Additional sources

This might be useful in matching station codes to airports:

1. http://dss.ucar.edu/datasets/ds353.4/inventories/station-list.html (http://dss.ucar.edu/datasets/ds353.4/inventories/station-list.html)
2. https://www.world-airport-codes.com/ (https://www.world-airport-codes.com/)

```
from pyspark.sql import functions as f
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, IntegerType, NullType, ShortType, DateType, BooleanType,
BinaryType
from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)
```

```
display(dbutils.fs.ls("dbfs:/mnt/mids-w261/datasets_final_project/"))
```

	path	name	size
1	dbfs:/mnt/mids-w261/datasets_final_project/airlines_data/	airlines_data/	0
2	dbfs:/mnt/mids-w261/datasets_final_project/allstate-claims-severity.zip	allstate-claims-severity.zip	512048
3	dbfs:/mnt/mids-w261/datasets_final_project/dac.tar.gz	dac.tar.gz	457682
4	dbfs:/mnt/mids-w261/datasets_final_project/kdd-cup-2014-predicting-excitement-at-donors-choose.zip	kdd-cup-2014-predicting-excitement-at-donors-choose.zip	971133
5	dbfs:/mnt/mids-w261/datasets_final_project/parquet_airlines_data/	parquet_airlines_data/	0
6	dbfs:/mnt/mids-w261/datasets_final_project/parquet_airlines_data_3m/	parquet_airlines_data_3m/	0
7	dbfs:/mnt/mids-w261/datasets_final_project/parquet_airlines_data_6m/	parquet_airlines_data_6m/	0
8	dbfs:/mnt/mids-w261/datasets_final_project/porto-seguro-safe-driver-prediction.zip	porto-seguro-safe-driver-prediction.zip	802475

Showing all 12 rows.

```
display(dbutils.fs.ls("dbfs:/mnt/mids-w261/datasets_final_project/weather_data"))
```

	path	name	size
1	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather-miss.parquet/	weather-miss.parquet/	0
2	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2015a.parquet/	weather2015a.parquet/	0
3	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2016a.parquet/	weather2016a.parquet/	0
4	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2017a.parquet/	weather2017a.parquet/	0
5	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2018a.parquet/	weather2018a.parquet/	0
6	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2019a.parquet/	weather2019a.parquet/	0

Showing all 6 rows.

```
airlines = spark.read.option("header", "true").parquet(f"dbfs:/mnt/mids-w261/datasets_final_project/parquet_airlines_data/201*.parquet")
display(airlines.sample(False, 0.00001))
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE
1	2019	2	6	2	7	2019-06-02	OO	20304
2	2019	2	6	17	1	2019-06-17	DL	19790
3	2019	2	6	3	1	2019-06-03	UA	19977
4	2019	2	6	20	4	2019-06-20	MQ	20398
5	2016	3	8	22	1	2016-08-22	AA	19805
6	2016	3	8	20	6	2016-08-20	EV	20366
7	2016	3	8	11	4	2016-08-11	UA	19977
8	2016	3	8	31	3	2016-08-31	WN	19393

Showing all 343 rows.

```
airlines.printSchema()
```

```
root
 |-- YEAR: integer (nullable = true)
 |-- QUARTER: integer (nullable = true)
 |-- MONTH: integer (nullable = true)
 |-- DAY_OF_MONTH: integer (nullable = true)
 |-- DAY_OF_WEEK: integer (nullable = true)
 |-- FL_DATE: string (nullable = true)
 |-- OP_UNIQUE_CARRIER: string (nullable = true)
 |-- OP_CARRIER_AIRLINE_ID: integer (nullable = true)
 |-- OP_CARRIER: string (nullable = true)
 |-- TAIL_NUM: string (nullable = true)
 |-- OP_CARRIER_FL_NUM: integer (nullable = true)
 |-- ORIGIN_AIRPORT_ID: integer (nullable = true)
 |-- ORIGIN_AIRPORT_SEQ_ID: integer (nullable = true)
 |-- ORIGIN_CITY_MARKET_ID: integer (nullable = true)
 |-- ORIGIN: string (nullable = true)
 |-- ORIGIN_CITY_NAME: string (nullable = true)
 |-- ORIGIN_STATE_ABR: string (nullable = true)
 |-- ORIGIN_STATE_FIPS: integer (nullable = true)
 |-- ORIGIN_STATE_NM: string (nullable = true)
 |-- ORIGIN_WAC: integer (nullable = true)
```

```
f'{airlines.count():,}'
```

```
Out[7]: '31,746,841'
```

```
display(airlines.describe())
```

	summary	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUI
1	count	31746841	31746841	31746841	31746841	31746841	31746841	31746841
2	mean	2017.1512498204152	2.5174877084620797	6.552106365480585	15.749554640727876	3.9346285509162944	null	null
3	stddev	1.4316532810214522	1.1053295681781878	3.3994302561415366	8.774238088354506	1.9917635387471744	null	null
4	min	2015	1	1	1	1	2015-01-01	9E
5	max	2019	4	12	31	7	2019-12-31	YX

Showing all 5 rows.

```
airlines.where('MONTH == "MONTH"').count()
```

```
Out[11]: 0
```

Weather

https://data.nodc.noaa.gov/cgi-bin/iso?id=gov.noaa.ncdc:C00532 (https://data.nodc.noaa.gov/cgi-bin/iso?id=gov.noaa.ncdc:C00532)

```
display(dbutils.fs.ls("dbfs:/mnt/mids-w261/datasets_final_project/weather_data"))
```

	path	name	size
1	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/teamjvml1/	teamjvml1/	0
2	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather-miss.parquet/	weather-miss.parquet/	0
3	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2015a.parquet/	weather2015a.parquet/	0
4	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2016a.parquet/	weather2016a.parquet/	0
5	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2017a.parquet/	weather2017a.parquet/	0
6	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2018a.parquet/	weather2018a.parquet/	0
7	dbfs:/mnt/mids-w261/datasets_final_project/weather_data/weather2019a.parquet/	weather2019a.parquet/	0

Showing all 7 rows.

```
weather = spark.read.option("header", "true")\
                    .parquet(f"dbfs:/mnt/mids-w261/datasets_final_project/weather_data/*.parquet")
```

```
f'{weather.count():,}'
```

```
Out[3]: '630,904,436'
```

```
display(weather.where('DATE =="DATE"'))
```

Query returned no results

```
display(weather.describe())
```

	summary	STATION	SOURCE	LATITUDE	LONGITUDE	ELEVATION	NAME	REPOF
1	count	626994336	548964297	630904436	630904436	630904436	630904436	630904
2	mean	5.900842664264165E10	4.945690792346738	36.82263507708492	-36.70028299553371	369.4329781232484	null	null
3	stddev	3.2730239773023293E10	1.3644544181414946	22.617392665562907	78.71810434001523	534.4094770398781	null	null
4	min	10000199999	1	-90.0	-179.999	-999.9		CRN05
	max	A5125600451	8	83.65	179.998	7070.0	ZYRYANKA, RS	SY-MT

Showing all 5 rows.

Stations

```
stations = spark.read.option("header", "true").csv("dbfs:/mnt/mids-w261/DEMO8/gsod/stations.csv.gz")
```

```
display(stations)
```

```
from pyspark.sql import functions as f
stations.where(f.col('name').contains('JAN MAYEN NOR NAVY'))
```

```
stations.select('name').distinct().count()
```

```
display(stations.select('name').distinct())
```

```
weather.select('NAME').distinct().count()
```

```
display(weather.select('name').distinct())
```

```
display(dbutils.fs.ls())
```

```
  TypeError: ls() missing 1 required positional argument: 'path'
```

Format data

- Introduce newly imported airport_locations dataset (airports with longitude and latitudes)
- Find distinct airports in airlines dataset and join onto airport_locations for longitudes and latitudes
- For each airport, find closest weather station using haversine distance

```
# %sql
# /* Query the created temp table in a SQL cell */
# -- select * from `globalairportdatabase_1_txt`
# -- alter table `globalairportdatabase_1_txt` rename to `team_29_global_airport_database`
```

```
# airport_locations = spark.sql("select * from team_29_global_airport_database")
# #SAVING Spark Dataframe to Directory
# userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'     #CHANGE USERNAME IF FILE STORED ELSEWHERE
# finalproject_path = userhome + "/FINAL_PROJECT/"
# dbutils.fs.mkdirs(finalproject_path)                      #Make Directory in DataBricks
# file_to_store = airport_locations                #name of Spark Dataframe (to save in database)
# filename = "airport_locations"                   #new file name in database
# dbutils.fs.rm(finalproject_path+filename, True)           #remove file if there already is an existing one, be careful with this!!!
# file_to_store.write.format("parquet").save(finalproject_path+filename)


# read in airlines data from group folder
airlines = spark.read.option("header", "true").parquet(f"dbfs:/mnt/mids-w261/datasets_final_project/parquet_airlines_data/201*.parquet")
airlines.createOrReplaceTempView('airlines')

# read in and set airport_locations dataset to variable
airport_locations = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airport_locations")
airport_locations.createOrReplaceTempView('airport_locations')

# make temporary tables to access with SQL
weather.createOrReplaceTempView('weather')
stations.createOrReplaceTempView('stations')


test = 'SELECT op_carrier, op_carrier_fl_num, fl_date, origin FROM airlines'
# test = 'SELECT * from airlines'

display(sqlContext.sql(test))
```

	op_carrier	op_carrier_fl_num	fl_date	origin
1	B6	620	2019-06-01	SAN
2	B6	623	2019-06-01	JFK
3	B6	624	2019-06-01	LAX
4	B6	626	2019-06-01	BWI
5	B6	629	2019-06-01	MCO
6	DL	1584	2019-06-25	ROC
7	DL	1585	2019-06-25	LGA
8	DL	1586	2019-06-25	ATL

Showing the first 1000 rows.

```
test = 'SELECT iata as airport_code, latitude_decimal_degrees, longitude_degrees from airport_locations WHERE iata in
("SAN","JFK","LAX","BWI","MCO","ROC","LGA")'
# test = 'SELECT * from airlines'

display(sqlContext.sql(test))
```

	airport_code	latitude_decimal_degrees	longitude_degrees
1	BWI	39.175	76
2	JFK	40.64	73
3	LAX	33.942	118
4	LGA	40.777	73
5	MCO	28.429	81
6	ROC	43.119	77
7	SAN	32.733	117

Showing all 7 rows.

```
test = 'SELECT distinct station as weather_station_id, latitude, longitude from weather'

display(sqlContext.sql(test))
```

	weather_station_id	latitude	longitude
1	70019726558	66.6	-159.98611
2	6662199999	47.181628	7.417189

3	99999994059	48.4887	-105.2096
4	99999923909	37.6344	-91.7226
5	6670099999	47.464722	8.549167
6	6673099999	47.2166666	8.6833333
7	99999926562	60.1951	-154.3196
8	99999994044	40.13	-105.24

Showing the first 1000 rows.

```
stations_cleaned = sqlContext.sql("SELECT DISTINCT STATION, NAME, LATITUDE, LONGITUDE FROM weather")
stations_cleaned.createOrReplaceTempView('stations_cleaned')

## there seems to be a couple hundred stations that move (have different longitude latitudes over time)
# these are either hot air balloons or floating buoys that seem to move across the ocean
# for clarity's sake, we're going to remove any weather station that has more than 1 distinct location (latitude, longitude)
stations_with_only_one_location = sqlContext.sql("SELECT station as station_id, count(*) as num_rows FROM stations_cleaned group by 1
HAVING num_rows = 1")


## join from the one location stations with the rest of the station location data
stations_cleaned_one_location = stations_with_only_one_location.join(stations_cleaned, stations_with_only_one_location.station_id ==
stations_cleaned.STATION, how = 'left')

# drop all other columns from the airport_locations dataset
columns_to_drop = ['station_id', 'num_rows']
stations_cleaned_one_location = stations_cleaned_one_location.drop(*columns_to_drop)

stations_cleaned_one_location.createOrReplaceTempView('stations_cleaned_one_location')

display(stations_cleaned_one_location)


## cleaning aiport_locations data to only hold important columns
# rename columns
airport_locations_cleaned = airport_locations.withColumnRenamed("iata", "airport_code")

# drop all other columns from the airport_locations dataset
columns_to_drop = ['icao'
                   ,'latitude_degrees'
                   ,'latitude_minutes'
                   ,'latitude_seconds'
                   ,'latitude_direction'
                   ,'longitude_degrees'
                   ,'longitude_minutes'
                   ,'longitude_seconds'
                   ,'longitude_direction'
                   ,'altitude']

airport_locations_cleaned = airport_locations_cleaned.drop(*columns_to_drop)
airport_locations_cleaned.createOrReplaceTempView('airport_locations_cleaned')


## purposefully cross join these two datasets so that every airport has a corresponding row with every weather station
airports_and_stations_exploded = stations_cleaned_one_location.join(airport_locations_cleaned)
display(airports_and_stations_exploded)
airports_and_stations_exploded.count()
##this should be counting around ~140M rows, we will be quickly shrinking this down
```

```
## formula for haversine distance calculation found here:
## https://stackoverflow.com/questions/60086180/pyspark-how-to-apply-a-python-udf-to-pyspark-dataframe-columns

from pyspark.sql.functions import col, radians, asin, sin, sqrt, cos

airports_and_stations_exploded_with_distance = airports_and_stations_exploded.withColumn("dlon",
radians(col("longitude_decimal_degrees")) - radians(col("LONGITUDE"))) \
                                                                    .withColumn("dlat",
radians(col("latitude_decimal_degrees")) - radians(col("LATITUDE"))) \
                                                                    .withColumn("haversine_dist", asin(sqrt(
                                                                                                        sin(col("dlat") /
2) ** 2 + cos(radians(col("LATITUDE")))
                                                                                                        *
cos(radians(col("latitude_decimal_degrees"))) * sin(col("dlon") / 2) ** 2
                                                                                                        )
                                                                                                ) * 2 * 3963 * 5280) \
                                                                    .drop("dlon", "dlat")

airports_and_stations_exploded_with_distance.createOrReplaceTempView('airports_and_stations_exploded_with_distance')


## rank each row by ascending order in distance
airports_and_stations_exploded_with_distance_ranked = sqlContext.sql('SELECT *, dense_rank() over (partition by airport_code order by
haversine_dist) as rnk FROM airports_and_stations_exploded_with_distance')
airports_and_stations_exploded_with_distance_ranked.createOrReplaceTempView('airports_and_stations_exploded_with_distance_ranked')

## filter on rank = 1 so only the shortest distance remains
airports_and_stations_ranked_and_trimmed = sqlContext.sql('SELECT * FROM airports_and_stations_exploded_with_distance_ranked WHERE rnk =
1')
airports_and_stations_ranked_and_trimmed.createOrReplaceTempView('airports_and_stations_ranked_and_trimmed')


## remove rows where latitudes and longitudes for both the weather station and the airport are 0 - this must be an error in the data
that we can't rely on
airport_weather_station_key = sqlContext.sql('SELECT * FROM airports_and_stations_ranked_and_trimmed WHERE LATITUDE <> 0 AND LONGITUDE
<> 0 AND latitude_decimal_degrees <> 0 AND longitude_decimal_degrees <> 0')


## rename columns to be more clear, this will be our final mapping table for airports and weather stations
# rename columns
airport_weather_station_key = airport_weather_station_key.withColumnRenamed("STATION", "station_id")\
                                                                    .withColumnRenamed("NAME", "station_name")\
                                                                    .withColumnRenamed("city_town", "airport_city")\
                                                                    .withColumnRenamed("country", "airport_country")\
                                                                    .withColumnRenamed("LATITUDE", "station_latitude")\
                                                                    .withColumnRenamed("LONGITUDE", "station_longitude")\
                                                                    .withColumnRenamed("latitude_decimal_degrees",
"airport_latitude")\
                                                                    .withColumnRenamed("longitude_decimal_degrees",
"airport_longitude")\
                                                                    .withColumnRenamed("haversine_dist",
"haversine_dist_feet")

# drop all other columns from the airport_locations dataset
columns_to_drop = ['rnk']
airport_weather_station_key = airport_weather_station_key.drop(*columns_to_drop)


display(airport_weather_station_key)

  NameError: name 'airport_weather_station_key' is not defined

## after trimming, there are only 2,911 rows left in this dataset - need to make sure that this roughly matches the unique airports in
the airlines dataset
airport_weather_station_key.count()

### checking the airlines dataset, there seem to be only around 300 airport codes


# #SAVING this final airport to weather station mapper key to final project folder

# userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'     #CHANGE USERNAME IF FILE STORED ELSEWHERE
# finalproject_path = userhome + "/FINAL_PROJECT/"
# # dbutils.fs.mkdirs(finalproject_path)                       #Make Directory in DataBricks
# file_to_store = airport_weather_station_key                 #name of Spark Dataframe (to save in database)
# filename = "airport_weather_station_key"                    #new file name in database
# dbutils.fs.rm(finalproject_path+filename, True)            #remove file if there already is an existing one, be careful with this!!!
# file_to_store.write.format("parquet").save(finalproject_path+filename)
```

Join airlines data onto newly created airport_weather_station_key dataframe

```
airport_weather_station_key = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airport_weather_station_key")
airport_weather_station_key.createOrReplaceTempView('airport_weather_station_key')


test = 'SELECT station_id, station_latitude, station_longitude, airport_code, airport_latitude, airport_longitude, haversine_dist_feet
FROM airport_weather_station_key'
display(sqlContext.sql(test))
```

	station_id	station_latitude	station_longitude	airport_code	airport_latitude	airport_longitude	haversine_dist_feet
1	72512614736	40.29639	-78.32028	AOO	40.296	-78.32	161.96709662798654
2	84752099999	-16.341072	-71.583083	AQP	-16.341	-71.583	39.26067867327079
3	72219013874	33.6301	-84.4418	ATL	33.64	-84.427	5772.012020316871
4	72228013876	33.56556	-86.745	BHM	33.563	-86.753	2607.411638777335
5	96441099999	3.12385	113.020472	BTU	3.172	113.044	19565.70347813837
6	83208099999	-12.694375	-60.098269	BVH	-12.694	-60.098	167.16911411449553
7	16020099999	46.460194	11.326383	BZO	46.461	11.326	309.19762061213015
8	59287099999	23.392436	113.298786	CAN	23.184	113.266	76912.07784504181

Showing the first 1000 rows.


```
airlines = spark.read.option("header", "true").parquet(f"dbfs:/mnt/mids-w261/datasets_final_project/parquet_airlines_data/201*.parquet")
airport_weather_station_key = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airport_weather_station_key")

airlines.createOrReplaceTempView('airlines')
airport_weather_station_key.createOrReplaceTempView('airport_weather_station_key')
weather.createOrReplaceTempView('weather')
stations.createOrReplaceTempView('stations')
```

```
# join dataset on origin location
airlines_with_origin = airlines.join(airport_weather_station_key, airlines.ORIGIN == airport_weather_station_key.airport_code, how =
'left')

# rename columns
airlines_with_origin = airlines_with_origin.withColumnRenamed("station_id", "origin_station_id")\
                                           .withColumnRenamed("station_name", "origin_station_name")\
                                           .withColumnRenamed("station_latitude", "origin_station_latitude")\
                                           .withColumnRenamed("station_longitude", "origin_station_longitude")\
                                           .withColumnRenamed("airport_latitude", "origin_airport_latitude")\
                                           .withColumnRenamed("airport_longitude", "origin_airport_longitude")\
                                           .withColumnRenamed("haversine_dist_feet", "origin_haversine_distance_feet")

# drop all other columns from the airport_locations dataset
columns_to_drop = ['airport_code'
                   ,'airport_name'
                   ,'airport_city'
                   ,'airport_country']
airlines_with_origin = airlines_with_origin.drop(*columns_to_drop)

# join dataset on destination location
airlines_with_origin_and_destination = airlines_with_origin.join(airport_weather_station_key, airlines_with_origin.DEST ==
airport_weather_station_key.airport_code, how = 'left')

# rename columns
airlines_with_origin_and_destination = airlines_with_origin_and_destination.withColumnRenamed("station_id", "destination_station_id")\
                                                                            .withColumnRenamed("station_name",
"destination_station_name")\
                                                                            .withColumnRenamed("station_latitude",
"destination_station_latitude")\
                                                                            .withColumnRenamed("station_longitude",
"destination_station_longitude")\
                                                                            .withColumnRenamed("airport_latitude",
"destination_airport_latitude")\
                                                                            .withColumnRenamed("airport_longitude",
"destination_airport_longitude")\
                                                                            .withColumnRenamed("haversine_dist",
"destination_haversine_distance_feet")

# drop all other columns from the airport_locations dataset
columns_to_drop = ['airport_code'
                   ,'airport_name'
                   ,'airport_city'
                   ,'airport_country']

# final airlines dataset with nearest weather station at both origin and destination
airlines_with_weather_station = airlines_with_origin_and_destination.drop(*columns_to_drop)


# ## SAVING this final airport to weather station mapper key to final project folder

# userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'     #CHANGE USERNAME IF FILE STORED ELSEWHERE
# finalproject_path = userhome + "/FINAL_PROJECT/"
# # dbutils.fs.mkdirs(finalproject_path)                       #Make Directory in DataBricks
# file_to_store = airlines_with_weather_station                 #name of Spark Dataframe (to save in database)
# filename = "airlines_with_weather_station"                    #new file name in database
# dbutils.fs.rm(finalproject_path+filename, True)            #remove file if there already is an existing one, be careful with this!!!
# file_to_store.write.format("parquet").save(finalproject_path+filename)
```

EDA on the airlines data set with new columns to join weather data set

- airlines_with_weather_station now has the columns origin_station_id and destination_station_id
- these station id columns match onto the weather dataset's STATION column
- joining on these columns, along with some datetime logic to make sure we're only looking at weather data for the origin and destination at the appropriate time will get us the correctly joined dataset

```
airlines_with_weather_station = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_weather_station")
airlines_with_weather_station.createOrReplaceTempView('airlines_with_weather_station')
display(airlines_with_weather_station)
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE
1	2019	2	6	27	4	2019-06-27	OO	20304
2	2019	2	6	27	4	2019-06-27	OO	20304
3	2019	2	6	27	4	2019-06-27	OO	20304
4	2019	2	6	27	4	2019-06-27	OO	20304
5	2019	2	6	27	4	2019-06-27	OO	20304
6	2019	2	6	27	4	2019-06-27	OO	20304
7	2019	2	6	27	4	2019-06-27	OO	20304
8	2019	2	6	27	4	2019-06-27	OO	20304

Showing the first 1000 rows.

```
# export just the weather station data
station_data = spark.sql('''select origin_station_name,
                                   origin_Station_latitude,
                                   origin_Station_longitude,
                                   origin_airport_latitude,
                                   origin_airport_longitude
                            from airlines_with_weather_station
                            group by 1,2,3,4,5
''')
station_data.display(100000)
```

	origin_station_name	origin_Station_latitude	origin_Station_longitude	origin_airport_latitude	origin
1	ALAMEDA, CA US	37.772	-122.298	37.721	-122.2
2	TALLAHASSEE REGIONAL AIRPORT, FL US	30.39306	-84.35333	30.396	-84.35
3	BOSTON, MA US	42.3606	-71.0097	42.364	-71.00
4	CHRISTIANSTED HAMILTON FIELD AIRPORT, VI US	17.6997	-64.8125	17.702	-64.79
5	MOBILE REGIONAL AIRPORT, AL US	30.68833	-88.24556	30.691	-88.24
6	WATERTOWN AIRPORT, NY US	43.98867	-76.02623	43.992	-76.02
7	KANSAS CITY INTERNATIONAL AIRPORT, MO US	39.2972	-94.7306	39.297	-94.71
8	VALPARAISO EGLIN AFB, FL US	30.48333	-86.51667	30.483	-86.52

Showing all 208 rows.

```
## There are 164 airports that we could not find a match with in the airport_locations dataset, and therefore, we could not find a
nearest weather station for
## These airports are some really small ones that you can see below
## Flights that involve these 164 airports make up about 10% of the entire airlines dataset (~3M out out of ~31M)
## We can come back and revisit these but it seems like we can move forward since we have the vast majority of airports covered
test = 'SELECT origin_city_name, count(*) FROM airlines_with_weather_station WHERE origin_station_id is null group by 1'
display(sqlContext.sql(test))

test = 'SELECT count(*) FROM airlines_with_weather_station WHERE origin_station_id is null or destination_station_id is null'
display(sqlContext.sql(test))
```

	origin_city_name	count(1)
1	Medford, OR	17848
2	Redding, CA	5409
3	St. Cloud, MN	364
4	Devils Lake, ND	3049
5	Dubuque, IA	3073
6	Huntsville, AL	31233
7	Columbus, GA	5797
8	Pasco/Kennewick/Richland, WA	16144

Showing all 164 rows.

	count(1)
1	3437823

Showing all 1 rows.

Make airlines data useable for join and change into UTC format

- load and read in airport timezone dataset
- update the Airline + Weather Station dataset to include local time
- join the updated Airline + Weather Station + local time dataset with the timezone dataset
- use the spark built-in function to convert the local time to UTC time based on timezone

```
# import functions
from pyspark.sql import types
import pyspark.sql.functions as f
from pyspark.sql.functions import *
from pytz import timezone
import pytz


# # pull data from database. Data contains airport and timezone information. source [https://openflights.org/data.html]
# airport_timezones = spark.sql("select * from team_29_airport_tz")

# #SAVING Spark Dataframe to Directory
# userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'         #CHANGE USERNAME IF FILE STORED ELSEWHERE
# finalproject_path = userhome + "/FINAL_PROJECT/"
# file_to_store = airport_timezones                                     #name of Spark Dataframe (to save in database)
# filename = "airport_timezones"                                        #new file name in database
# dbutils.fs.rm(finalproject_path+filename, True)                       #remove file if there already is an existing one, be careful
with this!!!
# file_to_store.write.format("parquet").save(finalproject_path+filename)


# read in and set airport_timezones dataset to variable airport_tz
airport_tz = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airport_timezones")
airport_tz.createOrReplaceTempView('airport_tz')


# read in the airlines_with_weather_station dataset
airlines_with_weather_station = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_weather_station")
airlines_with_weather_station.createOrReplaceTempView('airlines_with_weather_station')


# Convert the date and time in the airlines dataset to timestamps

# UDF for converting year, month, day to timestamps
def create_datetime_from_parts(year, month, day, local_time):
    time = str(local_time)
    num_str = time.zfill(4)
    hour = num_str[0:2]
    min = num_str[2:4]
    return f'{year}-{month}-{day} {hour}:{min}:0'

# Apply UDF to create column for local departure time
create_datetime_udf = f.udf(create_datetime_from_parts, types.StringType())
airlines_with_localtime = airlines_with_weather_station.withColumn("LOCAL_DEP_TIME", create_datetime_udf('YEAR', 'MONTH',
'DAY_OF_MONTH','CRS_DEP_TIME').cast(types.TimestampType()))


# join airlines dataset with timezone dataset. join based on departure airport 'ORIGIN' and airport 'IATA'
airlines_with_timezone = airlines_with_localtime.join(f.broadcast(airport_tz), airport_tz.IATA==airlines_with_localtime.ORIGIN, 'inner')


# use the spark built-in function to convert the local time to UTC time based on timezone
airlines_with_utc = airlines_with_timezone.withColumn("UTC_DEP_TIME",to_utc_timestamp(col("LOCAL_DEP_TIME"), col("timezone")))


# drop the columns from the timezone dataset and save final updated dataset airlines_with_weather_station_utc
drop_column_tz = ['id','name','city','country','IATA','ICAO','lat','long','UTC_offset']
airlines_with_weather_station_utc_dep = airlines_with_utc.drop(*drop_column_tz)
```

```
# #SAVING Spark Dataframe to Directory
# userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'         #CHANGE USERNAME IF FILE STORED ELSEWHERE
# finalproject_path = userhome + "/FINAL_PROJECT/"
# file_to_store = airlines_with_weather_station_utc_dep                 #name of Spark Dataframe (to save in database)
# filename = "airlines_with_weather_station_utc_dep"                    #new file name in database
# dbutils.fs.rm(finalproject_path+filename, True)                       #remove file if there already is an existing one, be careful
with this!!!
# file_to_store.write.format("parquet").save(finalproject_path+filename)
```

```
# read in airlines_with_weather_station_utc_dep
airlines_with_weather_station_utc_dep = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_weather_station_utc_dep")
airlines_with_weather_station_utc_dep.createOrReplaceTempView('airlines_with_weather_station_utc_dep')
display(airlines_with_weather_station_utc_dep)
airlines_with_weather_station_utc_dep.count()
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE
1	2019	2	4	5	5	2019-04-05	NK	20416
2	2019	2	4	6	6	2019-04-06	NK	20416
3	2019	2	4	7	7	2019-04-07	NK	20416
4	2019	2	4	8	1	2019-04-08	NK	20416
5	2019	2	4	9	2	2019-04-09	NK	20416
6	2019	2	4	10	3	2019-04-10	NK	20416
7	2019	2	4	11	4	2019-04-11	NK	20416
8	2019	2	4	12	5	2019-04-12	NK	20416

Showing the first 1000 rows.

```
Out[1]: 31573695
```

Create new variable "PREV_DEP_DEL15" that records if the previous flight leg is delayed

- load and read in airlines with weather station UTC dataset
- remove rows with nulls in DEP_DEL15 column
- partition by tail number and order by UTC departure time
- create column "PREV_UTC_DEP_TIME" that has the previous flight leg UTC departure time
- create column "DEP_TIME_DIFF" that calculates the time difference between the previous departure time and the current departure time, in hours.
- the time difference, "DEP_TIME_DIFF", must be greater than 2 hours because we can only use data that we have access to 2 hours prior to departure. In addition, DEP_TIME_DIFF must be less than 7.5 hours. This is the cutoff we selected because time differences greater than 7.5 hours typically indicates a change in day. First flights in the morning are usually not impacted by the last flight from the night before.
- create column "PREV_DEP_DEL15" that denotes if the previous flight leg departure was delayed by 15 minutes or more. When the time difference between departures is between 2 and 7.5 hours, "PREV_DEP_DEL15" is equal to the previous flight leg's "DEP_DEL15". Else, "PREV_DEP_DEL15" is set to 0.

```
# Remove the rows with nulls in the DEP_DEL15 column (should remove 474,582 records)
airlines_with_weather_station_remove_nulls =
airlines_with_weather_station_utc_dep.filter(airlines_with_weather_station_utc_dep.DEP_DEL15.isNotNull())
```

```
# import functions
from pyspark.sql import functions as F
from pyspark.sql.functions import col, lag, hour, when
from pyspark.sql.window import Window

# create window that partitions on tail number and orders by UTC departure time
my_window = Window.partitionBy("TAIL_NUM").orderBy('UTC_DEP_TIME')

# create column PREV_UTC_DEP_TIME for previous flight leg UTC departure time
# create column DEP_TIME_DIFF to calculate difference (in hours) between current departure time and previous departure time
# create column PREV_DEP_DEL15 to denote if the previous flight was delayed 15 minutes (DEP_DEL15 = 1) given that DEP_TIME DIFF is
between 2 and 7.5 hours
airports_and_stations_with_prev_del15 = airlines_with_weather_station_remove_nulls.withColumn("PREV_UTC_DEP_TIME",
F.lag(col('UTC_DEP_TIME')).over(my_window)) \
                                    .withColumn("DEP_TIME_DIFF", ((col("UTC_DEP_TIME").cast("long") -
col("PREV_UTC_DEP_TIME").cast("long"))/3600)) \
                                    .fillna({'DEP_TIME_DIFF': 0 }) \
                                    .withColumn("PREV_DEP_DEL15", when((col("DEP_TIME_DIFF") >= 2) & (col("DEP_TIME_DIFF") <= 7.5),
F.lag(col("DEP_DEL15")).over(my_window))
.otherwise(0))

# display dataframe
display(airports_and_stations_with_prev_del15)
airports_and_stations_with_prev_del15.count()
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE
1	2015	1	1	1	4	2015-01-01	UA	19977
2	2015	1	1	1	4	2015-01-01	UA	19977
3	2015	1	1	4	7	2015-01-04	UA	19977
4	2015	1	1	5	1	2015-01-05	UA	19977
5	2015	1	1	5	1	2015-01-05	UA	19977
6	2015	1	1	5	1	2015-01-05	UA	19977
7	2015	1	1	5	1	2015-01-05	UA	19977
8	2015	1	1	6	2	2015-01-06	UA	19977

Showing the first 1000 rows.

```
Out[3]: 31099113

# Verify that the "PREV_DEP_DEL15" variable has been recorded correctly
verify_df = airports_and_stations_with_prev_del15[['TAIL_NUM','LOCAL_DEP_TIME','UTC_DEP_TIME',"PREV_UTC_DEP_TIME",
"DEP_TIME_DIFF",'DEP_DEL15', 'PREV_DEP_DEL15']]
display(verify_df)
verify_df.count()
```

	TAIL_NUM	LOCAL_DEP_TIME	UTC_DEP_TIME	PREV_UTC_DEP_TIME	DEP_TIME_DIFF	DEP_DEL15	PR
1	N12004	2019-02-14T09:00:00.000+0000	2019-02-14T14:00:00.000+0000	null	0	0	0
2	N12004	2019-02-14T14:00:00.000+0000	2019-02-14T22:00:00.000+0000	2019-02-14T14:00:00.000+0000	8	1	0
3	N12004	2019-02-15T07:00:00.000+0000	2019-02-15T12:00:00.000+0000	2019-02-14T22:00:00.000+0000	14	0	0
4	N12004	2019-02-15T14:00:00.000+0000	2019-02-15T22:00:00.000+0000	2019-02-15T12:00:00.000+0000	10	1	0
5	N12004	2019-02-16T07:00:00.000+0000	2019-02-16T12:00:00.000+0000	2019-02-15T22:00:00.000+0000	14	0	0
6	N12004	2019-02-16T13:15:00.000+0000	2019-02-16T21:15:00.000+0000	2019-02-16T12:00:00.000+0000	9.25	0	0
7	N12004	2019-02-17T09:00:00.000+0000	2019-02-17T14:00:00.000+0000	2019-02-16T21:15:00.000+0000	16.75	0	0
8	N12004	2019-02-17T14:00:00.000+0000	2019-02-17T22:00:00.000+0000	2019-02-17T14:00:00.000+0000	8	1	0

Showing the first 1000 rows.

```
Out[5]: 31099113

# drop the columns from the timezone dataset and save final updated dataset airlines_with_weather_station_utc
drop_column_prev_dep_del15 = ['PREV_UTC_DEP_TIME']
airlines_with_weather_station_utc = airports_and_stations_with_prev_del15.drop(*drop_column_prev_dep_del15)
display(airlines_with_weather_station_utc)
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE
1	2018	1	1	1	1	2018-01-01	G4	20368
2	2018	1	1	1	1	2018-01-01	G4	20368
3								

4	2018	1	1	2	2	2018-01-02	G4	20368
5	2018	1	1	2	2	2018-01-02	G4	20368
6	2018	1	1	2	2	2018-01-02	G4	20368
7	2018	1	1	2	2	2018-01-02	G4	20368
8	2018	1	1	2	2	2018-01-02	G4	20368

Showing the first 1000 rows.

```
# #SAVING Spark Dataframe to Directory
# userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'     #CHANGE USERNAME IF FILE STORED ELSEWHERE
# finalproject_path = userhome + "/FINAL_PROJECT/"
# file_to_store = airlines_with_weather_station_utc                 #name of Spark Dataframe (to save in database)
# filename = "airlines_with_weather_station_utc"                    #new file name in database
# dbutils.fs.rm(finalproject_path+filename, True)                   #remove file if there already is an existing one, be careful with
this!!!
# file_to_store.write.format("parquet").save(finalproject_path+filename)


# read in airlines_with_weather_station_utc
airlines_with_weather_station_utc = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_weather_station_utc")
airlines_with_weather_station_utc.createOrReplaceTempView('airlines_with_weather_station_utc')
display(airlines_with_weather_station_utc)
airlines_with_weather_station_utc.count()
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE
1	2015	1	1	1	4	2015-01-01	US	20355
2	2015	1	1	1	4	2015-01-01	US	20355
3	2015	1	1	2	5	2015-01-02	US	20355
4	2015	1	1	3	6	2015-01-03	US	20355
5	2015	1	1	3	6	2015-01-03	US	20355
6	2015	1	1	3	6	2015-01-03	US	20355
7	2015	1	1	3	6	2015-01-03	US	20355
8	2015	1	1	4	7	2015-01-04	US	20355

Showing the first 1000 rows.

```
Out[1]: 31099113
```

Pull in fields from Airline Data

- put the column name and a group by number for each field that we want to bring in

```
# Hourly Data
# make a time stamp
airlines_with_updated_weather_hourly = spark.sql('''select fl_date,
concat(utc_dep_time,op_unique_carrier,origin_station_id, op_carrier_fl_num) as id,
                                        op_carrier_airline_id,
                                        op_unique_carrier,
                                        year,
                                        quarter,
                                        month,
                                        day_of_month,
                                        day_of_week,
                                        origin_city_name,
                                        op_carrier_fl_num,
                                        op_carrier,
                                        origin_State_abr,
                                        dest_state_abr,
                                        origin,
                                        dest,
                                        arr_delay_new,
                                        distance,
                                        weather_delay,
                                        nas_delay,
                                        security_delay,
                                        carrier_delay,
                                        late_aircraft_delay,

                    origin_station_id,
                    local_dep_time,
                    timezone,
                    utc_dep_time,
                    to_timestamp(concat(date(utc_dep_time), ' ' , hour(utc_dep_time), ':' ,'00', ':' , '00'))- interval 2 hours as
updated_dep_time,
                    dep_delay_new,
                    dep_del15,
                    dep_time_diff,
                    prev_dep_del15
                    from airlines_with_weather_station_utc
                    where origin_station_id is not null
                    group by 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32
                    order by utc_dep_time''')
# move it to a temp table
airlines_with_updated_weather_hourly.createOrReplaceTempView('airlines_with_updated_weather_hourly')

display(airlines_with_updated_weather_hourly)

airlines_with_updated_weather_hourly.count()
```

	fl_date	id	op_carrier_airline_id	op_unique_carrier	year	quarter	month	day_c
1	2015-01-01	2015-01-01 06:55:00NK72622564776451	20416	NK	2015	1	1	1
2	2015-01-01	2015-01-01 07:00:00NK72528704724647	20416	NK	2015	1	1	1
3	2015-01-01	2015-01-01 07:15:00NK72658414927597	20416	NK	2015	1	1	1
4	2015-01-01	2015-01-01 07:30:00DL725650030172336	19790	DL	2015	1	1	1
5	2015-01-01	2015-01-01 07:40:00DL72572024127232 4	19790	DL	2015	1	1	1
6	2015-01-01	2015-01-01 08:10:00AA722950231742336	19805	AA	2015	1	1	1
7	2015-01-01	2015-01-01 08:20:00AA725650030172392	19805	AA	2015	1	1	1
8	2015-01-01	2015-01-01 08:20:00AA72295023174258	19805	AA	2015	1	1	1

Showing the first 1000 rows.

```
Out[2]: 29384608

# Store the Airlines Data

dbutils.fs.rm("dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_updated_weather_hourly", True)
airlines_with_updated_weather_hourly.write.parquet("dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_updated
_weather_hourly")


# Data Check to make sure no duplicates
airlines_with_updated_weather_hourly.select('id').distinct().count()

Out[4]: 29384603
```

Pull in fields from Weather Data

- put the column name and a group by number for each field that we want to bring in

```
# delete later once the final table has been made, this is just so we dont have to scroll to bring in weather data
weather = spark.read.option("header", "true")\
                    .parquet(f"dbfs:/mnt/mids-w261/datasets_final_project/weather_data/*.parquet")

f'{weather.count():,}'

weather.createOrReplaceTempView('weather')


# Take the weather data and find average per day
# join on snow indicator

weather_day = spark.sql('''select
                                to_date(date_trunc('DD',if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' ,
string(hour(date)) , ':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1
hour)),'YYYY-MM-DD') as updated_date_d,
                                a.station,
                                avg(int(left(AN1,3))) as snow_hours_d,
                                avg(int(substring(AN1,5,4))) as snow_depth_in_d,
                                avg(int(left(aa1,2))) as precipitation_period_time_min_d,
                                avg(int(substring(aa1,4,4))) as precipitation_inches_d,
                                ifnull(snow,0) as snow_ind
                                --ifnull(thunder,0) as thunder_ind
                                --ifnull(rain,0) as rain_ind
                                from weather a


                                left join (select station,
                                to_date(date_trunc('DD',if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' ,
string(hour(date)) , ':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1
hour)),'YYYY-MM-DD') as updated_date_snow,
                                                        '1' as snow
                                            from weather
                                            where int(left(AW1,2)) =24 or int(left(AW1,2)) between 70 and 78) b
                                  on to_date(date_trunc('DD',if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' ,
string(hour(date)) , ':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1
hour)),'YYY-MM-DD') = updated_date_snow    and a.station = b.station


                                where int(substring(aa1,4,4)) != 9999 and int(substring(AN1,5,4)) !=9999
                                group by 1,2,7
                                order by avg(int(substring(AN1,5,4))) desc
                                --having avg(int(right(substring_index(vis,',', 2),1))) is not null''')
weather_day.createOrReplaceTempView("weather_day")
display(weather_day)
```

	updated_date_d	station	snow_hours_d	snow_depth_in_d	precipitation_period_time_min_d	precipitation_inches_d	snow_ind
1	2018-02-01	72562804935	24	2515	24	0	0
2	2017-03-14	72502594741	24	1727	24	0	1
3	2019-02-22	72375003103	24	912	24	366	1
4	2017-12-26	72526014860	24	859	24	450	1
5	2015-01-28	72510094746	24	810	24	394	1
6	2017-03-15	72515004725	24	792	24	554	1
7	2016-01-24	74486094789	24	770	24	737	0
8	2016-01-24	72517014737	24	767	24	419	1

Showing the first 1000 rows.

```
# bring in the thunder and rain indicators
thunder_d = spark.sql('''select station,
                              to_date(date_trunc('DD',if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' ,
string(hour(date)) , ':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1
hour)),'YYYY-MM-DD') as updated_date_thunder,
                                                      '1' as thunder
                                               from weather
                                               where int(left(AW1,2)) =26 or int(left(AW1,2)) between 90 and 96''')

thunder_d.createOrReplaceTempView("thunder_d")
display(thunder_d)

rain_d = spark.sql('''select station,
                             to_date(date_trunc('DD',if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' ,
string(hour(date)) , ':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1
hour)),'YYYY-MM-DD') as updated_date_rain,
                                                    '1' as rain
                                             from weather
                                             where int(left(AW1,2)) =23 or int(left(AW1,2)) between 60 and 68''')

rain_d.createOrReplaceTempView("rain_d")
display(rain_d)
```

	station	updated_date_thunder	thunder
1	7650099999	2016-07-12	1
2	7650099999	2016-09-27	1
3	7650099999	2016-10-14	1
4	7650099999	2016-10-14	1
5	7650099999	2016-10-14	1
6	7650099999	2016-11-24	1
7	7650099999	2016-11-24	1
8	7650099999	2016-11-25	1

Showing the first 1000 rows.

	station	updated_date_rain	rain
1	7650099999	2016-01-01	1
2	7650099999	2016-01-02	1
3	7650099999	2016-01-05	1
4	7650099999	2016-01-05	1
5	7650099999	2016-01-20	1
6	7650099999	2016-01-26	1
7	7650099999	2016-02-14	1
8	7650099999	2016-02-14	1

Showing the first 1000 rows.

```
# Get the hourly rain and snow data with filters as well
rain_snow = spark.sql('''
select if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date))
,':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1 hour) as updated_date,
station,
                                                max(int(left(AW1,2))) as weather_type,
                                                avg(int(left(AN1,3))) as snow_hours,
                                                avg(int(substring(AN1,5,4))) as snow_depth_in,
                                                avg(int(left(aa1,2))) as precipitation_period_time_min,
                                                avg(int(substring(aa1,4,4))) as precipitation_inches
                                          from weather
                                         where int(substring(aa1,4,4)) != 9999
                                         group by 1,2
''')

rain_snow.createOrReplaceTempView("rain_snow")
display(rain_snow)
```

	updated_date	station	weather_type	snow_hours	snow_depth_in	precipitation_period_time_min	precipitation_inches
1	2019-11-05T08:00:00.000+0000	8186099999	21	null	null	1	1
2	2019-10-21T18:00:00.000+0000	8201099999	null	null	null	3	6

3	2019-09-15T18:00:00.000+0000	8202099999	null	null	null	3	40
4	2019-11-12T21:00:00.000+0000	8202099999	40	null	null	3	1
5	2019-01-20T16:00:00.000+0000	8210099999	61	null	null	1	0
6	2019-01-30T01:00:00.000+0000	8210099999	null	null	null	1	0
7	2019-01-31T21:00:00.000+0000	8210099999	10	null	null	3	0
8	2019-02-08T02:00:00.000+0000	8210099999	null	null	null	1	0

Showing the first 1000 rows.


```
# Take the weather data, fix the timestamp to record hourly, and take averages of each of the various weather conditions along with
fitlering by trustable data
# bring in the hours rain and snow data as well
# bring in rain and thunder indicator

weather_final = spark.sql('''select a.date,
                                  if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) ,
':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1 hour) as updated_date,
                                  concat(date, a.station) as date_station_id,
                                  a.station,
                                  --substring_index(wnd,',', 1) as wind_angle,
                                  --right(substring_index(wnd,',', 2),1) as wind_angle_qual,
                                  avg(int(right(substring_index(wnd,',', 4),4))) as wind_speed,
                                  avg(int(substring_index(vis,',', 1))) as horzn_dist,
                                  avg(int(right(substring_index(vis,',', 2),1))) as horzn_dist_qual,
                                  avg(int(right(substring_index(tmp,',', 1),4))) as temp_c,
                                  avg(int(right(substring_index(tmp,',', 2),1))) as temp_c_qual,
                                  avg(int(substring_index(dew,',', 1))) as dew_point_temp,
                                  max(weather_type) as weather_type,
                                  avg(snow_hours) as snow_hours,
                                  avg(snow_depth_in) as snow_depth_in,
                                  avg(precipitation_period_time_min) as precipitation_period_time_min,
                                  avg(precipitation_inches) as precipitation_inches,
                                  ifnull(rain,0) as rain_ind,
                                  ifnull(thunder,0) as thunder_ind
                                  from weather a
                                  left join rain_snow b
                                  on if(minute(date) <30, to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) ,
':00:00')),to_timestamp(concat(string(to_date(date)), ' ' , string(hour(date)) , ':00:00')) + interval 1 hour) = b.updated_date and
a.station = b.station

                                  left join rain_d c
                                 on a.date = c.updated_date_rain and a.station  = c.station

                                  left join thunder_d d
                                 on  a.date = d.updated_date_thunder and a.station = d.station

                                  where report_type not in ('SOD','SOM')
                                  --and substring_index(wnd,',', 1) != 999
                                  and right(substring_index(wnd,',', 4),4) != 999
                                  and substring_index(vis,',', 1) != 9999
                                  and int(substring_index(vis,',', 1)) != 999999
                                  and right(substring_index(vis,',', 2),1) in (1,5)
                                  and right(substring_index(tmp,',', 2),1) in (1,5)
                                  and a.station is not null --= 72528014733
                                  group by 1,2,3,4,16,17
                                  --having avg(int(right(substring_index(vis,',', 2),1))) is not null''')

weather_final.createOrReplaceTempView("weather_final")
display(weather_final)


weather_final.count()
```

	date	updated_date	date_station_id	station	wind_speed	horzn_dist	horzn_dis
1	2015-09-11T04:00:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:00:007629399999	7629399999	10	9000	1
2	2015-09-11T04:01:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:01:0070119600102	70119600102	31	16093	5
3	2015-09-11T04:04:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:04:0072508654734	72508654734	36	4828	5
4	2015-09-11T04:08:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:08:0071092099999	71092099999	57	4426	1
5	2015-09-11T04:15:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:15:0072026153976	72026153976	21	16093	5
6	2015-09-11T04:15:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:15:0072232403071	72232403071	26	16093	5
7	2015-09-11T04:16:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:16:0071882099999	71882099999	21	8047	1
8	2015-09-11T04:20:00.000+0000	2015-09-11T04:00:00.000+0000	2015-09-11 04:20:001088299999	1088299999	0	900	1

Showing the first 1000 rows.

Out[14]: 308505050

```
# bring in the average day and create the precipitation_inches_final

weather_final_2 = spark.sql('''select to_date(date_trunc('DD',date),'YYYY-MM-DD') as datee,
                                    updated_date,
                                    date_Station_id,
                                    a.station,
                                    wind_speed,
                                    horzn_dist,
                                    horzn_dist_qual,
                                    temp_c,
                                    temp_c_qual,
                                    dew_point_temp,
                                    weather_type,
                                    snow_hours,
                                    snow_depth_in,
                                    precipitation_period_time_min,
                                    precipitation_inches,
                                    precipitation_period_time_min_d,
                                    precipitation_inches_d,
                                    if(precipitation_inches/precipitation_period_time_min is null,
precipitation_inches_d/precipitation_period_time_min_d,precipitation_inches/precipitation_period_time_min) as
precipitation_inches_min_final,
                                    thunder_ind
                                from weather_final a

                                left join weather_day b
                                on date_trunc('DD',date) = b.updated_date_d and a.station = b.station
                                ''')
weather_final_2.createOrReplaceTempView('weather_final_2')
display(weather_final_2)

weather_final_2.count()
```

	datee	updated_date	date_Station_id	station	wind_speed	horzn_dist	horzn_dist_qual	temp_
1	2015-01-01	2015-01-01T18:00:00.000+0000	2015-01-01 18:00:0011157099999	11157099999	10	30000	1	16
2	2015-01-01	2015-01-01T09:00:00.000+0000	2015-01-01 09:00:0011157099999	11157099999	10	25000	1	14
3	2015-01-01	2015-01-01T21:00:00.000+0000	2015-01-01 21:00:0011157099999	11157099999	10	400	1	52
4	2015-01-01	2015-01-01T06:00:00.000+0000	2015-01-01 06:00:0011157099999	11157099999	10	12000	1	21
5	2015-01-01	2015-01-01T00:00:00.000+0000	2015-01-01 00:00:0011157099999	11157099999	10	8000	1	29
6	2015-01-01	2015-01-01T12:00:00.000+0000	2015-01-01 12:00:0011157099999	11157099999	10	25000	1	2
7	2015-01-01	2015-01-01T15:00:00.000+0000	2015-01-01 15:00:0011157099999	11157099999	20	25000	1	3
8	2015-01-01	2015-01-01T03:00:00.000+0000	2015-01-01 03:00:0011157099999	11157099999	10	9000	1	23

Showing the first 1000 rows.

Out[17]: 308505050

```
# Store the Weather Data

dbutils.fs.rm("dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/weather_final_2", True)
weather_final_2.write.parquet("dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/weather_final_2")


# Data check on weather to ensure no duplicates
weather_final_2.select('date_station_id').distinct().count()
```

Out[18]: 308505050

```
# # read in the final weather data set and the airlines that was previously saved before running the join
# weather_final_2 = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/weather_final_2")
# weather_final_2.createOrReplaceTempView('weather_final_2')
# airlines_with_updated_weather_hourly = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_updated_weather_hourly")
# airlines_with_updated_weather_hourly.createOrReplaceTempView('airlines_with_updated_weather_hourly')
```

Join

- put the column name and a group by number for each field that we want to bring in

```
# Join the airlines data to the weather data

airlines_with_weather_final = spark.sql('''select fl_date,
                                           concat(fl_date, op_carrier_fl_num, op_carrier_airline_id, origin_city_name) as id,
                                            op_carrier_airline_id,
                                           op_unique_carrier,
                                           year,
                                           quarter,
                                           month,
                                           day_of_month,
                                           day_of_week,
                                           origin_city_name,
                                           op_carrier_fl_num,
                                           op_carrier,
                                           origin_State_abr,
                                           dest_state_abr,
                                           origin,
                                           dest,
                                           arr_delay_new,
                                           distance,
                                           weather_delay,
                                           nas_delay,
                                           security_delay,
                                           carrier_delay,
                                           late_aircraft_delay,
                                           origin_station_id,
                                           local_dep_time,
                                           timezone,
                                           utc_dep_time,
                                           dep_delay_new,
                                           dep_del15,
                                           dep_time_diff,
                                           prev_dep_del15,
                                           max(thunder_ind) as thunder_ind,
                                           avg(temp_c) as temp_c,
                                           avg(wind_speed) as wind_speed,
                                           avg(horzn_dist) as horzn_dist,
                                           avg(precipitation_inches_min_final) as precipitation_period_time_min,
                                           int(if(day_of_week >=5,1,0)) as weekend
                                    from airlines_with_updated_weather_hourly a
                                    inner join weather_final_2 b
                                    on a.updated_dep_time = b.updated_date and a.origin_station_id = b.station
                                    group by 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
                                    order by concat(fl_date, op_carrier_fl_num, op_carrier_airline_id)
''')

airlines_with_weather_final.createOrReplaceTempView("airlines_with_weather_2")
display(airlines_with_weather_final)
airlines_with_weather_final.count()
```

	fl_date	id	op_carrier_airline_id	op_unique_carrier	year	quarter	month
1	2015-01-01	2015-01-01100119805Dallas/Fort Worth, TX	19805	AA	2015	1	1
2	2015-01-01	2015-01-01100119977Chicago, IL	19977	UA	2015	1	1
3	2015-01-01	2015-01-01100120409Buffalo, NY	20409	B6	2015	1	1
4	2015-01-01	2015-01-0110020409Los Angeles, CA	20409	B6	2015	1	1
5	2015-01-01	2015-01-01100219805Houston, TX	19805	AA	2015	1	1
6	2015-01-01	2015-01-01100219977Denver, CO	19977	UA	2015	1	1
7	2015-01-01	2015-01-01100220409Fort Lauderdale, FL	20409	B6	2015	1	1
8	2015-01-01	2015-01-01100319393Fort Lauderdale, FL	19393	WN	2015	1	1

Showing the first 1000 rows.

```
Out[19]: 28082932

airlines_with_weather_final.select('id').distinct().count()

Out[20]: 28082873
```

```
# drop the nulls from precipitation data
from pyspark.sql.functions import isnan, when, count, col

airlines_with_weather_final = airlines_with_weather_final.na.drop(subset=["precipitation_period_time_min"])

airlines_with_weather_final = airlines_with_weather_final.drop('local_dep_time').drop('utc_dep_time').drop('timezone')

display(airlines_with_weather_final.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in
airlines_with_weather_final.columns]))
```

	fl_date	id	op_carrier_airline_id	op_unique_carrier	year	quarter	month	day_of_month	day_of_week
1	0	0	0	0	0	0	0	0	0

Showing all 1 rows.

```
#SAVING Spark Dataframe to Directory
userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'         #CHANGE USERNAME IF FILE STORED ELSEWHERE
finalproject_path = userhome + "/FINAL_PROJECT/"
file_to_store = airlines_with_weather_final                                  #name of Spark Dataframe (to save in database)
filename = "airlines_with_weather_final"                                     #new file name in database
dbutils.fs.rm(finalproject_path+filename, True)                       #remove file if there already is an existing one, be careful with
this!!!
file_to_store.write.format("parquet").save(finalproject_path+filename)


# # read in the final weather data set and the airlines that was previously saved before running the join
# airlines_with_weather_train = spark.read.option("header",
"true").parquet(f"dbfs:/user/nathan.nusaputra@ischool.berkeley.edu/FINAL_PROJECT/airlines_with_weather_train")
# airlines_with_weather_train.createOrReplaceTempView('airlines_with_weather_train')


# Split train and test data
airlines_with_weather_train, airlines_with_weather_test = airlines_with_weather_final.randomSplit([0.9, 0.1], seed=12345)


#SAVING Spark Dataframe to Directory
userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'         #CHANGE USERNAME IF FILE STORED ELSEWHERE
finalproject_path = userhome + "/FINAL_PROJECT/"
file_to_store = airlines_with_weather_train                                  #name of Spark Dataframe (to save in database)
filename = "airlines_with_weather_train"                                     #new file name in database
dbutils.fs.rm(finalproject_path+filename, True)                       #remove file if there already is an existing one, be careful with
this!!!
file_to_store.write.format("parquet").save(finalproject_path+filename)


airlines_2 = airlines_with_weather_train.drop('local_dep_time').drop('utc_dep_time').drop('timezone')

airlines_2.columns
```

```
Out[24]: ['fl_date',
 'id',
 'op_carrier_airline_id',
 'op_unique_carrier',
 'year',
 'quarter',
 'month',
 'day_of_month',
 'day_of_week',
 'origin_city_name',
 'op_carrier_fl_num',
 'op_carrier',
 'origin_State_abr',
 'dest_state_abr',
 'origin',
 'dest',
 'arr_delay_new',
 'distance',
 'weather_delay',
 'nas_delay',
 'security_delay',
```

```
#SAVING Spark Dataframe to Directory
userhome = 'dbfs:/user/nathan.nusaputra@ischool.berkeley.edu'         #CHANGE USERNAME IF FILE STORED ELSEWHERE
finalproject_path = userhome + "/FINAL_PROJECT/"
file_to_store = airlines_with_weather_test                                  #name of Spark Dataframe (to save in database)
filename = "airlines_with_weather_test"                                     #new file name in database
dbutils.fs.rm(finalproject_path+filename, True)                       #remove file if there already is an existing one, be careful with
this!!!
file_to_store.write.format("parquet").save(finalproject_path+filename)
```