TaxiDriver

September 6, 2016

A Data analysis report by Giuseppe Di Bernardo date: “September 06, 2016”

1 Exploring the dataset

1.1 Preparing the notebook

In [1]:
```
# magic command to display matplotlib plots inline within the ipython notebook webpage
%matplotlib inline
% config InlineBackend.figure_format='retina'
```

In [2]:
```
# import relevant modules
import os

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

sns.set(font='sans')
```

1.2 Reading the input file

In [3]:
```
# dir paths
data_dir = ("../data/")
csv = "yellow_tripdata_2015-06.csv"
fullcsv = data_dir + csv
os.path.normpath(fullcsv)
# print(fullcsv)
```

Out[3]:
```
'../data/yellow_tripdata_2015-06.csv'
```

The data provided to the NYC Taxi and Limousine Commission (TLC) - by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP) - are stored in `CSV` format, and organized by year and month. In each file, each row represents a single taxi trip.
Let's take a look to the data. To this purpose, we will use `pandas` to do all the big data clean up and preparation. Each row of the `yellow_tripdata_.csv file` represents a trip, and the columns are the attributes for these trips.

In [4]:
```
# Create a pandas dataframe from the location data set.
# Load the location data set and, parse the dates so
# they're no longer strings but now rather Python datetime objects
# this lets us do date and time based operations on the data set
# our data frame
df = pd.read_csv(fullcsv, parse_dates=['tpep_pickup_datetime', 'tpep_dropoff_datetime'])
```

```
In [5]: # uncomment this if you want to get insights of the data types you are dealing with
        # df.info()
```

```
In [6]: # a first glimpse: the first five trips of the file
        df.head()
```

```
Out[6]:    VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
        0         2  2015-06-02 11:19:29   2015-06-02 11:47:52                1
        1         2  2015-06-02 11:19:30   2015-06-02 11:27:56                1
        2         2  2015-06-02 11:19:31   2015-06-02 11:30:30                1
        3         2  2015-06-02 11:19:31   2015-06-02 11:39:02                1
        4         1  2015-06-02 11:19:32   2015-06-02 11:32:49                1

           trip_distance  pickup_longitude  pickup_latitude  RateCodeID  \
        0           1.63        -73.954430        40.764141           1
        1           0.46        -73.971443        40.758942           1
        2           0.87        -73.978111        40.738434           1
        3           2.13        -73.945892        40.773529           1
        4           1.40        -73.979088        40.776772           1

          store_and_fwd_flag  dropoff_longitude  dropoff_latitude  payment_type  \
        0                  N         -73.974754         40.754093             2
        1                  N         -73.978539         40.761909             1
        2                  N         -73.990273         40.745438             1
        3                  N         -73.971527         40.760330             1
        4                  N         -73.982162         40.758999             2

           fare_amount  extra  mta_tax  tip_amount  tolls_amount  \
        0         17.0    0.0      0.5        0.00           0.0
        1          6.5    0.0      0.5        1.00           0.0
        2          8.0    0.0      0.5        2.20           0.0
        3         13.5    0.0      0.5        2.86           0.0
        4          9.5    0.0      0.5        0.00           0.0

           improvement_surcharge  total_amount
        0                    0.3         17.80
        1                    0.3          8.30
        2                    0.3         11.00
        3                    0.3         17.16
        4                    0.3         10.30
```

```
In [7]: databegin = len(df)
        print("We have " +str(databegin)+" trips in New York in June 2015")

        # a double-check
        # df.count(axis=0, level=None, numeric_only=False)
```

```
We have 12324935 trips in New York in June 2015
```

```
In [8]: # check it out if times are converted to datetime objects
        df.tpep_pickup_datetime.head()
        # df['tpep_pickup_datetime'].head()
```

```
Out[8]: 0   2015-06-02 11:19:29
        1   2015-06-02 11:19:30
```

```
2   2015-06-02 11:19:31
3   2015-06-02 11:19:31
4   2015-06-02 11:19:32
Name: tpep_pickup_datetime, dtype: datetime64[ns]
```

```
In [9]: Timedelta = df.tpep_pickup_datetime.iloc[-1] - df.tpep_pickup_datetime.iloc[0]
        print("We have " +str(Timedelta)+" of data observation for trips in New York in June 2015")
```

```
We have 28 days 10:34:53 of data observation for trips in New York in June 2015
```

Trip data looks like this. The file relative to the month of June has about ** 12 million rows **, and each row contains: `vendor id`, `rate code`, `store and forward flag`, `pickup date/time dropoff date/time`, `passenger count`, `trip distance`, and `latitude/longitude` coordinates for the pickup and dropoff locations. The possibilities are endless! I smell a tip analysis coming on :-)

1.3 Exploratory data analysis

The data set is organized as follows:

- `VendorID`: e.g., Verifone Transportation Systems (VTS), or Mobile Knowledge Systems Inc (CMT), implemented as part of the Technology Passenger Enhancements Project.
- `tpep_pickup_datetime`: start time of the trip, `mm-dd-yyyy hh24:mm:ss` EDT.
- `tpep_dropoff_datetime`: end time of the trip, `mm-dd-yyyy hh24:mm:ss` EDT.
- `RateCodeID`: taximeter rate, see NYCT&L description.
- `store_and_fwd_flag`: This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server. Y= store and forward trip, N= not a store and forward trip
- `passenger_count`: number of passengers on the trip, default value is one.
- `trip_distance`: trip distance measured by the taximeter in miles.
- `pickup_longitude` and `pickup_latitude`: GPS coordinates at the start of the trip.
- `dropoff_longitude` and `dropoff_latitude`: GPS coordinates at the end of the trip.
- `payment_type`: A numeric code signifying how the passenger paid for the trip. 1= Credit card, 2= Cash, 3= No charge, 4= Dispute, 5= Unknown, 6= Voided trip
- `fare_amount`: The time-and-distance fare calculated by the meter
- `extra`: Miscellaneous extras and surcharges. Currently, this only includes the `$0.50` and `$1` rush hour and overnight charges.
- `mta_tax`: `$0.50` MTA tax that is automatically triggered based on the metered rate in use.

- `tip_amount`: Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
- `tolls_amount`: Total amount of all tolls paid in trip.
- `improvement_surcharge`: `$0.30` improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in `2015`.

- `total_amount`: The total amount charged to passengers. Does not include cash tips.

A dictionary for the yellow taxi trip records can be found here.

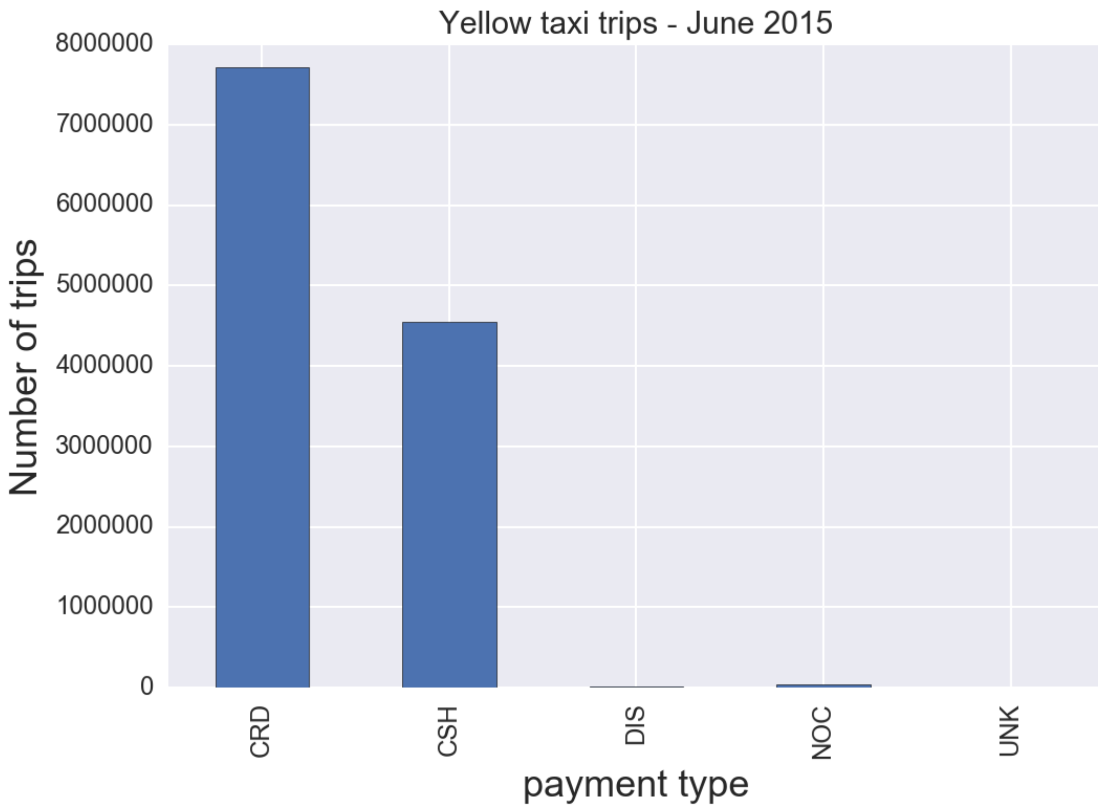
Let's use the `replace` method to modify a subset of values in an object. `replace` provides a simpler and more flexible way to do so.

```
In [10]: # the argument is passed as a dict:
         df.VendorID = df.VendorID.replace({1: 'CMT', 2: 'VFI'})
         df.RateCodeID = df.RateCodeID.replace({1: 'STD', 2: 'JFK', 3: 'NEW', 4: 'NOW', 5: 'NEG', 6: 'GI
         df.payment_type = df.payment_type.replace({1: 'CRD', 2: 'CSH', 3: 'NOC', 4: 'DIS', 5: 'UNK', 6
```

It is convenient to visualize some of these attributes, e.g., the `payment_type`, to get first insights in the distributions of these data values:

```
In [11]: ax = df.groupby(df['payment_type']).size().plot(kind='bar')

         ax.set_xlabel('payment type', fontsize=18)
         ax.set_ylabel('Number of trips', fontsize=18)
         plt.title('Yellow taxi trips - June 2015', fontsize=15)
         ax.tick_params(labelsize=12)
```



We can see that **credit card and cash** are the taxi's main payment types. The rows with the other strange values can be deleted. By doing this (we are going to do the same with the other attributes too) we are adding a bit of bias to the predictions, but those values are so unusual that will hardly affect to the prediction's performance.

```
In [12]: # we are going to drop trips with payment not cash or credit
         # types = ['CRD','CSH']
         # df = df[df.payment_type.isin(types)]
         payment_type = ((df.payment_type == 'CRD') | (df.payment_type == 'CSH'))
```

As the rest of the attributes are numeric, a way to help ourselves is by obtaining a few of statistical values from them.

```
In [13]: df.describe()
```

```
Out[13]:        passenger_count  trip_distance  pickup_longitude  pickup_latitude  \
         count     1.232494e+07   1.232494e+07      1.232494e+07     1.232494e+07
         mean      1.681898e+00   1.182908e+01     -7.291385e+01     4.016687e+01
         std       1.335180e+00   7.678550e+03      8.796762e+00     4.843162e+00
```

```
min      0.000000e+00  0.000000e+00     -7.592333e+02    -6.713696e+01
25%      1.000000e+00  1.010000e+00     -7.399190e+01     4.073614e+01
50%      1.000000e+00  1.750000e+00     -7.398154e+01     4.075323e+01
75%      2.000000e+00  3.230000e+00     -7.396646e+01     4.076793e+01
max      9.000000e+00  1.008332e+07      1.490285e+02     6.970258e+01

       dropoff_longitude  dropoff_latitude  fare_amount         extra  \
count       1.232494e+07      1.232494e+07  1.232494e+07  1.232494e+07
mean       -7.294474e+01      4.018478e+01  1.320408e+01  3.234049e-01
std         8.669562e+00      4.777574e+00  1.060766e+02  4.804153e-01
min        -7.541667e+02     -1.617787e+01 -3.000000e+02 -3.050000e+01
25%        -7.399130e+01      4.073463e+01  6.500000e+00  0.000000e+00
50%        -7.397962e+01      4.075380e+01  9.500000e+00  0.000000e+00
75%        -7.396248e+01      4.076879e+01  1.500000e+01  5.000000e-01
max         1.255356e+02      4.834500e+02  3.354137e+05  6.524200e+02

            mta_tax    tip_amount  tolls_amount  improvement_surcharge  \
count  1.232494e+07  1.232494e+07  1.232494e+07           1.232494e+07
mean   4.976184e-01  1.736538e+00  3.161518e-01           2.997213e-01
std    4.214822e-02  2.637613e+00  1.542573e+00           1.216386e-02
min   -5.000000e-01 -8.000000e+01 -1.400000e+01          -3.000000e-01
25%    5.000000e-01  0.000000e+00  0.000000e+00           3.000000e-01
50%    5.000000e-01  1.160000e+00  0.000000e+00           3.000000e-01
75%    5.000000e-01  2.350000e+00  0.000000e+00           3.000000e-01
max    6.035000e+01  9.809100e+02  9.009700e+02           7.000000e-01

       total_amount
count  1.232494e+07
mean   1.637827e+01
std    1.063828e+02
min   -3.000000e+02
25%    8.760000e+00
50%    1.230000e+01
75%    1.830000e+01
max    3.354145e+05
```

Concerning the `fare_amount`, this is an attribute that can be difficult to properly visualize. So, we can have a look to the above table. What we immediately observe is that there are negative values! We may thinking of a range of ordinary values for this attribute, something like between `$3.00` and `$200.00`

In [14]:
```python
fare_amount = ((df.fare_amount  >=3.0 ) & (df.fare_amount <=200.0))
```

From the dictionary above, concerning the attribute `improvement_surcharge` it is straightforward to save only trips with `$0.3` value. The same idea applies to the `mta_tax`:

In [15]:
```python
surcharge = (df.improvement_surcharge  == 0.3)
mta_tax = (df.mta_tax == 0.5)
```

An useful representation from `tolls_amount` is very difficult because of the huge range of values. A possible reason of that is that drivers manually introduced them. Also, these values can change as the course of the time, so they probably aren't going to be same in all the month. A solution for that might be to obtain the values that are repeated, for example, more that a thousand times:

In [16]:
```python
tolls = df.groupby(['tolls_amount']).size()

print(tolls[tolls >= 1000.00])
tolls = None
```

```
tolls_amount
0.00     11667901
2.54         5046
5.33         3170
5.54       609922
8.00         1059
9.75         7211
11.08        3433
11.75        9259
17.29        1099
dtype: int64
```

Therefore, a good range for this attribute could be something like `$0.00` and `$30.00`

```
In [17]: # Uncomment, and check it out by yourself!
         # ax = df.groupby(df['tolls_amount']).size().plot(kind='bar')

         # ax.set_xlabel('tolls amount', fontsize=18)
         # ax.set_ylabel('Number of trips', fontsize=18)
         # plt.title('Yellow taxi trips - June 2015')
         # ax.tick_params(labelsize=12)
```

```
In [18]: tolls_amount = ((df.tip_amount >=0.0) & (df.tip_amount <=30.0))
```

Regarding the `tip_amount`, as suspected, we notice that most cash fares have a tip of `$0`, which seems odd.

```
In [19]: ax = df[df['tip_amount'] > 0].groupby(['payment_type']).size().plot(kind = 'bar')

         ax.set_xlabel('payment_type', fontsize=18)
         ax.set_ylabel('Number of trips', fontsize=18)
         plt.title('Yellow taxi trips - June 2015', fontsize=15)
         ax.tick_params(labelsize=12)
```

A possible explanation is that it's very possible drivers are under-reporting cash tips, in order to pocket all of the cash themselves, which obviously skews our data quite a bit. So, we remove this annoying noise, deleting the `CSH payment type`. Moreover, it seems reasonable to assume an upper limit of a value of `$100.00`. Also, we drop unnecessary columns:

```
In [20]: tip_amount   = ((df.tip_amount >=0.0) & (df.tip_amount <=100.0))

         df = df[payment_type & fare_amount & surcharge & mta_tax & tip_amount & tolls_amount]
         payment_type = None
         surcharge = None
         fare_amount = None
         mta_tax = None
         tip_amount = None
         tolls_amount = None

         # drop unnecessary columns
         df.drop(['VendorID','RateCodeID','store_and_fwd_flag'], axis=1, inplace=True)
```

1.4 Cleaning the Data Set

We need to remove the noise: some bad data with 0 km trips, impossible gps coordinates and so on. . .

In an effort to determine trends in our data, in the next we add and reformat a number of columns to our dataframe for features that we thought might be interesting to predict or might be valuable as explanatory factors in predicting other features, such as:

- trip time in seconds
- trip time in minutes
- trip time in hours
- time of day (minutes since midnight)
- hour of the trip
- time of day (morning, afternoon, etc.)
- speed of trip (mph)
- day of the week
- month of the year
- cost of trip (total cost minus tip)
- percent tip (tip/cost)

In [21]:
```
trip_time = df.tpep_dropoff_datetime - df.tpep_pickup_datetime
# ATTENTION, this in timedelta64[ns].
# we need to convert it in either in seconds or minutes or in hours, as you prefer
```

In [22]:
```
# generate a column for trip time in secs
trip_time_in_secs = (trip_time / np.timedelta64(1,'s'))
# generate a column for trip time in minutes
trip_time_in_mins = (trip_time / np.timedelta64(1,'m'))
# generate a column for trip time in hours
trip_time_in_hours = (trip_time / np.timedelta64(1,'h'))
```

In [23]:
```
df['trip_time_in_secs'] = trip_time_in_secs
df['trip_time_in_mins'] = trip_time_in_mins
df['trip_time_in_hours'] = trip_time_in_hours
```

In [24]:
```
# A brief statistical view of what is going on...
# df.describe()
```

In [25]:
```
# generate column for minutes since midnight
df['time'] = [a.hour*60 + a.minute for a in df['tpep_pickup_datetime']]

# generate a column for hour of the day
df['hour'] = [a.hour for a in df['tpep_pickup_datetime']]

# create a column for time of day
df['time_of_day'] = ['morning' if (4 <= time.hour < 12) else 'afternoon' \
                     if (12 <= time.hour < 17) else 'evening' \
                     if (17 <= time.hour < 21) else 'night' \
                     for time in df['tpep_pickup_datetime']]

# generate a column for average speed of the trip
# speed in mph
df['speed'] = df['trip_distance']/df['trip_time_in_hours']

# generate a column for the day of the week
df['weekday'] = [a.weekday() for a in df['tpep_pickup_datetime']]

# generate a column to note whether it was a weekday or a weekend
df['weekend'] = [1 if (a == 5 or a == 6) else 0 for a in df['weekday']]
```

For obtaining a better value to predict, we can obtain a normalized version of the tip, the tip percentage.

In [26]:
```
# Let's create a new column for the percentage tip
subtotal = df['fare_amount'] + df['improvement_surcharge'] + df['mta_tax']
```

```python
tip = df['tip_amount'] / subtotal
df['percent_tip'] = tip
# quick look to the statistical values of the new attributes
df.describe()
```

Out[26]:

```
       passenger_count  trip_distance  pickup_longitude  pickup_latitude  \
count     1.217787e+07   1.217787e+07      1.217787e+07     1.217787e+07
mean      1.684918e+00   1.190323e+01     -7.296978e+01     4.019767e+01
std       1.338153e+00   7.724734e+03      8.565742e+00     4.715785e+00
min       0.000000e+00   0.000000e+00     -7.592333e+02    -6.713696e+01
25%       1.000000e+00   1.030000e+00     -7.399190e+01     4.073632e+01
50%       1.000000e+00   1.750000e+00     -7.398155e+01     4.075331e+01
75%       2.000000e+00   3.210000e+00     -7.396658e+01     4.076798e+01
max       9.000000e+00   1.008332e+07      1.490285e+02     6.970258e+01

       dropoff_longitude  dropoff_latitude   fare_amount         extra  \
count       1.217787e+07      1.217787e+07  1.217787e+07  1.217787e+07
mean       -7.301819e+01      4.022546e+01  1.299820e+01  3.243466e-01
std         8.358233e+00      4.606182e+00  1.021219e+01  3.648137e-01
min        -7.541667e+02     -1.617787e+01  3.000000e+00 -3.050000e+01
25%        -7.399126e+01      4.073493e+01  6.500000e+00  0.000000e+00
50%        -7.397961e+01      4.075393e+01  9.500000e+00  0.000000e+00
75%        -7.396260e+01      4.076885e+01  1.500000e+01  5.000000e-01
max         1.255356e+02      4.834500e+02  2.000000e+02  4.005000e+01

          mta_tax    tip_amount      ...       total_amount  \
count  12177874.0  1.217787e+07      ...       1.217787e+07
mean          0.5  1.711114e+00      ...       1.612349e+01
std           0.0  2.294089e+00      ...       1.261241e+01
min           0.5  0.000000e+00      ...      -1.995000e+01
25%           0.5  0.000000e+00      ...       8.800000e+00
50%           0.5  1.200000e+00      ...       1.230000e+01
75%           0.5  2.360000e+00      ...       1.830000e+01
max           0.5  3.000000e+01      ...       6.364330e+03

       trip_time_in_secs  trip_time_in_mins  trip_time_in_hours          time  \
count       1.217787e+07       1.217787e+07        1.217787e+07  1.217787e+07
mean        9.282812e+02       1.547135e+01        2.578559e-01  8.406779e+02
std         2.348369e+03       3.913948e+01        6.523247e-01  3.918541e+02
min        -2.153600e+05      -3.589333e+03       -5.982222e+01  0.000000e+00
25%         4.110000e+02       6.850000e+00        1.141667e-01  5.620000e+02
50%         6.850000e+02       1.141667e+01        1.902778e-01  8.760000e+02
75%         1.111000e+03       1.851667e+01        3.086111e-01  1.175000e+03
max         1.188823e+06       1.981372e+04        3.302286e+02  1.439000e+03

               hour         speed       weekday       weekend   percent_tip
count  1.217787e+07  1.216682e+07  1.217787e+07  1.217787e+07  1.217787e+07
mean   1.351906e+01           inf  2.862225e+00  2.623815e-01  1.216904e-01
std    6.524266e+00           NaN  1.989614e+00  4.399289e-01  1.136837e-01
min    0.000000e+00 -2.910000e+03  0.000000e+00  0.000000e+00  0.000000e+00
25%    9.000000e+00  7.584366e+00  1.000000e+00  0.000000e+00  0.000000e+00
50%    1.400000e+01  1.046005e+01  3.000000e+00  0.000000e+00  1.438849e-01
75%    1.900000e+01  1.423636e+01  5.000000e+00  1.000000e+00  2.078125e-01
max    2.300000e+01           inf  6.000000e+00  1.000000e+00  7.894737e+00
```

```
[8 rows x 22 columns]
```

A tip with `~800,00%` of the fare! Let's eliminate trips with tips greater that `100%`. Perhaps, it is more reasonable to adjust the percentage to a more ordinary range, something like `0%` and `50%`.

```
In [27]: df = df[df['percent_tip'] <= 50]
```

Now that we have defined the columns above, we can do some simple counts to ensure our data set is truly a random sample. We can easily confirm that the distribution is relatively even.

```
In [28]: print(df.groupby(['weekday']).size())
         print(df.groupby(['hour']).size())
```

```
weekday
0    1878919
1    1988245
2    1668131
3    1712401
4    1734929
5    1717102
6    1478147
dtype: int64
hour
0     484383
1     348094
2     249919
3     176491
4     132908
5     129905
6     283058
7     462703
8     565501
9     569051
10    556866
11    575996
12    595904
13    589618
14    606758
15    565060
16    490649
17    591176
18    714831
19    747566
20    686963
21    728522
22    705159
23    620793
dtype: int64
```

Let's move on the other physical attributes. When first examining the data, we discovered several outliers in the data that simply did not make any sense, such as:

- trips that apparently lasted for $>$ `10000` minutes
- cabs that hit speeds above `60mph`
- cabs that traveled at `0mph`

- tips 800% of the fare
- pickup and dropoff locations in Antarctica

We therefore removed these outliers with cutoff values we deemed reasonable for each feature, and dropped unncessary columns in order to simplify the dataframe.

In [29]:
```
# eliminate trips with unreasonable speeds (in excess of 60mph)
df = df[(df.speed < 60.0) & (df.speed > 0.0)]
```

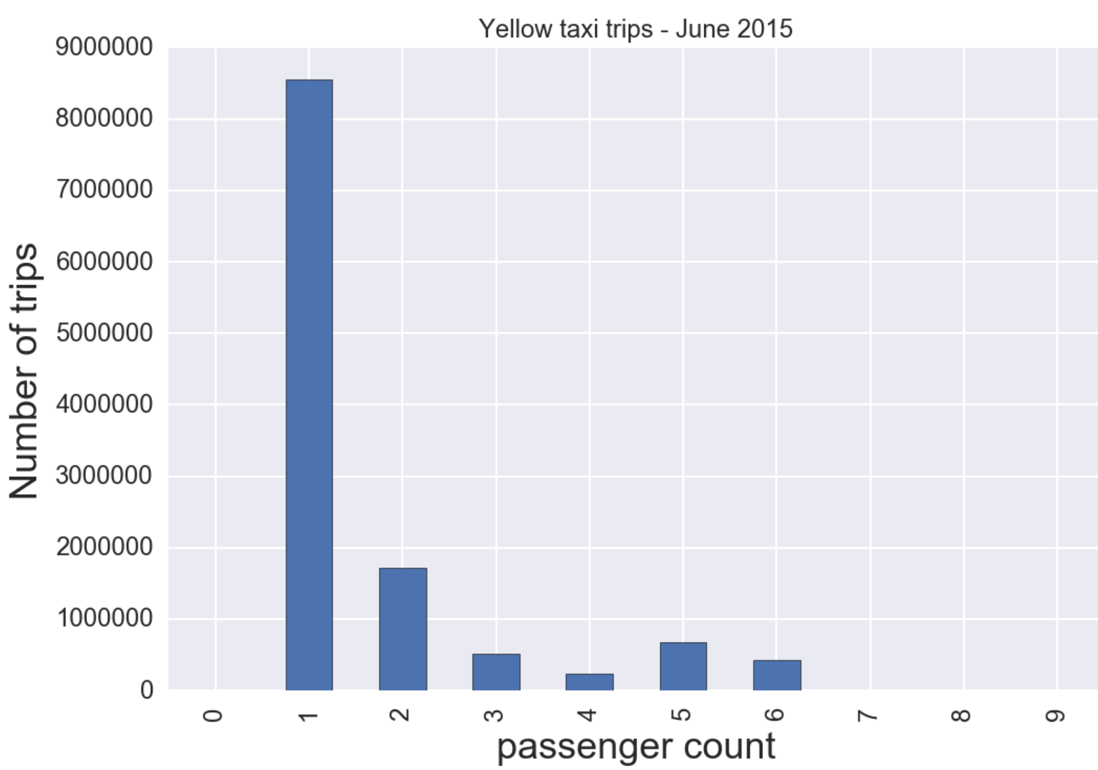
Concerning the cleaning part, we are not done yet... Negative values, trips lasting more than 10 days traveling million of miles. A crazy thing! So, for fixing that, we can use Google Maps and look for a long, but usual trip, like this one. A trip around 50 minutes for travelling 21.1 miles. Therefore, we can use a maximum of 1 hour (3,600 seconds) and 25 miles.

In [30]:
```
# A usual trip has 1 to 6 passengers. So, we can discard the others.
ax = df.groupby('passenger_count').size().plot(kind='bar')
ax.set_xlabel('passenger count', fontsize=18)
ax.set_ylabel('Number of trips', fontsize=18)
plt.title('Yellow taxi trips - June 2015')
ax.tick_params(labelsize=12)

passenger_count = ((df.passenger_count >= 1.0) & (df.passenger_count <= 6.0))
trip_time_in_secs = ((df.trip_time_in_secs > 0.0) & (df.trip_time_in_secs <= 3600.0))
trip_distance = ((df.trip_distance > 0.0) & (df.trip_distance <= 25.0))

df = df[passenger_count & trip_time_in_secs & trip_distance]

passenger_count = None
trip_time_in_secs = None
trip_distance = None
```



Continuing with the attributes, it's the turn of the coordinates, longitude and latitude for pickups and dropoffs. By observing the above dataframe, we notice coordinates that don't even exist! For fix that, we can use only the coordinates satisfying the conditions in the following table:

	Min	Max
Latitude	40.459518	41.175342
Longitude	−74.361107	−71,903083

```
In [31]: # eliminate outliers based on location
         pickup_latitude = ((df.pickup_latitude >= 40.459518) & (df.pickup_latitude <= 41.175342))
         pickup_longitude = ((df.pickup_longitude >= -74.361107) & (df.pickup_longitude <= -71.903083))
         dropoff_latitude = ((df.dropoff_latitude >= 40.459518) & (df.dropoff_latitude <= 41.175342))
         dropoff_longitude = ((df.dropoff_longitude >= -74.361107) & (df.dropoff_longitude <= -71.90308

         df = df[pickup_latitude & pickup_longitude & dropoff_latitude & dropoff_longitude]

         pickup_latitude = None
         pickup_longitude = None
         dropoff_latitude = None
         dropoff_longitude = None
```

Now, we would like to be able to subset the taxi data by the neighborhoods people were travelling to and from in order to discover any trends that may be there; thus, we create columns for pickup and dropoff neighborhood using geographic boundaries defined using Google maps.

```
In [32]: # Make a tuple column for pickup and dropoff latitudes and longitudes
         df['pickup_lat_long'] = list(zip(df.pickup_latitude, df.pickup_longitude))
         df['dropoff_lat_long'] = list(zip(df.dropoff_latitude, df.dropoff_longitude))
         ### COMMENT: In Python 3, zip returns an iterator of tuples, like itertools.izip in Python2.
         ### To get a list of tuples, use list(zip(foo, bar)). And to zip until both iterators are exha
         ## you would use itertools.zip_longest.

         # Define the pickup neighborhood column
         df['pickup_neighborhood'] = ['Upper East Side' if ((-73.93269 <= longitude <= -73.958506 and (
                                      or (-73.958506 < longitude <= -73.955760 and (1.351248*longitude
                                      or (-73.955760 < longitude <= -73.938250 and (1.351248*longitude
                                      else 'Upper West Side' if ((-73.996349 <= longitude <= -73.98192
                                      or (-73.981929 < longitude <= -73.971286 and (1.411006*longitude
                                      or (-73.971286 < longitude <= -73.958669 and (1.411006*longitude
                                      else 'East Harlem' if ((-73.955810 <= longitude <= -73.941562 an
                                      or (-73.941562 < longitude <= -73.934009 and (1.364892*longitude
                                      or (-73.934009 < longitude <= -73.927400 and (1.364892*longitude
                                      else 'Harlem' if ((-73.970920 <= longitude <= -73.949376 and (-
                                      or (-73.949376 < longitude <= -73.950406 and (1.335965*longitude
                                      or (-73.950406 < longitude <= -73.933669 and (1.335965*longitude
                                      else 'Washington Heights' if ((-73.952561 <= longitude <= -73.93
                                      or (-73.934450 < longitude <= -73.938313 and (2.030519*longitude
                                      or (-73.938313 < longitude <= -73.921147 and (2.030519*longitude
                                      else 'Chelsea' if ((-74.012918 <= longitude <= -74.004936 and (-
                                      or (-74.004936 < longitude <= -73.996181 and (-.425864*longitude
                                      or (-73.996181 < longitude <= -73.987684 and (1.46181*longitude
                                      else "Hell's Kitchen" if ((-74.005267 <= longitude <= -73.99402
```

```
                                or (-73.994023 < longitude <= -73.993423 and (-.455589*longitude
                                or (-73.993423 < longitude <= -73.982265 and (1.380893*longitude
                                else 'Midtown' if ((-73.993851 <= longitude <= -73.984495 and (
                                or (-73.984495 < longitude <= -73.981491 and (1.389750*longitude
                                or (-73.981491 < longitude <= -73.973080 and (1.389750*longitude
                                else 'Midtown East' if ((-73.984495 <= longitude <= -73.972908 a
                                or (-73.972908 < longitude <= -73.966986 and (-.425289*longitude
                                or (-73.966986 < longitude <= -73.959004 and (1.369121*longitude
                                else 'Murray Hill and Gramercy' if ((-73.996782 <= longitude <=
                                or (-73.987684 < longitude <= -73.971634 and (-.398282*longitude
                                or (-73.971634 < longitude <= -73.963909 and (1.422401*longitude
                                else 'East Village' if ((-73.992711 <= longitude <= -73.989621 a
                                or (-73.989621 < longitude <= -73.972626 and (-.430977*longitude
                                or (-73.972626 < longitude <= -73.971511 and (3.283819*longitude
                                else 'West Village' if ((-74.014761 <= longitude <= -74.009354 a
                                or (-74.009354 < longitude <= -74.003603 and (-.391182*longitude
                                or (-74.003603 < longitude <= -73.996222 and (1.374746*longitude
                                else 'Greenwich Village' if ((-74.002925 <= longitude <= -73.99
                                or (-73.996230 < longitude <= -73.992711 and (-.454489*longitude
                                or (-73.992711 < longitude <= -73.989792 and (3.498458*longitude
                                else 'Financial District' if ((-74.017118 <= longitude <= -74.0
                                or (-74.012741 < longitude <= -74.010166 and (-.689918*longitude
                                or (-74.010166 < longitude <= -73.999351 and (3.348184*longitude
                                else 'Lower East Side' if ((-74.001139 <= longitude <= -73.9926
                                or (-73.992642 < longitude <= -73.978223 and (-.296245*longitude
                                or (-73.978223 < longitude <= -73.973759 and (2.520648*longitude
                                else 'Soho' if ((-74.017018 <= longitude <= -74.011096 and (-.3
                                or (-74.011096 < longitude <= -74.001225 and (-.313095*longitude
                                or (-74.001225 < longitude <= -73.992814 and (1.631911*longitude
                                else 'Central Park' if ((-73.981834 <= longitude <= -73.972994 a
                                or (-73.972994 < longitude <= -73.957716 and (1.364612*longitude
                                or (-73.957716 < longitude <= -73.949133 and (1.364612*longitude
                                else 'New Jersey' if (latitude >= 1.691689*longitude + 165.9601
                                else 'Brooklyn' if (-74.042158 <= longitude <= -73.858137 and la
                                else 'Laguardia Airport' if (-73.889398 <= longitude <= -73.855
                                else 'JFK Airport' if (-73.833340 <= longitude <= -73.747166 an
                                else 'Queens' if (-73.940543 <= longitude <= -73.724937 and 40.
                                else 'Other' for latitude, longitude in df['pickup_lat_long']]

# Define the dropoff neighborhood column
df['dropoff_neighborhood'] = ['Upper East Side' if ((-73.93269 <= longitude <= -73.958506 and
                                or (-73.958506 < longitude <= -73.955760 and (1.351248*longitude
                                or (-73.955760 < longitude <= -73.938250 and (1.351248*longitude
                                else 'Upper West Side' if ((-73.996349 <= longitude <= -73.9819
                                or (-73.981929 < longitude <= -73.971286 and (1.411006*longitude
                                or (-73.971286 < longitude <= -73.958669 and (1.411006*longitude
                                else 'East Harlem' if ((-73.955810 <= longitude <= -73.941562 an
                                or (-73.941562 < longitude <= -73.934009 and (1.364892*longitude
                                or (-73.934009 < longitude <= -73.927400 and (1.364892*longitude
                                else 'Harlem' if ((-73.970920 <= longitude <= -73.949376 and (-
                                or (-73.949376 < longitude <= -73.950406 and (1.335965*longitude
                                or (-73.950406 < longitude <= -73.933669 and (1.335965*longitude
                                else 'Washington Heights' if ((-73.952561 <= longitude <= -73.9
                                or (-73.934450 < longitude <= -73.938313 and (2.030519*longitude
```

```
or (-73.938313 < longitude <= -73.921147 and (2.030519*longitud
else 'Chelsea' if ((-74.012918 <= longitude <= -74.004936 and (
or (-74.004936 < longitude <= -73.996181 and (-.425864*longitud
or (-73.996181 < longitude <= -73.987684 and (1.46181*longitude
else "Hell's Kitchen" if ((-74.005267 <= longitude <= -73.99402
or (-73.994023 < longitude <= -73.993423 and (-.455589*longitud
or (-73.993423 < longitude <= -73.982265 and (1.380893*longitud
else 'Midtown' if ((-73.993851 <= longitude <= -73.984495 and (
or (-73.984495 < longitude <= -73.981491 and (1.389750*longitud
or (-73.981491 < longitude <= -73.973080 and (1.389750*longitud
else 'Midtown East' if ((-73.984495 <= longitude <= -73.972908 a
or (-73.972908 < longitude <= -73.966986 and (-.425289*longitud
or (-73.966986 < longitude <= -73.959004 and (1.369121*longitud
else 'Murray Hill and Gramercy' if ((-73.996782 <= longitude <=
or (-73.987684 < longitude <= -73.971634 and (-.398282*longitud
or (-73.971634 < longitude <= -73.963909 and (1.422401*longitud
else 'East Village' if ((-73.992711 <= longitude <= -73.989621 a
or (-73.989621 < longitude <= -73.972626 and (-.430977*longitud
or (-73.972626 < longitude <= -73.971511 and (3.283819*longitud
else 'West Village' if ((-74.014761 <= longitude <= -74.009354 a
or (-74.009354 < longitude <= -74.003603 and (-.391182*longitud
or (-74.003603 < longitude <= -73.996222 and (1.374746*longitud
else 'Greenwich Village' if ((-74.002925 <= longitude <= -73.99
or (-73.996230 < longitude <= -73.992711 and (-.454489*longitud
or (-73.992711 < longitude <= -73.989792 and (3.498458*longitud
else 'Financial District' if ((-74.017118 <= longitude <= -74.0
or (-74.012741 < longitude <= -74.010166 and (-.689918*longitud
or (-74.010166 < longitude <= -73.999351 and (3.348184*longitud
else 'Lower East Side' if ((-74.001139 <= longitude <= -73.9926
or (-73.992642 < longitude <= -73.978223 and (-.296245*longitud
or (-73.978223 < longitude <= -73.973759 and (2.520648*longitud
else 'Soho' if ((-74.017018 <= longitude <= -74.011096 and (-.3
or (-74.011096 < longitude <= -74.001225 and (-.313095*longitud
or (-74.001225 < longitude <= -73.992814 and (1.631911*longitud
else 'Central Park' if ((-73.981834 <= longitude <= -73.972994 a
or (-73.972994 < longitude <= -73.957716 and (1.364612*longitud
or (-73.957716 < longitude <= -73.949133 and (1.364612*longitud
else 'New Jersey' if (latitude >= 1.691689*longitude + 165.9601
else 'Brooklyn' if (-74.042158 <= longitude <= -73.858137 and l
else 'Laguardia Airport' if (-73.889398 <= longitude <= -73.855
else 'JFK Airport' if (-73.833340 <= longitude <= -73.747166 an
else 'Queens' if (-73.940543 <= longitude <= -73.724937 and 40.
else 'Other' for latitude, longitude in df['dropoff_lat_long']]
```

At this stage, we believe that the data set is pretty much clean, and set up to analyze actual trends. This is our dataframe we begin to work with

1.5 Predictive Task

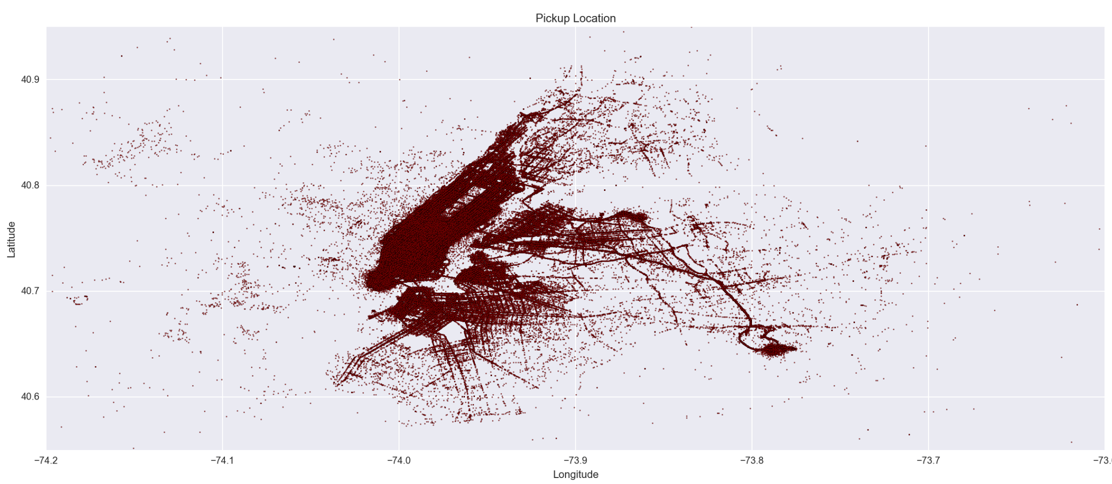
The predictive task that is being analyzed is the percentage of tip in relation to the total amount paid for taxi trips in New York City (NYC). The predictive task was chosen due to our curiosity of what factors causes people to tip higher percentages. Ultimately, this analysis can be used to assist taxi drivers in considering these factors in order to better understand their business and how they can maximize the tip received.

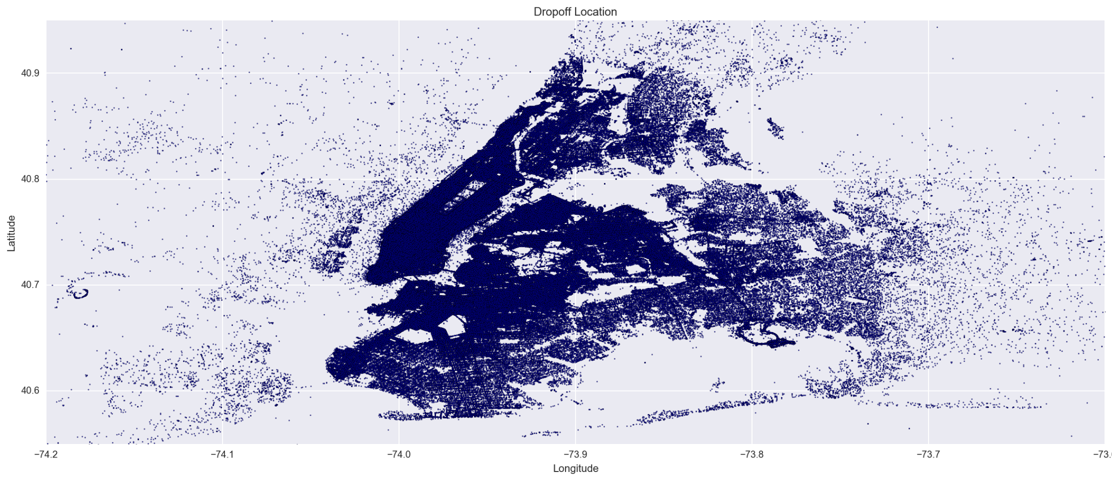
1.5.1 Preparing the notebook

We think it might be a good idea to actually plot the geographic locations of the taxi pickups and dropoffs in order to gain a better understanding of where taxi trips are most concentrated:

In [33]:
```
plt.figure(figsize =(20,8))
plt.scatter(df['pickup_longitude'],df['pickup_latitude'],s=1,alpha=0.8,c='r')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Pickup Location')
plt.xlim(-74.2, -73.6)
plt.ylim(40.55, 40.95)
plt.show()

plt.figure(figsize =(20,8))
plt.scatter(df['dropoff_longitude'],df['dropoff_latitude'],s=1,alpha=0.8,c='b')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Dropoff Location')
plt.xlim(-74.2, -73.6)
plt.ylim(40.55, 40.95)
plt.show()
```





We see that the vast majority of trips are clustered in Manhattan and in two smaller spots outside the immediate vacinity of the city. Searching for the values of those latitudes and longitudes in Google Maps led us to discover that those smaller spots are reflective of the Laguardia and JFK Airports. As it turns out, $52 is the flat rate for a trip from Manhattan to JFK Airport, explaining the spike in fare amount noted in the histogram above. We then set about trying to figure out if we could determine any trends in the data that would suggest a high correlation between them and the amount of a fare. We assumed that trip time and trip distance would likely be the most highly correlated measurements, as taxi companies use a direct calculation in determining fare that relies upon trip time and distance. To confirm this, we plotted them below.

```
In [ ]: # plot a simple histogram
        import seaborn as sns
        plt.figure(figsize =(20,8))
        plt.hist(df['fare_amount'],bins=100)
        plt.xlabel('Fare')
        plt.ylabel('Frequency')
        plt.title('Fare Amount')
```

We looked at tip as a percentage of the total cost of a trip, as clearly longer trips with higher fares will be correlated with higher tips.

```
In [ ]: df.percent_tip.mean() * 100
```

Clearly, the average tip for people paying with cards is considerably higher than for people paying with cash, to the point of being totally unreasonable. This led us to the conclusion that there must be some kind of fraud going on when people pay with cash - specifically, cab drivers are likely not reporting their cash tips and simply pocketing the entirety of it. Thus for the purpose of an accurate analysis of tips, we decided to only look at trips where passengers paid card. This way we could be sure that the tip was accurately reported. We also decided that an interesting goal of this analysis might be to predict tip percentage in order to determine whether or not a taxi driver is committing fraud by under-reporting tips - if our predictions are accurate, we will be able to guess what tip a driver should have received based on the features of their trip, and then if the reported tip is much less than this we will have reason to be suspicious.

```
In [ ]:
```

```
In [ ]: import matplotlib as plt
        import warnings
```

```python
warnings.filterwarnings('ignore')

pd.options.display.mpl_style = 'default' # Better Styling
new_style = {'grid': False} # Remove grid
matplotlib.rc('axes', **new_style)
from matplotlib import rcParams
rcParams['figure.figsize'] = (15, 15) # Size of figure
rcParams['figure.dpi'] = 125

P = df.plot(kind='scatter', x='pickup_longitude', y='pickup_latitude',\
            color='white',xlim=(-74.36,-71,90),ylim=(40.61, 40.91),s=.02,alpha=.6)

P.set_axis_bgcolor('black') # Background Color
```