Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Географически аппроксимировать и данных интерполировать погоду за месяц

Студент ИУ5Ц-82Б _____ **А.Е. Чиварзин**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

Консультант _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой __ИУ5__
(Индекс)
_____ _В. М. Черненький_
(И.О.Фамилия)
« _9_ » __февраля__ 20 _22_ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме

_____Географически апексиммировать и интерполировать погоду за месяц._____

Студент группы ИУ5Ц-82Б

_____Чиварзин Александр Евгеньевич_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

_____Научно-исследовательская работа_____

Источник тематики (кафедра, предприятие, НИР)

_____ ___Научно-исследовательская работа _____

График выполнения НИР: 25% к _3_ нед., 50% к _9_ нед., 75% к _12_ нед., 100% к _15_ нед.

Техническое задание

____Спроектировать (смоделировать) систему анализа и визуализации данных _____
_____«Погода» на языке Python_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на __1234_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____

Дата выдачи задания « _9_ » __февраля___ 20_22_ г.

Руководитель НИР _____ __Ю. Е. Гапанюк_____
(Подпись, дата) (И.О.Фамилия)

Студент _____ __А. Е. Чиварзин____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Поиск и выбор набора данных для построения моделей машинного обучения

В качестве набора данных мы будем использовать набор данных, содержащий историю погоды в США с 1964 по 2013 год - https://data.world/carlvlewis/u-s-weather-outliers-1964

Эта задача является очень актуальной для...

Датасет состоит иис одного файла `weather-anomalies-1964-2013.csv`

Файл содержит следующие колонки:

- `date_str` – дата измерения погоды
- `degrees_from_mean`
- `id`
- `longitude`
- `latitude`
- `max_temp`
- `min_temp`
- `station_name`
- `type`
- `serialid`

В данной работе будем решать задачи регрессии.

Импорт библиотек

Импортируем библиотеки с помощью команды import. Как правило, все команды import размещают в первых ячейках ноутбука.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import datetime
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absol
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
%matplotlib inline
sns.set(style="ticks")
# import gmaps # не подходит, поскольку невозможно использовать этот API без платёжной системы Google
import pygeoplot as gp # Пакета нет в pip, установка мописана в gitHub
```

Загрузка данных

Загрузим файлы датасета в помощью библиотеки Pandas.

Файл представляет собой данные в формате CSV (https://ru.wikipedia.org/wiki/CSV). Часто в файлах такого формата в качестве разделителей используются символы ",", ";" или табуляция. Поэтому вызывая метод `read_csv` всегда стоит явно указывать разделитель данных с помощью параметра `sep` . Чтобы узнать какой разделитель используется в файле его рекомендуется предварительно посмотреть в любом текстовом редакторе.

In [2]:
```
original = pd.read_csv('weather-anomalies-1964-2013.csv', sep=",")
```

Пос кольку наборп данных очень большой, то будем использовать только первые 2000 строк

In [3]:
```
original2000 = original.head(2000)
```

Удалим дкбликаты записей, если они присутствуют

In [4]:
```
data = original2000.drop_duplicates()
```

Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

Основные характенристики набора данных

Первые и последние 5 строк выборки из датасета (далее - датасета)

In [5]:
```
data
```

Out[5]:

	date_str	degrees_from_mean	id	longitude	latitude	max_temp	min_temp	station_name	type	serialid
0	1977-02-19	8.61	USC00103882	-113.5472	43.7186	10.0	-12.8	GROUSE	Weak Hot	1
1	1977-02-19	10.74	USC00053951	-107.1097	37.7717	11.1	-8.9	HERMIT 7 ESE	Weak Hot	2
2	1977-02-19	20.46	USC00040379	-119.5128	37.0919	25.6	12.8	AUBERRY 2 NW	Strong Hot	3
3	1977-02-19	8.60	USC00020808	-109.7517	33.4783	20.0	-3.9	BLACK RIVER PUMPS	Weak Hot	4
4	1977-02-19	10.30	USC00042598	-115.4508	33.8089	30.6	13.9	EAGLE MTN	Weak Hot	5
...	...	...	...	...	...	...	...	...	...	...
1995	1977-08-28	-12.11	USC00355142	-121.1461	44.6633	17.2	7.8	MADRAS 2 N	Weak Cold	1996
1996	1977-08-28	-14.36	USC00248809	-113.9847	48.5003	11.7	5.6	WEST GLACIER	Weak Cold	1997
1997	1977-08-28	8.28	USW00014739	-71.0106	42.3606	35.0	20.6	BOSTON LOGAN INTL AP	Weak Hot	1998
1998	1977-08-28	9.83	USW00023244	-122.0481	37.4058	34.4	16.7	MOFFETT FEDERAL AIRFIELD	Weak Hot	1999
1999	1977-08-28	8.12	USC00207280	-84.5542	43.0114	33.3	22.8	SAINT JOHNS	Weak Hot	2000

2000 rows × 10 columns

Список колонок с типами данных

In [6]:
```
data.dtypes
```

Out[6]:
```
date_str              object
degrees_from_mean    float64
id                    object
longitude            float64
latitude             float64
max_temp             float64
min_temp             float64
station_name          object
type                  object
serialid               int64
dtype: object
```

Как выидим - тип даты определён неверно. Исправим это

In [7]:
```
data['date_str'] = pd.to_datetime(data['date_str'], format="%Y-%m-%d")
```

Проверяем типы данных ещё раз

In [8]:
```
data.dtypes
```

Out[8]:
```
date_str             datetime64[ns]
degrees_from_mean           float64
id                           object
longitude                   float64
latitude                    float64
max_temp                    float64
min_temp                    float64
station_name                 object
type                         object
serialid                      int64
dtype: object
```

Как видим - типы данных стали верными. Выведем первые 5 строк датасета

In [9]:
```
data.head()
```

Out[9]:

	date_str	degrees_from_mean	id	longitude	latitude	max_temp	min_temp	station_name	type	serialid
0	1977-02-19	8.61	USC00103882	-113.5472	43.7186	10.0	-12.8	GROUSE	Weak Hot	1
1	1977-02-19	10.74	USC00053951	-107.1097	37.7717	11.1	-8.9	HERMIT 7 ESE	Weak Hot	2
2	1977-02-19	20.46	USC00040379	-119.5128	37.0919	25.6	12.8	AUBERRY 2 NW	Strong Hot	3
3	1977-02-19	8.60	USC00020808	-109.7517	33.4783	20.0	-3.9	BLACK RIVER PUMPS	Weak Hot	4
4	1977-02-19	10.30	USC00042598	-115.4508	33.8089	30.6	13.9	EAGLE MTN	Weak Hot	5

Провекряем наличие пропущенных значений

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
date_str             0
degrees_from_mean    0
id                   0
longitude            0
latitude             0
max_temp             0
min_temp             0
station_name         0
type                 0
serialid             0
dtype: int64
```

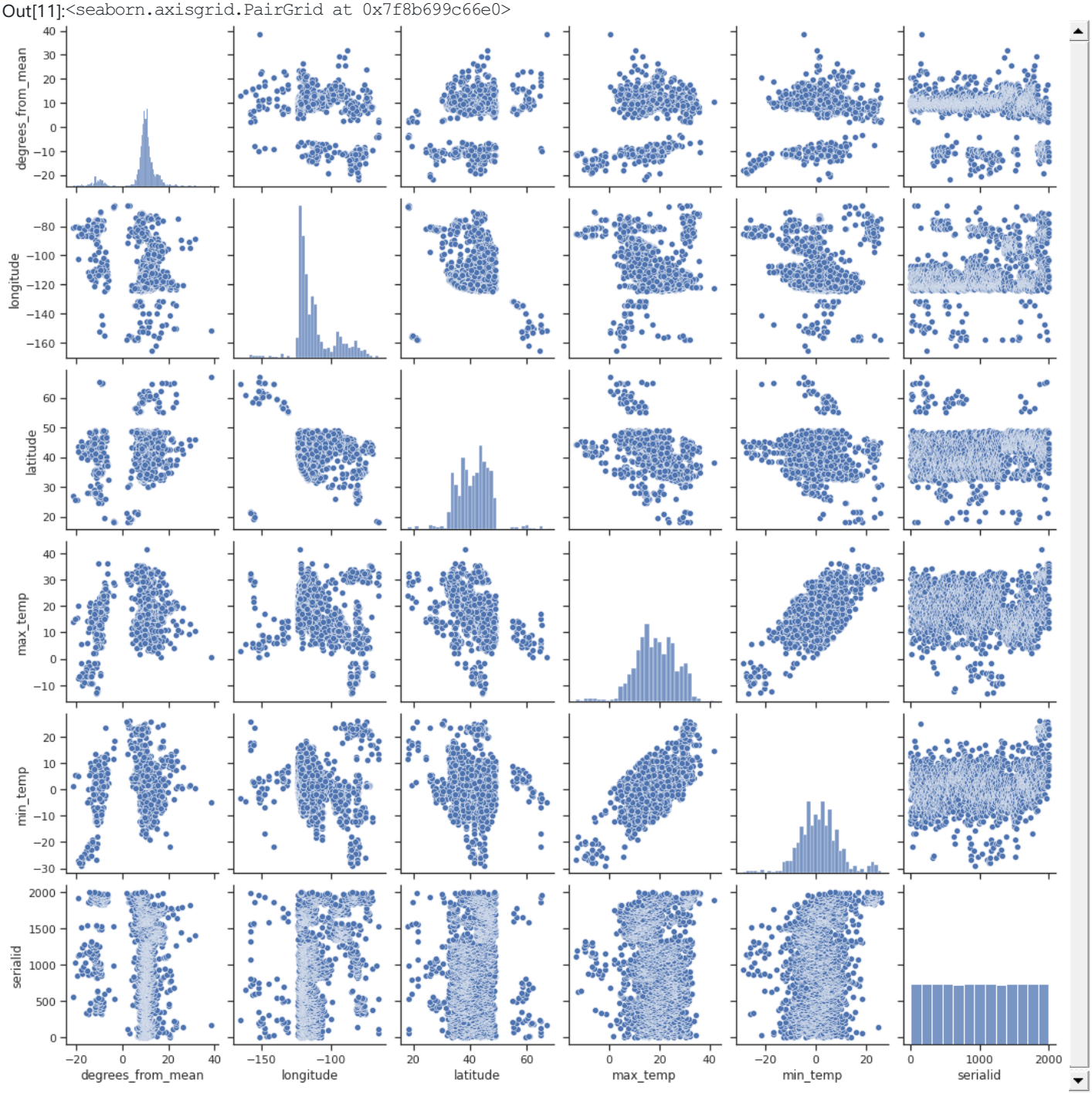
Вывод: Представленный набор данных не содержит пропусков.

Построение графиков для понимания структуры данных

Парные диаграммы

In [11]:
```
sns.pairplot(data)
```

Out[11]:<seaborn.axisgrid.PairGrid at 0x7f8b699c66e0>



Выведем измерения погоды на карте с помощью библиотеки PyGeoPlot (https://github.com/romovpa/pygeoplot)

Поскольку оригинальный репозиторий содержит неактуальный код (см. раздел `issues`), то в данной научной исследовательской работе будем использовать форк - https://github.com/irina-goltsman/pygeoplot

Данный форк тоже содержит неактуальный код, поэтому приведу инструкцию по установке:

1. Выполнить следующие команды в теминале

```
$ git clone https://github.com/irina-goltsman/pygeoplot
$ cd pygeoplot
$ python setup.py install
```

1. Отредактировать `__init__.py` в установленном пакете (дописать символ `.` так, чтобы имена были относительными):

```
from .api import *
from .display import *
from .util import *
```

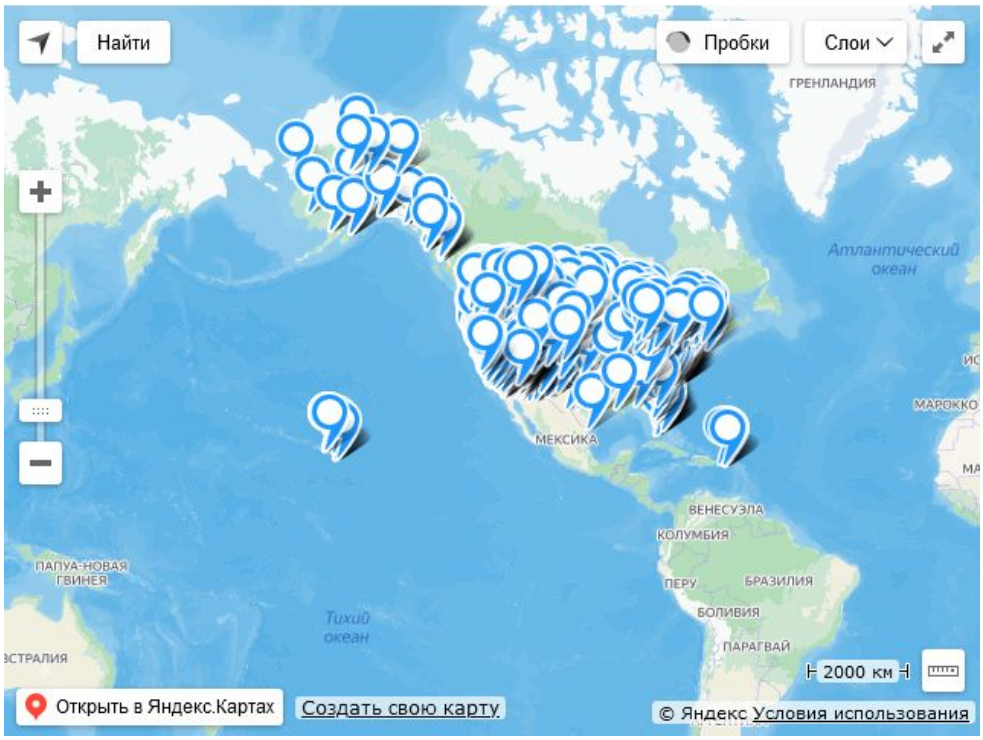
После выполнения этих дейсчтвий можно пользоваться библиотекой.
Ниже показаны на карте все данные из выбранного фрагмента датасета.

In [12]:
```
m = gp.api.Map() # Создаём объект карты
```

Добавляем точки на карту

In [13]:
```
for i in range(data.shape[0]):
    m.add_placemark((data.iloc[i]['latitude'], data.iloc[i]['longitude']), hint=data.iloc[i]['station_na
```

In [14]:
```
# Оцентруем карту относительно одной из точек набора данных
m.set_state((data.iloc[2]['latitude'], data.iloc[2]['longitude']), zoom=2)
m.display() # Выводим карту
```



Обработка данных

Кодирование категориальных признпков

Поскольку алгоритмы машинного обучения вог многих библиотеках не работают с категориальными признаками, то закодируекм их с помощью `LabelEncoder`

In [15]:
```
le = LabelEncoder()
    # "date_str"
le.fit(data.date_str.drop_duplicates())
data.date_str = le.transform(data.date_str)
    # "id"
le.fit(data.id.drop_duplicates())
data.id = le.transform(data.id)
    # "station_name"
le.fit(data.station_name.drop_duplicates())
data.station_name = le.transform(data.station_name)
    # "type"
le.fit(data.type.drop_duplicates())
data.type = le.transform(data.type)
```

In [16]:
```
data_clone = data.copy()
```

Масштабирование данных

Для улучшения качества алгоритмов машинного обучения отмасштабируем данные.

In [17]:
```
# Числовые колонки для масштабирования
scale_cols = ['degrees_from_mean', 'longitude', 'latitude', 'max_temp', 'min_temp']
```

In [18]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data_clone[scale_cols])
```

Добавим масштабированные данные в набор данных

In [19]:
```
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
```

```
        data_clone[new_col_name] = sc1_data[:,i]
```

In [20]:
```
data_clone
```

Out[20]:

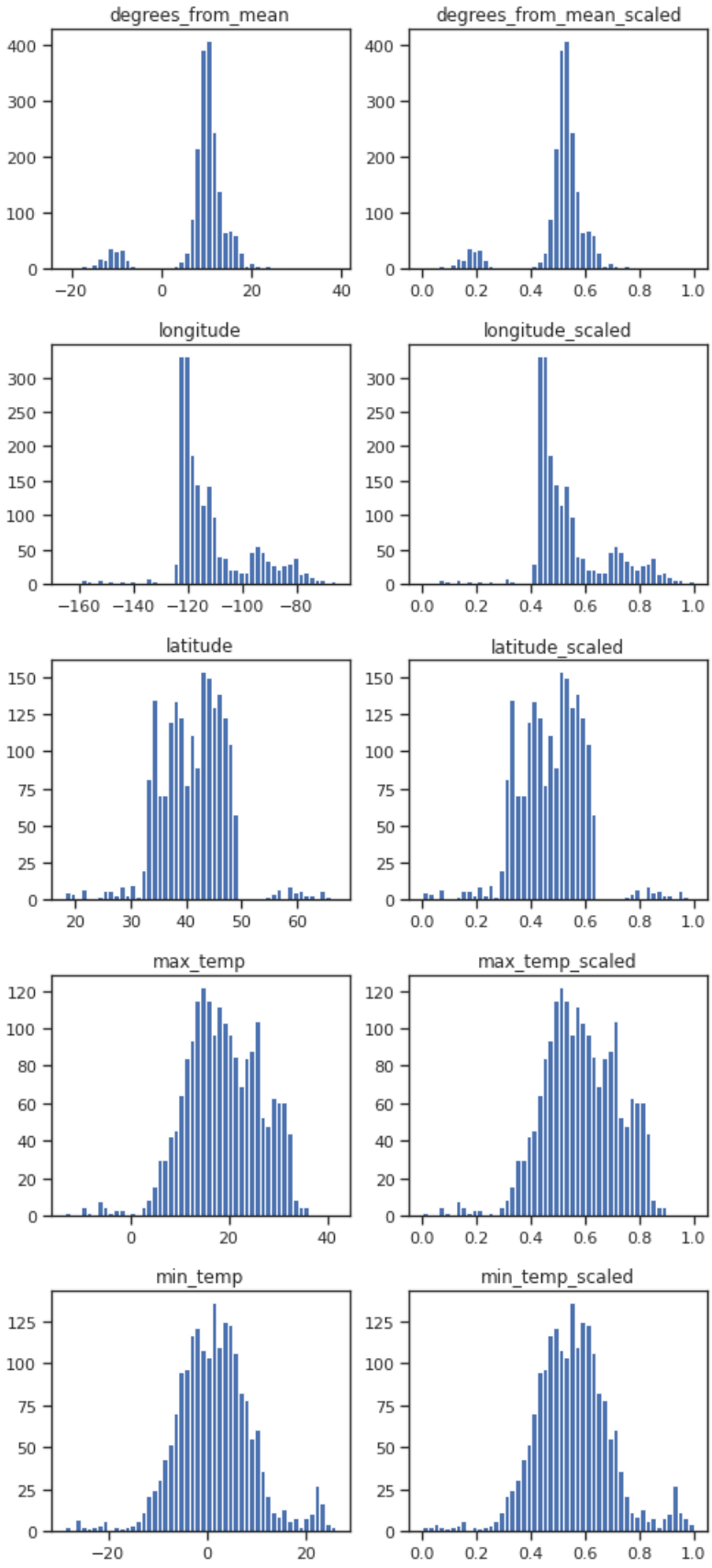
	date_str	degrees_from_mean	id	longitude	latitude	max_temp	min_temp	station_name	type	serialid	degrees_from_mean_scal
0	11	8.61	207	-113.5472	43.7186	10.0	-12.8	312	3	1	0.5009
1	11	10.74	160	-107.1097	37.7717	11.1	-8.9	345	3	2	0.5363
2	11	20.46	42	-119.5128	37.0919	25.6	12.8	30	1	3	0.6974
3	11	8.60	10	-109.7517	33.4783	20.0	-3.9	70	3	4	0.5008
4	11	10.30	65	-115.4508	33.8089	30.6	13.9	225	3	5	0.5296
...	...	...	...	...	...	...	...	...	...	...	
1995	12	-12.11	502	-121.1461	44.6633	17.2	7.8	454	2	1996	0.1574
1996	12	-14.36	373	-113.9847	48.5003	11.7	5.6	865	2	1997	0.1201
1997	12	8.28	751	-71.0106	42.3606	35.0	20.6	81	3	1998	0.4955
1998	12	9.83	811	-122.0481	37.4058	34.4	16.7	514	3	1999	0.5212
1999	12	8.12	296	-84.5542	43.0114	33.3	22.8	712	3	2000	0.4928

2000 rows × 15 columns

Проверяем, что масштабирование не повлияло на распределение данных

In [21]:
```
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data_clone[col], 50)
    ax[1].hist(data_clone[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```



Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

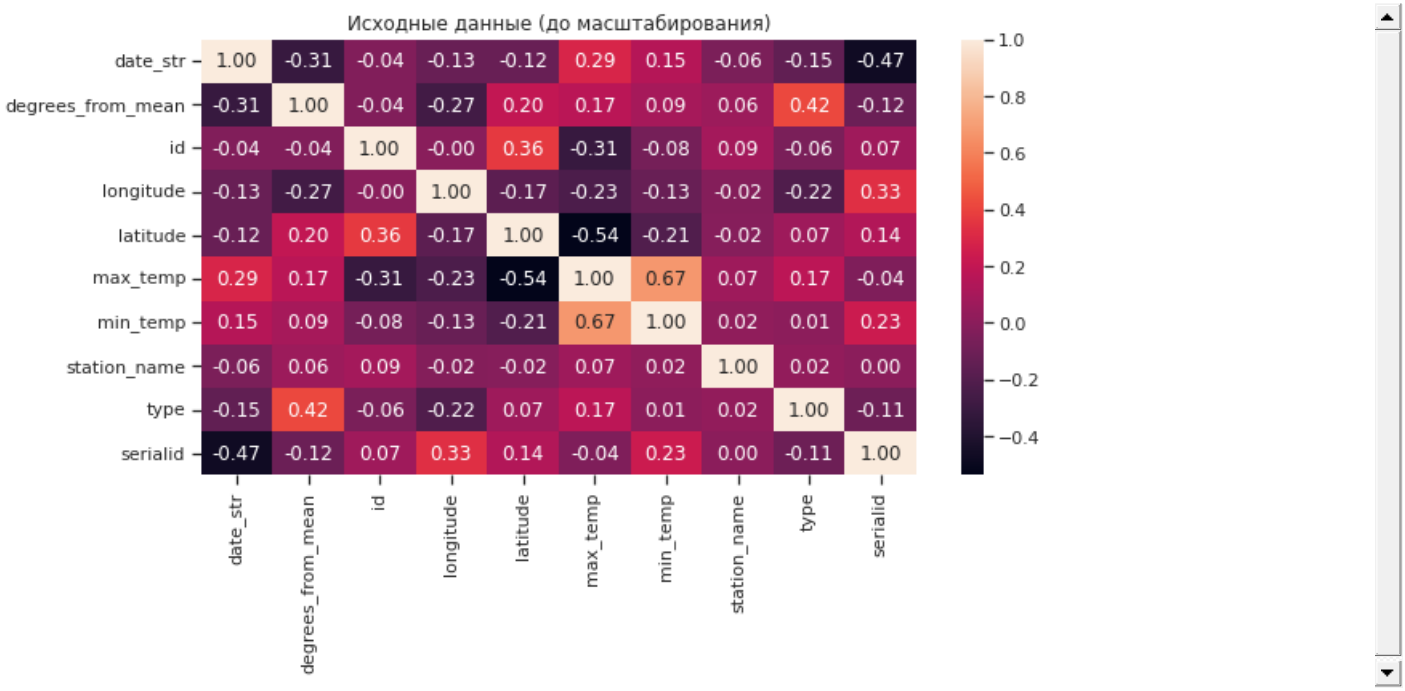
In [22]:
```
corr_cols_1 = data_clone.columns[:-5]
corr_cols_1
```

Out[22]:
```
Index(['date_str', 'degrees_from_mean', 'id', 'longitude', 'latitude',
       'max_temp', 'min_temp', 'station_name', 'type', 'serialid'],
      dtype='object')
```

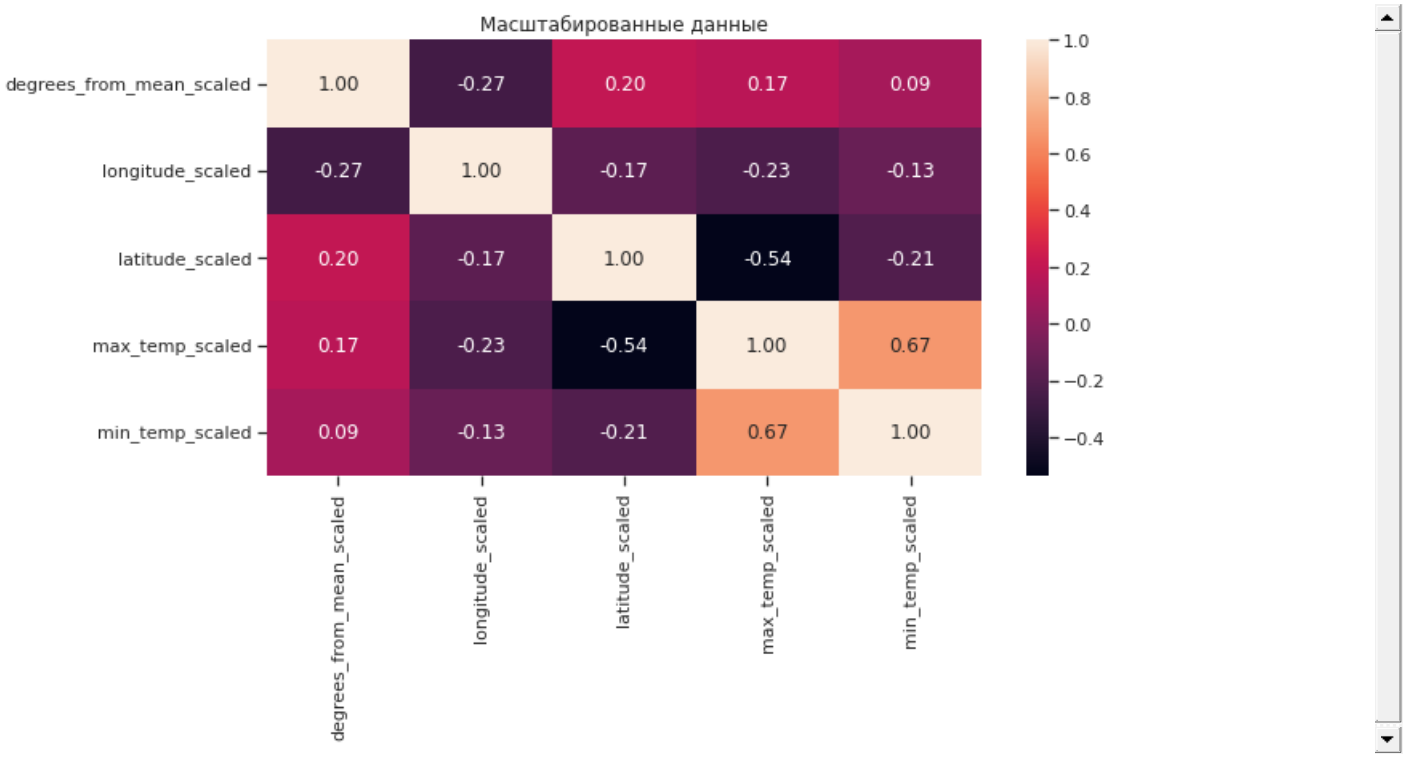
In [23]:
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix
corr_cols_2
```

Out[23]:
```
['degrees_from_mean_scaled',
 'longitude_scaled',
 'latitude_scaled',
 'max_temp_scaled',
 'min_temp_scaled']
```

In [24]:
```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_clone[corr_cols_1].corr(), annot=True, fmt='.2f')
ax.set_title('Исходные данные (до масштабирования)')
plt.show()
```



In [25]:
```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_clone[corr_cols_2].corr(), annot=True, fmt='.2f')
ax.set_title('Масштабированные данные')
plt.show()
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- ...

Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать:

Сохранение и визуализация метрик

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества.

In [26]:
```
class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = pd.concat([self.df, pd.DataFrame(temp)], ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.

Для задачи регрессии будем использовать следующие модели:

- Линейная регрессия
- Метод ближайших соседей
- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

Формирование обучающей и тестовой выборок на основе исходного набора данных.

На основе масштабированных данных выделим обучающую и тестовую выборки с помощью фильтра

In [27]:
```
X = data_clone[["date_str", "degrees_from_mean_scaled", "id", "longitude_scaled", "latitude_scaled", "mi
Y = data_clone["max_temp_scaled"]
print('Входные данные:\n\n', X.head(), '\n\nВыходные данные:\n\n', Y.head())
```

```
Входные данные:

    date_str  degrees_from_mean_scaled   id  longitude_scaled  latitude_scaled  \
0        11                  0.500995  207          0.519972         0.524524
1        11                  0.536306  160          0.584477         0.402632
2        11                  0.697447   42          0.460196         0.388698
3        11                  0.500829   10          0.558003         0.314631
4        11                  0.529012   65          0.500898         0.321407

   min_temp_scaled  station_name  type  serialid
0         0.292727           312     3         1
1         0.363636           345     3         2
2         0.758182            30     1         3
3         0.454545            70     3         4
4         0.778182           225     3         5

Выходные данные:

 0    0.423636
1    0.443636
2    0.707273
3    0.605455
4    0.798182
Name: max_temp_scaled, dtype: float64
```

In [28]:
```
X_train,  X_test,  Y_train,  Y_test = train_test_split(X,  Y, random_state = 2022, test_size = 0.1)
```

In [ ]:

Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

Решение задачи регрессии

In [29]:
```
# Модели
regr_models = {'LR': LinearRegression(),
               'KNN_5':KNeighborsRegressor(n_neighbors=5),
               'SVR':SVR(),
               'Tree':DecisionTreeRegressor(),
               'RF':RandomForestRegressor(),
               'GB':GradientBoostingRegressor()}
```

In [30]:
```
# Сохранение метрик
regrMetricLogger = MetricLogger()
```

In [31]:
```
def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)

    mae = mean_absolute_error(Y_test, Y_pred)
    mse = mean_squared_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)

    regrMetricLogger.add('MAE', model_name, mae)
    regrMetricLogger.add('MSE', model_name, mse)
    regrMetricLogger.add('R2', model_name, r2)

    print('{} \t MAE={}, MSE={}, R2={}'.format(
        model_name, round(mae, 3), round(mse, 3), round(r2, 3)))
```

In [32]:
```
for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
LR   MAE=0.05, MSE=0.005, R2=0.764
KNN_5   MAE=0.075, MSE=0.01, R2=0.466
SVR   MAE=0.081, MSE=0.012, R2=0.385
Tree   MAE=0.031, MSE=0.003, R2=0.86
RF   MAE=0.022, MSE=0.001, R2=0.946
GB   MAE=0.031, MSE=0.002, R2=0.904
```

Подбор гиперпараметров для выбранных моделей с использованием методов кросс-валидации.

In [67]:
```
n_range = np.array(range(1,1000,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[67]:
```
[{'n_neighbors': array([  1,   6,  11,  16,  21,  26,  31,  36,  41,  46,  51,  56,  61,
         66,  71,  76,  81,  86,  91,  96, 101, 106, 111, 116, 121, 126,
        131, 136, 141, 146, 151, 156, 161, 166, 171, 176, 181, 186, 191,
        196, 201, 206, 211, 216, 221, 226, 231, 236, 241, 246, 251, 256,
        261, 266, 271, 276, 281, 286, 291, 296, 301, 306, 311, 316, 321,
        326, 331, 336, 341, 346, 351, 356, 361, 366, 371, 376, 381, 386,
        391, 396, 401, 406, 411, 416, 421, 426, 431, 436, 441, 446, 451,
        456, 461, 466, 471, 476, 481, 486, 491, 496, 501, 506, 511, 516,
        521, 526, 531, 536, 541, 546, 551, 556, 561, 566, 571, 576, 581,
        586, 591, 596, 601, 606, 611, 616, 621, 626, 631, 636, 641, 646,
        651, 656, 661, 666, 671, 676, 681, 686, 691, 696, 701, 706, 711,
        716, 721, 726, 731, 736, 741, 746, 751, 756, 761, 766, 771, 776,
        781, 786, 791, 796, 801, 806, 811, 816, 821, 826, 831, 836, 841,
        846, 851, 856, 861, 866, 871, 876, 881, 886, 891, 896, 901, 906,
        911, 916, 921, 926, 931, 936, 941, 946, 951, 956, 961, 966, 971,
        976, 981, 986, 991, 996])}]
```

In [68]:
```
%%time
regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scoring='neg_mean_squared_error')
regr_gs.fit(X_train, Y_train)
```

```
CPU times: user 2min 11s, sys: 1min 55s, total: 4min 6s
Wall time: 30.2 s
```

Out[68]:
```
GridSearchCV(cv=5, estimator=KNeighborsRegressor(),
             param_grid=[{'n_neighbors': array([  1,   6,  11,  16,  21,  26,  31,  36,  41,  46,  51,
       56,  61,
        66,  71,  76,  81,  86,  91,  96, 101, 106, 111, 116, 121, 126,
       131, 136, 141, 146, 151, 156, 161, 166, 171, 176, 181, 186, 191,
       196, 201, 206, 211, 216, 221, 226, 231, 236, 241, 246, 251, 256,
       261, 266, 271, 276, 281, 286, 291, 296, 301, 306, 311, 316, 321,
       326, 331, 336, 341, 346, 351, 356,...
       586, 591, 596, 601, 606, 611, 616, 621, 626, 631, 636, 641, 646,
       651, 656, 661, 666, 671, 676, 681, 686, 691, 696, 701, 706, 711,
       716, 721, 726, 731, 736, 741, 746, 751, 756, 761, 766, 771, 776,
       781, 786, 791, 796, 801, 806, 811, 816, 821, 826, 831, 836, 841,
       846, 851, 856, 861, 866, 871, 876, 881, 886, 891, 896, 901, 906,
       911, 916, 921, 926, 931, 936, 941, 946, 951, 956, 961, 966, 971,
       976, 981, 986, 991, 996])}],
             scoring='neg_mean_squared_error')
```

In [69]:
```
# Лучшая модель
regr_gs.best_estimator_
```

Out[69]:
```
KNeighborsRegressor(n_neighbors=16)
```

In [70]:
```
# Лучшее значение параметров
regr_gs.best_params_
```

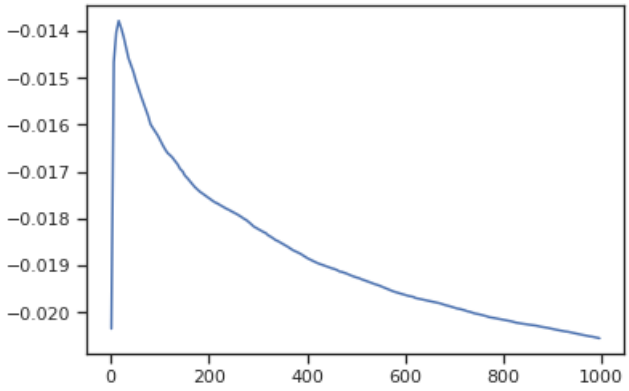
Out[70]:
```
{'n_neighbors': 16}
```

In [71]:
```
regr_gs_best_params_txt = str(regr_gs.best_params_['n_neighbors'])
regr_gs_best_params_txt
```

Out[71]:
```
'16'
```

In [72]:
```
# Изменение качества на тестовой выборке в зависимости от К-соседей
plt.plot(n_range, regr_gs.cv_results_['mean_test_score'])
```

Out[72]: [<matplotlib.lines.Line2D at 0x7f8b605b2c20>]



Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

In [73]:
```
regr_models_grid = {'KNN_5':KNeighborsRegressor(n_neighbors=5), str('KNN_'+regr_gs_best_params_txt):regr
```

In [74]:
```
for model_name, model in regr_models_grid.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
KNN_5   MAE=0.075, MSE=0.01, R2=0.466
KNN_16   MAE=0.074, MSE=0.01, R2=0.464
```

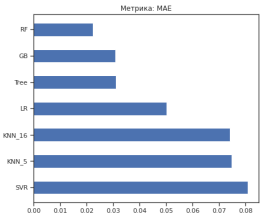
Формирование выводов о качестве построенных моделей на основе выбранных метрик.

Метрики качества модели

In [75]:
```
regr_metrics = regrMetricLogger.df['metric'].unique()
regr_metrics
```

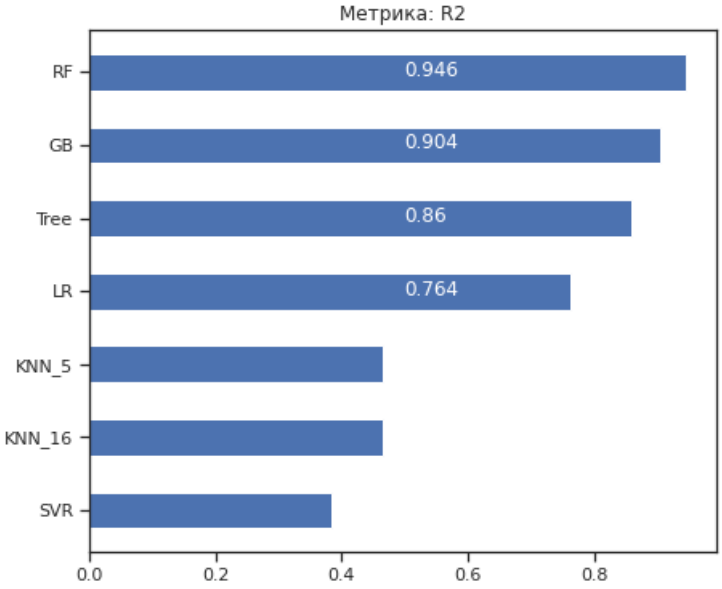
Out[75]: array(['MAE', 'MSE', 'R2'], dtype=object)

In [85]:
```
regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(7, 6))
```



In [86]:
```
regrMetricLogger.plot('Метрика: ' + 'MSE', 'MSE', ascending=False, figsize=(7, 6))
```

In [87]:
```
regrMetricLogger.plot('Метрика: ' + 'R2', 'R2', ascending=True, figsize=(7, 6))
```



Вывод

На основании двух метрик из трёх используемых, лучшей оказалась модель на основе метода опорных векторов (SVR)