Explore and create ML datasets

In this notebook, we will explore data corresponding to taxi rides in New York City to build a Machine Learning model in support of a fare-estimation tool. The idea is to suggest a likely fare to taxi riders so that they are not surprised, and so that they can protest if the charge is much higher than expected.

Let's start off with the Python imports that we need.

In [1]:

```
from google.cloud import bigquery
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import shutil
```

Extract sample data from BigQuery

The dataset that we will use is a BigQuery public dataset (https://bigquery.cloud.google.com/table/nyc-tlc:yellow.trips). Click on the link, and look at the column names. Switch to the Details tab to verify that the number of records is one billion, and then switch to the Preview tab to look at a few rows.

Let's write a SQL query to pick up interesting fields from the dataset.

In [2]:

```
sql = """
  SELECT
    pickup_datetime, pickup_longitude, pickup_latitude, dropoff_longitude,
    dropoff_latitude, passenger_count, trip_distance, tolls_amount,
    fare_amount, total_amount
  FROM `nyc-tlc.yellow.trips`
  LIMIT 10
"""
```

In [3]:

```
client = bigquery.Client()
trips = client.query(sql).to_dataframe()
trips
```

Out[3]:

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	trip_distance	tolls_amount	f
0	2010-03-04 00:35:16+00:00	-74.035201	40.721548	-74.035201	40.721548	1	0.0	0.0	
1	2010-03-15 17:18:34+00:00	0.000000	0.000000	0.000000	0.000000	1	0.0	0.0	
2	2015-03-18 01:07:02+00:00	0.000000	0.000000	0.000000	0.000000	5	0.0	0.0	
3	2015-03-09 18:24:03+00:00	-73.937248	40.758202	-73.937263	40.758190	1	0.0	0.0	
4	2010-03-06 06:33:41+00:00	-73.785514	40.645400	-73.784564	40.648681	2	4.1	0.0	
5	2013-08-07 00:42:45+00:00	-74.025817	40.763044	-74.046752	40.783240	1	4.8	0.0	
6	2015-04-26 02:56:37+00:00	-73.987656	40.771656	-73.987556	40.771751	1	0.0	0.0	
7	2015-04-29 18:45:03+00:00	0.000000	0.000000	0.000000	0.000000	1	1.0	0.0	
8	2010-03-11 21:24:48+00:00	-74.571511	40.910800	-74.628928	40.964321	1	68.4	0.0	
9	2013-08-24 01:58:23+00:00	-73.972171	40.759439	0.000000	0.000000	4	0.0	0.0	

Let's increase the number of records so that we can do some neat graphs. There is no guarantee about the order in which records are returned, and so no guarantee about which records get returned if we simply increase the LIMIT. To properly sample the dataset, let's use the HASH of the pickup time and return 1 in 100,000 records -- because there are 1 billion records in the data, we should get back approximately 10,000 records if we do this.

In [5]:

```
sql = """
  SELECT
    pickup_datetime,
    pickup_longitude, pickup_latitude,
    dropoff_longitude, dropoff_latitude,
    passenger_count,
    trip_distance,
    tolls_amount,
    fare_amount,
    total_amount
  FROM
    `nyc-tlc.yellow.trips`
  WHERE
    ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)), 100000)) = 1
"""
```

In [6]:

```
trips = client.query(sql).to_dataframe()
trips[:10]
```

Out[6]:

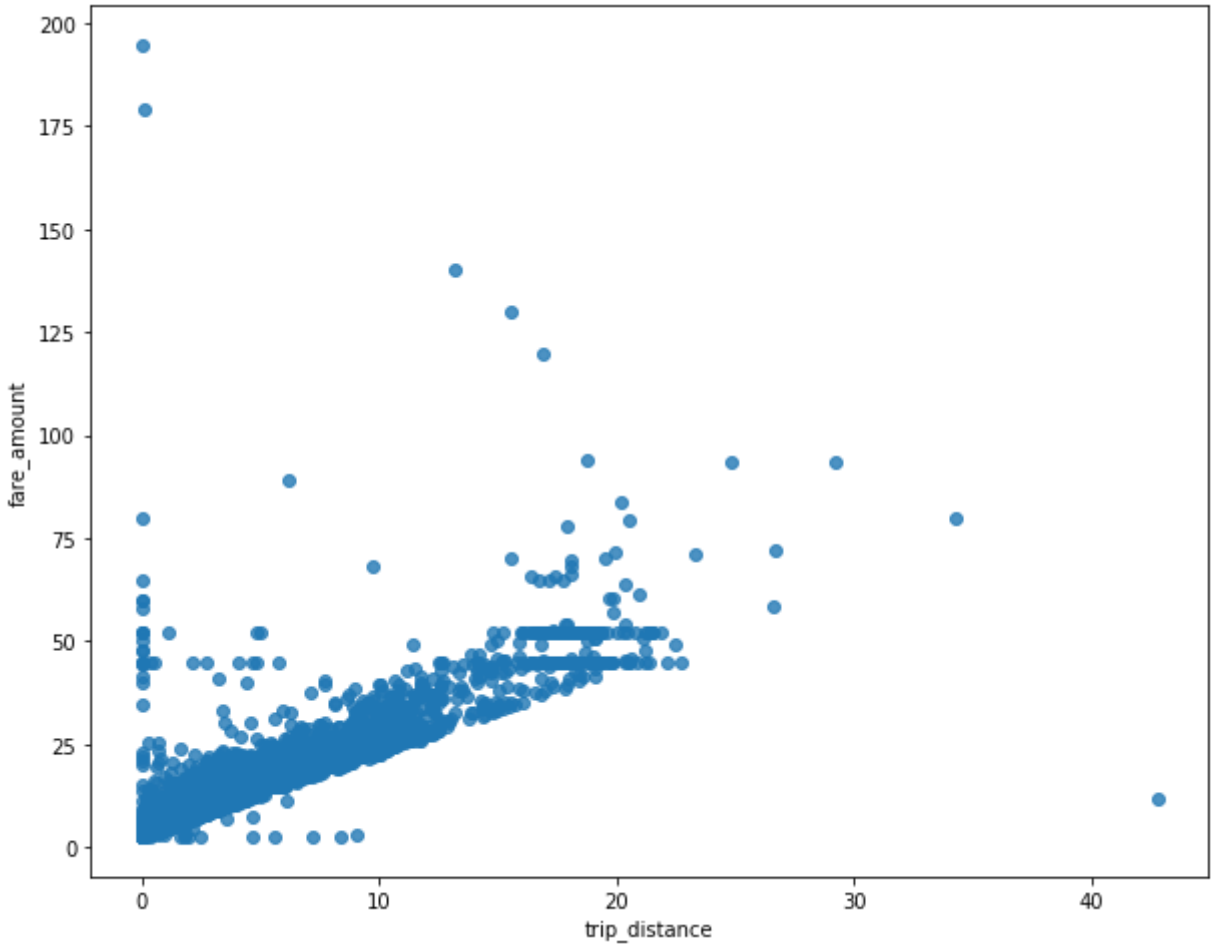
	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	trip_distance	tolls_amount	f
0	2014-12-08 21:50:00+00:00	-73.994802	40.720612	-73.949125	40.668893	1	5.33	0.00	
1	2012-03-04 00:57:00+00:00	-74.005625	40.734517	-73.952492	40.725197	1	7.33	0.00	
2	2013-11-26 14:33:11+00:00	-73.983840	40.728959	-73.972363	40.762258	1	2.80	0.00	
3	2012-05-05 22:46:05+00:00	-74.009790	40.712483	-73.959293	40.768908	1	5.20	0.00	
4	2010-12-21 13:08:00+00:00	-73.982422	40.739847	-73.981658	40.768732	2	2.64	0.00	
5	2013-12-09 15:03:00+00:00	-73.990950	40.749772	-73.870807	40.774070	1	9.44	5.33	
6	2014-07-25 20:07:41+00:00	-73.964157	40.754548	-74.002540	40.760683	1	2.60	0.00	
7	2014-05-17 15:15:00+00:00	-73.999550	40.760600	-73.999650	40.725220	1	5.57	0.00	
8	2014-10-06 15:16:00+00:00	-73.980130	40.760910	-73.861730	40.768330	2	11.47	5.33	
9	2014-12-08 21:50:00+00:00	-73.870867	40.773782	-74.003297	40.708215	2	11.81	0.00	

Exploring data

Let's explore this dataset and clean it up as necessary. We'll use the Python Seaborn package to visualize graphs and Pandas to do the slicing and filtering.

In [7]:

```
ax = sns.regplot(x="trip_distance", y="fare_amount", fit_reg=False, ci=None, truncate=True, data=trips)
ax.figure.set_size_inches(10, 8)
```



Hmm ... do you see something wrong with the data that needs addressing?

It appears that we have a lot of invalid data that is being coded as zero distance and some fare amounts that are definitely illegitimate. Let's remove them from our analysis. We can do this by modifying the BigQuery query to keep only trips longer than zero miles and fare amounts that are at least the minimum cab fare ($2.50).

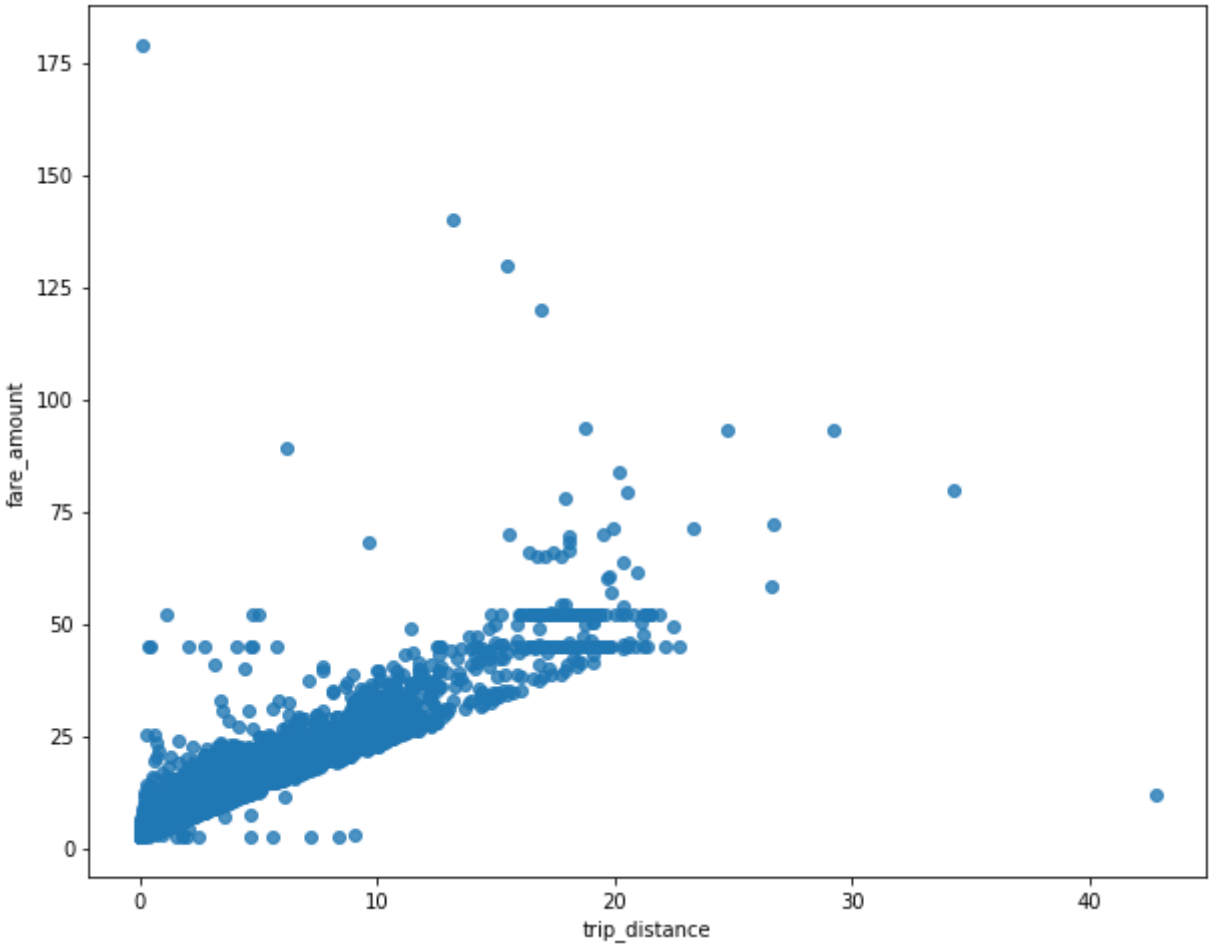
Note the extra WHERE clauses.

In [8]:

```
sql = """
  SELECT
    pickup_datetime,
    pickup_longitude, pickup_latitude,
    dropoff_longitude, dropoff_latitude,
    passenger_count,
    trip_distance,
    tolls_amount,
    fare_amount,
    total_amount
  FROM
    `nyc-tlc.yellow.trips`
  WHERE
    ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)), 100000)) = 1
    AND trip_distance > 0 AND fare_amount >= 2.5
"""
```

In [9]:

```
trips = client.query(sql).to_dataframe()
ax = sns.regplot(x="trip_distance", y="fare_amount", fit_reg=False, ci=None, truncate=True, data=trips)
ax.figure.set_size_inches(10, 8)
```


fare_amount
trip_distance
0
25
50
75
100
125
150
175
0
10
20
30
40

What's up with the streaks at \$45 and \$50? Those are fixed-amount rides from JFK and La Guardia airports into anywhere in Manhattan, i.e. to be expected. Let's list the data to make sure the values look reasonable.

Let's examine whether the toll amount is captured in the total amount.

In [26]:

```
#trips.head()
#trips.describe()
print ("size of the dataset before:", trips.shape)
print ("size of the dataset with only tolls_amount:", trips[trips["tolls_amount"] >0].shape)
```

```
size of the dataset before: (10716, 10)
size of the dataset with only tolls_amount: (453, 10)
```

In [27]:

```
tollrides = trips[trips['tolls_amount'] > 0]
tollrides[tollrides['pickup_datetime'] == '2010-04-29 12:28:00']
```

Out[27]:

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	trip_distance	tolls_amoun
333	2010-04-29 12:28:00+00:00	-73.991303	40.749965	-73.714585	40.745767	2	18.68	4.5
439	2010-04-29 12:28:00+00:00	-73.865723	40.770543	-73.984790	40.758760	1	12.32	5.5
678	2010-04-29 12:28:00+00:00	-74.008322	40.735337	-74.177383	40.695083	1	15.86	10.0
808	2010-04-29 12:28:00+00:00	-74.006398	40.738450	-73.872652	40.774357	2	10.67	4.5
868	2010-04-29 12:28:00+00:00	-73.969748	40.759790	-73.872892	40.774297	1	10.15	4.5
1354	2010-04-29 12:28:00+00:00	-73.862715	40.768987	-74.007195	40.707480	1	13.17	4.5
1396	2010-04-29 12:28:00+00:00	-73.950105	40.827105	-73.861490	40.768172	1	9.06	4.5
1425	2010-04-29 12:28:00+00:00	-73.870773	40.773753	-73.984963	40.757590	1	10.97	4.5
1575	2010-04-29 12:28:00+00:00	-73.870928	40.773747	-73.983638	40.752948	1	8.63	4.5
6756	2010-04-29 12:28:00+00:00	-73.789942	40.646943	-73.974362	40.756418	2	16.84	4.5

Looking a few samples above, it should be clear that the total amount reflects fare amount, toll and tip somewhat arbitrarily -- this is because when customers pay cash, the tip is not known. So, we'll use the sum of fare_amount + tolls_amount as what needs to be predicted. Tips are discretionary and do not have to be included in our fare estimation tool.

Let's also look at the distribution of values within the columns.

In [28]:

```
trips.describe()
```

Out[28]:

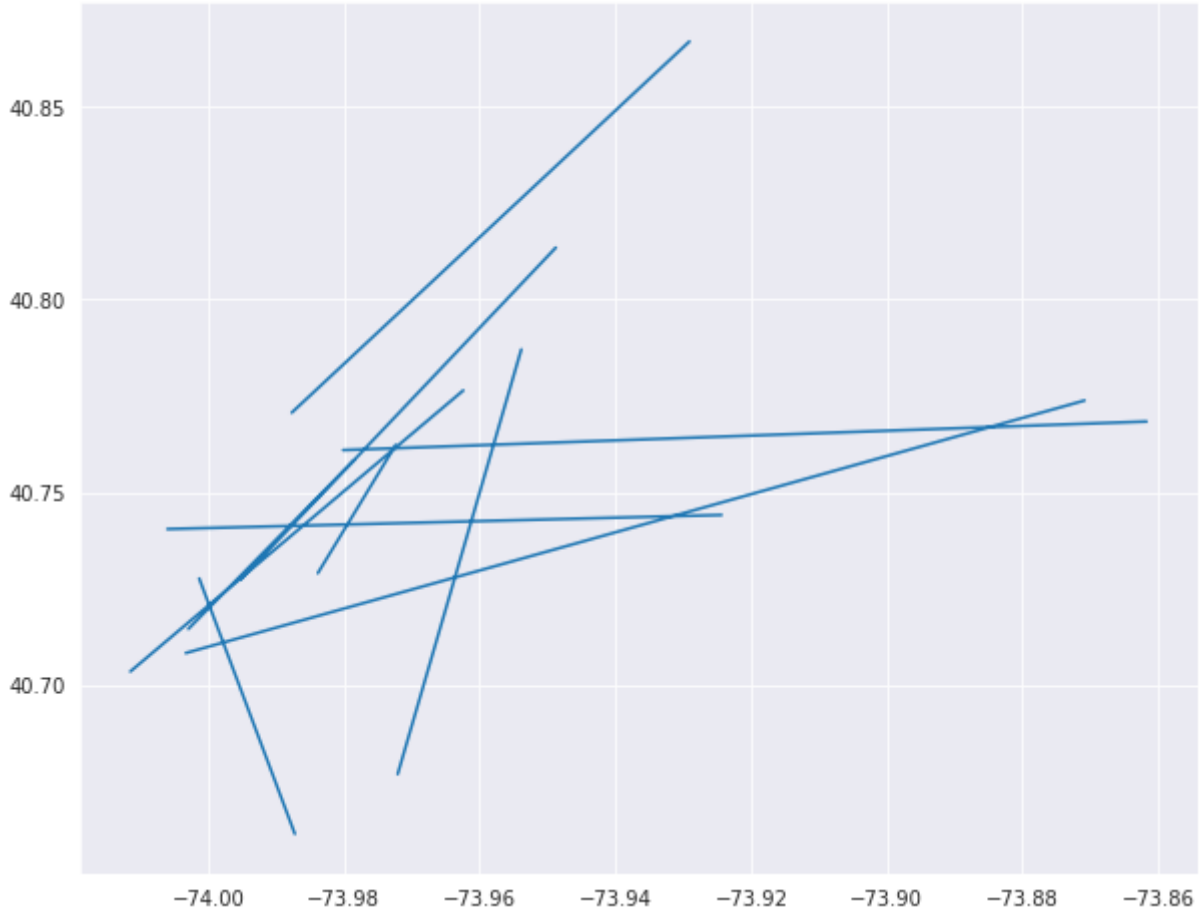
	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	trip_distance	tolls_amount	fare_amount
count	10716.000000	10716.000000	10716.000000	10716.000000	10716.000000	10716.000000	10716.000000	10716.000000
mean	-72.602192	40.002372	-72.594838	40.002052	1.650056	2.856395	0.226428	11.109446
std	9.982373	5.474670	10.004324	5.474648	1.283577	3.322024	1.135934	9.137710
min	-74.258183	0.000000	-74.260472	0.000000	0.000000	0.010000	0.000000	2.500000
25%	-73.992153	40.735936	-73.991566	40.734310	1.000000	1.040000	0.000000	6.000000
50%	-73.981851	40.753264	-73.980373	40.752956	1.000000	1.770000	0.000000	8.500000
75%	-73.967400	40.767340	-73.964142	40.767510	2.000000	3.160000	0.000000	12.500000
max	0.000000	41.366138	0.000000	41.366138	6.000000	42.800000	16.000000	179.000000

Hmm ... The min, max of longitude look strange.

Finally, let's actually look at the start and end of a few of the trips.

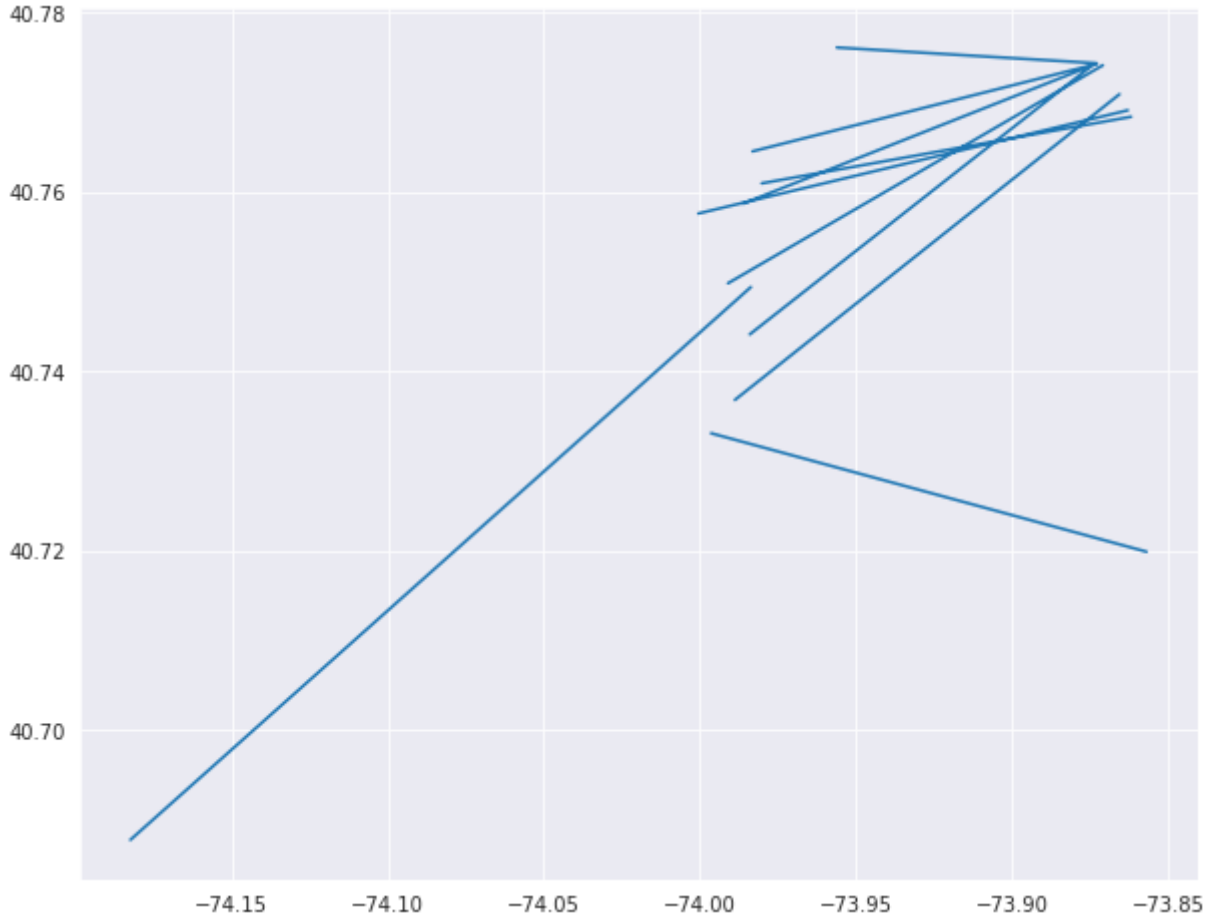
In [29]:

```
def showrides(df, numlines):
  lats = []
  lons = []
  for iter, row in df[:numlines].iterrows():
    lons.append(row['pickup_longitude'])
    lons.append(row['dropoff_longitude'])
    lons.append(None)
    lats.append(row['pickup_latitude'])
    lats.append(row['dropoff_latitude'])
    lats.append(None)

  sns.set_style("darkgrid")
  plt.figure(figsize=(10,8))
  plt.plot(lons, lats)

showrides(trips, 10)
```


40.85
40.80
40.75
40.70
−74.00
−73.98
−73.96
−73.94
−73.92
−73.90
−73.88
−73.86

In [30]:

```
showrides(tollrides, 10)
```



As you'd expect, rides that involve a toll are longer than the typical ride.

Quality control and other preprocessing

We need to some clean-up of the data:

1. New York city longitudes are around -74 and latitudes are around 41.
2. We shouldn't have zero passengers.
3. Clean up the total_amount column to reflect only fare_amount and tolls_amount, and then remove those two columns.
4. Before the ride starts, we'll know the pickup and dropoff locations, but not the trip distance (that depends on the route taken), so remove it from the ML dataset
5. Discard the timestamp

We could do preprocessing in BigQuery, similar to how we removed the zero-distance rides, but just to show you another option, let's do this in Python. In production, we'll have to carry out the same preprocessing on the real-time input data.

This sort of preprocessing of input data is quite common in ML, especially if the quality-control is dynamic.

In [31]:

```
def preprocess(trips_in):
  trips = trips_in.copy(deep=True)
  trips.fare_amount = trips.fare_amount + trips.tolls_amount
  del trips['tolls_amount']
  del trips['total_amount']
  del trips['trip_distance']
  del trips['pickup_datetime']
  qc = np.all([\
              trips['pickup_longitude'] > -78, \
              trips['pickup_longitude'] < -70, \
              trips['dropoff_longitude'] > -78, \
              trips['dropoff_longitude'] < -70, \
              trips['pickup_latitude'] > 37, \
              trips['pickup_latitude'] < 45, \
              trips['dropoff_latitude'] > 37, \
              trips['dropoff_latitude'] < 45, \
              trips['passenger_count'] > 0,
             ], axis=0)
  return trips[qc]

tripsqc = preprocess(trips)
tripsqc.describe()
```

Out[31]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	fare_amount
count	10476.000000	10476.000000	10476.000000	10476.000000	10476.000000	10476.000000
mean	-73.975206	40.751526	-73.974373	40.751199	1.653303	11.349003
std	0.038547	0.029187	0.039086	0.033147	1.278827	9.878630
min	-74.258183	40.452290	-74.260472	40.417750	1.000000	2.500000
25%	-73.992336	40.737600	-73.991739	40.735904	1.000000	6.000000
50%	-73.982090	40.754020	-73.980780	40.753597	1.000000	8.500000
75%	-73.968517	40.767774	-73.965851	40.767921	2.000000	12.500000
max	-73.137393	41.366138	-73.137393	41.366138	6.000000	179.000000

The quality control has removed about 300 rows (11400 - 11101) or about 3% of the data. This seems reasonable.

Let's move on to creating the ML datasets.

Create ML datasets

Let's split the QCed data randomly into training, validation and test sets.

In [32]:

```
shuffled = tripsqc.sample(frac=1)
trainsize = int(len(shuffled['fare_amount']) * 0.70)
validsize = int(len(shuffled['fare_amount']) * 0.15)

df_train = shuffled.iloc[:trainsize, :]
df_valid = shuffled.iloc[trainsize:(trainsize+validsize), :]
df_test = shuffled.iloc[(trainsize+validsize):, :]
```

In [33]:

```
df_train.describe()
```

Out[33]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	fare_amount
count	7333.000000	7333.000000	7333.000000	7333.000000	7333.000000	7333.000000
mean	-73.975244	40.751280	-73.974289	40.751282	1.652802	11.273657
std	0.038193	0.029379	0.038731	0.033316	1.277082	9.909542
min	-74.187541	40.452290	-74.187541	40.417750	1.000000	2.500000
25%	-73.992457	40.737217	-73.991626	40.735775	1.000000	6.000000
50%	-73.982149	40.753607	-73.980582	40.753595	1.000000	8.500000
75%	-73.968482	40.767537	-73.965776	40.768177	2.000000	12.500000
max	-73.137393	41.366138	-73.137393	41.366138	6.000000	179.000000

In [34]:

```
df_valid.describe()
```

Out[34]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	fare_amount
count	1571.000000	1571.000000	1571.000000	1571.000000	1571.000000	1571.000000
mean	-73.975696	40.752144	-73.973965	40.750974	1.667728	11.419548
std	0.037428	0.025896	0.036884	0.031326	1.300307	9.316296
min	-74.258183	40.641319	-74.260472	40.569997	1.000000	2.500000
25%	-73.992378	40.738817	-73.991887	40.736654	1.000000	6.100000
50%	-73.981949	40.754570	-73.980958	40.754488	1.000000	8.500000
75%	-73.968771	40.768057	-73.964320	40.767248	2.000000	12.500000
max	-73.330783	40.847460	-73.711521	40.879257	6.000000	80.250000

In [35]:

```
df_test.describe()
```

Out[35]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	fare_amount
count	1572.000000	1572.000000	1572.000000	1572.000000	1572.000000	1572.000000
mean	-73.974540	40.752054	-73.975177	40.751037	1.641221	11.629975
std	0.041220	0.031319	0.042718	0.034131	1.265929	10.272940
min	-74.116582	40.633522	-74.181608	40.597695	1.000000	2.500000
25%	-73.991641	40.737664	-73.991959	40.735843	1.000000	6.100000
50%	-73.981907	40.754516	-73.981365	40.753043	1.000000	8.500000
75%	-73.968650	40.768689	-73.967764	40.767212	2.000000	12.900000
max	-73.137393	41.366138	-73.137393	41.366138	6.000000	93.750000

Let's write out the three dataframes to appropriately named csv files. We can use these csv files for local training (recall that these files represent only 1/100,000 of the full dataset) until we get to point of using Dataflow and Cloud ML.

In [36]:

```
def to_csv(df, filename):
  outdf = df.copy(deep=False)
  outdf.loc[:, 'key'] = np.arange(0, len(outdf)) # rownumber as key
  # reorder columns so that target is first column
  cols = outdf.columns.tolist()
  cols.remove('fare_amount')
  cols.insert(0, 'fare_amount')
  print (cols)  # new order of columns
  outdf = outdf[cols]
  outdf.to_csv(filename, header=False, index_label=False, index=False)

to_csv(df_train, 'taxi-train.csv')
to_csv(df_valid, 'taxi-valid.csv')
to_csv(df_test, 'taxi-test.csv')
```

```
['fare_amount', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'pas
senger_count', 'key']
['fare_amount', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'pas
senger_count', 'key']
['fare_amount', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'pas
senger_count', 'key']
```

In [37]:

```
!head -10 taxi-valid.csv
```

```
3.7,-73.999574,40.73405,-74.004335,40.734673,1,0
14.9,-74.008213,40.707913,-73.962378,40.719655,1,1
4.5,-73.968298,40.762442,-73.963288,40.765837,1,2
14.0,-73.98870849609375,40.73094177246094,-73.97443389892578,40.69493865966797,6,3
4.5,-73.96257,40.772997,-73.968792,40.764408,1,4
37.0,-73.87164306640625,40.77409362792969,-73.7763442993164,40.64540100097656,1,5
6.0,-73.982061,40.743172,-73.993733,40.735817,1,6
6.5,-73.989645,40.762371,-74.005482,40.7401,2,7
12.1,-73.988779,40.774127,-73.947591,40.776189,1,8
10.5,-73.976268,40.74424,-74.005255,40.729097,1,9
```

Verify that datasets exist

In [38]:

```
!ls -l *.csv
```

```
-rw-r--r-- 1 jupyter jupyter  85667 Jan 17 19:51 taxi-test.csv
-rw-r--r-- 1 jupyter jupyter 403297 Jan 17 19:51 taxi-train.csv
-rw-r--r-- 1 jupyter jupyter  85371 Jan 17 19:51 taxi-valid.csv
```

We have 3 .csv files corresponding to train, valid, test. The ratio of file-sizes correspond to our split of the data.

In [39]:

```
%%bash
head taxi-train.csv
```

```
11.3,-73.870875,40.77374,-73.858582,40.736037,1,0
8.5,-74.009828,40.711247,-74.005105,40.732552,1,1
3.5,-73.980872,40.754292,-73.97834,40.757527,1,2
16.5,-73.96735,40.756807,-74.01306,40.714675,1,3
11.3,-73.982523,40.757095,-73.947024,40.776801,1,4
15.7,-74.010788,40.709463,-73.968095,40.754745,5,5
14.9,-73.956615,40.76695,-73.992755,40.748563,5,6
6.1,-73.9678,40.765859,-73.952213,40.783851,1,7
5.7,-73.964115,40.773741,-73.971997,40.765649,2,8
79.0,-73.974253,40.786518,-74.177625,40.695465,1,9
```

Looks good! We now have our ML datasets and are ready to train ML models, validate them and evaluate them.

Benchmark

Before we start building complex ML models, it is a good idea to come up with a very simple model and use that as a benchmark.

My model is going to be to simply divide the mean fare_amount by the mean trip_distance to come up with a rate and use that to predict. Let's compute the RMSE of such a model.

For the Haversine formula Here is the wikipedia page https://en.wikipedia.org/wiki/Haversine_formula (https://en.wikipedia.org/wiki/Haversine_formula).

In [42]:

```
def distance_between(lat1, lon1, lat2, lon2):
  # haversine formula to compute distance "as the crow flies".  Taxis can't fly of course.
  dist = np.degrees(np.arccos(np.minimum(1,np.sin(np.radians(lat1)) * np.sin(np.radians(lat2)) + np.cos(np.radi
ans(lat1)) * np.cos(np.radians(lat2)) * np.cos(np.radians(lon2 - lon1))))) * 60 * 1.515 * 1.609344
  return dist

def estimate_distance(df):
  return distance_between(df['pickuplat'], df['pickuplon'], df['dropofflat'], df['dropofflon'])

def compute_rmse(actual, predicted):
  return np.sqrt(np.mean((actual-predicted)**2))

def print_rmse(df, rate, name):
  print ("{1} RMSE = {0}".format(compute_rmse(df['fare_amount'], rate*estimate_distance(df)), name))

FEATURES = ['pickuplon','pickuplat','dropofflon','dropofflat','passengers']
TARGET = 'fare_amount'
columns = list([TARGET])
columns.extend(FEATURES) # in CSV, target is the first column, after the features
columns.append('key')
df_train = pd.read_csv('taxi-train.csv', header=None, names=columns)
df_valid = pd.read_csv('taxi-valid.csv', header=None, names=columns)
df_test = pd.read_csv('taxi-test.csv', header=None, names=columns)
rate = df_train['fare_amount'].mean() / estimate_distance(df_train).mean()
#print ("Rate = ${0}/km".format(rate))
print ("Rate = %.2f km" %(rate))
print_rmse(df_train, rate, 'Train')
print_rmse(df_valid, rate, 'Valid')
print_rmse(df_test, rate, 'Test')
```

```
Rate = 2.60 km
Train RMSE = 7.905437258302899
Valid RMSE = 7.449881477209603
Test RMSE = 6.111828389723749
```

Benchmark on same dataset

The RMSE depends on the dataset, and for comparison, we have to evaluate on the same dataset each time. We'll use this query in later labs:

In [43]:

```
def create_query(phase, EVERY_N):
  """
  phase: 1=train 2=valid
  """
  base_query = """
SELECT
  (tolls_amount + fare_amount) AS fare_amount,
  CONCAT(CAST(pickup_datetime AS STRING), CAST(pickup_longitude AS STRING), CAST(pickup_latitude AS STRING), CA
ST(dropoff_latitude AS STRING), CAST(dropoff_longitude AS STRING)) AS key,
  EXTRACT(DAYOFWEEK FROM pickup_datetime)*1.0 AS dayofweek,
  EXTRACT(HOUR FROM pickup_datetime)*1.0 AS hourofday,
  pickup_longitude AS pickuplon,
  pickup_latitude AS pickuplat,
  dropoff_longitude AS dropofflon,
  dropoff_latitude AS dropofflat,
  passenger_count*1.0 AS passengers
FROM
  `nyc-tlc.yellow.trips`
WHERE
  trip_distance > 0
  AND fare_amount >= 2.5
  AND pickup_longitude > -78
  AND pickup_longitude < -70
  AND dropoff_longitude > -78
  AND dropoff_longitude < -70
  AND pickup_latitude > 37
  AND pickup_latitude < 45
  AND dropoff_latitude > 37
  AND dropoff_latitude < 45
  AND passenger_count > 0
  """

  if EVERY_N == None:
    if phase < 2:
      # training
      query = "{0} AND ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)), 4)) < 2".format(base_query)
    else:
      query = "{0} AND ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)), 4)) = {1}".format(base_query,
phase)
  else:
```

```
      query = "{0} AND ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)), {1})) = {2}".format(base_quer
y, EVERY_N, phase)

  return query

query = create_query(2, 100000)
df_valid = client.query(query).to_dataframe()
print_rmse(df_valid, 2.56, 'Final Validation Set')
```

```
Final Validation Set RMSE = 7.4158766166380445
```

The simple distance-based rule gives us a RMSE of **$7.42**. We have to beat this, of course, but you will find that simple rules of thumb like this can be surprisingly difficult to beat.

Let's be ambitious, though, and make our goal to build ML models that have a RMSE of less than $6 on the test set.