Взаимодействие участников в шоу продакшена "Medium Quality" на платформе VK Видео

С начала 2023 года на платформе VK Видео публикуются многочисленные проекты продакшена "Medium Quality" (возглавляемый бывшим генеральным продюсером телеканала ТНТ). Распространено мнение, что продакшен задействует *небольшое* количество известных комиков в качестве гостей и ведущих в *большом* количестве шоу. В данной работе представлена визуализация и анализ такого взаимодействия между участниками проектов продакшена.

Большим подспорьем при сборе данных стал сайт Humorpedia, где перечислены проекты Medium Quality и плейлисты с эпизодами шоу. Так как информация на сайте была не полностью актуальной, недостающие плейлисты были добавлены в датасет на основе принадлежности к каналам, которые перечислены на Humorpedia.

Сбор даных проходил в два этапа: 1) формирование списка шоу вручную с опорой на сайт Humorpedia 2) скрейпинг эпизодов указанных шоу с помощью сапомисного web-scraper'a (ссылка на блокнот). У ВК есть собственный API, но извлечь нужные данные в этой ситуации с помощью библиотеки `requests` было проще, чем использовать официальный API. 3) извлечение из описаний шоу и эпизодов ведущих и их гостей с помощью spacy и вручную.

Данные в датасете актуальны на момент 2 декабря 2025 года и только для платформы ВК видео. Продакшен также производит проекты и на других платформах -- они в датасет не вошли.

Датасет состоит из двух листов:

- `data`: список эпизодов с указанием ведущих и гостей
  - `label` -- название лэйбла/канала (LABELSMART, Площадка, SlowSlowCow и др.)
  - `show_title` -- название шоу
  - `host_norm` -- ведущие
  - `link` -- ссылка на плейлист
  - `owner_id_y` -- id владельца плейлиста
  - `album_id` -- id плейлиста
  - `id` -- id видео
  - `title` -- название видео
  - `direct_url` -- ссылка на видео
  - `date` -- дата загрузки видео на платформу
  - `description` -- описание видео
  - `duration` -- длительность видео
  - `guests_full` -- список гостей
- `shows`: список шоу

- `label` -- название лэйбла/канала (LABELSMART, Площадка, SlowSlowCow и др.)
- `show_title` -- название шоу
- `host_norm` -- ведущие
- `link` -- ссылка на плейлист
- `owner_id` -- id владельца плейлиста
- `album_id` -- id плейлиста
- `videos_count` -- количество видео в плейлисте
- `status` -- предусмотрено ли участие "известных" гостей

Далее анализируется совместное появление только **"известных"**, публичных людей.

Из списка шоу не для всех получилось извлечь имена гостей, т.к. они не были указаны ни в заголовке, ни в описании видео, также в датасет вошли шоу, которые предположительно принадлежат продакшену, но опубликованы не на ВК видео, а на других платформах -- такие эпизоды не вошли в основной датасет.

В данной работе вершинами являются персоналии, а ребрами -- совместное появление в эпизоде. Граф является ненаправленным, гомогенным, взвешенным по количеству совместных появлений, или взаимодействий.

После обработки данных получившийся граф состоит из 1 371 вершин и 11 825 ребер.

Предобработка данных

```
import random
import numpy as np
import os
def seed_everything(seed_value):
    random.seed(seed_value)
    np.random.seed(seed_value)
    os.environ['PYTHONHASHSEED'] = str(seed_value)

seed = 42
seed_everything(seed)
```

Загружаем данные из гитхаб-репозитория

```
!wget
https://raw.githubusercontent.com/liminovna/HSE_SocialNetworks_noteboo
ks/main/final_project/mq_data.xlsx
```

```
--2025-12-25 20:26:01--
https://raw.githubusercontent.com/liminovna/HSE_SocialNetworks_noteboo
ks/main/final_project/mq_data.xlsx
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.108.133|:443... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 1446877 (1.4M) [application/octet-stream]
Saving to: ‘mq_data.xlsx.1’

mq_data.xlsx.1      100%[===================>]   1.38M  8.74MB/s    in
0.2s

2025-12-25 20:26:02 (8.74 MB/s) - ‘mq_data.xlsx.1’ saved
[1446877/1446877]
```

```
import pandas as pd
```

Извлекаем данные о шоу: из этого файла нам понадобится только название шоу `show_title` и колонка `status`, на основе которой мы будем избавляться от эпизодов, для которых не получилось определить "известных" участников

```
df_shows = pd.read_excel('mq_data.xlsx', sheet_name='shows',
engine='openpyxl')
df_shows.shape
```

```
(188, 9)
```

```
print('Всего шоу:', len(df_shows))
```

```
Всего шоу: 188
```

Избавляемся от шоу, где не получилось определить "известных" участников

```
# читаем данные об эпизодах (=основной датасет) в датафрейм
cols = ['label', 'show_title', 'host_norm',
       'link', 'album_id', 'videos_count', 'id',
       'owner_id_y', 'title', 'direct_url', 'adding_date', 'date',
       'description', 'duration', 'guests_full']
df = pd.read_excel('mq_data.xlsx', sheet_name='data',
engine='openpyxl')[cols]
df.shape
```

```
(3378, 15)
```

```
to_delete = df_shows[df_shows['status'].isna()]['show_title']
to_delete_df = df[df['show_title'].isin(to_delete)]
df.drop(to_delete_df.index, inplace=True)

# удаляем строки, где не указаны ни ведущие, ни гости
df.drop(df[(df['host_norm'].isna()) &
(df['guests_full'].isna())].index, inplace=True)

# удялаем шоу, для которых не получилось извлечь названия и описания
эпизодов
df.drop(df[df['videos_count']==0].index, inplace=True)
```

```
print('Всего шоу после удаления "лишних" строк из датасета:',
df['show_title'].nunique())
```

```
Всего шоу после удаления "лишних" строк из датасета: 168
```

```
import ast

def str2list(x):
    """
    Конвертировать строковое значение из поля guests_full в список.
    Если конвертация невозможна, распечатать это значение.
    """
    try:
        if x != 'нет':
            res = ast.literal_eval(x)
            return list(map(lambda s: s.strip(), res))
    except:
        print('error', x)

        return ['error']
```

Приводим значения в полях с ведущими и гостями к нижнему регистру

```
df['host_norm'] = df['host_norm'].str.lower()
df['guests_full'] = df['guests_full'].str.lower()
df['host_norm'] = df['host_norm'].str.split(',')
df['guests_full'] = df['guests_full'].fillna('нет').apply(str2list)

# df[['host_norm', 'guests_full']]
```

Разведочный анализ

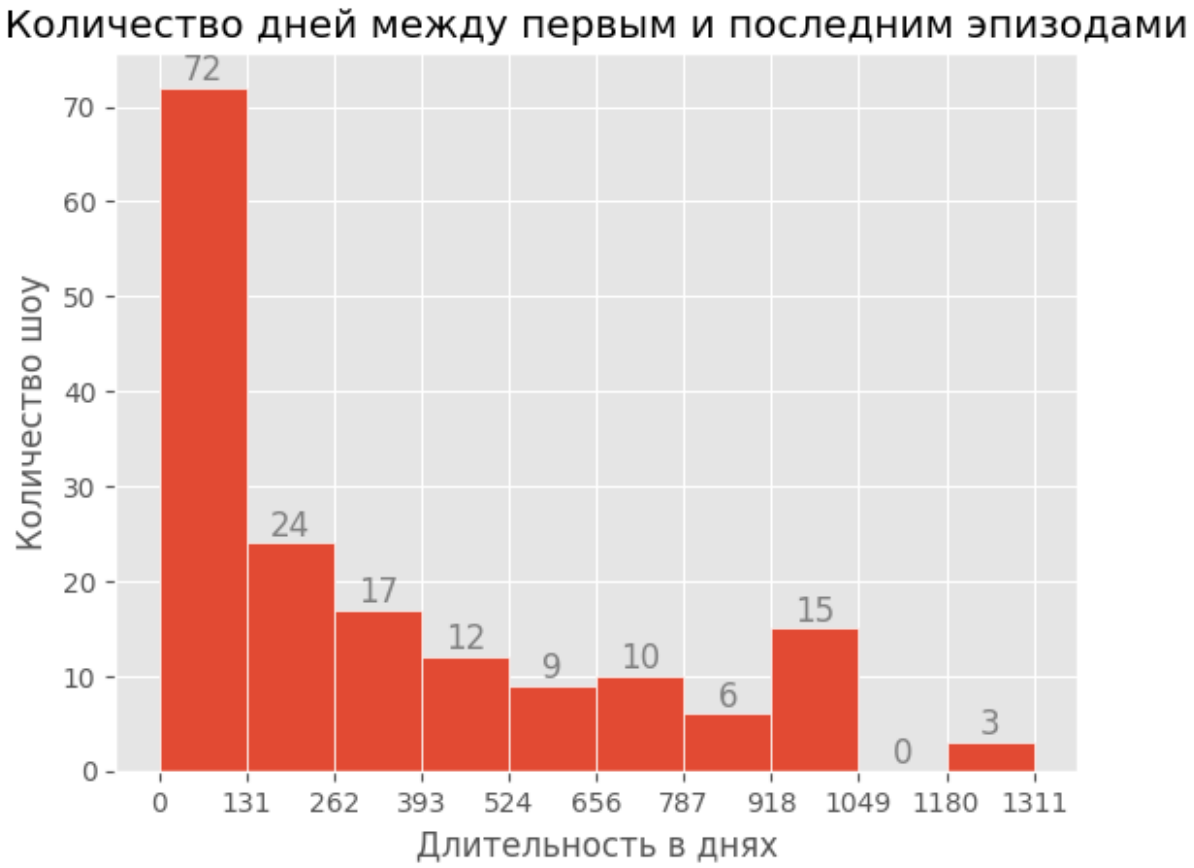
Активность каналов

Извлекаем дату загрузки видео

```
df['date'] = pd.to_datetime(df['date'],unit='s')
# df['date'].head()

pd.set_option('display.max_rows', None)
aggregation_dict = {
    'date': ['min', 'max'],
    'id': 'count'
}
stats = df.groupby('show_title').agg(aggregation_dict)
# stats

pd.reset_option("display.max_rows")
```

Визуализируем длительность шоу

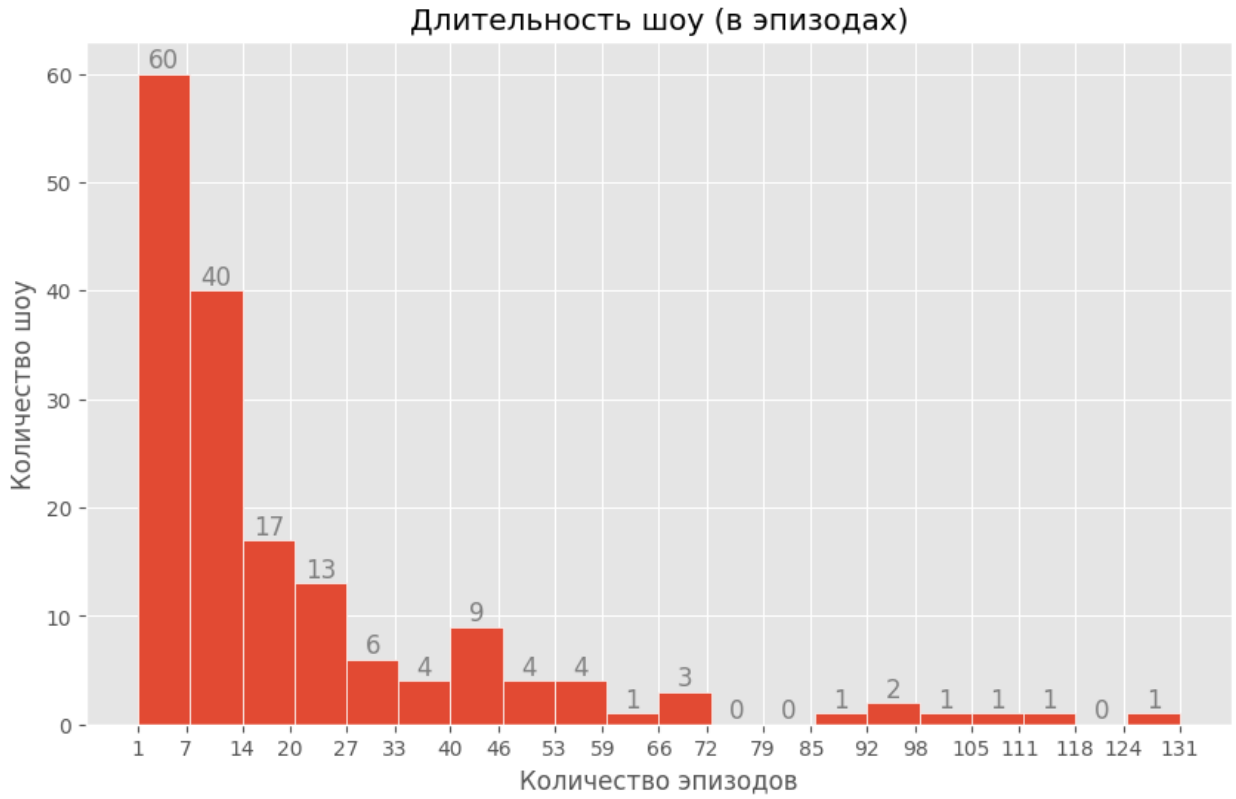
```
import matplotlib.pyplot as plt

plt.style.use('ggplot')

intervals = (stats['date']['max'] - stats['date']['min']).dt.days
values, bins, bars = plt.hist(intervals, bins=10, edgecolor='white')

plt.bar_label(bars, fontsize=12, color='grey')

plt.title('Количество дней между первым и последним эпизодами')
plt.xticks(np.linspace(intervals.min(), intervals.max(), 11))
plt.xlabel('Длительность в днях')
plt.ylabel('Количество шоу');
```



Визуализируем количество эпизодов

```
plt.figure(figsize=(10,6))
values, bins, bars = plt.hist(stats['id']['count'], bins=20,
edgecolor='white')

plt.bar_label(bars, fontsize=12, color='grey')

plt.title('Длительность шоу (в эпизодах)')
```

```
plt.xticks(np.linspace(stats['id']['count'].min(), stats['id']
['count'].max(), 21, dtype=int))
plt.xlabel('Количество эпизодов')
plt.ylabel('Количество шоу');
```




```
# когда выходили эпизоды
plt.figure(figsize=(18, 5))

values, bins, bars = plt.hist(df['date'].dropna().dt.round('D'),
bins=50, edgecolor='white')

plt.bar_label(bars, fontsize=12, color='grey')

plt.title('Загрузка видео по дням')
plt.xlabel('Дата загрузки видео')
plt.ylabel('Количество шоу')

plt.xticks(np.arange(min(df['date'].dt.date),max(df['date'].dt.date),
27))
plt.xticks(rotation=90);
```

В течение первых двух месяцеев 2023 года наблюдается пик по количеству загруженных видео.

Участники проектов

Извлекаем отдельных ведущих

```
hosts = df['host_norm'].dropna().explode().str.strip()
hosts = hosts.rename('name')
# hosts
```

Извлекаем отдельных гостей

```
guests = df['guests_full'].explode().str.strip()
guests = guests.rename('name')
# guests
```

Объединяем оба списка -- ведущих и гостей -- в сет

```
all_pers = list(set(hosts.dropna()) | set(guests.dropna()))
print('Количество уникальных участников:', len(all_pers))
```

```
Количество уникальных участников: 1381
```

Объединяем в одну последовательность, в которой считаем количество раз, человек участвовал в эпизодах в качестве ведущего или гостя

```
partic = pd.merge(left=hosts.value_counts().reset_index(name='count'),
right=guests.value_counts().reset_index(name='count'), on='name',
how='outer')
partic.columns = ['name', 'as_host', 'as_guest']
partic = partic.set_index('name')
# partic
```

```
partic['sum'] = partic.sum(axis=1)
# partic
```

Проверяем, что все посчиталось правильно

```
assert partic.loc['зоя яровицына']['sum'] == hosts.value_counts()['зоя яровицына'] + guests.value_counts()['зоя яровицына']
```

Топ-20 "популярных" участников

```
# partic.sort_values(by='sum', ascending=False).head(20).style.background_gradient(axis=0, cmap='YlOrRd')

THRESHOLD = 10 # количество эпизодов

print(f'Снимались в менее, чем {THRESHOLD} эпизодах:\t', str(round(len(partic[partic['sum']<THRESHOLD]) / len(partic) * 100, 2)) + '%')

print(f'Снимались в {THRESHOLD} и более эпизодах:\t', str(round(len(partic[partic['sum']>=THRESHOLD])/ len(partic) * 100, 2)) + '%')
```

```
Снимались в менее, чем 10 эпизодах:    85.37%
Снимались в 10 и более эпизодах:  14.63%
```

Только 15% от общего числа участников снимались в более, чем 10 эпизодах.

Посмотрим, кого чаще всего приглашают в качестве гостей на пилотные выпуски

```
firsts = df.sort_values(by='date').drop_duplicates(subset='show_title', keep='first')
print(firsts['guests_full'].explode().value_counts()[:20])
```

```
guests_full
игорь джабраилов      15
павел дедищев         12
расул чабдаров        11
александр ваш          9
иван абрамов           9
зоя яровицына          8
антон шастун           8
андрей бебуришвили     7
денис дорохов          7
гурам амарян           7
евгений чебатков       6
варвара щербакова      6
```

```
алексей сапрыкин        5
роман косицын           5
азамат мусагалиев       4
гарик харламов          4
артур бабич             4
артем винокур           4
дмитрий позов           4
алексей шальнов         4
Name: count, dtype: int64
```

Анализ социальной сети

Далее в качестве вершин выступают отдельно взятые персоналии, а в качестве ребер -- совместное появление в эпизоде, независимо от роли.

Общее описание сети

Подготавливаем список ребер

```
edges = df[['host_norm', 'guests_full']]
# edges

# разделяем списки ведущих и гостей горизонтально, чтобы в одной
строке в отдельных столбцах содержались все участники
left = edges['host_norm'].fillna('').apply(lambda x:
','.join(x)).str.split(',', expand=True)
right = edges['guests_full'].fillna('').apply(lambda x:
','.join(x)).str.split(',', expand=True)

t = pd.concat([left, right], axis=1).replace(np.nan, 0)
t['combined_list'] = t.apply(lambda row: set(row.values), axis=1)
# t

# составляем список парных взаимодействий -- ребер
import itertools
edges = list()
for idx, row in t.iterrows():
    edges.extend(list(itertools.combinations(list(filter(lambda x: x
not in ('', 0), row['combined_list'])), 2)))

from collections import Counter

interactions = Counter(edges)

print('Количество парных взаимодействий (в т.ч. повторных): ',
len(edges))
print('Количество уникальных пар: ', len(interactions))
```

```
Количество парных взаимодействий (в т.ч. повторных):  31685
Количество уникальных пар:  12152
```

Составляем список уникальных пар (ребер)

```
import networkx as nx

interactions = pd.DataFrame(interactions.items(), columns=[0,
'count'])
# interactions

interactions[['left', 'right']] = interactions[0].apply(pd.Series)
interactions = interactions.drop(columns=0)
# interactions
```

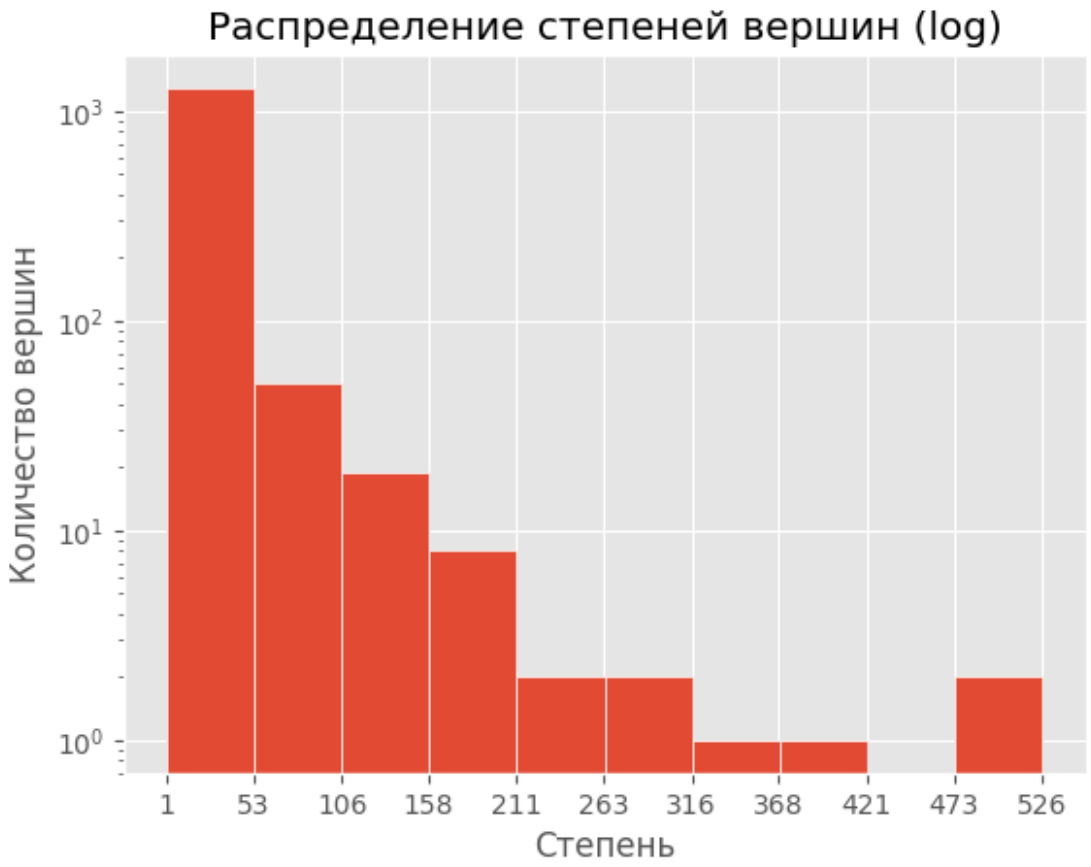
Создаем граф

```
G = nx.from_pandas_edgelist(
    df=interactions,
    source='left',
    target='right',
    edge_attr=['count'],
    # create_using=nx.MultiDiGraph,
)
print(G)

Graph with 1374 nodes and 11825 edges
```

Граф состоит из 1 374 вершин и 11 825 ребер.

Распределение степеней вершин (Degree distribution)

```
degree_seq = [degree for (node, degree) in G.degree]
bins, freq = np.unique(degree_seq, return_counts=True)
# plt.bar(bins, freq, align='center', edgecolor='white')
plt.hist(degree_seq, edgecolor='white', align='mid', bins=10,
log=True)
plt.xticks(np.linspace(np.min(degree_seq), np.max(degree_seq), 11,
dtype='int'))
plt.xlabel('Степень')
plt.ylabel('Количество вершин')
plt.title('Распределение степеней вершин (log)');
```




```
print('Минимальная степень вершины (взвешенная):', np.min(degree_seq))
print('Максимальная степень вершины (взвешенная):',
np.max(degree_seq))
```

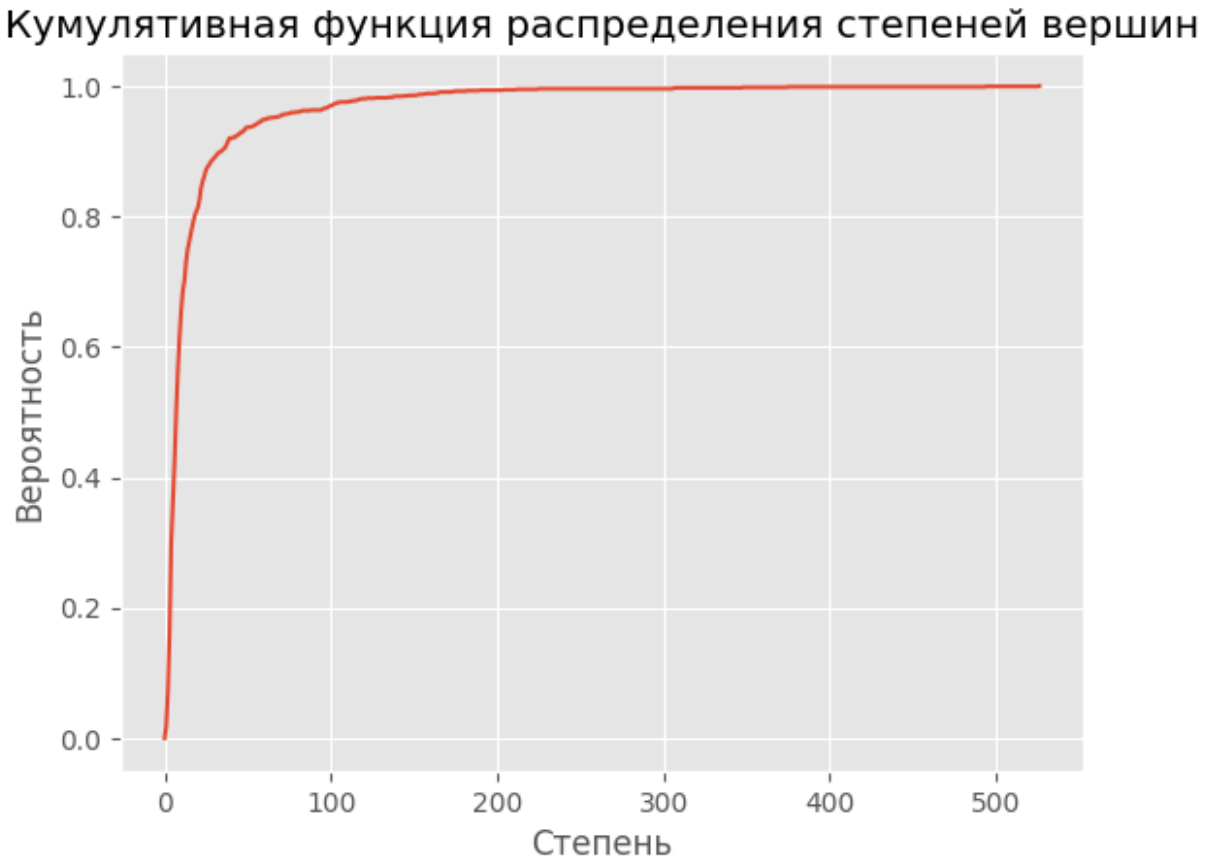
```
Минимальная степень вершины (взвешенная): 1
Максимальная степень вершины (взвешенная): 526
```

Большинство вершин имеет малую степень.

Проверим, является ли сеть scale-free.

```
def empirical_cdf(g: nx.Graph):
    """
    Вычислить кумулятивную функцию распределения (CDF)
    """
    hist = nx.degree_histogram(g)
    return np.cumsum(hist) / np.sum(hist)

ecdf = empirical_cdf(G)

plt.plot(ecdf)
plt.xlabel('Степень')
plt.ylabel('Вероятность')
```

```
plt.title('Кумулятивная функция распределения степеней вершин')
plt.show();
```

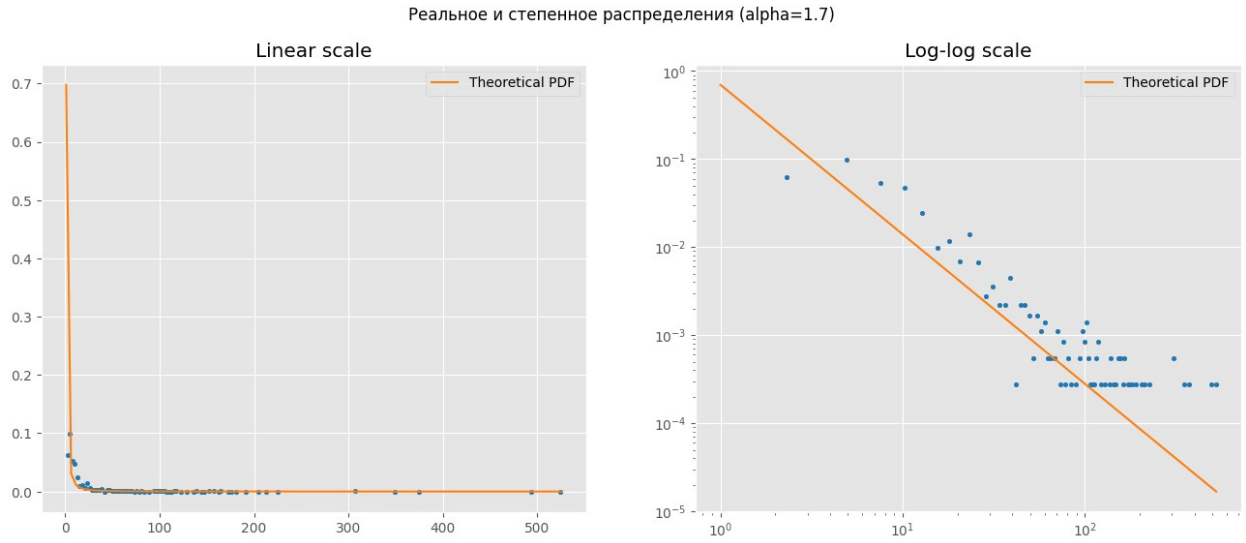



```
# Находим alpha
from sklearn.linear_model import LinearRegression

def alpha_lin_bins(x_train, bins):
    """
    Вычислить параметр alpha для степенного распределения
    """
    hist, bin_edges = np.histogram(x_train, bins=bins, density=True)
    bin_centers = (bin_edges[1:] + bin_edges[:-1]) / 2

    idx = np.argwhere(hist > 0)

    log_centers = np.log(bin_centers[idx][:,0]).reshape(-1, 1)
    log_hist = np.log(hist[idx])

    lr = LinearRegression().fit(log_centers, log_hist)
    return -lr.coef_[0][0]
```

```
alpha = alpha_lin_bins(degree_seq, bins)
print('Alpha =', alpha);
```

```
Alpha = 1.6977387131536303
```

```
def power_law_pdf(x, alpha=3.5, x_min=1):
    C = (alpha - 1) / x_min ** (1 - alpha)
    return C * x ** (-alpha)

x_min = 1

x_space = np.linspace(min(degree_seq), max(degree_seq), 100)

fig, ax = plt.subplots(1,2, figsize=(16,6))

plt.suptitle(f'Реальное и степенное распределения
(alpha={round(alpha,2)})')

hist, bin_edges = np.histogram(degree_seq, bins=200, density=True)
bin_centers = (bin_edges[1:] + bin_edges[:-1]) / 2
ax[0].scatter(bin_centers[hist > 0], hist[hist > 0], s=10,
c="tab:blue")
ax[0].plot(x_space, power_law_pdf(x_space, alpha, x_min),
label='Theoretical PDF', c='tab:orange')
ax[0].legend()
ax[0].set_title('Linear scale')

hist, bin_edges = np.histogram(degree_seq, bins=200, density=True)
bin_centers = (bin_edges[1:] + bin_edges[:-1]) / 2
ax[1].scatter(bin_centers, hist, s=10, c="tab:blue")
ax[1].plot(x_space, power_law_pdf(x_space, alpha, x_min),
         label='Theoretical PDF', c='tab:orange')
ax[1].legend()
ax[1].set_xscale('log')
ax[1].set_yscale('log')
ax[1].set_title('Log-log scale')

plt.show();
```



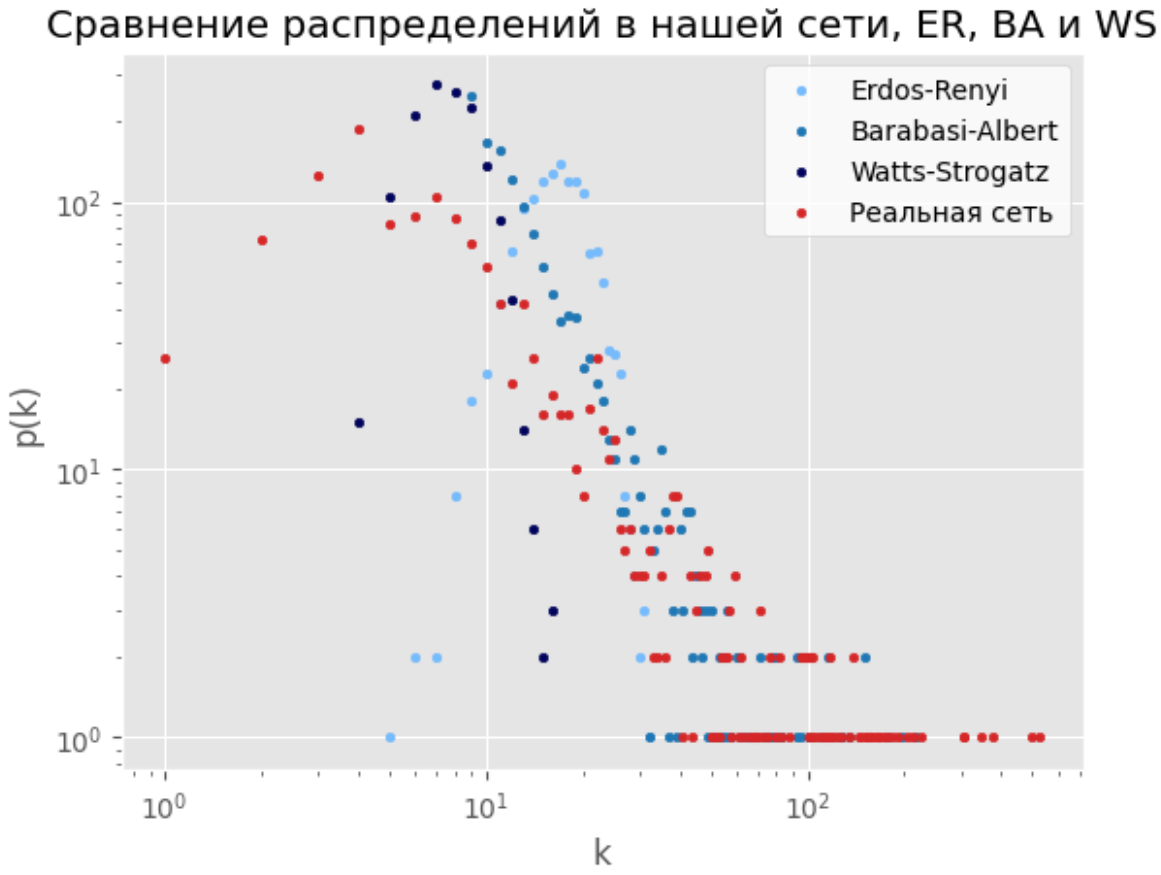
Из графиков видим, что наша сеть -- scale-free, т.к. степени вершин следуют степенному распределению.

Сравним наш граф с RE, BA и WS

```
# ER
def random_from_real(graph):

    n = len(graph.nodes)
    mean_degree = np.mean(list(dict(graph.degree).values()))
    p = mean_degree / (n - 1)
    return nx.fast_gnp_random_graph(n, p)

ER = random_from_real(G)
degree_hist_ER = np.array(nx.degree_histogram(ER))
idx = np.argwhere(degree_hist_ER > 0)
plt.scatter(idx, degree_hist_ER[idx], s=10, label='Erdos-Renyi',
c='xkcd:sky blue')


# BA
def estimate_m_from_graph(G):
    #
https://www.math.nagoya-u.ac.jp/~richard/teaching/s2024/SML_Adam_2.pdf
    N = G.number_of_nodes()
    E = G.number_of_edges()
    m_hat = E/N
    return round(m_hat)

BA = nx.barabasi_albert_graph(n=len(G.nodes),
m=estimate_m_from_graph(G)) #
degree_hist_BA = np.array(nx.degree_histogram(BA))
idx = np.argwhere(degree_hist_BA > 0)
```

```
plt.scatter(idx, degree_hist_BA[idx], s=10, label='Barabasi-Albert',
c='tab:blue')


# WS
def estimate_ws_params(degrees, tol=1e-5, max_iter=100):
    n = len(degrees)
    mean_deg = np.mean(degrees)
    K = mean_deg / 2
    var = np.var(degrees, ddof=1)
    p_hat = binary_search_p(degrees, K, tol, max_iter)
    return n, K, p_hat

def binary_search_p(degrees, K, tol, max_iter):
    pl, pr = 0.0, 1.0
    for _ in range(max_iter):
        p = (pl + pr) / 2
        model_var = theoretical_ws_var(degrees, K, p)
        if abs(model_var - np.var(degrees, ddof=1)) < tol:
            return p
        if model_var > np.var(degrees, ddof=1):
            pr = p
        else:
            pl = p
    return p

def theoretical_ws_var(degrees, K, p):
    n = len(degrees)
    alpha = 0.5 * p * (n - 1 - 2 * K) / n
    beta = p / n
    mean = 2 * K
    second = 2 * K * (1 - 2 * alpha + 2 * beta)
    variance = second - mean**2
    return max(variance, 0)
n, K, p = estimate_ws_params(list(dict(G.degree).values()))

WS = nx.watts_strogatz_graph(n, int(K), p)
degree_hist_WS = np.array(nx.degree_histogram(WS))
idx = np.argwhere(degree_hist_WS > 0)
plt.scatter(idx, degree_hist_WS[idx], s=10, label='Watts-Strogatz',
c='xkcd:dark blue')

# наша сеть
degree_hist = np.array(nx.degree_histogram(G))
idx = np.argwhere(degree_hist > 0)
plt.scatter(idx, degree_hist[idx], s=10, label='Реальная сеть',
c='tab:red')


plt.legend(facecolor="white")
```

```
plt.xlabel('k')
plt.ylabel('p(k)')
plt.title('Сравнение распределений в нашей сети, ER, BA и WS')
plt.xscale('log')
plt.yscale('log')
plt.show();
```



Наше распределение похоже на модель Barabasi-Albert и значительно отличается от моделей Erdos-Renyi и Watts-Strogatz.

Среднее расстояние (Average distance) и коэффициент кластеризации (Clustering coefficient)

Поскольку в сети есть вершины, между которыми нет ни прямой, ни опосредованной связи до некоторых других, для вычисления статистик выберем наибольший соединенный подграф

```
def select_largest_component(g: nx.Graph) -> nx.Graph:
    nodes = max(nx.connected_components(g), key=len)
    return g.subgraph(nodes)

G_sub = select_largest_component(G)
```

```
print('Вершин в подграфе:', len(G_sub.nodes()))
print('Ребер в подграфе:', len(G_sub.edges()))
```

```
Вершин в подграфе: 1358
Ребер в подграфе: 11801
```

```
c = {}
l = {}

l_ = nx.average_shortest_path_length(G_sub)
c_ = nx.average_clustering(G_sub)

c['Реальная сеть'] = c_
l['Реальная сеть'] = l_

print('Радиус: ', nx.radius(G_sub))
print('Диаметр: ', nx.diameter(G_sub))
print('Средняя длина кратчайшего пути: ', l_)
print('Срединй коэффициент кластеризации: ', c_)
```

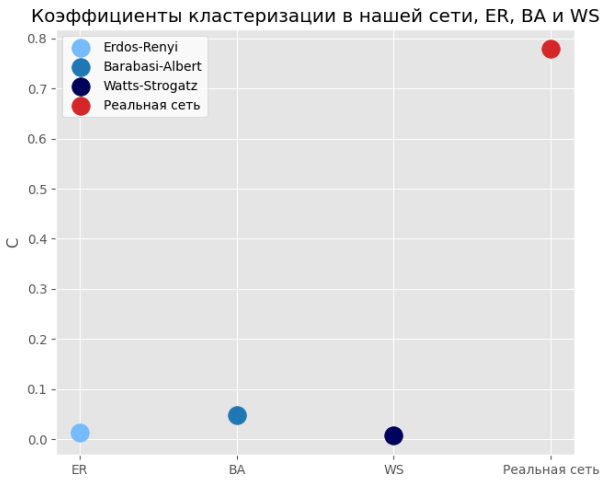
```
Радиус:  3
Диаметр:  6
Средняя длина кратчайшего пути:  2.6446006796157597
Срединй коэффициент кластеризации:  0.7787389396531752
```

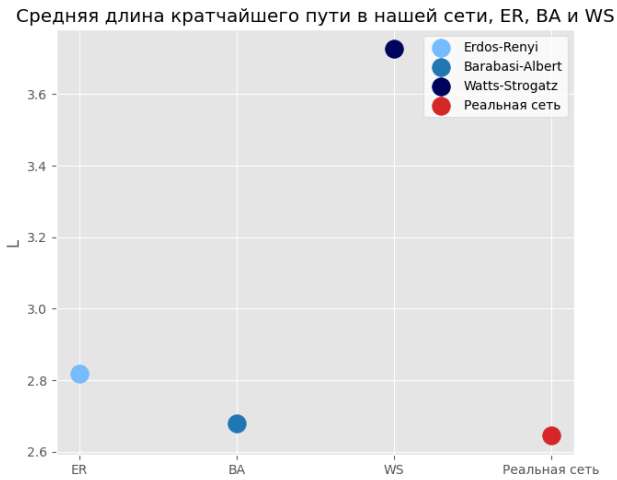
Расчитываем коэффициенты кластеризации и длины кратчайшего пути для трех моделей -- ER, BA, WS

```
for m_ in ['ER', 'BA', 'WS']:
    m = eval(m_)
    # print('Радиус: ', nx.radius(m))
    # print('Диаметр: ', nx.diameter(m))
    c_ = nx.average_clustering(m)
    l_ = nx.average_shortest_path_length(m)

    c[m_] = c_
    l[m_] = l_
    print(f'Средняя длина кратчайшего пути в {m_}: ', l_)
    print(f'Срединй коэффициент кластеризации в {m_}: ', c_)
    print()
```

```
Средняя длина кратчайшего пути в ER:  2.81866650552186
Срединй коэффициент кластеризации в ER:  0.012778473027273534

Средняя длина кратчайшего пути в BA:  2.6783401236786393
Срединй коэффициент кластеризации в BA:  0.04798587159725889

Средняя длина кратчайшего пути в WS:  3.7267864014986465
Срединй коэффициент кластеризации в WS:  0.006942192531712182
```

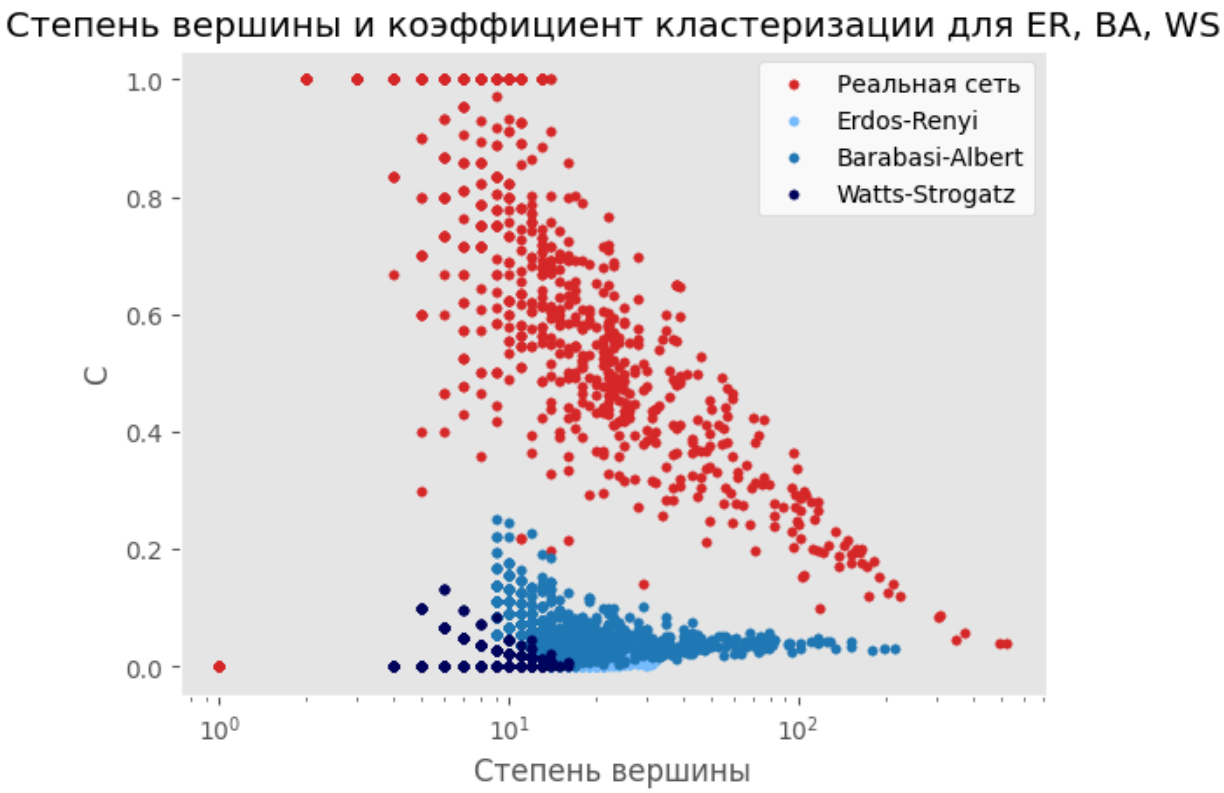
```
fix, ax = plt.subplots(1,2, figsize=(16,6))

ax[0].scatter('ER', c['ER'], s=200, label='Erdos-Renyi', c='xkcd:sky
blue')
ax[0].scatter('BA', c['BA'], s=200, label='Barabasi-Albert',
c='tab:blue')
ax[0].scatter('WS', c['WS'], s=200, label='Watts-Strogatz',
c='xkcd:dark blue')
ax[0].scatter('Реальная сеть', c['Реальная сеть'], s=200, c='tab:red',
label='Реальная сеть')

ax[0].legend(facecolor="white")
# plt.xlabel('k')
ax[0].set_ylabel('C')
ax[0].set_title('Коэффициенты кластеризации в нашей сети, ER, BA и
WS')



ax[1].scatter('ER', l['ER'], s=200, label='Erdos-Renyi', c='xkcd:sky
blue')
ax[1].scatter('BA', l['BA'], s=200, label='Barabasi-Albert',
c='tab:blue')
ax[1].scatter('WS', l['WS'], s=200, label='Watts-Strogatz',
c='xkcd:dark blue')
ax[1].scatter('Реальная сеть', l['Реальная сеть'], s=200, c='tab:red',
label='Реальная сеть')

ax[1].legend(facecolor="white")
# plt.xlabel('k')
ax[1].set_ylabel('L')
ax[1].set_title('Средняя длина кратчайшего пути в нашей сети, ER, BA и
WS')


plt.show();
```





Нашу сеть можно назвать small-world, поскольку ей свойственна высокая степень кластеризации и короткие расстояния между вершинами.

```
def node_degree_clustering(graph):
    return np.array(list(dict(graph.degree).values())),
np.array(list(dict(nx.clustering(graph)).values()))

degree, clustering = node_degree_clustering(G_sub)
plt.scatter(degree, clustering, s=15, linewidths=0.3, c='tab:red',
label='Реальная сеть')

degree_ER, clustering_ER = node_degree_clustering(ER)
plt.scatter(degree_ER, clustering_ER, s=15, linewidths=0.3,
label='Erdos-Renyi', c='xkcd:sky blue')

degree_BA, clustering_BA = node_degree_clustering(BA)
plt.scatter(degree_BA, clustering_BA, s=15, linewidths=0.3,
label='Barabasi-Albert', c='tab:blue')

degree_WS, clustering_WS = node_degree_clustering(WS)
plt.scatter(degree_WS, clustering_WS, s=15, linewidths=0.3,
label='Watts-Strogatz', c='xkcd:dark blue')

plt.xlabel('Степень вершины')
plt.ylabel('C')
plt.title('Степень вершины и коэффициент кластеризации для ER, BA,
WS')
plt.grid()
plt.legend(facecolor="white")

plt.xscale('log')
# plt.yscale('log')

plt.show();
```



Чем выше степень вершины, тем меньше коэффициент кластеризации

Определение хабов

Хабами мы будем считать вершины, степень которых выше 90го перцентиля.

```
top_degree =
pd.Series(dict(G.degree(weight='count')).values()).quantile(0.90,
interpolation="nearest")
print('Минимально возможная степень хаба:', top_degree)
```

```
Минимально возможная степень хаба: 88
```

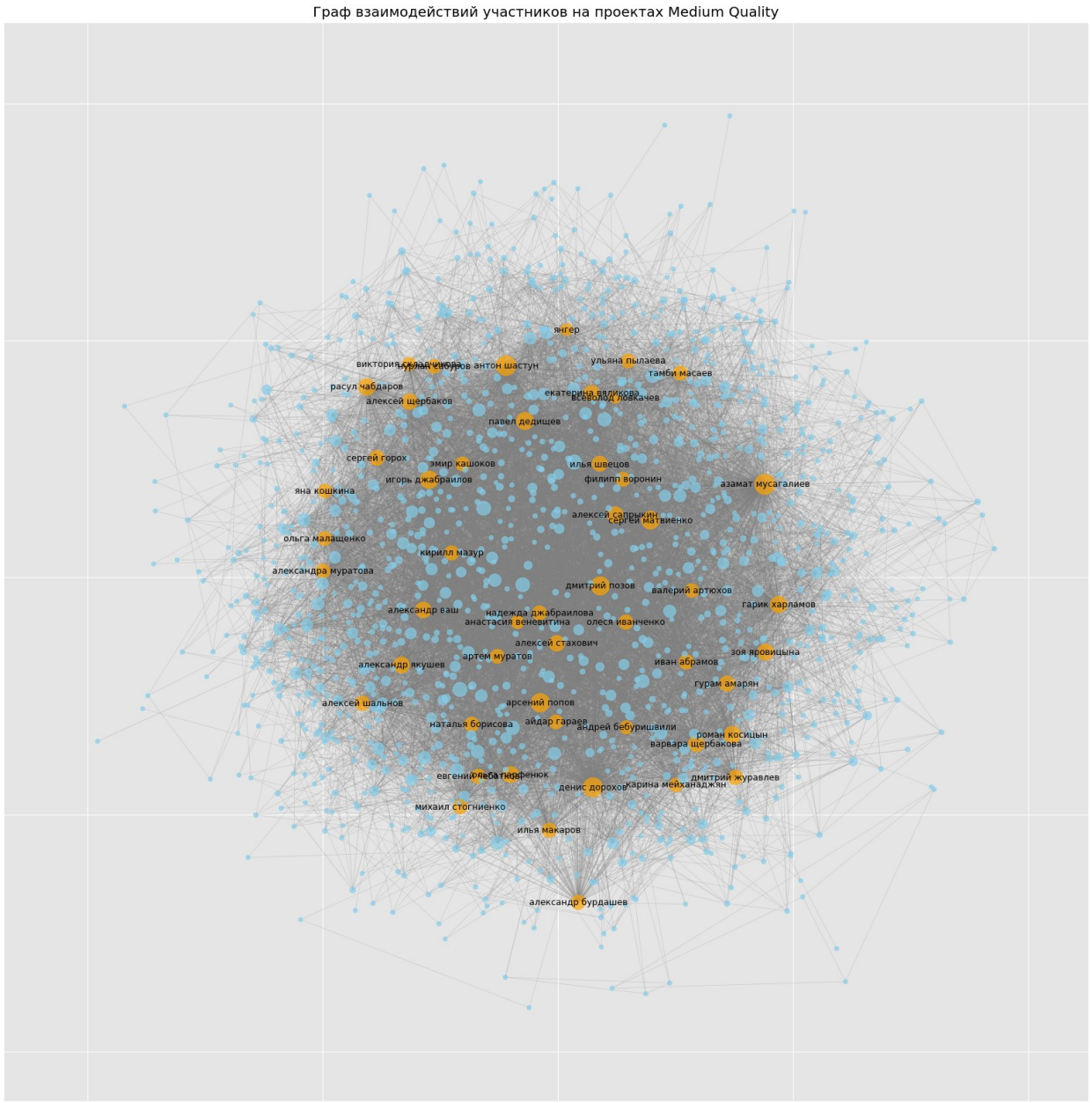
```
# THRESHOLD = 100 # @param
{"type":"slider","min":0,"max":600,"step":10}

hubs = list(filter(lambda p: G.degree[p] > top_degree,
dict(G.degree(weight='count')).keys()))
print('Количество хабов:', len(hubs))
```

```
Количество хабов: 51
```

Визуализация

```
# на визуализации будут отражены и выделены цветом только имена
"хабов"
```

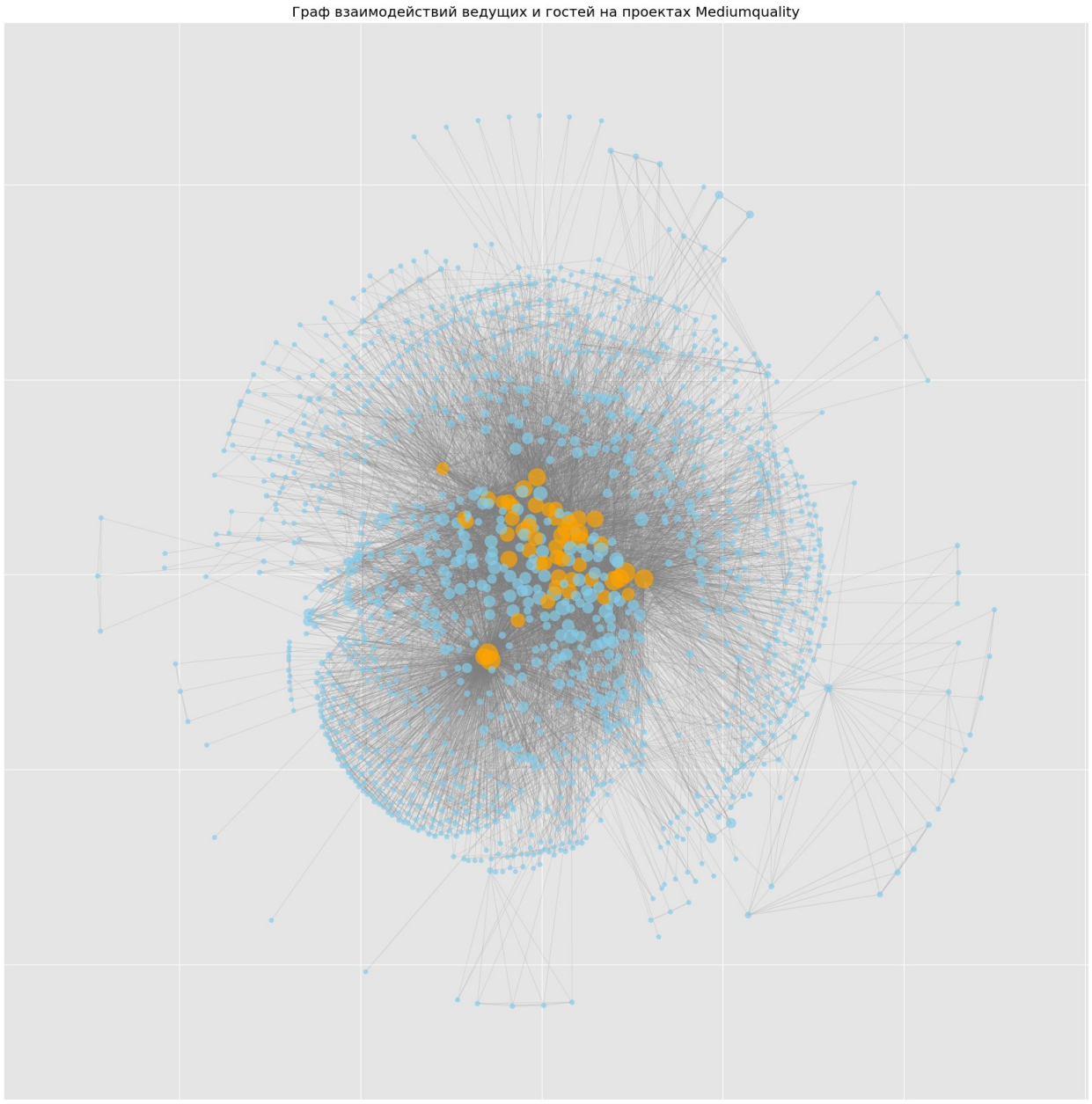
```
labels = {}
color_map = []
for node in G.nodes():
    if node in hubs:
        labels[node] = node
        color_map.append('orange')
    else:
        color_map.append('skyblue')

pos = nx.forceatlas2_layout(G, scaling_ratio=200)

plt.figure(figsize=(20,20))

nx.draw_networkx_nodes(G, pos, node_size=[np.log(x)**3 + 20 for x in
dict(G.degree(weight='count')).values()], alpha=0.7,
node_color=color_map)
nx.draw_networkx_edges(G, pos, edge_color='gray', width=[(np.log(x)
+2)/5 for x in list(nx.get_edge_attributes(G, 'count').values())],
alpha=0.3)
nx.draw_networkx_labels(G, pos,labels,font_size=9,font_color='black')

plt.title("Граф взаимодействий участников на проектах Medium Quality")
plt.show();
```



Другой способ визуализации

```
pos = nx.kamada_kawai_layout(G, scale=5)

plt.figure(figsize=(20,20))

nx.draw_networkx_nodes(G, pos, node_size=[np.log(x)**3 + 20 for x in
dict(G.degree(weight='count')).values()], alpha=0.7,
node_color=color_map)
nx.draw_networkx_edges(G, pos, edge_color='gray', width=[(np.log(x)
+2)/5 for x in list(nx.get_edge_attributes(G, 'count').values())],
alpha=0.3)
```

```
# nx.draw_networkx_labels(G,
pos,labels,font_size=9,font_color='black')

plt.title("Граф взаимодействий ведущих и гостей на проектах
Mediumquality")
plt.show();
```



Так как вершин и рбеер слишком много, визуализировать граф статично на плоскости недостаточно. Создадим интерактивную сеть с помощью библиотеки ploty.

Визуализация в plotly

```
# pos = nx.forceatlas2_layout(G, scaling_ratio=200)
pos = nx.kamada_kawai_layout(G, scale=5)

import plotly.graph_objects as go


edge_x = []
edge_y = []
for edge in G.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_x.append(x0)
    edge_x.append(x1)
    edge_x.append(None)
    edge_y.append(y0)
    edge_y.append(y1)
    edge_y.append(None)

edge_trace = go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=1, color='rgba(0.5, 0.5, 0.5, 0.3)'),
    hoverinfo='none',
    mode='lines')

node_x = []
node_y = []
for node in G.nodes():
    x, y = pos[node]
    node_x.append(x)
    node_y.append(y)

node_trace = go.Scatter(
    x=node_x, y=node_y,
    mode='markers',
    hoverinfo='text',
    marker=dict(
        showscale=True,
        # colorscale options
        #'Greys' | 'YlGnBu' | 'Greens' | 'YlOrRd' | 'Bluered' | 'RdBu'
|
        #'Reds' | 'Blues' | 'Picnic' | 'Rainbow' | 'Portland' | 'Jet'
|
        #'Hot' | 'Blackbody' | 'Earth' | 'Electric' | 'Viridis' |
        colorscale='YlGnBu',
        reversescale=True,
        color=[],
        size=5,
        colorbar=dict(
            thickness=15,
```

```
            title=dict(
              text='Node Connections',
              side='right'
            ),
            xanchor='left',
        ),
        line_width=2))

node_adjacencies = []
node_text = []
# for node, adjacencies in enumerate(G.adjacency()):
#     node_adjacencies.append(np.log(len(adjacencies[1])))
#     # node_text.append('# of connections:
'+str(len(adjacencies[1])))
#     node_text.append(adjacencies[0])

for p, d in dict(G.degree(weight='count')).items():
    node_adjacencies.append(np.log(d)**2)
    node_text.append(p)

node_trace.marker.color = node_adjacencies
node_trace.text = node_text
node_trace.marker.size = node_adjacencies

import plotly.graph_objects as go

fig = go.Figure(data=[edge_trace, node_trace],
             layout=go.Layout(
                title=dict(
                    text="<br>Граф взаимодействий ведущих и гостей на
проектах Medium Quality",
                    font=dict(
                        size=16
                    ),
                ),
                height=1200,
                width=1600,
                showlegend=False,
                hovermode='closest',
                margin=dict(b=20,l=5,r=5,t=40),
                annotations=[ dict(
                    showarrow=True,
                    xref="paper", yref="paper",
                    x=0.005, y=-0.002 ) ],
                xaxis=dict(showgrid=False, zeroline=False,
showticklabels=False),
                yaxis=dict(showgrid=False, zeroline=False,
showticklabels=False))
                )
fig.show();
```

```
fig.write_html("plotly_network.html")
```

Структурный анализ

Сравнение с моделелями ER, BA, WS

Применяем тест Колмогорова-Смирнова, для сравнения с тремя моделями.

```
from scipy.stats import ks_2samp

ER_degree_seq = [degree for (node, degree) in ER.degree]
print('p-value (ER) =', round(ks_2samp(degree_seq,
ER_degree_seq).pvalue,4))

BA_degree_seq = [degree for (node, degree) in BA.degree]
print('p-value (BA) =', round(ks_2samp(degree_seq,
BA_degree_seq).pvalue,4))

WS_degree_seq = [degree for (node, degree) in WS.degree]
print('p-value (WS) =', round(ks_2samp(degree_seq,
WS_degree_seq).pvalue,4))
```

```
p-value (ER) = 0.0
p-value (BA) = 0.0
p-value (WS) = 0.0
```

Как мы заметили выше, визуально наша сеть в значительной степени похожа на модель Barabasi-Albert, однако тест Колмогорова-Смирнова говорит о том, что она имеет распределение степеней, отличное от всех трех моделей при уровне значимости 0.05.

Центральность (Centrality measures)

Рассмотрим меры центральности нашей сети и выделим персоналии с наивысшимы значениями

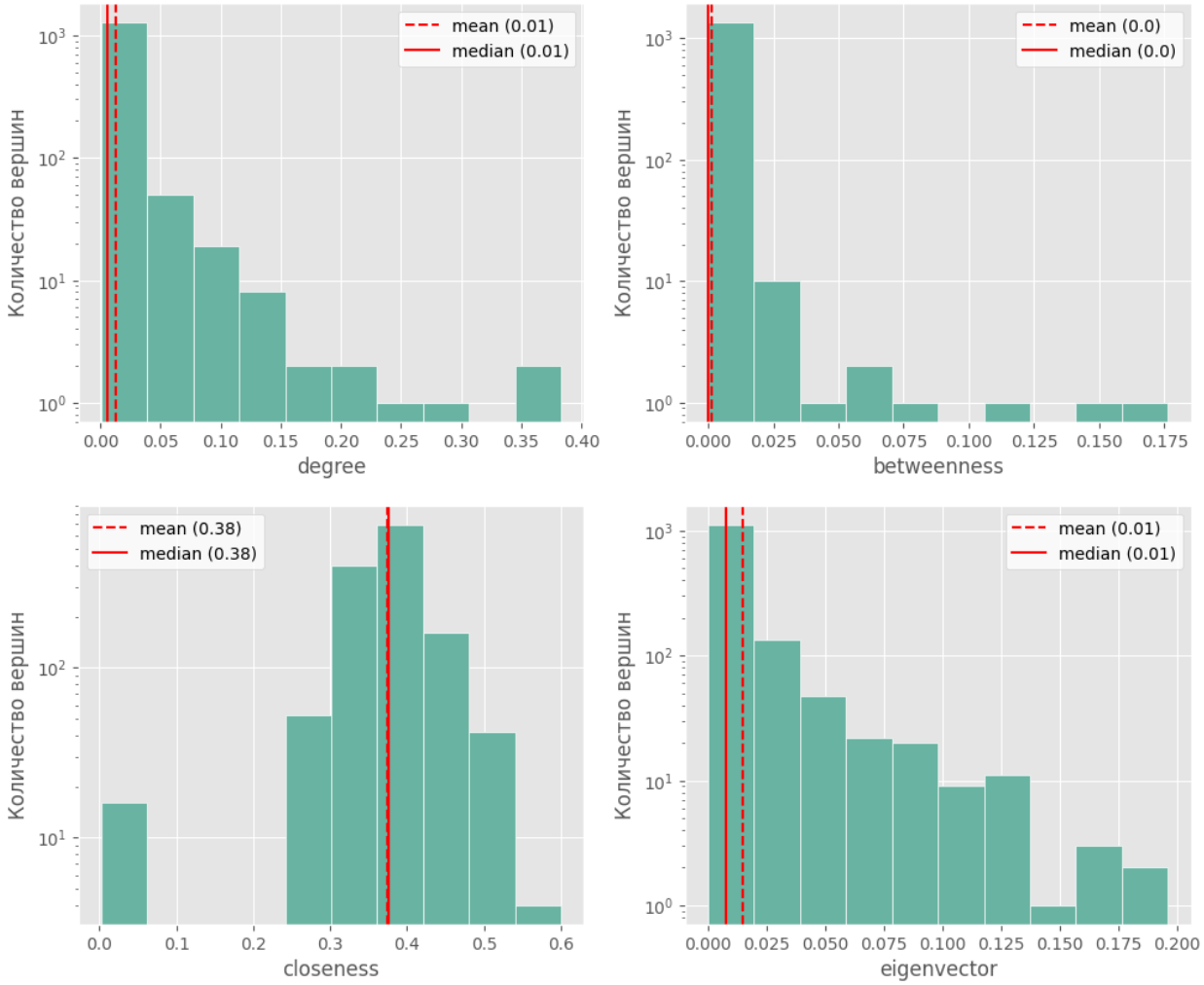
```
degrees_c = pd.DataFrame(dict(nx.degree_centrality(G)).items(),
columns=('name', 'degree')).set_index('name')

betweenness_c =
pd.DataFrame(dict(nx.betweenness_centrality(G)).items(),
columns=('name', 'betweenness')).set_index('name')

closeness_c = pd.DataFrame(dict(nx.closeness_centrality(G)).items(),
columns=('name', 'closeness')).set_index('name')

eig_c = pd.DataFrame(dict(nx.eigenvector_centrality(G)).items(),
columns=('name', 'eigenvector')).set_index('name')
```

```
centralities = pd.concat([degrees_c, betweenness_c, closeness_c,
eig_c],axis=1)
print(centralities.head())
```

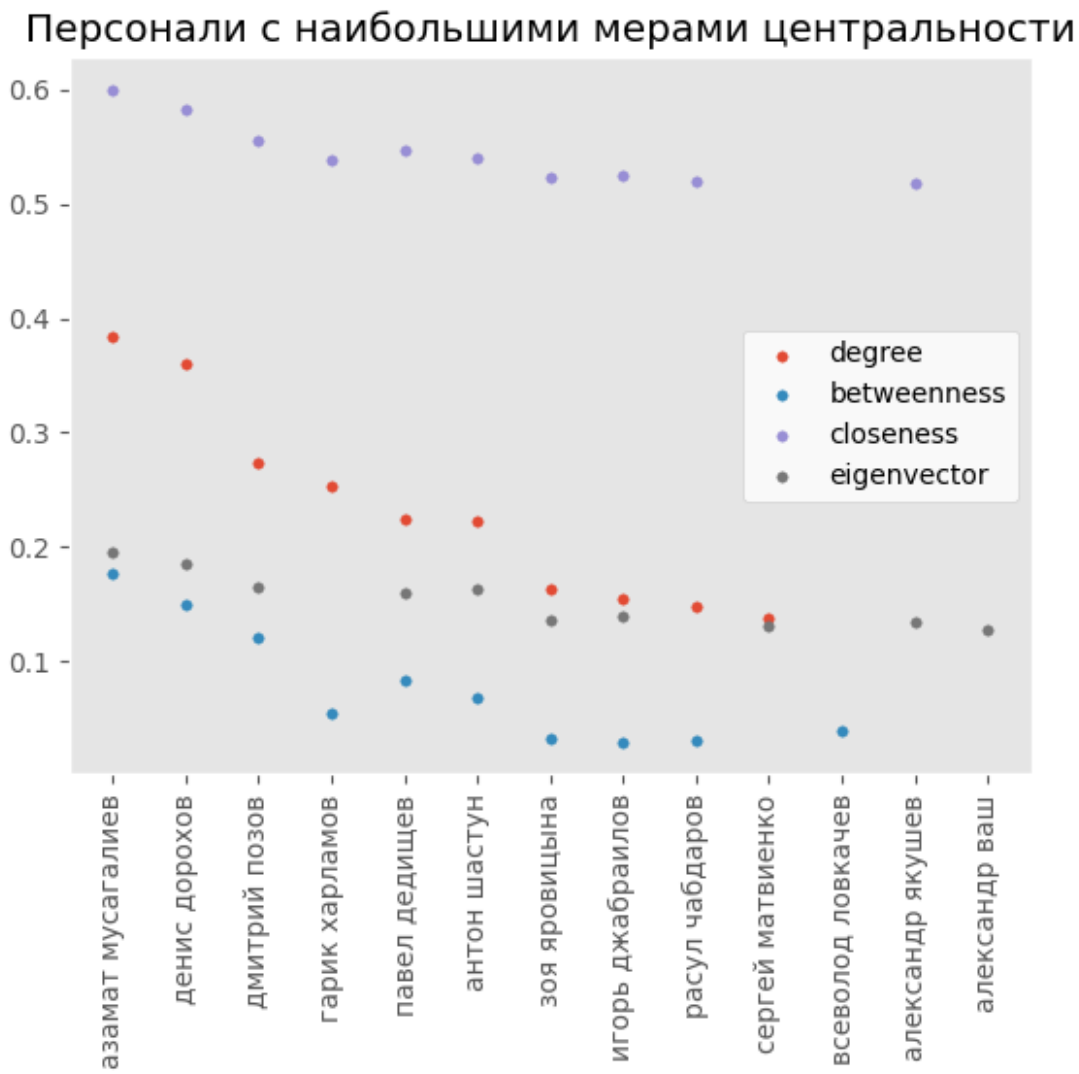
```
                      degree  betweenness  closeness  eigenvector
name
антон шастун        0.222870     0.068270   0.540148     0.163192
сергей матвиенко    0.138383     0.018115   0.510539     0.130802
александр забродин  0.044428     0.002415   0.467150     0.051399
сергей горох        0.077203     0.006008   0.491457     0.097242
валерий равдин      0.055353     0.001531   0.479680     0.080411
```

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
axes = axes.flatten()

for i, column in enumerate(centralities.columns):

    centralities[column].hist(ax=axes[i], edgecolor='white',
color='#69b3a2')
    axes[i].axvline(x=centralities[column].mean(), color='r',
linestyle='--', label=f'mean ({round(centralities[column].mean(),
2)})')
    axes[i].axvline(x=centralities[column].median(), color='r',
linestyle='-', label=f'median ({round(centralities[column].mean(),
2)})')

    # axes[i].set_title(f'{column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Количество вершин')
    axes[i].set_yscale('log')
    axes[i].legend(facecolor='white')

# plt.tight_layout()
plt.suptitle('Меры центральности')
plt.show();
```

Меры центральности




```
tops = set()
for m in ['degree', 'betweenness', 'closeness', 'eigenvector']:
    top10 = centralities.sort_values(by=m, ascending=False)[:10]

    plt.scatter(top10.index, top10[m], s=15, linewidths=0.3, label=m)

    tops.update(list(top10.index))

plt.legend(facecolor="white")
plt.grid()
plt.title('Персонали с наибольшими мерами центральности')
plt.xticks(rotation=90);
```




```
tops_shows = df[(df['host_norm'].fillna('').apply(set).apply(lambda s:
bool(s & tops))) |
df['guests_full'].fillna('').apply(set).apply(lambda s: bool(s &
tops))]
print('Персоналии из "топа" участвовали в', str(round(len(tops_shows)
/ len(df) * 100, 2)) + '% всех эпизодов в датасете, ')
print('или в', str(round(tops_shows['show_title'].nunique() /
df['show_title'].nunique() * 100, 2)), '% всех шоу в датасете.')
```

```
Персоналии из "топа" участвовали в 68.42% всех эпизодов в датасете,
или в 80.95 % всех шоу в датасете.
```

```
perc = []
for p in tops:
    cur_ = df[(df['host_norm'].fillna('').apply(set).apply(lambda s:
bool(s & {p}))) | df['guests_full'].fillna('').apply(set).apply(lambda
s: bool(s & {p}))]
```

```
    perc.append((p, len(cur_)))

for p in sorted(perc, key=lambda _: _[1], reverse=True):
    print(f'{p[0]}:\t\t', f'{p[1]}/{len(df)}' , f'эпизодов
({str(round(p[1] / len(df) * 100, 2))}%)')
```

```
антон шастун:           689/3322 эпизодов (20.74%)
дмитрий позов:          635/3322 эпизодов (19.11%)
азамат мусагалиев:           391/3322 эпизодов (11.77%)
денис дорохов:          360/3322 эпизодов (10.84%)
сергей матвиенко:            344/3322 эпизодов (10.36%)
игорь джабраилов:            324/3322 эпизодов (9.75%)
павел дедищев:          294/3322 эпизодов (8.85%)
расул чабдаров:         228/3322 эпизодов (6.86%)
гарик харламов:         180/3322 эпизодов (5.42%)
александр якушев:            158/3322 эпизодов (4.76%)
зоя яровицына:          143/3322 эпизодов (4.3%)
александр ваш:          134/3322 эпизодов (4.03%)
всеволод ловкачев:           56/3322 эпизодов (1.69%)
```

```
perc = []
for p in tops:
    cur_ = df[(df['host_norm'].fillna('').apply(set).apply(lambda s:
bool(s & {p}))) | df['guests_full'].fillna('').apply(set).apply(lambda
s: bool(s & {p}))]['show_title'].nunique()
    perc.append((p, cur_))

for p in sorted(perc, key=lambda _: _[1], reverse=True):
    print(f'{p[0]}:\t\t', f'{p[1]}/{df['show_title'].nunique()}' ,
f'шоу ({str(round(p[1] / df['show_title'].nunique() * 100, 2))}%)')
```

```
игорь джабраилов:            64/168 шоу (38.1%)
павел дедищев:          63/168 шоу (37.5%)
александр ваш:          54/168 шоу (32.14%)
дмитрий позов:          53/168 шоу (31.55%)
зоя яровицына:          49/168 шоу (29.17%)
денис дорохов:          49/168 шоу (29.17%)
расул чабдаров:         48/168 шоу (28.57%)
сергей матвиенко:            45/168 шоу (26.79%)
антон шастун:           43/168 шоу (25.6%)
азамат мусагалиев:           36/168 шоу (21.43%)
александр якушев:            24/168 шоу (14.29%)
гарик харламов:         15/168 шоу (8.93%)
всеволод ловкачев:           7/168 шоу (4.17%)
```

```
print('Средний процент шоу, в которых участвовали персоналии из
топа:', round(np.mean(list(dict(perc).values())) /
df['show_title'].nunique() * 100, 1), '%')
```

```
Средний процент шоу, в которых участвовали персоналии из топа: 25.2 %
```

```
plt.bar(dict(sorted(perc, key=lambda _: _[1], reverse=True)).keys(),
dict(sorted(perc, key=lambda _: _[1], reverse=True)).values())
plt.xticks(rotation=90);
plt.title('Количество эпизодов, в которых участвовали персоналии с
наибольшими значениями центральности', wrap=True);
```

В "топе" оказались известные комики, которые суммарно участвовали в 80% всех рассматриваемых шоу (25% шоу в среднем).

Ассортативность (Degree mixing pattern)

```
r = nx.degree_assortativity_coefficient(G)

print(f"Ассортативность по степени вершин: {r:3.1f}")
```

```
Ассортативность по степени вершин: -0.2
```

Значение ассортативности близко к нулю, что означает слабую зависимость вероятности присоединения от степени вершины, или небольшую склонность вершин с высокой степенью присоединяться к вершиной с низкой степенью.

Определение сообществ

```
def visualize_community(com_number, com_set, title=None,
dims=(20,20)):
    """
    Визуализировать вершины и связи внутри конкретного сообщества
    """
    cur_comm = com_set[com_number]

    # edgelist_sub = [e for e in edges if e[0] in cur_comm or e[1] in
cur_comm] # включаем членов сообщества + всех, кто снимался с ними
хотя бы раз
    edgelist_sub = [e for e in edges if e[0] in cur_comm and e[1] in
cur_comm] # включаем только членов сообщества

    G_sub = nx.from_edgelist(edgelist_sub)
    print(G_sub)

    # 3. Process the results for visualization
    # Create a dictionary to map each node to its community ID
    node2color = {}
    for pair in G_sub.edges:
        for node in pair:
            if node in cur_comm:
                node2color[node] = 'red'
            else:
                node2color[node] = 'blue'

    # Get a list of community IDs in the order of the nodes
    node_colors = [node2color[node] for node in G_sub.nodes()]

    # pos = nx.kamada_kawai_layout(G_sub)
    pos = nx.forceatlas2_layout(G_sub, scaling_ratio=100)

    top_20 = list(dict(sorted(dict(G_sub.degree).items(), key=lambda
x: x[1], reverse=True)[:20]).keys())

    labels = {}
    for node in G_sub.nodes():
        if node in cur_comm and node in top_20:
            labels[node] = node


    plt.figure(figsize=dims)

    nx.draw_networkx_nodes(G_sub, pos, node_size=[x*10 + 20 for x in
dict(G_sub.degree(weight='count')).values()], alpha=0.7,
```

```
node_color=node_colors)
    nx.draw_networkx_edges(G_sub, pos, arrows=True, edge_color='gray',
alpha=0.3) #  width=[(np.log(x)+2)/5 for x in
list(nx.get_edge_attributes(G_sub, 'count').values())]
    nx.draw_networkx_labels(G_sub,
pos,labels,font_size=9,font_color='black')

    if not title:
        title = f"Сообщество {com_number}"
    plt.title(title)
    # plt.legend()
    plt.show();
```

Clauset-Newman-Moore Greedy Modularity Maximization

```
comm_greedy =
list(nx.algorithms.community.greedy_modularity_communities(G))
print('Количество сообществ по Clauset-Newman-Moore Greedy Modularity
Maximization:', len(comm_greedy))
modularity_score = nx.community.modularity(G, comm_greedy)
print('Модулярность по Clauset-Newman-Moore Greedy Modularity
Maximization', round(modularity_score,2))
```

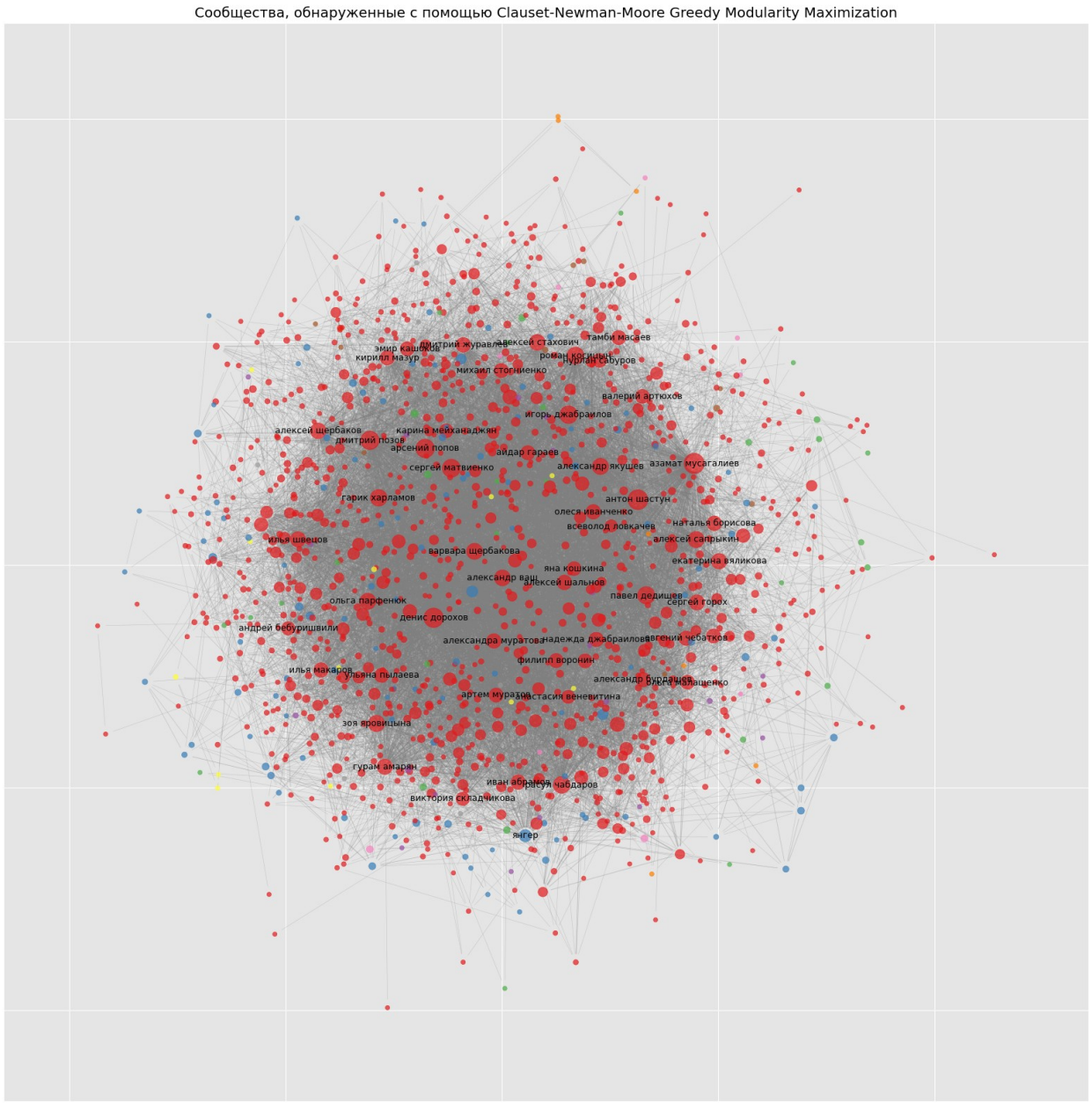
```
Модулярность по Clauset-Newman-Moore Greedy Modularity Maximization
0.35
```

Значение модулярности 0.35 говорит о наличии плотных связей между узлами внутри сообществ, но слабые связи между вершинами в различных сообществах.

Визуализируем полную сеть, выделяя цветом отдельные сообщества

```
node_to_community = {}
for i, comm in enumerate(comm_greedy):
    for node in comm:
        node_to_community[node] = i

node_colors = [node_to_community[node] for node in G.nodes()]

pos = nx.forceatlas2_layout(G, scaling_ratio=200)

labels = {}
for node in G.nodes():
    if node in hubs:
        labels[node] = node

edges_to_draw = []
for e in G.edges:
    if e[0] != e[1]:
        edges_to_draw.append(e)
```

```
plt.figure(figsize=(20,20))

nx.draw_networkx_nodes(G, pos, node_size=[np.log(x)**3 + 20 for x in
dict(G.degree(weight='count')).values()], alpha=0.7,
node_color=node_colors, cmap='Set1')
nx.draw_networkx_edges(G, pos, edgelist=edges_to_draw, arrows=True,
edge_color='gray', width=[(np.log(x)+2)/5 for x in
list(nx.get_edge_attributes(G, 'count').values())], alpha=0.3)
nx.draw_networkx_labels(G, pos,labels,font_size=9,font_color='black')

plt.title("Сообщества, обнаруженные с помощью Clauset-Newman-Moore
Greedy Modularity Maximization")
# plt.legend()
plt.show();
```

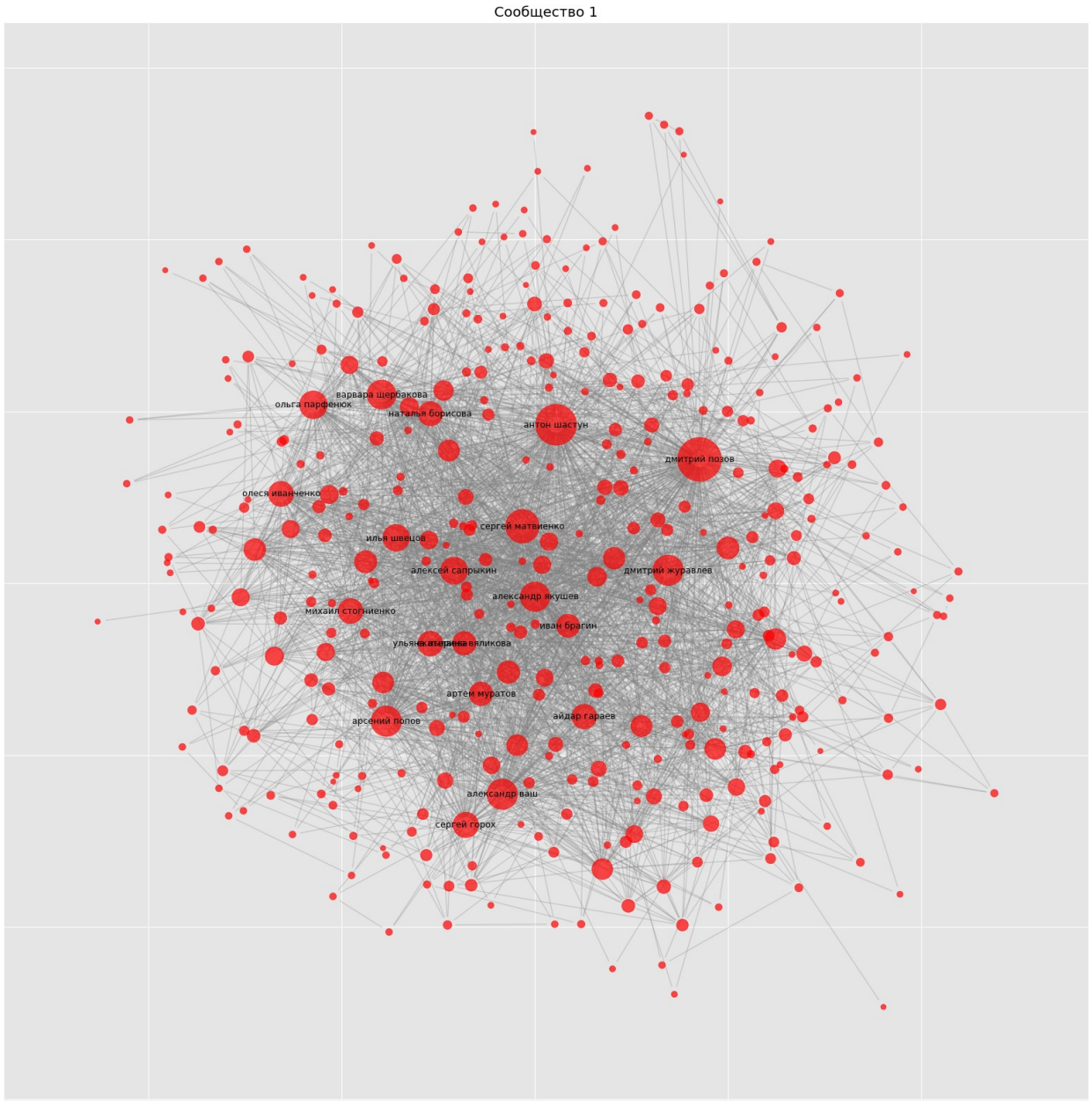



```
# print(f"Обнаружено {len(comm_greedy)} сообществ:")
# for i, comm in enumerate(comm_greedy):
#     print(f"Сообщество №{i}\t({len(comm)} человек):\
t{sorted(list(comm))}")
```

Визуализируем сообщества и подпишем топ-20 ее членов по степени вершины

```
COMMUNITY_NUM = 1 # @param {"type":"integer"}

visualize_community(COMMUNITY_NUM, comm_greedy)
```

```
Graph with 382 nodes and 2583 edges
```



Louvain method

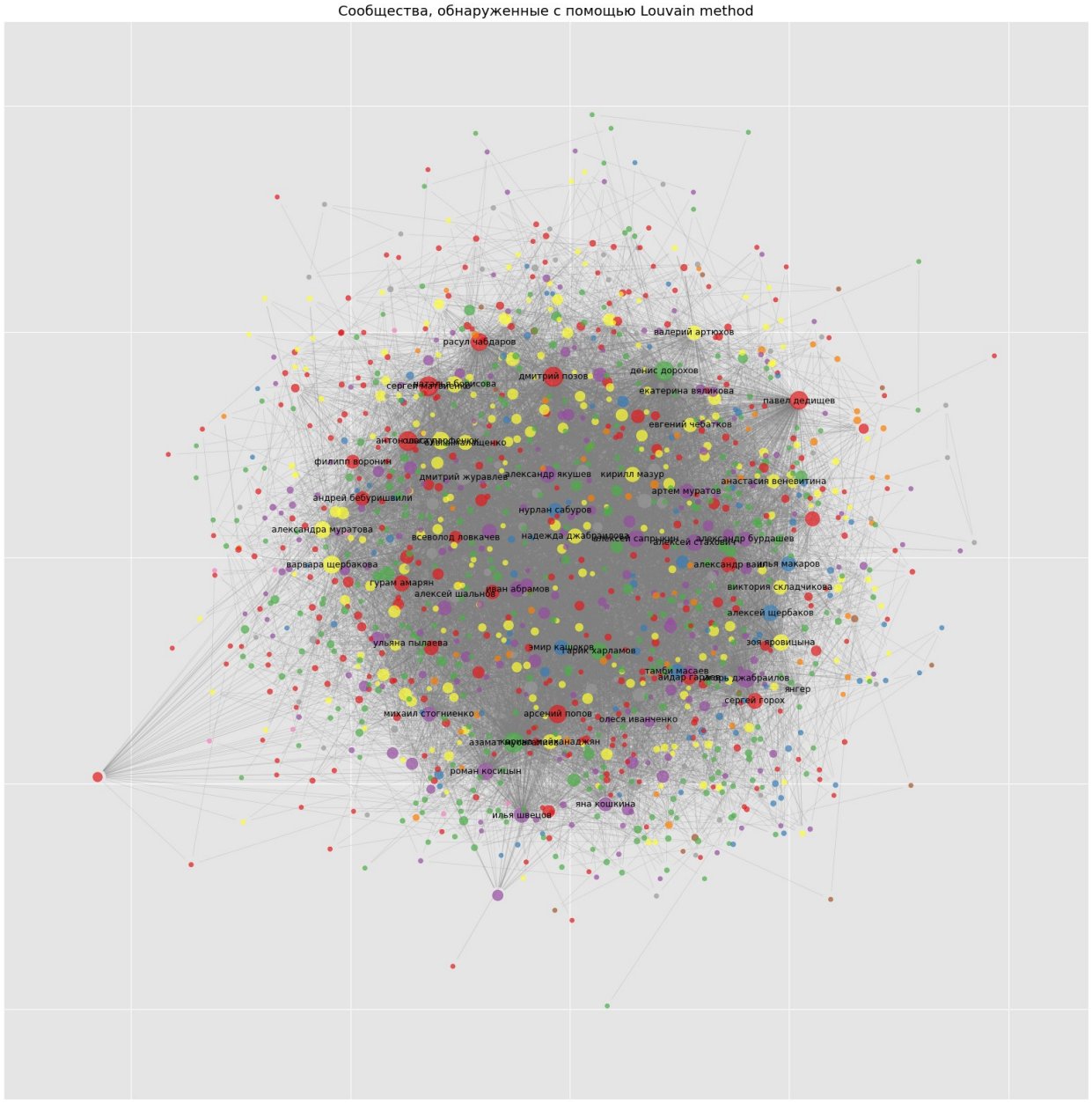
```
# определяем сообщества
comm_louvian = nx.community.louvain_communities(G, weight='count',
seed=42)
print('Количество сообществ по Louvain method:', len(comm_louvian))
# считаем модулярность
modularity_score = nx.community.modularity(G, comm_louvian)
print('Модулярность по Louvain method', round(modularity_score,2))
```

```
Количество сообществ по Louvain method: 15
Модулярность по Louvain method 0.36
```

Значение модулярности 0.36 говорит о наличии плотных связей между узлами внутри сообществ, но слабые связи между вершинами в различных сообществах.

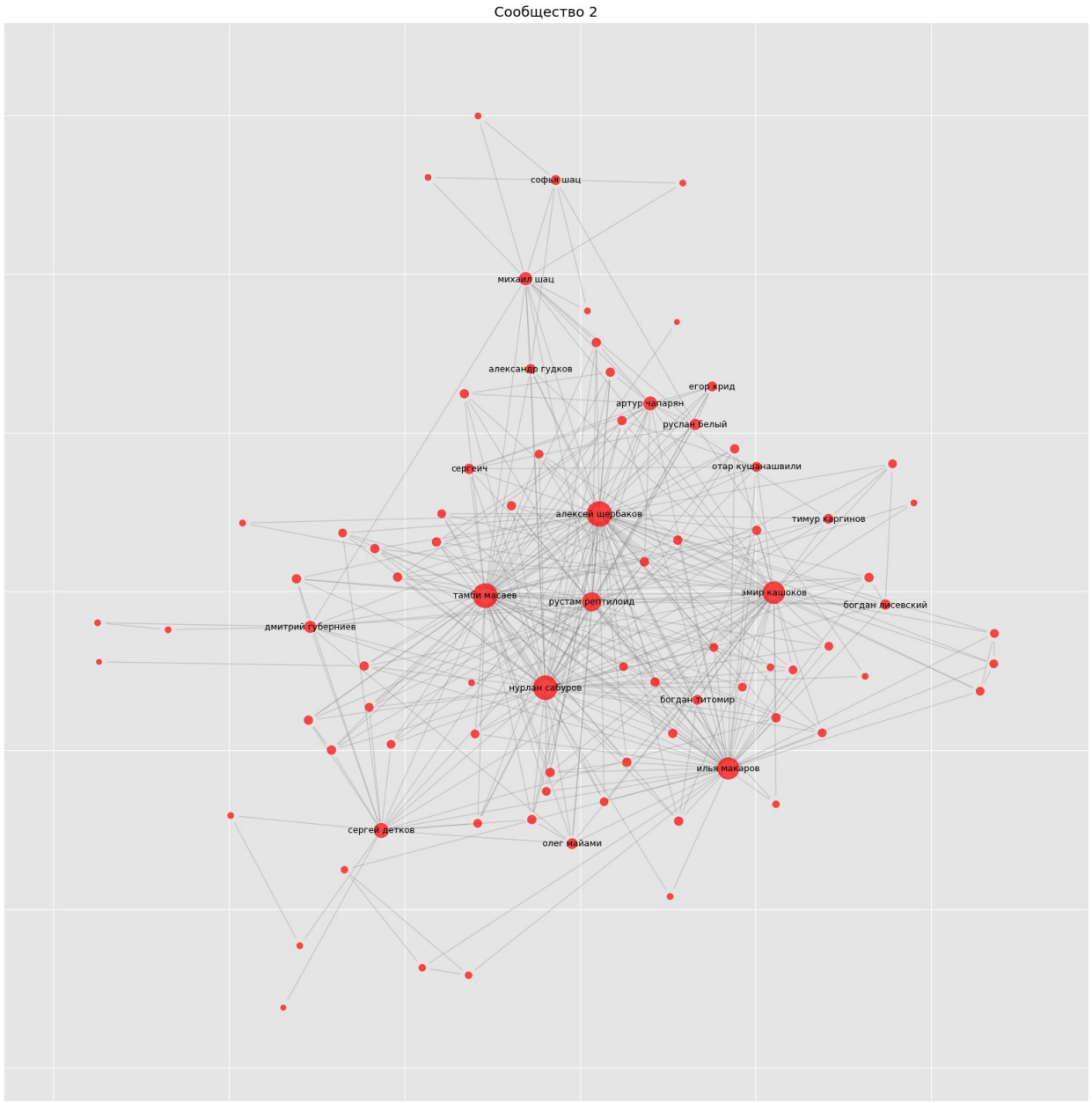
Визуализируем

```
node_to_community = {}
for i, comm in enumerate(comm_louvian):
    for node in comm:
        node_to_community[node] = i

node_colors = [node_to_community[node] for node in G.nodes()]

pos = nx.forceatlas2_layout(G, scaling_ratio=200)

labels = {}
for node in G.nodes():
    if node in hubs:
        labels[node] = node

edges_to_draw = []
for e in G.edges:
    if e[0] != e[1]:
        edges_to_draw.append(e)

plt.figure(figsize=(20,20))

nx.draw_networkx_nodes(G, pos, node_size=[np.log(x)**3 + 20 for x in
dict(G.degree(weight='count')).values()], alpha=0.7,
node_color=node_colors, cmap='Set1')
nx.draw_networkx_edges(G, pos, edgelist=edges_to_draw, arrows=True,
edge_color='gray', width=[(np.log(x)+2)/5 for x in
list(nx.get_edge_attributes(G, 'count').values())], alpha=0.3)
nx.draw_networkx_labels(G, pos,labels,font_size=9,font_color='black')

plt.title("Сообщества, обнаруженные с помощью Louvain method")
# plt.legend()
plt.show();
```




```
# print(f"Обнаружено {len(comm_louvian)} сообществ:")
# for i, comm in enumerate(comm_louvian):
#     # print(f"Сообщество №{i}\t({len(comm)} человек):\
t{sorted(list(comm))}")
#     print(f"Сообщество №{i}\t({len(comm)} человек)")

COMMUNITY_NUM = 2 # @param {"type":"integer"}

visualize_community(COMMUNITY_NUM, comm_louvian)
```

```
Graph with 84 nodes and 367 edges
```



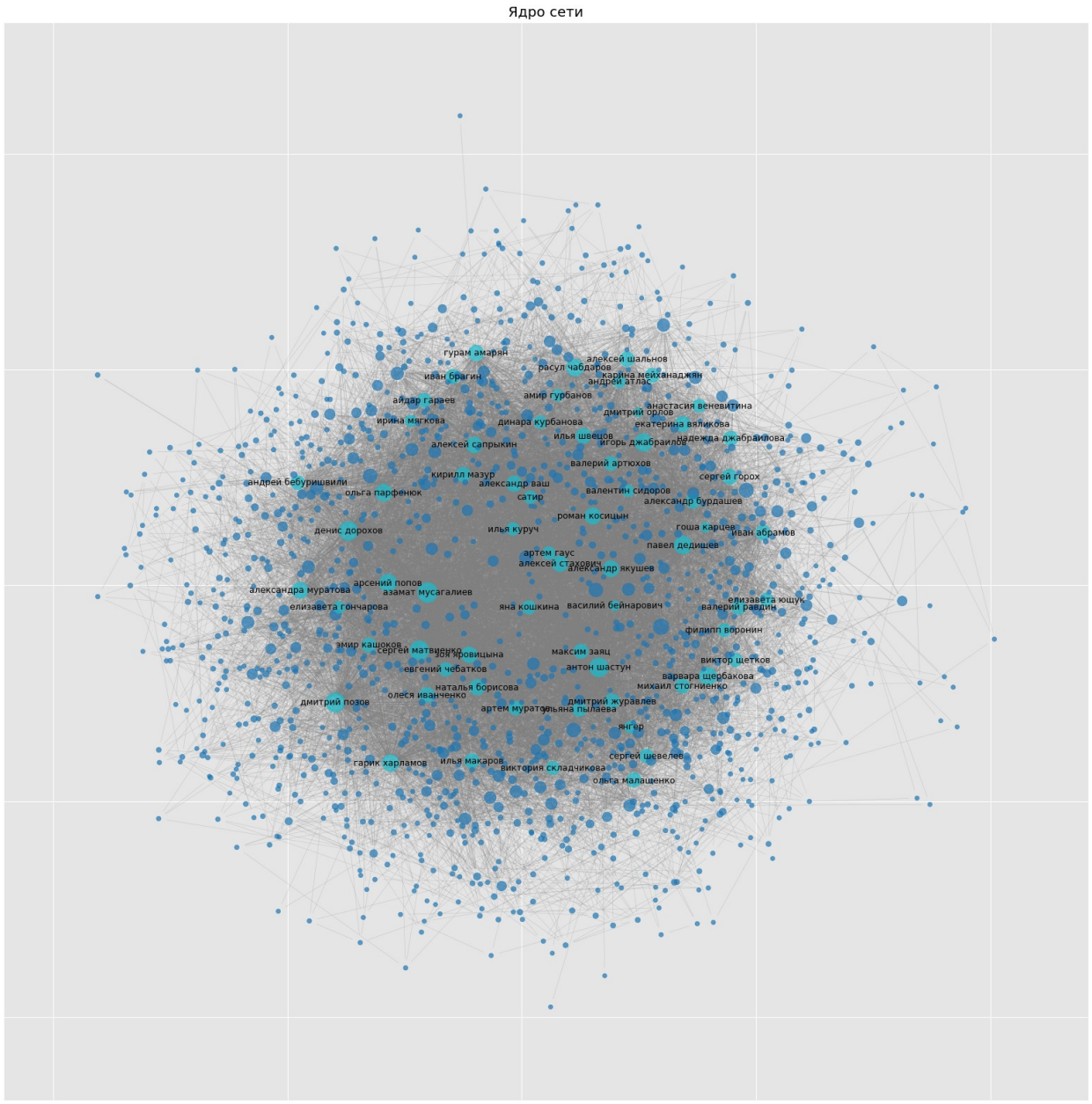
K-core decomposition

```
# находим "ядро" сети
main_core = nx.k_core(G)
# main_core.nodes

print('Минимальная степень вершины в ядре:',
np.min(list(dict(main_core.degree).values())))
```

```
Минимальная степень вершины в ядре: 31
```

```
node_colors = [1 if node in main_core.nodes else 0 for node in
G.nodes()]
```

```
pos = nx.forceatlas2_layout(G, scaling_ratio=200)

labels = {}
for node in G.nodes():
    if node in main_core.nodes:
        labels[node] = node

edges_to_draw = []
for e in G.edges:
    if e[0] != e[1]:
        edges_to_draw.append(e)

plt.figure(figsize=(20,20))

nx.draw_networkx_nodes(G, pos, node_size=[np.log(x)**3 + 20 for x in
dict(G.degree(weight='count')).values()], alpha=0.7,
node_color=node_colors, cmap='tab10')
nx.draw_networkx_edges(G, pos, edgelist=edges_to_draw, arrows=True,
edge_color='gray', width=[(np.log(x)+2)/5 for x in
list(nx.get_edge_attributes(G, 'count').values())], alpha=0.3)
nx.draw_networkx_labels(G, pos,labels,font_size=9,font_color='black')

plt.title("Ядро сети")
# plt.legend()
plt.show()
```

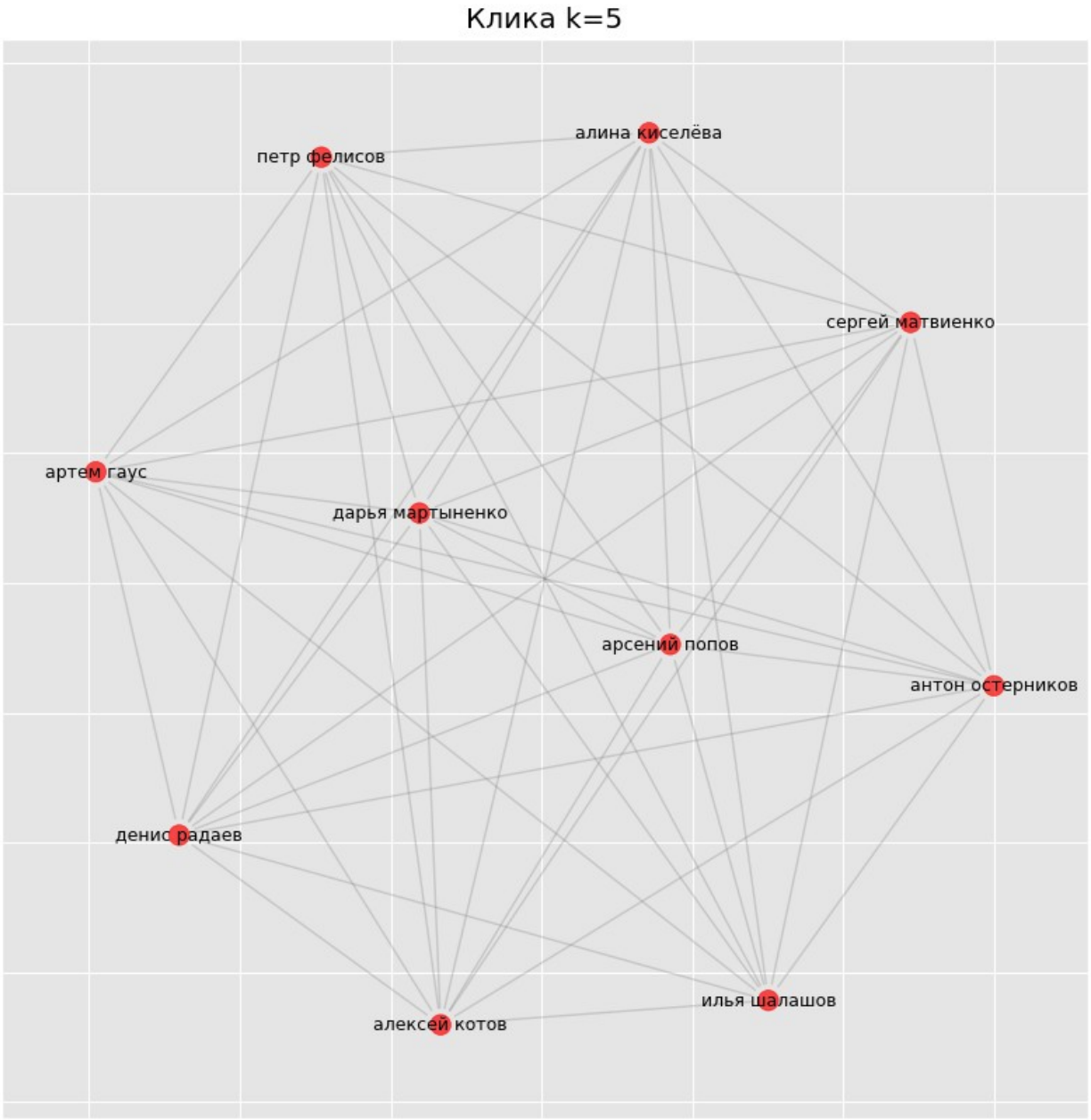



```
# определяем ядра
cores = nx.core_number(G)
core_vert = dict()

for pers, k in cores.items():
    if not core_vert.get(k):
        core_vert[k] = []
    core_vert[k] = core_vert[k] + [pers]
```

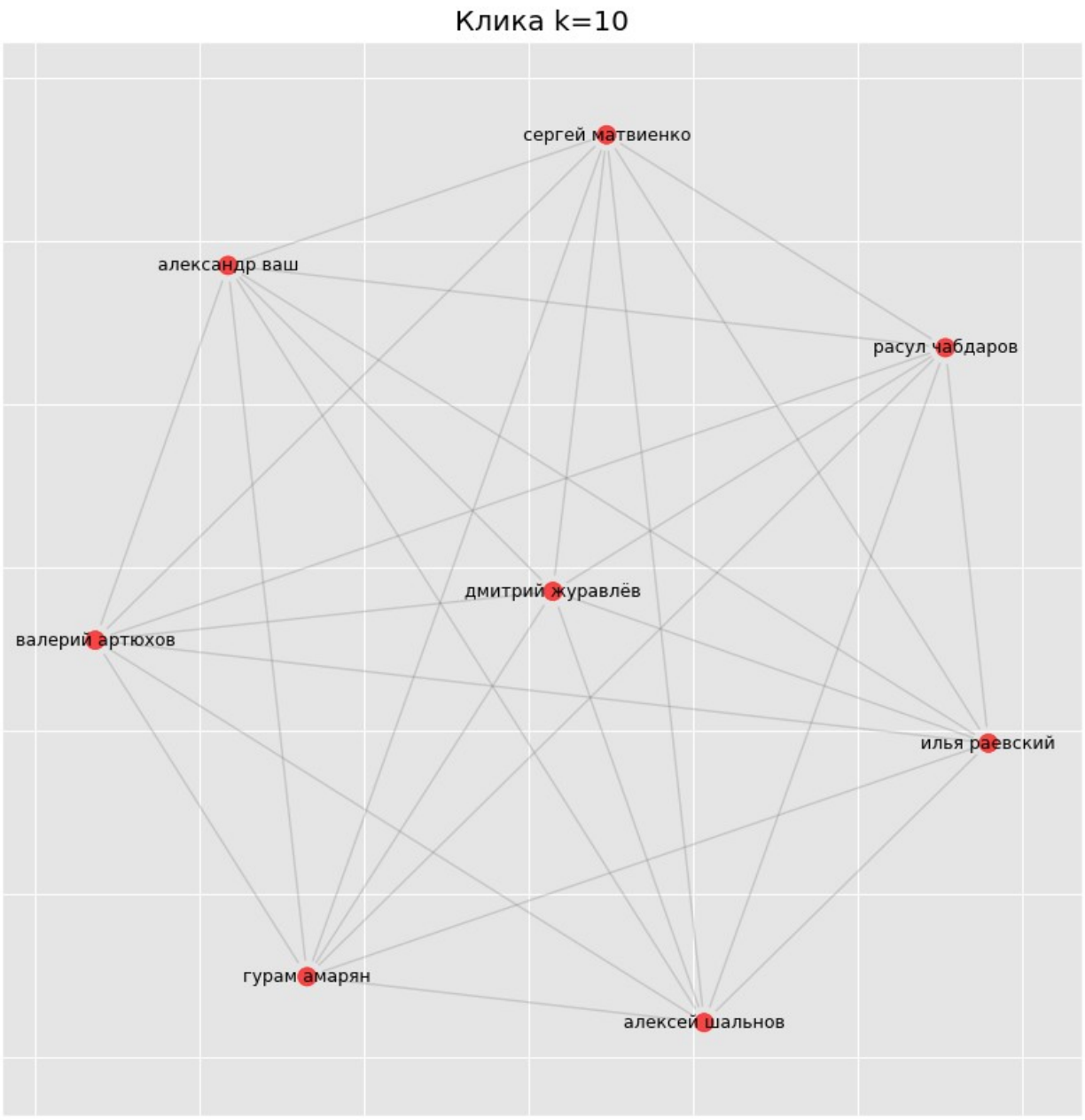
Визуализируем 6 клик

```
cores = nx.core_number(G)
max_k = np.max(list(cores.values()))
for k in [5, 10, 15, 20, 25, max_k]:
    k_core = nx.k_core(G, k)
    clique = next(nx.find_cliques(k_core))

    visualize_community(0, [clique], dims=(10,10), title=f'Клика
k={k}')
```

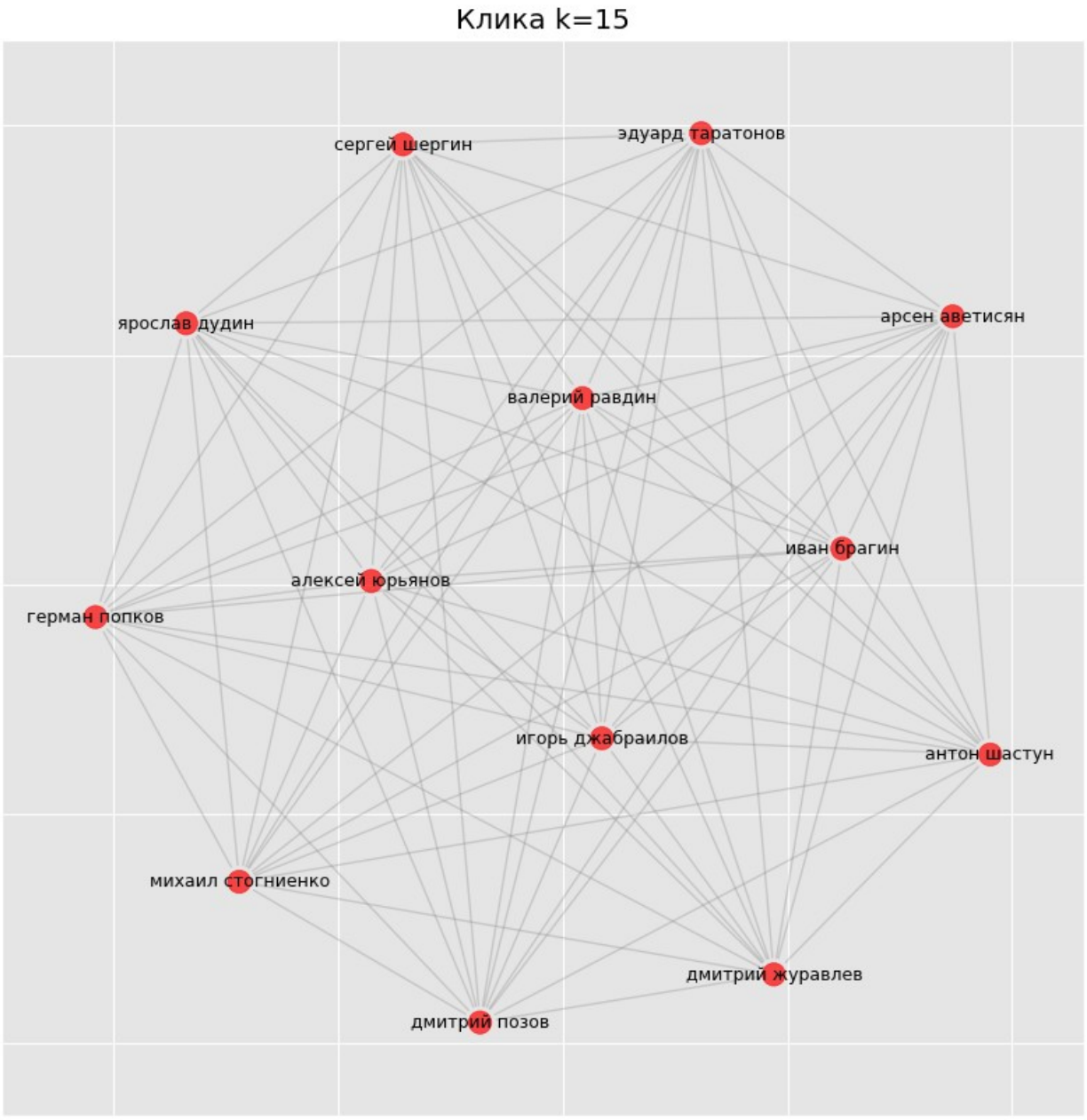
```
Graph with 10 nodes and 45 edges
```



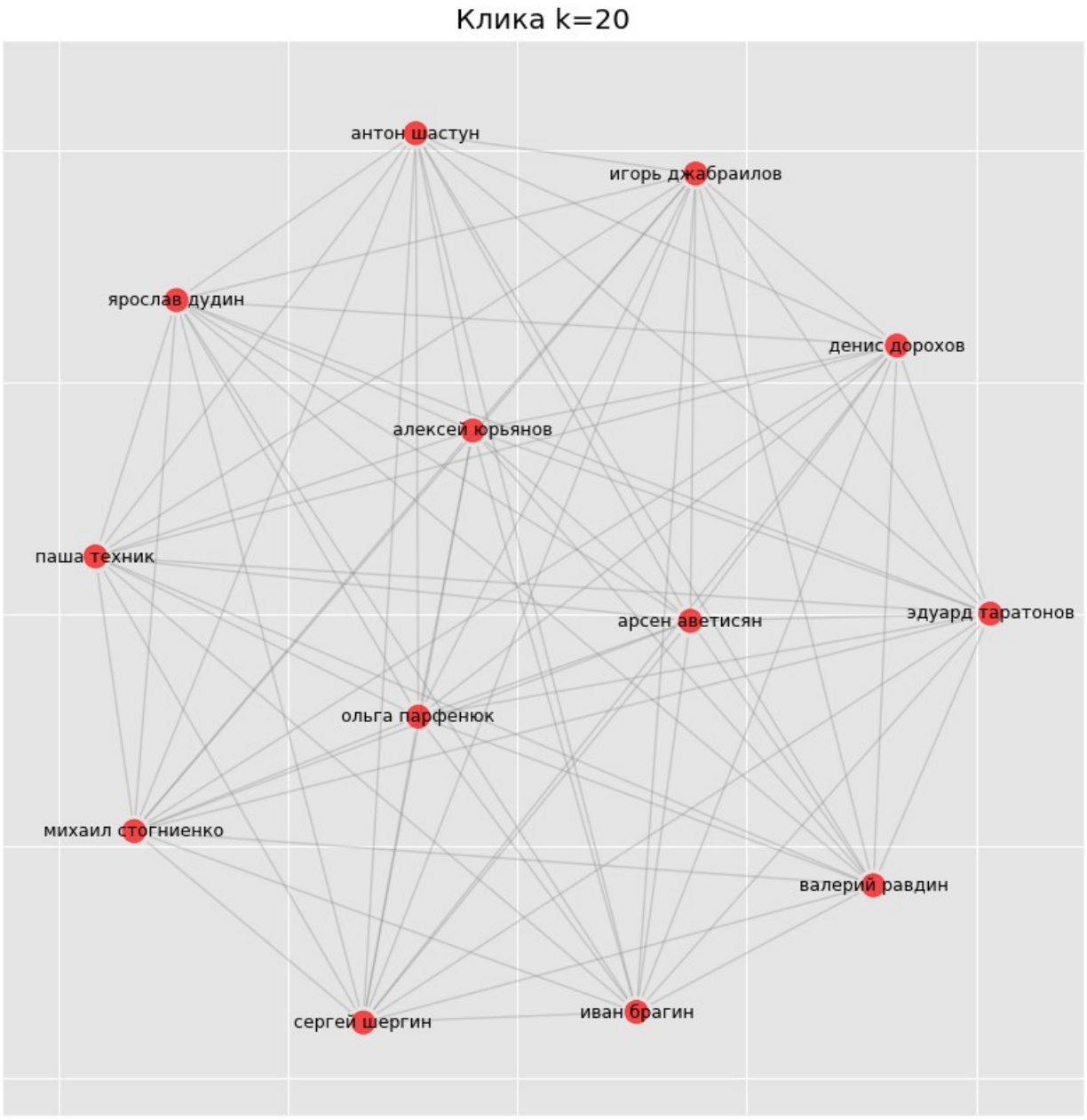
```
Graph with 8 nodes and 28 edges
```



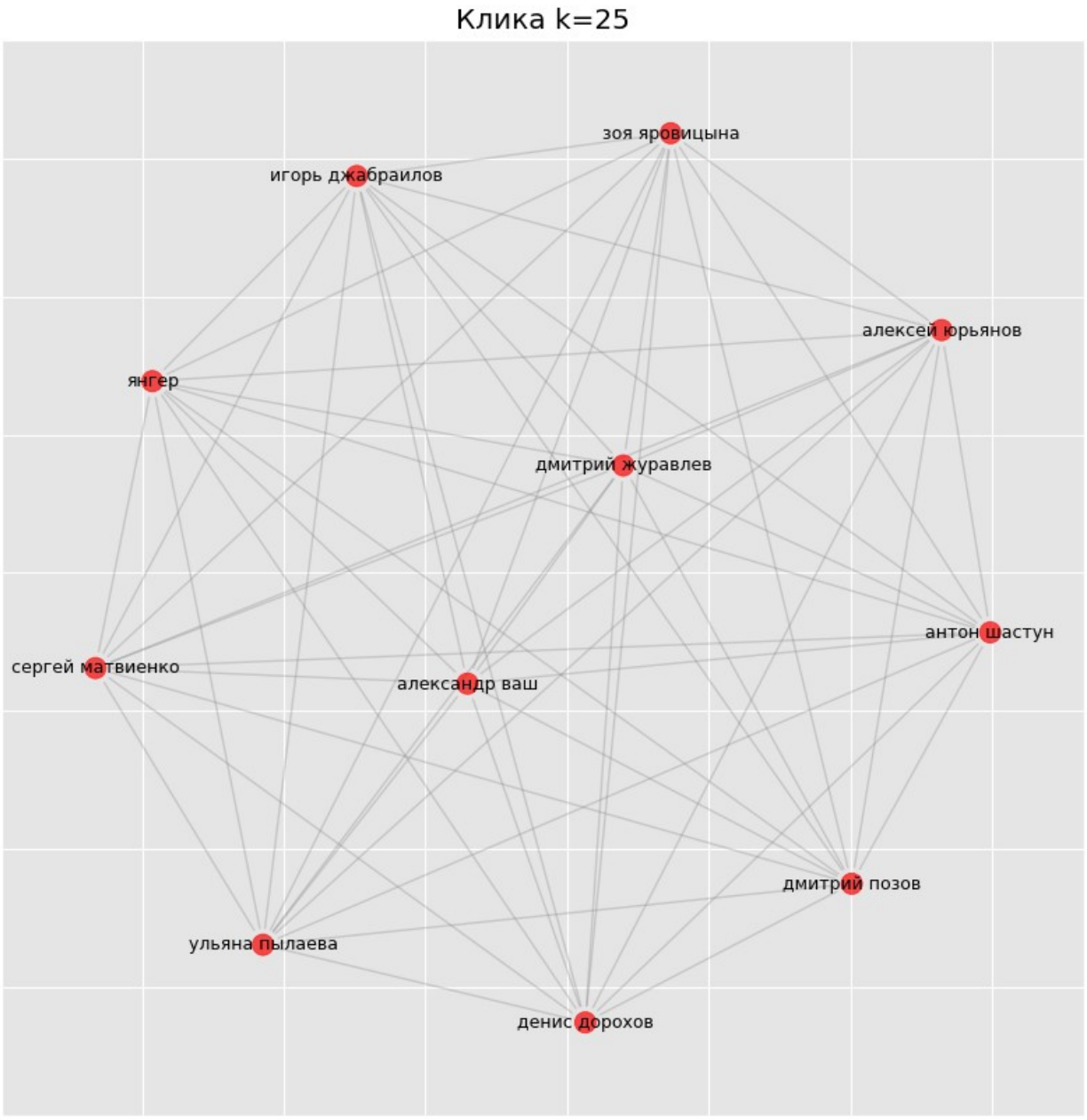

```
Graph with 13 nodes and 78 edges
```



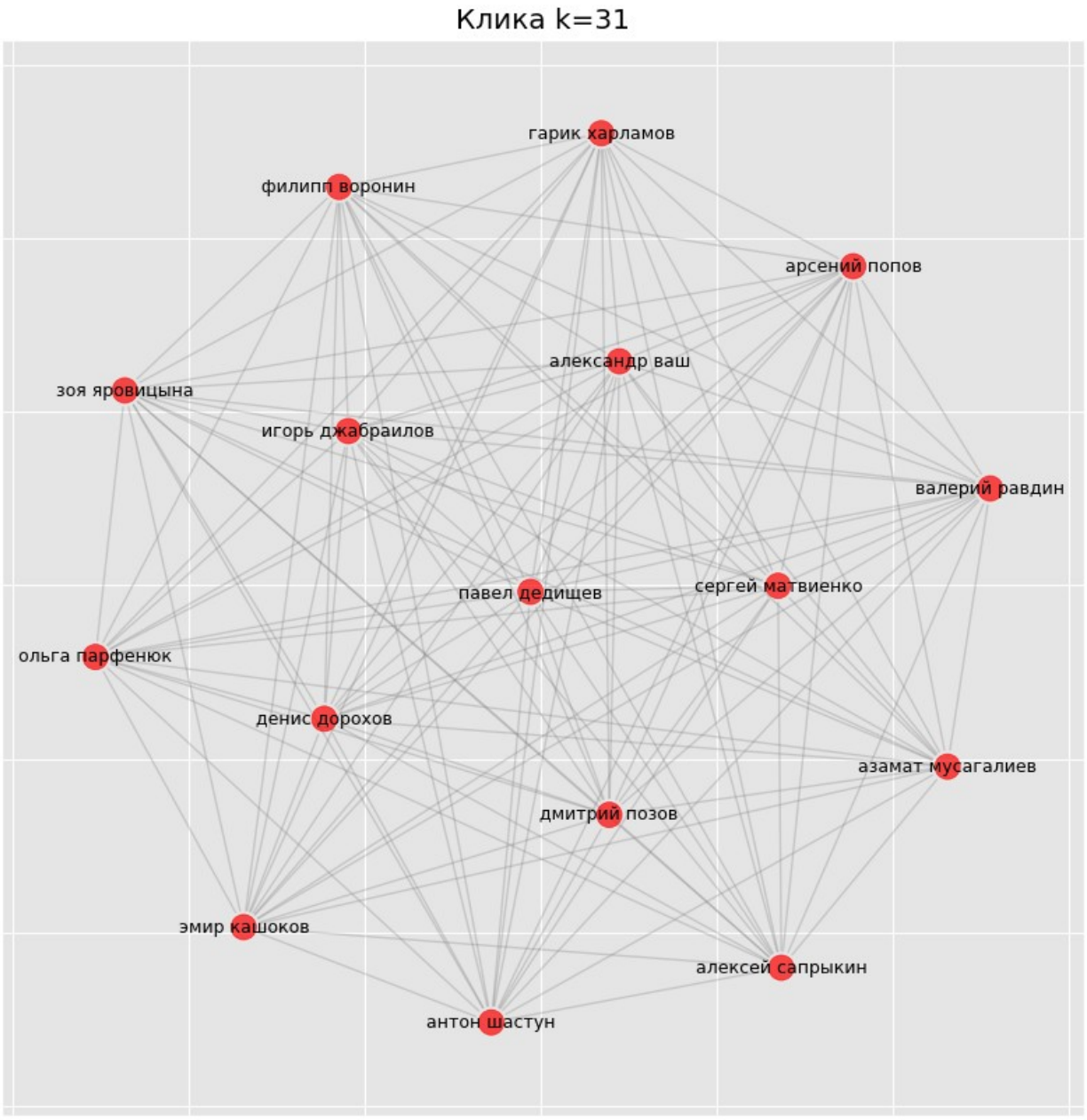

```
Graph with 13 nodes and 78 edges
```




```
Graph with 11 nodes and 55 edges
```




```
Graph with 16 nodes and 120 edges
```



В клику k=25 вошли ведущие проекта "Импровизация" Антон Шастун, Сергей Матвиенко и Дмитрий Позов. В числе ведущих также числится Арсений Попов, но в клику он не вошел. Остальную часть этой клики, как и состав других клик описать сложно.