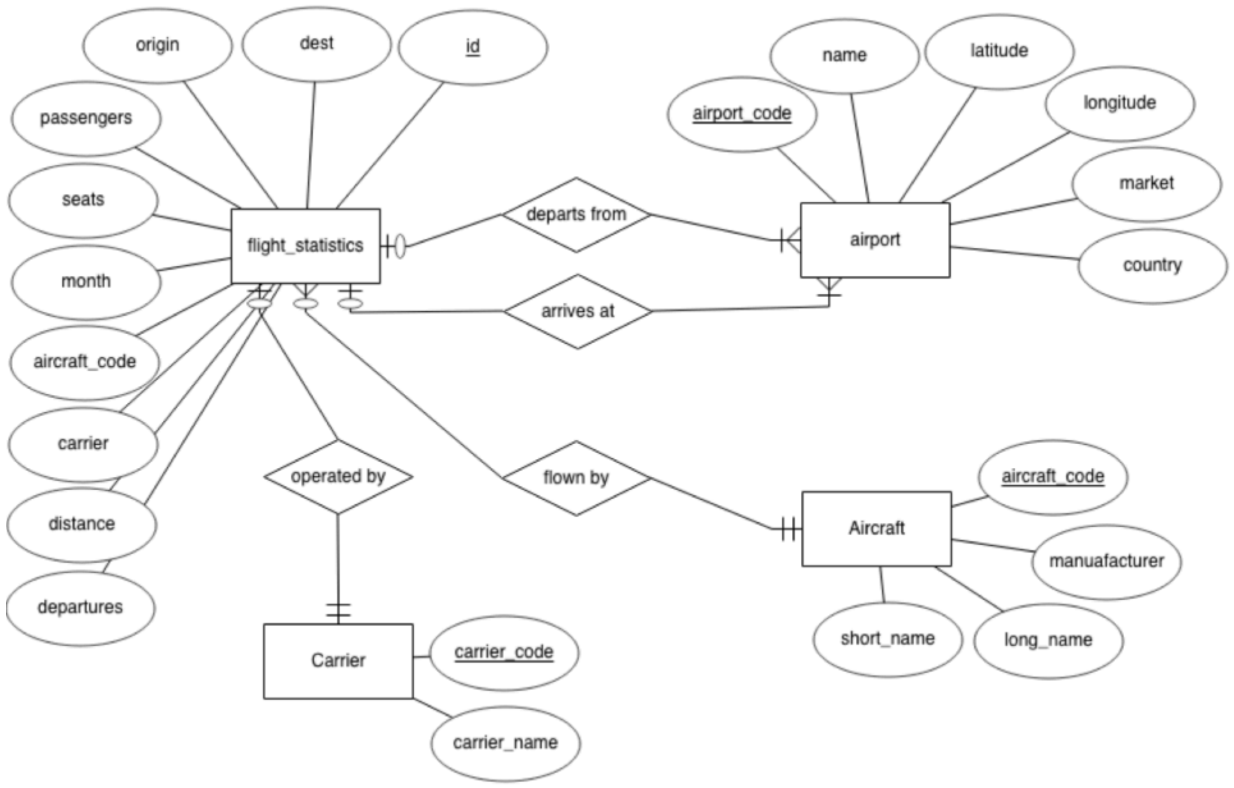

Figure 1. Entity Relationship diagram of data after normalization

The primary key for the flight statistics table is an added unique identifier integer “ID” that was added when the flight statistics table was normalized. Each flight statistic record has one carrier and multiple flight statistic records can have the same carrier. The primary key for the carrier table is the carrier code from the original source data, making the matching carrier field in the flight statistics table a foreign key. Each flight statistics record has one aircraft type and multiple flight statistics records can have the same aircraft type. The primary key in the aircraft table is the aircraft code from the original source data and the matching aircraft code field in the flight statistics table is a foreign key.

A flight statistic record has two airports, one that it originates from and one that is the destination. In the airport table, the airport code is the primary key. In the flight statistics table, the origin and destination fields are both foreign keys and both join to the airport code in the airport table. An airport entity has many origins and destinations in the flight statistics table.

The airport entity was also given a geometry field using the GeoSpark package. The geometry is created from the latitude and longitude fields and is stored in a field with its own geometry data type, similar to the PostGIS extension on Postgres. That geometry can then be used to join to other tables with geometries or to perform other geospatial calculations such as measuring the distance between two airports.

This analysis was all done in Apache Spark, which doesn’t have functionality for enforcing primary key and foreign key restraints, so they are omitted from the actual table creation code later shown. Informational referential integrity constraints such as primary keys and foreign keys have been shown to speed up joins in Spark and work is currently underway to add them (Delaney, 2017 & Kambhampati, 2018).

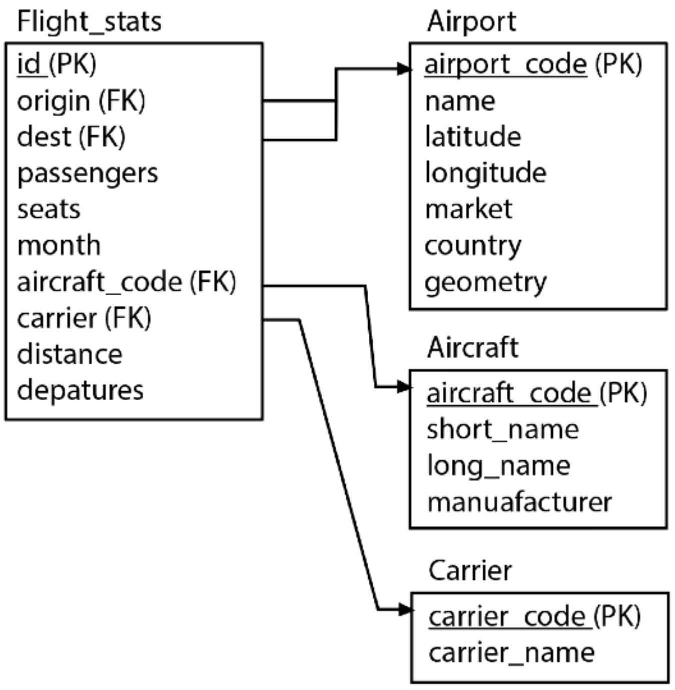

Figure 2. Relational schema diagram

Data loading and table creation

In Apache Spark, you first define the schema, then load the data into a new data frame using the schema as a parameter. For the flights table, the schema setup scala code for the original raw data from the DOT T-100 database is as follows. Many of the columns were skipped here for brevity but can be seen in appendix A.

```
// raw flight statistics schema
var flightsSchema = StructType(Array(
    StructField("DEPARTURES_SCHEDULED", DoubleType, true),
    StructField("DEPARTURES_PERFORMED", DoubleType, false),
    StructField("PAYLOAD", DoubleType, true),
    StructField("SEATS", DoubleType, false),
    StructField("PASSENGERS", DoubleType, false),
    StructField("FREIGHT", DoubleType, true),
    StructField("MAIL", DoubleType, true),
    StructField("DISTANCE", DoubleType, false),
   …
    StructField("QUARTER", IntegerType, true),
    StructField("MONTH", IntegerType, false),
    StructField("DISTANCE_GROUP", IntegerType, true),
    StructField("CLASS", StringType, true),
    StructField("DATA_SOURCE", StringType, true)
))
```

The raw flight statistics data is then loaded to a new data frame with:

```
val flightsFullDF = spark.read.format(“csv")
      .schema(flightsSchema)
      .option("header", “true")
      .load("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/
      Assignments/project/data/dot_airline/
      90313096_T_T100_SEGMENT_ALL_CARRIER/
      90313096_T_T100_SEGMENT_ALL_CARRIER_2018_All.csv"
      )

flightsFullDF.createOrReplaceTempView("flightsfulldf")
```

Then the data is transformed to the final data frame needed for analysis:

```
// flight statistics table
var flightsDF = flightsFullDF.where("ORIGIN_COUNTRY = 'US' and
DEST_COUNTRY = 'US'").selectExpr(
        "ORIGIN as origin",
        "DEST as dest",
        "PASSENGERS as passengers",
        "SEATS as seats",
        "MONTH as month",
        "AIRCRAFT_TYPE as aircraft_code",
        "CARRIER as carrier",
        "DISTANCE as distance",
        "DEPARTURES_PERFORMED as departures").
    withColumn("id",monotonicallyIncreasingId)
```

```
flightsDF.createOrReplaceTempView("flights")
```

The airports data is loaded using a similar method with the addition of creating a geometry field in the transformed data frame.

```
var airportDF = airportFullDF.selectExpr(
    "AIRPORT as airport_code",
    "DISPLAY_AIRPORT_NAME as airport_name",
    "LATITUDE as latitude",
    "LONGITUDE as longitude",
    "DISPLAY_CITY_MARKET_NAME_FULL as market",
    "AIRPORT_COUNTRY_NAME as country",
    "ST_Point(CAST(LONGITUDE AS Decimal(24,20)), CAST(LATITUDE AS 
Decimal(24,20))) as geometry"
).where("AIRPORT_IS_LATEST = 1”)

airportDF.createOrReplaceTempView("airport")
```

Analysis

Major carriers by passenger miles
The first round of analysis compares passenger miles for each airline to find the major carriers. Passenger miles is the count of passengers multiplied by the route length in miles. The analysis steps where mostly done using SQL which is available as part of the spark data frames API and doesn’t cause any reduction in performance. The following query gets the carrier names and sum of passenger miles, ordered with the carriers with highest passenger miles first. Full carrier names are being pulled from the carrier table via a join. The output can be viewed in the shell window using “.show()” but in this case is being saved to a csv by using “.write”.

```
// major airlines by passenger miles
spark.sql("""
    select
        carrier.carrier_name,
        cast(sum(passengers * distance) as long) as passenger_miles
    from flights
    join carrier
        on flights.carrier = carrier.carrier_code
    group by 1
    order by 2 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep",
",").option("header", "true").save("/Users/tomlee/OneDrive - UW-
Madison/Classes/GEOG574/Assignments/project/results/
carrier_passenger_miles.csv")
```

Major airports by passenger count
The next query gets the major airports by passenger count. The passenger count is the count of passengers that embarked or disembarked at the airport. The passenger count is being pulled from the flights table and the airport name is being pulled from the airport table via a join.

```
// major airports by passenger count
spark.sql("""
    select
        airport.airport_code,
        airport.airport_name,
        cast(sum(flights.passengers) as long) as passenger_count
    from airport
    join flights
        on airport.airport_code = flights.origin or
airport.airport_code = flights.dest
    group by 1, 2
    order by 3 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep",
",").option("header", "true").save("/Users/tomlee/OneDrive - UW-
Madison/Classes/GEOG574/Assignments/project/results/
airport_passengers.csv”)
```

Major markets by passenger count
An alternative analysis of the major airports by passenger count is the major markets by passenger count. A market is a greater city area and can have multiple airports. For example, the market New York, NY has three major airports: JFK, EWR, and LGA. This is more indicative of the number of passengers traveling to or from an area. Passengers may be able to travel to multiple airports that are in the same market, but the assumption is that passengers choose to travel to a specific market and are less concerned about which airport they use within a market. This query pulls the market name from the airport table and the passenger counts from the flights table.

```
// major markets by passenger count
spark.sql("""
    select
        airport.market,
        cast(sum(flights.passengers) as long) as passenger_count
    from airport
    join flights
        on airport.airport_code = flights.origin or
airport.airport_code = flights.dest
    group by 1
    order by 2 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep",
",").option("header", "true").save("/Users/tomlee/OneDrive - UW-
Madison/Classes/GEOG574/Assignments/project/results/
city_passengers.csv")
```

Major market to market routes by passenger miles
This query is again more focused on markets than airports and is an adaption of the major routes by passenger miles query.

```
// major market routes by passenger miles
spark.sql("""
    select
        origin.market as origin_market,
        dest.market dest_market,
        cast(sum(flights.passengers * flights.distance) as long) as
passenger_miles
    from flights
    join airport as origin
        on flights.origin = origin.airport_code
    join airport as dest
        on flights.dest = dest.airport_code
    group by 1, 2
    order by 3 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep",
",").option("header", "true").save("/Users/tomlee/OneDrive - UW-
Madison/Classes/GEOG574/Assignments/project/results/
route_city_passenger_miles.csv")
```

Market to market route competition
The get a measure of competition for each route I calculated the share of passengers that the carrier with the greatest share has. Lower competition is indicated where the major carrier has a higher share. For example, if the major carrier on a route has a 100% share, then it indicates that there are not other carriers flying passengers on that route. On highly competitive routes, the major carriers have a share of around 30%.

For the competition analysis, I first set up a new data frame with the route share of each carrier as a row. I then queried this data frame to get the aggregated statistics for each individual research question. The count of passengers and the share of route passengers per carrier were each calculated using window functions.

```
// market to market competition
var marketRouteCompDF = spark.sql("""
    with carrier_pct_of_route_passengers_table as (
        select distinct
            origin_airport.market as origin_market,
            dest_airport.market as dest_market,
            round(100.0 * sum(flights.passengers) over (partition by
origin_airport.market, dest_airport.market, carrier) /
sum(flights.passengers) over (partition by origin_airport.market,
dest_airport.market), 2) as carrier_pct_of_route_passengers
        from flights
        join airport as origin_airport
            on flights.origin = origin_airport.airport_code
        join airport as dest_airport
            on flights.dest = dest_airport.airport_code
        order by 1, 2
    )
    select
        origin_market,
        dest_market,
        max(carrier_pct_of_route_passengers) as max_carrier_share
    from carrier_pct_of_route_passengers_table
    group by 1, 2
    order by 1, 2
“"")

marketRouteCompDF.createOrReplaceTempView(“marketRouteComp")

// market to market route competition
spark.sql("""
    select
        origin_airport.market as origin_market,
        dest_airport.market dest_market,
        competition.max_carrier_share,
        cast(sum(flights.passengers * flights.distance) as long) as
passenger_miles
    from flights
    join airport as origin_airport
        on flights.origin = origin_airport.airport_code
    join airport as dest_airport
        on flights.dest = dest_airport.airport_code
    join marketRouteComp as competition
        on origin_airport.market = competition.origin_market
        and dest_airport.market = competition.dest_market
    group by 1, 2, 3
    order by 4 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep",
",").option("header", "true").save("/Users/tomlee/OneDrive - UW-
Madison/Classes/GEOG574/Assignments/project/results/
market_route_competition.csv")
```

Competition on a passenger mile basis was also calculated using a similar method (see appendix A).

To create maps in QGIS, data was exported with the geometry saved as a field with the data in Well Known Text format using the ST_GeomFromWKT function as shown below. You can also connect QGIS to Apache Spark using a JDBC connection. Queries for airports and competition with geometry can be found in Appendix A.

```
// major routes by passenger miles with linestring geometry
spark.sql("""
    select
        flights.origin,
        flights.dest,
        cast(
            ST_GeomFromWKT('LINESTRING(' || origin.longitude || ' ' ||
origin.latitude || ', ' || dest.longitude || ' ' || dest.latitude ||
')')
        as varchar(100)) as geom,
        cast(sum(passengers * distance) as long) as passenger_miles,
        cast(sum(passengers) as long) as passengers
    from flights
    join airport as origin on flights.origin = origin.airport_code
    join airport as dest on flights.dest = dest.airport_code
    group by 1, 2, 3
    order by 4 desc
    limit 10000
""").write.format("csv").mode("overwrite").option("sep",
",").option("header", "true").save("/Users/tomlee/OneDrive - UW-
Madison/Classes/GEOG574/Assignments/project/results/
route_passenger_count_geom.csv")
```

Results

Major carriers by passenger miles

Southwest has the greatest passenger miles followed closely by American, Delta, and United.

Rank	Carrier	Passenger miles
1	Southwest Airlines Co.	128,600,984,310
2	American Airlines Inc.	128,505,881,430
3	Delta Air Lines Inc.	121,803,316,002
4	United Air Lines Inc.	108,486,948,174
5	Alaska Airlines Inc.	44,035,166,268
6	JetBlue Airways	40,149,372,369
7	Spirit Air Lines	27,954,476,174
8	Frontier Airlines Inc.	19,732,569,370
9	SkyWest Airlines Inc.	19,504,833,900
10	Allegiant Air	12,310,130,288

Table 1. Major carriers by passenger miles

Major airports by passenger count

Atlanta has the highest count of passengers at 92.2 million passengers embarking or disembarking in 2018. Fan favorite Dane County Regional Airport in Madison, WI had 2.1 million passengers.

Rank	Airport code	Airport name	Passenger count
1	ATL	Hartsfield-Jackson Atlanta International	92,254,219
2	ORD	Chicago O'Hare International	66,552,404
3	LAX	Los Angeles International	60,549,128
4	DEN	Denver International	60,437,401
5	DFW	Dallas/Fort Worth International	57,491,836
6	LAS	McCarran International	44,432,222
7	SEA	Seattle/Tacoma International	42,905,818
8	SFO	San Francisco International	42,195,162
9	PHX	Phoenix Sky Harbor International	41,712,280
10	CLT	Charlotte Douglas International	41,592,965

Table 2. Major airports by passenger count

Major market by passenger count
A market is a greater city area and can be served by multiple airports. Atlanta has the greatest passenger count at 92.2 million passengers in 2019, matching the Atlanta airport passenger count as Atlanta has only one major airport. New York City did not have any of its three major airports in the top ten airports list, but as a combined market gets the number 2 rank with 91.8 million passengers.

Rank	Market	Passenger count
1	Atlanta, GA (Metropolitan Area)	92,264,104
2	New York City, NY (Metropolitan Area)	91,773,624
3	Chicago, IL	87,611,949
4	Los Angeles, CA (Metropolitan Area)	84,956,909
5	Dallas/Fort Worth, TX	73,783,374
6	San Francisco, CA (Metropolitan Area)	68,280,335
7	Washington, DC (Metropolitan Area)	63,940,146
8	Denver, CO	60,437,665
9	Miami, FL (Metropolitan Area)	48,145,311
10	Houston, TX	45,605,294

Table 3. Major market by passenger count

Major airport to airport routes by passenger count
LAX to San Francisco and San Francisco to LAX had the highest passenger counts for a route at 2 million passenger each way in 2018.

Rank	Origin	Dest	Passenger count
1	LAX	SFO	2,006,236
2	SFO	LAX	1,989,114
3	LAX	JFK	1,799,164
4	JFK	LAX	1,789,829
5	LGA	ORD	1,607,187
6	ORD	LGA	1,598,374
7	LAS	LAX	1,523,904
8	LAX	LAS	1,512,539
9	ATL	MCO	1,505,486
10	MCO	ATL	1,495,775

Table 4. Major airport to airport routes by passenger count

Major airport to airport routes by passenger miles
Including passenger miles when comparing routes gives a better picture of revenue generated by a route. Longer routes have higher costs and should return higher revenue. LAX to JFK and return had the greatest passenger miles with passengers flying around 4.4 billion miles each way on that route in 2018. This is because that route had a high passenger count and also because it is one of the longest routes. LAX to Honolulu took the third and forth spots due to the very high route length.

Rank	Origin	Dest	Passenger miles
1	LAX	JFK	4,452,930,900
2	JFK	LAX	4,429,826,775
3	HNL	LAX	2,971,526,364
4	LAX	HNL	2,965,647,564
5	SFO	JFK	2,656,926,222
6	JFK	SFO	2,656,463,328
7	SFO	EWR	2,480,385,780
8	LAX	ORD	2,469,572,016
9	EWR	SFO	2,445,068,295
10	ORD	LAX	2,437,145,824

Table 5. Major airport to airport routes by passenger miles

The following map shows the count of passengers for each airport to airport route, as well as the count of passengers embarking and disembarking at each airport. You can see that Atlanta has one of the highest passenger counts and that many flights originate from the denser areas of population in the north east and on the west coast.

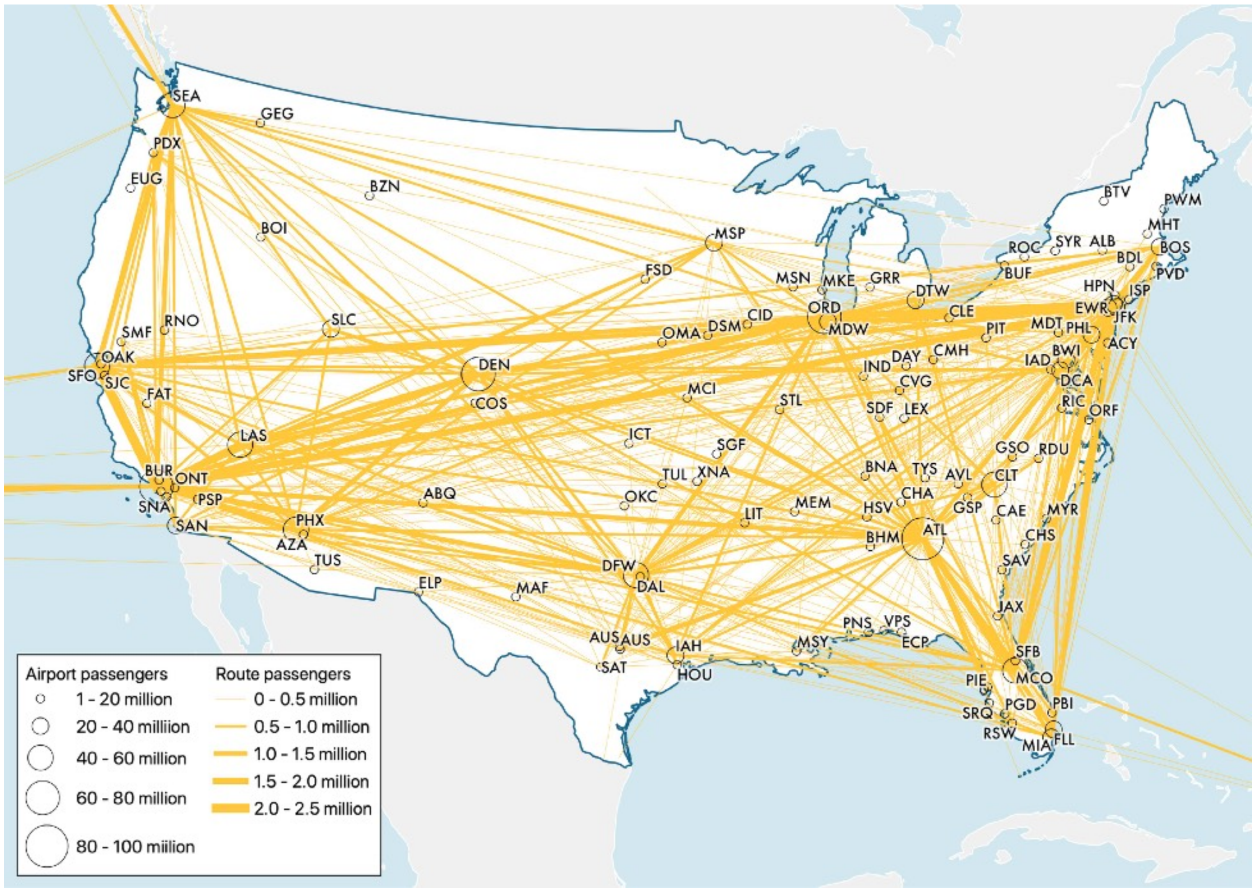

Figure 3. Airport to airport passenger count

Major market to market routes by passenger miles
Aggregating the route passenger miles to the market level gives a better indication of passengers traveling to or from a city. Los Angeles to New York had the greatest passenger miles at 7.1 billion passenger miles each way.

Rank	Origin market	Destination market	Passenger miles
1	Los Angeles, CA (Metropolitan Area)	New York City, NY (Metropolitan Area	7,142,820,412
2	New York City, NY (Metropolitan Area)	Los Angeles, CA (Metropolitan Area)	7,109,505,016
3	San Francisco, CA (Metropolitan Area)	New York City, NY (Metropolitan Area	5,765,783,865
4	New York City, NY (Metropolitan Area)	San Francisco, CA (Metropolitan Area	5,706,669,222
5	Miami, FL (Metropolitan Area)	New York City, NY (Metropolitan Area	4,489,493,179
6	New York City, NY (Metropolitan Area)	Miami, FL (Metropolitan Area)	4,466,401,230
7	Los Angeles, CA (Metropolitan Area)	Chicago, IL	3,645,442,525
8	Chicago, IL	Los Angeles, CA (Metropolitan Area)	3,617,355,006
9	Honolulu, HI	Los Angeles, CA (Metropolitan Area)	3,055,955,326
10	Los Angeles, CA (Metropolitan Area)	Honolulu, HI	3,050,503,969

Table 6. Market to market passenger miles

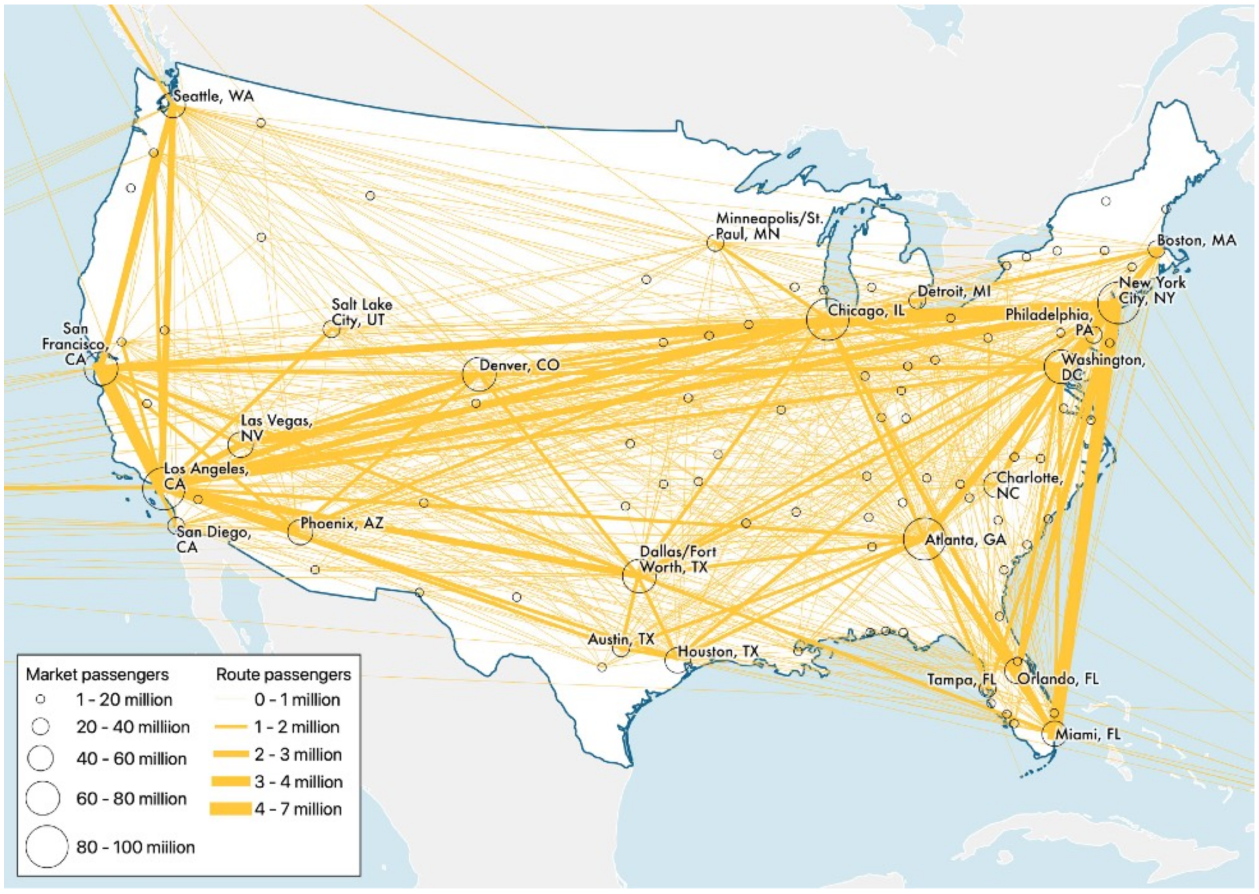

Figure 4. Market to market passenger counts

Market to market competition
The following table shows the major carrier's share of passengers on market to market routes with the routes with the highest passenger miles listed first.

Rank	Origin market	Destination market	Major carrier share	Passenger mile
1	Los Angeles, CA	New York City, NY	26.11%	7,142,820,412
2	New York City, NY	Los Angeles, CA	26.09%	7,109,505,016
3	San Francisco, CA	New York City, NY	37.21%	5,765,783,865
4	New York City, NY	San Francisco, CA	37.22%	5,706,669,222
5	Miami, FL	New York City, NY	26.1%	4,489,493,179
6	New York City, NY	Miami, FL	25.88%	4,466,401,230
7	Los Angeles, CA	Chicago, IL	37.38%	3,645,442,525
8	Chicago, IL	Los Angeles, CA	37.7%	3,617,355,006
9	Honolulu, HI	Los Angeles, CA	26.46%	3,055,955,326
10	Los Angeles, CA	Honolulu, HI	25.83%	3,050,503,969

Table 7. Market to market route competition

The Los Angeles to New York route has 7.1 billion passenger miles and the carrier with the greatest passenger count on that route flies 26% of the route's passengers. This low share indicates that there is high competition for this route.

This data is better visualized on a map as shown below. Passenger counts are symbolized by the route line width and competition is symbolized by the route line color.

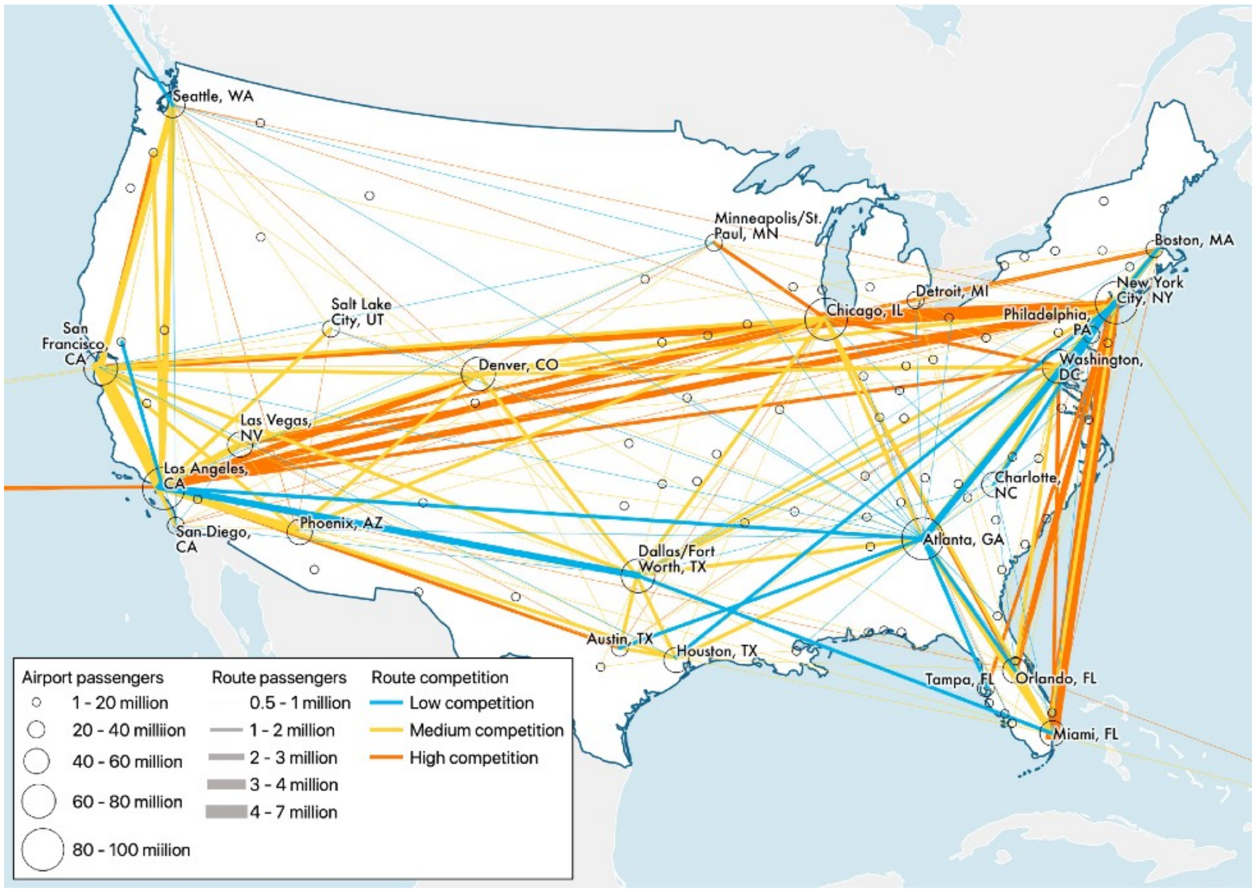

Figure 5. Market to market route competition

You can see that the routes that cross the country between Los Angleles, Chicago, and New York all have high competition and passenger miles. This indicates more choice of carriers for passengers on those routes and a higher level of difficulty for new carriers to enter the route profitably.

The route with the highest passenger count from Los Angeles to San Francisco only has moderate competition. Routes to and from Atlanta have consistently low competition, indicating less choice for Atlanta passengers and more opportunity for a new carrier if they were looking to fly to a new market.

Carrier market share
The following table shows the passenger miles for each carrier and the average share of each route for routes that the carrier operates on, on a passenger mile basis.

Rank	Carrier	Total passenger miles	Average route share
1	American Airlines Inc.	130,582,073,946	68.6%
2	Southwest Airlines Co.	129,323,307,996	70.98%
3	Delta Air Lines Inc.	121,803,530,413	63.95%
4	United Air Lines Inc.	110,064,184,284	61.58%
5	Alaska Airlines Inc.	44,035,166,268	49.22%
6	JetBlue Airways	40,149,372,369	51.29%
7	Spirit Air Lines	28,462,440,173	28.29%
8	Frontier Airlines Inc.	19,864,814,792	35.16%
9	SkyWest Airlines Inc.	19,862,488,160	44.7%
10	Allegiant Air	12,360,282,048	90.26%

Table 8. Carrier market share

American Airlines has the highest passenger miles at 130.6 billion passenger miles, followed closely by Southwest at 129.3 billion passenger miles. Both have have market share on their routes at 68.6% and 71.0% respectively.

JetBlue and Spirit both have comparatively low market share on their routes and therefore need to work harder to differentiate from the competition. Spirit differentiates with cut rate fares and Jet Blue focuses on better onboard experience compared to other major carriers.

Allegiant Air has the highest average route share out of the top ten carriers at over 90%. This means they Allegiant operates on routes with very few other carriers competing against them. Allegiant would be an attractive option for carriers looking to buy a smaller carrier based on route share.

Conclusion

This study aims to investigate the U.S. domestic aviation using Apache Spark with GeoSpark and passenger flight summary data from the U.S. Department of Transportation. Apache Spark is setup and operates in a slightly different method to a traditional RDMS but inclusion of SQL in the data frames API makes it easy for people with knowledge of SQL to query and manipulate the data.

The results of the study show that routes with high passenger miles tend to also have high competition, indicating better choice for consumers and less opportunity for new carriers to enter the routes. Overall, the four major carriers have much higher passenger miles for 2018 than the rest of the carriers and also have less competition on routes that they operate on.

References

US DOT (2019) Air Carrier Statistics Database (T-100), US Department of Transportation, Bureau of Transport Statistics. Downloaded from https://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=111

Delaney, I., Thalamati, S., Kambhampati, S., (2017) Informational Referential Integrity Constraints Support in Spark, Center fo Open-Source Data & AI Technologies, IBM. Downloaded from https://issues.apache.org/jira/browse/SPARK-19842 on 8/16/2019.

Delaney, I., Thalamati, S. (2017) Informational Referential Integrity Constraints Support in Apache Spark, presented at Spark Summit 2017 and viewable on Databricks website: https://databricks.com/session/informational-referential-integrity-constraints-support-in-apache-spark

Kambhampati, S., Delaney, I. (2018) Blog post: Performance enhancements in Spark using informational referential integrity constraints, IBM Developer Blog. Accessed at https://developer.ibm.com/blogs/performance-enhancements-in-spark-using-referential-integrity-constraints/ on 8/16/2019

Appendix A - Full code for setting up database and running queries

```
spark-shell --packages org.datasyslab:geospark:1.2.0,org.datasyslab:geospark-
sql_2.3:1.2.0,org.datasyslab:geospark-viz_2.3:1.2.0

// setup
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.serializer.KryoSerializer;
import org.apache.spark.sql.types.{StructField, StructType, DoubleType, IntegerType,
StringType}
import org.datasyslab.geospark.serde.GeoSparkKryoRegistrator
import org.datasyslab.geosparksql.utils.GeoSparkSQLRegistrator

val conf = new SparkConf()
conf.setAppName("GeoSparkApp") // Change this to a proper name
conf.setMaster("local[*]") // Delete this if run in cluster mode
// Enable GeoSpark custom Kryo serializer
conf.set("spark.serializer", classOf[KryoSerializer].getName)
conf.set("spark.kryo.registrator", classOf[GeoSparkKryoRegistrator].getName)

var sparkSession = SparkSession.builder().
    config("spark.serializer",classOf[KryoSerializer].getName).
    config("spark.kryo.registrator", classOf[GeoSparkKryoRegistrator].getName).
    master("local[*]").appName("myGeoSparkSQLdemo").getOrCreate()
GeoSparkSQLRegistrator.registerAll(sparkSession)

// raw flight statistics schema
var flightsSchema = StructType(Array(
    StructField("DEPARTURES_SCHEDULED", DoubleType, true),
    StructField("DEPARTURES_PERFORMED", DoubleType, false),
    StructField("PAYLOAD", DoubleType, true),
    StructField("SEATS", DoubleType, false),
    StructField("PASSENGERS", DoubleType, false),
    StructField("FREIGHT", DoubleType, true),
    StructField("MAIL", DoubleType, true),
    StructField("DISTANCE", DoubleType, false),
    StructField("RAMP_TO_RAMP", DoubleType, true),
    StructField("AIR_TIME", DoubleType, true),
    StructField("UNIQUE_CARRIER", StringType, true),
    StructField("AIRLINE_ID", IntegerType, true),
    StructField("UNIQUE_CARRIER_NAME", StringType, true),
    StructField("UNIQUE_CARRIER_ENTITY", StringType, true),
    StructField("REGION", StringType, true),
    StructField("CARRIER", StringType, false),
    StructField("CARRIER_NAME", StringType, true),
    StructField("CARRIER_GROUP", IntegerType, true),
    StructField("CARRIER_GROUP_NEW", IntegerType, true),
    StructField("ORIGIN_AIRPORT_ID", IntegerType, true),
    StructField("ORIGIN_AIRPORT_SEQ_ID", IntegerType, true),
    StructField("ORIGIN_CITY_MARKET_ID", IntegerType, true),
    StructField("ORIGIN", StringType, false),
    StructField("ORIGIN_CITY_NAME", StringType, true),
    StructField("ORIGIN_STATE_ABR", StringType, true),
    StructField("ORIGIN_STATE_FIPS", StringType, true),
    StructField("ORIGIN_STATE_NM", StringType, true),
    StructField("ORIGIN_COUNTRY", StringType, true),
    StructField("ORIGIN_COUNTRY_NAME", StringType, true),
    StructField("ORIGIN_WAC", IntegerType, true),
    StructField("DEST_AIRPORT_ID", IntegerType, true),
    StructField("DEST_AIRPORT_SEQ_ID", IntegerType, true),
    StructField("DEST_CITY_MARKET_ID", IntegerType, true),
    StructField("DEST", StringType, false),
    StructField("DEST_CITY_NAME", StringType, true),
    StructField("DEST_STATE_ABR", StringType, true),
    StructField("DEST_STATE_FIPS", StringType, true),
    StructField("DEST_STATE_NM", StringType, true),
    StructField("DEST_COUNTRY", StringType, true),
    StructField("DEST_COUNTRY_NAME", StringType, true),
    StructField("DEST_WAC", IntegerType, true),
    StructField("AIRCRAFT_GROUP", IntegerType, true),
    StructField("AIRCRAFT_TYPE", IntegerType, false),
    StructField("AIRCRAFT_CONFIG", IntegerType, true),
    StructField("YEAR", IntegerType, true),
    StructField("QUARTER", IntegerType, true),
    StructField("MONTH", IntegerType, false),
    StructField("DISTANCE_GROUP", IntegerType, true),
    StructField("CLASS", StringType, true),
    StructField("DATA_SOURCE", StringType, true)
))

val flightsFullDF = spark.read.format("csv").schema(flightsSchema).option("header",
"true").load("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
data/dot_airline/90313096_T_T100_SEGMENT_ALL_CARRIER/
90313096_T_T100_SEGMENT_ALL_CARRIER_2018_All.csv"
)

import org.apache.spark.sql.functions._

flightsFullDF.createOrReplaceTempView("flightsfulldf")
```

// flight statistics table

```
var flightsDF = flightsFullDF.where("ORIGIN_COUNTRY = 'US' and DEST_COUNTRY =
'US'").selectExpr(
        "ORIGIN as origin",
        "DEST as dest",
        "PASSENGERS as passengers",
        "SEATS as seats",
        "MONTH as month",
        "AIRCRAFT_TYPE as aircraft_code",
        "CARRIER as carrier",
        "DISTANCE as distance",
        "DEPARTURES_PERFORMED as departures").
    withColumn("id",monotonicallyIncreasingId)


flightsDF.createOrReplaceTempView("flights")
```

// carriers table

```
var carrierDF = flightsFullDF.selectExpr(
    "CARRIER as carrier_code",
    "CARRIER_NAME as carrier_name"
).distinct().orderBy("carrier_code")

carrierDF.createOrReplaceTempView("carrier")
```

// aircraft table

```
var aircraftSchema = StructType(Array(
    StructField("AC_TYPEID", IntegerType, false),
    StructField("AC_GROUP", StringType, true),
    StructField("SSD_NAME", StringType, true),
    StructField("MANUFACTURER", StringType, false),
    StructField("LONG_NAME", StringType, false),
    StructField("SHORT_NAME", StringType, false),
    StructField("BEGIN_DATE", StringType, true),
    StructField("END_DATE", StringType, true)
))

var aircraftFullDF = spark.read.format("csv").schema(aircraftSchema).option("header", 
"true").load("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
data/dot_airline/90313096_T_AIRCRAFT_TYPES/90313096_T_AIRCRAFT_TYPES_All_All.csv"
)
aircraftFullDF.createOrReplaceTempView("aircraftfulldf")

var aircraftDF = aircraftFullDF.selectExpr(
    "AC_TYPEID as aircraft_code",
    "SHORT_NAME as short_name",
    "LONG_NAME as long_name",
    "MANUFACTURER as manufacturer"
)
aircraftDF.createOrReplaceTempView("aircraft")
```

// airports

```
var airportSchema = StructType(Array(
    StructField("AIRPORT_SEQ_ID", IntegerType, true),
    StructField("AIRPORT_ID", IntegerType, true),
    StructField("AIRPORT", StringType, false),
    StructField("DISPLAY_AIRPORT_NAME", StringType, false),
    StructField("DISPLAY_AIRPORT_CITY_NAME_FULL", StringType, true),
    StructField("AIRPORT_WAC_SEQ_ID2", IntegerType, true),
    StructField("AIRPORT_WAC", IntegerType, true),
    StructField("AIRPORT_COUNTRY_NAME", StringType, false),
    StructField("AIRPORT_COUNTRY_CODE_ISO", StringType, true),
    StructField("AIRPORT_STATE_NAME", StringType, true),
    StructField("AIRPORT_STATE_CODE", StringType, true),
    StructField("AIRPORT_STATE_FIPS", IntegerType, true),
    StructField("CITY_MARKET_SEQ_ID", IntegerType, true),
    StructField("CITY_MARKET_ID", IntegerType, true),
    StructField("DISPLAY_CITY_MARKET_NAME_FULL", StringType, false),
    StructField("CITY_MARKET_WAC_SEQ_ID2",  IntegerType,  true),
    StructField("CITY_MARKET_WAC", IntegerType, true),
    StructField("LAT_DEGREES", IntegerType, true),
    StructField("LAT_HEMISPHERE", StringType, true),
    StructField("LAT_MINUTES", IntegerType, true),
    StructField("LAT_SECONDS", IntegerType, true),
    StructField("LATITUDE", DoubleType, false),
    StructField("LON_DEGREES", IntegerType, true),
    StructField("LON_HEMISPHERE", StringType, true),
    StructField("LON_MINUTES", IntegerType, true),
    StructField("LON_SECONDS", IntegerType, true),
    StructField("LONGITUDE", DoubleType, false),
    StructField("UTC_LOCAL_TIME_VARIATION", IntegerType, true),
    StructField("AIRPORT_START_DATE", StringType, true),
    StructField("AIRPORT_THRU_DATE", StringType, true),
    StructField("AIRPORT_IS_CLOSED", IntegerType, true),
    StructField("AIRPORT_IS_LATEST", IntegerType, true)
))

var airportFullDF = spark.read.format("csv").schema(airportSchema).option("header", 
"true").load("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
data/dot_airline/90313096_T_MASTER_CORD/90313096_T_MASTER_CORD_All_All.csv")

import org.datasyslab.geospark.spatialRDD.PointRDD;
import com.vividsolutions.jts.geom.Point;
import org.apache.spark.sql.geosparksql.expressions.ST_Point

var airportDF = airportFullDF.selectExpr(
    "AIRPORT as airport_code",
    "DISPLAY_AIRPORT_NAME as airport_name",
    "LATITUDE as latitude",
    "LONGITUDE as longitude",
    "DISPLAY_CITY_MARKET_NAME_FULL as market",
    "AIRPORT_COUNTRY_NAME as country",
    "ST_Point(CAST(LONGITUDE AS Decimal(24,20)), CAST(LATITUDE AS Decimal(24,20))) as 
geometry"
).where("AIRPORT_IS_LATEST = 1")

airportDF.createOrReplaceTempView("airport")
```

```
// major airlines by passenger miles
spark.sql("""
    select
        carrier.carrier_name,
        cast(sum(passengers * distance) as long) as passenger_miles
    from flights
    join carrier
        on flights.carrier = carrier.carrier_code
    group by 1
    order by 2 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header", 
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/carrier_passenger_miles.csv")

// major airports by passenger count
spark.sql("""
    select
        airport.airport_code,
        airport.airport_name,
        cast(sum(flights.passengers) as long) as passenger_count
    from airport
    join flights
        on airport.airport_code = flights.origin or airport.airport_code = 
flights.dest
    group by 1, 2
    order by 3 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header", 
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/airport_passengers.csv")

// major cities by passenger count
spark.sql("""
    select
        airport.market,
        cast(sum(flights.passengers) as long) as passenger_count
    from airport
    join flights
        on airport.airport_code = flights.origin or airport.airport_code = 
flights.dest
    group by 1
    order by 2 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header", 
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/city_passengers.csv")

// major routes by passenger count
spark.sql("""
    select
        origin,
        dest,
        cast(sum(passengers) as long) as passenger_count
    from flights
    group by 1, 2
    order by 3 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/route_passenger_count.csv")
```

```
// major routes by passenger miles
spark.sql("""
    select
        origin,
        dest,
        cast(sum(passengers * distance) as long) as passenger_miles
    from flights
    group by 1, 2
    order by 3 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/route_passenger_miles.csv")

// major city routes by passenger miles
spark.sql("""
    select
        origin.market as origin_market,
        dest.market dest_market,
        cast(sum(flights.passengers * flights.distance) as long) as passenger_miles
    from flights
    join airport as origin
        on flights.origin = origin.airport_code
    join airport as dest
        on flights.dest = dest.airport_code
    group by 1, 2
    order by 3 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/route_city_passenger_miles.csv")

// market - market competition data frame
var marketRouteCompDF = spark.sql("""
    with carrier_pct_of_route_passengers_table as (
        select distinct
            origin_airport.market as origin_market,
            dest_airport.market as dest_market,
            round(100.0 * sum(flights.passengers) over (partition by
origin_airport.market, dest_airport.market, carrier) / sum(flights.passengers) over
(partition by origin_airport.market, dest_airport.market), 2) as
carrier_pct_of_route_passengers
        from flights
        join airport as origin_airport
            on flights.origin = origin_airport.airport_code
        join airport as dest_airport
            on flights.dest = dest_airport.airport_code
        order by 1, 2
    )
    select
        origin_market,
        dest_market,
        max(carrier_pct_of_route_passengers) as max_carrier_share
    from carrier_pct_of_route_passengers_table
    group by 1, 2
    order by 1, 2
""")
marketRouteCompDF.createOrReplaceTempView("marketRouteComp")

// market to market route competition
spark.sql("""
    select
        origin_airport.market as origin_market,
        dest_airport.market dest_market,
        competition.max_carrier_share,
        cast(sum(flights.passengers * flights.distance) as long) as passenger_miles
    from flights
    join airport as origin_airport
        on flights.origin = origin_airport.airport_code
    join airport as dest_airport
        on flights.dest = dest_airport.airport_code
    join marketRouteComp as competition
        on origin_airport.market = competition.origin_market
        and dest_airport.market = competition.dest_market
    group by 1, 2, 3
    order by 4 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/market_route_competition.csv")

// carrier market share per passenger mile
var routeCompDF = spark.sql("""
    select distinct
        origin,
        dest,
        carrier,
        distance,
        sum(passengers) over (partition by origin, dest) as route_passengers,
        sum(passengers) over (partition by origin, dest, carrier) as
carrier_passengers,
        round(100.0 * sum(passengers) over (partition by origin, dest, carrier) /
sum(passengers) over (partition by origin, dest), 2) as
carrier_pct_of_route_passengers
    from flights
    order by 6 desc
""")
routeCompDF.createOrReplaceTempView("routeComp")

spark.sql("""
    select
        carrier.carrier_name,
        cast(sum(carrier_passengers * distance) as long) as total_passenger_miles,
        round((
            sum(carrier_pct_of_route_passengers * carrier_passengers * distance) /
            sum(carrier_passengers * distance)
        ), 2) as route_share_per_passenger_mile
    from routecomp as route
    join carrier
        on route.carrier = carrier.carrier_code
    group by 1
    order by 2 desc
    limit 100
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/carrier_market_share.csv")

// major routes by passenger miles with linestring geometry
spark.sql("""
    select
        flights.origin,
        flights.dest,
        cast(
            ST_GeomFromWKT('LINESTRING(' || origin.longitude || ' ' || origin.latitude
|| ', ' || dest.longitude || ' ' || dest.latitude || ')')
        as varchar(100)) as geom,
        cast(sum(passengers * distance) as long) as passenger_miles,
        cast(sum(passengers) as long) as passengers
    from flights
    join airport as origin on flights.origin = origin.airport_code
    join airport as dest on flights.dest = dest.airport_code
    group by 1, 2, 3
    order by 4 desc
    limit 10000
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/route_passenger_count_geom.csv")

// major airports by passenger count with geom
spark.sql("""
    select
        airport.airport_code,
        airport.airport_name,
        cast(ST_GeomFromWKT(
            'POINT(' || airport.longitude || ' ' || airport.latitude || ')'
        ) as varchar(100)) as geom,
        cast(sum(flights.passengers) as long) as passenger_count
    from airport
    join flights
        on airport.airport_code = flights.origin or airport.airport_code =
flights.dest
    group by 1, 2, 3
    order by 4 desc
    limit 10000
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/airport_passengers_geom.csv")

// major cities by passenger count with geom
spark.sql("""
    with airports_table as (
    select
        airport.airport_code,
        airport.airport_name,
        airport.market,
        airport.latitude,
        airport.longitude,
        cast(sum(flights.passengers) as long) as passenger_count
    from airport
    join flights
        on airport.airport_code = flights.origin or airport.airport_code =
flights.dest
    group by 1, 2, 3, 4, 5
    having sum(flights.passengers) > 1000000
    order by airport.market
    limit 10000
    )
    select
        market,
        avg(longitude) as lng,
        avg(latitude) as lat,
        cast(ST_GeomFromWKT(
            'POINT(' || avg(longitude) || ' ' || avg(latitude) || ')'
        ) as varchar(100)) as geom,
        sum(passenger_count) as passengers
    from airports_table
    group by 1
    order by sum(passenger_count) desc
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/market_passengers_geom.csv")
```

// market - market route passengers with geom

```
spark.sql("""
    select
        origin_airport.market as origin_market,
        dest_airport.market dest_market,
        competition.max_carrier_share,
        cast(sum(flights.passengers * flights.distance) as long) as passenger_miles,
        cast(sum(flights.passengers) as long) as passengers,
        cast(
            ST_GeomFromWKT('LINESTRING(' || avg(origin_airport.longitude) || ' ' ||
avg(origin_airport.latitude) || ', ' || avg(dest_airport.longitude) || ' ' ||
avg(dest_airport.latitude) || ')')
        as varchar(100)) as geom
    from flights
    join airport as origin_airport
        on flights.origin = origin_airport.airport_code
    join airport as dest_airport
        on flights.dest = dest_airport.airport_code
    join marketRouteComp as competition
        on origin_airport.market = competition.origin_market
        and dest_airport.market = competition.dest_market
    group by 1, 2, 3
    having sum(flights.passengers * flights.distance) > 100000000
    order by 4 desc
    limit 10000
""").write.format("csv").mode("overwrite").option("sep", ",").option("header",
"true").save("/Users/tomlee/OneDrive - UW-Madison/Classes/GEOG574/Assignments/project/
results/market_route_passengers_geom.csv")
```