

# МОДЕЛЬ СОЗДАНИЯ АРХИВА РАСЧЁТНЫХ ПОГОДНЫХ ДАННЫХ для конкретной локации по данным нескольких референсных метеостанций

**Цель:** Воссоздать архив погодных явлений для локации Агробиостанции МГУ в пос. Чашниково Солнечногорского р-она Московской области

*==== Тетрадь 2: Исследование аномалий, пропусков и ошибок, а также восстановление, исправление и моделирование значений для каждого индивидуального параметра: численные показатели температуры воздуха ( $T$ ,  $T_{min}$ ,  $T_{Max}$ ) ===*

## 0. Подготовка данных 2-й тетради

### 0.0. Импорт необходимых библиотек и настройки представления

```
In [1]: # Python interpreter version: 3.9.12
# Системная конфигурация
import sys

# Работа с файловой системой
from os import listdir
from os.path import isfile, join
from os import makedirs

# Вывод данных
from pprint import pprint
from io import StringIO
```

```
# Вычисления
from math import degrees, radians, cos, sin, asin, atan, sqrt, floor, log
import numpy as np # v. 1.22.3
import pandas as pd # v. 1.4.4
from scipy.stats import norm # v.1.9.1
from scipy.spatial.distance import pdist # v.1.9.1

# Работа с временем и датами
import datetime as dt
import time

# Работа со строками
import re
import pymorphy2 # v. 0.9.1

# Машинное обучение
#...
# - подготовка данных
from sklearn.model_selection import train_test_split # v.1.1.2
# - метрики качества
from sklearn.metrics import max_error, mean_absolute_error, mean_squared_error, r2_score # v.1.1.2
# - Библиотека SciKit GStat:
import skgstat as skg # v. 1.0.1

# Построение визуализаций
import matplotlib as mpl # v. 3.5.2
import matplotlib.pyplot as plt # v. 3.5.2
import matplotlib.lines as mlines # v. 3.5.2
import seaborn as sns # v. 0.12.0
# import seaborn.objects as so
import missingno as msno # v. 0.4.2

# EDA tools
import sweetviz as sv
#from pandas_profiling import ProfileReport

# Настройки
import warnings
```

Настроим отображение вывода результатов кода в нескольких ячейках

```
In [2]: from IPython.core.interactiveshell import InteractiveShell  
InteractiveShell.ast_node_interactivity = "all"
```

Для удобства отображения данных изменим опции максимум отображаемых строк и столбцов.

```
In [3]: pd.set_option('display.max_rows', 400) # изменим максимум отображаемых строк  
pd.set_option('display.max_columns', 50) # изменим максимум отображаемых столбцов
```

Установим для Seaborn настройки темы по умолчанию.

```
In [4]: sns.set_theme()
```

## 0.1. Импорт данных

```
In [5]: # Определим значения переменных path  
path = path = 'data/csv/' # общих путь к данным  
raw_path1 = path + 'raw/locations/' # путь к архивам метеостанций  
raw_path2 = path + 'raw/' # путь к архивам параметров
```

### 0.1.1. Данные метеостанций

#### 0.1.1.1. Информация о метеостанциях

```
In [6]: # Загружаем файл с общей информацией о метеостанциях  
df_stations = pd.read_csv(filepath_or_buffer=path + 'df_stations.csv', index_col=0)  
df_stations.head(2)  
# Загружаем файл с расстояниями между метеостанциями  
df_station_dists = pd.read_csv(filepath_or_buffer=path + 'df_station_dists.csv', index_col=0)  
df_station_dists.head(2)  
# Загружаем файл с начальными азимутами между метеостанциями  
df_station_bearings = pd.read_csv(filepath_or_buffer=path + 'df_station_bearings.csv', index_col=0)  
df_station_bearings.head(2)  
# Загружаем файл с линейными координатами метеостанций  
df_stations_lin_coords = pd.read_csv(filepath_or_buffer=path + 'df_stations_lin_coords.csv', index_col=0)  
df_stations_lin_coords.head(2)
```

Out[6]:

	station_name	station_ID	height	latitude	longitude	degree_lo	minute_lo	lo	degree_la	minute_la	la	comment
0	Вышний Волочек	26393	161	N57.583333	E34.566667	57	35	N	34	34	E	validate 26393.11.07.2005.09.06.2022.1.0
1	Старица	26499	185	N56.500000	E34.933333	56	30	N	34	56	E	validate 26499.01.02.2005.09.06.2022.1.0

Out[6]:

	station_ID	station	LaN	LoE	height	V_Volochek	Staritsa	Kashyn	Tver	Klin	Dmitrov	Volokolamsk	Mozhais
0	26393	V_Volochek	57.583333	34.566667	161	0.000000	122.520236	182.287149	114.810932	190.589931	225.030989	193.100035	246.07462
1	26499	Staritsa	56.500000	34.933333	185	122.520236	0.000000	186.562842	72.307124	111.270810	160.570705	81.891761	127.90687

Out[6]:

	station_ID	station	LaN	LoE	height	V_Volochek	Staritsa	Kashyn	Tver	Klin	Dmitrov	Volokolamsk	Mozhais
0	26393	V_Volochek	57.583333	34.566667	161	0.000000	349.720491	279.454332	315.263759	317.734920	308.198351	335.03763	339.67437
1	26499	Staritsa	56.500000	34.933333	185	169.41282	0.000000	240.670335	237.073671	280.332008	276.381371	311.44173	329.20554

Out[6]:

	station_ID	station	LaN	LoE	height	lin_ver	lin_hor
0	26393	V_Volochek	57.583333	34.566667	161	296.603273	0.00000
1	26499	Staritsa	56.500000	34.933333	185	176.108200	23.44064

## 0.1.2. Данные архивов погоды

### 0.1.2.1. Чтение архивов метеостанций

In [7]:

```
# Создадим список файлов с архивами метеостанций
list_df_locations_files = [file_name for file_name in listdir(raw_path1) if isfile(join(raw_path1, file_name))]
list_df_locations_files
```

```
Out[7]: ['df_Dmitrov.csv',
 'df_Kashyn.csv',
 'df_Klin.csv',
 'df_Mozhaisk.csv',
 'df_Naro_Fominsk.csv',
 'df_Nemchinovka.csv',
 'df_N_Jerusalem.csv',
 'df_Serpukhov.csv',
 'df_Staritsa.csv',
 'df_Tver.csv',
 'df_Volokolamsk.csv',
 'df_V_Volochev.csv']
```

```
In [8]: dict_df_locations = {} # Инициализируем словарь датафреймов
for file_name in list_df_locations_files: # По списку csv файлов
    name_df = file_name[:-4] # Вычленяем название датафрейма из названия файла
    file_path = raw_path1 + file_name # Формируем путь к файлу
    print(file_path, ' - ', end='') # КОНТРОЛЬ: Обрабатываемый файл
    # Создаём ключ (название DF) из названия файла
    # и записываем в словарь по этому ключу соответствующий DF
    dict_df_locations[f'{name_df}'] = pd.read_csv(file_path, index_col=0, parse_dates=True, infer_datetime_format = True, low_memory=True)
    print('O.K.')
# Выводим полученные ключи словаря
dict_df_locations.keys()
```

```
data/csv/raw/locations/df_Dmitrov.csv - O.K.
data/csv/raw/locations/df_Kashyn.csv - O.K.
data/csv/raw/locations/df_Klin.csv - O.K.
data/csv/raw/locations/df_Mozhaisk.csv - O.K.
data/csv/raw/locations/df_Naro_Fominsk.csv - O.K.
data/csv/raw/locations/df_Nemchinovka.csv - O.K.
data/csv/raw/locations/df_N_Jerusalem.csv - O.K.
data/csv/raw/locations/df_Serpukhov.csv - O.K.
data/csv/raw/locations/df_Staritsa.csv - O.K.
data/csv/raw/locations/df_Tver.csv - O.K.
data/csv/raw/locations/df_Volokolamsk.csv - O.K.
data/csv/raw/locations/df_V_Volochev.csv - O.K.
```

```
Out[8]: dict_keys(['df_Dmitrov', 'df_Kashyn', 'df_Klin', 'df_Mozhaisk', 'df_Naro_Fominsk', 'df_Nemchinovka', 'df_N_Jerusalem', 'df_Serpukhov', 'df_Staritsa', 'df_Tver', 'df_Volokolamsk', 'df_V_Volochev'])
```

```
In [9]: for name in dict_df_locations.keys():
    dict_df_locations[name].sample(3)
```

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_ci
Dmitrov_Local_time													
<b>2012-07-05 21:00:00</b>	22.8	748.7	764.3	NaN	61.0	ENE	67.5	1125.0	1.0	NaN	NaN	от 90 менее 100%	No
<b>2021-10-19 15:00:00</b>	3.8	744.3	760.9	NaN	79.0	W	270.0	4500.0	3.0	NaN	NaN	от 90 менее 100%	Облача в целе образовывали и развивали
<b>2010-10-13 12:00:00</b>	0.2	730.7	747.3	NaN	94.0	NW	315.0	5250.0	4.0	NaN	NaN	100%.	Ливневый сн слабый в ср наблюден или зд

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
Kashyn_Local_time													
2020-09-28 12:00:00	15.8	757.2	769.7	-0.3	48.0	SE	135.0	2250.0	2.0	NaN	NaN	70 – 80%.	NaN
2019-12-01 21:00:00	-6.9	746.5	760.0	0.1	89.0	SW	225.0	3750.0	1.0	NaN	NaN	20–30%.	NaN
2008-02-20 09:00:00	-10.4	743.7	757.1	NaN	85.0	NW	315.0	5250.0	3.0	NaN	NaN	60%.	Состояние неба в общем не изменилось.

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr	Prc
Klin_Local_time														
2007-03-31 09:00:00	7.4	751.6	767.0	NaN	63.0	штиль	360.0	6000.0	0.0	NaN	NaN	0%	NaN	
2010-06-09 09:00:00	16.1	743.7	758.4	NaN	85.0	W	270.0	4500.0	2.0	NaN	NaN	от 90 менее 100%	NaN	
2017-02-15 06:00:00	-2.9	750.4	766.2	1.9	73.0	NNW	337.5	5625.0	4.0	NaN	11.0	70 – 80%.	Состояние неба в общем не изменилось.	Сс изм

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
<b>Mozhaisk_Local_time</b>													
<b>2007-07-25 03:00:00</b>	18.2	738.6	754.6	NaN	62.0	SE	135.0	2250.0	2.0	NaN	NaN	от 90 менее 100%	NaN
<b>2005-09-18 12:00:00</b>	7.6	752.0	769.0	NaN	78.0	NNW	337.5	5625.0	2.0	NaN	NaN	от 90 менее 100%	NaN
<b>2013-01-16 09:00:00</b>	-9.0	743.6	761.6	NaN	90.0	SSE	157.5	2625.0	1.0	NaN	NaN	100%.	Ливневый снег слабый в срок наблюдения или за ...

Out[9]:

T P\_station P\_sea P\_drift Humid Wind\_dir Wind\_dir360 Wind\_dir6k Wind\_speed Gusts Gusts\_3h Cloudness Wt

Naro Fominsk Local time

Погодные условия													Состояние	Время
Погодные условия													Состояние	Время
2016-04-23 03:00:00	1.2	733.3	751.0	0.7	97.0	WNW	292.5	4875.0	2.0	NaN	NaN	NaN	от 90 менее 100%	от 90 менее 100%
2010-01-21 00:00:00	-13.3	764.1	783.8	NaN	85.0	ENE	67.5	1125.0	2.0	NaN	NaN	NaN	от 90 менее 100%	непрер сл
2021-04-27 00:00:00	1.9	740.2	758.0	-0.7	86.0	штиль	360.0	6000.0	0.0	NaN	NaN	NaN	100%.	наблю

Out[9]:

T P station P sea P drift Humid Wind dir Wind dir360 Wind dir6k Wind speed Gusts Gusts 3h Cloudness Wth

Nemchinovka Local time

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_cri
	N_Jerusalem_Local_time												
	<b>2021-07-25 03:00:00</b>	10.5	749.8	764.2	0.7	95.0	штиль	360.0	6000.0	0.0	NaN	NaN	40%.
	<b>2007-01-13 12:00:00</b>	1.8	730.6	745.4	NaN	92.0	SE	135.0	2250.0	1.0	NaN	NaN	от 90 менее 100%
	<b>2012-07-23 06:00:00</b>	12.3	753.4	767.9	NaN	95.0	W	270.0	4500.0	2.0	NaN	NaN	20–30%.

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_cri
	Serpukhov_Local_time												
	<b>2019-06-28 09:00:00</b>	13.9	728.1	742.3	NaN	94.0	W	270.0	4500.0	3.0	NaN	10.0	100%.
	<b>2014-03-12 12:00:00</b>	4.5	750.2	765.4	0.1	53.0	N	0.0	0.0	5.0	NaN	NaN	70 – 80%.
	<b>2017-11-12 12:00:00</b>	4.4	738.7	753.7	NaN	93.0	SW	225.0	3750.0	3.0	NaN	NaN	100%.

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
Staritsa_Local_time													
<b>2021-09-12 15:00:00</b>	22.6	742.6	758.5	-0.4	60.0	WNW	292.5	4875.0	2.0	NaN	NaN	60%.	NaN
<b>2018-05-17 09:00:00</b>	15.2	744.4	760.8	-0.1	90.0	SSE	157.5	2625.0	3.0	NaN	NaN	100%.	Ливневый(ые) дождь(и).
<b>2019-10-11 21:00:00</b>	8.1	741.1	758.0	0.3	90.0	SW	225.0	3750.0	2.0	NaN	NaN	от 90 менее 100%	NaN

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
Tver_Local_time													
<b>2015-11-11 06:00:00</b>	0.8	732.3	745.7	0.9	98.0	WNW	292.5	4875.0	1.0	NaN	NaN	100%.	Дымка.
<b>2005-02-09 12:00:00</b>	-11.4	763.0	777.7	NaN	90.0	WSW	247.5	4125.0	1.0	NaN	NaN	100%.	Снег непрерывный в слабый в срок наблюдения.
<b>2020-12-17 18:00:00</b>	1.1	750.7	764.5	1.0	92.0	WSW	247.5	4125.0	1.0	NaN	NaN	70 – 80%.	NaN

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_c
Volokolamsk_Local_time													
<b>2020-09-25 03:00:00</b>	8.5	746.2	764.3	NaN	93.0	S	180.0	3000.0	1.0	NaN	NaN	NaN	N
<b>2015-01-12 00:00:00</b>	0.4	719.4	737.2	1.2	92.0	WSW	247.5	4125.0	2.0	NaN	NaN	100%.	Ливнев снег слаб в ср наблюд или з.
<b>2021-06-17 21:00:00</b>	19.1	751.3	768.7	NaN	51.0	штиль	360.0	6000.0	0.0	NaN	NaN	0%	N

Out[9]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_c
V_Volochek_Local_time													
2014-02-21 09:00:00	-3.4	745.3	761.3	1.4	82.0	SE	135.0	2250.0	1.0	NaN	NaN	от 90 менее 100%	N
2007-12-25 03:00:00	-1.3	754.6	771.4	NaN	86.0	SW	225.0	3750.0	2.0	NaN	NaN	от 90 менее 100%	N
2019-01-20 09:00:00	-3.6	737.0	753.0	0.7	89.0	SSW	202.5	3375.0	4.0	NaN	NaN	100%. непрерывн слабы сф наблюден	Ci

### 0.1.2.2. Чтение архивов параметров

In [10]:

```
# Создадим список файлов с архивами параметров
list_df_parameters_files = [file_name for file_name in.listdir(raw_path2) if isfile(join(raw_path2, file_name))]
list_df_parameters_files
```

```
Out[10]: ['df_Cloudness.csv',
 'df_Cl_bottom.csv',
 'df_Cl_cirrus.csv',
 'df_Cl_Cumls.csv',
 'df_Cl_cumls_hi.csv',
 'df_Cl_viewd.csv',
 'df_Depo_diam_mm.csv',
 'df_Dew_point.csv',
 'df_Gusts.csv',
 'df_Gusts_3h.csv',
 'df_Humid.csv',
 'df_Prcptn.csv',
 'df_Prcptn_depo.csv',
 'df_Prcptn_like.csv',
 'df_Prcptn_tdelts.csv',
 'df_P_drift.csv',
 'df_P_sea.csv',
 'df_P_station.csv',
 'df_Snow_height.csv',
 'df_Soil.csv',
 'df_Soil_cover.csv',
 'df_Soil_T.csv',
 'df_T.csv',
 'df_T_max.csv',
 'df_T_min.csv',
 'df_Visibility.csv',
 'df_Wind_dir.csv',
 'df_Wind_dir360.csv',
 'df_Wind_dir6k.csv',
 'df_Wind_speed.csv',
 'df_Wthr_3h.csv',
 'df_Wthr_3h2.csv',
 'df_Wthr_curr.csv']
```

```
In [11]: # Запишем данные архивов параметров в словарь
dict_df_parameters = {} # Инициализируем словарь датафреймов
for file_name in list_df_parameters_files: # По списку csv файлов
    name_df = file_name[:-4] # Вычленяем название датафрейма из названия файла
    file_path = raw_path2 + file_name # Формируем путь к файлу
    print(file_path, ' - ', end='') # КОНТРОЛЬ: Обрабатываемый файл
    # Создаём ключ (название DF) из названия файла
    # и записываем в словарь по этому ключу соответствующий DF
    dict_df_parameters[f'{name_df}'] = pd.read_csv(
        file_path, index_col=0, parse_dates=True, infer_datetime_format = True, low_memory=False)
```

```
print('O.K.')
# Выводим полученные ключи словаря
dict_df_parameters.keys()

data/csv/raw/df_Cloudness.csv - O.K.
data/csv/raw/df_Cl_bottom.csv - O.K.
data/csv/raw/df_Cl_cirrus.csv - O.K.
data/csv/raw/df_Cl_Cumls.csv - O.K.
data/csv/raw/df_Cl_cumls_hi.csv - O.K.
data/csv/raw/df_Cl_viewd.csv - O.K.
data/csv/raw/df_Depo_diam_mm.csv - O.K.
data/csv/raw/df_Dew_point.csv - O.K.
data/csv/raw/df_Gusts.csv - O.K.
data/csv/raw/df_Gusts_3h.csv - O.K.
data/csv/raw/df_Humid.csv - O.K.
data/csv/raw/df_Prcpttn.csv - O.K.
data/csv/raw/df_Prcpttn_depo.csv - O.K.
data/csv/raw/df_Prcpttn_like.csv - O.K.
data/csv/raw/df_Prcpttn_tdelt.csv - O.K.
data/csv/raw/df_P_drift.csv - O.K.
data/csv/raw/df_P_sea.csv - O.K.
data/csv/raw/df_P_station.csv - O.K.
data/csv/raw/df_Snow_height.csv - O.K.
data/csv/raw/df_Soil.csv - O.K.
data/csv/raw/df_Soil_cover.csv - O.K.
data/csv/raw/df_Soil_T.csv - O.K.
data/csv/raw/df_T.csv - O.K.
data/csv/raw/df_T_max.csv - O.K.
data/csv/raw/df_T_min.csv - O.K.
data/csv/raw/df_Visibility.csv - O.K.
data/csv/raw/df_Wind_dir.csv - O.K.
data/csv/raw/df_Wind_dir360.csv - O.K.
data/csv/raw/df_Wind_dir6k.csv - O.K.
data/csv/raw/df_Wind_speed.csv - O.K.
data/csv/raw/df_Wthr_3h.csv - O.K.
data/csv/raw/df_Wthr_3h2.csv - O.K.
data/csv/raw/df_Wthr_curr.csv - O.K.

Out[11]: dict_keys(['df_Cloudness', 'df_Cl_bottom', 'df_Cl_cirrus', 'df_Cl_Cumls', 'df_Cl_cumls_hi', 'df_Cl_viewd', 'df_Depo_diam_mm', 'df_Dew_point', 'df_Gusts', 'df_Gusts_3h', 'df_Humid', 'df_Prcpttn', 'df_Prcpttn_depo', 'df_Prcpttn_like', 'df_Prcpttn_tdelt', 'df_P_drift', 'df_P_sea', 'df_P_station', 'df_Snow_height', 'df_Soil', 'df_Soil_cover', 'df_Soil_T', 'df_T', 'df_T_max', 'df_T_min', 'df_Visibility', 'df_Wind_dir', 'df_Wind_dir360', 'df_Wind_dir6k', 'df_Wind_speed', 'df_Wthr_3h', 'df_Wthr_3h2', 'df_Wthr_curr'])
```

```
In [12]: for name in dict_df_parameters.keys():
    dict_df_parameters[name].sample(3)
```

	Cloudness#V_Volochev	Cloudness#Staritsa	Cloudness#Kashyn	Cloudness#Tver	Cloudness#Klin	Cloudness#Dmitrov	Cloudness#Volokolamsk	Cloudness#Leningrad
<b>2011-12-16 03:00:00</b>	от 90 менее 100%	от 90 менее 100%	от 90 менее 100%	60%.	100%.	100%.	100%.	100%.
<b>2013-04-08 21:00:00</b>	0%	20–30%.	60%.	40%.	больше 0 до 10%	больше 0 до 10%	60%.	
<b>2006-10-22 00:00:00</b>	NaN	100%.	NaN	100%.	100%.	100%.	100%.	

	Cl_bottom#V_Volochev	Cl_bottom#Staritsa	Cl_bottom#Kashyn	Cl_bottom#Tver	Cl_bottom#Klin	Cl_bottom#Dmitrov	Cl_bottom#Volokolamsk	Cl_bottom#Leningrad
<b>2017-07-21 06:00:00</b>	600-1000	NaN	300-600	2500 или более, или облаков нет.	600-1000	NaN	NaN	NaN
<b>2020-04-08 09:00:00</b>	600-1000	600-1000	600-1000	600-1000	600-1000	300-600	NaN	NaN
<b>2010-01-26 15:00:00</b>	2500 или более, или облаков нет.	2000-2500	300-600	2500 или более, или облаков нет.				

Out[12]:

	Cl_cirrus#V_Volochev	Cl_cirrus#Staritsa	Cl_cirrus#Kashyn	Cl_cirrus#Tver	Cl_cirrus#Klin	Cl_cirrus#Dmitrov	Cl_cirrus#Volokolamsk	Cl_cirrus#N
2019-05-14 15:00:00	Перистые нитевидные, иногда когтевидные, не ра...	Перистые нитевидные, иногда когтевидные, не ра...	Перистых, перисто-кучевых или перисто-слоистых...	Перисто-или г... сл...				
2014-04-06 00:00:00	Перистых, перисто-кучевых или перисто-слоистых...	Перистые нитевидные, иногда когтевидные, не ра...	Перистых, перисто-кучевых или перисто-слоистых...	NaN	Перистые нитевидные, иногда когтевидные, не ра...	Перисто-слоистые, не распространяющиеся по неб...	Перистые плотные в виде кла... скрученных...	Перистые в виде кла... скрученных...
2011-03-28 06:00:00	NaN	NaN	NaN	Перистые нитевидные, иногда когтевидные, не ра...	NaN	Перистые нитевидные, иногда когтевидные, не ра...	Перистые нитевидные, иногда когтевидные, не ра...	Перистые и в виде кло... скрученных...

Out[12]:

	Cl_Cumls#V_Volochev	Cl_Cumls#Staritsa	Cl_Cumls#Kashyn	Cl_Cumls#Tver	Cl_Cumls#Klin	Cl_Cumls#Dmitrov	Cl_Cumls#Volokolamsk	Cl_Cur
2013-11-13 00:00:00	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоистые туманообразные или слоистые разорванн...	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоистые туманообразные или слоистые разорванн...	Слоисто-кучевые, образовавшиеся не из кучевых.	NaN	NaN	NaN
2014-05-26 03:00:00	Слоисто-кучевых, слоистых, кучевых или кучево-...	Слоисто-кучевые, слоистых, кучевых или кучево-...	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.			
2021-07-28 12:00:00	Кучевые плоские или кучевые разорванные, или т...	Кучево-дождевые волокнистые (часто с наковальн...	Слоисто-кучевых, слоистых, кучевых или кучево-...	Кучево-дождевые лысые с кучевыми, слоисто-куче...	Кучевые и слоисто-кучевые (но не слоисто-куче...	Кучево-дождевые лысые с кучевыми, слоисто-куче...	Кучево-дождевые волокнистые (часто с наковальн...	Кучевые волокнистые

	Cl_cumls_hi#V_Volochev	Cl_cumls_hi#Staritsa	Cl_cumls_hi#Kashyn	Cl_cumls_hi#Tver	Cl_cumls_hi#Klin	Cl_cumls_hi#Dmitrov	Cl_cumls_hi#Volokolamsk
2007-02-23 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2012-02-21 18:00:00	Высококучевых, высокослоистых или слоисто-дожд...	Высококучевые просвечающие, расположенные на...	Высококучевые просвечающие, расположенные на...	Высококучевые просвечающие, полосами, либо о...	Высококучевых, высокослоистых или слоисто- дожд...	Высококучевых, высокослоистых или слоисто-дожд...	Высококуч высокослоисты слоисто-д
2012-05-04 09:00:00	Высококучевых, высокослоистых или слоисто-дожд...	NaN	NaN	NaN	NaN	NaN	NaN

	Cl_viewd#V_Volochev	Cl_viewd#Staritsa	Cl_viewd#Kashyn	Cl_viewd#Tver	Cl_viewd#Klin	Cl_viewd#Dmitrov	Cl_viewd#Volokolamsk	Cl_viewd#M
2017-08-24 03:00:00	90 или более, но не 100%	100%.	40%.	60%.	100%.	100%.	100%.	NaN
2014-03-29 00:00:00	NaN	NaN	NaN	NaN	40%.	60%.	70 – 80%.	
2019-07-29 21:00:00	70 – 80%.	90 или более, но не 100%	20–30%.	10% или менее, но не 0	60%.	90 или более, но не 100%	90 или более, но не 100%	

	Depo_diam_mm#V_Volochev	Depo_diam_mm#Staritsa	Depo_diam_mm#Kashyn	Depo_diam_mm#Tver	Depo_diam_mm#Klin	Depo_diam_mm#Dm
2016-03-27 18:00:00	0.0	0.0	0.0	0.0	0.0	0.0
2017-12-20 09:00:00	0.0	0.0	0.0	0.0	0.0	1.0
2020-09-21 21:00:00	0.0	0.0	0.0	0.0	0.0	0.0

Out[12]:	Dew_point#V_Volochev	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
<b>2020-10-15 00:00:00</b>	6.9	10.0	7.8	8.2	11.8	11.7	NaN
<b>2012-07-14 00:00:00</b>	13.8	11.9	13.0	12.6	13.4	13.7	12.6
<b>2009-12-19 21:00:00</b>	-16.4	-17.2	-18.3	-17.6	-21.1	-17.4	-17.0



Out[12]:	Prcptn_like#V_Volochev	Prcptn_like#Staritsa	Prcptn_like#Kashyn	Prcptn_like#Tver	Prcptn_like#Klin	Prcptn_like#Dmitrov	Prcptn_like#Volokolamsk		
2013-05-31 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN		
2016-08-24 03:00:00	Ливневый(ые) дождь(и) слабый(ые) в срок наблюд...	NaN	Состояние неба в общем не изменилось.	Туман или ледяной туман, неба не видно, без за...	Гроза (с осадками или без них).	Гроза слабая или умеренная без града, но с дож...	Облака в рассеиваются становятся...		
2016-04-04 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN		
Out[12]:	Prcptn_tdelt#V_Volochev	Prcptn_tdelt#Staritsa	Prcptn_tdelt#Kashyn	Prcptn_tdelt#Tver	Prcptn_tdelt#Klin	Prcptn_tdelt#Dmitrov	Prcptn_tdelt#Volokolamsk		
2020-11-27 21:00:00	12.0	12.0	12.0	12.0	12.0	12.0	12.0		
2017-12-08 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN		
2009-03-13 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN		
Out[12]:	P_drift#V_Volochev	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#Volokolamsk
2012-07-08 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-11-07 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2018-09-20 21:00:00	0.3	0.6	0.6	0.6	0.5	NaN	NaN	NaN	NaN





Out[12]:	T#V_Volochek	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2019-08-01 15:00:00	12.3	12.6	13.4	13.4	13.4	14.5	13.4	13.8	13.9	15.2	14
2020-04-22 12:00:00	6.2	6.8	5.1	6.9	5.6	5.4	6.1	6.7	6.1	5.3	6
2017-04-30 21:00:00	7.5	9.4	17.0	11.5	19.1	19.9	20.0	20.6	17.8	19.1	18
Out[12]:	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N_Jer		
2009-09-10 18:00:00	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
2009-09-25 00:00:00	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
2015-02-04 12:00:00	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
Out[12]:	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer		
2021-06-28 15:00:00	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
2011-03-09 15:00:00	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
2021-02-13 06:00:00	-17.5	-16.1	-17.0	-15.6	-16.6	Nan	Nan	Nan	Nan	Nan	Nan

Out[12]:	Visibility#V_Volochek	Visibility#Staritsa	Visibility#Kashyn	Visibility#Tver	Visibility#Klin	Visibility#Dmitrov	Visibility#Volokolamsk	Visibility#Moscow
2021-02-19 03:00:00	10.0	20.0	10.0	10.0	10.0	10.0	10.0	10.0
2010-07-23 18:00:00	NaN	20.0	NaN	10.0	20.0	12.0	40.0	
2009-04-15 00:00:00	NaN	10.0	NaN	2.0	20.0	10.0	20.0	
Out[12]:	Wind_dir#V_Volochek	Wind_dir#Staritsa	Wind_dir#Kashyn	Wind_dir#Tver	Wind_dir#Klin	Wind_dir#Dmitrov	Wind_dir#Volokolamsk	Wind_dir#Moscow
2005-05-13 21:00:00	NaN	SE	NaN	SE	SSE	S	SSE	
2008-01-01 09:00:00	SE	SE	S	SE	штиль	SSE	SSE	
2010-08-18 09:00:00	SE	SSE	SE	SSE	S	SSW	S	
Out[12]:	Wind_dir360#V_Volochek	Wind_dir360#Staritsa	Wind_dir360#Kashyn	Wind_dir360#Tver	Wind_dir360#Klin	Wind_dir360#Dmitrov	Wind_dir360#Volokolamsk	Wind_dir360#Moscow
2013-01-05 18:00:00	22.5	0.0	360.0	270.0	360.0	337.5		
2005-12-05 15:00:00	NaN	135.0	112.5	112.5	157.5	157.5		
2011-01-08 12:00:00	NaN	180.0	NaN	225.0	180.0	180.0		

Out[12]:	Wind_dir6k#V_Volochek	Wind_dir6k#Staritsa	Wind_dir6k#Kashyn	Wind_dir6k#Tver	Wind_dir6k#Klin	Wind_dir6k#Dmitrov	Wind_dir6k#Volokola
2015-02-16 21:00:00	6000.0	6000.0	5250.0	4500.0	6000.0	4875.0	5625.0
2007-06-27 06:00:00	Nan	2250.0	Nan	2250.0	3000.0	2250.0	2625.0
2017-12-27 12:00:00	4125.0	3750.0	3375.0	3375.0	3750.0	3750.0	3750.0

Out[12]:	Wind_speed#V_Volochev	Wind_speed#Staritsa	Wind_speed#Kashyn	Wind_speed#Tver	Wind_speed#Klin	Wind_speed#Dmitrov	Wind_speed#Vol
2012-12-19 18:00:00	1.0	0.0	1.0	1.0	1.0	3.0	
2019-12-29 03:00:00	1.0	4.0	2.0	1.0	2.0	2.0	
2013-02-14 15:00:00	0.0	1.0	2.0	2.0	2.0	1.0	

Out[12]:		Wthr_3h2#V_Volochev	Wthr_3h2#Staritsa	Wthr_3h2#Kashyn	Wthr_3h2#Tver	Wthr_3h2#Klin	Wthr_3h2#Dmitrov	Wthr_3h2#Volokolamsk	Wthr...
	2014-05-10 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	2005-07-07 06:00:00	NaN	NaN	NaN	NaN	Облака покрывали более половины неба в течение...	NaN	Облака покрывали более половины неба в течение...	
	2017-08-03 00:00:00	Облака покрывали более половины неба в течение...	Ливень (ливни).	Ливень (ливни).	Ливень (ливни).	Ливень (ливни).	Ливень (ливни).	Ливень (ливни).	NaN

Out[12]:		Wthr_curr#V_Volochev	Wthr_curr#Staritsa	Wthr_curr#Kashyn	Wthr_curr#Tver	Wthr_curr#Klin	Wthr_curr#Dmitrov	Wthr_curr#Volokolamsk	Wt...
	2020-06-26 09:00:00	NaN	NaN	NaN	NaN	Состояние неба в общем не изменилось.	Дымка.	NaN	
	2016-02-24 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	2019-11-23 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

# 1. Теоретические основы

## 1.1. Описание задачи

Основная цель: получить статистически значимые изолированные наблюдения без ошибок и пропусков во всём поле метеостанций, по каждому параметру, на каждый заданный момент наблюдения, и на их основе смоделировать значения тех же параметров для Агробиостанции МГУ. Для реализации поставленной цели необходимо будет подобрать оптимальный способ восстановления данных для каждого параметра, который может быть положен в основу составления архивов

**метеонаблюдений для локации Агробиостанции МГУ в пос. Чашниково Солнечногорского района Московской области. ВАЖНО:**  
время наблюдений, фиксируемое в архивах погодных явлений, определяется по Гринвичу.

*На первом этапе необходимо отделить выбросы, имеющие физический смысл, от выбросов, являющихся ошибками ведения архивов. На выходе мы должны получить такие значения параметров, которые на самом деле являются реальными измерениями реальных погодных явлений (то есть имеют физический и метеорологический смысл) и потому должны быть исследованы и смоделированы, даже если они могут восприниматься как выбросы.*

Изначально нами была предпринята попытка анализа всех имеющихся данных одновременно и поиска ошибок на основе анализа выбросов за пределами трёх сигм от средней величины (то есть, вне пределов вероятности в 99,7%). Эта попытка оказалась неудачной.

- Во-первых, совокупность метеостанций составляет всего 12. При таком количестве любое аномальное значение будет приводить к сильному смещению распределения в свою сторону, и, как следствие, аномальное значение может оказаться внутри доверительного интервала, хотя и близко к его границе.
- Во-вторых, несмотря на географическую и природную близость, из-за особенностей метеорологических явлений и иногда их локального характера, запределенные значения не всегда могут свидетельствовать об ошибках.
- В-третьих, при наличии нескольких пропущенных значений совокупность еще более сужается, ошибочное значение может трактоваться, как нормальное и также находится внутри доверительного интервала.

Поэтому ниже мы будем принимать во внимание, среди прочего:

- отклонение индивидуальных значений от средней между станциями в пределах от не только 3 сигм, но и менее; при этом подсчёт средней и показателей вариации не будет включать само проверяемое значение,
- нахождение индивидуального значения в пределах заданного доверительного интервала для гипотетического нормального распределения,
- отклонение индивидуальных значений от ближайших значений наблюдения по времени (до и после),
- климатические и сезонные нормы (например, <https://ru.wikipedia.org/> - климат Московской области, климат Тверской области). Каждый параметр (группу родственных параметров) будем рассматривать отдельно. Заполнение пропусков будем выполнять отдельно и после удаления ошибок (кроме случаев, когда ошибки возможно исправить на основе вычисления взаимозависимых значений).

Для более детального анализа будем учитывать временные границы сезонов. Исходя из данных о климате Московской области, временные границы сезонов определены следующим образом.

- Зима (ниже 0°): в среднем длится с 5 ноября по 4 апреля.
- Весна (от 0° до +10°): в среднем длится с 5 апреля по 18 мая.
- Лето (выше +10°): в среднем длится с 19 мая по 14-15 сентября.
- Осень (от +10° до 0°): в среднем длится с 14-15 сентября по 4 ноября.

Следует иметь в виду следующий порядок организации метеонаблюдений, принятый в Российской Федерации:

- наблюдения на всех метеорологических станциях проводят синхронно восемь раз в сутки в 00, 03, 06, 09, 12, 15, 18 и 21 ч по гринвичскому времени;
- во все сроки измеряют температуру воздуха и почвы, влажность воздуха, скорость ветра и его направление, метеорологическую дальность видимости, атмосферное давление, определяют характеристики облачности;
  - другие величины, не имеющие хорошо выраженного суточного хода, определяют не во все сроки и даже между сроками. Так, состояние поверхности почвы и осадки определяют два раза в сутки в сроки, ближайшие к 8 и 20 ч местного времени пояса, в котором расположена станция. Высоту снежного покрова, глубину промерзания почвы измеряют один раз в утренний срок, ближайший к 08 ч декретного времени данного пояса. Снегомерные съемки производят один раз в 10 дней, а весной перед началом и в период таяния снега – один раз в 5 дней. Испарение измеряют один раз в 5 дней, влажность почвы – один раз в 10 дней (на 8-й день 30 декады). Ленты термографа, гигрографа, барографа меняют в срок, ближайший к 13 ч, а плювиографа – к 20 ч местного времени.
- за начало суток на каждой станции принимают единый срок, ближайший к 20 ч, а за первый срок наблюдений – срок, ближайший к 23 ч местного времени;
- так как произвести измерения всеми приборами точно в срок наблюдений нельзя, принято при восьмисрочных наблюдениях температуру и влажность воздуха измерять за 10 мин, а давление воздуха – за 2 мин до срочного часа. Все остальные измерения начинают за 30 мин до срока и заканчивают после срока. Общая продолжительность наблюдений составляет 30 – 40 мин.

Однако используемые нами архивы метеорологических наблюдений, полученные с интернет ресурса [www.rp5.ru](http://www.rp5.ru) используют местное время метеорологических наблюдений.

## **1.2. Общий алгоритм поиска ошибок, их исправления, а также восстановления пропущенных значений**

*Основная цель при работе с выбросами: отделить выбросы имеющие физический смысл от ошибок. По своей структуре наши данные могут быть описаны в следующих измерениях:*

1. Географическое пространство метеостанций (поле):

- географическое положение (координаты и высота над уровнем моря),
- производные от них расстояния и начальные азимуты.

2. Временной ряд:

- однонаправленный, приведённой к частоте дискретизации в 1 наблюдение за 3 часа,
- определяет суточные колебания и сезонные колебания.

3. Набор параметров с различной степенью взаимной зависимости.

- дискретный ряд параметров,
- взаимозависимые параметры, которые могут служить подтверждением или опровержением для значений в увязке с временным рядом и географическими параметрами.

Для выявления ошибок необходимо:

1. Определить диапазон допустимых значений для каждого исследуемого параметра и границы вероятности отклонения от среднего наблюдаемого значения в географическом пространстве и во времени. Для каждого параметра такие границы следует устанавливать индивидуально.

2. Найти выбросы:

- в поле метеостанций на момент наблюдения
- в окне временного ряда (с учётом исследуемого параметра, его суточных и сезонных колебаний и с учётом климатических норм, если таковые существуют).

1. Если выброс является одновременно и выбросом в поле метеостанций, и в окне временного ряда, и тем более, если он не совпадает с расчётными значениями из других параметров (если зависимость между ними существует и взаимозависимые параметры не содержат ошибок), то этот выброс следует расценивать как ошибку.

Мы исходим из того, что все ошибки в архивах являются ошибками ввода данных и в большинстве случаев являются одной или одновременно несколькими следующими ошибками:

- ошибкой в разряде,
- ошибкой в знаке,
- ошибкой лишней или недостающей цифры,
- ошибкой неверной цифры (чаще всего схожей по начертанию).

Все ошибки такого рода буду приводить к очень значительными выбросам.

Для исправления ошибок возможен один или несколько способов:

1. Привести значение в соответствие с допустимым интервалом значений для поля метеостанций,
2. Привести значение в соответствие с допустимым интервалом значений для временного ряда,
3. Если возможно, произвести расчёт корректного значения, исходя из корректных значений других параметров,
4. Логически исправить ошибку (вручную, если количество ошибок позволяет это сделать),
5. Удалить ошибочное значение и заменить его прогнозом, исходя из выбранной модели экстраполяции данных.

Необходимо подобрать модель экстраполяции данных, критерием здесь могут служить:

- метрики качества,
- для хороших метрик качества - соотношение трудоемкости использования других моделей и потенциального дальнейшего улучшения метрик.

Далее нужно произвести моделирование значений для:

- пропусков в архивах,
- условной метеостанции Чашниково.

В конечном итоге полученные значения будут записаны в архивы.

## 1.3. Подход к моделированию отсутствующих значений (пропущенных и намеренно удалённых)

### 1.3.1. Метеорологические показатели, подлежащие моделированию

Конечное назначение данной модели - воссоздание тех данных, которые будут критичными для анализа снегового покрова. Поэтому была составлена иерархия данных по степени критичности для последующего исследования снегового покрова.

Сортировка параметров приводится *по убыванию степени критичности*.

1. Необходимые данные о предмете исследования:

- Snow\_height (sss) Высота снежного покрова, см;

- Soil\_cover (E') Состояние поверхности почвы со снегом или измеримым ледяным покровом.

1. Существенные данные для анализа предмета исследования:

- T(T) Температура воздуха на высоте 2 м над поверхностью земли, градусов Цельсия;
- P\_sea (P) Атмосферное давление, приведённое к среднему уровню моря, мм рт. столба;
- Humid (U) Относительная влажность воздуха на высоте 2 м над поверхностью земли, %;
- Wind\_dir (DD) Направление ветра на высоте 10-12 м над поверхностью земли, усреднённое за 10 минут непосредственно перед наблюдением, румбы;
- Wind\_speed, (Ff) - Скорость ветра на высоте 10-12 м над поверхностью земли, усреднённая за 10 минут непосредственно перед наблюдением, м/с.;
- Prcpttn (RRR) Количество выпавших осадков, мм;
- Prcpttn\_tdelt (tR) Период времени, за который выпало указанное количество осадков, ч.;

1. Желательные данные:

- Wthr\_curr (WW) Текущая погода, сообщаемая метеостанцией - *в части описания осадков только!*
- Wthr\_3h (W1) Прошедшая погода между сроками наблюдения 1 - *в части описания осадков только!*
- Wthr\_3h2 (W2) Прошедшая погода между сроками наблюдения 2 - *в части описания осадков только!*

1. Некритичные данные:

- Soil (E) Состояние поверхности почвы без снега или измеримого ледяного покрова (*желательно для смежных исследований*);
- Soil\_T (Tg) Минимальная температура поверхности почвы за ночь, градусов Цельсия (*желательно для смежных исследований*);
- Gusts (ff10) Максимальная скорость порыва ветра на высоте 10-12 м над поверхностью земли, за 10 минут непосредственно перед наблюдением, м/с.;
- Gusts\_3h (ff3) Максимальная скорость порыва ветра на высоте 10-12 м над поверхностью земли, за период между моментами наблюдения 3 часа, м/с.;
- Visibility (VV) Горизонтальная дальность видимости, км.;
- Cloudness (N) Общая облачность, %;
- Cl\_bottom (H) Высота основания самых низких облаков, м;
- Cl\_Cumls (Cl) Слоисто-кучевые, слоистые, кучевые и кучево-дождевые облака;
- Cl\_viewed (Nh) Количество всех наблюдающихся облаков Cl, или при отсутствии облаков Cl, количество всех наблюдающихся облаков Cm, %;

- Cl\_cumuls\_hi (Cm) Высоко-кучевые, высоко-слоистые и слоисто-дождевые облака;
- Cl\_cirrus (Ch) Перистые, перисто-кучевые и перисто-слоистые облака.

1. Вычисляемые и зависимые данные:

- P\_station (Po) Атмосферное давление на уровне станции, мм рт. столба;
- P\_drift (Pa) Барическая тенденция: изменение атмосферного давления за последние 3 часа, мм рт. столба;
- T\_min (Tn) Минимальная температура воздуха за прошедший период не более 12 часов, градусов Цельсия;
- T\_max (Tx) Максимальная температура воздуха за прошедший период не более 12 часов, градусов Цельсия;
- Dew\_poin (Td) Температура точки росы на высоте 2 м над поверхностью земли, градусов Цельсия.

### **1.3.2. Последовательность моделирования метеорологических показателей.**

Не смотря на заданную последовательность значимости показателей для предмета исследования, последовательность моделирования будет несколько иной.

1. Мы начнём с основополагающих групп показателей: температура, давление, влажность. Связанные с ними вычисляемые показатели будем рассчитывать сразу же. Эти данные содержат менее всего пропусков, хорошо коррелируются между собой и являются наиболее значимыми для анализа погодных явлений, по ним можно ориентироваться при оценке корректности значений других показателей.
2. Далее перейдём к моделированию хорошо коррелирующихся численных значений состояния почвы и высоты снегового покрова.
3. После чего перейдём к моделированию других показателей с учётом их зависимости между собой и значимости для предмета исследования (снегового покрова).

В первую очередь будут исследованы и смоделированы числовые непрерывные показатели, имеющие между собой лучшую корреляцию. И лишь после этого - дискретные и категориальные. Более того, для моделирования значений нам могут понадобиться показатели, отнесенные к группе "некритичных данных", в этом случае их придется исследовать и обрабатывать в первую очередь.

### **1.3.3. Выяснение параметров (фич) для модели**

Архивные данные содержат большое количество пропусков и ошибок. Как было показано в первой тетради, пропуски по некоторым показателями наблюдения зачастую занимают большие периоды по временной оси. Некоторые метеостанции не фиксировали отдельные показатели годами. В связи с этим попытка моделирования данных, опираясь на значения по временной оси не

представляется перспективной. В ряде случаев (где проущено очень много моментов наблюдения подряд) они просто не дадут результата. Надо также учитывать, что сезонные и суточные колебания, хоть и подобны, но практически никогда не дублируются. Поэтому предсказания на их основе будут иметь достаточно низкую точность. Гораздо более точные предсказания можно получить из наблюдений в различных, относительно близко расположенных, точках, объединённых одним моментом времени.

**A.** Для каждого зафиксированного момента времени  $t$  для каждого наблюдаемого параметра  $Z$  мы должны найти такую функцию  $f$  для совокупности  $station_n$ , которая с большой долей вероятности даст значение показателя  $(Z(t))$  для условной метеостанции *chashnikovo*. То есть:  $chashnikovo(Z(t)) = f(station_1(Z(t)), \dots, station_n(Z(t)))$ . Для каждого метеорологического показателя эта зависимость может быть разной, поэтому у нас может быть столько моделей, сколько и показателей метеонаблюдений.

Фичами для нас являются не данные архива, а данные о метеостанциях. А это - данные о географическом положении метеостанций. А географические параметры (высота над уровнем моря и положение метеостанций в пространстве относительно друг друга и метеостанции *chashnikovo*) - суть величины постоянные для каждой  $station_n$ , то есть, их влияние как факторов на модель для каждого данного параметра  $Z$  будет тоже постоянным.

Временной ряд показывает только итерации наблюдений. Чем длиннее временной ряд, тем больше вариаций значений параметров мы имеем для поиска искомой функции.

В идеале, мы можем вывести для каждого наблюдаемого значения погодного явления функцию для определения значения этого явления в заданной точке по значениям на  $n$  заданных метеостанциях. Эта функция, скорее всего, будет рабочей в рамках одной климатической зоны. Коррелирующие между собой метеорологические явления проявятся в подобии функций поиска их значений для Чашниково. Но на качество предсказываемой величины корреляция параметров погоды не повлияет, потому что для каждого параметра будет искааться своя функция.

Тем не менее, используя жестко коррелирующие между собой метеорологические показатели, мы можем облегчить себе задачу и сократить количество показателей, для которых нужно составлять индивидуальную модель. Если сейчас их 29, то мы можем взять меньшее число, и уменьшить количество индивидуальных моделей. Мы знаем, что некоторые показатели погодных явлений имеют физическую зависимость между собой. Следовательно, зная (или выявив) уравнения для такой зависимости, мы можем в искомой точке воссоздать их значения, не прибегая к моделированию зависимостей между метеостанциями.

### Ограничения.

1. Для создания моделей нам нельзя брать разные метеостанции для обучения, валидации и предсказания. Нам нужно брать разные данные по моментам наблюдения.

2. Важно, чтобы между или в непосредственной близости рядом с метеостанциями и искомой точкой не было зон с другим микроклиматом (больших водоёмов, высоких неровностей рельефа, зон плотной городской застройки и климатической "грелки" вроде Москвы). Именно по этой причине мы не можем взять для расчётов показания метеостанций, расположенных в черте Москвы.

**При условии достаточно высокой корреляции между данными метеостанций, для каждой модели параметра наблюдения определяющими фичами будут географическое положение метеостанций и их результирующая дальность от искомой точки.**

**Б.** Пространственная экстраполяция может оказаться не всегда возможной. Как показано в первой тетради, целый ряд показателей имеет недостаточную корреляцию между метеостанциями. В этом случае придётся использовать другой подход. Для плохо географически коррелируемых показателей потребуется выбирать фичи исходя из физического смысла показателя и его зависимостей от других показателей. Отбор фич здесь надо будет делать индивидуально для каждого такого показателя. Принципиальным будет то, что для поиска целевой величины нам нужно будет использовать показатели, предварительно очищенные от ошибок и пропусков, чтобы получить максимально чистые предсказания целевых величин. Этот момент определяет последовательность обработки и моделирования метеорологических явлений.

**При условии недостаточно высокой корреляции между метеостанции, для каждого показателя придется создавать свою модель с учётом физических особенностей явления, которое он выражает. Набор параметров для модели в этом случае придётся определять индивидуально.**

### 1.3.4. Модели пространственной экстраполяции геостатистических данных

Основная идея пространственной экстраполяции заключена в моделях IDW (от английского *Inverse distance weighting* - взвешивание по обратным расстояниям).

Основная идея заключается в предположении, что значения  $x$  в точке  $i$  ( $x_i$ ) и  $x$  в точке  $i + h$  ( $x_{i+h}$ ) тем ближе между собой, чем меньше расстояние  $h$  между ними. Функция зависимости индивидуального значения показателя от значений того же показателя в других географических точках таким образом имеет вид:

$$Z_u = \frac{\sum_i^N w_i * Z(i)}{\sum_i^N w_i}$$

где  $Z_u$  - предсказываемое значение показателя в точке с отсутствием наблюдения, находящейся среди  $N$  наблюдаемых точек.

Значения показателей в наблюдаемых точках  $Z(i)$  взвешиваются и усредняются. Веса рассчитываются как:

$$w_i = \frac{1}{\|\overrightarrow{ux_i}\|}$$

где  $u$  - точка с отсутствием наблюдаемого значения, а  $x_i$  - точка с имеющимися значениями наблюдения.

Таким образом,  $\|\overrightarrow{ux_i}\|$  - это нормированный вектор между двумя точками - Евклидово расстояние в пространстве координат (которое отнюдь не ограничивается именно  $\mathbb{R}^2$ )

Так как более близкие точки имеют большее влияние на исходную точку, то используется обратная пропорциональность расстояниям. Отсюда и взвешивание по обратным расстояниям (IDW).

Распространённым подходом к созданию таких моделей является кригинг, основанный на вариограммах - реализован в библиотечных функциях SciKit GStat, GTools. Мы так же можем использовать взвешивание по обратным степеням расстояний (согласно закону обратных квадратов, сила воздействия многих физических явлений изменяется обратнопропорционально квадрату расстояния от источника), тогда

$$w_i = \frac{1}{\|\overrightarrow{ux_i}\|} \text{ будет иметь вид } w_i = \frac{1}{(ux_i)^p}$$

где  $p$  - степень, в которую возводятся веса.

### 1.3.5. Иные модели.

В тех случаях, когда модели пространственной экстраполяции окажутся неприменимыми, конкретные модели предсказания целевых значений придётся выбирать индивидуально. В зависимости от корреляции выработанных фич и целевой величиной это могут быть регрессоры, деревья, нейросети, классификаторы и кластеризаторы, ансамбли моделей. Если для пространственной экстраполяции значения параметров имеют общую с моделируемым значением структуру распределения и диапазон значений, то здесь надо будет иметь в виду необходимость предварительной нормализации входных данных. Дополнительно необходимо оговорить, что не все показатели имеют числовое выражение. Часть содержащихся в архивах данных являются категориальными.

## 2. Общие функции для поиска ошибок, моделирования и исправления значений параметров

## 2.1. Функция рассчёта средней, взвешенной по обратной степени расстояний

```
In [13]: def inverse_distance_avg(row_, param_, station_='Rfrnce_point', df_dists_=df_station_dists, power_=2):
    """
    Расчитывает среднюю, взвешенную по обратным степеням расстояния, исключая данную метеостанцию.
    РАБОТАЕТ ТОЛЬКО ДЛЯ АРХИВА МЕТЕОНАБЛЮДЕНИЙ ПО ПАРАМЕТРАМ!
    РАБОТАЕТ ТОЛЬКО ДЛЯ ПРОСТРАНСТВА МЕТЕОСТАНЦИЙ!
    Принимает:
    - серия значений из которых необходимо рассчитать среднюю, взвешенную по обратным степеням расстояния;
    - название параметра, для которого рассчитывается средняя;
    - название метеостанции для которой рассчитывается средняя (её значение исключается из подсчёта средней!)
        default='Rfrnce_point' - геометрический центр поля метеостанций;
    - df_dists_ DF с расстояниями между метеостанциями default=df_station_dists;
    - power_ степень для вычисления обратных весов default=2
    Возвращает:
    - значение средней, взвешенной по обратным степеням расстояния от точки, заданной параметром stations_
        (вес w = 1000000(d**power_), во избежание слишком малых значений). Во избежание представления 1 в виде
        десятичной дроби с 99..998 после запятой, результат округляется до 12 знака после запятой.
    ПРИМЕЧАНИЕ: чтобы вычислить общую среднюю для всего поля метеостанций без исключений необходимо использовать значение
        staton_ по умолчанию - 'Rfrnce_pint'
    """
    # удаляем из серии значений row_ все значения не являющиеся значениями параметров (символ # в названии индекса -
    # признак того, что этому индексу соответствует значение параметра)
    counter_ = 0 # счётчик для индекса
    ##
    # print(row_.index.tolist())
    for item_ in (row_.index.tolist()): # название индекса по индексу серии row_
        if '#' not in item_: # если в названии индекса нет символа '#'
            row_ = row_.drop(row_.index[counter_]) # удаляем из серии элемент (не являющийся значением параметра)
        else:
            counter_ += 1 # в противном случае увеличиваем счётчик на 1

    list_param_per_inv_dists_ = [] # Список значений параметров, умноженных на обратную степень расстояний
    weight_sum_ = 0 # сумма весов для вычисления средней

    # объединяем в zip названия df станций (индекс серии) и значений параметра им соответствующих (собственно серия)
    # имя df станции и значение параметра в zip
    for name_st_, param_val_ in zip(row_.index, row_):
        # к каждому названию df станции применяем уменьшение слева на длинну строки с названием параметра + 1 знак ('#')
        # так как название столбцов состоит из названия параметра + '#' + название станции
        name_st_ = name_st_[len(param_) + 1 :]
```

```

# расстояние равно значению строке в df_station_dists где station равно станции, для которой вычисляется средняя
# а name_st - столбцу в df_station_dists
## 
#     print(station_, name_st_)
distance_ = df_station_dists.loc[(df_station_dists.station == station_), name_st_].values[0]

# если расстояние не равно 0 (чтобы исключить искомую станцию) и текущее значение не является NaN
if (distance_ != 0) and (not pd.isna(param_val_)):
    # добавляем в список значение параметра для этой станции умноженное на обратную степень расстояния
    # умножим его на 1000 в степени power_, чтобы исключить чрезвычайно малые значения в делителе при расчёте средней
    # множитель 1000 в степени power_ нужен чтобы компенсировать увеличение делителя при увеличении степени
    list_param_per_inv_dists_.append(((1000/distance_)**power_)*param_val_)
    # увеличиваем сумму весов на текущую обратную степень расстояния, умноженную также 1000 в степени power_
    weight_sum_ += (1000/distance_)**power_

else:
    pass

# результат: сумма значений параметров (кроме искомой станции), умноженных на обратные степени расстояний
# до соответствующей станции от искомой станции и, делённая на сумму обратных степеней расстояний.
result_ = np.around(np.nanmean(list_param_per_inv_dists_)/weight_sum_, 12) if weight_sum_ != 0 else np.nan
# Сумма без NaN

return result_

```

## 2.2. Функция пространственной экстраполяции данных методом кригинга

Сначала определим необходимые для функции значения

Для нашего случая, погрешность в исчислении расстояний между угловыми координатами и прямоугольными (в соответствующей проекции) крайне незначительна. Поэтому, для вариаграммы будем использовать углы широт и долгот.

```
In [14]: # Определим df с полным списком координат всех точек: Нужно для данной функции
df_coords_full = df_station_dists[["LoE", "LaN"]]
df_coords_full.index = df_station_dists.station
# df_coords_full = df_stations_lin_coords[["lin_hor", "lin_ver"]]
# df_coords_full.index = df_stations_lin_coords.station
```

```
In [15]: # Вывод на экран через print() сильно замедляет работу. Чтобы временно заблокировать вывод на экран предупреждений типа:  
# 'Warning: for %d locations, not enough neighbors were found within the range.' % self.no_points_error  
# определим отдельный класс с пустым выводом и направим на него вывод функции  
class NullIO(StringIO): # Класс нулевого вывода  
    def write(self, txt):  
        pass
```

```
In [16]: # сохраним стандартные настройки вывода  
real_stdout = sys.stdout # сделаем backup стандартного вывода
```

```
In [17]: def kriging_extrapolation(row_, df_coords_=df_coords_full):  
    """  
    Расчитывает значения для пространственной экстраполяции данных с помощью модели обычного кригинга.  
    РАБОТАЕТ ТОЛЬКО ДЛЯ АРХИВА МЕТЕОНАБЛЮДЕНИЙ ПО ПАРАМЕТРАМ!  
    РАБОТАЕТ ТОЛЬКО ДЛЯ ПРОСТРАНСТВА МЕТЕОСТАНЦИЙ!  
  
    ВНИМАНИЕ! Данная функция привязана к структуре данных в df_stations (и производных от него) и dict_df_parameters!  
    Предполагается, что перечень метеостанций в df_stations идёт в той же последовательности, что и в row_!  
  
    ПРИМЕЧАНИЕ: Параметры вариограммы зафиксированы (estimator='matheron', model='spherical', dist_func='euclidean',  
    bin_func='ward', maxlag=0.99999, n_lags=4, normalize=False, use_nugget=False, samples=len(vals_v), fit_method='trf',)  
  
    Принимает:  
    - row_: серию значений из которых необходимо произвести пространственную экстраполяцию;  
    - df_coords_ (по умолчанию df_coords_full) датафрейм с координатами метеостанций, сообразно индексу row_.  
    Возвращает:  
    - массив предиктов для всех метеостанций в row_  
    ЕСЛИ В ПРОЦЕССЕ ОПРЕДЕЛЕНИЯ ВАРИОГРАММЫ ВОЗНИКАЮТ ОШИБКИ  
    (КАК ПРАВИЛО ИЗ-ЗА МАЛОГО КОЛИЧЕСТВА ЗНАЧЕНИЙ В ПОЛЕ МЕТЕОСТАНЦИЙ), ТО ФУНКЦИЯ ПРЕРЫВАЕТСЯ БЕЗ ВОЗВРАЩЕНИЯ ЗНАЧЕНИЙ  
    """  
  
    # Для построения вариограммы нужны только точки, для которых есть значения:  
    # - получаем массив координат  
    # - получаем массив значений  
    # Берём координаты только соответствующие ряду значений и без NaN:  
    coords_v_ = np.array(df_coords_full)[:len(row_)][~np.isnan(row_)]  
    vals_v_ = np.array(row_.dropna())  
  
    try:  
        # Определяем вариограмму  
        V_ = skg.Variogram(coordinates=coords_v_,
```

```
        values=vals_v_,
        estimator='matheron',
        model='spherical',
        dist_func='euclidean',
        bin_func='ward',
        maxlag=0.99999, # Используем всю матрицу расстояний
        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
        normalize=False,
        use_nugget=False,
        samples=len(vals_v),
        fit_method='trf'
    )

except ValueError: # Уберём количество сэмплов из определения вариограммы (когда количество сэмплов слишком велико)
###
##    print('\nВозникла ошибка ValueError, строим вариограмму без учета количества сэмплов.')
###
try:
    V_ = skg.Variogram(coordinates=coords_v_,
                         values=vals_v_,
                         estimator='matheron',
                         model='spherical',
                         dist_func='euclidean',
                         bin_func='ward',
                         maxlag=0.99999, # Используем всю матрицу расстояний
                         n_lags=4,
                         normalize=False,
                         use_nugget=False,
                         fit_method='trf'
    );
except:
###
##    print('\nНовая ошибка вариограммы. Выход из функции.')
###
    return
except:
###
##    print('\nИная ошибка вариограммы. Выход из функции.')
###
    return

# Определяем модель кrigинга
model_ok_ = skg.OldinaryKriging(V_, min_points=1, max_points=14, mode='exact');
```

```

# координаты точки предикта:
predict_coords_ = np.array(df_coords_full)[:len(row_)] # здесь берём все координаты, кроме Reference_point

sys.stdout = NullIO() # определим вывод print() в класс нулевого вывода

# Получаем массив предиктов для всех точек
arr_predict_ = model_ok_.transform(predict_coords_[:,0], predict_coords_[:,1]);

sys.stdout = real_stdout # восстановим стандартный вывод

return arr_predict_ # Возвращаем массив предиктов для всех точек

```

## 2.3. Функция рассчёта пределов n сигм относительно средней (как опция: взвешенной по степени обратных расстояний)

In [18]:

```

# функция вычисления пределов n сигм для ряда значений row_ в DF
def sigma_n_limits(row_, n_sigma_ = 3, IDW_ = False, param_ = None, station_='Rfrnce_point', inclusive_= True):
    """ ТОЛЬКО ДЛЯ АРХИВА ПАРАМЕТРОВ
    Вычисляет пределы в n сигм от средней по обратным квадратам или средней арифметической для серии значений.
    Принимает:
    - row_ значения из ряда DF (default = None),
    - n_sigma_ количество n на которое умножается среднеквадратическое отклонение для вычисления доверительного интервала
    (default=3),
    - IDW_ (boolean) использовать ли усреднение по обратным квадратам расстояний (default=False),
    - param_ название параметра для вычисления IDW (default=None). Обязателен для IDW_=True, а так же
        при установки sigma_inclusive_ = False, в противном случае вернёт ошибку,
    - station_ - название метеостанции, для которой вычисляются пределы (значение для неё исключается из подсчёта
        средней и сигм (default='Rfrnce_point' - геометрический центр поля метеостанций),
    - inclusive_ (при IDW_=True имеет смысл только при указании station_, отличной от 'Rfrnce_point') -
        включать ли значение для данной метеостанции в расчёт средней,
        при station_='Rfrnce_point' значение для данной метеостанции будет включено
        в подсчёт средней независимо от значения inclusive_ (boolean, deafault=True).
    Возвращает:
    - кортеж - нижний и верхний порог доверительного интервала.
    """
    # Во избежание разночтений установим в True включение всех значений в подсчёт средней и сигмы
    if station_ == 'Rfrnce_point':
        inclusive_ = True
        # Rfrnce_point не входит в число метеостанций, а это значит, что при его указании в расчёте обеих величин
        # должны участвовать все реальные метеостанции. Так как для Rfrnce_point значения не определены,

```

```

# при расчётах средней и сигмы NaN будет выброшен, и сам Rfrnce_point в подсчёт не войдёт,
# но войдут все без исключения метеостанции, для которых значение не равно NaN.

# Только если индекс row_ не является datetime64
if row_.index.inferred_type != "datetime64":
    # удаляем из серии значений row_ все значения не являющиеся значениями параметров (символ # в названии индекса -
    # признак того, что этому индексу соответствует значение параметра)
    counter_ = 0 # счётчик для индекса
    for item_ in (row_.index.tolist()): # название индекса по индексу серии row_
        if '#' not in item_: # если в названии индекса нет символа '#'
            row_ = row_.drop(row_.index[counter_]) # удаляем из серии элемент (не являющийся значением параметра)
        else:
            counter_ += 1 # в противном случае увеличиваем счётчик на 1

    # Ниже перебираем все возможные сочетания булевых значений:
    # использовать IDV, включить в подсчёт средней данную метеостанцию
    # и включить в подсчёт сигмы данную метеостанцию
    if IDW_ and inclusive_ : # Если IDW_=True, использовать данную станцию для подсчёта
        # средней и сигмы
        # Находим среднюю по обратным квадратам расстояния относительно центра поля метеостанций
        # при station_='Rfrnce_point'
        mean_value_ = inverse_distance_avg(row_, param_, station_ ='Rfrnce_point', df_dists_ = df_station_dists, power_=2)
        # сигма по серии, игнорируя nan, степень свободы=1 (где количество нeNaN больше 1)
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan
    elif IDW_ and not inclusive_:
        # Находим среднюю по обратным квадратам расстояния относительно данной метеостанций
        mean_value_ = inverse_distance_avg(row_, param_, station_ , df_dists_ = df_station_dists, power_=2)
        row_ = row_.drop(labels=param_ + '#' + station_) # удаляем значение данной метеостанции из ряда
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan
    elif not IDW_ and inclusive_:
        mean_value_ = np.nanmean(row_) if len(row_.dropna()) > 0 else np.nan # средняя по серии, игнорируя nan
        # сигма по серии, игнорируя nan, степень свободы=1 (где количество нeNaN больше 1)
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan
    elif not IDW_ and not inclusive_:
        row_ = row_.drop(labels=param_ + '#' + station_) # удаляем значение данной метеостанции из ряда
        mean_value_ = np.nanmean(row_) if len(row_.dropna()) > 0 else np.nan # средняя по серии, игнорируя nan
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan

    upper_level_ = mean_value_ + n_sigma_* std_value_ # верхний порог п сигм
    lower_level_ = mean_value_ - n_sigma_* std_value_ # нижний порог п сигм

return (lower_level_ , upper_level_ )

```

## 2.4. Функция вычисления вероятностного интервала нормального распределения относительно средней (как опция: взвешенной по степени обратных расстояний)

```
In [19]: # функция оценки границ вероятности для нормального распределения относительно средней, или средней,
# взвешенной по обратным квадратам расстояния с учетом среднеквадратического отклонения для данного ряда значений.
def norm_probability_limits(
    row_, confidence_=0.95, IDW_=False, param_=None, station_='Rfrnce_point', inclusive_=True):
    """ ТОЛЬКО ДЛЯ АРХИВА ПАРАМЕТРОВ
    Вычисляет пределы в n сигм от средней по обратным квадратам или средней арифметической для серии значений.
    Принимает:
        - row_ - значения из ряда DF (default = None),
        - confidence_ - значение вероятности для оценки границ (float в диапазоне от 0 до 1) (default=0.95),
        - IDW_ (boolean) использовать ли усреднение по обратным квадратам расстояний (default=False),
        - param_ - название параметра для вычисления IDW (default=None). Обязателен для IDW_=True, а так же
            при установки sigma_inclusive_ = False, в противном случае вернёт ошибку,
        - station_ - название метеостанции, для которой вычисляются пределы (значение для неё исключается из подсчёта
            средней и сигм (default='Rfrnce_point' - геометрический центр поля метеостанций),
        - inclusive_ (при IDW_=True имеет смысл только при указании station_, отличной от 'Rfrnce_point') -
            включать ли значение для данной метеостанции в расчёт средней,
            при station_=='Rfrnce_point' значение для данной метеостанции будет включено
            в подсчёт средней независимо от значения inclusive_ (boolean, deafault=True).
    Возвращает:
        - кортеж - нижний и верхний порог доверительного интервала.
    """
    # Во избежание разночтений установим в True включение всех значений в подсчёт средней и сигмы
    if station_ == 'Rfrnce_point':
        # Rfrnce_point не входит в число метеостанций, а это значит, что при его указании в расчёте обеих величин
        # должны участвовать все реальные метеостанции. Так как для Rfrnce_point значения не определены,
        # при расчётах средней и сигма NaN будет выброшен, и сам Rfrnce_point в подсчёте не войдёт,
        # но войдут все без исключения метеостанции, для которых значение не равно NaN.
        inclusive_ = True

    # Только если индекс row_ не является datetime64
    if row_.index.inferred_type != "datetime64":
        # удаляем из серии значений row_ все значения не являющиеся значениями параметров
        # (символ # в названии индекса - признак того, что этому индексу соответствует значение параметра)
        counter_ = 0 # счётчик для индекса
        for item_ in (row_.index.tolist()): # название индекса по индексу серии row_
            if '#' not in item_: # если в названии индекса нет символа '#'
```

```

        row_ = row_.drop(row_.index[counter_]) # удаляем из серии элемент (не являющийся значением параметра)
    else:
        counter_ += 1 # в противном случае увеличиваем счётчик на 1

    # Ниже перебираем все возможные сочетания булевых значений:
    # использовать IDV, включить в подсчёт средней данную метеостанцию
    # и включить в подсчёт сигмы данную метеостанцию
    if IDW_ and inclusive_ : # Если IDW_=True, использовать данную станцию для подсчёта
        # средней и сигмы
        # Находим среднюю по обратным квадратам расстояния относительно центра поля метеостанций
        # при station_='Rfrnce_point'
        mean_value_ = inverse_distance_avg(row_, param_, station_='Rfrnce_point', df_dists_= df_station_dists, power_=2)
        # сигма по серии, игнорируя nan, степень свободы=1 (где количество нeNaN больше 1)
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan
    elif IDW_ and not inclusive_:
        row_ = row_.drop(labels=param_ + '#' + station_) # удаляем значение данной метеостанции из ряда
        # Находим среднюю по обратным квадратам расстояния относительно данной метеостанции
        mean_value_ = inverse_distance_avg(row_, param_, station_, df_dists_= df_station_dists, power_=2)
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan
    elif not IDW_ and inclusive_:
        mean_value_ = np.nanmean(row_) if len(row_.dropna()) > 0 else np.nan # средняя по серии, игнорируя nan
        # сигма по серии, игнорируя nan, степень свободы=1 (где количество нeNaN больше 1)
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan
    elif not IDW_ and not inclusive_:
        row_ = row_.drop(labels=param_ + '#' + station_) # удаляем значение данной метеостанции из ряда
        mean_value_ = np.nanmean(row_) if len(row_.dropna()) > 0 else np.nan # средняя по серии, игнорируя nan
        std_value_ = np.nanstd(row_, ddof=1) if len(row_.dropna()) > 1 else np.nan

    # вычисляем границы вероятности распределения confidence для нормального распределения
    # относительно mean_value_ с scale равном std_value_, если std_value_ равно NaN или 0 - возвращаем NaN
    result_ = norm.interval(confidence=confidence_,
                           loc=mean_value_,
                           scale=std_value_
                           ) if (pd.notna(std_value_) and (std_value_ != 0)) else np.nan

return result_

```

## 2.5. Функция оценки индивидуального отклонения признака в отношении к средней

```
In [20]: def individ_variance(row_, value_, mean_):
    """Вычисляет отношение абсолютного отклонения частной величины от средней к этой средней
ПРИНИМАЕТ:
- индивидуальное значение,
- среднюю.
ВОЗВРАЩАЕТ:
- частное от индивидуального абсолютного отклонения и средней"""

    return (abs(value_ - mean_)) / mean_ if mean_ != 0 else (abs(value_ + 1 - mean_)) / (mean_+1*len(row_))
# Если знаменатель - mean_ окажется равным нулю, сдвинем значение на 1 и изменим среднюю
# (если к каждому значению прибавить k, то средняя увеличится на k*n)
```

## 2.6. Функция вычисления выбросов в поле метеостанций вне пределов доверительного интервала, определённого либо в количестве сигм, либо в диапазоне границ вероятности нормального распределения

```
In [21]: # функция для вычисления выбросов вне пределов доверительного интервала, определённого либо в количестве сигм,
# либо в диапазоне границ вероятности нормального распределения
def field_outliers(
    row_, method_='sigma', criterium_=3, IDW_=False, param_=None, station_='Rfrnce_point', inclusive_=True):
    """ ТОЛЬКО ДЛЯ АРХИВА ПАРАМЕТРОВ
    Вычисляет выбросы в поле метеостанций и выводит их характеристики
    Принимает:
    - row_ - значения из ряда DF (default = None),
    - method_{'sigma' | 'norm'} - какую функцию использовать для оценки выбросов:
        среднеквадратическое отклонение или границы вероятности нормального распределения (default='sigma'),
    - criterium_ - числовое значение передаваемое функции оценки границ распределения, зависит от параметра 'method':
        для 'sigma' - множитель для среднеквартатического отклонения, для 'norm' - вероятностные границы (от 0 до 1),
    - IDW_ (boolean) использовать ли усреднение по обратным квадратам расстояний (default=False),
    - param_ - название параметра для вычисления IDW (default=None). Обязателен для IDW_=True, а так же
        при установки sigma_inclusive_ = False, в противном случае вернёт ошибку,
    - station_ - название метеостанции, для которой вычисляются пределы (значение для неё исключается из подсчёта
        средней и сигм
        (default='Rfrnce_point' - геометрический центр поля метеостанций),
    - inclusive_ (при IDW_=True имеет смысл только при указании station_, отличной от 'Rfrnce_point') -
        включать ли значение для данной метеостанции в расчёт средней,
        при station_='Rfrnce_point' значение для данной метеостанции будет включено
        в подсчёт средней независимо от значения inclusive_ (boolean, default=True).
    Возвращает:
```

- Если выбросы обнаружены - список из кортежей (на каждый выброс свой кортеж):  
значение выброса, его индекс в серии, границы  $\sigma$  или вероятности, ему соответствующие, и индивидуальная вариация (абсолютное отклонение в отношении к средней)
  - Если выбросов нет - возвращает NaN
- ВНИМАНИЕ:** Единственное значение в строке будет всегда расцениваться как выброс
- """

```

list_outliers_ = [] # Список для значений и параметров выбросов
## 
if len(row_.dropna()) == 1: # Если имеем единственное значение в строке, сразу вывести его атрибуты, как выброса
    val_ = row_.dropna().values[0] # Получаем единственное значение, отбросив все NaN
    # Вычисляем индекс этого значения
    idx_ = row_.tolist().index(val_)
    # добавить выброс и его параметры в список:
    list_outliers_.append((val_, idx_, val_, val_))
return list_outliers_ # Вывести список
## 

# В зависимости от 'method_' определим границы доверительного интервала вызовом соответствующей функции
if method_ == 'sigma':
    limits_ = sigma_n_limits(row_=row_, n_sigma_ = criterium_, IDW_ = IDW_, param_ = param_, station_=station_,
                             inclusive_=inclusive_)
elif method_ == 'norm':
    limits_ = norm_probability_limits(row_=row_, confidence_=criterium_, IDW_ = IDW_,
                                       param_ = param_, station_=station_, inclusive_=inclusive_)

list_outliers_ =[] # Список для значений и параметров выбросов
for i_, val_ in enumerate(row_): # по порядковому номеру и значению в полученной строке DF
    # Если limits_!=NaN и значение вне границ доверительного интервала
    if pd.notna(limits_) and (val_ < limits_[0] or val_ > limits_[1]):
        # порядковый номер значения в row_, приведённом к списку
        idx_ = i_
        # добавить выброс и его параметры в список:
        list_outliers_.append((val_, idx_, limits_[0], limits_[1]))

if len(list_outliers_) > 0: # Если выбросы есть (длина списка больше 0)
    return list_outliers_ # Вывести список
else:
    return np.nan # Иначе вывести NaN

```

## 2.7. Функция вычисления выбросов во временном ряду по каждой метеостанции вне пределов доверительного интервала, определённого либо в количестве сигм, либо в диапазоне границ вероятности нормального распределения

In [22]:

```
def time_outliers(df_, row_, td_symbol_='D', td_quant_='5', equal_hours_=False,
                  method_='sigma', criterium_=3, inclusive_=True):
    """
    Вычисляет выбросы во временных рядах по каждой метеостанции в поле метеостанций и выводит их характеристики
    ПРИНИМАЕТ:
        - df_ - датафрейм, в котором ищутся выбросы,
        - row_ строка со значением параметра по метеостанциям на определённый момент наблюдения,
        - td_symbol_ - строковое значение периода для передачи timedelta (default='D' - day),
        - td_quant_ - строковое значение количества периодов для передачи timedelta (default='5')
        - equal_hours_ - boolean использовать ли для вычислений наблюдения на один и тот же час (default=False)
        - method_ {'sigma'|'norm'} - использовать для оценки выбросов количество сигм или вероятностные границы нормального
            распределения (default='sigma')
        - criterium (float) значения для критерия определения выброса: для метода 'sigma' - количество сигм,
            для метода 'norm' - значение вероятности (от 0 до 1) (default - для 'sigma' - =3)?
        - inclusive - boolean использовать ли текущее значение параметра для подсчёта средней и сигмы (deafult=True).

    ВОЗВРАЩАЕТ:
        - Если выбросы обнаружены - список из кортежей (на каждый выброс свой кортеж):
            значение выброса, его индекс в серии, границы n сигм или вероятности, ему соответствующие.
        - Если выбросов нет - возвращает NaN

    ВАЖНО: при методе 'norm' значения для timedelta должны включать в себя более 3 моментов наблюдения
    ВНИМАНИЕ: Единственное значение в строке будет всегда расцениваться как выброс
    """

    td_string_ = f'{td_quant_}{td_symbol_}' # Стока для определения периода timedelta
    start_time_ = row_.name - pd.Timedelta(td_string_) # Время начала периода выборки для вычисления средней и сигмы
    end_time_ = row_.name + pd.Timedelta(td_string_) # Время окончания периода выборки для вычисления средней и сигмы
    obsrvtn_hour_ = row_.name.hour # Час наблюдения

    # Создаём маску для выбора значений из df_,
    mask_ = (df_.index >= start_time_) & (df_.index <= end_time_) # временной интервал для выборки
    if not inclusive_: # если данное значение параметра не включать в подсчёт средний и сигмы:
        mask_ = mask_ & (df_.index != row_.name)
    if equal_hours_: # если используем фиксированный час наблюдений
        mask_ = mask_ & (df_.index.hour == obsrvtn_hour_)
```

```

list_outliers_ =[] # Список для значений и параметров выбросов

# Проходим по ряду значений параметра между метеостанциями на данный момент времени:
for label_ in row_.index: # Проходим по ряду значений для метеостанций
    ser_ = df_.loc[mask_][label_] # для каждой станции берём серию значений в соответствии с временным интервалом
    checked_value_ = row_[label_] # значение, проверяющееся на выброс - текущее значение, определённое row_

    # В зависимости от 'method_' определим границы доверительного интервала вызовом соответствующей функции
    if method_ == 'sigma':
        limits_ = sigma_n_limits(row_=ser_, n_sigma=criterium_, IDW=False, param=None, station=None,
                                  inclusive=True) # мы уже сделали все операции, связанные с inclusive_ - нечего исключать
    elif method_ == 'norm':
        limits_ = norm_probability_limits(row_=ser_, confidence=criterium_,
                                           IDW=False, param=None, station=None,
                                           inclusive=True) # мы уже сделали все операции, связанные с inclusive_ - нечего исключать

    # Если limits_ !=NaN и значение вне границ доверительного интервала, или границы неопределены
    # (т.е. единственное значение в серии)
    if (
        pd.notna(limits_) and (
            (checked_value_ < limits_[0] or checked_value_ > limits_[1]) or
            (pd.isna(limits_[0]) or pd.isna(limits_[1])))
        )
    ):
        # порядковый номер столбца для значения с label
        idx_ = df_.columns.get_loc(label_)
        # добавить выброс и его параметры в список:
        list_outliers_.append((checked_value_, idx_, limits_[0], limits_[1]))
    elif pd.isna(limits_): # norm_probability_limits вернула NaN - может единственное значение в серии
        idx_ = df_.columns.get_loc(label_)
        # добавить выброс и его параметры в список:
        list_outliers_.append((checked_value_, idx_, np.nan, np.nan))

    if len(list_outliers_) > 0: # Если выбросы есть (длина списка больше 0)
        return list_outliers_ # Вывести список значений и параметров выбросов
    else:
        return np.nan # Иначе вывести NaN

```

## 2.8. Функция поиска референсного значения для параметра для сравнения с расчётным значением

```
In [23]: def reference_param_extractor(station_, param_, dict_archive_, idx_station_):
    """
    Функция поиска референсного значения параметра в словаре DFs архивов метеостанций на основе подобных индексов DFs
    Принимает:
    - название станции;
    - параметр, для которого надо найти значение;
    - название словаря DF, где надо искать значение;
    - индекс DateTime переданного функции текущего значения station_ в текущем DF.
    Выводит:
    - значение искомого параметра.
    ПРЕДПОЛАГАЕТСЯ, что поиск значений производится по индексу в обоих сопоставляемых DFs
    """
    name_df_ = f'df_{station_}' # преобразуем название станции в название ключа её архива в словаре архивов
    df_arch_ = dict_archive_[name_df_] # DF архива по ключу в словаре архивов
    # значение параметра в DF архива, где индекс совпадает с индексом station_:
    value_ = df_arch_.at[idx_station_, param_]
    # print(idx_station_, value_)
    return value_
```

## 2.9. ФУНКЦИЯ ВЫВОДА СПИСКА НАЗВАНИЯ СТАНЦИЙ ИЗ СПИСКА КОРТЕЖЕЙ ВЫБРОСОВ

```
In [24]: def stations_from_outliers(row_, column_name_: str, param_: str):
    """
    Функция вывода списка названия станций из списка кортежей выбросов
    Принимает:
    - серию (строку) из датафрейма
    - название столбца, содержащего списки кортежей с параметрами выбросов
    - название параметра по которому произошёл выброс - без 'df_'
    Возвращает:
    - список названий метеостанций к которым относятся выбросы значений
    ПРЕДПОЛАГАЕТСЯ, что столбец column_name_ в качестве значений содержит списки кортежей
    """
    # По списку кортежей в row row_[column_name_] получаем значение индекса столбца метеостанции и
    # вычленяем из названия этого столбца называние станции - название столбца после символа #.
    list_stations_ = [row_.index[i[1]][len(param_)+1 :] for i in row_[column_name_]]
    return list_stations_
```

## 2.10. ФУНКЦИЯ СОЗДАНИЯ ЛОГИЧЕСКИХ МАСОК ДЛЯ РАЗБИЕНИЯ АРХИВНОГО ДАТАФРЕЙМА НА КЛИМАТИЧЕСКИЕ СЕЗОНЫ

```
In [25]: def season_masks(df_:pd.DataFrame):
    """
    Функция создания логических временных масок для разбиения датафрейма на климатические сезоны.
    Принимает:
    - Датафрейм с индексом TimeStamp
    Возвращает
    - Словарь с масками, где ключами являются названия сезонов (spring, summer, autumn, winter)
    """
    # Определим логические маски для климатических сезонов
    mask_winter_ = (
        (df_.index.month == 11) & (df_.index.day >= 5)
    ) | (
        (df_.index.month > 11) | (df_.index.month < 4)
    ) | (
        (df_.index.month == 4) & (df_.index.day <= 4)
    )

    mask_spring_ = (
        (df_.index.month == 4) & (df_.index.day >= 5)
    ) | (
        (df_.index.month == 5) & (df_.index.day <= 18)
    )

    mask_summer_ = (
        (df_.index.month == 5) & (df_.index.day >= 19)
    ) | (
        (df_.index.month > 5) & (df_.index.month < 9)
    ) | (
        (df_.index.month == 9) & (df_.index.day <= 14)
    )

    mask_autumn_ = (
        (df_.index.month == 9) & (df_.index.day >= 15)
    ) | (
        (df_.index.month == 10)
    ) | (
        (df_.index.month == 11) & (df_.index.day <= 4)
    )
    dict_season_masks_ = {'spring': mask_spring_, 'summer': mask_summer_, 'autumn': mask_autumn_, 'winter': mask_winter_}
    return dict_season_masks_
```

## 2.11. Функция замены некорректных значений в архивах *метеостанций* на корректные

```
In [26]: # Функция замены значений в архивах МЕТЕОСТАНЦИЙ
def correct_errors_stations(x_corr_, station_, param_, dict_archive_, idx_x_):
    """
    Заменяет значения в архивах на корректные: только архивы по метеостанциям!
    Принимает:
        корректное значение параметра;
        название станции;
        параметр, для которого надо заменить значение в архиве;
        название словаря DF архивов, где надо заменить значение;
        индекс переданного функции текущего x_ в df_x_.

    Выводит:
        - (Строку с подтверждением замены значений.)
    Возвращает:
        ПРЕДПОЛАГАЕТСЯ, что поиск значений производится по индексу в обоих сопоставляемых DFs

    ВНИМАНИЕ! ПОЛЬЗОВАТЬСЯ С ОСТОРОЖНОСТЬЮ, ЧТОБЫ НЕ ПОВРЕДИТЬ АРХИВЫ!
    """
    name_df_ = f'df_{station_}' # преобразуем название станции в название ключа её архива в словаре архивов
    df_arch_ = dict_archive_[name_df_] # DF архива по ключу в словаре архивов
    # записываем значение параметра в DF архив, где индекс совпадает с индексом x_:
    x_faulty_ = df_arch_.at[idx_x_, param_] # значение, подлежащее замене
    df_arch_.at[idx_x_, param_] = x_corr_
###
#     print(f'Параметр {param_} в архиве {name_df_} на момент наблюдения {idx_x_} установлен в значение {x_corr_}, '
#           f'(прежнее значение {x_faulty_})')
#     return None
```

## 2.12. Функция замены некорректных значений в архивах *параметров* на корректные

```
In [27]: # Функция замены значений в архивах ПАРАМЕТРОВ
def correct_errors_parameters(x_corr_, station_, param_, dict_archive_=dict_df_parameters, idx_x_=""):
    """
    Заменяет значения в архивах на корректные: только архивы по параметрам!
    Принимает:
```

```

корректное значение параметра;
название станции;
параметр, для которого надо заменить значение в архиве;
название словаря DF архивов, где надо заменить значение;
индекс переданного функции текущего x_ в df_x_.

Выводит: -
(Строку с подтверждением замены значения.

Возвращает: -
ПРЕДПОЛАГАЕТСЯ, что поиск значений производится по индексу в обоих сопоставляемых DFs

ВНИМАНИЕ! ПОЛЬЗОВАТЬСЯ С ОСТОРОЖНОСТЬЮ, ЧТОБЫ НЕ ПОВРЕДИТЬ АРХИВЫ!
"""
name_df_ = f'df_{param_}' # преобразуем название параметра в название ключа её архива в словаре архивов
df_arch_ = dict_archive_[name_df_] # DF архива по ключу в словаре архивов
col_name = f'{param_}#{station_}' # Формируем название столбца
# записываем значение параметра в DF архив, где индекс совпадает с индексом x_:
x_faulty_ = df_arch_.at[idx_x_, col_name] # значение, подлежащее замене
df_arch_.at[idx_x_, col_name] = x_corr_

## 
# print(f'Параметр {col_name} в архиве {name_df_} на момент наблюдения {idx_x_} установлен в значение {x_corr_}, '
#       f'(прежнее значение {x_faulty_})')
# return None

```

## 2.13. Функция замены NaN на среднюю величину, взвешенную по обратным степеням расстояний между метеостанциями

In [28]:

```

def row_nan_idw_correct (row_, name_param_, power_):
    """
    Функция замены NaN на средневзвешенные по обратным квадратам расстояний idw = inverted distance weight
    Для каждого NaN в строке row_ :
        - вычисляет необходимые для inverse_distance_avg параметры;
        - вызывает inverse_distance_avg;
        - полученный результат использует сразу для:
            - замены NaN в текущем ряду (для последующих вычислений),
            - замены NaN в архивах метеостанций,
            - замены NaN в архивах параметров.

    Принимает:
        - ряд значений,
        - название параметра,
        - степень для весов для функции inverse_distance_avg

```

```

Возвращает: -
"""
list_columns_ = row_.isna()[lambda y: y].index.tolist() # Список столбцов в которых есть NaN в данном ряду
if len(list_columns_) == 0: # Если NaNов нет
    return # Выходим из функции

for col_name_ in list_columns_:
    station_ = col_name_[len(name_param_) + 1 :] # Убираем слева в названии столбца признак параметра и '#'
    # для данной станции вычисляем средневзвешенную по обратным квадратам из строки значений
    result_ = inverse_distance_avg(row_ = row_,
                                    param_ = name_param_,
                                    station_ = station_,
                                    df_dists_ = df_station_dists,
                                    power_=power_
                                    )

    # Заменяем NaN в текущем ряду значений
    row_.loc[col_name_] = result_

    # Заменяем значения в архивах метеостанций на корректные.
    correct_errors_stations(x_corr_ = result_,
                            station_ = station_,
                            param_ = name_param_,
                            dict_archive_ = dict_df_locations,
                            idx_x_ = row_.name)

    # Заменяем значения в архивах параметров на корректные.
    correct_errors_parameters(x_corr_ = result_,
                            station_ = station_,
                            param_ = name_param_,
                            dict_archive_ = dict_df_parameters,
                            idx_x_ = row_.name)

#     return None

```

## 2.14. Функция замены NaN на значение, полученное пространственной экстраполяцией методом кригинга

In [29]: `def row_nan_kriging_correct (row_, name_param_):`

ВНИМАНИЕ! Данная функция привязана к структуре данных в `df_stations` (и производных от него) и `dict_df_parameters`!

Предполагается, что перечень метеостанций в `df_stations` идёт в той же последовательности, что и в `row_`!

Функция замены `NaN` на значение, полученное пространственной экстраполяцией методом кrigинга

Для каждого `NaN` в строке `row_`:

- вычисляет индексы пропущенных значений;
- определяет вариограмму;
- вызывает модель `ordinary.kriging`;
- полученный результат использует сразу для:
  - замены `NaN` в текущем ряду (для последующих вычислений),
  - замены `NaN` в архивах метеостанций,
  - замены `NaN` в архивах параметров.

ЕСЛИ В РЯДУ ВСЕ ЗНАЧЕНИЯ ОПРЕДЕЛЕНЫ, ИЛИ В ПРОЦЕССЕ В ПРОЦЕССЕ ОПРЕДЕЛЕНИЯ ВАРИОГРАММЫ ВОЗНИКАЮТ ОШИБКИ (КАК ПРАВИЛО ИЗ-ЗА МАЛОГО КОЛИЧЕСТВА ЗНАЧЕНИЙ В ПОЛЕ МЕТЕОСТАНЦИЙ), ТО ФУНКЦИЯ ПРЕРЫВАЕТСЯ

Принимает:

- ряд значений,
- название параметра для поиска значений

Возвращает:

"""

```
# Определяем индексы значений NaN в row_
arr_idx_nan_ = np.where(np.isnan(row_))[0] # np.where только с условием - выводит кортеж из массива пнтур!
# Если в массив индексов NaN пустой (есть все нужные значения) или
# если разница между длинной ряда и длинной массива индексов NaN меньше 3 (ограничение сэмплов для вариограммы)
# то выйти из функции
if (len(arr_idx_nan_) == 0) or ((len(row_) - len(arr_idx_nan_)) < 3):
    #
    #     print(f"\nВыход из функции из-за количества значений."
    #           f"Количество переданных определённых значений={len(row_) - len(arr_idx_nan_)}."
    #
    #
    return # выход из функции

# Вызовем функцию kriging_extrapolation и получаем массив предиктов для row_
arr_predict_ = kriging_extrapolation(row_=row_,
                                       df_coords_=df_coords_full)
# если массив предиктов неопределён из-за невозможности построить вариограмму (не возвращён np.ndarray)
if not isinstance(arr_predict_, np.ndarray):
    return # выход из функции

# Заменяем NaN в текущем ряду значений
row_[arr_idx_nan_] = arr_predict_[arr_idx_nan_] # присваиваем элементам row_ предикты по индексу бывших NaN-ов

for idx_ in arr_idx_nan_:
```

```
station_ = row_.index[idx_][len(name_param_) + 1 :]
# Заменяем значения в архивах метеостанций на корректные.
correct_errors_stations(x_corr_ = arr_predict_[idx_],
                         station_ = station_,
                         param_ = name_param_,
                         dict_archive_ = dict_df_locations,
                         idx_x_ = row_.name)

# Заменяем значения в архивах параметров на корректные.
correct_errors_parameters(x_corr_ = arr_predict_[idx_],
                           station_ = station_,
                           param_ = name_param_,
                           dict_archive_ = dict_df_parameters,
                           idx_x_ = row_.name)

# return None
```

### 3. Исследование и обработка индивидуальных показателей метеорологических наблюдений: Температура воздуха ( $T$ , $T_{min}$ , $T_{max}$ )

#### 3.1. Температура воздуха: $T$ (температура на высоте 2 м над поверхностью земли на момент наблюдения, градусов Цельсия ).

##### 3.1.1. Поиск и удаление ошибок показателя температуры $T$

Температура воздуха  $T$  измеряется метеостанциями в каждый момент наблюдений с интервалом в 3 часа.

Для данного раздела обозначим константу названия параметра

In [30]: `PARAMETER31 = 'T'`

Для проверки корректности значений нам может понадобится проверить согласованность значений температуры со значениями других параметров. Для этого определим функцию вычисления температуры в градусах Цельсия, исходя из давления на уровне моря, на

уровне станции и высоты станции над уровнем моря.

```
In [31]: # Функция расчёта температуры, исходя из давления на уровне моря и на поверхности земли.
def t_from_P(pHg_, p0Hg_, h_):
    """ Расчёт температуры в градусах Цельсия, исходя из давления на уровне моря, давления на уровне точки, высоты точки
    Принимает:
    - давление на уровне точки в mm Hg,
    - давление на уровне моря в mm Hg,
    - высоту в метрах
    Возвращает:
    - температуру в градусах Цельсия
    """
    p_ = pHg_ * 133.321995 # Переводим давление на уровне точки в Па
    p0_ = p0Hg_ * 133.321995 # Переводим давление на уровне моря в Па
    T_ = -(0.029 * 9.81 * h_) / (log(p_ / p0_) * 8.31)
    t_ = T_ - 273.15 # Переводим температуру в градусы Цельсия

    return t_
```

Определим также функцию вычисления температуры в градусах Цельсия, исходя из влажности и температуры точки росы (основана на приближении Августа-Роша-Магнуса).

```
In [32]: def t_from_dew_point(dew_point_=10, humid_=50):
    """
    Функция расчёта температуры по относительной влажности и значению точки росы
    Принимает:
    - температуру точки росы в градусах С,
    - значение относительной влажности воздуха в %
    Возвращает:
    - значение температуры в градусах С.
    """
    numerator_ = (17.27 * dew_point_) / (237.71 + dew_point_) - log(humid_ / 100)
    denominator_ = 17.27 + log(humid_ / 100) - 17.27 * (dew_point_ / (237.71 + dew_point_))

    temp_ = 237.71 * (numerator_ / denominator_)
    return temp_
```

## Создаём временный DF для работы с параметром температуры

```
In [33]: param_df_name = f'df_{PARAMETER31}' # преобразуем полученное значение в df_PARAMETER31 - ключ словаря dict_df_parameters
# Создадим временный df
```

```
df_tmp31 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp31.sample(5, random_state=56)
```

Out[33]:

	T#V_Volochek	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2014-02-22 03:00:00	-3.5	-2.8	-3.0	-4.3	-4.9	-4.8	-2.9	-2.4	-3.4	-3.6	-2
2015-05-16 03:00:00	8.6	8.1	10.8	8.0	8.3	10.9	7.7	7.9	8.4	8.8	8
2020-06-18 18:00:00	28.7	28.8	24.6	28.3	27.7	27.5	30.1	29.1	29.6	25.8	30
2019-12-02 21:00:00	-2.2	-3.2	-2.9	-2.2	-3.1	-3.6	-3.2	-2.8	-2.9	-2.6	-3
2008-06-05 18:00:00	NaN	18.5	NaN	19.2	17.2	16.2	17.1	17.0	16.9	NaN	16

Определим последовательность действий, для поиска ошибок в параметре "Температура".

Здесь следуем иметь в виду, что локальные температурные колебания могут быть достаточно значительными и зависят от локальных метеорологических явлений. Сам по себе выброс в поле метеостанций еще не значит, что мы имеем ошибочное значение. Очевидно, что одновременный выброс по нескольким метеостанциям во временном ряду за один и тот же период означает резкое изменение в природных явлениях, затрагивающих всё поле метеостанций. В этих случаях выбросов в поле метеостанций фиксироваться не будет, а будут наблюдаться выбросы по некоторым метеостанциям во временном ряду. Такие выбросы не являются ошибкой. Поэтому рассмотрим только те ситуации, когда выброс в поле метеостанций является одновременно выбросом во временном ряду.

1. Определяем выбросы в поле метеостанций - формируем кандидатов в ошибки
2. Отдельно проверяем, есть ли единичные значения в рядах - это тоже кандидаты в ошибки
3. Проверяем каждый из этих кандидатов во временном ряду (только эти выбросы, а не весь DF!) - не подтвердилось - отбрасываем
4. Проверяем оставшихся кандидатов на соответствие температуры давлению и точке росы: соответствуют обоим - неошибка.
5. То, что осталось и есть ошибка.

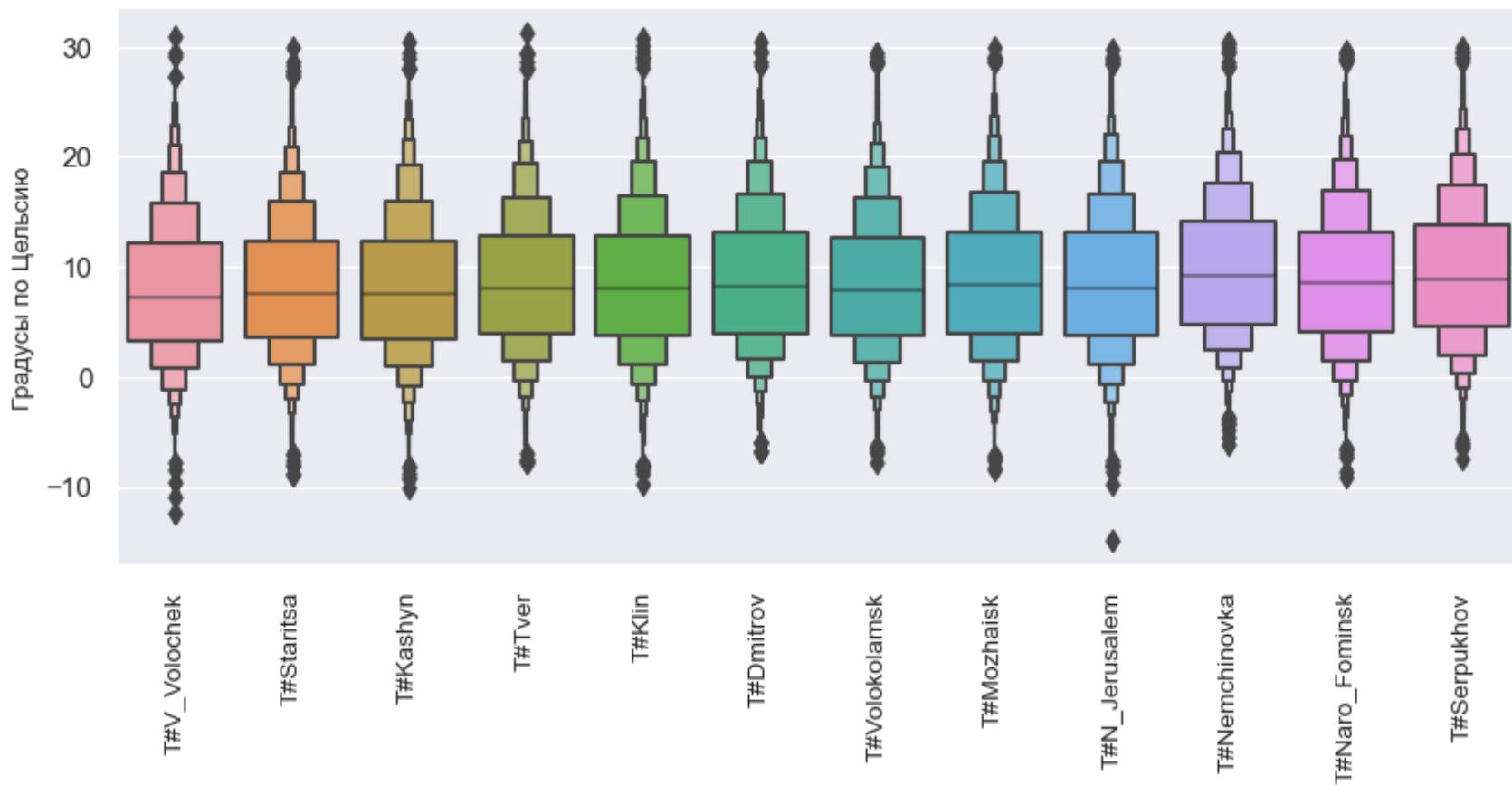
## Визуализируем архив температуры (T) по сезонам

Исходя из данных о климате Московской области, временные границы сезонов определены следующим образом.

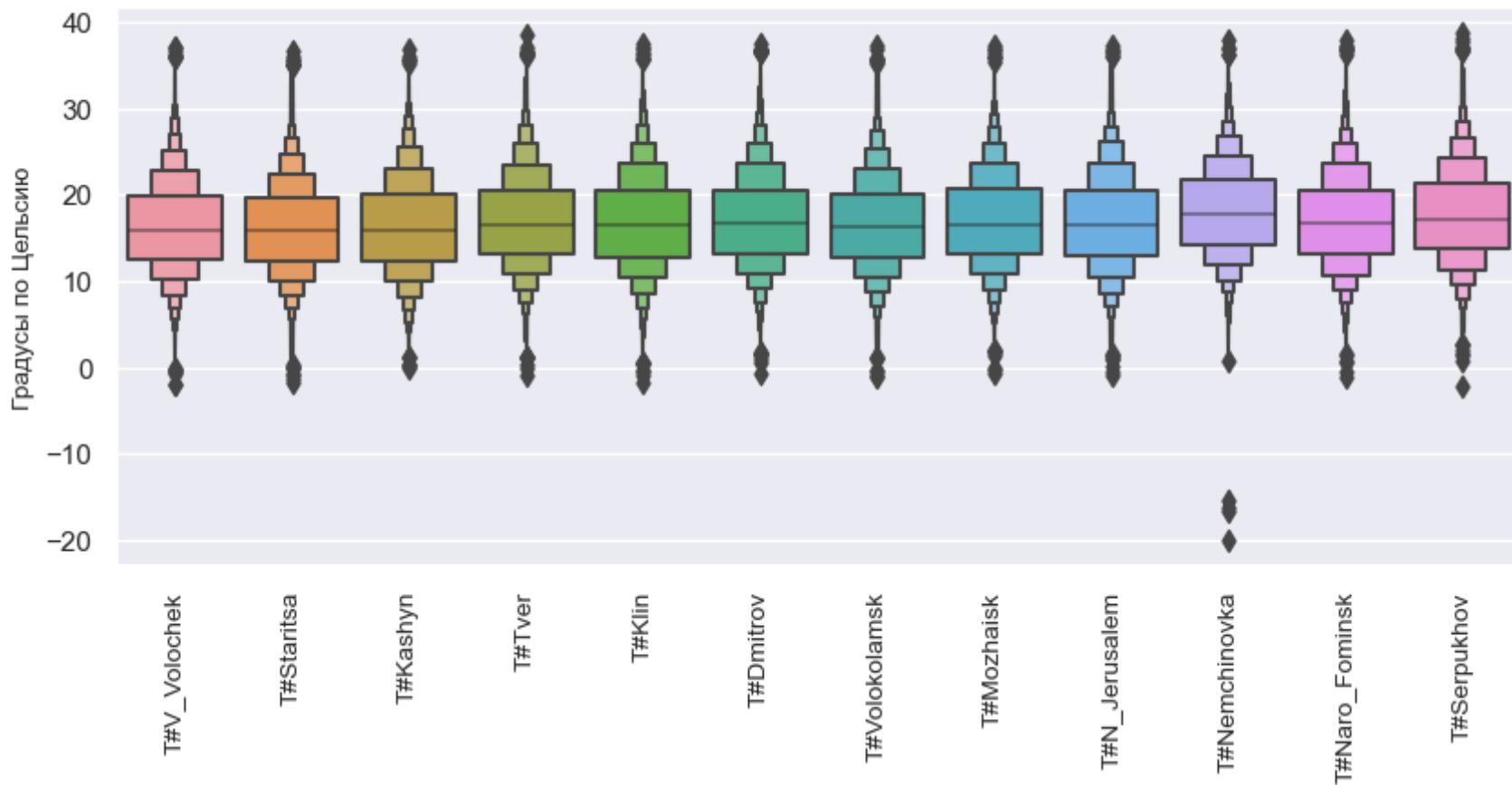
- Зима (ниже 0°): В среднем длится с 5 ноября по 4 апреля
- Весна (от 0° до +10°): В среднем длится с 5 апреля по 18 мая
- Лето (выше +10°): В среднем длится с 19 мая по 14-15 сентября
- Осень (от +10° до 0°): В среднем длится с 14-15 сентября по 4 ноября

```
In [34]: # В цикле выведем графики температур по метеостанциям в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp31).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp31[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Градусы по Цельсию', size=10)
    dummy = g.set_title(f'Распределение значений T в разрезе метеостанций:\n{season_name}')
plt.show()
```

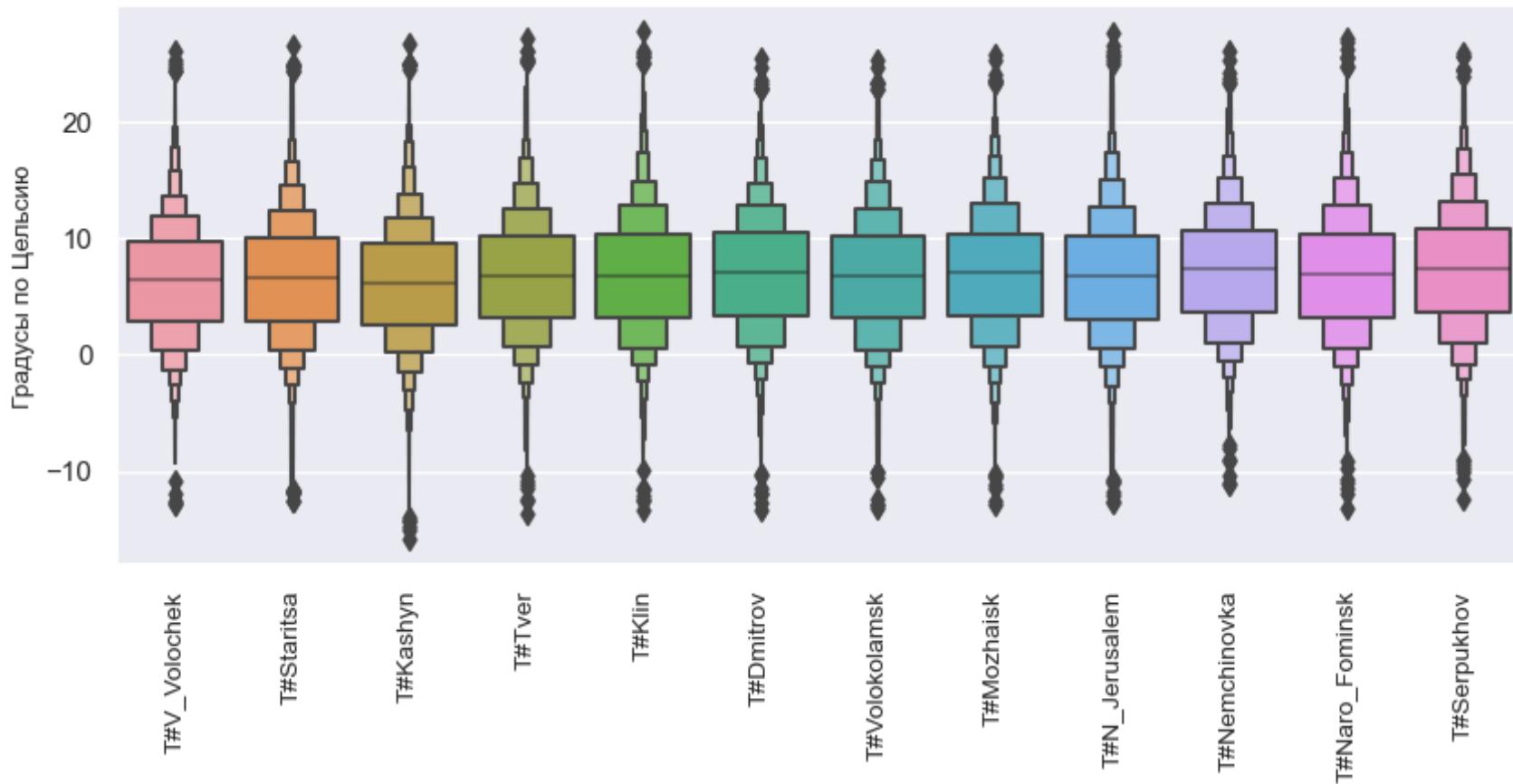
Распределение значений Т в разрезе метеостанций:  
spring



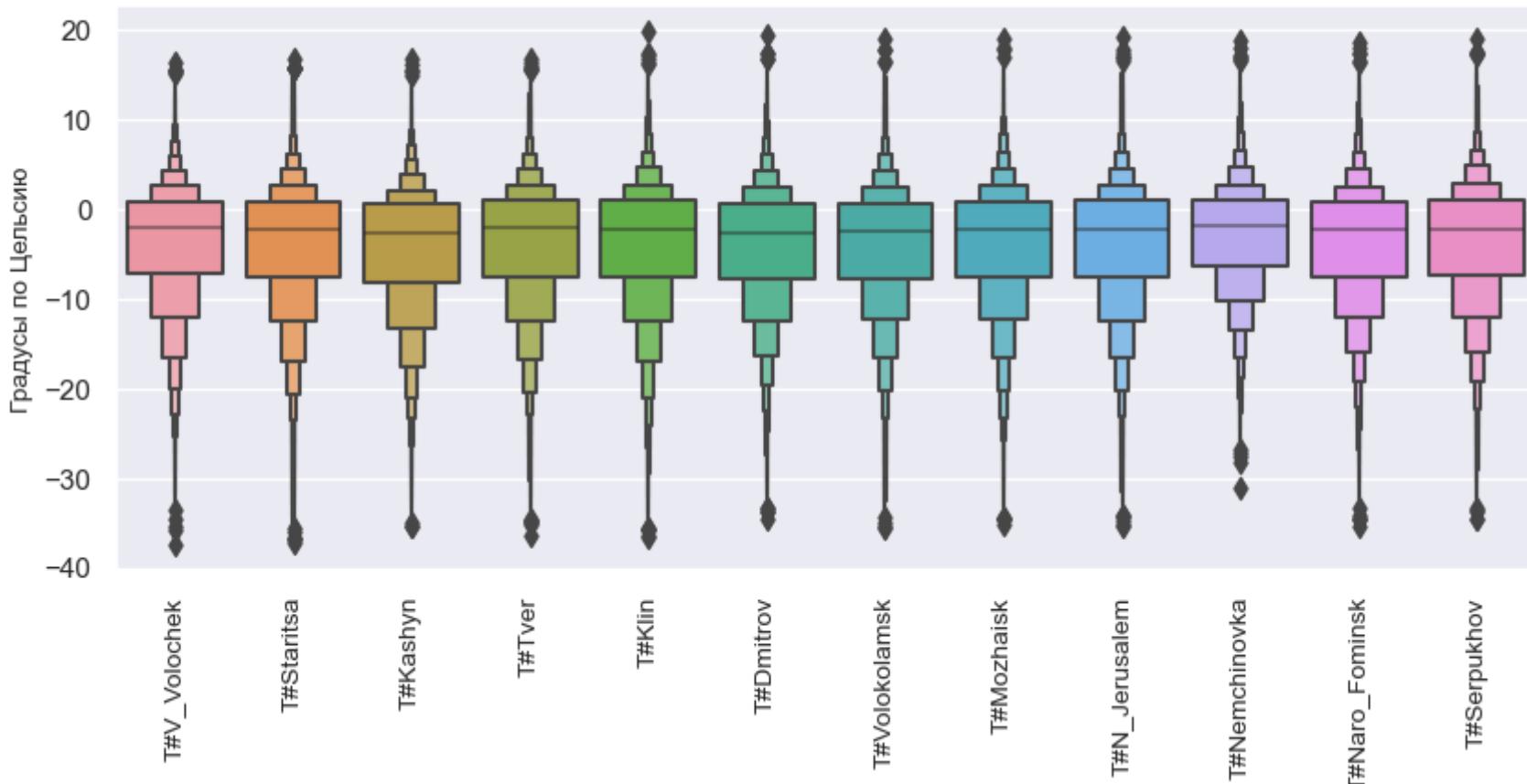
Распределение значений Т в разрезе метеостанций:  
summer



Распределение значений Т в разрезе метеостанций:  
autumn



### Распределение значений Т в разрезе метеостанций: winter



Выведем минимальное и максимальное значения, а также значение медианы и средней для всего DF.

```
In [35]: print(f'Минимальное значение: {np.nanmin(df_tmp31)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp31)},\n'
      f'Средняя: {np.nanmean(df_tmp31)},\n'
      f'Медиана: {np.nanmedian(df_tmp31)})')
```

Минимальное значение: -37.5,  
Максимальное значение: 38.7,  
Средняя: 5.818075706239234,  
Медиана: 5.5

Экстремальные выбросы на общем уровне пока не обнаруживаются

### Найдем выбросы в поле метеостанций.

Параметры:

- используем среднюю по обратным квадратам расстояния от центра поля,
- включаем проверяющее значение в подсчёт средней (автоматически, так как средняя рассчитывается от центра поля для всех станций),
- определим границы доверительного интервала в 3,0 сигмы

```
In [36]: start_time = time.time() # для замера времени выполнения кода

df_tmp31 = df_tmp31.assign(field_out=df_tmp31.apply(lambda x: field_outliers(row=x,
                                                               method='sigma',
                                                               criterium=3.0,
                                                               IDW=True,
                                                               param=PARAMETER31,
                                                               station='Rfrnce_point',
                                                               inclusive=True),
                           axis=1)
                           )

end_time = time.time()
elapsed_time = end_time - start_time
time_format = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f"На выполнение кода ушло: {time_format}")
```

На выполнение кода ушло: 00:05:13

```
In [37]: df_tmp31.sample(5, random_state=56)
```

Out[37]:

	T#V_Volochek	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2014-02-22 03:00:00	-3.5	-2.8	-3.0	-4.3	-4.9	-4.8	-2.9	-2.4	-3.4	-3.6	-2
2015-05-16 03:00:00	8.6	8.1	10.8	8.0	8.3	10.9	7.7	7.9	8.4	8.8	8
2020-06-18 18:00:00	28.7	28.8	24.6	28.3	27.7	27.5	30.1	29.1	29.6	25.8	30
2019-12-02 21:00:00	-2.2	-3.2	-2.9	-2.2	-3.1	-3.6	-3.2	-2.8	-2.9	-2.6	-3
2008-06-05 18:00:00	NaN	18.5	NaN	19.2	17.2	16.2	17.1	17.0	16.9	NaN	16

Используемые формулы определения выбросов специально обозначают как выброс в поле метеостанций случай, когда в ряду есть единственное значение. Проверим, есть ли ситуации, когда существует только одно значение параметра в поле метеостанций.

In [38]:

```
# Если значение является единственным в строке, то True, иначе False,
# убираем NaN (берём ряд без столбца field_out) и суммируем результат
single_values_count = df_tmp31.apply(lambda x : True if (len(x[:-1].dropna()) == 1) else False, axis = 1).dropna().sum()
print(f'Количество случаев единичных значений в рядах моментов наблюдений: {single_values_count}')

Количество случаев единичных значений в рядах моментов наблюдений: 9
```

Проверим максимальное количество выбросов в поле метеостанций на момент наблюдения.

In [39]:

```
# Находим максимальное количество выбросов в строке поля метеостанций
max_field_outliers= df_tmp31.field_out.dropna().apply(lambda x: len(x)).max()
print(f'Максимальное количество выбросов в поле метеостанций за весь период наблюдения не превышает '
      f'{max_field_outliers}')

Максимальное количество выбросов в поле метеостанций за весь период наблюдения не превышает 1
```

Для дальнейшей работы с ошибками создадим столбец error\_at, куда запишем кортеж из TimeStamp и названия станции.

In [40]:

```
# Определяем столбец error_at.
# Поскольку в field_out у нас всегда не более 1 выброса, мы можем не анализировать весь список из вывода функции
# stations_from_outliers, а взять из вывода-списка первый (и единственный) элемент-кортеж

df_tmp31 = df_tmp31.assign(error_at =
    df_tmp31.
        apply(lambda x: # Вычленим DateTime index и название станции с выбросом
              # пропустим NaN, проверив, является ли x.field_out списком
              (x.name,
               stations_from_outliers(
                   row_=x,
                   column_name_= 'field_out', # используем выбросы в поле метеостанций
                   param_=PARAMETER31)[0]
               ) if isinstance(x.field_out, list) else np.nan,
              axis=1
            )
        )
df_tmp31.sample(5, random_state=56)
```

Out[40]:

	T#V_Volochev	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2014-02-22 03:00:00	-3.5	-2.8	-3.0	-4.3	-4.9	-4.8	-2.9	-2.4	-3.4	-3.6	-2
2015-05-16 03:00:00	8.6	8.1	10.8	8.0	8.3	10.9	7.7	7.9	8.4	8.8	8
2020-06-18 18:00:00	28.7	28.8	24.6	28.3	27.7	27.5	30.1	29.1	29.6	25.8	30
2019-12-02 21:00:00	-2.2	-3.2	-2.9	-2.2	-3.1	-3.6	-3.2	-2.8	-2.9	-2.6	-3
2008-06-05 18:00:00	NaN	18.5	NaN	19.2	17.2	16.2	17.1	17.0	16.9	NaN	16

## Для подтверждения, являются ли отобранные значения температуры ошибками, найдём выбросы во временном ряду

Среди выбросов в поле метеостанций найдем выбросы во временном окне.

1й Вариант (для одного и того же часа наблюдения за несколько дней) Параметры:

- используем границы нормального распределения,
- не включаем проверяемое значение в подсчёт средней и сигмы,
- определим границы доверительного интервала в 98%,
- определим временное окно в +/- 3 дня ('3D'),
- включим в подсчёт только моменты наблюдения на данный час (equalhours=True).

2й Вариант (для всех моментов наблюдения за несколько дней) Параметры:

- используем границы нормального распределения,
- не включаем проверяемое значение в подсчёт средней и сигмы,
- определим границы доверительного интервала в 98%,
- определим временное окно в +/- 3 48 часов ('48H'),
- включим в подсчёт все моменты наблюдения (equalhours=False).

In [41]: `start_time = time.time() # для замера времени выполнения кода`

```
# Помним, мы не должны включать столбец field_out в поиск средних, поэтому исключаем последнюю колонку из df_ и row_
# Нельзя удалять NaN в поле field_out (Это приведёт к некорректным времененным рядам!)
# Удалим NaN в столбце "field_out" в результате
df_tmp31=(
df_tmp31.assign(
    time_out_3D=df_tmp31.
    apply(lambda x: time_outliers(df_=df_tmp31,
                                    # Возьмём из row_ только значения, соответствующие выбросам
                                    # в поле метеостанций:
                                    # пропустим NaN, проверив, является ли x.field_out списком
                                    row_=x[x.index == x.index[x.field_out[0][1]]],
                                    td_symbol_='D',
                                    td_quant_='3',
                                    equal_hours_=True,
                                    method_='norm',
                                    criterium_=0.98,
                                    inclusive_=False) if isinstance(x.field_out, list) else np.nan,
```

```
        axis=1),
time_out_48H=df_tmp31.
apply(lambda x: time_outliers(df_=df_tmp31,
                                # Возьмём из row_ только значения, соответствующие выбросам
                                # в поле метеостанций:
                                # пропустим NaN, проверив, является ли x.field_out списком
                                row_=x[x.index == x.index[x.field_out[0][1]]],
                                td_symbol_='H',
                                td_quant_='48',
                                equal_hours_=False,
                                method_='norm',
                                criterium_=0.98,
                                inclusive_=False) if isinstance(x.field_out, list) else np.nan,
                                axis=1)).
dropna(subset=["field_out"])
)

end_time = time.time()
elapsed_time = end_time - start_time
time_format = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f"На выполнение кода ушло: {time_format}")
```

На выполнение кода ушло: 00:00:15

In [42]: df\_tmp31.sample(5, random\_state=56)

Out[42]:

	T#V_Volocheok	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
<b>2019-12-25 21:00:00</b>	1.9	0.6	-0.3	-0.1	-0.4	-0.6	-0.5	-0.2	-0.8	-0.4	-0
<b>2018-01-30 21:00:00</b>	-3.8	-0.9	-1.3	-0.9	0.2	0.0	0.0	0.2	0.0	0.1	0
<b>2020-07-12 21:00:00</b>	14.0	13.6	12.7	12.6	13.9	14.5	13.7	13.6	14.9	16.9	16
<b>2014-02-26 18:00:00</b>	1.4	0.7	0.4	0.3	0.9	1.1	1.1	0.9	0.2	0.2	0
<b>2008-06-23 21:00:00</b>	17.4	17.0	17.7	17.3	15.4	17.6	17.4	17.1	16.4	10.4	16



**Для подтверждения, являются ли отобранные значения температуры ошибками, вычислим значения температуры по разнице давления на уровне моря и на уровне метеостанции, используя сокращённую формулу Лапласа.**

На всех официальных метеостанциях, посылающих синоптические телеграммы давление, наблюдаемое на уровне метеостанции, приводится к уровню моря не по отдельному измерению, а по формуле приведения - барометрической (сокращённой) Лапласа. ([https://studbooks.net/1916726/estestvoznanie/privedenie\\_davleniya\\_urovnyu\\_morya](https://studbooks.net/1916726/estestvoznanie/privedenie_davleniya_urovnyu_morya)). Поэтому, проведя вычисления температуры из разницы давлений, мы выявим те случаи, когда для приведения давления к уровню моря было использовано одно - верное - значение температуры, а в архив занесено другое - неверное. Однако, если для приведения давления к уровню моря было использовано неверное значение температуры, то и давление было вычислено ошибочно. Обратный расчёт температуры приведёт нас к тому же неверному значению, которое составило выброс. При этом не исключается случай, когда и давление было определено неверно. Поэтому эту проверку еще нельзя считать конечным критерием ошибки.

Для расчёта возьмём из архивов:

- название метеостанции с выбросом по температуре(df\_tmp31.error\_at),
- значение давления на уровне моря для этой (dict\_df\_locations) ,

- значение давления на уровне этой метеостанции (dict\_df\_locations),
- высоту метеостанции (df\_stations).

В результирующий столбец выведем расчётное значение температуры из разницы давлений. Для всех этих процедур используем определённые ранее функции.

```
In [43]: df_tmp31=(
    df_tmp31.assign(P0=df_tmp31.apply(lambda x: # Давление на уровне моря
        reference_param_extractor(
            station=x.error_at[1],
            param_='P_sea',
            dict_archive_=dict_df_locations,
            idx_station_=x.name),
        axis=1
    ),
    P1=df_tmp31.apply(lambda x: # Давление на уровне станции
        reference_param_extractor(
            station=x.error_at[1],
            param_='P_station',
            dict_archive_=dict_df_locations,
            idx_station_=x.name),
        axis=1
    ),
    h_ref=df_tmp31.apply(lambda x: # Высота станции над уровнем моря
        (df_stations[df_stations.name_of_df == f'df_{x.error_at[1]}']['height']).values[0],
        axis=1
    )
)
# Последовательное вычисление из вновь созданных столбцов в нашем случае не работает
# (вопреки заявлению в описании Pandas)
# Назначим столбец Значения температуры, вычисленного из давления, отдельно
df_tmp31=(
    df_tmp31.assign(t_from_P=df_tmp31.apply(lambda x: # Значение температуры, вычисленное из давления
        t_from_P(
            pHg_=x["P1"],
            p0Hg_=x.P0,
            h_=x.h_ref),
        axis=1
    )
).
drop(columns=["P0", "P1", "h_ref"]) # Удалим столбцы, использованные только для вычисления.
```

```
)
```

```
df_tmp31.sample(5, random_state=56)
```

Out[43]:

	T#V_Volochek	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2019-12-25 21:00:00	1.9	0.6	-0.3	-0.1	-0.4	-0.6	-0.5	-0.2	-0.8	-0.4	-0
2018-01-30 21:00:00	-3.8	-0.9	-1.3	-0.9	0.2	0.0	0.0	0.2	0.0	0.1	0
2020-07-12 21:00:00	14.0	13.6	12.7	12.6	13.9	14.5	13.7	13.6	14.9	16.9	16
2014-02-26 18:00:00	1.4	0.7	0.4	0.3	0.9	1.1	1.1	0.9	0.2	0.2	0
2008-06-23 21:00:00	17.4	17.0	17.7	17.3	15.4	17.6	17.4	17.1	16.4	10.4	16

Для подтверждения, являются ли отобранные значения температуры ошибками, вычислим значения температуры в градусах Цельсия, исходя из влажности и температуры точки росы (на основании приближения Августа-Роша-Магнуса).

In [44]:

```
df_tmp31=(  
    df_tmp31.assign(DP=df_tmp31.apply(lambda x: # Давление на уровне моря  
        reference_param_extractor(  
            station_=x.error_at[1],  
            param_='Dew_point',  
            dict_archive_=dict_df_locations,  
            idx_station_=x.name),  
        axis=1  
    ),  
    Humid=df_tmp31.apply(lambda x: # Давление на уровне станции  
        reference_param_extractor(  
            station_=x.error_at[1],  
            param_='Humid',
```

```
        dict_archive_=dict_df_locations,
        idx_station_=x.name),
    axis=1
)
)

# Последовательное вычисление из вновь созданных столбцов в нашем случае не работает
# (вопреки заявленному в описании Pandas)
# Назначим столбец Значения температуры, вычисленного из температуры точки росы, отдельно
df_tmp31=(
    df_tmp31.assign(t_from_DP=df_tmp31.apply(lambda x: # Значение температуры, вычисленное из температуры точки росы
        t_from_dew_point(
            dew_point_=x.DP,
            humid_=x.Humid),
        axis=1
    )
).
drop(columns=["DP", "Humid"]) # Удалим столбцы, использованные только для вычисления.
)

df_tmp31.sample(5, random_state=56)
```

Out[44]:

	T#V_Volochev	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
<b>2019-12-25 21:00:00</b>	1.9	0.6	-0.3	-0.1	-0.4	-0.6	-0.5	-0.2	-0.8	-0.4	-0
<b>2018-01-30 21:00:00</b>	-3.8	-0.9	-1.3	-0.9	0.2	0.0	0.0	0.2	0.0	0.1	0
<b>2020-07-12 21:00:00</b>	14.0	13.6	12.7	12.6	13.9	14.5	13.7	13.6	14.9	16.9	16
<b>2014-02-26 18:00:00</b>	1.4	0.7	0.4	0.3	0.9	1.1	1.1	0.9	0.2	0.2	0
<b>2008-06-23 21:00:00</b>	17.4	17.0	17.7	17.3	15.4	17.6	17.4	17.1	16.4	10.4	16

◀ ▶

Оценим корректность значений температуры исходя из параметров максимальной и минимальной температур за последние 12 часов наблюдений

In [45]:

```
df_tmp31=df_tmp31.assign(T_min=df_tmp31.apply(lambda x: reference_param_extractor(
    station_=x.error_at[1],
    param_='T_min',
    dict_archive_=dict_df_locations,
    idx_station_=x.name),
    axis=1
),
T_max=df_tmp31.apply(lambda x: reference_param_extractor(
    station_=x.error_at[1],
    param_='T_max',
    dict_archive_=dict_df_locations,
    idx_station_=x.name),
    axis=1)
)
```

```
In [46]: df_tmp31.sample(5, random_state=56)
```

	T#V_Volochek	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2019-12-25 21:00:00	1.9	0.6	-0.3	-0.1	-0.4	-0.6	-0.5	-0.2	-0.8	-0.4	-0.0
2018-01-30 21:00:00	-3.8	-0.9	-1.3	-0.9	0.2	0.0	0.0	0.2	0.0	0.1	0.0
2020-07-12 21:00:00	14.0	13.6	12.7	12.6	13.9	14.5	13.7	13.6	14.9	16.9	16.0
2014-02-26 18:00:00	1.4	0.7	0.4	0.3	0.9	1.1	1.1	0.9	0.2	0.2	0.0
2008-06-23 21:00:00	17.4	17.0	17.7	17.3	15.4	17.6	17.4	17.1	16.4	10.4	16.0

## Применимость критериев ошибочных значений

Рассмотрим полученные данные об аномальных значениях.

```
In [47]: print(f'В поле метеостанций выявлено аномальных значений: {df_tmp31.error_at.count()}, из них:\n'
      f'Подтверждается аномалиями в промежутке +/- 3 суток на тот же момент наблюдения: {df_tmp31.time_out_3D.count()}\n'
      f'Подтверждается аномалиями в промежутке +/- 48 часов по всем моментам наблюдения: {df_tmp31.time_out_48H.count()}' )
```

В поле метеостанций выявлено аномальных значений: 1263,

из них:

Подтверждается аномалиями в промежутке +/- 3 суток на тот же момент наблюдения: 124

Подтверждается аномалиями в промежутке +/- 48 часов по всем моментам наблюдения: 71

```
In [48]: t_from_P_min = df_tmp31.apply(lambda x: abs(x.field_out[0][0] - x.t_from_P), axis=1).min()
t_from_P_max = df_tmp31.apply(lambda x: abs(x.field_out[0][0] - x.t_from_P), axis=1).max()
dt_from_P = df_tmp31.error_at[df_tmp31.apply(lambda x: abs(x.field_out[0][0] - x.t_from_P), axis=1)>0.5].count()
dt_from_DP = df_tmp31.error_at[df_tmp31.apply(lambda x: abs(x.field_out[0][0] - x.t_from_DP), axis=1)>0.5].count()
```

```

print(f'Минимальное отклонение аномальных значений от расчётных по давлению: {t_from_P_min:.3f}\n'
      f'Максимальное отклонение аномальных значений от расчётных по давлению: {t_from_P_max:.3f}\n'
      f'Количество отклонений аномальных значений от расчётных по давлению в 0,5 и более градусов : {dt_from_P}\n'
      f'Количество отклонений аномальных значений от расчётных по точке росы в 0,5 и более градусов : {dt_from_DP}'
)

```

Минимальное отклонение аномальных значений от расчётных по давлению: 0.001  
 Максимальное отклонение аномальных значений от расчётных по давлению: 1871.051  
 Количество отклонений аномальных значений от расчётных по давлению в 0,5 и более градусов : 1134  
 Количество отклонений аномальных значений от расчётных по точке росы в 0,5 и более градусов : 0

- Параметры минимальной и максимальной температур имеют огромное количество пропущенных значений и потому вряд ли могут быть использованы для оценки корректности данных о текущей температуре на метеостанциях.
- Чрезмерные отклонения аномальных значений от расчётных по давлению говорит о наличии ошибок в данных о давлении.
- Отсутствие отклонений аномальных значений от расчётных по температуре точки росы указывает на то, что точка росы, скорее всего, рассчитывалась уже позже, по зафиксированным данным, включающим ошибки.

### Определим конечный критерий ошибочных значений:

1. Аномальное значение выявлено в поле метеостанций И ПРИ ЭТОМ

- Аномальное значение выявлено во временном ряду на один и тот же час за несколько суток (выбивается из общей тенденции изменения температуры) ИЛИ
- Аномальное значение выявлено во временном ряду по всем моментам наблюдения за несколько часов (несколько суток) (выбивается из среднесуточного диапазона) ИЛИ
- Расчётное значение температуры отличается от значения зафиксированного выброса на 0,5 градуса Цельсия или более - то есть выброшенное значение не подтверждается расчётым значением

In [49]:

```

# Создадим логические маски для указанного критерия
mask1 = pd.notna(df_tmp31.field_out)
mask2 = pd.notna(df_tmp31.time_out_3D)
mask3 = pd.notna(df_tmp31.time_out_48H)
mask4 = (df_tmp31.apply(lambda x: abs(x.t_from_P - x.field_out[0][0]), axis=1)) >= 0.5
mask = mask1 & (mask2 | mask3 | mask4)

```

Выведем только строки с аномальными значениями, которые в соответствии с указанными выше критериями являются ошибками

In [50]: `df_tmp31[mask]`

Out[50]:

	T#V_Volochek	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2022-06-04 18:00:00	14.4	18.8	19.6	20.7	20.4	20.7	20.0	19.3	21.3	20.0	20
2022-05-25 21:00:00	8.4	8.7	9.7	8.3	8.2	9.2	8.2	9.3	8.1	9.7	8
2022-05-21 21:00:00	11.2	11.8	7.1	10.2	10.3	10.5	11.9	11.0	10.7	9.9	10
2022-05-21 18:00:00	15.4	15.4	11.5	15.4	15.6	15.2	14.8	14.8	15.5	15.6	15
2022-05-08 12:00:00	10.5	15.5	17.0	17.9	17.5	15.8	17.9	18.2	17.9	17.5	16
...	...	...	...	...	...	...	...	...	...	...	...
2005-10-16 09:00:00	3.9	3.9	3.0	3.5	3.7	3.9	3.1	3.6	4.5	NaN	4
2005-10-08 09:00:00	7.9	8.6	8.7	9.0	8.8	8.2	8.2	7.9	8.3	NaN	8
2005-09-21 09:00:00	NaN	NaN	NaN	11.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-09-17 15:00:00	NaN	8.2	6.7	8.9	7.2	7.2	7.5	8.0	7.9	NaN	8
2005-08-11 15:00:00	NaN	15.8	14.6	15.6	15.3	15.6	14.0	15.4	15.6	NaN	15

1149 rows × 20 columns

## Удалим (заменим на NaN) все ошибочные значения

```
In [51]: # Создадим список координат ячеек, содержащих значения, определённые как ошибки
list_error_coords = df_tmp31[mask].error_at.tolist()
# list_error_coords
```

```
In [52]: # Восстановим исходное состояние df_tmp31
df_tmp31 = dict_df_parameters[param_df_name].copy(deep=True)
```

```
In [53]: for error_coords in list_error_coords: # по списку координат ошибочных значений
    # Преобразуем координату столбца и присваиваем ячейке NaN
    df_tmp31.at[error_coords[0], PARAMETER31+'#' + error_coords[1]] = np.nan
# df_tmp31
```

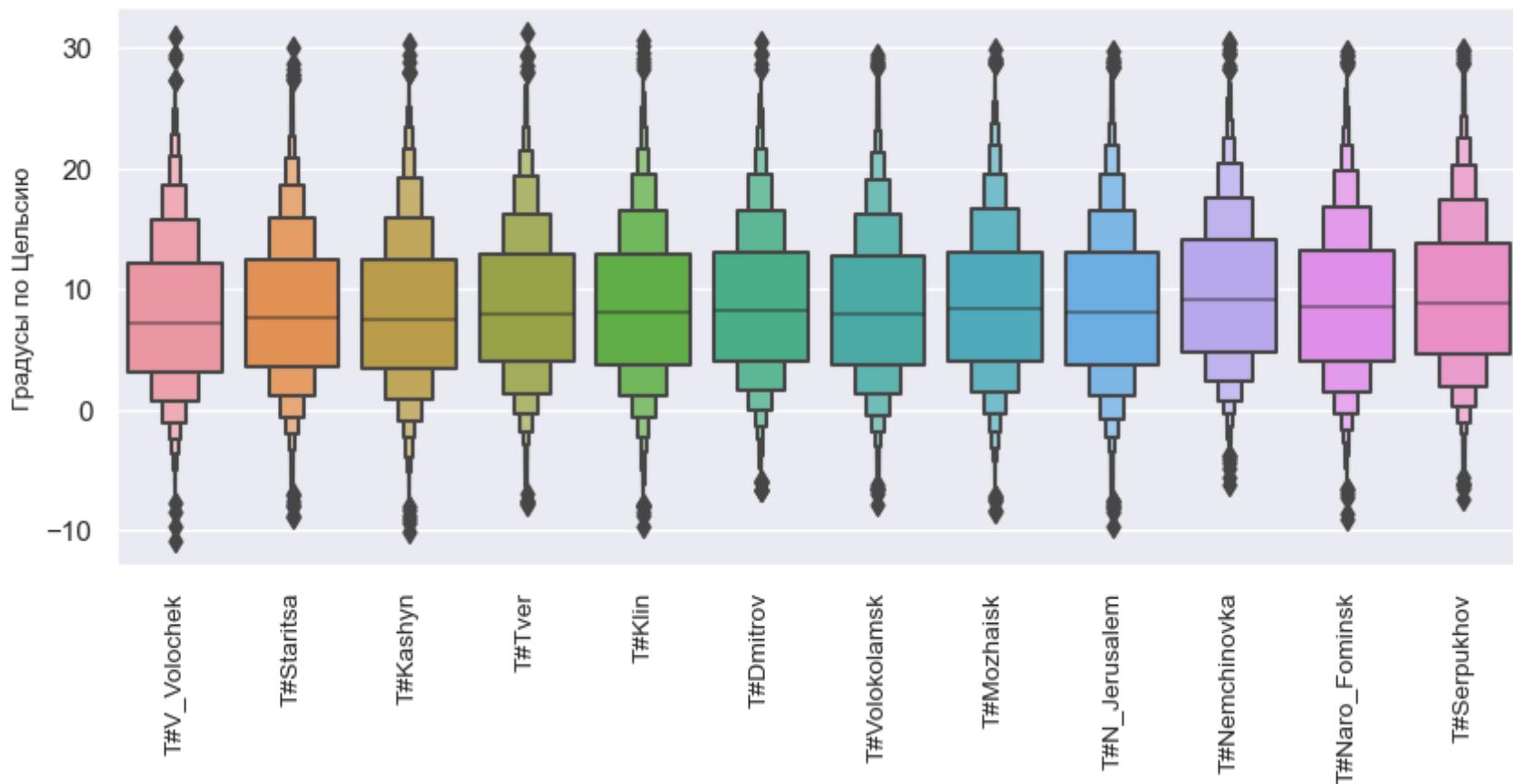
```
In [54]: # Проверим, все ли ошибки заменены на NaN
# Количество неNaN значений в df_tmp31 по координатам, указанным в списке list_error_coords
np.sum([pd.notna([df_tmp31.at[error_coords[0], PARAMETER31+'#' + error_coords[1]] for error_coords in list_error_coords])])
```

```
Out[54]: 0
```

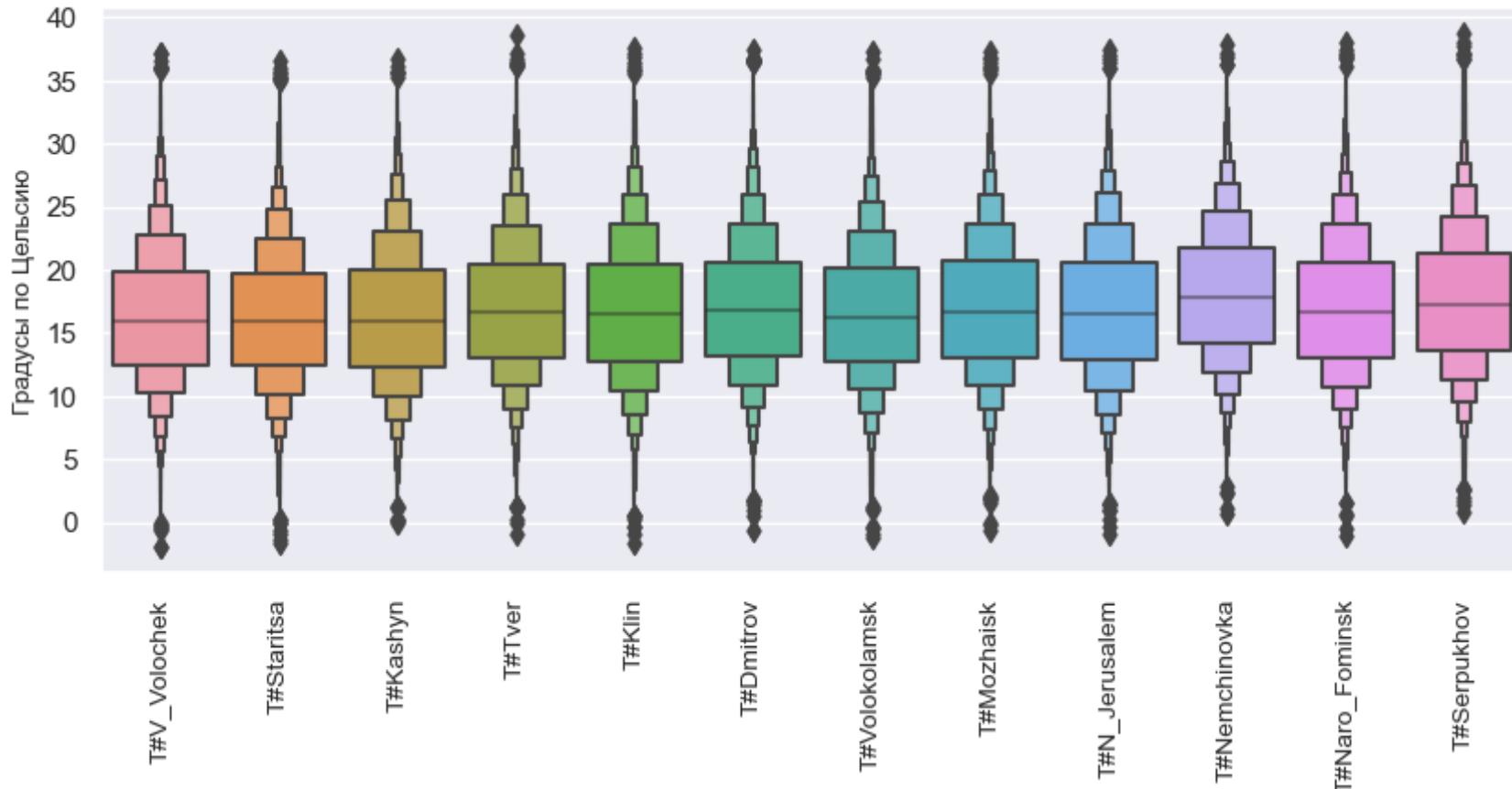
## Визуализируем архив температурь (T) по сезонам после удаления ошибок

```
In [55]: # В цикле выведем графики температур по метеостанциям в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp31).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp31[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Градусы по Цельсию', size=10)
    dummy = g.set_title(f'Распределение значений T в разрезе метеостанций после удаления ошибок:\n{season_name}')
    plt.show()
```

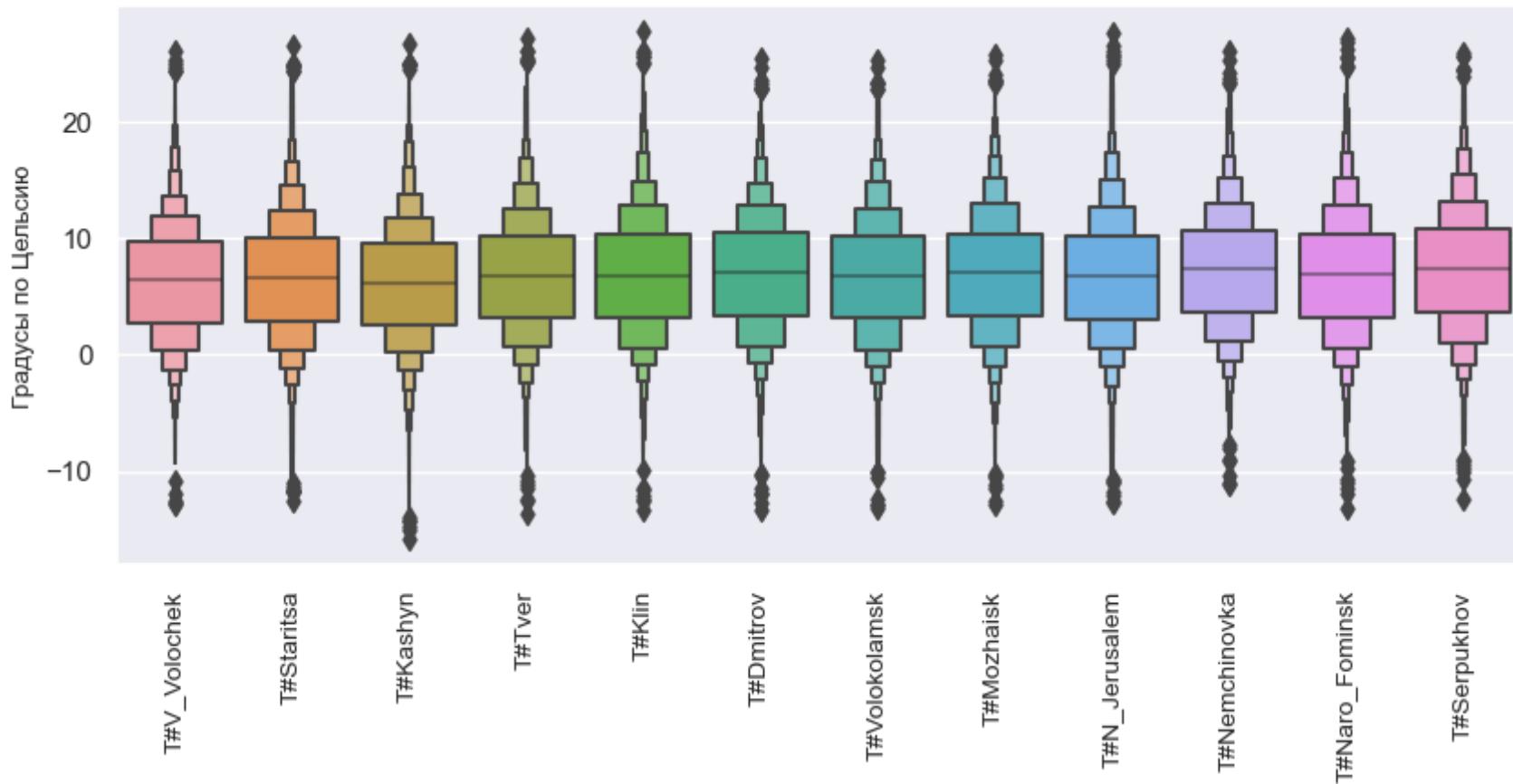
Распределение значений Т в разрезе метеостанций после удаления ошибок:  
spring



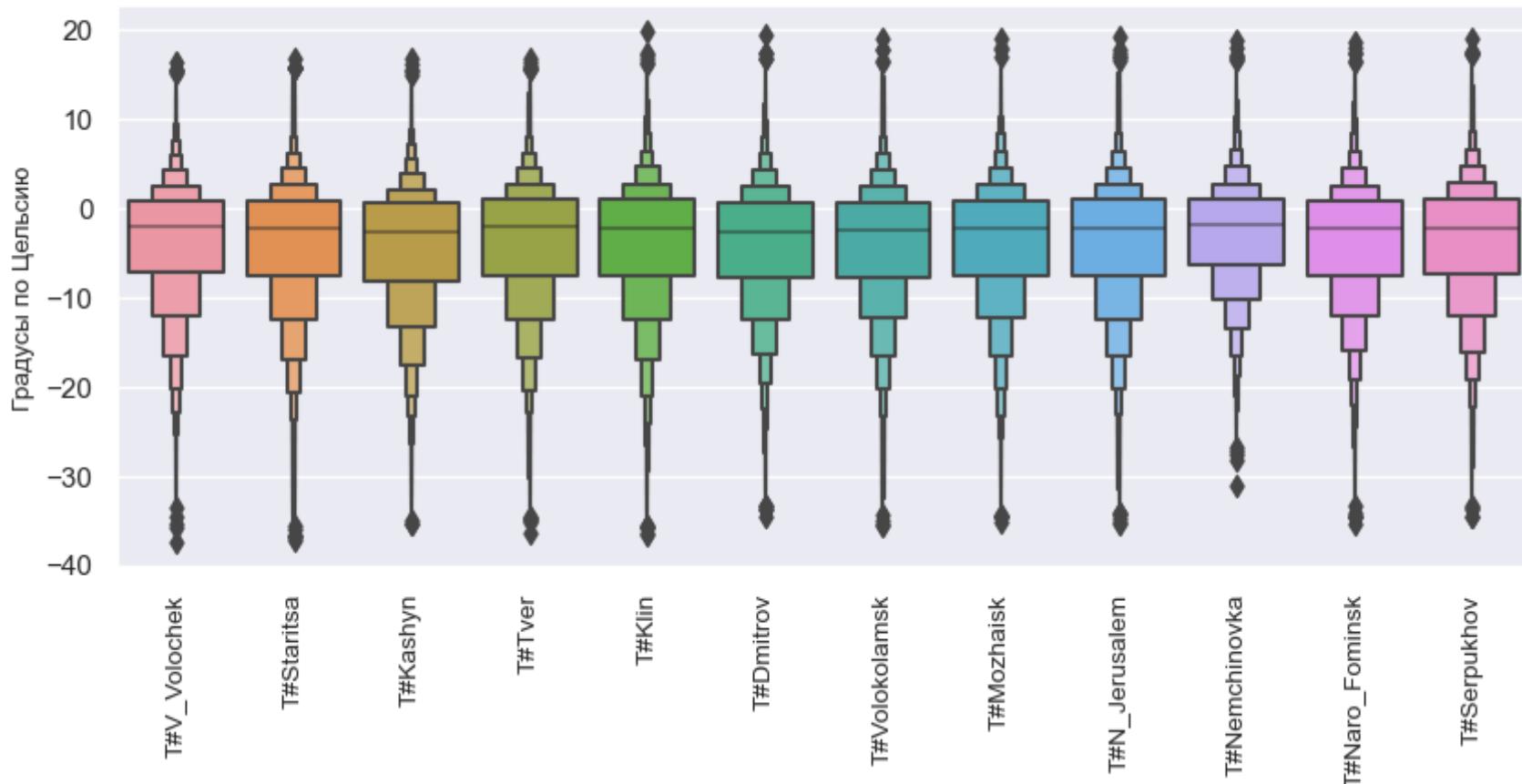
Распределение значений Т в разрезе метеостанций после удаления ошибок:  
summer



Распределение значений Т в разрезе метеостанций после удаления ошибок:  
autumn



### Распределение значений Т в разрезе метеостанций после удаления ошибок: winter



Выведем минимальное и максимальное значения, а также значение медианы и средней для всего DF после удаления ошибок

```
In [56]: print(f'Минимальное значение: {np.nanmin(df_tmp31)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp31)},\n'
      f'Средняя: {np.nanmean(df_tmp31)},\n'
      f'Медиана: {np.nanmedian(df_tmp31)})')
```

Минимальное значение: -37.5,  
Максимальное значение: 38.7,  
Средняя: 5.820500584922598,  
Медиана: 5.5

## Сохраним очищенные данные в файл параметров

```
In [57]: # # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
clean_path = f'{path}clean/'  
  
makedirs(clean_path, exist_ok=True)  
  
# Запишем текущие данные в файл  
print('df_'+PARAMETER31 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER31].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER31}.csv'  
)  
print('DONE!')  
  
df_T.csv -> DONE!
```

### 3.1.2. Восстановление "сплошных" NaN методом средней между соседними значениями соответствующего часа наблюдений для показателя температуры Т

Модели пространственной экстраполяции не могут экстраполировать значения в рамках одного момента наблюдения, если значения во всех точках наблюдения неизвестны. Такие случаи есть в наших архивах.

```
In [58]: # Подсчитаем количество строк со "сплошными" NaN в df_tmp31  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp31.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp31.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 167

Очевидно такие строки нужно заполнить до пространственной экстраполяции. Представляется, что лучшим способом заполнения пропущенных строк будет средняя по тем же часам наблюдения между двумя соседними днями

```
In [59]: # Создадим список DateTime индексов строк со сплошными NaN  
# Выбираем из датафрейма строки, где количество NaN равно количеству столбцов - то есть сплошные NaN
```

```
list_time_total_nans = df_tmp31[df_tmp31.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp31.keys())].index.tolist()
```

```
In [60]: # По списку
for idx in list_time_total_nans[:-1]: # кроме самого раннего TimeStamp, для которого нельзя вычислить start_date
    # определяем начало и конец временного интервала
    start_time = idx - pd.Timedelta('1D')
    end_time = idx + pd.Timedelta('1D')
    while sum(df_tmp31.loc[start_time].notna()) == 0: # Пока ряд от start_time тоже состоит из сплошных NaN
        start_time = start_time - pd.Timedelta('3H') # Уменьшаем start_time на 3 часа
    while sum(df_tmp31.loc[end_time].notna()) == 0: # Пока ряд от end_time тоже состоит из сплошных NaN
        end_time = end_time + pd.Timedelta('3H') # Увеличиваем end_time на 3 часа

    # по ряду сплошных NaN
    for label in df_tmp31.loc[idx].index:
        df_tmp31.at[idx, label] = np.mean([df_tmp31.at[start_time, label], df_tmp31.at[end_time, label]])
```

```
In [61]: # Подсчитаем количество строк со "сплошными" NaN в df_tmp31
# построчно подсчитаем сумму количества NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк
all_nans_count = sum(df_tmp31.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp31.keys())))
print(f'Количество строк со сплошными NaN равно {all_nans_count}' )
```

Количество строк со сплошными NaN равно 1

Всё верно, это самая начальная строка. Она должна остаться

Поскольку в процессе удаления ошибочных значений удалялись и единичные значения в строках, и часть строк могла превратиться в сплошные NaN. Проверим, сколько единичных значений исправлено методом восстановления сплошных NaN

```
In [62]: # Количество неNaN значений в df_tmp31 по координатам, указанным в списке list_error_coords
np.sum([pd.notna([df_tmp31.at[error_coords[0], PARAMETER31+'#'+error_coords[1]] for error_coords in list_error_coords])])
```

Out[62]: 9

Получившееся значение показывает количество исправленных ошибочных значений, там где был применён метод восстановления сплошных NaN.

**3.1.3. Подбор модели для восстановления пропущенных и удалённых значений показателя температуры T, а также для моделирования значений для искомой точки.**

### 3.1.3.1. Модель средней, взвешенной по степени обратных расстояний (далее по тексту эту модель будем называть сокращённо IDW)

Эта модель уже задана в виде функции *inverse\_distance\_avg*.

Модель применяется для каждого отдельно взятого момента наблюдения. Входные данные зафиксированы:

- Матрица расстояний между точками в поле метеостанций (df\_stations)
- Известные значения показателей для точек в поле метеостанций (архивы параметров).

Ожидаемые выходные данные:

- значение показателя для моделируемой точки (по сути, все значения NaN, включая значения для Агробиостанции МГУ в Чашниково).

Модель создана специально для имеющегося набора данных, поэтому для её работы не потребуется значительных преобразований архивов. Сама модель написана "вручную", без использования библиотечных функций машинного обучения.

Выше мы использовали формулу средней, взвешенной по обратным квадратам расстояния (по умолчанию в *inverse\_distance\_avg* задан параметр степени равный 2). Однако степень, в которую возводятся обратные величины расстояний - это единственный параметр, который можно менять в нашей реализации модели IDW.

Попробуем, как работает модель с различными значениями степени для обратных расстояний, и выделим ту степень, которая обеспечивает лучшие метрики качества.

Создадим набор данных для обучения и валидации модели IDW.

1. Используем данные в df\_tmp31.
2. Уберём все строки с NaN.
3. Выделим рандомно массив известных значений для разных метеостанций и разных моментов наблюдения - он потребуется для разделения на обучающую и валидационную часть и для расчёта метрик качества.

```
In [63]: # Создаём датафрейм для валидации модели IDW
df_test = df_tmp31.dropna(how='any')
```

```
In [64]: df_test.info()
df_test.shape
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 27543 entries, 2022-06-09 03:00:00 to 2007-09-26 15:00:00
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   T#V_Volochek    27543 non-null   float64
 1   T#Staritsa      27543 non-null   float64
 2   T#Kashyn        27543 non-null   float64
 3   T#Tver          27543 non-null   float64
 4   T#Klin          27543 non-null   float64
 5   T#Dmitrov       27543 non-null   float64
 6   T#Volokolamsk  27543 non-null   float64
 7   T#Mozhaisk      27543 non-null   float64
 8   T#N_Jerusalem   27543 non-null   float64
 9   T#Nemchinovka  27543 non-null   float64
 10  T#Naro_Fominsk 27543 non-null   float64
 11  T#Serpukhov    27543 non-null   float64
dtypes: float64(12)
memory usage: 2.7 MB
(27543, 12)
```

```
Out[64]:
```

Создадим массив индексов для выборки моделируемых и проверочных значений. Массив будет включать последовательно все индексы строк и случайный индекс столбца.

```
# Зафиксируем RandomState
rs56 = np.random.RandomState(56)
# Создаём 1мерный массив с последовательностью, равной количеству рядов в df_test
arr_row_index = np.arange(0, df_test.shape[0])
# Создаём 1мерный массив со случайными целыми числами в диапазоне количества столбцов в df_test
arr_column_index = rs56.randint(0, df_test.shape[1], df_test.shape[0])
# Соединяем 2 массива и транспонируем полученный массив
arr_idx_data = np.vstack((arr_row_index, arr_column_index)).T

arr_row_index
arr_column_index
arr_idx_data
```

```
Out[65]: array([    0,     1,     2, ..., 27540, 27541, 27542])
```

```
Out[65]: array([ 5,  4,  0, ...,  9, 11, 11])
```

```
Out[65]: array([[ 0,  5],
 [ 1,  4],
 [ 2,  0],
 ...,
 [27540,  9],
 [27541, 11],
 [27542, 11]])
```

Создадим тренировочный и обучающий массивы. Для этого:

- определим  $X$  как массив координат ячеек в архиве - (np.array),
- определим  $y$  как массив значений ячеек в множестве  $X$  - (np.array).

```
In [66]: # Разделим наши данные на обучающую и валидационную части.
# используем индексы значений как параметры модели
X = arr_idx_data
# результирующие значения (фактические значения измерений, соответствующие индексам), выведем в список и преобразуем в np.array
y = np.array([df_test.iloc[x, y] for x, y in arr_idx_data])

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.6, test_size=0.4, random_state=56)
x_train.shape
y_train.shape
x_test.shape
y_test.shape
```

```
Out[66]: (16525, 2)
```

```
Out[66]: (16525,)
```

```
Out[66]: (11018, 2)
```

```
Out[66]: (11018,)
```

**Обучающий датасет** Подберём лучшую степень для весов - обратных расстояний, ориентируясь на лучшие значения метрик качества

```
In [67]: start_time = time.time() # для замера времени выполнения кода
```

```
r2_idw_tr, mae_idw_tr, rmse_idw_tr = 0, 0, 0 # обнулим значения метрик качества
iterations = 0 # количество итераций
```

```
power = 3 # Начальное значение степени (опровергнуты начальные степени от 1)
power_increment = 0.25 # шаг увеличения степени
list_metrics=[] # список для фиксации результатов итераций

r2_idw_tr_old = 0 # Устанавливаем в 0 предыдущее значение R2
print('ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:\n')

# Зададим предел R2 для поиска оптимального веса
while r2_idw_tr_old <= r2_idw_tr:
    power += power_increment # Увеличиваем степень на 1 шаг
    iterations +=1 # Увеличиваем счётчик проходов цикла
    r2_idw_tr_old = r2_idw_tr # Запоминаем предыдущее значение R2

    # Создаём y_predict_idw_tr по индексам в массиве x_train
    # используем функцию inverse_distance_avg
    # Проходим по массиву и считываем из df_test данные по координатам, выводим результат списком
    y_predict_idw_tr = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param=PARAMETER31,
                             station=df_test.keys()[y][len(PARAMETER31)+1:],
                             df_dists=df_station_dists,
                             power=power)
        for x, y in x_train
    ]

    # Расчитываем метрики качества
    max_e_idw_tr = max_error(y_train, y_predict_idw_tr)
    mae_idw_tr = mean_absolute_error(y_train, y_predict_idw_tr)
    mse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr)
    rmse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr, squared=False)
    r2_idw_tr = r2_score(y_train, y_predict_idw_tr)

    if r2_idw_tr > r2_idw_tr_old:
        best_power_idw_tr = power
        best_max_e_idw_tr = max_e_idw_tr
        best_mae_idw_tr = mae_idw_tr
        best_mse_idw_tr = mse_idw_tr
        best_rmse_idw_tr = rmse_idw_tr
        best_r2_idw_tr = r2_idw_tr

    # замер времени:
    chk_time = time.time()
    elapsed_time = chk_time - start_time
    time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
```

```
print(f'Elapsed time={time_formatted}\n'
      f'Текущие значения: iterations={iterations}, power={power},\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_tr:.7f}\n'
    )
print(f'ЛУЧШИЕ значения: power={best_power_idw_tr},\n'
      f'Максимальная ошибка (MAX_E) IDW = {best_max_e_idw_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {best_mae_idw_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {best_mse_idw_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {best_rmse_idw_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {best_r2_idw_tr:.7f}'
```

ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:

Elapsed time=00:01:34

Текущие значения: iterations=1, power=3.25,  
Максимальная ошибка (MAX\_E) IDW = 10.9297118  
Средняя абсолютная ошибка (MAE) IDW = 0.8303679  
Средний квадрат ошибки (MSE) IDW = 1.4752679  
Средняя квадратическая ошибка (RMSE) IDW = 1.2146061  
Коэффициент детерминации (R2) IDW = 0.9871108

Elapsed time=00:03:11

Текущие значения: iterations=2, power=3.5,  
Максимальная ошибка (MAX\_E) IDW = 10.8494908  
Средняя абсолютная ошибка (MAE) IDW = 0.8275582  
Средний квадрат ошибки (MSE) IDW = 1.4673173  
Средняя квадратическая ошибка (RMSE) IDW = 1.2113287  
Коэффициент детерминации (R2) IDW = 0.9871803

Elapsed time=00:04:23

Текущие значения: iterations=3, power=3.75,  
Максимальная ошибка (MAX\_E) IDW = 10.7725369  
Средняя абсолютная ошибка (MAE) IDW = 0.8257495  
Средний квадрат ошибки (MSE) IDW = 1.4627272  
Средняя квадратическая ошибка (RMSE) IDW = 1.2094326  
Коэффициент детерминации (R2) IDW = 0.9872204

Elapsed time=00:05:37

Текущие значения: iterations=4, power=4.0,  
Максимальная ошибка (MAX\_E) IDW = 10.6991235  
Средняя абсолютная ошибка (MAE) IDW = 0.8248255  
Средний квадрат ошибки (MSE) IDW = 1.4609943  
Средняя квадратическая ошибка (RMSE) IDW = 1.2087160  
Коэффициент детерминации (R2) IDW = 0.9872355

Elapsed time=00:06:54

Текущие значения: iterations=5, power=4.25,  
Максимальная ошибка (MAX\_E) IDW = 10.6294415  
Средняя абсолютная ошибка (MAE) IDW = 0.8246861  
Средний квадрат ошибки (MSE) IDW = 1.4616664  
Средняя квадратическая ошибка (RMSE) IDW = 1.2089940  
Коэффициент детерминации (R2) IDW = 0.9872297

ЛУЧШИЕ значения: power=4.0,

Максимальная ошибка (MAX\_E) IDW = 10.6991235

Средняя абсолютная ошибка (MAE) IDW = 0.8248255  
Средний квадрат ошибки (MSE) IDW = 1.4609943  
Средняя квадратическая ошибка (RMSE) IDW = 1.2087160  
Коэффициент детерминации (R2) IDW = 0.9872355

Несколько запусков кода выше, с различными параметрами степени (от 1 до 10), показали, что, судя по R2, лучшим значением степени на обучающем массиве является 4.0. Оно даёт лучшие метрики качества.

Применим эту степень для валидационного массива.

## Валидационный датасет

```
In [68]: # Создаём y_predict_idw_vld по индексам в x_test
# используем функцию inverse_distance_avg
# Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком

y_predict_idw_vld = [
    inverse_distance_avg(row=df_test.iloc[x],
                          param=PARAMETER31,
                          station=df_test.keys()[y][len(PARAMETER31)+1:],
                          df_dists=df_station_dists,
                          power=best_power_idw_tr) # берём лучшее значение степени, полученное из кода выше на тренинговом датасете
    for x, y in x_test
]
# y_predict_idw_vld

max_e_idw_vld = max_error(y_test, y_predict_idw_vld)
mae_idw_vld = mean_absolute_error(y_test, y_predict_idw_vld)
mse_idw_vld = mean_squared_error(y_test, y_predict_idw_vld)
rmse_idw_vld = mean_squared_error(y_test, y_predict_idw_vld, squared=False)
r2_idw_vld = r2_score(y_test, y_predict_idw_vld)
print(f'ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld:.7f}')
)
```

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:

Максимальная ошибка (MAX\_E) IDW = 9.6052652  
Средняя абсолютная ошибка (MAE) IDW = 0.8256745  
Средний квадрат ошибки (MSE) IDW = 1.4599404  
Средняя квадратическая ошибка (RMSE) IDW = 1.2082799  
Коэффициент детерминации (R2) IDW = 0.9871001

## ВЫВОД

Модель средней, взвешенной по обратным расстояниям, изначально даёт очень хорошие метрики качества.

### 3.1.3.2. Кригинг и вариограммы в реализации библиотеки SciKit GStat

*Опробуем работу модели кригинга на уже сформированных тренинговой и валидационной выборках*

Координатная сетка для точек наблюдения в данной модели строится на основе "плоской" земной поверхности. Разность между расстояниями по прямой и по поверхности сферы игнорируется (впрочем, для нашего региона исследования это не приведёт к существенным ошибкам).

```
In [69]: # Загружаем координаты из df_coords_full (определён в разделе определения функции кригинга 2.2):  
# Создаём DF с координатами нужных нам точек  
df_coords = df_coords_full[["LoE", "LaN"]][:-2]  
  
df_coords
```

Out[69]:

LoE      NaN

station	LoE	NaN
V_Volochek	34.566667	57.583333
Staritsa	34.933333	56.500000
Kashyn	37.583333	57.350000
Tver	35.922000	56.857300
Klin	36.716667	56.333333
Dmitrov	37.533333	56.366667
Volokolamsk	35.933333	56.016700
Mozhaisk	36.000000	55.516700
N_Jerusalem	36.816667	55.900000
Nemchinovka	37.350000	55.716667
Naro_Fominsk	36.700000	55.383333
Serpukhov	37.416667	54.916667

## Обучающий датасет

Проверим работу модели кригинга сначала на данных обучающей выборки. На ней будем подбирать наиболее подходящие параметры вариограмм и модели кригинга (которые дают лучшие метрики и выдают меньшее количество ошибок из-за недостаточности наблюдений в поле метеостанций).

Представляется полезным сравнить работу модели обратных степеней расстояний и кригинга на одних и тех же данных.

В процессе исследования работоспособности модели код ниже многократно запускался с различными параметрами.

```
In [70]: # Намеренно оставим закомментированные части кода, они могут использоваться для отладки
# start_time = time.time() # для замера времени выполнения кода
# counter = 0
y_predict_kriging_tr = [] # список предиктов для x_test
```

```

for x in x_train: # x[0] ряд в df_test, x[1] столбец, соответствующий названию станции
#     counter += 1
##
#     if counter >15:
#         break
#     else:
#         counter +=1
##
# Найдем по координатам x_test ряд в df_test
row = df_test.iloc[x[0]]
# Присваиваем проверяемому значению NaN
row.iloc[x[1]] = np.nan
# Для построения вариограммы:
# - получаем массив координат без указанной точки
# (удаляем соответствующий ряд из df_coords, преобразуем оставшийся df в массив)
# - получаем массив значений
idx = x[1]
coords_v = np.array(df_coords.drop(index = df_coords.iloc[x[1]].name))[:,2]
vals_v = np.array(row.dropna())
try:
    # Определяем вариограмму
    V = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.99999, # Используем всю матрицу расстояний
                        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                        normalize=False,
                        use_nugget=False,
                        samples=len(vals_v) # количество сэмплов приравняем к количеству наблюдений
                        #fit_method='ml',
                        #entropy_bins = 1
                        )
    # V_North = skg.DirectionalVariogram(coordinates=coords_v,
    #                                     values=vals_v,
    #                                     estimator='matheron',
    #                                     model='spherical',
    #                                     dist_func='euclidean',
    #                                     bin_func='even',
    #                                     azimuth=90,
    #                                     tolerance=90,
    #                                     maxLag='full',

```

```

#                                     n_Lags=4)
# V_East = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=0,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)
# V_South = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=-90,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)
# V_West = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=180,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)

##
# V=V_West
##
```

**except ValueError:** # Уберём количество сэмплов из определения вариограммы (когда количество сэмплов слишком велико)

```

try:
    V_ = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.9999, # Используем всю матрицу расстояний

```

```

        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
        normalize=False,
        use_nugget=False,
        #fit_method='ml',
        #entropy_bins = 1
    );
except: # Возникает ошибка - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_tr.append(val_predict)
    continue
except: # возникает другая ошибка (помимо ValueError) - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_tr.append(val_predict)
    continue

# Определяем модель кригинга
model_ok = skg.OldinaryKriging(V, min_points=1, max_points=14, mode='exact')

# координаты точки предикта:
predict_coords = np.array(df_coords)
# Получаем предикт для тестируемой точки (получаем массив предиктов, выбираем из него индекс точки предикта)
# В процессе работы model_ok.transform выдается много сообщений типа:
# 'Warning: for %d Locations, not enough neighbors were found within the range.' % self.no_points_error
# "Заглушим" их, направив их в класс вывода, который ничего не делает (определен выше)

sys.stdout = NullIO() # определим вывод print() в класс нулевого вывода

val_predict = model_ok.transform(predict_coords[:,0], predict_coords[:,1])[x[1]]
sys.stdout = real_stdout # восстановим стандартный вывод

# добавляем предикт в список предиктов
y_predict_kriging_tr.append(val_predict)

## 
# if V.describe()['effective_range'] < 1:
#     dm = pdist(df_coords[['LoE', 'LaN']]) # массив пар расстояний
#     print(f'Итерация: {counter}, позиция в x_test: {x}\nКоличество пар расстояний: {len(dm)}\n'
#           f'Effective Range: {V.describe()["effective_range"]}\n'
#           f'Количество пар расстояний внутри Effective range: {sum(dm <= V.describe()["effective_range"])}\n'
#           f'Количество пар расстояний вне Effective range: {sum(dm > V.describe()["effective_range"])}\n'
#           f'Предикт: {val_predict}, координаты предикта: {predict_coords[x[1]]}, станция: {df_coords.iloc[x[1]].name}'
#           )
#     fig = V.plot(show=False)

```

```

#         plt.show()
## 
y_predict_kriging_tr = np.array(y_predict_kriging_tr) # превратим список предиктов в массив

# замер времени:
chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

```

```

In [71]: if np.isnan(np.array(y_predict_kriging_tr)).sum() == 0:
    max_e_kriging_tr = max_error(y_train, y_predict_kriging_tr)
    mae_kriging_tr = mean_absolute_error(y_train, y_predict_kriging_tr)
    mse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr)
    rmse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr, squared=False)
    r2_kriging_tr = r2_score(y_train, y_predict_kriging_tr)
else:
    max_e_kriging_tr = np.nan
    mae_kriging_tr = np.nan
    mse_kriging_tr = np.nan
    rmse_kriging_tr = np.nan
    r2_kriging_tr = np.nan

print('ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ')
print(f'Elapsed time={time_formatted}\n'
      f'Значения метрик:\n'
      f'R2={r2_kriging_tr:.7f}, MAX_E={max_e_kriging_tr}, MAE={mae_kriging_tr:.7f}, MSE={mse_kriging_tr}, '
      f'RMSE={rmse_kriging_tr:.7f}\n'
      )

print(f'Количество значений, которые не удалось предсказать: {pd.isna([y_predict_kriging_tr]).sum()}\n'
      f'Это составляет {pd.isna([y_predict_kriging_tr]).sum()/len(y_predict_kriging_tr):.4%} от обучающего массива данных')

```

ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ  
Elapsed time=00:05:08  
Значения метрик:  
R2=nan, MAX\_E=nan, MAE=nan, MSE=nan, RMSE=nan

Количество значений, которые не удалось предсказать: 635  
Это составляет 3.8427% от обучающего массива данных

Попробуем оценить качество работы модели в случаях, когда ей удалось найти предикты.

```
In [72]: # Оставим в y_train и y_predict_kriging_tr только значения неравные NaN  
y_train_kriging_shrunk = y_train[~np.isnan(y_predict_kriging_tr)]  
y_predict_kriging_tr_shrunk = y_predict_kriging_tr[~np.isnan(y_predict_kriging_tr)]
```

```
max_e_kriging_tr = max_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)  
mae_kriging_tr = mean_absolute_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)  
mse_kriging_tr = mean_squared_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)  
rmse_kriging_tr = mean_squared_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk, squared=False)  
r2_kriging_tr = r2_score(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
```

```
In [73]: print(f'КРИГИНГ на ОБУЧАЮЩЕМ датасете (для случаев, где удалось найти предикты):\n'  
      f'Максимальная ошибка (MAX_E) Kriging = {max_e_kriging_tr:.7f}, Kriging - IDW = '  
      f'{(max_e_kriging_tr - max_e_idw_tr):.7f}\n'  
      f'Средняя абсолютная ошибка (MAE) Kriging = {mae_kriging_tr:.7f}, Kriging - IDW = '  
      f'{(mae_kriging_tr - mae_idw_tr):.7f}\n'  
      f'Средний квадрат ошибки (MSE) Kriging = {mse_kriging_tr:.7f}, Kriging - IDW = '  
      f'{(mse_kriging_tr - mse_idw_tr):.7f}\n'  
      f'Средняя квадратическая ошибка (RMSE) Kriging = {rmse_kriging_tr:.7f}, '  
      f'Kriging - IDW = {(rmse_kriging_tr - rmse_idw_tr):.7f}\n'  
      f'Коэффициент детерминации (R2) Kriging = {r2_kriging_tr:.7f}, Kriging - IDW = '  
      f'{(r2_kriging_tr - r2_idw_tr):.7f}'  
)
```

КРИГИНГ на ОБУЧАЮЩЕМ датасете (для случаев, где удалось найти предикты):

Максимальная ошибка (MAX\_E) Kriging = 9.6527528, Kriging - IDW = -0.9766887  
Средняя абсолютная ошибка (MAE) Kriging = 0.7752786, Kriging - IDW = -0.0494074  
Средний квадрат ошибки (MSE) Kriging = 1.3060501, Kriging - IDW = -0.1556163  
Средняя квадратическая ошибка (RMSE) Kriging = 1.1428255, Kriging - IDW = -0.0661685  
Коэффициент детерминации (R2) Kriging = 0.9885618, Kriging - IDW = 0.0013322

## Валидационный датасет

Посмотрим, как модель будет отрабатывать на валидационной выборке.

```
In [74]: start_time = time.time() # для замера времени выполнения кода  
# counter = 0  
y_predict_kriging_vld = [] # список предиктов для x_test  
  
for x in x_test: # x[0] ряд в df_test, x[1] столбец, соответствующий названию станции  
#     counter += 1  
##  
#     if counter >15:
```

```

#         break
#     else:
#         counter +=1
##
# Найдем по координатам x_test ряд в df_test
row = df_test.iloc[x[0]]
# Присваиваем проверяемому значению NaN
row.iloc[x[1]] = np.nan
# Для построения вариограммы:
# - получаем массив координат без указанной точки
# (удаляем соответствующий ряд из df_coords, преобразуем оставшийся df в массив)
# - получаем массив значений
idx = x[1]
coords_v = np.array(df_coords.drop(index = df_coords.iloc[x[1]].name))[:, :2]
vals_v = np.array(row.dropna())

try:
    # Определяем вариограмму
    V = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.99999, # Используем всю матрицу расстояний
                        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                        normalize=False,
                        use_nugget=False,
                        samples=len(vals_v),
                        fit_method='trf',
                        )
except ValueError: # Уберём количество сэмплов из определения вариограммы (когда количество сэмплов слишком велико)
    try:
        V_ = skg.Variogram(coordinates=coords_v,
                            values=vals_v,
                            estimator='matheron',
                            model='spherical',
                            dist_func='euclidean',
                            bin_func='ward',
                            maxlag=0.99999, # Используем всю матрицу расстояний
                            n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                            normalize=False,
                            use_nugget=False,
                            #fit_method='ml',
    
```

```

        #entropy_bins = 1
    );
except: # Возникает ошибка - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_vld.append(val_predict)
    continue
except: # возникает другая ошибка (помимо ValueError) - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_vld.append(val_predict)
    continue

# Определяем модель кригинга
model_ok = skg.OldinaryKriging(V, min_points=1, max_points=14, mode='exact')

# координаты точки предикта:
predict_coords = np.array(df_coords)
# Получаем предикт для тестируемой точки (получаем массив предиктов, выбираем из него индекс точки предикта)
# В процессе работы model_ok.transform выдается много сообщений типа:
# 'Warning: for %d Locations, not enough neighbors were found within the range.' % self.no_points_error
# "Заглушим" их, направив их в класс вывода, который ничего не делает (определен выше)

sys.stdout = NullIO() # определим вывод print() в класс нулевого вывода

val_predict = model_ok.transform(predict_coords[:,0], predict_coords[:,1])[x[1]]
sys.stdout = real_stdout # восстановим стандартный вывод

# добавляем предикт в список предиктов
y_predict_kriging_vld.append(val_predict)

y_predict_kriging_vld = np.array(y_predict_kriging_vld)

# замер времени:
chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

```

In [75]:

```

if np.isnan(np.array(y_predict_kriging_vld)).sum() == 0:
    max_e_kriging_vld = max_error(y_test, y_predict_kriging_vld)
    mae_kriging_vld = mean_absolute_error(y_test, y_predict_kriging_vld)
    mse_kriging_vld = mean_squared_error(y_test, y_predict_kriging_vld)
    rmse_kriging_vld = mean_squared_error(y_test, y_predict_kriging_vld, squared=False)
    r2_kriging_vld = r2_score(y_test, y_predict_kriging_vld)

```

```

else:
    max_e_kriging_vld = np.nan
    mae_kriging_vld = np.nan
    mse_kriging_vld = np.nan
    rmse_kriging_vld = np.nan
    r2_kriging_vld = np.nan

print(f'Elapsed time={time_formatted}\n'
      f'Значения метрик:\n'
      f'R2={r2_kriging_vld:.7f}, MAX_E={max_e_kriging_vld:.7f}, MSE={mse_kriging_vld:.7f}, '
      f'RMSE={rmse_kriging_vld:.7f}\n')
print('ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ')
print(f'Количество значений, которые не удалось предсказать: {np.isnan([y_predict_kriging_vld]).sum()}\n'
      f'Это составляет {pd.isna([y_predict_kriging_vld]).sum()/len(y_predict_kriging_vld):.4%} '
      f'от валидационного массива данных')

```

Elapsed time=00:03:36

Значения метрик:

R2=nan, MAX\_E=nan, MAE=nan, MSE=nan, RMSE=nan

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ

Количество значений, которые не удалось предсказать: 455

Это составляет 4.1296% от валидационного массива данных

```
In [76]: y_test_kriging_shrunk = y_test[~np.isnan(y_predict_kriging_vld)]
y_predict_kriging_vld_shrunk = y_predict_kriging_vld[~np.isnan(y_predict_kriging_vld)]

max_e_kriging_vld = max_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
mae_kriging_vld = mean_absolute_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
mse_kriging_vld = mean_squared_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
rmse_kriging_vld = mean_squared_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk, squared=False)
r2_kriging_vld = r2_score(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
```

```
In [77]: print(f'Кригинг на ВАЛИДАЦИОННОМ датасете (для случаев, где удалось найти предикты):\n'
      f'Максимальная ошибка (MAX_E) Kriging = {max_e_kriging_vld:.7f}, Kriging - IDW = '
      f'{(max_e_kriging_vld - max_e_idw_vld):.7f}\n'
      f'Средняя абсолютная ошибка (MAE) Kriging = {mae_kriging_vld:.7f}, '
      f'Kriging - IDW = {(mae_kriging_vld - mae_idw_vld):.7f}\n'
      f'Средний квадрат ошибки (MSE) Kriging = {mse_kriging_vld:.7f}, Kriging - IDW = '
      f'{(mse_kriging_vld - mse_idw_vld):.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) Kriging = {rmse_kriging_vld:.7f}, '
      f'Kriging - IDW = {(rmse_kriging_vld - rmse_idw_vld):.7f}\n'
      f'Коэффициент детерминации (R2) Kriging = {r2_kriging_vld:.7f}, Kriging - IDW = '
```

```
f'{(r2_kriging_vld - r2_idw_vld):.7f}'  
)
```

Кригинг на ВАЛИДАЦИОННОМ датасете (для случаев, где удалось найти предикты):  
Максимальная ошибка (MAX\_E) Kriging = 9.1718611, Kriging - IDW = -0.4334040  
Средняя абсолютная ошибка (MAE) Kriging = 0.7819587, Kriging - IDW = -0.0437158  
Средний квадрат ошибки (MSE) Kriging = 1.3268325, Kriging - IDW = -0.1331079  
Средняя квадратическая ошибка (RMSE) Kriging = 1.1518821, Kriging - IDW = -0.0563978  
Коэффициент детерминации (R2) Kriging = 0.9882179, Kriging - IDW = 0.0011178

## ВЫВОД

Таким образом, модель кригинга не даёт 100% удовлетворительных результатов, так как в ряде случаев не может предсказать значения, и оказывается неприменимой. Это связано с незначительным размером поля метеостанций и встречающейся ситуацией, когда не удаётся найти ближайшие значения в effective range (расстоянии, вне пределов которого модель считает данные метеостанций статистически независимыми). То есть на этом и большем расстоянии корреляция между значениями показателя у метеостанций становится незначительной или отсутствует.

### 3.1.3.3. Модель кригинга, совмещённая с IDW (для значений, которые кригинг не может предсказать)

Отделим значения, которые не удалось предсказать моделью кригинга, от уже предсказанных значений и формируем новый (сокращённый) обучающий датасет для IDW

```
In [78]: # Поскольку длины массивов x_train, y_train и y_predict_tr, а также x_test, y_test и y_predict_vld соответственно одинаковы,  
# выбираем по индексу значения x_train и y_train, x_test и y_test  
# где значения y_predict_kriging_tr равны NaN  
# формируем новый массив истинных значений  
y_train_idw_shrunk = y_train[np.isnan(y_predict_kriging_tr)]  
x_train_idw_shrunk = x_train[np.isnan(y_predict_kriging_tr)]  
y_test_idw_shrunk = y_test[np.isnan(y_predict_kriging_vld)]  
x_test_idw_shrunk = x_test[np.isnan(y_predict_kriging_vld)]
```

Снова подберём лучшую степень для IDW

```
In [79]: start_time = time.time() # для замера времени выполнения кода  
  
r2_idw_tr_shrunk = 0 # обнулим значение метрики качества R2  
iterations = 0 # количество итераций  
power = 1.75 # Начальное значение степени (опробованы начальные степени от 1 до 10)  
power_increment = 0.25 # шаг увеличения степени
```

```
list_metrics=[] # список для фиксации результатов итераций

r2_idw_tr_shrunk_old = 0 # Устанавливаем в 0 предыдущее значение R2
print('НОВЫЙ ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:\n')

# Зададим предел R2 для поиска оптимального веса
while r2_idw_tr_shrunk_old <= r2_idw_tr_shrunk:
    power += power_increment # Увеличиваем степень на 1 шаг
    iterations +=1 # Увеличиваем счётчик проходов цикла
    r2_idw_tr_shrunk_old = r2_idw_tr_shrunk # Запоминаем предыдущее значение R2

    # Создаём y_predict_idw_tr_shrunk по индексам в массиве x_train_shrunk
    # используем функцию inverse_distance_avg
    # Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком
    y_predict_idw_tr_shrunk = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param_=PARAMETER31,
                             station_=df_test.keys()[y][len(PARAMETER31)+1:],
                             df_dists_= df_station_dists,
                             power_=power)
        for x, y in x_train_idw_shrunk
    ]

    # Расчитываем метрики качества
    max_e_idw_tr_shrunk = max_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    mae_idw_tr_shrunk = mean_absolute_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    mse_idw_tr_shrunk = mean_squared_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    rmse_idw_tr_shrunk = mean_squared_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk, squared=False)
    r2_idw_tr_shrunk = r2_score(y_train_idw_shrunk, y_predict_idw_tr_shrunk)

    if r2_idw_tr_shrunk > r2_idw_tr_shrunk_old:
        best_power_idw_tr_shrunk = power
        best_max_e_idw_tr_shrunk = max_e_idw_tr_shrunk
        best_mae_idw_tr_shrunk = mae_idw_tr_shrunk
        best_mse_idw_tr_shrunk = mse_idw_tr_shrunk
        best_rmse_idw_tr_shrunk = rmse_idw_tr_shrunk
        best_r2_idw_tr_shrunk = r2_idw_tr_shrunk

    # замер времени:
    chk_time = time.time()
    elapsed_time = chk_time - start_time
    time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

print(f'Elapsed time={time_formatted}\n'
```

```
f'Текущие значения: iterations={iterations}, power={power},\n'
f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_tr_shrunk:.7f}\n'
f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_tr_shrunk:.7f}\n'
f'Средний квадрат ошибки (MSE) IDW = {mse_idw_tr_shrunk:.7f}\n'
f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_tr_shrunk:.7f}\n'
f'Коэффициент детерминации (R2) IDW = {r2_idw_tr_shrunk:.7f}\n'
)
print(f'ЛУЧШИЕ значения: power={best_power_idw_tr_shrunk},\n'
f'Максимальная ошибка (MAX_E) IDW = {best_max_e_idw_tr_shrunk:.7f}\n'
f'Средняя абсолютная ошибка (MAE) IDW = {best_mae_idw_tr_shrunk:.7f}\n'
f'Средний квадрат ошибки (MSE) IDW = {best_mse_idw_tr_shrunk:.7f}\n'
f'Средняя квадратическая ошибка (RMSE) IDW = {best_rmse_idw_tr_shrunk:.7f}\n'
f'Коэффициент детерминации (R2) IDW = {best_r2_idw_tr_shrunk:.7f}'
```

НОВЫЙ ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:

Elapsed time=00:00:02

Текущие значения: iterations=1, power=2.0,  
Максимальная ошибка (MAX\_E) IDW = 8.8604790  
Средняя абсолютная ошибка (MAE) IDW = 1.0555680  
Средний квадрат ошибки (MSE) IDW = 2.1544735  
Средняя квадратическая ошибка (RMSE) IDW = 1.4678125  
Коэффициент детерминации (R2) IDW = 0.9817233

Elapsed time=00:00:05

Текущие значения: iterations=2, power=2.25,  
Максимальная ошибка (MAX\_E) IDW = 8.8246309  
Средняя абсолютная ошибка (MAE) IDW = 1.0534000  
Средний квадрат ошибки (MSE) IDW = 2.1469033  
Средняя квадратическая ошибка (RMSE) IDW = 1.4652315  
Коэффициент детерминации (R2) IDW = 0.9817875

Elapsed time=00:00:08

Текущие значения: iterations=3, power=2.5,  
Максимальная ошибка (MAX\_E) IDW = 8.7906343  
Средняя абсолютная ошибка (MAE) IDW = 1.0513751  
Средний квадрат ошибки (MSE) IDW = 2.1419043  
Средняя квадратическая ошибка (RMSE) IDW = 1.4635246  
Коэффициент детерминации (R2) IDW = 0.9818299

Elapsed time=00:00:11

Текущие значения: iterations=4, power=2.75,  
Максимальная ошибка (MAX\_E) IDW = 8.7586315  
Средняя абсолютная ошибка (MAE) IDW = 1.0496911  
Средний квадрат ошибки (MSE) IDW = 2.1393695  
Средняя квадратическая ошибка (RMSE) IDW = 1.4626584  
Коэффициент детерминации (R2) IDW = 0.9818514

Elapsed time=00:00:14

Текущие значения: iterations=5, power=3.0,  
Максимальная ошибка (MAX\_E) IDW = 8.7287271  
Средняя абсолютная ошибка (MAE) IDW = 1.0484586  
Средний квадрат ошибки (MSE) IDW = 2.1391280  
Средняя квадратическая ошибка (RMSE) IDW = 1.4625758  
Коэффициент детерминации (R2) IDW = 0.9818535

Elapsed time=00:00:18

Текущие значения: iterations=6, power=3.25,

Максимальная ошибка (MAX\_E) IDW = 8.7009844  
Средняя абсолютная ошибка (MAE) IDW = 1.0478186  
Средний квадрат ошибки (MSE) IDW = 2.1409602  
Средняя квадратическая ошибка (RMSE) IDW = 1.4632021  
Коэффициент детерминации (R2) IDW = 0.9818379

ЛУЧШИЕ значения: power=3.0,  
Максимальная ошибка (MAX\_E) IDW = 8.7287271  
Средняя абсолютная ошибка (MAE) IDW = 1.0484586  
Средний квадрат ошибки (MSE) IDW = 2.1391280  
Средняя квадратическая ошибка (RMSE) IDW = 1.4625758  
Коэффициент детерминации (R2) IDW = 0.9818535

Опробуем новое значение лучшей степени для IDW на сокращённом валидационном датасете

```
In [80]: # Создаём y_predict_idw_vld_shrunk по индексам в x_test_shrunk
# используем функцию inverse_distance_avg
# Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком

y_predict_idw_vld_shrunk = [
    inverse_distance_avg(row=df_test.iloc[x],
        param=PARAMETER31,
        station=df_test.keys()[y][len(PARAMETER31)+1:],
        df_dists=df_station_dists,
        power=best_power_idw_tr_shrunk) # берём лучшее значение степени, полученное из кода выше
    for x, y in x_test_idw_shrunk
]
# y_predict_idw_vld_shrunk

max_e_idw_vld_shrunk = max_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
mae_idw_vld_shrunk = mean_absolute_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
mse_idw_vld_shrunk = mean_squared_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
rmse_idw_vld_shrunk = mean_squared_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk, squared=False)
r2_idw_vld_shrunk = r2_score(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
print(f'ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld_shrunk:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld_shrunk:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld_shrunk:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld_shrunk:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld_shrunk:.7f}')
)
```

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:

Максимальная ошибка (MAX\_E) IDW = 6.1913436  
Средняя абсолютная ошибка (MAE) IDW = 1.0646196  
Средний квадрат ошибки (MSE) IDW = 2.0863885  
Средняя квадратическая ошибка (RMSE) IDW = 1.4444336  
Коэффициент детерминации (R2) IDW = 0.9831432

### Метрики обучающего и валидационного датасетов для совместной работы двух моделей

Данные работы моделей на своей части обучающего датасета у нас есть. Подсчитаем метрики для общего датасета

```
In [81]: # Восстановим y_predict для лучшего R2 IDW
y_predict_idw_tr_shrunk = [
    inverse_distance_avg(row=df_test.iloc[x],
                          param_=PARAMETER31,
                          station_=df_test.keys()[y][len(PARAMETER31)+1:],
                          df_dists_=df_station_dists,
                          power_=best_power_idw_tr_shrunk)
    for x, y in x_train_idw_shrunk
]
```

```
In [82]: # последовательно соединим датасеты для кrigинга (там, где есть предсказания) и для IDW
x_train_2 = np.append(x_train[~np.isnan(y_predict_kriging_tr)], x_train_idw_shrunk)
y_train_2 = np.append(y_train_kriging_shrunk, y_train_idw_shrunk)
y_predict_tr_2 = np.append(y_predict_kriging_tr_shrunk, y_predict_idw_tr_shrunk)
x_test_2 = np.append(x_test[~np.isnan(y_predict_kriging_vld)], x_test_idw_shrunk)
y_test_2 = np.append(y_test_kriging_shrunk, y_test_idw_shrunk)
y_predict_vld_2 = np.append(y_predict_kriging_vld_shrunk, y_predict_idw_vld_shrunk)
```

```
In [83]: # Расчитываем метрики качества
max_e_2_tr = max_error(y_train_2, y_predict_tr_2)
mae_2_tr = mean_absolute_error(y_train_2, y_predict_tr_2)
mse_2_tr = mean_squared_error(y_train_2, y_predict_tr_2)
rmse_2_tr = mean_squared_error(y_train_2, y_predict_tr_2, squared=False)
r2_2_tr = r2_score(y_train_2, y_predict_tr_2)

max_e_2_vld = max_error(y_test_2, y_predict_vld_2)
mae_2_vld = mean_absolute_error(y_test_2, y_predict_vld_2)
mse_2_vld = mean_squared_error(y_test_2, y_predict_vld_2)
rmse_2_vld = mean_squared_error(y_test_2, y_predict_vld_2, squared=False)
r2_2_vld = r2_score(y_test_2, y_predict_vld_2)
```

In [84]:

```
print(f'ОБЩИЙ ОБУЧАЮЩИЙ ДАТАСЕТ, Кригинг + IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_2_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_2_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_2_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_2_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_2_tr:.7f}\n'
    )
print(f'ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_2_vld:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_2_vld:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_2_vld:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_2_vld:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_2_vld:.7f}\n'
    )
print(f'Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld:.7f}\n'
    )
```

ОБЩИЙ ОБУЧАЮЩИЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX\_E) IDW = 9.6527528  
Средняя абсолютная ошибка (MAE) IDW = 0.7857760  
Средний квадрат ошибки (MSE) IDW = 1.3380625  
Средняя квадратическая ошибка (RMSE) IDW = 1.1567465  
Коэффициент детерминации (R2) IDW = 0.9883096

ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX\_E) IDW = 9.1718611  
Средняя абсолютная ошибка (MAE) IDW = 0.7936315  
Средний квадрат ошибки (MSE) IDW = 1.3581992  
Средняя квадратическая ошибка (RMSE) IDW = 1.1654180  
Коэффициент детерминации (R2) IDW = 0.9879991

Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:

Максимальная ошибка (MAX\_E) IDW = 9.6052652  
Средняя абсолютная ошибка (MAE) IDW = 0.8256745  
Средний квадрат ошибки (MSE) IDW = 1.4599404  
Средняя квадратическая ошибка (RMSE) IDW = 1.2082799  
Коэффициент детерминации (R2) IDW = 0.9871001

## ВЫВОД

Там, где с помощью кригинга удаётся получить предсказания, он превосходит по метрикам качества модель IDW. При этом сочетание этих двух моделей (применение IDW там, где кригинг оказывается бессильным) даёт в целом лучшее качество предсказания, чем только модель IDW. Поэтому, ниже применим комбинацию этих двух моделей к уже реальному датасету..

### 3.1.4. Применение выбранных моделей для исправления, восстановления данных и получения предиктов показателя температуры Т для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

```
In [85]: # Добавим в df_tmp31 столбцы для будущих значений для Чашниково и центральной точки, заполним их NaN  
# - это необходимо, чтобы вычислить значения для архивов  
# Создадим столбцы col_name со значениями np.nan  
# Переименуем столбец PARAMETER31#Chashnikovo и PARAMETER31#Rfrnce_point  
df_tmp31 = (df_tmp31.  
    assign(col_name1 = np.nan,  
          col_name2 = np.nan).  
    rename(columns={"col_name1": PARAMETER31+'#'+ 'Chashnikovo',  
              "col_name2": PARAMETER31+'#'+ 'Rfrnce_point'}))  
)  
  
df_tmp31.sample(3, random_state=56)
```

	T#V_Volochev	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2014-02-22 03:00:00	-3.5	-2.8	-3.0	-4.3	-4.9	-4.8	-2.9	-2.4	-3.4	-3.6	-2
2015-05-16 03:00:00	8.6	8.1	10.8	8.0	8.3	10.9	7.7	7.9	8.4	8.8	8
2020-06-18 18:00:00	28.7	28.8	24.6	28.3	27.7	27.5	30.1	29.1	29.6	25.8	30

```
In [86]: # Добавим в архив параметров Т столбец для будущей центральной точки, заполним их NaN  
# Создадим столбцы col_name со значениями np.nan
```

```

# Переименуем столбец PARAMETER31#Chashnikovo и PARAMETER31#Rfrnce_point
dict_df_parameters['df_'+PARAMETER31] = (dict_df_parameters['df_'+PARAMETER31].
                                         assign(col_name1 = np.nan,
                                                col_name2 = np.nan).
                                         rename(columns={"col_name1": PARAMETER31+'#'+ 'Chashnikovo',
                                                        "col_name2": PARAMETER31+'#'+ 'Rfrnce_point'})
                                         )
                                         )

dict_df_parameters['df_'+PARAMETER31].sample(3, random_state=56)

```

Out[86]:

	T#V_Volochek	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomin
2014-02-22 03:00:00	-3.5	-2.8	-3.0	-4.3	-4.9	-4.8	-2.9	-2.4	-3.4	-3.6	-2
2015-05-16 03:00:00	8.6	8.1	10.8	8.0	8.3	10.9	7.7	7.9	8.4	8.8	8
2020-06-18 18:00:00	28.7	28.8	24.6	28.3	27.7	27.5	30.1	29.1	29.6	25.8	30

In [87]: # Добавим в архив метеостанций df Чашниково и df для центральной точки,  
# Создадим в нём столбец с называнием PARAMETER31

```

dict_df_locations['df_Chashnikovo'] = (pd.DataFrame(index=df_tmp31.index).
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER31})
                                         )
                                         )

dict_df_locations['df_Rfrnce_point'] = (pd.DataFrame(index=df_tmp31.index).
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER31})
                                         )
                                         )

dict_df_locations['df_Chashnikovo'].sample(3, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(3, random_state=56)

```

Out[87]:

T

	T
2014-02-22 03:00:00	NaN
2015-05-16 03:00:00	NaN
2020-06-18 18:00:00	NaN

Out[87]:

T

	T
2014-02-22 03:00:00	NaN
2015-05-16 03:00:00	NaN
2020-06-18 18:00:00	NaN

## Перенесение уже исправленных сплошных NaN в архивы

In [88]:

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER31] = df_tmp31.copy(deep=True)

# Перенесём уже исправленные значения в dict_df_locations, оставшиеся NaN исправим ниже
for name_df in dict_df_locations.keys():
    dict_df_locations[name_df].loc[:, PARAMETER31] = df_tmp31.loc[:, PARAMETER31 + '#' + name_df[3:]]
```

In [89]:

```
# Подсчитаем количество строк со "сплошными" NaN dict_df_parameters['df_'+PARAMETER31]
# построчно подсчитаем сумму количества NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(
    dict_df_parameters['df_'+PARAMETER31]
    .apply(lambda x: sum(x.isna()), axis=1)==len(dict_df_parameters['df_'+PARAMETER31].keys()))
)

print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 1

In [90]:

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp31
# построчно подсчитаем сумму количества NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(
    df_tmp31
    .apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp31.keys()))
```

```
)
```

```
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

```
Количество строк со сплошными NaN равно 1
```

### Частичное заполнение оставшихся NaN расчётыми значениями с помощью кригинга

Вариограммы и модель кригинга имеют следующее свойство. Модель не всегда может рассчитать сразу все значения в заданном поле из-за нехватки данных для выявления полувариаций. При этом она может рассчитать часть из них. В таком случае, при еще одном вызове модели (.transform), в неё будут включены вновь рассчитанные данные, и модель сможет рассчитать еще часть недостающих значений. Поэтому применим модель кригинга в цикле, пока количество оставшихся в датафрейме NaN перестанет уменьшаться. Этим будут значения, которые модель уже никак не сможет рассчитать. Их можно будет восстановить методом IDW.

```
In [91]: start_time = time.time() # для замера времени выполнения кода

# Подсчитаем количество NaN в df_tmp31 и присвоим их переменной old_nan_count
# np.isnan(df_tmp31).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
new_nan_count = np.sum(np.isnan(df_tmp31).sum()) # результат всегда будет >= 0
# Определим начальное значение для переменной для обновления количества оставшихся NaN
old_nan_count = new_nan_count
counter = 0 # определим счётчик

# заменим значения в архивах, на исправленные и вычисленные из данных в df_tmp31
# используем функцию row_nan_kriging_correct
# функция настроена таким образом, что при возникновении ошибки, связной с недостаточностью данных,
# осуществляется выход из функции без возврата каких бы то ни было значений.
while (old_nan_count != new_nan_count) or (counter == 0):
    counter += 1
    print (f'\nИтерация № {counter}: осталось NaN: {new_nan_count}')

    # Построчно применяем функцию обработки NaN
    df_tmp31.apply(lambda x: row_nan_kriging_correct(row_=x,
                                                       name_param_=PARAMETER31
                                                       ),
                  axis=1
                 )
    old_nan_count = new_nan_count # сохраняем прежнее количество NaN в old_nan_count
    new_nan_count = np.sum(np.isnan(df_tmp31).sum()) # подсчитаем оставшиеся количество NaN

chk_time = time.time()
```

```
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f'Elapsed time={time_formatted}\n')
```

Итерация № 1: осталось NaN: 156861

Out[91]:

2022-06-09 21:00:00	None
2022-06-09 18:00:00	None
2022-06-09 15:00:00	None
2022-06-09 12:00:00	None
2022-06-09 09:00:00	None
...	
2005-02-01 12:00:00	None
2005-02-01 09:00:00	None
2005-02-01 06:00:00	None
2005-02-01 03:00:00	None
2005-02-01 00:00:00	None
Length:	50704, dtype: object

Итерация № 2: осталось NaN: 6390

Out[91]:

2022-06-09 21:00:00	None
2022-06-09 18:00:00	None
2022-06-09 15:00:00	None
2022-06-09 12:00:00	None
2022-06-09 09:00:00	None
...	
2005-02-01 12:00:00	None
2005-02-01 09:00:00	None
2005-02-01 06:00:00	None
2005-02-01 03:00:00	None
2005-02-01 00:00:00	None
Length:	50704, dtype: object

Итерация № 3: осталось NaN: 2152

Out[91]:

2022-06-09 21:00:00	None
2022-06-09 18:00:00	None
2022-06-09 15:00:00	None
2022-06-09 12:00:00	None
2022-06-09 09:00:00	None
...	
2005-02-01 12:00:00	None
2005-02-01 09:00:00	None
2005-02-01 06:00:00	None
2005-02-01 03:00:00	None
2005-02-01 00:00:00	None
Length:	50704, dtype: object

Итерация № 4: осталось NaN: 1708

```
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 5: осталось NaN: 1589  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 6: осталось NaN: 1535  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 7: осталось NaN: 1495
```

```
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 8: осталось NaN: 1478  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 9: осталось NaN: 1467  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 10: осталось NaN: 1457
```

```
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 11: осталось NaN: 1454  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 12: осталось NaN: 1449  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 13: осталось NaN: 1446
```

```
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 14: осталось NaN: 1443  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 15: осталось NaN: 1441  
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 16: осталось NaN: 1439
```

```
Out[91]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Elapsed time=00:19:54
```

Подсчитаем конечное количество оставшихся *NaN*, которые не могут быть рассчитаны моделью кригинга

```
In [92]: # np.isnan(df_tmp31).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp31).sum())
```

```
Out[92]: 1439
```

Восстановим оставшиеся значения через модель IDW. Используем для этого степень, полученную на обучающем датасете при совместном использовании модели IDW и модели кригинга.

```
# Произведём вычисление отсутствующих значений в df_tmp31 через модель IDW.  
# Заменим соответствующие значения в архивах на вновь вычисленные.  
# используем функцию row_nan_idw_correct  
  
start_time = time.time() # для замера времени выполнения кода  
  
# Построчно применяем функцию обработки NaN  
df_tmp31.apply(lambda x: row_nan_idw_correct(row_=x,  
                                              name_param_=PARAMETER31,  
                                              power_=best_power_idw_tr_shrunk),  
               axis=1  
)  
  
chk_time = time.time()  
elapsed_time = chk_time - start_time
```

```
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f'Elapsed time={time_formatted}\n')
```

Out[93]:

```
2022-06-09 21:00:00      None
2022-06-09 18:00:00      None
2022-06-09 15:00:00      None
2022-06-09 12:00:00      None
2022-06-09 09:00:00      None
...
2005-02-01 12:00:00      None
2005-02-01 09:00:00      None
2005-02-01 06:00:00      None
2005-02-01 03:00:00      None
2005-02-01 00:00:00      None
Length: 50704, dtype: object
Elapsed time=00:00:15
```

Подсчитаем окончательное количество NaN

In [94]:

```
# np.isnan(df_tmp31).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
np.sum(np.isnan(df_tmp31).sum())
```

Out[94]:

```
14
```

Всё корректно. 14 NaN находятся в самом первом моменте наблюдения. Он пустой.

Выведем, рандомные строки из df\_tmp31

In [95]:

```
df_tmp31.sample(7)
```

Out[95]:

	T#V_Volochev	T#Staritsa	T#Kashyn	T#Tver	T#Klin	T#Dmitrov	T#Volokolamsk	T#Mozhaisk	T#N_Jerusalem	T#Nemchinovka	T#Naro_Fomir
<b>2019-01-27 03:00:00</b>	-10.700000	-10.3	-9.800000	-9.8	-9.2	-9.7	-9.2	-8.8	-9.3	-8.600000	-
<b>2011-12-05 06:00:00</b>	1.200000	4.6	2.700000	4.5	4.6	3.9	4.6	5.0	4.8	4.320831	
<b>2007-11-17 18:00:00</b>	-6.050249	-5.4	-5.958933	-7.1	-6.0	-5.4	-3.8	-2.9	-3.1	-3.119970	-
<b>2017-04-24 12:00:00</b>	6.600000	5.2	3.200000	5.6	4.1	6.2	3.8	6.9	6.0	7.100000	
<b>2006-02-03 21:00:00</b>	-28.600000	-26.5	-24.200000	-25.0	-30.0	-23.2	-26.6	-25.1	-26.2	-24.655324	-2
<b>2016-12-11 00:00:00</b>	-6.600000	-6.6	-5.700000	-6.9	-6.5	-6.3	-6.3	-6.1	-6.3	-6.300000	-
<b>2011-04-13 12:00:00</b>	1.400000	0.6	1.300000	1.0	1.7	1.8	1.3	1.9	2.2	2.245325	

### 3.1.5. Визуализация графиков температуры Т для условной метеостанции Чашниково

In [96]:

```
# Построим список годов для графиков
list_years = df_tmp31.index.year.unique().tolist()
```

In [97]:

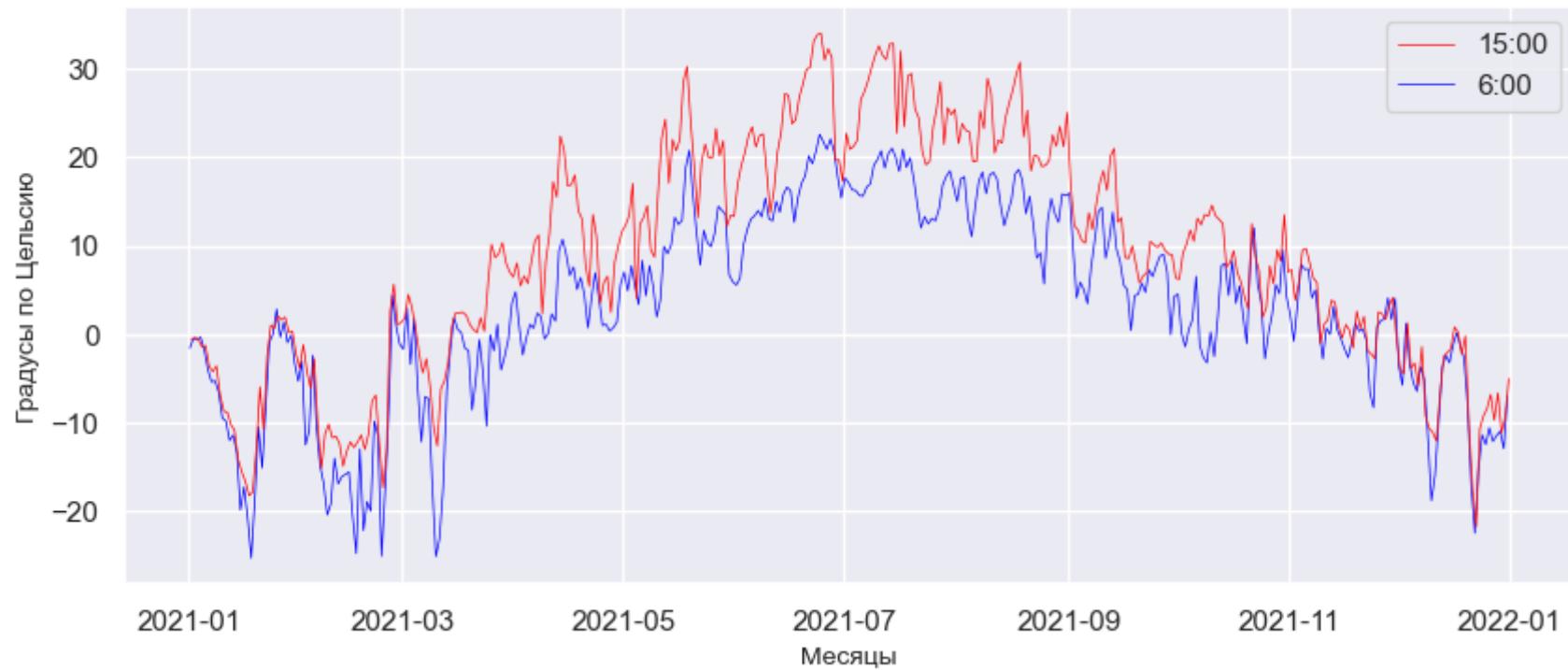
```
# Определим часы момента наблюдения для построения графиков
plot_hour1 = 6
plot_hour2 = 15
# Строим график в цикле для каждого года
for year in list_years:
    data1 = dict_df_parameters['df_T'][PARAMETER31+"#"+"Chashnikovo"]\
```

```
[(dict_df_parameters['df_T'].index.year == year) & (dict_df_parameters['df_T'].index.hour == plot_hour1)]\n\n    data2 = dict_df_parameters['df_T'][PARAMETER31+"#"+"Chashnikovo"]\\n\n    [(dict_df_parameters['df_T'].index.year == year) & (dict_df_parameters['df_T'].index.hour == plot_hour2)]\n\n    fig, ax = plt.subplots(figsize=(10, 4))\n    g1 = sns.lineplot(data=data1,\n                       color='blue',\n                       linewidth=0.5,\n                       ax=ax)\n    g2 = sns.lineplot(data=data2,\n                       color='red',\n                       linewidth=0.5,\n                       ax=ax)\n\n    # Добавляем легенду\n    blue_line = mlines.Line2D([], [], color='blue', label=f'{plot_hour1}:00', linewidth=0.5)\n    red_line = mlines.Line2D([], [], color='red', label=f'{plot_hour2}:00', linewidth=0.5)\n\n    dummy = ax.legend(handles=[red_line, blue_line])\n\n    dummy = ax.set_ylabel('Градусы по Цельсию', size=10)\n    dummy = ax.set_xlabel('Месяцы', size=10)\n    dummy = plt.title(f'Чашниково: Ежедневная динамика параметра {PARAMETER31} '\n                      f'на {plot_hour1} и {plot_hour2} часов, {year} год')\n\nplt.show()
```

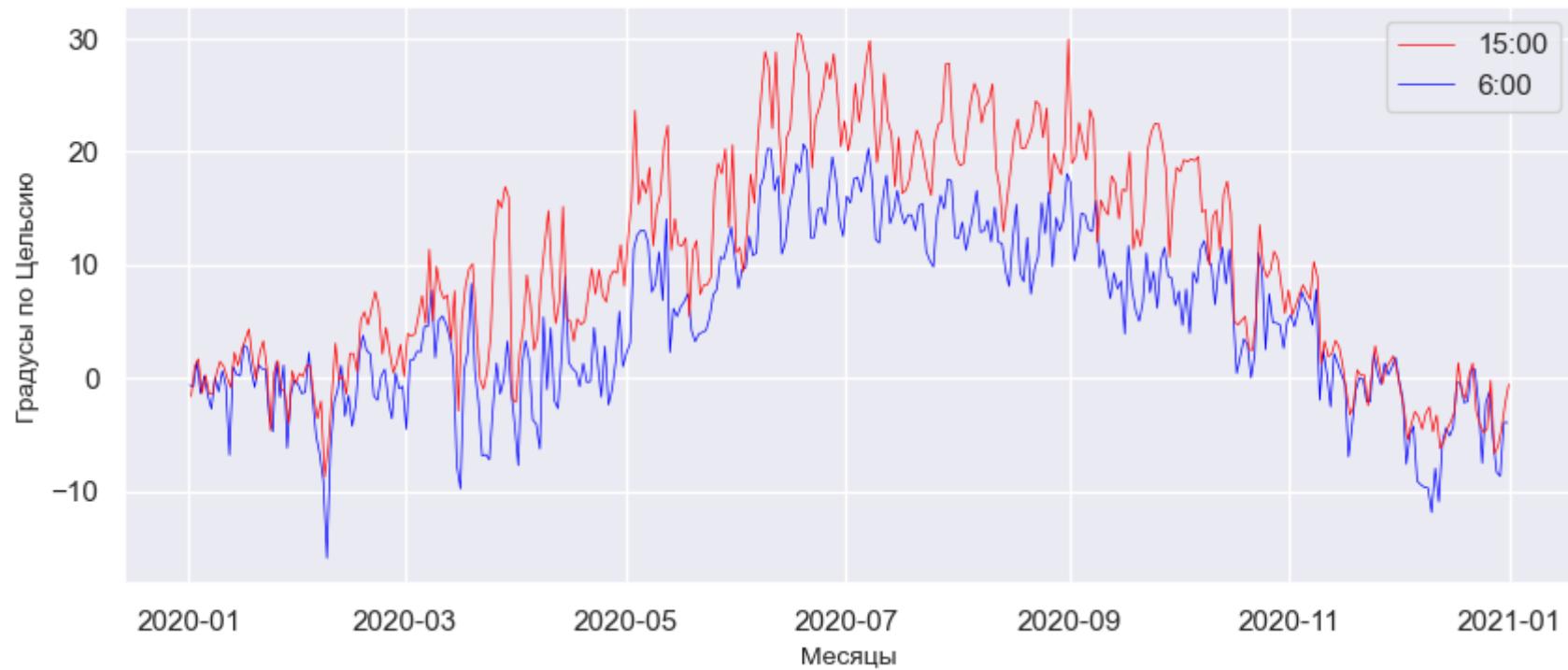
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2022 год



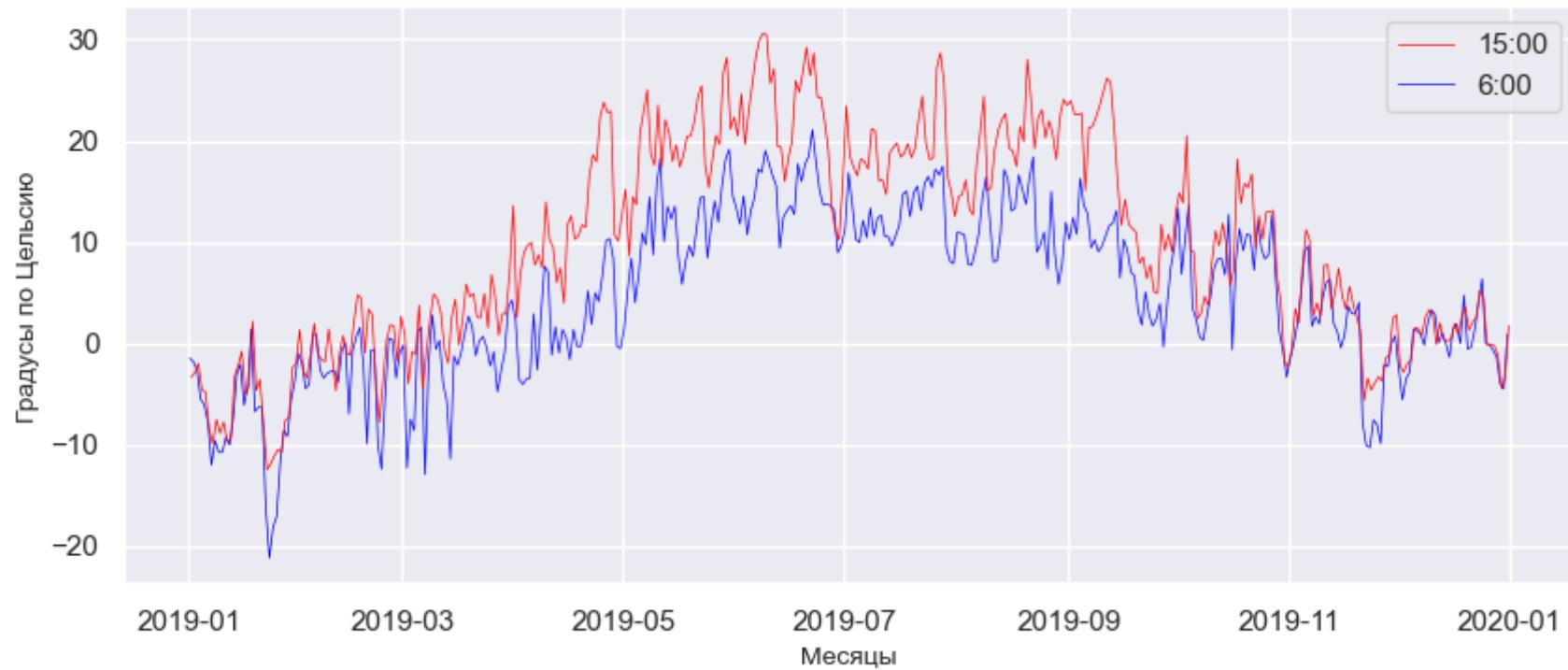
### Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2021 год



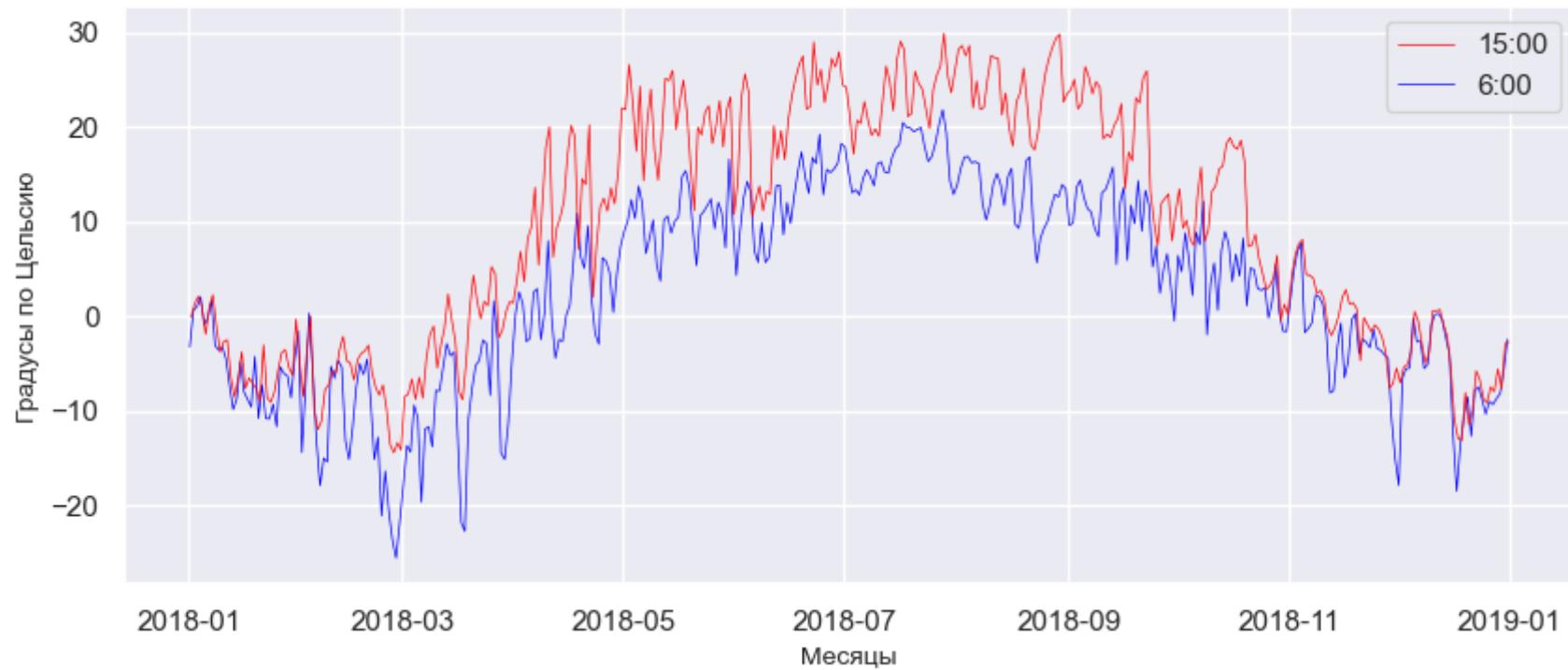
### Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2020 год



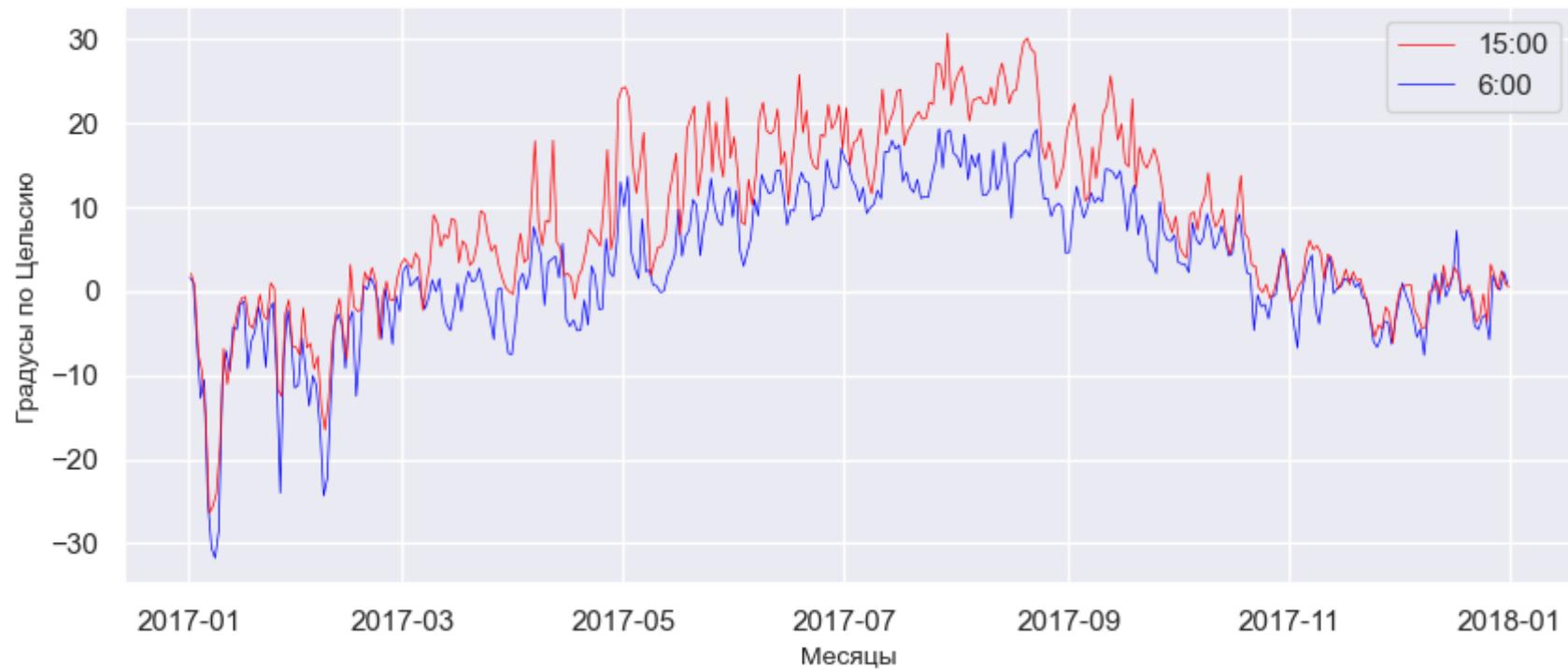
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2019 год



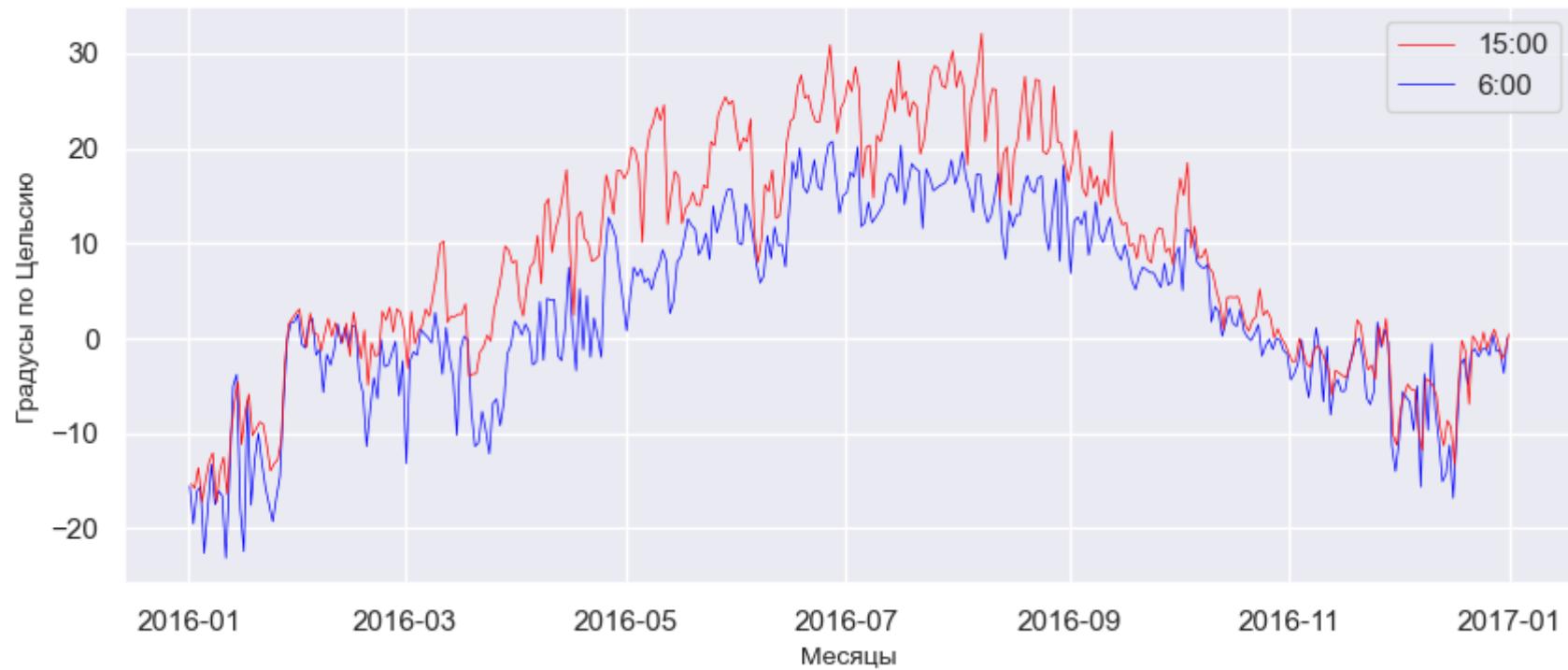
### Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2018 год



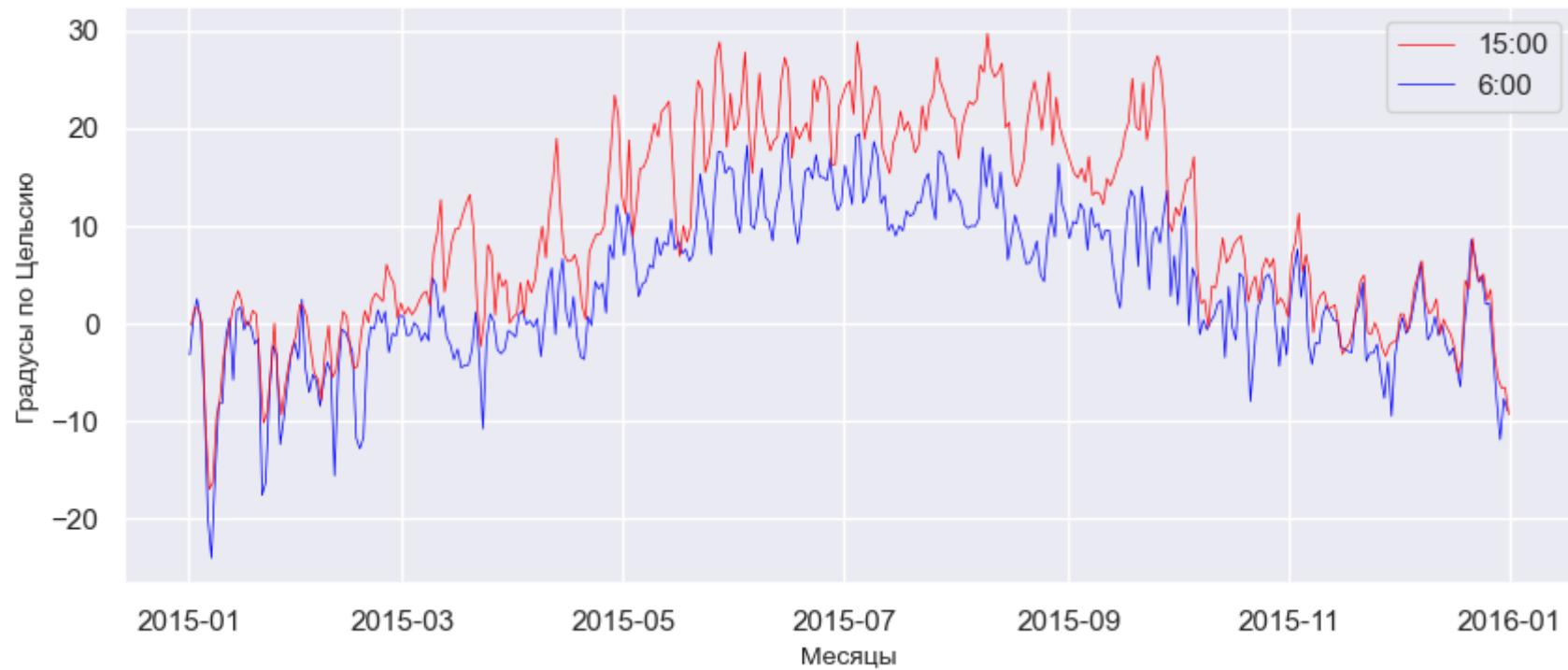
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2017 год



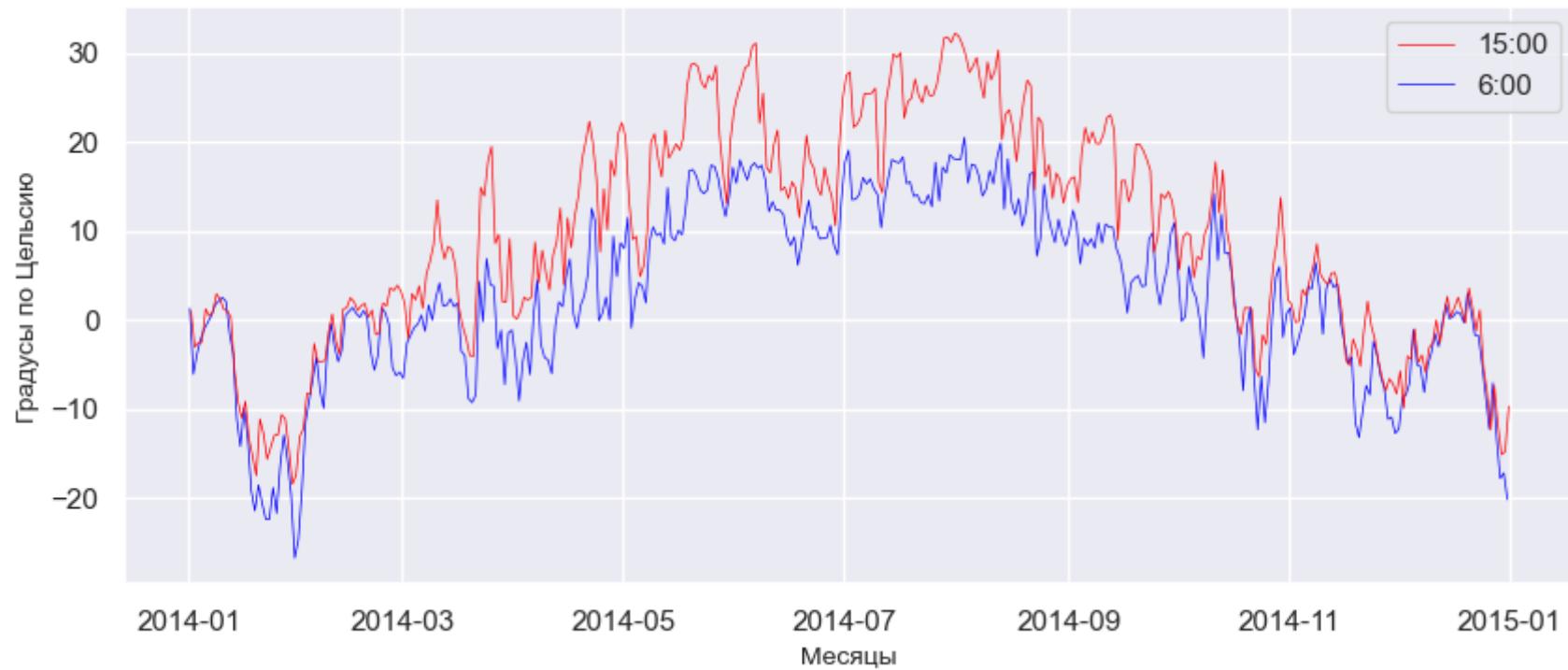
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2016 год



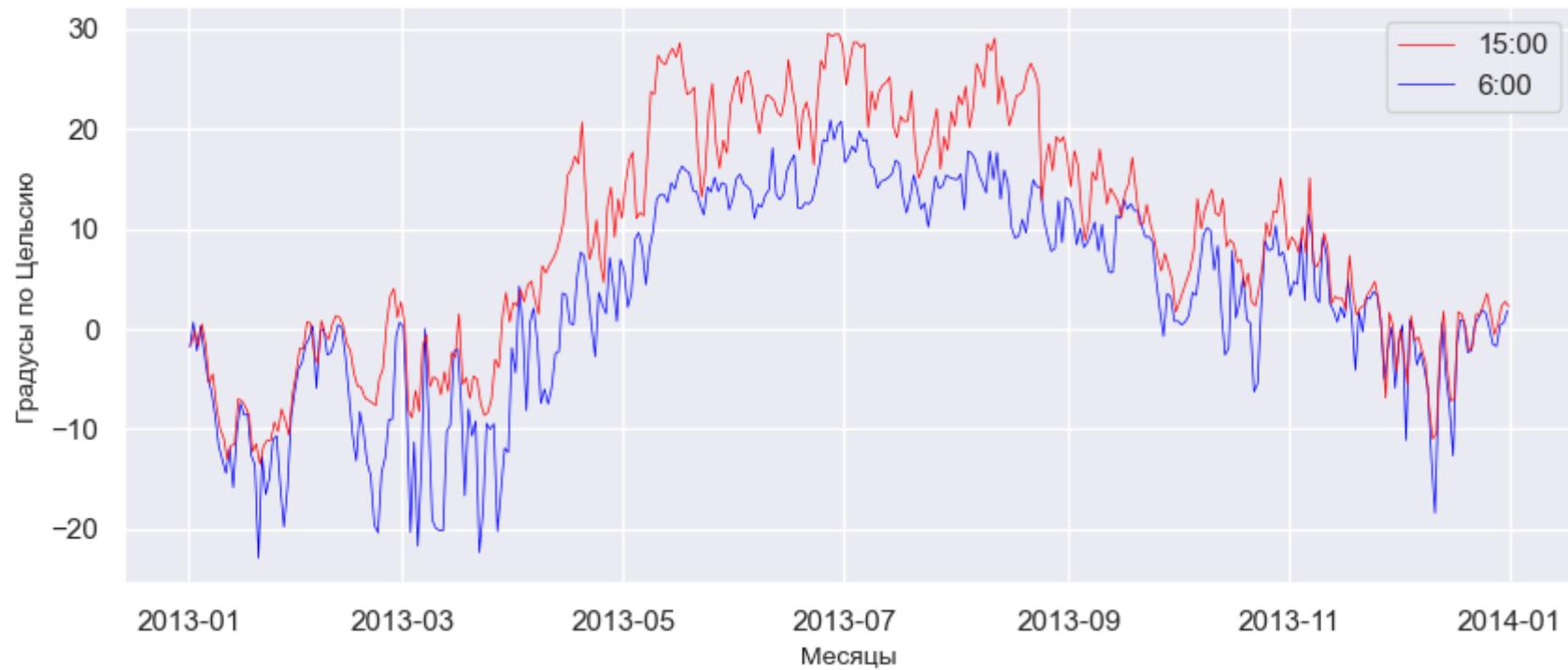
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2015 год



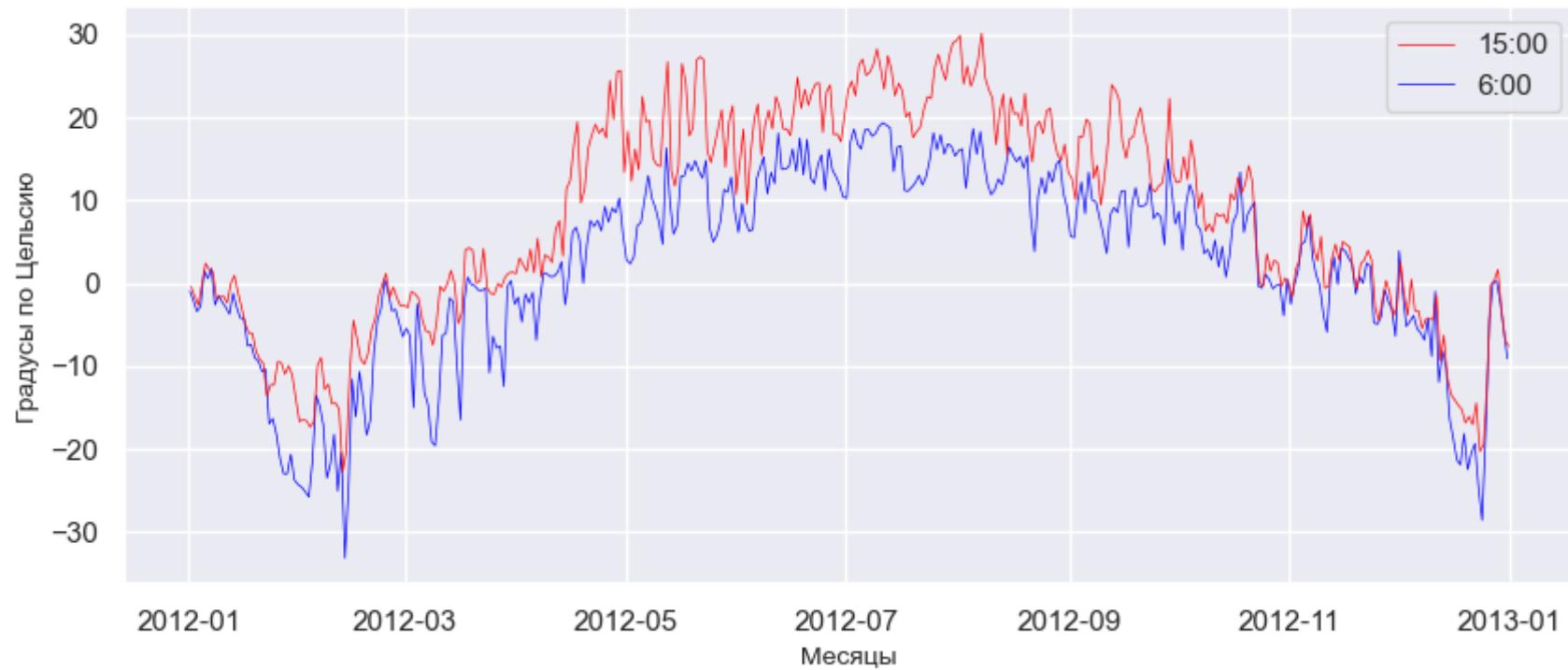
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2014 год



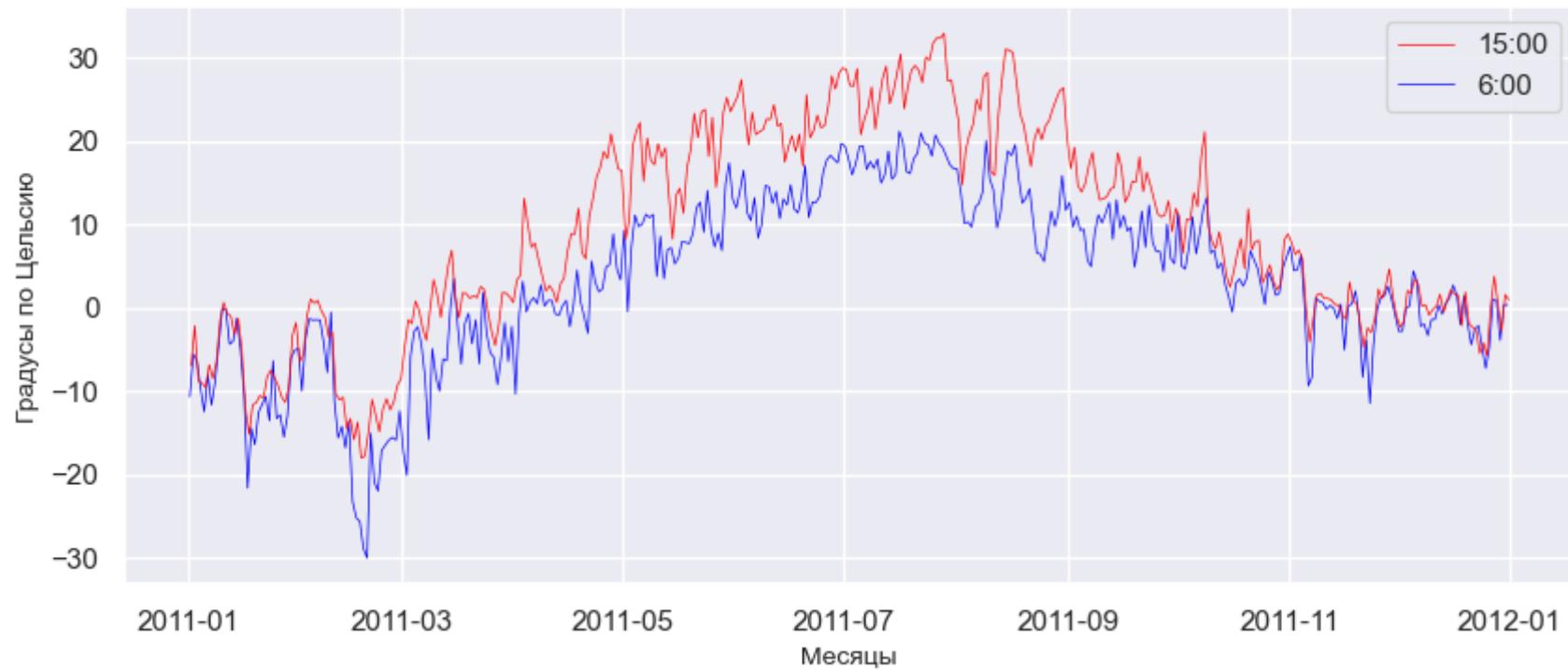
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2013 год



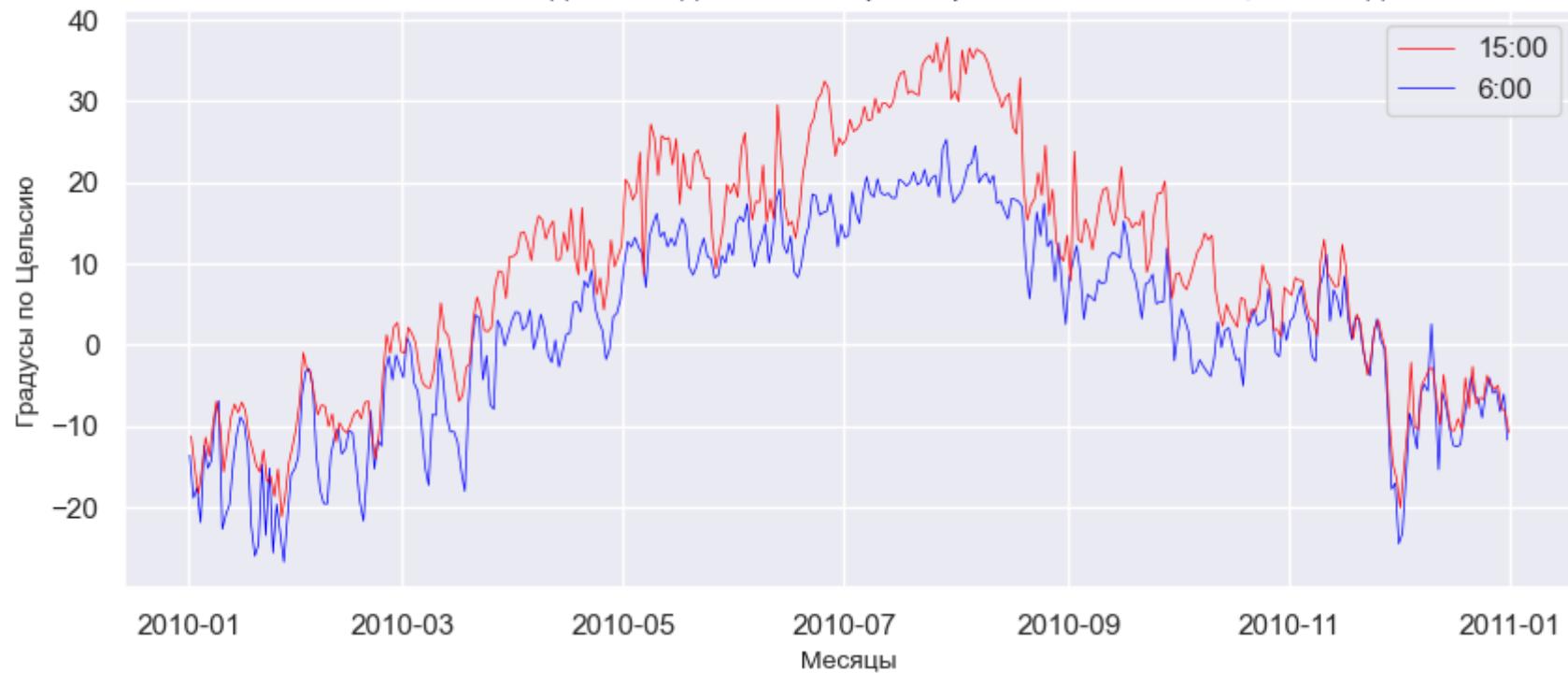
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2012 год



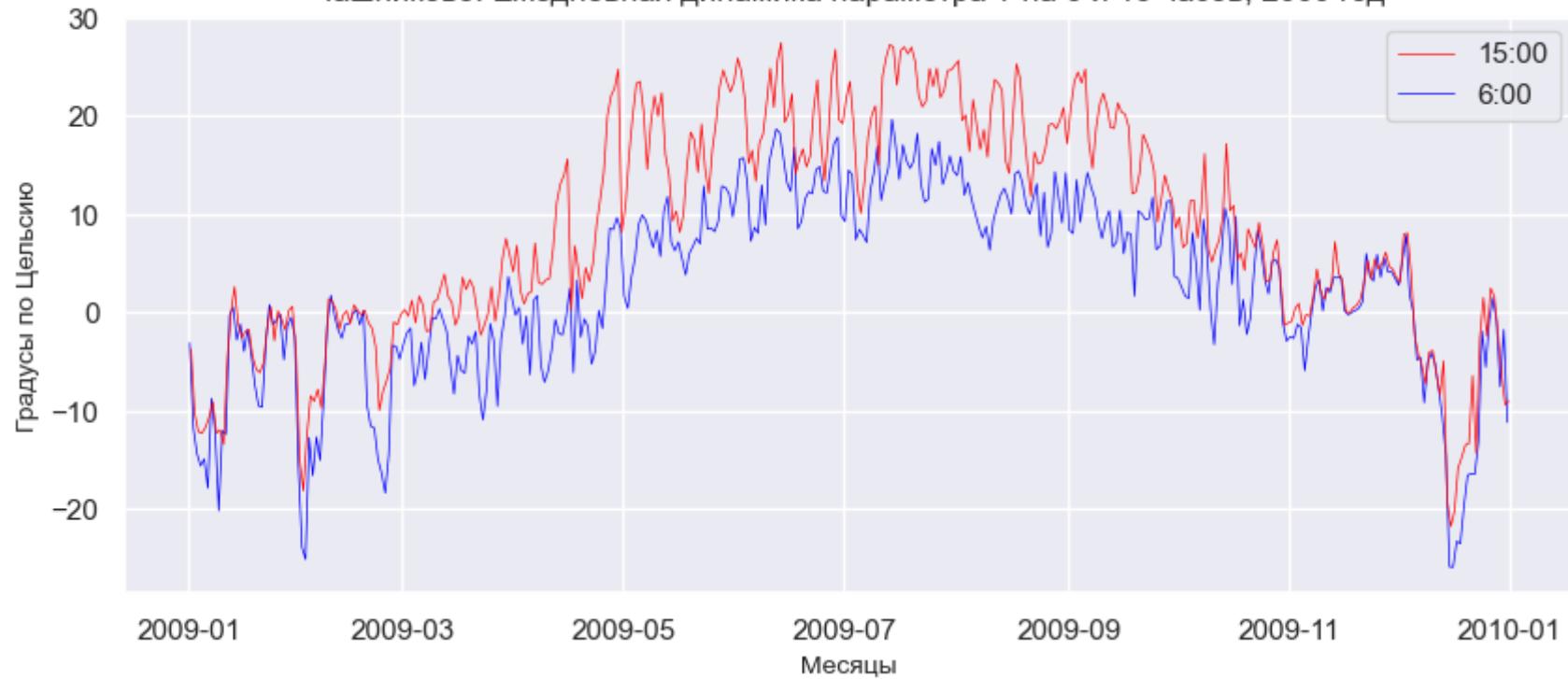
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2011 год



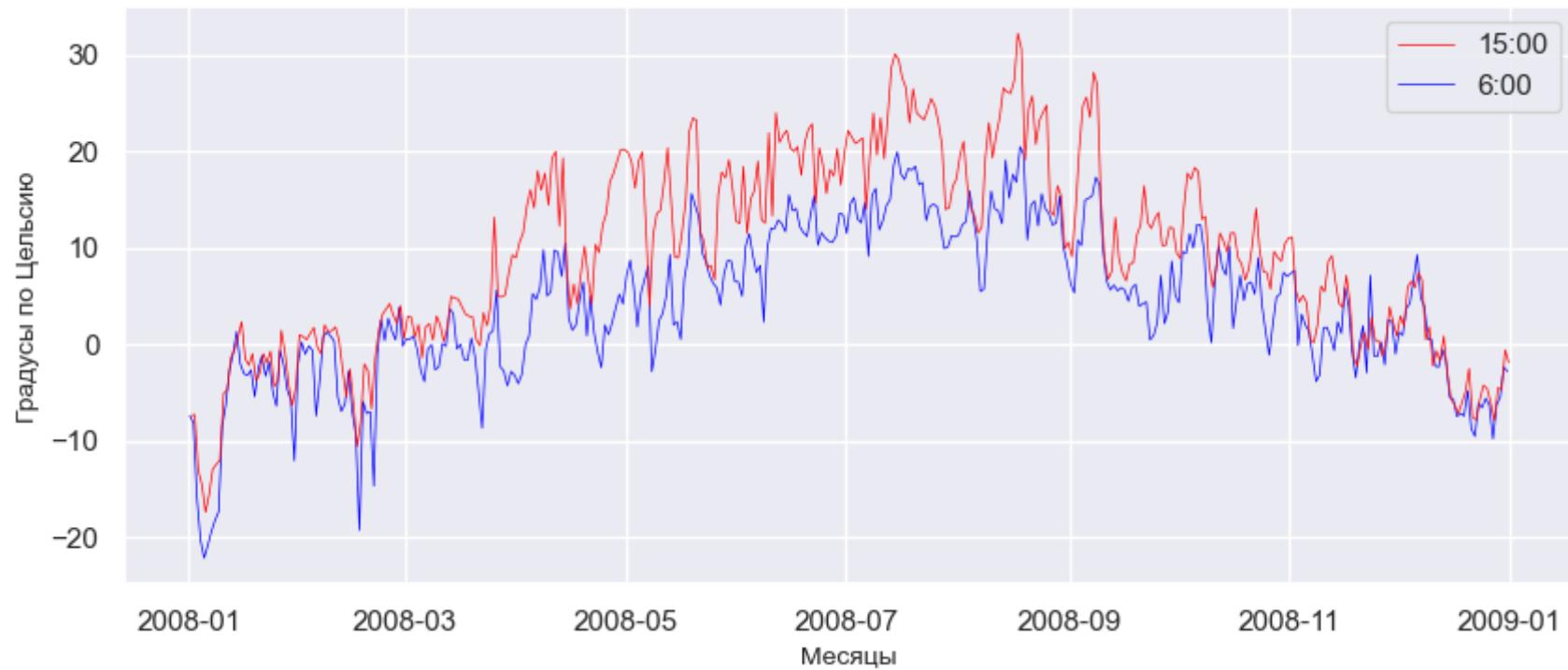
### Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2010 год



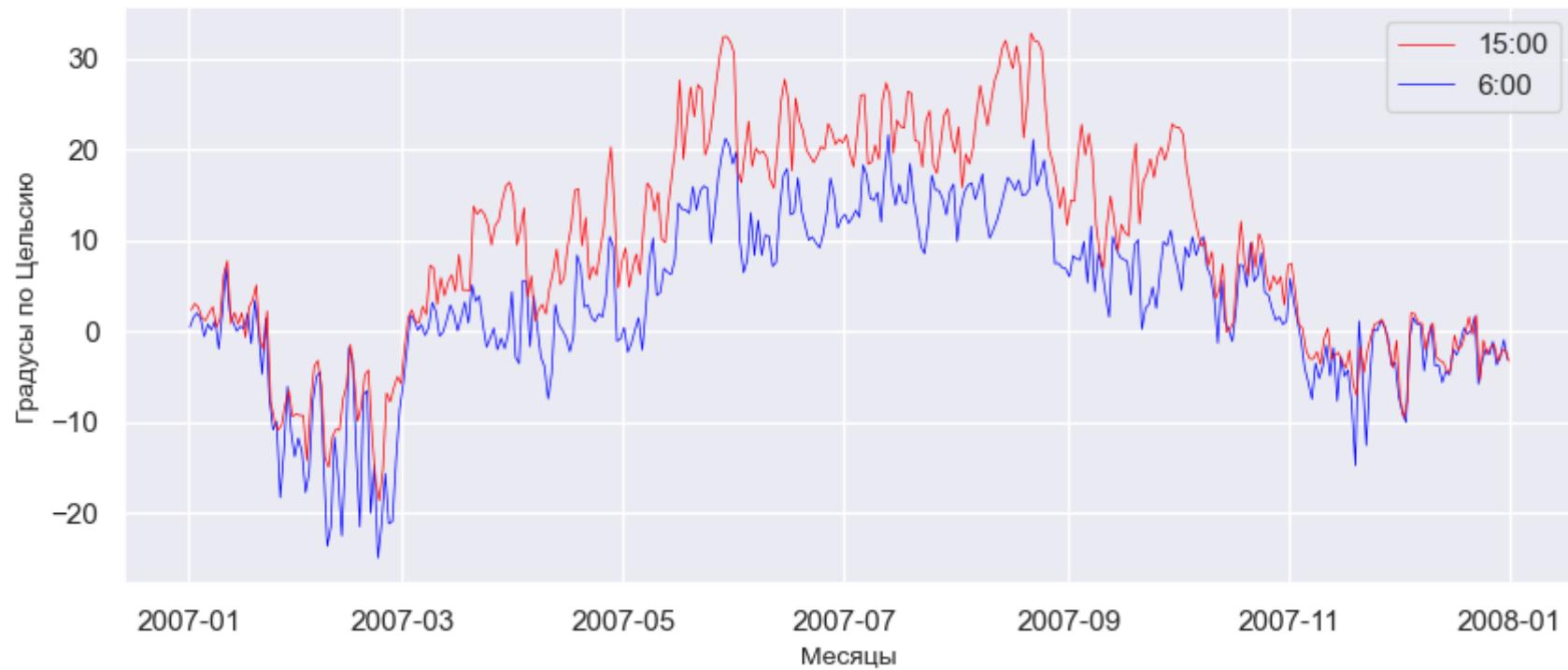
### Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2009 год



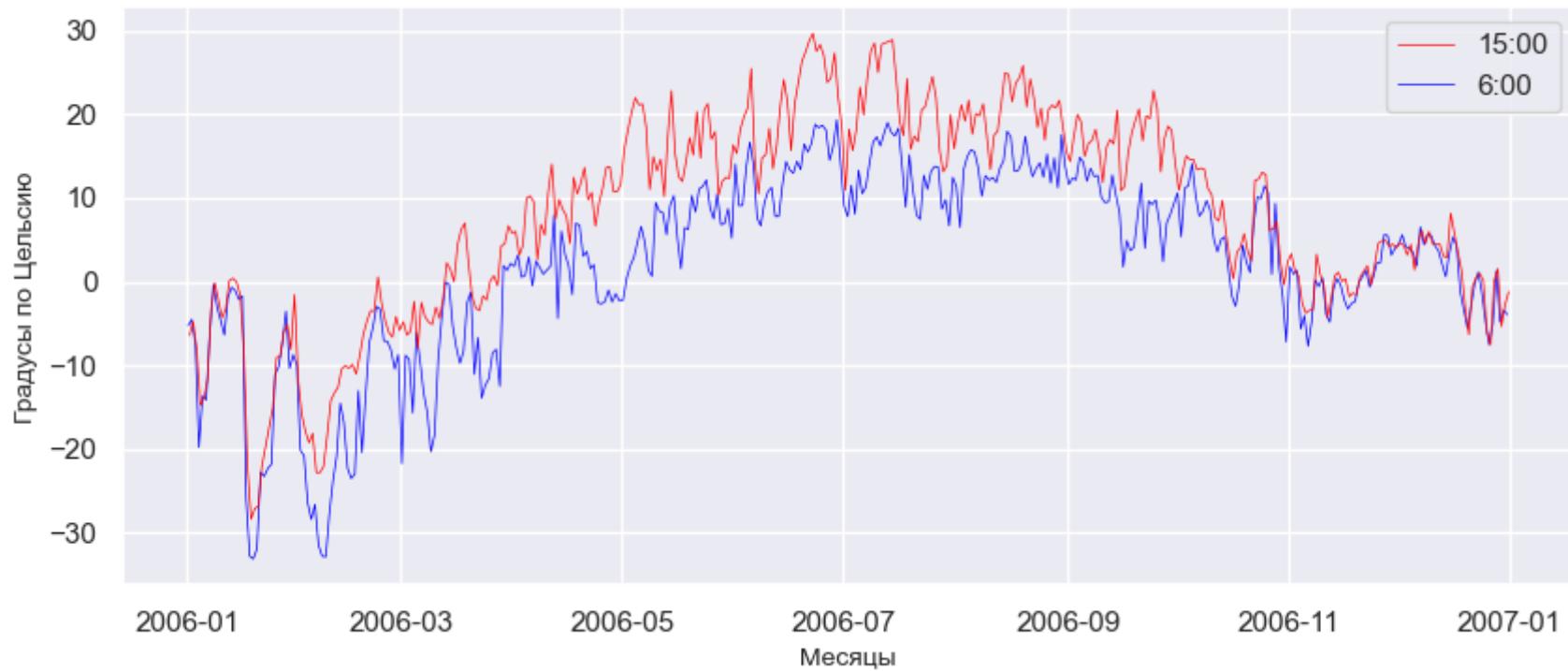
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2008 год



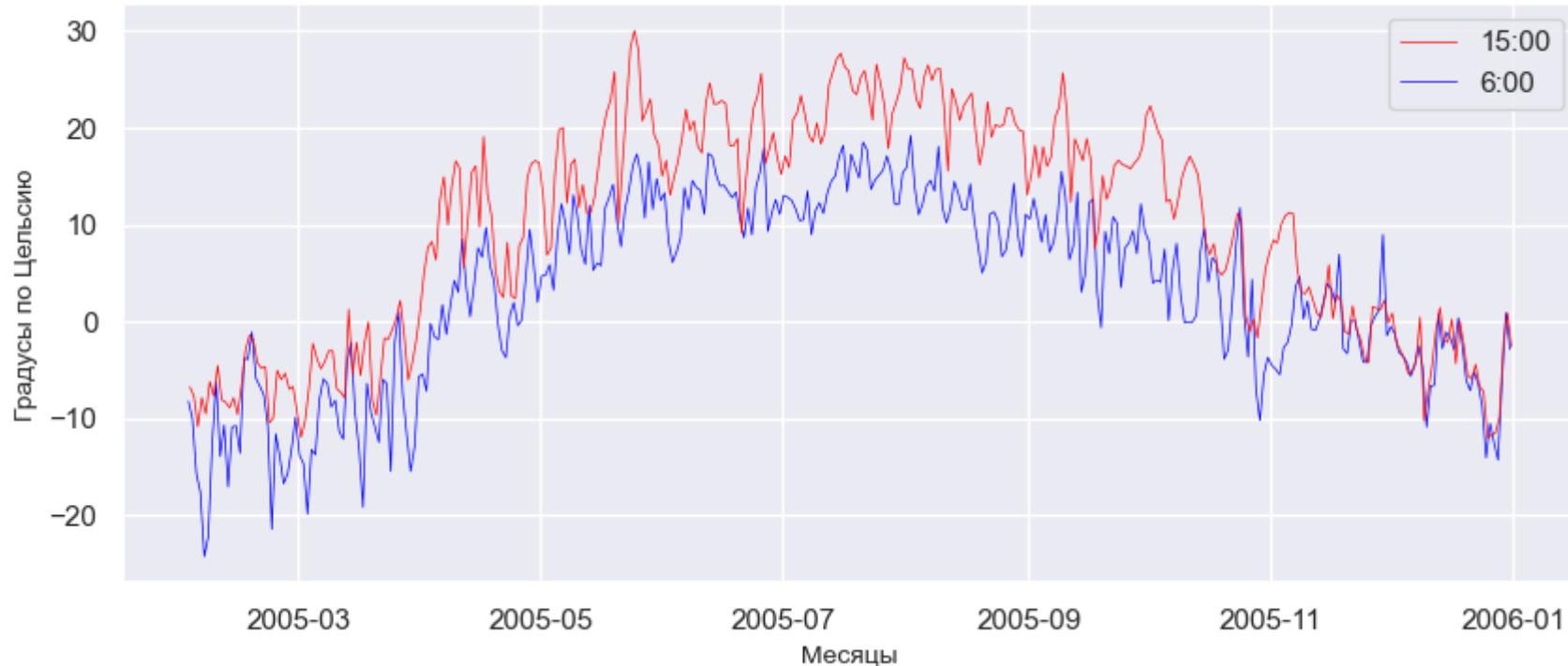
Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2007 год



Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2006 год



### Чашниково: Ежедневная динамика параметра Т на 6 и 15 часов, 2005 год



#### 3.1.6. Сохранение полученных данных в файлы

```
In [98]: # # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER31}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER31}'  
  
makedirs(predict_path1, exist_ok=True)  
makedirs(predict_path2, exist_ok=True)  
  
# Запишем текущие данные в файлы  
for name in dict_df_locations.keys():  
    print(name + '.csv ->', end=' ')
```

```

    dict_df_locations[name].to_csv(
        path_or_buf=f'{predict_path1}{name}.csv'
    )
    print('DONE! ')

    print('df_'+PARAMETER31 + '.csv ->', end=' ')
    dict_df_parameters['df_'+PARAMETER31].to_csv(
        path_or_buf=f'{predict_path2}df_{PARAMETER31}.csv'
    )
    print('DONE! ')

```

```

df_Dmitrov.csv -> DONE!
df_Kashyn.csv -> DONE!
df_Klin.csv -> DONE!
df_Mozhaisk.csv -> DONE!
df_Naro_Fominsk.csv -> DONE!
df_Nemchinovka.csv -> DONE!
df_N_Jerusalem.csv -> DONE!
df_Serpukhov.csv -> DONE!
df_Staritsa.csv -> DONE!
df_Tver.csv -> DONE!
df_Volokolamsk.csv -> DONE!
df_V_Volochev.csv -> DONE!
df_Chashnikovo.csv -> DONE!
df_Rfrnce_point.csv -> DONE!
df_T.csv -> DONE!

```

### 3.2. Минимальная температура воздуха: T\_min (за последние 12 часов, предшествующие наблюдению).

Минимальная температура должна измеряться не менее 2 раз в сутки. Однако, как показано в 1й тетради, а также при подсчёте фактической периодичности измерений, это положение далеко не всегда соблюдается. Более того, не всеми метеостанциями соблюдается синхронность в измерениях на определённый час. В связи с этим, чтобы привести измерения минимальной температуры к единой периодичности мы воссоздадим значения для этого показателя как если бы он фиксировался с интервалом в 3 часа за предыдущие 12 часов наблюдений. В реальной жизни такого не происходит (в том числе и из-за устройства самого прибора измерения), но нам важно привести значения к единым интервалам и моментам времени. В дальнейшем для анализа данных имеет смысл использовать данные на определённый утренний час за предшествующие 12 часов.

**Как будет показано ниже, основными моментами времени фиксации максимальной температуры является 09 часов утра и 06 часов утра. Поэтому, при анализе максимальной температуры следует использовать именно это время.**

### 3.2.1. Поиск и удаление ошибок показателя минимальной температуры T\_min

Для данного раздела обозначим константу названия параметра

```
In [99]: PARAMETER32 = 'T_min'
```

**Создаём временный DF для работы с параметром минимальной температуры**

```
In [100...]: param_df_name = f'df_{PARAMETER32}' # преобразуем полученное значение в df_PARAMETER32 - ключ словаря dict_df_parameters
# Создадим временный df
df_tmp32 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp32.sample(7, random_state=56)
```

Out[100]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
<b>2014-02-22 03:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2015-05-16 03:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2020-06-18 18:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2019-12-02 21:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2008-06-05 18:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2016-10-20 06:00:00</b>	0.1	-0.1	0.7	0.2	0.1	0.6	-0.7	-0.7	-0.7
<b>2006-09-11 03:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

### Визуализируем архив минимальной температуры (T\_min) по сезонам

Исходя из данных о климате Московской области, временные границы сезонов определены следующим образом.

- Зима (ниже 0°): В среднем длится с 5 ноября по 4 апреля
- Весна (от 0° до +10°): В среднем длится с 5 апреля по 18 мая
- Лето (выше +10°): В среднем длится с 19 мая по 14-15 сентября
- Осень (от +10° до 0°): В среднем длится с 14-15 сентября по 4 ноября

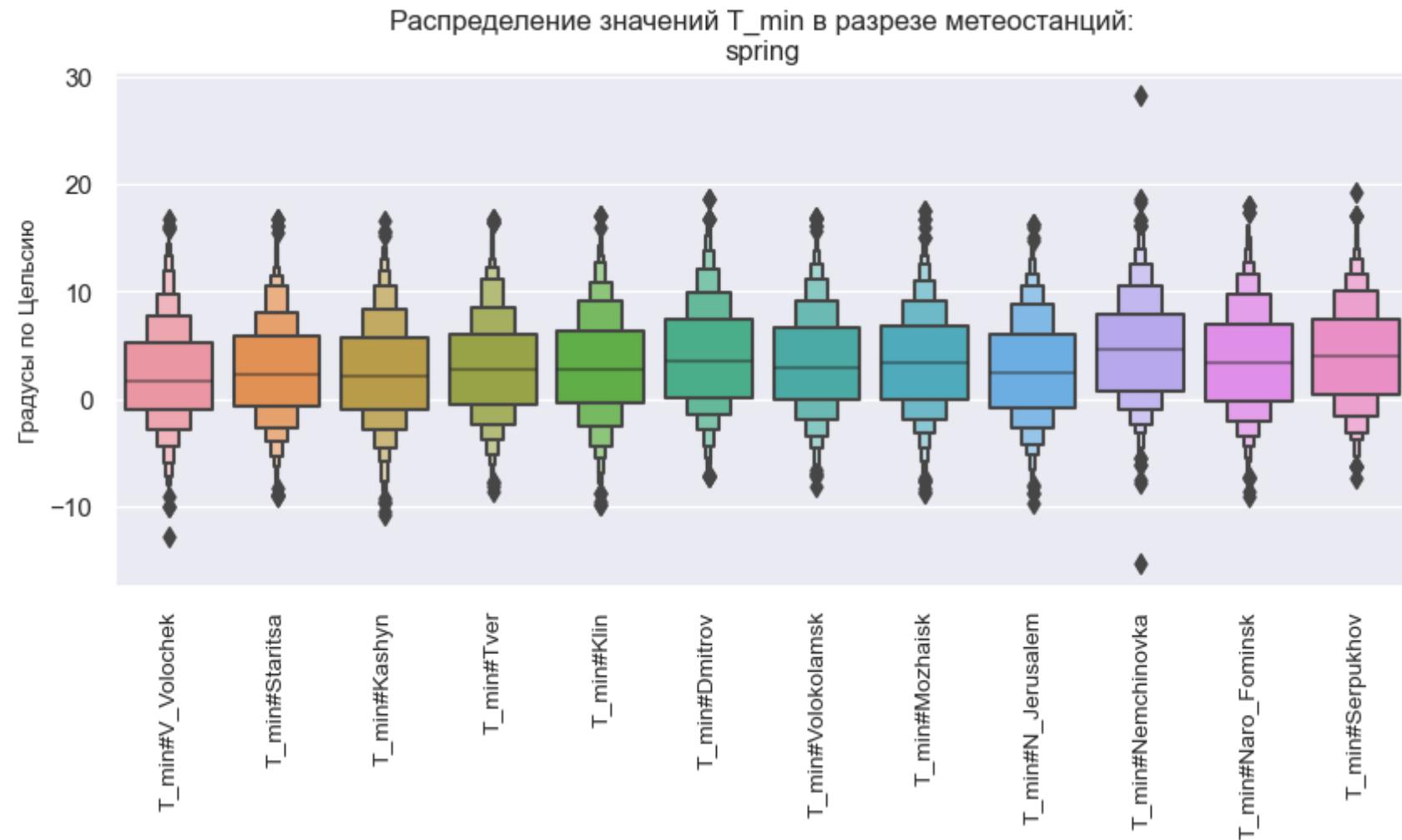
In [101...]

```
# В цикле выведем графики температур по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов (она возвращает словарь масок)
```

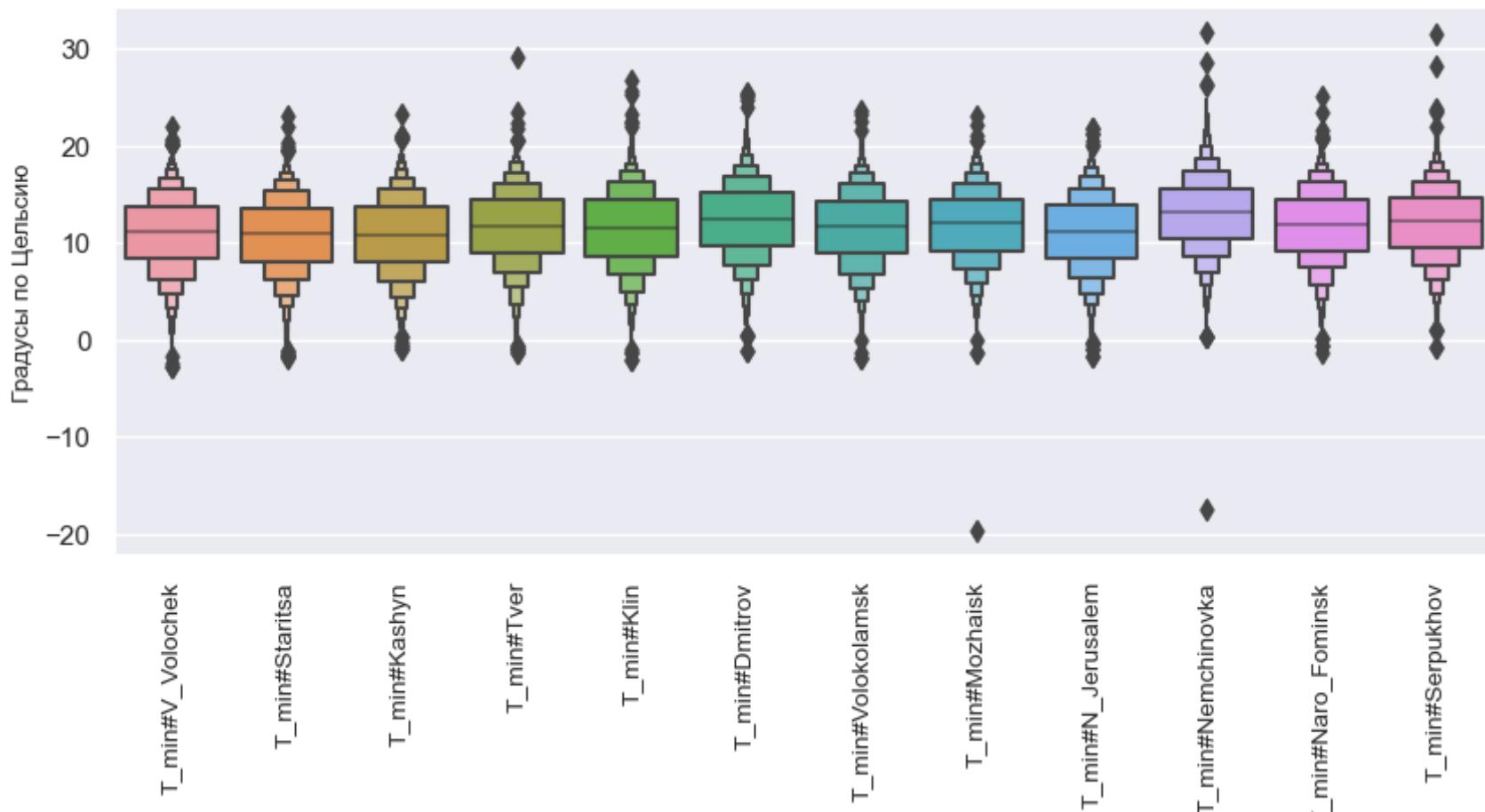
```

for season_name, season_mask in season_masks(df_tmp32).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp32[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Градусы по Цельсию', size=10)
    dummy = g.set_title(f'Распределение значений T_min в разрезе метеостанций:\n{season_name}')
plt.show()

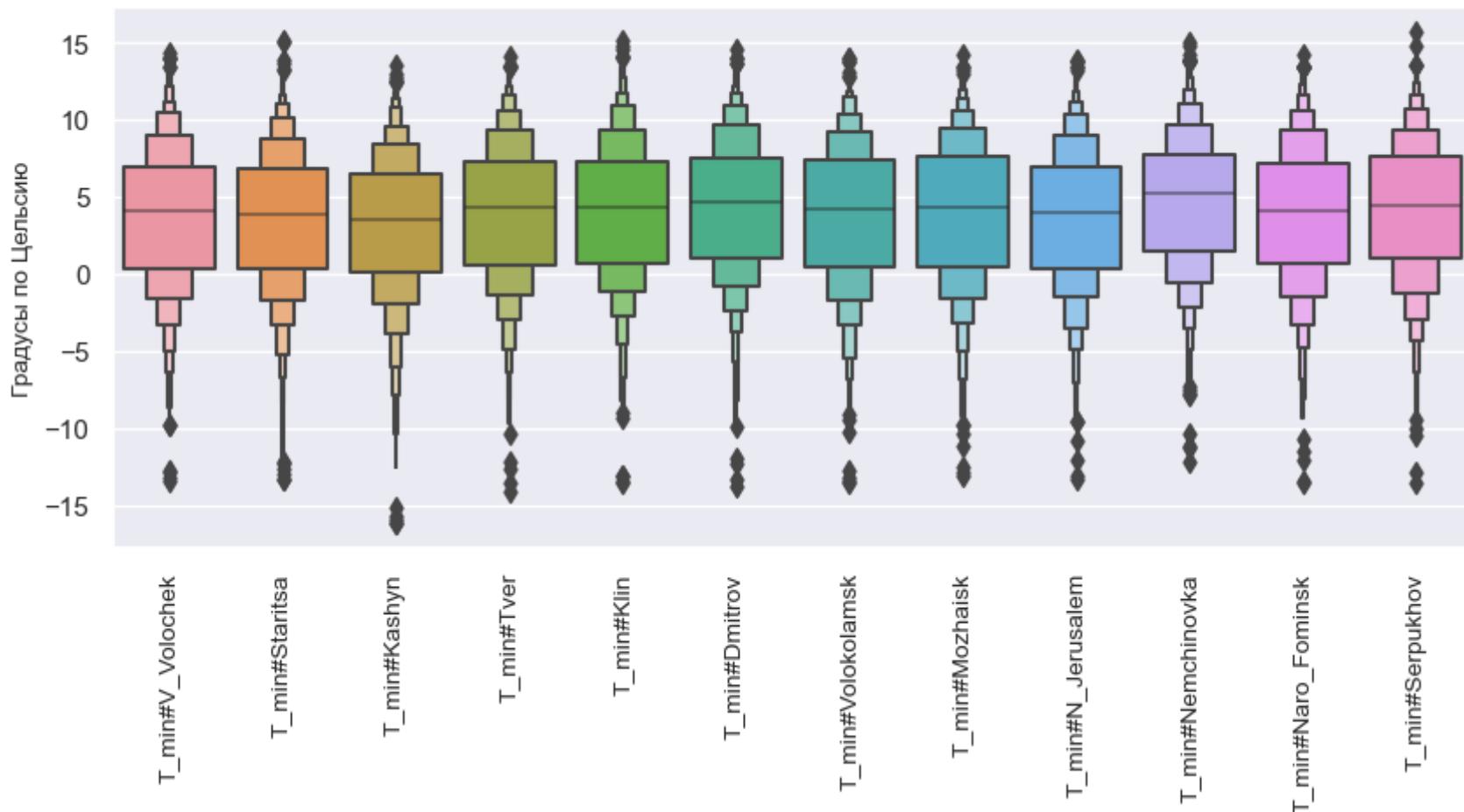
```



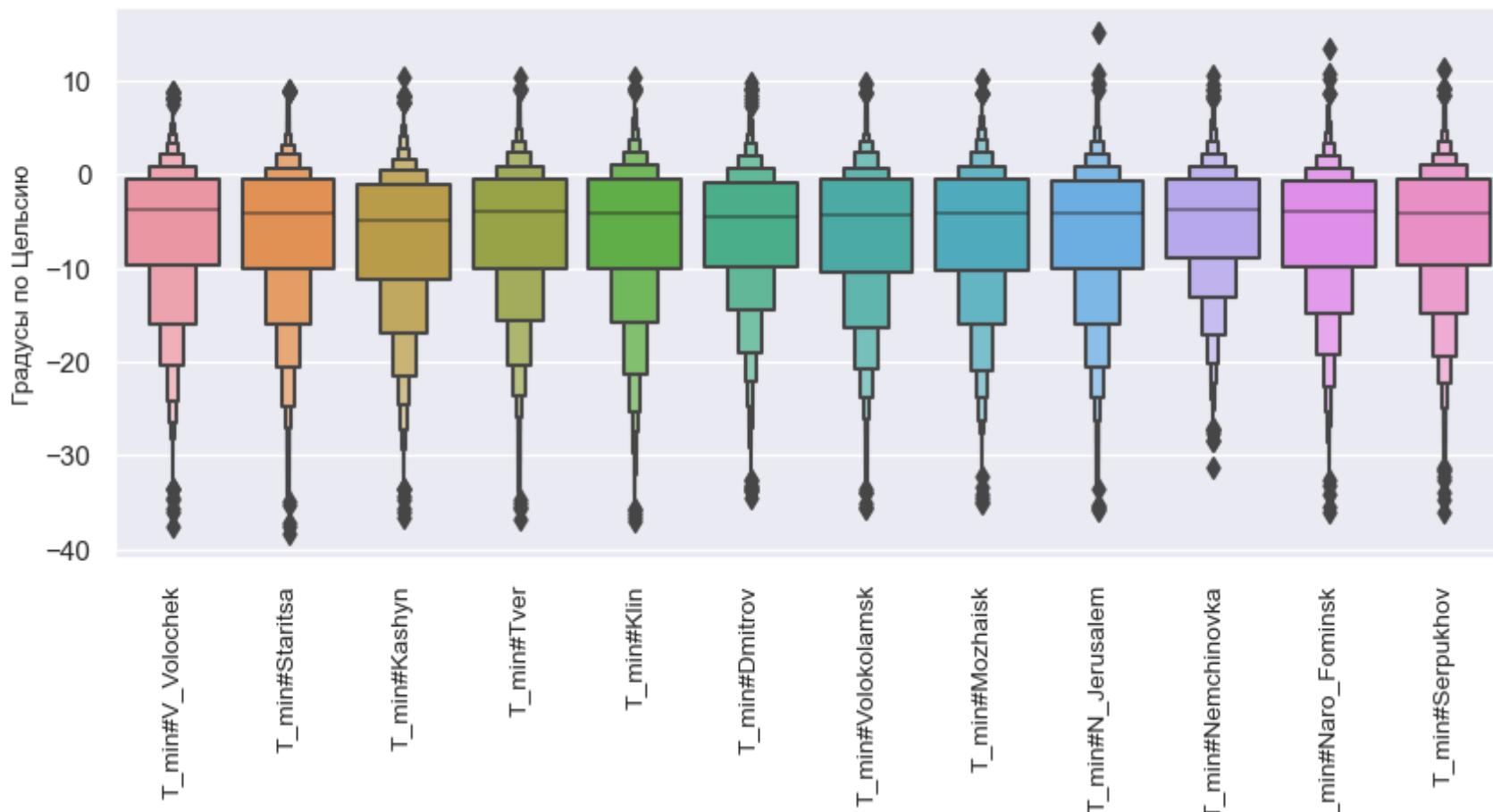
Распределение значений T\_min в разрезе метеостанций:  
summer



Распределение значений T\_min в разрезе метеостанций:  
autumn



## Распределение значений T\_min в разрезе метеостанций: winter



Как видно из графика, часть выбросов выглядит неестественно для своего сезона - это, скорее всего, ошибки.

Выведем минимальное и максимальное значения, а также значение медианы и средней для всего DF.

In [102...]

```
print(f'Минимальное значение: {np.nanmin(df_tmp32)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp32)},\n'
      f'Средняя: {np.nanmean(df_tmp32)},\n'
      f'Медиана: {np.nanmedian(df_tmp32)}')
```

```
Минимальное значение: -38.4,  
Максимальное значение: 31.6,  
Средняя: 2.0274550683631007,  
Медиана: 2.1
```

## Поиск и удаление ошибок

Как было показано в первой тетради, и как видно из случайно выбранных строк, показатель  $T_{min}$  содержит огромное количество пропущенных значений. Оценим объём имеющихся данных.

Удалим сплошные NaN (моменты наблюдения, по которым нет данных ни по одной метеостанции). Таким образом мы получим количество строк, в которых есть данные хотя бы по одной метеостанции. Сравним их с общим количеством моментов наблюдения.

```
In [103]:  
data_rows_frac = (df_tmp32.dropna(how='all', axis=0).shape[0] / df_tmp32.shape[0])  
print(f'Доля строк с хотя бы одним наблюдением T_min составляет {data_rows_frac:.2%}')
```

Доля строк с хотя бы одним наблюдением  $T_{min}$  составляет 19.94%

Очевидно, что фиксация минимальной температуры за последние 12 часов происходит не каждый момент наблюдения.

```
In [104]:  
# выведем часы и количество раз для каждого часа, когда фиксировались значения показателя T_min  
# получаем кортеж из 2x массивов:  
hours, counts = np.unique(df_tmp32.dropna(how='all', axis=0).index.hour, return_counts=True)  
# преобразуем его в массив и трансформируем (чтобы получить вертикальный вид)  
np.asarray((hours, counts)).T
```

```
Out[104]: array([[ 0,   54],  
                  [ 3,   54],  
                  [ 6, 3620],  
                  [ 9, 6322],  
                  [12,    4],  
                  [15,    9],  
                  [18,    1],  
                  [21,   45]], dtype=int64)
```

Таким образом основное время фиксации минимальной температуры приходится на 9 часов утра, и, почти в 2 раза реже, на 6 утра. Приблизительно 1,5% приходится на другие часы. Поэтому можно говорить о несистемности фиксации данных, что осложняет ситуацию.

*Определим последовательность действий, для поиска ошибок показателя "Минимальная температура".*

Здесь опять следует иметь в виду, что локальные температурные колебания могут быть достаточно значительными и зависят от

локальных метеорологических явлений.

Очевидным способом проверить корректность зафиксированных значений будет сравнить их с данными текущих наблюдений температуры воздуха Т, точнее с их минимумами за 12 предшествующих часов. При этом следует учесть, что минимумы температуры не обязательно будут совпадать по времени с моментами наблюдений, а могут случаться между ними.

1. Определяем выбросы в поле метеостанций - формируем кандидатов в ошибки, таким образом мы не пропустим "ложбины", когда понижение температуры затронуло большую часть метеостанций.
2. Отдельно проверяем, есть ли единичные значения в рядах - это тоже кандидаты в ошибки
3. Проверяем каждый из этих кандидатов на отклонение от минимумов значений по 12 часовому окну моментов наблюдения.
4. То, что осталось и есть ошибка.

### **Найдем выбросы в поле метеостанций.**

Параметры:

- используем среднюю по обратным квадратам расстояния от центра поля,
- включаем проверяемое значение в подсчёт средней (автоматически, так как средняя рассчитывается от центра поля для всех станций),
- определим границы доверительного интервала в 3,0 сигмы

In [105...]

```
start_time = time.time() # для замера времени выполнения кода
df_tmp32 = df_tmp32.dropna(how='all') # уберём все пустые строки
df_tmp32 = (df_tmp32.assign(field_out=df_tmp32
                             .apply(lambda x: field_outliers(row=x,
                                                               method_='sigma',
                                                               criterium_=3.0,
                                                               IDW_=True,
                                                               param_=PARAMETER32,
                                                               station_="Rfrnce_point",
                                                               inclusive_=True),
                                                               axis=1)
                             )
              .dropna(subset=["field_out"]) # убираем пустые ячейки во вновь созданном столбце
              )

end_time = time.time()
elapsed_time = end_time - start_time
time_format = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
```

```
print(f"На выполнение кода ушло: {time_format}")
df_tmp32.sample(7, random_state=56)
```

На выполнение кода ушло: 00:00:47

Out[105]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2006-07-15 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-10-14 06:00:00	NaN	3.7	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-05-07 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-02-13 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-01-22 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-01-04 06:00:00	0.3	-1.8	-1.6	-0.9	-1.5	-1.7	-2.0	-1.9	
2017-08-18 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [106...]

```
# Если значение является единственным в строке, то True, иначе False,
# убираем NaN (берём ряд без столбца field_out) и суммируем результатом
single_values_count = df_tmp32.apply(lambda x : True if (len(x[:-1].dropna()) == 1) else False, axis = 1).dropna().sum()
print(f'Количество случаев единичных значений в рядах моментов наблюдений: {single_values_count}'')
```

Используемые формулы определения выбросов специально обозначают как выброс в поле метеостанций случай, когда в ряду есть единственное значение. Проверим, есть ли ситуации, когда существует только одно значение параметра в поле метеостанций.

Количество случаев единичных значений в рядах моментов наблюдений: 217

Проверим максимальное количество выбросов в поле метеостанций на момент наблюдения.

In [107...]

```
# Находим максимальное количество выбросов в строке поля метеостанций
max_field_outliers = df_tmp32.field_out.dropna().apply(lambda x: len(x)).max()
print(f'Максимальное количество выбросов в поле метеостанций за весь период наблюдения не превышает '
      f'{max_field_outliers}')
```

Максимальное количество выбросов в поле метеостанций за весь период наблюдения не превышает 1

Для дальнейшей работы с ошибками создадим столбец `error_at`, куда запишем кортеж из `TimeStamp` и названия станции.

In 「108...

```
# Определяем столбец error_at.  
# Поскольку в field_out у нас всегда не более 1 выброса, мы можем не анализировать весь список из вывода функции  
# stations_from_outliers, а взять из вывода-списка первый (и единственный) элемент-кортеж  
  
df_tmp32 = df_tmp32.assign(error_at =  
    df_tmp32.  
    apply(lambda x: # Вычленим DateTime index и название станции с выбросом  
          # пропустим NaN, проверив, является ли x.field_out списком  
          (x.name,  
           stations_from_outliers(  
             row=x,  
             column_name_= 'field_out', # используем выбросы в поле метеостанций  
             param_=PARAMETER32)[0]  
           ) if isinstance(x.field_out, list) else np.nan,  
           axis=1  
          )  
    )  
df_tmp32.sample(5, random_state=56)
```

Out[108]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2006-07-15 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-10-14 06:00:00	NaN	3.7	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-05-07 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-02-13 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-01-22 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

◀ ▶

Для подтверждения, являются ли отобранные значения минимальной температуры ошибками, рассмотрим их отклонение от наблюдаемых значений показателя температуры Т за предыдущие 12 часов по моментам наблюдения (наблюдение 12 часов назад не включаем, текущее наблюдение включаем)

In [109...]

```
# построим датафрейм с минимальными значениями температуры по моментам наблюдения за предшествующие 12 часов
# Используем скользящее окно с вычислением минимума и со сдвигом вверх на 4 строки (-4),
# чтобы обозначить минимумы за предшествующие периоды.
# Исключим из него 2 последних столбца с условными метеостанциями
df_t1 = df_tmp31.rolling(window='12H', center=False, closed='left').min().shift(-4).iloc[:, :-2]
```

In [110...]

```
# преобразуем один из DF в массив пирту, чтобы избежать deprecation warning,
# возьмём только значения соответствующие индексам в датафрейме df_tmp32.
# Найдём абсолютное отклонение между элементами df_t1 и df_tmp32
# не берём 2 последних новых столбца df_tmp32
df_t2 = np.absolute(
    np.subtract(
        df_t1[df_t1.index.isin(df_tmp32.index)], np.array(df_tmp32.iloc[:, :-2])
    )
)
```

```
)  
# df_t2
```

In [111...]

```
# Выберем из df_t2 значения отклонений, соответствующие "кандидатам в ошибки"  
# и поместим их в отдельный столбец df_tmp32: Tn_deviation  
df_tmp32 = (df_tmp32  
    .assign(Tn_deviation=df_tmp32.error_at  
        .apply(lambda x:df_t2.loc[x[0], "T#" + x[1]]))  
    )  
df_tmp32.sample(5, random_state=56)
```

Out[111]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2006-07-15 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-10-14 06:00:00	NaN	3.7	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-05-07 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-02-13 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-01-22 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [112...]

```
# Выберем из них отклонения превышающие 5 градусов  
df_tmp32[df_tmp32.Tn_deviation > 5]
```

Out[112]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2022-02-05 06:00:00	-7.9	-7.9	-8.2	-7.1	-7.7	-8.5	-7.2	-10.0	
2022-01-09 09:00:00	-7.3	-6.0	-14.6	-5.8	-5.5	-5.7	-6.0	-5.9	
2021-03-21 09:00:00	-5.3	-5.9	-11.8	-5.7	-5.5	-5.9	-6.4	-5.4	
2021-03-21 06:00:00	-5.1	-5.9	-11.8	-5.7	-5.3	-5.9	-5.8	-5.4	
2020-03-16 09:00:00	-7.4	-11.5	-9.9	-11.5	-10.5	-10.7	-11.9	-1.2	
2019-11-06 09:00:00	1.9	8.8	8.3	9.0	8.9	9.1	8.5	8.3	
2019-07-06 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	18.1	
2017-08-18 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2015-06-04 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.2	
2014-09-24 06:00:00	4.4	9.9	8.9	10.2	9.7	9.8	10.0	9.9	
2014-08-16 21:00:00	NaN	23.1	NaN	NaN	NaN	NaN	NaN	NaN	

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2014-02-14 09:00:00	-0.7	0.0	0.3	0.3	1.0	0.8	0.7	1.1	
2014-02-14 06:00:00	-1.1	0.0	0.3	0.3	1.0	0.8	0.7	1.1	
2013-12-09 06:00:00	-8.6	-7.7	-10.3	-7.6	-6.7	-7.1	-7.7	-7.2	
2013-11-27 09:00:00	6.8	-8.1	-6.7	-7.5	-6.5	-6.0	-7.3	-6.4	
2013-10-09 09:00:00	10.5	9.9	2.7	10.5	8.7	7.1	10.1	9.9	
2013-10-09 06:00:00	10.5	9.9	2.7	10.5	8.7	7.1	10.1	9.9	
2013-09-08 09:00:00	2.4	8.7	10.4	9.6	10.4	10.4	9.8	10.2	
2013-09-08 06:00:00	2.4	8.9	10.4	9.6	10.4	10.4	9.8	10.3	
2013-08-30 03:00:00	NaN	NaN	NaN	4.3	NaN	NaN	NaN	NaN	
2013-06-04 21:00:00	NaN	NaN	NaN	29.0	NaN	NaN	NaN	NaN	
2013-05-16 09:00:00	14.3	10.6	11.9	12.1	13.3	14.8	11.8	12.4	

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2013-05-11 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013-02-13 09:00:00	-1.3	-2.0	-3.1	-2.1	-2.9	-3.7	-2.4	-2.3	
2012-06-23 09:00:00	9.4	13.8	8.9	13.0	10.4	14.5	15.1	15.6	
2012-06-16 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2012-04-20 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2011-07-16 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2011-07-09 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2011-03-30 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2011-01-15 09:00:00	-6.2	-6.7	-5.6	-5.4	-5.2	5.2	-5.7	-5.5	
2010-11-27 09:00:00	-8.6	-2.8	-4.0	-4.3	-0.7	-0.9	-1.0	-2.1	
2010-08-17 21:00:00	NaN	NaN	NaN	NaN	26.7	NaN	NaN	NaN	

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2009-09-28 09:00:00	11.5	11.0	10.6	10.9	11.3	10.7	11.1	11.4	
2009-09-12 09:00:00	3.5	8.6	7.7	7.9	9.8	11.2	10.2	10.0	
2009-09-04 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2009-06-03 21:00:00	NaN	NaN	NaN	NaN	22.4	NaN	NaN	NaN	
2009-04-09 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2009-02-20 09:00:00	-11.2	-12.3	-8.6	-11.1	-15.6	-9.6	-17.0	-16.7	
2008-12-25 09:00:00	-14.5	-8.5	-5.9	-5.2	-6.3	-6.1	-5.6	-6.2	
2008-12-05 09:00:00	-1.3	3.5	6.0	5.2	6.7	6.1	6.1	6.4	
2008-07-15 09:00:00	18.9	18.0	17.4	18.9	18.4	19.4	18.6	11.8	
2008-02-22 09:00:00	-0.9	-1.7	-9.5	-2.2	-2.6	-4.6	-1.9	-1.7	
2007-12-22 09:00:00	0.9	1.4	-7.8	-0.4	-0.6	-2.3	0.1	-0.2	

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2006-10-04 09:00:00	10.5	1.0	10.2	10.4	11.0	11.6	10.1	11.6	
2006-08-23 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-19.7	
2006-04-09 09:00:00	-5.3	-0.6	-0.3	-0.6	0.5	0.5	0.3	0.3	
2006-03-08 21:00:00	NaN	-5.2	NaN	NaN	NaN	NaN	NaN	NaN	
2006-02-03 09:00:00	-3.3	-34.4	-30.3	-29.5	-30.4	-28.0	-30.4	-29.7	
2006-01-11 09:00:00	-12.4	-7.4	-8.1	-7.4	-6.7	-7.0	-6.7	-7.1	
2005-08-10 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

### Применимость критериев ошибочных значений

Рассмотрим полученные данные об аномальных значениях.

In [113...]

```
print(f'В поле метеостанций выявлено аномальных значений минимальной температуры: {df_tmp32.error_at.count()}, из них:\n'
      f'Подтверждается отклонением от минимальных регулярно фиксируемых значений на более чем 5 градусов: '
      f'{df_tmp32[df_tmp32.Tn_deviation > 5].Tn_deviation.count()}\n')
```

В поле метеостанций выявлено аномальных значений минимальной температуры: 456,  
из них:

Подтверждается отклонением от минимальных регулярно фиксируемых значений на более чем 5 градусов: 51

## Окончательный критерий ошибочных значений

Ошибка считается выброс в поле метеостанций или единичное зафиксированное в поле метеостанций, которое отклоняется от минимальной наблюдаемой температуры (по моментам наблюдения) за 12 часов на более чем 5 градусов.  
Такие значения необходимо заменить на NaN.

## Удалим (заменим на NaN) все ошибочные значения

In [114...]

```
# Создадим список координат ячеек, содержащих значения, определённые как ошибки
list_error_coords = df_tmp32[df_tmp32.Tn_deviation > 5].error_at.tolist()
# list_error_coords
```

In [115...]

```
# Восстановим исходное состояние df_tmp32
df_tmp32 = dict_df_parameters[param_df_name].copy(deep=True)
```

In [116...]

```
for error_coords in list_error_coords: # по списку координат ошибочных значений
    # Преобразуем координату столбца и присваиваем ячейке NaN
    df_tmp32.at[error_coords[0], PARAMETER32+'#'+error_coords[1]] = np.nan
# df_tmp32
```

In [117...]

```
# Проверим, все ли ошибки заменены на NaN
# Количество неNaN значений в df_tmp31 по координатам, указанным в списке list_error_coords
np.sum([pd.notna([df_tmp32.at[error_coords[0], PARAMETER32+'#'+error_coords[1]] for error_coords in list_error_coords])])
```

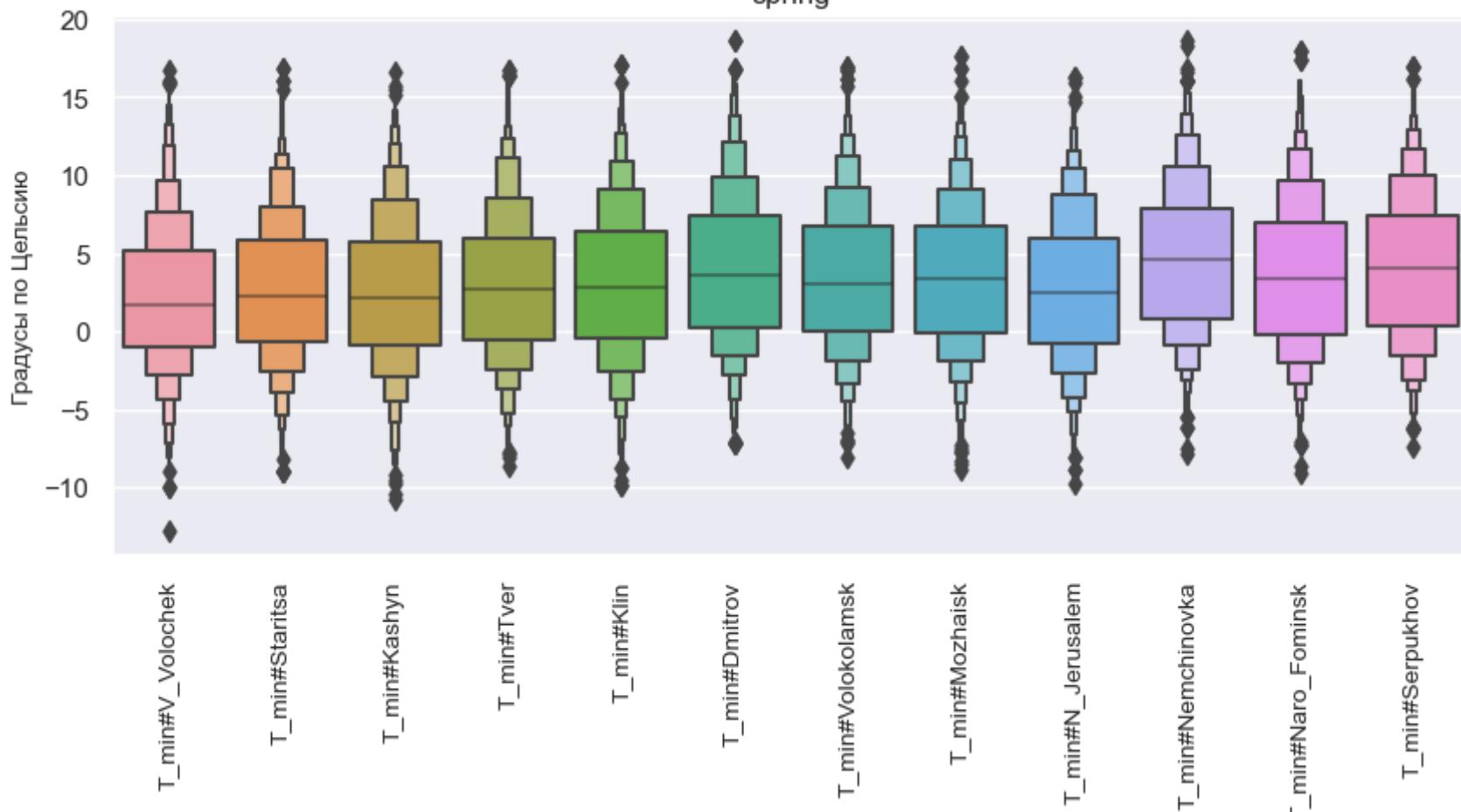
Out[117]:

## Визуализируем архив минимальных значений температуры (T\_min) по сезонам после удаления ошибок

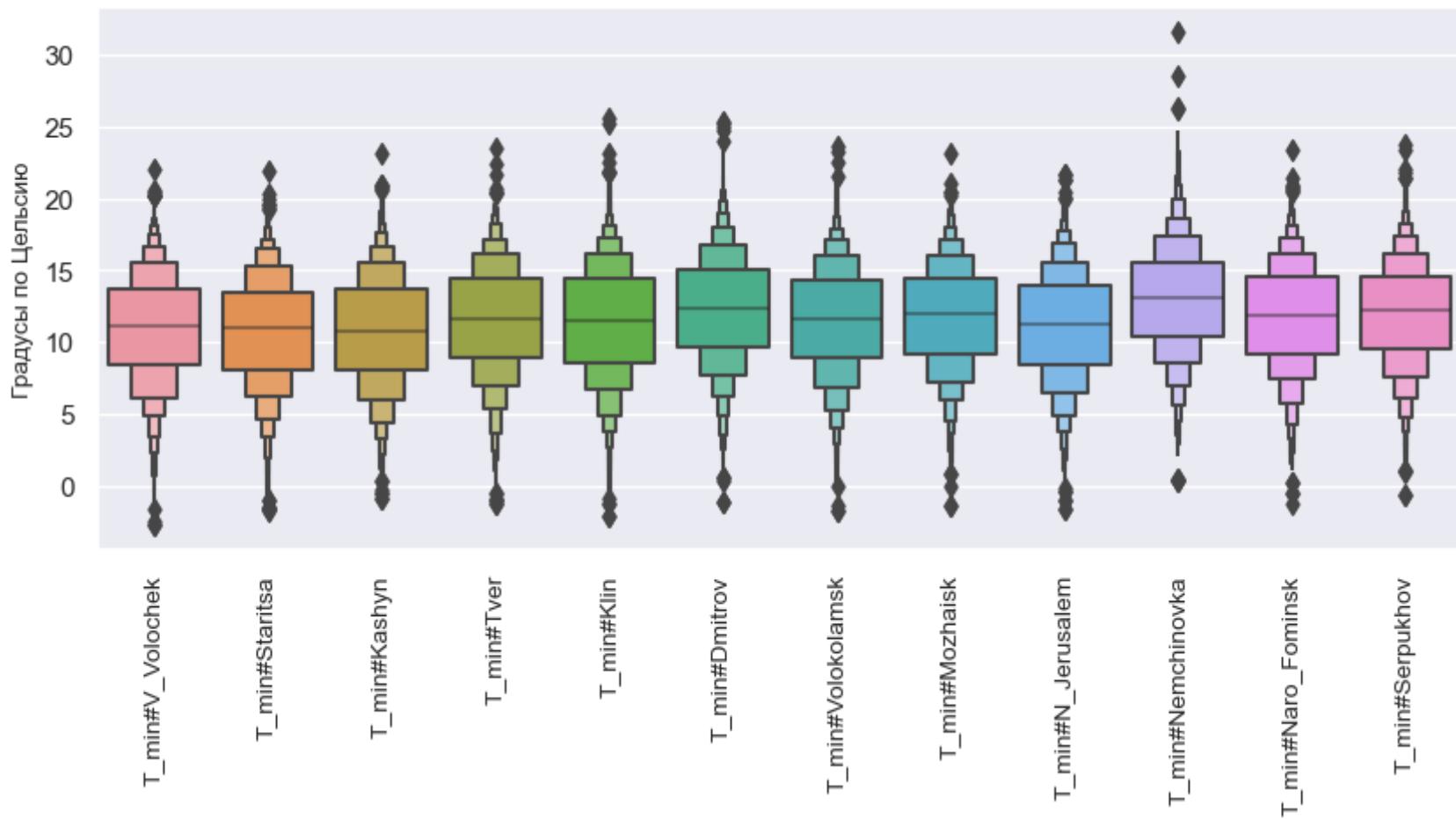
In [118...]

```
# В цикле выведем графики температур по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp32).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp32[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Градусы по Цельсию', size=10)
    dummy = g.set_title(f'Распределение значений T_min в разрезе метеостанций после удаления ошибок:\n{season_name}')
plt.show()
```

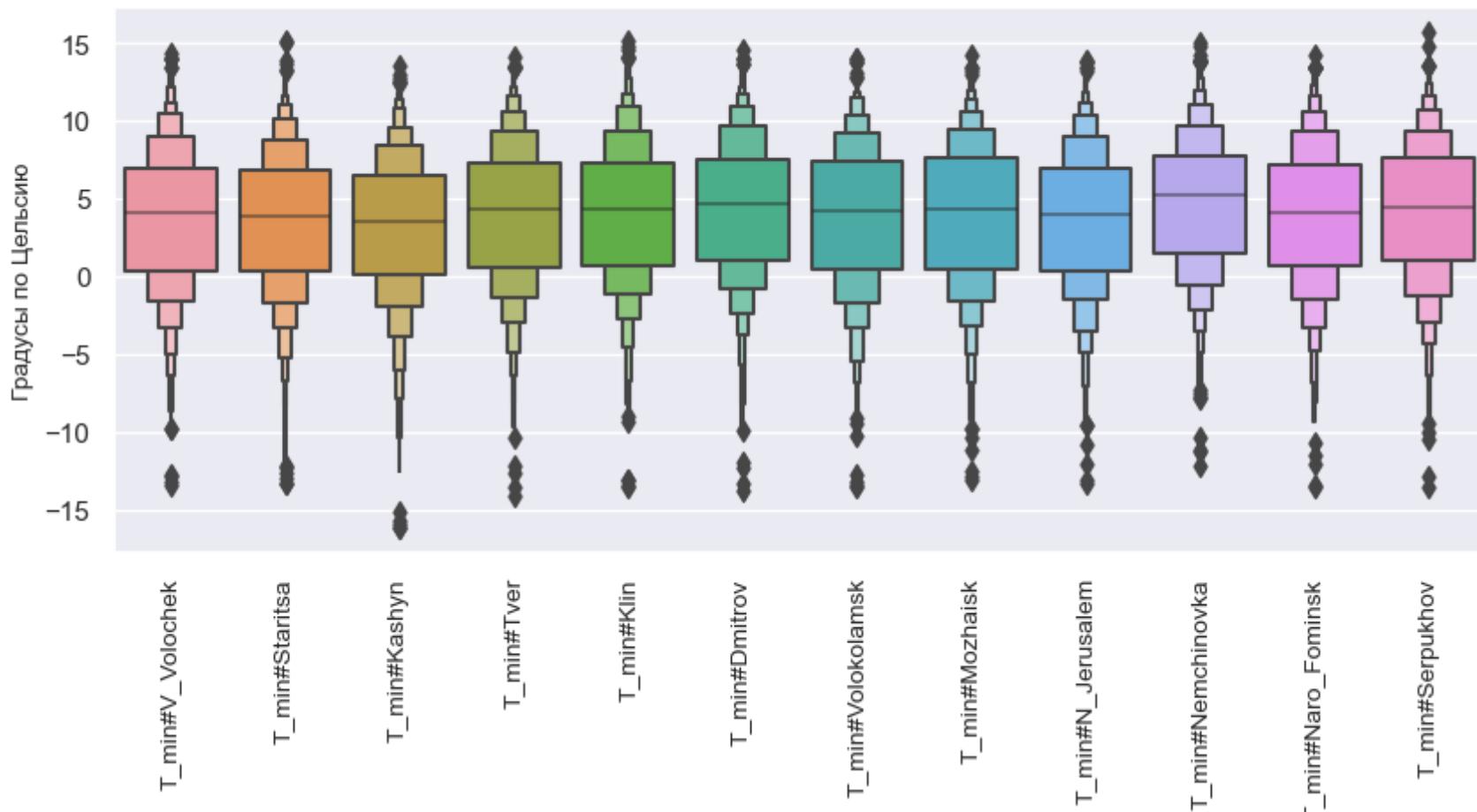
Распределение значений T\_min в разрезе метеостанций после удаления ошибок:  
spring



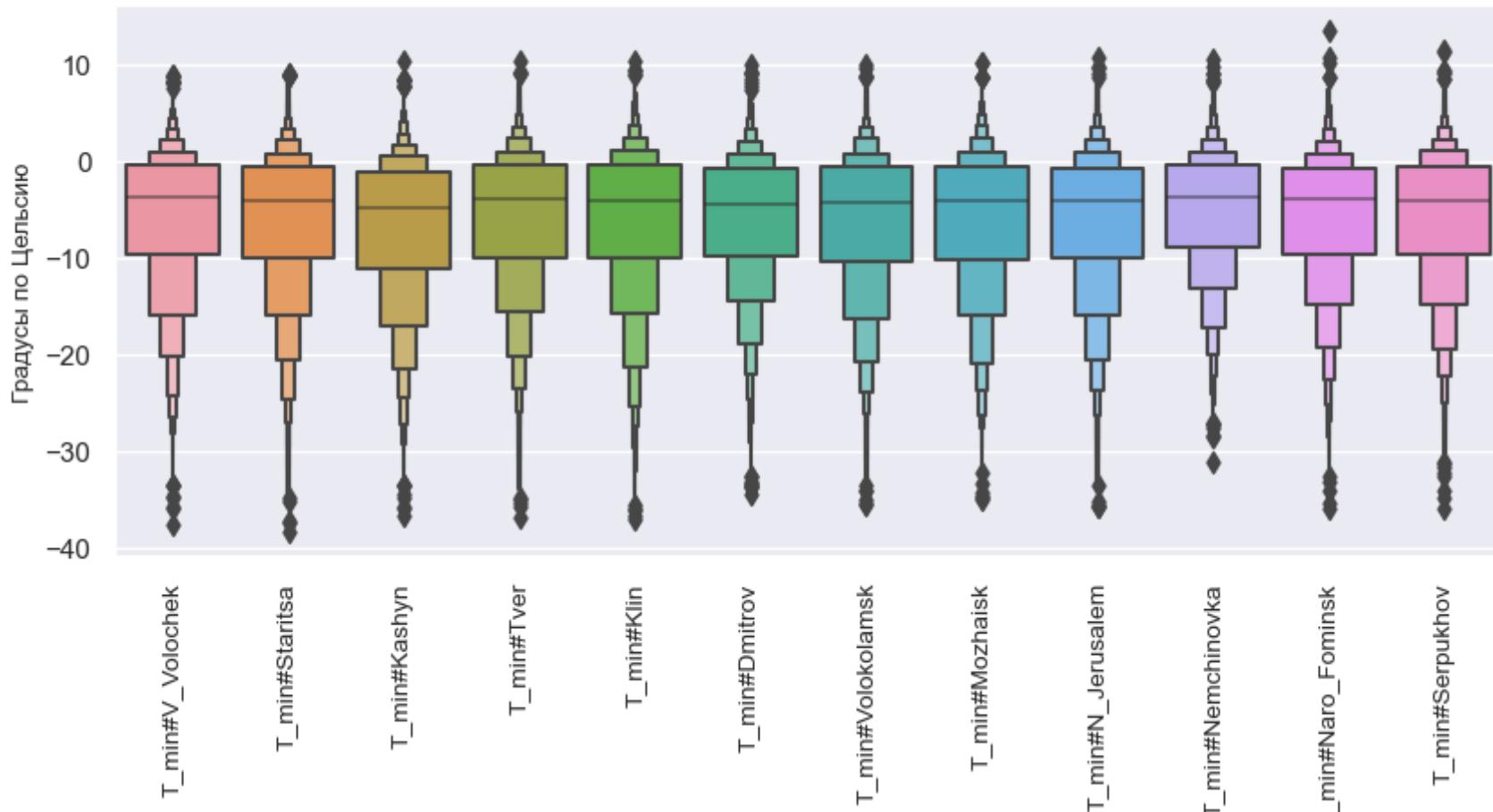
Распределение значений T\_min в разрезе метеостанций после удаления ошибок:  
summer



Распределение значений T\_min в разрезе метеостанций после удаления ошибок:  
autumn



## Распределение значений T\_min в разрезе метеостанций после удаления ошибок: winter



Выведем минимальное и максимальное значения, а также значение медианы и средней для всего DF после удаления ошибок

```
In [119...]: print(f'Минимальное значение: {np.nanmin(df_tmp32)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp32)},\n'
      f'Средняя: {np.nanmean(df_tmp32)},\n'
      f'Медиана: {np.nanmedian(df_tmp32)})'
```

Минимальное значение: -38.4,  
Максимальное значение: 31.6,  
Средняя: 2.026912189216873,  
Медиана: 2.1

### Сохраним очищенные данные в файл параметров

In [120...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
clean_path = f'{path}clean/'  
  
makedirs(clean_path, exist_ok=True)  
  
# Запишем текущие данные в файл  
print('df_'+PARAMETER32 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER32].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER32}.csv'  
)  
print('DONE!')  
  
df_T_min.csv -> DONE!
```

### 3.2.2. Восстановление "сплошных" NaN показателя минимальной температуры за 12 часов T\_min методом сользящего минимума из зафиксированных по моментам наблюдений фактических значений температуры Т

Модели пространственной экстраполяции не могут экстраполировать значения в рамках одного момента наблюдения, если значения во всех точках наблюдения неизвестны. Такие случаи есть в наших арихивах.

In [121...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp32  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp32.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp32.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 40614

Очевидно такие строки нужно заполнить до пространственной экстраполяции. Представляется, что лучшим способом заполнения пропущенных строк будет скользящий минимум наблюдений температуры за 12 часов. Эти значения уже определены в df\_t1.

In [122...]

```
# Создадим маску для выбора строк со сплошными NaN  
# Выбираем из датафрейма строки, где количество NaN равно количеству столбцов - то есть сплошные NaN  
mask_total_nans = df_tmp32.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp32.keys()))
```

In [123...]

```
# Выберем значения из df_t1, соответствующие индексам строк сплошных NaN в df_tmp32 и присвоим их ячейкам df_tmp32  
df_tmp32[mask_total_nans] = df_t1[df_t1.index.isin(df_tmp32[mask_total_nans].index)]  
df_tmp32[mask_total_nans].sample(5, random_state=56)
```

Out[123]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2018-04-30 21:00:00	12.1	16.8	9.4	13.9	15.3	15.2	13.8	15.3	
2009-06-30 03:00:00	9.4	10.4	7.7	11.0	8.1	8.9	10.1	10.9	
2019-03-09 12:00:00	0.7	2.6	1.1	3.5	2.2	1.9	2.7	2.3	
2019-12-19 12:00:00	1.7	2.7	1.6	3.5	3.5	3.7	3.3	3.7	
2008-09-28 21:00:00	8.0	9.9	9.5	10.3	10.6	10.4	9.9	10.2	

In [124...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp32  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp32.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp32.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 3

Всё верно, это самые начальные 3 строки, для которых невозможно определить 12 часовое окно.

Поскольку в процессе удаления ошибочных значений удалялись и единичные значения в строках, и часть строк могла превратиться в сплошные NaN. Проверим, сколько единичных значений исправлено методом восстановления сплошных NaN

```
In [125]: # Количество неNaN значений в df_tmp31 по координатам, указанным в списке list_error_coords  
np.sum([pd.notna([df_tmp32.at[error_coords[0], PARAMETER32+'#'+error_coords[1]] for error_coords in list_error_coords])])
```

Out[125]: 19

Получившееся значение показывает количество исправленных ошибочных значений, там где был применён метод восстановления сплошных NaN.

### 3.2.3. Подбор модели для восстановления пропущенных и удалённых значений показателя минимальной температуры T\_min, а также для моделирования значений для искомой точки.

Важным моментом для моделирования недостающих значений и значений для искомой точки является то, что большинство значений показателя T\_min было изначально состояло из сплошных строк NaN. Они были механически заполнены скользящими минимумами из значений температуры. Представляется более логичным не моделировать их для искомых точек, а также заполнить. Поэтому их следует исключить из обучающего и валидационного датасетов.

#### 3.2.3.1. Модель средней, взвешенной по степени обратных расстояний (далее по тексту эту модель будем называть сокращённо IDW)

Эта модель уже применялась выше

Попробуем, как работает модель с различными значениями степени для обратных расстояний, и выделим ту степень, которая обеспечивает лучшие метрики качества.

Создадим набор данных для обучения и валидации модели IDW.

1. Используем данные в df\_tmp32.
2. Уберём все строки с NaN.
3. Выделим рандомно массив известных значений для разных метеостанций и разных моментов наблюдения - он потребуется для разделения на обучающую и валидационную часть и для расчёта метрик качества.

In [126]:

```
# Создаём датафрейм для валидации модели IDW:  
# - не включаем изначальные строки сплошных NaN  
# - убираем все строки с хотя бы одним оставшимся NaN  
df_test = df_tmp32[~mask_total_nans].dropna(how='any')
```

In [127]:

```
df_test.info()  
df_test.shape  
  
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 5162 entries, 2022-06-08 09:00:00 to 2007-09-27 09:00:00  
Data columns (total 12 columns):  
 #   Column           Non-Null Count  Dtype     
---  --  
 0   T_min#V_Volochek    5162 non-null   float64  
 1   T_min#Staritsa      5162 non-null   float64  
 2   T_min#Kashyn        5162 non-null   float64  
 3   T_min#Tver          5162 non-null   float64  
 4   T_min#Klin          5162 non-null   float64  
 5   T_min#Dmitrov       5162 non-null   float64  
 6   T_min#Volokolamsk   5162 non-null   float64  
 7   T_min#Mozhaisk       5162 non-null   float64  
 8   T_min#N_Jerusalem    5162 non-null   float64  
 9   T_min#Nemchinovka   5162 non-null   float64  
 10  T_min#Naro_Fominsk  5162 non-null   float64  
 11  T_min#Serpukhov     5162 non-null   float64  
dtypes: float64(12)  
memory usage: 524.3 KB  
(5162, 12)
```

Out[127]:

Создадим массив индексов для выборки моделируемых и проверочных значений. Массив будет включать последовательно все индексы строк и случайный индекс столбца.

In [128]:

```
# Зафиксируем RandomState  
rs56 = np.random.RandomState(56)  
# Создаём 1мерный массив с последовательностью, равной количеству рядов в df_test  
arr_row_index = np.arange(0, df_test.shape[0])  
# Создаём 1мерный массив со случайными целыми числами в диапазоне количества столбцов в df_test  
arr_column_index = rs56.randint(0, df_test.shape[1], df_test.shape[0])  
# Соединяем 2 массива и транспонируем полученный массив  
arr_idx_data = np.vstack((arr_row_index, arr_column_index)).T
```

```
arr_row_index
arr_column_index
arr_idx_data

Out[128]: array([ 0,  1,  2, ..., 5159, 5160, 5161])
Out[128]: array([ 5,  4,  0, ..., 10,  9, 11])
Out[128]: array([[ 0,  5],
   [ 1,  4],
   [ 2,  0],
   ...,
   [5159, 10],
   [5160,  9],
   [5161, 11]])
```

Создадим тренировочный и обучающий массивы. Для этого:

- определим  $X$  как массив координат ячеек в архиве - (np.array),
- определим  $y$  как массив значений ячеек в множестве  $X$  - (np.array).

```
In [129...]:
# Разделим наши данные на обучающую и валидационную части.
# используем индексы значений как параметры модели
X = arr_idx_data
# результирующие значения (фактические значения измерений, соответствующие индексам), выведем в список и преобразуем в np.array
y = np.array([df_test.iloc[x, y] for x, y in arr_idx_data])

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.6, test_size=0.4, random_state=56)
x_train.shape
y_train.shape
x_test.shape
y_test.shape
```

Out[129]: (3097, 2)  
Out[129]: (3097,)  
Out[129]: (2065, 2)  
Out[129]: (2065,)

**Обучающий датасет** Подберём лучшую степень для весов - обратных расстояний, ориентируясь на лучшие значения метрик качества

In [130...]

```
start_time = time.time() # для замера времени выполнения кода

r2_idw_tr, mae_idw_tr, rmse_idw_tr = 0, 0, 0 # обнулим значения метрик качества
iterations = 0 # количество итераций
power = 3 # Начальное значение степени (опробованы начальные степени от 1)
power_increment = 0.25 # шаг увеличения степени
list_metrics=[] # список для фиксации результатов итераций

r2_idw_tr_old = 0 # Устанавливаем в 0 предыдущее значение R2
print('ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:\n')

# Зададим предел R2 для поиска оптимального веса
while r2_idw_tr_old <= r2_idw_tr:
    power += power_increment # Увеличиваем степень на 1 шаг
    iterations +=1 # Увеличиваем счётчик проходов цикла
    r2_idw_tr_old = r2_idw_tr # Запоминаем предыдущее значение R2

    # Создаём y_predict_idw_tr по индексам в массиве x_train
    # используем функцию inverse_distance_avg
    # Проходим по массиву и считываем из df_test данные по координатам, выводим результат списком
    y_predict_idw_tr = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param_=PARAMETER32,
                             station_=df_test.keys()[y][len(PARAMETER32)+1:],
                             df_dists_= df_station_dists,
                             power_=power)
        for x, y in x_train
    ]

    # Расчитываем метрики качества
    max_e_idw_tr = max_error(y_train, y_predict_idw_tr)
    mae_idw_tr = mean_absolute_error(y_train, y_predict_idw_tr)
    mse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr)
    rmse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr, squared=False)
    r2_idw_tr = r2_score(y_train, y_predict_idw_tr)

    if r2_idw_tr > r2_idw_tr_old:
        best_power_idw_tr = power
        best_max_e_idw_tr = max_e_idw_tr
```

```
best_mae_idw_tr = mae_idw_tr
best_mse_idw_tr = mse_idw_tr
best_rmse_idw_tr = rmse_idw_tr
best_r2_idw_tr = r2_idw_tr

# замер времени:
chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

print(f'Elapsed time={time_formatted}\n'
      f'Текущие значения: iterations={iterations}, power={power},\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_tr:.7f}\n'
      )
print(f'ЛУЧШИЕ значения: power={best_power_idw_tr},\n'
      f'Максимальная ошибка (MAX_E) IDW = {best_max_e_idw_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {best_mae_idw_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {best_mse_idw_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {best_rmse_idw_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {best_r2_idw_tr:.7f}'
```

ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:

```
Elapsed time=00:00:12
Текущие значения: iterations=1, power=3.25,
Максимальная ошибка (MAX_E) IDW = 10.6513920
Средняя абсолютная ошибка (MAE) IDW = 0.9495302
Средний квадрат ошибки (MSE) IDW = 1.8494611
Средняя квадратическая ошибка (RMSE) IDW = 1.3599489
Коэффициент детерминации (R2) IDW = 0.9795951
```

```
Elapsed time=00:00:25
Текущие значения: iterations=2, power=3.5,
Максимальная ошибка (MAX_E) IDW = 10.6881798
Средняя абсолютная ошибка (MAE) IDW = 0.9493181
Средний квадрат ошибки (MSE) IDW = 1.8521636
Средняя квадратическая ошибка (RMSE) IDW = 1.3609422
Коэффициент детерминации (R2) IDW = 0.9795653
```

ЛУЧШИЕ значения: power=3.25,  
Максимальная ошибка (MAX\_E) IDW = 10.6513920  
Средняя абсолютная ошибка (MAE) IDW = 0.9495302  
Средний квадрат ошибки (MSE) IDW = 1.8494611  
Средняя квадратическая ошибка (RMSE) IDW = 1.3599489  
Коэффициент детерминации (R2) IDW = 0.9795951

Несколько запусков кода выше, с различными параметрами степени (от 1 до 10), показали, что, судя по R2, лучшим значением степени на обучающем массиве является 3,25. Оно даёт лучшие метрики качества.

Применим эту степень для валидационного массива.

## Валидационный датасет

In [131...]

```
# Создаём y_predict_idw_vld по индексам в x_test
# используем функцию inverse_distance_avg
# Проходим по массиву и считываем из df_test данные по координатам, выводим результат списком

y_predict_idw_vld = [
    inverse_distance_avg(row_=df_test.iloc[x],
                          param_=PARAMETER32,
                          station_=df_test.keys()[y][len(PARAMETER32)+1:],
                          df_dists_=df_station_dists,
                          power_=best_power_idw_tr) # берём лучшее значение степени, полученное из кода выше на тренинговом датасете
    for x, y in x_test
```

```

]
# y_predict_idw_vld

max_e_idw_vld = max_error(y_test, y_predict_idw_vld)
mae_idw_vld = mean_absolute_error(y_test, y_predict_idw_vld)
mse_idw_vld = mean_squared_error(y_test, y_predict_idw_vld)
rmse_idw_vld = mean_squared_error(y_test, y_predict_idw_vld, squared=False)
r2_idw_vld = r2_score(y_test, y_predict_idw_vld)
print(f'ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld:.7f}')
)

```

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:

Максимальная ошибка (MAX\_E) IDW = 8.0291230  
 Средняя абсолютная ошибка (MAE) IDW = 0.9613331  
 Средний квадрат ошибки (MSE) IDW = 1.8916000  
 Средняя квадратическая ошибка (RMSE) IDW = 1.3753545  
 Коэффициент детерминации (R2) IDW = 0.9788415

## ВЫВОД

Модель средней, взвешенной по обратным расстояниям, изначально даёт очень хорошие метрики качества.

### 3.2.3.2. Кригинг и вариограммы в реализации библиотеки SciKit GStat

*Опробуем работу модели кригинга на уже сформированных тренинговой и валидационной выборках*

Координатная сетка для точек наблюдения в данной модели строится на основе "плоской" земной поверхности. Разность между расстояниями по прямой и по поверхности сферы игнорируется (впрочем, для нашего региона исследования это не приведёт к существенным ошибкам).

In [132...]

```

# Загружаем координаты из df_coords_full (определен в разделе определения функции кригинга 2.2):
# Создаём DF с координатами нужных нам точек
df_coords = df_coords_full[["LoE", "LaN"]][:-2]

df_coords

```

Out[132]:

LoE      NaN

station	LoE	NaN
V_Volochek	34.566667	57.583333
Staritsa	34.933333	56.500000
Kashyn	37.583333	57.350000
Tver	35.922000	56.857300
Klin	36.716667	56.333333
Dmitrov	37.533333	56.366667
Volokolamsk	35.933333	56.016700
Mozhaisk	36.000000	55.516700
N_Jerusalem	36.816667	55.900000
Nemchinovka	37.350000	55.716667
Naro_Fominsk	36.700000	55.383333
Serpukhov	37.416667	54.916667

## Обучающий датасет

Проверим работу модели кригинга сначала на данных обучающей выборки. На ней будем подбирать наиболее подходящие параметры вариограмм и модели кригинга (которые дают лучшие метрики и выдают меньшее количество ошибок из-за недостаточности наблюдений в поле метеостанций).

Представляется полезным сравнить работу модели обратных степеней расстояний и кригинга на одних и тех же данных.

В процессе исследования работоспособности модели код ниже многократно запускался с различными параметрами.

In [133...]

```
# Намеренно оставим закомментированные части кода, они могут использоваться для отладки
start_time = time.time() # для замера времени выполнения кода
# counter = 0
y_predict_kriging_tr = [] # список предиктов для x_test
```

```

for x in x_train: # x[0] ряд в df_test, x[1] столбец, соответствующий названию станции
#     counter += 1
##
#     if counter >15:
#         break
#     else:
#         counter +=1
##
# Найдем по координатам x_test ряд в df_test
row = df_test.iloc[x[0]]
# Присваиваем проверяющему значению NaN
row.iloc[x[1]] = np.nan
# Для построения вариограммы:
# - получаем массив координат без указанной точки
# (удаляем соответствующий ряд из df_coords, преобразуем оставшийся df в массив)
# - получаем массив значений
idx = x[1]
coords_v = np.array(df_coords.drop(index = df_coords.iloc[x[1]].name))[:, :2]
vals_v = np.array(row.dropna())
try:
    # Определяем вариограмму
    V = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.99999, # Используем всю матрицу расстояний
                        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                        normalize=False,
                        use_nugget=False,
                        samples=len(vals_v) # количество сэмплов приравняем к количеству наблюдений
                        #fit_method='ml',
                        #entropy_bins = 1
                        )
    #     V_North = skg.DirectionalVariogram(coordinates=coords_v,
    #                                         values=vals_v,
    #                                         estimator='matheron',
    #                                         model='spherical',
    #                                         dist_func='euclidean',
    #                                         bin_func='even',
    #                                         azimuth=90,
    #                                         tolerance=90,
    #                                         maxlag='full',
    #                                         )

```

```

#                                     n_Lags=4)
# V_East = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=0,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)
# V_South = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=-90,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)
# V_West = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=180,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)

##
# V=V_West
##
```

**except ValueError:** # Уберём количество сэмплов из определения вариограммы (когда количество сэмплов слишком велико)

```

try:
    V_ = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.9999, # Используем всю матрицу расстояний

```

```

        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
        normalize=False,
        use_nugget=False,
        #fit_method='ml',
        #entropy_bins = 1
    );
except: # Возникает ошибка - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_tr.append(val_predict)
    continue
except: # возникает другая ошибка (помимо ValueError) - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_tr.append(val_predict)
    continue

# Определяем модель кригинга
model_ok = skg.OldinaryKriging(V, min_points=1, max_points=14, mode='exact')

# координаты точки предикта:
predict_coords = np.array(df_coords)
# Получаем предикт для тестируемой точки (получаем массив предиктов, выбираем из него индекс точки предикта)
# В процессе работы model_ok.transform выдается много сообщений типа:
# 'Warning: for %d Locations, not enough neighbors were found within the range.' % self.no_points_error
# "Заглушим" их, направив их в класс вывода, который ничего не делает (определен выше)

sys.stdout = NullIO() # определим вывод print() в класс нулевого вывода

val_predict = model_ok.transform(predict_coords[:,0], predict_coords[:,1])[x[1]]
sys.stdout = real_stdout # восстановим стандартный вывод

# добавляем предикт в список предиктов
y_predict_kriging_tr.append(val_predict)

## 
# if V.describe()['effective_range'] < 1:
#     dm = pdist(df_coords[['LoE', 'LaN']]) # массив пар расстояний
#     print(f'Итерация: {counter}, позиция в x_test: {x}\nКоличество пар расстояний: {len(dm)}\n'
#           f'Effective Range: {V.describe()["effective_range"]}\n'
#           f'Количество пар расстояний внутри Effective range: {sum(dm <= V.describe()["effective_range"])}\n'
#           f'Количество пар расстояний вне Effective range: {sum(dm > V.describe()["effective_range"])}\n'
#           f'Предикт: {val_predict}, координаты предикта: {predict_coords[x[1]]}, станция: {df_coords.iloc[x[1]].name}'
#           )
#     fig = V.plot(show=False)

```

```

#         plt.show()
## 
y_predict_kriging_tr = np.array(y_predict_kriging_tr) # превратим список предиктов в массив

# замер времени:
chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

```

```

In [134... if np.isnan(np.array(y_predict_kriging_tr)).sum() == 0:
    max_e_kriging_tr = max_error(y_train, y_predict_kriging_tr)
    mae_kriging_tr = mean_absolute_error(y_train, y_predict_kriging_tr)
    mse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr)
    rmse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr, squared=False)
    r2_kriging_tr = r2_score(y_train, y_predict_kriging_tr)
else:
    max_e_kriging_tr = np.nan
    mae_kriging_tr = np.nan
    mse_kriging_tr = np.nan
    rmse_kriging_tr = np.nan
    r2_kriging_tr = np.nan

print('ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ')
print(f'Elapsed time={time_formatted}\n'
      f'Значения метрик:\n'
      f'R2={r2_kriging_tr:.7f}, MAX_E={max_e_kriging_tr}, MAE={mae_kriging_tr:.7f}, MSE={mse_kriging_tr}, '
      f'RMSE={rmse_kriging_tr:.7f}\n'
      )

print(f'Количество значений, которые не удалось предсказать: {pd.isna([y_predict_kriging_tr]).sum()}\n'
      f'Это составляет {pd.isna([y_predict_kriging_tr]).sum()/len(y_predict_kriging_tr):.4%} от обучающего массива данных')

```

ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ  
Elapsed time=00:00:56  
Значения метрик:  
R2=nan, MAX\_E=nan, MAE=nan, MSE=nan, RMSE=nan

Количество значений, которые не удалось предсказать: 159  
Это составляет 5.1340% от обучающего массива данных

Попробуем оценить качество работы модели в случаях, когда ей удалось найти предикты.

In [135...]

```
# Оставим в y_train и y_predict_kriging_tr только значения неравные NaN
y_train_kriging_shrunk = y_train[~np.isnan(y_predict_kriging_tr)]
y_predict_kriging_tr_shrunk = y_predict_kriging_tr[~np.isnan(y_predict_kriging_tr)]

max_e_kriging_tr = max_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
mae_kriging_tr = mean_absolute_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
mse_kriging_tr = mean_squared_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
rmse_kriging_tr = mean_squared_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk, squared=False)
r2_kriging_tr = r2_score(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
```

In [136...]

```
print(f'КРИГИНГ на ОБУЧАЮЩЕМ датасете (для случаев, где удалось найти предикты):\n'
      f'Максимальная ошибка (MAX_E) Kriging = {max_e_kriging_tr:.7f}, Kriging - IDW = '
      f'{(max_e_kriging_tr - max_e_idw_tr):.7f}\n'
      f'Средняя абсолютная ошибка (MAE) Kriging = {mae_kriging_tr:.7f}, Kriging - IDW = '
      f'{(mae_kriging_tr - mae_idw_tr):.7f}\n'
      f'Средний квадрат ошибки (MSE) Kriging = {mse_kriging_tr:.7f}, Kriging - IDW = '
      f'{(mse_kriging_tr - mse_idw_tr):.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) Kriging = {rmse_kriging_tr:.7f}, '
      f'Kriging - IDW = {(rmse_kriging_tr - rmse_idw_tr):.7f}\n'
      f'Коэффициент детерминации (R2) Kriging = {r2_kriging_tr:.7f}, Kriging - IDW = '
      f'{(r2_kriging_tr - r2_idw_tr):.7f}')
)
```

КРИГИНГ на ОБУЧАЮЩЕМ датасете (для случаев, где удалось найти предикты):

Максимальная ошибка (MAX\_E) Kriging = 10.7983493, Kriging - IDW = 0.1101695  
Средняя абсолютная ошибка (MAE) Kriging = 0.8866738, Kriging - IDW = -0.0626443  
Средний квадрат ошибки (MSE) Kriging = 1.6380413, Kriging - IDW = -0.2141223  
Средняя квадратическая ошибка (RMSE) Kriging = 1.2798599, Kriging - IDW = -0.0810823  
Коэффициент детерминации (R2) Kriging = 0.9817774, Kriging - IDW = 0.0022121

## Валидационный датасет

Посмотрим, как модель будет отрабатывать на валидационной выборке.

In [137...]

```
start_time = time.time() # для замера времени выполнения кода
# counter = 0
y_predict_kriging_vld = [] # список предиктов для x_test

for x in x_test: # x[0] ряд в df_test, x[1] столбец, соответствующий названию станции
    #     counter += 1
    ##     if counter >15:
```

```

#         break
#     else:
#         counter +=1
##
# Найдем по координатам x_test ряд в df_test
row = df_test.iloc[x[0]]
# Присваиваем проверяемому значению NaN
row.iloc[x[1]] = np.nan
# Для построения вариограммы:
# - получаем массив координат без указанной точки
# (удаляем соответствующий ряд из df_coords, преобразуем оставшийся df в массив)
# - получаем массив значений
idx = x[1]
coords_v = np.array(df_coords.drop(index = df_coords.iloc[x[1]].name))[:, :2]
vals_v = np.array(row.dropna())

try:
    # Определяем вариограмму
    V = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.99999, # Используем всю матрицу расстояний
                        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                        normalize=False,
                        use_nugget=False,
                        samples=len(vals_v),
                        fit_method='trf',
                        )
except ValueError: # Уберём количество сэмплов из определения вариограммы (когда количество сэмплов слишком велико)
    try:
        V_ = skg.Variogram(coordinates=coords_v,
                            values=vals_v,
                            estimator='matheron',
                            model='spherical',
                            dist_func='euclidean',
                            bin_func='ward',
                            maxlag=0.99999, # Используем всю матрицу расстояний
                            n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                            normalize=False,
                            use_nugget=False,
                            #fit_method='ml',
    
```

```

        #entropy_bins = 1
    );
except: # Возникает ошибка - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_vld.append(val_predict)
    continue
except: # возникает другая ошибка (помимо ValueError) - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_vld.append(val_predict)
    continue

# Определяем модель кригинга
model_ok = skg.OldinaryKriging(V, min_points=1, max_points=14, mode='exact')

# координаты точки предикта:
predict_coords = np.array(df_coords)
# Получаем предикт для тестируемой точки (получаем массив предиктов, выбираем из него индекс точки предикта)
# В процессе работы model_ok.transform выдается много сообщений типа:
# 'Warning: for %d Locations, not enough neighbors were found within the range.' % self.no_points_error
# "Заглушим" их, направив их в класс вывода, который ничего не делает (определен выше)

sys.stdout = NullIO() # определим вывод print() в класс нулевого вывода

val_predict = model_ok.transform(predict_coords[:,0], predict_coords[:,1])[x[1]]
sys.stdout = real_stdout # восстановим стандартный вывод

# добавляем предикт в список предиктов
y_predict_kriging_vld.append(val_predict)

y_predict_kriging_vld = np.array(y_predict_kriging_vld)

# замер времени:
chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

```

In [138...]

```

if np.isnan(np.array(y_predict_kriging_vld)).sum() == 0:
    max_e_kriging_vld = max_error(y_test, y_predict_kriging_vld)
    mae_kriging_vld = mean_absolute_error(y_test, y_predict_kriging_vld)
    mse_kriging_vld = mean_squared_error(y_test, y_predict_kriging_vld)
    rmse_kriging_vld = mean_squared_error(y_test, y_predict_kriging_vld, squared=False)
    r2_kriging_vld = r2_score(y_test, y_predict_kriging_vld)

```

```

else:
    max_e_kriging_vld = np.nan
    mae_kriging_vld = np.nan
    mse_kriging_vld = np.nan
    rmse_kriging_vld = np.nan
    r2_kriging_vld = np.nan

print(f'Elapsed time={time_formatted}\n'
      f'Значения метрик:\n'
      f'R2={r2_kriging_vld:.7f}, MAX_E={max_e_kriging_vld:.7f}, MSE={mse_kriging_vld:.7f}, '
      f'RMSE={rmse_kriging_vld:.7f}\n')
print('ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ')
print(f'Количество значений, которые не удалось предсказать: {np.isnan([y_predict_kriging_vld]).sum()}\n'
      f'Это составляет {pd.isna([y_predict_kriging_vld]).sum()/len(y_predict_kriging_vld):.4%} '
      f'от валидационного массива данных')

```

Elapsed time=00:00:37

Значения метрик:

R2=nan, MAX\_E=nan, MAE=nan, MSE=nan, RMSE=nan

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ

Количество значений, которые не удалось предсказать: 110

Это составляет 5.3269% от валидационного массива данных

In [139...]

```

y_test_kriging_shrunk = y_test[~np.isnan(y_predict_kriging_vld)]
y_predict_kriging_vld_shrunk = y_predict_kriging_vld[~np.isnan(y_predict_kriging_vld)]

max_e_kriging_vld = max_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
mae_kriging_vld = mean_absolute_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
mse_kriging_vld = mean_squared_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
rmse_kriging_vld = mean_squared_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk, squared=False)
r2_kriging_vld = r2_score(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)

```

In [140...]

```

print(f'Кригинг на ВАЛИДАЦИОННОМ датасете (для случаев, где удалось найти предикты):\n'
      f'Максимальная ошибка (MAX_E) Kriging = {max_e_kriging_vld:.7f}, Kriging - IDW = '
      f'{(max_e_kriging_vld - max_e_idw_vld):.7f}\n'
      f'Средняя абсолютная ошибка (MAE) Kriging = {mae_kriging_vld:.7f}, '
      f'Kriging - IDW = {(mae_kriging_vld - mae_idw_vld):.7f}\n'
      f'Средний квадрат ошибки (MSE) Kriging = {mse_kriging_vld:.7f}, Kriging - IDW = '
      f'{(mse_kriging_vld - mse_idw_vld):.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) Kriging = {rmse_kriging_vld:.7f}, '
      f'Kriging - IDW = {(rmse_kriging_vld - rmse_idw_vld):.7f}\n'
      f'Коэффициент детерминации (R2) Kriging = {r2_kriging_vld:.7f}, Kriging - IDW = '

```

```
f'{(r2_kriging_vld - r2_idw_vld):.7f}'

)
```

Кригинг на ВАЛИДАЦИОННОМ датасете (для случаев, где удалось найти предикты):  
Максимальная ошибка (MAX\_E) Kriging = 7.5000000, Kriging - IDW = -0.5291230  
Средняя абсолютная ошибка (MAE) Kriging = 0.8891294, Kriging - IDW = -0.0722036  
Средний квадрат ошибки (MSE) Kriging = 1.6819417, Kriging - IDW = -0.2096583  
Средняя квадратическая ошибка (RMSE) Kriging = 1.2968969, Kriging - IDW = -0.0784576  
Коэффициент детерминации (R2) Kriging = 0.9809893, Kriging - IDW = 0.0021477

## ВЫВОД

Таким образом, модель кригинга не даёт 100% удовлетворительных результатов, так как в ряде случаев не может предсказать значения, и оказывается неприменимой. Это связано с незначительным размером поля метеостанций и встречающейся ситуацией, когда не удаётся найти ближайшие значения в effective range (расстоянии, вне пределов которого модель считает данные метеостанций статистически независимыми). То есть на этом и большем расстоянии корреляция между значениями показателя у метеостанций становится незначительной или отсутствует.

### 3.2.3.3. Модель кригинга, совмещённая с IDW (для значений, которые кригинг не может предсказать)

Отделим значения, которые не удалось предсказать моделью кригинга, от уже предсказанных значений и формируем новый (сокращённый) обучающий датасет для IDW

In [141...]

```
# Поскольку длины массивов x_train, y_train и y_predict_tr, а также x_test, y_test и y_predict_vld соответственно одинаковы,
# выбираем по индексу значения x_train и y_train, x_test и y_test
# где значения y_predict_kriging_tr равны NaN
# формируем новый массив истинных значений
y_train_idw_shrunk = y_train[np.isnan(y_predict_kriging_tr)]
x_train_idw_shrunk = x_train[np.isnan(y_predict_kriging_tr)]
y_test_idw_shrunk = y_test[np.isnan(y_predict_kriging_vld)]
x_test_idw_shrunk = x_test[np.isnan(y_predict_kriging_vld)]
```

Снова подберём лучшую степень для IDW

In [142...]

```
start_time = time.time() # для замера времени выполнения кода

r2_idw_tr_shrunk = 0 # обнулим значение метрики качества R2
iterations = 0 # количество итераций
power = 1.75 # Начальное значение степени (опробованы начальные степени от 1 до 10)
power_increment = 0.25 # шаг увеличения степени
```

```

list_metrics=[] # список для фиксации результатов итераций

r2_idw_tr_shrunk_old = 0 # Устанавливаем в 0 предыдущее значение R2
print('НОВЫЙ ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:\n')

# Зададим предел R2 для поиска оптимального веса
while r2_idw_tr_shrunk_old <= r2_idw_tr_shrunk:
    power += power_increment # Увеличиваем степень на 1 шаг
    iterations +=1 # Увеличиваем счётчик проходов цикла
    r2_idw_tr_shrunk_old = r2_idw_tr_shrunk # Запоминаем предыдущее значение R2

    # Создаём y_predict_idw_tr_shrunk по индексам в массиве x_train_shrunk
    # используем функцию inverse_distance_avg
    # Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком
    y_predict_idw_tr_shrunk = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param_=PARAMETER32,
                             station_=df_test.keys()[y][len(PARAMETER32)+1:],
                             df_dists_= df_station_dists,
                             power_=power)
        for x, y in x_train_idw_shrunk
    ]

    # Расчитываем метрики качества
    max_e_idw_tr_shrunk = max_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    mae_idw_tr_shrunk = mean_absolute_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    mse_idw_tr_shrunk = mean_squared_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    rmse_idw_tr_shrunk = mean_squared_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk, squared=False)
    r2_idw_tr_shrunk = r2_score(y_train_idw_shrunk, y_predict_idw_tr_shrunk)

    if r2_idw_tr_shrunk > r2_idw_tr_shrunk_old:
        best_power_idw_tr_shrunk = power
        best_max_e_idw_tr_shrunk = max_e_idw_tr_shrunk
        best_mae_idw_tr_shrunk = mae_idw_tr_shrunk
        best_mse_idw_tr_shrunk = mse_idw_tr_shrunk
        best_rmse_idw_tr_shrunk = rmse_idw_tr_shrunk
        best_r2_idw_tr_shrunk = r2_idw_tr_shrunk

    # замер времени:
    chk_time = time.time()
    elapsed_time = chk_time - start_time
    time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

print(f'Elapsed time={time_formatted}\n'

```

```
f'Текущие значения: iterations={iterations}, power={power},\n'
f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_tr_shrunk:.7f}\n'
f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_tr_shrunk:.7f}\n'
f'Средний квадрат ошибки (MSE) IDW = {mse_idw_tr_shrunk:.7f}\n'
f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_tr_shrunk:.7f}\n'
f'Коэффициент детерминации (R2) IDW = {r2_idw_tr_shrunk:.7f}\n'
)
print(f'ЛУЧШИЕ значения: power={best_power_idw_tr_shrunk},\n'
f'Максимальная ошибка (MAX_E) IDW = {best_max_e_idw_tr_shrunk:.7f}\n'
f'Средняя абсолютная ошибка (MAE) IDW = {best_mae_idw_tr_shrunk:.7f}\n'
f'Средний квадрат ошибки (MSE) IDW = {best_mse_idw_tr_shrunk:.7f}\n'
f'Средняя квадратическая ошибка (RMSE) IDW = {best_rmse_idw_tr_shrunk:.7f}\n'
f'Коэффициент детерминации (R2) IDW = {best_r2_idw_tr_shrunk:.7f}'
```

НОВЫЙ ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:

Elapsed time=00:00:00

Текущие значения: iterations=1, power=2.0,  
Максимальная ошибка (MAX\_E) IDW = 4.9099820  
Средняя абсолютная ошибка (MAE) IDW = 0.9973685  
Средний квадрат ошибки (MSE) IDW = 1.7223001  
Средняя квадратическая ошибка (RMSE) IDW = 1.3123643  
Коэффициент детерминации (R2) IDW = 0.9833784

Elapsed time=00:00:01

Текущие значения: iterations=2, power=2.25,  
Максимальная ошибка (MAX\_E) IDW = 4.9010789  
Средняя абсолютная ошибка (MAE) IDW = 0.9902585  
Средний квадрат ошибки (MSE) IDW = 1.6916745  
Средняя квадратическая ошибка (RMSE) IDW = 1.3006439  
Коэффициент детерминации (R2) IDW = 0.9836740

Elapsed time=00:00:01

Текущие значения: iterations=3, power=2.5,  
Максимальная ошибка (MAX\_E) IDW = 4.8867338  
Средняя абсолютная ошибка (MAE) IDW = 0.9842385  
Средний квадрат ошибки (MSE) IDW = 1.6662379  
Средняя квадратическая ошибка (RMSE) IDW = 1.2908284  
Коэффициент детерминации (R2) IDW = 0.9839195

Elapsed time=00:00:02

Текущие значения: iterations=4, power=2.75,  
Максимальная ошибка (MAX\_E) IDW = 4.8678624  
Средняя абсолютная ошибка (MAE) IDW = 0.9795173  
Средний квадрат ошибки (MSE) IDW = 1.6457901  
Средняя квадратическая ошибка (RMSE) IDW = 1.2828835  
Коэффициент детерминации (R2) IDW = 0.9841168

Elapsed time=00:00:03

Текущие значения: iterations=5, power=3.0,  
Максимальная ошибка (MAX\_E) IDW = 4.8453721  
Средняя абсолютная ошибка (MAE) IDW = 0.9757826  
Средний квадрат ошибки (MSE) IDW = 1.6300161  
Средняя квадратическая ошибка (RMSE) IDW = 1.2767209  
Коэффициент детерминации (R2) IDW = 0.9842691

Elapsed time=00:00:04

Текущие значения: iterations=6, power=3.25,

Максимальная ошибка (MAX\_E) IDW = 4.8201291  
Средняя абсолютная ошибка (MAE) IDW = 0.9725600  
Средний квадрат ошибки (MSE) IDW = 1.6185182  
Средняя квадратическая ошибка (RMSE) IDW = 1.2722100  
Коэффициент детерминации (R2) IDW = 0.9843800

Elapsed time=00:00:04  
Текущие значения: iterations=7, power=3.5,  
Максимальная ошибка (MAX\_E) IDW = 4.7929320  
Средняя абсолютная ошибка (MAE) IDW = 0.9703315  
Средний квадрат ошибки (MSE) IDW = 1.6108467  
Средняя квадратическая ошибка (RMSE) IDW = 1.2691913  
Коэффициент детерминации (R2) IDW = 0.9844541

Elapsed time=00:00:05  
Текущие значения: iterations=8, power=3.75,  
Максимальная ошибка (MAX\_E) IDW = 4.7644929  
Средняя абсолютная ошибка (MAE) IDW = 0.9686441  
Средний квадрат ошибки (MSE) IDW = 1.6065287  
Средняя квадратическая ошибка (RMSE) IDW = 1.2674891  
Коэффициент детерминации (R2) IDW = 0.9844957

Elapsed time=00:00:06  
Текущие значения: iterations=9, power=4.0,  
Максимальная ошибка (MAX\_E) IDW = 4.7354261  
Средняя абсолютная ошибка (MAE) IDW = 0.9673178  
Средний квадрат ошибки (MSE) IDW = 1.6050914  
Средняя квадратическая ошибка (RMSE) IDW = 1.2669220  
Коэффициент детерминации (R2) IDW = 0.9845096

Elapsed time=00:00:06  
Текущие значения: iterations=10, power=4.25,  
Максимальная ошибка (MAX\_E) IDW = 4.7062439  
Средняя абсолютная ошибка (MAE) IDW = 0.9663846  
Средний квадрат ошибки (MSE) IDW = 1.6060801  
Средняя квадратическая ошибка (RMSE) IDW = 1.2673121  
Коэффициент детерминации (R2) IDW = 0.9845001

ЛУЧШИЕ значения: power=4.0,  
Максимальная ошибка (MAX\_E) IDW = 4.7354261  
Средняя абсолютная ошибка (MAE) IDW = 0.9673178  
Средний квадрат ошибки (MSE) IDW = 1.6050914  
Средняя квадратическая ошибка (RMSE) IDW = 1.2669220  
Коэффициент детерминации (R2) IDW = 0.9845096

Опробуем новое значение лучшей степени для IDW на сокращённом валидационном датасете

In [143...]

```
# Создаём y_predict_idw_vld_shrunk по индексам в x_test_shrunk
# используем функцию inverse_distance_avg
# Проходим по массиву и считываем из df_test данные по координатам, выводим результат списком

y_predict_idw_vld_shrunk = [
    inverse_distance_avg(row=df_test.iloc[x],
                          param=PARAMETER32,
                          station=df_test.keys()[y][len(PARAMETER32)+1:],
                          df_dists=df_station_dists,
                          power=best_power_idw_tr_shrunk) # берём лучшее значение степени, полученное из кода выше
    for x, y in x_test_idw_shrunk
]
# y_predict_idw_vld_shrunk

max_e_idw_vld_shrunk = max_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
mae_idw_vld_shrunk = mean_absolute_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
mse_idw_vld_shrunk = mean_squared_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
rmse_idw_vld_shrunk = mean_squared_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk, squared=False)
r2_idw_vld_shrunk = r2_score(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
print(f'ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld_shrunk:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld_shrunk:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld_shrunk:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld_shrunk:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld_shrunk:.7f}'
      )
```

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:

Максимальная ошибка (MAX\_E) IDW = 4.9311967  
Средняя абсолютная ошибка (MAE) IDW = 1.1431525  
Средний квадрат ошибки (MSE) IDW = 2.2184366  
Средняя квадратическая ошибка (RMSE) IDW = 1.4894417  
Коэффициент детерминации (R2) IDW = 0.9790424

## Метрики обучающего и валидационного датасетов для совместной работы двух моделей

Данные работы моделей на своей части обучающего датасета у нас есть. Подсчитаем метрики для общего датасета

In [144...]

```
# Восстановим y_predict для лучшего R2 IDW
y_predict_idw_tr_shrunk = [
```

```
        inverse_distance_avg(row_=df_test.iloc[x],
                             param_=PARAMETER32,
                             station_=df_test.keys()[-len(PARAMETER32)+1:],
                             df_dists_=df_station_dists,
                             power_=best_power_idw_tr_shrunk)
    for x, y in x_train_idw_shrunk
]
```

In [145...]

```
# последовательно соединим датасеты для кригинга (там, где есть предсказания) и для IDW
x_train_2 = np.append(x_train[~np.isnan(y_predict_kriging_tr)], x_train_idw_shrunk)
y_train_2 = np.append(y_train_kriging_shrunk, y_train_idw_shrunk)
y_predict_tr_2 = np.append(y_predict_kriging_tr_shrunk, y_predict_idw_tr_shrunk)
x_test_2 = np.append(x_test[~np.isnan(y_predict_kriging_vld)], x_test_idw_shrunk)
y_test_2 = np.append(y_test_kriging_shrunk, y_test_idw_shrunk)
y_predict_vld_2 = np.append(y_predict_kriging_vld_shrunk, y_predict_idw_vld_shrunk)
```

In [146...]

```
# Расчитываем метрики качества
max_e_2_tr = max_error(y_train_2, y_predict_tr_2)
mae_2_tr = mean_absolute_error(y_train_2, y_predict_tr_2)
mse_2_tr = mean_squared_error(y_train_2, y_predict_tr_2)
rmse_2_tr = mean_squared_error(y_train_2, y_predict_tr_2, squared=False)
r2_2_tr = r2_score(y_train_2, y_predict_tr_2)

max_e_2_vld = max_error(y_test_2, y_predict_vld_2)
mae_2_vld = mean_absolute_error(y_test_2, y_predict_vld_2)
mse_2_vld = mean_squared_error(y_test_2, y_predict_vld_2)
rmse_2_vld = mean_squared_error(y_test_2, y_predict_vld_2, squared=False)
r2_2_vld = r2_score(y_test_2, y_predict_vld_2)
```

In [147...]

```
print(f'ОБЩИЙ ОБУЧАЮЩИЙ ДАТАСЕТ, Кригинг + IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_2_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_2_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_2_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_2_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_2_tr:.7f}\n'
      )
print(f'ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_2_vld:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_2_vld:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_2_vld:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_2_vld:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_2_vld:.7f}\n'
```

```
)  
  
print(f'Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:\n'  
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld:.7f}\n'  
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld:.7f}\n'  
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld:.7f}\n'  
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld:.7f}\n'  
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld:.7f}'  
)
```

ОБЩИЙ ОБУЧАЮЩИЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX\_E) IDW = 10.7983493  
Средняя абсолютная ошибка (MAE) IDW = 0.8908140  
Средний квадрат ошибки (MSE) IDW = 1.6363497  
Средняя квадратическая ошибка (RMSE) IDW = 1.2791988  
Коэффициент детерминации (R2) IDW = 0.9819463

ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX\_E) IDW = 7.5000000  
Средняя абсолютная ошибка (MAE) IDW = 0.9026609  
Средний квадрат ошибки (MSE) IDW = 1.7105201  
Средняя квадратическая ошибка (RMSE) IDW = 1.3078685  
Коэффициент детерминации (R2) IDW = 0.9808670

Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:

Максимальная ошибка (MAX\_E) IDW = 8.0291230  
Средняя абсолютная ошибка (MAE) IDW = 0.9613331  
Средний квадрат ошибки (MSE) IDW = 1.8916000  
Средняя квадратическая ошибка (RMSE) IDW = 1.3753545  
Коэффициент детерминации (R2) IDW = 0.9788415

## ВЫВОД

Там, где с помощью кригинга удаётся получить предсказания, он превосходит по метрикам качества модель IDW. При этом сочетание этих двух моделей (применение IDW там, где кригинг оказывается бессильным) даёт в целом лучшее качество предсказания, чем только модель IDW. Поэтому, ниже применим комбинацию этих двух моделей к уже реальному датасету..

### 3.2.4. Применение выбранных модели для исправления, восстановления данных и получения предиктов показателя минимальной температуры T\_min для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

In [148..]

```
# Добавим в df_tmp32 столбцы для будущих значений для Чашниково и центральной точки, заполним их NaN
# - это необходимо, чтобы вычислить значения для архивов
# Создадим столбцы col_name со значениями pr.nan
# Переименуем столбец PARAMETER32#Chashnikovo и PARAMETER32#Rfrnce_point
df_tmp32 = (df_tmp32.
             assign(col_name1 = np.nan,
                   col_name2 = np.nan).
             rename(columns={"col_name1": PARAMETER32+'#'+ 'Chashnikovo',
                            "col_name2": PARAMETER32+'#'+ 'Rfrnce_point'}))
         )

df_tmp32.sample(3, random_state=56)
```

Out[148]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2014-02-22 03:00:00	-3.7	-2.8	-3.0	-4.4	-7.7	-4.8	-2.9	-2.4	
2015-05-16 03:00:00	8.6	8.1	9.5	8.0	8.3	9.8	7.7	7.9	
2020-06-18 18:00:00	23.2	23.1	18.1	22.7	23.5	23.4	27.0	25.8	

In [149..]

```
# Добавим в архив параметров T_min столбцы для будущих значений Чашниково и центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями pr.nan
# Переименуем столбец PARAMETER32#Chashnikovo и PARAMETER32#Rfrnce_point
dict_df_parameters['df_'+PARAMETER32] = (dict_df_parameters['df_'+PARAMETER32].
                                         assign(col_name1 = np.nan,
                                               col_name2 = np.nan).
                                         rename(columns={"col_name1": PARAMETER32+'#'+ 'Chashnikovo',
                                                        "col_name2": PARAMETER32+'#'+ 'Rfrnce_point'}))
                                         )

dict_df_parameters['df_'+PARAMETER32].sample(3, random_state=56)
```

Out[149]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
--	------------------	----------------	--------------	------------	------------	---------------	-------------------	----------------	-------------

<b>2014-02-22 03:00:00</b>	NaN								
<b>2015-05-16 03:00:00</b>	NaN								
<b>2020-06-18 18:00:00</b>	NaN								

In [150...]

```
# # Восстановление архивов Чашниково и центральной точки:
# dict_df_locations['df_Chashnikovo'] = pd.read_csv(
#     new_raw_path1 + '/' + 'df_Chashnikovo' + PARAMETER31 + '.csv',
#     index_col=0, parse_dates=True, infer_datetime_format = True, low_memory=False)
# print('df_Chashnikovo_T.csv -> O.K.')

# dict_df_locations['df_Rfrnce_point'] = pd.read_csv(
#     new_raw_path1 + '/' + 'df_Rfrnce_point' + PARAMETER31 + '.csv',
#     index_col=0, parse_dates=True, infer_datetime_format = True, low_memory=False)
# print('df_Rfrnce_point_T.csv -> O.K.')
```

In [151...]

```
# В архивах метеостанций df Чашниково и df для центральной точки
# создадим столбцы с называнием PARAMETER32

dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER32})
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER32}
                                         )
                                         )

dict_df_locations['df_Chashnikovo'].sample(3, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(3, random_state=56)
```

Out[151]:

	T	T_min
2014-02-22 03:00:00	-4.156558	NaN
2015-05-16 03:00:00	9.181730	NaN
2020-06-18 18:00:00	27.439052	NaN

Out[151]:

	T	T_min
2014-02-22 03:00:00	-3.589183	NaN
2015-05-16 03:00:00	7.789650	NaN
2020-06-18 18:00:00	29.404954	NaN

**Заполнение значений минимальной температуры T\_min для Чашниково и условной средней точки скользящими минимумами (аналогично заполнению строк сплошных NaN)**

In [152...]

```
# построим датафрейм с минимальными значениями температуры по моментам наблюдения за предшествующие 12 часов
# для условных метеостанций
# Используем скользящее окно с вычислением минимума и со сдвигом вверх на 4 строки (-4),
# чтобы обозначить минимумы за предшествующие периоды.
df_t1a = df_tmp31.iloc[:, -2:].rolling(window='12H', center=False, closed='left').min().shift(-4)
df_t1a.sample(7, random_state=56)
```

Out[152]:

	T#Chashnikovo	T#Rfrnce_point
2014-02-22 03:00:00	-4.156558	-3.794602
2015-05-16 03:00:00	9.181730	7.789650
2020-06-18 18:00:00	25.366130	25.572651
2019-12-02 21:00:00	-4.229056	-3.796107
2008-06-05 18:00:00	12.255838	11.754644
2016-10-20 06:00:00	0.117681	-0.382597
2006-09-11 03:00:00	9.855422	9.181940

In [153]:

```
# Выберем значения из df_t1a, соответствующие индексам строк сплошных NaN в df_tmp32 и присвоим их ячейкам df_tmp32
# используем .loc, чтобы избежать предупреждения о присовении значений для копии данных.
df_tmp32.loc[:, PARAMETER32+'#Chashnikovo'][mask_total_nans] = \
df_t1a.loc[:, PARAMETER31+'#Chashnikovo'][df_t1a.index.isin(df_tmp32[mask_total_nans].index)]

df_tmp32.loc[:, PARAMETER32+'#Rfrnce_point'][mask_total_nans] = \
df_t1a.loc[:, PARAMETER31+'#Rfrnce_point'][df_t1a.index.isin(df_tmp32[mask_total_nans].index)]

df_tmp32.sample(7, random_state=56)
```

Out[153]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jei
2014-02-22 03:00:00	-3.700000	-2.8	-3.0	-4.4	-7.7	-4.8	-2.9	-2.4	
2015-05-16 03:00:00	8.600000	8.1	9.5	8.0	8.3	9.8	7.7	7.9	
2020-06-18 18:00:00	23.200000	23.1	18.1	22.7	23.5	23.4	27.0	25.8	
2019-12-02 21:00:00	-2.831996	-3.9	-4.1	-3.9	-4.0	-4.3	-3.6	-3.3	
2008-06-05 18:00:00	11.600000	11.5	11.3	12.1	11.9	11.8	11.7	12.6	
2016-10-20 06:00:00	0.100000	-0.1	0.7	0.2	0.1	0.6	-0.7	-0.7	
2006-09-11 03:00:00	9.200000	8.1	10.3	9.6	9.9	10.0	8.9	10.4	



Перенесение уже исправленных сплошных NaN в архивы

In [154...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER32] = df_tmp32.copy(deep=True)

# Перенесём уже исправленные значения в dict_df_locations, оставшиеся NaN исправим ниже
for name_df in dict_df_locations.keys():
    dict_df_locations[name_df].loc[:, PARAMETER32] = df_tmp32.loc[:, PARAMETER32 + '#' + name_df[3:]]

dict_df_parameters['df_'+PARAMETER32].sample(7, random_state=56)
dict_df_locations['df_Chashnikovo'].sample(7, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(7, random_state=56)
```

Out[154]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2014-02-22 03:00:00	-3.700000	-2.8	-3.0	-4.4	-7.7	-4.8	-2.9	-2.4	
2015-05-16 03:00:00	8.600000	8.1	9.5	8.0	8.3	9.8	7.7	7.9	
2020-06-18 18:00:00	23.200000	23.1	18.1	22.7	23.5	23.4	27.0	25.8	
2019-12-02 21:00:00	-2.831996	-3.9	-4.1	-3.9	-4.0	-4.3	-3.6	-3.3	
2008-06-05 18:00:00	11.600000	11.5	11.3	12.1	11.9	11.8	11.7	12.6	
2016-10-20 06:00:00	0.100000	-0.1	0.7	0.2	0.1	0.6	-0.7	-0.7	
2006-09-11 03:00:00	9.200000	8.1	10.3	9.6	9.9	10.0	8.9	10.4	

Out[154]:

	T	T_min
2014-02-22 03:00:00	-4.156558	-4.156558
2015-05-16 03:00:00	9.181730	9.181730
2020-06-18 18:00:00	27.439052	25.366130
2019-12-02 21:00:00	-3.004683	-4.229056
2008-06-05 18:00:00	16.565949	12.255838
2016-10-20 06:00:00	0.117681	NaN
2006-09-11 03:00:00	9.855422	9.855422

Out[154]:

	T	T_min
2014-02-22 03:00:00	-3.589183	-3.794602
2015-05-16 03:00:00	7.789650	7.789650
2020-06-18 18:00:00	29.404954	25.572651
2019-12-02 21:00:00	-2.977448	-3.796107
2008-06-05 18:00:00	17.613615	11.754644
2016-10-20 06:00:00	-0.382597	NaN
2006-09-11 03:00:00	9.181940	9.181940

### Частичное заполнение оставшихся NaN расчётными значениями с помощью кригинга

Применим модель кригинга в цикле, пока количество оставшихся в датафрейме NaN перестанет уменьшаться. Этую будут значения, которые модель уже никак не сможет рассчитать. Их можно будет восстановить методом IDW.

In [155...]

```
start_time = time.time() # для замера времени выполнения кода

# Подсчитаем количество NaN в df_tmp32 (кроме бывших сплошных NaN) и присвоим их переменной old_nan_count
# np.isnan(df_tmp31).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
new_nan_count = np.sum(np.isnan(df_tmp32).sum()) # результат всегда будет >= 0
# Определим начальное значение для переменной для обновления количества оставшихся NaN
```

```

old_nan_count = new_nan_count
counter = 0 # определим счётчик

# заменим значения в архивах, на исправленные и вычисленные из данных в df_tmp31
# используем функцию row_nan_kriging_correct
# функция настроена таким образом, что при возникновении ошибки, связной с недостаточностью данных,
# осуществляется выход из функции без возврата каких бы то ни было значений.
while (old_nan_count != new_nan_count) or (counter == 0):
    counter += 1
    print (f'\nИтерация № {counter}: осталось NaN: {new_nan_count}')

    # Построчно применяем функцию обработки NaN
    df_tmp32.apply(lambda x: row_nan_kriging_correct(row=x,
                                                       name_param_=PARAMETER32
                                                       ),
                  axis=1
                 )
    old_nan_count = new_nan_count # сохраняем прежнее количество NaN в old_nan_count
    new_nan_count = np.sum(np.isnan(df_tmp32).sum()) # подсчитаем оставшиеся количество NaN

```

```

chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f'Elapsed time={time_formatted}\n')

```

Итерация № 1: осталось NaN: 33912

Out[155]:

2022-06-09 21:00:00	None
2022-06-09 18:00:00	None
2022-06-09 15:00:00	None
2022-06-09 12:00:00	None
2022-06-09 09:00:00	None
...	
2005-02-01 12:00:00	None
2005-02-01 09:00:00	None
2005-02-01 06:00:00	None
2005-02-01 03:00:00	None
2005-02-01 00:00:00	None

Length: 50704, dtype: object

Итерация № 2: осталось NaN: 4338

```
Out[155]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 3: осталось NaN: 3452  
Out[155]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 4: осталось NaN: 3390  
Out[155]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 5: осталось NaN: 3381
```

```
Out[155]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 6: осталось NaN: 3376  
Out[155]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 7: осталось NaN: 3373  
Out[155]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Elapsed time=00:03:57
```

Подсчитаем конечное количество оставшихся NaN, которые не могут быть рассчитаны моделью кригинга

```
In [156]: # np.isnan(df_tmp31).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp32).sum())
```

```
Out[156]: 3373
```

Восстановим оставшиеся значения через модель IDW. Используем для этого степень, полученную на обучающем датасете при совместном использовании модели IDW и модели кригинга.

```
In [157]: # Произведём вычисление отсутствующих значений в df_tmp32 через модель IDW.  
# Заменим соответствующие значения в архивах на вновь вычисленные.  
# используем функцию row_nan_idw_correct
```

```
start_time = time.time() # для замера времени выполнения кода  
  
# Построчно применяем функцию обработки NaN  
df_tmp32.apply(lambda x: row_nan_idw_correct(row_=x,  
                                              name_param_=PARAMETER32,  
                                              power_=best_power_idw_tr_shrunk),  
               axis=1  
)  
  
chk_time = time.time()  
elapsed_time = chk_time - start_time  
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))  
print(f'Elapsed time={time_formatted}\n')
```

```
Out[157]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Elapsed time=00:00:22
```

Подсчитаем окончательное количество NaN

```
In [158...]: # np.isnan(df_tmp31).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp32).sum())
```

```
Out[158]: 42
```

Всё корректно. 42 NaN находятся в 3х самых ранних моментах наблюдения. Они пустые.

Выведем, рандомные строки из df\_tmp32

```
In [159...]: df_tmp32.sample(15)
```

Out[159]:

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
<b>2010-11-09 21:00:00</b>	1.6	8.1	7.300000	7.3	9.7	8.9	9.800000	10.400000	
<b>2007-12-18 03:00:00</b>	-1.6	-1.4	-1.903946	-1.3	-1.8	-2.2	-1.800000	-2.200000	
<b>2021-10-08 09:00:00</b>	-2.5	-3.9	-2.800000	-2.2	-2.4	-1.8	-2.900000	-4.300000	
<b>2020-02-01 15:00:00</b>	-3.7	-3.5	-3.200000	-3.2	-1.7	-1.8	-2.700000	-2.200000	
<b>2017-01-06 06:00:00</b>	-28.5	-27.0	-29.500000	-28.7	-26.3	-26.8	-26.187992	-24.769770	
<b>2011-03-11 09:00:00</b>	-4.3	-7.5	-9.400000	-8.1	-9.4	-10.3	-10.600000	-10.500000	
<b>2021-09-01 03:00:00</b>	12.7	14.3	14.100000	14.2	16.6	17.6	15.800000	15.500000	
<b>2017-11-29 03:00:00</b>	-3.0	-3.7	-3.700000	-2.9	-4.1	-5.8	-4.500000	-5.600000	
<b>2012-05-02 21:00:00</b>	9.8	10.0	9.100000	10.1	9.8	8.8	9.000000	10.000000	
<b>2021-06-05 15:00:00</b>	10.3	11.0	12.800000	14.0	11.2	13.0	12.700000	12.800000	
<b>2013-05-20 09:00:00</b>	15.0	13.6	11.100000	12.7	13.7	13.4	12.400000	12.100000	

	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer
2021-12-31 21:00:00	-4.7	-5.5	-6.100000	-5.3	-5.7	-5.5	-5.900000	-5.400000	
2021-05-10 09:00:00	-1.8	-2.2	-0.600000	-1.5	-2.1	-0.2	-2.200000	-0.400000	
2018-11-16 06:00:00	-1.1	-0.9	-0.900000	-0.1	-4.2	-4.8	-3.902276	-5.648685	
2017-05-19 00:00:00	2.0	1.0	2.900000	1.7	3.4	6.3	2.143564	2.664419	

### 3.2.5. Визуализация графиков температуры T\_min для условной метеостанции Чашниково

In [160...]

```
# Построим список годов для графиков
list_years = df_tmp32.index.year.unique().tolist()
```

In [161...]

```
# Определим часы момента наблюдения для построения графиков
plot_hour1 = 9
plot_hour2 = 21
# Строим график в цикле для каждого года
for year in list_years:
    data1 = dict_df_parameters['df_T_min'][PARAMETER32+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_T_min'].index.year == year) & (dict_df_parameters['df_T_min'].index.hour == plot_hour1)]\n\n    data2 = dict_df_parameters['df_T_min'][PARAMETER32+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_T_min'].index.year == year) & (dict_df_parameters['df_T_min'].index.hour == plot_hour2)]\n\n    fig, ax = plt.subplots(figsize=(10, 4))\n    g1 = sns.lineplot(data=data1,\n                      color='blue',\n                      linewidth=0.5,\n                      ax=ax)\n    g2 = sns.lineplot(data=data2,\n                      color='red',\n                      linewidth=0.5,
```

```

    ax=ax)

# Добавляем легенду
blue_line = mlines.Line2D([], [], color='blue', label=f'{plot_hour1}:00', linewidth=0.5)
red_line = mlines.Line2D([], [], color='red', label=f'{plot_hour2}:00', linewidth=0.5)
dummy = ax.legend(handles=[red_line, blue_line])

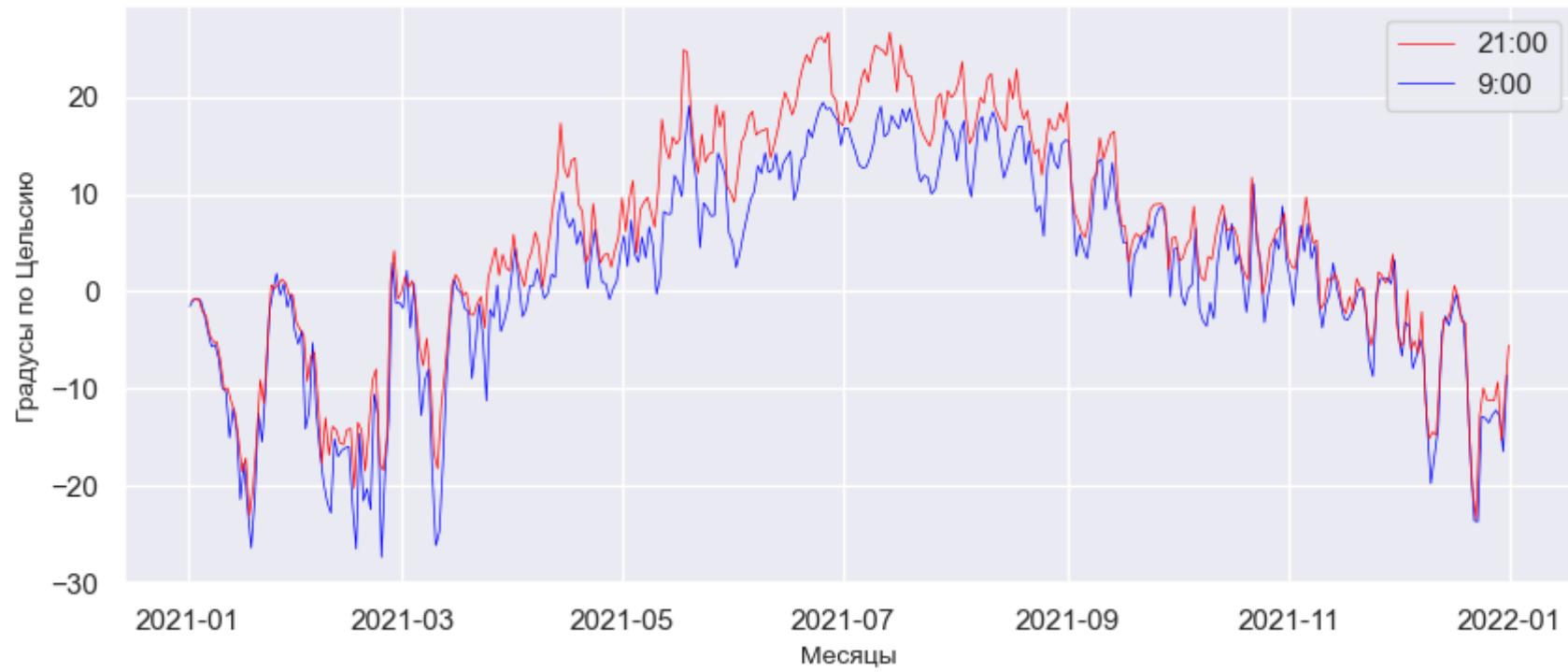
dummy = ax.set_ylabel('Градусы по Цельсию', size=10)
dummy = ax.set_xlabel('Месяцы', size=10)
dummy = plt.title(f'Чашниково: Ежедневная динамика параметра {PARAMETER32} за 12 предшествующих часов '
                  f'на {plot_hour1} и {plot_hour2} часов, {year} год')
plt.show()

```

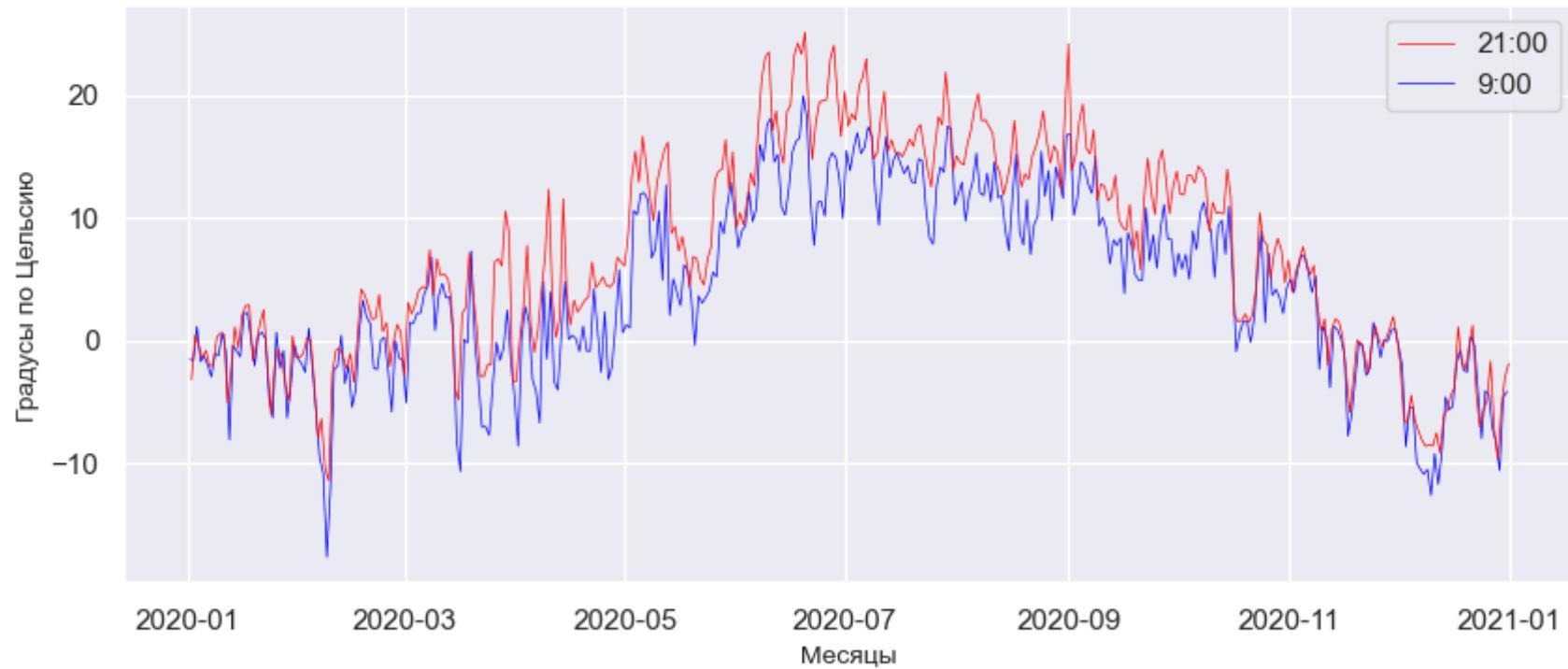
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2022 год



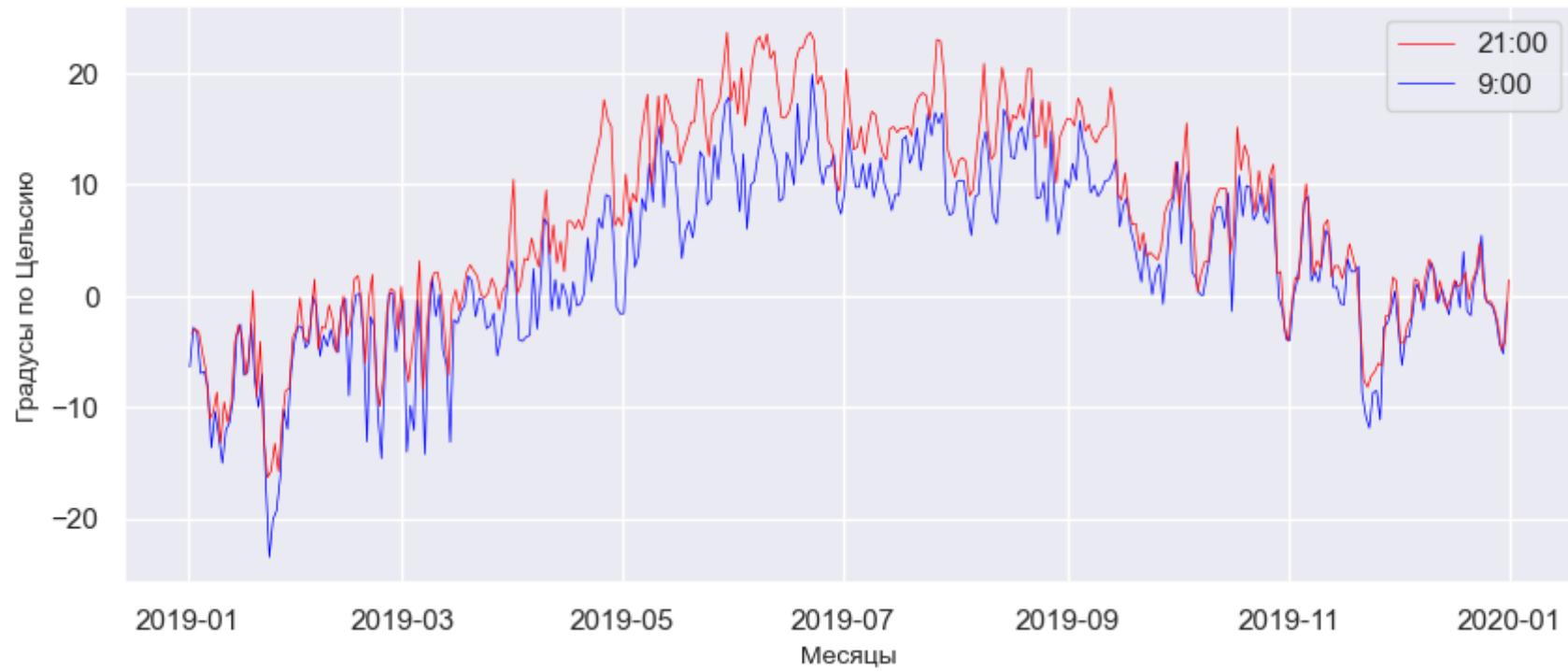
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2021 год



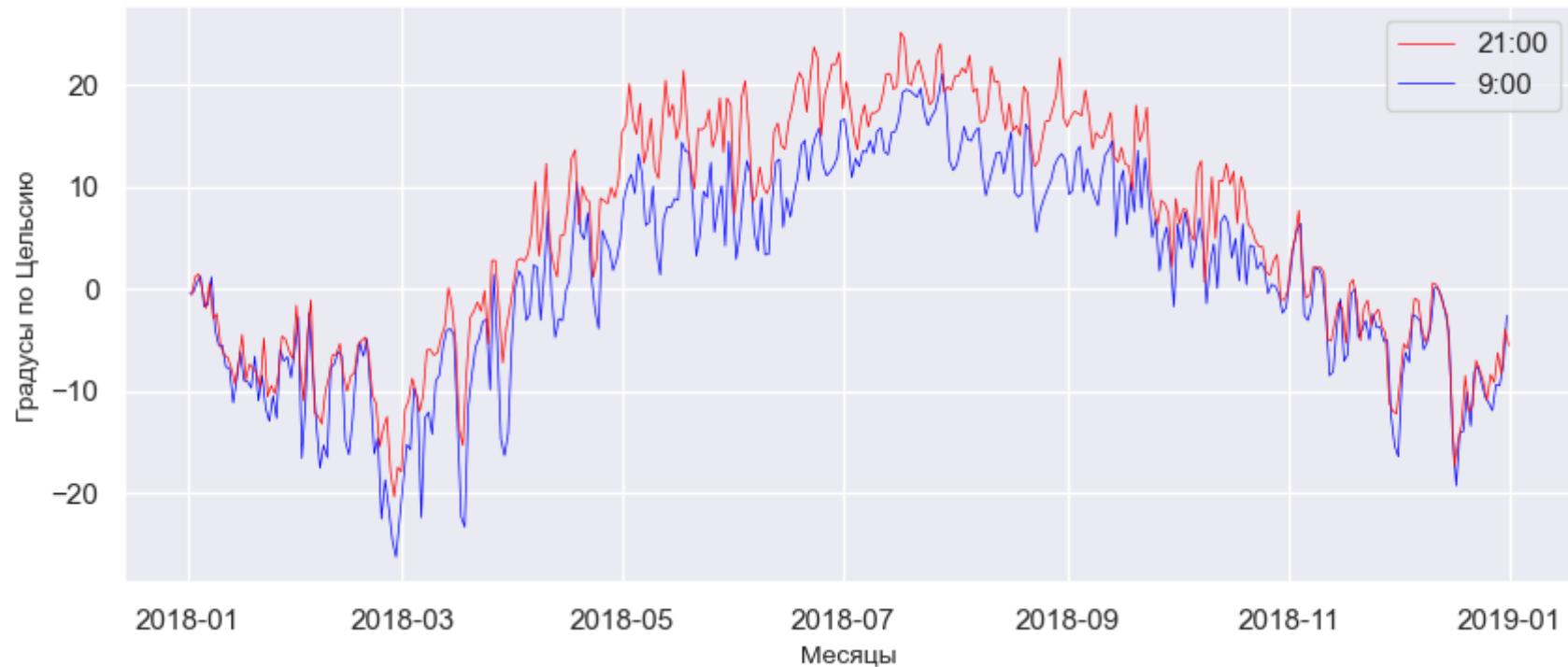
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2020 год



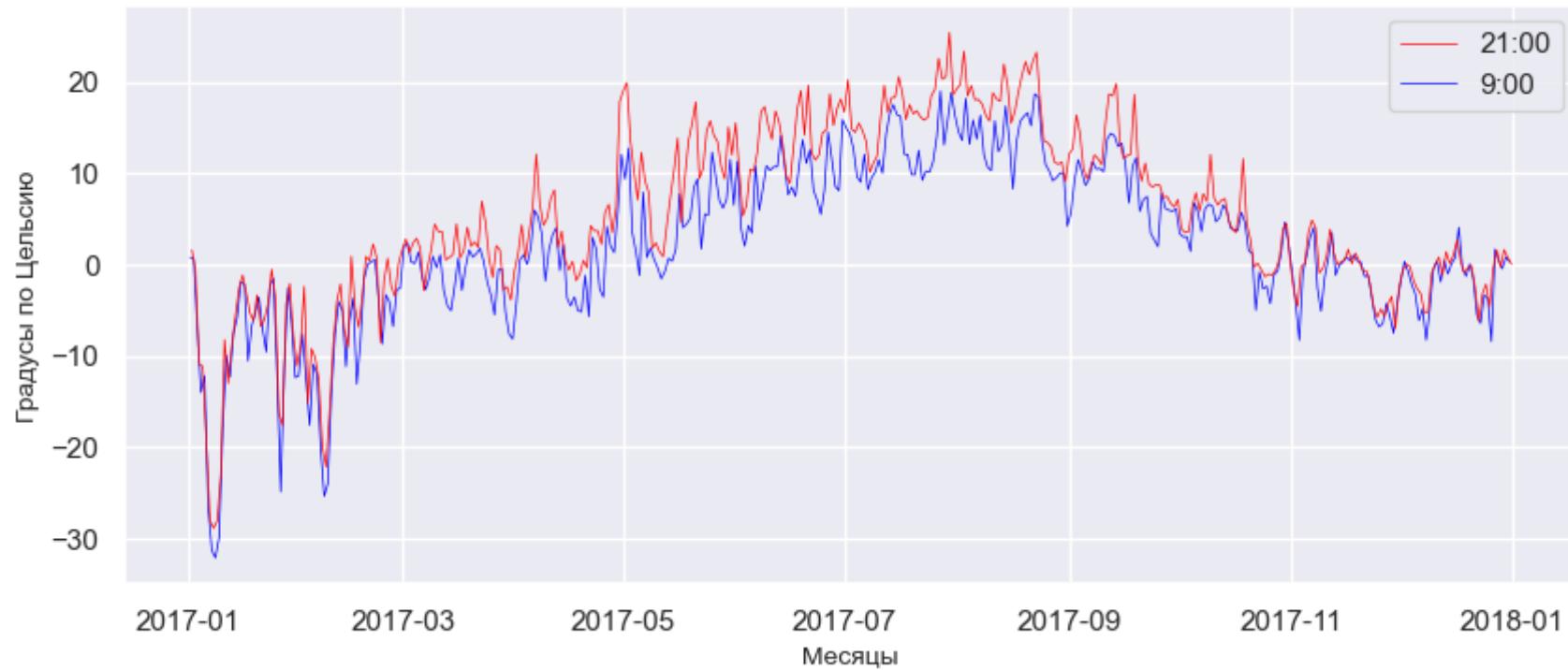
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2019 год



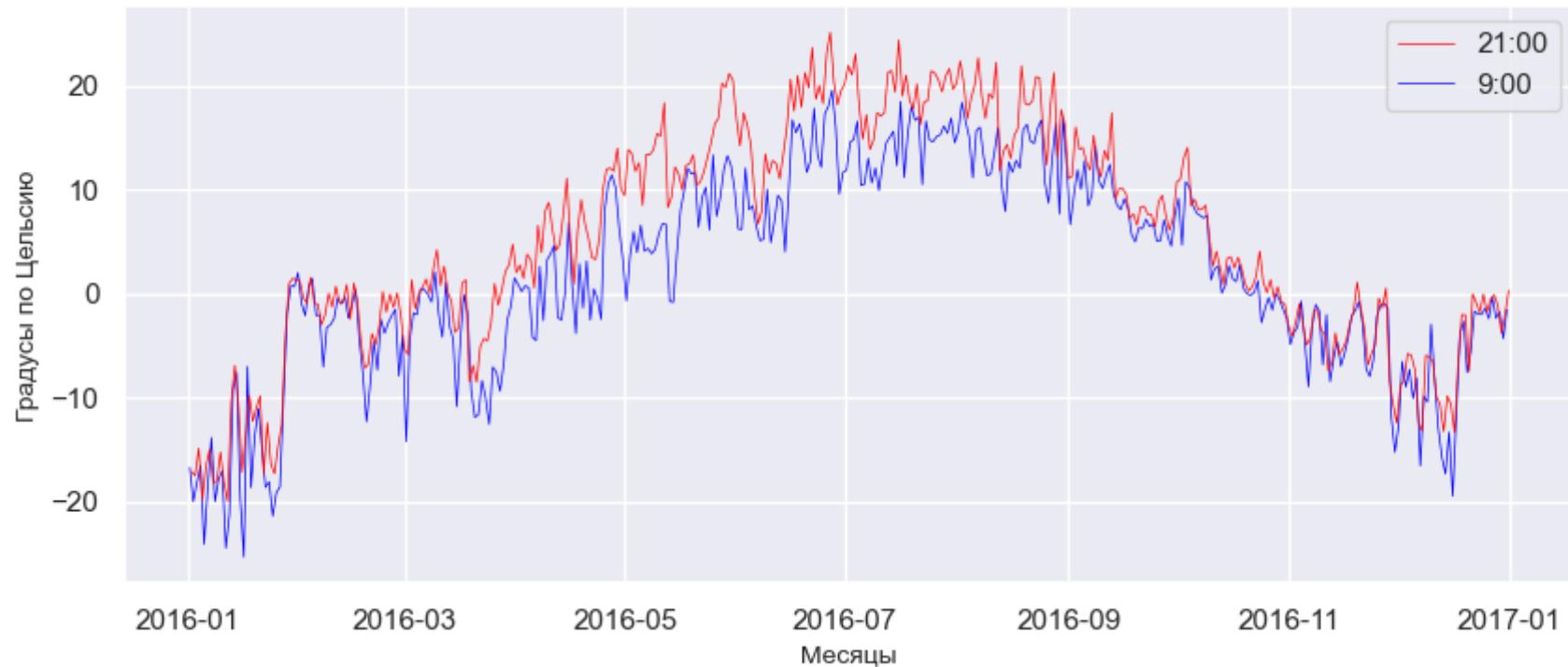
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2018 год



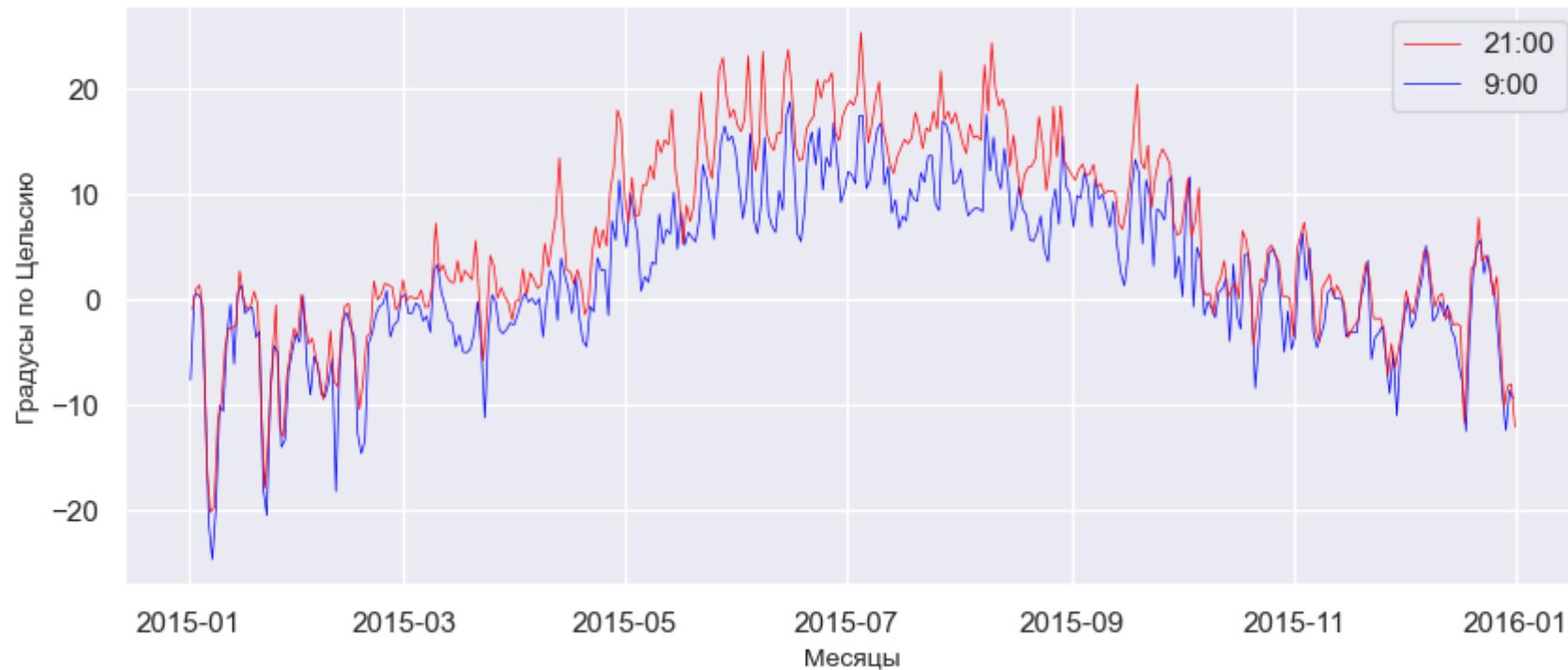
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2017 год



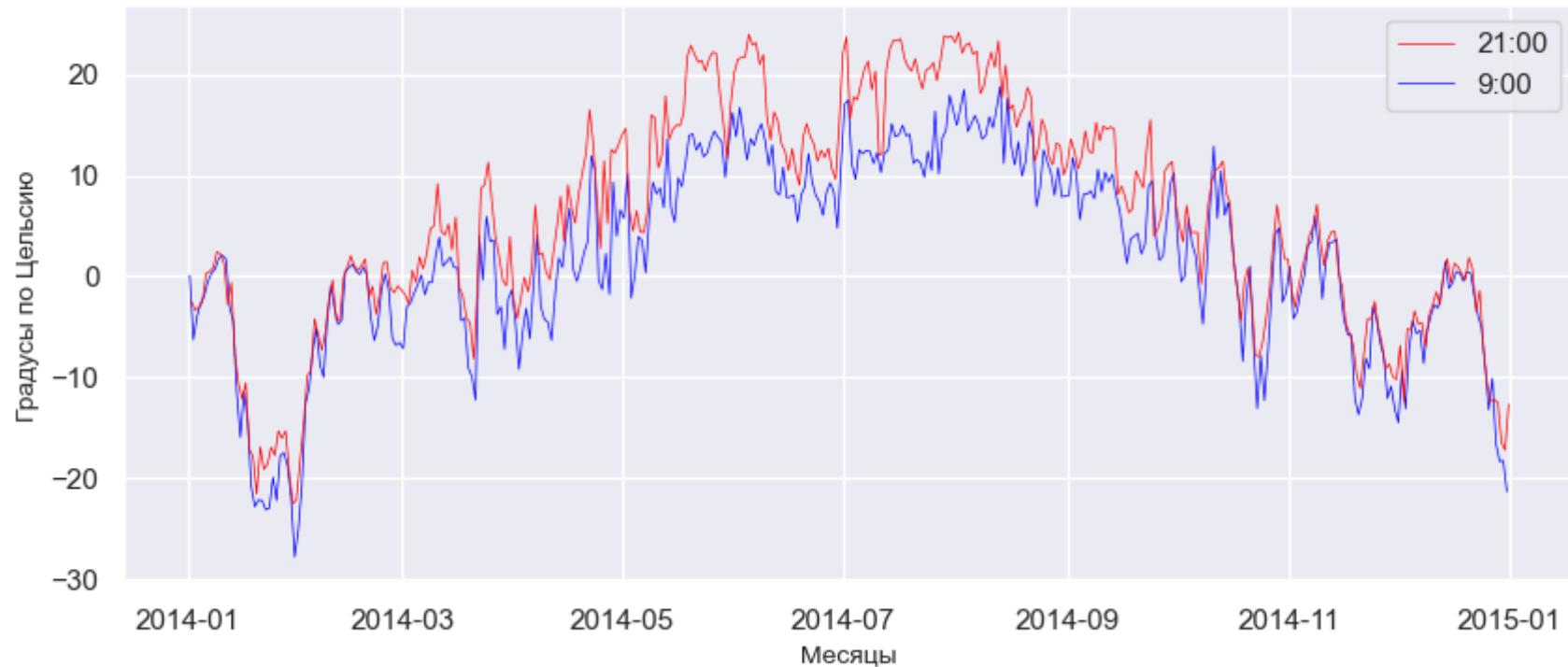
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2016 год



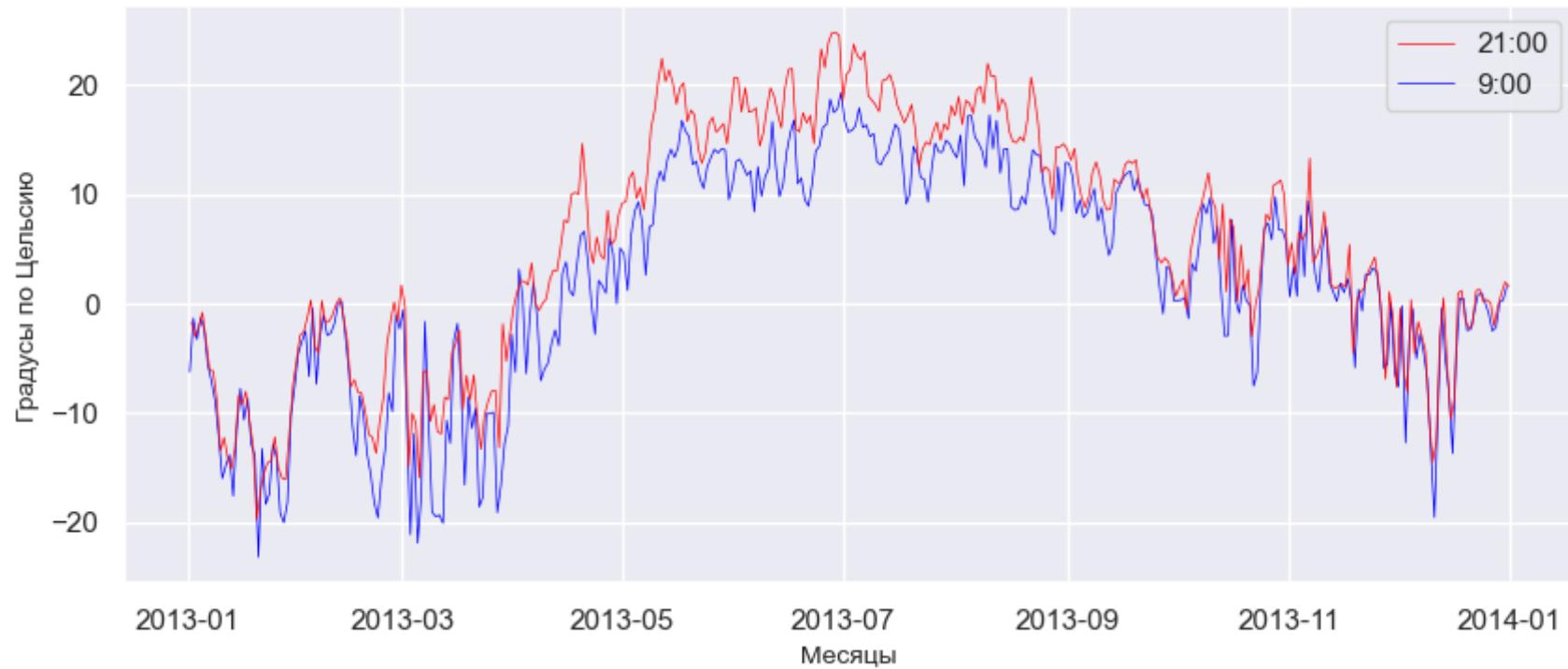
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2015 год



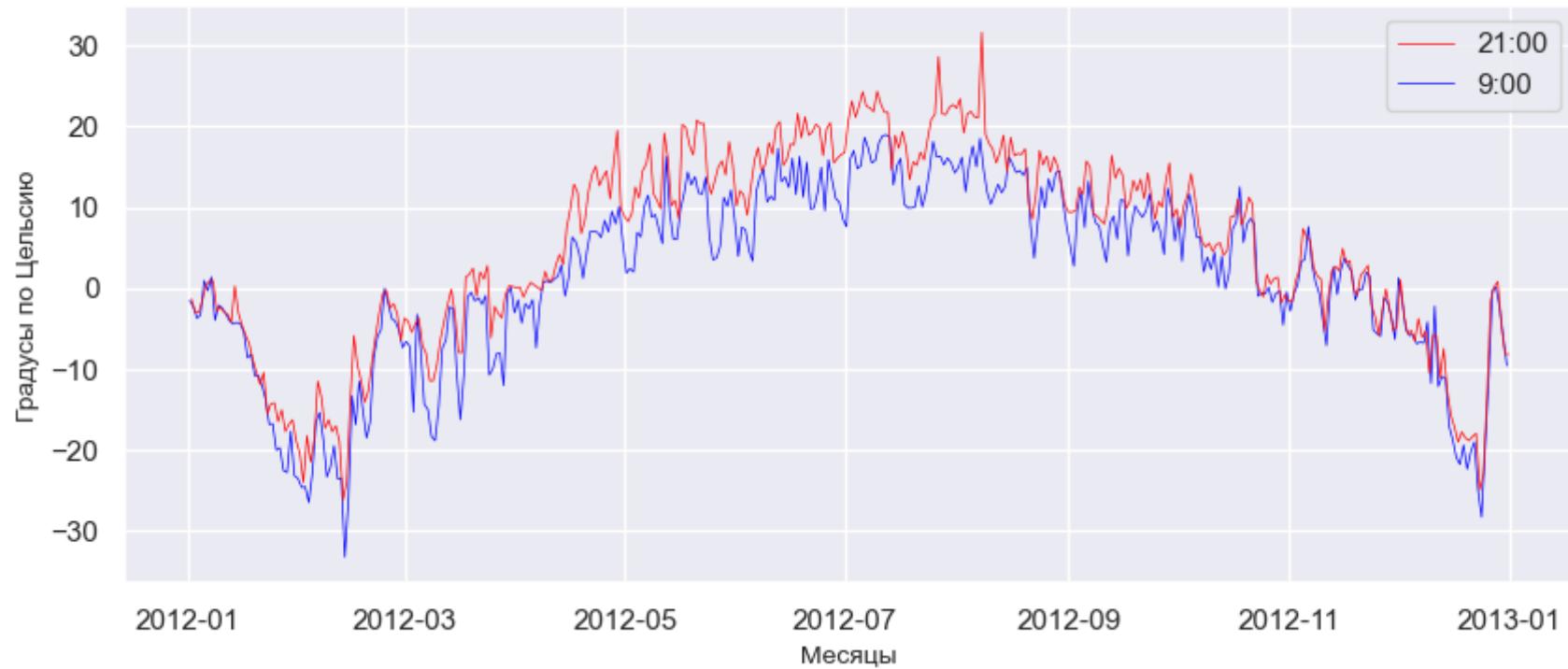
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2014 год



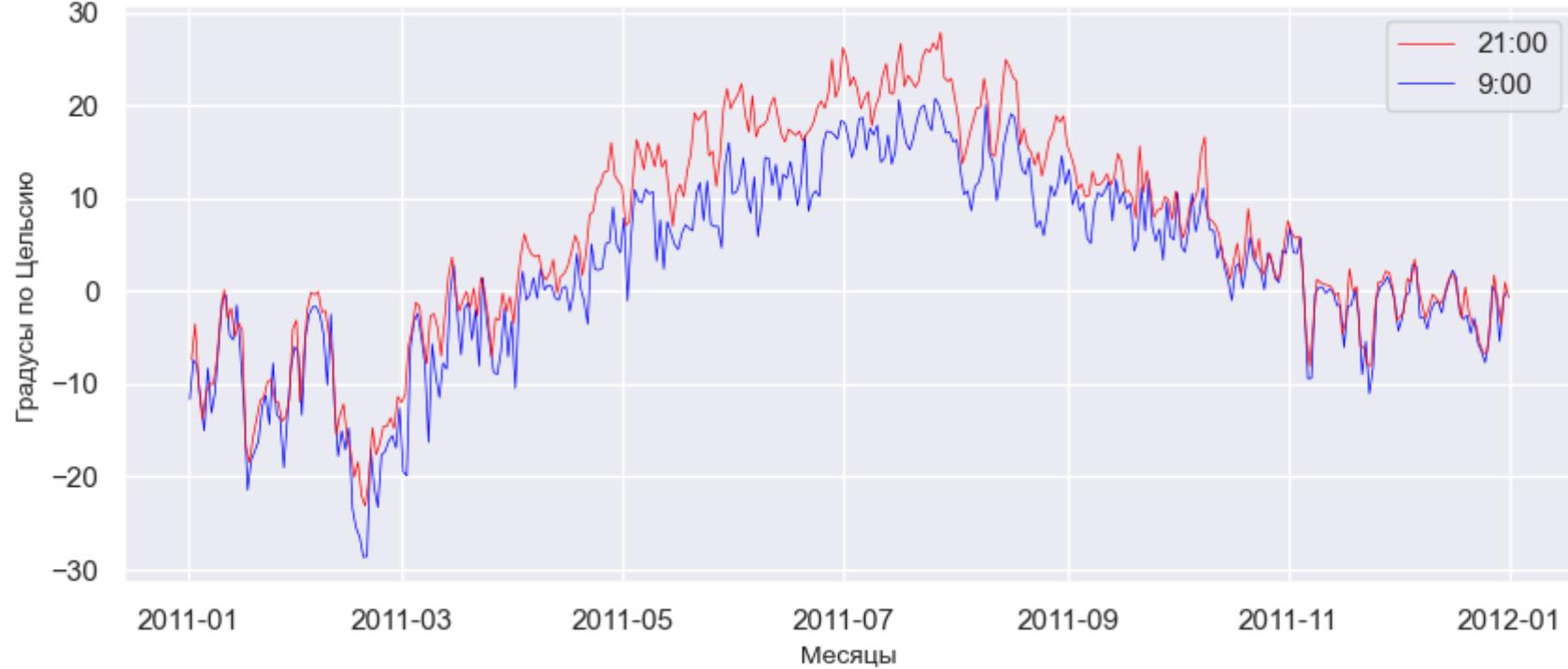
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2013 год



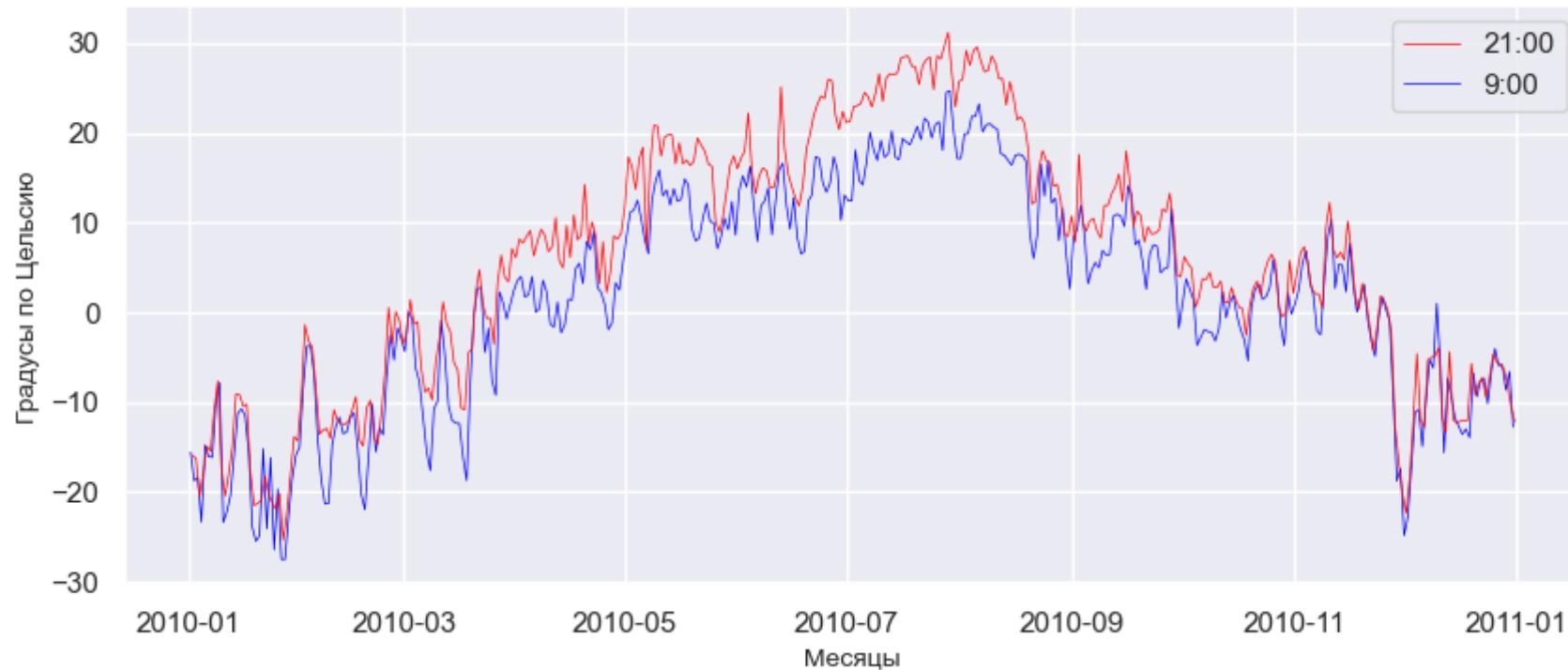
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2012 год



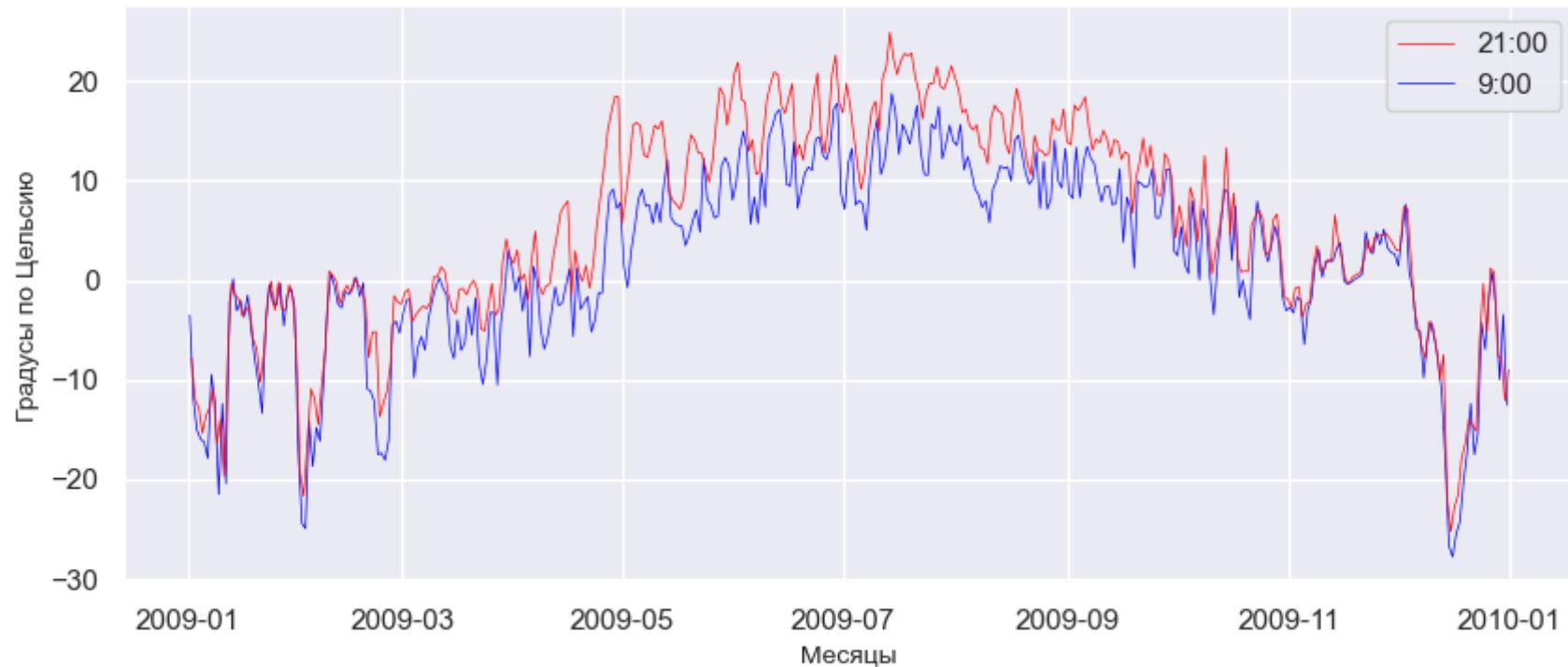
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2011 год



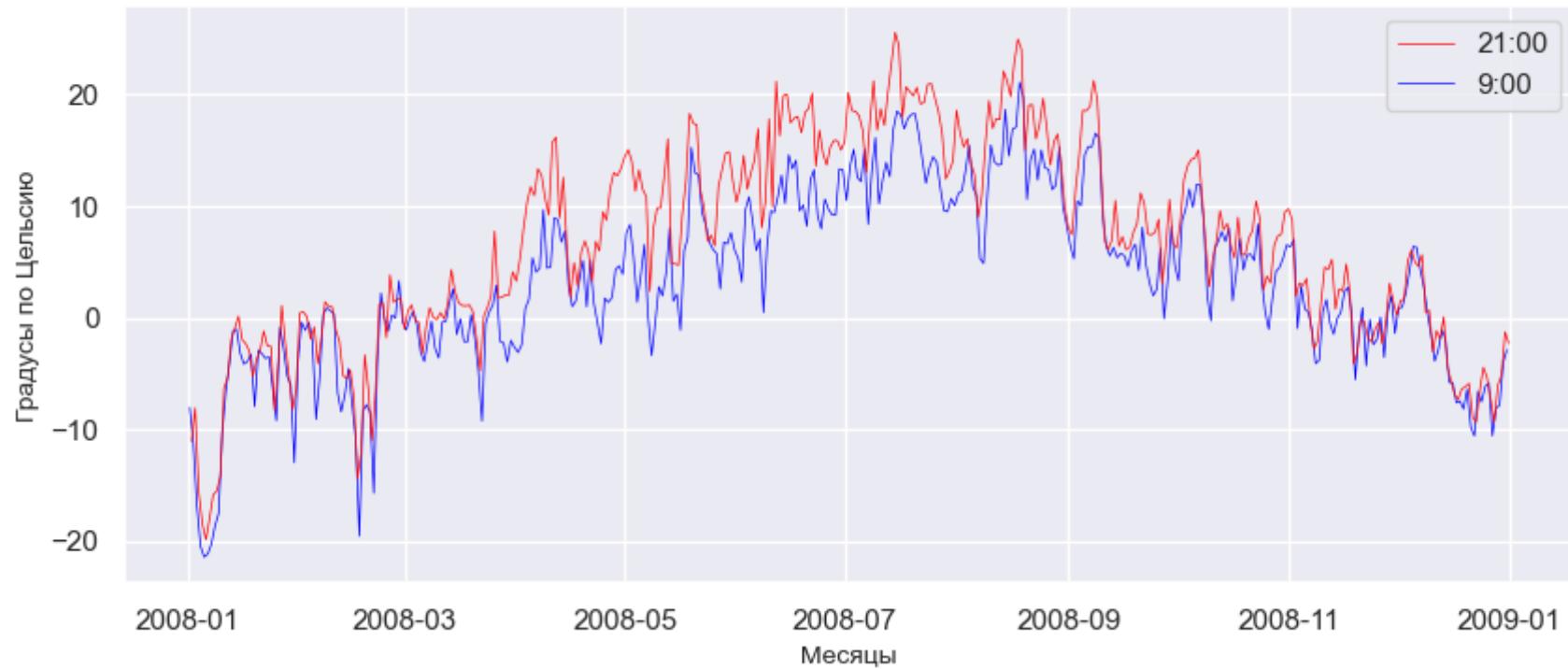
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2010 год



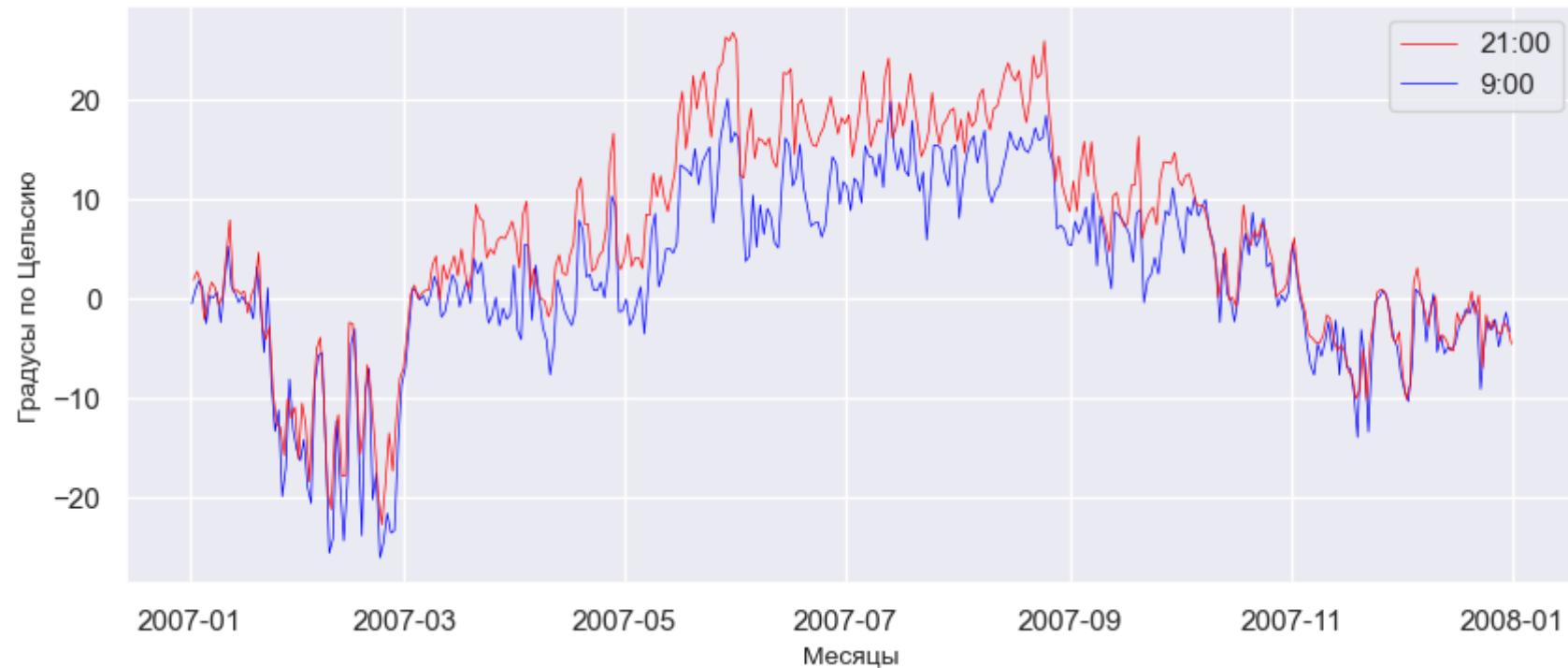
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2009 год



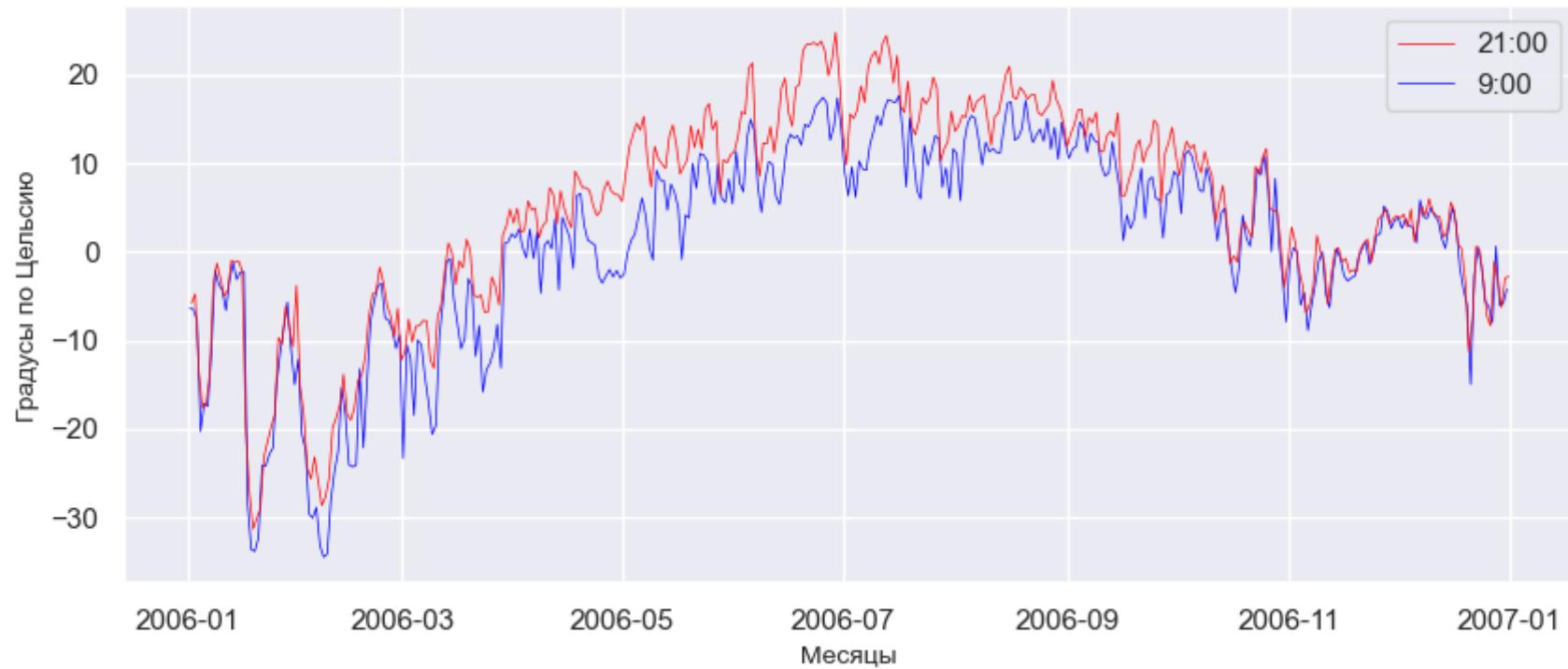
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2008 год



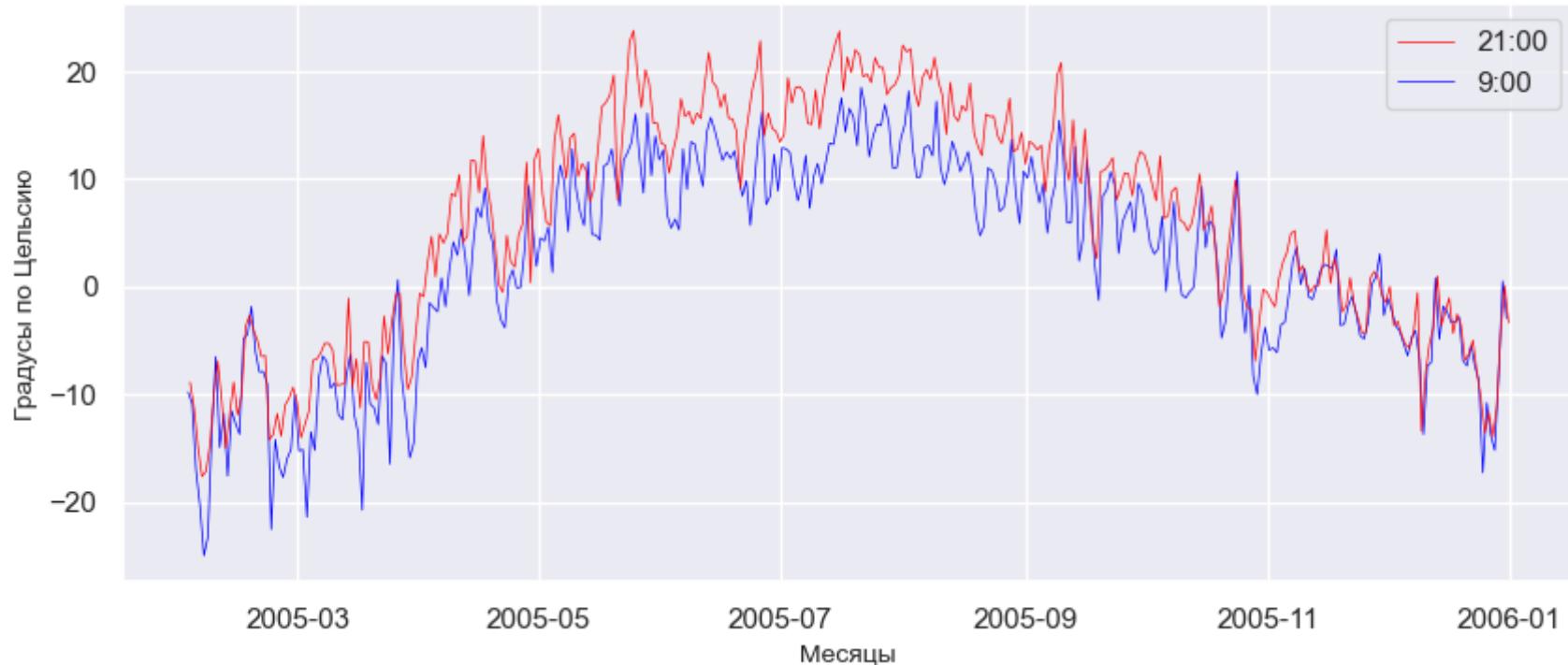
Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2007 год



Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2006 год



## Чашниково: Ежедневная динамика параметра T\_min за 12 предшествующих часов на 9 и 21 часов, 2005 год



### 3.2.6. Сохранение полученных данных в файлы

In [162...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER32}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER32}'  
  
makedirs(predict_path1, exist_ok=True)  
makedirs(predict_path2, exist_ok=True)  
  
# Запишем текущие данные в файлы  
for name in dict_df_locations.keys():  
    print(name + '.csv ->', end=' ')
```

```

    dict_df_locations[name].to_csv(
        path_or_buf=f'{predict_path1}{name}.csv'
    )
    print('DONE! ')

print('df_'+PARAMETER32 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER32].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER32}.csv'
)
print('DONE! ')

```

```

df_Dmitrov.csv -> DONE!
df_Kashyn.csv -> DONE!
df_Klin.csv -> DONE!
df_Mozhaisk.csv -> DONE!
df_Naro_Fominsk.csv -> DONE!
df_Nemchinovka.csv -> DONE!
df_N_Jerusalem.csv -> DONE!
df_Serpukhov.csv -> DONE!
df_Staritsa.csv -> DONE!
df_Tver.csv -> DONE!
df_Volokolamsk.csv -> DONE!
df_V_Volochev.csv -> DONE!
df_Chashnikovo.csv -> DONE!
df_Rfrnce_point.csv -> DONE!
df_T_min.csv -> DONE!

```

### 3.3. Максимальная температура воздуха: T\_max (за последние 12 часов, предшествующие наблюдению)

Максимальная температура должна измеряться не менее 2 раз в сутки. Однако, как показано в 1й тетради, а также при подсчёте фактической периодичности измерений, это положение далеко не всегда соблюдается. Более того, не всеми метеостанциями соблюдается синхронность в измерениях на определённый час. В связи с этим, чтобы привести измерения максимальной температуры к единой периодичности мы воссоздадим значения для этого показателя как если бы он фиксировался с интервалом в 3 часа за предыдущие 12 часов наблюдений. В реальной жизни такого не происходит (в том числе и из-за устройства самого прибора измерения), но нам важно привести значения к единым интервалам и моментам времени. В дальнейшем для анализа данных имеет смысл использовать данные на определённый вечерний час за предшествующие 12 часов. Как будет показано ниже, **основным моментом временем фиксации максимальной температуры является 21 час. Поэтому, при анализе максимальной температуры следует использовать именно это время.**

### 3.3.1. Поиск и удаление ошибок показателя максимальной температуры T\_max

Для данного раздела обозначим константу названия параметра

In [163...]

```
PARAMETER33 = 'T_max'
```

**Создаём временный DF для работы с параметром максимальной температуры**

In [164...]

```
param_df_name = f'df_{PARAMETER33}' # преобразуем полученное значение в df_PARAMETER33 - ключ словаря dict_df_parameters
# Создадим временный df
df_tmp33 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp33.sample(7, random_state=56)
```

Out[164]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2014-02-22 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-05-16 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-06-18 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-12-02 21:00:00	-1.8	-2.8	-2.9	-2.2	-2.3	-2.8	-2.7	-2.8	
2008-06-05 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2016-10-20 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2006-09-11 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN



### Визуализируем архив максимальной температуры (T\_max) по сезонам

Исходя из данных о климате Московской области, временные границы сезонов определены следующим образом.

- Зима (ниже 0°): В среднем длится с 5 ноября по 4 апреля
- Весна (от 0° до +10°): В среднем длится с 5 апреля по 18 мая
- Лето (выше +10°): В среднем длится с 19 мая по 14-15 сентября
- Осень (от +10° до 0°): В среднем длится с 14-15 сентября по 4 ноября

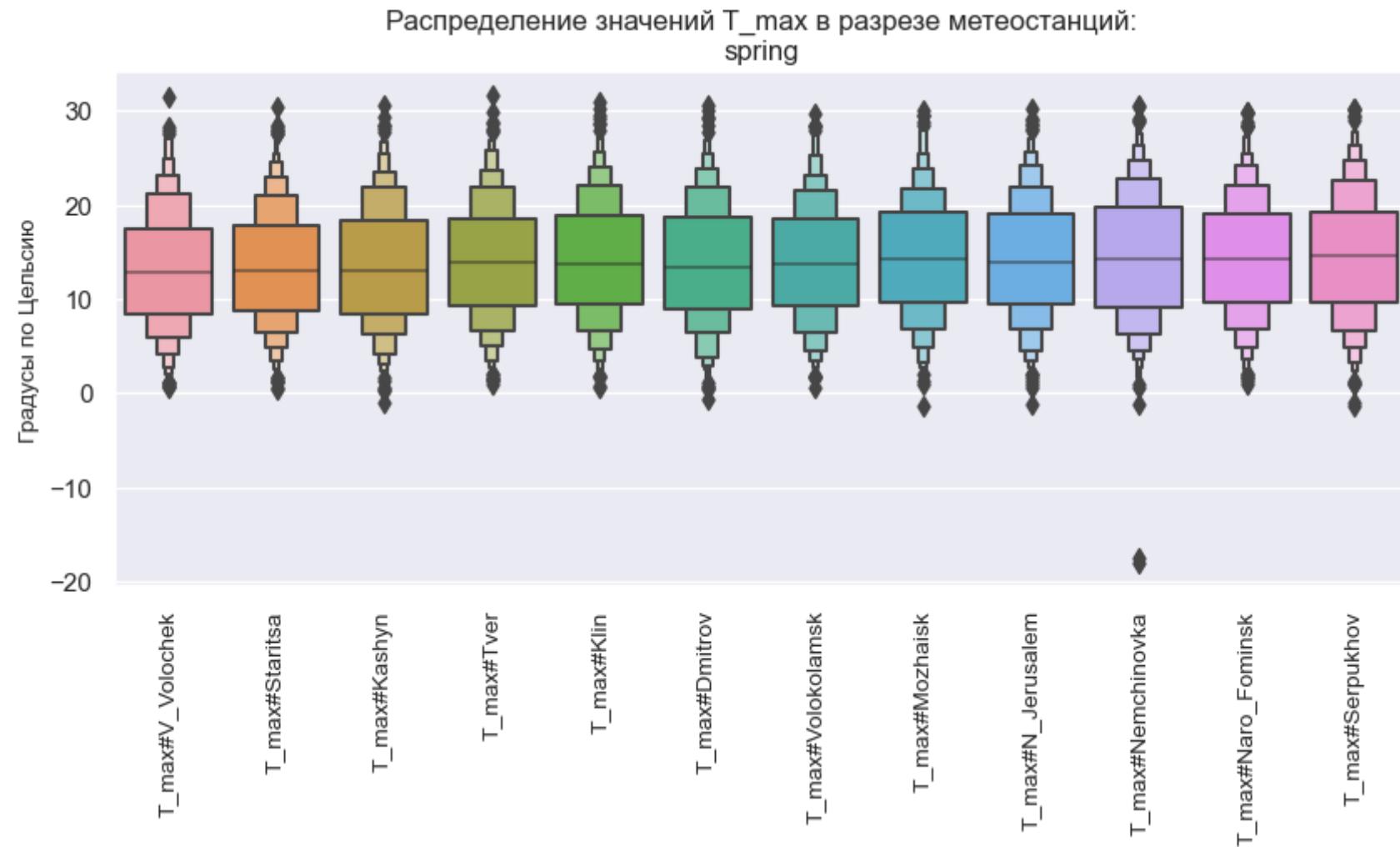
In [165...]

```
# В цикле выведем графики температур по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов (она возвращает словарь масок)
```

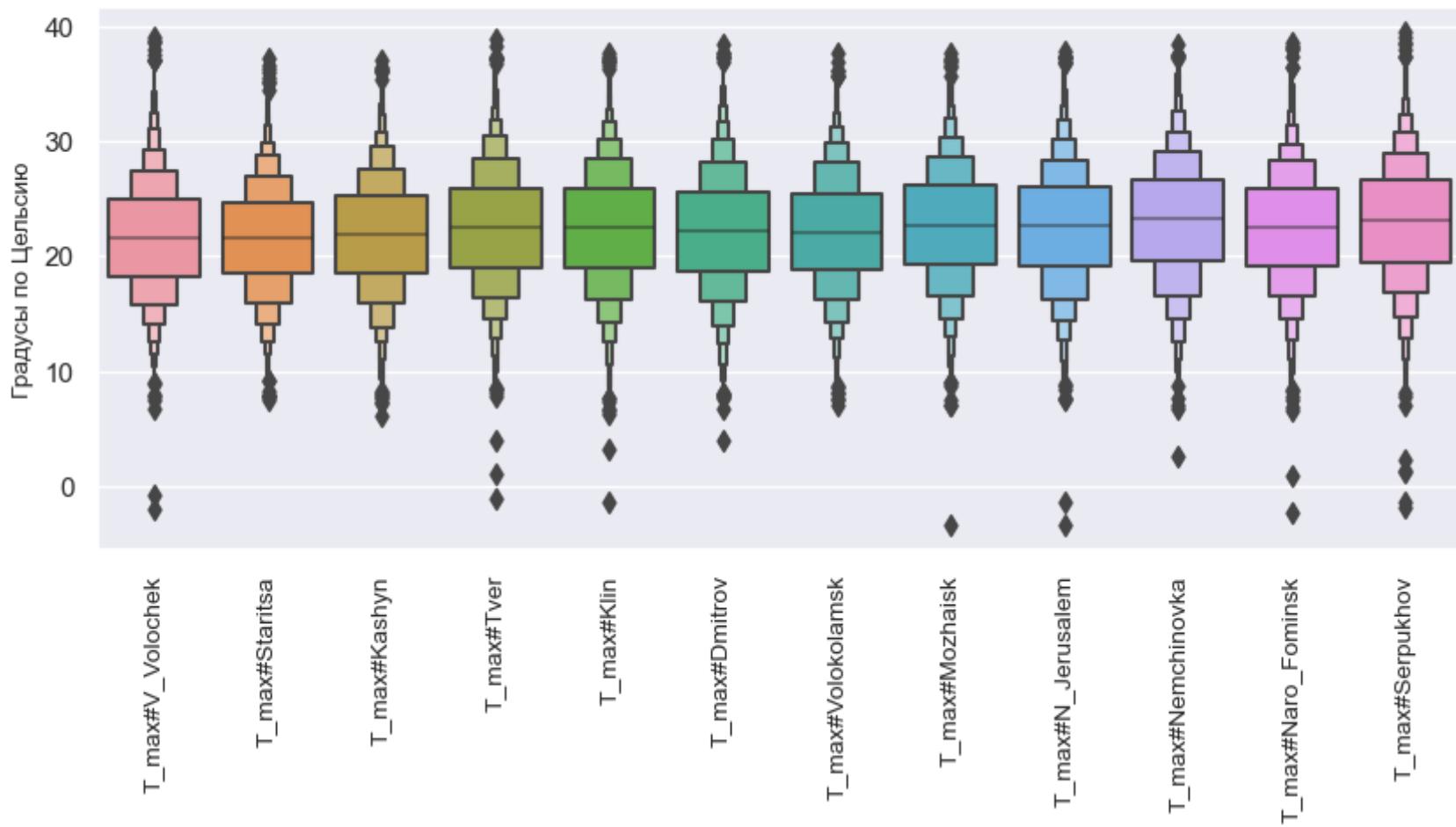
```

for season_name, season_mask in season_masks(df_tmp33).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp33[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Градусы по Цельсию', size=10)
    dummy = g.set_title(f'Распределение значений T_max в разрезе метеостанций:\n{season_name}')
plt.show()

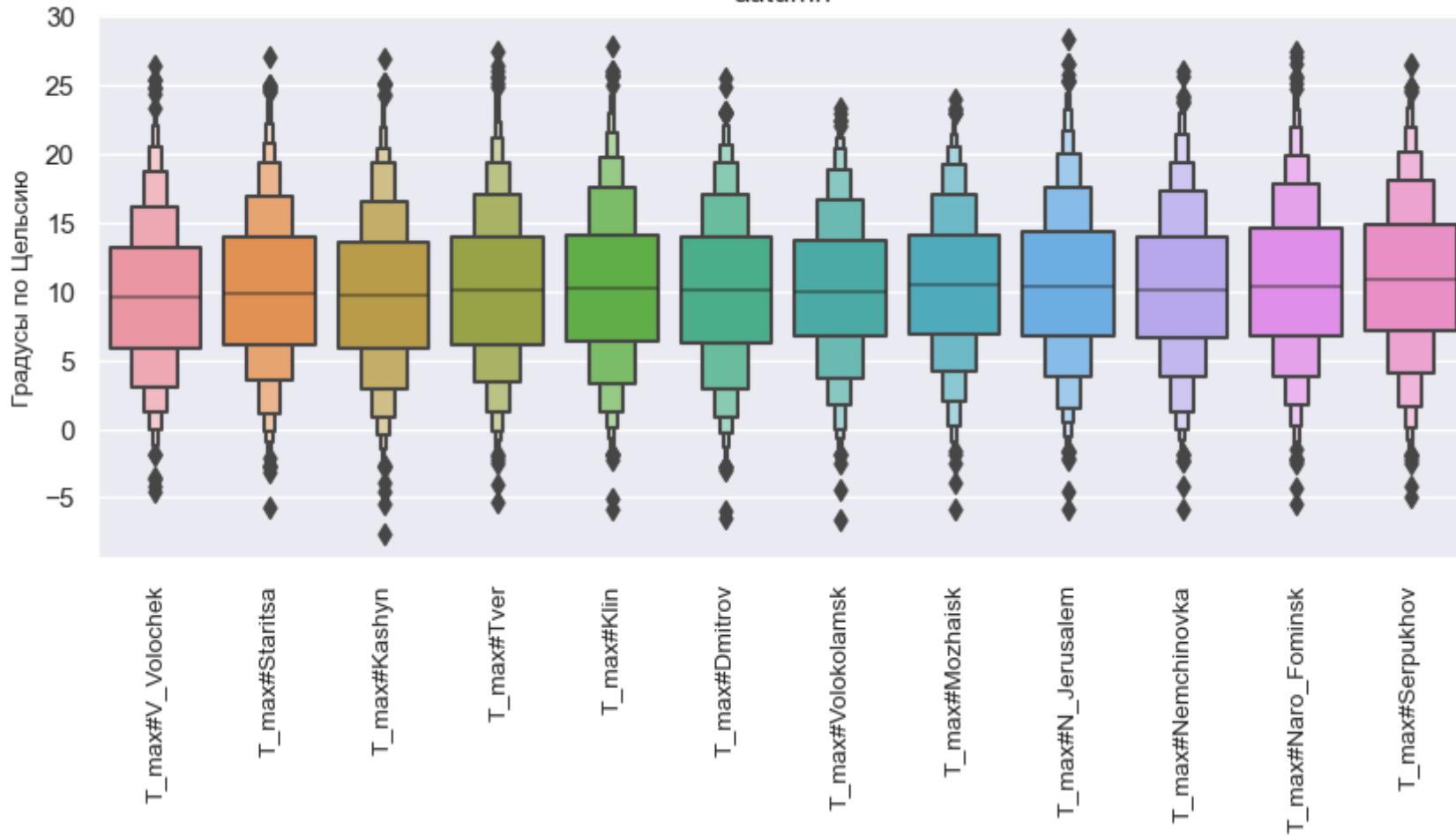
```

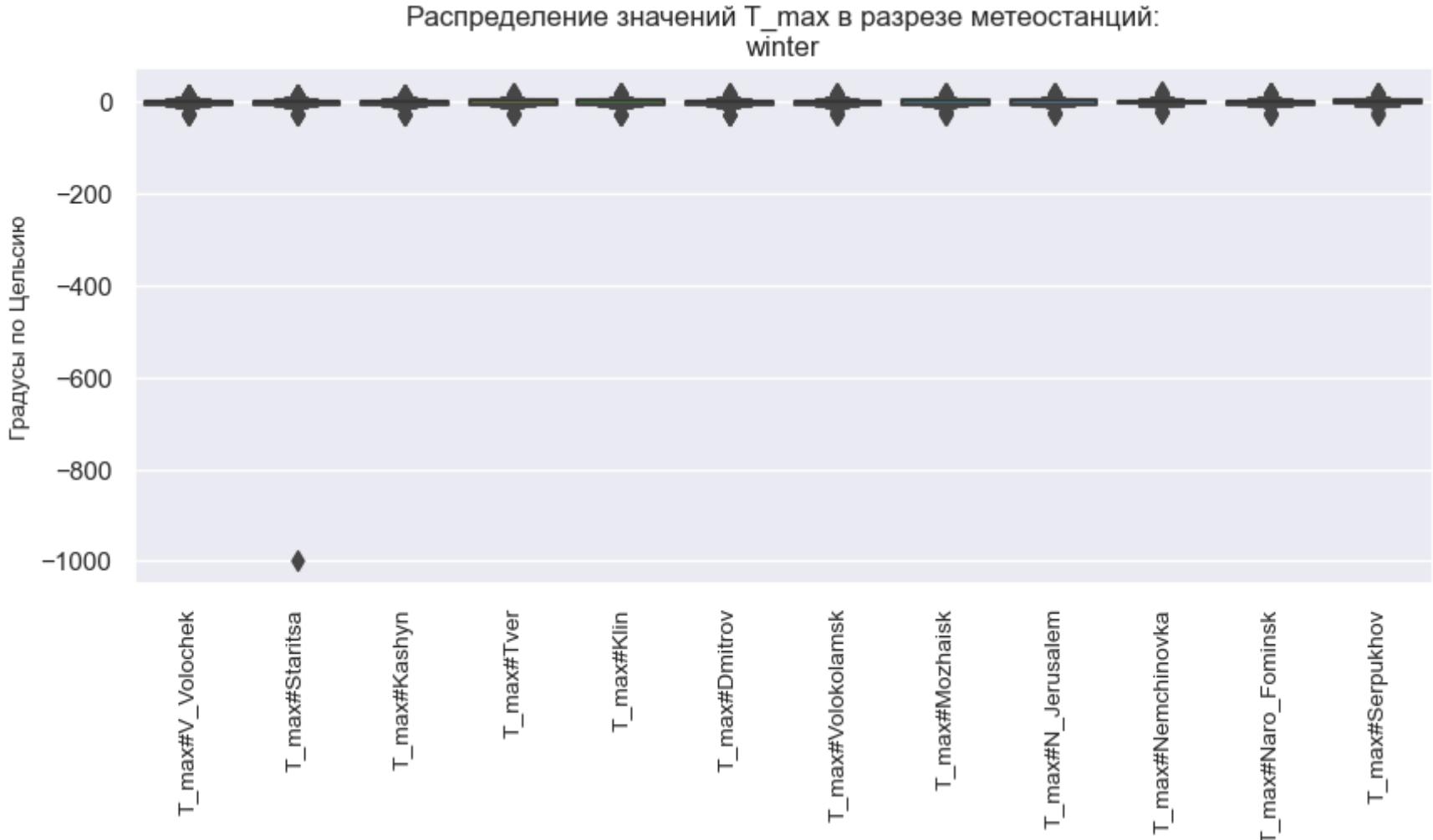


Распределение значений T\_max в разрезе метеостанций:  
summer



Распределение значений T\_max в разрезе метеостанций:  
autumn





Как видно из графика, часть выбросов выглядит неестественно для своего сезона - это, скорее всего, ошибки.

Выведем минимальное и максимальное значения, а также значение медианы и средней для всего DF.

```
In [166]: print(f'Минимальное значение: {np.nanmin(df_tmp33)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp33)},\n'
      f'Средняя: {np.nanmean(df_tmp33)},\n'
      f'Медиана: {np.nanmedian(df_tmp33)}')
```

Минимальное значение: -999.9,  
Максимальное значение: 39.4,  
Средняя: 9.728347806215723,  
Медиана: 9.5

Найдём эстремальные минимальные значения максимальной температуры за 12 часов.

```
In [167...]: list_ix = df_tmp33[df_tmp33 < -30].dropna(how='all').index.tolist()  
df_tmp33[df_tmp33.index.isin(list_ix)]
```

```
Out[167]:
```

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2005-02-25 21:00:00	NaN	-999.9	NaN	-5.0	-5.7	-5.4	-6.4	-6.7	

Это явная ошибка, и одна единственная такого рода. Скорее всего, из-за чрезмерно сильного смещения распределения значений в этом моменте наблюдений, и сильного смещения средней, такая ошибка попадёт в доверительный интервал 3х сигм и не будет определена как выброс. Поэтому при поиске выбросов потребуется использовать исключение данных по каждой конкретной станции из подсчёта средней.

### Поиск и удаление ошибок

Как было показано в первой тетради, и как видно из случайно выбранных строк, показатель T\_max содержит огромное количество пропущенных значений. Оценим объём имеющихся данных.

Удалим сплошные NaN (моменты наблюдения, по которым нет данных ни по одной метеостанции). Таким образом мы получим количество строк, в которых есть данные хотя бы по одной метеостанции. Сравним их с общим количеством моментов наблюдения.

```
In [168...]: data_rows_frac = (df_tmp33.dropna(how='all', axis=0).shape[0] / df_tmp33.shape[0])  
print(f'Доля строк с хотя бы одним наблюдением T_min составляет {data_rows_frac:.2%}')
```

Доля строк с хотя бы одним наблюдением T\_min составляет 12.73%

Очевидно, что фиксация максимальной температуры за последние 12 часов происходит не каждый момент наблюдения.

In [169]:

```
# выводим часы и количество раз для каждого часа, когда фиксировались значения показателя T_max
# получаем кортеж из 2х массивов:
hours, counts = np.unique(df_tmp33.dropna(how='all', axis=0).index.hour, return_counts=True)
# преобразуем его в массив и трансформируем (чтобы получить вертикальный вид)
np.asarray((hours, counts)).T
```

Out[169]:

```
array([[ 0,   9],
       [ 3,  21],
       [ 6,   4],
       [ 9,  55],
       [15,  33],
       [18,  12],
       [21, 6323]], dtype=int64)
```

Таким образом основное время фиксации максимальной температуры приходится на 21 час, на другие часы приходится незначительная часть фиксации максимальной температуры.

*Определим последовательность действий, для поиска ошибок показателя "максимальная температура".*

Здесь опять следует иметь в виду, что локальные температурные колебания могут быть достаточно значительными и зависят от локальных метеорологических явлений.

Очевидным способом проверить корректность зафиксированных значений будет сравнить их с данными текущих наблюдений температуры воздуха  $T$ , точнее с их минимумами за 12 предшествующих часов. При этом следует учесть, что максимумы температуры не обязательно будут совпадать по времени с моментами наблюдений, а могут случаться между ними.

1. Определяем выбросы в поле метеостанций - формируем кандидатов в ошибки, таким образом мы не пропустим "ложбины", когда понижение температуры затронуло большую часть метеостанций.
2. Отдельно проверяем, есть ли единичные значения в рядах - это тоже кандидаты в ошибки
3. Проверяем каждый из этих кандидатов на отклонение от максимумов значений по 12 часовому окну моментов наблюдения.
4. То, что осталось и есть ошибка.

### **Найдем выбросы в поле метеостанций.**

Параметры:

- используем среднюю по обратным квадратам расстояния от центра поля,
- включаем проверяемое значение в подсчёт средней (автоматически, так как средняя рассчитывается от центра поля для всех станций),

- определим границы доверительного интервала в 3,0 сигмы

```
In [170...]: # Определим небольшую функцию, которая будет перебирать все значения в ряду,
# последовательно исключая их из подсчёта средней
# и получая границы доверительного интервала для каждого такого расчёта.
# В ней же зададим параметры поиска выбросов
def station_exclusion_field_iterator(x_):
    """
    Функция для вызова field_outliers в цикле метеостанций для одного момента наблюдения
    и для формирования единого списка выбросов для этого момента
    Принимает - ряд из значений метеостанций для данного момента
    Возвращает - список выбросов для данного момента, удаляя дубликаты метеостанций.
    """

    list_outliers_ = [] # Определяем пустые списки для выбросов при проверке каждой станции отдельно
    list_station_idx_ =[] # и для индекса станции в серии
    for station_ in x_.index: # по индексу серии, вычилиняем метеостанции
        outlier_ = field_outliers( # вызываем функцию и передаём ей значения, согласно критериям поиска выбросов
            row_=x_,
            method_='sigma',
            criterium_=3,
            IDW_=True,
            param_=PARAMETER33,
            station_=station_[len(PARAMETER33)+1 :],
            inclusive_=False
        )
        # Оставляем в списке выбросов только одно упоминание о каждой станции, если таковое встречается
        if isinstance(outlier_, list): # Если функция вернула список, а не NaN
            for i_, tup_out_ in enumerate(outlier_): # По элементам списка-результата функции (список кортежей)
                if tup_out_[1] not in list_station_idx_: # Если индекс станции не в списке индексов станций
                    list_station_idx_.append(tup_out_[1]) # Добавим индекс станции в список
                    list_outliers_.append(outlier_[i_]) # Добавляем данный кортеж в список выбросов
            else:
                pass
        # Если список выбросов пустой, присвоим ему NaN
        list_outliers_ = np.nan if list_outliers_ == [] else list_outliers_
    #     print(list_outliers_, list_station_idx_, '\n')
    return list_outliers_ # Возвращаем список выбросов
```

```
In [171...]: start_time = time.time() # для замера времени выполнения кода
```

```
df_tmp33 = df_tmp33.assign(field_out=df_tmp33.iloc[:, :12].dropna(how='all'))
```

```

        .apply(lambda x: station_exclusion_field_iterator(x=x),
               axis=1)
    )

end_time = time.time()
elapsed_time = end_time - start_time
time_format = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f"На выполнение кода ушло: {time_format}")

```

На выполнение кода ушло: 00:05:58

In [172]: df\_tmp33.dropna(subset='field\_out').sample(7, random\_state=56)

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2019-11-02 21:00:00	2.5	2.9	1.7	3.3	3.2	3.5	3.6	3.7	
2019-03-05 21:00:00	5.6	4.4	3.8	4.5	4.4	3.8	NaN	NaN	
2009-11-26 21:00:00	5.9	6.1	5.0	5.6	5.6	5.1	5.8	6.0	
2018-11-03 21:00:00	8.4	7.9	7.6	8.1	8.0	7.8	NaN	NaN	
2015-10-08 21:00:00	2.5	2.8	3.1	3.3	3.1	NaN	NaN	NaN	
2021-04-18 21:00:00	16.1	15.2	15.9	16.0	15.9	15.8	15.4	14.9	
2021-02-24 21:00:00	-10.9	-11.8	-13.1	-12.3	-12.9	-13.1	-13.1	-13.0	

Проверим минимальные и максимальные значения выбросов в поле метеостанций.

In [173]:

```
# field_out содержит списки кортежей с характеристиками выбросов!
tup_out_stations = (
    df_tmp33.field_out.dropna().apply(lambda x: min([i[0] for i in x])).min(),
    df_tmp33.field_out.dropna().apply(lambda x: max([i[0] for i in x])).max()
)
tup_out_stations
```

Out[173]:

(-999.9, 34.6)

Используемые формулы определения выбросов специально обозначают как выброс в поле метеостанций случай, когда в ряду есть единственное значение. Проверим, есть ли ситуации, когда существует только одно значение параметра в поле метеостанций.

In [174]:

```
# Если значение является единственным в строке, то True, иначе False,
# убираем NaN (берём ряд без столбца field_out) и суммируем результат
single_values_count = df_tmp33.apply(
    lambda x : True if (len(x[:-1].dropna()) == 1) else False, axis = 1).dropna().sum()
print(f'Количество случаев единичных значений в рядах моментов наблюдений: {single_values_count}' )
```

Количество случаев единичных значений в рядах моментов наблюдений: 133

Проверим максимальное количество выбросов в поле метеостанций на момент наблюдения.

In [175]:

```
# Находим максимальное количество выбросов в строке поля метеостанций
max_field_outliers = df_tmp33.field_out.dropna().apply(lambda x: len(x)).max()
print(f'Максимальное количество выбросов в поле метеостанций за весь период наблюдения не превышает '
      f'{max_field_outliers}')
```

Максимальное количество выбросов в поле метеостанций за весь период наблюдения не превышает 2

**Для дальнейшей работы с ошибками создадим столбец error\_at, куда запишем кортеж из TimeStamp и названия станции.**

In [176]:

```
# Определяем столбец error_at (помним, что в field_out у нас может быть БОЛЕЕ 1 выброса),
# значит вторым элементом кортежа в error_at будет СПИСОК станций, по которым найдены выбросы

df_tmp33 = df_tmp33.assign(error_at =
    df_tmp33.
    apply(lambda x: # Вычленим DateTime index и название станции с выбросом
          # пропустим NaN, проверив, является ли x.field_out списком
          (x.name,
           stations_from_outliers(
             row_=x,
```

```
        column_name_ = 'field_out', # используем выбросы в поле метеостанций
        param_=PARAMETER33)
    ) if isinstance(x.field_out, list) else np.nan,
    axis=1
)
)
```

In [177]: `df_tmp33.dropna(how='all').sample(5, random_state=56)`

Out[177]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
<b>2015-03-19 21:00:00</b>	12.6	12.2	12.6	12.6	13.4	13.8	12.1	12.7	
<b>2006-03-19 21:00:00</b>	1.2	1.5	2.6	2.4	3.2	3.2	2.0	3.5	
<b>2007-06-16 21:00:00</b>	19.0	16.4	19.0	18.7	17.9	18.2	17.8	17.3	
<b>2009-06-10 21:00:00</b>	25.0	24.9	25.5	25.7	25.9	26.2	24.5	27.2	
<b>2013-01-21 21:00:00</b>	-12.3	-12.5	-12.4	-11.2	-11.6	-11.8	-12.8	-12.1	

Для подтверждения, являются ли отобранные значения максимальной температуры ошибками, рассмотрим их отклонение от наблюдаемых значений показателя температуры Т за предыдущие 12 часов по моментам наблюдения (наблюдение 12 часов назад не включаем, текущее наблюдение включаем)

In [178]:

```
# построим датафрейм с максимальными значениями температуры по моментам наблюдения за предшествующие 12 часов
# Используем скользящее окно с вчислением минимума и со сдвигом вверх на 4 строки (-4),
# чтобы обозначить минимумы за предшествующие периоды.
# Исключим из него 2 последних столбца с условными метеостанциями
df_t1 = df_tmp31.rolling(window='12H', center=False, closed='left').max().shift(-4).iloc[:, :-2]
```

In [179...]

```
# преобразуем один из DF в массив пятеру, чтобы избежать deprecation warning,
# возьмём только значения соответствующие индексам в датафрейме df_tmp33.
# Найдём абсолютное отклонение между элементами df_t1 и df_tmp33
# не берём 2 последних новых столбца df_tmp33
df_t2 = np.absolute(
    np.subtract(
        df_t1[df_t1.index.isin(df_tmp33.index)], np.array(df_tmp33.iloc[:, :-2])
    )
)
# df_t2
```

In [180...]

```
# выберем из df_t2 значения отклонений, соответствующие "кандидатам в ошибки"
# и поместим их в отдельный столбец df_tmp33: Tx_deviation в виде списка, соответствующего списку error_at
df_tmp33 = (df_tmp33
    .assign(Tx_deviation=df_tmp33.error_at.dropna()
           .apply(lambda x:[df_t2.loc[x[0], "T#" + station] for station in x[1]]))
    )
df_tmp33.dropna(how='all').sample(5, random_state=56)
```

Out[180]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
--	------------------	----------------	--------------	------------	------------	---------------	-------------------	----------------	---------

<b>2015-03-19 21:00:00</b>	12.6	12.2	12.6	12.6	13.4	13.8	12.1	12.7	
<b>2006-03-19 21:00:00</b>	1.2	1.5	2.6	2.4	3.2	3.2	2.0	3.5	
<b>2007-06-16 21:00:00</b>	19.0	16.4	19.0	18.7	17.9	18.2	17.8	17.3	
<b>2009-06-10 21:00:00</b>	25.0	24.9	25.5	25.7	25.9	26.2	24.5	27.2	
<b>2013-01-21 21:00:00</b>	-12.3	-12.5	-12.4	-11.2	-11.6	-11.8	-12.8	-12.1	

In [181...]

```
# Выберем из них отклонения превышающие 5 градусов
# Определяем столбец double_error_at (помним, что в field_out есть значения, указывающие на БОЛЕЕ чем 1 выброс),
# значит вторым элементом кортежа в error_at будет СПИСОК станций, по которым найдены выбросы,
df_tmp33 = (df_tmp33
    .assign(double_error_at =
        df_tmp33.dropna(subset=["Tx_deviation"])
        .apply(lambda x: # Вычленим DateTime index и название станции с выбросом если Tx_deviation>5
            (x.name,
            [station for i, station in enumerate(x.error_at[1]) if x.Tx_deviation[i] > 5]
            ),
        axis=1
        )
    )
# Заменим пустые списки в double_error_at на NaN
df_tmp33.double_error_at =(
    df_tmp33
    .double_error_at
    .dropna()
    .apply(lambda x: np.nan if len(x[1])==0 else x)
)
```

In [182...]

```
df_tmp33.dropna(subset=["double_error_at"])
```

Out[182]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2021-05-29 21:00:00	16.8	15.8	17.3	17.5	15.7	17.4	14.0	15.1	
2021-04-18 21:00:00	16.1	15.2	15.9	16.0	15.9	15.8	15.4	14.9	
2020-06-05 21:00:00	20.4	18.4	21.0	19.9	19.1	19.8	NaN	NaN	
2020-05-23 21:00:00	16.0	8.3	7.2	7.9	7.7	8.5	NaN	NaN	
2020-05-12 21:00:00	11.0	20.2	23.2	19.4	22.7	24.5	NaN	NaN	
2020-03-29 21:00:00	6.7	15.0	13.0	14.4	15.9	16.2	NaN	NaN	
2020-03-06 21:00:00	0.5	6.4	3.9	6.2	5.5	5.3	6.0	6.4	
2019-10-03 21:00:00	12.3	19.4	18.3	18.7	20.1	20.4	19.4	20.6	
2019-06-03 21:00:00	19.6	19.2	19.2	29.3	20.1	19.4	19.3	19.4	
2019-01-28 21:00:00	-13.7	-8.8	-8.3	-8.5	-7.4	-7.4	NaN	NaN	
2018-10-01 21:00:00	14.4	14.4	13.0	14.8	14.4	14.0	NaN	NaN	





	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2011-10-18 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-10 21:00:00	9.7	7.3	8.5	8.0	7.2	7.6	7.2	7.8	
2011-10-08 21:00:00	10.5	20.4	18.0	18.8	20.9	20.6	20.6	21.0	
2011-07-19 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-15 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-11 09:00:00	NaN	NaN	NaN	NaN	11.4	NaN	NaN	NaN	NaN
2011-05-15 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-1.4
2011-05-06 21:00:00	8.5	14.8	16.0	16.6	16.4	15.4	16.6	18.0	
2011-02-19 21:00:00	-14.2	-11.5	-13.8	-13.1	-14.0	-15.5	15.5	-14.3	
2011-01-27 21:00:00	-11.0	-10.0	-12.5	-10.6	-11.9	-12.0	-10.1	-9.7	
2010-11-27 21:00:00	-8.1	0.0	-2.2	-0.3	-0.1	-0.4	-0.4	-0.3	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2010-08-17 21:00:00	27.5	28.9	27.0	26.3	NaN	25.5	30.3	32.0	
2010-07-25 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-07-19 09:00:00	NaN	NaN	NaN	NaN	16.8	NaN	NaN	NaN	NaN
2010-07-02 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	10.0
2010-06-25 21:00:00	31.5	30.5	31.8	33.6	3.3	33.3	31.9	32.4	
2010-06-23 21:00:00	22.7	28.8	NaN	31.8	31.2	31.1	29.8	29.7	
2010-05-05 21:00:00	8.4	19.7	21.3	20.1	23.6	23.6	22.8	23.5	
2010-04-25 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-04-23 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-04-07 21:00:00	15.6	17.6	16.0	17.2	16.4	16.3	16.5	17.0	
2010-01-08 21:00:00	-5.1	-3.2	-7.2	4.8	-3.9	-4.6	-4.1	-3.5	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2009-12-15 21:00:00	-18.5	-19.2	-21.5	-22.4	-21.8	-22.6	-21.7	-20.8	
2009-12-08 21:00:00	-3.5	-3.8	-5.9	-4.6	-5.7	6.2	-5.5	-5.0	
2009-10-13 21:00:00	4.4	10.8	10.2	10.6	10.5	9.0	9.2	9.7	
2009-08-22 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2009-08-13 15:00:00	-2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2009-07-29 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-3.4
2009-07-28 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2009-05-27 09:00:00	NaN	NaN	NaN	NaN	6.3	NaN	NaN	NaN	NaN
2009-04-17 21:00:00	6.6	5.6	6.2	6.6	7.1	6.9	7.3	7.3	
2009-04-15 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-12-26 21:00:00	-6.9	-8.6	-6.6	-7.7	-6.8	6.4	-6.7	-5.7	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2008-10-08 21:00:00	5.5	12.7	12.6	13.2	13.0	13.4	12.4	13.0	
2008-09-26 21:00:00	11.0	11.2	10.0	10.9	10.3	10.2	11.1	11.2	
2008-09-08 21:00:00	18.0	26.9	20.6	23.7	26.8	26.1	27.7	28.5	
2008-09-04 21:00:00	25.4	24.9	25.3	25.7	25.4	24.9	24.9	25.0	
2008-08-19 03:00:00	Nan	Nan	Nan	1.1	Nan	Nan	Nan	Nan	
2008-07-19 21:00:00	18.4	22.5	26.1	24.9	26.6	26.4	23.6	25.0	
2008-06-24 09:00:00	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan	
2008-06-22 21:00:00	16.5	16.4	21.0	16.7	15.6	20.5	16.1	17.9	
2008-06-04 21:00:00	17.1	15.8	16.7	16.2	23.3	16.8	16.6	16.9	
2008-06-01 21:00:00	16.8	14.4	15.6	16.5	14.6	13.8	14.0	13.5	
2008-05-31 21:00:00	15.0	13.7	13.4	14.4	14.1	13.4	13.0	13.7	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2008-05-25 21:00:00	16.4	9.2	11.6	10.8	6.6	6.8	7.0	7.1	
2008-05-19 21:00:00	17.0	22.2	24.1	24.5	25.1	24.7	24.5	24.1	
2008-05-09 21:00:00	16.0	13.9	15.5	15.4	15.0	14.0	14.0	14.0	
2008-04-14 21:00:00	6.7	8.4	6.7	13.3	8.9	8.2	8.4	7.9	
2008-03-26 21:00:00	3.1	2.8	4.1	-3.3	4.8	5.7	4.5	5.2	
2008-02-24 21:00:00	-1.7	0.0	-4.4	-0.9	-0.6	-0.3	0.5	1.5	
2007-11-19 09:00:00	NaN	-18.3	NaN	NaN	NaN	NaN	NaN	NaN	
2007-09-25 03:00:00	NaN	0.3	NaN	NaN	NaN	NaN	NaN	NaN	
2007-08-27 21:00:00	18.0	17.9	17.6	17.6	18.9	18.8	18.2	11.9	
2007-08-06 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2007-05-16 21:00:00	14.3	22.2	24.1	21.4	27.4	27.7	27.1	27.9	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2007-05-16 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-03-29 15:00:00	-2.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-03-29 09:00:00	NaN	NaN	NaN	-2.8	NaN	NaN	NaN	NaN	NaN
2007-03-28 15:00:00	-2.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-03-08 21:00:00	8.0	7.1	4.8	5.9	7.3	6.9	6.8	6.9	
2007-02-05 21:00:00	-8.3	-2.6	-12.0	-6.3	-3.0	-3.8	-2.5	-2.0	
2006-11-07 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	13.9	NaN	
2006-09-23 03:00:00	NaN	NaN	NaN	1.3	NaN	NaN	NaN	NaN	
2006-09-14 03:00:00	-0.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2006-08-23 03:00:00	NaN	NaN	NaN	-1.1	NaN	NaN	NaN	NaN	
2006-04-16 21:00:00	7.0	8.2	6.5	1.5	6.3	6.0	7.3	7.8	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2006-02-24 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2006-01-29 21:00:00	-4.3	-3.2	-15.1	-8.3	-7.6	-9.5	-4.4	-3.8	
2005-09-29 03:00:00	NaN	NaN	NaN	0.5	NaN	NaN	NaN	NaN	
2005-07-31 21:00:00	19.9	25.7	23.4	25.9	26.8	27.5	28.0	28.3	
2005-07-29 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2005-07-24 09:00:00	NaN	NaN	NaN	NaN	12.7	NaN	NaN	NaN	
2005-07-11 09:00:00	NaN	NaN	NaN	NaN	8.7	NaN	NaN	NaN	
2005-06-10 09:00:00	NaN	NaN	NaN	NaN	-1.3	NaN	NaN	NaN	
2005-05-29 09:00:00	NaN	NaN	NaN	NaN	9.3	NaN	NaN	NaN	
2005-03-19 21:00:00	NaN	-8.0	NaN	-6.7	-7.3	-7.4	-8.4	-8.9	
2005-03-19 03:00:00	NaN	-18.1	NaN	NaN	NaN	NaN	NaN	NaN	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2005-02-25 21:00:00	NaN	-999.9	NaN	-5.0	-5.7	-5.4	-6.4	-6.7	

## Применимость критериев ошибочных значений

Рассмотрим полученные данные об аномальных значениях.

In [183...]

```
print(f'В поле метеостанций выявлено аномальных значений максимальной температуры: '
      f'{df_tmp33.error_at.dropna().apply(lambda x: len(x[1])).sum()},\n'
      f'из них:\n'
      f'Подтверждается отклонением от максимальных регулярно фиксируемых значений на более чем 5 градусов: '
      f'{df_tmp33.double_error_at.dropna().apply(lambda x: len(x[1])).sum()}\n')
```

В поле метеостанций выявлено аномальных значений максимальной температуры: 1652,  
из них:

Подтверждается отклонением от максимальных регулярно фиксируемых значений на более чем 5 градусов: 111

**Окончательный критерий ошибочных значений** Ошибкой считается выброс в поле метеостанций или единичное зафиксированное в поле метеостанций, которое отклоняется от максимальной наблюдаемой температуры (по моментам наблюдения) за 12 часов на более чем 5 градусов.

Такие значения необходимо заменить на NaN.

## Удалим (заменим на NaN) все ошибочные значения

In [184...]

```
# Создадим список координат ячеек, содержащих значения, определённые как ошибки
list_error_coords = df_tmp33.dropna(subset=['double_error_at']).double_error_at.tolist()
# list_error_coords
```

In [185...]

```
# Восстановим исходное состояние df_tmp33
df_tmp33 = dict_df_parameters[param_df_name].copy(deep=True)
```

In [186...]

```
for error_coords in list_error_coords: # по списку координат ошибочных значений
    for station in error_coords[1]: # по перечню метеостанций в каждом списке координат ошибочных значений для станций
        # Преобразуем координату столбца и присваиваем ячейке NaN
        df_tmp33.at[error_coords[0], PARAMETER33+'#'+ station] = np.nan
```

```
# df_tmp33
```

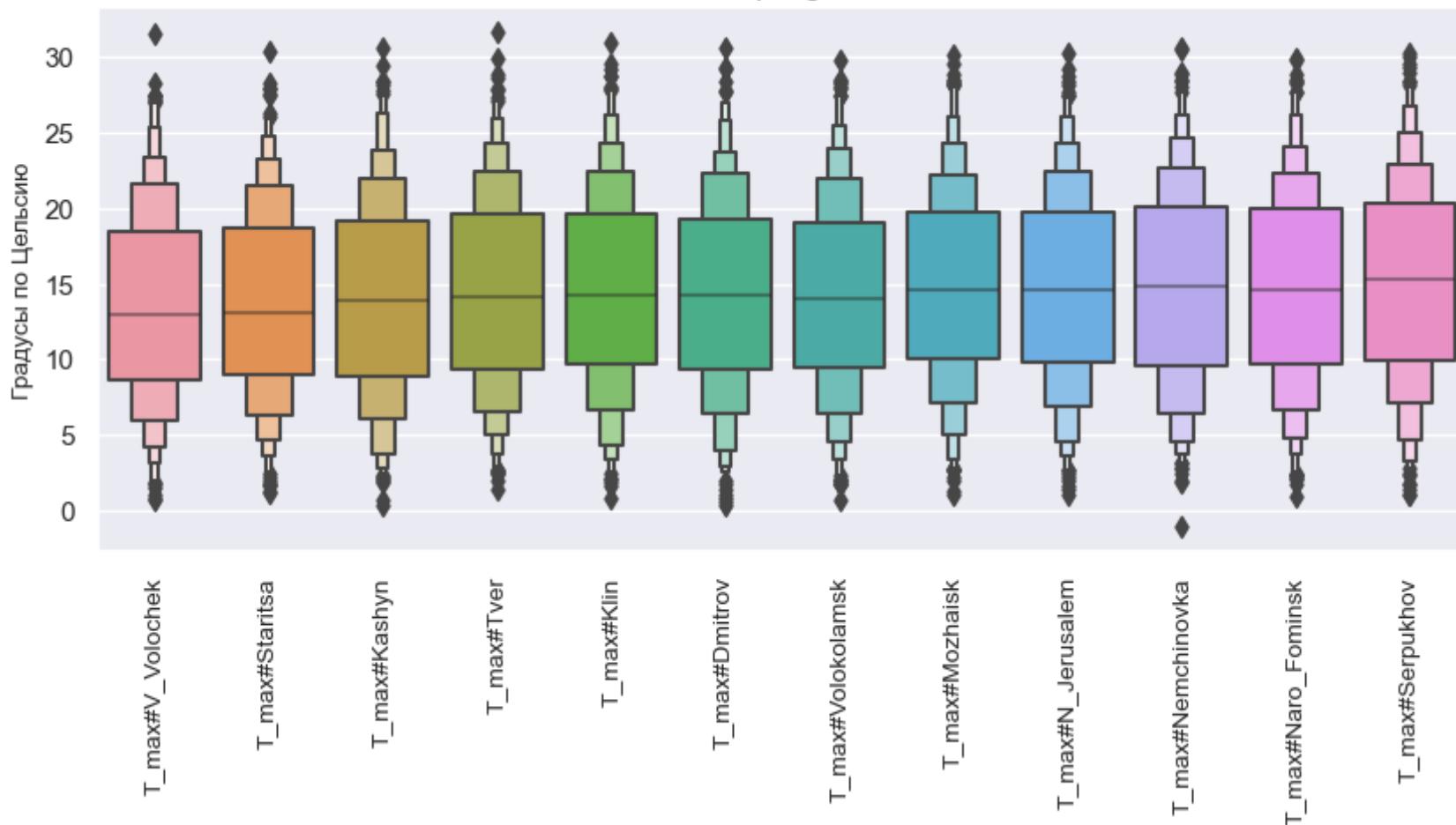
```
In [187...]  
# Проверим, все ли ошибки заменены на NaN  
# Количество неNaN значений в df_tmp33 по координатам, указанным в списках list_error_coords и list_error_at  
  
counter_notna1 = 0 # счётчик неNaN значений  
for error_coords in list_error_coords: # по списку координат ошибочных значений  
    for station in error_coords[1]: # перечисли метеостанций в каждом списке координат ошибочных значений для станций  
        # подсчитаем количество неNaN значений и прибавим их к счётчику неNaN значений.  
        counter_notna1 += np.sum(pd.notna(df_tmp33.at[error_coords[0], PARAMETER33+'#'+ station]))  
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna1}')  
  
# df_tmp33
```

По результатам удаления выявленных аномальных выбросов осталось значений: 0

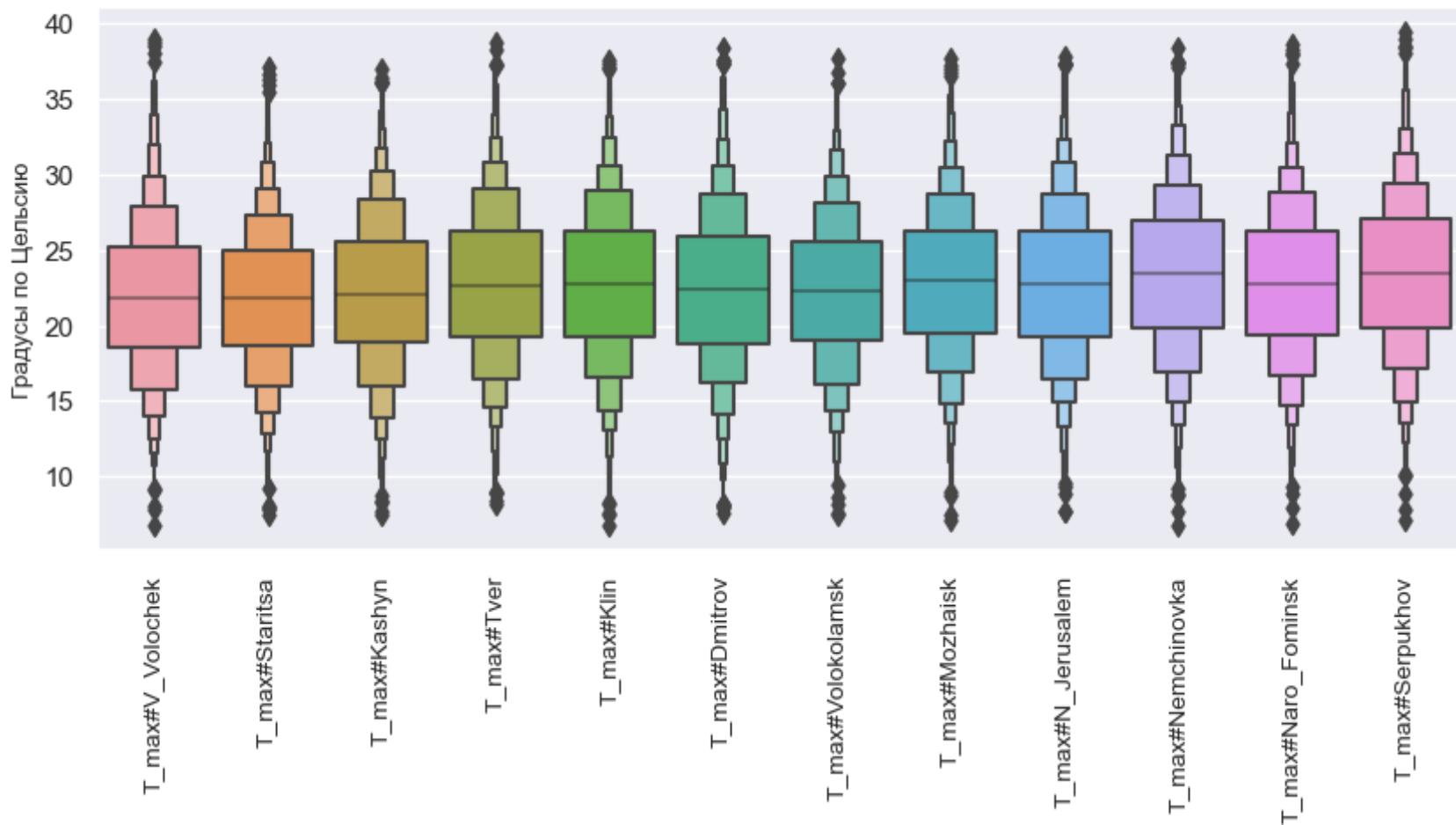
### Визуализируем архив максимальных значений температуры (T\_max) по сезонам после удаления ошибок

```
In [188...]  
# В цикле выведем графики температур по метеостанциями в зависимости от сезона  
# используем функцию создания масок климатических сезонов  
for season_name, season_mask in season_masks(df_tmp33.dropna()).items():  
    fig, ax = plt.subplots(figsize=(10, 4))  
    g = sns.boxenplot(data=df_tmp33.dropna().iloc[:, :12][season_mask],  
                       ax=ax)  
    dummy = plt.xticks(rotation=90, size=10)  
    dummy = g.set_ylabel('Градусы по Цельсию', size=10)  
    dummy = g.set_title(f'Распределение значений T_max в разрезе метеостанций после удаления ошибок:\n{season_name}')  
    plt.show()
```

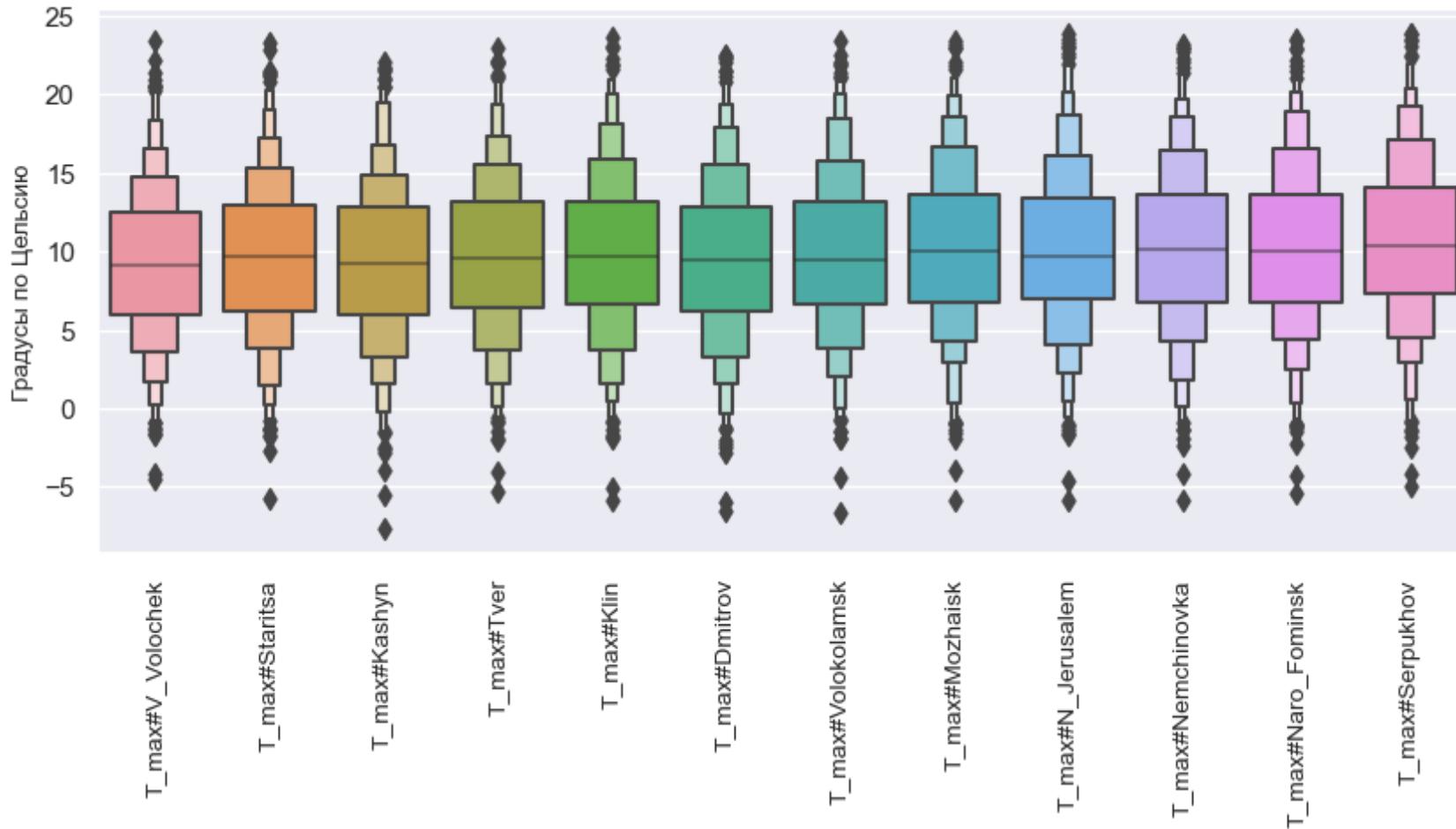
Распределение значений T\_max в разрезе метеостанций после удаления ошибок:  
spring



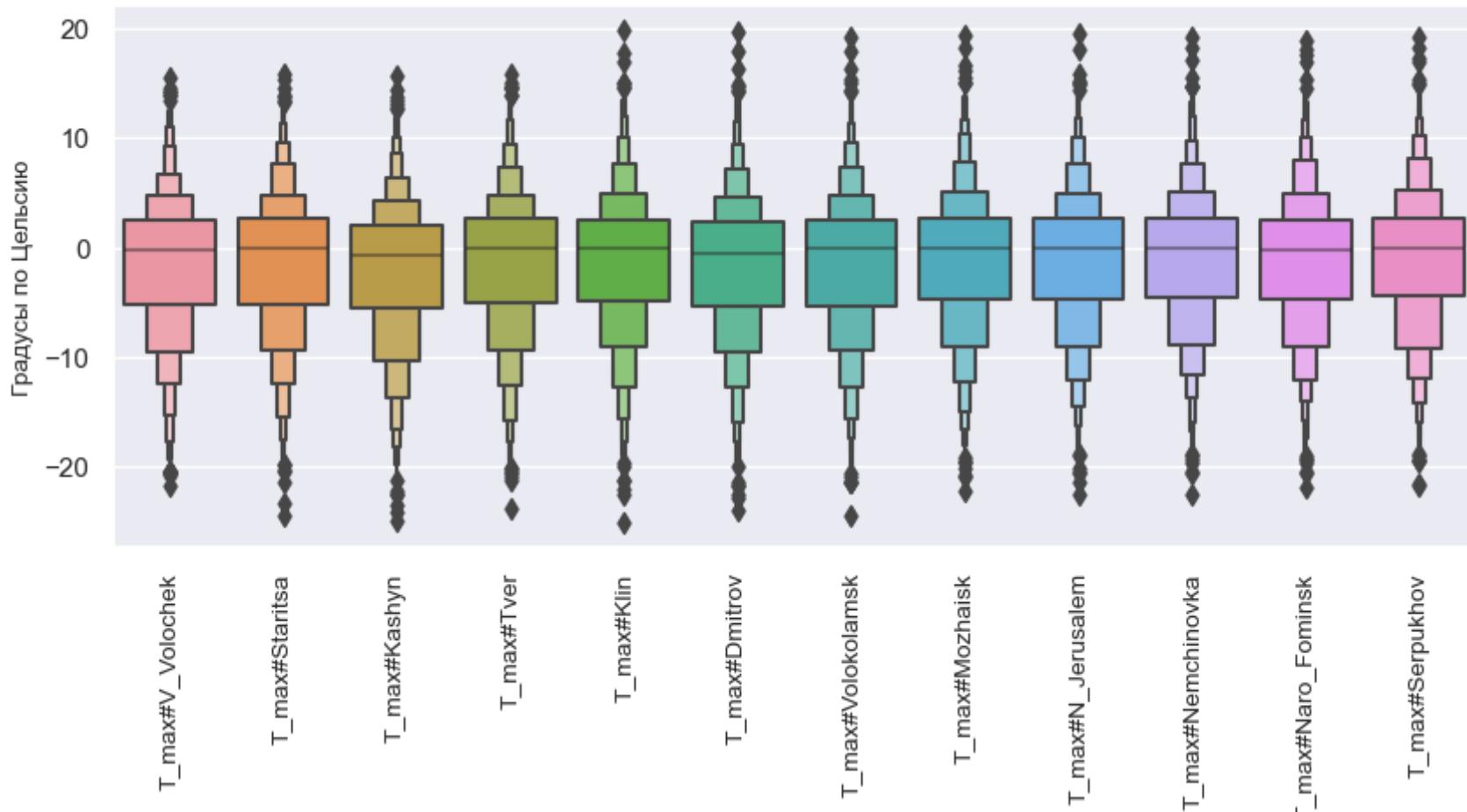
Распределение значений T\_max в разрезе метеостанций после удаления ошибок:  
summer



Распределение значений T\_max в разрезе метеостанций после удаления ошибок:  
autumn



## Распределение значений T\_max в разрезе метеостанций после удаления ошибок: winter



Выведем минимальное и максимальное значения, а также значение медианы и средней для всего DF после удаления ошибок

In [189]:

```
print(f'Минимальное значение: {np.nanmin(df_tmp33)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp33)},\n'
      f'Средняя: {np.nanmean(df_tmp33)},\n'
      f'Медиана: {np.nanmedian(df_tmp33)}')
```

Минимальное значение: -29.6,  
Максимальное значение: 39.4,  
Средняя: 9.747989414204994,  
Медиана: 9.5

### Сохраним очищенные данные в файл параметров

In [190...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
clean_path = f'{path}clean/'  
  
makedirs(clean_path, exist_ok=True)  
  
# Запишем текущие данные в файл  
print('df_'+PARAMETER33 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER33].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER33}.csv'  
)  
print('DONE!')  
  
df_T_max.csv -> DONE!
```

### 3.3.2. Восстановление "сплошных" NaN показателя максимальной температуры за 12 часов T\_max методом скользящего максимума зафиксированных по моментам наблюдений фактических значений температуры Т

Модели пространственной экстраполяции не могут экстраполировать значения в рамках одного момента наблюдения, если значения во всех точках наблюдения неизвестны. Такие случаи есть в наших архивах.

In [191...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp33  
# построчно подсчитаем сумму количества NaN,  
# и если оно равно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp33.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp33.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 44291

Очевидно такие строки нужно заполнить до пространственной экстраполяции. Представляется, что лучшим способом заполнения пропущенных строк будет скользящий максимум наблюдений температуры за 12 часов. Эти значения уже определены в df\_t1.

In [192...]

```
# Создадим маску для выбора строк со сплошными NaN  
# Выбираем из датафрейма строки, где количество NaN равно количеству столбцов - то есть сплошные NaN  
mask_total_nans = df_tmp33.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp33.keys()))
```

In [193...]

```
# Выберем значения из df_t1, соответствующие индексам строк сплошных NaN в df_tmp33 и присвоим их ячейкам df_tmp33  
df_tmp33[mask_total_nans] = df_t1[df_t1.index.isin(df_tmp33[mask_total_nans].index)]  
df_tmp33[mask_total_nans].sample(5, random_state=56)
```

Out[193]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2011-09-18 12:00:00	12.4	11.7	12.1	13.8	13.2	12.7	11.6	12.5	
2007-10-02 00:00:00	20.9	21.3	21.6	21.4	23.0	21.6	21.7	22.1	
2019-11-07 03:00:00	0.3	2.7	2.8	2.9	5.2	6.6	8.3	10.2	
2010-02-05 03:00:00	-6.2	-7.2	-8.5	-6.6	-7.1	-7.6	-8.0	-8.2	
2021-05-22 12:00:00	18.5	18.1	17.5	18.5	18.4	18.1	18.4	19.0	

In [194...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp33  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp33.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp33.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 3

Всё верно, это самые начальные 3 строки, для которых невозможно определить 12 часовое окно.

Поскольку в процессе удаления ошибочных значений удалялись и единичные значения в строках, и часть строк могла превратиться в сплошные NaN. Проверим, сколько единичных значений исправлено методом восстановления сплошных NaN

```
In [195...]
# Проверим, все ли ошибки заменены на NaN
# Количество нeNaN значений в df_tmp33 по координатам, указанным в списках list_error_coords и list_error_at

counter_notna1 = 0 # счётчик нeNaN значений
for error_coords in list_error_coords: # по списку координат ошибочных значений
    for station in error_coords[1]: # перечню метеостанций в каждом списке координат ошибочных значений для станций
        # подсчитаем количество нeNaN значений и прибавим их к счётуку нeNaN значений.
        counter_notna1 += np.sum(pd.notna(df_tmp33.at[error_coords[0], PARAMETER33+'#'+ station]))
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna1}')

# df_tmp33
```

По результатам удаления выявленных аномальных выбросов осталось значений: 44

Получившееся значение показывает количество исправленных ошибочных значений, там где был применён метод восстановления сплошных NaN.

### 3.3.3. Подбор модели для восстановления пропущенных и удалённых значений показателя максимальной температуры T\_max, а также для моделирования значений для искомой точки.

Важным моментом для моделирования недостающих значений и значений для искомой точки является то, что большинство значений показателя T\_max было изначально состояло из сплошных строк NaN. Они были механически заполнены скользящими максимумами из значений температуры. Представляется более логичным не моделировать их для искомых точек, а также заполнить. Поэтому их следует исключить из обучающего и валидационного датасетов.

#### 3.3.3.1. Модель средней, взвешенной по степени обратных расстояний (далее по тексту эту модель будем называть сокращённо IDW)

Эта модель уже применялась выше

Попробуем, как работает модель с различными значениями степени для обратных расстояний, и выделим ту степень, которая обеспечивает лучшие метрики качества.

Создадим набор данных для обучения и валидации модели IDW.

1. Используем данные в df\_tmp33.
2. Уберём все строки с NaN.
3. Выделим рандомно массив известных значений для разных метеостанций и разных моментов наблюдения - он потребуется для разделения на обучающую и валидационную часть и для расчёта метрик качества.

In [196]:

```
# Создаём датафрейм для валидации модели IDW:  
# - не включаем изначальные строки сплошных NaN  
# - убираем все строки с хотя бы одним оставшимся NaN  
df_test = df_tmp33[~mask_total_nans].dropna(how='any')
```

In [197]:

```
df_test.info()  
df_test.shape  
  
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 3624 entries, 2022-06-08 21:00:00 to 2007-09-26 21:00:00  
Data columns (total 12 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --    
 0   T_max#V_Volochek    3624 non-null   float64  
 1   T_max#Staritsa     3624 non-null   float64  
 2   T_max#Kashyn       3624 non-null   float64  
 3   T_max#Tver          3624 non-null   float64  
 4   T_max#Klin          3624 non-null   float64  
 5   T_max#Dmitrov      3624 non-null   float64  
 6   T_max#Volokolamsk  3624 non-null   float64  
 7   T_max#Mozhaisk     3624 non-null   float64  
 8   T_max#N_Jerusalem  3624 non-null   float64  
 9   T_max#Nemchinovka 3624 non-null   float64  
 10  T_max#Naro_Fominsk 3624 non-null   float64  
 11  T_max#Serpukhov    3624 non-null   float64  
dtypes: float64(12)  
memory usage: 368.1 KB  
(3624, 12)
```

Out[197]:

Создадим массив индексов для выборки моделируемых и проверочных значений. Массив будет включать последовательно все индексы строк и случайный индекс столбца.

In [198...]

```
# Зафиксируем RandomState
rs56 = np.random.RandomState(56)
# Создаём 1мерный массив с последовательностью, равной количеству рядов в df_test
arr_row_index = np.arange(0, df_test.shape[0])
# Создаём 1мерный массив со случайными целыми числами в диапазоне количества столбцов в df_test
arr_column_index = rs56.randint(0, df_test.shape[1], df_test.shape[0])
# Соединяем 2 массива и транспонируем полученный массив
arr_idx_data = np.vstack((arr_row_index, arr_column_index)).T

arr_row_index
arr_column_index
arr_idx_data
```

Out[198]: array([ 0, 1, 2, ..., 3621, 3622, 3623])

Out[198]: array([ 5, 4, 0, ..., 5, 11, 8])

Out[198]: array([[ 0, 5],
 [ 1, 4],
 [ 2, 0],
 ...,
 [3621, 5],
 [3622, 11],
 [3623, 8]])

Создадим тренировочный и обучающий массивы. Для этого:

- определим  $X$  как массив координат ячеек в архиве - (np.array),
- определим  $y$  как массив значений ячеек в множестве  $X$  - (np.array).

In [199...]

```
# Разделим наши данные на обучающую и валидационную части.
# используем индексы значений как параметры модели
X = arr_idx_data
# результирующие значения (фактические значения измерений, соответствующие индексам), выведем в список и преобразуем в np.array
y = np.array([df_test.iloc[x, y] for x, y in arr_idx_data])

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.6, test_size=0.4, random_state=56)
x_train.shape
y_train.shape
```

```
x_test.shape  
y_test.shape  
  
Out[199]: (2174, 2)  
Out[199]: (2174,)  
Out[199]: (1450, 2)  
Out[199]: (1450,)
```

**Обучающий датасет** Подберём лучшую степень для весов - обратных расстояний, ориентируясь на лучшие значения метрик качества

```
In [200... start_time = time.time() # для замера времени выполнения кода  
  
r2_idw_tr, mae_idw_tr, rmse_idw_tr = 0, 0, 0 # обнулим значения метрик качества  
iterations = 0 # количество итераций  
power = 3 # Начальное значение степени (опробованы начальные степени от 1)  
power_increment = 0.25 # шаг увеличения степени  
list_metrics=[] # список для фиксации результатов итераций  
  
r2_idw_tr_old = 0 # Устанавливаем в 0 предыдущее значение R2  
print('ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:\n')  
  
# Зададим предел R2 для поиска оптимального веса  
while r2_idw_tr_old <= r2_idw_tr:  
    power += power_increment # Увеличиваем степень на 1 шаг  
    iterations +=1 # Увеличиваем счётчик проходов цикла  
    r2_idw_tr_old = r2_idw_tr # Запоминаем предыдущее значение R2  
  
    # Создаём y_predict_idw_tr по индексам в массиве x_train  
    # используем функцию inverse_distance_avg  
    # Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком  
    y_predict_idw_tr = [  
        inverse_distance_avg(row=df_test.iloc[x],  
                             param_=PARAMETER33,  
                             station_=df_test.keys()[y][len(PARAMETER33)+1:],  
                             df_dists_= df_station_dists,  
                             power_=power)  
        for x, y in x_train  
    ]
```

```
# Расчитываем метрики качества
max_e_idw_tr = max_error(y_train, y_predict_idw_tr)
mae_idw_tr = mean_absolute_error(y_train, y_predict_idw_tr)
mse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr)
rmse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr, squared=False)
r2_idw_tr = r2_score(y_train, y_predict_idw_tr)

if r2_idw_tr > r2_idw_tr_old:
    best_power_idw_tr = power
    best_max_e_idw_tr = max_e_idw_tr
    best_mae_idw_tr = mae_idw_tr
    best_mse_idw_tr = mse_idw_tr
    best_rmse_idw_tr = rmse_idw_tr
    best_r2_idw_tr = r2_idw_tr

# замер времени:
chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

print(f'Elapsed time={time_formatted}\n'
      f'Текущие значения: iterations={iterations}, power={power},\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_tr:.7f}\n'
      )
print(f'ЛУЧШИЕ значения: power={best_power_idw_tr},\n'
      f'Максимальная ошибка (MAX_E) IDW = {best_max_e_idw_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {best_mae_idw_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {best_mse_idw_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {best_rmse_idw_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {best_r2_idw_tr:.7f}')
)
```

ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:

Elapsed time=00:00:08

Текущие значения: iterations=1, power=3.25,  
Максимальная ошибка (MAX\_E) IDW = 5.6724600  
Средняя абсолютная ошибка (MAE) IDW = 0.6981071  
Средний квадрат ошибки (MSE) IDW = 0.9872025  
Средняя квадратическая ошибка (RMSE) IDW = 0.9935806  
Коэффициент детерминации (R2) IDW = 0.9930601

Elapsed time=00:00:16

Текущие значения: iterations=2, power=3.5,  
Максимальная ошибка (MAX\_E) IDW = 5.6601241  
Средняя абсолютная ошибка (MAE) IDW = 0.6927352  
Средний квадрат ошибки (MSE) IDW = 0.9697069  
Средняя квадратическая ошибка (RMSE) IDW = 0.9847370  
Коэффициент детерминации (R2) IDW = 0.9931831

Elapsed time=00:00:24

Текущие значения: iterations=3, power=3.75,  
Максимальная ошибка (MAX\_E) IDW = 5.6523429  
Средняя абсолютная ошибка (MAE) IDW = 0.6885119  
Средний квадрат ошибки (MSE) IDW = 0.9554725  
Средняя квадратическая ошибка (RMSE) IDW = 0.9774827  
Коэффициент детерминации (R2) IDW = 0.9932832

Elapsed time=00:00:32

Текущие значения: iterations=4, power=4.0,  
Максимальная ошибка (MAX\_E) IDW = 5.6480793  
Средняя абсолютная ошибка (MAE) IDW = 0.6853325  
Средний квадрат ошибки (MSE) IDW = 0.9440712  
Средняя квадратическая ошибка (RMSE) IDW = 0.9716333  
Коэффициент детерминации (R2) IDW = 0.9933633

Elapsed time=00:00:40

Текущие значения: iterations=5, power=4.25,  
Максимальная ошибка (MAX\_E) IDW = 5.6464157  
Средняя абсолютная ошибка (MAE) IDW = 0.6828751  
Средний квадрат ошибки (MSE) IDW = 0.9351201  
Средняя квадратическая ошибка (RMSE) IDW = 0.9670161  
Коэффициент детерминации (R2) IDW = 0.9934263

Elapsed time=00:00:48

Текущие значения: iterations=6, power=4.5,

Максимальная ошибка (MAX\_E) IDW = 5.6465569  
Средняя абсолютная ошибка (MAE) IDW = 0.6811443  
Средний квадрат ошибки (MSE) IDW = 0.9282779  
Средняя квадратическая ошибка (RMSE) IDW = 0.9634718  
Коэффициент детерминации (R2) IDW = 0.9934744

Elapsed time=00:00:55  
Текущие значения: iterations=7, power=4.75,  
Максимальная ошибка (MAX\_E) IDW = 5.6478270  
Средняя абсолютная ошибка (MAE) IDW = 0.6800254  
Средний квадрат ошибки (MSE) IDW = 0.9232425  
Средняя квадратическая ошибка (RMSE) IDW = 0.9608551  
Коэффициент детерминации (R2) IDW = 0.9935098

Elapsed time=00:01:03  
Текущие значения: iterations=8, power=5.0,  
Максимальная ошибка (MAX\_E) IDW = 5.6496639  
Средняя абсолютная ошибка (MAE) IDW = 0.6793632  
Средний квадрат ошибки (MSE) IDW = 0.9197472  
Средняя квадратическая ошибка (RMSE) IDW = 0.9590345  
Коэффициент детерминации (R2) IDW = 0.9935343

Elapsed time=00:01:11  
Текущие значения: iterations=9, power=5.25,  
Максимальная ошибка (MAX\_E) IDW = 5.6516107  
Средняя абсолютная ошибка (MAE) IDW = 0.6791578  
Средний квадрат ошибки (MSE) IDW = 0.9175585  
Средняя квадратическая ошибка (RMSE) IDW = 0.9578928  
Коэффициент детерминации (R2) IDW = 0.9935497

Elapsed time=00:01:19  
Текущие значения: iterations=10, power=5.5,  
Максимальная ошибка (MAX\_E) IDW = 5.6533056  
Средняя абсолютная ошибка (MAE) IDW = 0.6793713  
Средний квадрат ошибки (MSE) IDW = 0.9164729  
Средняя квадратическая ошибка (RMSE) IDW = 0.9573259  
Коэффициент детерминации (R2) IDW = 0.9935573

Elapsed time=00:01:27  
Текущие значения: iterations=11, power=5.75,  
Максимальная ошибка (MAX\_E) IDW = 5.6544714  
Средняя абсолютная ошибка (MAE) IDW = 0.6798882  
Средний квадрат ошибки (MSE) IDW = 0.9163133  
Средняя квадратическая ошибка (RMSE) IDW = 0.9572426

Коэффициент детерминации (R2) IDW = 0.9935585

Elapsed time=00:01:35

Текущие значения: iterations=12, power=6.0,  
Максимальная ошибка (MAX\_E) IDW = 5.6549042  
Средняя абсолютная ошибка (MAE) IDW = 0.6807116  
Средний квадрат ошибки (MSE) IDW = 0.9169270  
Средняя квадратическая ошибка (RMSE) IDW = 0.9575630  
Коэффициент детерминации (R2) IDW = 0.9935542

ЛУЧШИЕ значения: power=5.75,  
Максимальная ошибка (MAX\_E) IDW = 5.6544714  
Средняя абсолютная ошибка (MAE) IDW = 0.6798882  
Средний квадрат ошибки (MSE) IDW = 0.9163133  
Средняя квадратическая ошибка (RMSE) IDW = 0.9572426  
Коэффициент детерминации (R2) IDW = 0.9935585

Несколько запусков кода выше, с различными параметрами степени (от 1 до 10), показали, что, судя по R2, лучшим значением степени на обучающем массиве является 5,25. Оно даёт лучшие метрики качества.

Применим эту степень для валидационного массива.

## Валидационный датасет

In [201...]

```
# Создаём y_predict_idw_vld по индексам в x_test
# используем функцию inverse_distance_avg
# Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком

y_predict_idw_vld = [
    inverse_distance_avg(row=df_test.iloc[x],
                          param=PARAMETER33,
                          station=df_test.keys()[y][len(PARAMETER33)+1:],
                          df_dists=df_station_dists,
                          power=best_power_idw_tr) # берём лучшее значение степени, полученное из кода выше на тренинговом датасете
    for x, y in x_test
]
# y_predict_idw_vld

max_e_idw_vld = max_error(y_test, y_predict_idw_vld)
mae_idw_vld = mean_absolute_error(y_test, y_predict_idw_vld)
mse_idw_vld = mean_squared_error(y_test, y_predict_idw_vld)
rmse_idw_vld = mean_squared_error(y_test, y_predict_idw_vld, squared=False)
r2_idw_vld = r2_score(y_test, y_predict_idw_vld)
```

```
print(f'ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld:.7f}')
)
```

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:

Максимальная ошибка (MAX\_E) IDW = 5.4204585  
Средняя абсолютная ошибка (MAE) IDW = 0.6852596  
Средний квадрат ошибки (MSE) IDW = 0.9087468  
Средняя квадратическая ошибка (RMSE) IDW = 0.9532821  
Коэффициент детерминации (R2) IDW = 0.9935060

## ВЫВОД

Модель средней, взвешенной по обратным расстояниям, изначально даёт очень хорошие метрики качества.

### 3.3.3.2. Кригинг и вариограммы в реализации библиотеки SciKit GStat

Опробуем работу модели кригинга на уже сформированных тренинговой и валидационной выборках

Координатная сетка для точек наблюдения в данной модели строится на основе "плоской" земной поверхности. Разность между расстояниями по прямой и по поверхности сферы игнорируется (впрочем, для нашего региона исследования это не приведёт к существенным ошибкам).

```
In [202...]: # Загружаем координаты из df_coords_full (определен в разделе определения функции кригинга 2.2):
# Создаём DF с координатами нужных нам точек
df_coords = df_coords_full[["LoE", "LaN"]][:-2]

df_coords
```

Out[202]:

LoE      NaN

station	LoE	NaN
V_Volochek	34.566667	57.583333
Staritsa	34.933333	56.500000
Kashyn	37.583333	57.350000
Tver	35.922000	56.857300
Klin	36.716667	56.333333
Dmitrov	37.533333	56.366667
Volokolamsk	35.933333	56.016700
Mozhaisk	36.000000	55.516700
N_Jerusalem	36.816667	55.900000
Nemchinovka	37.350000	55.716667
Naro_Fominsk	36.700000	55.383333
Serpukhov	37.416667	54.916667

## Обучающий датасет

Проверим работу модели кригинга сначала на данных обучающей выборки. На ней будем подбирать наиболее подходящие параметры вариограмм и модели кригинга (которые дают лучшие метрики и выдают меньшее количество ошибок из-за недостаточности наблюдений в поле метеостанций).

Представляется полезным сравнить работу модели обратных степеней расстояний и кригинга на одних и тех же данных.

В процессе исследования работоспособности модели код ниже многократно запускался с различными параметрами.

In [203...]

```
# Намеренно оставим закомментированные части кода, они могут использоваться для отладки
start_time = time.time() # для замера времени выполнения кода
# counter = 0
y_predict_kriging_tr = [] # список предиктов для x_test
```

```
for x in x_train: # x[0] ряд в df_test, x[1] столбец, соответствующий названию станции
#     counter += 1
##
#     if counter >15:
#         break
#     else:
#         counter +=1
##
# Найдем по координатам x_test ряд в df_test
row = df_test.iloc[x[0]]
# Присваиваем проверяемому значению NaN
row.iloc[x[1]] = np.nan
# Для построения вариограммы:
# - получаем массив координат без указанной точки
# (удаляем соответствующий ряд из df_coords, преобразуем оставшийся df в массив)
# - получаем массив значений
idx = x[1]
coords_v = np.array(df_coords.drop(index = df_coords.iloc[x[1]].name))[:,2]
vals_v = np.array(row.dropna())
try:
    # Определяем вариограмму
    V = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.99999, # Используем всю матрицу расстояний
                        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                        normalize=False,
                        use_nugget=False,
                        samples=len(vals_v) # количество сэмплов приравняем к количеству наблюдений
                        #fit_method='ml',
                        #entropy_bins = 1
                        )
    # V_North = skg.DirectionalVariogram(coordinates=coords_v,
    #                                     values=vals_v,
    #                                     estimator='matheron',
    #                                     model='spherical',
    #                                     dist_func='euclidean',
    #                                     bin_func='even',
    #                                     azimuth=90,
    #                                     tolerance=90,
    #                                     maxLag='full',
    #                                     )

```

```

#                                     n_Lags=4)
# V_East = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=0,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)
# V_South = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=-90,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)
# V_West = skg.DirectionalVariogram(coordinates=coords_v,
#                                     values=vals_v,
#                                     estimator='matheron',
#                                     model='spherical',
#                                     dist_func='euclidean',
#                                     bin_func='even',
#                                     azimuth=180,
#                                     tolerance=90,
#                                     maxlag='full',
#                                     n_Lags=4)

##
# V=V_West
##
```

**except ValueError:** # Уберём количество сэмплов из определения вариограммы (когда количество сэмплов слишком велико)

```

try:
    V_ = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.9999, # Используем всю матрицу расстояний

```

```

        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
        normalize=False,
        use_nugget=False,
        #fit_method='ml',
        #entropy_bins = 1
    );
except: # Возникает ошибка - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_tr.append(val_predict)
    continue
except: # возникает другая ошибка (помимо ValueError) - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_tr.append(val_predict)
    continue

# Определяем модель кригинга
model_ok = skg.OldinaryKriging(V, min_points=1, max_points=14, mode='exact')

# координаты точки предикта:
predict_coords = np.array(df_coords)
# Получаем предикт для тестируемой точки (получаем массив предиктов, выбираем из него индекс точки предикта)
# В процессе работы model_ok.transform выдается много сообщений типа:
# 'Warning: for %d Locations, not enough neighbors were found within the range.' % self.no_points_error
# "Заглушим" их, направив их в класс вывода, который ничего не делает (определен выше)

sys.stdout = NullIO() # определим вывод print() в класс нулевого вывода

val_predict = model_ok.transform(predict_coords[:,0], predict_coords[:,1])[x[1]]
sys.stdout = real_stdout # восстановим стандартный вывод

# добавляем предикт в список предиктов
y_predict_kriging_tr.append(val_predict)

## 
# if V.describe()['effective_range'] < 1:
#     dm = pdist(df_coords[['LoE', 'LaN']]) # массив пар расстояний
#     print(f'Итерация: {counter}, позиция в x_test: {x}\nКоличество пар расстояний: {len(dm)}\n'
#     f'Effective Range: {V.describe()["effective_range"]}\n'
#     f'Количество пар расстояний внутри Effective range: {sum(dm <= V.describe()["effective_range"])}\n'
#     f'Количество пар расстояний вне Effective range: {sum(dm > V.describe()["effective_range"])}\n'
#     f'Предикт: {val_predict}, координаты предикта: {predict_coords[x[1]]}, станция: {df_coords.iloc[x[1]].name}'
#     )
# fig = V.plot(show=False)

```

```
#         plt.show()
##  
y_predict_kriging_tr = np.array(y_predict_kriging_tr) # превратим список предиктов в массив  
  
# замер времени:  
chk_time = time.time()  
elapsed_time = chk_time - start_time  
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
```

```
In [204...]  
if np.isnan(np.array(y_predict_kriging_tr)).sum() == 0:  
    max_e_kriging_tr = max_error(y_train, y_predict_kriging_tr)  
    mae_kriging_tr = mean_absolute_error(y_train, y_predict_kriging_tr)  
    mse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr)  
    rmse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr, squared=False)  
    r2_kriging_tr = r2_score(y_train, y_predict_kriging_tr)  
else:  
    max_e_kriging_tr = np.nan  
    mae_kriging_tr = np.nan  
    mse_kriging_tr = np.nan  
    rmse_kriging_tr = np.nan  
    r2_kriging_tr = np.nan  
  
print('ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ')  
print(f'Elapsed time={time_formatted}\n'  
      f'Значения метрик:\n'  
      f'R2={r2_kriging_tr:.7f}, MAX_E={max_e_kriging_tr}, MAE={mae_kriging_tr:.7f}, MSE={mse_kriging_tr}, '  
      f'RMSE={rmse_kriging_tr:.7f}\n'  
      )  
  
print(f'Количество значений, которые не удалось предсказать: {pd.isna([y_predict_kriging_tr]).sum()}\n'  
      f'Это составляет {pd.isna([y_predict_kriging_tr]).sum()/len(y_predict_kriging_tr):.4%} от обучающего массива данных')
```

ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ  
Elapsed time=00:00:33  
Значения метрик:  
R2=nan, MAX\_E=nan, MAE=nan, MSE=nan, RMSE=nan

Количество значений, которые не удалось предсказать: 42  
Это составляет 1.9319% от обучающего массива данных

Попробуем оценить качество работы модели в случаях, когда ей удалось найти предикты.

In [205...]

```
# Оставим в y_train и y_predict_kriging_tr только значения неравные NaN
y_train_kriging_shrunk = y_train[~np.isnan(y_predict_kriging_tr)]
y_predict_kriging_tr_shrunk = y_predict_kriging_tr[~np.isnan(y_predict_kriging_tr)]

max_e_kriging_tr = max_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
mae_kriging_tr = mean_absolute_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
mse_kriging_tr = mean_squared_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
rmse_kriging_tr = mean_squared_error(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk, squared=False)
r2_kriging_tr = r2_score(y_train_kriging_shrunk, y_predict_kriging_tr_shrunk)
```

In [206...]

```
print(f'КРИГИНГ на ОБУЧАЮЩЕМ датасете (для случаев, где удалось найти предикты):\n'
      f'Максимальная ошибка (MAX_E) Kriging = {max_e_kriging_tr:.7f}, Kriging - IDW = '
      f'{(max_e_kriging_tr - max_e_idw_tr):.7f}\n'
      f'Средняя абсолютная ошибка (MAE) Kriging = {mae_kriging_tr:.7f}, Kriging - IDW = '
      f'{(mae_kriging_tr - mae_idw_tr):.7f}\n'
      f'Средний квадрат ошибки (MSE) Kriging = {mse_kriging_tr:.7f}, Kriging - IDW = '
      f'{(mse_kriging_tr - mse_idw_tr):.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) Kriging = {rmse_kriging_tr:.7f}, '
      f'Kriging - IDW = {(rmse_kriging_tr - rmse_idw_tr):.7f}\n'
      f'Коэффициент детерминации (R2) Kriging = {r2_kriging_tr:.7f}, Kriging - IDW = '
      f'{(r2_kriging_tr - r2_idw_tr):.7f}')
)
```

КРИГИНГ на ОБУЧАЮЩЕМ датасете (для случаев, где удалось найти предикты):

Максимальная ошибка (MAX\_E) Kriging = 5.6971170, Kriging - IDW = 0.0422127  
Средняя абсолютная ошибка (MAE) Kriging = 0.6609031, Kriging - IDW = -0.0198085  
Средний квадрат ошибки (MSE) Kriging = 0.8646457, Kriging - IDW = -0.0522813  
Средняя квадратическая ошибка (RMSE) Kriging = 0.9298633, Kriging - IDW = -0.0276998  
Коэффициент детерминации (R2) Kriging = 0.9939234, Kriging - IDW = 0.0003692

## Валидационный датасет

Посмотрим, как модель будет отрабатывать на валидационной выборке.

In [207...]

```
start_time = time.time() # для замера времени выполнения кода
# counter = 0
y_predict_kriging_vld = [] # список предиктов для x_test

for x in x_test: # x[0] ряд в df_test, x[1] столбец, соответствующий названию станции
    #     counter += 1
    ##     if counter >15:
```

```

#         break
#     else:
#         counter +=1
##  

# Найдем по координатам x_test ряд в df_test
row = df_test.iloc[x[0]]
# Присваиваем проверяемому значению NaN
row.iloc[x[1]] = np.nan
# Для построения вариограммы:
# - получаем массив координат без указанной точки
# (удаляем соответствующий ряд из df_coords, преобразуем оставшийся df в массив)
# - получаем массив значений
idx = x[1]
coords_v = np.array(df_coords.drop(index = df_coords.iloc[x[1]].name))[:, :2]
vals_v = np.array(row.dropna())

try:
    # Определяем вариограмму
    V = skg.Variogram(coordinates=coords_v,
                        values=vals_v,
                        estimator='matheron',
                        model='spherical',
                        dist_func='euclidean',
                        bin_func='ward',
                        maxlag=0.99999, # Используем всю матрицу расстояний
                        n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                        normalize=False,
                        use_nugget=False,
                        samples=len(vals_v),
                        fit_method='trf',
                        )
except ValueError: # Уберём количество сэмплов из определения вариограммы (когда количество сэмплов слишком велико)
    try:
        V_ = skg.Variogram(coordinates=coords_v,
                            values=vals_v,
                            estimator='matheron',
                            model='spherical',
                            dist_func='euclidean',
                            bin_func='ward',
                            maxlag=0.99999, # Используем всю матрицу расстояний
                            n_lags=4, # количество бин не должно быть очень большим, чтобы не уменьшать effective distance
                            normalize=False,
                            use_nugget=False,
                            #fit_method='ml',
    
```

```

        #entropy_bins = 1
    );
except: # Возникает ошибка - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_vld.append(val_predict)
    continue
except: # возникает другая ошибка (помимо ValueError) - не можем построить вариаграмму
    val_predict = np.nan
    y_predict_kriging_vld.append(val_predict)
    continue

# Определяем модель кригинга
model_ok = skg.OldinaryKriging(V, min_points=1, max_points=14, mode='exact')

# координаты точки предикта:
predict_coords = np.array(df_coords)
# Получаем предикт для тестируемой точки (получаем массив предиктов, выбираем из него индекс точки предикта)
# В процессе работы model_ok.transform выдается много сообщений типа:
# 'Warning: for %d Locations, not enough neighbors were found within the range.' % self.no_points_error
# "Заглушим" их, направив их в класс вывода, который ничего не делает (определен выше)

sys.stdout = NullIO() # определим вывод print() в класс нулевого вывода

val_predict = model_ok.transform(predict_coords[:,0], predict_coords[:,1])[x[1]]
sys.stdout = real_stdout # восстановим стандартный вывод

# добавляем предикт в список предиктов
y_predict_kriging_vld.append(val_predict)

y_predict_kriging_vld = np.array(y_predict_kriging_vld)

# замер времени:
chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

```

In [208...]

```

if np.isnan(np.array(y_predict_kriging_vld)).sum() == 0:
    max_e_kriging_vld = max_error(y_test, y_predict_kriging_vld)
    mae_kriging_vld = mean_absolute_error(y_test, y_predict_kriging_vld)
    mse_kriging_vld = mean_squared_error(y_test, y_predict_kriging_vld)
    rmse_kriging_vld = mean_squared_error(y_test, y_predict_kriging_vld, squared=False)
    r2_kriging_vld = r2_score(y_test, y_predict_kriging_vld)

```

```

else:
    max_e_kriging_vld = np.nan
    mae_kriging_vld = np.nan
    mse_kriging_vld = np.nan
    rmse_kriging_vld = np.nan
    r2_kriging_vld = np.nan

print(f'Elapsed time={time_formatted}\n'
      f'Значения метрик:\n'
      f'R2={r2_kriging_vld:.7f}, MAX_E={max_e_kriging_vld:.7f}, MSE={mse_kriging_vld:.7f}, '
      f'RMSE={rmse_kriging_vld:.7f}\n')
print('ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ')
print(f'Количество значений, которые не удалось предсказать: {np.isnan([y_predict_kriging_vld]).sum()}\n'
      f'Это составляет {pd.isna([y_predict_kriging_vld]).sum()/len(y_predict_kriging_vld):.4%} '
      f'от валидационного массива данных')

```

Elapsed time=00:00:22

Значения метрик:

R2=nan, MAX\_E=nan, MAE=nan, MSE=nan, RMSE=nan

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ

Количество значений, которые не удалось предсказать: 45

Это составляет 3.1034% от валидационного массива данных

In [209...]

```

y_test_kriging_shrunk = y_test[~np.isnan(y_predict_kriging_vld)]
y_predict_kriging_vld_shrunk = y_predict_kriging_vld[~np.isnan(y_predict_kriging_vld)]

max_e_kriging_vld = max_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
mae_kriging_vld = mean_absolute_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
mse_kriging_vld = mean_squared_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)
rmse_kriging_vld = mean_squared_error(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk, squared=False)
r2_kriging_vld = r2_score(y_test_kriging_shrunk, y_predict_kriging_vld_shrunk)

```

In [210...]

```

print(f'Кригинг на ВАЛИДАЦИОННОМ датасете (для случаев, где удалось найти предикты):\n'
      f'Максимальная ошибка (MAX_E) Kriging = {max_e_kriging_vld:.7f}, Kriging - IDW = '
      f'{(max_e_kriging_vld - max_e_idw_vld):.7f}\n'
      f'Средняя абсолютная ошибка (MAE) Kriging = {mae_kriging_vld:.7f}, '
      f'Kriging - IDW = {(mae_kriging_vld - mae_idw_vld):.7f}\n'
      f'Средний квадрат ошибки (MSE) Kriging = {mse_kriging_vld:.7f}, Kriging - IDW = '
      f'{(mse_kriging_vld - mse_idw_vld):.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) Kriging = {rmse_kriging_vld:.7f}, '
      f'Kriging - IDW = {(rmse_kriging_vld - rmse_idw_vld):.7f}\n'
      f'Коэффициент детерминации (R2) Kriging = {r2_kriging_vld:.7f}, Kriging - IDW = '

```

```
f'{(r2_kriging_vld - r2_idw_vld):.7f}'

)
```

Кригинг на ВАЛИДАЦИОННОМ датасете (для случаев, где удалось найти предикты):  
Максимальная ошибка (MAX\_E) Kriging = 7.2695578, Kriging - IDW = 1.8490994  
Средняя абсолютная ошибка (MAE) Kriging = 0.6628003, Kriging - IDW = -0.0224593  
Средний квадрат ошибки (MSE) Kriging = 0.8812254, Kriging - IDW = -0.0275214  
Средняя квадратическая ошибка (RMSE) Kriging = 0.9387361, Kriging - IDW = -0.0145461  
Коэффициент детерминации (R2) Kriging = 0.9937231, Kriging - IDW = 0.0002170

## ВЫВОД

Таким образом, модель кригинга не даёт 100% удовлетворительных результатов, так как в ряде случаев не может предсказать значения, и оказывается неприменимой. Это связано с незначительным размером поля метеостанций и встречающейся ситуацией, когда не удаётся найти ближайшие значения в effective range (расстоянии, вне пределов которого модель считает данные метеостанций статистически независимыми). То есть на этом и большем расстоянии корреляция между значениями показателя у метеостанций становится незначительной или отсутствует.

### 3.3.3.3. Модель кригинга, совмещённая с IDW (для значений, которые кригинг не может предсказать)

Отделим значения, которые не удалось предсказать моделью кригинга, от уже предсказанных значений и формируем новый (сокращённый) обучающий датасет для IDW

In [211...]

```
# Поскольку длины массивов x_train, y_train и y_predict_tr, а также x_test, y_test и y_predict_vld соответственно одинаковы,
# выбираем по индексу значения x_train и y_train, x_test и y_test
# где значения y_predict_kriging_tr равны NaN
# формируем новый массив истинных значений
y_train_idw_shrunk = y_train[np.isnan(y_predict_kriging_tr)]
x_train_idw_shrunk = x_train[np.isnan(y_predict_kriging_tr)]
y_test_idw_shrunk = y_test[np.isnan(y_predict_kriging_vld)]
x_test_idw_shrunk = x_test[np.isnan(y_predict_kriging_vld)]
```

Снова подберём лучшую степень для IDW

In [212...]

```
start_time = time.time() # для замера времени выполнения кода

r2_idw_tr_shrunk = 0 # обнулим значение метрики качества R2
iterations = 0 # количество итераций
power = 1.75 # Начальное значение степени (опробованы начальные степени от 1 до 10)
power_increment = 0.25 # шаг увеличения степени
```

```

list_metrics=[] # список для фиксации результатов итераций

r2_idw_tr_shrunk_old = 0 # Устанавливаем в 0 предыдущее значение R2
print('НОВЫЙ ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:\n')

# Зададим предел R2 для поиска оптимального веса
while r2_idw_tr_shrunk_old <= r2_idw_tr_shrunk:
    power += power_increment # Увеличиваем степень на 1 шаг
    iterations +=1 # Увеличиваем счётчик проходов цикла
    r2_idw_tr_shrunk_old = r2_idw_tr_shrunk # Запоминаем предыдущее значение R2

    # Создаём y_predict_idw_tr_shrunk по индексам в массиве x_train_shrunk
    # используем функцию inverse_distance_avg
    # Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком
    y_predict_idw_tr_shrunk = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param_=PARAMETER33,
                             station_=df_test.keys()[y][len(PARAMETER33)+1:],
                             df_dists_= df_station_dists,
                             power_=power)
        for x, y in x_train_idw_shrunk
    ]

    # Расчитываем метрики качества
    max_e_idw_tr_shrunk = max_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    mae_idw_tr_shrunk = mean_absolute_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    mse_idw_tr_shrunk = mean_squared_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk)
    rmse_idw_tr_shrunk = mean_squared_error(y_train_idw_shrunk, y_predict_idw_tr_shrunk, squared=False)
    r2_idw_tr_shrunk = r2_score(y_train_idw_shrunk, y_predict_idw_tr_shrunk)

    if r2_idw_tr_shrunk > r2_idw_tr_shrunk_old:
        best_power_idw_tr_shrunk = power
        best_max_e_idw_tr_shrunk = max_e_idw_tr_shrunk
        best_mae_idw_tr_shrunk = mae_idw_tr_shrunk
        best_mse_idw_tr_shrunk = mse_idw_tr_shrunk
        best_rmse_idw_tr_shrunk = rmse_idw_tr_shrunk
        best_r2_idw_tr_shrunk = r2_idw_tr_shrunk

    # замер времени:
    chk_time = time.time()
    elapsed_time = chk_time - start_time
    time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))

print(f'Elapsed time={time_formatted}\n'

```

```
f'Текущие значения: iterations={iterations}, power={power},\n'
f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_tr_shrunk:.7f}\n'
f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_tr_shrunk:.7f}\n'
f'Средний квадрат ошибки (MSE) IDW = {mse_idw_tr_shrunk:.7f}\n'
f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_tr_shrunk:.7f}\n'
f'Коэффициент детерминации (R2) IDW = {r2_idw_tr_shrunk:.7f}\n'
)
print(f'ЛУЧШИЕ значения: power={best_power_idw_tr_shrunk},\n'
f'Максимальная ошибка (MAX_E) IDW = {best_max_e_idw_tr_shrunk:.7f}\n'
f'Средняя абсолютная ошибка (MAE) IDW = {best_mae_idw_tr_shrunk:.7f}\n'
f'Средний квадрат ошибки (MSE) IDW = {best_mse_idw_tr_shrunk:.7f}\n'
f'Средняя квадратическая ошибка (RMSE) IDW = {best_rmse_idw_tr_shrunk:.7f}\n'
f'Коэффициент детерминации (R2) IDW = {best_r2_idw_tr_shrunk:.7f}'
```

НОВЫЙ ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:

Elapsed time=00:00:00

Текущие значения: iterations=1, power=2.0,  
Максимальная ошибка (MAX\_E) IDW = 2.4819887  
Средняя абсолютная ошибка (MAE) IDW = 0.7282770  
Средний квадрат ошибки (MSE) IDW = 0.9846784  
Средняя квадратическая ошибка (RMSE) IDW = 0.9923096  
Коэффициент детерминации (R2) IDW = 0.9927749

Elapsed time=00:00:00

Текущие значения: iterations=2, power=2.25,  
Максимальная ошибка (MAX\_E) IDW = 2.4643692  
Средняя абсолютная ошибка (MAE) IDW = 0.7284702  
Средний квадрат ошибки (MSE) IDW = 0.9747929  
Средняя квадратическая ошибка (RMSE) IDW = 0.9873160  
Коэффициент детерминации (R2) IDW = 0.9928475

Elapsed time=00:00:00

Текущие значения: iterations=3, power=2.5,  
Максимальная ошибка (MAX\_E) IDW = 2.4479829  
Средняя абсолютная ошибка (MAE) IDW = 0.7284614  
Средний квадрат ошибки (MSE) IDW = 0.9671532  
Средняя квадратическая ошибка (RMSE) IDW = 0.9834395  
Коэффициент детерминации (R2) IDW = 0.9929035

Elapsed time=00:00:00

Текущие значения: iterations=4, power=2.75,  
Максимальная ошибка (MAX\_E) IDW = 2.4329814  
Средняя абсолютная ошибка (MAE) IDW = 0.7282718  
Средний квадрат ошибки (MSE) IDW = 0.9614267  
Средняя квадратическая ошибка (RMSE) IDW = 0.9805237  
Коэффициент детерминации (R2) IDW = 0.9929455

Elapsed time=00:00:00

Текущие значения: iterations=5, power=3.0,  
Максимальная ошибка (MAX\_E) IDW = 2.4194576  
Средняя абсолютная ошибка (MAE) IDW = 0.7280688  
Средний квадрат ошибки (MSE) IDW = 0.9572872  
Средняя квадратическая ошибка (RMSE) IDW = 0.9784106  
Коэффициент детерминации (R2) IDW = 0.9929759

Elapsed time=00:00:00

Текущие значения: iterations=6, power=3.25,

Максимальная ошибка (MAX\_E) IDW = 2.4074490  
Средняя абсолютная ошибка (MAE) IDW = 0.7280527  
Средний квадрат ошибки (MSE) IDW = 0.9544332  
Средняя квадратическая ошибка (RMSE) IDW = 0.9769510  
Коэффициент детерминации (R2) IDW = 0.9929969

Elapsed time=00:00:01  
Текущие значения: iterations=7, power=3.5,  
Максимальная ошибка (MAX\_E) IDW = 2.3969460  
Средняя абсолютная ошибка (MAE) IDW = 0.7283720  
Средний квадрат ошибки (MSE) IDW = 0.9525983  
Средняя квадратическая ошибка (RMSE) IDW = 0.9760114  
Коэффициент детерминации (R2) IDW = 0.9930103

Elapsed time=00:00:01  
Текущие значения: iterations=8, power=3.75,  
Максимальная ошибка (MAX\_E) IDW = 2.3879004  
Средняя абсолютная ошибка (MAE) IDW = 0.7311115  
Средний квадрат ошибки (MSE) IDW = 0.9515556  
Средняя квадратическая ошибка (RMSE) IDW = 0.9754771  
Коэффициент детерминации (R2) IDW = 0.9930180

Elapsed time=00:00:01  
Текущие значения: iterations=9, power=4.0,  
Максимальная ошибка (MAX\_E) IDW = 2.3802347  
Средняя абсолютная ошибка (MAE) IDW = 0.7340185  
Средний квадрат ошибки (MSE) IDW = 0.9511173  
Средняя квадратическая ошибка (RMSE) IDW = 0.9752524  
Коэффициент детерминации (R2) IDW = 0.9930212

Elapsed time=00:00:01  
Текущие значения: iterations=10, power=4.25,  
Максимальная ошибка (MAX\_E) IDW = 2.3738513  
Средняя абсолютная ошибка (MAE) IDW = 0.7368032  
Средний квадрат ошибки (MSE) IDW = 0.9511319  
Средняя квадратическая ошибка (RMSE) IDW = 0.9752599  
Коэффициент детерминации (R2) IDW = 0.9930211

ЛУЧШИЕ значения: power=4.0,  
Максимальная ошибка (MAX\_E) IDW = 2.3802347  
Средняя абсолютная ошибка (MAE) IDW = 0.7340185  
Средний квадрат ошибки (MSE) IDW = 0.9511173  
Средняя квадратическая ошибка (RMSE) IDW = 0.9752524  
Коэффициент детерминации (R2) IDW = 0.9930212

Опробуем новое значение лучшей степени для IDW на сокращённом валидационном датасете

In [213...]

```
# Создаём y_predict_idw_vld_shrunk по индексам в x_test_shrunk
# используем функцию inverse_distance_avg
# Проходим по массиву и считываем из df_test данные по координатам, выводим результат списком

y_predict_idw_vld_shrunk = [
    inverse_distance_avg(row=df_test.iloc[x],
                          param=PARAMETER33,
                          station=df_test.keys()[y][len(PARAMETER33)+1:],
                          df_dists=df_station_dists,
                          power=best_power_idw_tr_shrunk) # берём лучшее значение степени, полученное из кода выше
    for x, y in x_test_idw_shrunk
]
# y_predict_idw_vld_shrunk

max_e_idw_vld_shrunk = max_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
mae_idw_vld_shrunk = mean_absolute_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
mse_idw_vld_shrunk = mean_squared_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
rmse_idw_vld_shrunk = mean_squared_error(y_test_idw_shrunk, y_predict_idw_vld_shrunk, squared=False)
r2_idw_vld_shrunk = r2_score(y_test_idw_shrunk, y_predict_idw_vld_shrunk)
print(f'ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld_shrunk:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld_shrunk:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld_shrunk:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld_shrunk:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld_shrunk:.7f}'
      )
```

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, IDW:

Максимальная ошибка (MAX\_E) IDW = 2.7257166  
Средняя абсолютная ошибка (MAE) IDW = 0.7183273  
Средний квадрат ошибки (MSE) IDW = 0.8652198  
Средняя квадратическая ошибка (RMSE) IDW = 0.9301719  
Коэффициент детерминации (R2) IDW = 0.9930559

### Метрики обучающего и валидационного датасетов для совместной работы двух моделей

Данные работы моделей на своей части обучающего датасета у нас есть. Подсчитаем метрики для общего датасета

In [214...]

```
# Восстановим y_predict для лучшего R2 IDW
y_predict_idw_tr_shrunk = [
```

```
        inverse_distance_avg(row_=df_test.iloc[x],
                             param_=PARAMETER33,
                             station_=df_test.keys()[-1][len(PARAMETER33)+1:],
                             df_dists_=df_station_dists,
                             power_=best_power_idw_tr_shrunk)
    for x, y in x_train_idw_shrunk
]
```

In [215...]

```
# последовательно соединим датасеты для кригинга (там, где есть предсказания) и для IDW
x_train_2 = np.append(x_train[~np.isnan(y_predict_kriging_tr)], x_train_idw_shrunk)
y_train_2 = np.append(y_train_kriging_shrunk, y_train_idw_shrunk)
y_predict_tr_2 = np.append(y_predict_kriging_tr_shrunk, y_predict_idw_tr_shrunk)
x_test_2 = np.append(x_test[~np.isnan(y_predict_kriging_vld)], x_test_idw_shrunk)
y_test_2 = np.append(y_test_kriging_shrunk, y_test_idw_shrunk)
y_predict_vld_2 = np.append(y_predict_kriging_vld_shrunk, y_predict_idw_vld_shrunk)
```

In [216...]

```
# Расчитываем метрики качества
max_e_2_tr = max_error(y_train_2, y_predict_tr_2)
mae_2_tr = mean_absolute_error(y_train_2, y_predict_tr_2)
mse_2_tr = mean_squared_error(y_train_2, y_predict_tr_2)
rmse_2_tr = mean_squared_error(y_train_2, y_predict_tr_2, squared=False)
r2_2_tr = r2_score(y_train_2, y_predict_tr_2)

max_e_2_vld = max_error(y_test_2, y_predict_vld_2)
mae_2_vld = mean_absolute_error(y_test_2, y_predict_vld_2)
mse_2_vld = mean_squared_error(y_test_2, y_predict_vld_2)
rmse_2_vld = mean_squared_error(y_test_2, y_predict_vld_2, squared=False)
r2_2_vld = r2_score(y_test_2, y_predict_vld_2)
```

In [217...]

```
print(f'ОБЩИЙ ОБУЧАЮЩИЙ ДАТАСЕТ, Кригинг + IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_2_tr:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_2_tr:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_2_tr:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_2_tr:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_2_tr:.7f}\n'
      )
print(f'ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:\n'
      f'Максимальная ошибка (MAX_E) IDW = {max_e_2_vld:.7f}\n'
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_2_vld:.7f}\n'
      f'Средний квадрат ошибки (MSE) IDW = {mse_2_vld:.7f}\n'
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_2_vld:.7f}\n'
      f'Коэффициент детерминации (R2) IDW = {r2_2_vld:.7f}\n'
```

```
)  
  
print(f'Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:\n'  
      f'Максимальная ошибка (MAX_E) IDW = {max_e_idw_vld:.7f}\n'  
      f'Средняя абсолютная ошибка (MAE) IDW = {mae_idw_vld:.7f}\n'  
      f'Средний квадрат ошибки (MSE) IDW = {mse_idw_vld:.7f}\n'  
      f'Средняя квадратическая ошибка (RMSE) IDW = {rmse_idw_vld:.7f}\n'  
      f'Коэффициент детерминации (R2) IDW = {r2_idw_vld:.7f}'  
)
```

ОБЩИЙ ОБУЧАЮЩИЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX\_E) IDW = 5.6971170  
Средняя абсолютная ошибка (MAE) IDW = 0.6623156  
Средний квадрат ошибки (MSE) IDW = 0.8663163  
Средняя квадратическая ошибка (RMSE) IDW = 0.9307611  
Коэффициент детерминации (R2) IDW = 0.9939099

ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX\_E) IDW = 7.2695578  
Средняя абсолютная ошибка (MAE) IDW = 0.6645235  
Средний квадрат ошибки (MSE) IDW = 0.8807287  
Средняя квадратическая ошибка (RMSE) IDW = 0.9384715  
Коэффициент детерминации (R2) IDW = 0.9937062

Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:

Максимальная ошибка (MAX\_E) IDW = 5.4204585  
Средняя абсолютная ошибка (MAE) IDW = 0.6852596  
Средний квадрат ошибки (MSE) IDW = 0.9087468  
Средняя квадратическая ошибка (RMSE) IDW = 0.9532821  
Коэффициент детерминации (R2) IDW = 0.9935060

## ВЫВОД

Там, где с помощью кригинга удаётся получить предсказания, он превосходит по метрикам качества модель IDW. При этом сочетание этих двух моделей (применение IDW там, где кригинг оказывается бессильным) даёт в целом лучшее качество предсказания, чем только модель IDW. Поэтому, ниже применим комбинацию этих двух моделей к уже реальному датасету..

### 3.3.4. Применение выбранных модели для исправления, восстановления данных и получения предиктов показателя максимальной температуры T\_max для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

In [218...]

```
# Добавим в df_tmp33 столбцы для будущих значений для Чашниково и центральной точки, заполним их NaN
# - это необходимо, чтобы вычислить значения для архивов
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER33#Chashnikovo и PARAMETER33#Rfrnce_point
df_tmp33 = (df_tmp33.
             assign(col_name1 = np.nan,
                   col_name2 = np.nan).
             rename(columns={"col_name1": PARAMETER33+"#"+'Chashnikovo',
                            "col_name2": PARAMETER33+"#"+'Rfrnce_point'}))
         )

df_tmp33.sample(3, random_state=56)
```

Out[218]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2014-02-22 03:00:00	-2.3	-1.8	-2.3	-3.0	-3.9	-2.9	-2.2	-1.9	
2015-05-16 03:00:00	13.8	11.4	10.8	10.6	8.8	11.2	8.6	8.4	
2020-06-18 18:00:00	29.3	28.9	24.6	28.3	28.7	28.4	30.7	30.4	

In [219...]

```
# Добавим в архив параметров T_max столбцы для будущих значений Чашниково и центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER33#Chashnikovo и PARAMETER33#Rfrnce_point
dict_df_parameters['df_'+PARAMETER33] = (dict_df_parameters['df_'+PARAMETER33].
                                         assign(col_name1 = np.nan,
                                               col_name2 = np.nan).
                                         rename(columns={"col_name1": PARAMETER33+"#"+'Chashnikovo',
                                                        "col_name2": PARAMETER33+"#"+'Rfrnce_point'}))
                                         )

dict_df_parameters['df_'+PARAMETER33].sample(3, random_state=56)
```

Out[219]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
<b>2014-02-22 03:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2015-05-16 03:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2020-06-18 18:00:00</b>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [220...]

```
# # Восстановление архивов Чашниково и центральной точки:
# dict_df_locations['df_Chashnikovo'] = pd.read_csv(
#     new_raw_path1 + '/' + 'df_Chashnikovo' + PARAMETER32 + '.csv',
#     index_col=0, parse_dates=True, infer_datetime_format = True, low_memory=False)
# print('df_Chashnikovo_T.csv -> O.K.')

# dict_df_locations['df_Rfrnce_point'] = pd.read_csv(
#     new_raw_path1 + '/' + 'df_Rfrnce_point' + PARAMETER32 + '.csv',
#     index_col=0, parse_dates=True, infer_datetime_format = True, low_memory=False)
# print('df_Rfrnce_point_T.csv -> O.K.')
```

In [221...]

```
# В архивах метеостанций df Чашниково и df для центральной точки
# создадим столбцы с называнием PARAMETER33

dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER33})
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER33})
                                         )

dict_df_locations['df_Chashnikovo'].sample(3, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(3, random_state=56)
```

Out[221]:

	T	T_min	T_max
2014-02-22 03:00:00	-4.156558	-4.156558	NaN
2015-05-16 03:00:00	9.181730	9.181730	NaN
2020-06-18 18:00:00	27.439052	25.366130	NaN

Out[221]:

	T	T_min	T_max
2014-02-22 03:00:00	-3.589183	-3.794602	NaN
2015-05-16 03:00:00	7.789650	7.789650	NaN
2020-06-18 18:00:00	29.404954	25.572651	NaN

**Заполнение значений максимальной температуры T\_max для Чашниково и условной средней точки скользящими минимумами (аналогично заполнению строк сплошных NaN)**

In [222...]

```
# построим датафрейм с максимальными значениями температуры по моментам наблюдения за предшествующие 12 часов
# для условных метеостанций
# Используем скользящее окно с вычислением максимума и со сдвигом вверх на 4 строки (-4),
# чтобы обозначить минимумы за предшествующие периоды.
df_t1a = df_tmp31.iloc[:, -2:].rolling(window='12H', center=False, closed='left').max().shift(-4)
df_t1a.sample(7, random_state=56)
```

Out[222]:

	T#Chashnikovo	T#Rfrnce_point
2014-02-22 03:00:00	-2.109643	-2.700734
2015-05-16 03:00:00	10.434571	8.797603
2020-06-18 18:00:00	30.241632	29.941094
2019-12-02 21:00:00	-2.862178	-2.655566
2008-06-05 18:00:00	16.565949	17.613615
2016-10-20 06:00:00	0.916309	0.885072
2006-09-11 03:00:00	11.886722	11.628949

In [223]:

```
# Выберем значения из df_t1a, соответствующие индексам строк сплошных NaN в df_tmp33 и присвоим их ячейкам df_tmp33
# используем .loc, чтобы избежать предупреждения о присовении значений для копии данных.
df_tmp33.loc[:, PARAMETER33+'#Chashnikovo'][mask_total_nans] = \
df_t1a.loc[:, PARAMETER31+'#Chashnikovo'][df_t1a.index.isin(df_tmp33[mask_total_nans].index)]

df_tmp33.loc[:, PARAMETER33+'#Rfrnce_point'][mask_total_nans] = \
df_t1a.loc[:, PARAMETER31+'#Rfrnce_point'][df_t1a.index.isin(df_tmp33[mask_total_nans].index)]

df_tmp33.sample(7, random_state=56)
```

Out[223]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2014-02-22 03:00:00	-2.300000	-1.8	-2.300000	-3.0	-3.9	-2.9	-2.2	-1.9	
2015-05-16 03:00:00	13.800000	11.4	10.800000	10.6	8.8	11.2	8.6	8.4	
2020-06-18 18:00:00	29.300000	28.9	24.600000	28.3	28.7	28.4	30.7	30.4	
2019-12-02 21:00:00	-1.800000	-2.8	-2.900000	-2.2	-2.3	-2.8	-2.7	-2.8	
2008-06-05 18:00:00	18.600000	18.5	16.961854	19.2	17.2	16.2	17.1	17.0	
2016-10-20 06:00:00	1.100000	1.3	1.000000	0.9	1.1	1.3	0.8	0.2	
2006-09-11 03:00:00	11.966017	12.0	11.801339	12.1	11.5	12.0	11.5	11.8	



Перенесение уже исправленных сплошных NaN в архивы

In [224...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER33] = df_tmp33.copy(deep=True)

# Перенесём уже исправленные значения в dict_df_locations, оставшиеся NaN исправим ниже
for name_df in dict_df_locations.keys():
    dict_df_locations[name_df].loc[:, PARAMETER33] = df_tmp33.loc[:, PARAMETER33 + '#' + name_df[3:]]

dict_df_parameters['df_'+PARAMETER33].sample(7, random_state=56)
dict_df_locations['df_Chashnikovo'].sample(7, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(7, random_state=56)
```

Out[224]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
<b>2014-02-22 03:00:00</b>	-2.300000	-1.8	-2.300000	-3.0	-3.9	-2.9	-2.2	-1.9	
<b>2015-05-16 03:00:00</b>	13.800000	11.4	10.800000	10.6	8.8	11.2	8.6	8.4	
<b>2020-06-18 18:00:00</b>	29.300000	28.9	24.600000	28.3	28.7	28.4	30.7	30.4	
<b>2019-12-02 21:00:00</b>	-1.800000	-2.8	-2.900000	-2.2	-2.3	-2.8	-2.7	-2.8	
<b>2008-06-05 18:00:00</b>	18.600000	18.5	16.961854	19.2	17.2	16.2	17.1	17.0	
<b>2016-10-20 06:00:00</b>	1.100000	1.3	1.000000	0.9	1.1	1.3	0.8	0.2	
<b>2006-09-11 03:00:00</b>	11.966017	12.0	11.801339	12.1	11.5	12.0	11.5	11.8	

Out[224]:

	T	T_min	T_max
<b>2014-02-22 03:00:00</b>	-4.156558	-4.156558	-2.109643
<b>2015-05-16 03:00:00</b>	9.181730	9.181730	10.434571
<b>2020-06-18 18:00:00</b>	27.439052	25.366130	30.241632
<b>2019-12-02 21:00:00</b>	-3.004683	-4.229056	NaN
<b>2008-06-05 18:00:00</b>	16.565949	12.255838	16.565949
<b>2016-10-20 06:00:00</b>	0.117681	0.074269	0.916309
<b>2006-09-11 03:00:00</b>	9.855422	9.855422	11.886722

Out[224]:

	T	T_min	T_max
<b>2014-02-22 03:00:00</b>	-3.589183	-3.794602	-2.700734
<b>2015-05-16 03:00:00</b>	7.789650	7.789650	8.797603
<b>2020-06-18 18:00:00</b>	29.404954	25.572651	29.941094
<b>2019-12-02 21:00:00</b>	-2.977448	-3.796107	NaN
<b>2008-06-05 18:00:00</b>	17.613615	11.754644	17.613615
<b>2016-10-20 06:00:00</b>	-0.382597	-0.369287	0.885072
<b>2006-09-11 03:00:00</b>	9.181940	9.181940	11.628949

### Частичное заполнение оставшихся NaN расчётными значениями с помощью кригинга

Применим модель кригинга в цикле, пока количество оставшихся в dataфрейме NaN перестанет уменьшаться. Этую будут значения, которые модель уже никак не сможет рассчитать. Их можно будет восстановить методом IDW.

In [225...]

```
start_time = time.time() # для замера времени выполнения кода

# Подсчитаем количество NaN в df_tmp33 (кроме бывших сплошных NaN) и присвоим их переменной old_nan_count
# np.isnan(df_tmp331).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
new_nan_count = np.sum(np.isnan(df_tmp33).sum()) # результат всегда будет >= 0
# Определим начальное значение для переменной для обновления количества оставшихся NaN
```

```

old_nan_count = new_nan_count
counter = 0 # определим счётчик

# заменим значения в архивах, на исправленные и вычисленные из данных в df_tmp331
# используем функцию row_nan_kriging_correct
# функция настроена таким образом, что при возникновении ошибки, связной с недостаточностью данных,
# осуществляется выход из функции без возврата каких бы то ни было значений.
while (old_nan_count != new_nan_count) or (counter == 0):
    counter += 1
    print (f'\nИтерация № {counter}: осталось NaN: {new_nan_count}')

    # Построчно применяем функцию обработки NaN
    df_tmp33.apply(lambda x: row_nan_kriging_correct(row=x,
                                                       name_param_=PARAMETER33
                                                       ),
                  axis=1
                 )
    old_nan_count = new_nan_count # сохраняем прежнее количество NaN в old_nan_count
    new_nan_count = np.sum(np.isnan(df_tmp33).sum()) # подсчитаем оставшиеся количество NaN

```

```

chk_time = time.time()
elapsed_time = chk_time - start_time
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f'Elapsed time={time_formatted}\n')

```

Итерация № 1: осталось NaN: 19919

Out[225]:

2022-06-09 21:00:00	None
2022-06-09 18:00:00	None
2022-06-09 15:00:00	None
2022-06-09 12:00:00	None
2022-06-09 09:00:00	None
...	
2005-02-01 12:00:00	None
2005-02-01 09:00:00	None
2005-02-01 06:00:00	None
2005-02-01 03:00:00	None
2005-02-01 00:00:00	None
Length: 50704, dtype: object	

Итерация № 2: осталось NaN: 1601

```
Out[225]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 3: осталось NaN: 1320  
Out[225]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 4: осталось NaN: 1305  
Out[225]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 5: осталось NaN: 1304
```

```
Out[225]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Elapsed time=00:02:10
```

Подсчитаем конечное количество оставшихся *NaN*, которые не могут быть рассчитаны моделью кригинга

```
In [226]: # np.isnan(df_tmp331).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp33).sum())
```

```
Out[226]: 1304
```

Восстановим оставшиеся значения через модель IDW. Используем для этого степень, полученную на обучающем датасете при совместном использовании модели IDW и модели кригинга.

```
In [227]: # Произведём вычисление отсутствующих значений в df_tmp33 через модель IDW.  
# Заменим соответствующие значения в архивах на вновь вычисленные.  
# используем функцию row_nan_idw_correct  
  
start_time = time.time() # для замера времени выполнения кода  
  
# Построчно применяем функцию обработки NaN  
df_tmp33.apply(lambda x: row_nan_idw_correct(row_=x,  
                                              name_param_=PARAMETER33,  
                                              power_=best_power_idw_tr_shrunk),  
               axis=1  
)  
  
chk_time = time.time()  
elapsed_time = chk_time - start_time
```

```
time_formatted = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
print(f'Elapsed time={time_formatted}\n')
```

Out[227]:

```
2022-06-09 21:00:00      None
2022-06-09 18:00:00      None
2022-06-09 15:00:00      None
2022-06-09 12:00:00      None
2022-06-09 09:00:00      None
...
2005-02-01 12:00:00      None
2005-02-01 09:00:00      None
2005-02-01 06:00:00      None
2005-02-01 03:00:00      None
2005-02-01 00:00:00      None
Length: 50704, dtype: object
Elapsed time=00:00:13
```

Подсчитаем окончательное количество NaN

In [228...]

```
# np.isnan(df_tmp331).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
np.sum(np.isnan(df_tmp33).sum())
```

Out[228]:

```
42
```

Всё корректно. 42 NaN находятся в 3х самых ранних моментах наблюдения. Они пустые.

Выведем, рандомные строки из df\_tmp33

In [229...]

```
df_tmp33.sample(15)
```

Out[229]:

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
<b>2009-03-05 06:00:00</b>	-1.191654	-1.2	-1.4	-0.2	-0.5	-2.800000	-1.0	-1.4	
<b>2013-05-01 06:00:00</b>	11.257862	11.5	9.5	10.6	11.4	11.100000	11.7	11.0	
<b>2016-10-27 06:00:00</b>	-0.300000	-1.0	-0.2	-0.8	0.8	2.300000	-1.0	-0.6	
<b>2020-01-26 06:00:00</b>	-1.000000	-0.6	-1.3	-0.3	-1.2	-0.900000	-0.9	-1.0	
<b>2008-06-01 15:00:00</b>	15.000000	12.8	14.3	14.2	13.1	12.400000	12.8	13.4	
<b>2011-09-09 09:00:00</b>	11.200000	10.3	12.1	11.3	12.1	11.900000	10.1	11.1	
<b>2015-04-16 15:00:00</b>	2.000000	6.0	2.4	1.9	5.4	3.400000	6.5	6.9	
<b>2009-04-10 09:00:00</b>	-1.500000	-1.3	-2.5	-1.0	0.1	-0.901697	-1.2	-0.9	
<b>2018-08-24 00:00:00</b>	19.600000	18.6	19.3	20.4	19.7	18.700000	19.3	20.3	
<b>2019-12-24 09:00:00</b>	7.100000	6.7	5.4	7.0	6.8	6.100000	7.3	7.6	
<b>2013-09-29 03:00:00</b>	3.900000	3.8	4.9	4.8	5.3	4.600000	5.2	4.3	

	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N
2017-07-21 15:00:00	19.100000	18.8	18.1	21.3	20.4	20.500000	20.7	23.1	
2011-03-27 21:00:00	0.000000	0.1	-1.5	0.3	-0.9	-1.100000	0.8	0.3	
2018-04-24 00:00:00	7.300000	8.1	7.0	8.6	7.7	7.300000	7.6	8.1	
2013-03-04 15:00:00	-7.600000	-7.8	-7.9	-6.6	-7.3	-6.800000	-7.2	-6.6	

### 3.3.5. Визуализация графиков температуры T\_max для условной метеостанции Чашниково

In [230...]

```
# Построим список годов для графиков
list_years = df_tmp33.index.year.unique().tolist()
```

In [231...]

```
# Определим часы момента наблюдения для построения графиков
plot_hour1 = 9
plot_hour2 = 21
# Строим график в цикле для каждого года
for year in list_years:
    data1 = dict_df_parameters['df_T_max'][PARAMETER33+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_T_max'].index.year == year) & (dict_df_parameters['df_T_max'].index.hour == plot_hour1)]\n\n    data2 = dict_df_parameters['df_T_max'][PARAMETER33+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_T_max'].index.year == year) & (dict_df_parameters['df_T_max'].index.hour == plot_hour2)]\n\n    fig, ax = plt.subplots(figsize=(10, 4))
    g1 = sns.lineplot(data=data1,
                      color='blue',
                      linewidth=0.5,
                      ax=ax)
    g2 = sns.lineplot(data=data2,
                      color='red',
                      linewidth=0.5,
```

```

    ax=ax)

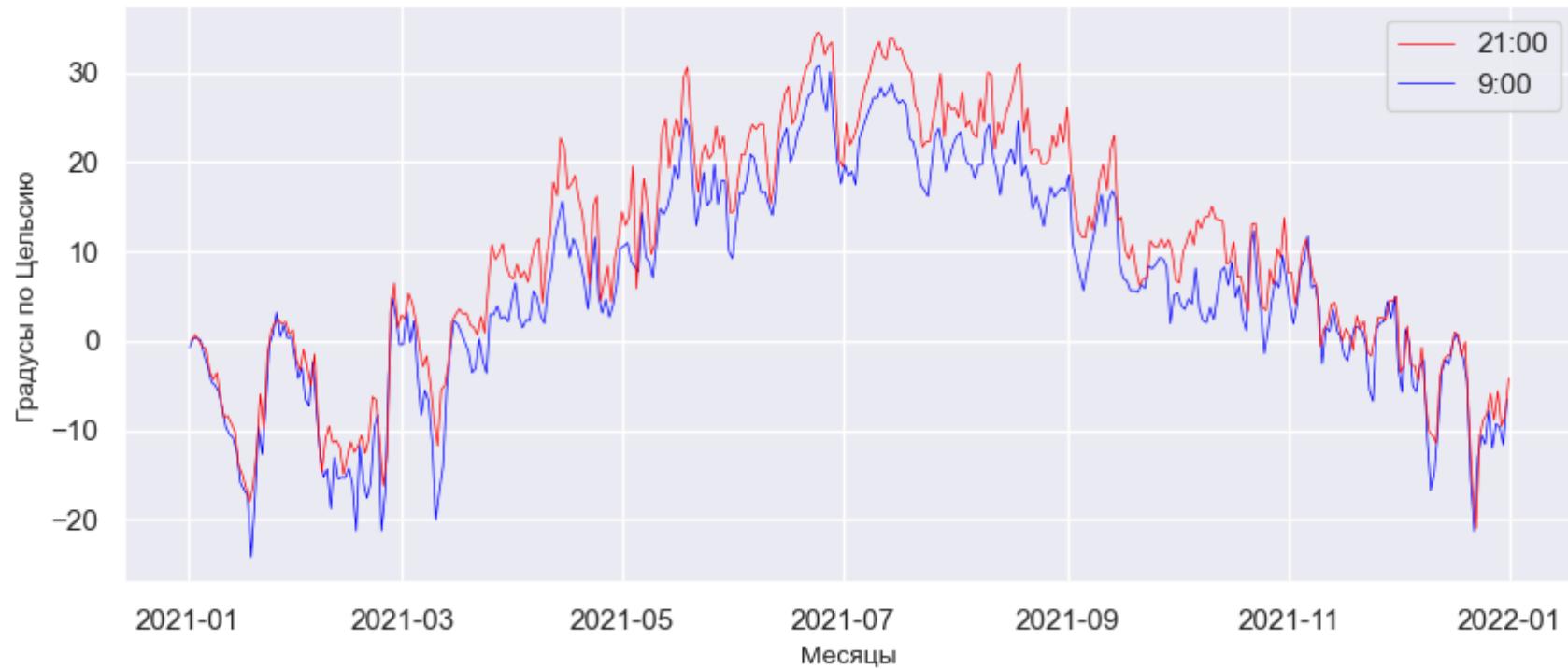
# Добавляем легенду
blue_line = mlines.Line2D([], [], color='blue', label=f'{plot_hour1}:00', linewidth=0.5)
red_line = mlines.Line2D([], [], color='red', label=f'{plot_hour2}:00', linewidth=0.5)
dummy = ax.legend(handles=[red_line, blue_line])

dummy = ax.set_ylabel('Градусы по Цельсию', size=10)
dummy = ax.set_xlabel('Месяцы', size=10)
dummy = plt.title(f'Чашниково: Ежедневная динамика параметра {PARAMETER33} за 12 предшествующих часов '
                  f'на {plot_hour1} и {plot_hour2} часов, {year} год')
plt.show()

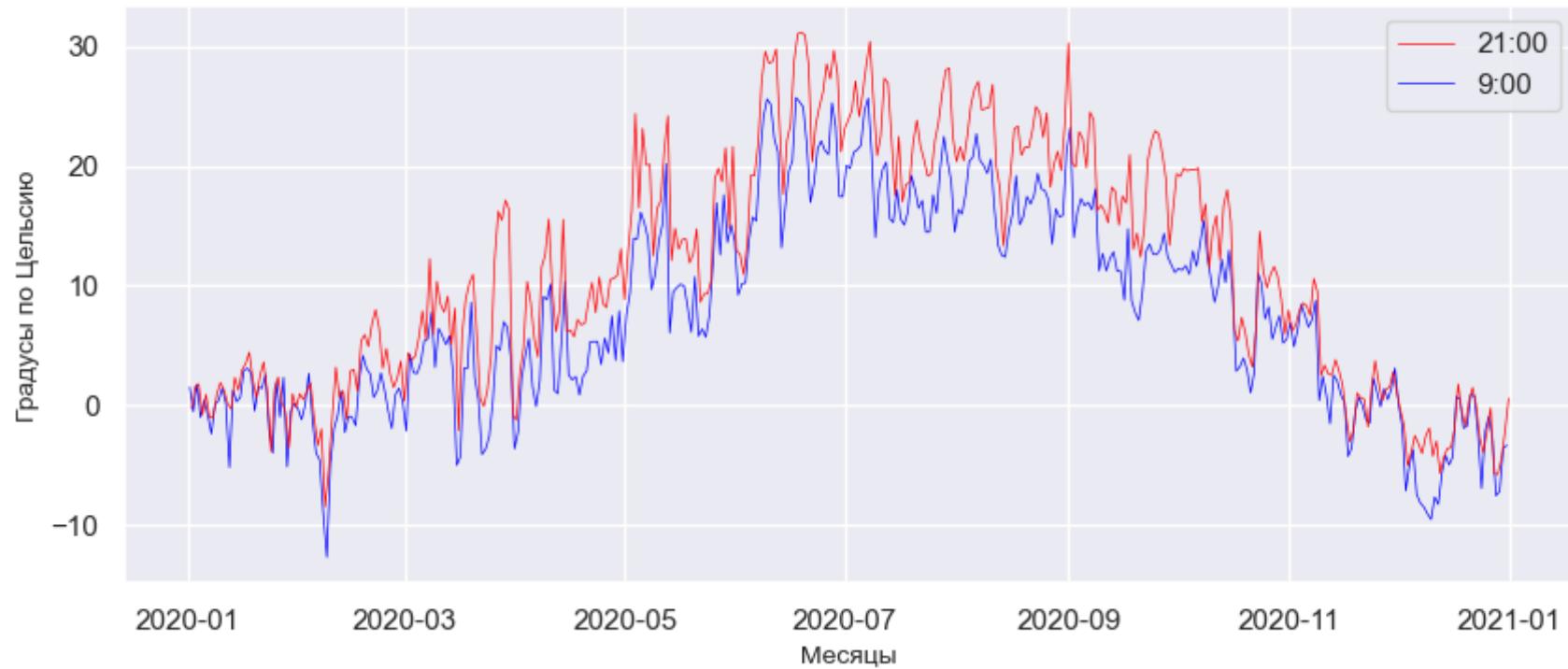
```



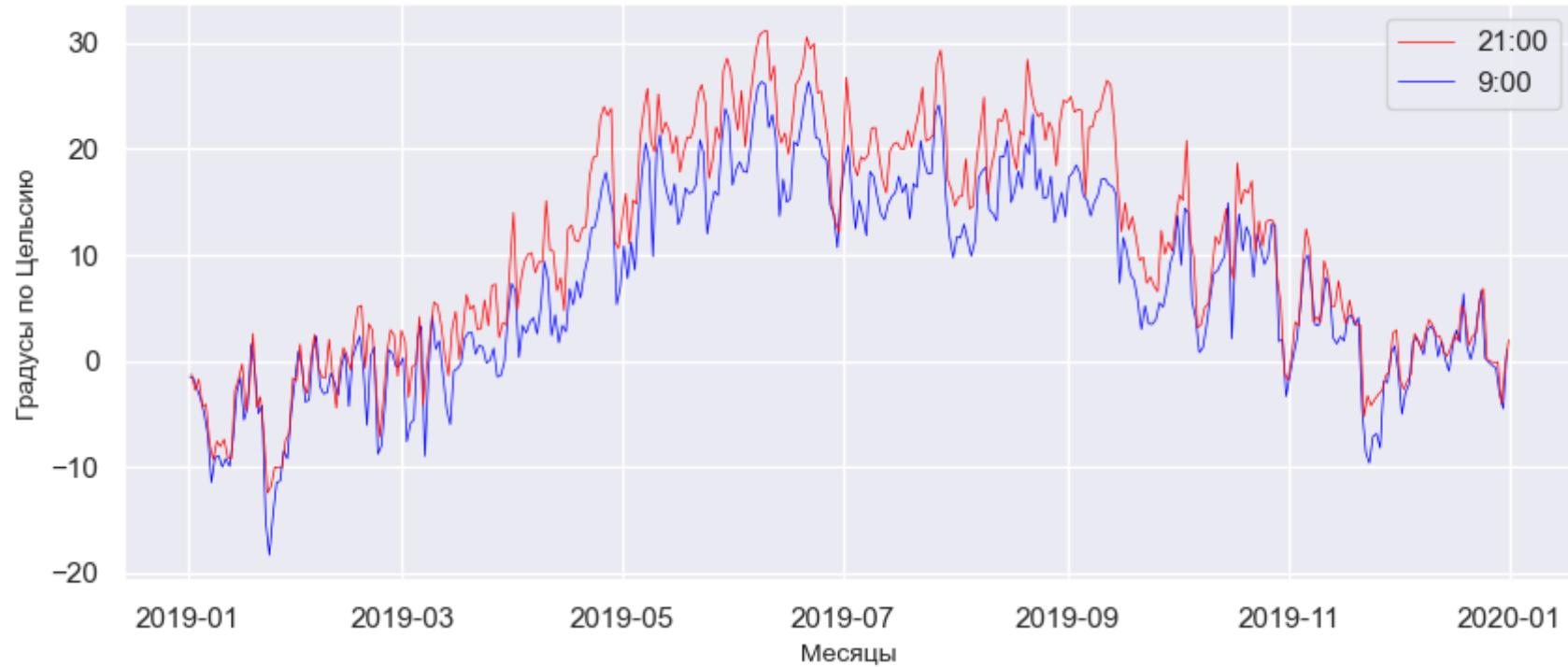
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2021 год



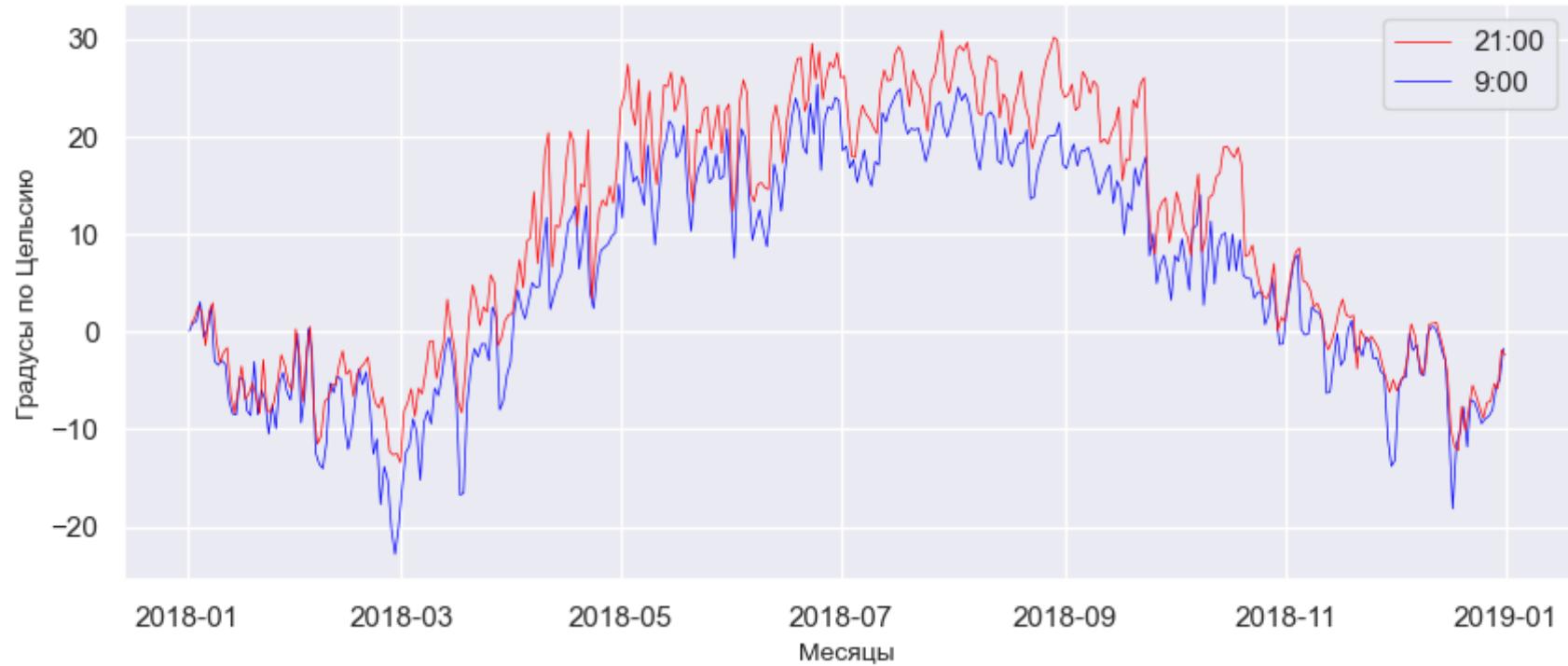
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2020 год



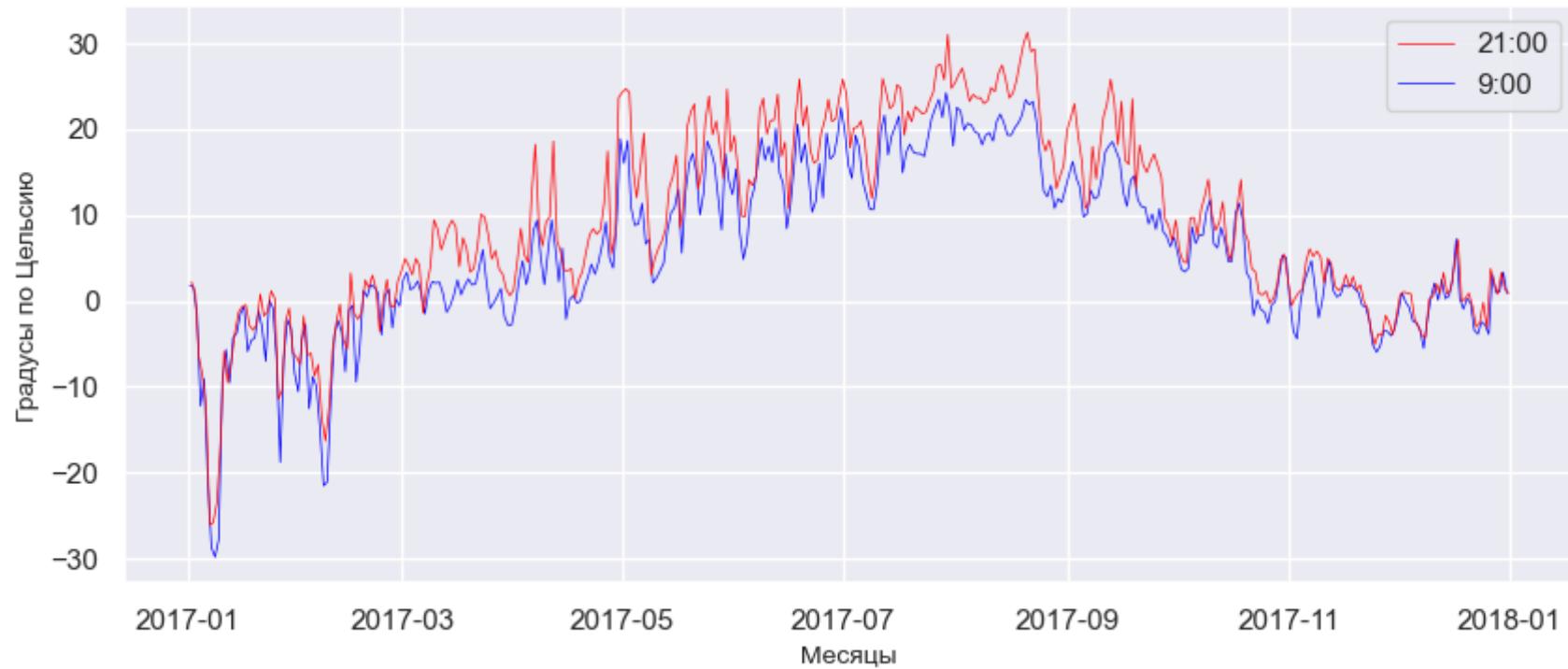
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2019 год



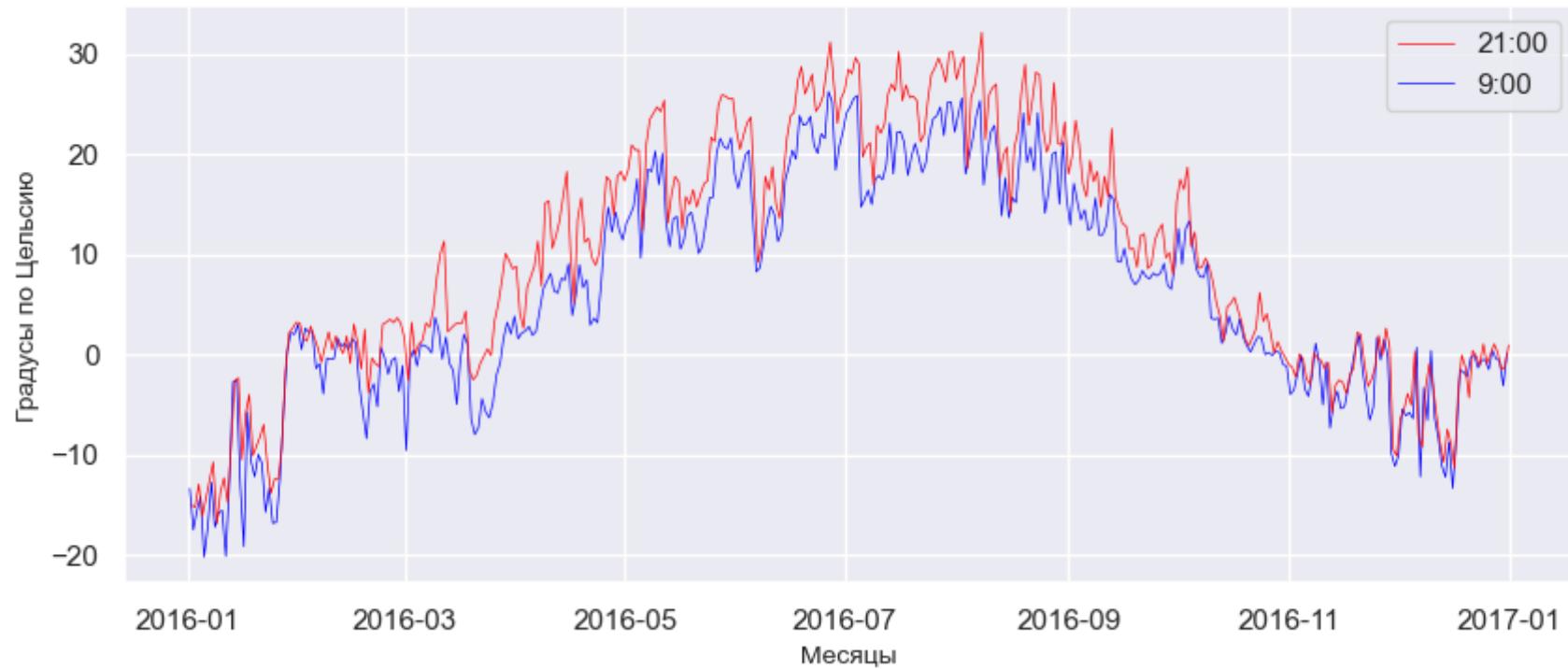
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2018 год



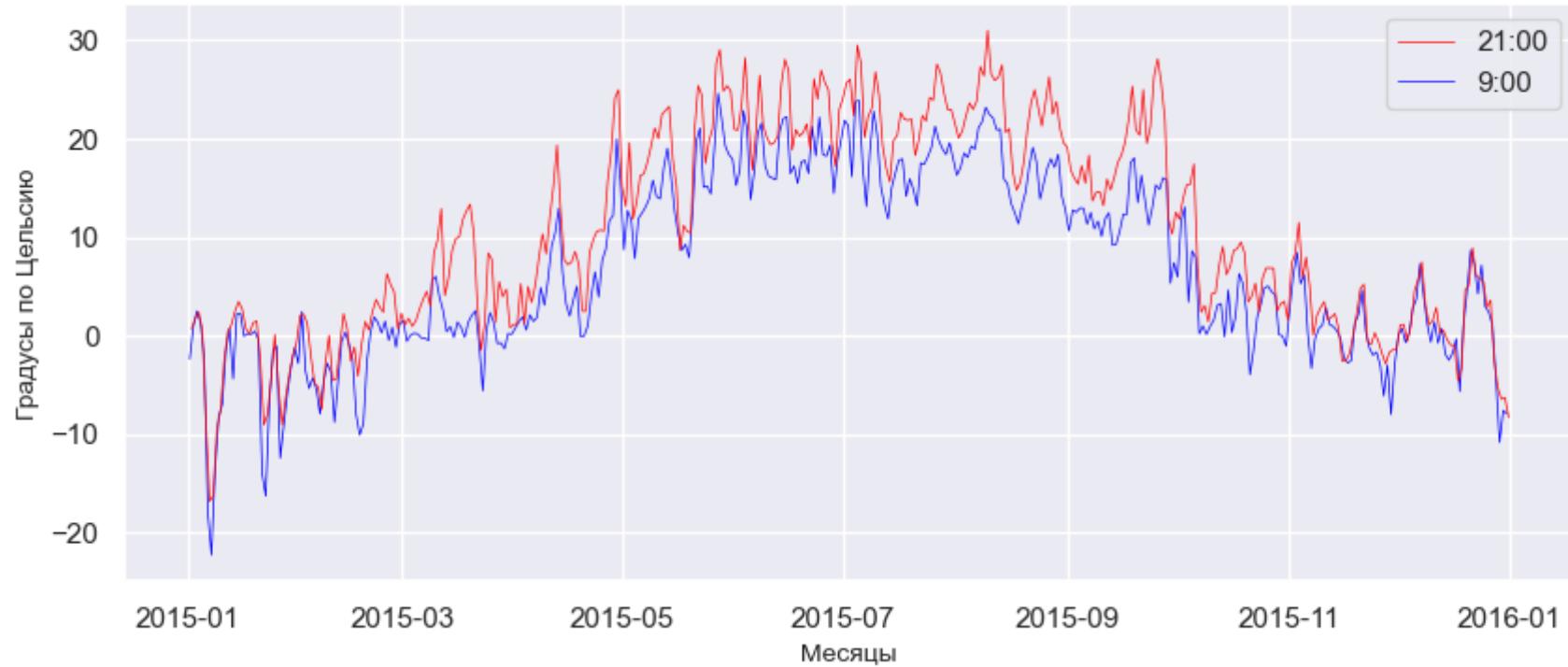
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2017 год



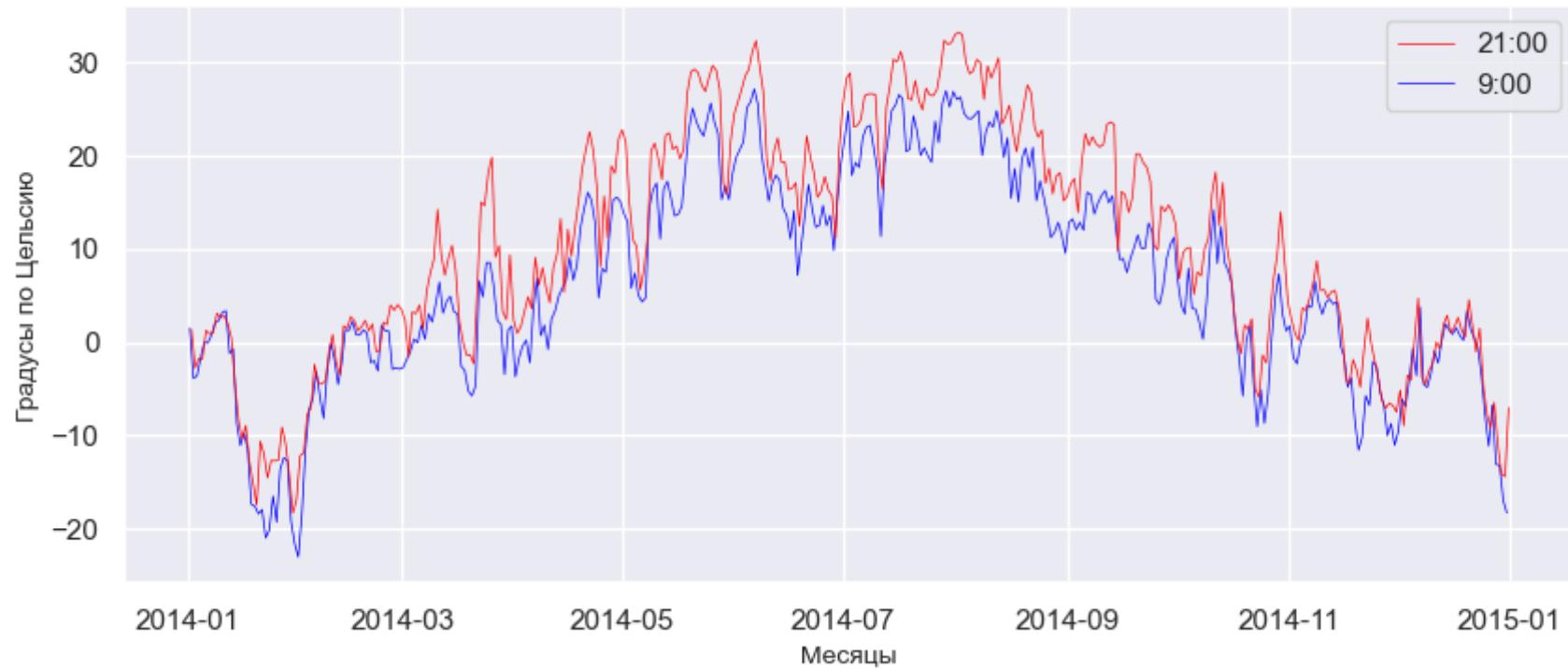
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2016 год



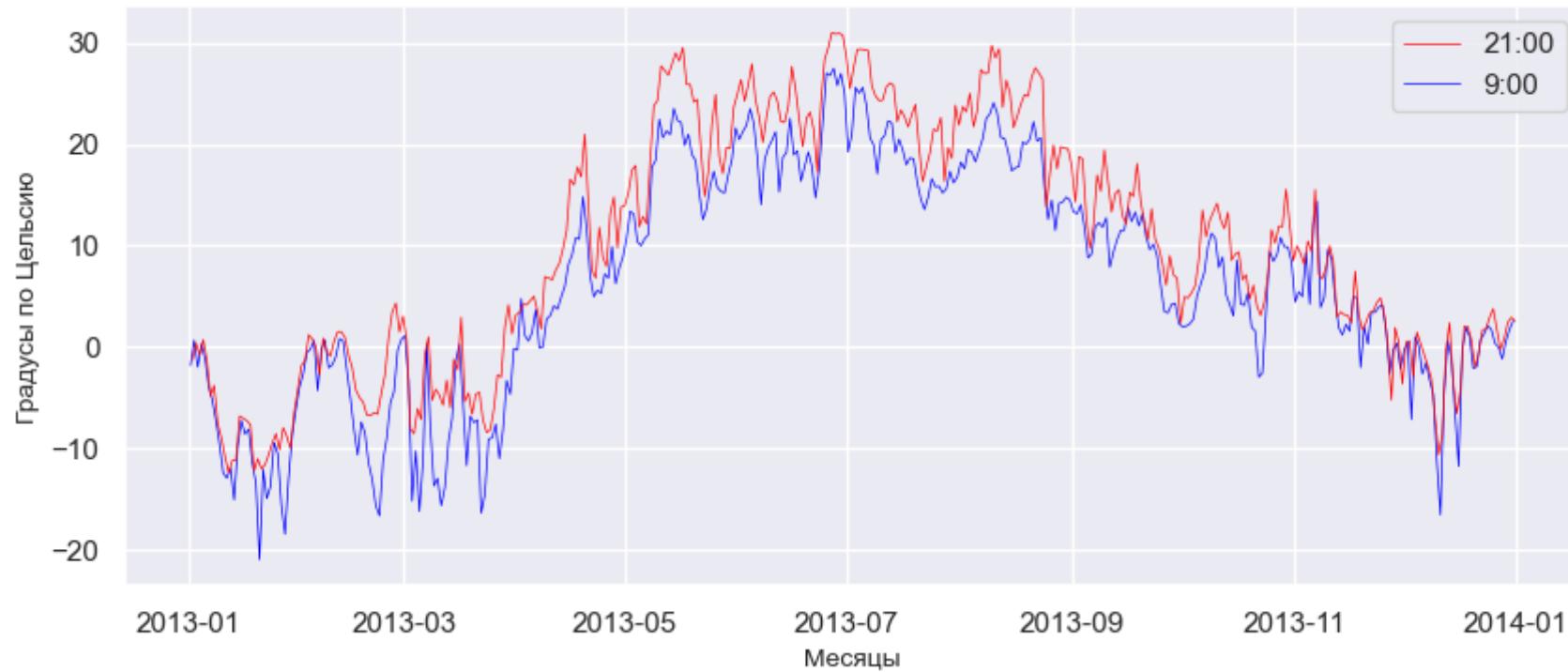
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2015 год



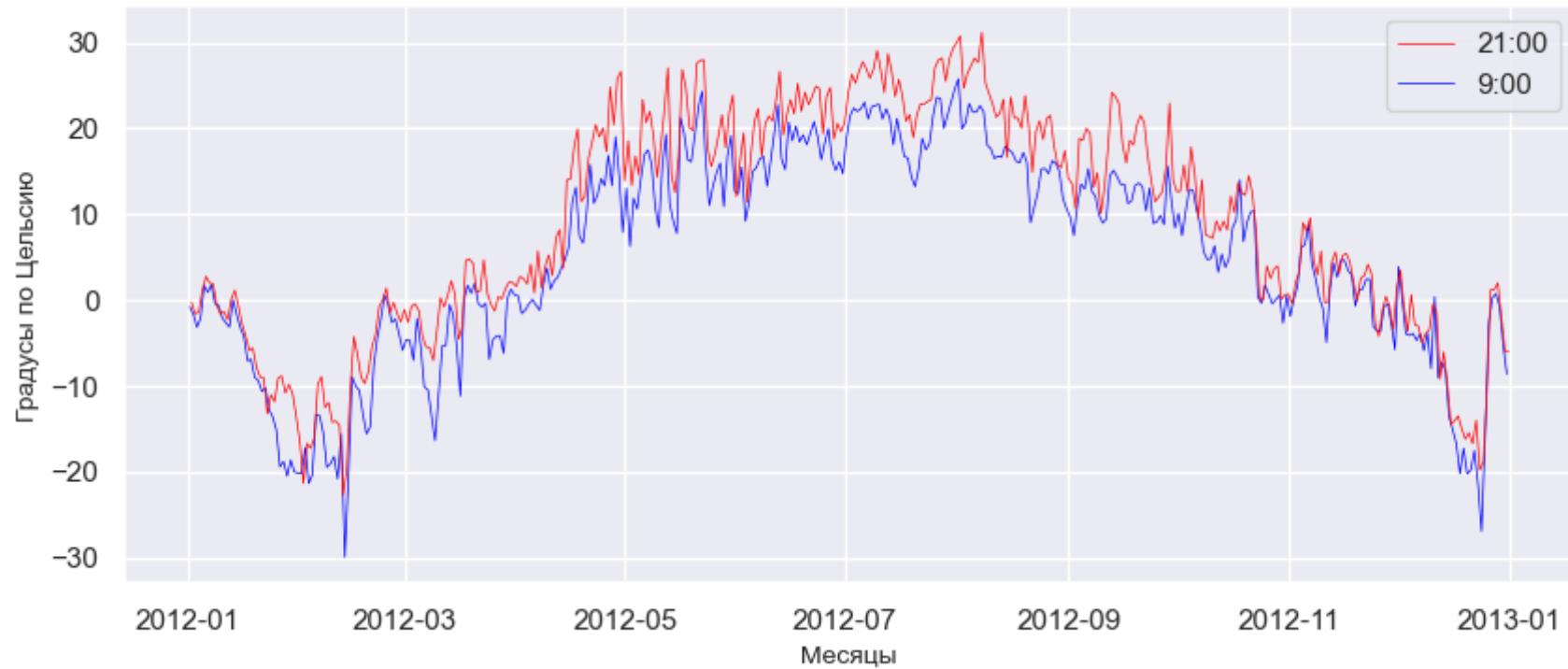
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2014 год



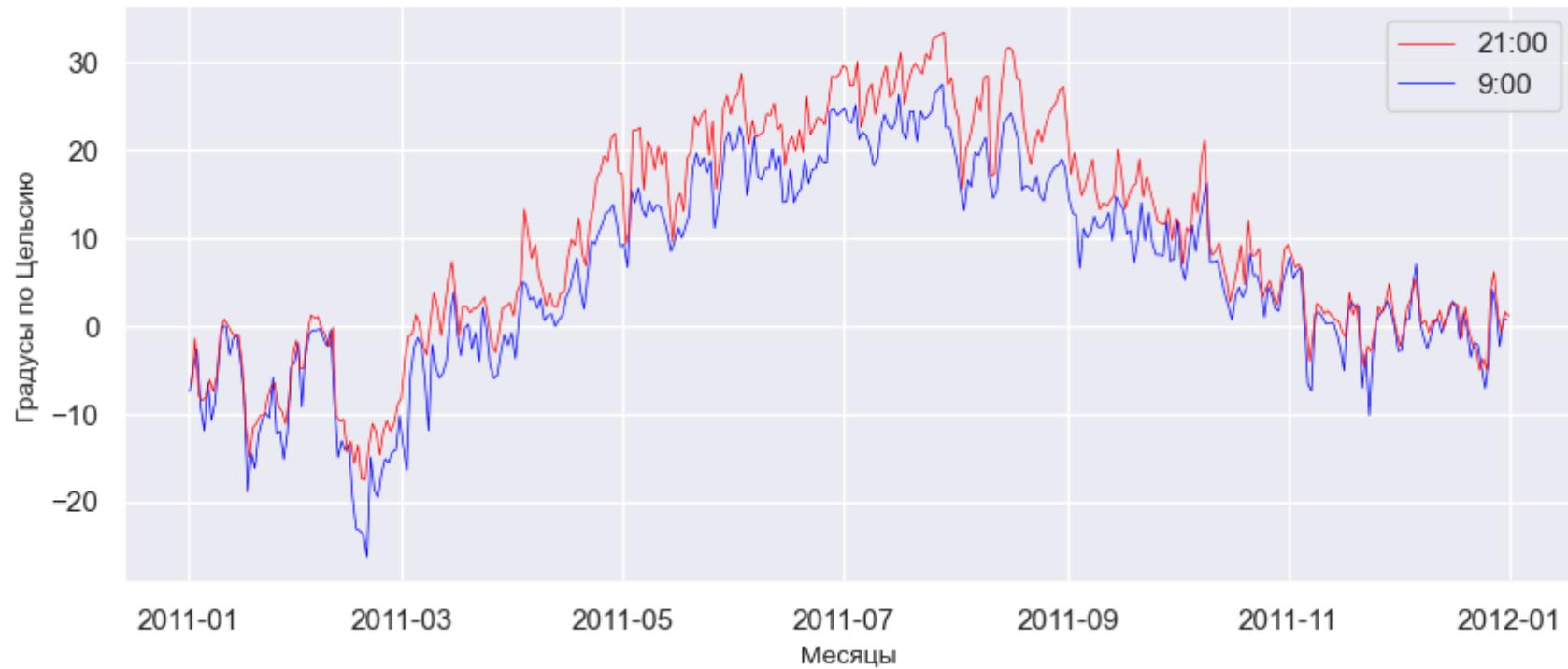
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2013 год



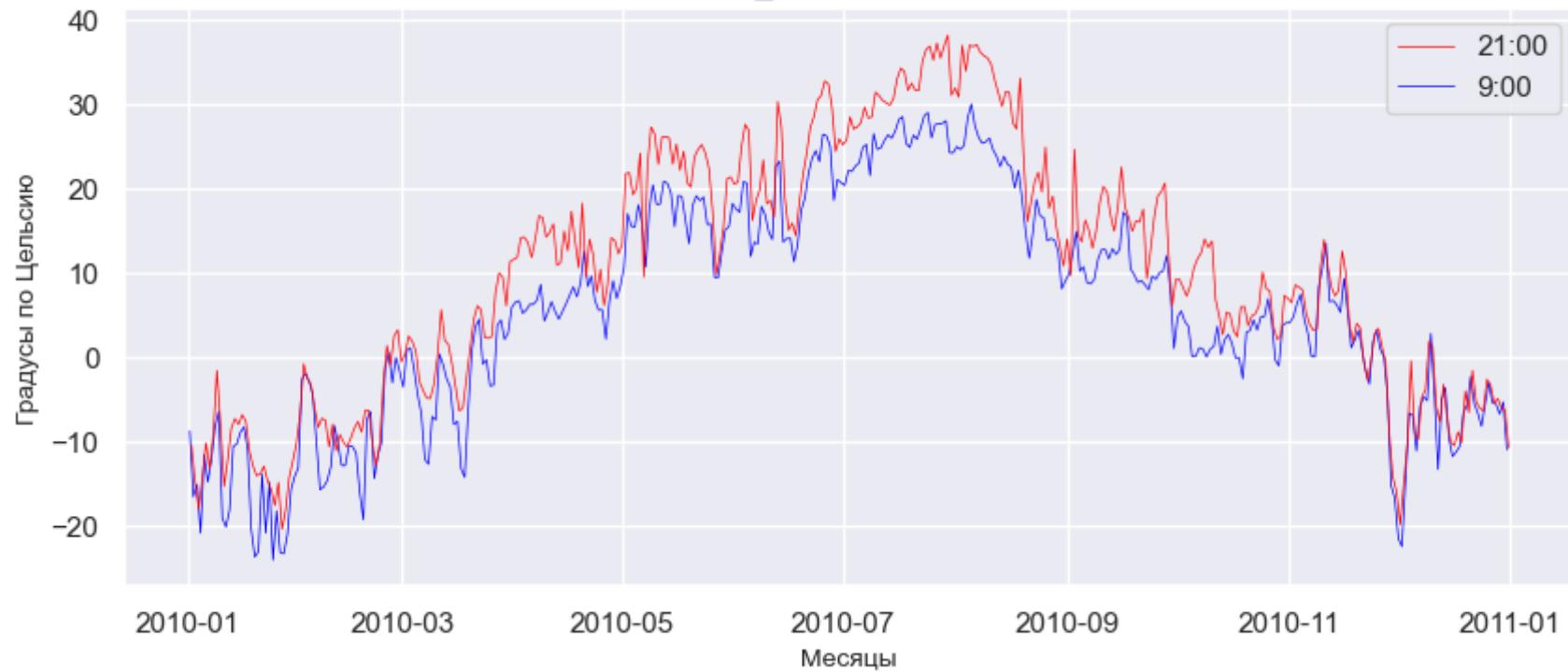
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2012 год



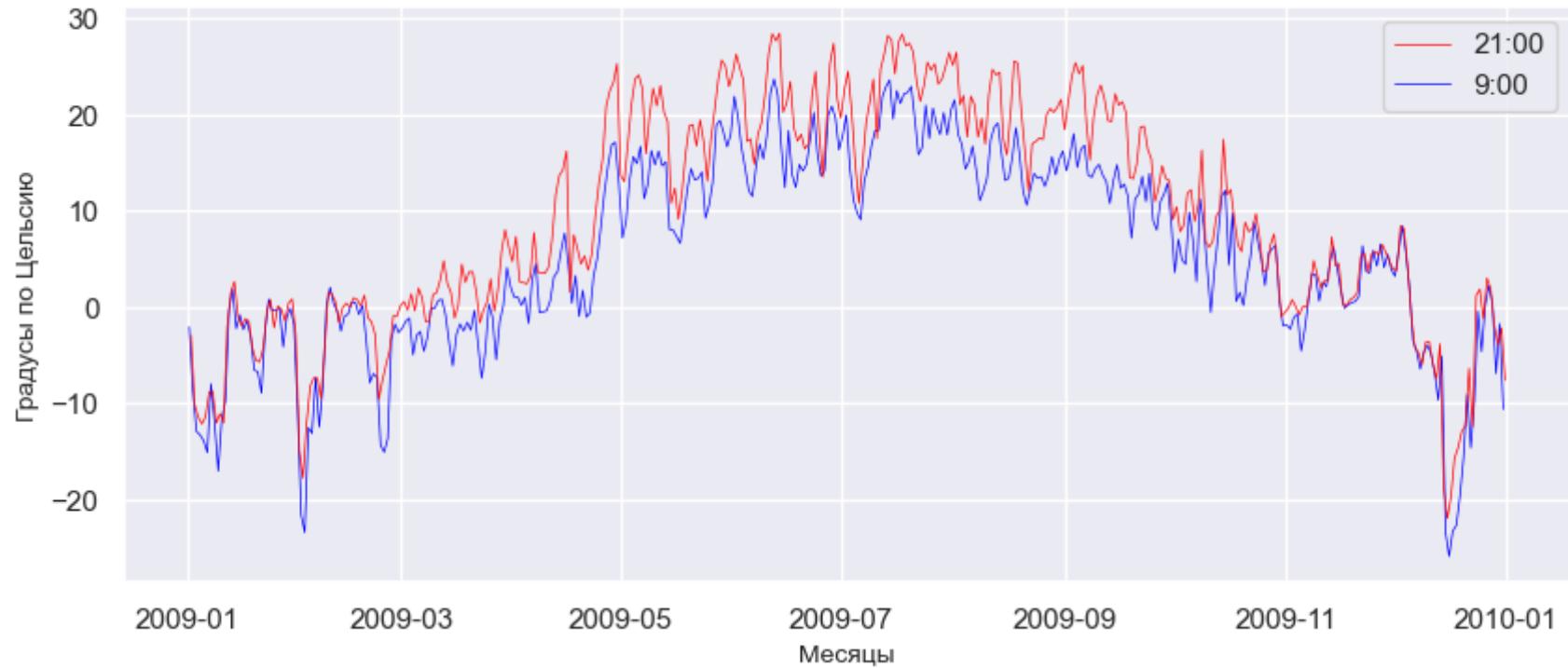
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2011 год



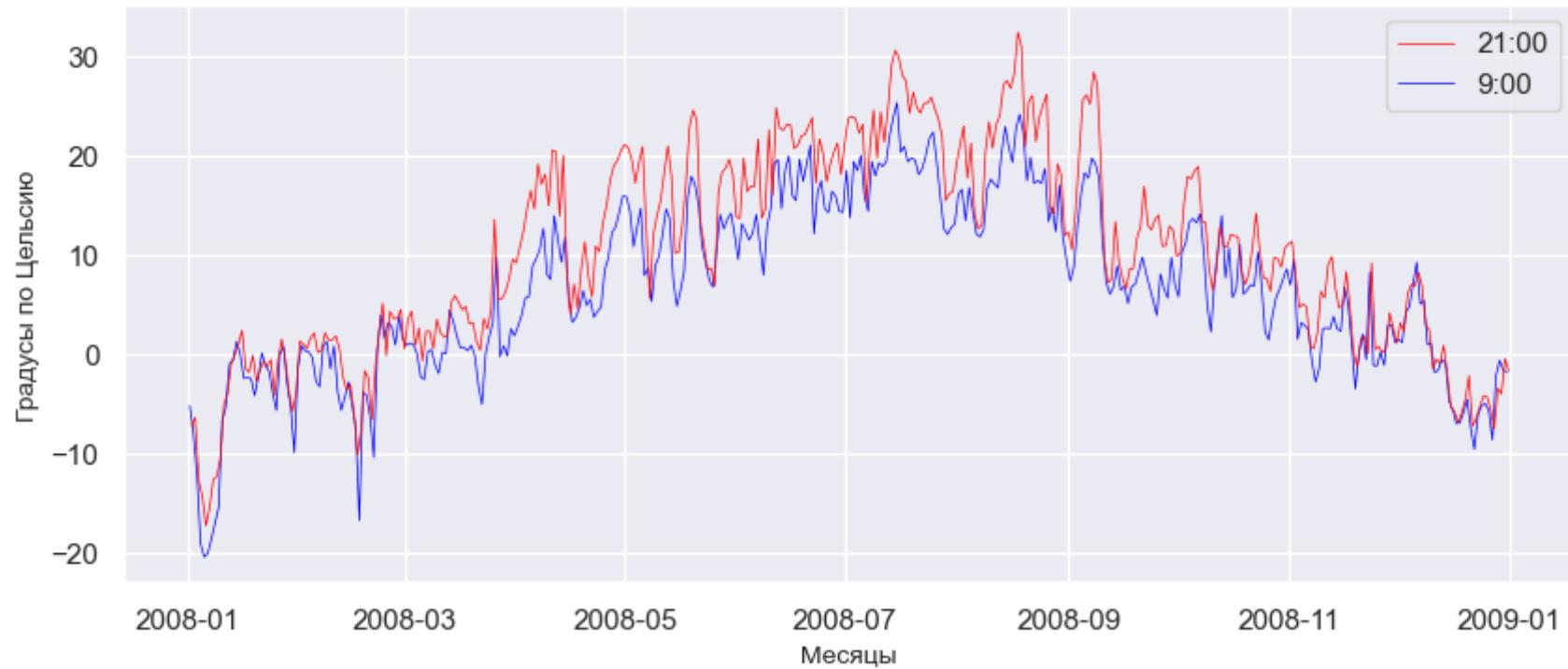
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2010 год



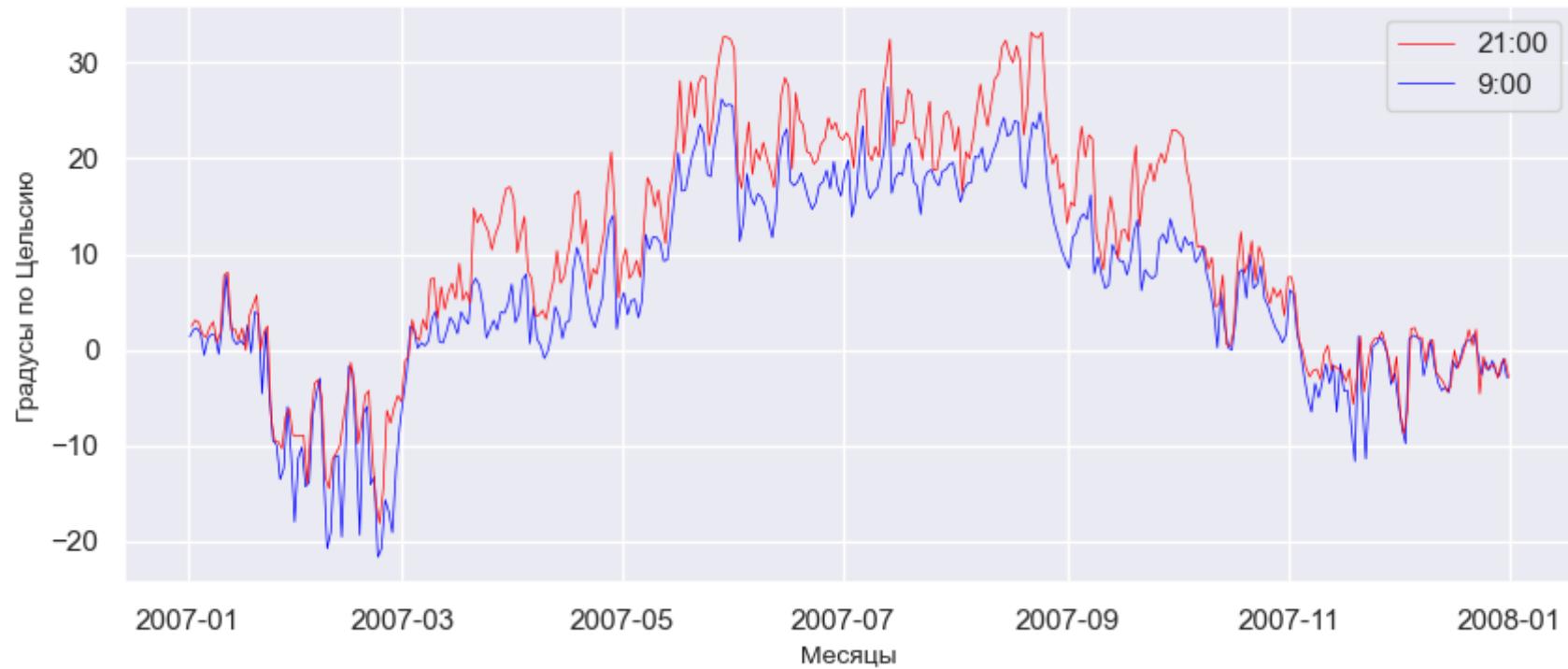
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2009 год



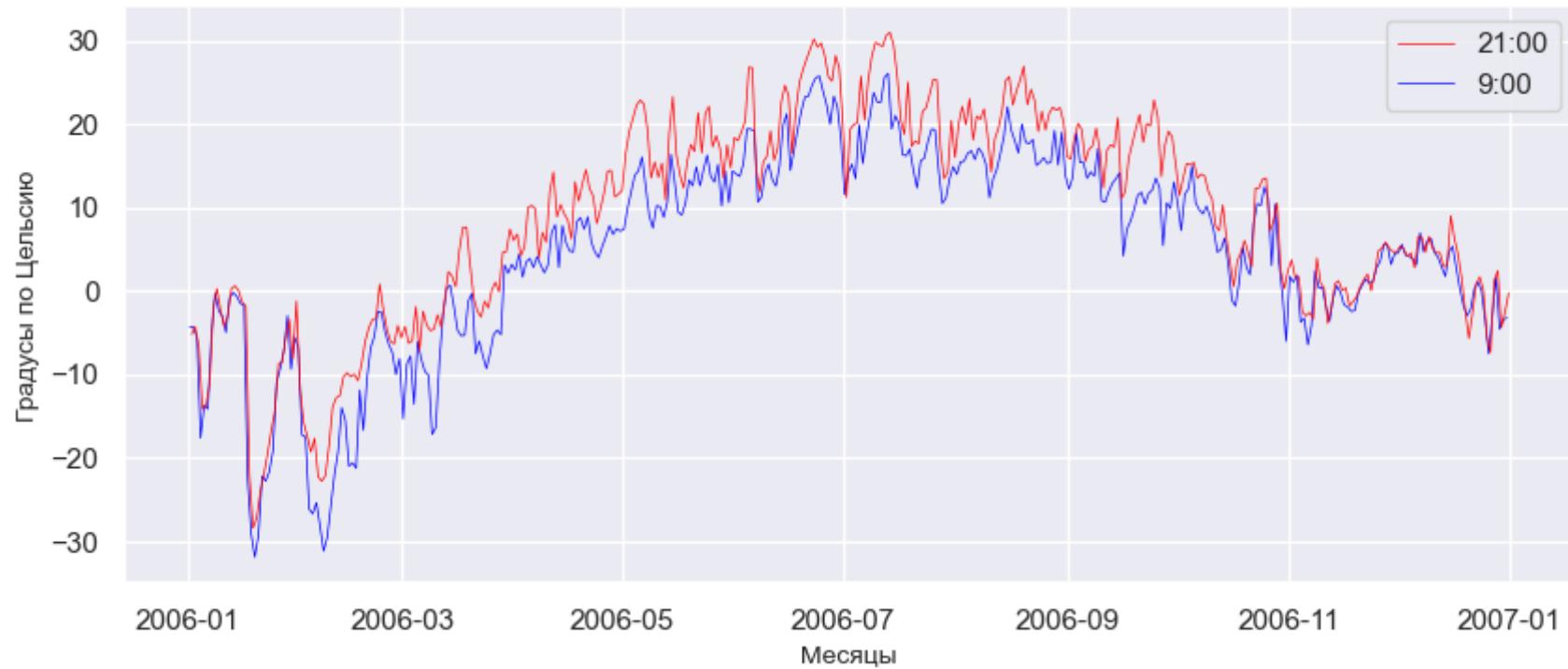
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2008 год



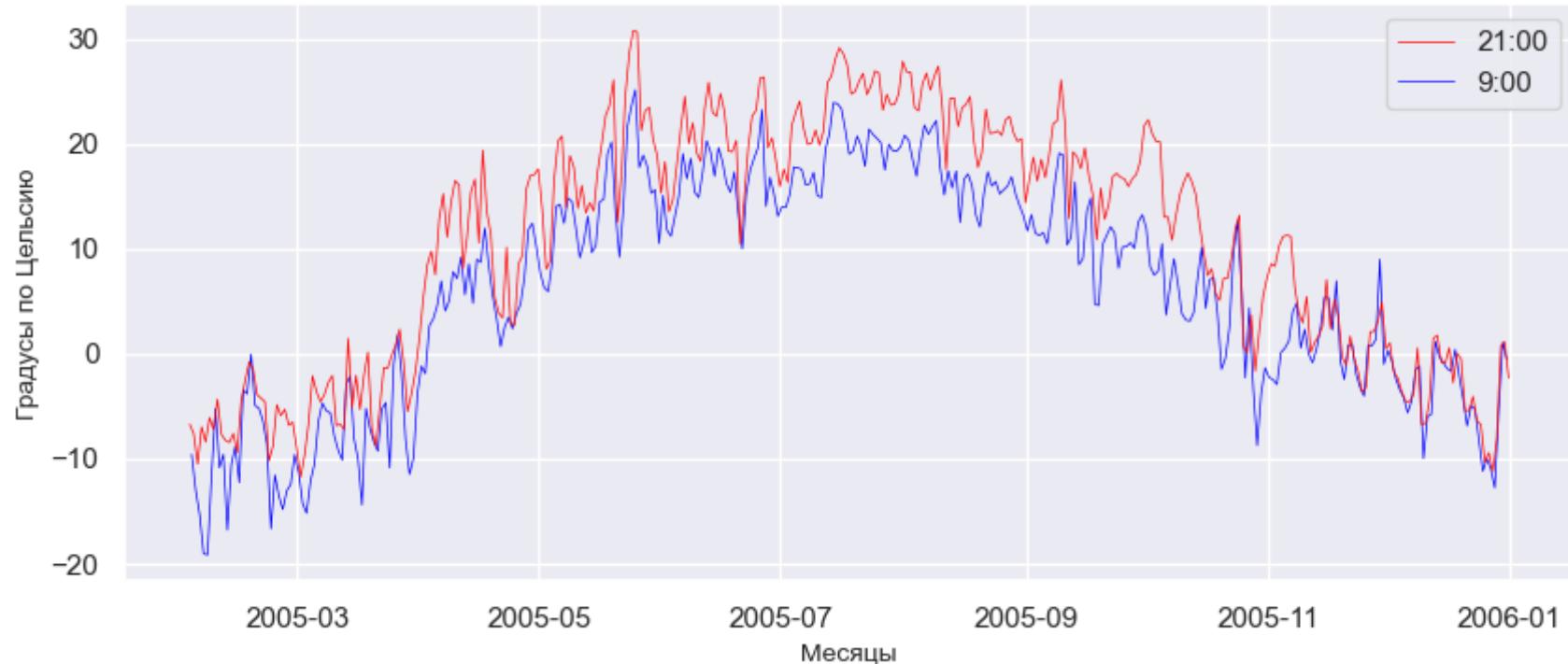
Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2007 год



Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2006 год



Чашниково: Ежедневная динамика параметра T\_max за 12 предшествующих часов на 9 и 21 часов, 2005 год



### 3.3.6. Сохранение полученных данных в файлы

In [232...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER33}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER33}'  
  
makedirs(predict_path1, exist_ok=True)  
makedirs(predict_path2, exist_ok=True)  
  
# Запишем текущие данные в файлы  
for name in dict_df_locations.keys():  
    print(name + '.csv ->', end=' ')
```

```

    dict_df_locations[name].to_csv(
        path_or_buf=f'{predict_path1}{name}.csv'
    )
    print('DONE! ')

print('df_'+PARAMETER33 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER33].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER33}.csv'
)
print('DONE!')

```

```

df_Dmitrov.csv -> DONE!
df_Kashyn.csv -> DONE!
df_Klin.csv -> DONE!
df_Mozhaisk.csv -> DONE!
df_Naro_Fominsk.csv -> DONE!
df_Nemchinovka.csv -> DONE!
df_N_Jerusalem.csv -> DONE!
df_Serpukhov.csv -> DONE!
df_Staritsa.csv -> DONE!
df_Tver.csv -> DONE!
df_Volokolamsk.csv -> DONE!
df_V_Volochev.csv -> DONE!
df_Chashnikovo.csv -> DONE!
df_Rfrnce_point.csv -> DONE!
df_T_max.csv -> DONE!

```

## ПРОДОЛЖЕНИЕ В ТЕТЕРАДИ 3

# Разное

Вернём значения отображения к настройкам по умолчанию

```
In [233...]: pd.set_option('display.max_rows', 60) # восстановим значение по умолчанию максимума отображаемых строк
pd.set_option('display.max_columns', 20) # восстановим значение по умолчанию максимума отображаемых столбцов
```

```
In [234...]: InteractiveShell.ast_node_interactivity = "last_expr" # Отображение результата только последней строки кода
```

```
In [ ]:
```

