Solution of "Value Based Order Location Anticipation" by Dmitrii Khizbullin

In [ ]:

```
!pip install matplotlib
!pip install pandas
# We will use google maps API
!pip install gmplot
```

In [ ]:

```
# We will need this for clusterzation. This package uses OR-Tools under the hood.
!pip install k-means-constrained
```

In [3]:

```
import math
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import gmplot

from k_means_constrained import KMeansConstrained
```

In [4]:

```
df_orig = pd.read_csv("robotex5.csv")
print("Original data size", len(df_orig))
# Let's 10x decimate the points to limit the further clusterization time
df = df_orig.sample(frac=1/10)
df.sort_index(inplace=True)
print("Decimated data size", len(df))
```

```
Original data size 627210
Decimated data size 62721
```

Convert time to datetime

In [5]:

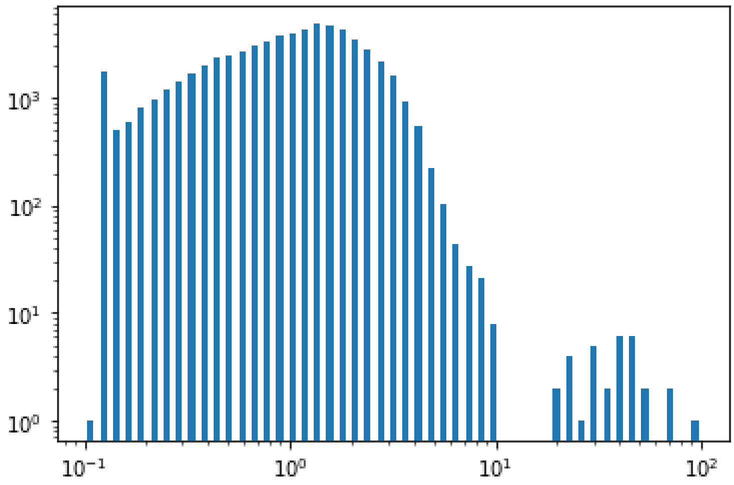
```
df['start_time'] = pd.to_datetime(df['start_time'])
```

Filter ride_value outliers by capping them

In [6]:

```
ride_vaules = df['ride_value'].to_numpy()
fig, ax = plt.subplots()
ax.hist(ride_vaules, bins=np.logspace(np.log10(0.1), np.log10(100), 50), rwidth=0.5)
ax.set_xscale('log')
ax.set_yscale('log')
print("median ride_vaules", np.median(ride_vaules))
```

```
median ride_vaules 1.0525
```




Seems that ride values are in something like euros (maybe in 1/10 of euros). Let's filter out top 0.1% of prices.

In [7]:

```
# We need this to clamp outliers in ride cost
def cap_by_percentile(array: np.ndarray, percentile=0.001):
    array_sorted = np.sort(array)
    loc = int(len(array_sorted) * (1 - percentile))
    max_val = array_sorted[loc]
    print("max_val", max_val)
    array_capped = np.copy(array)
    array_capped[array_capped > max_val] = max_val
    return array_capped

df['ride_value'] = cap_by_percentile(df['ride_value'].to_numpy())
```

```
max_val 28.57025
```

We have capped the ride value at maybe around 280 euro. Not sure a ride can be more expensive.

In [8]:

```
df.head(5)
```

Out[8]:

| | start_time | start_lat | start_lng | end_lat | end_lng | ride_value |
|---|---|---|---|---|---|---|
| **0** | 2022-03-06 15:02:39.329452 | 59.407910 | 24.689836 | 59.513027 | 24.831630 | 3.51825 |
| **9** | 2022-03-17 16:20:20.028387 | 59.410783 | 24.721219 | 59.439901 | 24.771756 | 1.06975 |
| **16** | 2022-03-28 22:54:32.854802 | 59.440720 | 24.747952 | 59.440104 | 24.782386 | 0.47800 |
| **17** | 2022-03-01 10:51:09.123023 | 59.429038 | 24.772361 | 59.414550 | 24.740206 | 0.60050 |
| **26** | 2022-03-15 13:13:24.389791 | 59.443171 | 24.699707 | 59.435390 | 24.749013 | 0.71725 |

In [9]:

```
print("Start and end of the time span")
df['start_time'].min(), df['start_time'].max()
```

```
Start and end of the time span
```

Out[9]:

```
(Timestamp('2022-03-01 00:00:07.936317'),
 Timestamp('2022-03-28 23:57:07.776690'))
```

In [10]:

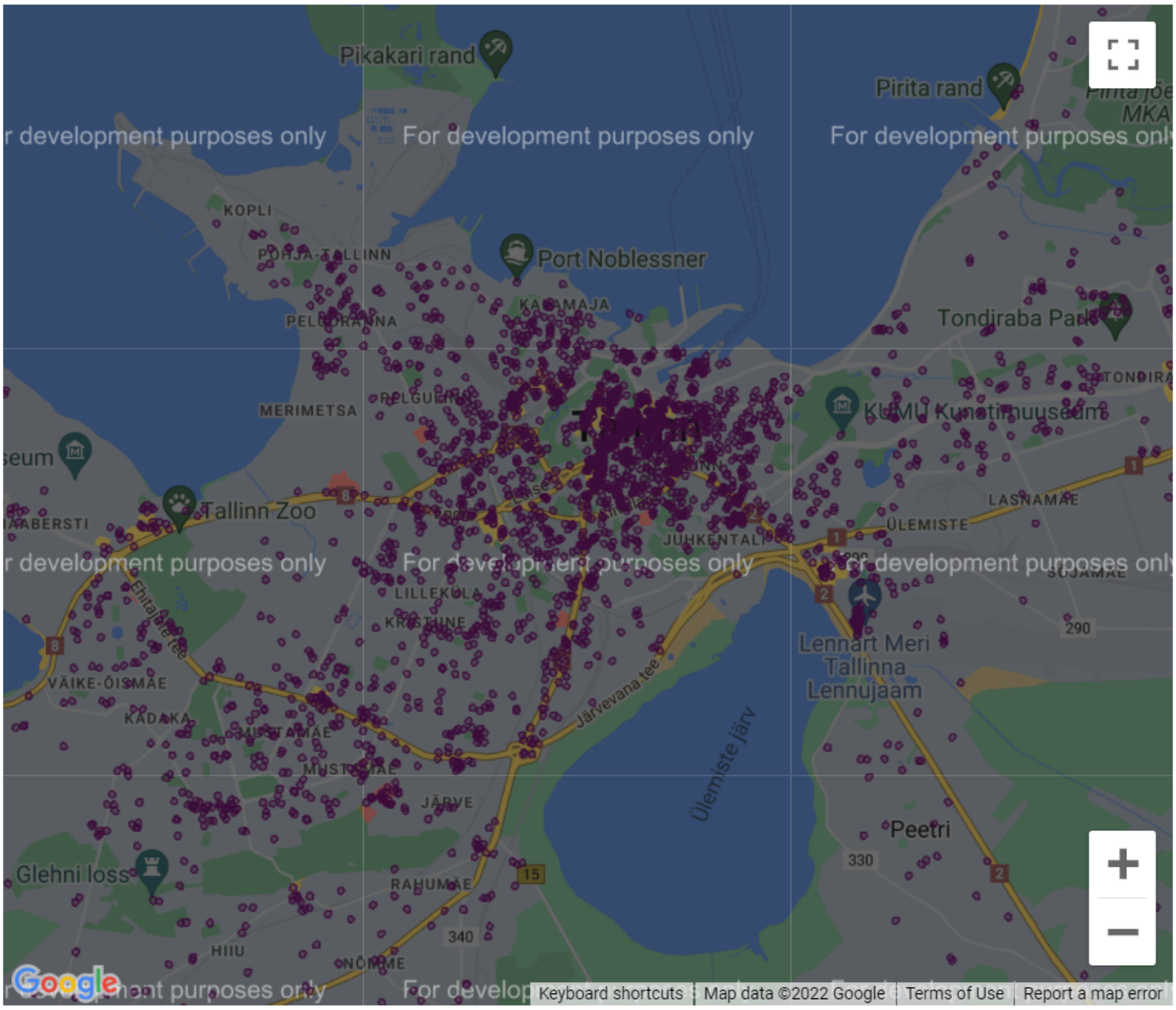
```
coord = (df['start_lat'], df['start_lng'])
coord_mean = [v.mean() for v in (df['start_lat'], df['start_lng'])]
print("Mean coordinates", coord_mean)
```

```
gmap = gmplot.GoogleMapPlotter(*coord_mean, 12, apikey='')

gmap.scatter(*[v[::20] for v in coord], color='#3B0B39', size=40, marker=False)
gmap.draw('map.html')
```

```
Mean coordinates [59.428738125527275, 24.743462078563457]
```

In [ ]:

```
from IPython.display import IFrame
IFrame(src='./map.html', width=700, height=600)
```




In [12]:

```
# Helper functions

# This function computes the distance in meters between two points given by coordina
def distance(coords_1: np.ndarray, coords_2: np.ndarray):
    # coords_n shape [N, 2], @0 - longitude (x), @1 - latitude (y)
    earth_radius = 6.371e6 # m
    start_lat_rad = math.pi * coords_1[:, 1] / 180
    diffs_deg = np.abs(coords_2 - coords_1)
    diffs_rad = math.pi * diffs_deg / 180
    diff_long_m = diffs_rad[:, 0] * earth_radius * np.cos(start_lat_rad)
    diff_lat_m = diffs_rad[:, 1] * earth_radius
    distance_m = np.sqrt(np.square(diff_long_m) + np.square(diff_lat_m))
    return distance_m

# This function does K-means clusterization with a constraint
# on the minimal size of a cluster. The constraint is important since
# we do not want a single distant point to create a degenerate single-point cluster.
def get_cluster_ids(
        loc_np,
        num_clusters,
```

```
        max_fraction_of_even = 0.2):
    mean_latitude = np.mean(loc_np[:, 1])
    # loc_rectified are fake locations that are kind-of coordinates but are
    # equally strethed in meters for latitude and longitude directions
    loc_rectified = loc_np.copy()
    loc_rectified[:, 1] /= math.cos(math.pi * mean_latitude / 180)
    size_min = int(max_fraction_of_even * len(loc_rectified) / num_clusters)
    size_max = len(loc_rectified)
    clf = KMeansConstrained(
        n_clusters=num_clusters,
        size_min=size_min,
        size_max=size_max,
        random_state=0,
        n_jobs=-1
    )
    cluster_ids = clf.fit_predict(loc_rectified)
    return cluster_ids
```

Let's group all points into 10 clusters using constrained K-means. This gives smaller clusters in dense areas and bigger clusters in sparse areas.

In [13]:

```
%%time

num_clusters = 10

loc_start = (df['start_lng'].to_numpy(), df['start_lat'].to_numpy())
loc_start_np = np.array(loc_start).T
loc_start_np.shape
cluster_ids = get_cluster_ids(loc_start_np, num_clusters)
df['cluster_id'] = cluster_ids
```

```
CPU times: total: 62.5 ms
Wall time: 50.3 s
```

Display the numbers of orders in clusters

In [14]:

```
df.groupby(['cluster_id']).size()
```

Out[14]:

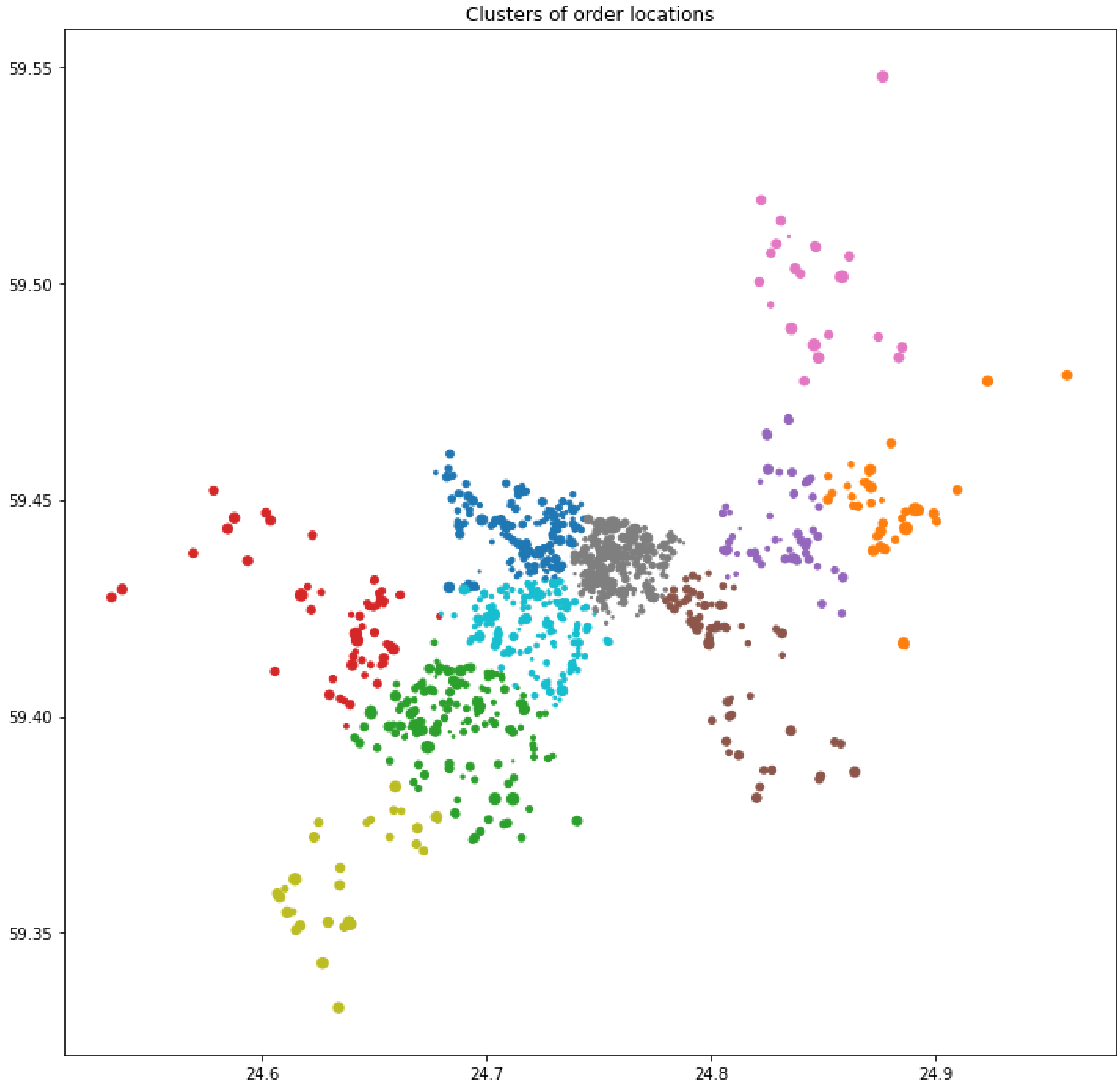
```
cluster_id
0      8498
1      2333
2      7920
3      3256
4      3341
5      4003
6      1618
7     21760
8      1288
9      8704
dtype: int64
```

Let's visualize clusters with colors. Size of the circles denotes ride value.

In [15]:

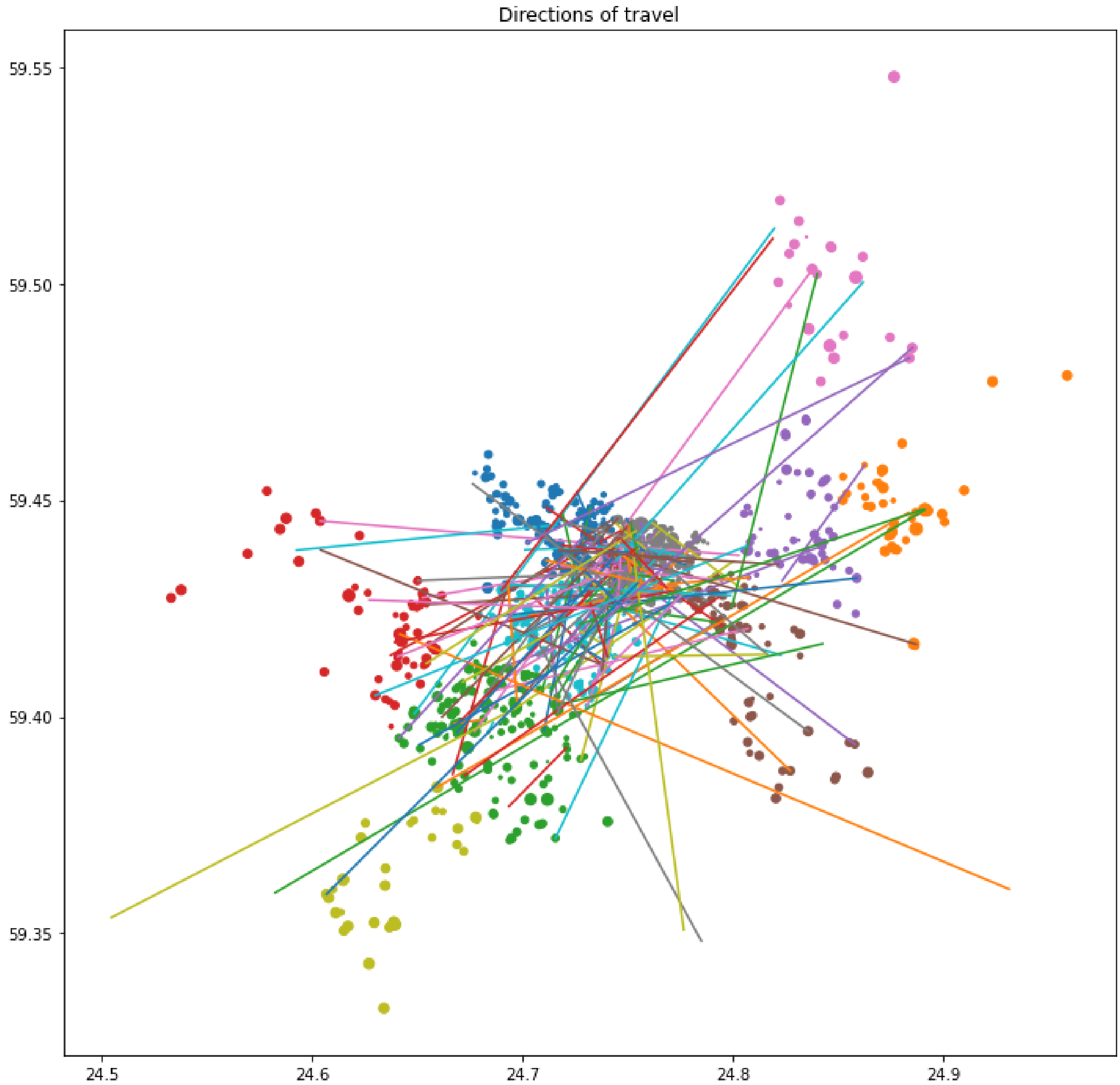
```
df_sel = df.sample(frac=1/50).sort_index()
loc_start = (df_sel['start_lng'].to_numpy(), df_sel['start_lat'].to_numpy())
loc_end = (df_sel['end_lng'].to_numpy(), df_sel['end_lat'].to_numpy())
ride_values = df_sel['ride_value'].to_numpy()
loc_start_np = np.array(loc_start).T
loc_end_np = np.array(loc_end).T
distances_m = distance(loc_start_np, loc_end_np)
sizes = ride_values * 50
cluster_ids = df_sel['cluster_id'].to_numpy()
```

```
plt.figure(figsize=(12, 12))
plt.title("Clusters of order locations")
sc = plt.scatter(*loc_start, marker='.', s=sizes, c=cluster_ids, cmap='tab10')
cluster_colors = np.array([sc.to_rgba(cid) for cid in range(num_clusters)])
plt.show()
```




Let's visualize each 10th ride to get the intuition of where the riders go.

In [16]:
```
plt.figure(figsize=(12, 12))
plt.title("Directions of travel")
plt.scatter(*loc_start, marker='.', s=sizes, c=cluster_ids, cmap='tab10')

for i_point, (start_pt, end_pt) in enumerate(zip(loc_start_np, loc_end_np)):
    if float(distance(np.expand_dims(start_pt, 0), np.expand_dims(end_pt, 0))) > 100
        continue
    if i_point % 10 == 0:
        plt.plot((start_pt[0], end_pt[0]), (start_pt[1], end_pt[1]))

plt.show()
```

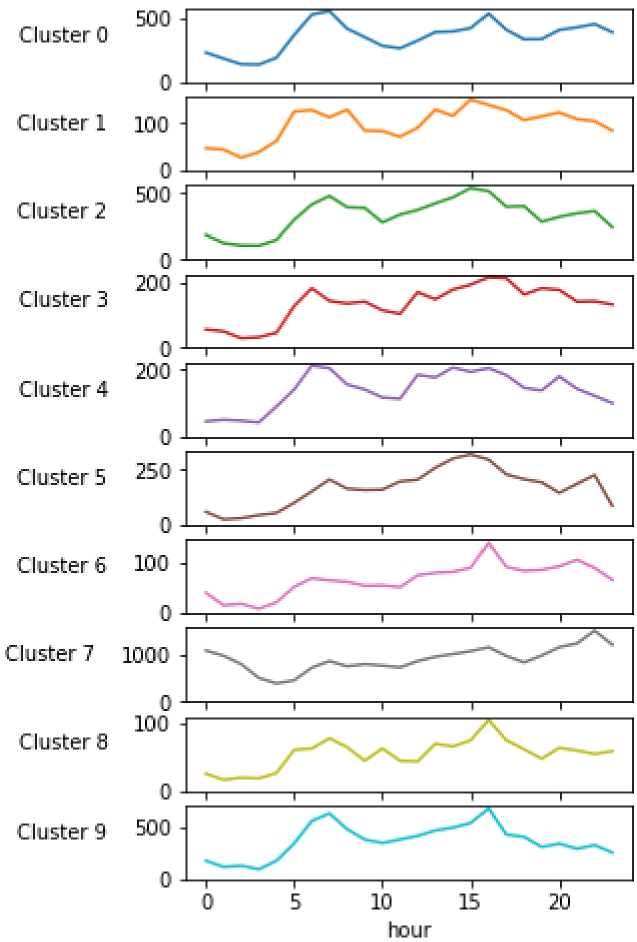



One idea is to check if after a ride the taxi driver will have to go back from the low-demand area back to high-demand, and if so, lower the value of this ride. I have skipped the implementation of this idea into the model this time around.

Once we have formed clusters, we can analyze the profiles of demand over hours of day. The graphs below show that the damand profiles somewhat differ across clusters meaning that some of them are business areas, some are residential, and of course the most important are PARTY SPOTS with bars and night clubs and thus the highest demand for taxi to drive tired but satisfied data scientists from Telliskivi to their beds on Friday evenings.

In [17]:

```
pd.options.mode.chained_assignment = None
fig, axs = plt.subplots(num_clusters, 1, figsize=(4, 8))
for cluster_id in range(num_clusters):
    df_cluster = df[df['cluster_id'] == cluster_id]
    df_cluster['day'] = df_cluster['start_time'].dt.hour
    counts_series = df_cluster.groupby(['day']).size()
    ax = axs[cluster_id]
    ax.plot(counts_series.index.to_numpy(), counts_series.to_numpy(),
            color=cluster_colors[cluster_id])
    ax.set_ylim([0, None])
    ax.set_ylabel(f"Cluster {cluster_id}" + 20*" ", rotation='horizontal')
plt.xlabel("hour");
```




The graphs above do not pay due respect to the days of week (of which Friday is the most important one), so it may make sense to introduce the graphs across the "hour of week" as below.

In [18]:

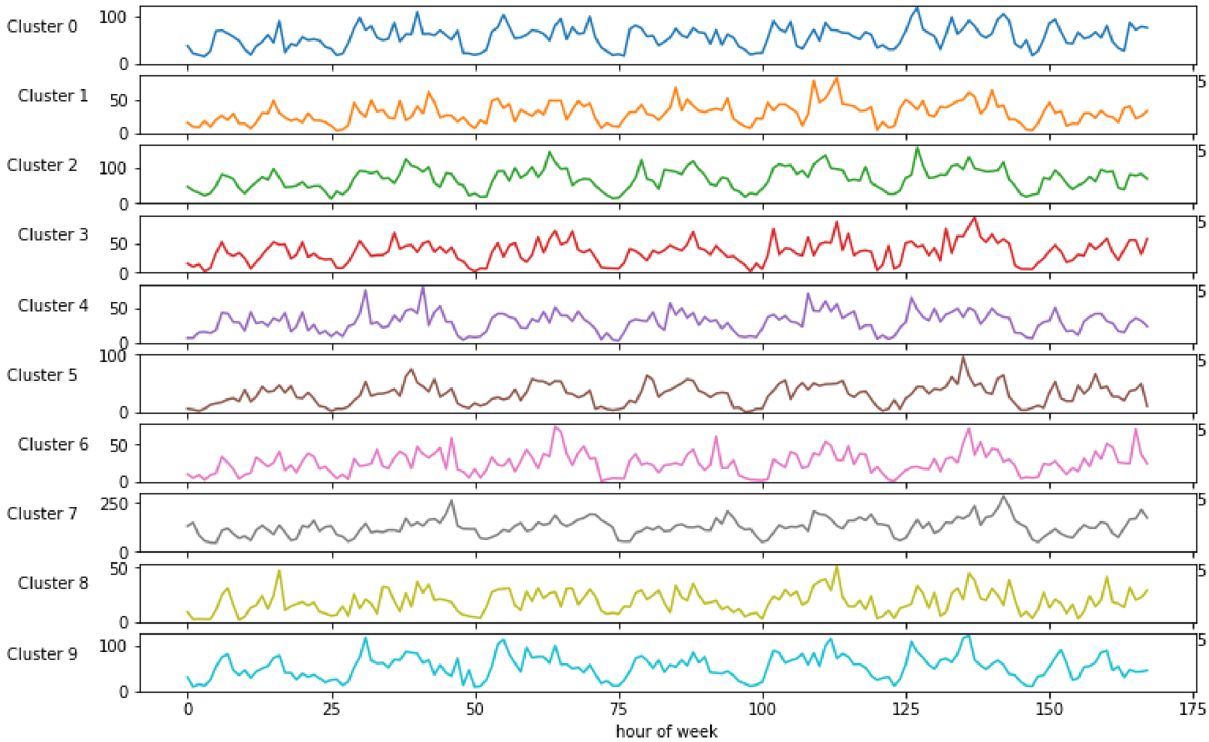
```
num_hours = 24
num_days = 7
num_hours_per_week = num_hours * num_days

def hour_of_week(ts):
    return ts.dt.dayofweek * num_hours + ts.dt.hour

df['hour_of_week'] = hour_of_week(df['start_time'])
```

In [42]:

```
df_count = df.groupby(['cluster_id', 'hour_of_week']).size()
df_value = df.groupby(['cluster_id', 'hour_of_week'])[['ride_value']].sum()
```

In [43]:

```
demand_mat = np.zeros((num_clusters, num_hours_per_week), dtype=np.float32)
fig, axs = plt.subplots(num_clusters, 1, figsize=(12, 8))
for cluster_id in range(num_clusters):
    cluster_value_df = df_value.loc[cluster_id]
    demand_values = cluster_value_df.to_numpy().squeeze(1)
    demand_index = cluster_value_df.index.to_numpy()
    demand_mat[cluster_id, demand_index] = demand_values
    ax = axs[cluster_id]
    ax.plot(demand_index, demand_values,
            color=cluster_colors[cluster_id])
    ax.set_ylim([0, None])
    ax.set_ylabel(f"Cluster {cluster_id}" + 20*" ", rotation='horizontal')
plt.xlabel("hour of week");
```

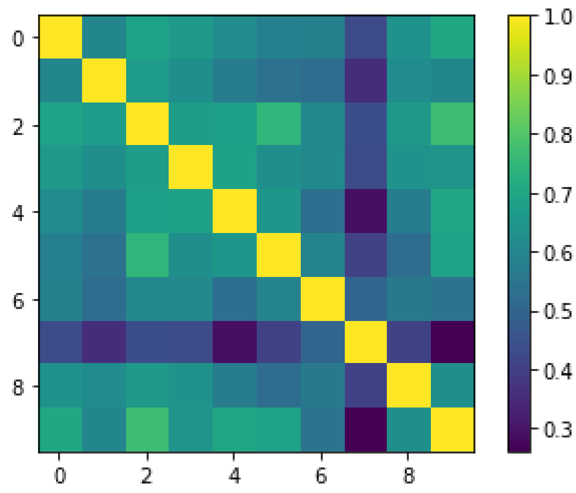



From the graphs above we are not sure how significant the geographical split into clusters is. Visual examination is not very reliable, so let's check the correlation matrix between different clusters.

In [41]:

```
demand_corr = np.corrcoef(demand_mat)
print(demand_corr.shape)
fig, ax = plt.subplots()
pos = ax.imshow(demand_corr, cmap='viridis')
fig.colorbar(pos, ax=ax)
np.set_printoptions(precision=2)
print(demand_corr)
```

```
(10, 10)
[[1.   0.6  0.69 0.66 0.62 0.58 0.58 0.43 0.64 0.7 ]
 [0.6  1.   0.67 0.63 0.57 0.54 0.53 0.36 0.62 0.6 ]
 [0.69 0.67 1.   0.67 0.68 0.75 0.61 0.44 0.66 0.77]
 [0.66 0.63 0.67 1.   0.68 0.63 0.61 0.43 0.64 0.64]
 [0.62 0.57 0.68 0.68 1.   0.65 0.53 0.29 0.57 0.7 ]
 [0.58 0.54 0.75 0.63 0.65 1.   0.59 0.41 0.53 0.69]
 [0.58 0.53 0.61 0.61 0.53 0.59 1.   0.5  0.56 0.55]
 [0.43 0.36 0.44 0.43 0.29 0.41 0.5  1.   0.4  0.26]
 [0.64 0.62 0.66 0.64 0.57 0.53 0.56 0.4  1.   0.63]
 [0.7  0.6  0.77 0.64 0.7  0.69 0.55 0.26 0.63 1.  ]]
```




Here we see that clusters 2 and 9 are highly correlated (0.77), and indeed they both belong to the same Kristiine district. And more obviously we can see that cluster 7 has the lowest correlation with other clusters and indeed the gray cluster is Kesklinn where we know that the morning influx and evening exodus of office workers happens.

We will go with the combination of ['cluster_id', 'hour_of_week'] to build the model.

There are two strategies to make a design choice:

1. Provides the best quality of service
2. Generates highest profit

The highest QoS is achieved by counting the number of orders regardless of their value. The highest profit is achieved by counting the total value of rides from some specific area, however this strategy can drop the QoS for areas with sparse cheap rides.

Basically the profit prediction model is as simple as a look-up table below:

In [46]:

```
df_value
```

Out[46]:

| | | ride_value |
|---|---|---|
| **cluster_id** | **hour_of_week** | |
| **0** | **0** | 37.005244 |
| | **1** | 21.434736 |
| | **2** | 18.315266 |
| | **3** | 14.868956 |
| | **4** | 26.932222 |
| **...** | **...** | ... |
| **9** | **163** | 30.421394 |
| | **164** | 47.029851 |
| | **165** | 42.198421 |
| | **166** | 42.807714 |
| | **167** | 45.544008 |

1675 rows × 1 columns

The model of the number of drivers that have to be there in the area is also a look-up table:

In [45]:

```
df_count
```

Out[45]:

```
cluster_id  hour_of_week
0           0               38
            1               23
            2               17
            3               17
            4               24
                            ..
9           163             29
            164             50
            165             44
            166             40
            167             47
Length: 1675, dtype: int64
```

To be able to AB test the proposal we could switch to this strategy alltogether for 1 day a week, so that the experiment would last for 7 weeks with days of week selected in a round-robin fashion, ex: Monday on week 1, Tuesday on week 2, Sunday on week 7. This would be a large-scale experiment. Probably there are ways to make it less intruding and faster.