In [2]:

```
import scipy as sps
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tqdm.notebook import tqdm

import plotly.graph_objects as go
import plotly.express as px
import plotly.offline


from mpl_toolkits.basemap import Basemap
sns.set(font_scale=1.5)
```

1. Станции велопроката

Попробуем начать с исследования датасета со станциями велопроката.

In [3]:

```
stations = pd.read_csv('cycle-share-dataset/station.csv')

stations.head()
```

Out[3]:

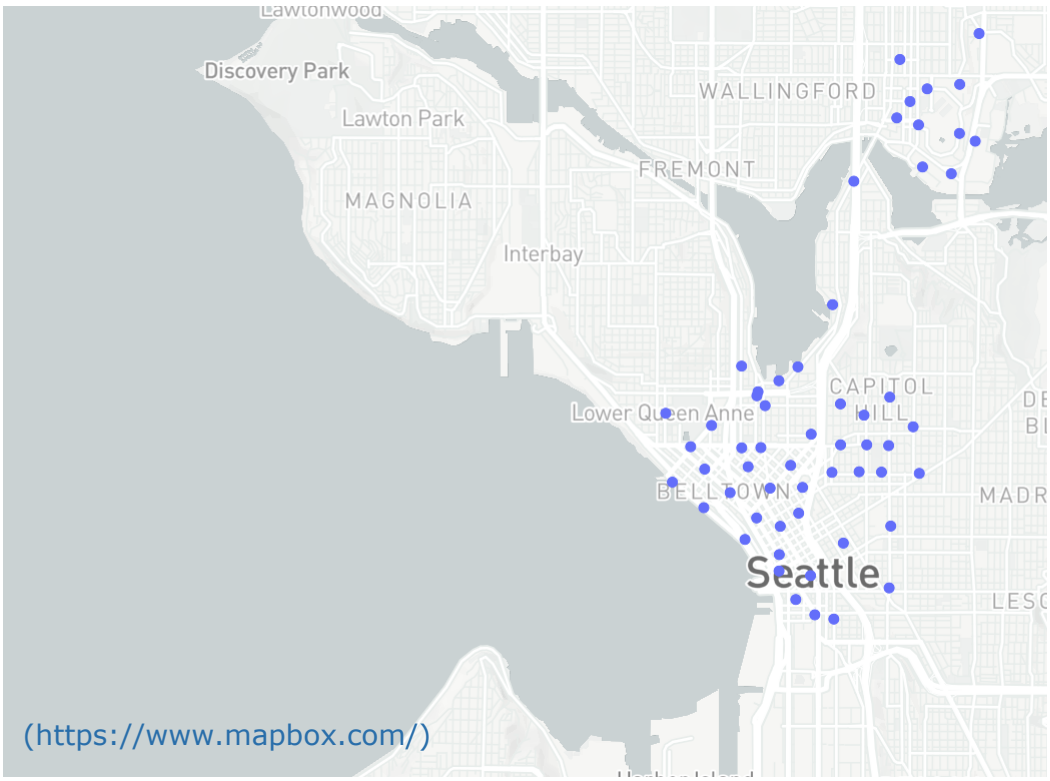
	station_id	name	lat	long	install_date	install_dockcount	modification_date
0	BT-01	3rd Ave & Broad St	47.618418	-122.350964	10/13/2014	18	NaN
1	BT-03	2nd Ave & Vine St	47.615829	-122.348564	10/13/2014	16	NaN
2	BT-04	6th Ave & Blanchard St	47.616094	-122.341102	10/13/2014	16	NaN
3	BT-05	2nd Ave & Blanchard St	47.613110	-122.344208	10/13/2014	14	NaN
4	CBD-03	7th Ave & Union St	47.610731	-122.332447	10/13/2014	20	NaN

Визуализируем их на карте.

In [4]:

```
px.set_mapbox_access_token(open("public_key").read())
fig = px.scatter_mapbox(stations, lat="lat", lon="long",
                        zoom=11)

fig.show()
```

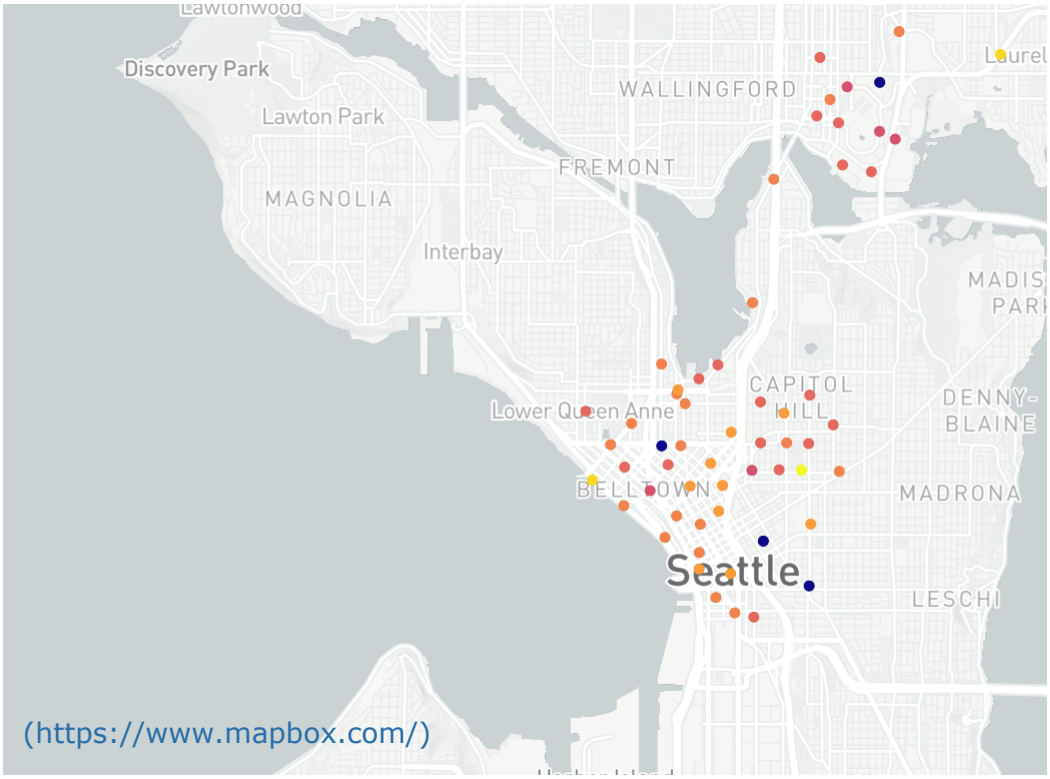


Можно видеть, что координаты точек разбиваются на два кластера.

Для каждой в поле станции `current_dockcount` указана её вметсимость, т.е. количество слотов для велосипедов. Визуализируем это информацию, используя радиус и цвет точки, указывающей местоположение станции.

In [5]:

```
px.set_mapbox_access_token(open("public_key").read())
fig = px.scatter_mapbox(stations, lat="lat", lon="long",
                        color="current_dockcount",
                        size_max=12,
                        zoom=11)
fig.show()
```



Видно, что существуют три станции, в которых current_dockcount = 0. Это значит, что они были выведены из строя.

По графику ничего нельзя сказать о зависимости вместимости станции от её расположения.

2. Поездки

Загрузим теперь данные о поездках.

In [6]:

```
trips = pd.read_csv('cycle-share-dataset/trip.csv',
                    error_bad_lines=False,
                    parse_dates=[1, 2])
```

```
b'Skipping line 50794: expected 12 fields, saw 20\n'
```

In [7]:

```
1 trips.head()
```

Out[7]:

	trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_station_name	from
0	431	2014-10-13 10:31:00	2014-10-13 10:48:00	SEA00298	985.935	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
1	432	2014-10-13 10:32:00	2014-10-13 10:48:00	SEA00195	926.375	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
2	433	2014-10-13 10:33:00	2014-10-13 10:48:00	SEA00486	883.831	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
3	434	2014-10-13 10:34:00	2014-10-13 10:48:00	SEA00333	865.937	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	
4	435	2014-10-13 10:34:00	2014-10-13 10:49:00	SEA00202	923.923	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	

In [8]:

```
1 trips.shape
```

Out[8]:

```
(286857, 12)
```

Визуализируем поездки с помощью прямых линий, соединяющих станции.

In [9]:

```
1  trips_general = trips[['trip_id', 'from_station_id', 'to_station_id']]
2
3  joined = trips_general.merge(stations, left_on='from_station_id',
4                               right_on = 'station_id', how='left',
5                               suffixes=('_st','_to'))
6
7  joined = joined[['trip_id', 'from_station_id',
8                   'to_station_id', 'lat', 'long']]
9  joined = joined.merge(stations, left_on='to_station_id',
10                       right_on = 'station_id', how='left',
11                       suffixes=('_from','_to'))
12
13 joined = joined[['trip_id', 'from_station_id',
14                  'to_station_id', 'lat_from', 'long_from',
15                  'lat_to', 'long_to']]
```

In [10]:

```
1  trips_count = joined.reset_index().groupby(['from_station_id',
2                                               'to_station_id']).count()['index']
```

In [11]:

```
unique_trips = joined.drop(columns='trip_id').drop_duplicates()
```

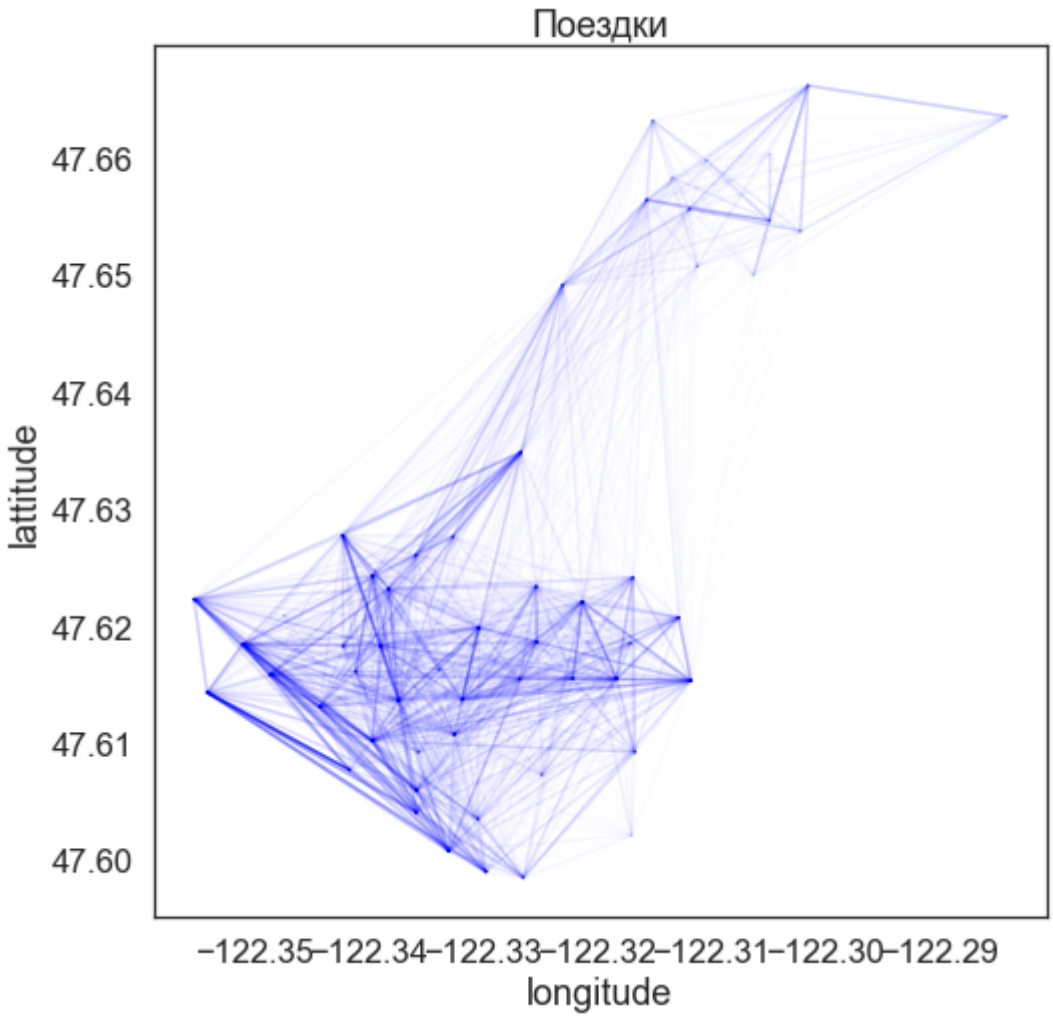
In [12]:

```
plt.figure(figsize=(8, 8))
sns.set_style('white')

for index, line in tqdm(unique_trips.iterrows()):
    plt.plot([line.long_from, line.long_to], [line.lat_from, line.lat_to],
             color='blue',
             alpha=trips_count.loc[line.from_station_id, line.to_station_id]/
                      np.max(trips_count))

plt.title('Поездки')
plt.xlabel('longitude')
plt.ylabel('lattitude')
plt.show()
```

Error rendering Jupyter widget: missing widget manager



Можно видеть, что станции разбиваются на два кластера в разных частях города. Перемещения велосипедов из кластера в кластер случаются реже, чем перемещения внутри кластеров.

По графику (прозрачности отрезков), также, видно, что наиболее частыми были поездки в центральном районе города.

Займемся исследованием популярности поездок более подробно.

3. Поиск популярных маршрутов

Для начала посмотрим, сколько уникальных маршрутов всего представлено в датасете.

In [13]:

```
routes = set()

for route in trips.values:
    routes.add((route[7], route[8]))
    routes.add((route[8], route[7]))

print(f'Кол-во маршрутов: {len(routes)//2}')
```

```
Кол-во маршрутов: 1554
```

In [14]:

```
print(f'Кол-во возможных маршрутов: {stations.shape[0]*(stations.shape[0] - 1)//2
```

```
Кол-во возможных маршрутов: 1653
```

Почти все возможные маршруты представлены в датасете.

Напишем универсальный метод, который по датасету возвращает словарь {маршрут: его количество в датасете}, ключи (маршруты) в котором отсортированы по убыванию популярности.

In [15]:

```
def most_popular(trip):
    d = {}

    for route in trip.values:
        from_to = (route[7], route[8])
        if from_to in d:
            d[from_to] += 1
        else:
            d[from_to] = 1

    d = {k: v for k, v in sorted(d.items(), key=lambda item: item[1])[::-1]}

    return d
```

Также напишем функцию отрисовки n самых популярных маршрутов на карте.

In [16]:

```
def plot_popular(popular_dict, n=10):
    px.set_mapbox_access_token(open("public_key").read())
    fig = px.scatter_mapbox(stations, lat="lat", lon="long",
                            zoom=11)
    top = list(popular_dict.keys())[:n]

    for route in top:
        st_from = route[0]
        st_to = route[1]
        from_lon = stations[stations.station_id == st_from].long.values[0]
        from_lat = stations[stations.station_id == st_from].lat.values[0]
        to_lon = stations[stations.station_id == st_to].long.values[0]
        to_lat = stations[stations.station_id == st_to].lat.values[0]

        fig.add_trace(go.Scattermapbox(mode = "markers+lines",
                                       lon = [from_lon, to_lon],
                                       lat = [from_lat, to_lat],
                                       marker = {'size': 10}))

    fig.show()
```

3.1. В зависимости от выходного или рабочего дня

Для этого заведем отдельный признак - is_holiday.

In [17]:

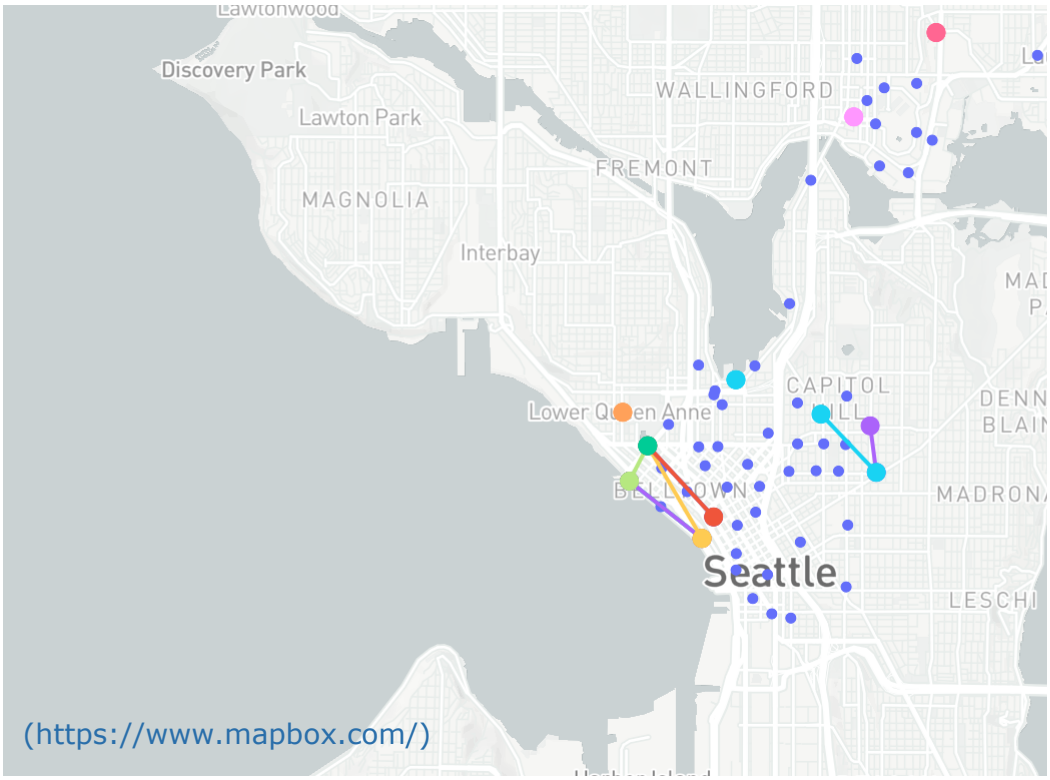
```
trips['day_of_week'] = trips.starttime.dt.dayofweek
trips['is_holiday'] = (trips.day_of_week.values == 5).astype('int') + \
                      (trips.day_of_week.values == 6).astype('int')
```

In [18]:

```
working_popular = most_popular(trips[trips.is_holiday == 1])
```

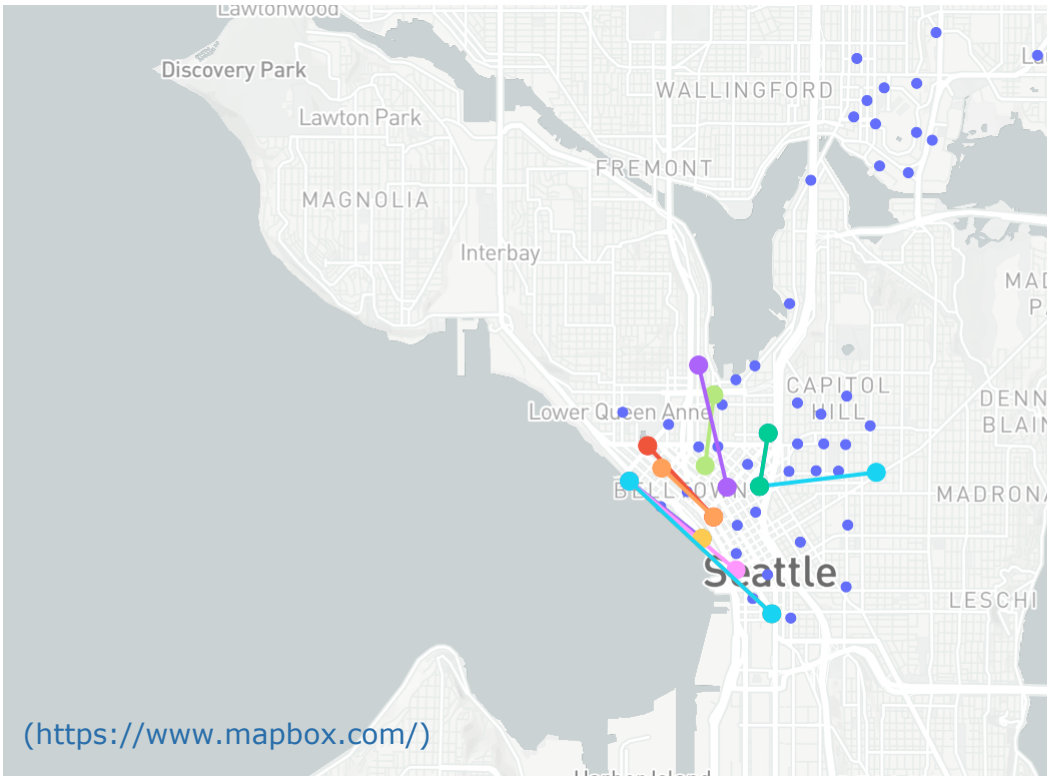
In [19]:

```
1 plot_popular(working_popular, 15)
```



In [20]:

```
nonworking_popular = most_popular(trips[trips.is_holiday == 0])
plot_popular(nonworking_popular, 15)
```



Результат: как видим, вне зависимости от того, рабочий день или нет, самые популярные маршруты оказываются в центре города. Можно заметить, что в рабочие дни маршруты короче по дистанции, причем есть такие, которые вырождаются в одну точку (где взял велосипед, там его и сдал). Это может быть связано с тем, что люди, работающие в центре, берут велосипед на обеденный перерыв, или чтобы доехать до деловой встречи, расположенной недалеко.

В выходные же маршруты длиннее, не вырождаются в одну точку, зачастую протянуты вдоль набережной. Все эти факторы указывают на то, что эти поездки скорее "прогулочные", нежели "деловые".

3.2. В зависимости от времени суток

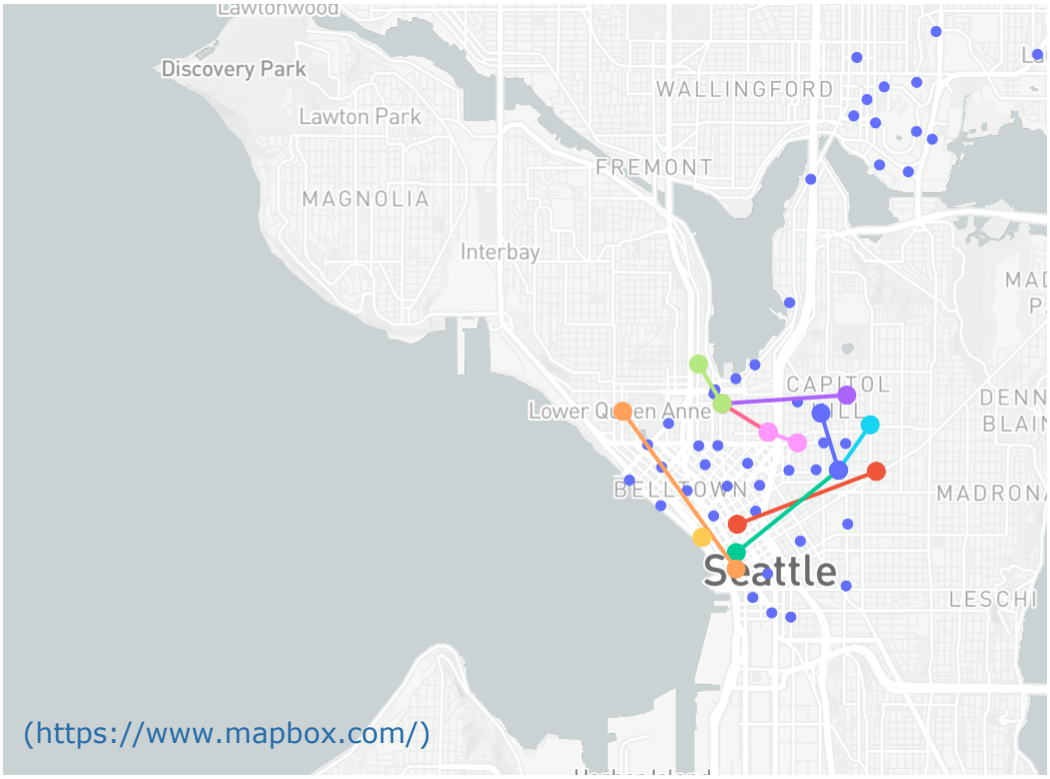
In [21]:

```
trips['hour'] = pd.to_datetime(trips.starttime.values).hour
trips['time_of_day'] = (trips.hour.values >= 6).astype('int') + \
                       (trips.hour.values >= 12).astype('int') + \
                       (trips.hour.values >= 18).astype('int')
```

Ночь:

In [22]:

```
night_popular = most_popular(trips[trips.time_of_day == 0])
plot_popular(night_popular, 10)
```

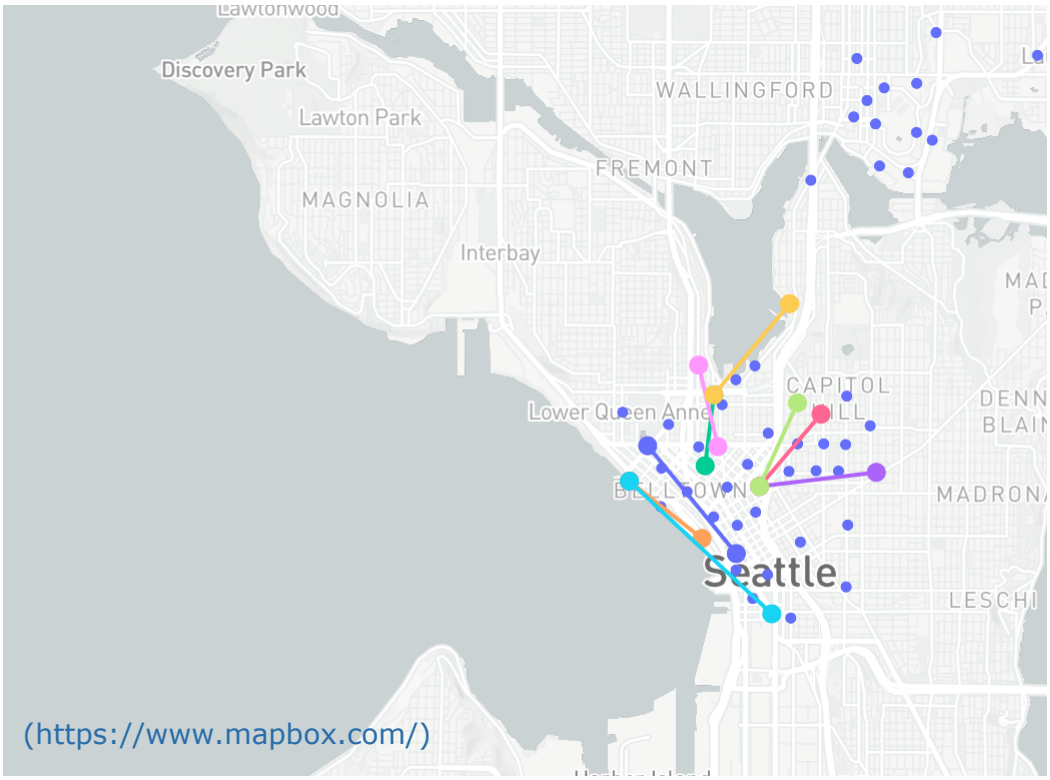


Много популярных маршрутов ночью связаны с районами South Lake Union и Capitol Hill. Если посчитать про эти районы в интернете, то они отличаются обилием баров и ночных клубов, что объясняет получившийся результат.

Утро:

In [23]:

```
morning_popular = most_popular(trips[trips.time_of_day == 1])
plot_popular(morning_popular, 10)
```

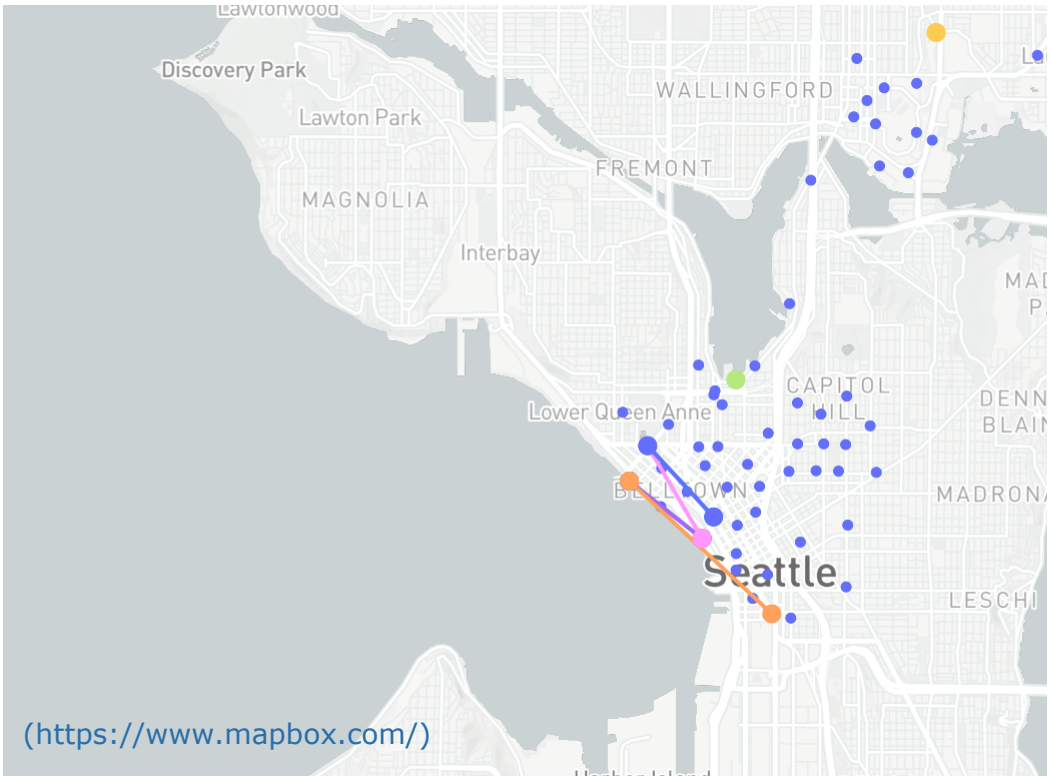


Здесь много маршрутов ведут в Belltown - деловой центр города. С утра туда едут на работу люди, которые живут не очень далеко и могут добираться на работу на велосипеде.

День:

In [24]:

```
day_popular = most_popular(trips[trips.time_of_day == 2])
plot_popular(day_popular, 10)
```

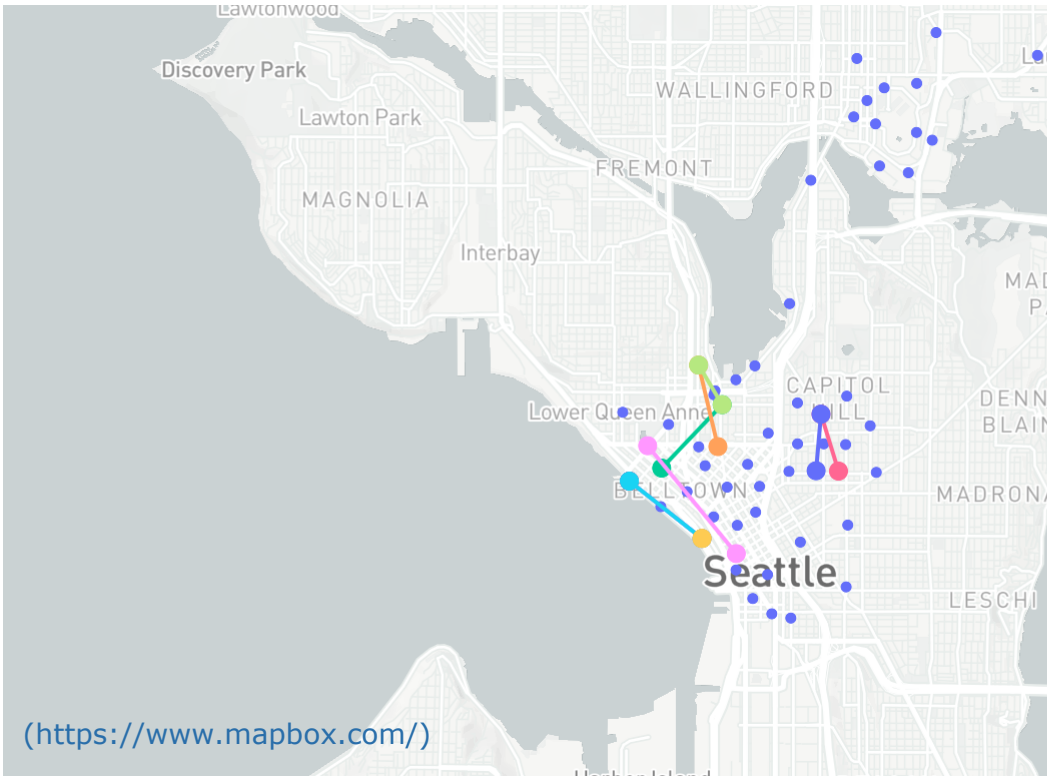


Результат похож на самые популярные маршруты в рабочие дни, основные поездки происходят в самом центре, где много офисов, что подтверждает нашу ранее высказанную догадку о том, что это поездки "по делам".

Вечер:

In [25]:

```
evening_popular = most_popular(trips[trips.time_of_day == 3])
plot_popular(evening_popular, 10)
```



Самый интересный результат получился здесь. Видно, что вечером остается чуть-чуть деловой активности, но и в районах с большим количеством баров появляется много маршрутов :)