Certain part of this code, such as distance formulas are reference from Kaggle competition.

Code Cite

[1]
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

[2]
```
%matplotlib inline
import pandas as pd
import time
import numpy as np
import xgboost as xgb
from datetime import date
import holidays
import json
import math
import seaborn as sns
!pip install mapboxgl
from mapboxgl.utils import *
from mapboxgl.viz import *
%matplotlib inline

from datetime import timedelta
import datetime as dt
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [16, 10]
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.cluster import MiniBatchKMeans
import warnings
warnings.filterwarnings('ignore')

# !pip install pyowm
```

```
# import pyowm

!pip install wwo-hist
from wwo_hist import retrieve_hist_data

!pip install -U tables
os.environ['MAPBOX_ACCESS_TOKEN'] =
"pk.eyJ1IjoiY2VydWxlYW5ndSIsImEiOiJjazJ6cGs2NGUwYjhvM2JwZGVqZzl4N
mx6In0.aIKAagrGcWYMZ2x7JbDrWg"
```

```
Collecting mapboxgl
  Downloading
https://files.pythonhosted.org/packages/4f/e1/cdaa6c2f6d3a7a29b0b9a675dc
fc25f4c481d577d137da8c769f13014ce5/mapboxgl-0.10.2-py2.py3-none-any.whl
(43kB)
     |████████████████████████████████| 51kB 3.9MB/s eta 0:00:011
Requirement already satisfied: jinja2 in /usr/local/lib/python3.6/dist-
packages (from mapboxgl) (2.10.3)
Collecting chroma-py
  Downloading
https://files.pythonhosted.org/packages/23/3c/39d07abb9d4bcda64d9c50a4fd
a2ecfee4d76a436bd590d232a4e1a5ad43/chroma-py-0.1.0.dev1.tar.gz
Collecting colour
  Downloading
https://files.pythonhosted.org/packages/74/46/e81907704ab203206769dee138
5dc77e1407576ff8f50a0681d0a6b541be/colour-0.1.5-py2.py3-none-any.whl
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.6/dist-packages (from mapboxgl) (3.1.2)
Collecting geojson
  Downloading
https://files.pythonhosted.org/packages/e4/8d/9e28e9af95739e6d2d2f8d4bef
0b3432da40b7c3588fbad4298c1be09e48/geojson-2.5.0-py2.py3-none-any.whl
Requirement already satisfied: MarkupSafe>=0.23 in
/usr/local/lib/python3.6/dist-packages (from jinja2->mapboxgl) (1.1.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
in /usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(2.4.5)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(1.1.0)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(2.6.1)
Requirement already satisfied: numpy>=1.11 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(1.17.4)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(0.10.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1-
>matplotlib->mapboxgl) (42.0.2)
```

```
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.1-
>matplotlib->mapboxgl) (1.12.0)
Building wheels for collected packages: chroma-py
  Building wheel for chroma-py (setup.py) ... done
  Created wheel for chroma-py: filename=chroma_py-0.1.0.dev1-cp36-none-
any.whl size=5107
sha256=9d085665b0d81f237e418457dbb6514bde5aba5d51fa92b09632b8bf1dc8af05
  Stored in directory:
/root/.cache/pip/wheels/43/3b/8c/3f6d7536b8bef26b7c3be5989f8103513eb949e
50a4f9f81cf
Successfully built chroma-py
Installing collected packages: chroma-py, colour, geojson, mapboxgl
Successfully installed chroma-py-0.1.0.dev1 colour-0.1.5 geojson-2.5.0
mapboxgl-0.10.2
Collecting wwo-hist
  Downloading
https://files.pythonhosted.org/packages/9e/4b/e4f82813f1bd33195ca5eca204
c7c312850592aba45dd819de0f2250d7fd/wwo_hist-0.0.4.tar.gz
Building wheels for collected packages: wwo-hist
  Building wheel for wwo-hist (setup.py) ... done
  Created wheel for wwo-hist: filename=wwo_hist-0.0.4-cp36-none-any.whl
size=4217
sha256=adfb2c34b14cb94a95c5d0a37629d4a17dab69766390b5fc0dbf82de7dfe1e68
  Stored in directory:
/root/.cache/pip/wheels/c5/0e/16/329d9233f3b0b7e5fe81b09c5519a193727999c
6ae77577ca9
Successfully built wwo-hist
Installing collected packages: wwo-hist
Successfully installed wwo-hist-0.0.4
Collecting tables
  Downloading
https://files.pythonhosted.org/packages/ed/c3/8fd9e3bb21872f9d69eb93b301
4c86479864cca94e625fd03713ccacec80/tables-3.6.1-cp36-cp36m-
manylinux1_x86_64.whl (4.3MB)
     |████████████████████████████████| 4.3MB 9.5MB/s
Requirement already satisfied, skipping upgrade: numpy>=1.9.3 in
/usr/local/lib/python3.6/dist-packages (from tables) (1.17.4)
Requirement already satisfied, skipping upgrade: numexpr>=2.6.2 in
/usr/local/lib/python3.6/dist-packages (from tables) (2.7.0)
Installing collected packages: tables
  Found existing installation: tables 3.4.4
    Uninstalling tables-3.4.4:
      Successfully uninstalled tables-3.4.4
Successfully installed tables-3.6.1
```

[0]

```
def get_coords():
  with open('/content/drive/My Drive/10701/cx.json') as
json_file:
    longitude = json.load(json_file)
    longitude['1'] = -74.17446239999998
```

```
  with open('/content/drive/My Drive/10701/cy.json') as
json_file:
    latitude = json.load(json_file)
    latitude['1'] = 40.6895314

  return longitude, latitude

longitude, latitude = get_coords()
```

```
[0]
def haversine_distance(origin, destination):
    """
    Formula to calculate the spherical distance between 2
coordinates, with each specified as a (lat, lng) tuple

    :param origin: (lat, lng)
    :type origin: tuple
    :param destination: (lat, lng)
    :type destination: tuple
    :return: haversine distance
    :rtype: float
    """
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371  # km

    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2) * math.sin(dlat / 2) +
math.cos(math.radians(lat1)) * math.cos(
        math.radians(lat2)) * math.sin(dlon / 2) * math.sin(dlon
/ 2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = radius * c

    return d
```

```
[5]
raw_data = pd.read_hdf('/content/drive/My Drive/10701/subset.h5',
                       parse_dates=['tpep_pickup_datetime',
'tpep_dropoff_datetime']).sample(frac=1)
total_data = raw_data.shape[0]
split = int(total_data*0.8)
raw_data.rename(columns = {'PULocationX':'pickup_longitude',
'PULocationY':'pickup_latitude',
                              'DOLocationX':'dropoff_longitude',
'DOLocationY':'dropoff_latitude'}, inplace = True)
raw_train = raw_data.head(split)
raw_test = raw_data.tail(total_data - split)
print('Raw data size: ', raw_data.shape)
```

```
print('train size:', raw_train.shape)
print('test size:', raw_test.shape)

raw_train.head()
```

```
Raw data size:  (1000000, 14)
train size: (800000, 14)
test size: (200000, 14)
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	p
32613858	2	2017-04-13 03:30:38	2017-04-13 03:41:31	1
12485674	1	2017-02-10 20:53:26	2017-02-10 21:29:45	1
5301729	1	2017-01-10 11:28:15	2017-01-10 11:42:53	1
38768138	2	2017-04-12 13:21:13	2017-04-12 13:31:46	1
65467061	2	2017-07-23 18:06:50	2017-07-23 18:16:15	1

[0]

```
def get_pu_location(row):
  return latitude[str(row['PULocationID'])],
longitude[str(row['PULocationID'])]
def get_do_location(row):
  return latitude[str(row['DOLocationID'])],
longitude[str(row['DOLocationID'])]

us_holidays = holidays.UnitedStates()
def is_holiday(x):
  if x['tpep_pickup_datetime'] in us_holidays:
    return 1
  else:
    return 0

def process_test(data):
  data.drop_duplicates(inplace = True)
  data.dropna(inplace = True)

  data = data[data['PULocationID'] < 264]
  data = data[data['DOLocationID'] < 264]

  data['distance_haversine'] = data.apply(lambda x:
haversine_distance((x['pickup_latitude'],

x['pickup_longitude']),

(x['dropoff_latitude'],

x['dropoff_longitude'])), axis=1)
```

```
  # data['Month'] = data['tpep_pickup_datetime'].dt.month
  # data['DayofMonth'] = data['tpep_pickup_datetime'].dt.day
  # data['DayofWeek'] = data['tpep_pickup_datetime'].dt.dayofweek
  # data['Hour'] = data['tpep_pickup_datetime'].dt.hour
  # data['Minute'] = data['tpep_pickup_datetime'].dt.minute

  data['pickup_date'] = data['tpep_pickup_datetime'].dt.date
  data['dropoff_date'] = data['tpep_dropoff_datetime'].dt.date

  data['duration'] = (data['tpep_dropoff_datetime'] -
data['tpep_pickup_datetime']).dt.total_seconds() / 60.0
  data = data[data['duration'] > 0]
  data['log_duration'] = data['duration'].map(lambda x: np.log(x)
+ 1)

  # data['isHoliday'] = data.apply(lambda x: is_holiday(x),
axis=1)

  # train = pd.get_dummies(train, columns=['passenger_count',
'VendorID', 'Hour', 'Minute', 'Month', 'DayofMonth', 'DayofWeek',
'PULocationID', 'DOLocationID'])
  # ret = data.as_matrix().astype('float32')
  return data


def process_train(data):
  data = process_test(data)

  q = data['duration'].quantile(0.999)
  data = data[data['duration'] < q]
  # train = pd.get_dummies(train, columns=['passenger_count',
'VendorID', 'Hour', 'Minute', 'Month', 'DayofMonth', 'DayofWeek',
'PULocationID', 'DOLocationID'])
  # ret = data.as_matrix().astype('float32')
  return data
```

[7]

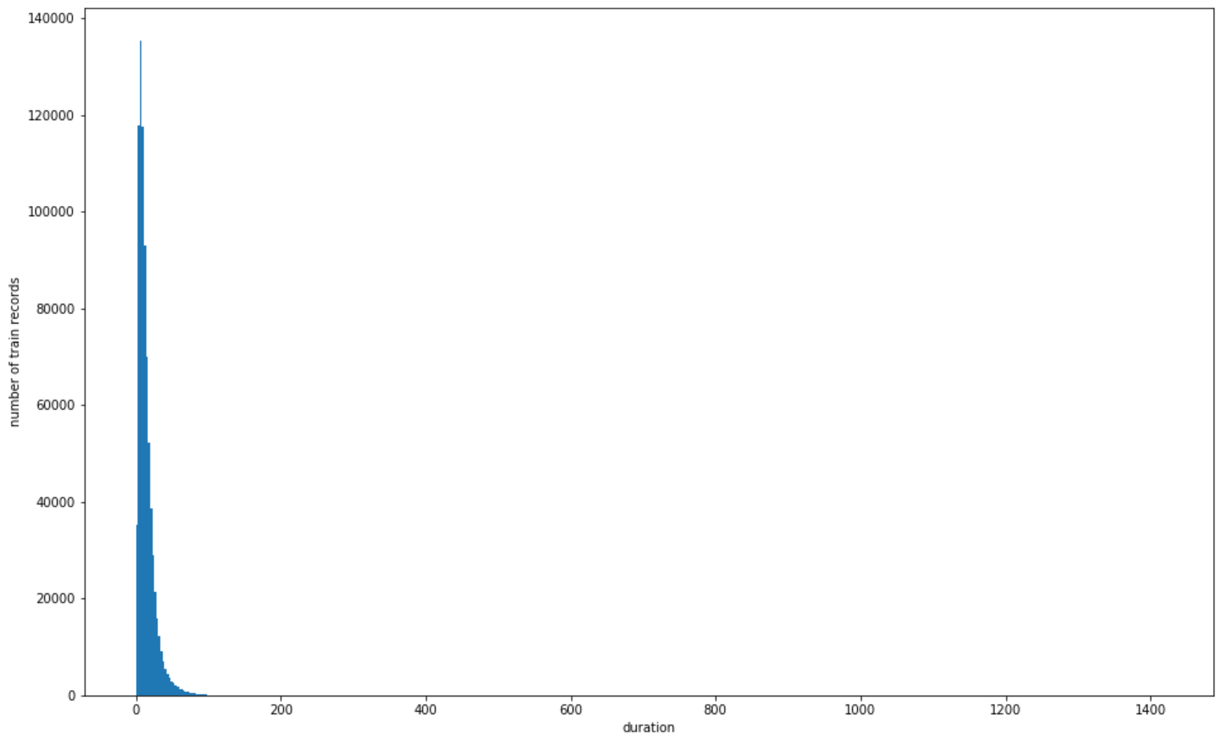
```
# data_df = load_csv('/content/drive/My
Drive/10701/assignment1_data-1.csv')
# data_df

train = process_train(raw_train)
test = process_test(raw_test)

print('train size:', train.shape)
print('test size:', test.shape)

train.head()
```

```
train size: (785092, 19)
test size: (196420, 19)
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	p
32613858	2	2017-04-13 03:30:38	2017-04-13 03:41:31	1
12485674	1	2017-02-10 20:53:26	2017-02-10 21:29:45	1
5301729	1	2017-01-10 11:28:15	2017-01-10 11:42:53	1
38768138	2	2017-04-12 13:21:13	2017-04-12 13:31:46	1
65467061	2	2017-07-23 18:06:50	2017-07-23 18:16:15	1

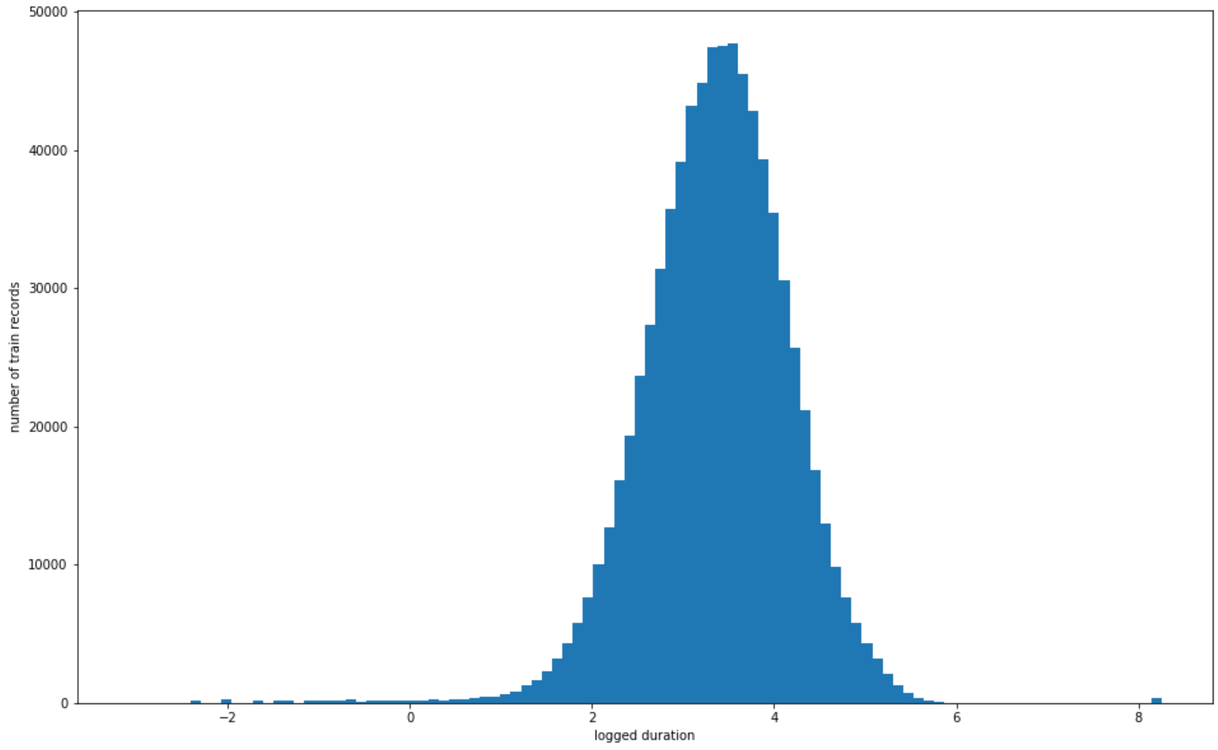
[14]

```
plt.hist(train['duration'].values, bins=500)
plt.xlabel('duration')
plt.ylabel('number of train records')
plt.show()
```



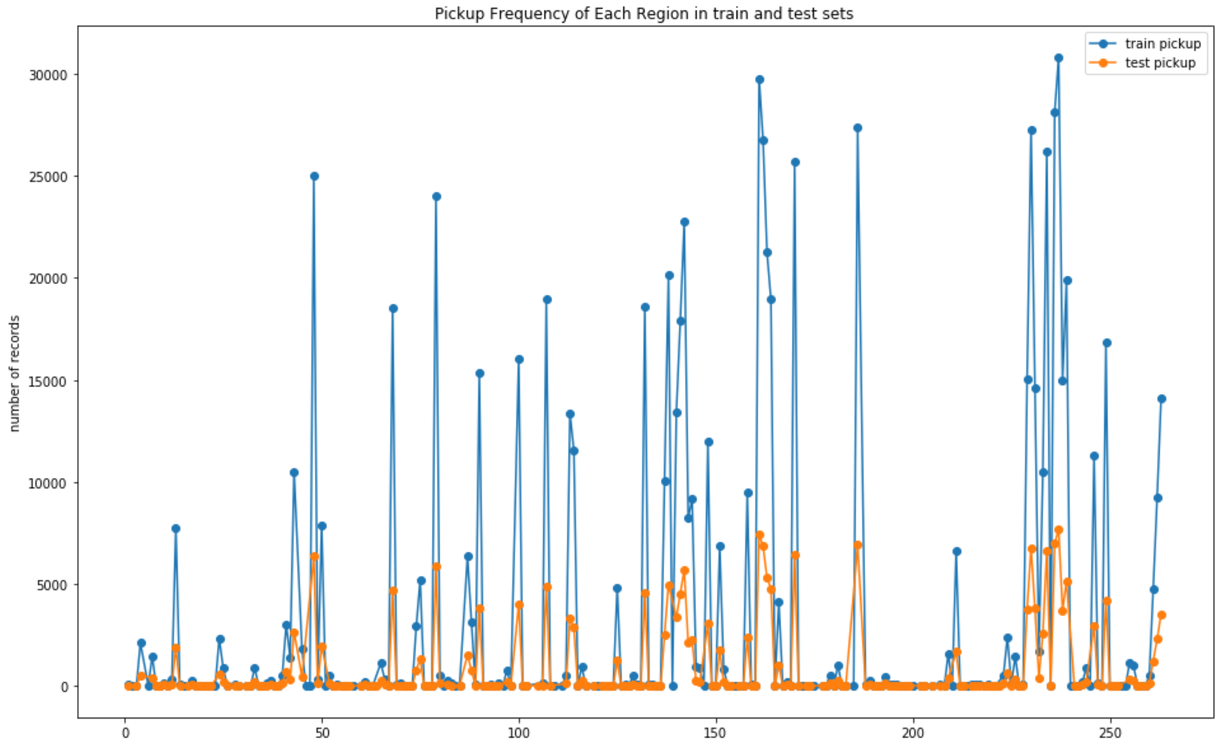
[17]

```
plt.hist(train['log_duration'].values, bins=100)
plt.xlabel('logged duration')
plt.ylabel('number of train records')
plt.show()
```



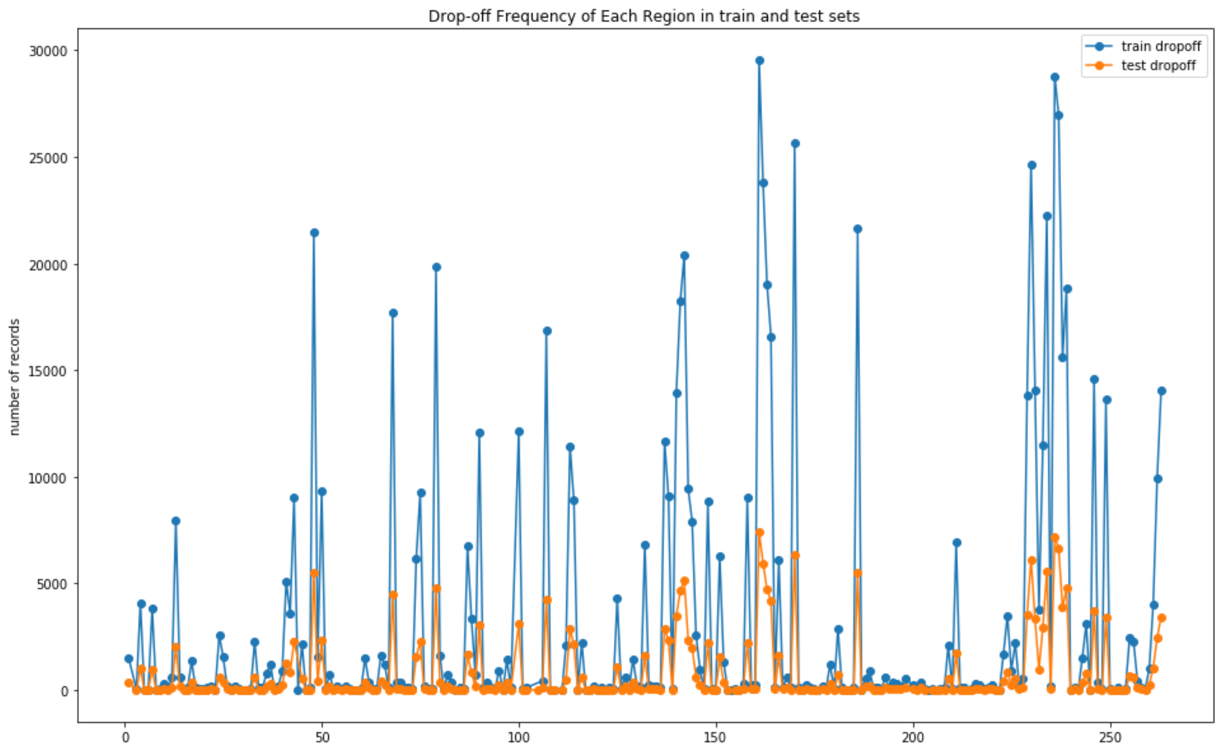
[11]

```
plt.plot(train.groupby('PULocationID').count()
[['tpep_pickup_datetime']], 'o-', label='train pickup')
plt.plot(test.groupby('PULocationID').count()
[['tpep_pickup_datetime']], 'o-', label='test pickup')

# plt.plot(test.groupby('pickup_date').count()[['id']], 'o-',
label='test')
plt.title('Pickup Frequency of Each Region in train and test
sets')
plt.legend(loc=0)
plt.ylabel('number of records')
plt.show()
```



[18]

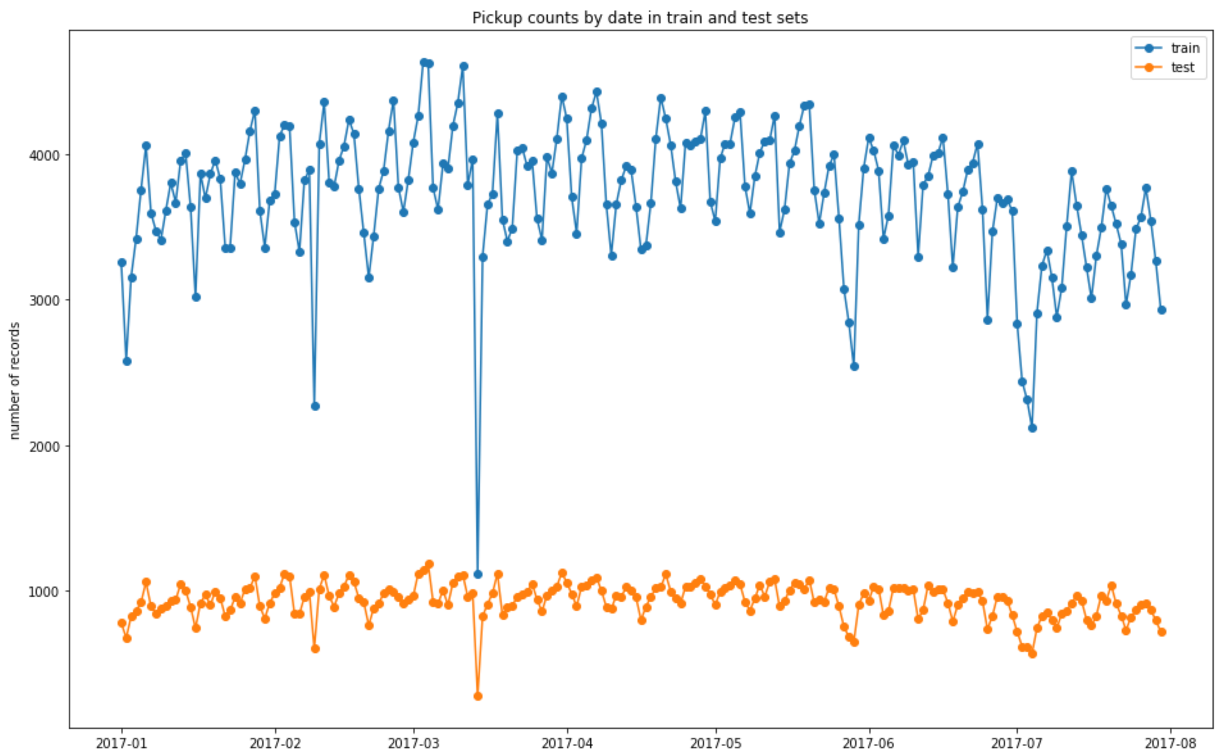
```
plt.plot(train.groupby('DOLocationID').count()
[['tpep_dropoff_datetime']], 'o-', label='train dropoff')
plt.plot(test.groupby('DOLocationID').count()
[['tpep_dropoff_datetime']], 'o-', label='test dropoff')

plt.title('Drop-off Frequency of Each Region in train and test
sets')
plt.legend(loc=0)
plt.ylabel('number of records')
plt.show()
```



let's check the train test split. It helps to decide our validation strategy and gives ideas about feature engineering

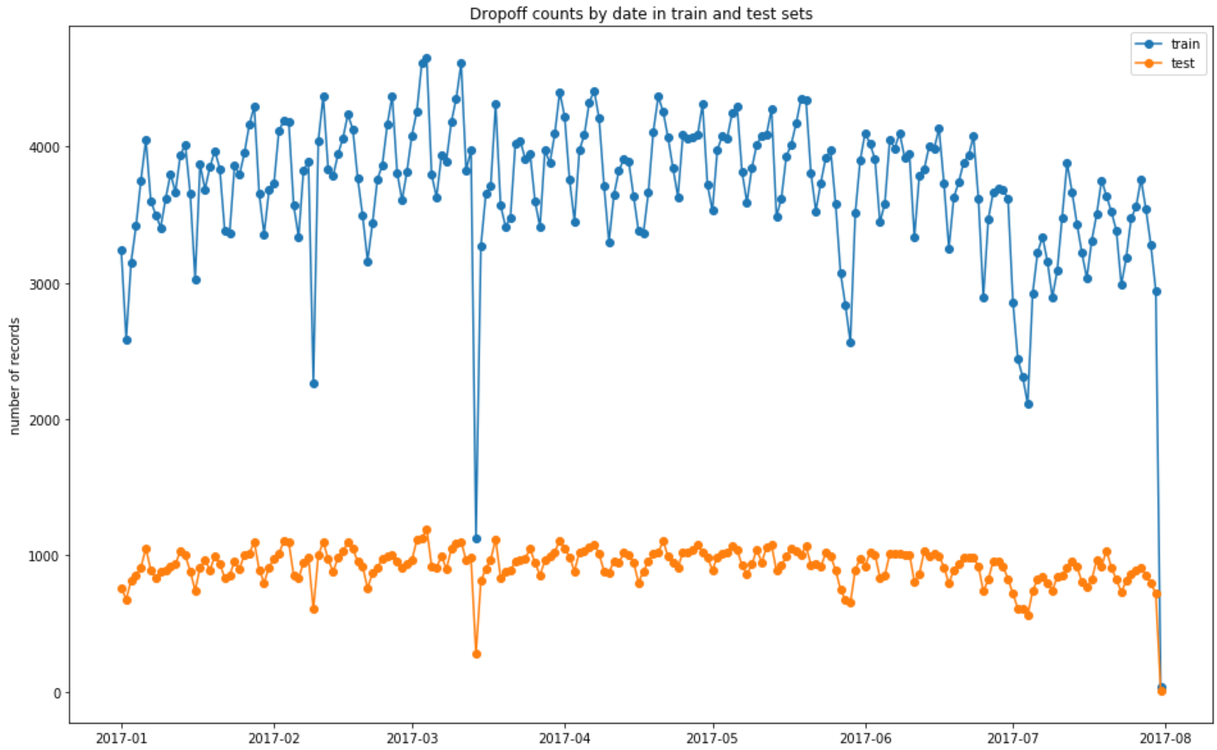
[26]

```
plt.plot(train.groupby('pickup_date').count()
[['tpep_pickup_datetime']], 'o-', label='train')
plt.plot(test.groupby('pickup_date').count()
[['tpep_pickup_datetime']], 'o-', label='test')
plt.title('Pickup counts by date in train and test sets')
plt.legend(loc=0)
plt.ylabel('number of records')
plt.show()
```



[25]

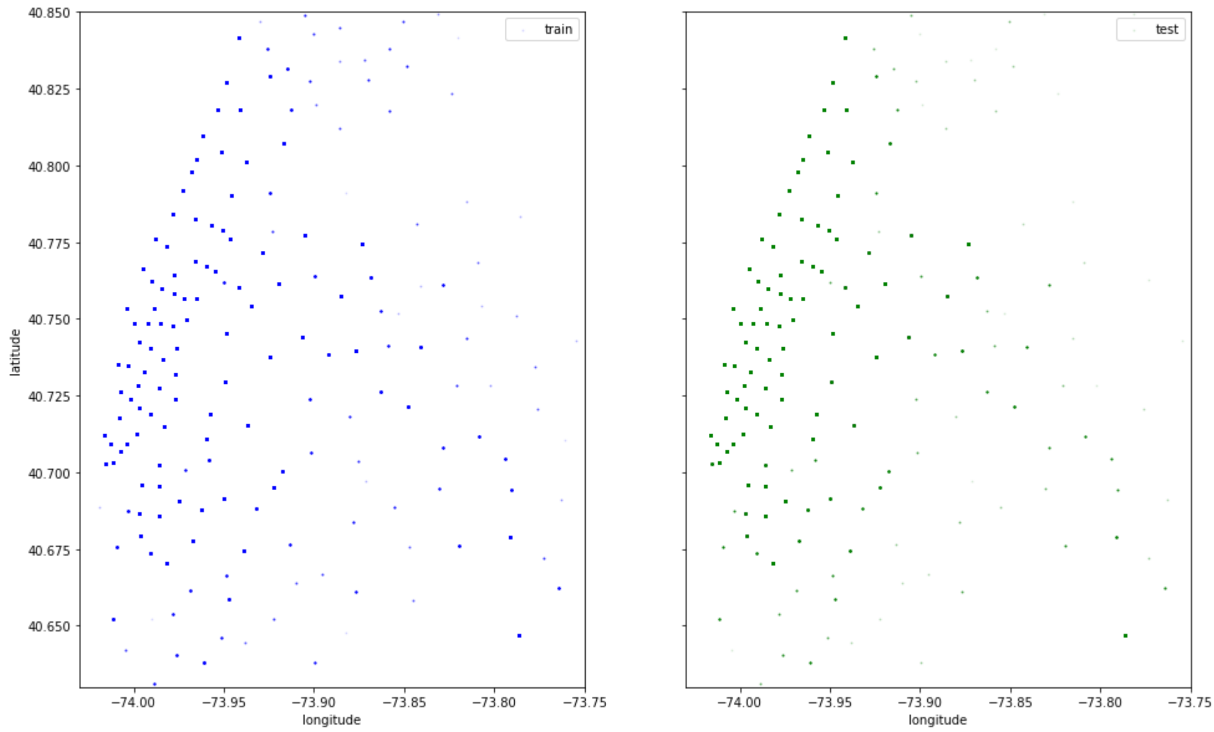
```
plt.plot(train.groupby('dropoff_date').count()
[['tpep_dropoff_datetime']], 'o-', label='train')
plt.plot(test.groupby('dropoff_date').count()
[['tpep_dropoff_datetime']], 'o-', label='test')
plt.title('Dropoff counts by date in train and test sets')
plt.legend(loc=0)
plt.ylabel('number of records')
plt.show()
```



[0]

```
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)
N = 10000
fig, ax = plt.subplots(ncols=2, sharex=True, sharey=True)
ax[0].scatter(train['pickup_longitude'].values,
train['pickup_latitude'].values,
              color='blue', s=1, label='train', alpha=0.1)
ax[1].scatter(test['pickup_longitude'].values,
test['pickup_latitude'].values,
              color='green', s=1, label='test', alpha=0.1)
fig.suptitle('Train and test area complete overlap.')
ax[0].legend(loc=0)
ax[0].set_ylabel('latitude')
ax[0].set_xlabel('longitude')
ax[1].set_xlabel('longitude')
ax[1].legend(loc=0)
plt.ylim(city_lat_border)
plt.xlim(city_long_border)
plt.show()
```

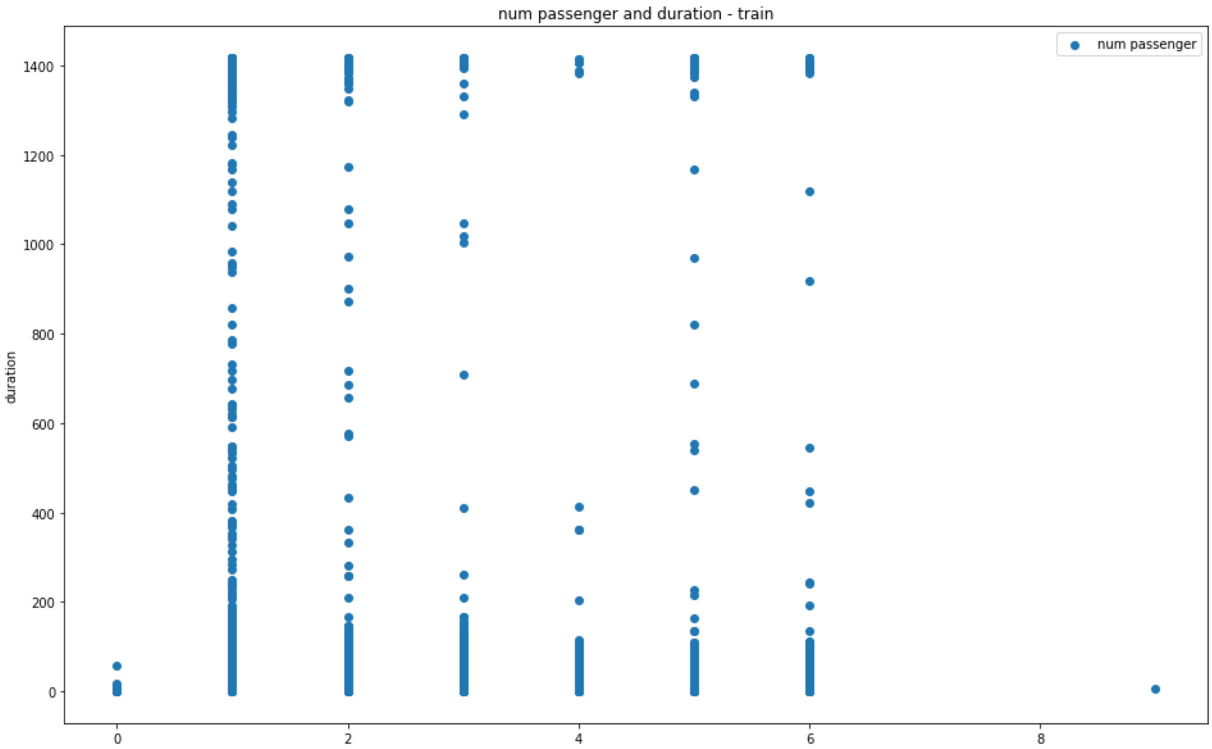
Train and test area complete overlap.



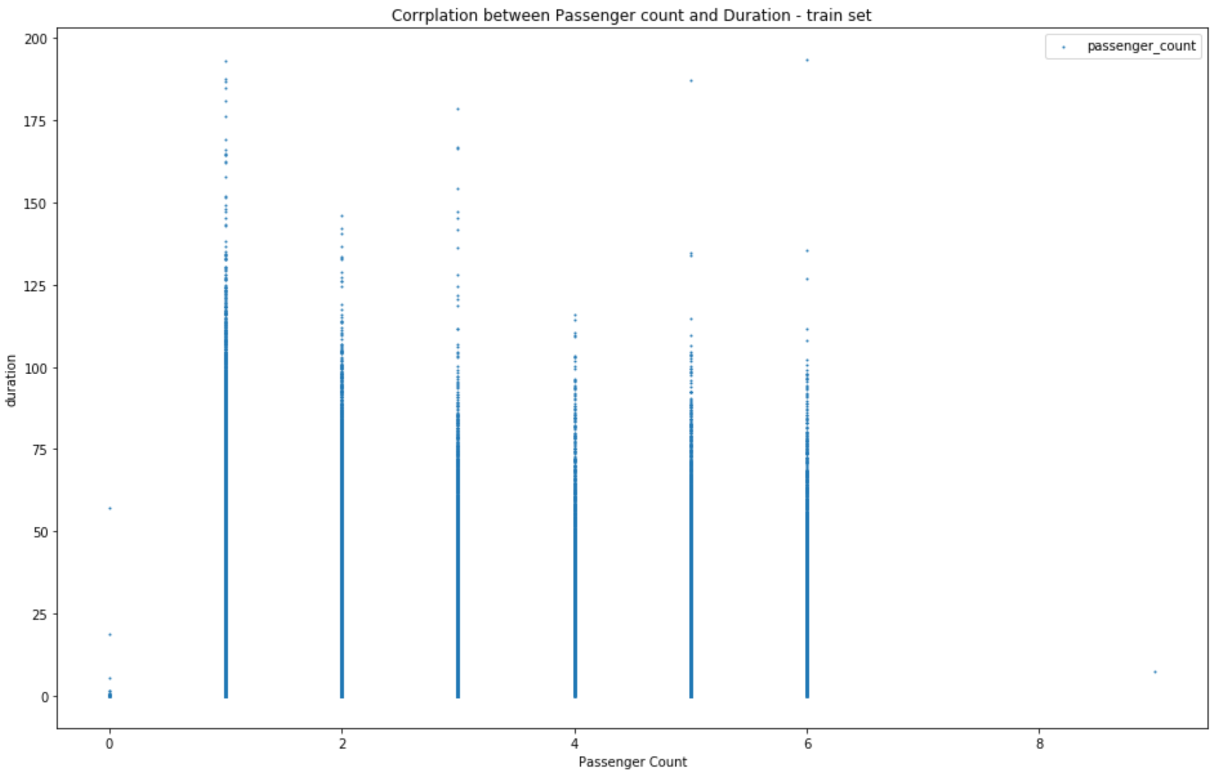
correaltion

[0]

```
def plot_corr(col, label, title):
  plt.scatter(col, train['duration'],label=label)

  plt.title(title)
  plt.legend(loc=0)
  plt.ylabel('duration')
  plt.show()

plot_corr(train['passenger_count'], 'num passenger', 'num
passenger and duration - train')
```

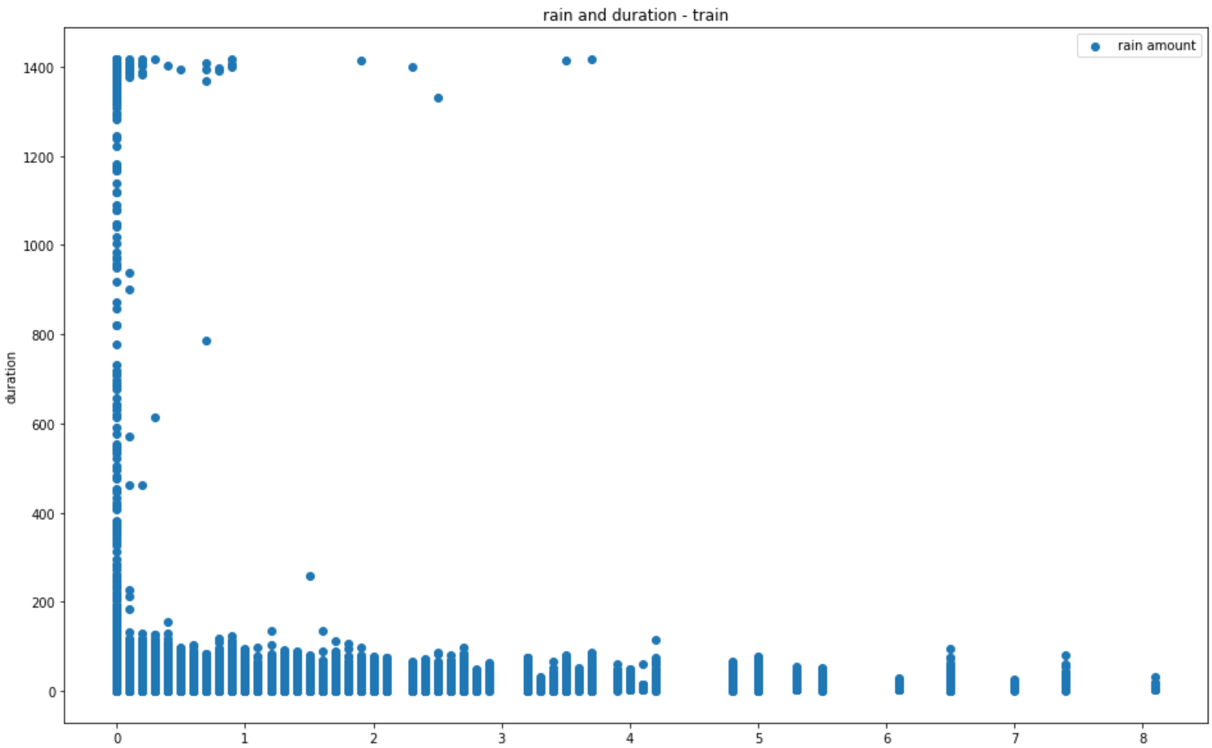



```
[27]  train_sub = train[train['duration'] < 200]
      plt.scatter(train_sub['passenger_count'],
      train_sub['duration'],label='passenger_count', s=1)

      plt.title('Corrplation between Passenger count and Duration -
      train set')
      plt.legend(loc=0)
      plt.ylabel('duration')
      plt.xlabel('Passenger Count')
      plt.show()
```



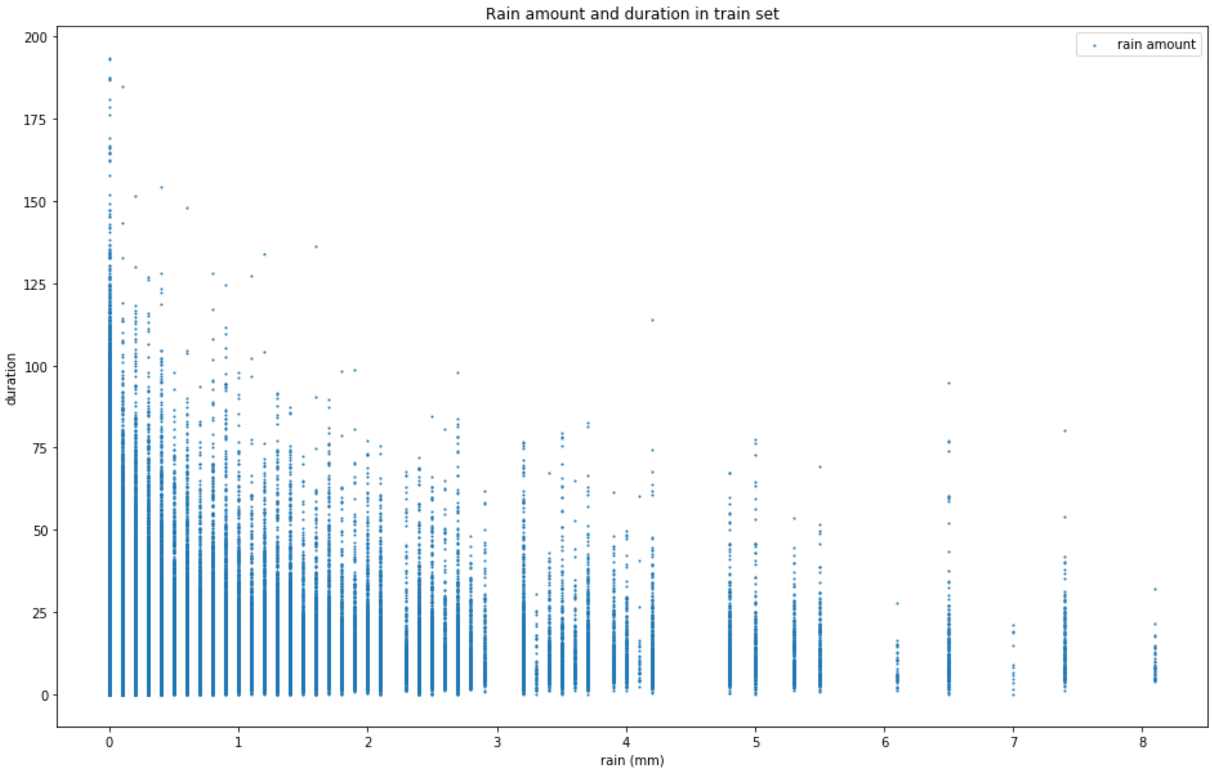
[0]

```
plot_corr(train['rain'], 'rain amount', 'rain and duration -
train')
```

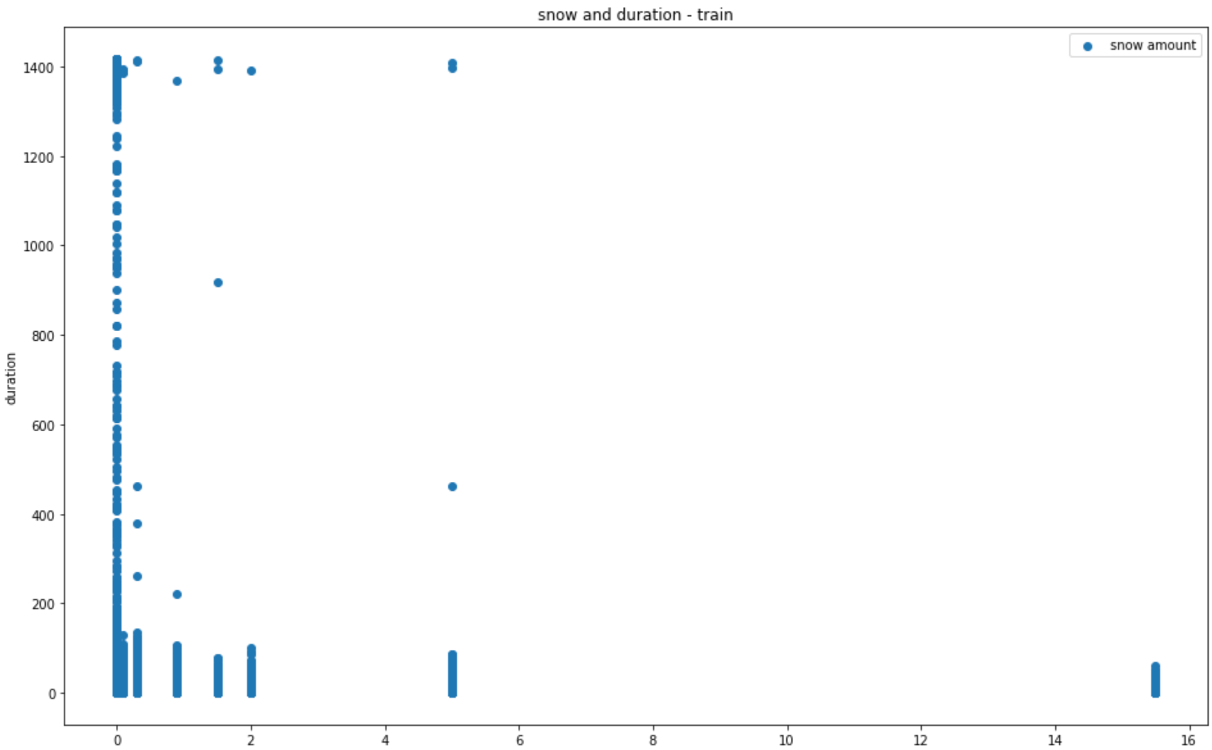


[29]

```
train_sub = train[train['duration'] < 200]
plt.scatter(train_sub['rain'], train_sub['duration'],label='rain
amount', s=1)

plt.title('Rain amount and duration in train set')
plt.legend(loc=0)
plt.ylabel('duration')
plt.xlabel('rain (mm)')
plt.show()
```

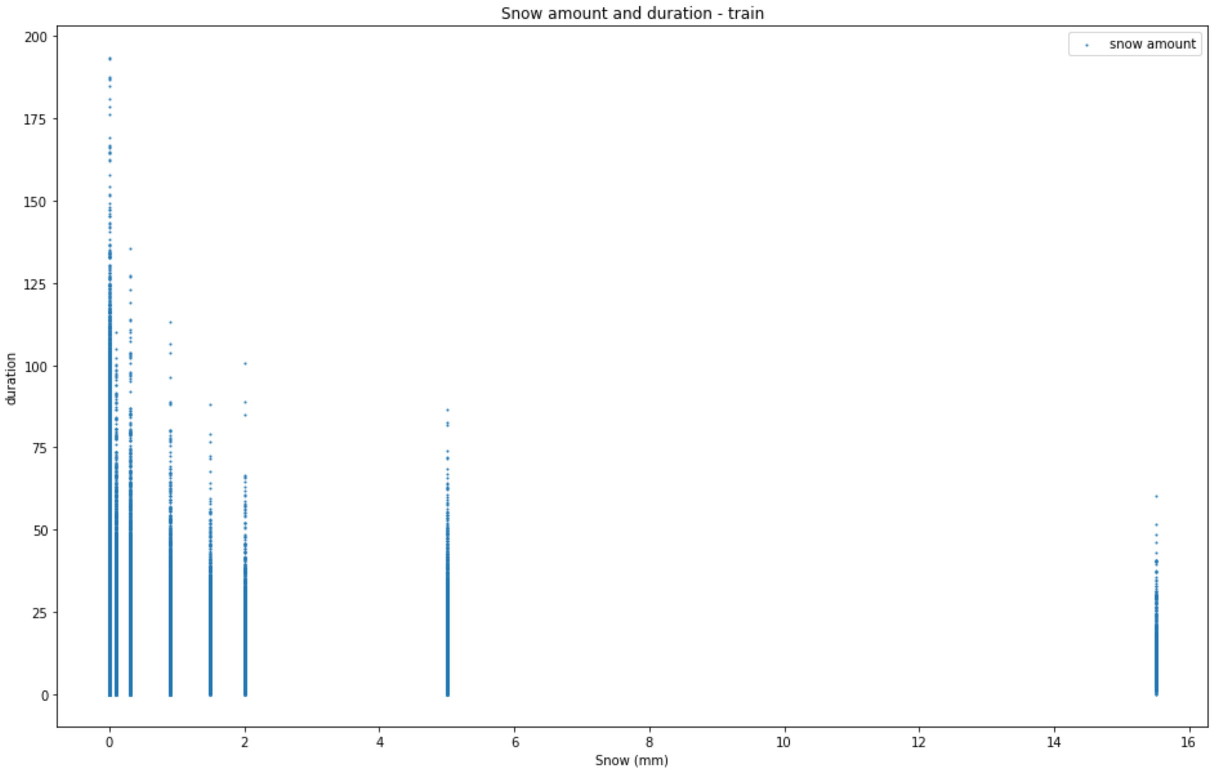


[0]
```
plot_corr(train['snow'], 'snow amount', 'snow and duration -
train')
```



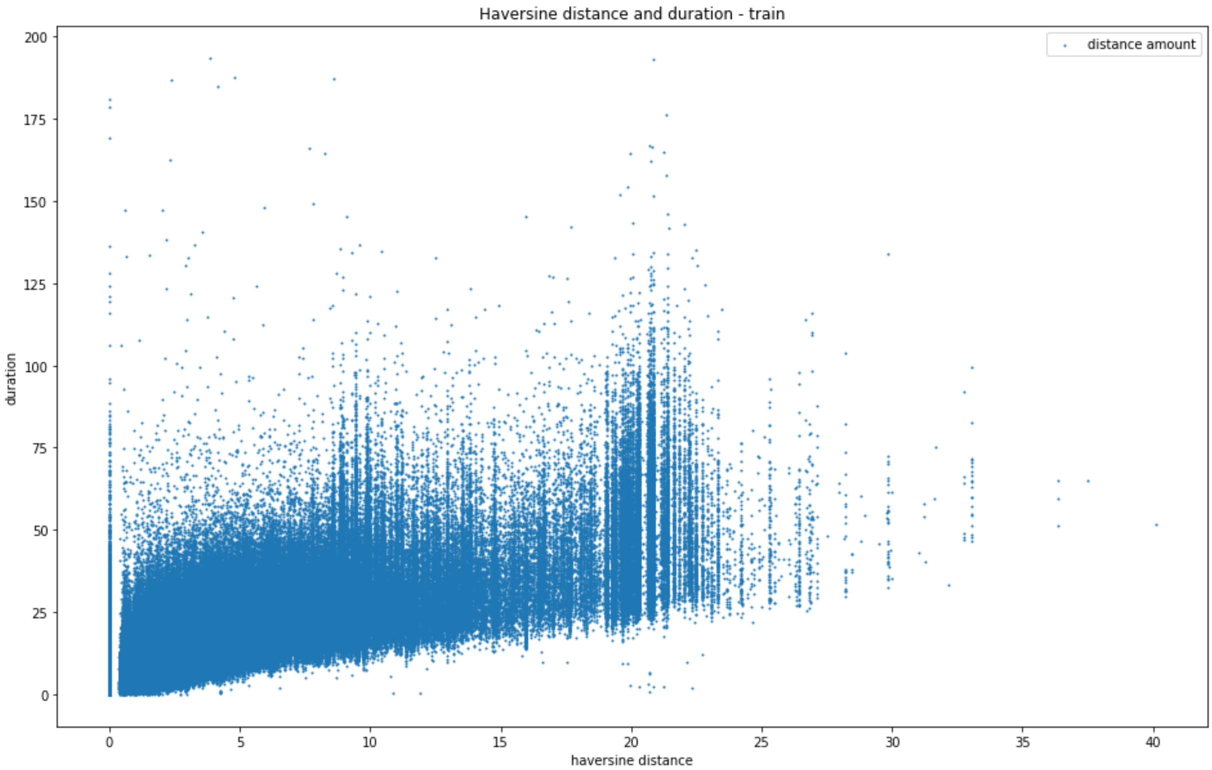
[30]
```
train_sub = train[train['duration'] < 200]
plt.scatter(train_sub['snow'], train_sub['duration'],label='snow
amount', s=1)

plt.title('Snow amount and duration - train')
```

```
plt.legend(loc=0)
plt.ylabel('duration')
plt.xlabel('Snow (mm)')
plt.show()
```



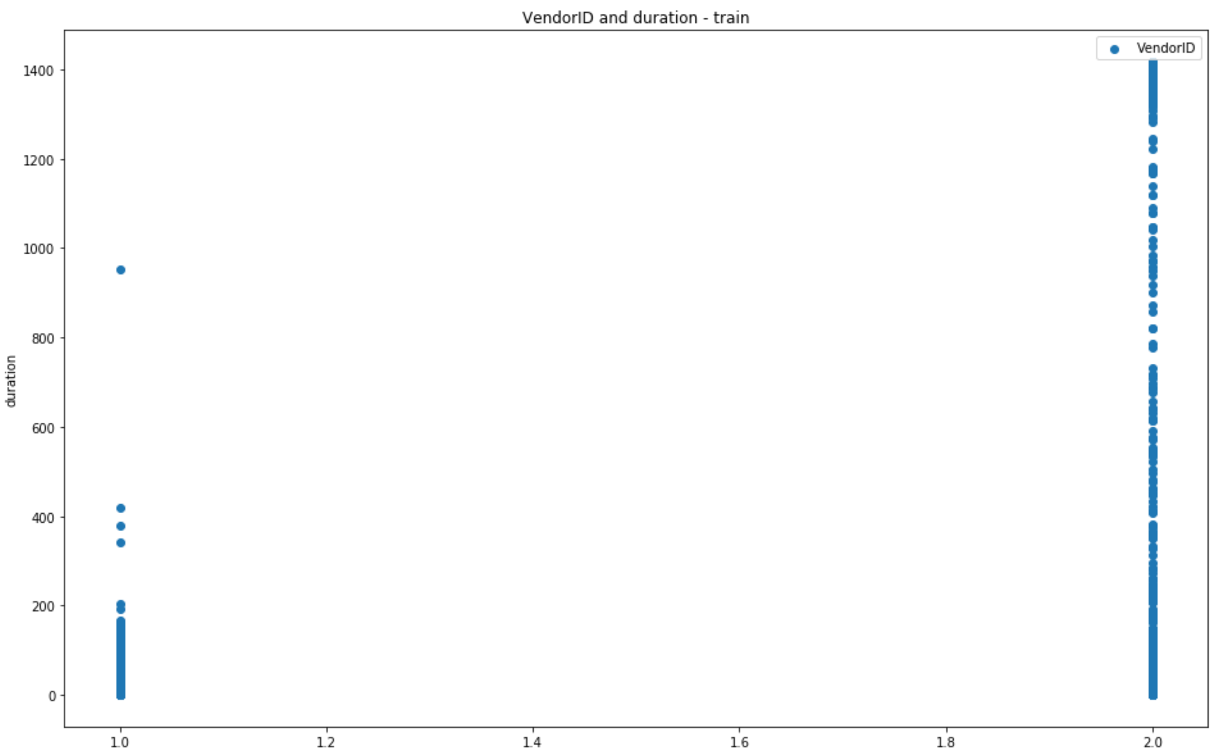
[31]

```
train_sub = train[train['duration'] < 200]
plt.scatter(train_sub['distance_haversine'],
train_sub['duration'],label='distance amount', s=1)

plt.title('Haversine distance and duration - train')
plt.legend(loc=0)
plt.ylabel('duration')
plt.xlabel('haversine distance')
plt.show()
```



[0]

```
plot_corr(train['VendorID'], 'VendorID', 'VendorID and duration - train')
```



[0]

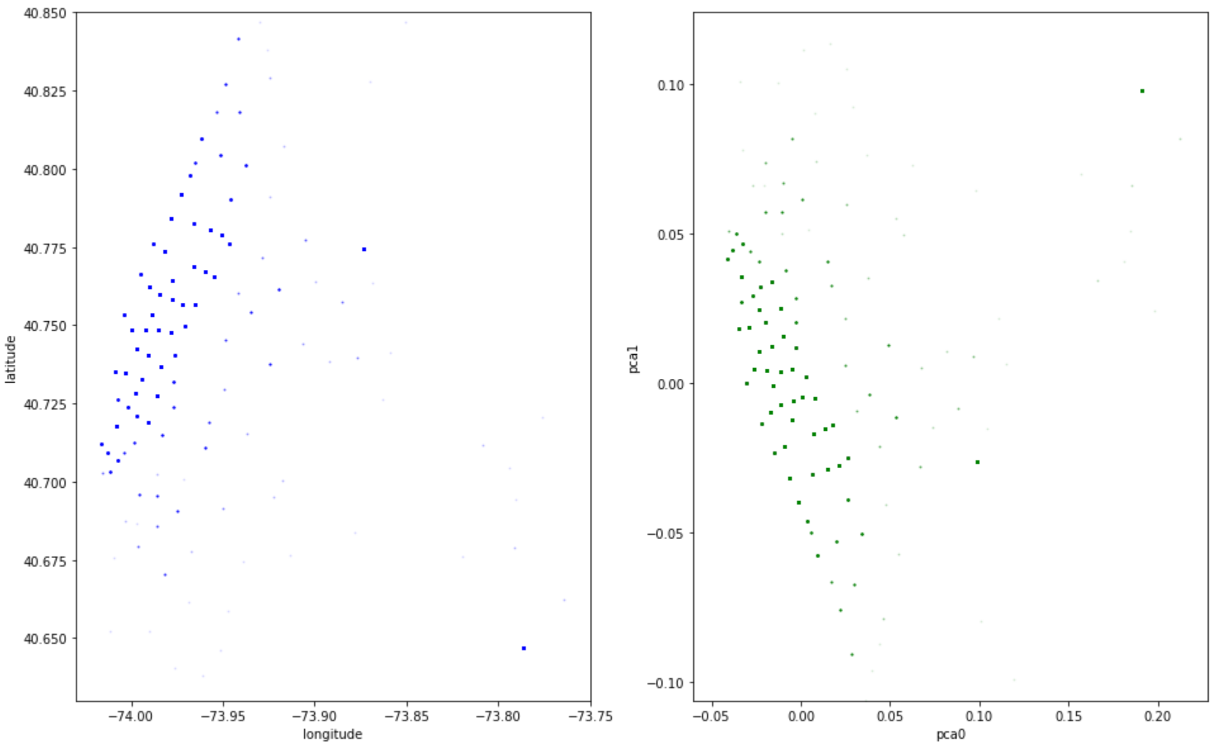
PCA

```
[0]  coords = np.vstack((train[['pickup_latitude',
     'pickup_longitude']].values,
                         train[['dropoff_latitude',
     'dropoff_longitude']].values,
                         test[['pickup_latitude',
     'pickup_longitude']].values,
                         test[['dropoff_latitude',
     'dropoff_longitude']].values))

     pca = PCA().fit(coords)
     train['pickup_pca0'] = pca.transform(train[['pickup_latitude',
     'pickup_longitude']])[:, 0]
     train['pickup_pca1'] = pca.transform(train[['pickup_latitude',
     'pickup_longitude']])[:, 1]
     train['dropoff_pca0'] = pca.transform(train[['dropoff_latitude',
     'dropoff_longitude']])[:, 0]
     train['dropoff_pca1'] = pca.transform(train[['dropoff_latitude',
     'dropoff_longitude']])[:, 1]
     test['pickup_pca0'] = pca.transform(test[['pickup_latitude',
     'pickup_longitude']])[:, 0]
     test['pickup_pca1'] = pca.transform(test[['pickup_latitude',
     'pickup_longitude']])[:, 1]
     test['dropoff_pca0'] = pca.transform(test[['dropoff_latitude',
     'dropoff_longitude']])[:, 0]
     test['dropoff_pca1'] = pca.transform(test[['dropoff_latitude',
     'dropoff_longitude']])[:, 1]
```

```
[0]  fig, ax = plt.subplots(ncols=2)
     ax[0].scatter(train['pickup_longitude'].values[:N],
     train['pickup_latitude'].values[:N],
                   color='blue', s=1, alpha=0.1)
     ax[1].scatter(train['pickup_pca0'].values[:N],
     train['pickup_pca1'].values[:N],
                   color='green', s=1, alpha=0.1)
     fig.suptitle('Pickup lat long coords and PCA transformed
     coords.')
     ax[0].set_ylabel('latitude')
     ax[0].set_xlabel('longitude')
     ax[1].set_xlabel('pca0')
     ax[1].set_ylabel('pca1')
     ax[0].set_xlim(city_long_border)
     ax[0].set_ylim(city_lat_border)
     pca_borders = pca.transform([[x, y] for x in city_lat_border for
     y in city_long_border])
```

```
ax[1].set_xlim(pca_borders[:, 0].min(), pca_borders[:, 0].max())
ax[1].set_ylim(pca_borders[:, 1].min(), pca_borders[:, 1].max())
plt.show()
```

Pickup lat long coords and PCA transformed coords.



distance

[0]

```
def haversine_array(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2,
lng2))
    AVG_EARTH_RADIUS = 6371  # in km
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) *
np.sin(lng * 0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

def dummy_manhattan_distance(lat1, lng1, lat2, lng2):
    a = haversine_array(lat1, lng1, lat1, lng2)
    b = haversine_array(lat1, lng1, lat2, lng1)
    return a + b

def bearing_array(lat1, lng1, lat2, lng2):
    AVG_EARTH_RADIUS = 6371  # in km
```

```
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2,
lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2)
* np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))
```

[0]

```
# train.loc[:, 'distance_haversine'] =
haversine_array(train['pickup_latitude'].values,
train['pickup_longitude'].values,
train['dropoff_latitude'].values,
train['dropoff_longitude'].values)
train.loc[:, 'distance_dummy_manhattan'] =
dummy_manhattan_distance(train['pickup_latitude'].values,
train['pickup_longitude'].values,
train['dropoff_latitude'].values,
train['dropoff_longitude'].values)
train.loc[:, 'direction'] =
bearing_array(train['pickup_latitude'].values,
train['pickup_longitude'].values,
train['dropoff_latitude'].values,
train['dropoff_longitude'].values)
train.loc[:, 'pca_manhattan'] = np.abs(train['dropoff_pca1'] -
train['pickup_pca1']) + np.abs(train['dropoff_pca0'] -
train['pickup_pca0'])

# test.loc[:, 'distance_haversine'] =
haversine_array(test['pickup_latitude'].values,
test['pickup_longitude'].values, test['dropoff_latitude'].values,
test['dropoff_longitude'].values)
test.loc[:, 'distance_dummy_manhattan'] =
dummy_manhattan_distance(test['pickup_latitude'].values,
test['pickup_longitude'].values, test['dropoff_latitude'].values,
test['dropoff_longitude'].values)
test.loc[:, 'direction'] =
bearing_array(test['pickup_latitude'].values,
test['pickup_longitude'].values, test['dropoff_latitude'].values,
test['dropoff_longitude'].values)
test.loc[:, 'pca_manhattan'] = np.abs(test['dropoff_pca1'] -
test['pickup_pca1']) + np.abs(test['dropoff_pca0'] -
test['pickup_pca0'])

train.loc[:, 'center_latitude'] =
(train['pickup_latitude'].values +
train['dropoff_latitude'].values) / 2
train.loc[:, 'center_longitude'] =
(train['pickup_longitude'].values +
train['dropoff_longitude'].values) / 2
test.loc[:, 'center_latitude'] = (test['pickup_latitude'].values
+ test['dropoff_latitude'].values) / 2
```

```
test.loc[:, 'center_longitude'] =
(test['pickup_longitude'].values +
test['dropoff_longitude'].values) / 2
```

time

```
[0]
train.loc[:, 'pickup_weekday'] =
train['tpep_pickup_datetime'].dt.weekday
train.loc[:, 'pickup_hour_weekofyear'] =
train['tpep_pickup_datetime'].dt.weekofyear
train.loc[:, 'pickup_hour'] =
train['tpep_pickup_datetime'].dt.hour
train.loc[:, 'pickup_minute'] =
train['tpep_pickup_datetime'].dt.minute
train.loc[:, 'pickup_dt'] = (train['tpep_pickup_datetime'] -
train['tpep_pickup_datetime'].min()).dt.total_seconds()
train.loc[:, 'pickup_week_hour'] = train['pickup_weekday'] * 24 +
train['pickup_hour']

test.loc[:, 'pickup_weekday'] =
test['tpep_pickup_datetime'].dt.weekday
test.loc[:, 'pickup_hour_weekofyear'] =
test['tpep_pickup_datetime'].dt.weekofyear
test.loc[:, 'pickup_hour'] = test['tpep_pickup_datetime'].dt.hour
test.loc[:, 'pickup_minute'] =
test['tpep_pickup_datetime'].dt.minute
test.loc[:, 'pickup_dt'] = (test['tpep_pickup_datetime'] -
train['tpep_pickup_datetime'].min()).dt.total_seconds()
test.loc[:, 'pickup_week_hour'] = test['pickup_weekday'] * 24 +
test['pickup_hour']
```

```
[0]
X['pickup_dt']
```

```
35515473     9537529.0
61924539    16582639.0
46134238    11564679.0
1005434      1361251.0
31561727     8328406.0
                ...
36893916     9873813.0
11468716     3057299.0
18258392     4868125.0
3829424       316606.0
```

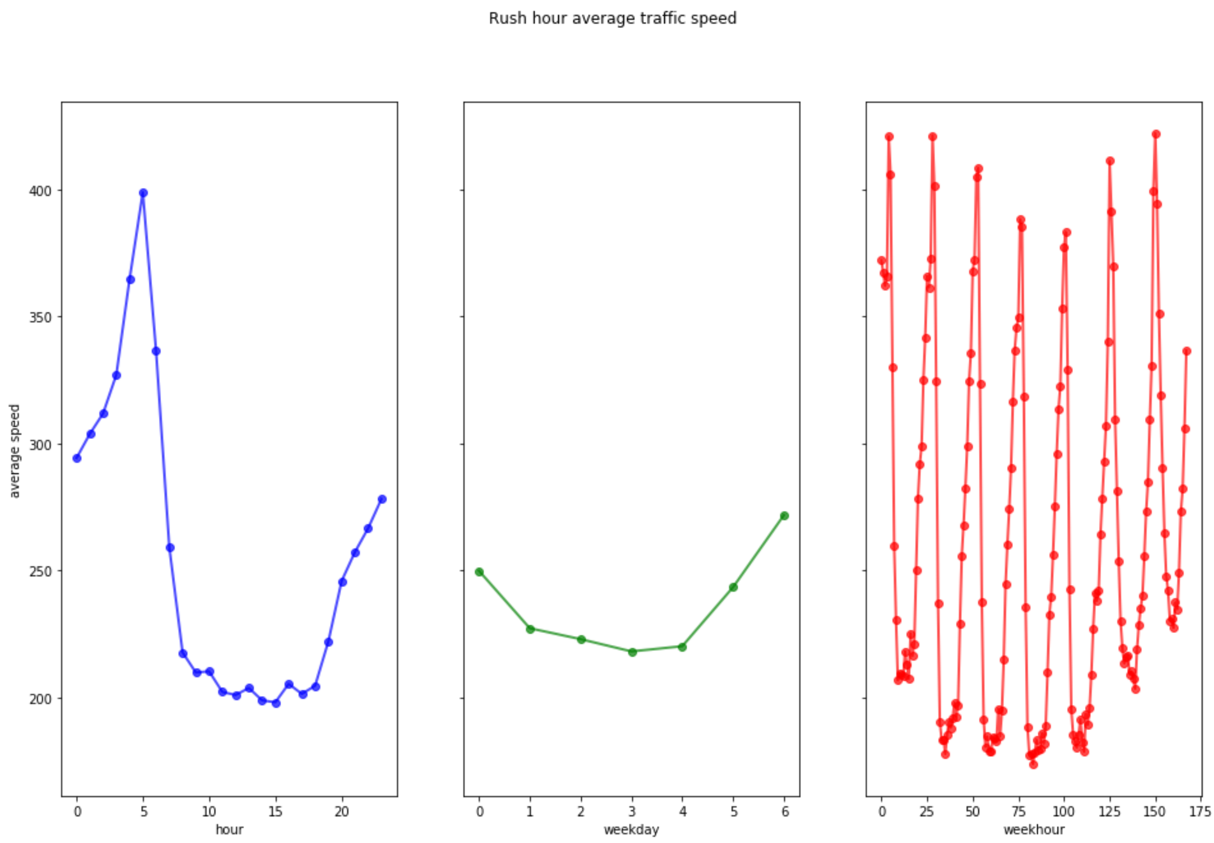
```
12914768     3625735.0
Name: pickup_dt, Length: 785020, dtype: float64
```

speed

```
[0]  train.columns
```

```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'PULocationID', 'DOLocationID',
'payment_type',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'snow', 'rain', 'isHoliday',
'distance_haversine',
       'pickup_date', 'dropoff_date', 'duration', 'log_duration',
       'pickup_pca0', 'pickup_pca1', 'dropoff_pca0', 'dropoff_pca1',
       'center_latitude', 'center_longitude', 'pickup_weekday',
       'pickup_hour_weekofyear', 'pickup_hour', 'pickup_minute',
'pickup_dt',
       'pickup_week_hour', 'avg_speed_h', 'distance_dummy_manhattan',
       'direction', 'pca_manhattan'],
      dtype='object')
```

```
[0]  train.loc[:, 'avg_speed_h'] = 1000 * train['distance_haversine']
     / train['duration']
     train.loc[:, 'avg_speed_m'] = 1000 *
     train['distance_dummy_manhattan'] / train['duration']
     fig, ax = plt.subplots(ncols=3, sharey=True)
     ax[0].plot(train.groupby('pickup_hour').mean()['avg_speed_h'],
     'bo-', lw=2, alpha=0.7)
     ax[1].plot(train.groupby('pickup_weekday').mean()['avg_speed_h'],
     'go-', lw=2, alpha=0.7)
     ax[2].plot(train.groupby('pickup_week_hour').mean()
     ['avg_speed_h'], 'ro-', lw=2, alpha=0.7)
     ax[0].set_xlabel('hour')
     ax[1].set_xlabel('weekday')
     ax[2].set_xlabel('weekhour')
     ax[0].set_ylabel('average speed')
     fig.suptitle('Rush hour average traffic speed')
     plt.show()
```

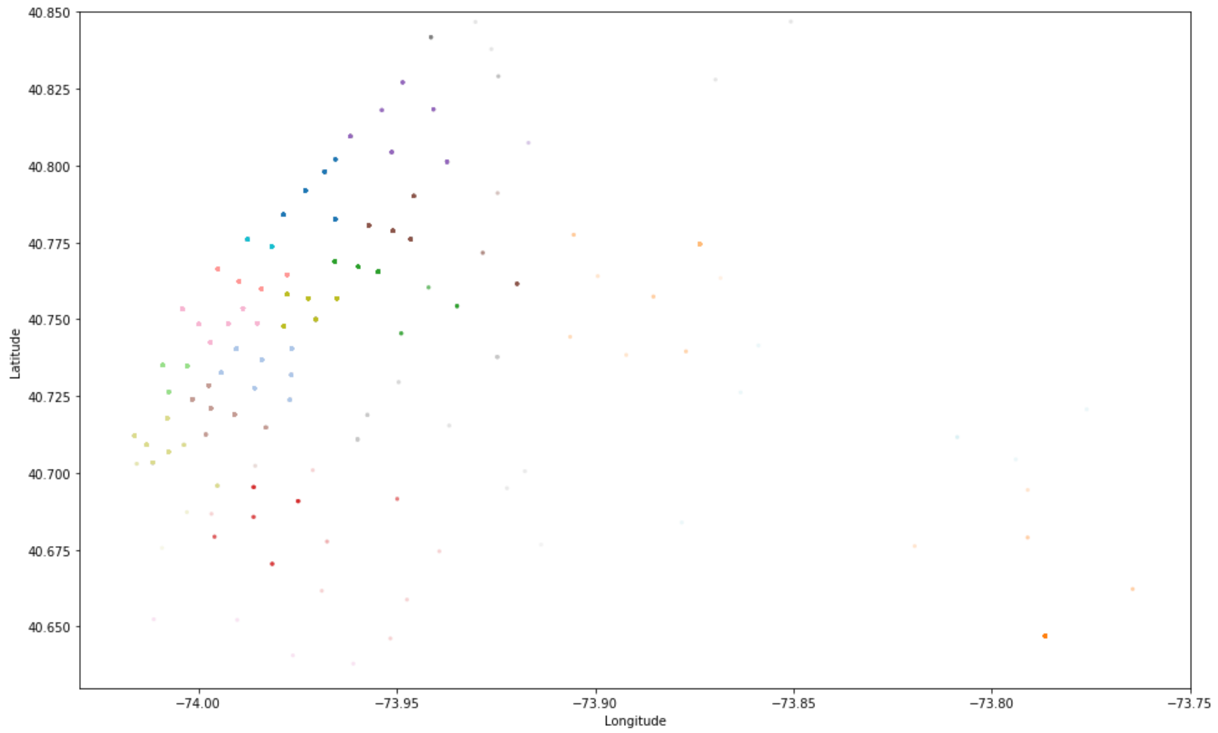



```
[0]   t0 = dt.datetime.now()
      sample_ind = np.random.permutation(len(coords))[:500000]
      kmeans = MiniBatchKMeans(n_clusters=20,
      batch_size=10000).fit(coords[sample_ind])

      train.loc[:, 'pickup_cluster'] =
      kmeans.predict(train[['pickup_latitude', 'pickup_longitude']])
      train.loc[:, 'dropoff_cluster'] =
      kmeans.predict(train[['dropoff_latitude', 'dropoff_longitude']])
      test.loc[:, 'pickup_cluster'] =
      kmeans.predict(test[['pickup_latitude', 'pickup_longitude']])
      test.loc[:, 'dropoff_cluster'] =
      kmeans.predict(test[['dropoff_latitude', 'dropoff_longitude']])
      t1 = dt.datetime.now()
      print('Time till clustering: %i seconds' % (t1 - t0).seconds)
```

```
Time till clustering: 2 seconds
```

```
[0]   fig, ax = plt.subplots(ncols=1, nrows=1)
      ax.scatter(train.pickup_longitude.values[:N],
      train.pickup_latitude.values[:N], s=10, lw=0,
                 c=train.pickup_cluster[:N].values, cmap='tab20',
      alpha=0.2)
      ax.set_xlim(city_long_border)
      ax.set_ylim(city_lat_border)
      ax.set_xlabel('Longitude')
      ax.set_ylabel('Latitude')
```

```
plt.show()
```



[0]

```
!pip install mapboxgl
from mapboxgl.utils import *
from mapboxgl.viz import *
%matplotlib inline

from datetime import timedelta
import datetime as dt
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [16, 10]
```

```
Requirement already satisfied: mapboxgl in
/usr/local/lib/python3.6/dist-packages (0.10.2)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.6/dist-packages (from mapboxgl) (3.1.1)
Requirement already satisfied: geojson in /usr/local/lib/python3.6/dist-
packages (from mapboxgl) (2.5.0)
Requirement already satisfied: chroma-py in
/usr/local/lib/python3.6/dist-packages (from mapboxgl) (0.1.0.dev1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.6/dist-
packages (from mapboxgl) (2.10.3)
Requirement already satisfied: colour in /usr/local/lib/python3.6/dist-
packages (from mapboxgl) (0.1.5)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(2.6.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(1.1.0)
```

```
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
in /usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(2.4.5)
Requirement already satisfied: numpy>=1.11 in
/usr/local/lib/python3.6/dist-packages (from matplotlib->mapboxgl)
(1.17.4)
Requirement already satisfied: MarkupSafe>=0.23 in
/usr/local/lib/python3.6/dist-packages (from jinja2->mapboxgl) (1.1.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.1-
>matplotlib->mapboxgl) (1.12.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1-
>matplotlib->mapboxgl) (41.6.0)
```

[0]

```
pickup_counts = train['PULocationID'].value_counts().to_dict()
dropoff_counts = train['DOLocationID'].value_counts().to_dict()

def freq(x, pu_flag):
  if pu_flag:
    return pickup_counts[x['PULocationID']]
  else:
    return dropoff_counts[x['DOLocationID']]

train['pickup_loc_count'] = train.apply(lambda x: freq(x, 1),
axis=1)
train['dropoff_loc_count'] = train.apply(lambda x: freq(x, 0),
axis=1)
```

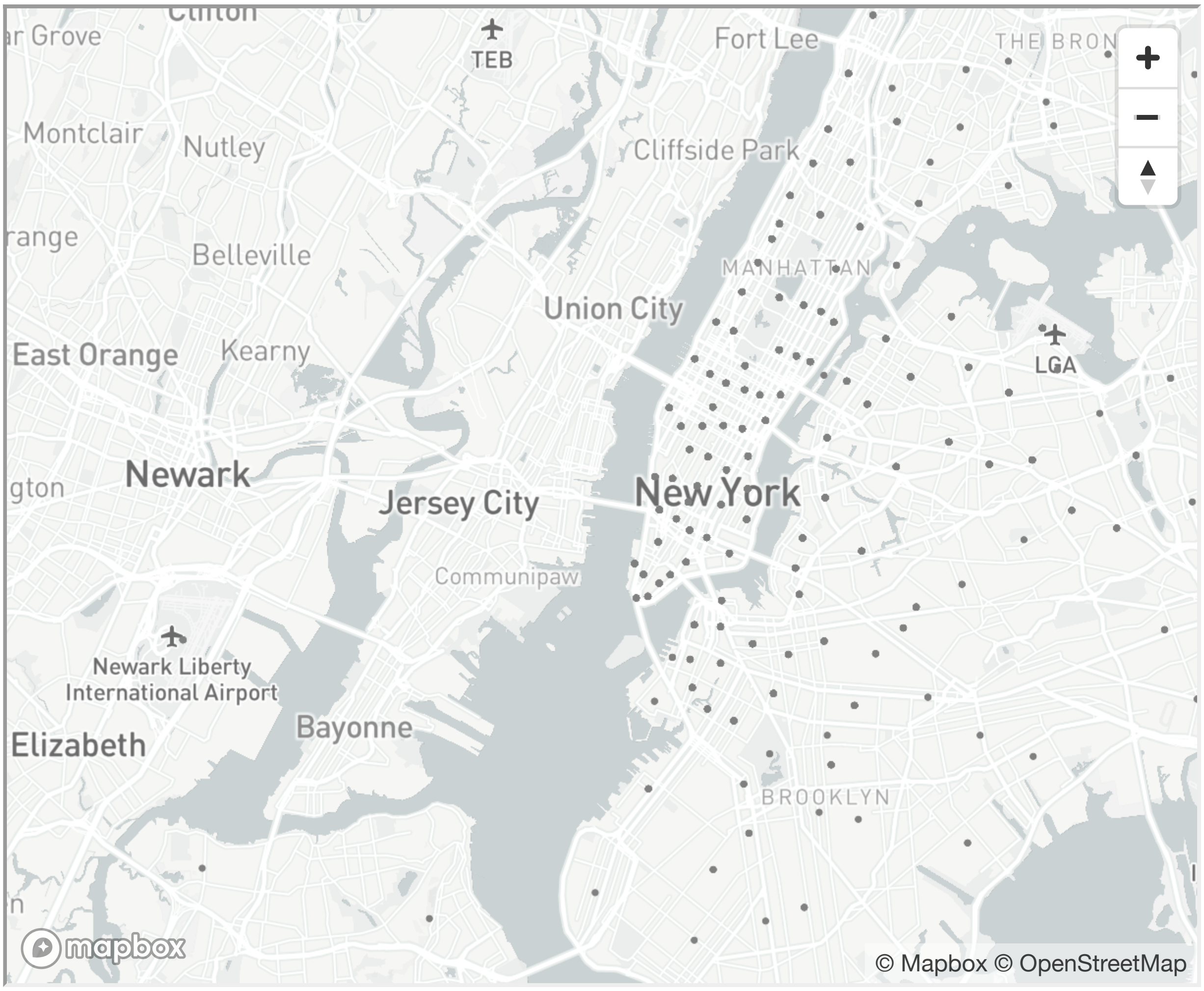
[0]

```
pickup_counts_test =
test['PULocationID'].value_counts().to_dict()
dropoff_counts_test =
test['DOLocationID'].value_counts().to_dict()

def freq_test(x, pu_flag):
  if pu_flag:
    return pickup_counts_test[x['PULocationID']]
  else:
    return dropoff_counts_test[x['DOLocationID']]

test['pickup_loc_count'] = test.apply(lambda x: freq_test(x, 1),
axis=1)
test['dropoff_loc_count'] = test.apply(lambda x: freq_test(x, 0),
axis=1)
```

```
[0]  def visualize_map(data, center, zoom):
         """
         This is a sample method for you to get used to spatial data
     visualization using the Mapboxgl-jupyter library.

         Mapboxgl-jupyter is a location based data visualization
     library for Jupyter Notebooks.
         To better understand this, you may want to read the
     documentation:
         https://mapbox-mapboxgl-jupyter.readthedocs-
     hosted.com/en/latest/

         To use the library, you need to register for a token by
     accessing:
         https://account.mapbox.com/access-tokens/
         You need to create an account and login. Then you can see
     your access token by revisiting the above URL.
         """
         # Create the viz from the dataframe
         viz = CircleViz(data,
                         access_token =
     os.environ['MAPBOX_ACCESS_TOKEN'],
                         center = center,
                         zoom = zoom,
                       )
         # It could take several minutes to show the map
         print("showing map...")
         viz.show();

     # set the center of the map
     center_of_nyc = (-74, 40.73)

     data = df_to_geojson(train.head(50000), lat='pickup_latitude',
     lon='pickup_longitude')
```

```
[0]  # visualize map of New York City
     visualize_map(data=data, center=center_of_nyc, zoom=10)
```

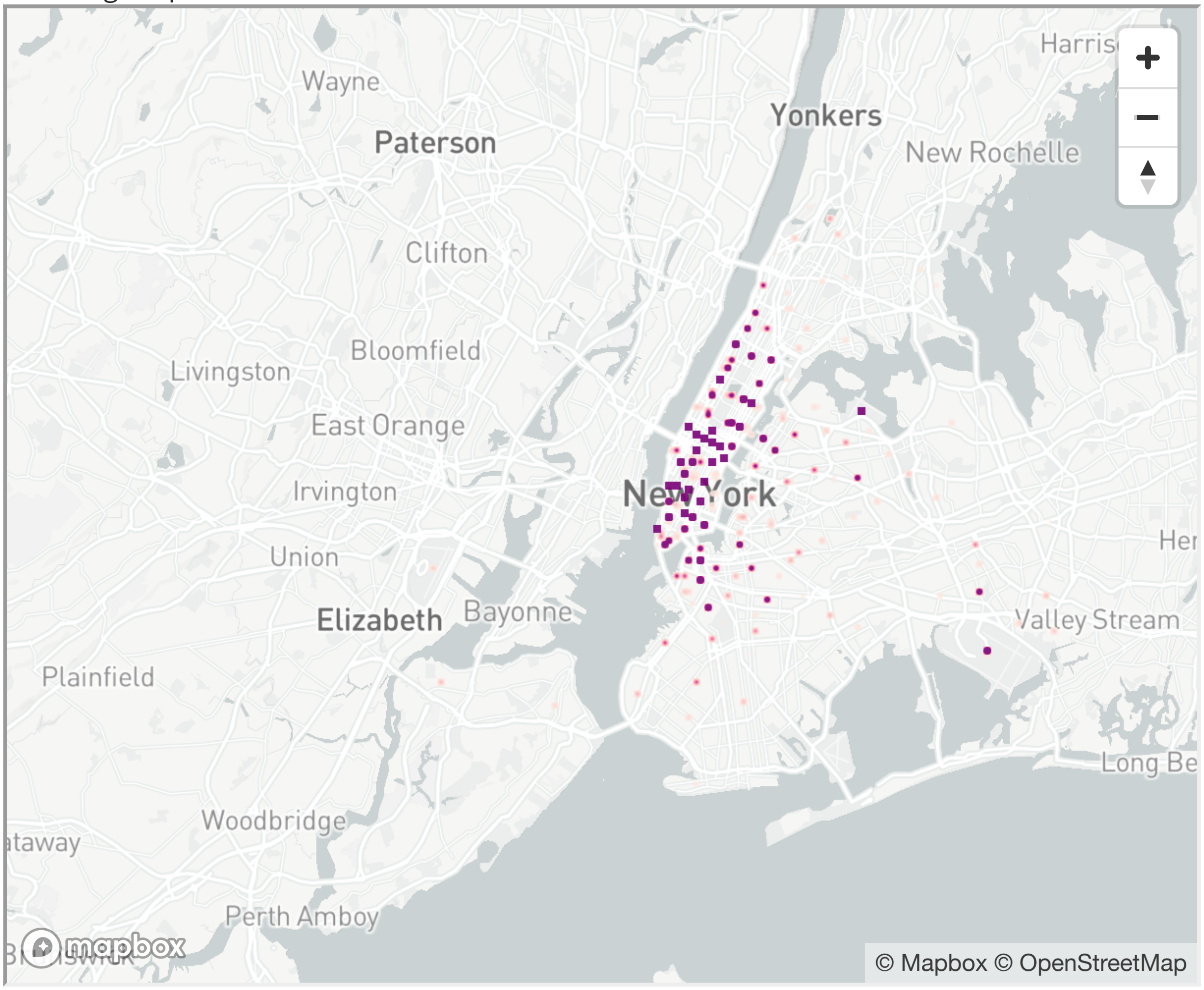
```
showing map...
```




```
[0]   def draw_heatmap(data, center, zoom):
          """
          Method to draw a heat map. You should use this method to
      identify the most popular pickup location in the southeast of
      NYC.

          :param geodata: name of GeoJSON file or object or JSON join-
      data weight_property
          :type geodata: string
          :param center: map center point
          :type center: tuple
          :param zoom: starting zoom level for map
          :type zoom: float
          """
          # set features for the heatmap
          heatmap_color_stops =
      create_color_stops([0.01,0.25,0.5,0.75,1], colors='RdPu')
          heatmap_radius_stops = [[10,1],[20,2]] #increase radius with
      zoom

          # create a heatmap
          viz = HeatmapViz(data,

      access_token=os.environ['MAPBOX_ACCESS_TOKEN'],
```

```
                              color_stops=heatmap_color_stops,
                              radius_stops=heatmap_radius_stops,
                              height='500px',
                              opacity=0.9,
                              center=center,
                              zoom=zoom)
    print("drawing map...")
    viz.show()

draw_heatmap(data=data, center=center_of_nyc, zoom=9)
```

```
drawing map...
```



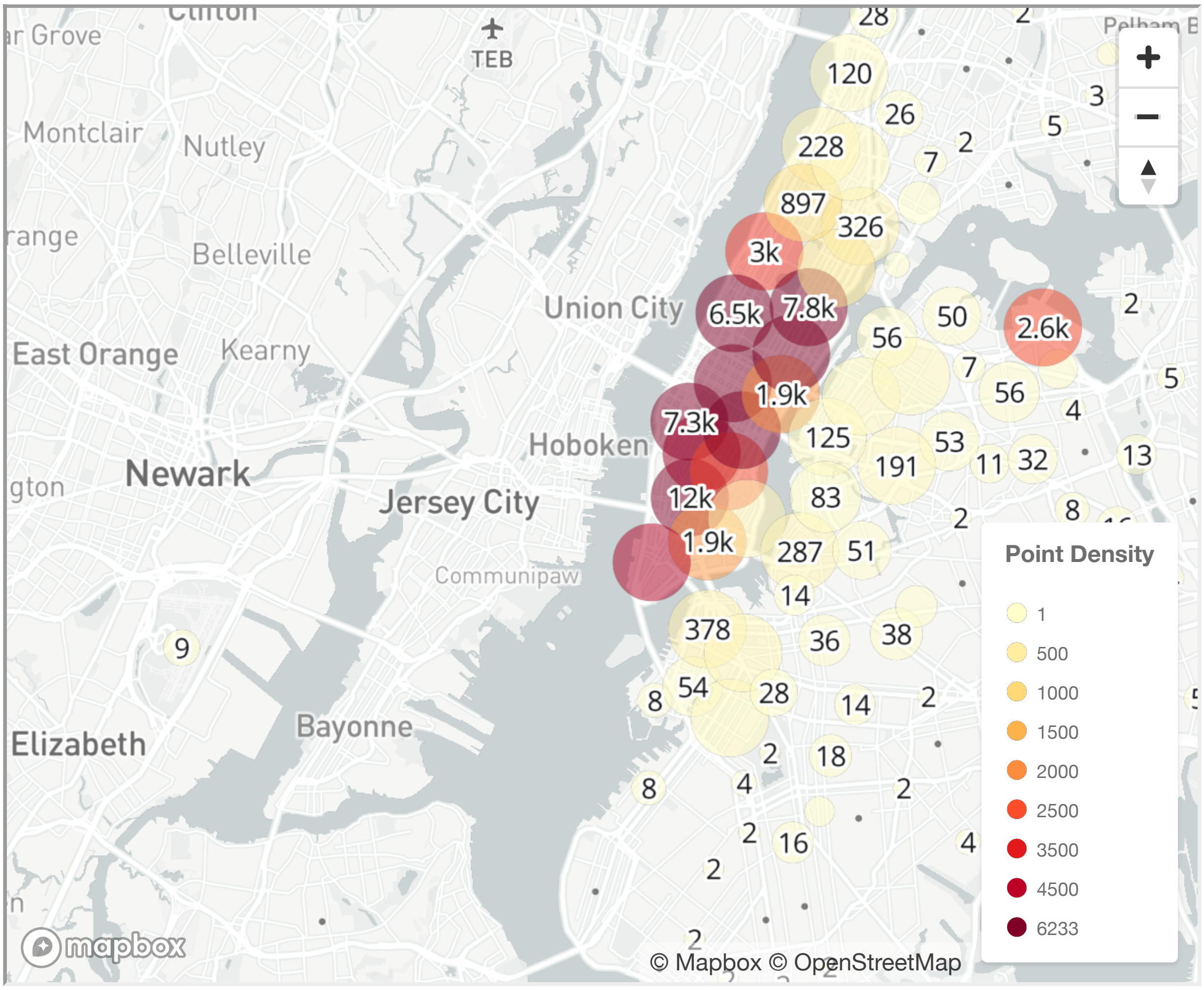

```
[0]  pickup_counts = train['PULocationID'].value_counts().to_dict()
     dropoff_counts = train['DOLocationID'].value_counts().to_dict()

     def freq(x, pu_flag):
       if pu_flag:
         return pickup_counts[x['PULocationID']]
       else:
         return dropoff_counts[x['DOLocationID']]

     train['pickup_loc_count'] = train.apply(lambda x: freq(x, 1),
     axis=1)
```

```
train['dropoff_loc_count'] = train.apply(lambda x: freq(x, 0),
axis=1)
```

```
[0]  train_sub = train.head(100000)
     pickup_counts =
     train_sub['PULocationID'].value_counts().to_dict()
     dropoff_counts =
     train_sub['DOLocationID'].value_counts().to_dict()

     def freq(x, pu_flag):
       if pu_flag:
         return pickup_counts[x['PULocationID']]
       else:
         return dropoff_counts[x['DOLocationID']]

     pu_freq = []
     for p in train_sub['PULocationID']:
       pu_freq.append([p, longitude[str(p)], latitude[str(p)]])

     do_freq = []
     for d in train_sub['DOLocationID']:
       do_freq.append([d, longitude[str(d)], latitude[str(d)]])

     # Create the pandas DataFrame
     pickup_df = pd.DataFrame(pu_freq, columns = ['PULocationID',
     'longitude', 'latitude'])
     pickup_df['freq'] = pickup_df.apply(lambda x: freq(x, 1), axis=1)

     dropoff_df = pd.DataFrame(do_freq, columns = ['DOLocationID',
     'longitude', 'latitude'])
     dropoff_df['freq'] = dropoff_df.apply(lambda x: freq(x, 0),
     axis=1)
```

```
[0]  # Create a geojson Feature Collection export from a Pandas
     dataframe
     points = df_to_geojson(pickup_df,
                            properties=['freq'],
                            lat='latitude', lon='longitude')

     #Create a clustered circle map
     color_stops = create_color_stops([1, 500, 1000, 1500, 2000, 2500,
     3500, 4500, 6233], colors='YlOrRd')

     viz = ClusteredCircleViz(points,

     access_token=os.environ['MAPBOX_ACCESS_TOKEN'],
                              color_stops=color_stops,
```

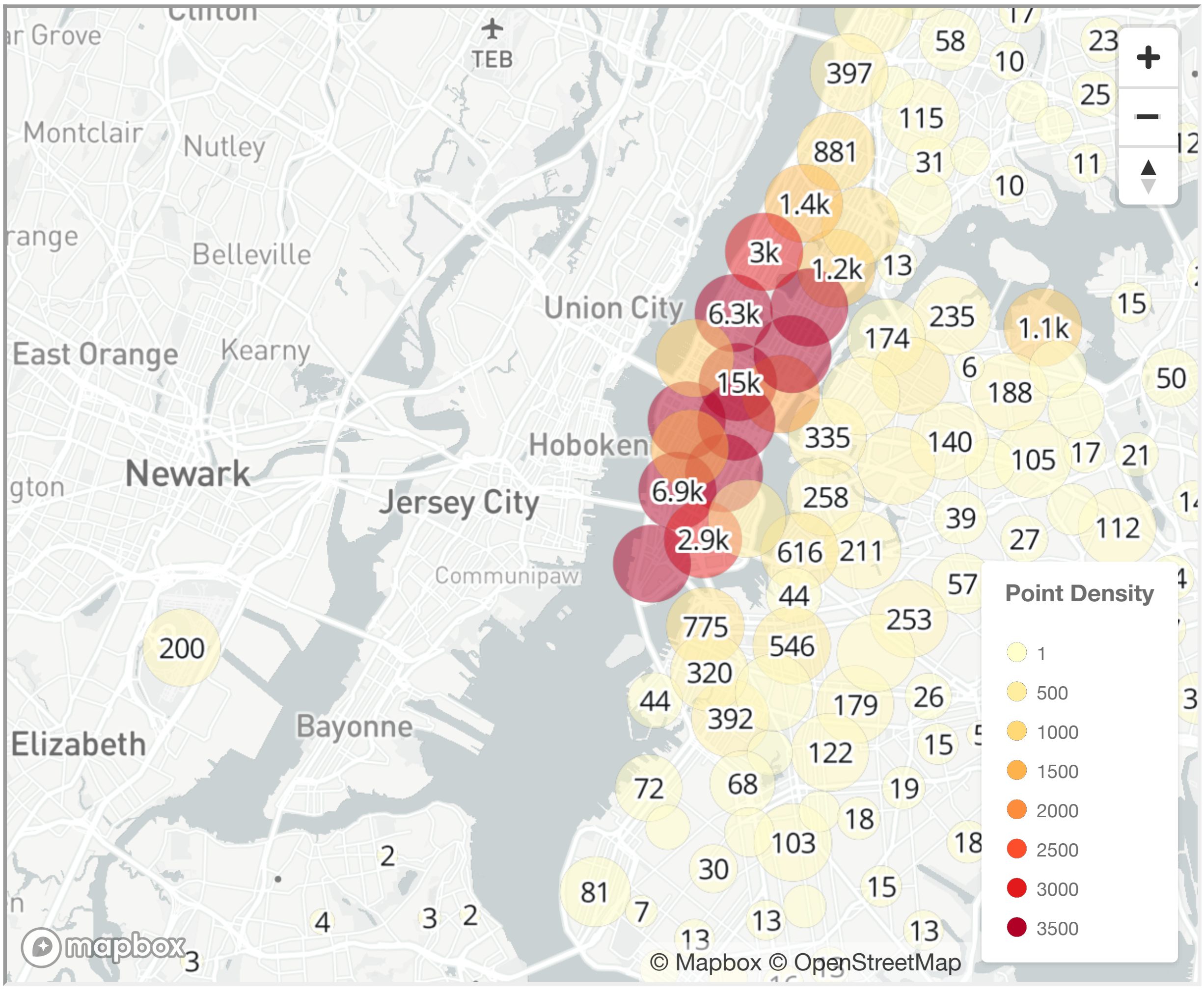
```
                                radius_stops=[[1,5], [10, 10], [50, 15],
[100, 20]],
                                radius_default=1,
                                cluster_maxzoom=20,
                                cluster_radius=20,
                                label_size=12,
                                opacity=0.5,
                                center=center_of_nyc,
                                zoom=10)

viz.show()
```



[0]

```
# Create a geojson Feature Collection export from a Pandas
dataframe
points_do = df_to_geojson(dropoff_df,
                          properties=['freq'],
                          lat='latitude', lon='longitude')

#Create a clustered circle map
color_stops_do = create_color_stops([1, 500, 1000, 1500, 2000,
2500, 3000, 3500], colors='YlOrRd')

viz = ClusteredCircleViz(points_do,
```

```
access_token=os.environ['MAPBOX_ACCESS_TOKEN'],
                          color_stops=color_stops_do,
                          radius_stops=[[1,5], [10, 10], [50, 15],
[100, 20]],
                          radius_default=1,
                          cluster_maxzoom=20,
                          cluster_radius=20,
                          label_size=12,
                          opacity=0.5,
                          center=center_of_nyc,
                          zoom=10)

viz.show()
```




```
weather_df = pd.read_csv('/content/drive/My
Drive/10701/weather.csv', parse_dates=['date_time'])

# pd.Timestamp(2017, 1, 1, 2)
def get_snow(x):
  time = x['tpep_pickup_datetime'].dt
  comp_time = pd.Timestamp(2017, time.month, time.day, time.hour)
  return weather_df[weather_df['date_time'] == comp_time]
['totalSnow_cm'].iloc[0]
```

```
# pd.Timestamp(2017, 1, 1, 2)
def get_rain(x):
  time = x['tpep_pickup_datetime'].dt
  comp_time = pd.Timestamp(2017, time.month, time.day, time.hour)
  return weather_df[weather_df['date_time'] == comp_time]
['precipMM'].iloc[0]

data['snow'] = data.apply(lambda x: get_snow(x), axis=1)
data['rain'] = data.apply(lambda x: get_rain(x), axis=1)
```

xgboost

[0]

```
train.columns
```

```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'PULocationID', 'DOLocationID',
'payment_type',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'snow', 'rain', 'isHoliday',
'distance_haversine',
       'pickup_date', 'dropoff_date', 'duration', 'log_duration',
       'pickup_pca0', 'pickup_pca1', 'dropoff_pca0', 'dropoff_pca1',
       'center_latitude', 'center_longitude', 'pickup_weekday',
       'pickup_hour_weekofyear', 'pickup_hour', 'pickup_minute',
'pickup_dt',
       'pickup_week_hour', 'avg_speed_h', 'distance_dummy_manhattan',
       'direction', 'pca_manhattan', 'avg_speed_m', 'pickup_cluster',
       'dropoff_cluster', 'pickup_loc_count', 'dropoff_loc_count'],
      dtype='object')
```

[0]

```
# Remove fields that we do not want to train with
X = train.drop(['duration',
                'tpep_pickup_datetime' ,
                'tpep_pickup_datetime',
                'tpep_dropoff_datetime',
                'tpep_dropoff_datetime',
                'payment_type',
                'pickup_date' ,
                'dropoff_date',
                'pickup_pca0',
                'pickup_pca1',
                'dropoff_pca0',
                'dropoff_pca1',
```

```
                    'avg_speed_h',
                    'dropoff_cluster',
                     'pickup_cluster',
                    'avg_speed_m',
                    'log_duration',
                    'center_latitude',
                    'center_longitude',
                    'pca_manhattan',
                     'pickup_dt'
                    ], axis=1, errors='ignore')

# Extract the value you want to predict
Y = train['duration']
print('Shape of the feature matrix: {}'.format(X.shape))
```

```
Shape of the feature matrix: (785020, 21)
```

[0]

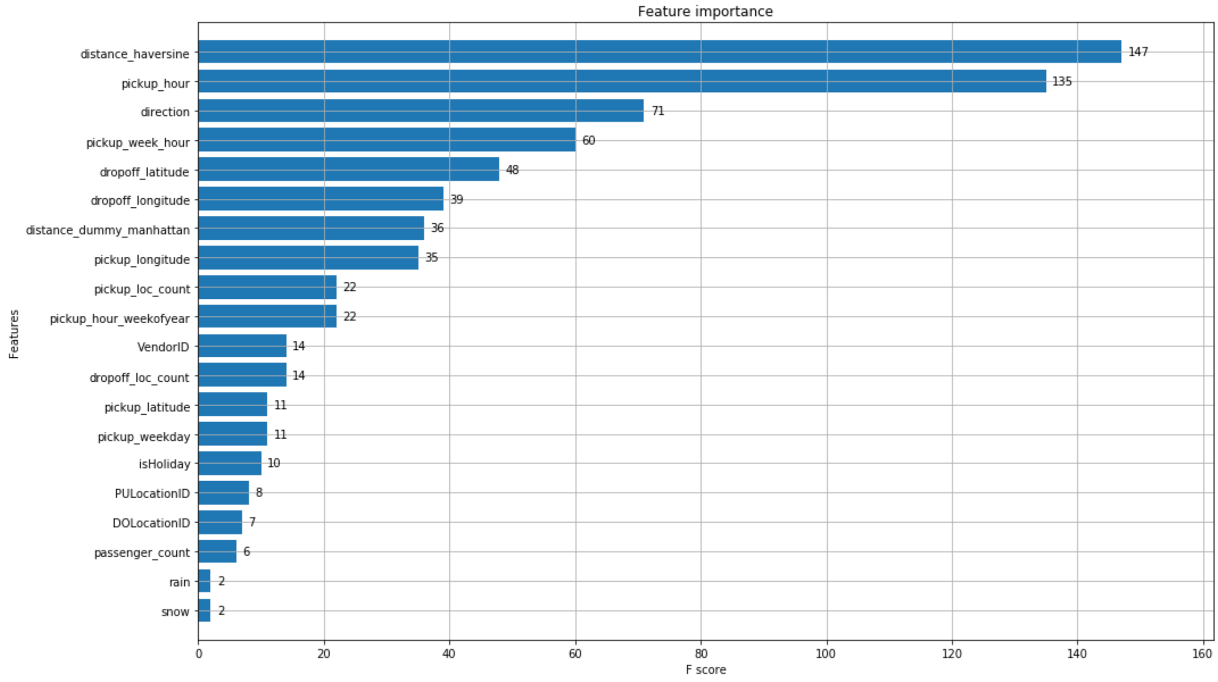
```
X.columns
```

```
Index(['VendorID', 'passenger_count', 'PULocationID', 'DOLocationID',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'snow', 'rain', 'isHoliday',
'distance_haversine',
       'pickup_weekday', 'pickup_hour_weekofyear', 'pickup_hour',
       'pickup_minute', 'pickup_week_hour', 'distance_dummy_manhattan',
       'direction', 'pickup_loc_count', 'dropoff_loc_count'],
      dtype='object')
```

[0]

```
from xgboost.sklearn import XGBRegressor
# Train the model again with the entire training set
model = XGBRegressor(objective ='reg:squarederror')
model.fit(X, Y)
xgb.plot_importance(model, height=0.8)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5e91c98400>
```



[0]

```
X.columns
```

```
Index(['VendorID', 'passenger_count', 'PULocationID', 'DOLocationID',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'snow', 'rain', 'isHoliday',
'distance_haversine',
       'pickup_weekday', 'pickup_hour_weekofyear', 'pickup_hour',
       'pickup_minute', 'pickup_week_hour', 'distance_dummy_manhattan',
       'direction', 'pickup_loc_count', 'dropoff_loc_count'],
      dtype='object')
```

[0]

```
from sklearn.model_selection import cross_val_score
# Evaluate features with K-fold cross validation
# The higher K is, the longer it takes to run, and the higher
your confidence in the score
K = 5
model = XGBRegressor(objective ='reg:squarederror')
scores = cross_val_score(model, X, Y, cv=K,
scoring='neg_mean_squared_error', verbose=False)
avg_rmse = math.sqrt(abs(np.mean(scores)))

print('Average RMSE with {}-fold Cross Validation:
{:.3f}'.format(K, avg_rmse))
```

```
Average RMSE with 5-fold Cross Validation: 29.489
```

[0]

```
feature_names = X.columns
```

[0]

```
X_test = test[feature_names]
y_test = test['duration']
```

[0]

```
Xtr, Xv, ytr, yv = train_test_split(X.values, Y, test_size=0.2,
random_state=1987)
dtrain = xgb.DMatrix(Xtr, label=ytr)
dvalid = xgb.DMatrix(Xv, label=yv)
dtest = xgb.DMatrix(test[feature_names].values)
watchlist = [(dtrain, 'train'), (dvalid, 'valid')]

# Try different parameters! My favorite is random search :)
# xgb_pars = {'min_child_weight': 50, 'eta': 0.3,
'colsample_bytree': 0.3, 'max_depth': 10,
#              'subsample': 0.8, 'lambda': 1., 'nthread': 4,
'booster' : 'gbtree', 'silent': 1,
#              'eval_metric': 'rmse', 'objective': 'reg:linear'}
```

[0]

```
xgb_pars = {'min_child_weight': 1, 'eta': 0.5,
'colsample_bytree': 0.9,
            'max_depth': 6,
'subsample': 0.9, 'lambda': 1., 'nthread': -1, 'booster' :
'gbtree', 'silent': 1,
'eval_metric': 'rmse', 'objective': 'reg:linear'}
model = xgb.train(xgb_pars, dtrain, 10, watchlist,
early_stopping_rounds=2,
      maximize=False, verbose_eval=1)
print('Modeling RMSLE %.5f' % model.best_score)
```

```
[0]     train-rmse:30.2333      valid-rmse:33.2554
Multiple eval metrics have been passed: 'valid-rmse' will be used for
early stopping.

Will train until valid-rmse hasn't improved in 2 rounds.
[1]     train-rmse:29.0727      valid-rmse:32.3971
[2]     train-rmse:28.7171      valid-rmse:32.1716
[3]     train-rmse:28.5662      valid-rmse:32.0916
[4]     train-rmse:28.4996      valid-rmse:32.0657
[5]     train-rmse:28.4555      valid-rmse:32.0578
[6]     train-rmse:28.4067      valid-rmse:32.0514
[7]     train-rmse:28.3275      valid-rmse:32.0575
[8]     train-rmse:28.2677      valid-rmse:32.0612
Stopping. Best iteration:
[6]     train-rmse:28.4067      valid-rmse:32.0514

Modeling RMSLE 32.05145
```

```
[0]
```