Bike sharing in Oslo

Runar Helin and Jon Nordby
September 2017

Abstract

The bike sharing system in Oslo publishes open data (https://developer.oslobysykkel.no/data) about trips made by their users. We analysed one month usage (June 2016) and looked at typical trip durations, daily traffic patterns as well as movement at various times. Our results indicate that Affinity Propagation clustering based on trips between stations can distinguish movement patterns between weekends and typical commute times during weekdays.

In [1]:
```
%load_ext autoreload

%matplotlib inline
%autoreload 2

import math
# Supress nosiy deprecation warning from inside matplotlib
import warnings
warnings.filterwarnings('ignore')

# Part of Anaconda distribution
import numpy
import pandas
import matplotlib
import matplotlib.pyplot as plt
import sklearn.cluster

# External dependencies
# pip install geopy graphviz folium
import folium

# Custom code developed for the notebook, ./oslo.py
import oslo

matplotlib.rcParams['figure.figsize'] = (14, 8)
```

In [2]:
```
# Download raw data
start = (2016, 6)
end = (2016, 7)
notexisting = [(2017, 1), (2017, 2), (2017, 3) ]
periods = sorted(set(oslo.months_between(start, end)).difference(notexisting))
for period in periods:
    try:
        filename = oslo.download_trip(*period)
    except Exception as e:
        raise RuntimeError("Could not download %d-%d: %s" % (*period, e.msg))
```

```
using existing trips-2016.6.1-2016.6.30.csv.zip
```

In [3]:
```
stations = oslo.read_stations()
print("%d stations" % len(stations.keys()))
dict(filter(lambda kv: kv[0] != 'bounds', stations[157].items()))
```

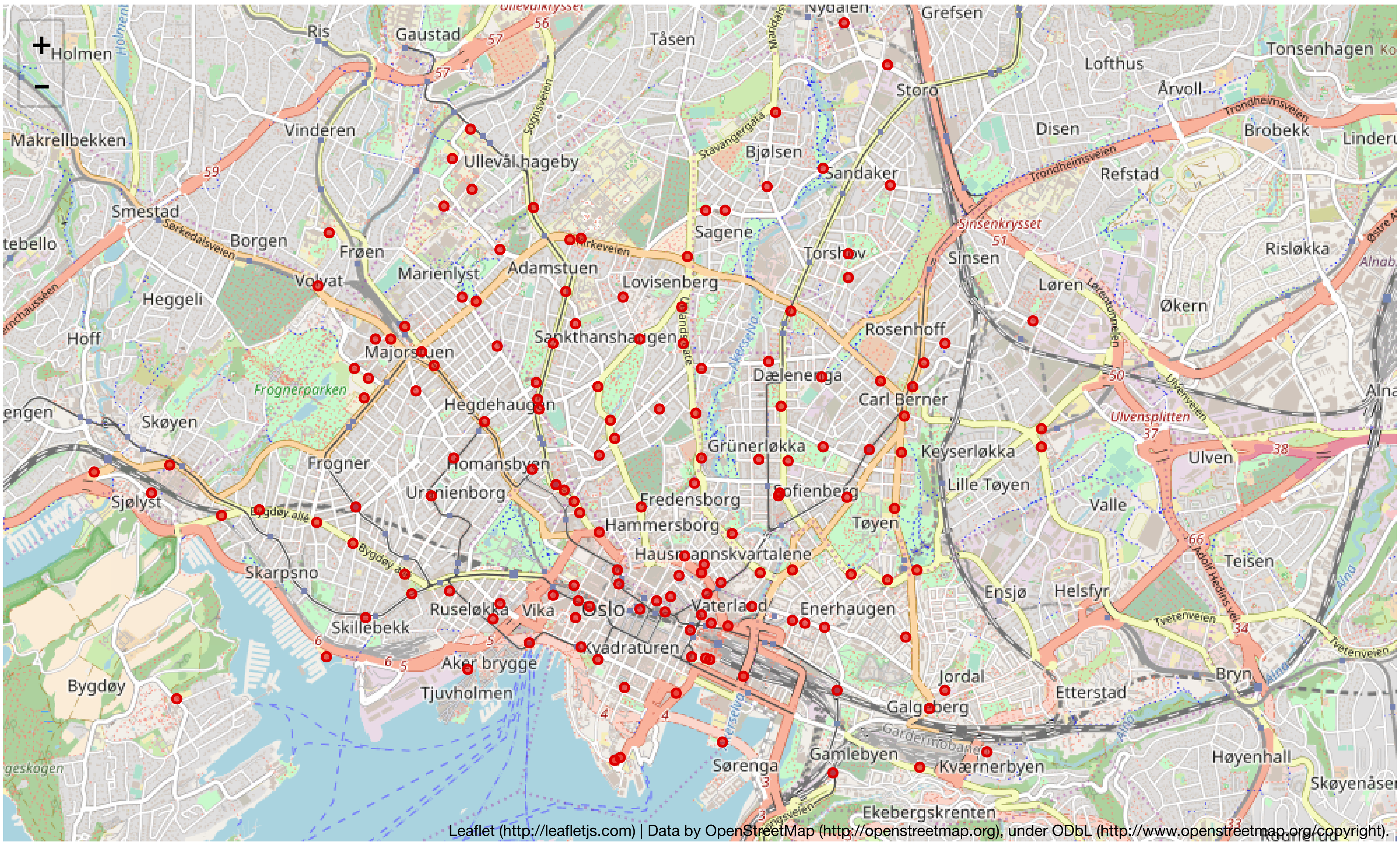
```
153 stations
```

Out[3]:
```
{'center': {'latitude': 59.91562, 'longitude': 10.762248},
 'id': 157,
 'in_service': True,
 'number_of_locks': 30,
 'subtitle': 'mellom Norbygata og Urtegata',
 'title': 'Nylandsveien'}
```

In [29]:

```
map_center = [59.925, 10.75]
map_zoom = 12
station_map = folium.Map(location=map_center, zoom_start=map_zoom)

for _, station in stations.items():
    center = station['center']
    lon, lat = center['longitude'], center['latitude']
    folium.CircleMarker([lat, lon],
                radius=3, popup=station['title'],
                color='#dd0000', fill_color='#dd0000',
    ).add_to(station_map)

station_map
```

Out[29]:



In [5]:

```
# Read in the files
trips = pandas.DataFrame()
for period in periods:
    filename = "data/"+ oslo.trips_basename(*period)+'.csv.zip'
    print('reading', filename)
    frame = pandas.read_csv(filename, index_col=None, header=0, parse_dates=[1, 3])
    trips = pandas.concat([trips, frame])
trips[:3]
```

```
reading data/trips-2016.6.1-2016.6.30.csv.zip
```

Out[5]:

	Start station	Start time	End station	End time
0	226	2016-06-01 03:59:59	243	2016-06-01 04:02:14
1	206	2016-06-01 04:00:02	212	2016-06-01 04:18:46
2	290	2016-06-01 04:00:06	261	2016-06-01 04:02:14

In [6]:

```
# Find total number of trips in data set
number_trips = trips.shape[0]
number_trips
```

Out[6]: 292302

Enriching the data

In [7]:

```
# Index based on time
trips.set_index(['Start time'], drop=False, inplace=True)

# Add trip durations intervals
trips['Duration'] = trips['End time'] - trips['Start time']
# Convert from nanoseconds, remove timedelta type
trips['Duration Seconds'] = pandas.Series(trips['Duration'], dtype='int64').abs() / (1000*1000*1000)
```

In [8]:
```
# Add distance of the trip
# Note: a bit slow, since doing geometric calculations in pure Python
subs = trips
subs['Distance'] = subs.apply(lambda r: oslo.calculate_distance(stations, r), 'columns')
```

In [9]:
```
# Add overall velocity (m/s)
trips['Velocity'] = trips['Distance'] / trips['Duration Seconds']
trips[:3]
```

Out[9]:

	Start station	Start time	End station	End time	Duration	Duration Seconds	Distance	Velocity
Start time								
2016-06-01 03:59:59	226	2016-06-01 03:59:59	243	2016-06-01 04:02:14	00:02:15	135.0	610.115142	4.519371
2016-06-01 04:00:02	206	2016-06-01 04:00:02	212	2016-06-01 04:18:46	00:18:44	1124.0	2917.746920	2.595860
2016-06-01 04:00:06	290	2016-06-01 04:00:06	261	2016-06-01 04:02:14	00:02:08	128.0	354.690116	2.771017

Cleaning the data

In [10]:
```
### Do people return to same station often?
self_ = trips[trips['Start station'].eq(trips['End station'])]
print("%d %% of the trips go back to same stations" % (len(self_)/len(trips)*100,))
```

```
7 % of the trips go back to same stations
```

In [11]:
```
# 30% of return-to-same trips are below 1 minute.
# Where as < 3% of all trips are below. Probably errors!
minimum_duration = 1*60
own_trips = trips[trips['Start station'] == (trips['End station'])]
veryshort = trips[trips['Duration Seconds'] < minimum_duration]
invalid = own_trips[own_trips['Duration Seconds'] < minimum_duration]

possibly_valid = (len(veryshort)-len(invalid))/len(trips)
print("""
Eliminating trips under %d sec.
%.3f%% of start!=end trips
%d start==trips""" % (minimum_duration, 100*possibly_valid, len(invalid)))
```

```

Eliminating trips under 60 sec.
0.021% of start!=end trips
7797 start==trips
```

In [12]:
```
# Eliminating
valid = trips['Duration Seconds'] > minimum_duration
trips = trips[valid]
len(trips)
# Total number of trips after elimination
```

Out[12]: 284379

In [13]:
```
# Trips with missing start/end stations
# Nothing for June 2016, but happens for other months
missing_start = trips[pandas.isnull(trips['Start station'])]
missing_end = trips[pandas.isnull(trips['End station'])]
len(missing_start + missing_end)
```

Out[13]: 0

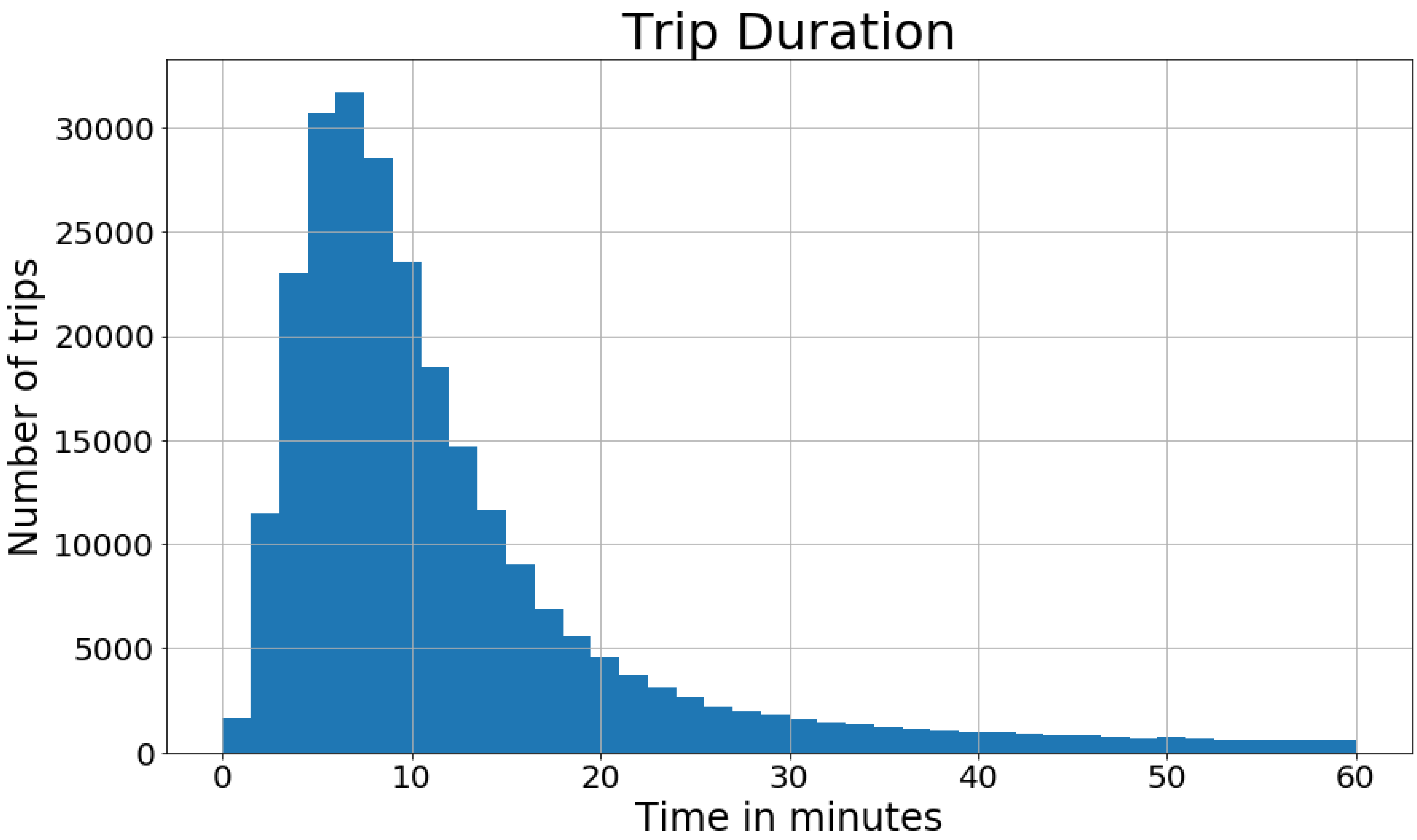
In [14]:
```
# Find missing station info
# Means we cannot know the geographical position, needed for plotting on map
def not_nan(n):
    return not math.isnan(n)
known_stations = set(stations.keys())
start_stations = set(filter(not_nan, trips['Start station'].unique()))
end_stations =  set(filter(not_nan, trips['End station'].unique()))
trip_stations = start_stations | end_stations
unknown_stations = trip_stations - known_stations
print('Unknown stations:', unknown_stations)

trips_unknown_end = trips[trips['End station'].isin(unknown_stations)]
trips_unknown_start = trips[trips['Start station'].isin(unknown_stations)]
print('Number of trips with missing start/end station: %d %d' % (len(trips_unknown_start), len(trips_unknown_end)))
```

```
Unknown stations: {288, 172, 173, 271, 186}
Number of trips with missing start/end station: 9403 10500
```

Basic characteristics

In [15]:

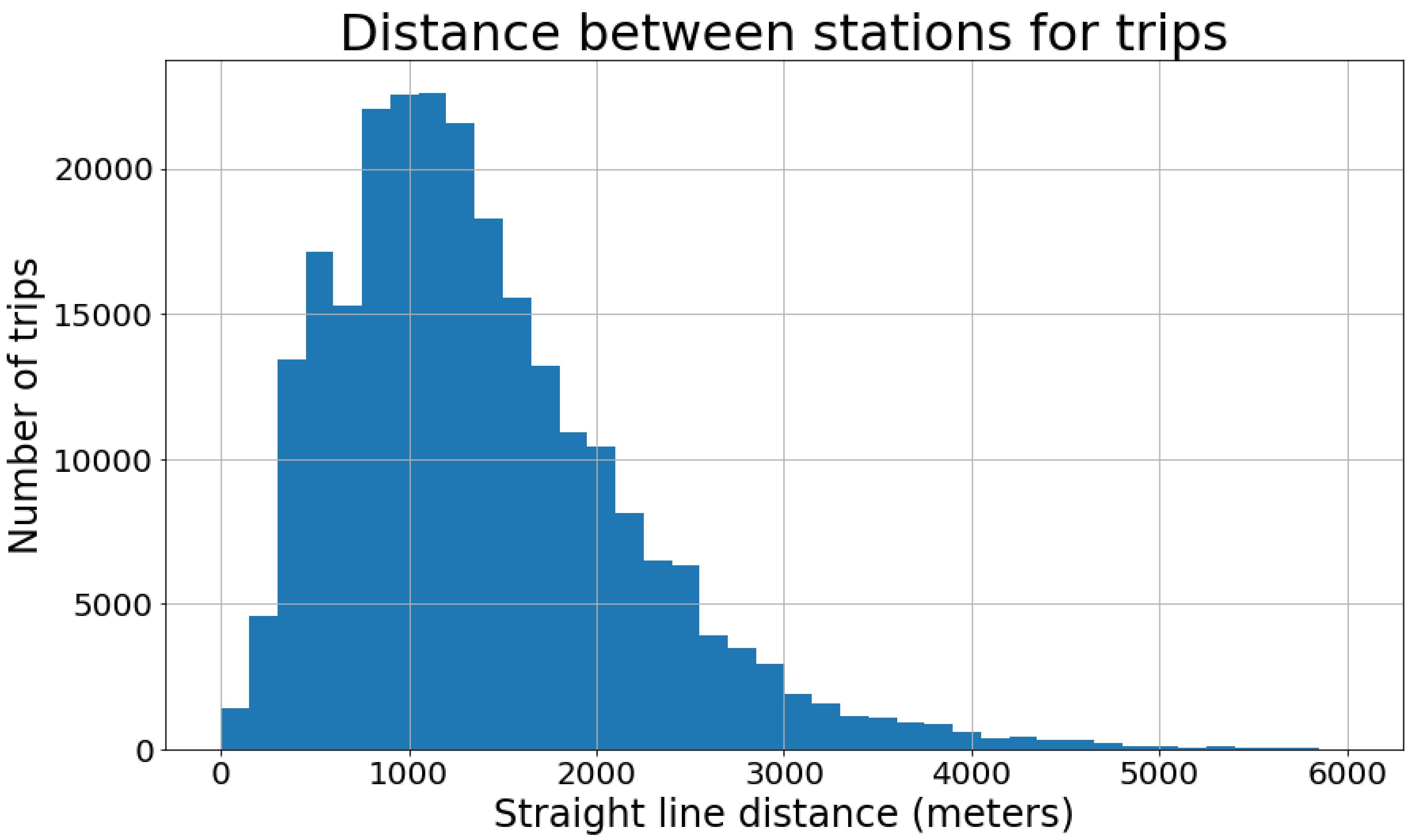
```
percentile_lim = 0.75
duration_mean = trips['Duration Seconds'].median() / 60
percentile = trips['Duration Seconds'].quantile(percentile_lim) / 60
(trips['Duration Seconds'] / 60).hist(bins=40, range=(0, 60), xlabelsize=20, ylabelsize=20)
#plt.text(30.1, 20500, 'Median: %.2f' % (duration_mean), fontsize=30)
#plt.text(30.1, 15500, '%d %% percentile: %.2f' % (percentile_lim*100 ,percentile), fontsize=30)
plt.xlabel('Time in minutes', fontsize=24)
plt.ylabel('Number of trips', fontsize=24)
plt.title('Trip Duration', fontsize=32)
plt.savefig('Trip Duration.png', format='png', dpi=300)
```



A typical trip lasts about 10 minutes

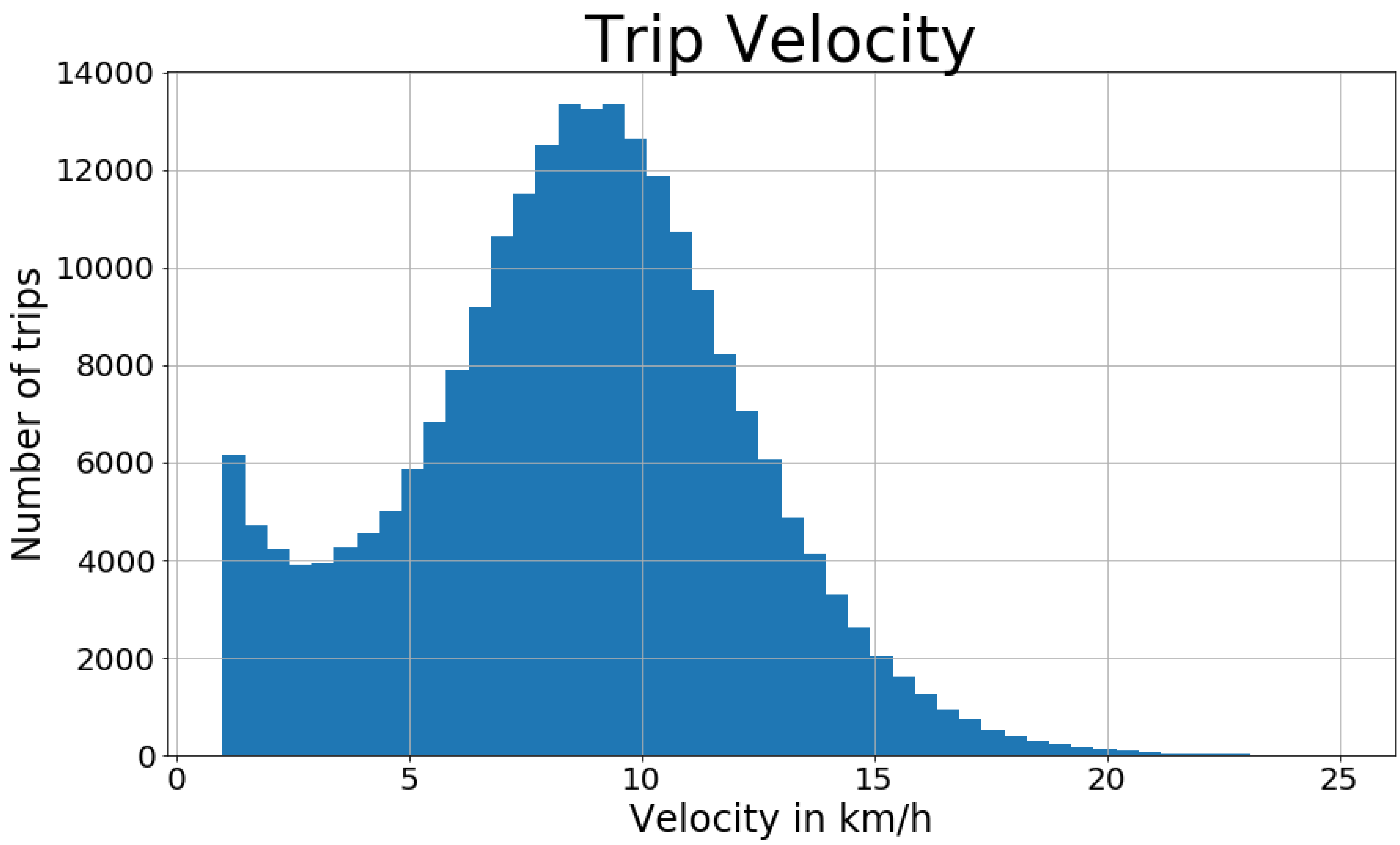
In [16]:

```
# 0 is a very common distance (travel back to self), ignore it
trips['Distance'].hist(bins=40, range=(1, 6000), xlabelsize=20, ylabelsize=20)
plt.xlabel('Straight line distance (meters)', fontsize=24)
plt.ylabel('Number of trips', fontsize=24)
plt.title('Distance between stations for trips', fontsize=32)
plt.savefig('Trip Distance.png', format='png', dpi=300)
```



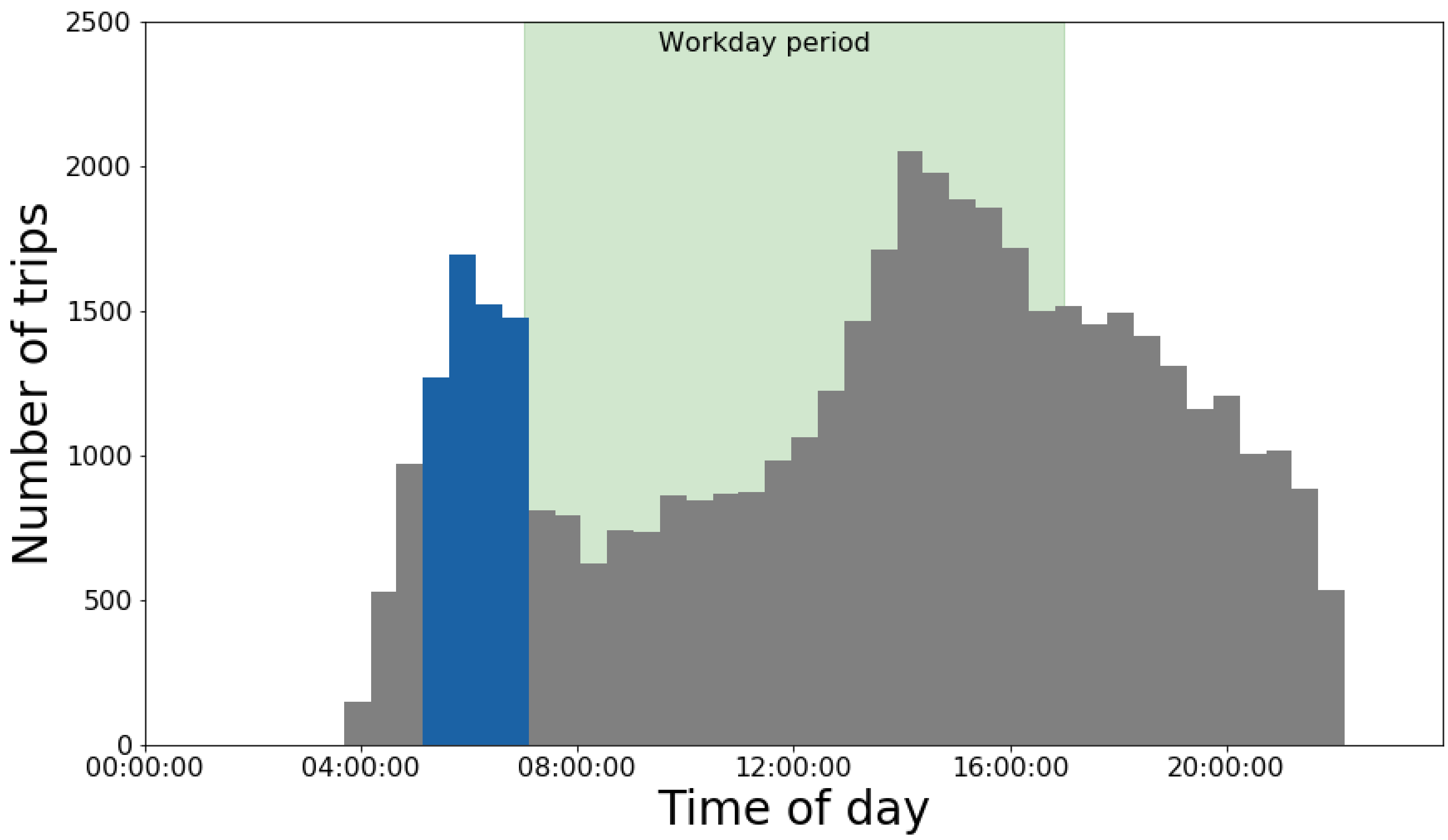
A typical trip is about 1 km in straight line distance

In [17]:
```
(trips['Velocity'] * 3.6).hist(bins=50, range=(1,25), xlabelsize=20, ylabelsize=20)
plt.xlabel('Velocity in km/h', fontsize=24)
plt.ylabel('Number of trips', fontsize=24)
plt.title('Trip Velocity', fontsize=40)
plt.savefig('Trip Velocity.png', format='png', dpi=300)
```



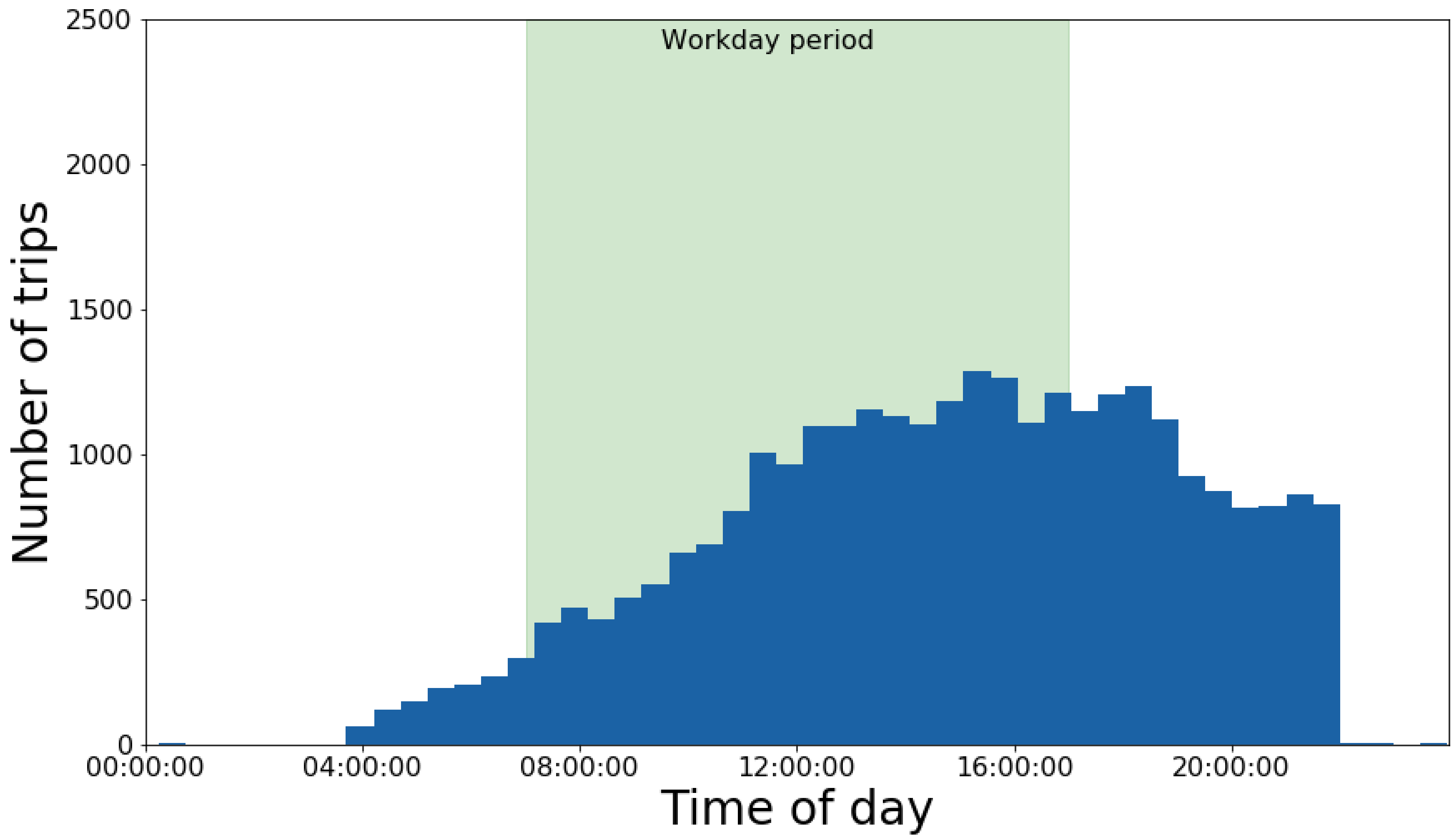
The velocity of a typical trip is about 10 km/h

In [18]:
```
to_work = (5*3600, 7*3600)
timeofday = trips
timeofday['timeofday'] = trips.index.time
series = timeofday['timeofday']
perday = series[trips.index.weekday < 5]
oslo.plot_trip_times(perday, title='towork-trips', selected=to_work, scale=5.0);
```



The spike at the period 05:00 - 07:00 is marked in blue and is presumed to be the time when people go to work. The period marked in green is the workday period. In the afternoon the spike is not as well defined, and the people returning from work is not as easily seen.

In [19]:
```
oslo.plot_trip_times(series[trips.index.weekday >= 5], title='weekend-trips', scale=2.0);
```



There are no clear spikes in the weekends, and people generally start their bike trips later in the day

Estimating neighbourhood areas using Spectral Clustering

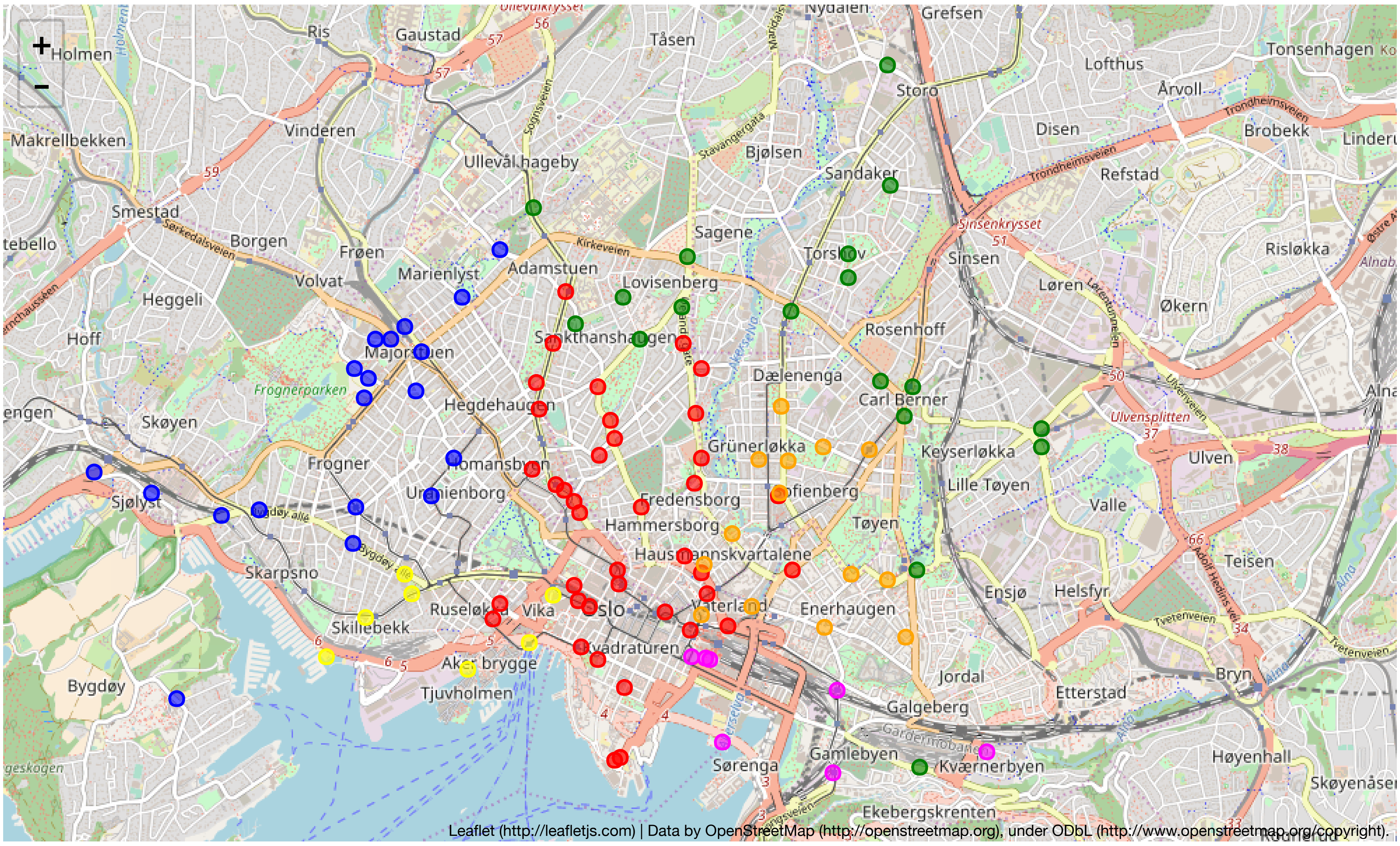
We want to see how the bikes moves around in the city. We use the number of trips between the stations as a metric for how strongly they are 'connected'.

The Spectral Clustering method tries to find clusters that have a minimum number of trips between different clusters, but maximize trips inside them. Hence it is an estimate for 'areas' which people tend to stay inside of.

In [20]:
```
# Returns an affinity matrix
def station_connectivity(frame):
    outbound = pandas.crosstab(frame['Start station'], frame['End station'])
    inbound = pandas.crosstab(frame['End station'], frame['Start station'])
    # Using the sum gives us an undirected affinity
    # This makes the matrix symmetrical across the diagonal, required by spectral clustering
    connectivity = inbound + outbound
    # Spectral clustering also requires diagonal to be zero
    numpy.fill_diagonal(connectivity.values, 0)
    return connectivity

# Map clusters back to station IDs
def cluster_labels_to_station_ids(connectivity, labels):
    no_clusters = len(set(labels))

    station_clusters = [ [] for n in range(0, no_clusters) ]
    for idx, label in enumerate(labels):
        station = connectivity.columns[idx]
        station_clusters[label].append(station)

    # Largest clusters first
    station_clusters = sorted(station_clusters, key=len, reverse=True)
    return station_clusters

# Perform clustering using Spectral Clustering
def cluster_spectral(frame, n_clusters=9):
    connectivity = station_connectivity(frame)
    cluster = sklearn.cluster.SpectralClustering(n_clusters=n_clusters, affinity='precomputed')
    labels = cluster.fit_predict(connectivity)

    station_clusters = cluster_labels_to_station_ids(connectivity, labels)

    return station_clusters
```

In [21]:
```
df = trips
df = df[df.index.weekday >= 5]
clustered = cluster_spectral(df, n_clusters=6)
oslo.plot_station_groups(stations, clustered)
```

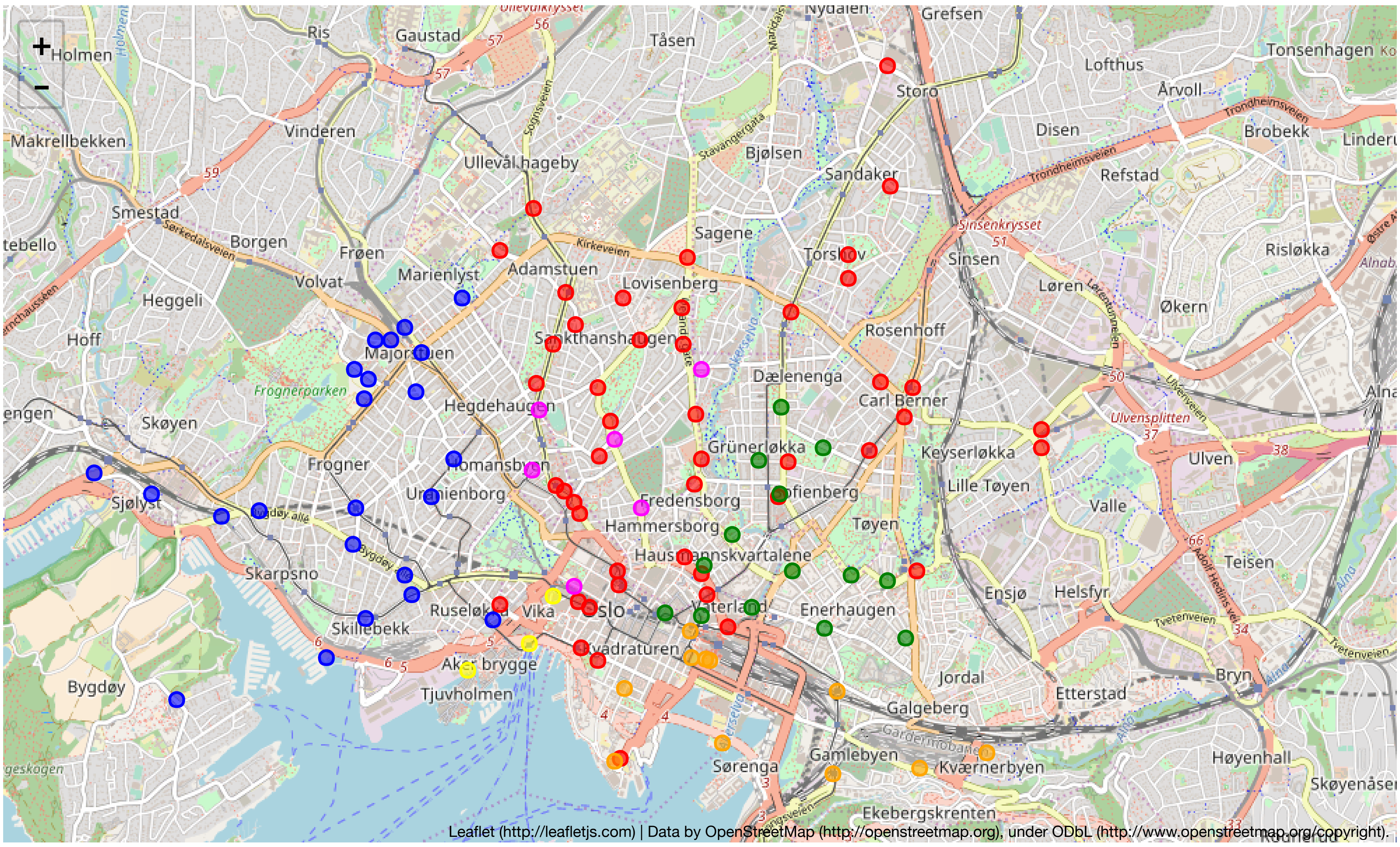
Out[21]:



On weekends, we can clearly see that the clusters is separated into geographically parts of Oslo.

In [22]:
```
df = trips[trips.index.weekday < 5 ]
clustered = cluster_spectral(df, n_clusters=6)
oslo.plot_station_groups(stations, clustered)
```

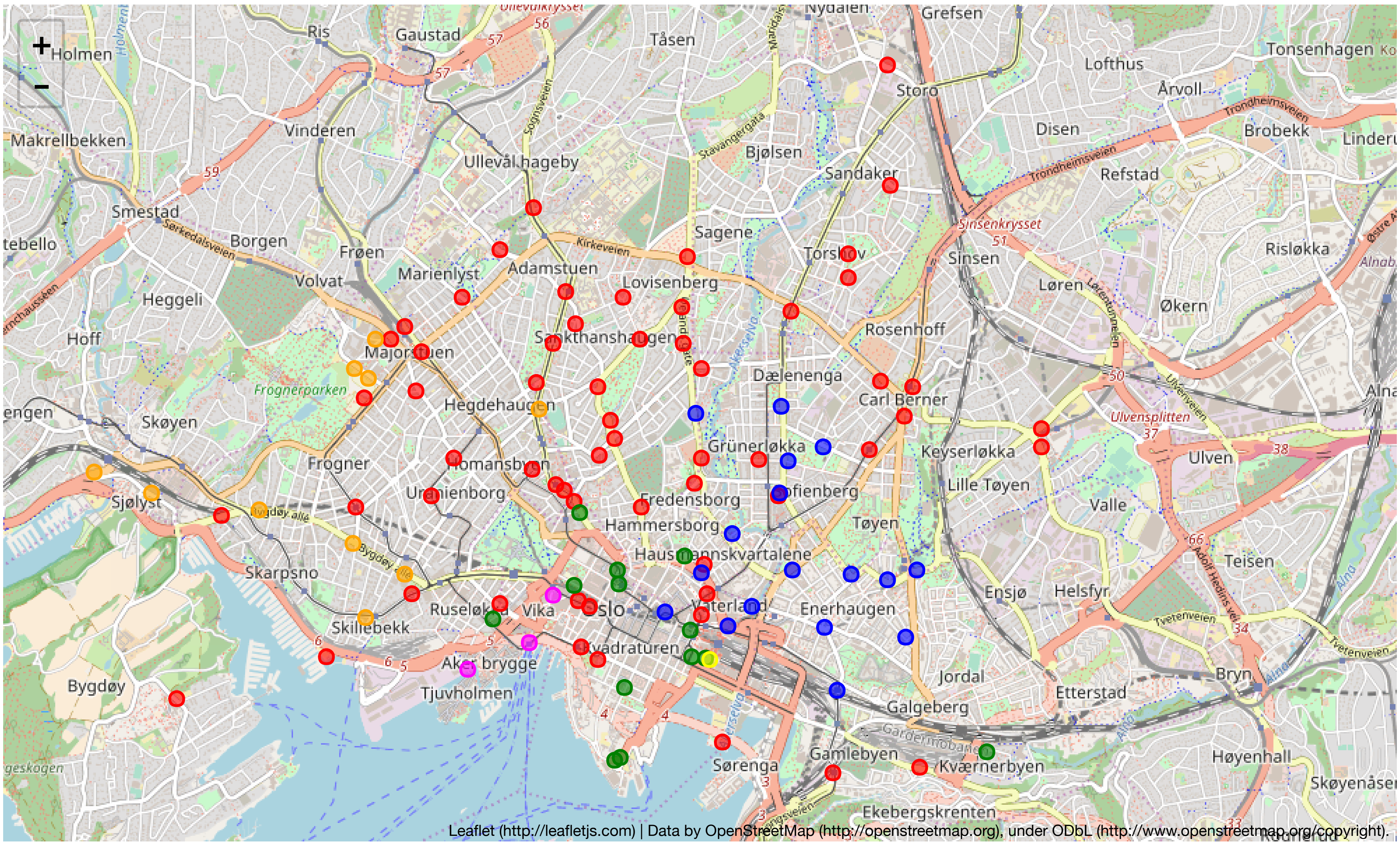
Out[22]:



We see a similar clustering for weekdays

In [23]:
```
df = trips[trips.index.weekday < 5 ]
df = df.between_time('5:00', '7:00')
clustered = cluster_spectral(df, n_clusters=6)
oslo.plot_station_groups(stations, clustered)
```

Out[23]:



For the commute period on weekdays between 05:00 and 07:00 the largest cluster (in red) contains nearly half the stations. The cluster spreads across almost all of Oslo, also overlapping with other clusters. The distinct geographical boundaries are gone, and it is not clear what the explanation is.

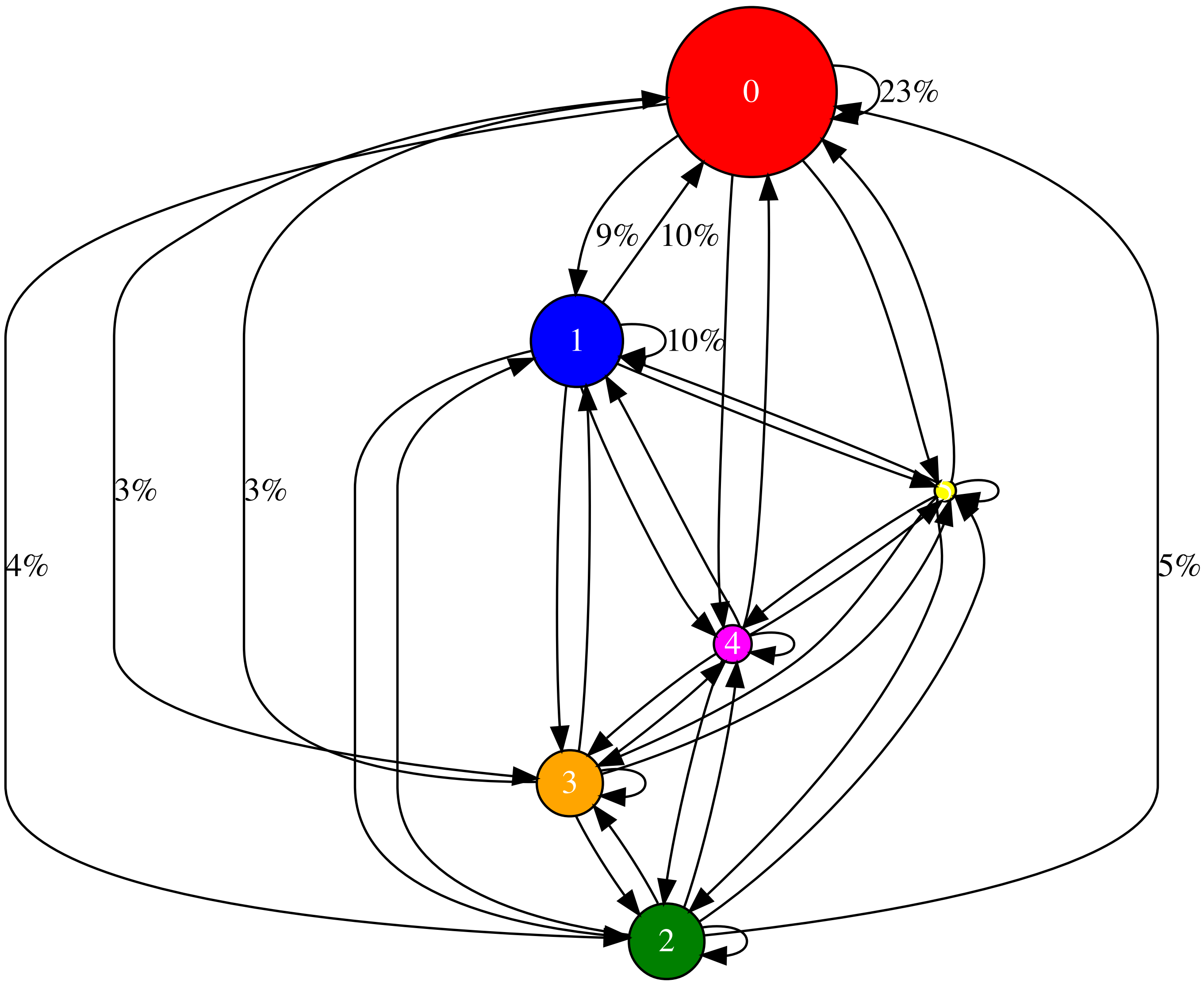
Visualizing cluster connectivity as a graph

In an attempt to better understand the clusters we get, we try to visualize them as a directed graph.

In [24]:
```
# Calculate number of trips between and inside clusters
def cluster_trips(stations, df, clusters):

    out = numpy.empty((len(clusters), len(clusters)))

    for from_cluster in range(0, len(clusters)):
        for to_cluster in range(0, len(clusters)):

            to_stations = clusters[to_cluster]
            from_stations = clusters[from_cluster]

            is_outbound = df['Start station'].isin(from_stations) & df['End station'].isin(to_stations)
            is_inbound = df['End station'].isin(from_stations) & df['Start station'].isin(to_stations)

            out[from_cluster][to_cluster] = df[is_inbound].shape[0]
            out[to_cluster][from_cluster] = df[is_outbound].shape[0]

    return pandas.DataFrame(data=out)
```

In [25]:
```
# Visualize clustered connectivity
stats = cluster_trips(stations, trips, clustered)
dot = oslo.cluster_digraph(clustered, stats)

dot
```

Out[25]:



Nodes areas are proportional to the number of stations in the cluster. Edges represent the percentage of total trips between the clusters. Traffic internally in cluster. Colors correspond to the color used in the map plot.

We believe this kind of visualization can be helpful understand inter-cluster behavior.

However it did not help us understand why we get a single large cluster during commute times.

Finding hubs using Affinity Propagation clustering

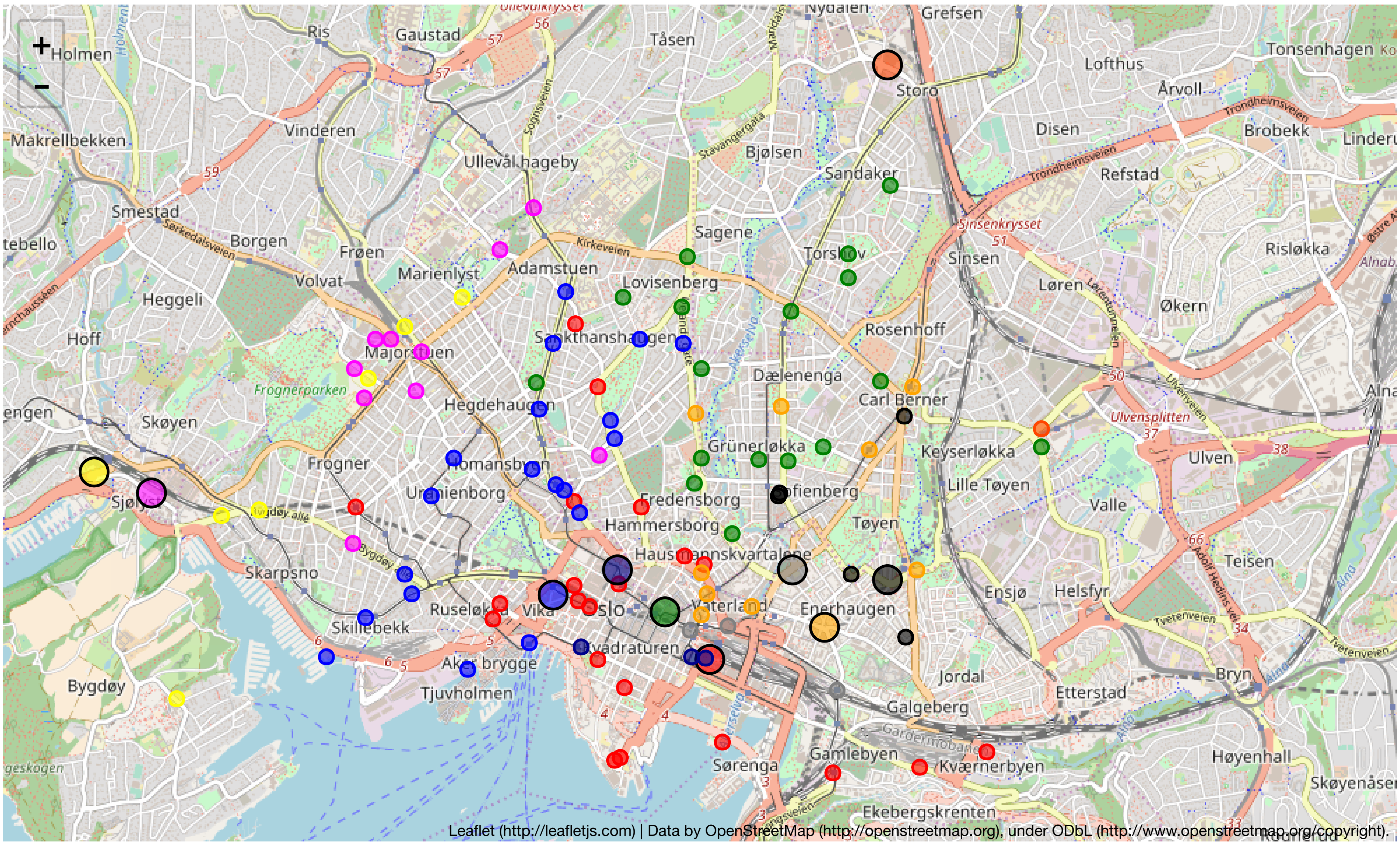
We switch to use the Affinity Propagation (https://en.wikipedia.org/wiki/Affinity_propagation) clustering method, which identifies an 'examplar' and attemps to build a cluster based on this exemplar. By using the number of trips as the affinity, we should get clusters with stations that are highly connected to the exemplar.

In [26]:
```
def cluster_affinity(frame):
    connectivity = station_connectivity(frame)
    cluster = sklearn.cluster.AffinityPropagation(affinity='precomputed')
    labels = cluster.fit_predict(connectivity)

    centers = cluster.cluster_centers_indices_
    center_stations = [ connectivity.columns.values[idx] for idx in centers ]
    station_clusters = cluster_labels_to_station_ids(connectivity, labels)

    return (station_clusters, center_stations)
```

In [27]:

```
# Commute time on weekdays
df = (trips[trips.index.weekday < 5 ]).between_time('5:00', '7:00')
clustered, centers = cluster_affinity(df)
m = oslo.plot_station_groups(stations, clustered, centers)
m.save('affinity-cluster-workday-commute.html')
m
```

Out[27]:

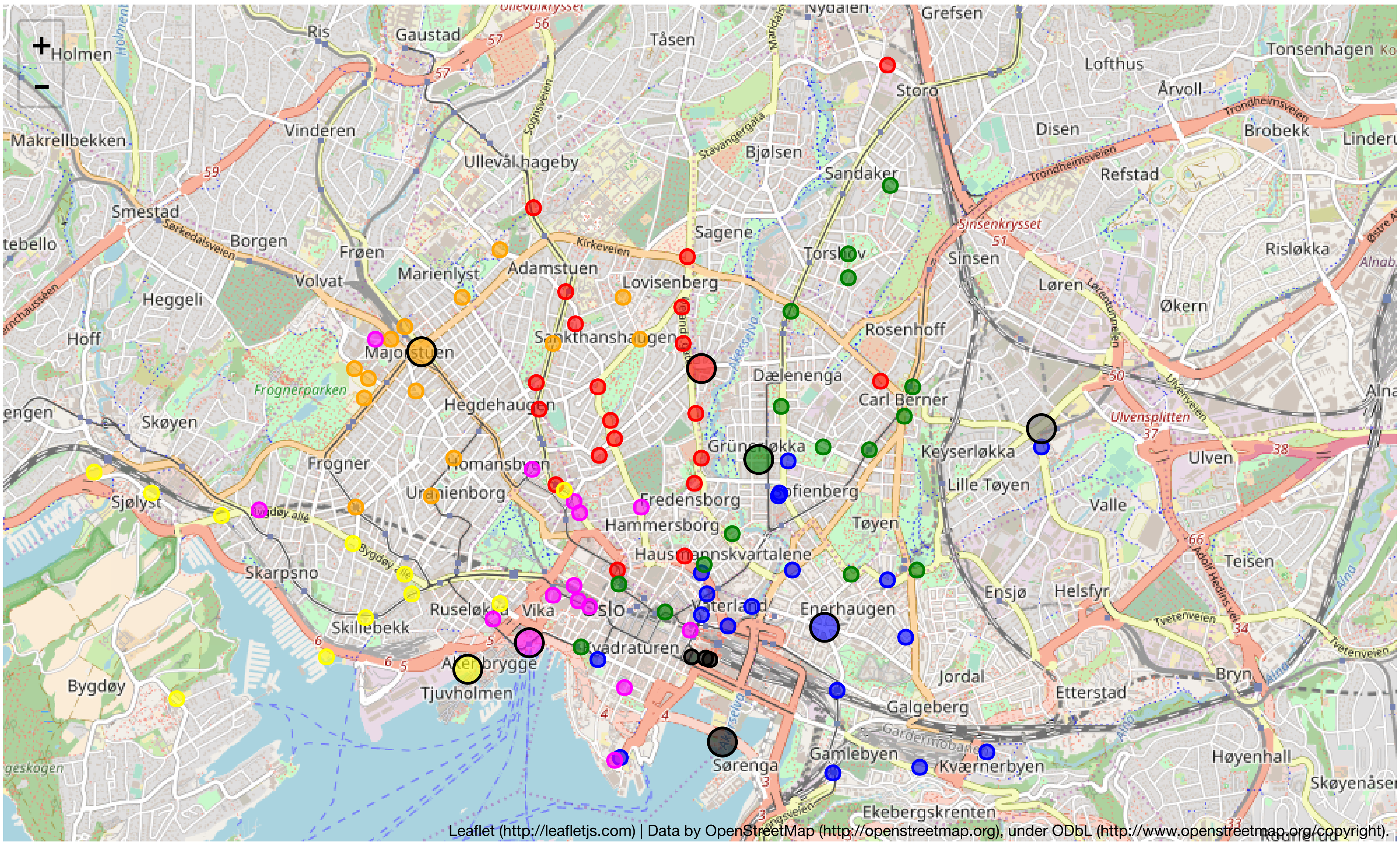


We see that trips are centered on traffic hubs around Oslo S and Nationaltheatret and office areas in Skøyen.

In [28]:

```
# In weekends
df = (trips[trips.index.weekday >= 5 ])
clustered, centers = cluster_affinity(df)
m = oslo.plot_station_groups(stations, clustered, centers)
m.save('affinity-cluster-weekend.html')
m
```

Out[28]:



In the weekends the trips are centered around social areas like Aker brygge, Grünerløkka, Majorstuen and Oslo Opera House.

Ideas for further work

Use a directed affinity matrix, with only inbound or outbound trips, to see which direction bikes are moving. Check the affinity propagation clustering method on other timeperiods of the year. Use cross-validation to establish how well the clustering fits.

Use bike data in combination with public transportation data to get an idea for how people combine bike sharing with other transportation modes.