

# Taxi Fare Analysis: CS 512 Final Project

Lisa Wilson

3/11/2020

## Overview of the dataset

This dataset contains three sets of data related to New York City taxi services in 2013. The first is information on fares, including time of pickup, payment type (e.g., cash or card), fare amount, tip amount, and total amount. The second is information on the trips themselves, including time of pickup and dropoff, pickup and dropoff latitude and longitude, trip distance, trip time, and passenger count. Both of these sets were obtained from Academic Torrents, where they had been uploaded by Chris Whong, who had submitted a Freedom of Information Law (FOIL) request for them in 2014. Each torrent file contained zipped folders of csv files for data from each month. The third is information on the latitude and longitude of landmarks in New York City, which was scraped from the Wikipedia page “List of National Historic Landmarks” and saved as a csv file.

## Sample of records

The following is a sample of records from one of the fare data csv files (first 100 rows included in zip file as fareTrip2013\_8\_samp.csv):

	medallion	hack_license	vendor_id	pickup_datetime	payment_type	fare_amount	surcharge	mta_tax	tip_amount	tolls_amount	total_amount
1	3418135604	B25386A1F259C87449430593E904FDBC	VTS	8/30/13 7:57	CSH	41.5	0	0.5	0	0	42
2	6D382A7682C30DC64F3F12976EF9386	A603A905FAA46E8FF2A97A143328D938	CMT	8/30/13 23:26	CSH	31	0.5	0.5	0	5.33	37.33
3	6D49E494913752B75B2685E0019FF3D5	3F0BF90A5D71741840B25600A89E225	CMT	8/30/13 9:18	CSH	5.5	0	0.5	0	0	6
4	4C4A0AFC432A1A87E97ED8F18403FF6E	BA20A20E2CF85EF7B00162D711394C7E	CMT	8/26/13 23:27	CSH	23	0.5	0.5	0	5.33	29.33
5	1258CA1DF5E2A9E9A9F7848408A7AAE8	8C140CF69CAA2A9A0DFAFD99E00536A1	CMT	8/29/13 10:57	CSH	14	0	0.5	0	0	14.5
6	3B0E8DC736D1E0D35C698D7B0BF9CBFA	95E18898C718FB1DC76B939937E043F3	CMT	8/27/13 11:37	CSH	31.5	0	0.5	0	5.33	37.33
7	1A575E3980D761E6496F0027F9E34D19	FAE278EFC2171382209D06992A5FE59D	CMT	8/29/13 5:41	CSH	13	0.5	0.5	0	0	14
8	A3A0C08B131E9C95B012947F22081F	B56ECC02B67AD98E9EBEF80E8A9FB6D	CMT	8/28/13 19:53	CSH	31.5	1	0.5	0	5.33	38.33
9	EF74C463981D4D28888BCF0694DC821	3251A220F065378A4E358E26E611DA8D	CMT	8/26/13 1:08	CSH	13	0.5	0.5	0	0	14
10	C647516A7B73EC8105E8DE5494B4CDB9	80C4140E2FE17888549FE2825D49238F	CMT	8/27/13 21:59	CSH	10.5	0.5	0.5	0	0	11.5

The following is a sample of records from one of the trip data csv files (first 100 rows included in zip file as tripTrip2013\_8\_samp.csv):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	medallion	hack_license	vendor_id	rate_code	store_and_fwd_flag	pickup_datetime	dropoff_datetime	passenger	trip_time_in_secs	trip_distance	pickup_long	pickup_latitude	dropoff_long	dropoff_latitude
2	3418135604	B25386A1F2	VTS	1		8/30/13 7:57	8/30/13 8:30	5	1980	14.58	-73.791359	40.645657	-73.922501	40.758766
3	6D382A7682	A603A905FA	CMT	1	N	8/30/13 23:26	8/30/13 23:46	2	1177	11	-73.862724	40.769062	-73.976845	40.764595
4	6D49E49491	3F0BF90A5	CMT	1	N	8/30/13 9:18	8/30/13 9:24	1	357	0.8	-73.991653	40.750324	-73.98642	40.742924
5	4C4A0AFC43	BA20A20E2	CMT	1	N	8/26/13 23:27	8/26/13 23:42	4	938	7.7	-73.975372	40.756237	-73.867119	40.721886
6	1258CA1DF5	8C140CF69	CMT	1	N	8/29/13 10:57	8/29/13 11:19	2	1270	2.1	-73.99102	40.750912	-73.996727	40.767578
7	3B0E8DC736	95E18898C7	CMT	1	N	8/27/13 11:37	8/27/13 12:00	1	1392	10.9	-73.872948	40.774212	-73.981438	40.743946
8	1A575E3980	FAE278EFC2	CMT	1	N	8/29/13 5:41	8/29/13 5:55	3	811	3.5	-73.991791	40.757999	-73.99514	40.717392
9	A3A0C08B1	B56ECC02B	CMT	1	N	8/28/13 19:53	8/28/13 20:21	1	1678	10	-73.873009	40.773956	-73.997437	40.730167
10	EF74C46398	3251A220F0	CMT	1	Y	8/26/13 1:08	8/26/13 1:19	1	697	3.6	-73.999733	40.72847	-73.983963	40.769341
11	C647516A7B	80C4140E2F	CMT	1	N	8/27/13 21:59	8/27/13 22:10	1	638	2.6	-73.972351	40.756321	-74.000221	40.730488

The following is a sample of records from the landmark data table from Wikipedia:

## National Historic Landmarks in New York City [\[ edit \]](#)

[1] ↕	Landmark name ↕	Image ↕	Date designated <sup>[2]</sup> ↕	Location ↕	County ↕	Description ↕
1	69th Regiment Armory	 <a href="#">More images</a>	June 19, 1996 (#93001538 <a href="#">↗</a> )	Manhattan  40°44′30″N 73°59′01″W	New York	Home of the watershed <a href="#">Armory Show</a> in 1913, which introduced America to modern art
2	Admiral David Glasgow Farragut Gravesite	 <a href="#">More images</a>	October 17, 2012 (#12001008 <a href="#">↗</a> )	Bronx  40°53′32″N 73°51′57″W	Bronx	Only intact known property directly associated with Admiral <a href="#">David Farragut</a>
3	African Burial Ground	 <a href="#">More images</a>	April 19, 1993 (#93001597 <a href="#">↗</a> )	Manhattan  40°42′52″N 74°00′16″W	New York	Dedicated as National Monument on October 5, 2007; burial site in Lower Manhattan of over 400 Africans from the 17th and 18th centuries

The following is a sample of records from the landmark data csv file (included in zip file as `lm_df.csv`):

1	Landmark name	Location name	Lat	Long
2	69th Regiment Armory	Manhattan	40.741648	73.983607
3	Admiral David Glasgow Farragut Gravesite	Bronx	40.892165	73.86586
4	African Burial Ground	Manhattan	40.714558	74.004384
5	Ambrose (lightship)	Manhattan	40.704844	74.002467
6	American Stock Exchange Building	Manhattan	40.709	74.0126
7	Louis Armstrong House	Corona	40.754556	73.861557
8	Chester A. Arthur House	Manhattan	40.74279	73.982196
9	Alice Austen House	Rosebank	40.615129	74.062952
10	Bartow-Pell Mansion	Pelham Bay Park	40.871748	73.805578
11	Bayard-Condict Building	Manhattan	40.7263	73.9956

As noted in a later section, the landmark longitudes were converted to negative values in Dataprep when I realized they were supposed to be negative when in decimal form.

## Data wrangling process

### Fare and trip torrent files

While I had originally tried downloading the torrent files for the fare and trip data from Academic Torrents using BitTorrent and then uploading the zip files to my VM instance, I realized it would be faster and easier to use `aria2` to download the files to my instance. Once I had the VM instance created, I installed `aria2`, then used the line `aria2c <URL to the torrent file>` to download the fare and trip files. After unzipping the fare file, I noticed that each of the 12 month files were stored as `.csv.zip` files, so I unzipped each of these files using `unzip *.csv.zip` for the fare data. For the trip data, I got a “bad zipfile offset” error when trying to unzip, so I instead used `jar xvf tripData2013.zip` and then `unzip *.csv.zip` to access the csv files themselves. Finally, I used `gsutil -m cp "*.csv" gs://taxi-final/fareData2013` and `gsutil -m cp "*.csv" gs://taxi-final/tripData2013` to move the fare and trip csv files to the Google Cloud Storage bucket.

Next, I imported the fare and trip data separately to flows in Dataprep, where I selected the columns I wanted for analysis, ensured that the float or decimal type columns had the correct data type, and extracted

month, day, day of the week, and hour as separate variables from the pickup and dropoff time columns. The Dataprep recipes are provided below:

The image shows three screenshots of the Dataprep interface, each displaying a recipe for a different dataset. Each recipe is a list of steps to be applied to the data.

- fareData2013 - 2:**
  - 1 Change tip\_amount type to Decimal
  - 2 Change tolls\_amount type to Decimal
  - 3 Create month\_pickup from MONTH(pickup\_datetime)
  - 4 Create day\_pickup from DAY(pickup\_datetime)
  - 5 Create weekday\_pickup from WEEKDAY(pickup\_datetime)
  - 6 Create hour\_pickup from HOUR(pickup\_datetime)
  - 7 Delete medallion
  - 8 Delete pickup\_datetime
  - 9 Delete surcharge
  - 10 Delete mta\_tax
  - 11 Delete tolls\_amount
- tripData2013 - 2:**
  - 1 Set trip\_distance to NUMFORMAT(\$col, '#.##')
  - 2 Create month\_pickup from MONTH(pickup\_datetime)
  - 3 Create day\_pickup from DAY(pickup\_datetime)
  - 4 Create weekday\_pickup from WEEKDAY(pickup\_datetime)
  - 5 Create hour\_pickup from HOUR(pickup\_datetime)
  - 6 Create month\_dropoff from MONTH(dropoff\_datetime)
  - 7 Create day\_dropoff from DAY(dropoff\_datetime)
  - 8 Create weekday\_dropoff from WEEKDAY(dropoff\_datetime)
- tripData2013 - 2:**
  - 7 Create day\_dropoff from DAY(dropoff\_datetime)
  - 8 Create weekday\_dropoff from WEEKDAY(dropoff\_datetime)
  - 9 Create hour\_dropoff from HOUR(dropoff\_datetime)
  - 10 Delete pickup\_datetime
  - 11 Delete store\_and\_fwd\_flag
  - 12 Delete rate\_code
  - 13 Delete dropoff\_datetime
  - 14 Delete medallion
  - 15 Delete vendor\_id

Finally, I ran the job to export each dataset to BigQuery tables “fare” and “trip.” In BigQuery, I created test sets of 100 to 1000 rows on which to test my PySpark code, and I also created the table “chrysler” that included the pickup day of the week and the trip distance for trips where the dropoff was within a one-block radius of the Chrysler Building. To convert one-twentieth of a mile (which is approximately one block) to latitude and longitude, I used the following code, which was suggested on a StackExchange answer (<https://gis.stackexchange.com/a/15572>).

```
lat <- 0.05/69
long <- lat/cos((40.7*pi)/180)
```

Having obtained 0.0007 for the latitude margin and 0.0010 for the longitude margin, I used the following query to create the table of trips where the taxi dropped off within a block of the Chrysler Building.

```
SELECT weekday_pickup, trip_distance, dropoff_latitude, dropoff_longitude FROM taxi.trip
WHERE dropoff_latitude < 40.7518 AND dropoff_latitude > 40.7516
AND dropoff_longitude < -73.9752 AND dropoff_longitude > -73.9754;
```

## Landmark data from Wikipedia

To obtain the latitudes and longitudes of NYC landmarks from the Wikipedia page, I used the `read_html` function from `pandas`, then formatted the table as a dataframe of only the landmark name and the location coordinates:

```
wiki = "https://en.wikipedia.org/wiki/List_of_National_Historic_Landmarks_in_New_York_City"
table = pd.read_html(wiki)[0]
lm_df = pd.DataFrame(table)[["Landmark name", "Location"]]
```

The latitudes and longitudes in decimal form were within strings, so I used the `str.split` method several times and subsetted the results to get separate columns for latitude and longitude:

```

new = lm_df["Location"].str.split(r' (?<!^)([0-9]+)', n = 1, expand = True)
newnew = lm_df["Location"].str.split("/", n = 1, expand = True)
lat = newnew[1].str.split(u'°', n = 2, expand = True)
long = lat[1].str.split("N ", n = 1, expand = True)

lm_temp = pd.DataFrame()
lm_temp["Lat"] = lat[0]
lm_temp["Long"] = long[1]

```

One consequence of the above code is that longitude was saved as a float, which was the desired outcome, while latitude was saved as a string. After trying to adjust my `str.split` process to correct this, I instead landed on the following code for removing a unicode character from the beginning of the latitude values, then converting latitude to a float and ensuring that longitude was a float:

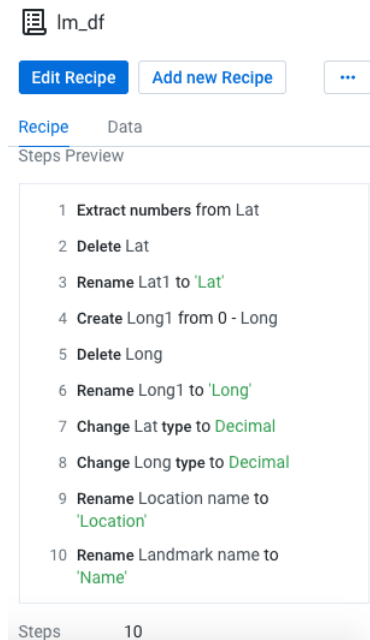
```

for i in range(len(lm_temp["Lat"])):
    lm_temp["Lat"][i] = float(lm_temp["Lat"][i].replace('\uffff', ''))

lm_temp["Lat"] = lm_temp["Lat"].astype(dtype=np.float64)
lm_temp["Long"] = lm_temp["Long"].astype(dtype=np.float64)

```

Finally, I compiled the landmark name, latitude, and longitude into a dataframe and saved that dataframe to a csv file, which I then uploaded directly to my Google Cloud Storage bucket because of its small file size. I then imported this csv to a Dataprep flow and used the following recipe to prepare the data for export into a BigQuery table “landmark”:



The screenshot shows the Dataprep interface for a dataset named 'lm\_df'. At the top, there are buttons for 'Edit Recipe', 'Add new Recipe', and a menu icon. Below these, there are tabs for 'Recipe' and 'Data', with 'Recipe' selected. The main area is titled 'Steps Preview' and contains a list of 10 steps:

- 1 Extract numbers from Lat
- 2 Delete Lat
- 3 Rename Lat1 to 'Lat'
- 4 Create Long1 from 0 - Long
- 5 Delete Long
- 6 Rename Long1 to 'Long'
- 7 Change Lat type to Decimal
- 8 Change Long type to Decimal
- 9 Rename Location name to 'Location'
- 10 Rename Landmark name to 'Name'

At the bottom, there are tabs for 'Steps' and '10', with 'Steps' selected.

One major adjustment I made in Dataprep was to make the longitude values negative. This was because I realized that the scraped data from the Wikipedia page used the directions East and West rather than positive and negative, which was lost when I stripped the strings apart. To match the format of the latitude and longitude in the trip data, I converted the landmark longitudes to negative.

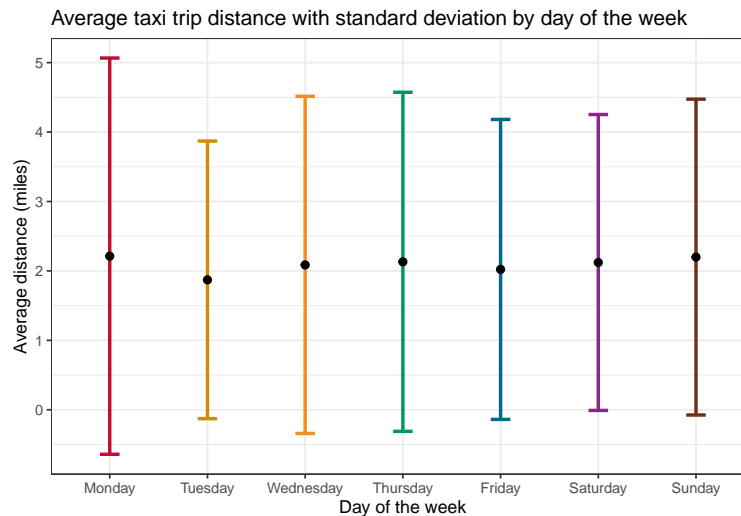
(Script to scrape table data and create csv file included in zip file as `nyclm_latlong.py`.)

## Data analysis questions

### For dropoffs near the Chrysler Building, how do average trip distances vary depending on the day of the week?

For this question, I wanted to see how some trip metric varied based on the day of the week for dropoffs near some New York City landmark. I chose the Chrysler Building arbitrarily, mainly because it is beautiful. My guess is that the distances would be higher for weekdays, due to the work week, than the weekends.

To answer this question, I created key-value tuples for each row in the table I created of dropoffs within a block of the Chrysler Building, with the key being the day of the week and the value being the trip distance. I then used `reduceByKey` and `countByKey` to calculate the average and standard deviation of trip distance for each day of the week.

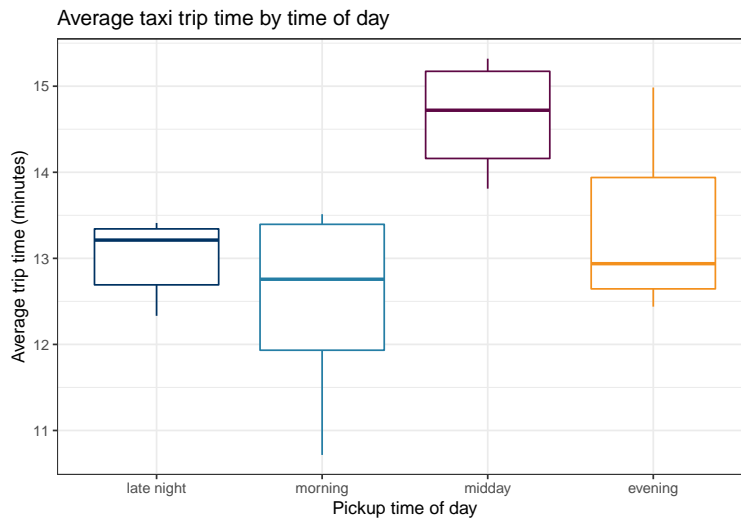
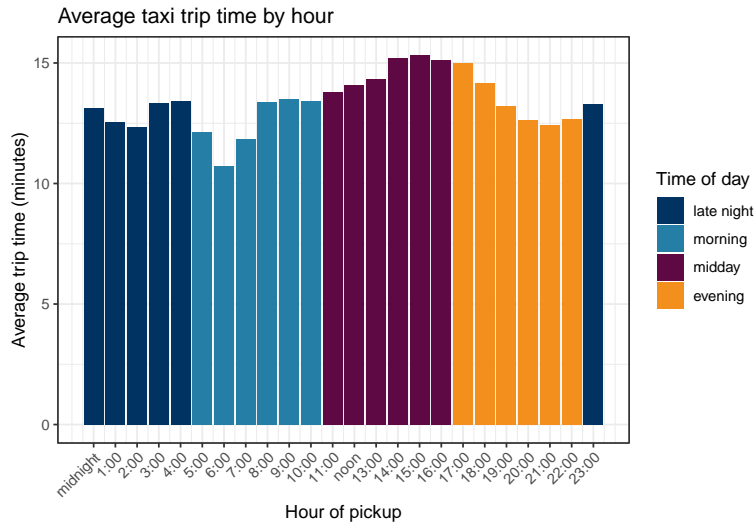


The average distance does not vary much by day of the week, although the trip distance is most variable on Mondays and least variable on Tuesdays. All the average distances were slightly above 2 miles, with the exception of Tuesday, which is slightly below at 1.87 miles.

### How does average trip time vary depending on the pickup hour?

For this question, I wanted to look at the variation in average trip time depending on what hour the passenger(s) was picked up. I anticipated that trip times would be longer during the evening rush hour and in the late night and early morning when people are returning home from a night out.

To answer this question, I created key-value tuples of hour of the day for the pickup as the key and trip time in seconds as the value, then calculated the average of trip time in minutes for each hour of the day.

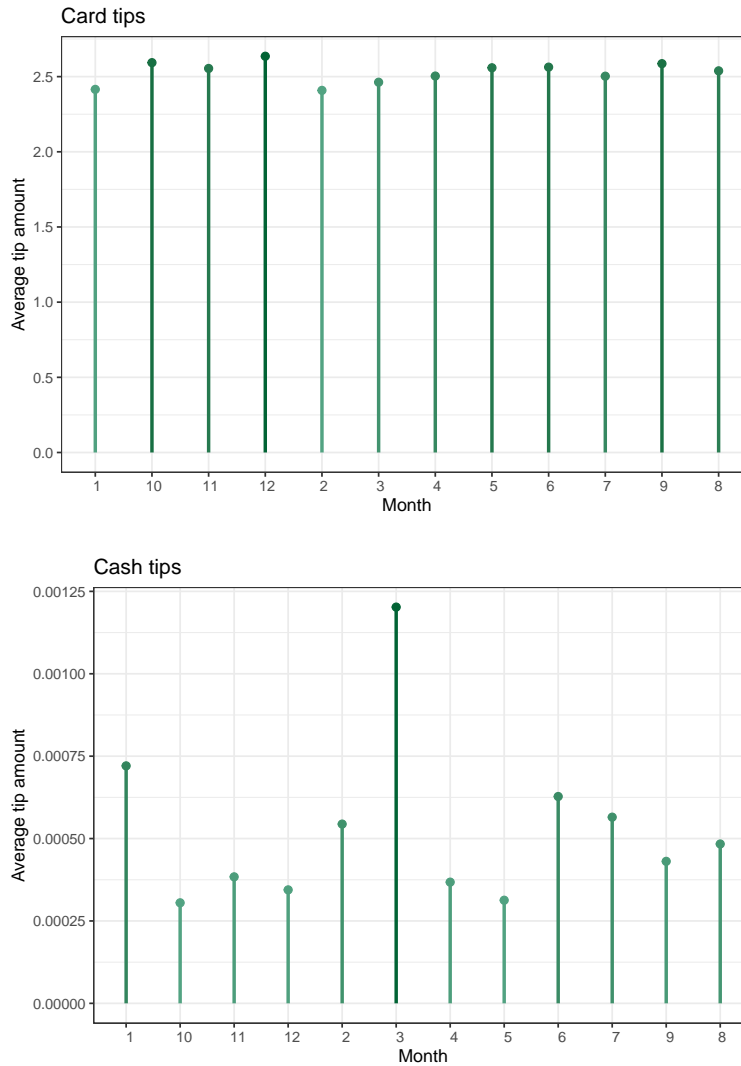


The two pickup hours with the highest average trip times are 14:00 at 15.2 minutes and 15:00 at 15.3 minutes, and the two with the lowest average trip times are 6:00 at 10.7 minutes and 7:00 at 11.9 minutes. The midday hours (which I defined as 11:00 to 16:00) have the highest average trip times out of the four times of day.

### How do tip amounts – for both cash and card payments – vary depending on month?

For this question, my hunch is that cash tips are lower than card tips overall – because of the ease of putting a higher tip on a card – and that tips may be higher in December around the holidays, or in the tourist-season summer months.

To answer this question, I filtered the RDDs by payment type, created key-value tuples of hour of the day for the pickup as the key and trip time in seconds as the value, and then calculated the average of trip time in minutes for each hour of the day.



The average tips for card payments are clearly higher than for cash payments: the average tip across the months for card payments is around \$2.50, and the average tip for cash payments is less than 1 cent. While the average tips for card payments are relatively close to each other, the month with the highest average tip is December at \$2.64 while the month with the lowest average tip is February at \$2.41. Again, all the average cash tips are less than 1 cent, but the month with the highest is March at \$0.0012 and with the lowest is May at \$0.0003. In addition, the maximum card tip in 2013 was \$200.00, and the maximum cash tip was \$121.50.

(Code for visualizations included in zip file as final\_proj\_plots.R.)

## Process of going from the original data to BigQuery and to Spark

### Fare and trip data

Part of the process of importing the fare and trip data into BigQuery is described in the data wrangling section, particularly the steps that involved the VM instance to upload the data in the torrent files into Google Cloud Storage. Importing the data from the Google Cloud Storage bucket into Dataprep was straightforward, and from there, I was able to ensure that the data was in a form that would work for my analysis. Because I had previously encountered problems with the data type not correctly carrying over from Dataprep into BigQuery, I changed the data type of the tip amount to decimal in the fare data so that it would export to

BigQuery as a float and changed the data type of the trip distance to a number. For the pickup datetime in both the fare and trip datasets and dropoff datetime in the trip dataset, I extracted the month, day, weekday, and hour using the Dataprep functions. I also deleted the columns that I did not anticipate using in the analysis. Using the Append BigQuery option when running the Dataprep job, the formatted fare data exported automatically to the “fare” table within the “taxi” dataset I had created in BigQuery, and the formatted trip data exported automatically to the “trip” table within the “taxi” dataset I had created in BigQuery.

To get the data into Spark, I used code very similar to that used for the Spark Plane Distance assignments that was originally presented in the “Running Real Jobs” video from the Week 7 module. The code below is an example of defining a SparkContext and connecting to the “trip” table within the BigQuery dataset I used for this project:

```
sc = pyspark.SparkContext()

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output'.format(bucket)

conf = {
    # Input Parameters.
    'mapred.bq.project.id': project,
    'mapred.bq.gcs.bucket': bucket,
    'mapred.bq.temp.gcs.path': input_directory,
    'mapred.bq.input.project.id': 'final-project-269604',
    'mapred.bq.input.dataset.id': 'taxi',
    'mapred.bq.input.table.id': 'trip2',
}

table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf=conf
)
```

To read the data into PySpark as an RDD, the following code was used:

```
vals = table_data.values()
vals = vals.map(lambda line: json.loads(line))
```

From here, PySpark code, including mapping lambda functions and `reduceByKey` on the RDDs, was used to do the appropriate analysis.

## Landmark data

Again, part of the process of importing the landmark data into BigQuery is described in the data wrangling section. In Dataprep, I primarily made the longitude values negative (for reasons described previously), ensured that the latitude and longitude values were in decimal form, and changed the names of the location name (e.g., Manhattan) and landmark name columns. Using the Append BigQuery option when running the Dataprep job, the formatted landmark data exported automatically to the “landmark” table within the “taxi” dataset I had created in BigQuery.

Because the landmark data was only used to determine the latitude and longitude bounds to use for analyzing trip data around landmarks of interest, the landmark data itself was not itself sent to Spark.



## Sample of BigQuery tables

Full “trip” table

trip

QUERY TABLE

COPY TABLE

DELETE TABLE

EXPORT

Schema

Details

Preview

Row	hack_license	hour_pickup	weekday_pickup	day_pickup	month_pickup	hour_dropoff	weekday_dropoff	day_dropoff	month_dropoff
1	A690B149890E6813644A42DB3B15AD28	10	1	1	7	11	1	1	7
2	A690B149890E6813644A42DB3B15AD28	10	1	1	7	10	1	1	7
3	A690B149890E6813644A42DB3B15AD28	10	1	1	7	10	1	1	7
4	A690B149890E6813644A42DB3B15AD28	10	1	1	7	10	1	1	7
5	A690B149890E6813644A42DB3B15AD28	8	1	1	7	8	1	1	7
6	A690B149890E6813644A42DB3B15AD28	9	1	1	7	9	1	1	7

Rows per page:

100

1 - 100 of 173179769

First page

<

>

>|

passenger_count	trip_time_in_secs	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
1	2249	14.8	-73.975502	40.7542	-73.799599	40.66851
1	3	0.0	-73.934891	40.741241	-73.934875	40.741241
1	3	0.0	-73.986794	40.7593	-73.986794	40.7593
1	3	0.0	-73.934853	40.741222	-73.93486	40.741219
1	449	1.7	-74.002831	40.739288	-74.002335	40.720345
1	537	3.3	-74.008003	40.714844	-73.985825	40.754723

Rows per page:

100

1 - 100 of 173179769

First page

<

>

>| Last page

“trip2” table with reduced columns

Query used:

```
SELECT hour_pickup, trip_time_in_secs AS time_sec FROM taxi.trip
WHERE hour_pickup IS NOT NULL AND trip_time_in_secs IS NOT NULL;
```

trip2		
Schema Details Preview		
Row	hour_pickup	time_sec
1	0	917
2	0	1044
3	0	319
4	0	568
5	0	495
6	0	2348

Full “fare” table

fare

QUERY TABLE

COPY TABLE

DELETE TABLE

EXPORT

Schema Details Preview

Row	hack_license	vendor_id	hour_pickup	weekday_pickup	day_pickup	month_pickup	payment_type	fare_amount	tip_amount	total_amount
1	D812F96FB918ABF2C4B4EE37372055A0	VTs	0	5	6	12	CRD	34.5	8.07	48.9
2	E22079BDD935685FA9BCD862913F96DF	CMT	0	5	27	12	CRD	4.5	1.1	6.6
3	7C653CC679536FAA4C1582017D15023B	VTs	0	3	4	12	CRD	3.5	0.8	5.3
4	9F1A1A09D34B59FCBA780CC580D40954	VTs	0	3	4	12	CRD	8.0	2.55	11.55
5	0D4AAFEDBEA23D0B920F724D49DEB382	VTs	0	3	4	12	CRD	25.0	2.0	27.5
6	D16C32B0E2463004C869B9F17E3D14E0	VTs	0	3	4	12	CRD	6.5	1.75	9.25

Rows per page: 100

1 - 100 of 173179759

First page

<

>

Last page

“fare2” table with reduced columns

Query used:

```
SELECT payment_type, month_pickup, tip_amount FROM taxi.fare;
```

fare2

Schema Details Preview

Row	payment_type	month_pickup	tip_amount
1	CRD	1	4.9
2	CRD	1	6.62
3	CRD	1	6.4
4	CRD	1	5.7
5	CRD	1	6.7
6	CRD	1	8.5

“landmark table”

landmark

QUERY TABLE

COPY TABLE

Schema Details Preview

Row	Name	Location	Lat	Long
1	Admiral David Glasgow Farragut Gravesite	Bronx	40.892165	-73.86586
2	New York Botanical Garden	Bronx	40.863611	-73.878333
3	United Workers Cooperatives	Bronx	40.866389	-73.869722
4	University Heights Campus (Bronx Community College of the City University of New York)	Bronx	40.857778	-73.912222
5	Woodlawn Cemetery	Bronx	40.889167	-73.873333
6	Louis Armstrong House	Corona	40.754556	-73.861557

Rows per page: 100

1 - 100 of 116

First p

PySpark scripts and output of Dataproc jobs

Average trip distance by day of the week for dropoffs near the Chrysler Building (chrysler.py)

```

import json
import pyspark
import pprint
from operator import add
import math

sc = pyspark.SparkContext()

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output3'.format(bucket)

conf = {
    # Input Parameters.
    'mapred.bq.project.id': project,
    'mapred.bq.gcs.bucket': bucket,
    'mapred.bq.temp.gcs.path': input_directory,
    'mapred.bq.input.project.id': 'final-project-269604',
    'mapred.bq.input.dataset.id': 'taxi',
    'mapred.bq.input.table.id': 'chrysler',
}

table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf=conf
)

vals = table_data.values()
vals = vals.map(lambda line: json.loads(line))
vals_wd = vals.map(lambda x: (str(x['weekday_pickup']), float(x['trip_distance'])))
sum_wd = vals_wd.reduceByKey(add)
count_wd = vals_wd.countByKey()
avg_wd = sum_wd.map(lambda x: (x[0], float(x[1]/count_wd[x[0]])))

# standard deviation
# had trouble using reduceByKey with built-in standard deviation functions in various packages
# so used code from the following blog post
# https://blog.scottlogic.com/2016/12/19/spark-unaffordable-britain.html
countByKey = vals_wd.map(lambda kvp: (kvp[0], 1)).reduceByKey(lambda a,b: a + b)
totalByKey = vals_wd.reduceByKey(lambda a,b: a + b)
sumSqByKey = vals_wd.map(lambda kvp: (kvp[0], kvp[1]**2)).reduceByKey(lambda a,b: a + b)
mean = totalByKey.join(countByKey).map(lambda kvp: (kvp[0], kvp[1][0] / kvp[1][1]))
avgSquare = sumSqByKey.join(countByKey).map(lambda kvp: (kvp[0], kvp[1][0] / kvp[1][1]))
std_wd = avgSquare.join(mean).map(lambda kvp: (kvp[0], math.sqrt(kvp[1][0] - kvp[1][1]**2)))

pprint.pprint(mean.collect())

pprint.pprint(avg_wd.collect())
pprint.pprint(std_wd.collect())

```

```
wd_avgstd = avg_wd.union(std_wd)
pprint.pprint(wd_avgstd.collect())
```

```
wd_avgstd.saveAsTextFile(output_directory)
```

```
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)
```

```
20/03/10 22:24:34 INFO org.spark_project.jetty.util.log: Logging initialized @3848ms
20/03/10 22:24:34 INFO org.spark_project.jetty.server.Server: jetty-9.3.z-SNAPSHOT, build timestamp: unknown, git hash: unknown
20/03/10 22:24:34 INFO org.spark_project.jetty.server.Server: Started @3974ms
20/03/10 22:24:34 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@1a986d85{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
20/03/10 22:24:34 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
  Fair Scheduler configuration file not found so jobs will be scheduled in FIFO order.
To use fair scheduling, configure pools in fairscheduler.xml or set spark.scheduler.allocation.file
to a file that contains the configuration.
20/03/10 22:24:35 INFO org.apache.hadoop.yarn.client.RMProxy:
  Connecting to ResourceManager at cluster-bf71-m/10.138.0.12:8032
20/03/10 22:24:35 INFO org.apache.hadoop.yarn.client.AHSProxy:
  Connecting to Application History server at cluster-bf71-m/10.138.0.12:10200
20/03/10 22:24:38 INFO org.apache.hadoop.yarn.client.impl.YarnClientImpl:
  Submitted application application_1583868653199_0007
class
com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.DefaultPlatform:
  cannot cast result of calling 'com.google.cloud.hadoop.repackaged.gcs.com.google.common.flogger.backend.log4j.Log4jBackendFactory#getInstance'
to 'com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.BackendFactory':
  java.lang.ClassCastException: Cannot cast com.google.cloud.hadoop.repackaged.gcs.com.google.common.flogger.backend.log4j.Log4jBackendFactory
to com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.BackendFactory

20/03/10 22:24:46 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: BigQuery connector version hadoop2-1.1.0
20/03/10 22:24:46 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from default credential.
20/03/10 22:24:46 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from given credential.
20/03/10 22:24:46 INFO com.google.cloud.hadoop.io.bigquery.BigQueryConfiguration: Using working path: 'gs://taxi-final/hadoop/tmp/bigquery/pyspark_input'
20/03/10 22:25:00 INFO com.google.cloud.hadoop.io.bigquery.UnshardedExportToCloudStorage: Setting FileInputFormat's inputPath to 'gs://taxi-final/hadoop/tmp/bigquery/pyspark_input'
20/03/10 22:25:00 INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat: Total input files to process : 1
[('1', 2.2125680933852125),
 ('3', 2.086708860759492),
 ('2', 1.8712714776632287),
 ('5', 2.0215102974828376),
 ('4', 2.131322620519159),
 ('7', 2.1995274102079394),
 ('6', 2.1216275659824033)]
[('1', 2.2125680933852125),
 ('3', 2.086708860759492),
 ('2', 1.8712714776632287),
 ('5', 2.0215102974828376),
 ('4', 2.131322620519159),
 ('7', 2.1995274102079394),
 ('6', 2.1216275659824033)]
[('1', 2.854095951997802),
 ('3', 2.4283288725307774),
 ('5', 2.160650912181427),
 ('7', 2.274957936773339),
 ('2', 1.9994917133778995),
 ('4', 2.4421914323679315),
 ('6', 2.131348716430275)]
[('1', 2.2125680933852125),
 ('3', 2.086708860759492),
 ('2', 1.8712714776632287),
 ('5', 2.0215102974828376),
 ('4', 2.131322620519159),
 ('7', 2.1995274102079394),
 ('6', 2.1216275659824033),
 ('1', 2.854095951997802),
 ('3', 2.4283288725307774),
 ('5', 2.160650912181427),
 ('7', 2.274957936773339),
 ('2', 1.9994917133778995),
 ('4', 2.4421914323679315),
 ('6', 2.131348716430275)]
20/03/10 22:25:12 INFO org.spark_project.jetty.server.AbstractConnector:
  Stopped Spark@1a986d85{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}

Job output is complete
```

Average trip time by hour of the day (final.py)

```
import json
import pyspark
import pprint
from operator import add
import math

sc = pyspark.SparkContext()

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
```

```

project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output_f'.format(bucket)

conf = {
    # Input Parameters.
    'mapred.bq.project.id': project,
    'mapred.bq.gcs.bucket': bucket,
    'mapred.bq.temp.gcs.path': input_directory,
    'mapred.bq.input.project.id': 'final-project-269604',
    'mapred.bq.input.dataset.id': 'taxi',
    'mapred.bq.input.table.id': 'trip2',
}

table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf=conf
)

vals = table_data.values()
vals = vals.map(lambda line: json.loads(line))
vals_hour = vals.map(lambda x: (str(x['hour_pickup']), float(x['time_sec'])))
sum_time = vals_hour.reduceByKey(add)
count_time = vals_hour.countByKey()
hour_avg = sum_time.map(lambda x: (x[0], x[1]/count_time[x[0]]))
hour_avg_min = hour_avg.map(lambda x: (x[0], float(x[1])/60))

pprint.pprint(hour_avg_min.collect())

hour_avg_min.saveAsTextFile(output_directory)

input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)

```

```

20/03/13 20:15:08 INFO org.spark_project.jetty.util.log: Logging initialized @3617ms
20/03/13 20:15:08 INFO org.spark_project.jetty.server.Server: jetty-9.3.z-SNAPSHOT, build timestamp: unknown, git hash: unknown
20/03/13 20:15:08 INFO org.spark_project.jetty.server.Server: Started @3752ms
20/03/13 20:15:08 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@6f86d094[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
20/03/13 20:15:08 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair Scheduler configuration file not found so jobs will be scheduled in FIFO order.
To use fair scheduling, configure pools in fairscheduler.xml or set spark.scheduler.allocation.file to a file that contains the configuration.
20/03/13 20:15:09 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at cluster-925d-m/10.138.0.18:8032
20/03/13 20:15:09 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at cluster-925d-m/10.138.0.18:10200
20/03/13 20:15:12 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1584120636332_0006
class com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.DefaultPlatform: cannot cast result of calling
'com.google.cloud.hadoop.repackaged.gcs.com.google.common.flogger.backend.log4j.Log4jBackendFactory#getInstance' to
'com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.BackendFactory':
java.lang.ClassCastException: Cannot cast com.google.cloud.hadoop.repackaged.gcs.com.google.common.flogger.backend.log4j.Log4jBackendFactory to
com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.BackendFactory
20/03/13 20:15:18 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: BigQuery connector version hadoop2-1.1.1
20/03/13 20:15:19 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from default credential.
20/03/13 20:15:19 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from given credential.
20/03/13 20:15:19 INFO com.google.cloud.hadoop.io.bigquery.BigQueryConfiguration: Using working path: 'gs://taxi-final/hadoop/tmp/bigquery/pyspark_input'
20/03/13 20:16:05 INFO com.google.cloud.hadoop.io.bigquery.UnshardedExportToCloudStorage: Setting FileInputFormat's inputPath to 'gs://taxi-final/hadoop/tmp/bigquery/pyspark_input'
20/03/13 20:16:05 INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat: Total input files to process : 18
[('0', 13.116088302606041),
 ('8', 13.362108856591925),
 ('2', 12.332166335541071),
 ('4', 13.411120474052414),
 ('6', 10.715828169962315),
 ('20', 12.633682481180138),
 ('21', 12.438711716165274),
 ('22', 12.680503097552416),
 ('23', 13.308431712401264),
 ('1', 12.550689087173204),
 ('9', 13.512928708257583),
 ('3', 13.353709769199515),
 ('5', 12.152232766805971),
 ('7', 11.859358037489168),

```

```
(('11', 13.80971135174218),
 ('10', 13.406249326041582),
 ('13', 14.334573157420524),
 ('12', 14.102550077113754),
 ('15', 15.319806813196555),
 ('14', 15.195048194374161),
 ('17', 14.984775430967824),
 ('16', 15.107008028556073),
 ('19', 13.19534124521038),
 ('18', 14.18719243574185)])
20/03/13 20:37:59 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark06f86d094{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}

Job output is complete
```

## Average tip amount by month for cash and card (tip.py)

```
import json
import pyspark
import pprint
from operator import add
import math

sc = pyspark.SparkContext()

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output_f2'.format(bucket)

conf = {
    # Input Parameters.
    'mapred.bq.project.id': project,
    'mapred.bq.gcs.bucket': bucket,
    'mapred.bq.temp.gcs.path': input_directory,
    'mapred.bq.input.project.id': 'final-project-269604',
    'mapred.bq.input.dataset.id': 'taxi',
    'mapred.bq.input.table.id': 'fare2',
}

table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf=conf
)

vals = table_data.values()
vals = vals.map(lambda line: json.loads(line))
vals_crd = vals.filter(lambda x: str(x['payment_type'])=='CRD').map(lambda x: (str(x['month_pickup']),
float(x['tip_amount'])))

sum_crd = vals_crd.reduceByKey(add)
count_crd = vals_crd.countByKey()
avg_crd = sum_crd.map(lambda x: (x[0], float(x[1]/count_crd[x[0]])))
max_crd = vals_crd.max()
min_crd = vals_crd.min()

vals_csh = vals.filter(lambda x: str(x['payment_type'])=='CSH').map(lambda x: (str(x['month_pickup']),
float(x['tip_amount'])))

sum_csh = vals_csh.reduceByKey(add)
```

```

count_csh = vals_csh.countByKey()
avg_csh = sum_csh.map(lambda x: (x[0], float(x[1]/count_csh[x[0]])))
max_csh = vals_csh.max()
min_csh = vals_csh.min()

pprint.pprint(avg_crd.collect())

pprint.pprint(avg_csh.collect())

pprint.pprint([max_crd, min_crd, max_csh, min_csh])

crdcsh = avg_crd.union(avg_csh)

crdcsh.saveAsTextFile(output_directory)

input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)

```

```

20/03/13 20:54:18 INFO org.spark_project.jetty.util.log: Logging initialized @3488ms
20/03/13 20:54:18 INFO org.spark_project.jetty.server.Server: jetty-9.3.z-SNAPSHOT, build timestamp: unknown, git hash: unknown
20/03/13 20:54:18 INFO org.spark_project.jetty.server.Server: Started @3612ms
20/03/13 20:54:18 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@463d3822{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
20/03/13 20:54:18 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair Scheduler configuration file not found so jobs will be scheduled in FIFO order.
To use fair scheduling, configure pools in fairscheduler.xml or set spark.scheduler.allocation.file to a file that contains the configuration.
20/03/13 20:54:19 INFO org.apache.hadoop.yarn.client.RMPProxy: Connecting to ResourceManager at cluster-925d-m/10.138.0.18:8032
20/03/13 20:54:19 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at cluster-925d-m/10.138.0.18:10200
20/03/13 20:54:22 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1584120636332_0007
class com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.DefaultPlatform: cannot cast result of calling
'com.google.cloud.hadoop.repackaged.gcs.com.google.common.flogger.backend.log4j.Log4jBackendFactory#getInstance' to
'com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.BackendFactory':
java.lang.ClassCastException: Cannot cast com.google.cloud.hadoop.repackaged.gcs.com.google.common.flogger.backend.log4j.Log4jBackendFactory to
com.google.cloud.hadoop.repackaged.bigquery.com.google.common.flogger.backend.system.BackendFactory

20/03/13 20:54:29 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: BigQuery connector version hadoop2-1.1.1
20/03/13 20:54:29 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from default credential.
20/03/13 20:54:29 INFO com.google.cloud.hadoop.io.bigquery.BigQueryFactory: Creating BigQuery from given credential.
20/03/13 20:54:29 INFO com.google.cloud.hadoop.io.bigquery.BigQueryConfiguration: Using working path: 'gs://taxi-final/hadoop/tmp/bigquery/pyspark_input'
20/03/13 20:56:09 INFO com.google.cloud.hadoop.io.bigquery.UnshardedExportToCloudStorage: Setting FileInputFormat's inputPath to 'gs://taxi-final/hadoop/tmp/bigquery/pyspark_input'
20/03/13 20:56:09 INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat: Total input files to process : 18
[('8', 2.5385533208419697),
 ('2', 2.4085676038819046),
 ('4', 2.503969905514294),
 ('6', 2.5631580818483655),
 ('9', 2.586271805970852),
 ('1', 2.415242544916568),
 ('3', 2.4627967892529035),
 ('5', 2.5588796453961717),
 ('7', 2.502943442658389),
 ('11', 2.5544210586648166),
 ('10', 2.593016324273577),
 ('12', 2.6356888474337397)]
[('8', 0.00048356422869125953),
 ('2', 0.0005439561235407734),
 ('4', 0.00036776090653769),
 ('6', 0.0006276523843120803),
 ('9', 0.00043088520125064207),
 ('1', 0.000720563452342532),
 ('3', 0.0012023247603891066),
 ('5', 0.00031289659188713176),
 ('7', 0.0005649380484220094),
 ('11', 0.00038384538824000173),
 ('10', 0.000305059466221933),
 ('12', 0.00034430176777324123)]
[('9', 200.0), ('1', 0.0), ('9', 121.5), ('1', 0.0)]
20/03/13 22:27:22 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@463d3822{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}

Job output is complete

```

## Links to data sources

Trip data: <http://academictorrents.com/details/6c594866904494b06aae51ad97ec7f985059b135>

Fare data: <http://academictorrents.com/details/107a7d997f331ef4820cf5f7f654516e1704dccf>

Landmark data: [https://en.wikipedia.org/wiki/List\\_of\\_National\\_Historic\\_Landmarks\\_in\\_New\\_York\\_City](https://en.wikipedia.org/wiki/List_of_National_Historic_Landmarks_in_New_York_City)