Это ***вспомогательный ноутбук***, необходимый лишь для обработки всех данных и формирования общего файла с предсказаниями.

Никаких новых функций в нём нет.

In [3]:
```
import pandas as pd
import numpy as np
from tqdm import tqdm_notebook
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from umap.umap_ import UMAP

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import \
    KMeans, \
    AgglomerativeClustering, \
    DBSCAN, \
    SpectralClustering

warnings.filterwarnings('ignore')
sns.set_style('whitegrid')
sns.set(font_scale=1.5)
```

Файл со всеми ~115000 сессиями.

In [4]:
```
all_data = pd.read_csv('data/processed_data.csv')
```

Генерация новых признаков.

In [5]:

```
def add_features(data):

    new_data = data.copy()

    new_data['delay'] = new_data['ts']
    new_data.loc[1:, 'delay'] = (new_data['delay'].values[1:] -
                                 new_data['delay'].values[:-1])

    dist = np.sqrt((new_data['x'].values[1:] - new_data['x'].values[:-1])**2 +
                   (new_data['y'].values[1:] - new_data['y'].values[:-1])**2)

    new_data['dist'] = 0
    new_data.loc[1:, 'dist'] = dist

    new_data.loc[new_data['begin'] == 1, 'delay'] = 0
    new_data.loc[new_data['begin'] == 1, 'av_speed'] = 0

    new_data['av_speed'] = new_data['dist'] / new_data['delay']

    new_data['x_diff'] = new_data.x
    new_data.loc[1:, 'x_diff'] = new_data.x.values[1:] - new_data.x.values[:-1]

    new_data['y_diff'] = new_data.y
    new_data.loc[1:, 'y_diff'] = new_data.y.values[1:] - new_data.y.values[:-1]

    new_data.loc[new_data['begin'] == 1, 'x_diff'] = 0
    new_data.loc[new_data['begin'] == 1, 'y_diff'] = 0

    angles = [np.arctan2(y_diff, x_diff) for y_diff, x_diff in zip(new_data.y_diff, new_data.x_diff)]

    new_data['angle'] = angles
    new_data.loc[new_data.angle == 0, 'angle'] = np.nan
    new_data.loc[new_data['begin'] == 1, 'angle'] = 0.

    # когда такси стоит, оно сохраняет направление
    new_data.loc[:, 'angle'].fillna(method='pad', inplace=True)

    return new_data
```

In [6]:

```
new_all_data = add_features(all_data)
```

Полученные данные.

In [7]:
```
1 new_all_data.head()
```

Out[7]:

	status	y	ts	x	begin	session	delay	dist	av_speed	x_diff	y_diff	angle
0	0.0	0.000000	0.0	0.000000	1	0	0.0	0.000000	NaN	0.000000	0.000000	0.000000
1	0.0	0.291129	9.0	-0.644802	0	0	9.0	0.707478	0.078609	-0.644802	0.291129	2.717491
2	0.0	1.229173	17.0	-0.243663	0	0	8.0	1.020216	0.127527	0.401139	0.938044	1.166697
3	0.0	13.056778	25.0	2.447144	0	0	8.0	12.129826	1.516228	2.690807	11.827605	1.347101
4	0.0	21.475057	33.0	-3.184499	0	0	8.0	10.128318	1.266040	-5.631643	8.418279	2.160397

Предсказания с использованием методов кластеризации.

In [10]:

```
def clustering_prediction(fit_predict_methods, data):
    session_values = np.unique(data.session.values)

    session_predictions = []
    cluster_distances = []
    for i, session in tqdm_notebook(enumerate(session_values), total=len(session_values)):

        # Данные рассматриваемой сессии
        session_data = data[data.session == session]
        session_data = session_data[session_data.ts != 0]

        columns = ['x', 'y', 'ts', 'av_speed', 'angle', 'dist']

        # Выделяем из данных сессии данные с нужными статусами
        status_1_data = session_data[session_data.status == 1].loc[:, columns].copy()
        status_0_data = session_data[session_data.status == 0].loc[:, columns].copy()
        status_2_data = session_data[session_data.status == 2].loc[:, columns].copy()

        # Соединяем всё вместе
        status_data_w_dist = pd.concat([status_0_data.iloc[-2:, :],
                                        status_1_data,
                                        status_2_data.iloc[:2, :]],
                                        axis=0)

        if status_data_w_dist.shape[0] < 3:
            status_data_w_dist = status_data_w_dist.append(status_data_w_dist.iloc[0, :])

        status_data = status_data_w_dist.loc[:, columns[:-1]].copy()

        if status_data.shape[0] < 3:
            status_data = status_data.append(status_data.iloc[0, :])


        #Стандартизуем данные
        scaler = StandardScaler(with_mean=False)
        status_data.iloc[:, :] = scaler.fit_transform(status_data.copy())


        x_preds = []
        y_preds = []
        dist_preds = []
```

```
        for method in fit_predict_methods:
            pred = method(status_data)

            # Найдём точки, где сменяется кластер
            candidates = []
            for j in range(len(pred) - 1):
                if pred[j] != pred[j+1]:
                    candidates.append(j)

            # Если кластера всего два, то возьмём последнюю рассматриваемую
            # точку в качестве правой границы
            if len(candidates) == 1:
                candidates.append(len(pred) - 1)

            # Предсказания и среднее расстояние в кластере
            x_preds.append(status_data_w_dist['x'].values[candidates[-1]])
            y_preds.append(status_data_w_dist['y'].values[candidates[-1]])
            dist_preds.append(np.mean(status_data_w_dist['dist'].values[candidates[0]:candidates[-1]+1]))


        x_pred = np.mean(x_preds)
        y_pred = np.mean(y_preds)

        session_predictions.append([i, x_pred, y_pred])
        cluster_distances.append(np.mean(dist_preds))

    return np.array(session_predictions), np.array(cluster_distances)

fit_predict_methods = [KMeans(n_clusters=3,
                              random_state=42,
                              n_init=100,
                              max_iter=1000,
                              n_jobs=-1).fit_predict,
                       AgglomerativeClustering(n_clusters=3,
                                               linkage='ward').fit_predict,
                       SpectralClustering(n_clusters=3, n_jobs=-1).fit_predict]
```

In [11]:
```
# Очень долго выполняющийся код
# all_predictions, all_cluster_distances = clustering_prediction(fit_predict_methods, new_all_data)
```

100% 115204/115204 [4:46:09<00:00, 6.71it/s]

Предсказание с помощью поиска первой точки со статусом 2.

In [12]:
```
def first_status_2(data):
    d = {}

    session_values = np.unique(data.session.values)
    for session in tqdm_notebook(session_values):
        session_data = data[data.session == session]
        pickup_row = session_data[session_data.status == 2].head(1)
        d[session] = (pickup_row.x.values[0], pickup_row.y.values[0])

    return d
```

Формирование словаря всех предсказаний.

In [16]:
```
def final_preds(predictions, cluster_distances, data):
    # пороговое значение
    threshold = np.quantile(cluster_distances, 0.1)
    print(f'Пороговое значение межкластерного расстояния: {threshold}')

    valid_predictions = predictions[cluster_distances < threshold]

    preds = {}
    for pred in valid_predictions:
        preds[int(pred[0])] = (pred[1], pred[2], 0)

    first2 = first_status_2(data)

    for key in first2:
        if key not in preds:
            preds[key] = (first2[key][0], first2[key][1], 1)

    return preds
```

In [17]:
```
preds = final_preds(all_predictions, all_cluster_distances, all_data)
```

```
Пороговое значение межкластерного расстояния: 1.1775877242421593
```

100% 115204/115204 [30:07<00:00, 63.73it/s]

In [18]:
```
preds[1], preds[100]
```

Out[18]:
```
((467.8795440394896, 1009.9926489507196, 1),
 (373.0633040684311, -1113.3754465019024, 1))
```

Сохранение данных.

In [19]:
```
preds_data = pd.DataFrame(preds).T
preds_data.columns = ['x', 'y', 'method']
preds_data = preds_data.sort_index()

preds_data.head()
```

Out[19]:

	x	y	method
0	537.323154	73.614187	1.0
1	467.879544	1009.992649	1.0
2	-698.044056	-39.040766	1.0
3	-140.035784	2103.199406	1.0
4	-1033.518299	-107.762341	1.0

In [20]:
```
preds_data.to_csv('data/all_predictions.csv')
```

In [ ]:
```

```