

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 4: Исследование аномалий, пропусков и ошибок, а также восстановление, исправление и моделирование значений для каждого индивидуального параметра: численные показатели состояния почвы - температура почвы (*Soil_T*), - и снегонакопления - высота снежного покрова (*Snow_height*) ===*

0. Подготовка данных 4-й тетради

0.0. Импорт необходимых библиотек, настройки представления, константы

```
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
# Последовательность исполнения
# ... #
# - регрессоры
# ... #
# - классификаторы
# ... #
# - 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 [611...]: from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

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

```
In [612...]: 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 [613...]: sns.set_theme()
```

Определим константы и значения (сообразно предыдущим тетрадям)

```
# Константы параметров
PARAMETER31 = 'T'
PARAMETER32 = 'T_min'
PARAMETER33 = 'T_max'
PARAMETER41 = 'P_sea'
PARAMETER42 = 'P_station'
PARAMETER43 = 'P_drift'
PARAMETER51 = 'Humid'
PARAMETER52 = 'Dew_point'
list_const_param = [PARAMETER31,
                    PARAMETER32,
                    PARAMETER33,
                    PARAMETER41,
                    PARAMETER42,
                    PARAMETER43,
                    PARAMETER51,
                    PARAMETER52]
list_const_param
```

```
Out[614]: ['T', 'T_min', 'T_max', 'P_sea', 'P_station', 'P_drift', 'Humid', 'Dew_point']
```

0.1. Импорт данных

In [615...]

```
# Определим значения переменных path (по результатам обработки данных в 3-й тетради)
# path = 'data/csv/' # общий путь к данным
# path1 = path + 'locations/' # путь к архивам метеостанций
# path2 = path + 'parameters/' # путь к архивам параметров

path = 'data/csv/' # общий путь к данным
path1 = f'{path}predict/{PARAMETER52}/locations/' # путь к архивам метеостанций, в последней редакции (Dew_point)
path2 = path + 'predict/' # путь к архивам параметров с рассчётыми значениями
path3 = f'{path}raw/' # путь к исходным архивам параметров
```

0.1.1. Данные метеостанций

0.1.1.1. Информация о метеостанциях

In [616...]

```
# 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[616]:

	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[616]:

	station_ID	station	LaN	LoE	height	V_Voloche	Staritsa	Kashyn	Tver	Klin	Dmitrov	Volokolamsk	Mozha
0	26393	V_Voloche	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[616]:

	station_ID	station	LaN	LoE	height	V_Voloche	Staritsa	Kashyn	Tver	Klin	Dmitrov	Volokolamsk	Mozha
0	26393	V_Voloche	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[616]:

	station_ID	station	LaN	LoE	height	lin_ver	lin_hor
0	26393	V_Voloche	57.583333	34.566667	161	296.603273	0.000000
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 [617...]

```
# Создадим список файлов с архивами метеостанций
list_df_locations_files = [file_name for file_name in listdir(path1) if isfile(join(path1, file_name))]
```

Out[617]:

```
['df_Chashnikovo.csv',
 'df_Dmitrov.csv',
 'df_Kashyn.csv',
 'df_Klin.csv',
 'df_Mozhaisk.csv',
 'df_Naro_Fominsk.csv',
 'df_Nemchinovka.csv',
 'df_N_Jerusalem.csv',
 'df_Rfrnce_point.csv',
 'df_Serpukhov.csv',
 'df_Staritsa.csv',
 'df_Tver.csv',
 'df_Volokolamsk.csv',
 'df_V_Voloche.csv']
```

In [618]:

```

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 = 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=False)
    print('O.K.')
# Выводим полученные ключи словаря
dict_df_locations.keys()

```

```

data/csv/predict/Dew_point/locations/df_Chashnikovo.csv - O.K.
data/csv/predict/Dew_point/locations/df_Dmitrov.csv - O.K.
data/csv/predict/Dew_point/locations/df_Kashyn.csv - O.K.
data/csv/predict/Dew_point/locations/df_Klin.csv - O.K.
data/csv/predict/Dew_point/locations/df_Mozhaisk.csv - O.K.
data/csv/predict/Dew_point/locations/df_Naro_Fominsk.csv - O.K.
data/csv/predict/Dew_point/locations/df_Nemchinovka.csv - O.K.
data/csv/predict/Dew_point/locations/df_N_Jerusalem.csv - O.K.
data/csv/predict/Dew_point/locations/df_Rfrnce_point.csv - O.K.
data/csv/predict/Dew_point/locations/df_Serpukhov.csv - O.K.
data/csv/predict/Dew_point/locations/df_Staritsa.csv - O.K.
data/csv/predict/Dew_point/locations/df_Tver.csv - O.K.
data/csv/predict/Dew_point/locations/df_Volokolamsk.csv - O.K.
data/csv/predict/Dew_point/locations/df_V_Volochev.csv - O.K.

```

Out[618]:

```

dict_keys(['df_Chashnikovo', 'df_Dmitrov', 'df_Kashyn', 'df_Klin', 'df_Mozhaisk', 'df_Naro_Fominsk', 'df_Nemchinovka', 'df_N_Jerusalem', 'df_Rfrnce_point', 'df_Serpukhov', 'df_Staritsa', 'df_Tver', 'df_Volokolamsk', 'df_V_Volochev'])

```

In [619]:

```

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

```

Out[619]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point
2015-05-12 09:00:00	17.038501	6.673042	17.038501	765.489827	746.317823	0.333208	41.094161	3.715406
2009-12-07 00:00:00	-5.095486	-5.095486	-4.235719	773.586566	752.633804	-0.092392	79.890020	-8.017601
2005-10-21 03:00:00	-2.046775	-2.046775	2.045295	767.717147	747.154037	-0.380747	82.059286	-4.691684

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr	Pi
Dmitrov_Local_time														
2011-08-05 15:00:00	22.5	744.4	759.9	-0.4	36.0	NNE	22.5	375.0	3.0	NaN	NaN	20–30%.	NaN	
2018-08-20 06:00:00	16.5	745.6	761.5	0.1	99.0	SW	225.0	3750.0	1.0	NaN	NaN	от 90 менее 100%	NaN	
2014-11-25 03:00:00	-2.7	761.3	778.7	-0.8	91.0	S	180.0	3000.0	2.0	NaN	NaN	100%.	NaN	

Out[619]:

T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_c
---	-----------	-------	---------	-------	----------	-------------	------------	------------	-------	----------	-----------	--------

Kashyn_Local_time

2015-06-10 21:00:00	13.756662	750.7	763.3	-0.8	57.0	штиль	360.0	6000.0	0.0	NaN	NaN	40%.	N
--------------------------------	-----------	-------	-------	------	------	-------	-------	--------	-----	-----	-----	------	---

2020-02-29 09:00:00	-2.700000	737.8	750.8	0.3	86.0	W	270.0	4500.0	4.0	NaN	NaN	от 90 менее 100%	Состоян неб общем изменилс
--------------------------------	-----------	-------	-------	-----	------	---	-------	--------	-----	-----	-----	------------------------	-------------------------------------

2018-08-11 12:00:00	25.100000	753.7	765.7	-0.8	46.0	WSW	247.5	4125.0	4.0	NaN	NaN	20–30%.	N
--------------------------------	-----------	-------	-------	------	------	-----	-------	--------	-----	-----	-----	---------	---

Out[619]:

T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr	Prcp
---	-----------	-------	---------	-------	----------	-------------	------------	------------	-------	----------	-----------	-----------	------

Klin_Local_time

2021-03-08 21:00:00	-8.6	741.3	757.3	2.0	85.0	N	0.0	0.0	2.0	NaN	NaN	100%.	Состояние неба в общем не изменилось.	Со изм
--------------------------------	------	-------	-------	-----	------	---	-----	-----	-----	-----	-----	-------	--	-----------

2020-09-19 09:00:00	7.6	743.4	758.5	0.5	97.0	W	270.0	4500.0	2.0	NaN	NaN	100%.	NaN
--------------------------------	-----	-------	-------	-----	------	---	-------	--------	-----	-----	-----	-------	-----

2017-05-24 06:00:00	8.5	744.1	759.1	0.6	90.0	S	180.0	3000.0	1.0	NaN	NaN	0%	NaN
--------------------------------	-----	-------	-------	-----	------	---	-------	--------	-----	-----	-----	----	-----

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr	1
Mozhaisk_Local_time														
2018-09-08 00:00:00	10.5	749.7	766.6	0.6	97.0	штиль	360.0	6000.0	0.0	NaN	NaN	NaN	NaN	NaN
2007-08-01 18:00:00	19.9	746.0	762.1	-0.1	46.0	WNW	292.5	4875.0	2.0	NaN	NaN	от 90 менее 100%	NaN	NaN
2017-11-12 03:00:00	5.0	736.2	753.1	-0.7	95.0	SE	135.0	2250.0	4.0	NaN	NaN	NaN	NaN	NaN

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wth
Naro_Fominsk_Local_time													
2020-10-27 09:00:00	5.0	752.3	770.2	0.2	91.0	SSE	157.5	2625.0	1.0	NaN	NaN	от 90 менее 100%	NaN
2021-11-02 03:00:00	-3.6	748.9	767.2	-0.3	100.0	штиль	360.0	6000.0	0.0	NaN	NaN	100%.	Туман ле
2010-11-16 12:00:00	8.7	745.3	762.8	1.0	96.0	W	270.0	4500.0	2.0	NaN	NaN	100%.	Ливневый дождь слабый нас

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Clo
Nemchinovka_Local_time												
2017-07-19 18:00:00	20.80000	746.800000	762.300000	-0.100000	52.000000	NW		315.0	5250.0	2.0	NaN	NaN
2007-08-28 21:00:00	14.67880	739.521917	755.076148	-0.197874	73.219207		NaN	NaN	NaN	NaN	NaN	NaN
2010-12-23 18:00:00	-6.95499	749.778059	766.844117	1.778059	87.158674		NaN	NaN	NaN	NaN	NaN	NaN

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	WtI
N_Jerusalem_Local_time													
2007-03-12 12:00:00	3.3	752.8	768.0	1.4	69.0	штиль		360.0	6000.0	0.0	NaN	NaN	100%.
2009-11-01 06:00:00	-2.5	752.6	768.0	0.0	87.0	NNW		337.5	5625.0	2.0	NaN	NaN	100%. Ливневъ слабый наблк иј
2021-09-23 09:00:00	7.9	745.1	759.6	-0.4	98.0	NNE		22.5	375.0	1.0	NaN	NaN	100%. незамерза непрер слабый в

Out[619]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point
2010-10-24 21:00:00	5.844843	5.844843	9.734693	762.061578	749.265789	-0.112527	71.361757	1.065133
2014-03-10 03:00:00	2.976921	2.976921	6.282096	771.570964	758.482092	-0.593379	64.811328	-2.991229
2018-07-21 12:00:00	22.734008	19.034243	22.734008	755.514419	743.546911	0.796240	77.176946	18.522621

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Clou
Serpukhov_Local_time												
2016-02-23 00:00:00	0.662593	727.102132	741.423181	3.038708	89.645541	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-02-26 06:00:00	1.800000	733.700000	749.000000	1.900000	89.000000	WNW	292.5	4875.0	3.0	NaN	NaN	70 -
2018-03-02 03:00:00	-11.100000	753.400000	769.600000	-0.900000	71.000000	E	90.0	1500.0	3.0	NaN	NaN	

Out[619]:

	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													
2018-10-28 15:00:00	6.3	741.3	758.3	2.7	74.0	W	270.0	4500.0	2.0	NaN	NaN	от 90 менее 100%	NaN
2010-08-06 03:00:00	19.8	746.7	762.9	1.3	72.0	штиль	360.0	6000.0	0.0	NaN	NaN	40%.	NaN
2009-11-01 21:00:00	-2.0	751.6	769.4	0.0	77.0	WNW	292.5	4875.0	2.0	NaN	NaN	от 90 менее 100%	Состояние неба в общем не изменилось.

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_cu
Tver_Local_time													
2013-09-19 03:00:00	9.7	744.800000	757.518637	-0.200000	94.0	E	90.0	1500.0	2.0	NaN	NaN	20–30%.	N&
2022-05-02 09:00:00	10.2	751.750096	763.835410	-0.606894	37.0	SSW	202.5	3375.0	2.0	NaN	NaN	40%.	N&
2009-03-17 00:00:00	-4.9	758.108598	770.987974	-0.129132	96.0	штиль	360.0	6000.0	0.0	NaN	NaN	40%.	N&

Out[619]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_cu
Volokolamsk_Local_time													
2012-03-05 12:00:00	-5.4	743.900000	762.9	0.500000	75.0	NNE	22.5	375.0	4.0	NaN	NaN	от 90% менее 100%	Лив снег с наблю иц
2008-06-27 09:00:00	15.4	743.000000	760.5	0.400000	77.0	WNW	292.5	4875.0	1.0	NaN	NaN	60%.	
2016-11-22 18:00:00	-5.8	757.64488	777.1	-0.114287	74.0	S	180.0	3000.0	4.0	NaN	NaN	NaN	

Out[619]:

V_Volochek_Local_time	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudnes
2010-03-15 12:00:00	-4.959113	732.098455	747.3	3.098455	64.969178	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-07-08 12:00:00	29.100000	754.100000	768.5	0.100000	41.000000	WSW	247.5	4125.0	2.0	NaN	NaN	60%
2016-01-07 06:00:00	-21.000000	743.000000	760.3	-0.300000	85.000000	WSW	247.5	4125.0	1.0	NaN	NaN	от 9 мене 100%

0.1.2.2. Чтение архивов параметров

In [620...]

```
# Создадим список файлов с архивами параметров
list_df_parameters_files = [file_name for file_name in.listdir(path3) if.isfile(join(path3, file_name))]
list_df_parameters_files
```

```
Out[620]: ['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 [621...]: # 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
    # проверяем, есть ли файл с исправленными значениями параметра (по списку)
    if name_df[3:] in list_const_param:
        file_path = f'{path2}{name_df[3:]}/{file_name}' # Формируем путь к файлу
    else:
        file_path = f'{path3}{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_C1_bottom.csv - O.K.
data/csv/raw/df_C1_cirrus.csv - O.K.
data/csv/raw/df_C1_Cumls.csv - O.K.
data/csv/raw/df_C1_cumls_hi.csv - O.K.
data/csv/raw/df_C1_viewd.csv - O.K.
data/csv/raw/df_Depo_diam_mm.csv - O.K.
data/csv/predict/Dew_point/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/predict/Humid/df_Humid.csv - O.K.
data/csv/raw/df_Prcptn.csv - O.K.
data/csv/raw/df_Prcptn_depo.csv - O.K.
data/csv/raw/df_Prcptn_like.csv - O.K.
data/csv/raw/df_Prcptn_tdelt.csv - O.K.
data/csv/predict/P_drift/df_P_drift.csv - O.K.
data/csv/predict/P_sea/df_P_sea.csv - O.K.
data/csv/predict/P_station/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/predict/T/df_T.csv - O.K.
data/csv/predict/T_max/df_T_max.csv - O.K.
data/csv/predict/T_min/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[621]: 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_tdel', '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_cmr'])
```

```
In [622...]: for name in dict_df_parameters.keys():
    dict_df_parameters[name].sample(3)
```

Out[622]:	Cloudness#V_Volochev	Cloudness#Staritsa	Cloudness#Kashyn	Cloudness#Tver	Cloudness#Klin	Cloudness#Dmitrov	Cloudness#Volokolamsk	Cloudness#K
2018-02-09 09:00:00	100%.	100%.	от 90 менее 100%	100%.	100%.	100%.	100%.	NaN
2013-07-13 06:00:00	0%	40%.	20–30%.	20–30%.	больше 0 до 10%	40%.	Небо вне видимости	
2007-07-04 18:00:00	Nan	от 90 менее 100%	Nan	40%.	40%.	70 – 80%.	40%.	

Out[622]:	Cl_bottom#V_Volochev	Cl_bottom#Staritsa	Cl_bottom#Kashyn	Cl_bottom#Tver	Cl_bottom#Klin	Cl_bottom#Dmitrov	Cl_bottom#Volokolamsk	Cl_bottom#K
2014-02-06 03:00:00	600-1000	300-600	300-600	200-300	600-1000	300-600	600-1000	
2011-07-06 18:00:00	600-1000	1000-1500	300-600	1000-1500	1000-1500	600-1000	1000-1500	
2012-01-14 15:00:00	600-1000	2000-2500	2500 или более, или облаков нет.	2500 или более, или облаков нет.	600-1000	300-600	600-1000	

Out[622]:

	Cl_cirrus#V_Volochev	Cl_cirrus#Staritsa	Cl_cirrus#Kashyn	Cl_cirrus#Tver	Cl_cirrus#Klin	Cl_cirrus#Dmitrov	Cl_cirrus#Volokolamsk	Cl_cirrus#Moz
2020-08-26 15:00:00	Перистых, перисто-кучевых или перисто-слоистых...	Перистых, перисто-кучевых или перисто-слоистых...		NaN	NaN	NaN	NaN	NaN
2010-02-04 00:00:00	Перисто-кучевые одни или перисто-кучевые, сопр...	Перисто-кучевые одни или перисто-кучевые, сопр...		NaN	Перисто-кучевые одни или перисто-кучевые, сопр...			
2008-03-01 21:00:00		NaN	NaN	Перисто-кучевые одни или перисто-кучевые, сопр...	NaN	NaN	NaN	NaN

Out[622]:

	Cl_Cumls#V_Volochev	Cl_Cumls#Staritsa	Cl_Cumls#Kashyn	Cl_Cumls#Tver	Cl_Cumls#Klin	Cl_Cumls#Dmitrov	Cl_Cumls#Volokolamsk	Cl_Cumls
2022-05-08 06:00:00	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевых, слоистых, кучевых или кучево...	Слоисто-кучевых, слоистых, кучевых или кучево...		NaN	NaN	Слоисто-кучевые, образовавшиеся не из кучевых.
2015-01-19 12:00:00	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевых, слоистых, кучевых или кучево...	Слоисто-кучевых, слоистых, кучевых или кучево...	Слоисто-кучевых, слоистых, кучевых или кучево...	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевых, слоистых, кучевых или кучево...	Слоисто-кучевые, образовавшиеся не из кучевых.
2020-11-02 00:00:00	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевых, слоистых, кучевых или кучево...	Слоисто-кучевые, образовавшиеся не из кучевых.	Кучево-дождевые волокнистые (часто с наковалн...	Слоистые разорванные или кучевые разорванные о...		NaN

Out[622]:	Cl_cumls_hi#V_Volochek	Cl_cumls_hi#Staritsa	Cl_cumls_hi#Kashyn	Cl_cumls_hi#Tver	Cl_cumls_hi#Klin	Cl_cumls_hi#Dmitrov	Cl_cumls_hi#Vol
2007-04-29 00:00:00	NaN	Высококучевые башенообразные или хлопьевидные.	NaN	Высококучевые просвечивающие, расположенные на...	Высококучевые просвечивающие, расположенные на...	NaN	Высокобашенообразные хлопьевидные.
2012-10-28 18:00:00	Высокослоистые непросвечивающие или слоисто-до...	NaN	Высокослоистые непросвечивающие или слоисто-до...	NaN	Высокослоистые непросвечивающие или слоисто-до...	Высокослоистые непросвечивающие или слоисто-до...	Высоконепросвечивающие слои
2019-10-28 12:00:00	Высококучевых, высокослоистых или слоисто-дожд...	Высоко высокослоистые					

Out[622]:	Cl_viewd#V_Volochek	Cl_viewd#Staritsa	Cl_viewd#Kashyn	Cl_viewd#Tver	Cl_viewd#Klin	Cl_viewd#Dmitrov	Cl_viewd#Volokolamsk	Cl_viewd#M
2022-06-04 09:00:00	60%.	NaN	NaN	70 – 80%. менее, но не 0	10% или	NaN	NaN	NaN
2015-02-19 00:00:00	90 или более, но не 100%	20–30%.	100%.	100%.	100%.	70 – 80%.	100%.	
2021-01-28 18:00:00	100%.	100%.	90 или более, но не 100%	100%.	100%.	100%.	100%.	

Out[622]:	Depo_diam_mm#V_Volochek	Depo_diam_mm#Staritsa	Depo_diam_mm#Kashyn	Depo_diam_mm#Tver	Depo_diam_mm#Klin	Depo_diam_mm#Dm
2010-07-28 21:00:00	0.0	0.0	0.0	0.0	0.0	0.0
2008-10-20 09:00:00	0.0	0.0	0.0	0.0	0.0	0.0
2013-11-18 21:00:00	0.0	0.0	0.0	0.0	0.0	0.0

Out[622]:	Dew_point#V_Volochek	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
2020-12-11 03:00:00	-11.1	-10.4	-13.3	-11.7	-11.0	-13.0	-12.200000
2010-09-03 15:00:00	8.8	6.5	8.9	7.9	6.7	5.4	8.000000
2018-03-14 15:00:00	-0.7	-2.9	-4.0	-3.1	-6.0	-5.8	-5.864993

Out[622]:	Prcpttn_like#V_Volocheok	Prcpttn_like#Staritsa	Prcpttn_like#Kashyn	Prcpttn_like#Tver	Prcpttn_like#Klin	Prcpttn_like#Dmitrov	Prcpttn_like#Volokolamsk	Prcpttn_like#Mozhaisk	Prcpttn_like#Voronezh
2014-07-17 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-02-27 18:00:00	Ливневый снег умеренный или сильный в срок наб...	Снег непрерывный слабый в срок наблюдения.	Снег непрерывный слабый в срок наблюдения.	Дымка.	Дождь (незамерз нели)				
2006-10-14 09:00:00	NaN	NaN	NaN	NaN	Дождь незамерзающий непрерывный слабый в срок ...	Ливневый(ые) дождь(и).	NaN	Облака т образовывал разви	
Out[622]:	Prcpttn_tdelt#V_Volocheok	Prcpttn_tdelt#Staritsa	Prcpttn_tdelt#Kashyn	Prcpttn_tdelt#Tver	Prcpttn_tdelt#Klin	Prcpttn_tdelt#Dmitrov	Prcpttn_tdelt#Volokolamsk	Prcpttn_tdelt#Mozhaisk	Prcpttn_tdelt#Voronezh
2019-08-06 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2014-06-25 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2005-04-06 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Out[622]:	P_drift#V_Volocheok	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#Voronezh
2011-02-03 15:00:00	0.600000	0.3	1.100000	0.694232	0.9	1.0	0.6	0.1	
2007-09-28 03:00:00	-1.573815	0.0	-0.760602	-0.298328	0.1	0.0	0.4	0.6	
2014-06-28 21:00:00	0.600000	0.9	0.600000	0.600000	0.5	0.5	0.8	1.1	

Out[622]:	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
2011-12-10 15:00:00	0.0	0.0	-0.5	-0.2	-0.2	-0.60000	-0.5000	-0.200000	-0.1	0.000000	-0
2013-11-14 21:00:00	2.5	1.7	2.5	2.3	2.3	2.49486	2.4128	2.417609	2.7	2.436728	2
2017-09-10 06:00:00	11.6	10.7	10.3	11.8	12.3	11.80000	10.8000	9.400000	9.7	9.504240	9
Out[622]:	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_Jerusalem	T_max#Nemchinovka	T_max#Naro_Fomin
2020-02-13 21:00:00	2.8	2.7	-1.5	1.2	-0.3	-1.60000	0.200000	0.3			
2009-09-17 09:00:00	12.4	10.5	12.8	13.2	12.6	13.400000	12.500000	10.7			
2015-10-17 06:00:00	2.1	2.3	1.5	3.3	0.8	0.736374	2.083685	2.1			
Out[622]:	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_Jerusalem	T_min#Nemchinovka	T_min#Naro_Fomin
2008-01-31 12:00:00	0.621984	0.5	-1.5	-0.2	-1.8	-3.2	-1.4	-3.9			
2018-05-03 18:00:00	12.000000	14.6	13.8	15.2	16.6	16.6	15.4	19.3			
2010-02-13 15:00:00	-14.000000	-14.2	-14.9	-13.7	-11.9	-13.4	-13.2	-13.3			

Out[622]:	Visibility#V_Volochek	Visibility#Staritsa	Visibility#Kashyn	Visibility#Tver	Visibility#Klin	Visibility#Dmitrov	Visibility#Volokolamsk	Visibility#Moscow
2019-01-26 03:00:00	10.0	10.0	50.0	10.0	10.0	10.0	10.0	10.0
2017-01-27 09:00:00	10.0	10.0	50.0	10.0	10.0	10.0	10.0	4.0
2006-02-22 03:00:00	NaN	2.0	12.0	2.0	2.0	6.0	4.0	
Out[622]:	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
2008-06-13 12:00:00	NaN	E	NaN	WNW	W	WNW	SW	
2016-04-03 09:00:00	NNW	NW	NNW	WNW	WNW	NaN	NaN	
2016-02-03 09:00:00	SW	SW	SW	S	S	NaN	NaN	
Out[622]:	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
2008-08-16 09:00:00	202.5	157.5	135.0	112.5	112.5	90.0		
2018-07-26 09:00:00	135.0	135.0	135.0	135.0	112.5	90.0		
2008-03-07 06:00:00	NaN	225.0	NaN	202.5	225.0	202.5		

Out[622]:	Wind_dir6k#V_Volocheok	Wind_dir6k#Staritsa	Wind_dir6k#Kashyn	Wind_dir6k#Tver	Wind_dir6k#Klin	Wind_dir6k#Dmitrov	Wind_dir6k#Volokola	
2014-09-21 06:00:00	1500.0	3000.0	6000.0	6000.0	2625.0	2625.0	30	
2016-10-09 12:00:00	1125.0	1500.0	1125.0	1125.0	1125.0	1125.0	15	
2021-10-08 00:00:00	3000.0	3750.0	6000.0	6000.0	6000.0	3000.0	30	
Out[622]:	Wind_speed#V_Volocheok	Wind_speed#Staritsa	Wind_speed#Kashyn	Wind_speed#Tver	Wind_speed#Klin	Wind_speed#Dmitrov	Wind_speed#Vol	
2005-06-25 21:00:00	NaN	2.0	NaN	1.0	0.0	1.0		
2005-06-01 06:00:00	NaN	5.0	NaN	2.0	1.0	2.0		
2013-05-10 03:00:00	2.0	3.0	2.0	1.0	1.0	2.0		
Out[622]:	Wthr_3h#V_Volocheok	Wthr_3h#Staritsa	Wthr_3h#Kashyn	Wthr_3h#Tver	Wthr_3h#Klin	Wthr_3h#Dmitrov	Wthr_3h#Volokolamsk	Wthr_3h#Moz
2009-09-18 21:00:00	Ливень (ливни).	NaN	NaN	NaN	Ливень (ливни).	Ливень (ливни).	NaN	Ливень (ли
2008-09-18 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2016-04-18 06:00:00	Ливень (ливни).	Ливень (ливни).	NaN	NaN	NaN	NaN	NaN	NaN

Out[622]:

	Wthr_3h2#V_Volochev	Wthr_3h2#Staritsa	Wthr_3h2#Kashyn	Wthr_3h2#Tver	Wthr_3h2#Klin	Wthr_3h2#Dmitrov	Wthr_3h2#Volokolamsk	Wthr_3h2#Moscow
2017-06-27 00:00:00	NaN	NaN	Облака покрывали более половины неба в течение...	NaN	Ливень (ливни).	Облака покрывали более половины неба в течение...	NaN	NaN
2009-08-29 03:00:00	NaN	Облака покрывали более половины неба в течение...	Облака покрывали половину неба или менее в теч...	Облака покрывали половину неба или менее в теч...	NaN	Облака покрывали более половины неба в течение...	NaN	NaN
2020-01-28 15:00:00	Облака покрывали более половины неба в течение...	Сн						

Out[622]:

	Wthr_curr#V_Volochev	Wthr_curr#Staritsa	Wthr_curr#Kashyn	Wthr_curr#Tver	Wthr_curr#Klin	Wthr_curr#Dmitrov	Wthr_curr#Volokolamsk	Wthr_curr#Moscow
2005-12-02 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Облака в целом рассеиваются или становятся мен...
2007-11-07 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-12-25 15:00:00	Дымка.	Состояние неба в общем не изменилось.	NaN	Дымка.	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 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:

- односторонний, 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 будут 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_tdelta (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_cumls_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 коррелирующихя 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 показателям 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 проупщено 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 $hashnikovo$. То есть: $hashnikovo(Z(t)) = f(station_1(Z(t)), \dots, station_n(Z(t)))$. 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 $hashnikovo$) - 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_i в точке i (x_{i+}) и x_h 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:

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

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

Таким образом, $\|\overrightarrow{u_i}\|$ - 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 искомую 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

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

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 выделенных 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 [623...]

```
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
        station_ по умолчанию - 'Rfrnce_point'
    """

    # 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 пересчур 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 [624...]

```
# 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 [625...]

```
# 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 [626...]

```
# сохраним стандартные настройки вывода
real_stdout = sys.stdout # сделаем backup стандартного вывода
```

In [627...]

```
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 вариаграммы 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 [628...]

```

# 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 n сигм
        lower_level_ = mean_value_ - n_sigma_ * std_value_ # нижний порог n сигм

    return (lower_level_, upper_level_)

```

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

In [629...]

```

# 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 сигма 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, default=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
    # pru_station_='Rfrnce_point'
    mean_value_ = inverse_distance_avg(row_, param_, station_='Rfrnce_point',
                                         df_dists_=df_station_dists, power_=2)
    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 метеостанций
    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 ненан 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_ только,
# если std_value_ не является NaN и std_value_ != 0
if (pd.notna(std_value_) and (std_value_ != 0)):
    result_ = norm.interval(
        confidence=confidence_,

```

```

        loc=mean_value_,
        scale=std_value_
    )
elif (pd.notna(std_value_) and (std_value_ == 0)): # std_value_ существует и равно 0 (все элементы равны)
    result_ = (mean_value_, mean_value_) # Чтобы далее это не было расценено как выброс
else:
    result_ = np.nan # иначе - вернём NaN

return result_

```

2.5. Функция оценки индивидуального отклонения признака в отношении к средней

In [630...]

```

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 [631...]

```

# 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 (deafault='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 усредненение 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:

- Если выбросы обнаружены - список из кортежей (на каждый выброс свой кортеж):
 значение выброса, его индекс в серии, границы п сигм 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 [632...]

```

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 (default=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, проверяемое 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 - може 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 [63...]

```

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 (словарь архивов метеостанций!);
    - 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 [64...]

```

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 название 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 [635...]

```

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 [636...]

```

# 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 [637...]

```
# 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 [638...]

```
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 [639...]

```
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 кригинга  
    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 пнтрю!  
    # 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

# Вызываем 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)

См. тетрадь 2

4. Исследование и обработка индивидуальных показателей метеорологических наблюдений: Атмосферное давление (P_{sea} , P_{station} , P_{drift})

См. тетрадь 3

5. Исследование и обработка индивидуальных показателей метеорологических наблюдений: Концентрация водяных паров в воздухе - относительная влажность воздуха и температура точки росы (Humid , Dew_point)

См. тетрадь 3

6. Исследование и обработка индивидуальных показателей метеорологических наблюдений: Численные показатели состояния почвы - температура почвы, высота снегового покрова (Soil_T , Snow_height)

6.1. Температура почвы: Soil_T

Температура почвы - это температура ее верхнего слоя (толщиной несколько миллиметров), свободного от растительного покрова, хорошо взрыхленного и не затеняемого от солнца, а в зимнее время при наличии снежного покрова – температура поверхности снега. Измеряется термометром, лежащим открыто на поверхности почвы и снежного покрова, при этом резервуар термометра погружен в почву (снежный покров). Измерения температуры поверхности почвы представляют большие методические трудности из-за невозможности затенить термометр от действия радиации и вследствие различия радиационных свойств резервуара и почвы (снега).

Показатель температуры почвы должен фиксироваться метеостанциями ежесуточно в одно и то же время: 9 часов утра Московского декретного времени. Как показано в 1-й тетради, этот показатель хорошо коррелируется между метеостанциями.

6.1.1. Поиск и удаление ошибок показателя температуры почвы Soil_T

Для данного раздела обозначим константу названия параметра.

```
In [640...]: PARAMETER61 = 'Soil_T'
```

Создаём временный DF для работы с параметром Soil_T

```
In [641...]: param_df_name = f'df_{PARAMETER61}' # преобразуем полученное значение в df_PARAMETER61 - ключ словаря dict_df_parameters  
# Создадим временный df  
df_tmp61 = dict_df_parameters[param_df_name].copy(deep=True)  
df_tmp61.sample(5, random_state=56)
```

	Soil_T#V_Volochek	Soil_T#Staritsa	Soil_T#Kashyn	Soil_T#Tver	Soil_T#Klin	Soil_T#Dmitrov	Soil_T#Volokolamsk	Soil_T#Mozhaisk	Soil_T#N_Jeru
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	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2008-06-05 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Определим последовательность действий, для поиска ошибок показателя "Температура почвы".

1. Проверим синхронность наблюдения температуры почвы всеми метеостанциями.
2. Определяем выбросы в поле метеостанций - формируем кандидатов в ошибки.
3. Отдельно проверяем, есть ли единичные значения в рядах - это тоже кандидаты в ошибки.
4. Проверяем каждый из этих кандидатов во временном ряду (только эти выбросы, а не весь DF!) - не подтвердилось - отбрасываем
5. То, что осталось и есть ошибка.

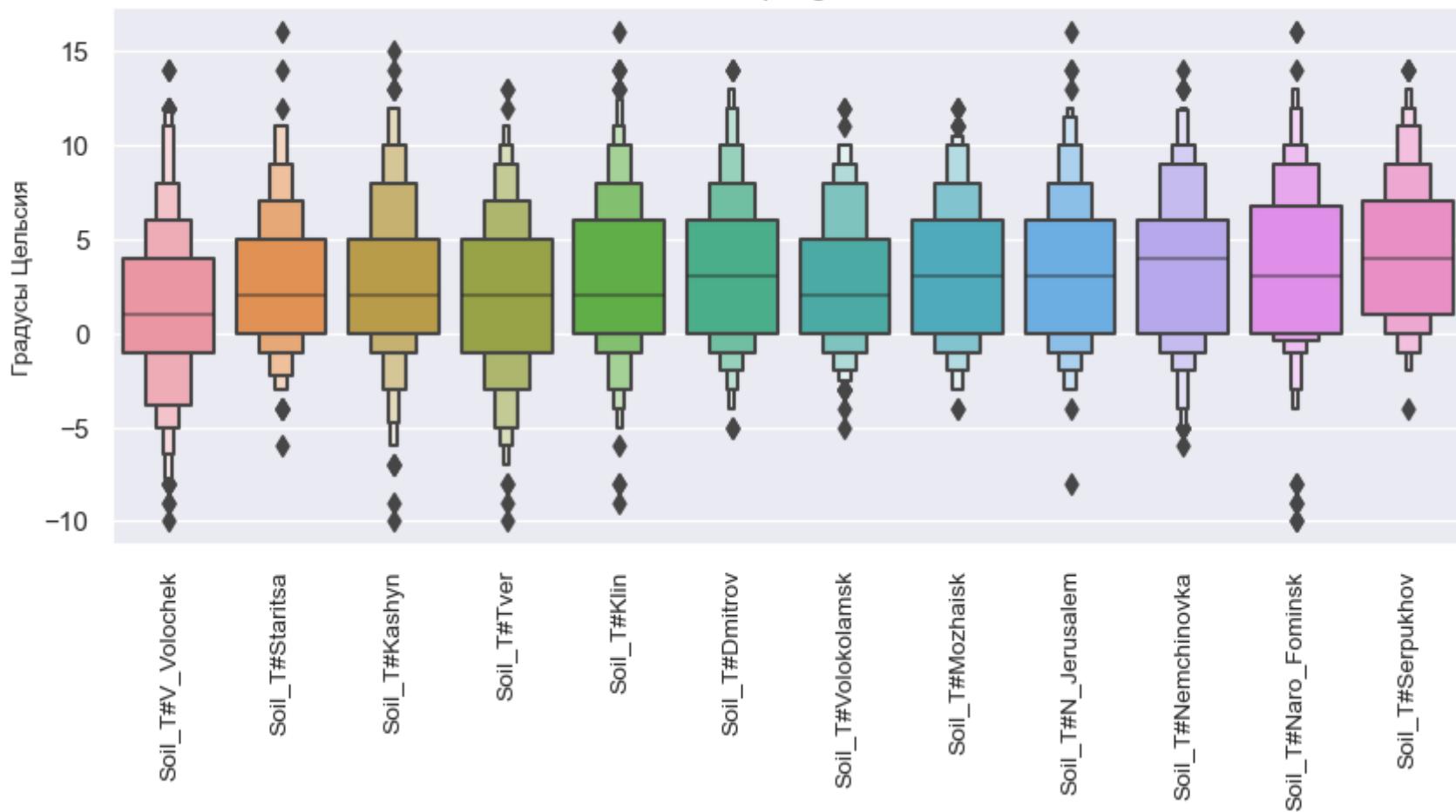
Визуализируем архив температуры почвы (Soil_T) по сезонам

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

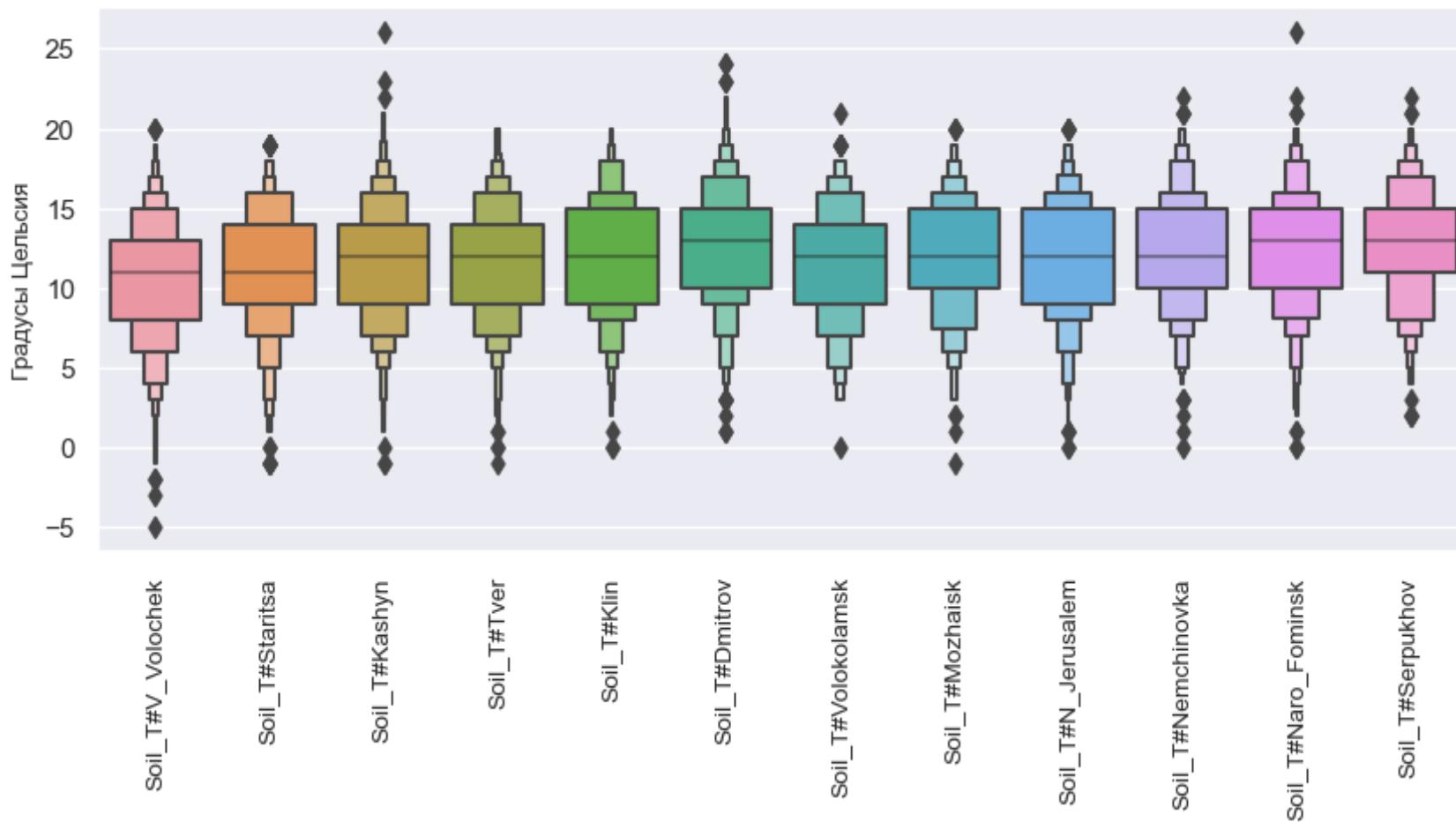
- Зима (ниже 0°): В среднем длится с 5 ноября по 4 апреля
- Весна (от 0° до +10°): В среднем длится с 5 апреля по 18 мая
- Лето (выше +10°): В среднем длится с 19 мая по 14-15 сентября
- Осень (от +10° до 0°): В среднем длится с 14-15 сентября по 4 ноября

```
In [642...]: # В цикле выведем графики давления по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp61).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp61[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Градусы Цельсия', size=10)
    dummy = g.set_title(f'Распределение значений Soil_T в разрезе метеостанций:\n{season_name}')
plt.show()
```

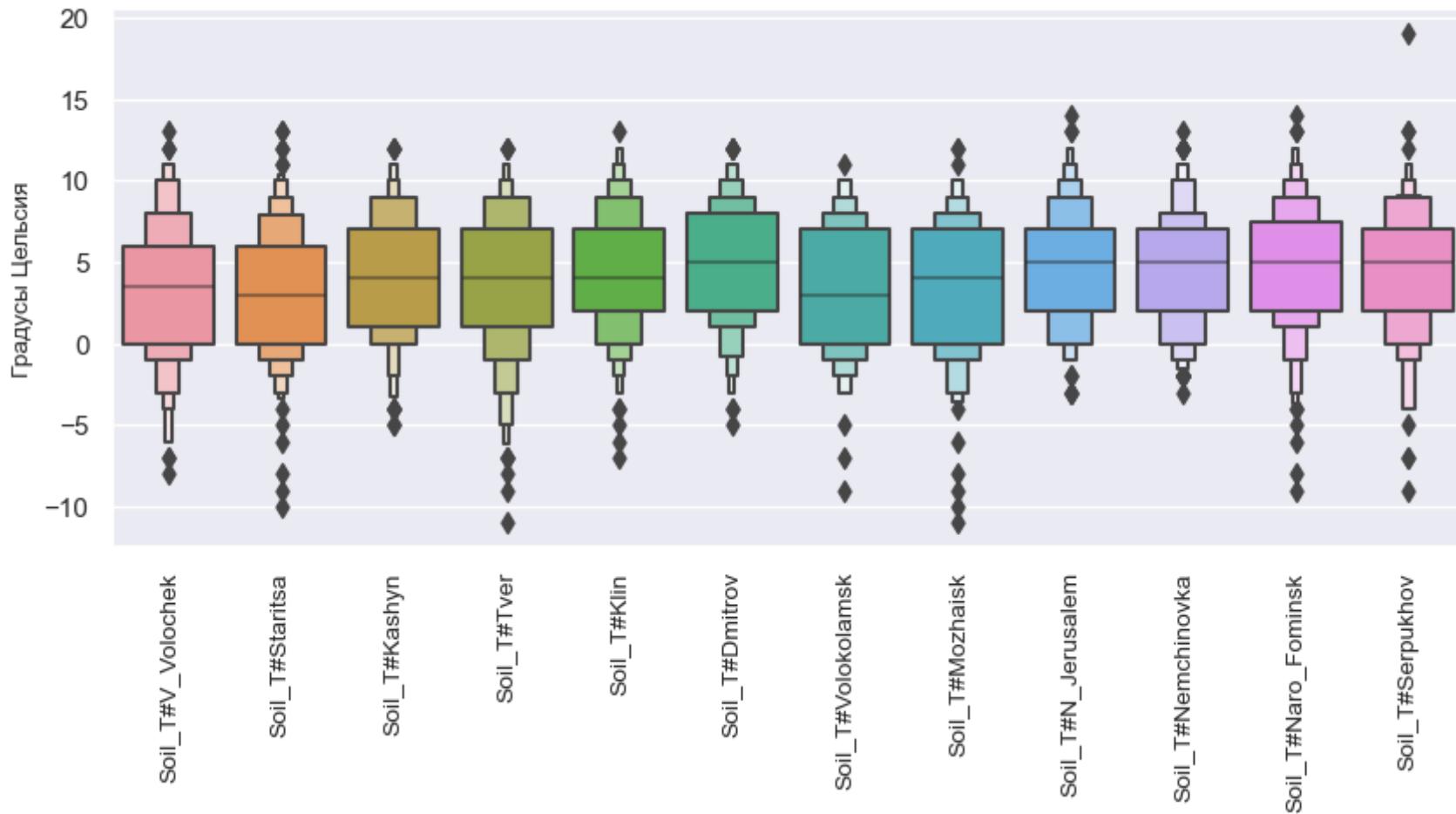
Распределение значений Soil_T в разрезе метеостанций: spring



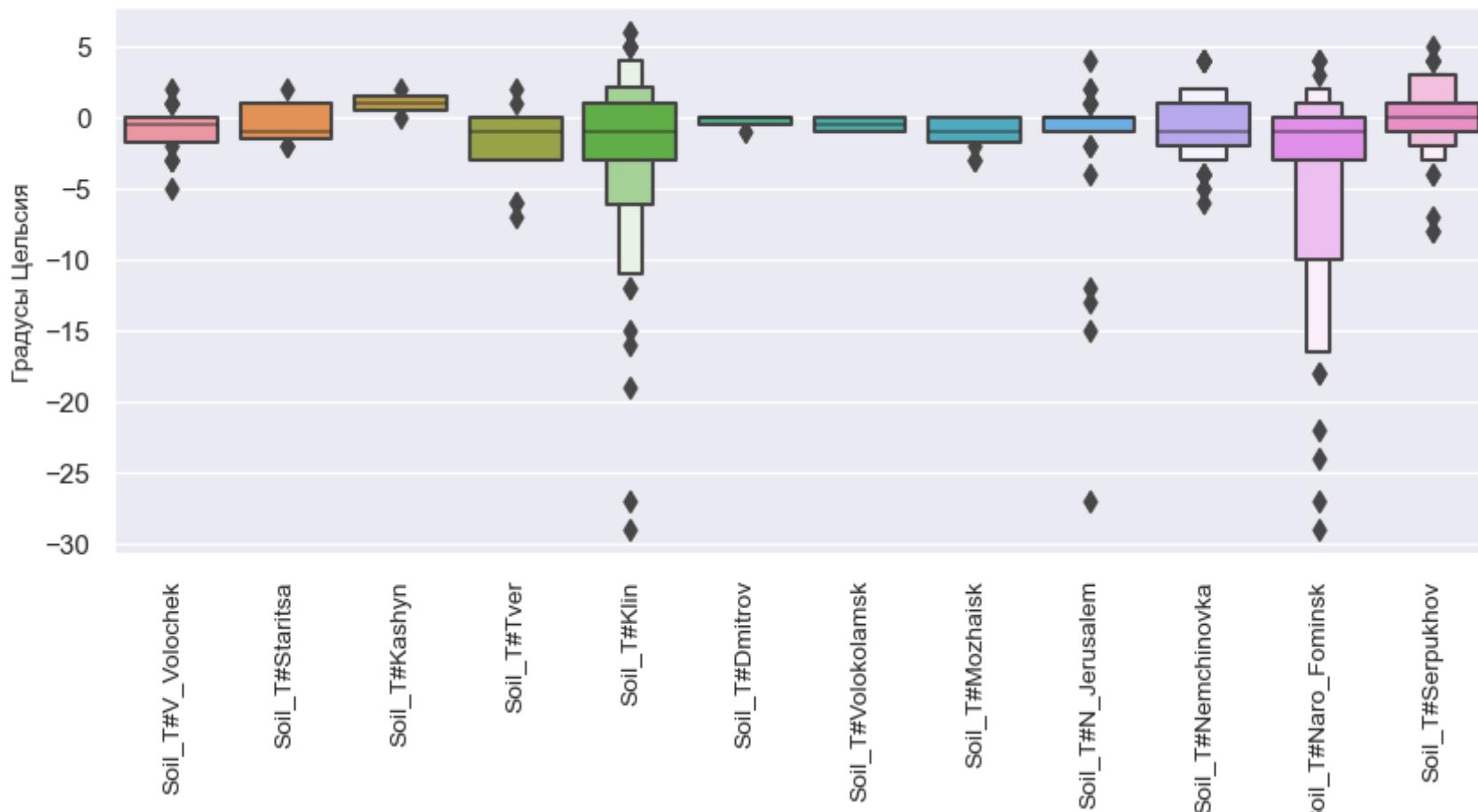
Распределение значений Soil_T в разрезе метеостанций:
summer



Распределение значений Soil_T в разрезе метеостанций:
autumn



Распределение значений Soil_T в разрезе метеостанций: winter



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

```
In [643]: print(f'Минимальное значение: {np.nanmin(df_tmp61)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp61)},\n'
      f'Средняя: {np.nanmean(df_tmp61)},\n'
      f'Медиана: {np.nanmedian(df_tmp61)}')
```

```
Минимальное значение: -29.0,
Максимальное значение: 26.0,
Средняя: 8.113353794933152,
Медиана: 9.0
```

Как видно из графиков и рассчитанных величин, температура почвы не имеет значимых аномальных вбросов.

Определим, с какого момента началась фиксация температуры почвы метеостанциями.

```
In [644]: df_tmp61.dropna(how='all').index.min() # удаляем сплошные NaNы, выводим минимум индекса
```

```
Out[644]: Timestamp('2013-03-10 09:00:00')
```

Получается, что с 2005 года до весны 2013 года наблюдения температуры почвы исследуемыми метеостанциями не фиксировалось.

Выясним в какие часы происходила фактическая фиксация температуры почвы

```
In [645...]: arr_moments = np.array(df_tmp61.dropna(how="all").index.hour.unique().sort_values())
arr_moments
```

```
Out[645]: array([ 3,  6,  9, 15], dtype=int64)
```

```
In [646...]: for moment in arr_moments: # по массиву моментов фиксации метеонаблюдений
    print(f'В {moment} час(а/ов) встречается '
          f'{df_tmp61[df_tmp61.index.hour == moment].dropna(how="all").count().sum().astype(int)} раз(а)')
```

```
В 3 час(а/ов) встречается 9 раз(а)
В 6 час(а/ов) встречается 8 раз(а)
В 9 час(а/ов) встречается 19205 раз(а)
В 15 час(а/ов) встречается 1 раз(а)
```

Составим список координат нестандартных моментов наблюдения температуры почвы.

```
In [647...]: # Выберем координаты ячеек с нестандартным временем фиксации наблюдений
list_error_coords = [] # список координат ячеек
# дополняем список кортежами (чтобы избежать лишнего вывода, присваиваем результат некой переменной)
dummy = (df_tmp61[df_tmp61.index.hour != 9] # выбираем нестандартное время фиксации температуры почвы
         .dropna(how='all') # удаляем сплошные NaN
         .apply(lambda x: list_error_coords.append(
             (x.name, [col[(len(PARAMETER61) + 1):] for col in x.dropna().index.tolist()])), # список станций
                axis=1 # получаем кортеж:(время фиксации, список столбцов)
```

```
)  
print(f'Всего найдено строк с нестандартными временем фиксации температуры почвы: {len(list_error_coords)}')
```

Всего найдено строк с нестандартными временем фиксации температуры почвы: 18

Проверим, зафиксированы ли в наблюдения в нестандартные моменты так же ещё и в стандартное время в диапазоне +/- 12 часов.

```
In [648...]  
for idx in list_error_coords: # по списку нестандартных моментов наблюдения  
    # получаем массив с индексами в диапазоне +/- 12 часов  
    arr_ix = (  
        df_tmp61[(df_tmp61.index >= (idx[0] - pd.Timedelta('12H'))) &  
                 (df_tmp61.index <= (idx[0] + pd.Timedelta('12H')))] # отбираем строки из df_tmp61  
        .dropna(how='all')  
        .index) # удаляем сплошные NaNы (если в стандартное время наблюдений нет - такая строка удалится)  
  
    if arr_ix.max() - arr_ix.min() == pd.Timedelta('0 days 00:00:00'): # Соседних строк с наблюдениями нет  
        print(f'У момента {idx} нет соседних стандартных моментов наблюдения')
```

У момента (Timestamp('2019-01-20 06:00:00'), ['V_Volochev']) нет соседних стандартных моментов наблюдения

Поскольку у нас есть единицы нестандартных моментов наблюдения без соседних стандартных моментов, их можно безболезненно удалить. Остальные нестандартные моменты подтверждены ближайшими стандартными и тоже могут быть безболезненно удалены.

```
In [649...]  
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_tmp61.at[error_coords[0], PARAMETER61+'#'+ station] = np.nan
```

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 сигмы.

```
In [650...]  
start_time = time.time() # для замера времени выполнения кода
```

```

df_tmp61 = df_tmp61.assign(field_out=df_tmp61.apply(lambda x: field_outliers(row=x,
                                                               method='sigma',
                                                               criterium=3,
                                                               IDW=True,
                                                               param=PARAMETER61,
                                                               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: 00:05:31

In [651]: `df_tmp61.dropna(subset=["field_out"]).sample(5, random_state=56)`

	<code>Soil_T#V_Volochek</code>	<code>Soil_T#Staritsa</code>	<code>Soil_T#Kashyn</code>	<code>Soil_T#Tver</code>	<code>Soil_T#Klin</code>	<code>Soil_T#Dmitrov</code>	<code>Soil_T#Volokolamsk</code>	<code>Soil_T#Mozhaisk</code>	<code>Soil_T#N_Jeru</code>
<code>2021-11-16 09:00:00</code>	NaN	NaN	NaN	NaN	-3.0	NaN	NaN	NaN	NaN
<code>2017-11-05 09:00:00</code>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<code>2021-11-09 09:00:00</code>	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN
<code>2014-11-02 09:00:00</code>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<code>2019-03-31 09:00:00</code>	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	

Проверим минимальные и максимальные значения выбросов в поле метеостанций.

In [652...]

```
# field_out содержит списки кортежей с характеристиками выбросов!
tup_out_stations = (
    df_tmp61.field_out.dropna().apply(lambda x: min([i[0] for i in x])).min(),
    df_tmp61.field_out.dropna().apply(lambda x: max([i[0] for i in x])).max()
)
tup_out_stations
```

Out[652]:

```
(-29.0, 26.0)
```

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 [653...]

```
# 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_tmp61.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: 104

Проверим максимальное количество выбросов в поле метеостанций на момент наблюдения.

In [654...]

```
# Находим максимальное количество выбросов в строке поля метеостанций
max_field_outliers = df_tmp61.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 названий станций с выбросами.

In [655...]

```
# 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_tmp61 = df_tmp61.assign(error_at =
    df_tmp61.
    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_=PARAMETER61)
) if isinstance(x.field_out, list) else np.nan,
axis=1
)
```

```
In [656]: df_tmp61.dropna(subset=["error_at"]).sample(5, random_state=56)
```

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

Для анализа выбросов во временном окне следует иметь в виду, что температура почвы измеряется 1 раз в день и не подвержена резким колебаниям. Поэтому целесообразно изучить выбросы во временном окне с интервалом в несколько дней.

Используем только один вариант поиска выбросов (для всех моментов наблюдения за одни сутки). Параметры:

- используем границы нормального распределения,

- не включаем проверяемое значение в подсчёт средней и сигмы,
- определим границы доверительного интервала в 95%,
- определим временное окно в +/- 5 дня ('5D'),
- включим в подсчёт моменты наблюдения только в 9 часов (equalhours=True).

Поскольку выбросы за один момент наблюдения при различных параметрах поиска могут существовать одновременно в нескольких метеостанциях, мы сформируем список координат выбросов и выведем их соответствующие значения в отдельную серию

In [657...]

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

# Нельзя удалять NaN в поле field_out непосредственно в df_tmp61 (Это приведёт к некорректным временным рядам!)
# Удалим NaN в столбце "field_out" в результирующем df

# Определим серию для записи списков с данными выбросов, индекс равен общему индексу архивов, тип данных - объект,
# название серии - будущее имя соответствующего столбца в DF
ser_time_out = pd.Series(index = df_tmp61.index, dtype='object', name="time_out_5D")

for elem in df_tmp61.dropna(subset=["error_at"]).error_at.tolist(): # поэлементно в списке значений столбца error_at
    # присваиваем элементу серии по соответствующему datetime индексу значение - пустой список
    ser_time_out.at[elem[0]] = []

    for station in elem[1]: # по метеостанциям, указанным в списке (2й элемент кортежа error_at)
        # Определяем вербальные координаты ячеек с выбросами: TimeStamp и название столбца, соответствующего станции:
        idxs = (elem[0], PARAMETER61+'#'+station)

        # Вызываем функцию time_outliers с обозначенными выше параметрами
        val_func = time_outliers(df=df_tmp61,
                                 # В качестве серии row_ передаём серию из одного значения: на момент наблюдения,
                                 # где индекс серии соответствует названию столбца (idxs[1])
                                 row_=df_tmp61.loc[idxs[0]][df_tmp61.loc[idxs[0]].index == idxs[1]],
                                 td_symbol_='D',
                                 td_quant_='5',
                                 equal_hours_=True,
                                 method_='norm',
                                 criterium_=0.95,
                                 inclusive_=False)

        # Присваиваем элементу серии по соответствующему datetime индексу результат вызова функции (список)
        if isinstance(val_func, float): # Если time_outliers вернула NaN (то есть значение float, то ничего не делаем
            pass
        else: # Иначе, если time_outliers вернула список
```

```
list_out = ser_time_out.at[idxs[0]] # Получаем список, находящийся в серии по индексу
list_out = list_out + val_func # Расширяем его за счёт вывода функции (контакенация)
ser_time_out.at[idxs[0]] = list_out # Записываем в серию обновлённый список

# ser_time_out.at[idxs[0]]

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: 00:00:00

Присоединим полученную серию к df_tmp61

```
In [658...]: # Индекс серии совпадает с индексом всех архивов, а значения в серии упорядочены по этому индексу.
# Поэтому произведём объединение по индексу
df_tmp61 = (df_tmp61
             .merge(ser_time_out, left_index=True, right_index=True) # производим объединение
             )
```

```
In [659...]: # Выводим случайные строки из df_tmp61, удалив NaN в столбце time_out_24H
df_tmp61.dropna(subset=["time_out_5D"]).sample(5, random_state=56)
```

Out[659]:

	Soil_T#V_Volochek	Soil_T#Staritsa	Soil_T#Kashyn	Soil_T#Tver	Soil_T#Klin	Soil_T#Dmitrov	Soil_T#Volokolamsk	Soil_T#Mozhaisk	Soil_T#N_Jeru
--	-------------------	-----------------	---------------	-------------	-------------	----------------	--------------------	-----------------	---------------

2021-11-16 09:00:00	NaN	NaN	NaN	NaN	-3.0	NaN	NaN	NaN
2017-11-05 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-11-09 09:00:00	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN
2014-11-02 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-03-31 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0



Заменим пустые списки в столбце time_out_5D на NaN

In [660...]

```
df_tmp61.time_out_5D = df_tmp61.time_out_5D.apply(lambda x: np.nan if x == [] else x)
```

In [661...]

```
df_tmp61.dropna(subset=["time_out_5D"]).sample(5, random_state=56)
```

Out[661]:

	Soil_T#V_Volochek	Soil_T#Staritsa	Soil_T#Kashyn	Soil_T#Tver	Soil_T#Klin	Soil_T#Dmitrov	Soil_T#Volokolamsk	Soil_T#Mozhaisk	Soil_T#N_Jeru
--	-------------------	-----------------	---------------	-------------	-------------	----------------	--------------------	-----------------	---------------

2020-11-17 09:00:00	NaN	NaN	NaN	NaN	-6.0	NaN	NaN	NaN
2013-03-27 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-04-20 09:00:00	-1.0	3.0	1.0	4.0	4.0	3.0	5.0	4.0
2014-11-09 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-12-21 09:00:00	NaN	NaN	NaN	NaN	6.0	NaN	NaN	NaN



Проверим минимальные и максимальные значения выбросов во временном окне.

In [662...]

```
# field_out содержит списки кортежей с характеристиками выбросов!
tup_out_time = (
    df_tmp61.time_out_5D.dropna().apply(lambda x: min([i[0] for i in x])).min(),
    df_tmp61.time_out_5D.dropna().apply(lambda x: max([i[0] for i in x])).max()
)
tup_out_time
print(f'Проверка равенства экстремальных выбросов в поле метеостанций и во временном окне: '
      f'{tup_out_time == tup_out_stations}')
```

Out[662]: (-29.0, 26.0)

Проверка равенства экстремальных выбросов в поле метеостанций и во временном окне: True

Применимость критериев ошибочных значений

Рассмотрим полученные данные об аномальных значениях.

In [663...]

```
# 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_tmp61 = (df_tmp61
    .assign(double_error_at =
        df_tmp61.dropna(subset=["time_out_5D"])
        .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_= 'time_out_5D', # используем выбросы во временном окне
                    param_=PARAMETER61)
                 ) if isinstance(x.field_out, list) else np.nan,
               axis=1
            )
        )
    )
```

In [664... df_tmp61.dropna(subset=["double_error_at"])

Out[664]:

Soil_T#V_Volochek Soil_T#Staritsa Soil_T#Kashyn Soil_T#Tver Soil_T#Klin Soil_T#Dmitrov Soil_T#Volokolamsk Soil_T#Mozhaisk Soil_T#N_Jeru

Найдём общие выбросы как в поле метеостанций, так и во временном окне. (У нас могут быть случаи, когда при проверке выброса в поле метеостанций, выбросов для того же значения во временном окне не обнаруживается).

Для контроля, найдём пересечение списков в столбцах error_at и double_error_at, выведем его в отдельный столбец cross_check

```
In [665...]: df_tmp61 = df_tmp61.assign(  
    cross_check = df_tmp61  
        .dropna(subset=["error_at", "double_error_at"])  
        .apply(  
            lambda x: list(set(x.error_at[1]) & set(x.double_error_at[1])),  
            axis=1  
        )  
    )
```

```
In [666...]: df_tmp61.dropna(subset=["cross_check"]).sample(7, random_state=56)
```

Out[666]:

	Soil_T#V_Volochek	Soil_T#Staritsa	Soil_T#Kashyn	Soil_T#Tver	Soil_T#Klin	Soil_T#Dmitrov	Soil_T#Volokolamsk	Soil_T#Mozhaisk	Soil_T#N_Jeru
2020-11-17 09:00:00	NaN	NaN	NaN	NaN	-6.0	NaN	NaN	NaN	NaN
2013-03-27 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-04-20 09:00:00	-1.0	3.0	1.0	4.0	4.0	3.0	5.0	4.0	
2014-11-09 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-12-21 09:00:00	NaN	NaN	NaN	NaN	6.0	NaN	NaN	NaN	NaN
2013-10-14 09:00:00	NaN	NaN	NaN	NaN	-1.0	NaN	NaN	NaN	NaN
2021-11-30 09:00:00	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN

In [667...]

```
# Подсчитаем количество выбросов в поле метеостанций
count_field_out = (df_tmp61
    .error_at # по столбцу error_at
    .dropna()
    .apply(lambda x: len(x[1]) # вычислим длины списков с перечнем метеостанций с выбросами
          )
    ).sum() # просуммируем их в общий итог

# Подсчитаем из них количество выбросов во временном окне
count_time_out = (df_tmp61
    .double_error_at # по столбцу double_error_at
    .dropna()
    .apply(lambda x: len(x[1]) # вычислим длины списков с перечнем метеостанций с выбросами
          )
```

```

        )
    ).sum() # просуммируем их в общий итог

# Подсчитаем из них количество выбросов в контрольном столбце
count_cross_out = (df_tmp61
    .cross_check # по столбцу double_error_at
    .dropna()
    .apply(lambda x: len(x) # вычислим длины списков с перечнем метеостанций с выбросами
          )
    ).sum() # просуммируем их в общий итог

print(f'В поле метеостанций выявлено аномальных значений: {count_field_out}, из них:\n'
      f'Подтверждается аномалиями в промежутке +/- 5 дней по всем моментам наблюдения: '
      f'{count_time_out}\n'
      f'Проверка: пересечение множеств выбросов в поле метеостанций и во временном окне насчитывает '
      f'{count_cross_out} элемент(ов)')

```

В поле метеостанций выявлено аномальных значений: 120,

из них:

Подтверждается аномалиями в промежутке +/- 5 дней по всем моментам наблюдения: 22

Проверка: пересечение множеств выбросов в поле метеостанций и во временном окне насчитывает 22 элемент(ов)

Определим конечный критерий ошибочных значений:

1. Аномальное значение выявлено в поле метеостанций И ПРИ ЭТОМ

А. Аномальное значение выявлено во временном ряду в промежутке +/- 5 дней (выбивается из общей тенденции изменения температуры почвы)

Удалим (заменим на NaN) все ошибочные значения

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

In [668]: df_tmp61.dropna(subset=['cross_check'])

Out[668]:

Soil_T#V_Volochek Soil_T#Staritsa Soil_T#Kashyn Soil_T#Tver Soil_T#Klin Soil_T#Dmitrov Soil_T#Volokolamsk Soil_T#Mozhaisk Soil_T#N_Jeru

In [669...]

```
# Создадим список координат ячеек, содержащих значения, определённые как ошибки
# list_error_coords = df_tmp61.dropna(subset=['double_error_at']).double_error_at.tolist()
# list_error_coords

# Добавим список координат ячеек, содержащих значения, определённые как ошибки
list_error_coords.extend(df_tmp61.dropna(subset=['double_error_at']).double_error_at.tolist())
#list_error_coords
```

In [670...]

```
# Восстановим исходное состояние df_tmp61
df_tmp61 = dict_df_parameters[param_df_name].copy(deep=True)
```

In [671...]

```
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_tmp61.at[error_coords[0], PARAMETER61+'#'+ station] = np.nan
```

In [672...]

```
# Проверим, все ли ошибки заменены на NaN
# Количество нeNaN значений в df_tmp61 по координатам, указанным в списках list_error_coords и list_error_at

counter_notna = 0 # счётчик нeNaN значений
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
        # подсчитаем количество нeNaN значений и прибавим их к счётчику нeNaN значений.
        counter_notna += np.sum(pd.notna(df_tmp61.at[error_coords[0], PARAMETER61+'#'+ station]))
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna}')

# df_tmp61
```

По результатам удаления выявленных аномальных выбросов осталось значений: 0

Визуализируем архив температуры почвы (Soil_T) по сезонам после удаления ошибок

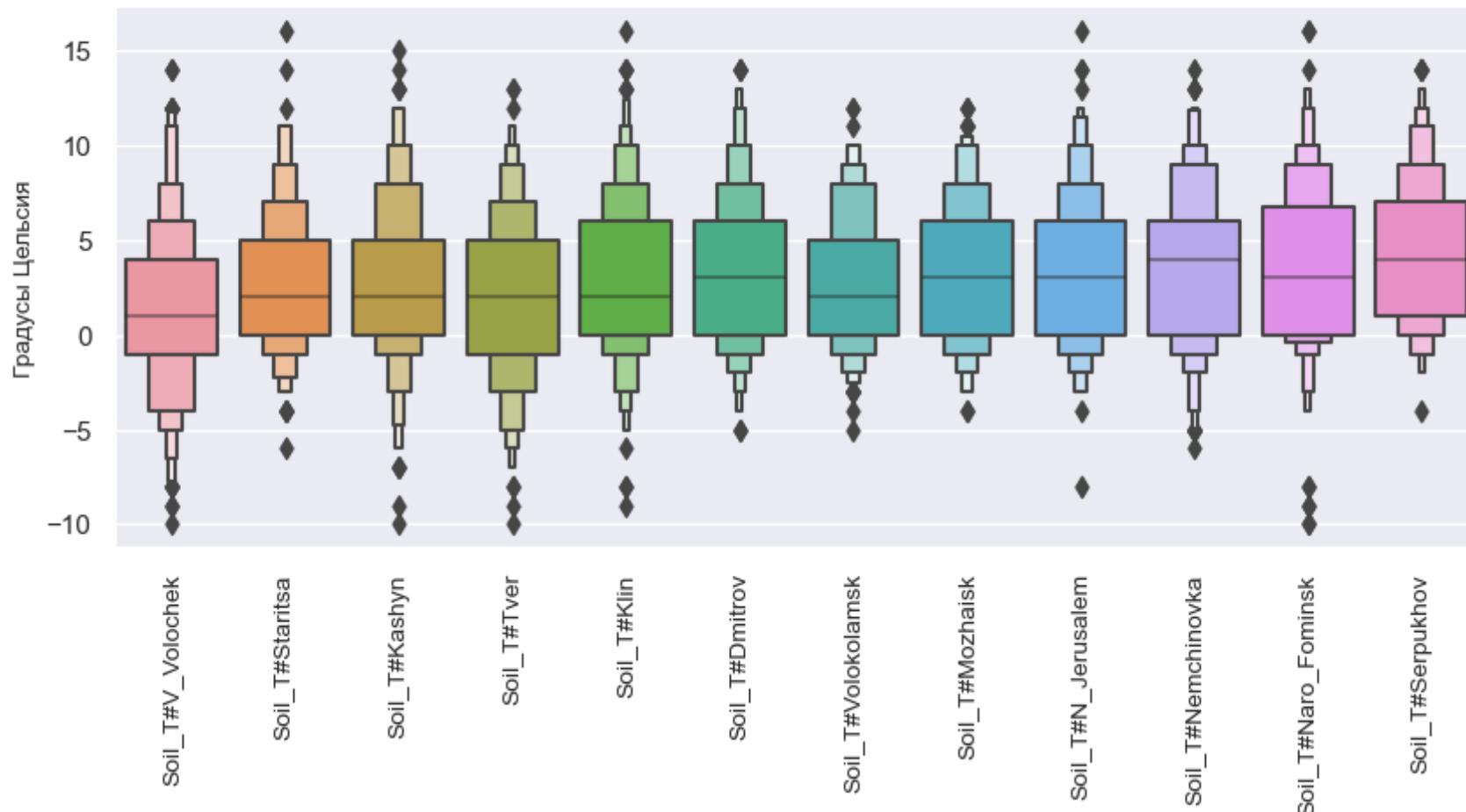
In [673...]

```
# В цикле выведем графики давления по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp61).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp61[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Градусы Цельсия', size=10)
    dummy = g.set_title(f'Распределение значений Soil_T в разрезе метеостанций после удаления ошибок:\n'
```

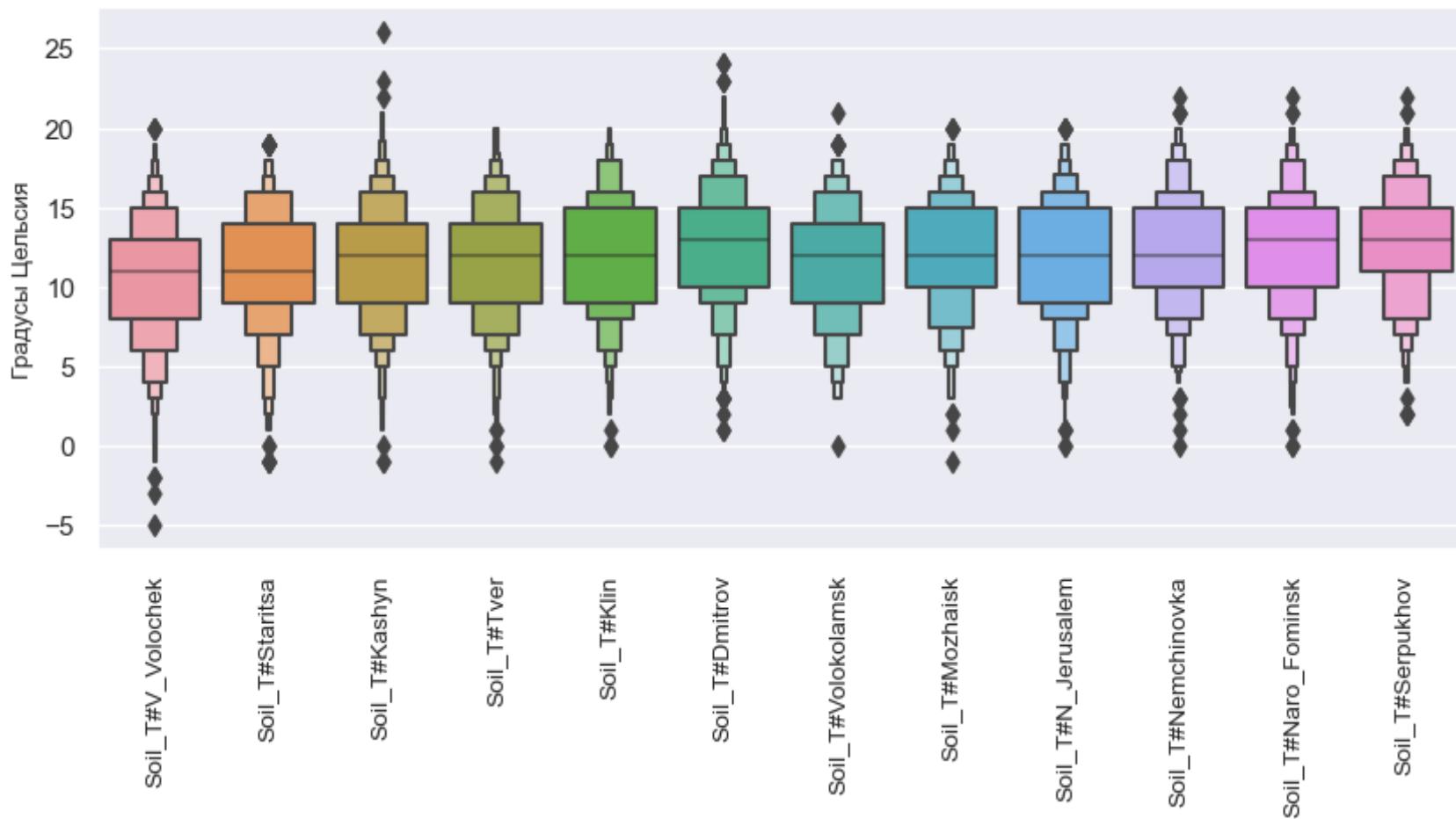
```
f'{season_name}')
```

```
plt.show()
```

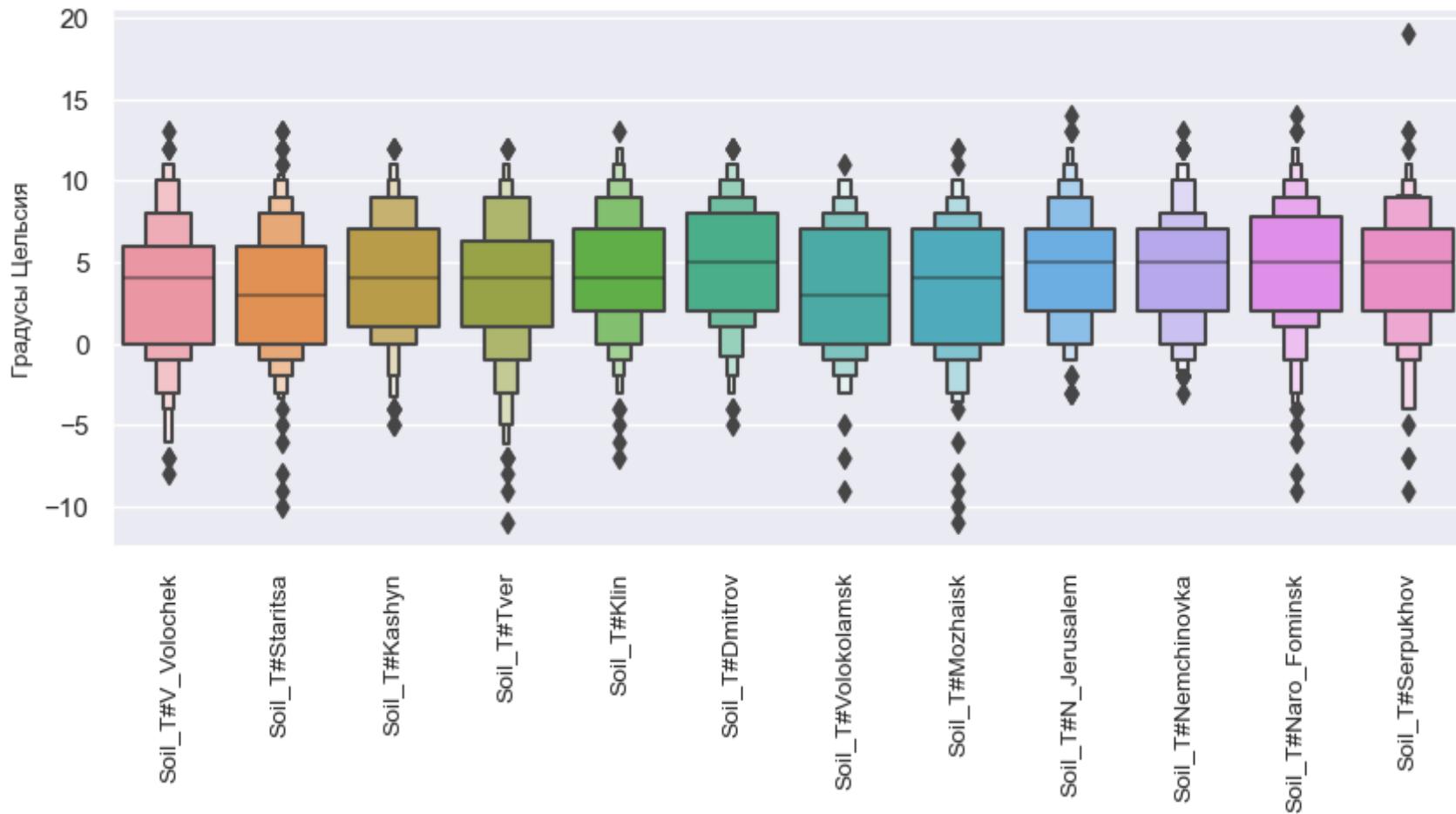
Распределение значений Soil_T в разрезе метеостанций после удаления ошибок:
spring



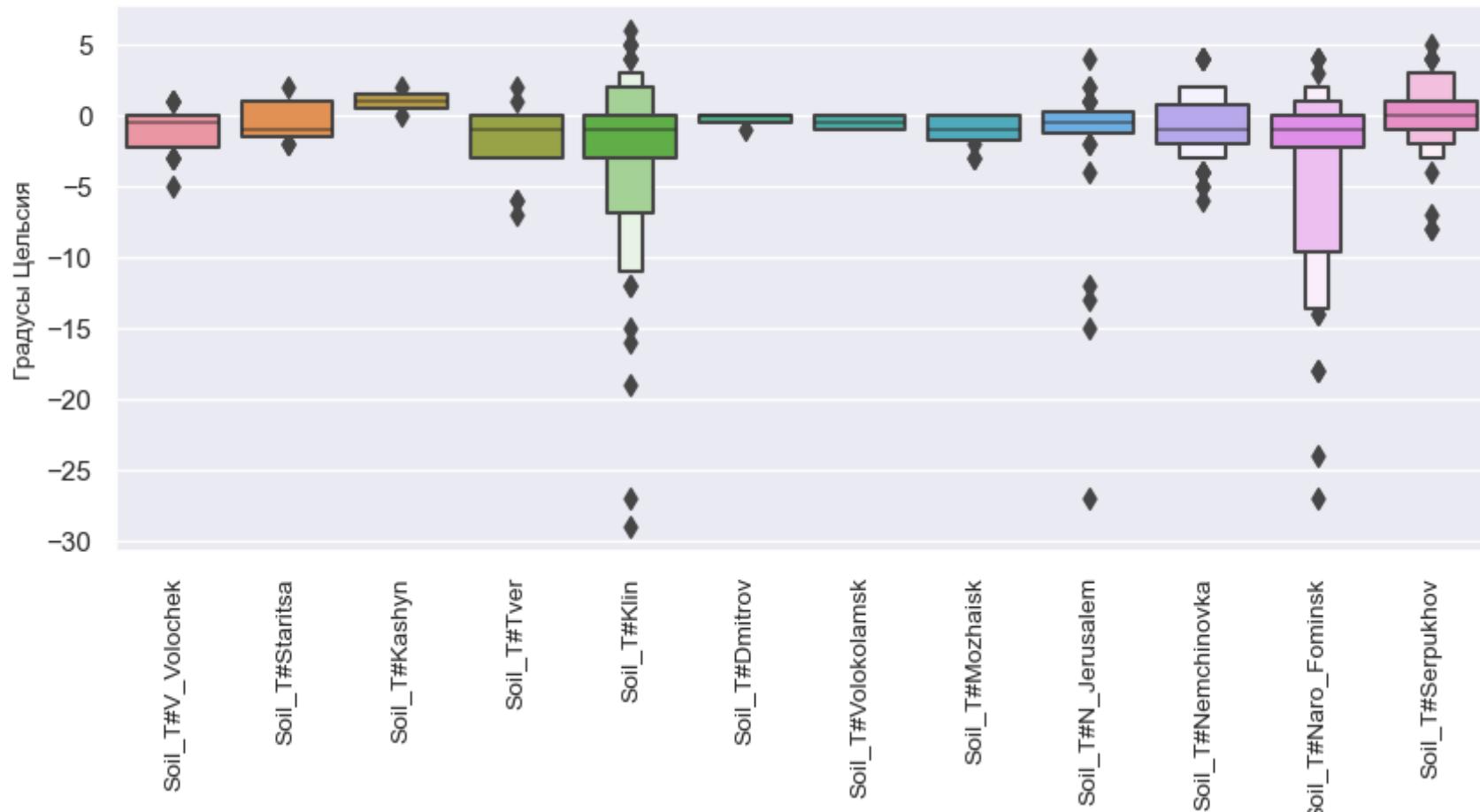
Распределение значений Soil_T в разрезе метеостанций после удаления ошибок:
summer



Распределение значений Soil_T в разрезе метеостанций после удаления ошибок:
autumn



Распределение значений Soil_T 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 [674]: print(f'Минимальное значение: {np.nanmin(df_tmp61)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp61)},\n'
      f'Средняя: {np.nanmean(df_tmp61)},\n'
      f'Медиана: {np.nanmedian(df_tmp61)}')
```

Минимальное значение: -29.0,
Максимальное значение: 26.0,
Средняя: 8.124120314862118,
Медиана: 9.0

Сохраним очищенные данные в файл параметров

In [675...]

```
# # 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_'+PARAMETER61 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER61].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER61}.csv'  
)  
print('DONE!')
```

df_Soil_T.csv -> DONE!

6.1.2. Восстановление "сплошных" NaN методом средней между соседними моментами наблюдения для показателя температуры почвы Soil_T

Модели пространственной экстраполяции не могут экстраполировать значения в рамках одного момента наблюдения, если значения во всех точках наблюдения неизвестны. Есть ли такие случаи в архиве температуры почвы (в периоды, когда такие наблюдения проводились)?

In [676...]

```
# Определим временный DF для стандартных моментов наблюдения и с учётом того,  
# что зимой наблюдения не фиксируются,  
# и с учётом начала фиксации этих данных в архивах  
df_tmp61d = df_tmp61[(df_tmp61.index.hour == 9) & (df_tmp61.index >= '2013-03-10 09:00:00')]  
  
# Подсчитаем количество строк со "сплошными" NaN в df_tmp61d  
# построчно подсчитаем сумму количества NaN,
```

```
# и если оно количество столбцов в DF (boolean), подсчитаем через сумму количество таких строк
all_nans_count = sum(df_tmp61d.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp61d.keys())))
print(f'Количество дней со сплошными NaN равно {all_nans_count}')
```

Количество дней со сплошными NaN равно 1280

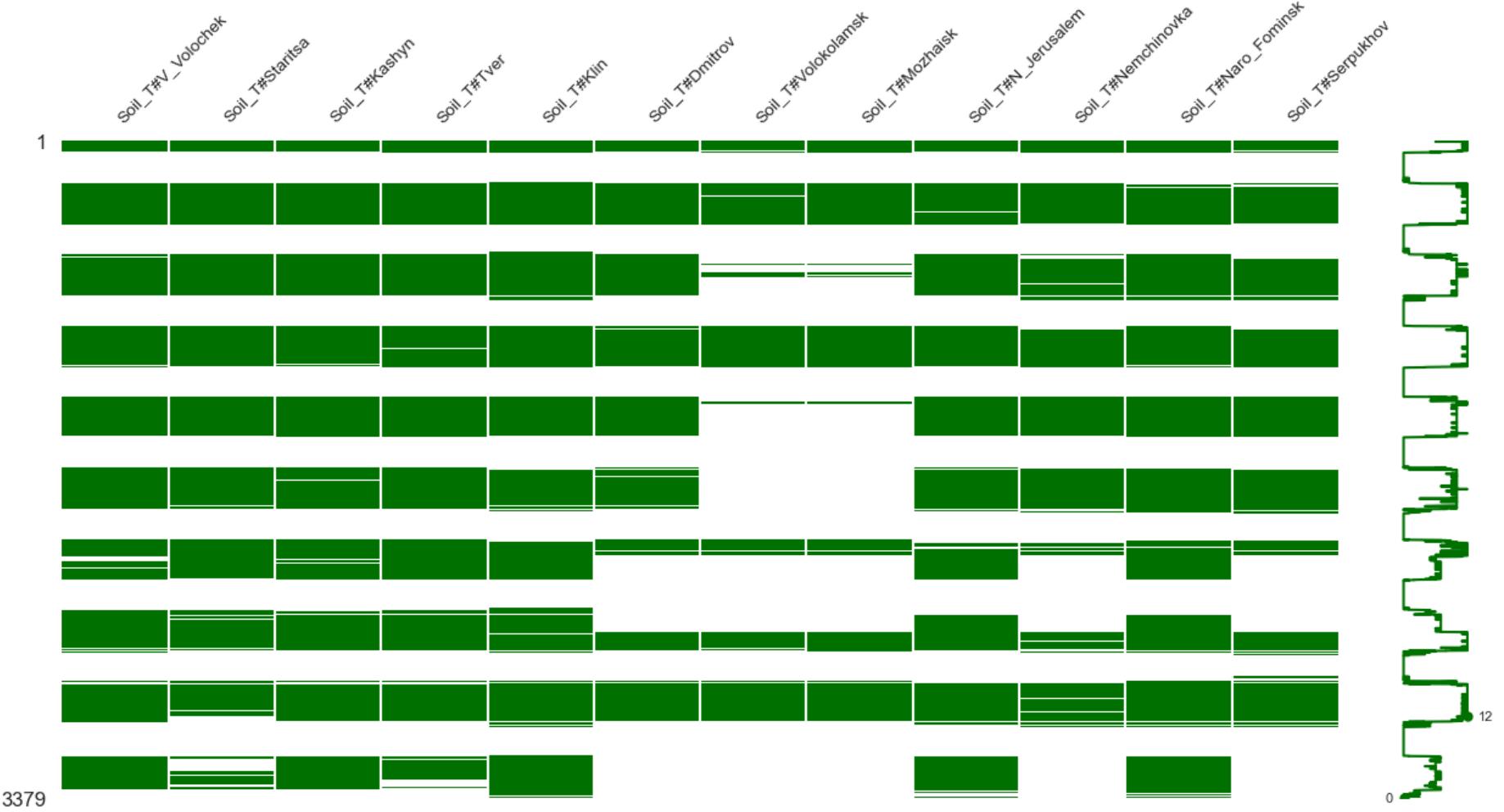
Визуализируем структуру пропущенных данных о температуре почвы.

In [677...]

```
msno.matrix(df_tmp61d,
            color=(0, 100/225, 0),
            figsize=(15, 7),
            fontsize=10); # выводим график из библиотеки "missingno"
plt.show()
```

Out[677]:

<AxesSubplot:>



Из графика следует, что наблюдения имеют сезонный характер. В зимний период наблюдения носят эпизодический характер. В середине зимы наблюдений нет. Проверим количество наблюдений, приходящийся на каждый сезон.

In [678...]

```

for season, mask in season_masks(df_tmp61d).items():
    print(f'Количество наблюдений за температурой почвы. Сезон: {season}')
    pd.notna(df_tmp61d[mask]).sum() # количество неNaN значений в разбиении по сезонам
    msno.matrix(
        df_tmp61d[mask],
        color=(10/225, 10/225, 150/225),
        figsize=(15, 5),
    )

```

```
    fontsize=10); # выводим график из библиотеки "missingno"
plt.show()
```

Количество наблюдений за температурой почвы. Сезон: spring

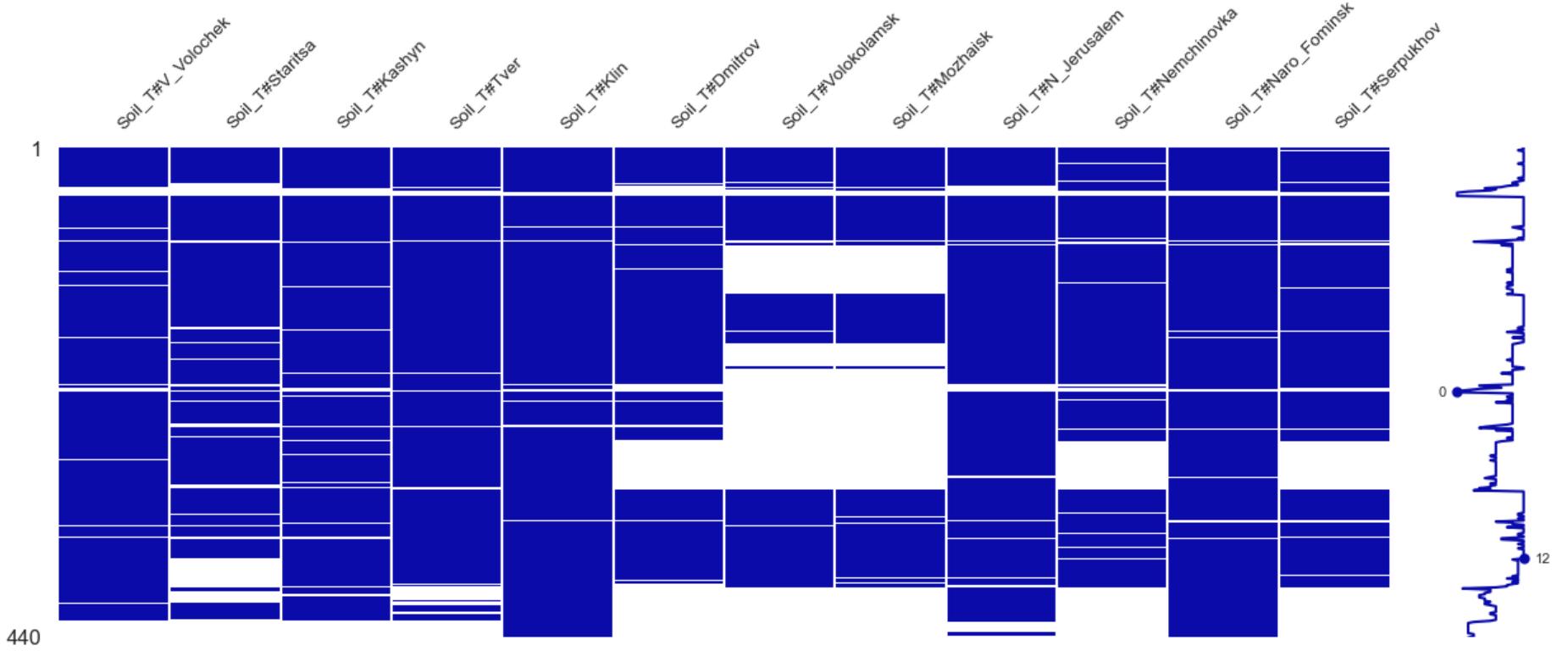
```
Out[678]:
```

Soil_T#V_Volochek	401
Soil_T#Staritsa	349
Soil_T#Kashyn	394
Soil_T#Tver	391
Soil_T#Klin	424
Soil_T#Dmitrov	320
Soil_T#Volokolamsk	208
Soil_T#Mozhaisk	209
Soil_T#N_Jerusalem	400
Soil_T#Nemchinovka	326
Soil_T#Naro_Fominsk	422
Soil_T#Serpukhov	330

dtype: int64

<AxesSubplot:>

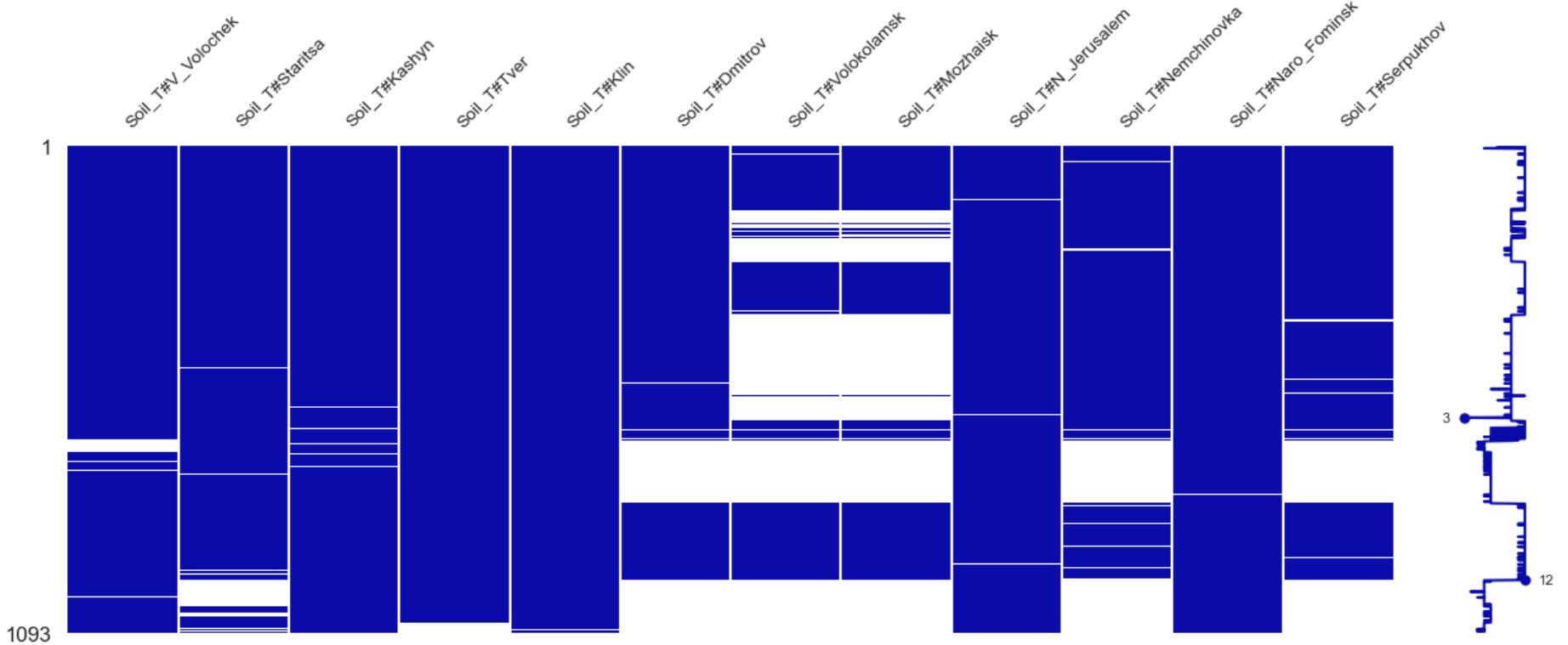
```
Out[678]:
```



Количество наблюдений за температурой почвы. Сезон: summer

```
Out[678]: 
Soil_T#V_Volochek      1054
Soil_T#Staritsa         1016
Soil_T#Kashyn           1076
Soil_T#Tver              1061
Soil_T#Klin              1088
Soil_T#Dmitrov          823
Soil_T#Volokolamsk      489
Soil_T#Mozhaisk          493
Soil_T#N_Jerusalem       1087
Soil_T#Nemchinovka      809
Soil_T#Naro_Fominsk      1089
Soil_T#Serpukhov          813
dtype: int64
```

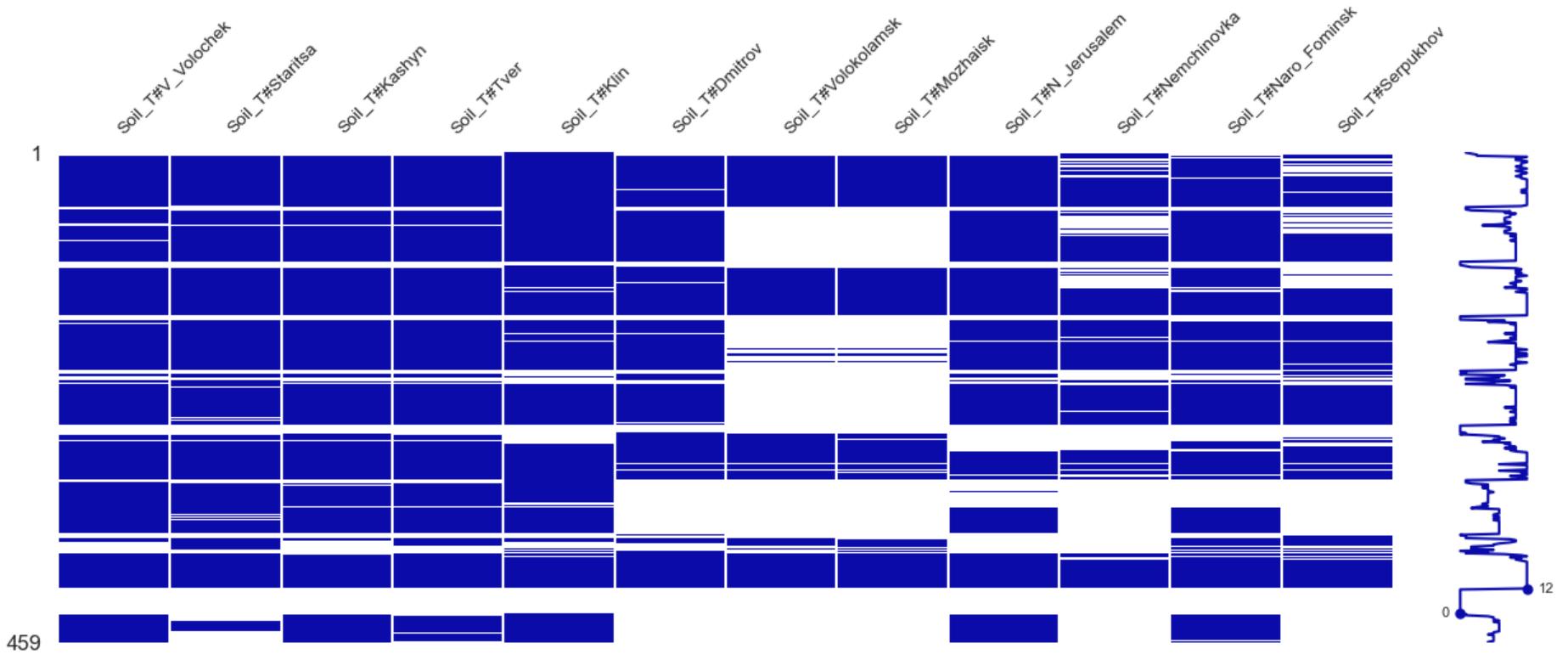
```
Out[678]: <AxesSubplot:>
```



Количество наблюдений за температурой почвы. Сезон: autumn

```
Out[678]: 
Soil_T#V_Volochev      375
Soil_T#Staritsa        360
Soil_T#Kashyn          371
Soil_T#Tver            376
Soil_T#Klin             377
Soil_T#Dmitrov         308
Soil_T#Volokolamsk    177
Soil_T#Mozhaisk        176
Soil_T#N_Jerusalem     335
Soil_T#Nemchinovka    238
Soil_T#Naro_Fominsk   342
Soil_T#Serpukhov       256
dtype: int64
```

```
Out[678]: <AxesSubplot:>
```



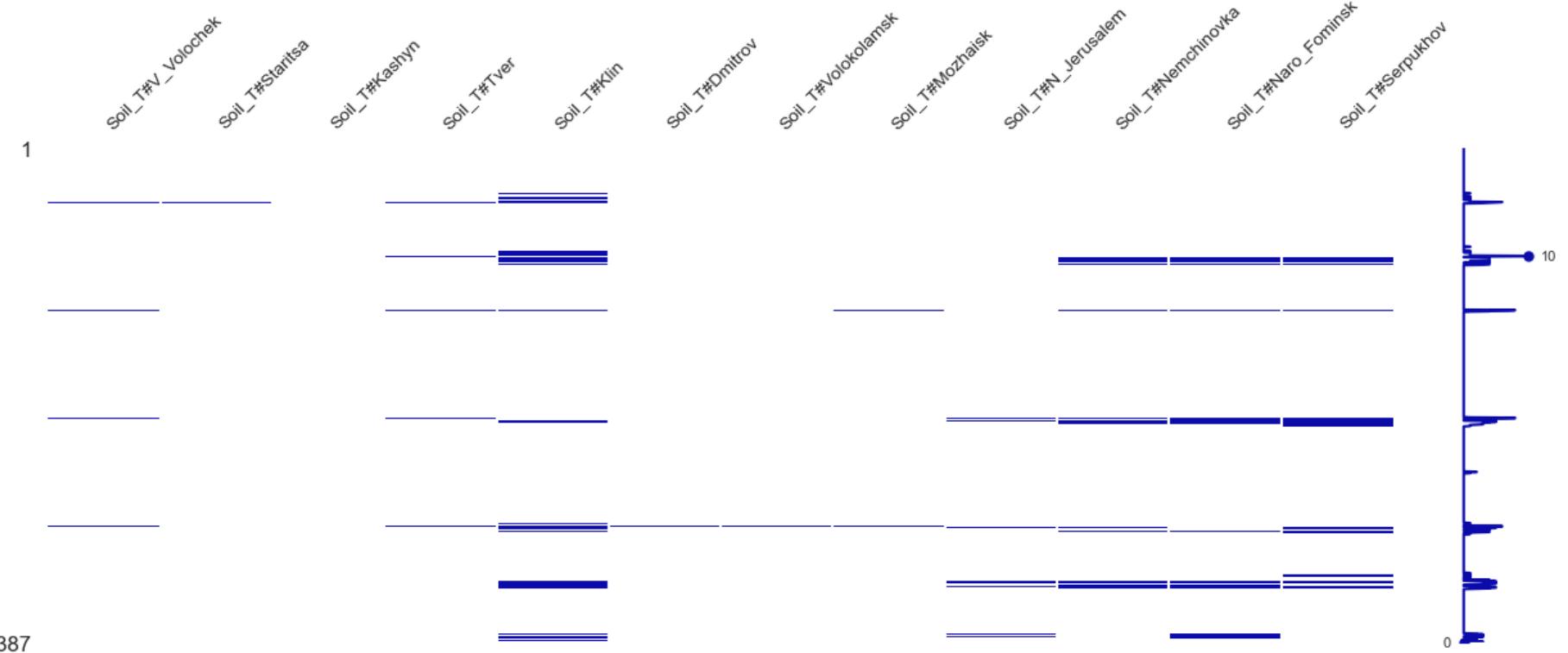
Количество наблюдений за температурой почвы. Сезон: winter

Out[678]:

Soil_T#V_Volochek	16
Soil_T#Staritsa	7
Soil_T#Kashyn	2
Soil_T#Tver	14
Soil_T#Klin	114
Soil_T#Dmitrov	3
Soil_T#Volokolamsk	4
Soil_T#Mozhaisk	10
Soil_T#N_Jerusalem	28
Soil_T#Nemchinovka	58
Soil_T#Naro_Fominsk	84
Soil_T#Serpukhov	80

dtype: int64

<AxesSubplot:>



Заменим строки сплошных NaN средними значениями из соседних моментов наблюдения, кроме периода отсутствия наблюдений зимой.

In [679...]

```
fix_nan_counter = 0 # счётчик исправленных строк
for year in df_tmp61d.index.year.unique(): # по перечню уникальных лет в df_tmp61d, по годам
    # определим логическую маску: соответствие данному году, и не менее 1 неNaN значения в ряду
    yearly_mask = (df_tmp61d.index.year == year) & (df_tmp61d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))

    # Определим границы периодов наблюдения
    start_moment = (
        df_tmp61d[yearly_mask]
        .dropna(how='all')
        .index
        .min()
    ) # минимальный момент
    end_moment = (
        df_tmp61d[yearly_mask]
```

```

    .dropna(how='all')
    .index
    .max()
) # максимальный момент

# Список: выбираем из датафрейма строки, где количество NaN равно количеству столбцов - то есть сплошные NaN,
# если эти строки находятся внутри промежутка наблюдений для каждого года
list_time_total_nans = (
    df_tmp61d[(df_tmp61d.index >= start_moment) &
               (df_tmp61d.index <= end_moment) &
               (df_tmp61d.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp61d.keys())))
)
.list_time_total_nans
    .index.tolist()
)
# year, start_moment, end_moment
# list_time_total_nans
f'{year}: {start_moment} to {end_moment}'

# По списку
for idx in list_time_total_nans:
    # определяем начало и конец временного интервала
    start_time = idx - pd.Timedelta('1D')
    end_time = idx + pd.Timedelta('1D')

    # Пока в двух строках нет значений одновременно хотя бы в одном общем столбце
    # Проверяем, какое количество элементов содержит пересечение индекса серии start_date без NaN и
    # индекса серии end_date без NaN, и пока оно не будет содержать хотя бы один элемент сдвигаем границы.
    # (np.isin выдает булев массив для каждого элемента в первом списке, содержит ли он во втором списке)
    while (
        (np.isin(df_tmp61d.loc[start_time].dropna().index,
                 df_tmp61d.loc[end_time].dropna().index).sum() == 0) |
        (df_tmp61d.loc[start_time].notna().sum() == 0) |
        (df_tmp61d.loc[end_time].notna().sum() == 0)
    ): # границы периода не должны быть сплошными NaN

        if start_time > start_moment: # Пока не дошли до начала наблюдений данного года
            start_time = start_time - pd.Timedelta('1D') # Уменьшаем start_time на 1 день

        if end_time < end_moment: # Пока не дошли до оконания наблюдений данного года
            end_time = end_time + pd.Timedelta('1D') # Увеличиваем end_time на 1 день

    # по ряду сплошных NaN заменяем NaN в df_tmp61d на средние значения их соседних двух рядов
    for label in df_tmp61d.loc[idx].index:
        df_tmp61d.at[idx, label] = np.mean([df_tmp61d.at[start_time, label], df_tmp61d.at[end_time, label]])

```

```
#         df_tmp61.loc[idx]
fix_nan_counter += len(list_time_total_nans) # увеличиваем счётчик исправленных слошных NaN

Out[679]: '2022: 2022-04-09 09:00:00 to 2022-06-09 09:00:00'
Out[679]: '2021: 2021-04-01 09:00:00 to 2021-11-29 09:00:00'
Out[679]: '2020: 2020-03-10 09:00:00 to 2020-11-30 09:00:00'
Out[679]: '2019: 2019-03-31 09:00:00 to 2019-10-31 09:00:00'
Out[679]: '2018: 2018-04-07 09:00:00 to 2018-10-31 09:00:00'
Out[679]: '2017: 2017-03-12 09:00:00 to 2017-11-05 09:00:00'
Out[679]: '2016: 2016-03-31 09:00:00 to 2016-10-28 09:00:00'
Out[679]: '2015: 2015-03-13 09:00:00 to 2015-11-12 09:00:00'
Out[679]: '2014: 2014-03-10 09:00:00 to 2014-11-23 09:00:00'
Out[679]: '2013: 2013-03-10 09:00:00 to 2013-10-12 09:00:00'
```

```
In [680... df_tmp61.update(df_tmp61d) # обновим значения df_tmp61 за счёт значений df_tmp61d
```

```
print(f'Изначальное количество дней со сплошными NaN равно {all_nans_count}')
# Снова, по результатам исправления
# подсчитаем количество строк со "сплошными" NaN в df_tmp61d
# построчно подсчитаем сумму количества NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(df_tmp61d.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp61d.keys())))
print(f'Оставшееся количество дней со сплошными NaN равно {all_nans_count}')
print(f'Исправлено дней со сплошным NaN {fix_nan_counter}')
print(f'{all_nans_count - fix_nan_counter} дней с отсутствием наблюдений приходится на зимний период')
```

Изначальное количество дней со сплошными NaN равно 1280
Оставшееся количество дней со сплошными NaN равно 1213
Исправлено дней со сплошным NaN 67
1146 дней с отсутствием наблюдений приходится на зимний период

Проверим, на какие месяцы оставшиеся сплошные NaN

```
In [682... set(df_tmp61[(df_tmp61.index.hour==9) &
(df_tmp61.index >= '2013-03-10 09:00:00') &
```

```
(df_tmp61.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp61d.keys())))
.index.month
.tolist()
)
Out[682]: {1, 2, 3, 4, 10, 11, 12}
```

6.1.3. Подбор модели для восстановления пропущенных и удалённых значений показателя температуры почвы, а также для моделирования значений для искомой точки

6.1.3.1. Модель средней, взвешенной по степени обратных расстояний (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 степень 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_tmp71.

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 [683]:

```
# Создаём датафрейм для валидации модели IDW
df_test = df_tmp61.dropna(how='any')
```

In [684]:

```
df_test.info()
df_test.shape

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 704 entries, 2022-06-08 09:00:00 to 2014-05-02 09:00:00
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   Soil_T#V_Volochek    704 non-null   float64 
 1   Soil_T#Staritsa     704 non-null   float64 
 2   Soil_T#Kashyn       704 non-null   float64 
 3   Soil_T#Tver          704 non-null   float64 
 4   Soil_T#Klin          704 non-null   float64 
 5   Soil_T#Dmitrov      704 non-null   float64 
 6   Soil_T#Volokolamsk   704 non-null   float64 
 7   Soil_T#Mozhaisk      704 non-null   float64 
 8   Soil_T#N_Jerusalem   704 non-null   float64 
 9   Soil_T#Nemchinovka   704 non-null   float64 
 10  Soil_T#Naro_Fominsk  704 non-null   float64 
 11  Soil_T#Serpukhov     704 non-null   float64 
dtypes: float64(12)
memory usage: 71.5 KB
(704, 12)
```

Out[684]:

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

In [685]:

```
# 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[685]: array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
       13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
       26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
       39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
       52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
       65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
       78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
       91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
      104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
      117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
      130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
      143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
      156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
      169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
      182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
      195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
      208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
      221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
      234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
      247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,
      260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,
      273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
      286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
      299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311,
      312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
      325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337,
      338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
      351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,
      364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,
      377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
      390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,
      403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,
      416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
      429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
      442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
      455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,
      468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
      481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
      494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506,
      507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
      520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532,
      533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545,
      546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,
      559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571,
```

572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703])

```
Out[685]: array([ 5,  4,  0,  2, 11, 10,  9, 11,  7,  6,  4,  9, 10,  7,  1,  8,  2,
   11, 11,  0,  5,  6,  1,  9, 10,  5,  5,  2,  9,  3,  5,  9,  2, 10,
   1,  0,  4,  6,  2,  0,  8,  6,  4,  0,  1, 11,  1,  3,  9,  6, 10,
   2,  1,  3,  8,  8,  3,  2,  0,  3,  0,  5,  8,  0,  8,  6, 10,  0,
   9,  4, 11, 10,  6,  0, 10,  2,  3,  7,  1, 11, 11,  6,  5,  0, 11,
   6,  8,  2,  8,  7,  6,  0,  3,  4,  2,  7,  2,  4,  5,  0,  7, 10,
   1,  1,  8,  6,  7,  4,  1,  7,  8,  4, 11,  3,  6,  2, 10,  5,  0,
   8,  0,  5,  3,  7,  8,  7,  0, 10, 10,  9, 11,  1,  3,  5,  3,  2,
   5,  1,  5,  1, 11, 10,  4,  3,  9,  7,  6,  7,  7,  0,  4,  8,  9,
   0,  1, 11,  1,  3,  6,  7,  7,  5,  2,  6,  0,  7, 11,  7,  9,  8,
   3,  3,  8,  8,  8,  9, 11,  4, 11,  5, 10,  1,  5, 10,  8, 10,  2,
   0,  8,  7,  1,  7,  6,  6,  9,  3,  2,  5,  5,  4,  9,  3,  2,  2,
   9, 10,  3,  0,  4, 11,  5,  8,  3,  5,  0,  0,  9, 11,  0,  8, 11,
   9,  5,  0,  5,  3,  5,  6,  2,  4,  0,  8,  8,  6,  7,  0,  1,  1,
   5,  6,  4,  4,  9,  8,  5,  1,  1,  9, 10,  6,  9,  0,  6,  3, 11,
   11, 11,  9,  9,  7,  3,  6,  0,  3,  4,  5,  7,  2, 10,  4,  8,  2,
   10,  6,  6, 10,  5,  2, 11,  0,  5,  7,  3,  3, 10,  9,  0, 10,  0,
   9,  0, 11,  3,  2,  4,  0,  7, 10, 10,  7,  1,  4, 10,  3,  0,  4,
   10,  7, 11,  6,  8,  2,  0,  8,  3,  1,  9,  5,  0,  4,  9,  7,  0,
   6, 11,  1,  7,  4,  5,  6,  7,  8,  4,  4,  3,  4,  6,  0,  5,  5,
   6,  9,  9,  8,  7,  1,  8, 10, 10,  9,  8,  7,  4, 10,  8,  1,  4,
   5,  7,  8,  5, 10,  7, 10, 10,  8,  4,  2,  7, 10,  7,  8,  7, 10,
   9,  7,  6,  8,  2,  2,  9,  7,  4,  7,  3,  3,  4, 10,  9,  5,  3,
   11,  8,  4,  6,  2,  8,  6,  1,  9,  4, 11,  7,  0,  1,  3,  8,  7,
   7,  7,  3,  4,  7,  7, 10,  9, 10, 10, 11,  4,  6, 10, 10,  0, 10,
   9,  7,  2,  8,  2,  8,  3,  3,  5, 11,  1,  7,  9,  4,  6, 11,  6,
   3,  7,  5,  9,  4,  5,  4,  9,  1, 10,  2,  6,  8, 10, 11,  1,  9,
   0,  2,  5,  7,  2,  5,  2, 11,  0,  7,  7,  4,  3,  2,  0,  4, 10,
   4,  9,  2,  0,  7,  5, 10, 11, 10,  1,  2,  8,  5, 11,  2,  5,  1,
   8,  0,  3,  6,  7,  2,  9,  6,  7,  4,  9,  7,  1,  5,  6,  0,  5,
   4,  7,  1,  8,  3,  1,  0,  9, 10,  5,  8, 10,  6,  6,  8,  0,  8,
   0, 11,  8,  7,  2,  5,  7,  9,  1,  2,  9, 10, 10,  4,  3, 11,  3,
   0,  7,  2,  6,  8, 11,  5,  2,  2, 11, 11,  8,  7,  0,  8,  9,  0,
   2, 10, 11,  3,  3,  8, 10,  3,  9,  6,  8, 10,  1,  8,  8,  2,  5,
   8,  4,  7,  0,  0,  4,  3,  6,  8,  9,  6,  3,  8,  6,  4,  5,  8,
   1,  9,  7,  3,  4,  0,  7,  8,  7,  1,  6,  4,  1,  0,  3,  0,  8,
   7,  5,  8, 10,  2,  5,  6,  0,  4,  7,  6,  9,  2,  1,  2,  3, 11,
   2,  5,  1, 10,  0,  9,  5,  6,  4,  3,  0, 10,  9,  0,  5,  6, 10,
   11, 11, 11,  7,  1, 10,  0,  0,  0,  7,  6,  3,  3,  8,  9,  7,  4,
   11,  4, 10,  7,  1,  3,  2,  8,  1,  4,  2,  8, 11,  3,  5,  0, 11,
   11,  8,  4,  3,  9, 10,  6,  1,  9,  8,  0,  3,  5,  0,  3,  5,  2,
   6,  7,  6,  4, 11, 10,  6])
```

```
Out[685]: array([[ 0,  5],
   [ 1,  4],
   [ 2,  0],
   ...,
  [701, 11],
  [702, 10],
  [703,  6]])
```

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 [686]

```
# 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[686]: (422, 2)

Out[686]: (422,)

Out[686]: (282, 2)

Out[686]: (282,)

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

In [687]

```
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=PARAMETER61,
                             station=df_test.keys()[y][len(PARAMETER61)+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=00:00:01

Текущие значения: iterations=1, power=3.25,
Максимальная ошибка (MAX_E) IDW = 6.9634850
Средняя абсолютная ошибка (MAE) IDW = 0.9925245
Средний квадрат ошибки (MSE) IDW = 1.7446693
Средняя квадратическая ошибка (RMSE) IDW = 1.3208593
Коэффициент детерминации (R2) IDW = 0.9377220

Elapsed time=00:00:03

Текущие значения: iterations=2, power=3.5,
Максимальная ошибка (MAX_E) IDW = 6.9714008
Средняя абсолютная ошибка (MAE) IDW = 0.9918982
Средний квадрат ошибки (MSE) IDW = 1.7410028
Средняя квадратическая ошибка (RMSE) IDW = 1.3194706
Коэффициент детерминации (R2) IDW = 0.9378528

Elapsed time=00:00:06

Текущие значения: iterations=3, power=3.75,
Максимальная ошибка (MAX_E) IDW = 6.9792872
Средняя абсолютная ошибка (MAE) IDW = 0.9921267
Средний квадрат ошибки (MSE) IDW = 1.7404704
Средняя квадратическая ошибка (RMSE) IDW = 1.3192689
Коэффициент детерминации (R2) IDW = 0.9378718

Elapsed time=00:00:08

Текущие значения: iterations=4, power=4.0,
Максимальная ошибка (MAX_E) IDW = 6.9868674
Средняя абсолютная ошибка (MAE) IDW = 0.9929059
Средний квадрат ошибки (MSE) IDW = 1.7426089
Средняя квадратическая ошибка (RMSE) IDW = 1.3200791
Коэффициент детерминации (R2) IDW = 0.9377955

ЛУЧШИЕ значения: power=3.75,

Максимальная ошибка (MAX_E) IDW = 6.9792872
Средняя абсолютная ошибка (MAE) IDW = 0.9921267
Средний квадрат ошибки (MSE) IDW = 1.7404704
Средняя квадратическая ошибка (RMSE) IDW = 1.3192689
Коэффициент детерминации (R2) IDW = 0.9378718

Несколько запусков кода выше, с различными параметрами степени (от 1 до 10), показали, что, судя по R2, лучшим значением степени на обучающем массиве является 3.75 Оно даёт лучшие метрики качества.

Применим эту степень для валидационного массива.

Валидационный датасет

In [688...]

```
# 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=PARAMETER61,
                          station=df_test.keys()[y][len(PARAMETER61)+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 = 4.3841678
Средняя абсолютная ошибка (MAE) IDW = 1.0117870
Средний квадрат ошибки (MSE) IDW = 1.6608545
Средняя квадратическая ошибка (RMSE) IDW = 1.2887415
Коэффициент детерминации (R2) IDW = 0.9366787

ВЫВОД

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

6.1.3.2. Кrigинг 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 [689]: # 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[689]:

station	LoE	LaN
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 [690...]

```
# 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=0.99999, # 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_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) # преобразуем 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 [691...]

```

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=00:00:07

Значения метрик:

R2=nan, MAX_E=nan, MAE=nan, MSE=nan, RMSE=nan

Количество значений, которые не удалось предсказать: 17

Это составляет 4.0284% от обучающего массива данных

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

In [692...]

```
# 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 [693...]

```
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 = 6.3295999, Kriging - IDW = -0.6572676
Средняя абсолютная ошибка (MAE) Kriging = 0.9613482, Kriging - IDW = -0.0315577
Средний квадрат ошибки (MSE) Kriging = 1.6421216, Kriging - IDW = -0.1004873
Средняя квадратическая ошибка (RMSE) Kriging = 1.2814529, Kriging - IDW = -0.0386262
Коэффициент детерминации (R2) Kriging = 0.9413715, Kriging - IDW = 0.0035760

Валидационный датасет

Посмотрим, как модель будет отрабатывать на валидационной выборке.

```
In [694...]:  
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.OrdinaryKriging(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 [695...]

```
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}, MAE={mae_kriging_vld:.7f}, MSE={mse_kriging_vld}, '
      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=00:00:05

Значения метрик:

R2=nan, MAX_E=nan, MAE=nan, MSE=nan, RMSE=nan

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ

Количество значений, которые не удалось предсказать: 17

Это составляет 6.0284% от валидационного массива данных

In [696...]

```
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 [697...]

```
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 = 3.6755116, Kriging - IDW = -0.7086562
Средняя абсолютная ошибка (MAE) Kriging = 0.9299475, Kriging - IDW = -0.0818396
Средний квадрат ошибки (MSE) Kriging = 1.4481212, Kriging - IDW = -0.2127334
Средняя квадратическая ошибка (RMSE) Kriging = 1.2033791, Kriging - IDW = -0.0853624
Коэффициент детерминации (R2) Kriging = 0.9452931, Kriging - IDW = 0.0086144

ВЫВОД

В данном случае модель не дала 100% результат, и не смогла предсказать 17 значений ни на тренировочном, ни на валидационном датасете. Однако для нашего географического расположения точек наблюдения, из-за незначительного размера поля метеостанций, встречаются ситуации, когда не удается 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. Поэтому при применении на рабочем датасете, есть большая вероятность, что модели кригинга будет недостаточно для расчёта всех необходимых предсказаний.

6.1.3.3. 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 [698..

```
# 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 [699..

```
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 <= 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 результат списком
    y_predict_idw_tr_shrunk = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param_=PARAMETER61,
                             station_=df_test.keys()[y][len(PARAMETER61)+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 = 2.5538798
Средняя абсолютная ошибка (MAE) IDW = 1.1933338
Средний квадрат ошибки (MSE) IDW = 2.0620167
Средняя квадратическая ошибка (RMSE) IDW = 1.4359724
Коэффициент детерминации (R2) IDW = 0.9130040
```

```
Elapsed time=00:00:00
Текущие значения: iterations=2, power=2.25,
Максимальная ошибка (MAX_E) IDW = 2.5692115
Средняя абсолютная ошибка (MAE) IDW = 1.1989076
Средний квадрат ошибки (MSE) IDW = 2.0704860
Средняя квадратическая ошибка (RMSE) IDW = 1.4389183
Коэффициент детерминации (R2) IDW = 0.9126467
```

ЛУЧШИЕ значения: power=2.0,
Максимальная ошибка (MAX_E) IDW = 2.5538798
Средняя абсолютная ошибка (MAE) IDW = 1.1933338
Средний квадрат ошибки (MSE) IDW = 2.0620167
Средняя квадратическая ошибка (RMSE) IDW = 1.4359724
Коэффициент детерминации (R2) IDW = 0.9130040

Опробуем новое значение лучшей степени для IDW на сокращённом валидационном датасете

```
In [700...]: # 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=PARAMETER61,
                          station=df_test.keys()[y][len(PARAMETER61)+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 = 3.0000000
 Средняя абсолютная ошибка (MAE) IDW = 1.3196677
 Средний квадрат ошибки (MSE) IDW = 2.4166427
 Средняя квадратическая ошибка (RMSE) IDW = 1.5545555
 Коэффициент детерминации (R2) IDW = 0.8907027

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 [701...]

```
# Восстановим y_predict для лучшего R2 IDW
y_predict_idw_tr_shrunk = [
    inverse_distance_avg(row=df_test.iloc[x],
                          param=PARAMETER61,
                          station=df_test.keys()[y][len(PARAMETER61)+1:],
                          df_dists=df_station_dists,
                          power=best_power_idw_tr_shrunk)
    for x, y in x_train_idw_shrunk
]
```

In [702...]

```
# 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 [703...]

```
# 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 [704...]

```
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 = 6.3295999

Средняя абсолютная ошибка (MAE) IDW = 0.9706936

Средний квадрат ошибки (MSE) IDW = 1.6590368

Средняя квадратическая ошибка (RMSE) IDW = 1.2880360

Коэффициент детерминации (R2) IDW = 0.9407787

ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX_E) IDW = 3.6755116

Средняя абсолютная ошибка (MAE) IDW = 0.9534412

Средний квадрат ошибки (MSE) IDW = 1.5065072

Средняя квадратическая ошибка (RMSE) IDW = 1.2273986

Коэффициент детерминации (R2) IDW = 0.9425633

Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:

Максимальная ошибка (MAX_E) IDW = 4.3841678

Средняя абсолютная ошибка (MAE) IDW = 1.0117870

Средний квадрат ошибки (MSE) IDW = 1.6608545

Средняя квадратическая ошибка (RMSE) IDW = 1.2887415

Коэффициент детерминации (R2) IDW = 0.9366787

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..

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

```
In [705...]: # Добавим в df_tmp61 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 PARAMETER61#Chashnikovo и PARAMETER61#Rfrnce_point
df_tmp61 = (df_tmp61.
             assign(col_name1 = np.nan,
                   col_name2 = np.nan).
             rename(columns={"col_name1": PARAMETER61+'#'+ 'Chashnikovo',
                           "col_name2": PARAMETER61+'#'+ 'Rfrnce_point'}))
)
```

```
df_tmp61.sample(3, random_state=56)
```

Out[705]:	Soil_T#V_Volochek	Soil_T#Staritsa	Soil_T#Kashyn	Soil_T#Tver	Soil_T#Klin	Soil_T#Dmitrov	Soil_T#Volokolamsk	Soil_T#Mozhaisk	Soil_T#N_Jeru
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

In [706...]

```
# Добавим в архив параметров Soil_T столбец для будущей центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER61#Chashnikovo и PARAMETER61#Rfrnce_point
dict_df_parameters['df_'+PARAMETER61] = (dict_df_parameters['df_'+PARAMETER61].
                                             assign(col_name1 = np.nan,
                                                   col_name2 = np.nan).
                                             rename(columns={"col_name1": PARAMETER61+"#"+'Chashnikovo',
                                                            "col_name2": PARAMETER61+"#"+'Rfrnce_point'}))
dict_df_parameters['df_'+PARAMETER61].sample(3, random_state=56)
```

Out[706]:

	Soil_T#V_Volochek	Soil_T#Staritsa	Soil_T#Kashyn	Soil_T#Tver	Soil_T#Klin	Soil_T#Dmitrov	Soil_T#Volokolamsk	Soil_T#Mozhaisk	Soil_T#N_Jeru
--	-------------------	-----------------	---------------	-------------	-------------	----------------	--------------------	-----------------	---------------

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

In [707...]

```
# В архивах метеостанций df Чашниково и df для центральной точки
# создадим столбцы с назначением PARAMETER61

dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER61})
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER61}
                                         )
                                         )

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

Out[707]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point	Soil_T
2014-02-22 03:00:00	-4.156558	-4.156558	-2.109643	768.336709	747.597791	0.812775	76.635021	-7.639754	NaN
2015-05-16 03:00:00	9.181730	9.181730	10.434571	745.095535	725.921747	-1.043037	95.837940	8.552121	NaN
2020-06-18 18:00:00	27.439052	25.366130	30.241632	761.480701	743.060948	-0.492627	58.118042	18.459938	NaN

Out[707]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point	Soil_T
2014-02-22 03:00:00	-3.589183	-3.794602	-2.700734	767.373725	754.041734	0.616282	75.006151	-7.367199	NaN
2015-05-16 03:00:00	7.789650	7.789650	8.797603	746.708428	734.256500	-0.826706	94.058645	6.893682	NaN
2020-06-18 18:00:00	29.404954	25.572651	29.941094	760.796222	749.008692	-0.723724	41.950321	15.114705	NaN

Перенесение уже исправленных сплошных NaN в архивы

In [708...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER61] = df_tmp61.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[:, PARAMETER61] = df_tmp61.loc[:, PARAMETER61 + '#' + name_df[3:]]
```

In [709...]

```
# Подсчитаем количество строк со "сплошными" NaN dict_df_parameters['df_'+PARAMETER61] для 9 часов момента наблюдения
# 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_'+PARAMETER61][(dict_df_parameters['df_'+PARAMETER61].index.hour == 9) &
                                             (dict_df_parameters['df_'+PARAMETER61].index >= '2013-03-10 09:00:00')]
    .apply(lambda x: sum(x.isna()), axis=1)==len(dict_df_parameters['df_'+PARAMETER61].keys()))
)

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

Количество строк со сплошными NaN равно 1213

In [710...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp61 для 9 часов момента наблюдения
# построчно подсчитаем сумму количества NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(
    df_tmp61[(df_tmp61.index.hour == 9) & (df_tmp61.index >= '2013-03-10 09:00:00')]
    .apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp61.keys()))
)

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

Количество строк со сплошными NaN равно 1213

Суммы дней с отсутствием наблюдений температуры почвы равны полученному по результатам исправления сплошных NaN значению.

Частичное заполнение оставшихся NaN расчётными 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.

Учитывая, что наблюдения за температурой почвы имеют сезонный характер без чётко фиксированных границ начала и окончания сезона, сделаем этот расчёт для каждого года отдельно. Воспользуемся циклом и сокращённым временным датафреймом df_tmp61d.

In [711...]

```
# ПЕРЕОПРЕДЕЛИМ временный DF для стандартных моментов наблюдения
# с учётом начала фиксации этих данных в архивах
df_tmp61d = df_tmp61[(df_tmp61.index.hour == 9) & (df_tmp61.index >= '2013-03-10 09:00:00')]
```

In [712...]

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

for year in df_tmp61d.index.year.unique(): # по перечню уникальных лет в df_tmp61d, по годам
    # определим логическую маску: соответствие данному году, и не менее 1 неNaN значения в ряду
    yearly_mask = (df_tmp61d.index.year == year) & (df_tmp61d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))

    # Определим границы периодов наблюдения для данного года
    start_moment = (
        df_tmp61d[yearly_mask]
        .dropna(how='all')
        .index
        .min()
    ) # минимальный момент
    end_moment = (
        df_tmp61d[yearly_mask]
        .dropna(how='all')
        .index
        .max()
    ) # максимальный момент

    # Исходя из границ наблюдения в текущем году определим
    border_mask = ((df_tmp61d.index >= start_moment) & (df_tmp61d.index <= end_moment))
    df_tmp61y = df_tmp61d[border_mask]
    # Подсчитаем количество NaN в df_tmp61d[border_mask] и присвоим их переменной old_nan_count
    # np.isnan(df_tmp61y).sum() выдаёт количество NaN по каждому столбцу.
```

```

# Поэтому дополнительно просуммируем полученные по столбцам значения
new_nan_count = np.sum(np.isnan(df_tmp61y).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

print(f'{year} год: Кригинг')
# заменим значения в архивах, на исправленные и вычисленные из данных в df_tmp61d[border_mask]
# 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, используем переменную dummy, чтобы избежать лишнего вывода
    dummy = df_tmp61y.apply(
        lambda x: row_nan_kriging_correct(row_=x,
                                            name_param_=PARAMETER61
                                            ),
        axis=1
    )
    old_nan_count = new_nan_count # сохраняем прежнее количество NaN в old_nan_count
    new_nan_count = np.sum(np.isnan(df_tmp61y).sum()) # подсчитаем оставшиеся количество NaN

print(f'Кригингом не удалось смоделировать {new_nan_count} значений, они переданы функции IDW')

# Для NaN, которые не могут быть заменены с помощью кригинга:
# Произведём вычисление отсутствующих значений в df_tmp61y через модель IDW.
# Заменим соответствующие значения в архивах на вновь вычисленные.
# используем функцию row_nan_idw_correct

# Построчно применяем функцию обработки NaN , используем переменную dummy, чтобы избежать лишнего вывода
dummy = df_tmp61y.apply(
    lambda x: row_nan_idw_correct(row_=x,
                                   name_param_=PARAMETER61,
                                   power_=best_power_idw_tr_shrunk),
    axis=1
)
# Перенесём полученные значения в df_tmp61
df_tmp61.update(df_tmp61y)

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')
```

2022 год: Кригинг

Итерация № 1: осталось NaN: 179

Итерация № 2: осталось NaN: 22

Итерация № 3: осталось NaN: 16

Итерация № 4: осталось NaN: 12

Кригингом не удалось смоделировать 12 значений, они переданы функции IDW

2021 год: Кригинг

Итерация № 1: осталось NaN: 890

Итерация № 2: осталось NaN: 419

Итерация № 3: осталось NaN: 408

Итерация № 4: осталось NaN: 407

Кригингом не удалось смоделировать 407 значений, они переданы функции IDW

2020 год: Кригинг

Итерация № 1: осталось NaN: 1492

Итерация № 2: осталось NaN: 636

Итерация № 3: осталось NaN: 569

Итерация № 4: осталось NaN: 551

Итерация № 5: осталось NaN: 550

Кригингом не удалось смоделировать 550 значений, они переданы функции IDW

2019 год: Кригинг

Итерация № 1: осталось NaN: 538

Итерация № 2: осталось NaN: 58

Итерация № 3: осталось NaN: 52

Кригингом не удалось смоделировать 52 значений, они переданы функции IDW

2018 год: Кригинг

Итерация № 1: осталось NaN: 875

Итерация № 2: осталось NaN: 58

Итерация № 3: осталось NaN: 34

Итерация № 4: осталось NaN: 32

Итерация № 5: осталось NaN: 31

Кригингом не удалось смоделировать 31 значений, они переданы функции IDW
2017 год: Кригинг

Итерация № 1: осталось NaN: 1264

Итерация № 2: осталось NaN: 461

Итерация № 3: осталось NaN: 410

Итерация № 4: осталось NaN: 402

Итерация № 5: осталось NaN: 400

Итерация № 6: осталось NaN: 399

Итерация № 7: осталось NaN: 398

Кригингом не удалось смоделировать 398 значений, они переданы функции IDW
2016 год: Кригинг

Итерация № 1: осталось NaN: 1268

Итерация № 2: осталось NaN: 223

Итерация № 3: осталось NaN: 167

Итерация № 4: осталось NaN: 160

Итерация № 5: осталось NaN: 158

Кригингом не удалось смоделировать 158 значений, они переданы функции IDW
2015 год: Кригинг

Итерация № 1: осталось NaN: 1491

Итерация № 2: осталось NaN: 628

Итерация № 3: осталось NaN: 497

Итерация № 4: осталось NaN: 489

Кригингом не удалось смоделировать 489 значений, они переданы функции IDW
2014 год: Кригинг

Итерация № 1: осталось NaN: 1163

Итерация № 2: осталось NaN: 615

Итерация № 3: осталось NaN: 575

Итерация № 4: осталось NaN: 573

Итерация № 5: осталось NaN: 572

Итерация № 6: осталось NaN: 571

Кригингом не удалось смоделировать 571 значений, они переданы функции IDW
2013 год: Кригинг

Итерация № 1: осталось NaN: 1883

Итерация № 2: осталось NaN: 626

Итерация № 3: осталось NaN: 561

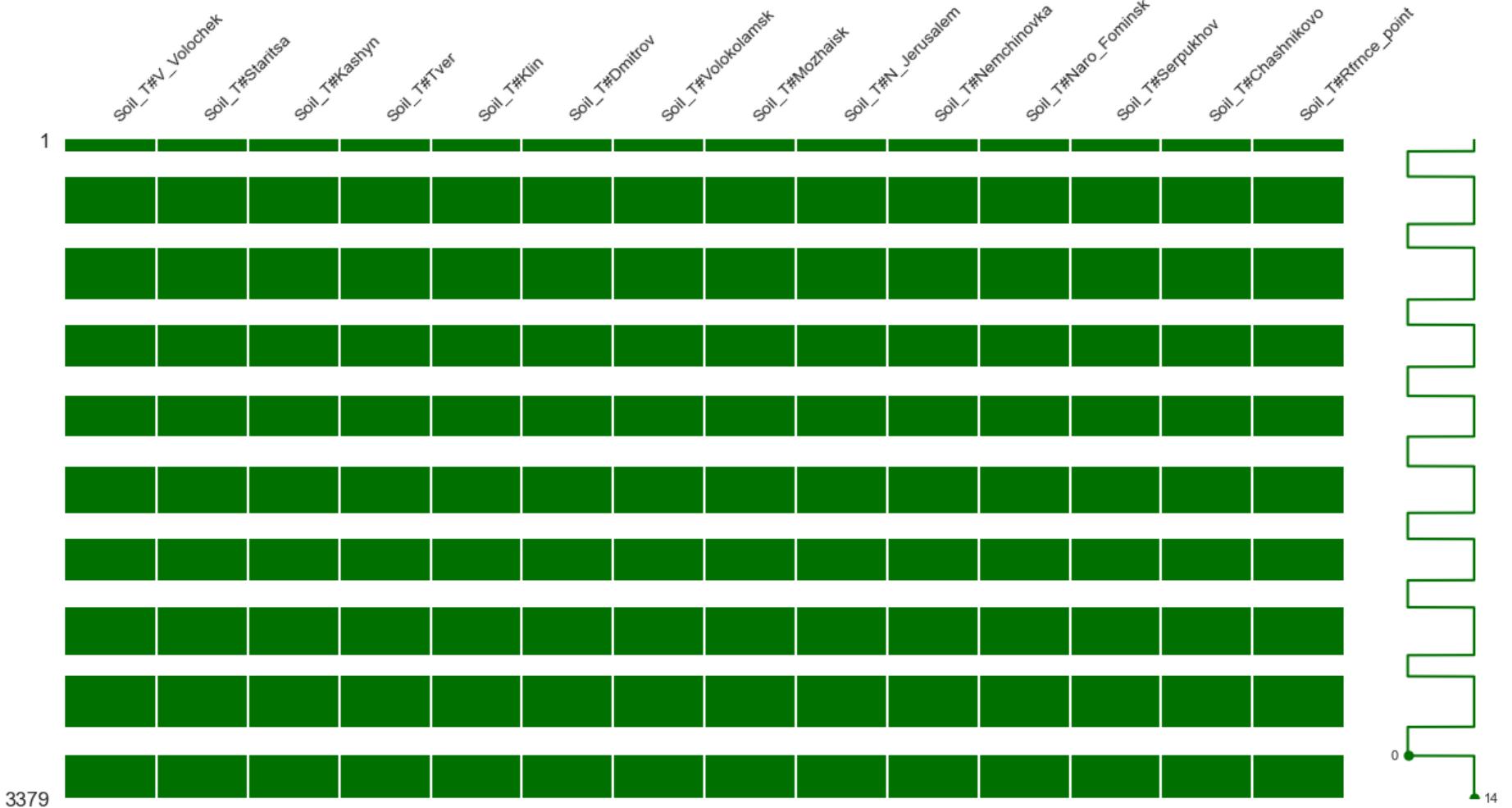
Итерация № 4: осталось NaN: 542

Кригингом не удалось смоделировать 542 значений, они переданы функции IDW
Elapsed time=00:01:21

Проверим полноту восстановления данных.

```
In [713]: msno.matrix(df_tmp61[(df_tmp61.index.hour == 9) & (df_tmp61.index >= '2013-03-10 09:00:00')],  
                     color=(0, 100/225, 0),  
                     figsize=(15, 7),  
                     fontsize=10); # выводим график из библиотеки "missingno"  
plt.show()
```

Out[713]: <AxesSubplot:>



Выведем, рандомные строки из df_tmp61

```
In [714...]: df_tmp61[(df_tmp61.index.hour == 9) & (df_tmp61.index >= '2013-03-10 09:00:00')].sample(7)
```

Out[714]:

	Soil_T#V_Volochek	Soil_T#Staritsa	Soil_T#Kashyn	Soil_T#Tver	Soil_T#Klin	Soil_T#Dmitrov	Soil_T#Volokolamsk	Soil_T#Mozhaisk	Soil_T#N_Jeru
2018-08-13 09:00:00	10.0	11.0	11.0	11.0	11.0	13.0	11.275313	11.856181	
2019-12-03 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-02-18 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-04-09 09:00:00	-4.0	-1.0	-5.0	-5.0	-2.0	-1.0	-1.852457	-0.945199	
2017-06-10 09:00:00	9.0	10.0	10.0	12.0	12.0	9.0	11.230365	10.765800	
2016-08-09 09:00:00	13.0	11.0	11.0	12.0	11.0	12.0	11.000000	12.000000	
2018-05-18 09:00:00	9.0	11.0	11.0	10.0	12.0	13.0	11.572274	12.152483	



6.1.5. Визуализация графиков температуры почвы Soil_T для условной метеостанции Чашниково

In [715...]

```
# Построим список годов для графиков
list_years = df_tmp61d.index.year.unique().tolist()
```

In [716...]

```
# Строим график в цикле для каждого года (для момента наблюдения в 9 часов)
for year in list_years:
    data1 = dict_df_parameters['df_Soil_T'][PARAMETER61+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_Soil_T'].index.year == year) &\n         (dict_df_parameters['df_Soil_T'].index.hour == 9)]
```

```

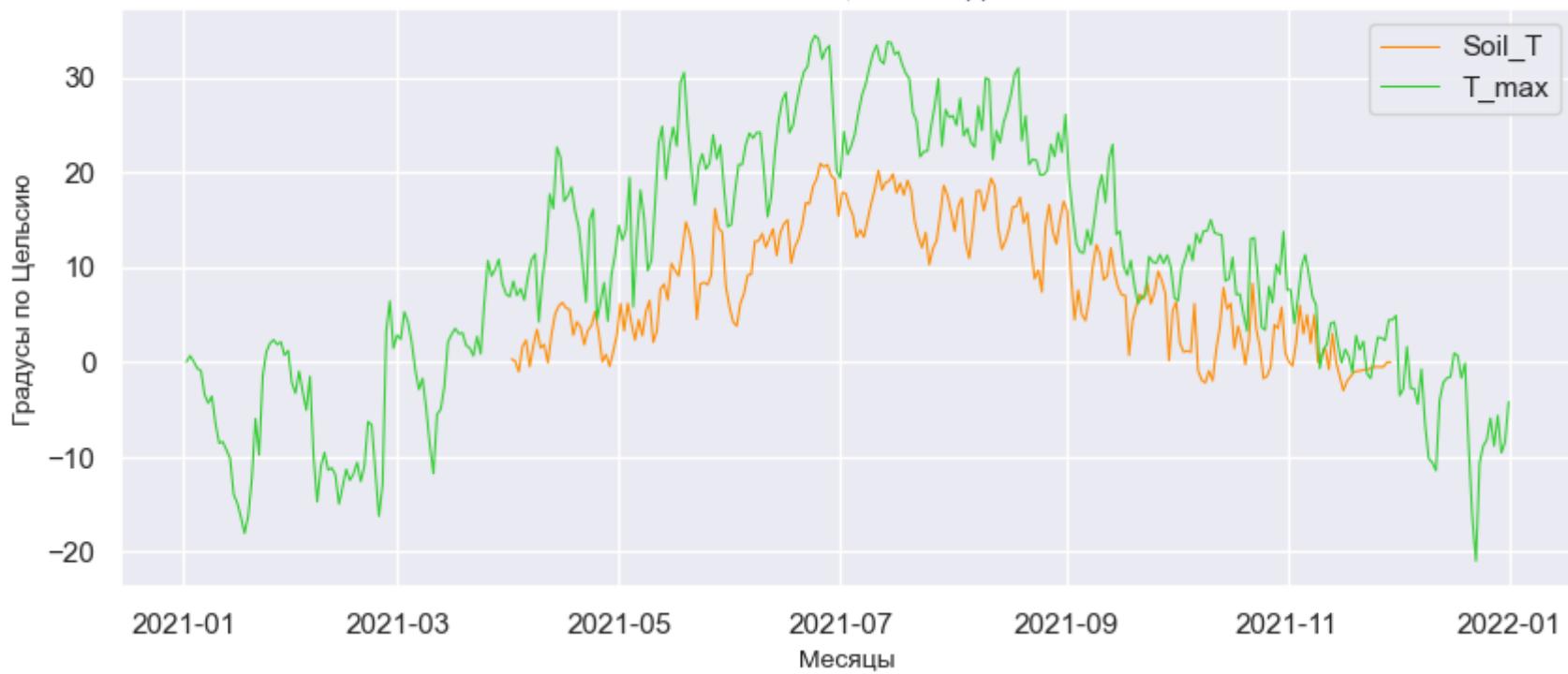
data2 = dict_df_parameters['df_T_max'][PARAMETER33+"#"+"Chashnikovo"]\n[(dict_df_parameters['df_T_max'].index.year == year) &\n (dict_df_parameters['df_T_max'].index.hour == 21)]\n\nfig, ax = plt.subplots(figsize=(10, 4))\ng1 = sns.lineplot(data=data1,\n                  color='darkorange',\n                  linewidth=0.75,\n                  ax=ax)\ng2 = sns.lineplot(data=data2,\n                  color='limegreen',\n                  linewidth=0.75,\n                  ax=ax)\n\n# Добавляем легенду\norange_line = mlines.Line2D([], [], color='darkorange', label=f'{PARAMETER61}', linewidth=0.75)\nlime_line = mlines.Line2D([], [], color='limegreen', label=f'{PARAMETER33}', linewidth=0.75)\ndummy = ax.legend(handles=[orange_line, lime_line])\ndummy = ax.set_ylabel('Градусы по Цельсию', size=10)\ndummy = ax.set_xlabel('Месяцы', size=10)\ndummy = plt.title(f'Чашниково: Ежедневная динамика параметров \n'\n                  f'температуры почвы {PARAMETER61} на 9 часов утра'\n                  f'и максимальной температуры воздуха {PARAMETER33} \n'\n                  f'на 21 час, {year} год')\nplt.show()

```

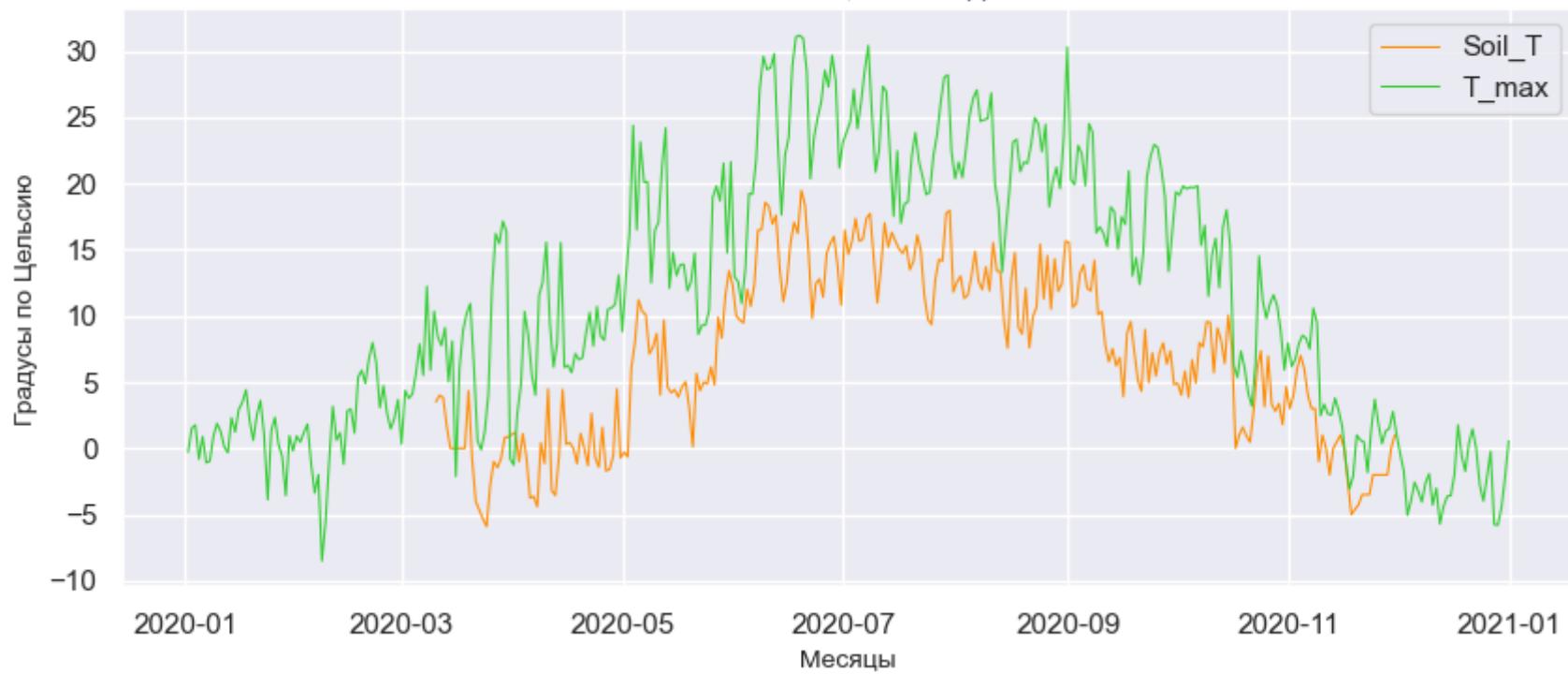
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2022 год



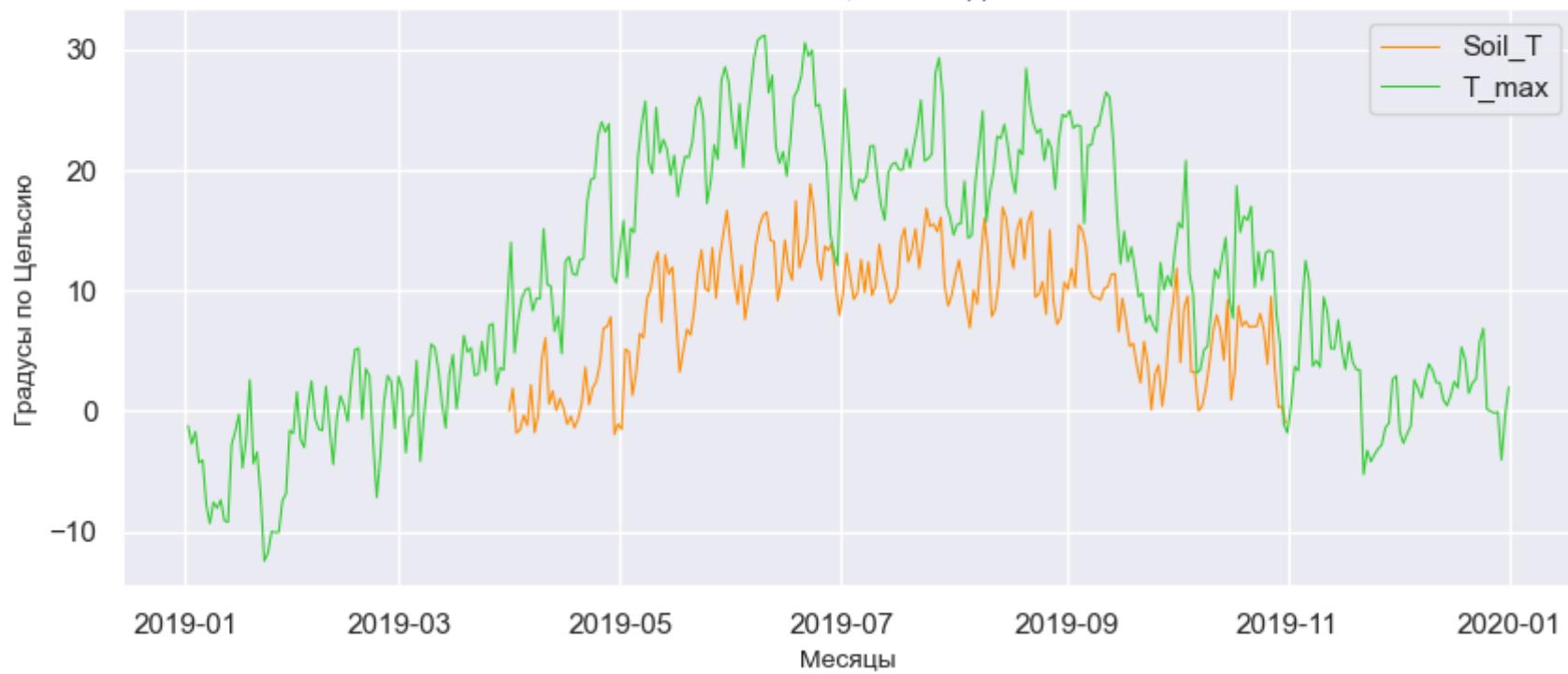
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2021 год



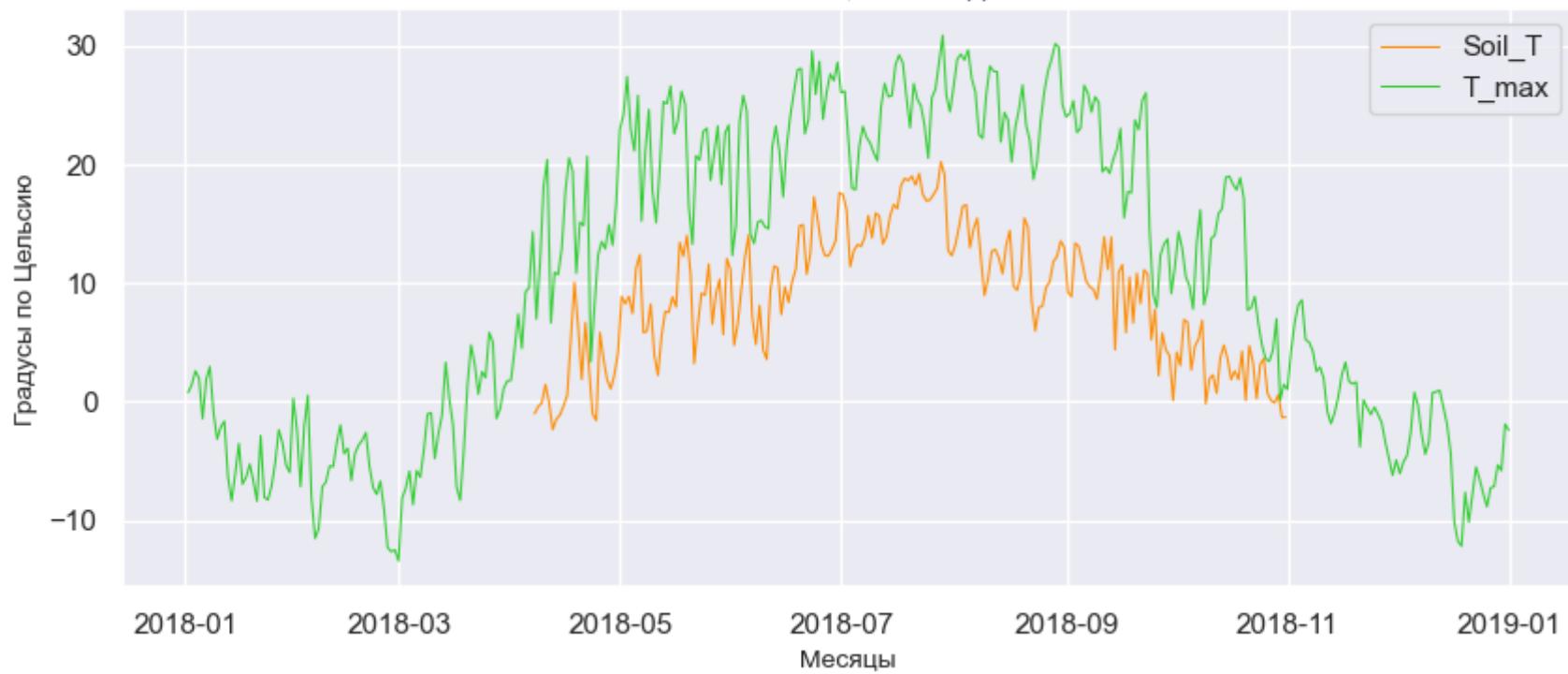
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2020 год



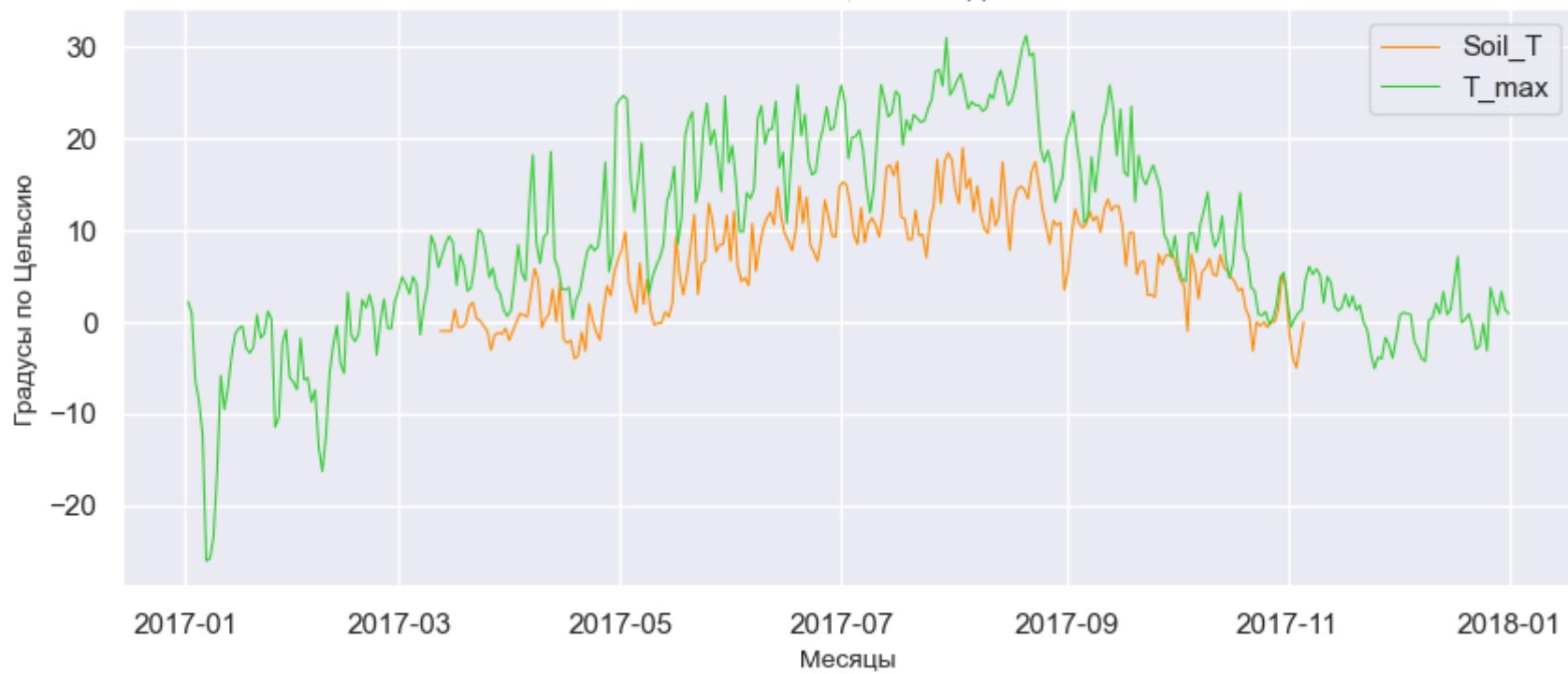
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2019 год



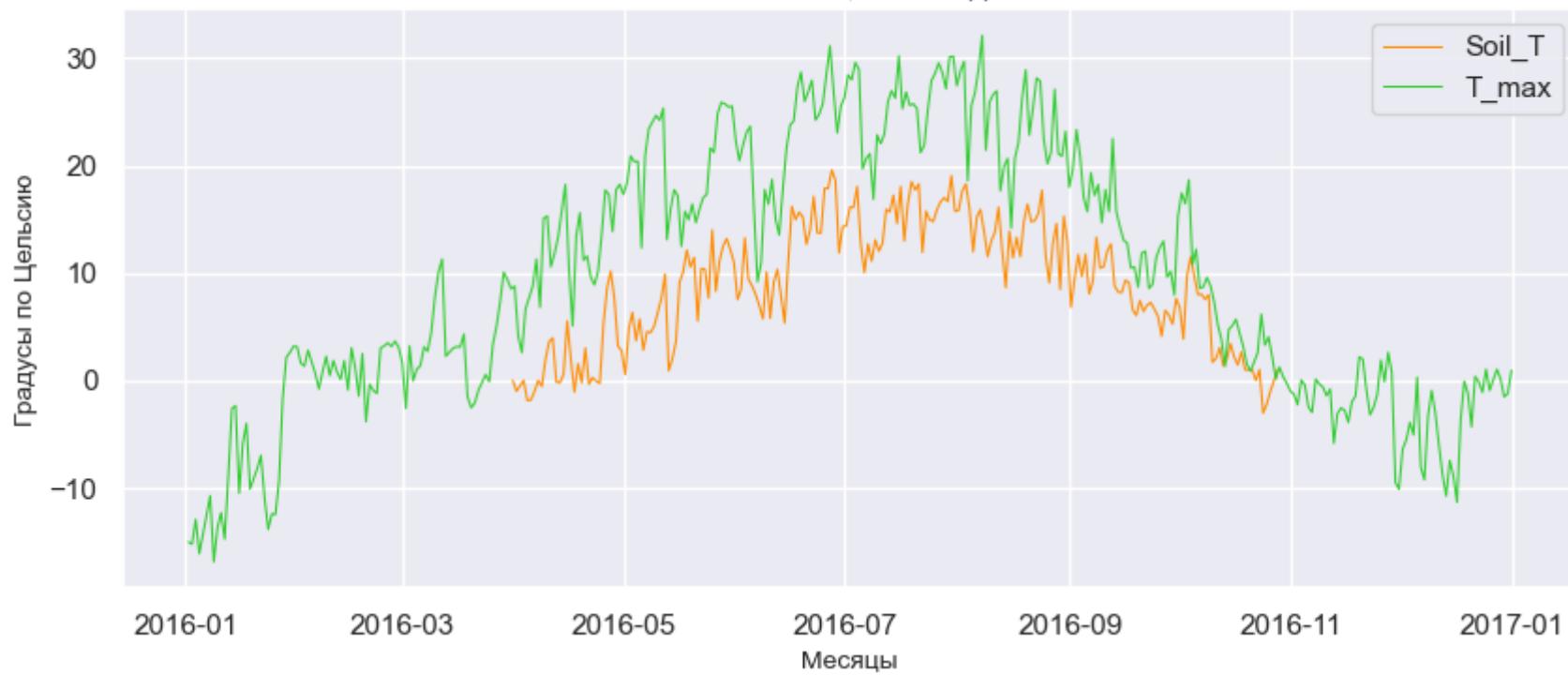
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2018 год



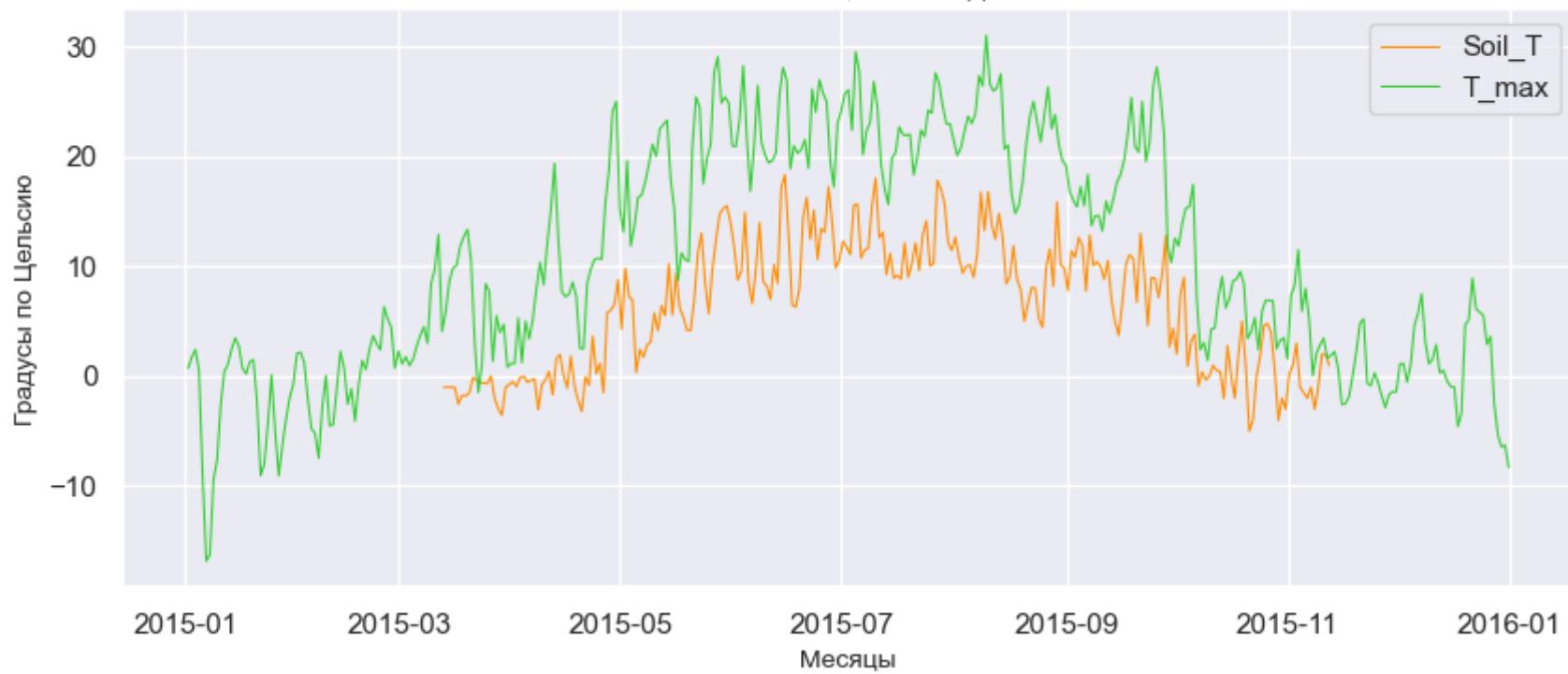
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2017 год



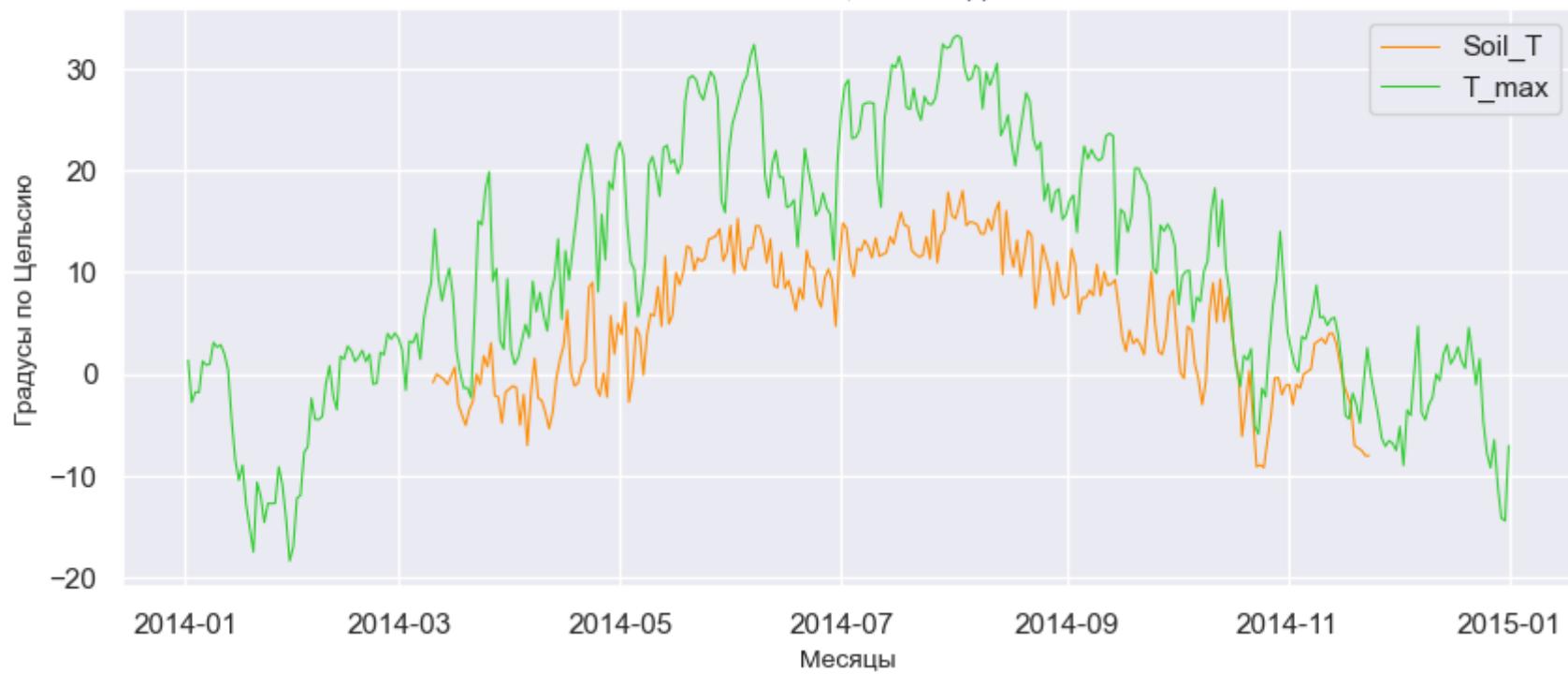
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2016 год



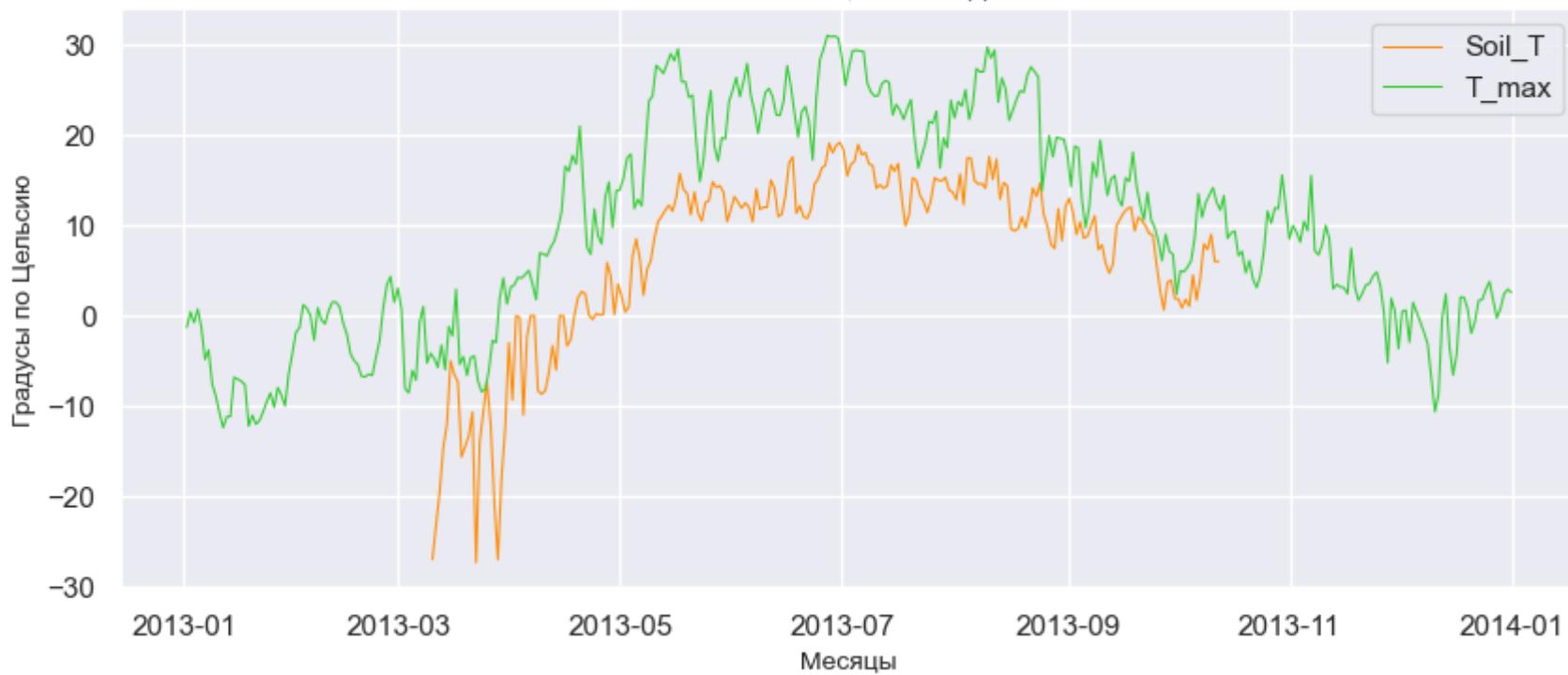
Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2015 год



Чашниково: Ежедневная динамика параметров
температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max
на 21 час, 2014 год



Чашниково: Ежедневная динамика параметров температуры почвы Soil_T на 9 часов утра максимальной температуры воздуха T_max на 21 час, 2013 год



6.1.6. Сохранение полученных данных в файлы

```
In [717...]: # # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER61}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER61}'/  
  
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_'+PARAMETER61 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER61].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER61}.csv'
)
print('DONE!')

```

df_Chashnikovo.csv -> 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_Rfrnce_point.csv -> DONE!
df_Serpukhov.csv -> DONE!
df_Staritsa.csv -> DONE!
df_Tver.csv -> DONE!
df_Volokolamsk.csv -> DONE!
df_V_Volochek.csv -> DONE!
df_Soil_T.csv -> DONE!

6.2. Высота снегового (снежного) покрова: Snow_height

Ежедневные наблюдения за снеговым покровом ведут с момента образования снежного покрова и до его исчезновения. При ежедневных наблюдениях за снеговым покровом определяют:

- степень покрытия окрестности станции снеговым покровом (балл);
- высоту снежного покрова на метеорологической площадке или на выбранном участке вблизи станции (см).

Ежедневные наблюдения за снеговым покровом должны проводиться при любых погодных условиях с постоянного места на метеорологической площадке или на выбранном месте вблизи площадки в срок, ближайший к 8 ч поясного декретного (зимнего) времени.

При отсутствии снега на поверхности почвы степень покрытия не оценивается.

Как показано в 1-й тетради, показатель снегового покрова хорошо коррелируется между метеостанциями.

6.2.1. Поиск и удаление ошибок показателя снегового покрова: Snow_height

Для данного раздела обозначим константу названия параметра.

In [718...]

```
PARAMETER62 = 'Snow_height'
```

Создаём временный DF для работы с параметром Snow_height

In [719...]

```
param_df_name = f'df_{PARAMETER62}' # преобразуем полученное значение в df_PARAMETER62 - ключ словаря dict_df_parameters
# Создадим временный df
df_tmp62 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp62.sample(5, random_state=56)
```

Out[719]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#...
--	------------------------	----------------------	--------------------	------------------	------------------	---------------------	-----------------

2014-02-22 03:00:00	NaN						
2015-05-16 03:00:00	NaN						
2020-06-18 18:00:00	NaN						
2019-12-02 21:00:00	NaN						
2008-06-05 18:00:00	NaN						



Определим последовательность действий, для поиска ошибок показателя "Высота снегового покрова".

1. Проверим синхронность наблюдения всеми метеостанциями.
2. Определяем выбросы в поле метеостанций - формируем кандидатов в ошибки.
3. Отдельно проверяем, есть ли единичные значения в рядах - это тоже кандидаты в ошибки.
4. Проверяем каждый из этих кандидатов во временном ряду (только эти выбросы, а не весь DF!) - не подтвердилось - отбрасываем
5. То, что осталось и есть ошибка.

Визуализируем архив высоты снегового покрова (Snow_height) по сезонам

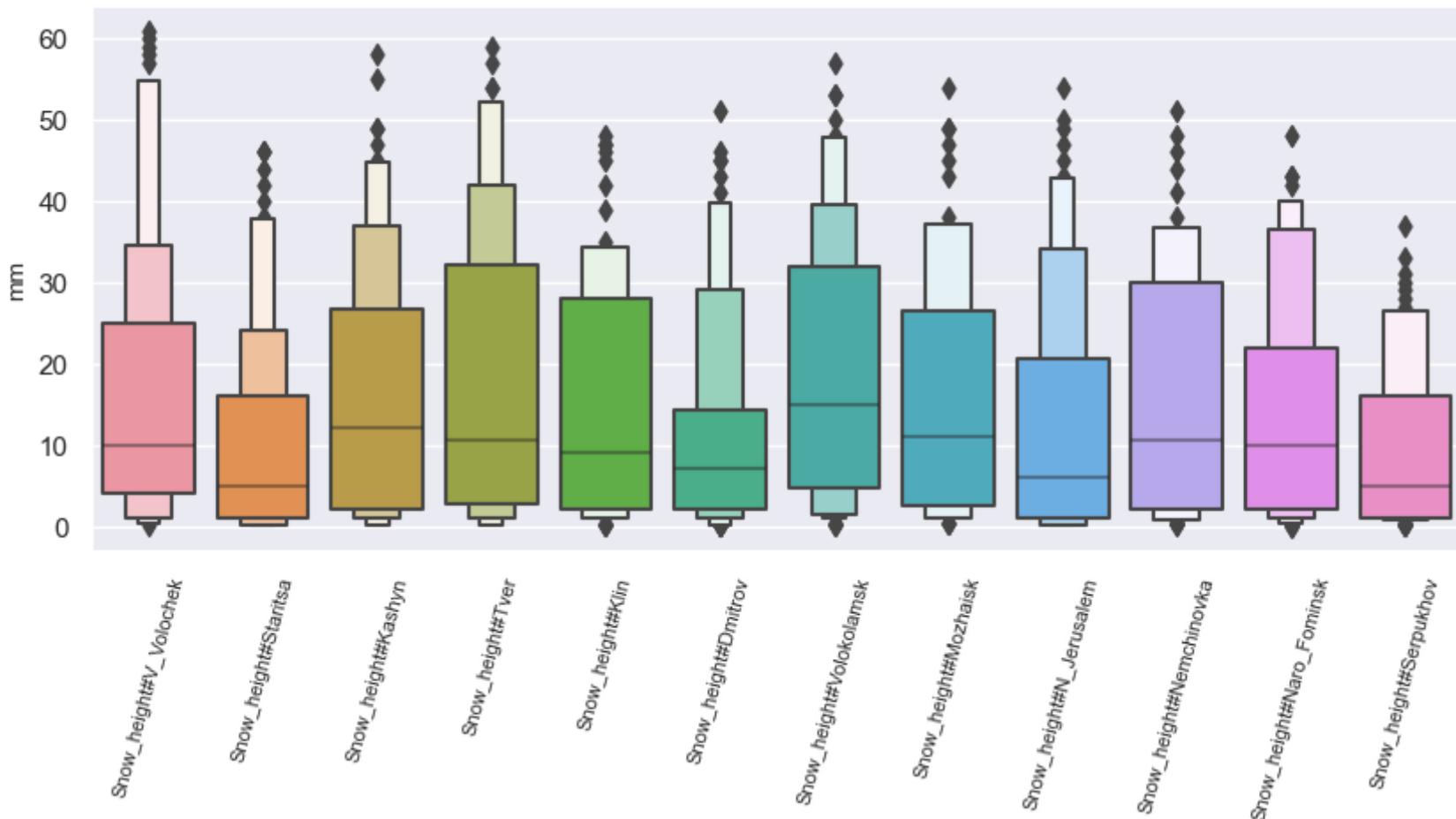
Исходя из данных о климате Московской области, временные границы сезонов определены следующим образом.

- 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

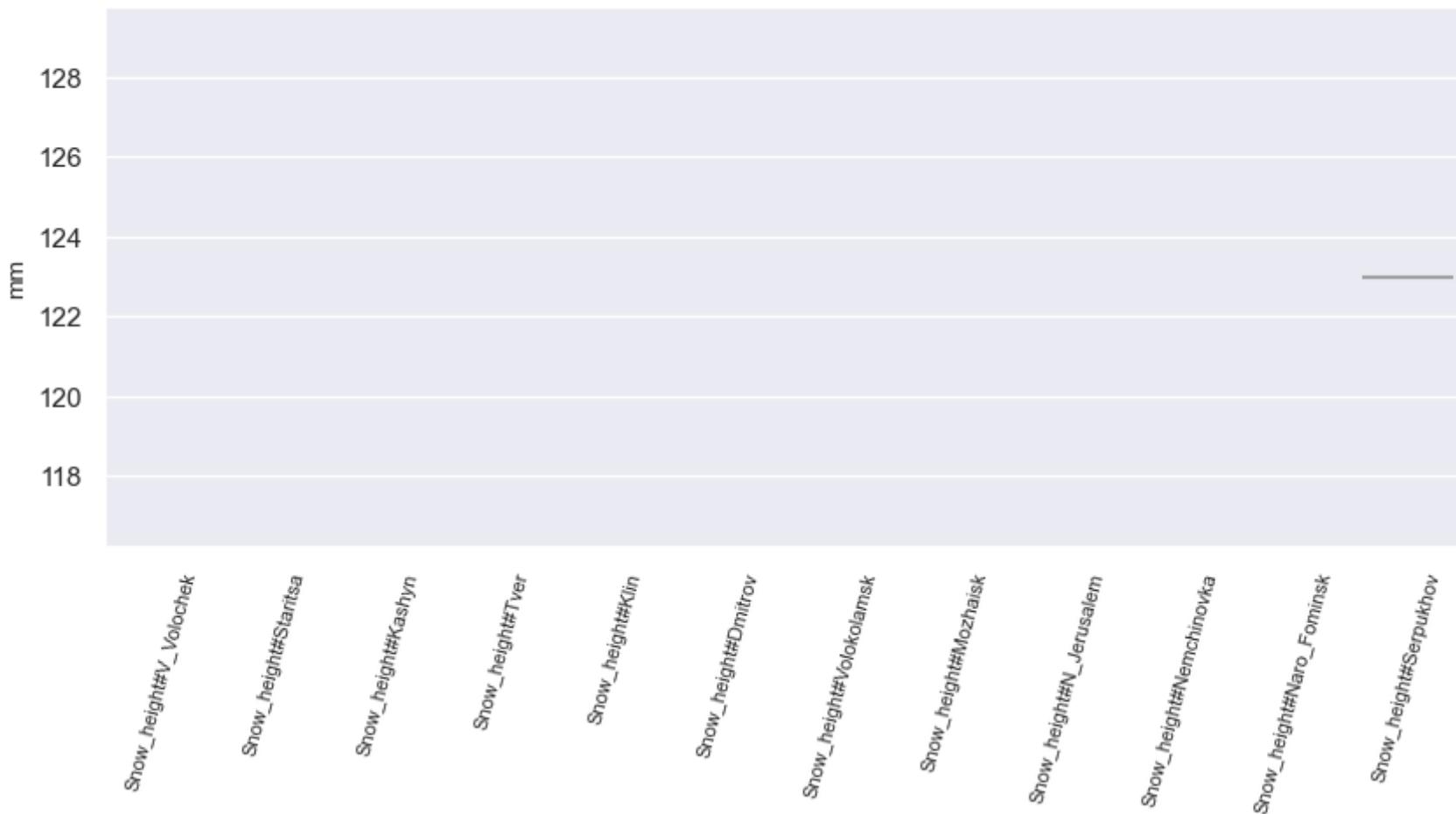
In [720...]

```
# В цикле выведем графики давления по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp62).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp62[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=75, size=8)
    dummy = g.set_ylabel('mm', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER62} в разрезе метеостанций:\n'
                        f'{season_name}')
plt.show()
```

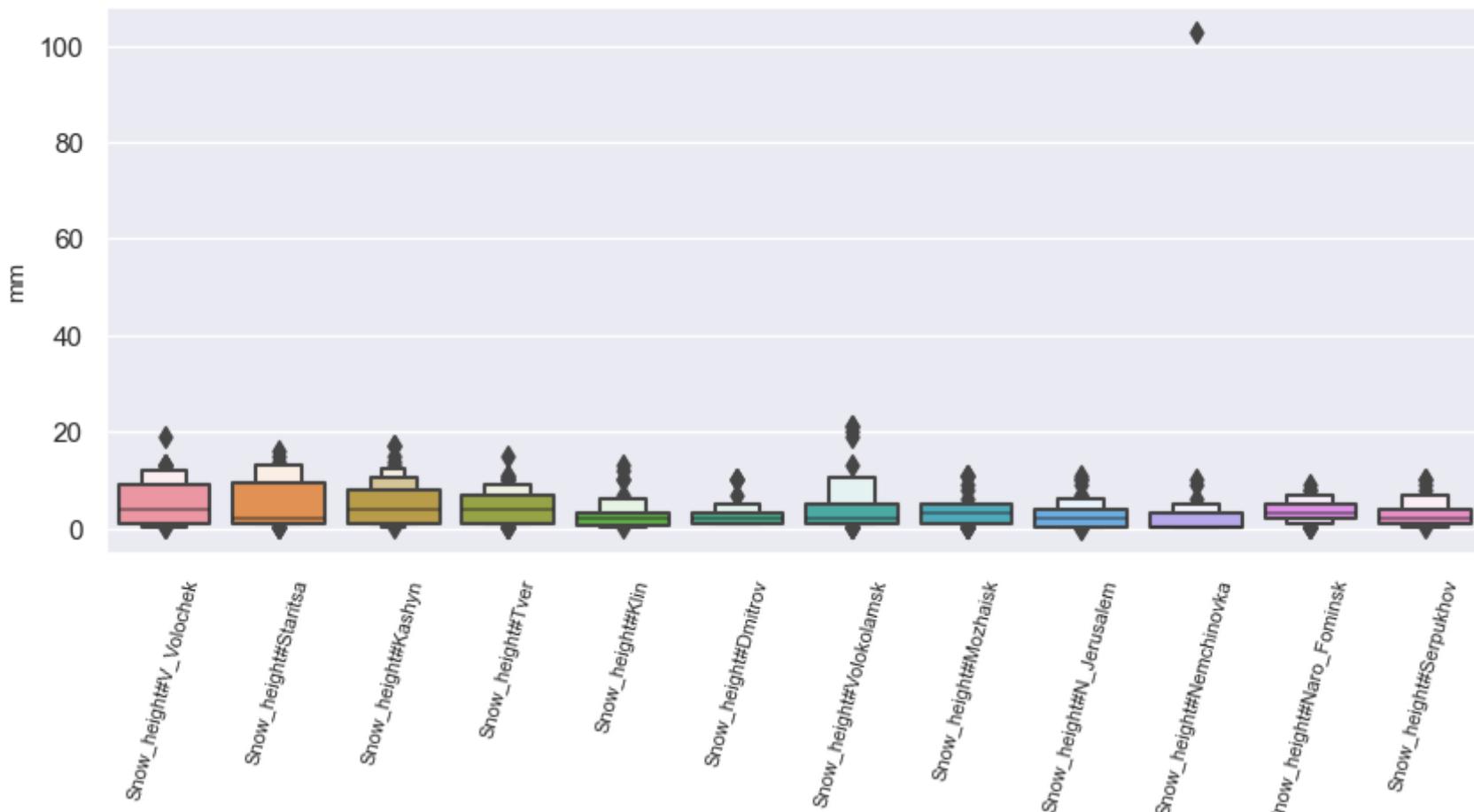
Распределение значений Snow_height в разрезе метеостанций:
spring



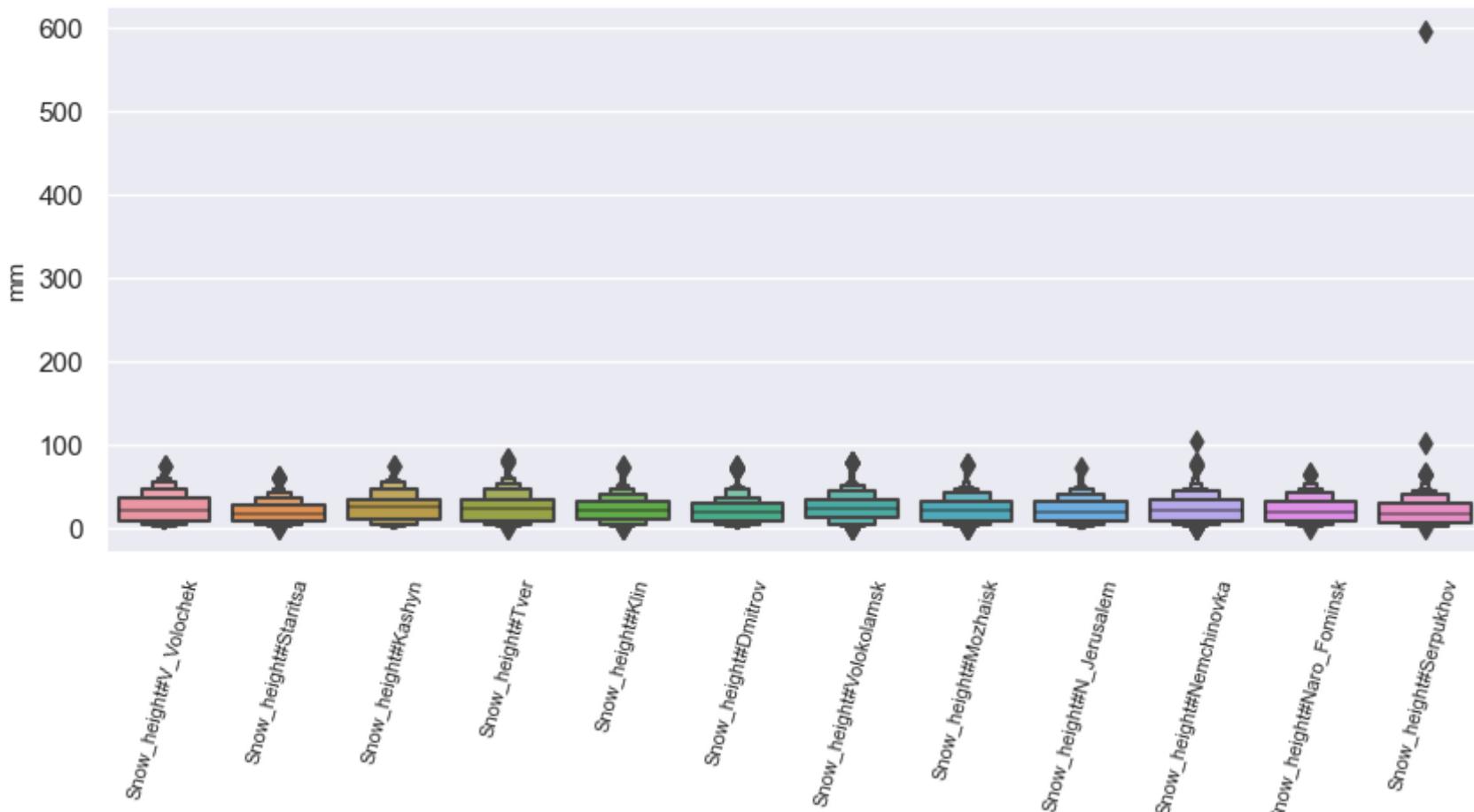
Распределение значений Snow_height в разрезе метеостанций:
summer



Распределение значений Snow_height в разрезе метеостанций:
autumn



Распределение значений Snow_height в разрезе метеостанций: winter



Как видно из графиков в данных о снеговом покрове есть аномальные выбросы, а также снеговой покров, зафиксированный летом.

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

```
In [721]: print(f'Минимальное значение: {np.nanmin(df_tmp62)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp62)},\n'
      f'Средняя: {np.nanmean(df_tmp62)},\n'
      f'Медиана: {np.nanmedian(df_tmp62)})'
```

Минимальное значение: 0.0,
Максимальное значение: 597.0,
Средняя: 20.78603751559795,
Медиана: 19.0

Определим, с какого момента началась фиксация высоты снегового покрова метеостанциями.

In [722]: `df_tmp62.dropna(how='all').index.min() # удаляем сплошные NaNы, выводим минимум индекса`

Out[722]: `Timestamp('2005-02-01 09:00:00')`

Снеговой покров фиксировался с начала архивов.

Выясним в какие часы происходила фактическая фиксация высоты снегового покрова По стандарту она должна фиксироваться в 9 часов МСК.

In [723]: `arr_moments = np.array(df_tmp62.dropna(how="all").index.hour.unique().sort_values())
arr_moments`

Out[723]: `array([3, 6, 9, 12, 15, 21], dtype=int64)`

In [724]: `for moment in arr_moments: # по массиву моментов фиксации метеонаблюдений
 print(f'В {moment} час(а/ов) встречается '
 f'{df_tmp62[df_tmp62.index.hour == moment].dropna(how="all").count().sum().astype(int)} раз(а)')`

В 3 час(а/ов) встречается 3 раз(а)
В 6 час(а/ов) встречается 6 раз(а)
В 9 час(а/ов) встречается 24827 раз(а)
В 12 час(а/ов) встречается 1 раз(а)
В 15 час(а/ов) встречается 4 раз(а)
В 21 час(а/ов) встречается 2 раз(а)

Составим список нестандартных моментов наблюдения высоты снегового покрова.

In [725]: `# Выберем координаты ячеек с нестандартным временем фиксации наблюдений
list_error_coords = [] # список координат ячеек
дополняем список кортежами (чтобы избежать лишнего вывода, присваиваем результат некой переменной)
dummy = (df_tmp62[df_tmp62.index.hour != 9] # выбираем нестандартное время фиксации температуры почвы
 .dropna(how='all') # удаляем сплошные NaN
 .apply(lambda x: list_error_coords.append(
 (x.name, [col[(len(PARAMETER62) + 1):] for col in x.dropna().index.tolist()])), # список станций`

```
        axis=1 # получаем кортеж:(время фиксации, список столбцов)
    )
print(f'Всего найдено строк с нестандартными временем фиксации температуры почвы: {len(list_error_coords)}')
```

Всего найдено строк с нестандартными временем фиксации температуры почвы: 16

Проверим, зафиксированы ли в стандартное время в диапазоне +/- 12 часов наблюдения, указанные в нестандартные моменты.

```
In [726...]:  
for idx in list_error_coords: # по списку нестандартных моментов наблюдения  
    # получаем массив с индексами в диапазоне +/- 12 часов  
    arr_ix = (  
        df_tmp62[(df_tmp62.index >= (idx[0] - pd.Timedelta('12H'))) &  
                  (df_tmp62.index <= (idx[0] + pd.Timedelta('12H')))] # отбираем строки из df_tmp62  
        .dropna(how='all')  
        .index) # удаляем сплошные NaNы (если в стандартное время наблюдений нет - такая строка удалится)  
  
    if arr_ix.max() - arr_ix.min() == pd.Timedelta('0 days 00:00:00'): # Соседних строк с наблюдениями нет  
        print(f'У момента {idx} нет соседних стандартных моментов наблюдения')
```

У момента (Timestamp('2008-08-24 03:00:00'), ['Serpukhov']) нет соседних стандартных моментов наблюдения

Полученный результат показывает, что все наблюдения высоты снегового покрова зафиксированные в нестандартное время отражены также и в стандартные часы. Исключение 24 августа 2008 года является ошибкой, так как в августе 2008 года снеговой покров не наблюдался.

Поскольку у нас есть только один (к тому же ошибочный) момент наблюдения без соседних стандартных моментов, его можно безболезненно удалить. Остальные нестандартные моменты подтверждены ближайшими стандартными и тоже могут быть безболезненно удалены.

```
In [727...]:  
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_tmp62.at[error_coords[0], PARAMETER62+'#'+station] = np.nan
```

Найдем выбросы в поле метеостанций. Как было показано в начале подраздела, максимальные величины снегового покрова доходят почти до 6 метров. Это явная ошибка. Но в ряду метеостанций распределение может оказаться настолько смещённым, что такие экстремальные значения окажутся внутри границ доверительного интервала. Поэтому будет необходимо расчитывать границы для каждого значения, последовательно исключая эти значения из вычисления средней. Параметры:

- используем среднюю по обратным квадратам расстояния от центра поля,
- исключаем проверяемое значение из подсчёта средней и высчитываем границы доверительного интервала для каждого отдельного значения,
- определим границы доверительного интервала в 3 сигмы.

Помимо этого, исключим из возможных выбросов условные значения:

- 'Измерение невозможно или неточно.' = 0,09
- 'Снежный покров не постоянный.' = 0,1

In [728...]

```
# 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_=PARAMETER62,
            station_=station_[len(PARAMETER62)+1 :],
            inclusive_=False
        )
        # Оставляем в списке выбросов только одно упоминание о каждой станции, если такое 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
                    (tup_out_[0] not in [0.1, 0.09])): # Если значение выброса не является условным [0.1, 0.09]
                    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 [729...]:

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

df_tmp62 = df_tmp62.assign(field_out=df_tmp62.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: 00:03:21

In [730...]:

```
df_tmp62.dropna(subset=["field_out"]).sample(5, random_state=56)
```

Out[730]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
--	------------------------	----------------------	--------------------	------------------	------------------	---------------------	--------------------

2016-04-05 09:00:00	NaN	0.10	1.0	0.1	NaN	NaN	NaN
2010-01-27 09:00:00	24.0	22.00	26.0	25.0	26.0	20.0	20.0
2010-03-11 09:00:00	51.0	38.00	43.0	52.0	43.0	37.0	37.0
2010-03-15 09:00:00	47.0	38.00	43.0	49.0	43.0	35.0	35.0
2021-11-23 09:00:00	1.0	0.25	1.0	1.0	2.0	1.0	1.0

◀ **▶**

Проверим минимальные и максимальные значения выбросов в поле метеостанций.

In [731...]

```
# field_out содержит списки кортежей с характеристиками выбросов!
tup_out_stations = (
    df_tmp62.field_out.dropna().apply(lambda x: min([i[0] for i in x])).min(),
    df_tmp62.field_out.dropna().apply(lambda x: max([i[0] for i in x])).max()
)
tup_out_stations
```

Out[731]:

```
(0.0, 597.0)
```

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 [732...]

```
# 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_tmp62.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: 107

Проверим максимальное количество выбросов в поле метеостанций на момент наблюдения.

In [733...]

```
# Находим максимальное количество выбросов в строке поля метеостанций
max_field_outliers = df_tmp62.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 названий станций с выбросами.

In [734...]

```
# 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_tmp62 = df_tmp62.assign(error_at =
                           df_tmp62.
                           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_=PARAMETER62)
                                 ) if isinstance(x.field_out, list) else np.nan,
                                 axis=1
                               )
                           )
```

In [735...]

```
df_tmp62.dropna(subset=["error_at"]).sample(5, random_state=56)
```

Out[735]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2016-04-05 09:00:00	NaN	0.10	1.0	0.1	NaN	NaN	NaN
2010-01-27 09:00:00	24.0	22.00	26.0	25.0	26.0	20.0	20.0
2010-03-11 09:00:00	51.0	38.00	43.0	52.0	43.0	37.0	37.0
2010-03-15 09:00:00	47.0	38.00	43.0	49.0	43.0	35.0	35.0
2021-11-23 09:00:00	1.0	0.25	1.0	1.0	2.0	1.0	1.0

◀ ▶

Для подтверждения, являются ли отобранные значения высоты снегового покрова ошибками, найдём выбросы во временном ряду

Для анализа выбросов во временном окне следует иметь в виду, что высота снегового покрова измеряется 1 раз в день и может интенсивно меняться из-за осадков или таяния. Поэтому целесообразно изучить выбросы во временном окне с интервалом в небольшое количество дней.

Используем только один вариант поиска выбросов. Параметры:

- используем границы нормального распределения,
- не включаем проверяемое значение в подсчёт средней и сигмы,
- определим границы доверительного интервала в 95%,
- определим временное окно в +/- 3 дня ('3D'),
- включим в подсчёт моменты наблюдения только в 9 часов (equalhours=True).

Поскольку выбросы за один момент наблюдения при различных параметрах поиска могут существовать одновременно в нескольких метеостанциях, мы сформируем список координат выбросов и выведем их соответствующие значения в отдельную серию

In [736..

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

# Нельзя удалять NaN в поле field_out непосредственно в df_tmp62 (Это приведёт к некорректным временным рядам!)
# Удалим NaN в столбце "field_out" в результирующем df

# Определим серию для записи списков с данными выбросов, индекс равен общему индексу архивов, тип данных - объект,
# название серии - будущее имя соответствующего столбца в DF
ser_time_out = pd.Series(index = df_tmp62.index, dtype='object', name="time_out_3D")

for elem in df_tmp62.dropna(subset=["error_at"]).error_at.tolist(): # поэлементно в списке значений столбца error_at
    # присваиваем элементу серии по соответствующему datetime индексу значение - пустой список
    ser_time_out.at[elem[0]] = []

for station in elem[1]: # по метеостанциям, указанным в списке (2й элемент кортежа error_at)
    # Определяем вербальные координаты ячеек с выбросами: TimeStamp и название столбца, соответствующего станции:
    idxs = (elem[0], PARAMETER62+'#'+station)

    # Вызываем функцию time_outliers с обозначенными выше параметрами
    val_func = time_outliers(df=df_tmp62,
                                # В качестве серии row_ передаём серию из одного значения: на момент наблюдения,
                                # где индекс серии соответствует названию столбца (idxs[1])
                                row_=df_tmp62.loc[idxs[0]][df_tmp62.loc[idxs[0]].index == idxs[1]],
                                td_symbol_='D',
                                td_quant_='3',
                                equal_hours_=True,
                                method_='norm',
                                criterium_=0.95,
                                inclusive_=False)

    # Присваиваем элементу серии по соответствующему datetime индексу результатом вызова функции (список)
    if isinstance (val_func, float): # Если time_outliers вернула NaN (то есть значение float, то ничего не делаем
        pass
    else: # Иначе, если time_outliers вернула список
        list_out = ser_time_out.at[idxs[0]] # Получаем список, находящийся в серии по индексу
        list_out = list_out + val_func # Расширяем его за счёт вывода функции (конкатенация)
        ser_time_out.at[idxs[0]] = list_out # Записываем в серию обновлённый список

# ser_time_out.at[idxs[0]]
```

```

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: 00:00:05

Присоединим полученную серию к df_tmp62

```

In [737...]: # Индекс серии совпадает с индексом всех архивов, а значения в серии упорядочены по этому индексу.
# Поэтому произведём объединение по индексу
df_tmp62 = (df_tmp62
             .merge(ser_time_out, left_index=True, right_index=True) # производим объединение
            )

```

```

In [738...]: # Выводим случайные строки из df_tmp62, удалив NaN в столбце time_out_24H
df_tmp62.dropna(subset=["time_out_3D"]).sample(5, random_state=56)

```

	Snow_height#V_Volochev	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#...
2016-04-05 09:00:00	NaN	0.10	1.0	0.1	NaN	NaN	
2010-01-27 09:00:00	24.0	22.00	26.0	25.0	26.0	20.0	
2010-03-11 09:00:00	51.0	38.00	43.0	52.0	43.0	37.0	
2010-03-15 09:00:00	47.0	38.00	43.0	49.0	43.0	35.0	
2021-11-23 09:00:00	1.0	0.25	1.0	1.0	2.0	1.0	

Заменим пустые списки в столбце time_out_3D на NaN

In [739]:

```
df_tmp62.time_out_3D = df_tmp62.time_out_3D.apply(lambda x: np.nan if x == [] else x)
```

In [740]:

```
df_tmp62.dropna(subset=["time_out_3D"]).sample(5, random_state=56)
```

Out[740]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Leningrad
2006-04-12 09:00:00	2.0	NaN	NaN	NaN	NaN	NaN	NaN
2009-11-01 09:00:00	NaN	1.0	3.0	NaN	NaN	NaN	1.0
2019-11-01 09:00:00	1.0	1.0	10.0	3.0	1.0	1.0	1.0
2011-11-18 09:00:00	NaN	NaN	1.0	NaN	NaN	NaN	NaN
2014-11-05 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Проверим минимальные и максимальные значения выбросов во временном окне.

In [741]:

```
# field_out содержит списки кортежей с характеристиками выбросов!
tup_out_time = (
    df_tmp62.time_out_3D.dropna().apply(lambda x: min([i[0] for i in x])).min(),
    df_tmp62.time_out_3D.dropna().apply(lambda x: max([i[0] for i in x])).max()
)
tup_out_time
print(f'Проверка равенства экстремальных выбросов в поле метеостанций и во временном окне: '
      f'{tup_out_time == tup_out_stations}')
```

Out[741]:

```
(0.0, 597.0)
```

Проверка равенства экстремальных выбросов в поле метеостанций и во временном окне: True

Применимость критериев ошибочных значений

Рассмотрим полученные данные об аномальных значениях.

In [742...]

```
# 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_tmp62 = (df_tmp62
    .assign(double_error_at =
        df_tmp62.dropna(subset=["time_out_3D"])
        .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_= 'time_out_3D', # используем выбросы во временном окне
                    param_=PARAMETER62)
               ) if isinstance(x.field_out, list) else np.nan,
               axis=1
            )
        )
    )
```

In [743...]

```
df_tmp62.dropna(subset=["double_error_at"])
```

Out[743]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2022-04-13 09:00:00	0.25	5.00	NaN	NaN	NaN	0.10	
2021-11-27 09:00:00	1.00	0.25	0.25	1.00	8.00	1.00	
2021-11-19 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2021-11-16 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2021-11-12 09:00:00	0.25	NaN	0.25	2.00	1.00	1.00	
2021-11-09 09:00:00	0.25	2.00	NaN	0.25	NaN	1.00	
2021-01-28 09:00:00	19.00	20.00	29.00	19.00	13.00	17.00	
2020-11-19 09:00:00	0.25	1.00	1.00	3.00	0.25	NaN	
2020-10-18 09:00:00	3.00	0.10	0.25	1.00	NaN	NaN	
2020-04-25 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	
2020-04-12 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2020-04-11 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2020-03-21 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-03-07 09:00:00	7.00	NaN	NaN	NaN	NaN	NaN	NaN
2020-02-26 09:00:00	0.25	0.25	4.00	1.00	1.00	1.00	1.00
2020-01-24 09:00:00	3.00	1.00	2.00	6.00	2.00	4.00	4.00
2020-01-22 09:00:00	1.00	0.25	NaN	NaN	NaN	NaN	NaN
2019-12-27 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-12-12 09:00:00	NaN	0.25	0.25	1.00	0.25	NaN	NaN
2019-12-11 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2019-11-27 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-11-09 09:00:00	NaN	NaN	4.00	NaN	NaN	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2019-11-08 09:00:00	0.25	NaN	7.00	6.00	NaN	NaN	NaN
2019-11-01 09:00:00	1.00	1.00	10.00	3.00	1.00	1.00	1.00
2019-10-29 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-10-08 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-04-15 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-04-10 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-04-04 09:00:00	NaN	0.10	0.10	NaN	NaN	NaN	0.25
2019-04-01 09:00:00	5.00	0.10	1.00	NaN	NaN	2.00	NaN
2019-03-01 09:00:00	28.00	24.00	45.00	26.00	26.00	38.00	NaN
2018-11-14 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2018-11-01 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2018-10-28 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2018-04-22 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2017-11-21 09:00:00	2.00	NaN	2.00	NaN	NaN	NaN	NaN
2017-11-20 09:00:00	4.00	NaN	2.00	1.00	NaN	NaN	NaN
2017-10-28 09:00:00	2.00	1.00	0.25	3.00	0.25	NaN	NaN
2017-10-25 09:00:00	NaN	NaN	2.00	0.25	0.25	0.25	1.00
2017-02-23 09:00:00	23.00	20.00	26.00	25.00	28.00	27.00	27.00
2016-04-17 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-12-26 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2015-12-06 09:00:00	NaN	0.10	NaN	1.00	NaN	NaN	NaN
2015-11-19 09:00:00	1.00	2.00	4.00	2.00	1.00	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2015-11-08 09:00:00	NaN	NaN	NaN	0.25	NaN	NaN	NaN
2015-10-30 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2015-10-28 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2015-10-10 09:00:00	NaN	NaN	0.25	NaN	4.00	NaN	NaN
2015-10-07 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2015-03-24 09:00:00	1.00	0.25	4.00	3.00	1.00	1.00	1.00
2015-01-13 09:00:00	27.00	25.00	28.00	30.00	29.00	25.00	25.00
2014-12-01 09:00:00	1.00	NaN	NaN	0.25	0.25	NaN	NaN
2014-11-05 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2014-10-16 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2014-03-21 09:00:00	14.00	5.00	4.00	5.00	4.00	5.00	5.00

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2014-03-10 09:00:00	0.25	0.10	0.10	NaN	NaN	NaN	NaN
2014-01-13 09:00:00	2.00	5.00	0.25	NaN	12.00	8.00	NaN
2013-11-25 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2013-03-20 09:00:00	74.00	60.00	73.00	81.00	70.00	68.00	NaN
2013-03-19 09:00:00	73.00	58.00	70.00	77.00	69.00	66.00	NaN
2012-11-04 09:00:00	NaN	NaN	6.00	NaN	NaN	NaN	NaN
2012-04-07 09:00:00	33.00	16.00	35.00	32.00	32.00	30.00	NaN
2012-03-30 09:00:00	33.00	20.00	39.00	39.00	34.00	31.00	NaN
2012-03-29 09:00:00	33.00	16.00	37.00	36.00	33.00	29.00	NaN
2011-12-18 09:00:00	NaN	0.00	NaN	NaN	NaN	NaN	NaN
2011-12-03 09:00:00	NaN	1.00	1.00	2.00	1.00	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2011-11-27 09:00:00	NaN	1.00	3.00	1.00	NaN	NaN	NaN
2011-11-25 09:00:00	NaN	NaN	1.00	NaN	2.00	1.00	1.00
2011-11-18 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2011-11-17 09:00:00	NaN	NaN	3.00	NaN	3.00	2.00	2.00
2011-11-09 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	1.00
2011-10-18 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-11-18 09:00:00	NaN	3.00	NaN	NaN	NaN	NaN	NaN
2010-11-09 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	NaN
2010-10-30 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2010-10-23 09:00:00	NaN	NaN	2.00	NaN	NaN	NaN	NaN
2010-03-11 09:00:00	51.00	38.00	43.00	52.00	43.00	37.00	37.00

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2010-03-05 09:00:00	51.00	37.00	43.00	50.00	73.00	32.00	
2009-11-03 09:00:00	NaN	NaN	2.00	5.00	NaN	1.00	
2009-11-01 09:00:00	NaN	1.00	3.00	NaN	NaN	1.00	
2009-10-31 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	
2009-10-12 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	
2008-11-23 09:00:00	11.00	NaN	3.00	4.00	2.00	NaN	
2008-04-15 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	
2008-03-28 09:00:00	2.00	1.00	1.00	NaN	NaN	NaN	
2008-03-20 09:00:00	5.00	NaN	15.00	3.00	NaN	NaN	
2008-03-18 09:00:00	3.00	NaN	12.00	NaN	3.00	2.00	
2007-11-04 09:00:00	NaN	1.00	1.00	NaN	1.00	2.00	

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2007-04-10 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-04-07 09:00:00	NaN	5.00	NaN	NaN	1.00	NaN	
2007-03-19 09:00:00	NaN	1.00	NaN	NaN	NaN	NaN	
2006-12-18 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2006-04-12 09:00:00	2.00	NaN	NaN	NaN	NaN	NaN	
2006-04-07 09:00:00	3.00	NaN	3.00	NaN	NaN	3.00	
2006-03-02 09:00:00	51.00	35.00	45.00	44.00	46.00	47.00	
2005-11-12 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	
2005-10-26 09:00:00	NaN	2.00	NaN	NaN	NaN	NaN	
2005-04-23 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2005-04-22 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	

```
Snow_height#V_Volochek Snow_height#Staritsa Snow_height#Kashyn Snow_height#Tver Snow_height#Klin Snow_height#Dmitrov Snow_height#Lipetsk
```

2005-02-21 00:00:00	NaN	26.00	NaN	46.00	33.00	22.00
---------------------	-----	-------	-----	-------	-------	-------

Найдём общие выбросы как в поле метеостанций, так и во временном окне. (У нас могут быть случаи, когда при проверке выброса в поле метеостанций, выбросов для того же значения во временном окне не обнаруживается).

Для контроля, найдём пересечение списков в столбцах error_at и double_error_at, выведем его в отдельный столбец cross_check

In [744...]

```
df_tmp62 = df_tmp62.assign(  
    cross_check = df_tmp62  
        .dropna(subset=["error_at", "double_error_at"])  
        .apply(  
            lambda x: list(set(x.error_at[1]) & set(x.double_error_at[1])),  
            axis=1  
        )  
    )
```

In [745...]

```
df_tmp62.dropna(subset=["cross_check"]).sample(7, random_state=56)
```

Out[745]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Leningrad
2006-04-12 09:00:00	2.0	NaN	NaN	NaN	NaN	NaN	NaN
2009-11-01 09:00:00	NaN	1.0	3.0	NaN	NaN	1.0	
2019-11-01 09:00:00	1.0	1.0	10.0	3.00	1.00	1.0	
2011-11-18 09:00:00	NaN	NaN	1.0	NaN	NaN	NaN	
2014-11-05 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2017-10-25 09:00:00	NaN	NaN	2.0	0.25	0.25	1.0	
2010-11-09 09:00:00	1.0	NaN	NaN	NaN	NaN	NaN	

In [746...]

```
# Подсчитаем количество выбросов в поле метеостанций
count_field_out = (df_tmp62
    .error_at # по столбцу error_at
    .dropna()
    .apply(lambda x: len(x[1]) # вычислим длины списков с перечнем метеостанций с выбросами
          )
    ).sum() # просуммируем их в общий итог

# Подсчитаем из них количество выбросов во временном окне
count_time_out = (df_tmp62
    .double_error_at # по столбцу double_error_at
    .dropna()
    .apply(lambda x: len(x[1]) # вычислим длины списков с перечнем метеостанций с выбросами
          )
```

```

        )
    ).sum() # просуммируем их в общий итог

# Подсчитаем из них количество выбросов в контрольном столбце
count_cross_out = (df_tmp62
    .cross_check # по столбцу double_error_at
    .dropna()
    .apply(lambda x: len(x) # вычислим длины списков с перечнем метеостанций с выбросами
          )
    ).sum() # просуммируем их в общий итог

print(f'В поле метеостанций выявлено аномальных значений: {count_field_out}, из них:\n'
      f'Подтверждается аномалиями в промежутке +/- 3 дня по всем моментам наблюдения: '
      f'{count_time_out}\n'
      f'Проверка: пересечение множеств выбросов в поле метеостанций и во временном окне насчитывает '
      f'{count_cross_out} элемент(а/ов)')

```

В поле метеостанций выявлено аномальных значений: 749,

из них:

Подтверждается аномалиями в промежутке +/- 3 дня по всем моментам наблюдения: 100

Проверка: пересечение множеств выбросов в поле метеостанций и во временном окне насчитывает 100 элемент(а/ов)

Определим конечный критерий ошибочных значений:

1. Аномальное значение выявлено в поле метеостанций И ПРИ ЭТОМ

А. Аномальное значение выявлено во временном ряду в промежутке +/- 3 дня (выбивается из общей тенденции изменения температуры почвы)

В. Аномальное значение не входит в список условных значений (учтено на этапе выявления выбросов):

- 'Измерение невозможно или неточно.' = 0,09
- 'Снежный покров не постоянный.' = 0,1

Удалим (заменим на NaN) все ошибочные значения

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

In [747...]

df_tmp62.dropna(subset=['cross_check'])

Out[747]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2022-04-13 09:00:00	0.25	5.00	NaN	NaN	NaN	0.10	
2021-11-27 09:00:00	1.00	0.25	0.25	1.00	8.00	1.00	
2021-11-19 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2021-11-16 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2021-11-12 09:00:00	0.25	NaN	0.25	2.00	1.00	1.00	
2021-11-09 09:00:00	0.25	2.00	NaN	0.25	NaN	1.00	
2021-01-28 09:00:00	19.00	20.00	29.00	19.00	13.00	17.00	
2020-11-19 09:00:00	0.25	1.00	1.00	3.00	0.25	NaN	
2020-10-18 09:00:00	3.00	0.10	0.25	1.00	NaN	NaN	
2020-04-25 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	
2020-04-12 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2020-04-11 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2020-03-21 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-03-07 09:00:00	7.00	NaN	NaN	NaN	NaN	NaN	NaN
2020-02-26 09:00:00	0.25	0.25	4.00	1.00	1.00	1.00	1.00
2020-01-24 09:00:00	3.00	1.00	2.00	6.00	2.00	4.00	4.00
2020-01-22 09:00:00	1.00	0.25	NaN	NaN	NaN	NaN	NaN
2019-12-27 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-12-12 09:00:00	NaN	0.25	0.25	1.00	0.25	NaN	NaN
2019-12-11 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2019-11-27 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-11-09 09:00:00	NaN	NaN	4.00	NaN	NaN	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2019-11-08 09:00:00	0.25	NaN	7.00	6.00	NaN	NaN	NaN
2019-11-01 09:00:00	1.00	1.00	10.00	3.00	1.00	1.00	1.00
2019-10-29 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-10-08 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-04-15 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-04-10 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2019-04-04 09:00:00	NaN	0.10	0.10	NaN	NaN	NaN	0.25
2019-04-01 09:00:00	5.00	0.10	1.00	NaN	NaN	2.00	NaN
2019-03-01 09:00:00	28.00	24.00	45.00	26.00	26.00	38.00	NaN
2018-11-14 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2018-11-01 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2018-10-28 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2018-04-22 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2017-11-21 09:00:00	2.00	NaN	2.00	NaN	NaN	NaN	NaN
2017-11-20 09:00:00	4.00	NaN	2.00	1.00	NaN	NaN	NaN
2017-10-28 09:00:00	2.00	1.00	0.25	3.00	0.25	NaN	NaN
2017-10-25 09:00:00	NaN	NaN	2.00	0.25	0.25	0.25	1.00
2017-02-23 09:00:00	23.00	20.00	26.00	25.00	28.00	27.00	27.00
2016-04-17 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-12-26 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2015-12-06 09:00:00	NaN	0.10	NaN	1.00	NaN	NaN	NaN
2015-11-19 09:00:00	1.00	2.00	4.00	2.00	1.00	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2015-11-08 09:00:00	NaN	NaN	NaN	0.25	NaN	NaN	NaN
2015-10-30 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2015-10-28 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2015-10-10 09:00:00	NaN	NaN	0.25	NaN	4.00	NaN	NaN
2015-10-07 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2015-03-24 09:00:00	1.00	0.25	4.00	3.00	1.00	1.00	1.00
2015-01-13 09:00:00	27.00	25.00	28.00	30.00	29.00	25.00	25.00
2014-12-01 09:00:00	1.00	NaN	NaN	0.25	0.25	NaN	NaN
2014-11-05 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2014-10-16 09:00:00	NaN	NaN	0.25	NaN	NaN	NaN	NaN
2014-03-21 09:00:00	14.00	5.00	4.00	5.00	4.00	5.00	5.00

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2014-03-10 09:00:00	0.25	0.10	0.10	NaN	NaN	NaN	NaN
2014-01-13 09:00:00	2.00	5.00	0.25	NaN	12.00	8.00	NaN
2013-11-25 09:00:00	0.25	NaN	NaN	NaN	NaN	NaN	NaN
2013-03-20 09:00:00	74.00	60.00	73.00	81.00	70.00	68.00	NaN
2013-03-19 09:00:00	73.00	58.00	70.00	77.00	69.00	66.00	NaN
2012-11-04 09:00:00	NaN	NaN	6.00	NaN	NaN	NaN	NaN
2012-04-07 09:00:00	33.00	16.00	35.00	32.00	32.00	30.00	NaN
2012-03-30 09:00:00	33.00	20.00	39.00	39.00	34.00	31.00	NaN
2012-03-29 09:00:00	33.00	16.00	37.00	36.00	33.00	29.00	NaN
2011-12-18 09:00:00	NaN	0.00	NaN	NaN	NaN	NaN	NaN
2011-12-03 09:00:00	NaN	1.00	1.00	2.00	1.00	NaN	NaN

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2011-11-27 09:00:00	NaN	1.00	3.00	1.00	NaN	NaN	NaN
2011-11-25 09:00:00	NaN	NaN	1.00	NaN	2.00	1.00	
2011-11-18 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	
2011-11-17 09:00:00	NaN	NaN	3.00	NaN	3.00	2.00	
2011-11-09 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	1.00
2011-10-18 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2010-11-18 09:00:00	NaN	3.00	NaN	NaN	NaN	NaN	NaN
2010-11-09 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	NaN
2010-10-30 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	NaN
2010-10-23 09:00:00	NaN	NaN	2.00	NaN	NaN	NaN	NaN
2010-03-11 09:00:00	51.00	38.00	43.00	52.00	43.00	37.00	

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2010-03-05 09:00:00	51.00	37.00	43.00	50.00	73.00	32.00	
2009-11-03 09:00:00	NaN	NaN	2.00	5.00	NaN	1.00	
2009-11-01 09:00:00	NaN	1.00	3.00	NaN	NaN	1.00	
2009-10-31 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	
2009-10-12 09:00:00	1.00	NaN	NaN	NaN	NaN	NaN	
2008-11-23 09:00:00	11.00	NaN	3.00	4.00	2.00	NaN	
2008-04-15 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	
2008-03-28 09:00:00	2.00	1.00	1.00	NaN	NaN	NaN	
2008-03-20 09:00:00	5.00	NaN	15.00	3.00	NaN	NaN	
2008-03-18 09:00:00	3.00	NaN	12.00	NaN	3.00	2.00	
2007-11-04 09:00:00	NaN	1.00	1.00	NaN	1.00	2.00	

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Moscow
2007-04-10 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-04-07 09:00:00	NaN	5.00	NaN	NaN	1.00	NaN	
2007-03-19 09:00:00	NaN	1.00	NaN	NaN	NaN	NaN	
2006-12-18 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2006-04-12 09:00:00	2.00	NaN	NaN	NaN	NaN	NaN	
2006-04-07 09:00:00	3.00	NaN	3.00	NaN	NaN	3.00	
2006-03-02 09:00:00	51.00	35.00	45.00	44.00	46.00	47.00	
2005-11-12 09:00:00	NaN	NaN	1.00	NaN	NaN	NaN	
2005-10-26 09:00:00	NaN	2.00	NaN	NaN	NaN	NaN	
2005-04-23 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	
2005-04-22 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	

```
Snow_height#V_Volochek Snow_height#Staritsa Snow_height#Kashyn Snow_height#Tver Snow_height#Klin Snow_height#Dmitrov Snow_height#Lipetsk
```

2005-02-21	NaN	26.00	NaN	46.00	33.00	22.00
00:00:00						

```
In [748]:
```

```
# Создадим список координат ячеек, содержащих значения, определённые как ошибки
# list_error_coords = df_tmp62.dropna(subset=['double_error_at']).double_error_at.tolist()
# list_error_coords

# Добавим список координат ячеек, содержащих значения, определённые как ошибки
list_error_coords.extend(df_tmp62.dropna(subset=['double_error_at']).double_error_at.tolist())
list_error_coords
```

```
Out[748]: [(Timestamp('2020-01-02 06:00:00'), ['Tver']),
(Timestamp('2018-03-16 06:00:00'), ['Serpukhov']),
(Timestamp('2016-11-30 03:00:00'), ['Staritsa']),
(Timestamp('2015-03-07 06:00:00'), ['Klin']),
(Timestamp('2013-04-05 06:00:00'), ['Staritsa']),
(Timestamp('2013-02-15 03:00:00'), ['Kashyn']),
(Timestamp('2011-03-30 21:00:00'), ['Nemchinovka']),
(Timestamp('2010-01-31 15:00:00'), ['Nemchinovka']),
(Timestamp('2008-08-24 03:00:00'), ['Serpukhov']),
(Timestamp('2008-02-13 15:00:00'), ['Nemchinovka']),
(Timestamp('2007-11-22 15:00:00'), ['Nemchinovka']),
(Timestamp('2006-01-29 12:00:00'), ['Tver']),
(Timestamp('2006-01-01 21:00:00'), ['N_Jerusalem']),
(Timestamp('2005-02-24 06:00:00'), ['Mozhaisk']),
(Timestamp('2005-02-12 06:00:00'), ['Klin']),
(Timestamp('2005-02-02 15:00:00'), ['Mozhaisk']),
(Timestamp('2022-04-13 09:00:00'), ['Staritsa']),
(Timestamp('2021-11-27 09:00:00'), ['Klin']),
(Timestamp('2021-11-19 09:00:00'), ['Serpukhov']),
(Timestamp('2021-11-16 09:00:00'), ['N_Jerusalem']),
(Timestamp('2021-11-12 09:00:00'), ['N_Jerusalem']),
(Timestamp('2021-11-09 09:00:00'), ['Staritsa']),
(Timestamp('2021-01-28 09:00:00'), ['Kashyn']),
(Timestamp('2020-11-19 09:00:00'), ['Tver']),
(Timestamp('2020-10-18 09:00:00'), ['V_Volochev']),
(Timestamp('2020-04-25 09:00:00'), ['V_Volochev']),
(Timestamp('2020-04-12 09:00:00'), ['V_Volochev']),
(Timestamp('2020-04-11 09:00:00'), ['Kashyn']),
(Timestamp('2020-03-21 09:00:00'), ['Mozhaisk']),
(Timestamp('2020-03-07 09:00:00'), ['V_Volochev']),
(Timestamp('2020-02-26 09:00:00'), ['Volokolamsk']),
(Timestamp('2020-01-24 09:00:00'), ['Tver']),
(Timestamp('2020-01-22 09:00:00'), ['V_Volochev']),
(Timestamp('2019-12-27 09:00:00'), ['V_Volochev']),
(Timestamp('2019-12-12 09:00:00'), ['Tver']),
(Timestamp('2019-12-11 09:00:00'), ['Kashyn']),
(Timestamp('2019-11-27 09:00:00'), ['V_Volochev']),
(Timestamp('2019-11-09 09:00:00'), ['Kashyn']),
(Timestamp('2019-11-08 09:00:00'), ['V_Volochev']),
(Timestamp('2019-11-01 09:00:00'), ['Kashyn']),
(Timestamp('2019-10-29 09:00:00'), ['V_Volochev']),
(Timestamp('2019-10-08 09:00:00'), ['Naro_Fominsk']),
(Timestamp('2019-04-15 09:00:00'), ['Serpukhov']),
(Timestamp('2019-04-10 09:00:00'), ['V_Volochev']),
```

```
(Timestamp('2019-04-04 09:00:00'), ['Dmitrov']),
(Timestamp('2019-04-01 09:00:00'), ['V_Volochev']),
(Timestamp('2019-03-01 09:00:00'), ['Kashyn']),
(Timestamp('2018-11-14 09:00:00'), ['V_Volochev']),
(Timestamp('2018-11-01 09:00:00'), ['Serpukhov']),
(Timestamp('2018-10-28 09:00:00'), ['V_Volochev']),
(Timestamp('2018-04-22 09:00:00'), ['Kashyn']),
(Timestamp('2017-11-21 09:00:00'), ['Serpukhov']),
(Timestamp('2017-11-20 09:00:00'), ['V_Volochev']),
(Timestamp('2017-10-28 09:00:00'), ['Serpukhov']),
(Timestamp('2017-10-25 09:00:00'), ['Kashyn']),
(Timestamp('2017-02-23 09:00:00'), ['Serpukhov']),
(Timestamp('2016-04-17 09:00:00'), ['N_Jerusalem']),
(Timestamp('2015-12-26 09:00:00'), ['Kashyn']),
(Timestamp('2015-12-06 09:00:00'), ['Tver']),
(Timestamp('2015-11-19 09:00:00'), ['Kashyn']),
(Timestamp('2015-11-08 09:00:00'), ['Tver']),
(Timestamp('2015-10-30 09:00:00'), ['Kashyn']),
(Timestamp('2015-10-28 09:00:00'), ['Kashyn']),
(Timestamp('2015-10-10 09:00:00'), ['Klin']),
(Timestamp('2015-10-07 09:00:00'), ['V_Volochev']),
(Timestamp('2015-03-24 09:00:00'), ['Kashyn']),
(Timestamp('2015-01-13 09:00:00'), ['Serpukhov']),
(Timestamp('2014-12-01 09:00:00'), ['V_Volochev']),
(Timestamp('2014-11-05 09:00:00'), ['Mozhaisk']),
(Timestamp('2014-10-16 09:00:00'), ['Kashyn']),
(Timestamp('2014-03-21 09:00:00'), ['Serpukhov']),
(Timestamp('2014-03-10 09:00:00'), ['V_Volochev']),
(Timestamp('2014-01-13 09:00:00'), ['Serpukhov']),
(Timestamp('2013-11-25 09:00:00'), ['V_Volochev']),
(Timestamp('2013-03-20 09:00:00'), ['Serpukhov']),
(Timestamp('2013-03-19 09:00:00'), ['Serpukhov']),
(Timestamp('2012-11-04 09:00:00'), ['Kashyn']),
(Timestamp('2012-04-07 09:00:00'), ['Staritsa']),
(Timestamp('2012-03-30 09:00:00'), ['Staritsa']),
(Timestamp('2012-03-29 09:00:00'), ['Staritsa']),
(Timestamp('2011-12-18 09:00:00'), ['Staritsa']),
(Timestamp('2011-12-03 09:00:00'), ['Naro_Fominsk']),
(Timestamp('2011-11-27 09:00:00'), ['Kashyn']),
(Timestamp('2011-11-25 09:00:00'), ['Serpukhov']),
(Timestamp('2011-11-18 09:00:00'), ['Kashyn']),
(Timestamp('2011-11-17 09:00:00'), ['Serpukhov']),
(Timestamp('2011-11-09 09:00:00'), ['Dmitrov']),
(Timestamp('2011-10-18 09:00:00'), ['Nemchinovka']),
```

```
(Timestamp('2010-11-18 09:00:00'), ['Staritsa']),
(Timestamp('2010-11-09 09:00:00'), ['V_Volochek']),
(Timestamp('2010-10-30 09:00:00'), ['Kashyn']),
(Timestamp('2010-10-23 09:00:00'), ['Kashyn']),
(Timestamp('2010-03-11 09:00:00'), ['Serpukhov']),
(Timestamp('2010-03-05 09:00:00'), ['Klin']),
(Timestamp('2009-11-03 09:00:00'), ['Tver']),
(Timestamp('2009-11-01 09:00:00'), ['Kashyn']),
(Timestamp('2009-10-31 09:00:00'), ['V_Volochek']),
(Timestamp('2009-10-12 09:00:00'), ['V_Volochek']),
(Timestamp('2008-11-23 09:00:00'), ['V_Volochek']),
(Timestamp('2008-04-15 09:00:00'), ['Kashyn']),
(Timestamp('2008-03-28 09:00:00'), ['V_Volochek']),
(Timestamp('2008-03-20 09:00:00'), ['Volokolamsk']),
(Timestamp('2008-03-18 09:00:00'), ['Kashyn']),
(Timestamp('2007-11-04 09:00:00'), ['Dmitrov']),
(Timestamp('2007-04-10 09:00:00'), ['Serpukhov']),
(Timestamp('2007-04-07 09:00:00'), ['Staritsa']),
(Timestamp('2007-03-19 09:00:00'), ['Staritsa']),
(Timestamp('2006-12-18 09:00:00'), ['Naro_Fominsk']),
(Timestamp('2006-04-12 09:00:00'), ['V_Volochek']),
(Timestamp('2006-04-07 09:00:00'), ['Naro_Fominsk']),
(Timestamp('2006-03-02 09:00:00'), ['Staritsa']),
(Timestamp('2005-11-12 09:00:00'), ['Kashyn']),
(Timestamp('2005-10-26 09:00:00'), ['Staritsa']),
(Timestamp('2005-04-23 09:00:00'), ['Naro_Fominsk']),
(Timestamp('2005-04-22 09:00:00'), ['Naro_Fominsk']),
(Timestamp('2005-02-21 09:00:00'), ['Tver'])]
```

In [749...]

```
# Восстановим исходное состояние df_tmp62
df_tmp62 = dict_df_parameters[param_df_name].copy(deep=True)
```

In [750...]

```
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_tmp62.at[error_coords[0], PARAMETER62+'#'+ station] = np.nan
```

In [751...]

```
# Проверим, все ли ошибки заменены на NaN
# Количество нeNaN значений в df_tmp62 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 неNaN значений и прибавим их к счётчику неNaN значений.
counter_notna1 += np.sum(pd.notna(df_tmp62.at[error_coords[0], PARAMETER62+'#'+ station]))
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna1}')

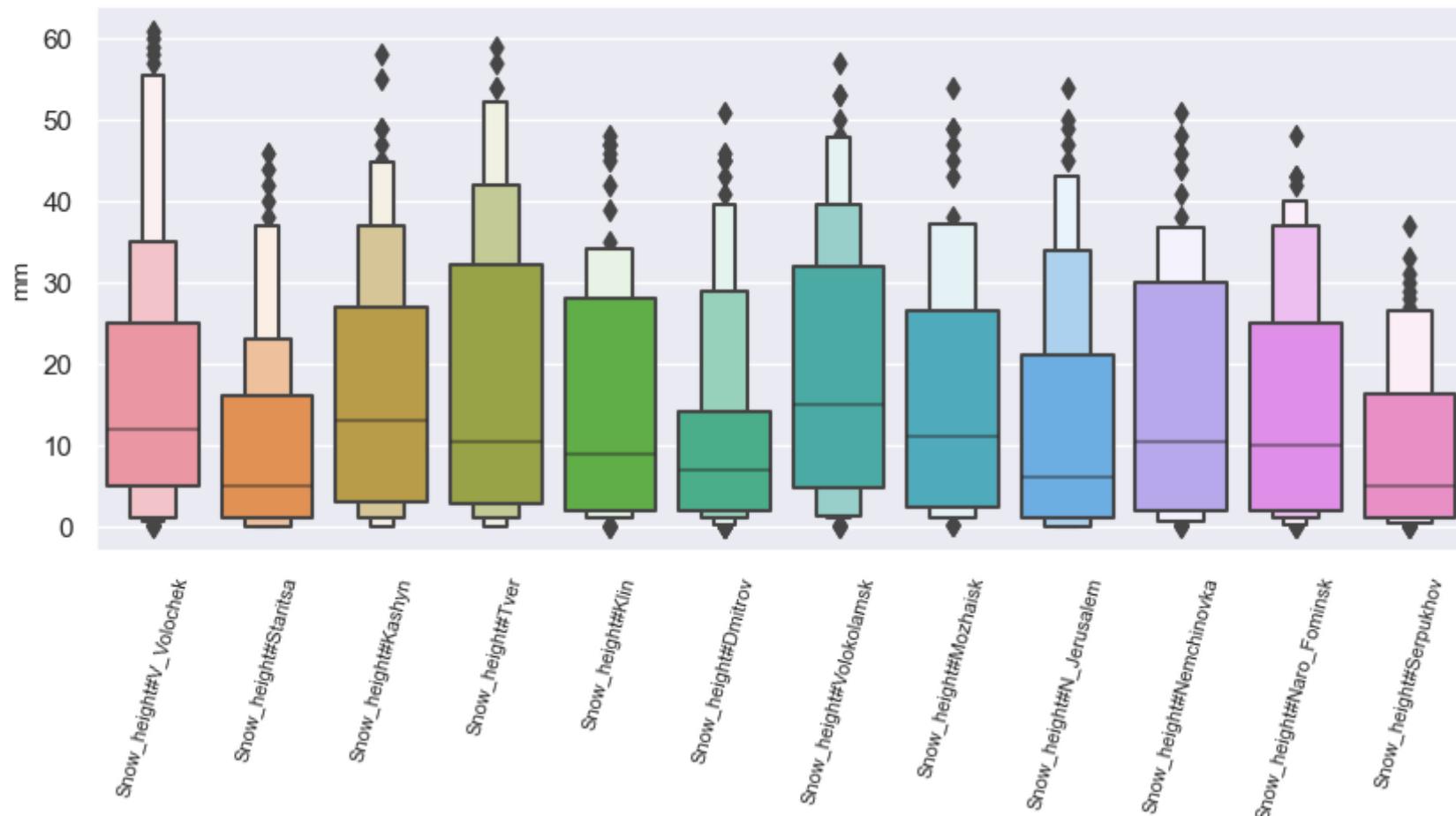
# df_tmp62
```

По результатам удаления выявленных аномальных выбросов осталось значений: 0

Визуализируем архив высоты снегового покрова (Snow_height) по сезонам после удаления ошибок

```
In [752...]: # В цикле выведем графики давления по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp62.dropna(how='all')).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp62.dropna(how='all').iloc[:, :12][season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=75, size=8)
    dummy = g.set_ylabel('мм', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER62} в разрезе метеостанций:\n'
                        f'{season_name}')
plt.show()
```

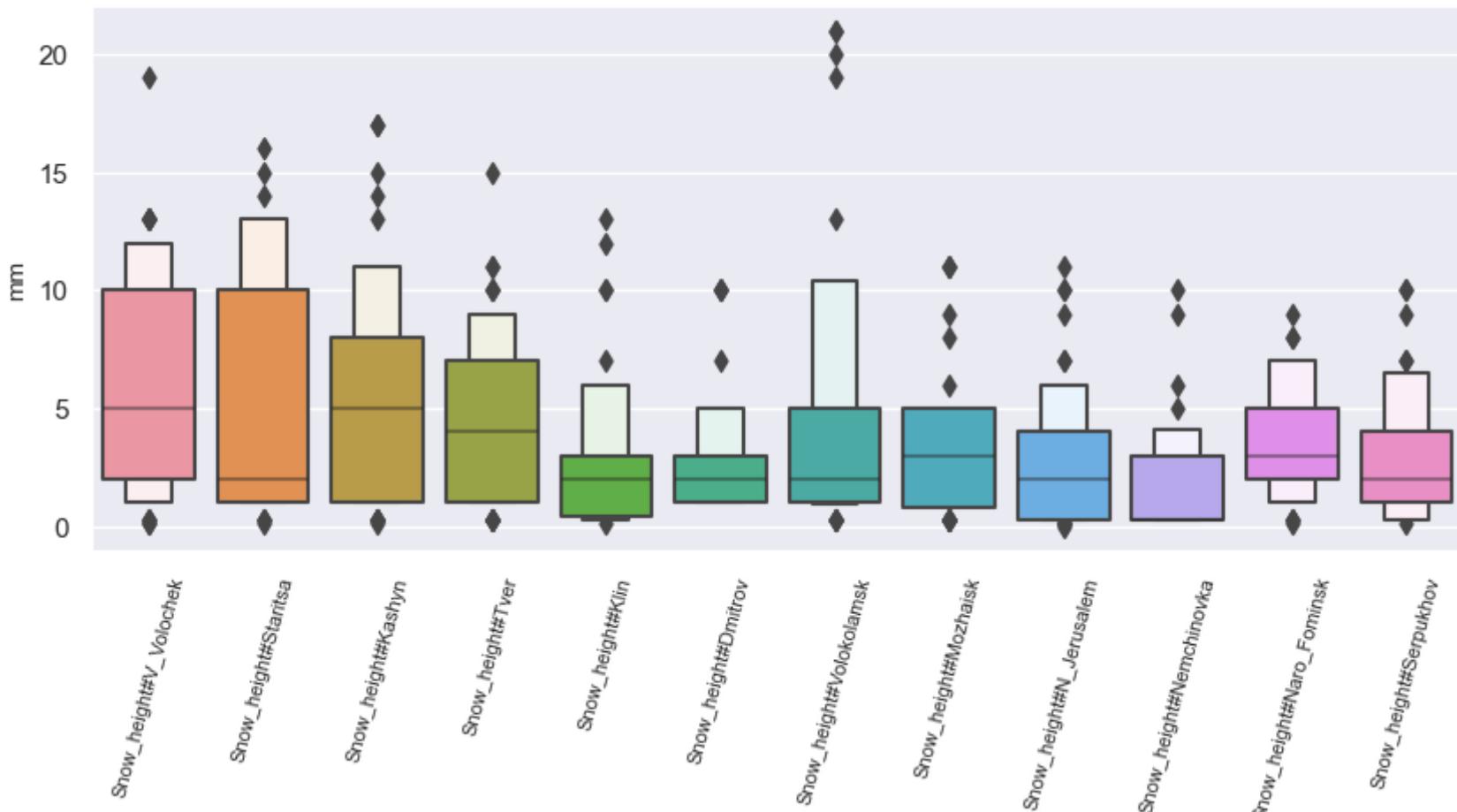
Распределение значений Snow_height в разрезе метеостанций:
spring



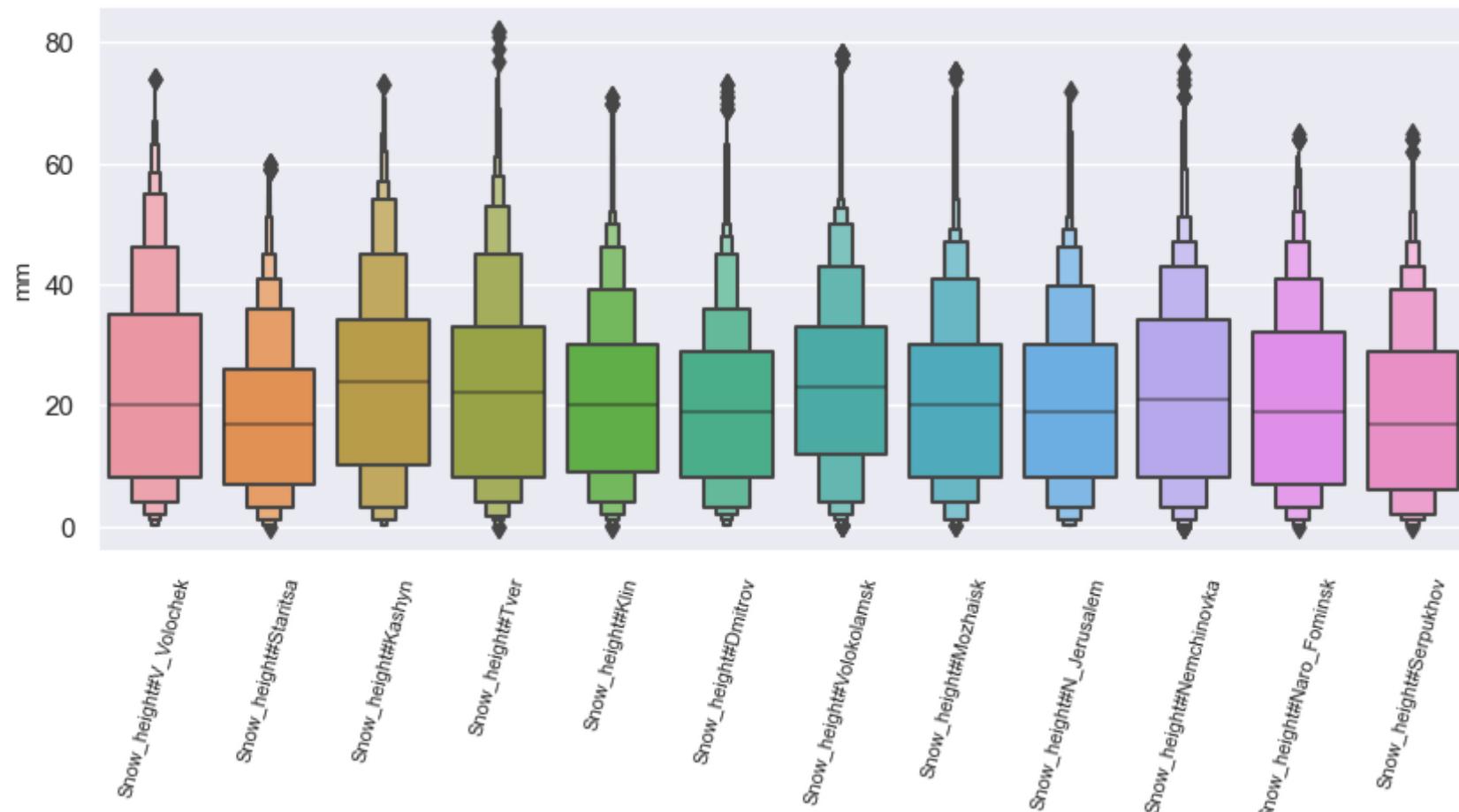
Распределение значений Snow_height в разрезе метеостанций:
summer



Распределение значений Snow_height в разрезе метеостанций:
autumn



Распределение значений Snow_height в разрезе метеостанций: winter



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

In [753]:

```
print(f'Минимальное значение: {np.nanmin(df_tmp62)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp62)},\n'
      f'Средняя: {np.nanmean(df_tmp62)},\n'
      f'Медиана: {np.nanmedian(df_tmp62)}')
```

Минимальное значение: 0.0,
Максимальное значение: 82.0,
Средняя: 20.801220528167587,
Медиана: 19.0

Сохраним очищенные данные в файл параметров

In [754...]

```
# # 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_'+PARAMETER62 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER62].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER62}.csv'  
)  
print('DONE!')  
  
df_Snow_height.csv -> DONE!
```

6.2.2. Восстановление "сплошных" NaN методом средней между соседними моментами наблюдения для показателя высоты снегового покрова (Snow_height)

Модели пространственной экстраполяции не могут экстраполировать значения в рамках одного момента наблюдения, если значения во всех точках наблюдения неизвестны. Есть ли такие случаи в архиве температуры почвы (в периоды, когда такие наблюдения проводились)?

In [755...]

```
# Определим временный DF для стандартных моментов наблюдения  
  
df_tmp62d = df_tmp62[df_tmp62.index.hour == 9]  
  
# Подсчитаем количество строк со "сплошными" NaN в df_tmp62d  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк
```

```
all_nans_count = sum(df_tmp62d.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp62d.keys())))
print(f'Количество дней со сплошными NaN равно {all_nans_count}')
```

Количество дней со сплошными NaN равно 3810

Визуализируем структуру пропущенных данных о снеговом покрове.

In [756]

```
msno.matrix(df_tmp62d,
            color=(0, 100/225, 0),
            figsize=(15, 7),
            fontsize=10); # выводим график из библиотеки "missingno"
plt.show()
```

Out[756]: <AxesSubplot:>



Из графика следует, что наблюдения имеют сезонный характер. Проверим количество наблюдений, приходящийся на каждый сезон.

In [757...]

```
for season, mask in season_masks(df_tmp62d).items():
    print(f'Количество наблюдений за высотой снегового покрова. Сезон: {season}')
    pd.notna(df_tmp62d[mask]).sum() # количество неNaN значений в разбиении по сезонам
    msno.matrix(
        df_tmp62d[mask],
        color=(10/225, 10/225, 150/225),
        figsize=(15, 5),
```

```
    fontsize=10); # выводим график из библиотеки "missingno"
plt.show()
```

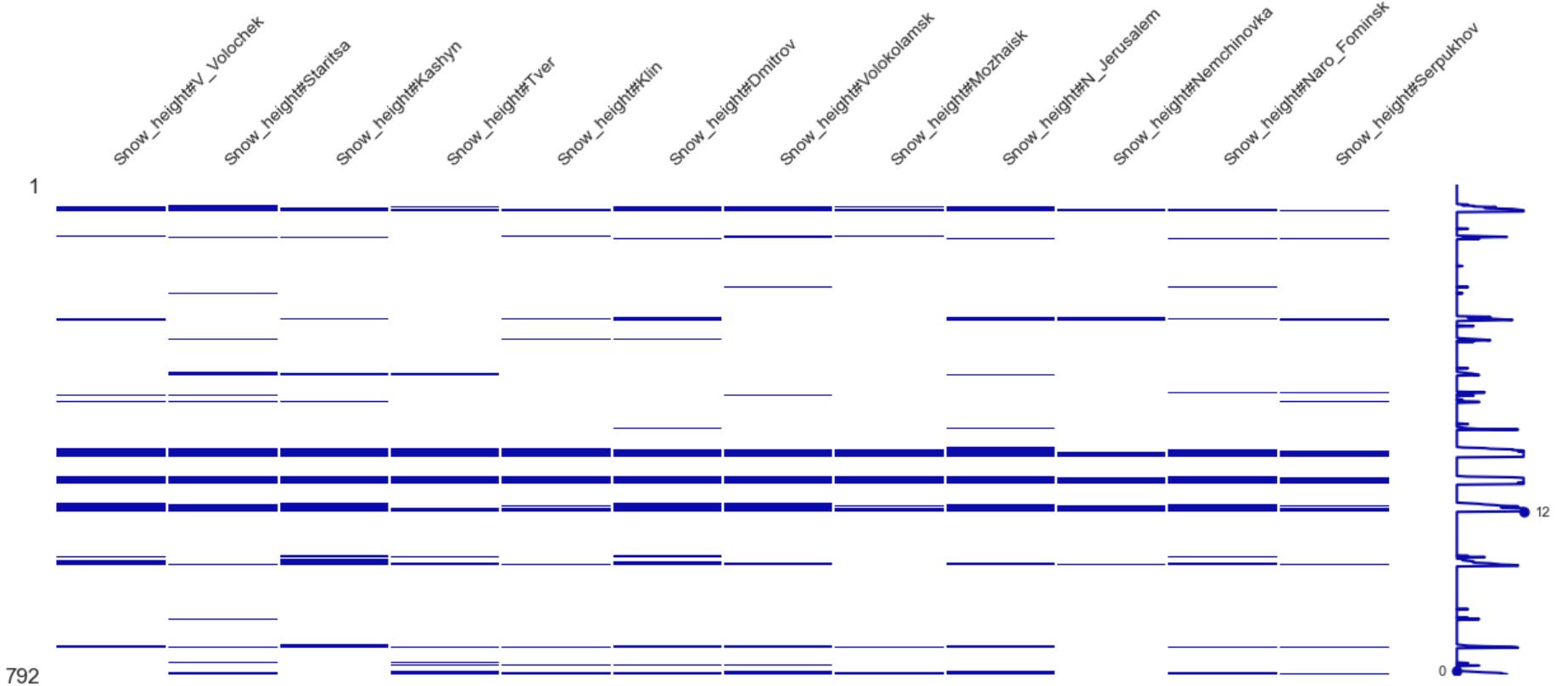
Количество наблюдений за высотой снегового покрова. Сезон: spring

```
Out[757]:
```

Snow_height#V_Volochek	73
Snow_height#Staritsa	81
Snow_height#Kashyn	83
Snow_height#Tver	64
Snow_height#Klin	59
Snow_height#Dmitrov	88
Snow_height#Volokolamsk	68
Snow_height#Mozhaisk	51
Snow_height#N_Jerusalem	81
Snow_height#Nemchinovka	46
Snow_height#Naro_Fominsk	66
Snow_height#Serpukhov	52

```
dtype: int64
<AxesSubplot:>
```

```
Out[757]:
```



Количество наблюдений за высотой снегового покрова. Сезон: summer

Out[757]:

```
Snow_height#V_Volochev      0
Snow_height#Staritsa        0
Snow_height#Kashyn          0
Snow_height#Tver            0
Snow_height#Klin             0
Snow_height#Dmitrov         0
Snow_height#Volokolamsk     0
Snow_height#Mozhaisk         0
Snow_height#N_Jerusalem      0
Snow_height#Nemchinovka     0
Snow_height#Naro_Fominsk    0
Snow_height#Serpukhov        0
dtype: int64
```

Out[757]:



Количество наблюдений за высотой снегового покрова. Сезон: autumn

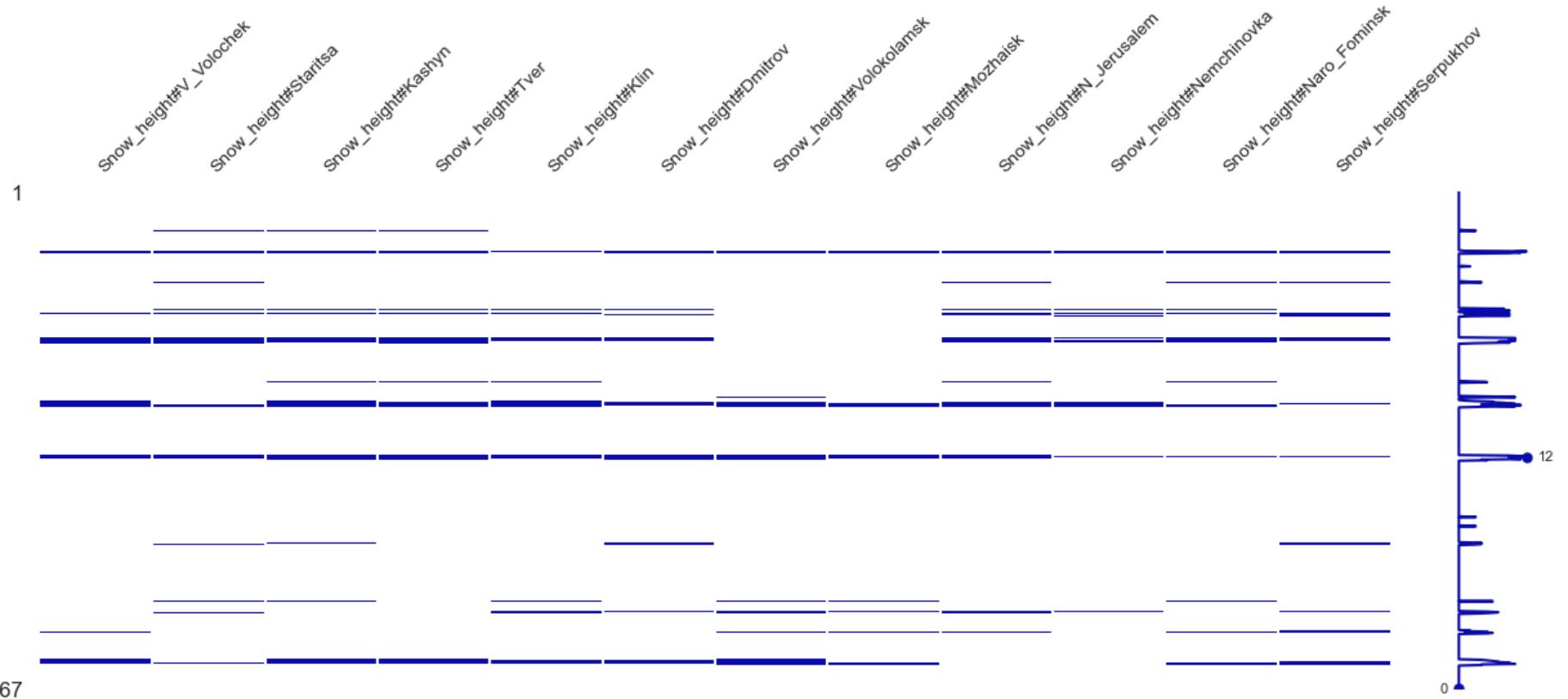
Out[757]:

Snow_height#V_Volochev	48
Snow_height#Staritsa	37
Snow_height#Kashyn	61
Snow_height#Tver	50
Snow_height#Klin	50
Snow_height#Dmitrov	50
Snow_height#Volokolamsk	40
Snow_height#Mozhaisk	28
Snow_height#N_Jerusalem	49
Snow_height#Nemchinovka	32
Snow_height#Naro_Fominsk	43
Snow_height#Serpukhov	37

dtype: int64

<AxesSubplot:>

Out[757]:



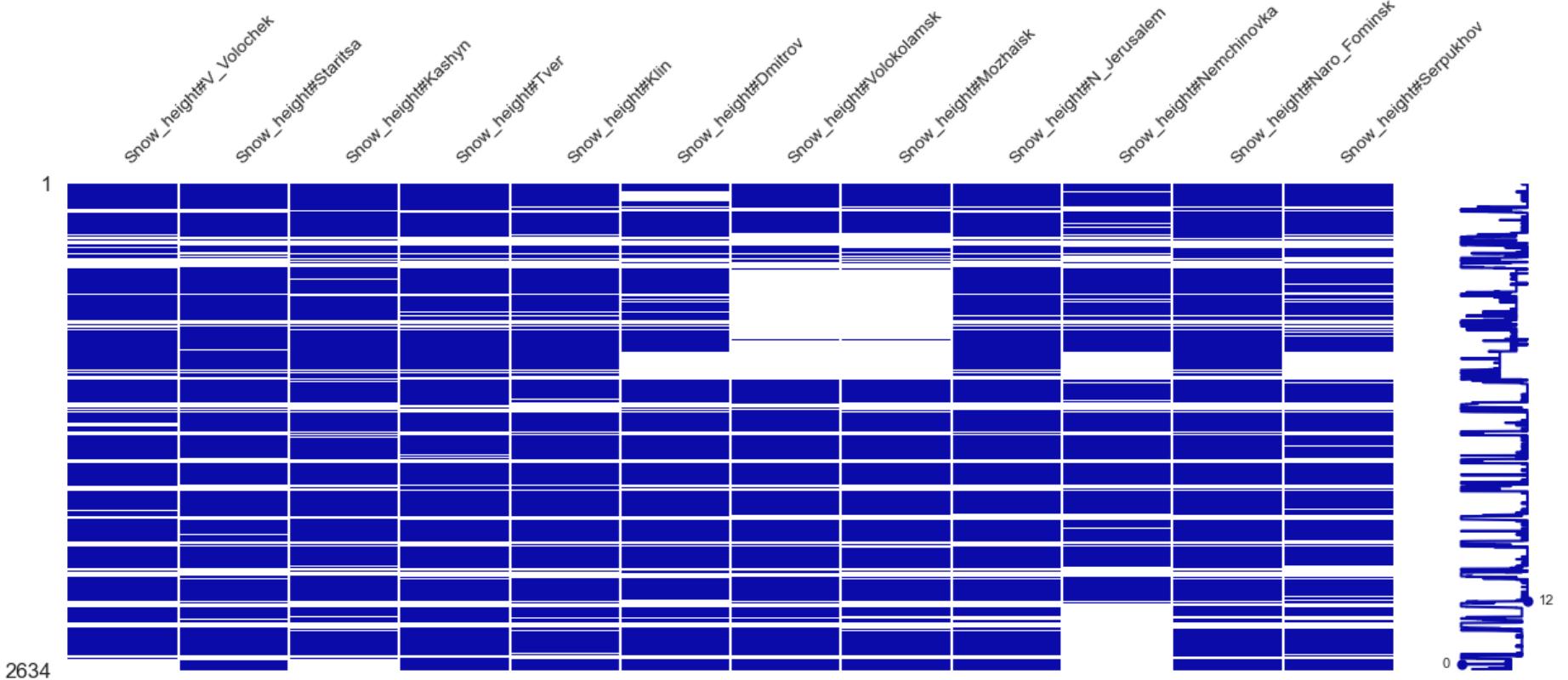
Количество наблюдений за высотой снегового покрова. Сезон: winter

Out[757]:

Snow_height#V_Volochev	2064
Snow_height#Staritsa	2121
Snow_height#Kashyn	2115
Snow_height#Tver	2123
Snow_height#Klin	2105
Snow_height#Dmitrov	1993
Snow_height#Volokolamsk	1618
Snow_height#Mozhaisk	1530
Snow_height#N_Jerusalem	2108
Snow_height#Nemchinovka	1627
Snow_height#Naro_Fominsk	2117
Snow_height#Serpukhov	1869

dtype: int64

Out[757]:



Заменим строки сплошных NaN средними значениями из соседних моментов наблюдения, кроме периода отсутствия наблюдений летом.

```
In [758...]: for year in df_tmp62d.index.year.unique(): # по перечню уникальных лет в df_tmp62d, по годам
    # определим логические маски: соответствие данному и предыдущему году, и не менее 1 неNaN значения в ряду
    yearly_mask = (
        (df_tmp62d.index.year == year) &
        (df_tmp62d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))
    )
    prev_yearly_mask = (
        (df_tmp62d.index.year == (year - 1)) &
        (df_tmp62d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))
    ) if year > df_tmp62d.index.year.min() else (
        (df_tmp62d.index.year == year) &
        (df_tmp62d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))
    ) # учитываем неполный сезон начального года
```

```

# Определим границы периодов наблюдения (начало во второй половине года, окончание - в первой)
end_moment = (
    df_tmp62d[yearly_mask & (df_tmp62d.index.month<=6)]
    .dropna(how='all')
    .index
    .max()
) # максимальный момент в первой половине года

start_moment = (
    df_tmp62d[prev_yearly_mask & (df_tmp62d.index.month>=7)]
    .dropna(how='all')
    .index
    .min()
) if year != df_tmp62d.index.year.min() else (
    df_tmp62d[prev_yearly_mask & (df_tmp62d.index.month<=6)]
    .dropna(how='all')
    .index
    .min()
) # минимальный момент во второй половине года, а для начала архива - начало архива снегового покрова

# создаём логическую маску для текущего сезона
snow_season_mask = ((df_tmp62d.index >= start_moment) & (df_tmp62d.index <= end_moment))

# Список: выбираем из датафрейма строки, где количество NaN равно количеству столбцов - то есть сплошные NaN,
# если эти строки находятся внутри промежутка наблюдений для каждого сезона
list_time_total_nans = (
    df_tmp62d[snow_season_mask & (df_tmp62d.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp62d.keys()))]
    .index.tolist()
)
f'{year-1}-{year}: {start_moment} to {end_moment}'
#     list_time_total_nans

# По списку
for idx in list_time_total_nans:
    # определяем начало и конец временного интервала
    start_time = idx - pd.Timedelta('1D')
    end_time = idx + pd.Timedelta('1D')

    # Пока в двух строках нет значений одновременно хотя бы в одном общем столбце
    # Проверяем, какое количество элементов содержит пересечение индекса серии start_time без NaN и
    # индекса серии end_time без NaN, и пока оно не будет содержать хотя бы один общий элемент
    # сдвигаем границы.
    # (np.isin выдает булев массив для каждого элемента в первом списке, содержится ли он во втором списке)

```

```

while ( # пока нет общих элементов в строках или пока хотя бы одна из границ периода это сплошные NaN
       (np.isin(df_tmp62d.loc[start_time].dropna().index,
                 df_tmp62d.loc[end_time].dropna().index).sum() == 0
    ) |
    (df_tmp62d.loc[start_time].notna().sum() == 0) |
    (df_tmp62d.loc[end_time].notna().sum() == 0)): # границы периода не должны быть сплошными NaN

    if start_time > start_moment: # Пока не дошли до начала наблюдений данного сезона
        start_time = start_time - pd.Timedelta('1D') # Уменьшаем start_time на 1 день
    if end_time < end_moment: # Пока не дошли до оконания наблюдений данного сезона
        end_time = end_time + pd.Timedelta('1D') # Увеличиваем end_time на 1 день

    # по ряду сплошных NaN заменяем NaN в df_tmp62d на средние значения их соседних двух рядов
    for label in df_tmp62d.loc[idx].index:
        df_tmp62d.at[idx, label] = np.mean([df_tmp62d.at[start_time, label], df_tmp62d.at[end_time, label]])
#     df_tmp62d.loc[idx]
    fix_nan_counter += len(list_time_total_nans) # увеличиваем счётчик исправленных сплошных NaN

```

```

Out[758]: '2021-2022: 2021-11-09 09:00:00 to 2022-04-16 09:00:00'
Out[758]: '2020-2021: 2020-10-18 09:00:00 to 2021-04-21 09:00:00'
Out[758]: '2019-2020: 2019-10-07 09:00:00 to 2020-04-05 09:00:00'
Out[758]: '2018-2019: 2018-10-30 09:00:00 to 2019-04-15 09:00:00'
Out[758]: '2017-2018: 2017-10-22 09:00:00 to 2018-04-11 09:00:00'
Out[758]: '2016-2017: 2016-10-26 09:00:00 to 2017-05-10 09:00:00'
Out[758]: '2015-2016: 2015-10-09 09:00:00 to 2016-04-16 09:00:00'
Out[758]: '2014-2015: 2014-10-17 09:00:00 to 2015-04-21 09:00:00'
Out[758]: '2013-2014: 2013-11-14 09:00:00 to 2014-04-14 09:00:00'
Out[758]: '2012-2013: 2012-10-26 09:00:00 to 2013-04-20 09:00:00'
Out[758]: '2011-2012: 2011-11-12 09:00:00 to 2012-04-17 09:00:00'
Out[758]: '2010-2011: 2010-10-13 09:00:00 to 2011-04-18 09:00:00'
Out[758]: '2009-2010: 2009-11-01 09:00:00 to 2010-04-03 09:00:00'
Out[758]: '2008-2009: 2008-11-18 09:00:00 to 2009-04-21 09:00:00'

```

```
Out[758]: '2007-2008: 2007-10-14 09:00:00 to 2008-03-28 09:00:00'  
Out[758]: '2006-2007: 2006-10-30 09:00:00 to 2007-04-23 09:00:00'  
Out[758]: '2005-2006: 2005-10-26 09:00:00 to 2006-04-09 09:00:00'  
Out[758]: '2004-2005: 2005-02-01 09:00:00 to 2005-04-24 09:00:00'
```

```
In [759...]: df_tmp62.update(df_tmp62d) # обновим значения df_tmp62 за счёт значений df_tmp62d
```

```
In [760...]: print(f'Изначальное количество дней со сплошными NaN равно {all_nans_count}')  
# Снова, по результатам исправления  
# подсчитаем количество строк со "сплошными" NaN в df_tmp62d  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк  
all_nans_count = sum(df_tmp62d.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp62d.keys()))  
print(f'Оставшееся количество дней со сплошными NaN равно {all_nans_count}')  
print(f'Исправлено дней со сплошным NaN {fix_nan_counter}')  
print(f'{all_nans_count - fix_nan_counter} дней с отсутствием наблюдений приходится на летний период')
```

Изначальное количество дней со сплошными NaN равно 3810

Оставшееся количество дней со сплошными NaN равно 3320

Исправлено дней со сплошным NaN 557

2763 дней с отсутствием наблюдений приходится на летний период

Проверим, на какие месяцы приходятся оставшиеся сплошные NaN

```
In [761...]: set(df_tmp62[(df_tmp62.index.hour==9) &  
                    (df_tmp62.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp62.keys()))]  
        .index.month  
        .tolist())  
)
```

```
Out[761]: {3, 4, 5, 6, 7, 8, 9, 10, 11}
```

6.2.3. Подбор модели для восстановления пропущенных и удалённых значений показателя температуры почвы, а также для моделирования значений для искомой точки

6.2.3.1. Модель средней, взвешенной по степени обратных расстояний (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_tmp62.
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 [762...]

```
# Создаём датафрейм для валидации модели IDW
df_test = df_tmp62.dropna(how='any')
```

In [763...]

```
df_test.info()
df_test.shape
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1081 entries, 2022-04-07 09:00:00 to 2007-11-14 09:00:00
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Snow_height#V_Volochek    1081 non-null   float64
 1   Snow_height#Staritsa     1081 non-null   float64
 2   Snow_height#Kashyn      1081 non-null   float64
 3   Snow_height#Tver        1081 non-null   float64
 4   Snow_height#Klin        1081 non-null   float64
 5   Snow_height#Dmitrov     1081 non-null   float64
 6   Snow_height#Volokolamsk 1081 non-null   float64
 7   Snow_height#Mozhaisk    1081 non-null   float64
 8   Snow_height#N_Jerusalem 1081 non-null   float64
 9   Snow_height#Nemchinovka 1081 non-null   float64
 10  Snow_height#Naro_Fominsk 1081 non-null   float64
 11  Snow_height#Serpukhov   1081 non-null   float64
dtypes: float64(12)
memory usage: 109.8 KB
(1081, 12)
```

Out[763]:

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

```
In [764...]: # 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[764]: array([0, 1, 2, ..., 1078, 1079, 1080])

Out[764]: array([5, 4, 0, ..., 3, 8, 6])

```
Out[764]: array([[ 0,  5],
   [ 1,  4],
   [ 2,  0],
   ...,
  [1078,  3],
  [1079,  8],
  [1080,  6]])
```

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 [765...]

```
# 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[765]: (648, 2)

Out[765]: (648,)

Out[765]: (433, 2)

Out[765]: (433,)

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

In [766...]

```
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=PARAMETER62,
                             station=df_test.keys()[y][len(PARAMETER62)+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=00:00:04

Текущие значения: iterations=1, power=3.25,
Максимальная ошибка (MAX_E) IDW = 21.8534082
Средняя абсолютная ошибка (MAE) IDW = 3.9961299
Средний квадрат ошибки (MSE) IDW = 29.7108706
Средняя квадратическая ошибка (RMSE) IDW = 5.4507679
Коэффициент детерминации (R2) IDW = 0.8728712

Elapsed time=00:00:08

Текущие значения: iterations=2, power=3.5,
Максимальная ошибка (MAX_E) IDW = 21.8324515
Средняя абсолютная ошибка (MAE) IDW = 4.0014570
Средний квадрат ошибки (MSE) IDW = 29.6781458
Средняя квадратическая ошибка (RMSE) IDW = 5.4477652
Коэффициент детерминации (R2) IDW = 0.8730112

Elapsed time=00:00:12

Текущие значения: iterations=3, power=3.75,
Максимальная ошибка (MAX_E) IDW = 21.8146535
Средняя абсолютная ошибка (MAE) IDW = 4.0083013
Средний квадрат ошибки (MSE) IDW = 29.6827635
Средняя квадратическая ошибка (RMSE) IDW = 5.4481890
Коэффициент детерминации (R2) IDW = 0.8729914

ЛУЧШИЕ значения: power=3.5,
Максимальная ошибка (MAX_E) IDW = 21.8324515
Средняя абсолютная ошибка (MAE) IDW = 4.0014570
Средний квадрат ошибки (MSE) IDW = 29.6781458
Средняя квадратическая ошибка (RMSE) IDW = 5.4477652
Коэффициент детерминации (R2) IDW = 0.8730112

Несколько запусков кода выше, с различными параметрами степени (от 1 до 10), показали, что, судя по R2, лучшим значением степени на обучающем массиве является 3.75. Оно даёт лучшие метрики качества.

Применим эту степень для валидационного массива.

Валидационный датасет

In [767...]

```
# 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_=PARAMETER62,
                         station_=df_test.keys()[y][len(PARAMETER62)+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 = 22.6863638
 Средняя абсолютная ошибка (MAE) IDW = 3.8278205
 Средний квадрат ошибки (MSE) IDW = 27.3411031
 Средняя квадратическая ошибка (RMSE) IDW = 5.2288721
 Коэффициент детерминации (R2) IDW = 0.8891143

ВЫВОД

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

6.2.3.2. Кrigинг 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 [768]:

```
# 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[768]:

LoE LaN

station	LoE	LaN
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 [769]: # 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 [770...]

```

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:.7f}, 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=00:00:14

Значения метрик:

R2=nan, MAX_E=nan, MAE=nan, MSE=nan, RMSE=nan

Количество значений, которые не удалось предсказать: 29

Это составляет 4.4753% от обучающего массива данных

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

In [771...]

```
# 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 [772...]

```
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}\n'
    )
```

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

Максимальная ошибка (MAX_E) Kriging = 20.9985079, Kriging - IDW = -0.8161456

Средняя абсолютная ошибка (MAE) Kriging = 4.1372472, Kriging - IDW = 0.1289459

Средний квадрат ошибки (MSE) Kriging = 30.7402145, Kriging - IDW = 1.0574510

Средняя квадратическая ошибка (RMSE) Kriging = 5.5443859, Kriging - IDW = 0.0961968

Коэффициент детерминации (R2) Kriging = 0.8693818, Kriging - IDW = -0.0036096

Валидационный датасет

Посмотрим, как модель будет отрабатывать на валидационной выборке.

In [773...]

```
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.OrdinaryKriging(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 [774...]

```
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}, MAE={mae_kriging_vld:.7f}, MSE={mse_kriging_vld}, '  
      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=00:00:09

Значения метрик:

R2=nan, MAX_E=nan, MAE=nan, MSE=nan, RMSE=nan

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ

Количество значений, которые не удалось предсказать: 17

Это составляет 3.9261% от валидационного массива данных

In [775...]

```
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 [776...]

```
print(f'Кrigинг 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 = 21.6274348, Kriging - IDW = -1.0589290
Средняя абсолютная ошибка (MAE) Kriging = 4.0810801, Kriging - IDW = 0.2532596
Средний квадрат ошибки (MSE) Kriging = 30.2839684, Kriging - IDW = 2.9428654
Средняя квадратическая ошибка (RMSE) Kriging = 5.5030872, Kriging - IDW = 0.2742151
Коэффициент детерминации (R2) Kriging = 0.8790778, Kriging - IDW = -0.0100364

ВЫВОД

В данном случае модель не дала 100% результат, и не смогла предсказать все значения ни на тренировочном, ни на валидационном дтасете. Однако для нашего географического расположения точек наблюдения, из-за незначительного размера поля метеостанций, встречаются ситуации, когда не удается 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. Поэтому при применении на рабочем датасете, есть большая вероятность, что модели кригинга будет недостаточно для расчёта всех необходимых предсказаний.

6.2.3.3. 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 [777...]

```
# 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 [778...]

```
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_=PARAMETER62,
                             station_=df_test.keys()[y][len(PARAMETER62)+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 = 21.2704485
Средняя абсолютная ошибка (MAE) IDW = 5.9103893
Средний квадрат ошибки (MSE) IDW = 63.2526216
Средняя квадратическая ошибка (RMSE) IDW = 7.9531517
Коэффициент детерминации (R2) IDW = 0.6231886

Elapsed time=00:00:00

Текущие значения: iterations=2, power=2.25,
Максимальная ошибка (MAX_E) IDW = 20.9818614
Средняя абсолютная ошибка (MAE) IDW = 5.9238529
Средний квадрат ошибки (MSE) IDW = 62.7097451
Средняя квадратическая ошибка (RMSE) IDW = 7.9189485
Коэффициент детерминации (R2) IDW = 0.6264227

Elapsed time=00:00:00

Текущие значения: iterations=3, power=2.5,
Максимальная ошибка (MAX_E) IDW = 20.7007917
Средняя абсолютная ошибка (MAE) IDW = 5.9362943
Средний квадрат ошибки (MSE) IDW = 62.2122294
Средняя квадратическая ошибка (RMSE) IDW = 7.8874729
Коэффициент детерминации (R2) IDW = 0.6293865

Elapsed time=00:00:00

Текущие значения: iterations=4, power=2.75,
Максимальная ошибка (MAX_E) IDW = 20.4302070
Средняя абсолютная ошибка (MAE) IDW = 5.9478318
Средний квадрат ошибки (MSE) IDW = 61.7616979
Средняя квадратическая ошибка (RMSE) IDW = 7.8588611
Коэффициент детерминации (R2) IDW = 0.6320704

Elapsed time=00:00:00

Текущие значения: iterations=5, power=3.0,
Максимальная ошибка (MAX_E) IDW = 20.1726593
Средняя абсолютная ошибка (MAE) IDW = 5.9586013
Средний квадрат ошибки (MSE) IDW = 61.3593339
Средняя квадратическая ошибка (RMSE) IDW = 7.8332199
Коэффициент детерминации (R2) IDW = 0.6344674

Elapsed time=00:00:01

Текущие значения: iterations=6, power=3.25,

Максимальная ошибка (MAX_E) IDW = 19.9302142
Средняя абсолютная ошибка (MAE) IDW = 5.9687479
Средний квадрат ошибки (MSE) IDW = 61.0057822
Средняя квадратическая ошибка (RMSE) IDW = 7.8106198
Коэффициент детерминации (R2) IDW = 0.6365736

Elapsed time=00:00:01
Текущие значения: iterations=7, power=3.5,
Максимальная ошибка (MAX_E) IDW = 19.7044173
Средняя абсолютная ошибка (MAE) IDW = 5.9784183
Средний квадрат ошибки (MSE) IDW = 60.7010704
Средняя квадратическая ошибка (RMSE) IDW = 7.7910892
Коэффициент детерминации (R2) IDW = 0.6383888

Elapsed time=00:00:01
Текущие значения: iterations=8, power=3.75,
Максимальная ошибка (MAX_E) IDW = 19.4962958
Средняя абсолютная ошибка (MAE) IDW = 5.9877524
Средний квадрат ошибки (MSE) IDW = 60.4445721
Средняя квадратическая ошибка (RMSE) IDW = 7.7746107
Коэффициент детерминации (R2) IDW = 0.6399169

Elapsed time=00:00:01
Текущие значения: iterations=9, power=4.0,
Максимальная ошибка (MAX_E) IDW = 19.3063881
Средняя абсолютная ошибка (MAE) IDW = 5.9968766
Средний квадрат ошибки (MSE) IDW = 60.2350183
Средняя квадратическая ошибка (RMSE) IDW = 7.7611222
Коэффициент детерминации (R2) IDW = 0.6411652

Elapsed time=00:00:01
Текущие значения: iterations=10, power=4.25,
Максимальная ошибка (MAX_E) IDW = 19.1347962
Средняя абсолютная ошибка (MAE) IDW = 6.0058994
Средний квадрат ошибки (MSE) IDW = 60.0705532
Средняя квадратическая ошибка (RMSE) IDW = 7.7505195
Коэффициент детерминации (R2) IDW = 0.6421450

Elapsed time=00:00:02
Текущие значения: iterations=11, power=4.5,
Максимальная ошибка (MAX_E) IDW = 18.9812505
Средняя абсолютная ошибка (MAE) IDW = 6.0149083
Средний квадрат ошибки (MSE) IDW = 59.9488237
Средняя квадратическая ошибка (RMSE) IDW = 7.7426626

Коэффициент детерминации (R2) IDW = 0.6428701

Elapsed time=00:00:02

Текущие значения: iterations=12, power=4.75,
Максимальная ошибка (MAX_E) IDW = 18.8451817
Средняя абсолютная ошибка (MAE) IDW = 6.0239692
Средний квадрат ошибки (MSE) IDW = 59.8670890
Средняя квадратическая ошибка (RMSE) IDW = 7.7373826
Коэффициент детерминации (R2) IDW = 0.6433571

Elapsed time=00:00:02

Текущие значения: iterations=13, power=5.0,
Максимальная ошибка (MAX_E) IDW = 18.7257921
Средняя абсолютная ошибка (MAE) IDW = 6.0331275
Средний квадрат ошибки (MSE) IDW = 59.8223376
Средняя квадратическая ошибка (RMSE) IDW = 7.7344901
Коэффициент детерминации (R2) IDW = 0.6436237

Elapsed time=00:00:02

Текущие значения: iterations=14, power=5.25,
Максимальная ошибка (MAX_E) IDW = 18.6221216
Средняя абсолютная ошибка (MAE) IDW = 6.0424093
Средний квадрат ошибки (MSE) IDW = 59.8113999
Средняя квадратическая ошибка (RMSE) IDW = 7.7337830
Коэффициент детерминации (R2) IDW = 0.6436888

Elapsed time=00:00:02

Текущие значения: iterations=15, power=5.5,
Максимальная ошибка (MAX_E) IDW = 18.5331070
Средняя абсолютная ошибка (MAE) IDW = 6.0518248
Средний квадрат ошибки (MSE) IDW = 59.8310506
Средняя квадратическая ошибка (RMSE) IDW = 7.7350534
Коэффициент детерминации (R2) IDW = 0.6435717

ЛУЧШИЕ значения: power=5.25,

Максимальная ошибка (MAX_E) IDW = 18.6221216
Средняя абсолютная ошибка (MAE) IDW = 6.0424093
Средний квадрат ошибки (MSE) IDW = 59.8113999
Средняя квадратическая ошибка (RMSE) IDW = 7.7337830
Коэффициент детерминации (R2) IDW = 0.6436888

Опробуем новое значение лучшей степени для IDW на сокращённом валидационном датасете

In [779...]

```
# 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_=PARAMETER62,
                          station_=df_test.keys()[y][len(PARAMETER62)+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 = 13.3662121
Средняя абсолютная ошибка (MAE) IDW = 5.3235998
Средний квадрат ошибки (MSE) IDW = 39.4166421
Средняя квадратическая ошибка (RMSE) IDW = 6.2782674
Коэффициент детерминации (R2) IDW = 0.7391718

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 [780...]

```
# Восстановим y_predict для лучшего R2 IDW
y_predict_idw_tr_shrunk = [
    inverse_distance_avg(row=df_test.iloc[x],
                          param_=PARAMETER62,
```

```
        station_=df_test.keys()[y][len(PARAMETER62)+1:],
        df_dists_= df_station_dists,
        power_=best_power_idw_tr_shrunk)
    for x, y in x_train_idw_shrunk
]
```

In [781...]

```
# 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 [782...]

```
# 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 [783...]

```
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 = 20.9985079
Средняя абсолютная ошибка (MAE) IDW = 4.2225091
Средний квадрат ошибки (MSE) IDW = 32.0412398
Средняя квадратическая ошибка (RMSE) IDW = 5.6604982
Коэффициент детерминации (R2) IDW = 0.8628998

ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX_E) IDW = 21.6274348
Средняя абсолютная ошибка (MAE) IDW = 4.1298626
Средний квадрат ошибки (MSE) IDW = 30.6425261
Средняя квадратическая ошибка (RMSE) IDW = 5.5355692
Коэффициент детерминации (R2) IDW = 0.8757249

Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:

Максимальная ошибка (MAX_E) IDW = 22.6863638
Средняя абсолютная ошибка (MAE) IDW = 3.8278205
Средний квадрат ошибки (MSE) IDW = 27.3411031
Средняя квадратическая ошибка (RMSE) IDW = 5.2288721
Коэффициент детерминации (R2) IDW = 0.8891143

ВЫВОД

Там, где с помощью кригинга удаётся получить предсказания, он по метрикам качества частично превосходит, а частично хуже модели IDW. При этом сочетание этих двух моделей (применение IDW там, где кригинг оказывается бессильным) даёт в целом не лучшее качество предсказания, чем только модель IDW. Однако, IDW проигрывает по максимальной абсолютной ошибке. Ниже применим к уже реальному датасету всё же комбинацию этих двух моделей.

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

In [784]:

```
# Добавим в df_tmp62 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 PARAMETER62#Chashnikovo и PARAMETER62#Rfrnce_point
df_tmp62 = (df_tmp62.
             assign(col_name1 = np.nan,
                   col_name2 = np.nan).
             rename(columns={"col_name1": PARAMETER62+'#'+ 'Chashnikovo',
                            "col_name2": PARAMETER62+'#'+ 'Rfrnce_point'}))
         )

df_tmp62.sample(3, random_state=56)
```

Out[784]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#
--	------------------------	----------------------	--------------------	------------------	------------------	---------------------	--------------

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

In [785]:

```
# Добавим в архив параметров Snow_height столбец для будущей центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями pr.nan
# Переименуем столбец PARAMETER62#Chashnikovo и PARAMETER62#Rfrnce_point
dict_df_parameters['df_'+PARAMETER62] = (dict_df_parameters['df_'+PARAMETER62].
                                         assign(col_name1 = np.nan,
                                               col_name2 = np.nan).
                                         rename(columns={"col_name1": PARAMETER62+'#'+ 'Chashnikovo',
                                                        "col_name2": PARAMETER62+'#'+ 'Rfrnce_point'}))
                                         )

dict_df_parameters['df_'+PARAMETER62].sample(3, random_state=56)
```

Out[785]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Chashnikovo
2014-02-22 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-05-16 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-06-18 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [786...]

```
# В архивах метеостанций df Чашниково и df для центральной точки
# создадим столбцы с называнием PARAMETER62

dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER62})
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point'].
                                         assign(col_name = np.nan).
                                         rename(columns={"col_name": PARAMETER62}
                                         )
                                         )

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

Out[786]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point	Soil_T	Snow_height
2014-02-22 03:00:00	-4.156558	-4.156558	-2.109643	768.336709	747.597791	0.812775	76.635021	-7.639754	NaN	NaN
2015-05-16 03:00:00	9.181730	9.181730	10.434571	745.095535	725.921747	-1.043037	95.837940	8.552121	NaN	NaN
2020-06-18 18:00:00	27.439052	25.366130	30.241632	761.480701	743.060948	-0.492627	58.118042	18.459938	NaN	NaN

Out[786]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point	Soil_T	Snow_height
2014-02-22 03:00:00	-3.589183	-3.794602	-2.700734	767.373725	754.041734	0.616282	75.006151	-7.367199	NaN	NaN
2015-05-16 03:00:00	7.789650	7.789650	8.797603	746.708428	734.256500	-0.826706	94.058645	6.893682	NaN	NaN
2020-06-18 18:00:00	29.404954	25.572651	29.941094	760.796222	749.008692	-0.723724	41.950321	15.114705	NaN	NaN

Перенесение уже исправленных сплошных NaN в архивы

In [787...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER62] = df_tmp62.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[:, PARAMETER62] = df_tmp62.loc[:, PARAMETER62 + '#' + name_df[3:]]
```

In [788...]

```
# Подсчитаем количество строк со "сплошными" NaN dict_df_parameters['df_'+PARAMETER62] для 9 часов момента наблюдения
# 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_'+PARAMETER62][(dict_df_parameters['df_'+PARAMETER62].index.hour == 9)]
    .apply(lambda x: sum(x.isna()), axis=1)==len(dict_df_parameters['df_'+PARAMETER62].keys()))
)

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

Количество строк со сплошными NaN равно 3320

In [789...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp62 для 9 часов момента наблюдения
# построчно подсчитаем сумму количества NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(
    df_tmp62[(df_tmp62.index.hour == 9)]
    .apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp62.keys()))
)

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

Количество строк со сплошными NaN равно 3320

In [790...]

```
set(df_tmp62[(df_tmp62.index.hour==9) &
              (df_tmp62.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp62.keys()))])
```

```
.index.month  
.tolist()  
)  
  
Out[790]: {3, 4, 5, 6, 7, 8, 9, 10, 11}
```

Суммы дней с отсутствием наблюдений высоты снегового покрова равны полученному по результатам исправления сплошных NaN значениям.

Частичное заполнение оставшихся NaN расчётными 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. Этую будут значения, которые модель уже никак не сможет рассчитать. Их можно будет восстановить методом IDW.

Учитывая, что наблюдения за температурой почвы имеют сезонный характер без чётко фиксированных границ начала и окончания сезона, сделаем этот расчёт для каждого года отдельно. Воспользуемся циклом и сокращённым временным датафреймом df_tmp61d.

```
In [791...]  
# ПЕРЕОПРЕДЕЛИМ временный DF для стандартных моментов наблюдения  
df_tmp62d = df_tmp62[df_tmp62.index.hour == 9]
```

```
In [792...]  
start_time = time.time() # для замера времени выполнения кода  
  
for year in df_tmp62d.index.year.unique(): # по перечню уникальных лет в df_tmp62d, по годам  
    # определим логические маски: соответствие данному и предыдущему году, и не менее 1 неNaN значения в ряду  
    yearly_mask = (  
        (df_tmp62d.index.year == year) &  
        (df_tmp62d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))  
    )  
    prev_yearly_mask = (  
        (df_tmp62d.index.year == (year - 1)) &  
        (df_tmp62d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))  
    ) if year > df_tmp62d.index.year.min() else (  
        (df_tmp62d.index.year == year) &  
        (df_tmp62d.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))  
    ) # учитываем неполный сезон начального года  
  
    # Определим границы периодов наблюдения (начало во второй половине года, окончание - в первой)
```

```

end_moment = (
    df_tmp62d[yearly_mask & (df_tmp62d.index.month<=6)]
    .dropna(how='all')
    .index
    .max()
) # максимальный момент в первой половине года

start_moment = (
    df_tmp62d[prev_yearly_mask & (df_tmp62d.index.month>=7)]
    .dropna(how='all')
    .index
    .min()
) if year != df_tmp62d.index.year.min() else (
    df_tmp62d[prev_yearly_mask & (df_tmp62d.index.month<=6)]
    .dropna(how='all')
    .index
    .min()
) # минимальный момент во второй половине года, а для начала архива - начало архива снегового покрова

# создаём логическую маску для текущего сезона
snow_season_mask = ((df_tmp62d.index >= start_moment) & (df_tmp62d.index <= end_moment))

df_tmp62y = df_tmp62d[snow_season_mask]
# Подсчитаем количество NaN в df_tmp62d[snow_season_mask] и присвоим их переменной old_nan_count
# np.isnan(df_tmp62y).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
new_nan_count = np.sum(np.isnan(df_tmp62y).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

print(f'Зима {year-1} - {year}: Кригинг')
# заменим значения в архивах, на исправленные и вычисленные из данных в df_tmp62d[snow_season_mask]
# 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, используем переменную dummy, чтобы избежать лишнего вывода
    dummy = df_tmp62y.apply(
        lambda x: row_nan_kriging_correct(row_=x,

```

```
        name_param_=PARAMETER62
            ),
    axis=1
)
old_nan_count = new_nan_count # сохраняем прежнее количество NaN в old_nan_count
new_nan_count = np.sum(np.isnan(df_tmp62y).sum()) # подсчитаем оставшиеся количество NaN

print(f'Кrigингом не удалось смоделировать {new_nan_count} значений, они переданы функции IDW\n')

# Для NaN, которые не могут быть заменены с помощью кригинга:
# Произведём вычисление отсутствующих значений в df_tmp62y через модель IDW.
# Заменим соответствующие значения в архивах на вновь вычисленные.
# используем функцию row_nan_idw_correct

# Построчно применяем функцию обработки NaN, используем переменную dummy, чтобы избежать лишнего вывода
dummy = df_tmp62y.apply(
    lambda x: row_nan_idw_correct(row_=x,
                                    name_param_=PARAMETER62,
                                    power_=best_power_idw_tr_shrunk),
    axis=1
)
# Перенесём полученные значения в df_tmp62
df_tmp62.update(df_tmp62y)

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')
```

Зима 2021 - 2022: Кригинг

Итерация № 1: осталось NaN: 516

Итерация № 2: осталось NaN: 137

Итерация № 3: осталось NaN: 125

Итерация № 4: осталось NaN: 124

Кригингом не удалось смоделировать 124 значений, они переданы функции IDW

Зима 2020 - 2021: Кригинг

Итерация № 1: осталось NaN: 1016

Итерация № 2: осталось NaN: 632

Итерация № 3: осталось NaN: 617

Итерация № 4: осталось NaN: 614

Кригингом не удалось смоделировать 614 значений, они переданы функции IDW

Зима 2019 - 2020: Кригинг

Итерация № 1: осталось NaN: 1532

Итерация № 2: осталось NaN: 1244

Итерация № 3: осталось NaN: 1220

Итерация № 4: осталось NaN: 1219

Итерация № 5: осталось NaN: 1218

Кригингом не удалось смоделировать 1218 значений, они переданы функции IDW

Зима 2018 - 2019: Кригинг

Итерация № 1: осталось NaN: 972

Итерация № 2: осталось NaN: 427

Итерация № 3: осталось NaN: 411

Кригингом не удалось смоделировать 411 значений, они переданы функции IDW

Зима 2017 - 2018: Кригинг

Итерация № 1: осталось NaN: 985

Итерация № 2: осталось NaN: 367

Итерация № 3: осталось NaN: 285

Итерация № 4: осталось NaN: 276

Итерация № 5: осталось NaN: 275

Кригингом не удалось смоделировать 275 значений, они переданы функции IDW

Зима 2016 - 2017: Кригинг

Итерация № 1: осталось NaN: 1255

Итерация № 2: осталось NaN: 572

Итерация № 3: осталось NaN: 545

Итерация № 4: осталось NaN: 542

Кригингом не удалось смоделировать 542 значений, они переданы функции IDW

Зима 2015 - 2016: Кригинг

Итерация № 1: осталось NaN: 1673

Итерация № 2: осталось NaN: 821

Итерация № 3: осталось NaN: 717

Итерация № 4: осталось NaN: 713

Итерация № 5: осталось NaN: 712

Кригингом не удалось смоделировать 712 значений, они переданы функции IDW

Зима 2014 - 2015: Кригинг

Итерация № 1: осталось NaN: 1045

Итерация № 2: осталось NaN: 665

Итерация № 3: осталось NaN: 638

Итерация № 4: осталось NaN: 635

Итерация № 5: осталось NaN: 634

Кригингом не удалось смоделировать 634 значений, они переданы функции IDW

Зима 2013 - 2014: Кригинг

Итерация № 1: осталось NaN: 711

Итерация № 2: осталось NaN: 383

Итерация № 3: осталось NaN: 340

Итерация № 4: осталось NaN: 324

Итерация № 5: осталось NaN: 322

Итерация № 6: осталось NaN: 320

Кригингом не удалось смоделировать 320 значений, они переданы функции IDW

Зима 2012 - 2013: Кригинг

Итерация № 1: осталось NaN: 650

Итерация № 2: осталось NaN: 238

Итерация № 3: осталось NaN: 235

Кригингом не удалось смоделировать 235 значений, они переданы функции IDW

Зима 2011 - 2012: Кригинг

Итерация № 1: осталось NaN: 614

Итерация № 2: осталось NaN: 295

Итерация № 3: осталось NaN: 269

Итерация № 4: осталось NaN: 266

Кригингом не удалось смоделировать 266 значений, они переданы функции IDW

Зима 2010 - 2011: Кригинг

Итерация № 1: осталось NaN: 849

Итерация № 2: осталось NaN: 487

Итерация № 3: осталось NaN: 469

Кригингом не удалось смоделировать 469 значений, они переданы функции IDW

Зима 2009 - 2010: Кригинг

Итерация № 1: осталось NaN: 676

Итерация № 2: осталось NaN: 367

Итерация № 3: осталось NaN: 353

Итерация № 4: осталось NaN: 352

Кригингом не удалось смоделировать 352 значений, они переданы функции IDW

Зима 2008 - 2009: Кригинг

Итерация № 1: осталось NaN: 532

Итерация № 2: осталось NaN: 183

Итерация № 3: осталось NaN: 169

Итерация № 4: осталось NaN: 166

Кригингом не удалось смоделировать 166 значений, они переданы функции IDW

Зима 2007 - 2008: Кригинг

Итерация № 1: осталось NaN: 704

Итерация № 2: осталось NaN: 318

Итерация № 3: осталось NaN: 249

Итерация № 4: осталось NaN: 229

Итерация № 5: осталось NaN: 222

Кригингом не удалось смоделировать 222 значений, они переданы функции IDW

Зима 2006 - 2007: Кригинг

Итерация № 1: осталось NaN: 1443

Итерация № 2: осталось NaN: 1033

Итерация № 3: осталось NaN: 1005

Итерация № 4: осталось NaN: 998

Итерация № 5: осталось NaN: 996

Кригингом не удалось смоделировать 996 значений, они переданы функции IDW

Зима 2005 - 2006: Кригинг

Итерация № 1: осталось NaN: 718

Итерация № 2: осталось NaN: 234

Итерация № 3: осталось NaN: 217

Итерация № 4: осталось NaN: 216

Кригингом не удалось смоделировать 216 значений, они переданы функции IDW

Зима 2004 - 2005: Кригинг

Итерация № 1: осталось NaN: 555

Итерация № 2: осталось NaN: 195

Итерация № 3: осталось NaN: 191

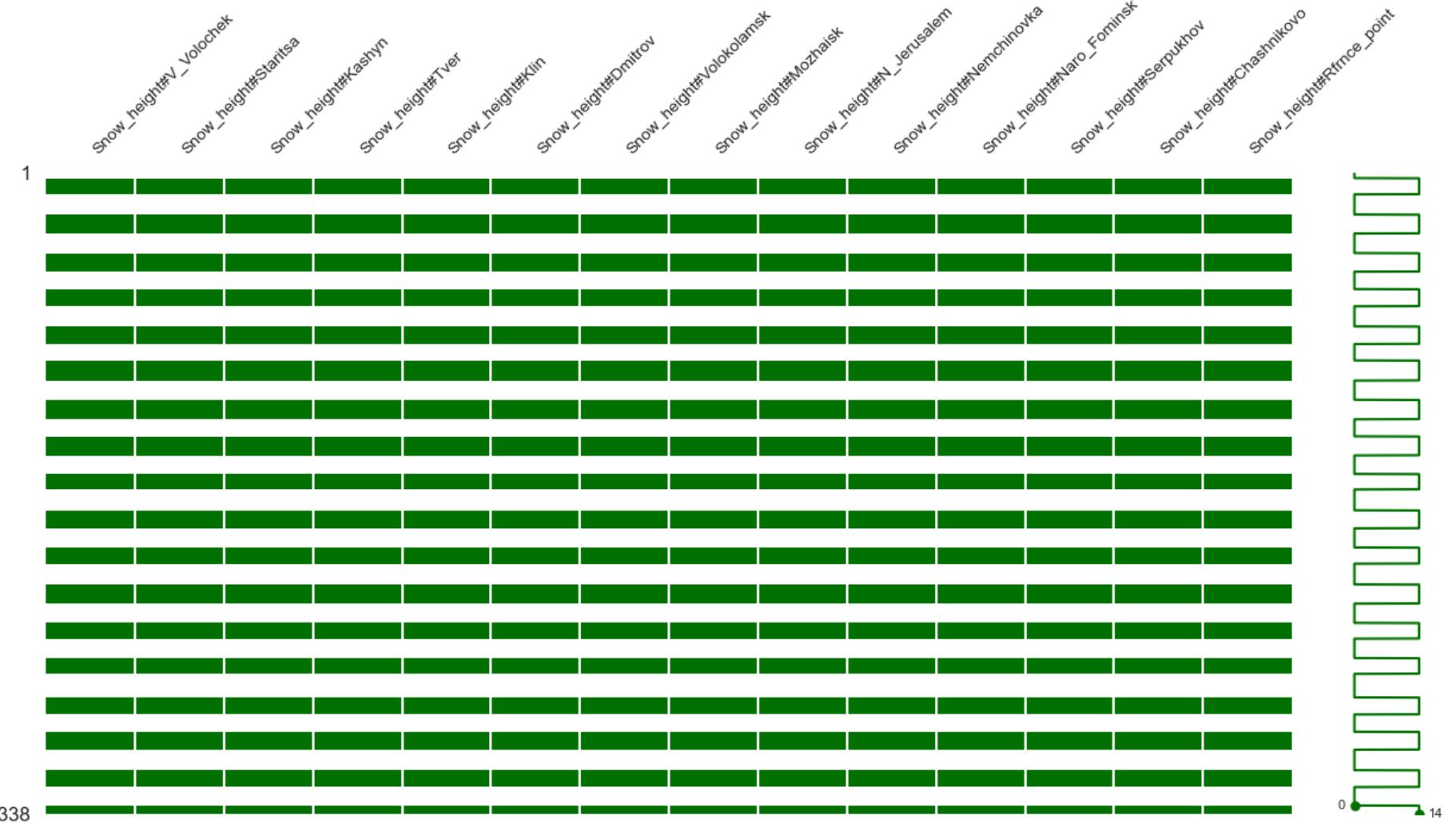
Кригингом не удалось смоделировать 191 значений, они переданы функции IDW

Elapsed time=00:02:36

Проверим полноту восстановления данных.

```
In [793]: msno.matrix(df_tmp62[df_tmp62.index.hour == 9],  
                     color=(0, 100/225, 0),  
                     figsize=(15, 7),  
                     fontsize=10); # выводим график из библиотеки "missingno"  
plt.show()
```

Out[793]: <AxesSubplot:>



Приведём минимальные значения к специальным значениям снегового покрова.

```
In [794...]: # В цикле заполним списки координат для 'Измерение невозможно или неточно.' = 0,09
# и для 'Снежный покров не постоянный.' = 0,1 по условиям ниже
list_special_coords009= []
list_special_coords01= []
for idx in df_tmp62.dropna(how='all').index:
    for col in df_tmp62.columns:
        if (df_tmp62.at[idx, col] < 0.1) and (df_tmp62.at[idx, col] != 0):
```

```

        list_special_coords009.append((idx, col))
    elif (df_tmp62.at[idx, col] < 0.25) and (df_tmp62.at[idx, col] > 0.09):
        list_special_coords01.append((idx, col))
    else:
        pass
# По спискам координат определим значения для
# Измерение невозможно или неточно. ' = 0,09 и
# 'Снежный покров не постоянный. ' = 0,1
for idx, col in list_special_coords009:
    df_tmp62.at[idx,col] = 0.09
for idx, col in list_special_coords01:
    df_tmp62.at[idx,col] = 0.1

```

In [795...]: `df_tmp62.dropna(how='all').sample(5, random_state=56)`

	Snow_height#V_Volochev	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Leningrad
2013-02-16 09:00:00	56.00	39.000000	55.00	53.0	46.0	46.0	46.0
2021-11-12 09:00:00	0.25	0.899457	0.25	2.0	1.0	1.0	1.0
2022-03-27 09:00:00	17.00	12.000000	12.00	12.0	3.0	8.0	8.0
2022-03-01 09:00:00	36.00	25.000000	29.00	23.0	17.0	17.0	17.0
2017-02-01 09:00:00	31.00	25.000000	31.00	31.0	36.0	35.0	35.0

In [796...]: `# Перенесём полученные значения df_tmp62 в архив параметров
dict_df_parameters['df_'+PARAMETER62].update(df_tmp62)`

In [797...]: `# Перенесём полученные значения df_tmp62 в архив метеостанций
for idx, col in list_special_coords009:`

```

dict_df_locations['df_'+ col[len(PARAMETER62)+1:]].at[idx, PARAMETER62] = 0.09
for idx, col in list_special_coords01:
    dict_df_locations['df_'+ col[len(PARAMETER62)+1:]].at[idx, PARAMETER62] = 0.1

```

Выведем, рандомные строки из df_tmp62

In [798...]

```

df_tmp62[df_tmp62.index.hour == 9].dropna(how='all').sample(5, random_state=56)
dict_df_parameters['df_'+PARAMETER62].dropna(how='all').sample(5, random_state=56)
dict_df_locations['df_Chashnikovo'].dropna(how='all').sample(5, random_state=21)

```

Out[798]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Chashnikovo
2013-02-16 09:00:00	56.00	39.000000	55.00	53.0	46.0	46.0	
2021-11-12 09:00:00	0.25	0.899457	0.25	2.0	1.0	1.0	
2022-03-27 09:00:00	17.00	12.000000	12.00	12.0	3.0	8.0	
2022-03-01 09:00:00	36.00	25.000000	29.00	23.0	17.0	17.0	
2017-02-01 09:00:00	31.00	25.000000	31.00	31.0	36.0	35.0	

Out[798]:

	Snow_height#V_Volochek	Snow_height#Staritsa	Snow_height#Kashyn	Snow_height#Tver	Snow_height#Klin	Snow_height#Dmitrov	Snow_height#Chashnikovo
2013-02-16 09:00:00	56.00	39.000000	55.00	53.0	46.0	46.0	
2021-11-12 09:00:00	0.25	0.899457	0.25	2.0	1.0	1.0	
2022-03-27 09:00:00	17.00	12.000000	12.00	12.0	3.0	8.0	
2022-03-01 09:00:00	36.00	25.000000	29.00	23.0	17.0	17.0	
2017-02-01 09:00:00	31.00	25.000000	31.00	31.0	36.0	35.0	

Out[798]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point	Soil_T	Snow_height
2013-06-07 06:00:00	12.457735	11.757238	17.900912	764.421382	744.973014	0.002886	91.830991	11.165276	NaN	NaN
2007-08-15 12:00:00	26.242078	16.933303	26.242078	757.079526	738.693914	0.303775	55.587077	16.638887	NaN	NaN
2018-07-25 15:00:00	23.863420	16.862702	23.863420	761.316804	742.682051	-0.396625	50.000322	12.799734	NaN	NaN
2008-12-21 09:00:00	-9.767089	-9.793221	-7.432882	768.020575	746.854737	0.766392	69.049727	-14.359883	NaN	3.488988
2007-10-24 03:00:00	8.505875	7.612161	8.505875	776.938707	756.898136	0.018927	92.747411	7.399059	NaN	NaN

6.2.5. Визуализация графиков температуры почвы Soil_T для условной метеостанции Чашниково

In [799...]

```

data1 = dict_df_parameters['df_'+PARAMETER62]
# Строим график в цикле для каждого года (для момента наблюдения в 9 часов)
for year in data1.index.year.unique(): # по перечню уникальных лет в data1, по годам
    # определим логические маски: соответствие данному и предыдущему году, и не менее 1 неNaN значения в ряду
    yearly_mask = (
        (data1.index.year == year) &

```

```

        (data1.index.hour == 9) &
        (data1.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))
    )
prev_yearly_mask = (
    (data1.index.year == (year - 1)) &
    (data1.index.hour == 9) &
    (data1.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))
) if year > data1.index.year.min() else (
    (data1.index.year == year) &
    (data1.index.hour == 9) &
    (data1.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))
) # учитываем неполный сезон начального года

# Определим границы периодов наблюдения (начало во второй половине года, окончание - в первой)
end_moment = (
    data1[yearly_mask & (data1.index.month<=6)]
    .dropna(how='all')
    .index
    .max()
) # максимальный момент в первой половине года

start_moment = (
    data1[prev_yearly_mask & (data1.index.month>=7)]
    .dropna(how='all')
    .index
    .min()
) if year != data1.index.year.min() else (
    data1[prev_yearly_mask & (data1.index.month<=6)]
    .dropna(how='all')
    .index
    .min()
) # минимальный момент во второй половине года, а для начала архива - начало архива снегового покрова

# создаём логическую маску для текущего сезона
snow_season_mask = ((data1.index >= start_moment) &
                     (data1.index <= end_moment) &
                     (data1.index.hour == 9))

data_y = data1[snow_season_mask][PARAMETER62+"#"+"Chashnikovo"]

fig, ax = plt.subplots(figsize=(10, 4))
g1 = sns.lineplot(data=data_y,

```

```
        color='mediumblue',
        linewidth=0.75,
        ax=ax)

# Добавляем легенду
blue_line = mlines.Line2D([], [], color='mediumblue', label=f'{PARAMETER62}', linewidth=0.75)

dummy = ax.legend(handles=[blue_line])

dummy = ax.set_ylabel('Высота снегового покрова, см', size=10)
dummy = ax.set_xlabel('Месяцы', size=10)
dummy = plt.title(f'Чашниково: Ежедневная динамика параметра \n'
                  f'высоты снегового покрова {PARAMETER62} на 9 часов утра, \n'
                  f'зима {year-1}-{year} годов')
plt.show()
```

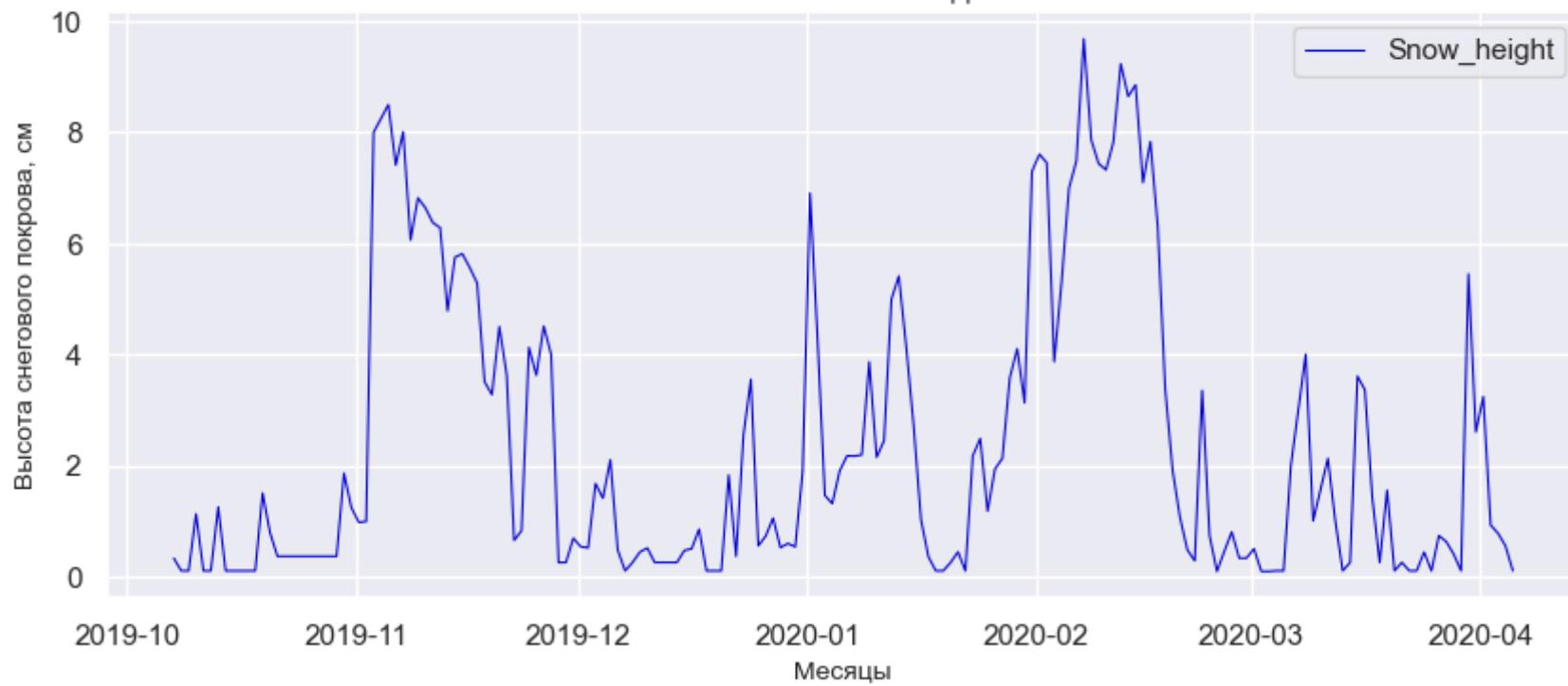
Чашниково: Ежедневная динамика параметра
высоты снежного покрова Snow_height на 9 часов утра,
зима 2021-2022 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2020-2021 годов



Чашниково: Ежедневная динамика параметра
высоты снежного покрова Snow_height на 9 часов утра,
зима 2019-2020 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2018-2019 годов



Чашниково: Ежедневная динамика параметра
высоты снежного покрова Snow_height на 9 часов утра,
зима 2017-2018 годов



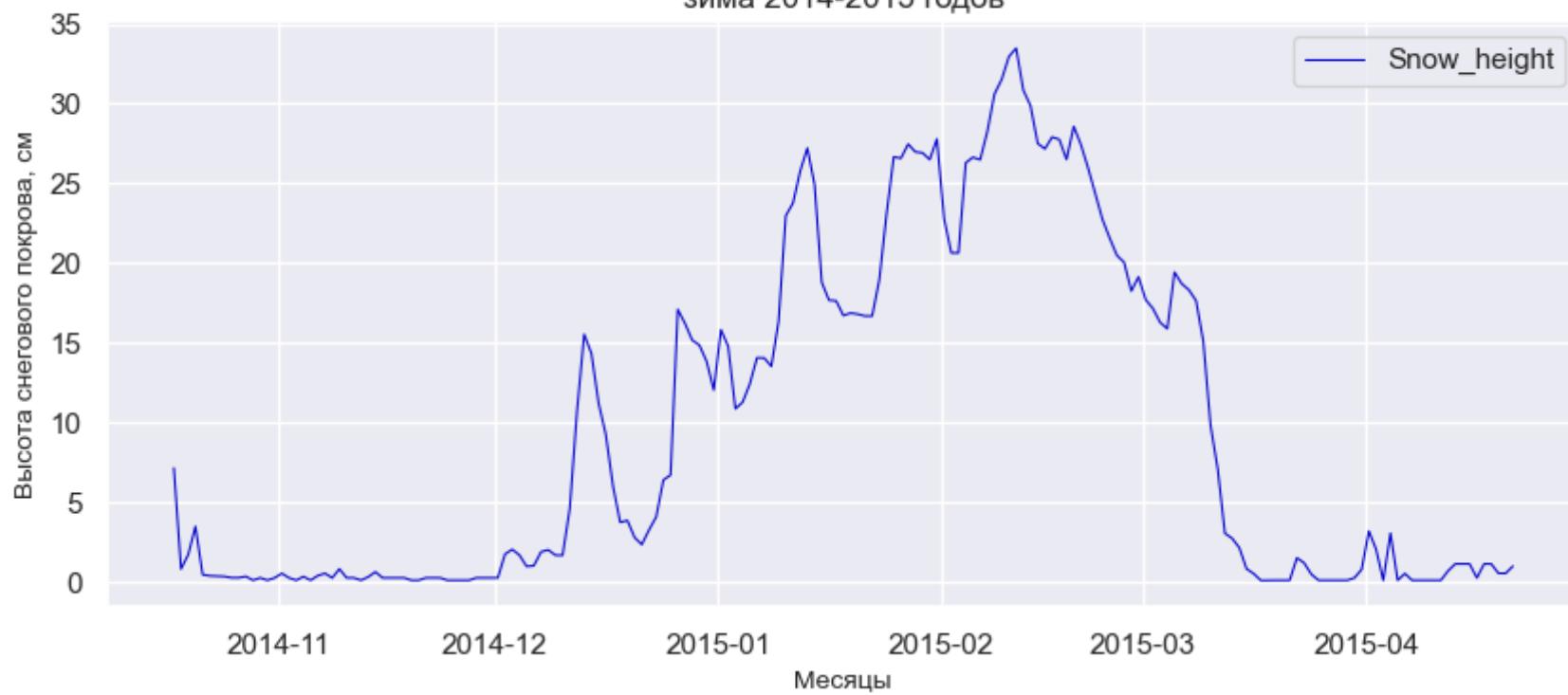
Чашниково: Ежедневная динамика параметра
высоты снежного покрова Snow_height на 9 часов утра,
зима 2016-2017 годов



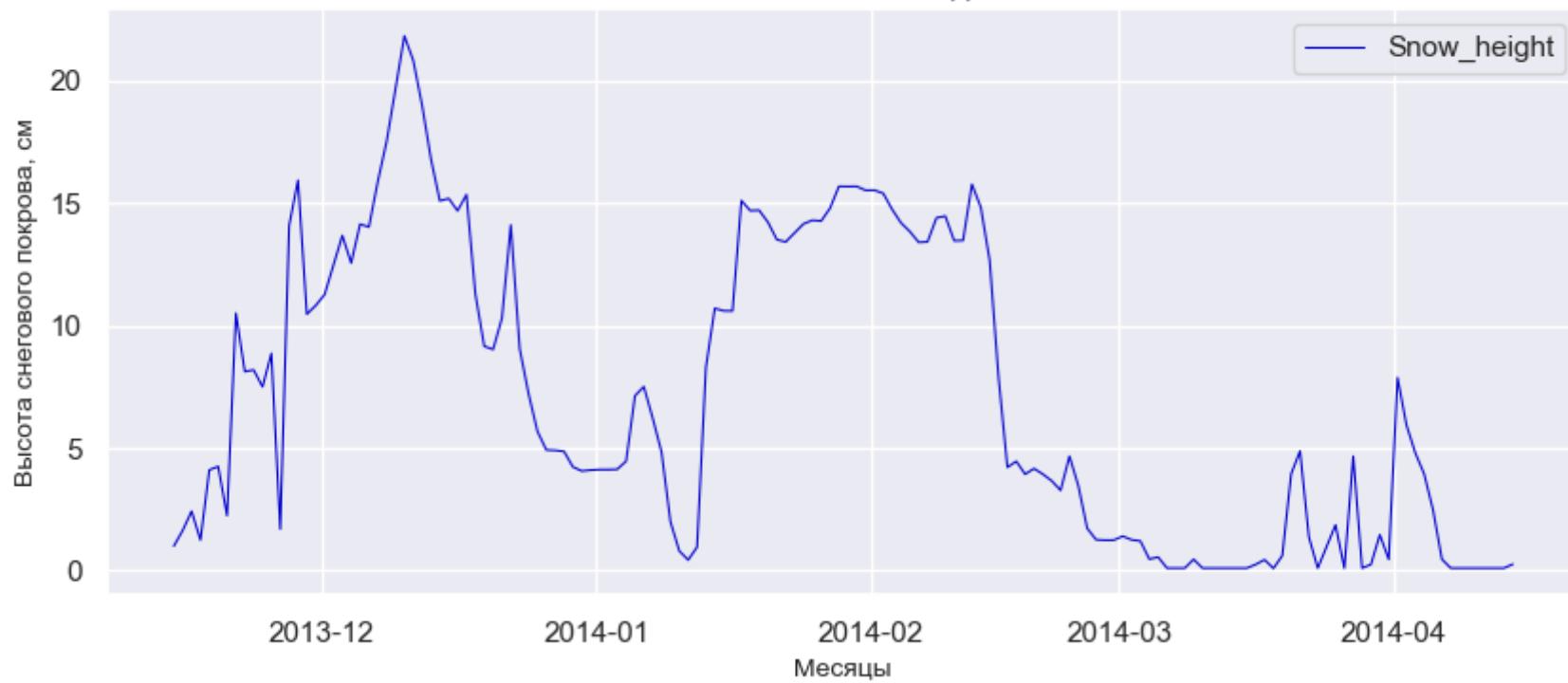
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2015-2016 годов



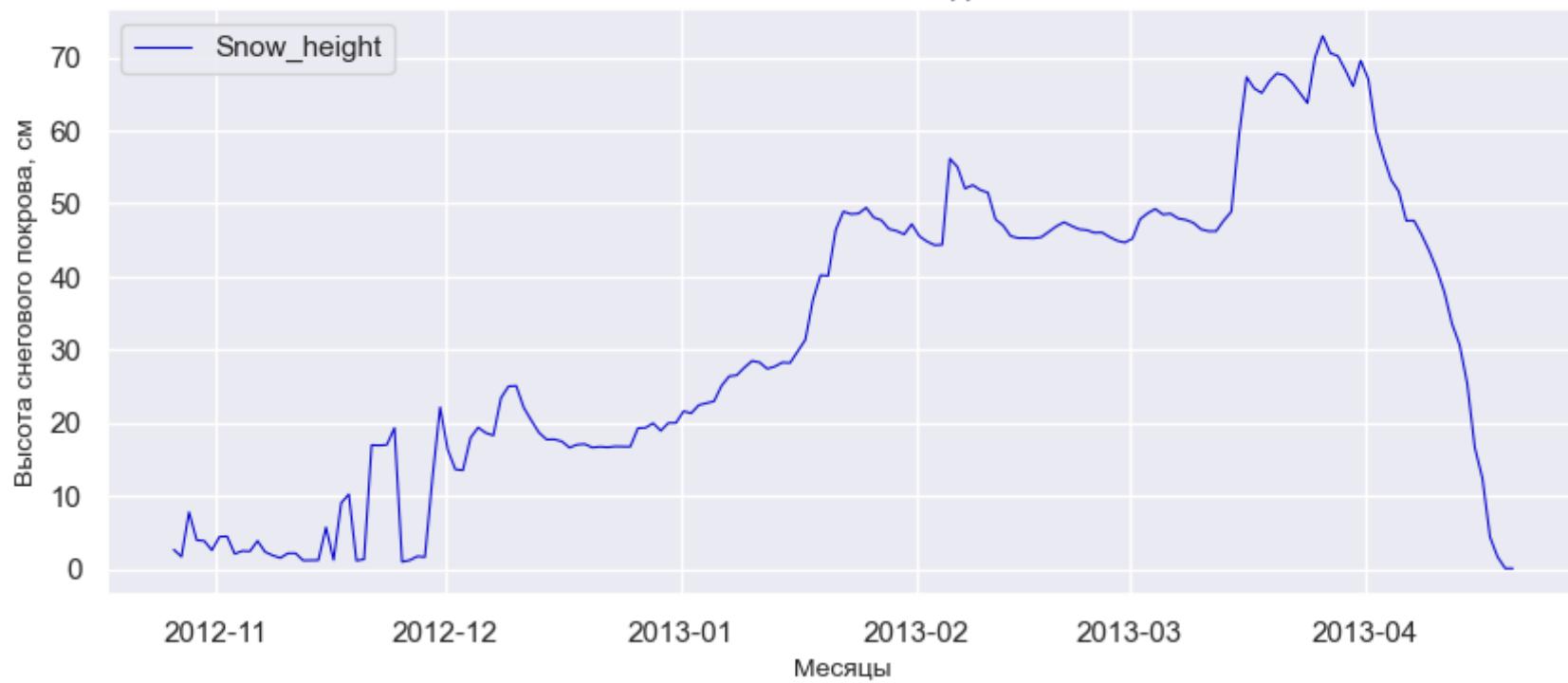
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2014-2015 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2013-2014 годов



Чашниково: Ежедневная динамика параметра
высоты снежного покрова Snow_height на 9 часов утра,
зима 2012-2013 годов



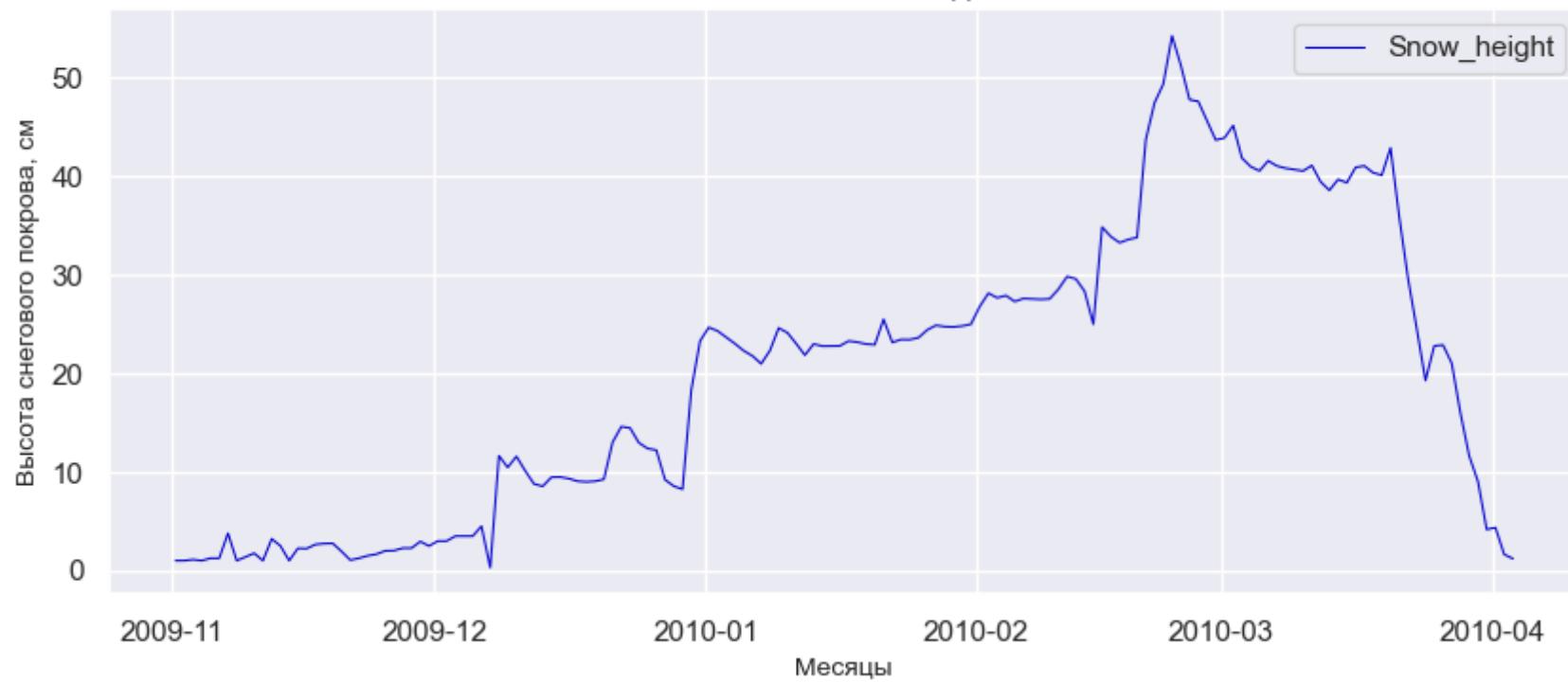
Чашниково: Ежедневная динамика параметра
высоты снежного покрова Snow_height на 9 часов утра,
зима 2011-2012 годов



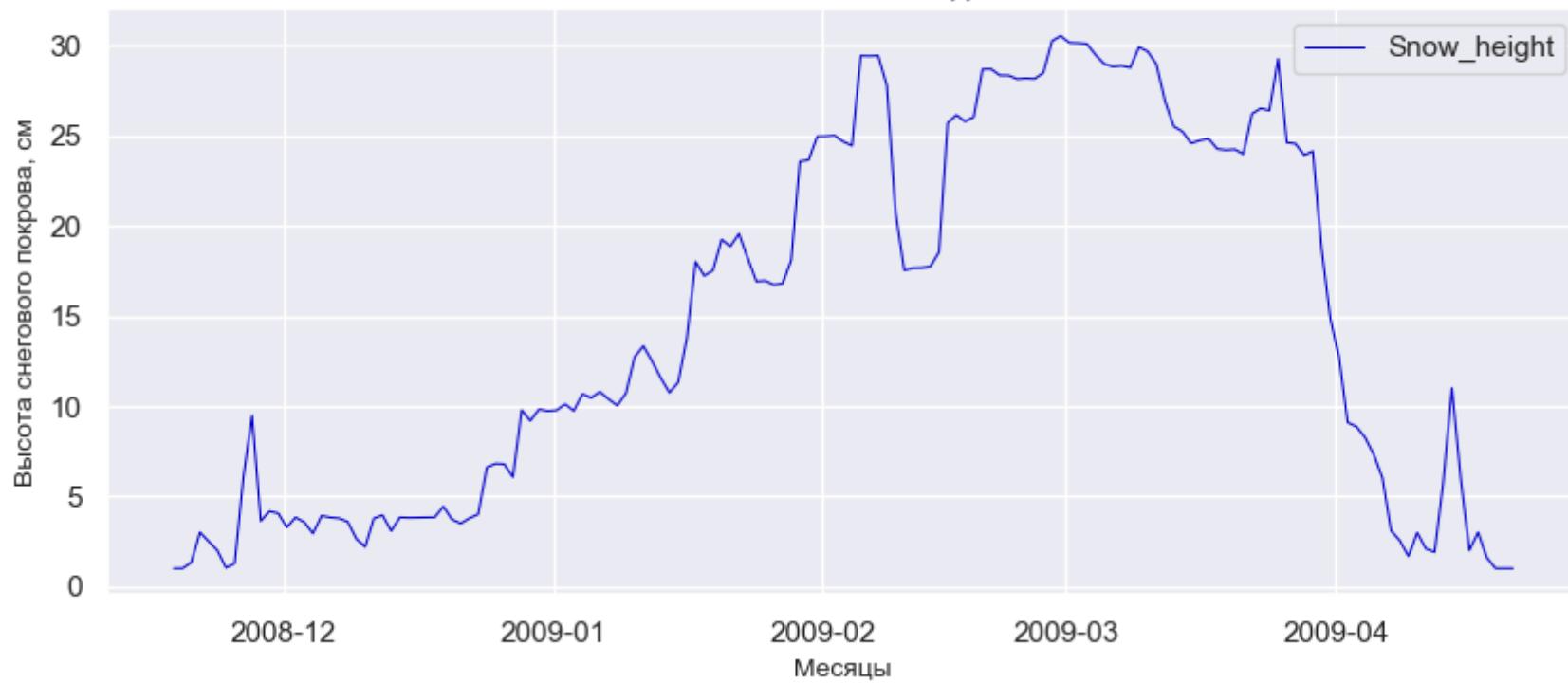
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2010-2011 годов



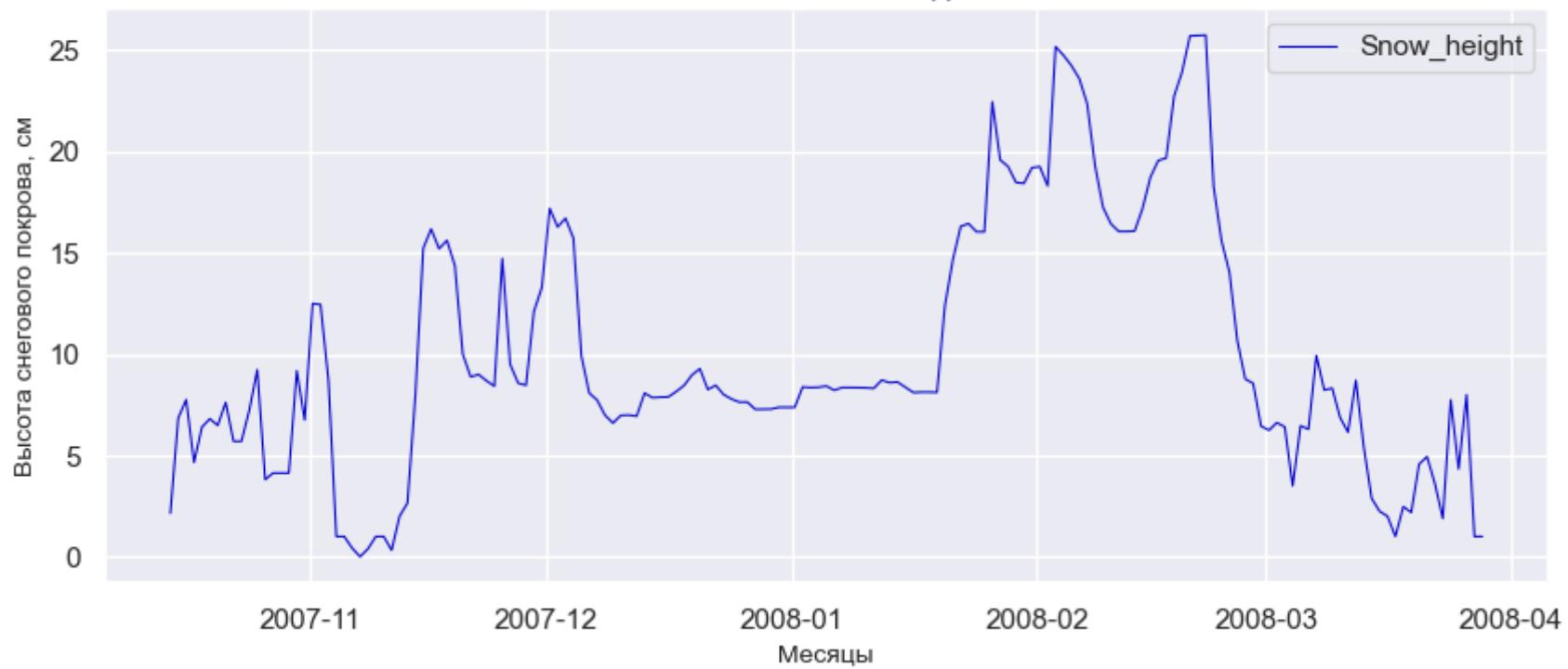
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2009-2010 годов



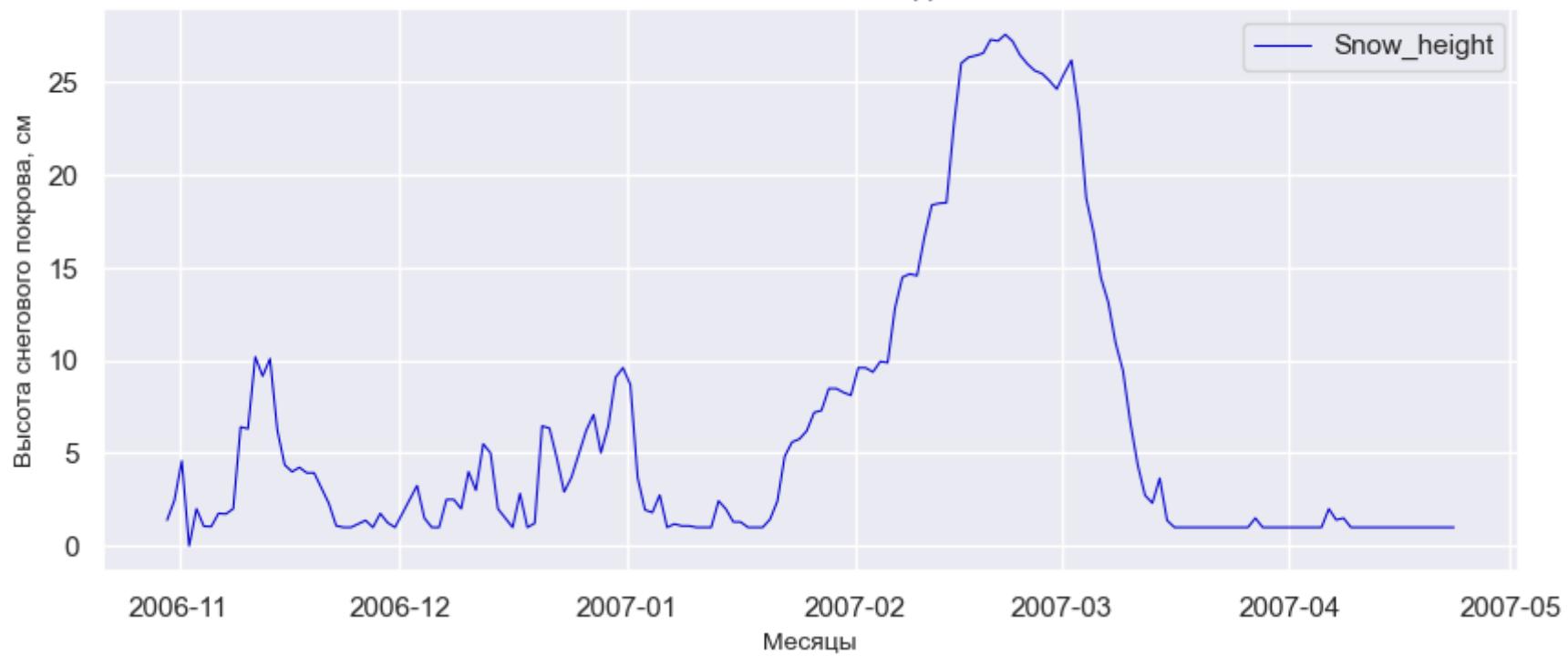
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2008-2009 годов



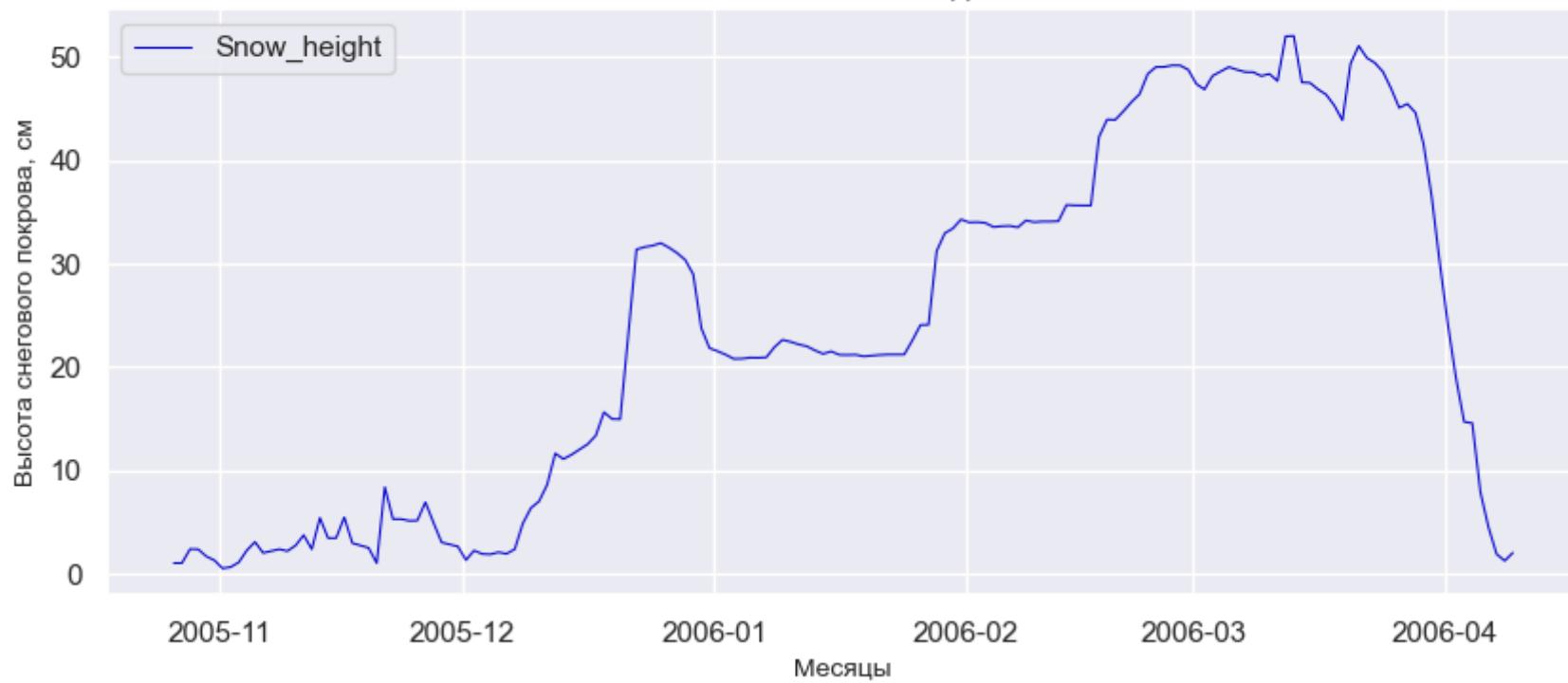
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2007-2008 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2006-2007 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height на 9 часов утра,
зима 2005-2006 годов



Чашниково: Ежедневная динамика параметра
высоты снежного покрова Snow_height на 9 часов утра,
зима 2004-2005 годов



```
In [800...]:  
data1 = dict_df_parameters['df_'+PARAMETER62]  
data2 = dict_df_parameters['df_'+PARAMETER32]  
# Строим график в цикле для каждого года (для момента наблюдения в 9 часов)  
for year in data1.index.year.unique(): # по перечню уникальных лет в data1, по годам  
    # определим логические маски: соответствие данному и предыдущему году, и не менее 1 неNaN значения в ряду  
    yearly_mask = (  
        (data1.index.year == year) &  
        (data1.index.hour == 9) &  
        (data1.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))  
    )  
    prev_yearly_mask = (  
        (data1.index.year == (year - 1)) &  
        (data1.index.hour == 9) &  
        (data1.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))  
    ) if year > data1.index.year.min() else (  
        (data1.index.year == year) &
```

```

        (data1.index.hour == 9) &
        (data1.apply(lambda x: pd.notna(x).sum() > 0, axis = 1))
    ) # учитываем неполный сезон начального года

# Определим границы периодов наблюдения (начало во второй половине года, окончание - в первой)
end_moment = (
    data1[yearly_mask & (data1.index.month<=6)]
    .dropna(how='all')
    .index
    .max()
) # максимальный момент в первой половине года

start_moment = (
    data1[prev_yearly_mask & (data1.index.month>=7)]
    .dropna(how='all')
    .index
    .min()
) if year != data1.index.min() else (
    data1[prev_yearly_mask & (data1.index.month<=6)]
    .dropna(how='all')
    .index
    .min()
) # минимальный момент во второй половине года, а для начала архива - начало архива снегового покрова

# создаём логическую маску для текущего сезона
snow_season_mask = ((data1.index >= start_moment) &
                     (data1.index <= end_moment) &
                     (data1.index.hour == 9))

data_y1 = data1[snow_season_mask][PARAMETER62+"#"+"Chashnikovo"]
data_y2 = data2[snow_season_mask][PARAMETER32+"#"+"Chashnikovo"]

fig, ax = plt.subplots(figsize=(10, 4))
twin1 = ax.twinx() # определим второй график на той же оси X
g1 = sns.lineplot(data=data_y1,
                   color='mediumblue',
                   linewidth=0.75,
                   ax=ax)
g2 = sns.lineplot(data=data_y2,
                   color='indianred',
                   linewidth=0.75,
                   ax=twin1)

```

```
# Добавляем легенду
blue_line = mlines.Line2D([], [], color='mediumblue', label=f'{PARAMETER62}', linewidth=0.75)
coral_line = mlines.Line2D([], [], color='indianred', label=f'{PARAMETER32}', linewidth=0.75)
dummy = ax.legend(handles=[blue_line, coral_line])

dummy = ax.set_ylabel('Высота снегового покрова, см', size=10)
dummy = ax.tick_params(axis='y', colors='mediumblue', size=4, width=1.5)
dummy = twin1.set_ylabel('Минимальная температура воздуха, °C', size=10)
dummy = twin1.tick_params(axis='y', colors='indianred', size=4, width=1.5)
dummy = ax.set_xlabel('Месяцы', size=10)

dummy = plt.title(f'Чашниково: Ежедневная динамика параметра \n'
                  f'высоты снегового покрова {PARAMETER62} и минимальной температуры воздуха {PARAMETER32}\n'
                  f'на 9 часов утра, зима {year-1}-{year} годов')
plt.show()
```

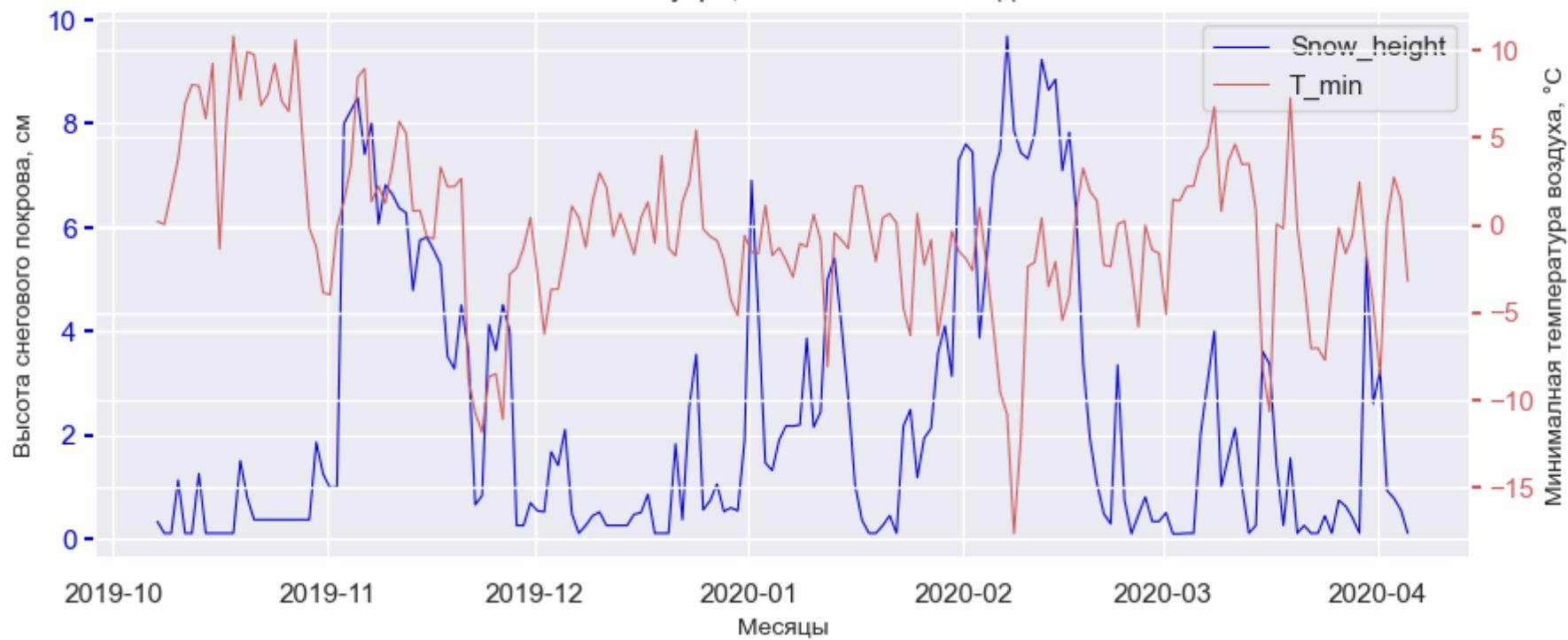
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2021-2022 годов



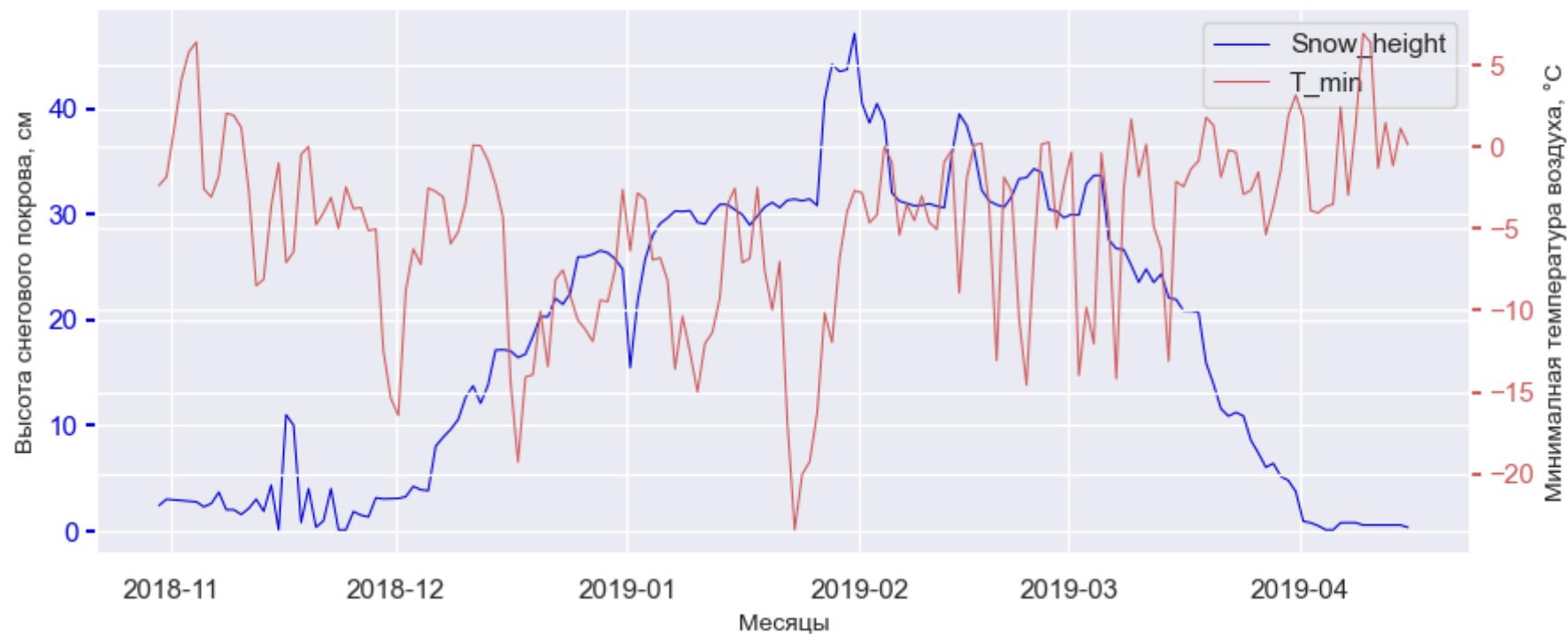
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2020-2021 годов



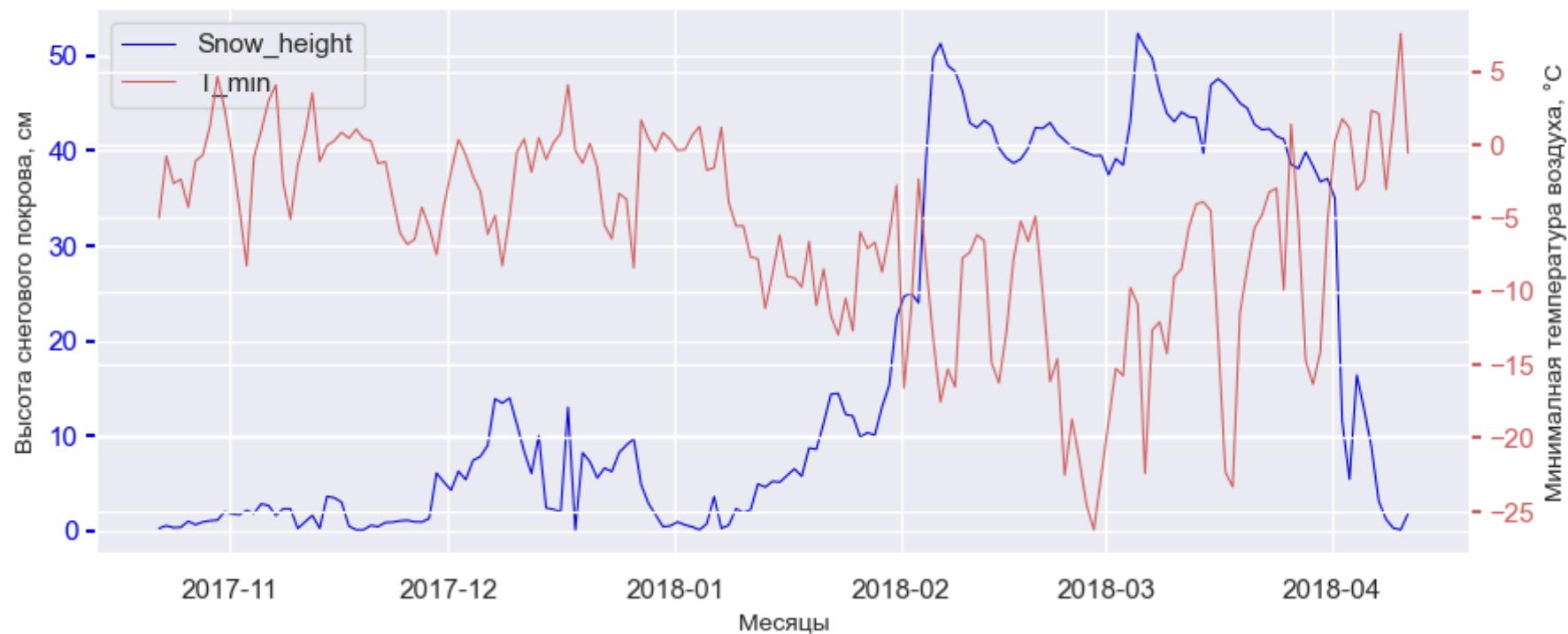
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2019-2020 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2018-2019 годов



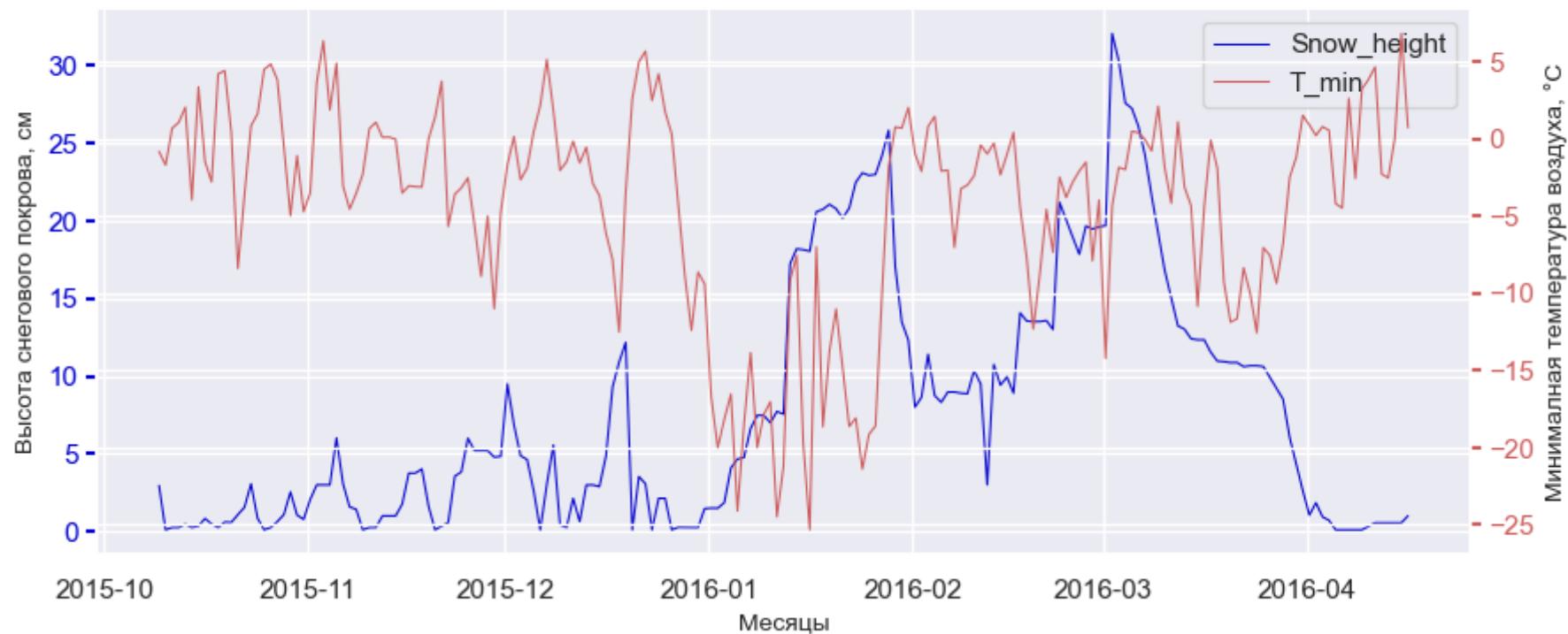
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2017-2018 годов



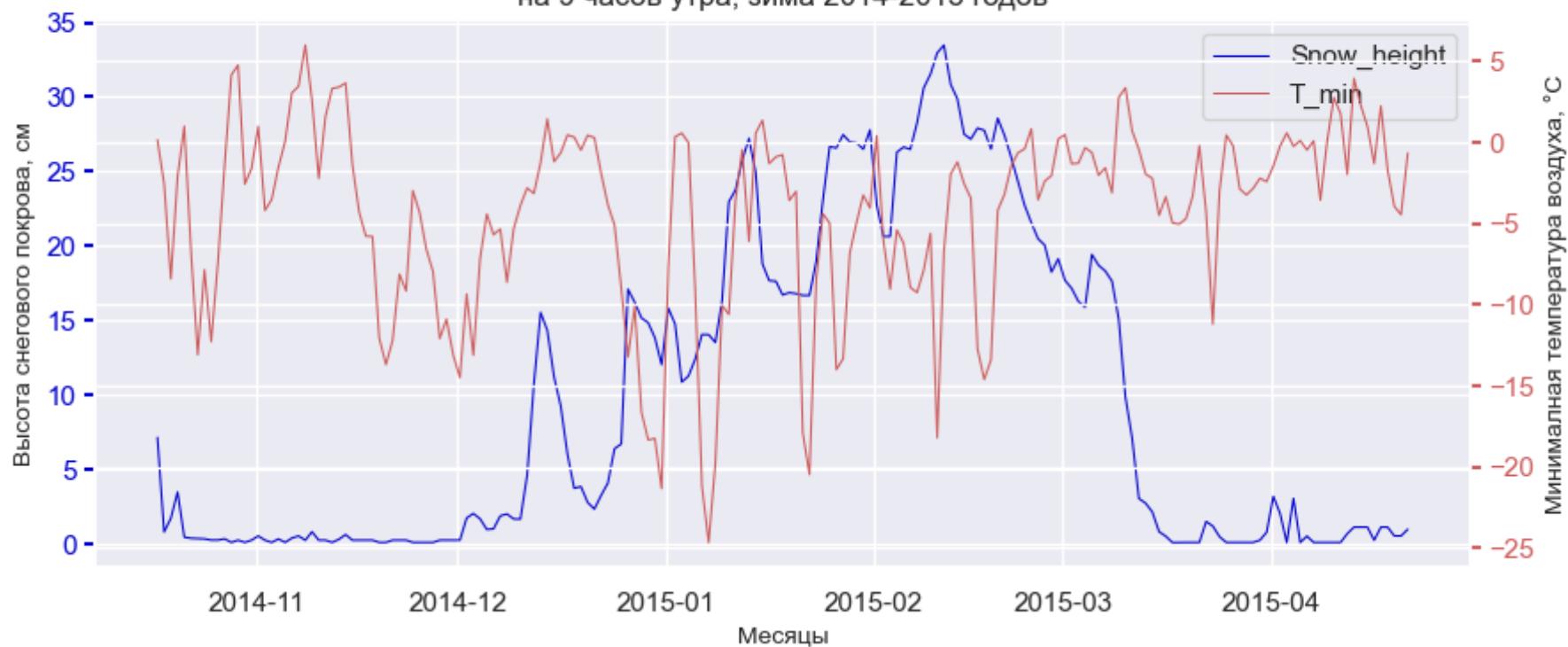
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2016-2017 годов



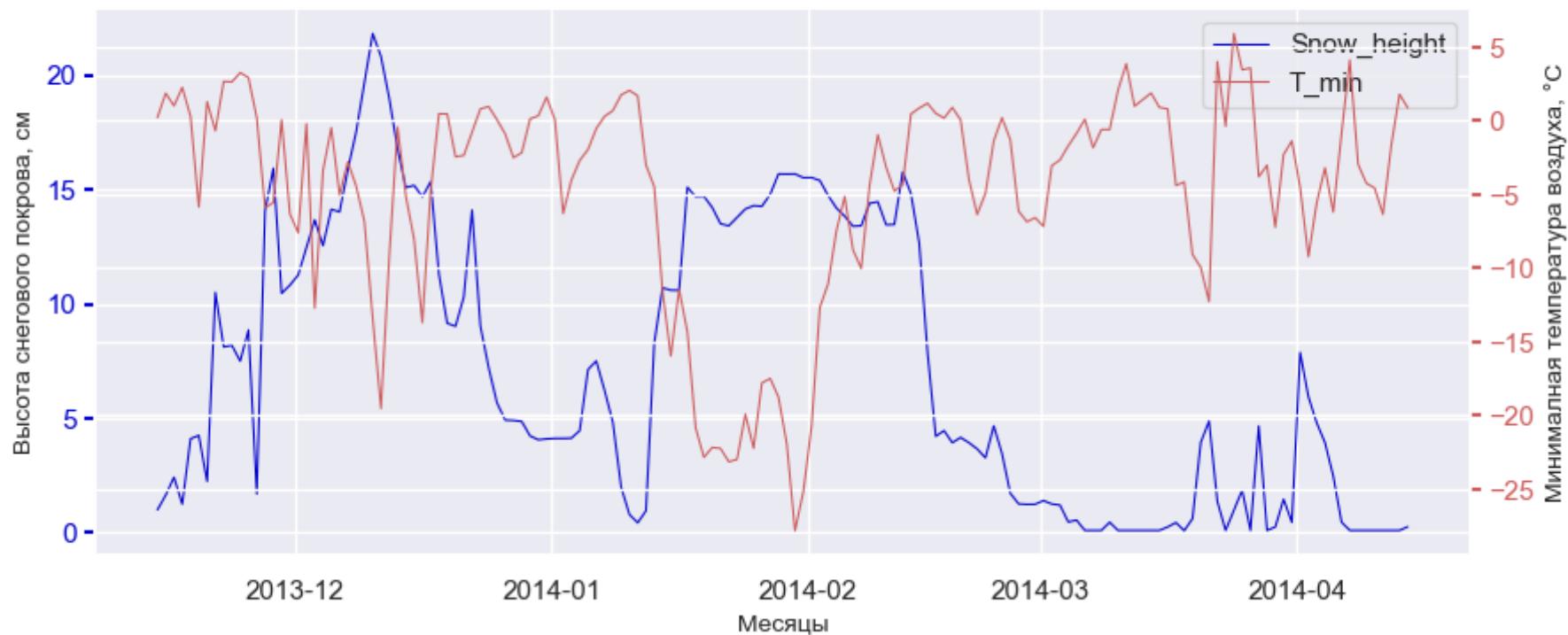
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2015-2016 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2014-2015 годов



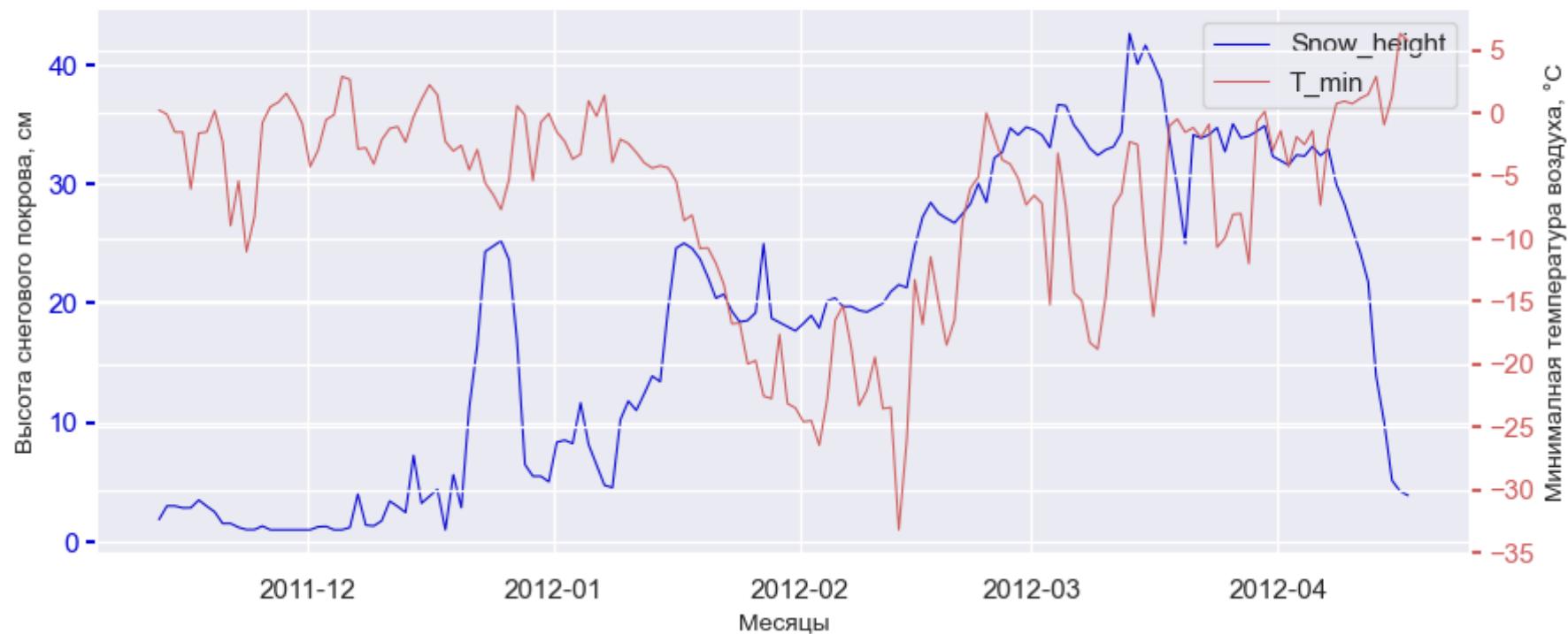
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2013-2014 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2012-2013 годов



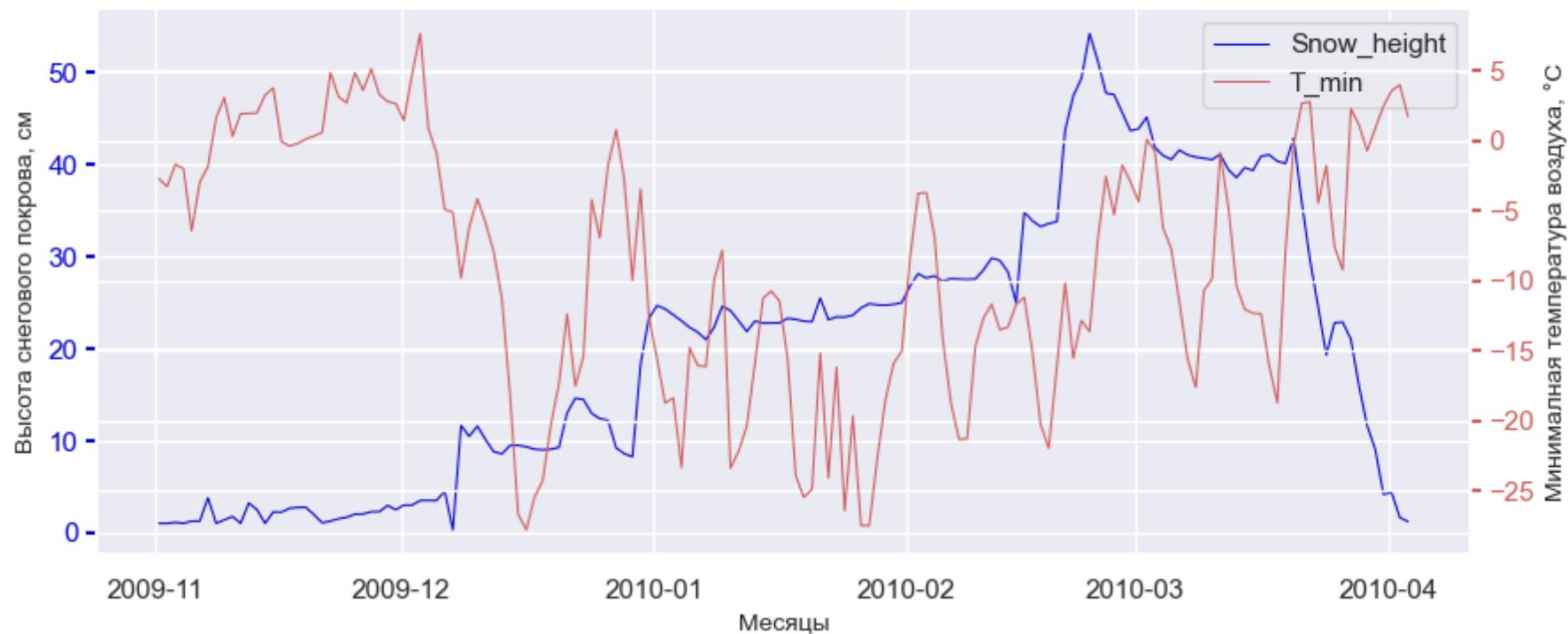
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2011-2012 годов



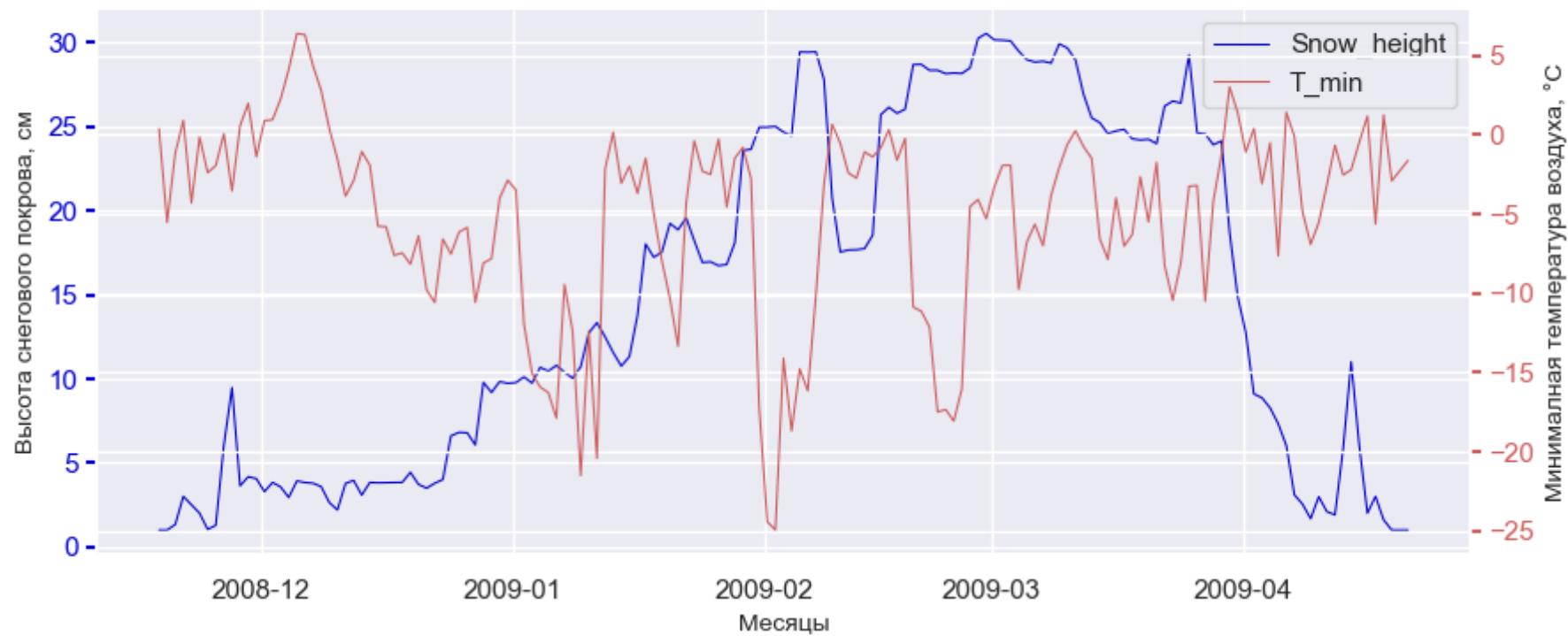
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2010-2011 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2009-2010 годов



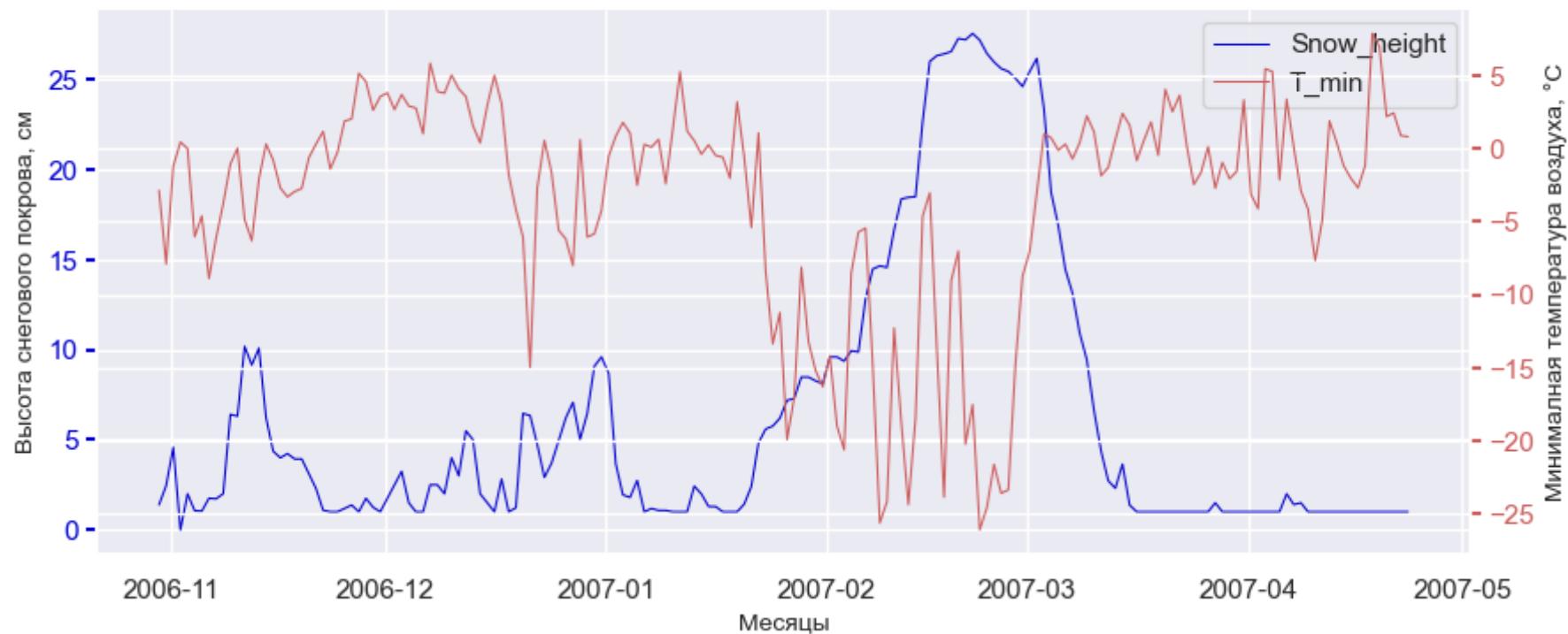
Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2008-2009 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2007-2008 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2006-2007 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2005-2006 годов



Чашниково: Ежедневная динамика параметра
высоты снегового покрова Snow_height и минимальной температуры воздуха T_min
на 9 часов утра, зима 2004-2005 годов



6.2.6. Сохранение полученных данных в файлы

```
In [801...]: # # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER62}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER62}'/  
  
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_'+PARAMETER62 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER62].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER62}.csv'
)
print('DONE!')

```

df_Chashnikovo.csv -> 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_Rfrnce_point.csv -> DONE!
df_Serpukhov.csv -> DONE!
df_Staritsa.csv -> DONE!
df_Tver.csv -> DONE!
df_Volokolamsk.csv -> DONE!
df_V_Volochev.csv -> DONE!
df_Snow_height.csv -> DONE!

ПРОДОЛЖЕНИЕ В ТЕТЕРАДИ 5

Разное

Вернём значения отображения к настройкам по умолчанию

In [802...]

```

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 [803...]

```

InteractiveShell.ast_node_interactivity = "last_expr" # Отображение результата только последней строки кода

```

