

# 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 3: Исследование аномалий, пропусков и ошибок, а также восстановление, исправление и моделирование значений для каждого индивидуального параметра: численные показатели атмосферного давления ( $P_{station}$ ,  $P_{sea}$ ,  $P_{drift}$ ), показатели относительной влажности воздуха и температуры точки росы ===*

## 0. Подготовка данных 3-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
#...
# - 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

# Настстройки
import warnings
```

Настроим отображение вывода результатов кода в нескольких ячейках

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

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

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

Установим для Seaborn настройки темы по умолчанию.

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

Определим константы и значения (сообразно предыдущим тетрадям)

```
In [5]: # Константы параметров  
PARAMETER31 = 'T'  
PARAMETER32 = 'T_min'  
PARAMETER33 = 'T_max'  
list_const_param = [PARAMETER31, PARAMETER32, PARAMETER33]  
list_const_param
```

```
Out[5]: ['T', 'T_min', 'T_max']
```

## 0.1. Импорт данных

```
In [6]: # Определим значения переменных path (по результатам обработки данных во 2-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/{PARAMETER33}/locations/' # путь к архивам метеостанций, в последней редакции (T_max)  
path2 = f'{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 [7]:

```
# 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[7]:

	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[7]:

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

Out[7]:

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

Out[7]:

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

## 0.1.2. Данные архивов погоды

### 0.1.2.1. Чтение архивов метеостанций

In [8]:

```
# 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[8]:

```
['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_Volochev.csv']
```

In [9]:

```
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/T_max/locations/df_Chashnikovo.csv - O.K.  
data/csv/predict/T_max/locations/df_Dmitrov.csv - O.K.  
data/csv/predict/T_max/locations/df_Kashyn.csv - O.K.  
data/csv/predict/T_max/locations/df_Klin.csv - O.K.  
data/csv/predict/T_max/locations/df_Mozhaisk.csv - O.K.  
data/csv/predict/T_max/locations/df_Naro_Fominsk.csv - O.K.  
data/csv/predict/T_max/locations/df_Nemchinovka.csv - O.K.  
data/csv/predict/T_max/locations/df_N_Jerusalem.csv - O.K.  
data/csv/predict/T_max/locations/df_Rfrnce_point.csv - O.K.  
data/csv/predict/T_max/locations/df_Serpukhov.csv - O.K.  
data/csv/predict/T_max/locations/df_Staritsa.csv - O.K.  
data/csv/predict/T_max/locations/df_Tver.csv - O.K.  
data/csv/predict/T_max/locations/df_Volokolamsk.csv - O.K.  
data/csv/predict/T_max/locations/df_V_Volochev.csv - O.K.  
Out[9]: 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 [10]: for name in dict_df_locations.keys():  
    dict_df_locations[name].sample(3)
```

```
Out[10]:
```

	T	T_min	T_max
<b>2012-03-19 21:00:00</b>	2.472873	2.472873	4.276204
<b>2020-06-18 09:00:00</b>	25.366130	16.469550	25.366130
<b>2012-05-14 06:00:00</b>	5.927060	5.927060	10.211642

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
Dmitrov_Local_time													
<b>2007-11-20 06:00:00</b>	0.4	740.8	757.6	NaN	96.0	WNW	292.5	4875.0	4.0	NaN	NaN	100%.	Дождь со снегом или ледяная крупа неливневые.
<b>2019-06-14 00:00:00</b>	13.6	752.9	769.1	NaN	49.0	SE	135.0	2250.0	3.0	NaN	NaN	от 90 менее 100%	NaN
<b>2014-08-19 21:00:00</b>	17.8	745.0	760.8	0.3	59.0	SSE	157.5	2625.0	1.0	NaN	NaN	20–30%.	NaN

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
Kashyn_Local_time													
<b>2006-02-04 09:00:00</b>	-26.6	748.0	762.4	NaN	80.0	штиль	360.0	6000.0	0.0	NaN	NaN	от 90% менее 100%	NaN
<b>2022-05-14 00:00:00</b>	9.6	737.5	750.0	0.1	72.0	W	270.0	4500.0	5.0	NaN	15.0	0%	NaN
<b>2017-04-08 12:00:00</b>	4.8	742.1	754.9	0.9	85.0	NNW	337.5	5625.0	1.0	NaN	NaN	от 90% менее 100%	Состояние неба в общем не изменилось.

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
Klin_Local_time													
<b>2017-03-11 03:00:00</b>	3.0	748.8	764.3	-0.1	71.0	S	180.0	3000.0	3.0	NaN	NaN	100%.	NaN
<b>2015-08-17 06:00:00</b>	10.6	748.9	763.9	0.4	92.0	N	0.0	0.0	1.0	NaN	NaN	100%.	Состояние неба в общем не изменилось. I
<b>2021-11-09 00:00:00</b>	3.2	736.3	751.5	1.5	91.0	WSW	247.5	4125.0	3.0	NaN	10.0	100%.	Ливневый(ые) дождь(и) слабый(ые) в срок наблюд...

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
<b>Mozhaisk_Local_time</b>													
<b>2013-04-18 09:00:00</b>	12.4	749.6	766.4	0.0	43.0	W	270.0	4500.0	2.0	NaN	NaN	50%.	NaN
<b>2014-05-01 00:00:00</b>	6.9	736.2	752.9	0.3	75.0	штиль	360.0	6000.0	0.0	NaN	NaN	0%	NaN
<b>2016-10-30 15:00:00</b>	-0.3	NaN	754.6	NaN	90.0	NE	45.0	750.0	2.0	NaN	NaN	NaN	умеренный или сильный в срок наб...

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wth
<b>Naro_Fominsk_Local_time</b>													
<b>2016-12-04 12:00:00</b>	-5.6	742.4	760.8	0.7	89.0	штиль	360.0	6000.0	0.0	NaN	NaN	от 90% менее 100%	ливнёвый снег слаг наблюд ил
<b>2022-06-02 03:00:00</b>	13.9	747.9	765.0	-0.1	100.0	ESE	112.5	1875.0	1.0	NaN	NaN	100%.	
<b>2005-07-21 18:00:00</b>	23.9	734.5	750.8	NaN	74.0	SSW	202.5	3375.0	2.0	NaN	NaN	от 90% менее 100%	ливнёвый дождь

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	W
<b>Nemchinovka_Local_time</b>													
<b>2022-04-23 18:00:00</b>	14.300000	745.0	760.9	NaN	42.0	S	180.0	3000.0	1.0	NaN	NaN	70 – 80%.	
<b>2017-01-09 09:00:00</b>	-21.800000	753.1	771.6	-0.3	81.0	NW	315.0	5250.0	1.0	NaN	NaN	100%.	
<b>2008-06-17 00:00:00</b>	16.009681	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_
<b>N_Jerusalem_Local_time</b>													
<b>2012-07-01 06:00:00</b>	9.5	751.3	766.0	NaN	93.0	штиль	360.0	6000.0	0.0	NaN	NaN	20–30%.	
<b>2015-08-16 21:00:00</b>	12.6	747.6	761.9	0.6	87.0	WNW	292.5	4875.0	2.0	NaN	NaN	70 – 80%.	
												Обла це рассекива станов M	
<b>2006-07-17 09:00:00</b>	15.3	747.3	761.6	NaN	74.0	N	0.0	0.0	3.0	NaN	NaN	70 – 80%.	

Out[10]:

	T	T_min	T_max
2012-05-26 06:00:00	5.485245	3.855456	13.303483
2008-06-06 15:00:00	21.126501	8.073209	21.126501
2017-10-12 03:00:00	6.007353	6.007353	7.583748

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
<b>Serpukhov_Local_time</b>													
2008-07-05 12:00:00	17.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-03-25 12:00:00	8.2	746.5	761.5	0.5	46.0	W	270.0	4500.0	2.0	NaN	NaN	от 90 менее 100%	NaN
2014-07-31 21:00:00	22.7	746.9	761.1	0.3	49.0	E	90.0	1500.0	1.0	NaN	NaN	0%	NaN

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_cui
<b>Staritsa_Local_time</b>													
<b>2016-01-28 15:00:00</b>	2.9	731.9	748.9	-1.5	95.0	SSW	202.5	3375.0	5.0	NaN	NaN	100%.	Морос незамерзающая с перерывами слабая в сро
<b>2006-08-14 00:00:00</b>	16.8	747.4	763.7	NaN	87.0	E	90.0	1500.0	1.0	NaN	NaN	70 – 80%.	Nal
<b>2014-06-20 12:00:00</b>	20.9	735.0	750.9	-0.4	53.0	S	180.0	3000.0	6.0	11.0	11.0	40%.	Nal

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_curr
<b>Tver_Local_time</b>													
<b>2011-03-26 03:00:00</b>	-7.8	732.2	746.0	NaN	93.0	SW	225.0	3750.0	1.0	NaN	NaN	100%.	Снег непрерывный умеренный в срок наблюдения. не уи н
<b>2014-06-22 06:00:00</b>	7.7	742.4	755.6	0.2	90.0	W	270.0	4500.0	1.0	NaN	NaN	20–30%.	NaN
<b>2013-08-25 21:00:00</b>	7.8	750.5	763.9	0.8	93.0	штиль	360.0	6000.0	0.0	NaN	NaN	0%	NaN

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_
Volokolamsk_Local_time													
2010-09-13 21:00:00	8.9	750.4	768.6	NaN	92.0	штиль	360.0	6000.0	0.0	NaN	NaN	0%	I
2018-03-15 00:00:00	-3.2	735.9	754.6	NaN	93.0	S	180.0	3000.0	3.0	NaN	NaN	NaN	I
2005-02-15 06:00:00	-2.6	742.3	761.0	NaN	93.0	SE	135.0	2250.0	4.0	NaN	15.0	100%.	Обла це рассевиа станов м

Out[10]:

	T	P_station	P_sea	P_drift	Humid	Wind_dir	Wind_dir360	Wind_dir6k	Wind_speed	Gusts	Gusts_3h	Cloudness	Wthr_
V_Volochev_Local_time													
2013-11-24 15:00:00	4.800000	740.5	756.0	-0.6	98.0	SSW	202.5	3375.0	3.0	NaN	NaN	100%.	
2006-01-10 15:00:00	-6.335351	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2010-09-18 06:00:00	8.500000	741.9	757.2	NaN	93.0	SSE	157.5	2625.0	1.0	NaN	NaN	NaN	от 90 менее 100%

## 0.1.2.2. Чтение архивов параметров

In [11]:

```
# 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[11]: ['df_Cloudness.csv',
 'df_C1_bottom.csv',
 'df_C1_cirrus.csv',
 'df_C1_Cumls.csv',
 'df_C1_cumls_hi.csv',
 'df_C1_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 [12]: # 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/raw/df_Dew_point.csv - O.K.
data/csv/raw/df_Gusts.csv - O.K.
data/csv/raw/df_Gusts_3h.csv - O.K.
data/csv/raw/df_Humid.csv - O.K.
data/csv/raw/df_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_tdelta.csv - O.K.
data/csv/raw/df_P_drift.csv - O.K.
data/csv/raw/df_P_sea.csv - O.K.
data/csv/raw/df_P_station.csv - O.K.
data/csv/raw/df_Snow_height.csv - O.K.
data/csv/raw/df_Soil.csv - O.K.
data/csv/raw/df_Soil_cover.csv - O.K.
data/csv/raw/df_Soil_T.csv - O.K.
data/csv/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[12]: 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 [13]: for name in dict_df_parameters.keys():
    dict_df_parameters[name].sample(3)
```

Out[13]:	Cloudness#V_Volochev	Cloudness#Staritsa	Cloudness#Kashyn	Cloudness#Tver	Cloudness#Klin	Cloudness#Dmitrov	Cloudness#Volokolamsk	Cloudness#K
2005-03-06 15:00:00	NaN	100%.	NaN	100%.	100%.	100%.	100%.	100%.
2016-05-23 03:00:00	70 – 80%.	100%.	100%.	от 90% менее 100%	100%.	NaN	NaN	NaN
2021-03-18 21:00:00	60%.	100%.	от 90% менее 100%	от 90% менее 100%	от 90% менее 100%	100%.	100%.	100%.

Out[13]:	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
2007-04-19 03:00:00	600-1000	300-600	300-600	2500 или более, или облаков нет.	300-600	300-600	300-600	300-600
2011-07-20 12:00:00	600-1000	NaN	600-1000	600-1000	600-1000	600-1000	600-1000	1000-1500
2011-10-21 21:00:00	600-1000	600-1000	600-1000	600-1000	600-1000	300-600	300-600	300-600

Out[13]:

	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
2022-04-23 12:00:00	Перистых, перисто-кучевых или перисто-слоистых...	Перистые нитевидные, иногда когтевидные, не ра...	Перистых, перисто-кучевых или перисто-слоистых...	Перистых, перисто-кучевых или перисто-слоистых...	Перистые нитевидные, иногда когтевидные, не ра...	Перистые нитевидные, иногда когтевидные, не ра...	Перистые плотные в виде кла... скрученных...	Перистые плотные в виде кла... скрученных...
2021-07-05 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Перистые перисто-кучевые или перисто-слоистые...
2008-06-25 15:00:00	NaN	NaN	NaN	NaN	Перистые нитевидные, иногда когтевидные, не ра...	NaN	Перистые плотные в виде кла... скрученных...	

Out[13]:

	Cl_Cumls#V_Volochev	Cl_Cumls#Staritsa	Cl_Cumls#Kashyn	Cl_Cumls#Tver	Cl_Cumls#Klin	Cl_Cumls#Dmitrov	Cl_Cumls#Volokolamsk	Cl_Cumls#Moz
2008-05-01 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Слоисто-кучевые, образовавшиеся не из кучевых.
2021-08-08 12:00:00	Кучевые средние или мощные или вместе с кучевы...	Кучевые средние или мощные или вместе с кучевы...	Кучево-дождевые лысые с кучевыми, слоисто-куче...	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Кучево-дождевые лысые с кучевыми, слоисто-куче...	Кучевые плоские или кучевые разорванные, или т...	Слоисто-образованные...
2011-10-09 06:00:00	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, слоистых, кучевых или кучево-...	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-кучевые, образовавшиеся не из кучевых.	Слоисто-образованные...

Out[13]:	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#Volok
2008-08-22 03:00:00	Высококучевые просвечивающие, расположенные на...	NaN	NaN	Высокослоистые непросвечивающие или слоисто-до...	NaN	NaN	Высокок просвечив. расположенн
2020-10-08 18:00:00	Высококучевых, высокослоистых или слоисто-дожд...	Высококучевые, просвечивающие или плотные в дв...	Высококучевые просвечивающие, расположенные на...	Высококучевые просвечивающие, расположенные на...	Высококучевых, высокослоистых или слоисто-дожд...	Высококучевые просвечивающие, расположенные на...	

Out[13]:	Cl_viewd#V_Volochev	Cl_viewd#Staritsa	Cl_viewd#Kashyn	Cl_viewd#Tver	Cl_viewd#Klin	Cl_viewd#Dmitrov	Cl_viewd#Volokolamsk	Cl_viewd#M
<b>2014-01-11 03:00:00</b>	100%.	90 или более, но не 100%	90 или более, но не 100%	90 или более, но не 100%	100%.	100%.	100%.	100%.
<b>2006-06-06 12:00:00</b>	NaN	90 или более, но не 100%	NaN	90 или более, но не 100%	100%.	100%.	100%.	100%.
<b>2021-09-10 18:00:00</b>	40%.	100%.	40%.	90 или более, но не 100%	90 или более, но не 100%	40%.	90 или более, но не 100%	90 или более, но не 100%

Out[13]:	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
2015-10-04 18:00:00	0.0	0.0	0.0	0.0	0.0	0.0
2006-06-30 15:00:00	0.0	0.0	0.0	0.0	0.0	0.0
2015-04-13 03:00:00	0.0	0.0	0.0	0.0	0.0	0.0

Out[13]:	Dew_point#V_Volochev	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
<b>2009-11-02 12:00:00</b>	NaN	-4.1	NaN	-2.9	-1.6	-4.1	-4.5
<b>2016-10-19 15:00:00</b>	-2.7	-4.2	-2.4	-3.3	-2.7	-1.4	-4.6
<b>2020-05-27 03:00:00</b>	5.2	7.5	6.7	4.7	8.3	7.5	NaN



Out[13]:	Prcptn_like#V_Volochev	Prcptn_like#Staritsa	Prcptn_like#Kashyn	Prcptn_like#Tver	Prcptn_like#Klin	Prcptn_like#Dmitrov	Prcptn_like#Volokolamsk		
2011-03-12 09:00:00	NaN	Состояние неба в общем не изменилось.	Снег непрерывный слабый в срок наблюдения.	Снег неливневый.	Снег непрерывный слабый в срок наблюдения.	Снег непрерывный слабый в срок наблюдения.	Ливневый снег с срок наблюдений		
2015-09-23 09:00:00	Дымка.	Туман или ледяной туман, небо видно, ослабел з...	NaN	Туман или ледяной туман.	NaN	NaN	NaN		
2020-02-05 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN		
Out[13]:	Prcptn_tdelt#V_Volochev	Prcptn_tdelt#Staritsa	Prcptn_tdelt#Kashyn	Prcptn_tdelt#Tver	Prcptn_tdelt#Klin	Prcptn_tdelt#Dmitrov	Prcptn_tdelt#Volokolamsk		
2017-10-27 06:00:00	12.0	12.0	12.0	12.0	12.0	12.0	12.0		
2013-07-22 09:00:00	12.0	12.0	12.0	12.0	12.0	12.0	12.0		
2007-06-15 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN		
Out[13]:	P_drift#V_Volochev	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#Volokolamsk
2015-04-26 09:00:00	-0.7	-1.1	-0.2	-0.8	-0.8	-0.5	-0.9	-0.6	0.0
2017-06-07 00:00:00	0.5	0.7	0.2	0.5	0.4	NaN	NaN	NaN	NaN
2021-07-14 15:00:00	0.2	-1.0	-0.3	-0.7	-1.1	NaN	NaN	NaN	NaN





Out[13]:	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
2016-09-21 18:00:00	9.800000	11.3	9.9	10.7	9.1	9.3	10.1	9.5	9.7	10.000000	9
2010-02-25 21:00:00	-8.600000	-5.2	-13.9	-7.9	-6.3	-6.2	-3.3	-1.7	-2.9	-0.800000	-0
2008-05-30 12:00:00	17.003057	17.0	15.2	17.1	17.0	15.2	16.8	15.4	17.4	16.666774	16
Out[13]:	T_max#V_Volochek	T_max#Staritsa	T_max#Kashyn	T_max#Tver	T_max#Klin	T_max#Dmitrov	T_max#Volokolamsk	T_max#Mozhaisk	T_max#N_Jer		
2020-09-05 03:00:00	18.100000		19.7	18.400000		19.7	20.1	19.9	19.8		21.4
2006-12-27 18:00:00	1.546717		1.5	1.228236		1.7	1.1	1.1	1.4		1.4
2017-04-05 12:00:00	9.400000		9.4	7.100000		9.4	6.8	5.9	7.2		8.5
Out[13]:	T_min#V_Volochek	T_min#Staritsa	T_min#Kashyn	T_min#Tver	T_min#Klin	T_min#Dmitrov	T_min#Volokolamsk	T_min#Mozhaisk	T_min#N_Jer		
2010-12-09 00:00:00	-8.0		-7.0	-7.1		-7.4	-6.4	-6.3	-6.3		-4.9
2011-09-10 12:00:00	11.8		11.2	11.0		11.2	10.2	10.2	10.7		10.7
2017-01-22 00:00:00	-8.0		-7.8	-7.2		-7.2	-7.7	-8.0	-6.7		-7.7

Out[13]:	Visibility#V_Volochek	Visibility#Staritsa	Visibility#Kashyn	Visibility#Tver	Visibility#Klin	Visibility#Dmitrov	Visibility#Volokolamsk	Visibility#M
2005-05-10 09:00:00	NaN	20.0	NaN	10.0	20.0	12.0		40.0
2022-03-12 15:00:00	20.0	20.0	20.0	10.0	10.0	10.0		4.0
2011-04-30 18:00:00	23.0	20.0	50.0	10.0	10.0	12.0		29.0
Out[13]:	Wind_dir#V_Volochek	Wind_dir#Staritsa	Wind_dir#Kashyn	Wind_dir#Tver	Wind_dir#Klin	Wind_dir#Dmitrov	Wind_dir#Volokolamsk	Wind_dir#
2016-03-02 09:00:00	E	E	E	E	ESE		NaN	NaN
2007-07-14 18:00:00	NaN	W	NaN	W	SSW		SW	WNW
2006-10-25 21:00:00	SSW	SSW	WSW	SW	S		SW	SW
Out[13]:	Wind_dir360#V_Volochek	Wind_dir360#Staritsa	Wind_dir360#Kashyn	Wind_dir360#Tver	Wind_dir360#Klin	Wind_dir360#Dmitrov	Wind_dir360#	
2020-04-03 09:00:00	180.0	202.5	202.5	180.0	180.0		202.5	
2019-02-11 18:00:00	135.0	157.5	180.0	157.5	157.5		157.5	
2015-05-17 03:00:00	315.0	315.0	0.0	292.5	270.0		270.0	

Out[13]:	Wind_dir6k#V_Volochek	Wind_dir6k#Staritsa	Wind_dir6k#Kashyn	Wind_dir6k#Tver	Wind_dir6k#Klin	Wind_dir6k#Dmitrov	Wind_dir6k#Volokola
2014-10-19 00:00:00	4875.0	4875.0	5625.0	4500.0	4875.0	4875.0	5625.0
2011-05-21 18:00:00	4875.0	3750.0	5625.0	750.0	6000.0	4500.0	3000.0
2017-09-22 06:00:00	1500.0	1500.0	1500.0	1500.0	1125.0	1500.0	1500.0

Out[13]:	Wind_speed#V_Volochek	Wind_speed#Staritsa	Wind_speed#Kashyn	Wind_speed#Tver	Wind_speed#Klin	Wind_speed#Dmitrov	Wind_speed#Volokola
2014-01-15 12:00:00	2.0	0.0	2.0	1.0	1.0	1.0	1.0
2021-01-25 12:00:00	3.0	3.0	3.0	3.0	5.0	4.0	4.0
2013-12-28 03:00:00	3.0	5.0	4.0	2.0	2.0	3.0	3.0

Out[13]:	Wthr_3h#V_Volochev	Wthr_3h#Staritsa	Wthr_3h#Kashyn	Wthr_3h#Tver	Wthr_3h#Klin	Wthr_3h#Dmitrov	Wthr_3h#Volokolamsk	Wthr_3h#Moz
2005-11-22 21:00:00	NaN	NaN	Облака покрывали более половины неба в течение...	NaN	Облака покрывали более половины неба в течение...			
2014-08-06 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	Облака покрывали более половины неба в течение...	Гроза (грозы) с осадками и ливнями
2017-07-29 03:00:00	NaN	Облака покрывали более половины неба в течение...	NaN	NaN	NaN	NaN	NaN	NaN
Out[13]:	Wthr_3h2#V_Volochev	Wthr_3h2#Staritsa	Wthr_3h2#Kashyn	Wthr_3h2#Tver	Wthr_3h2#Klin	Wthr_3h2#Dmitrov	Wthr_3h2#Volokolamsk	Wthr_3h2#Moz
2015-05-31 18:00:00	NaN	NaN	NaN	NaN	Облака покрывали более половины неба в течение...	NaN	NaN	NaN
2011-02-21 18:00:00	Облака покрывали более половины неба в течение...	NaN	NaN	NaN	NaN	NaN	NaN	Облачно с дождем
2015-09-23 06:00:00	Облака покрывали половину неба или менее в теч...	Облака покрывали более половины неба в течение...	NaN	Облака покрывали половину неба или менее в теч...	NaN	NaN	NaN	NaN

Out[13]:

		Wthr_curr#V_Volochek	Wthr_curr#Staritsa	Wthr_curr#Kashyn	Wthr_curr#Tver	Wthr_curr#Klin	Wthr_curr#Dmitrov	Wthr_curr#Volokolamsk	Wt1
2014-12-29 18:00:00		Диаметр изморозевого отложения составляет 3 мм.	Диаметр изморозевого отложения составляет 13...	NaN	Диаметр изморозевого отложения составляет 7 мм.	Состояние неба в общем не изменилось.	Снег непрерывный слабый в срок наблюдения.	Состояние неба в общем не изменилось.	
2006-01-07 09:00:00	Замерзающая морось или замерзающий дождь нелив...	Морось замерзающая слабая.	Дымка.	Снег неливневый.	Снег непрерывный слабый в срок наблюдения.	Состояние неба в общем не изменилось.	Снег непрерывный слабый в срок наблюдения.	Л	л
2005-08-03 18:00:00		NaN	Состояние неба в общем не изменилось.	NaN	NaN	NaN	NaN	NaN	NaN

# 1. Теоретические основы

## 1.1. Описание задачи

**Основная цель: получить статистически значимые изолированные наблюдения без ошибок и пропусков во всём поле метеостанций, по каждому параметру, на каждый заданный момент наблюдения, и на их основе смоделировать значения тех же параметров для Агробиостанции МГУ. Для реализации поставленной цели необходимо будет подобрать оптимальный способ восстановления данных для каждого параметра, который может быть положен в основу составления архивов метеонаблюдений для локации Агробиостанции МГУ в пос. Чашниково Солнечногорского района Московской области. ВАЖНО: время наблюдений, фиксируемое в архивах погодных явлений, определяется по Гринвичу.**

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

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

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

Поэтому ниже мы будем принимать во внимание, среди прочего:

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

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

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

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

- наблюдения на всех метеорологических станциях проводят синхронно восемь раз в сутки в 00, 03, 06, 09, 12, 15, 18 и 21 ч по гринвичскому времени;
- во все сроки измеряют температуру воздуха и почвы, влажность воздуха, скорость ветра и его направление, метеорологическую дальность видимости, атмосферное давление, определяют характеристики облачности;

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

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

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

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

1. Географическое пространство метеостанций (поле):
  - географическое положение (координаты и высота над уровнем моря),
  - производные от них расстояния и начальные азимуты.
2. Временной ряд:
  - однодиректорный, 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.;

- Prcptn (RR) Количество выпавших осадков, мм;
- Prcptn\_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\_Cumuls (Cl) Слоисто-кучевые, слоистые, кучевые и кучево-дождевые облака;
- Cl\_viewed (Nh) Количество всех наблюдающихся облаков Cl, или при отсутствии облаков Cl, количество всех наблюдающихся облаков Cm, %;
- Cl\_cumuls\_hi (Cm) Высоко-кучевые, высоко-слоистые и слоисто-дождевые облака;
- Cl\_cirrus (Ch) Перистые, перисто-кучевые и перисто-слоистые облака.

1. Вычисляемые и зависимые данные:

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

### 1.3.2. Последовательность моделирования метеорологических показателей.

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

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

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

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

Архивные данные содержат большое количество пропусков и ошибок. Как было показано в первой тетради, пропуски по некоторым показателями наблюдения зачастую занимают большие периоды по временной оси. Некоторые метеостанции не фиксировали отдельные показатели годами. В связи с этим попытка моделирования данных, опираясь на значения по временной оси не представляется перспективной. В ряде случаев (где проущено очень много моментов наблюдения подряд) они просто не дадут результата. Надо также учитывать, что сезонные и суточные колебания, хоть и подобны, но практически никогда не дублируются. Поэтому предсказания на их основе будут иметь достаточно низкую точность. Гораздо более точные предсказания можно получить из наблюдений в различных, относительно близко расположенных, точках, объединённых одним моментом времени.

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

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

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

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

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

#### *Ограничения.*

1. Для создания моделей нам нельзя брать разные метеостанции для обучения, валидации и предсказания. Нам нужно брать разные данные по моментам наблюдения.
2. Важно, чтобы между или в непосредственной близости рядом с метеостанциями и искомой точкой не было зон с другим микроклиматом (больших водоёмов, высоких неровностей рельефа, зон плотной городской застройки и климатической "грелки" вроде Москвы). Именно по этой причине мы не можем взять для расчётов показания метеостанций, расположенных в черте Москвы.

***Таким образом, для каждой модели параметра наблюдения определяющими фичами будут географическое положение метеостанций и их результирующая дальность от искомой точки.***

### **1.3.4. Модели пространственной экстраполяции геостатистических данных**

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

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

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

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

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

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

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

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

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

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

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

### 1.3.5. Иные модели.

В тех случаях, когда модели пространственной экстраполяции окажутся неприменимыми, конкретные модели предсказания целевых значений придётся выбирать индивидуально. В зависимости от корреляции выанных 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 [14]:

```
def inverse_distance_avg(row_, param_, station_='Rfrnce_point', df_dists_=df_station_dists, power_=2):
    """
    Расчитывает среднюю, взвешенную по обратным степеням расстояния, исключая данную метеостанцию.
    РАБОТАЕТ ТОЛЬКО ДЛЯ АРХИВА МЕТЕОНАБЛЮДЕНИЙ ПО ПАРАМЕТРАМ!
    РАБОТАЕТ ТОЛЬКО ДЛЯ ПРОСТРАНСТВА МЕТЕОСТАНЦИЙ!
    Принимает:
    - серия значений из которых необходимо рассчитать среднюю, взвешенную по обратным степеням расстояния;
    - название параметра, для которого рассчитывается средняя;
    - название метеостанции для которой рассчитывается средняя (её значение исключается из подсчёта средней!)
        default='Rfrnce_point' - геометрический центр поля метеостанций;
    - df_dists_ DF с расстояниями между метеостанциями default=df_station_dists;
    - power_ степень для вычисления обратных весов default=2
    Возвращает:
    - значение средней, взвешенной по обратным степеням расстояния от точки, заданной параметром stations_
        (вес w = 1000000(d**power_), во избежание слишком малых значений). Во избежание представления 1 в виде
        десятичной дроби с 99..998 после запятой, результат округляется до 12 знака после запятой.
    ПРИМЕЧАНИЕ: чтобы вычислить общую среднюю для всего поля метеостанций без исключений необходимо использовать значение
        staton_ по умолчанию - 'Rfrnce_pint'
    """
    # удаляем из серии значений row_ все значения не являющиеся значениями параметров (символ # в названии индекса -
    # признак того, что этому индексу соответствует значение параметра)
    counter_ = 0 # счётчик для индекса
##    print(row_.index.tolist())
```

```

for item_ in (row_.index.tolist()): # название индекса по индексу серии row_
    if '#' not in item_: # если в названии индекса нет символа '#'
        row_ = row_.drop(row_.index[counter_]) # удаляем из серии элемент (не являющийся значением параметра)
    else:
        counter_ += 1 # в противном случае увеличиваем счётчик на 1

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

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

    # расстояние равно значению строке в df_station_dists где station равно станции, для которой вычисляется средняя
    # а name_st - столбцу в df_station_dists

## 
#     print(station_, name_st_)
distance_ = df_station_dists.loc[(df_station_dists.station == station_), name_st_].values[0]

# если расстояние не равно 0 (чтобы исключить искомую станцию) и текущее значение не является NaN
if (distance_ != 0) and (not pd.isna(param_val_)):
    # добавляем в список значение параметра для этой станции умноженное на обратную степень расстояния
    # умножим его на 1000 в степени power_, чтобы исключить чересчур 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 [15]: # 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 [16]: # 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 [17]: # сохраним стандартные настройки вывода
real_stdout = sys.stdout # сделаем backup стандартного вывода
```

```
In [18]: 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 [19]:

```

# 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 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:**

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

return (lower_level_, upper_level_)

```

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

In [20]:

```

# функция оценки границ вероятности для нормального распределения относительно средней, или средней,
# взвешенной по обратным квадратам расстояния с учетом среднеквартатического отклонения для данного ряда значений.
def norm_probability_limits(
    row_, confidence_=0.95, IDW_=False, param_=None, station_='Rfrnce_point', inclusive_=True):
    """ ТОЛЬКО ДЛЯ АРХИВА ПАРАМЕТРОВ
    Вычисляет пределы в n сигм от средней по обратным квадратам или средней арифметической для серии значений.
    Принимает:
    - row_ - значения из ряда DF (default = None),
    - confidence_ - значение вероятности для оценки границ (float в диапазоне от 0 до 1) (default=0.95),
    - IDW_ (boolean) использовать ли усреднение по обратным квадратам расстояний (default=False),
    - param_ - название параметра для вычисления IDW (default=None). Обязателен для IDW_=True, а так же
        при установки sigma_inclusive_ = False, в противном случае вернёт ошибку,
    - station_ - название метеостанции, для которой вычисляются пределы (значение для неё исключается из подсчёта
        средней и сигм (default='Rfrnce_point' - геометрический центр поля метеостанций),
    - inclusive_ (при IDW_=True имеет смысл только при указании station_, отличной от 'Rfrnce_point') -
        включать ли значение для данной метеостанции в расчёт средней,
        при station_='Rfrnce_point' значение для данной метеостанции будет включено
        в подсчёт средней независимо от значения inclusive_ (boolean, 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 подсчёт не войдёт,
    # но войдут все без исключения метеостанции, для которых значение не равно NaN.
    inclusive_ = True

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

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

# вычисляем границы вероятности распределения confidence для нормального распределения
# относительно mean_value_ с scale равном std_value_, если std_value_ равно NaN или 0 - возвращаем NaN

```

```

result_ = norm.interval(confidence=confidence_,
                      loc=mean_value_,
                      scale=std_value_
                     ) if (pd.notna(std_value_) and (std_value_ != 0)) else np.nan

return result_

```

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

In [21]:

```

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 [22]:

```

# 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_ - флаг использования метода IDW для обработки выбросов
    - param_ - параметры для функции оценки выбросов
    - station_ - название станции
    - inclusive_ - флаг, указывающий, должны ли быть включены в результат выбросы, лежащие на границах
    """

```

- IDW\_ (boolean) использовать ли усреднение по обратным квадратам расстояний (default=False),
- param\_ название параметра для вычисления IDW (default=None). Обязателен для IDW\_=True, а так же при установки sigma\_inclusive\_ = False, в противном случае вернёт ошибку,
- station\_ - название метеостанции, для которой вычисляются пределы (значение для неё исключается из подсчёта средней и сигм (default='Rfrnce\_point' - геометрический центр поля метеостанций),
- inclusive\_ (при IDW\_=True имеет смысл только при указании station\_, отличной от 'Rfrnce\_point') - включать ли значение для данной метеостанции в расчёт средней, при station\_='Rfrnce\_point' значение для данной метеостанции будет включено в подсчёт средней независимо от значения inclusive\_ (boolean, default=True).

Возвращает:

- Если выбросы обнаружены - список из кортежей (на каждый выброс свой кортеж): значение выброса, его индекс в серии, границы 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 [23]:

```

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 п 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 - может единственное значение в серии
        idx_ = df_.columns.get_loc(label_)
        # добавить выброс и его параметры в список:
        list_outliers_.append((checked_value_, idx_, np.nan, np.nan))

```

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

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

```
In [24]: 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 [25]: 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 [26]:

```
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 [27]:

```

# 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 [28]: # 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 [29]: 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 [30]: 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 питчу!
    # Если в массив индексов NaN пустой (есть все нужные значения) или
    # если разница между длинной ряда и длинной массива индексов NaN меньше 3 (ограничение сэмплов для вариограммы)
    # то выйти из функции
    if (len(arr_idx_nan_) == 0) or ((len(row_) - len(arr_idx_nan_)) < 3):
        #
        #     print(f"\nВыход из функции из-за количества значений."
        #           f"Количество переданных определённых значений={len(row_) - len(arr_idx_nan_)}."
        #
        #
        return # выход из функции
```

```

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

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

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

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

#     return None

```

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

См. тетрадь 2

### 4. Исследование и обработка индивидуальных показателей метеорологических наблюдений: Атмосферное давление ( $P_{sea}$ ,

## P\_station, P\_drift)

Параметры давления являются взаимновычисляемыми и должны согласовываться друг с другом.

Вычисление давления на высоте  $h$  можно произвести с помощью барометрической формулы, исходя из давления на уровне моря  $P_0$  и температуры воздуха  $T$ :

$$P = P_0 e^{-Mgh/RT}$$

где

$P_0$  — давление на уровне моря (Па);

$M$  — молярная масса сухого воздуха,  $M = 0,029\text{кг}/\text{моль}$ ;

$g$  — ускорение свободного падения,  $g = 9,81\text{м}/\text{с}^2$ ;

$R$  — универсальная газовая постоянная,  $R = 8,31\text{Дж}/\text{моль} \cdot \text{К}$ ;

$T$  — абсолютная температура воздуха, К,  $T = t + 273,15$ , где  $t$  — температура Цельсия, выражаемая в градусах Цельсия (обозначение: °C);

$h$  — высота, м.

Справочно:

$$1mmHg = 133.321995Pa$$

$$1Pa = 0.007501mmHg$$

Для небольших высот (до 1000 метров) в метеорологических наблюдениях используется формула Бабинэ:  $h = 16000(1 + 0.004t_m)^{\frac{p_1 - p_2}{p_1 + p_2}}$ , где

- $h$  — превышение, в м;
- $p_1$  и  $p_2$  — давление воздуха на нижнем и верхнем уровнях, в гПа, мбар или мм рт. ст.;
- $t_m$  — средняя температура воздуха слоя между уровнями;
- 0,004 — коэффициент расширения газа.

Но, для того, чтобы вычислить среднее значение температуры между уровнем моря и уровнем метеостанции, нам не хватает данных (температуры на уровне моря). Поэтому, будем использовать 1-ю барометрическую формулу.

**Определим две функции для расчёта давления в зависимости от высоты над уровнем моря и температуры**

Эти функции потребуются для работы с показателями давления

```
In [31]: # Функция расчёта давления на уровне поверхности на земле, исходя из давления на уровне моря.  
def P0_to_P(pHg0_, h_, t_):  
    """ Расчёт давления на уровне точки, исходя из давления на уровне моря  
    Принимает: давление на уровне моря в mm Hg, высоту, температуру в градусах Цельсия  
    Возвращает: давление на уровне точки в mm Hg"""\n    T_ = t_ + 273.15 # Переводим температуру в Кельвина\n    p0_ = pHg0_ * 133.321995 # Переводим давление в Па\n    p_ = p0_ * np.e ** (-(0.029 * 9.81 * h_)/(8.31 * T_))\n    pHg_ = p_ / 133.321995 # Переводим давление в mm Hg\n    return pHg_\n\n# Функция расчёта давления на уровне моря, исходя из давления на уровне поверхности на земле.  
def P_to_P0(pHg_, h_, t_):  
    """ Расчёт давления на уровне моря, исходя из давления на уровне точки  
    Принимает: давление на уровне точки в mm Hg, высоту, температуру в градусах Цельсия  
    Возвращает: давление на уровне моря в mm Hg"""\n    T_ = t_ + 273.15 # Переводим температуру в Кельвина\n    p_ = pHg_ * 133.321995 # Переводим давление в Па\n    p0_ = p_ / (np.e ** (-(0.029 * 9.81 * h_)/(8.31 * T_)))\n    pHg0_ = p0_ / 133.321995 # Переводим давление в mm Hg\n    return pHg0_
```

## 4.1. Атмосферное давление: P\_sea (атмосферное давление на уровне моря)

Показатели атмосферного давления должны фиксироваться метеостанциями каждые 3 часа в установленные часы. Показатель давления на уровне моря часто рассчитывается по барометрической формуле или по барометрическим таблицам из значений показателя давления на уровне метеостанции. Показатель давления на уровне моря является универсальным значением для всех метеостанций. Анализ аномалий и выбросов необходимо производить именно в нём, так как давление на уровне метеостанции зависит еще и от высоты метеостанции над уровнем моря. Единый атмосферный процесс, приводящий к одинаковым погодным явлениям, виден в первую очередь в едином изменении давления на уровне моря.

### 4.1.1. Поиск и удаление ошибок показателя давления P\_sea

Для данного раздела обозначим константу названия параметра

```
In [32]: PARAMETER41 = 'P_sea'
```

### Создаём временный DF для работы с параметром P\_sea

```
In [33]: param_df_name = f'df_{PARAMETER41}' # преобразуем полученное значение в df_PARAMETER41 - ключ словаря dict_df_parameters
# Создадим временный df
df_tmp41 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp41.sample(5, random_state=56)
```

Out[33]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2014-02-22 03:00:00	766.4	766.9	768.4	767.8	768.1	768.6	767.2	767.4	767.6
2015-05-16 03:00:00	749.2	748.5	745.7	747.1	745.8	745.0	746.8	747.0	747.1
2020-06-18 18:00:00	761.2	760.9	763.1	761.7	761.4	761.7	760.6	761.3	760.9
2019-12-02 21:00:00	756.0	757.4	757.3	757.3	758.2	758.5	758.2	759.1	758.5
2008-06-05 18:00:00	NaN	761.8	NaN	760.4	761.2	760.8	761.6	761.5	761.6

Определим последовательность действий, для поиска ошибок показателя "Атмосферное давление на уровне моря".

Изменения в давлении происходят гораздо более постепенно, чем изменения показателя температуры. Резкие скачки давления, как локальные, так и во временном окне, могут встречаться гораздо реже. Поэтому критерий, по которому выброс будет трактоваться как ошибка здесь жёстче.

Атмосферное давление - величина менее вариабельная по сравнению с температурой воздуха. Обычно на метеостанциях измеряемое атмосферное давление пересчитывается в давление на уровне моря.

1. Проверяем соответствие давления на уровне метеостанции давлению на уровне моря (кроме значений NaN для давления на уровне моря. При несоответствии - обозначаем как ошибки и удаляем.
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.

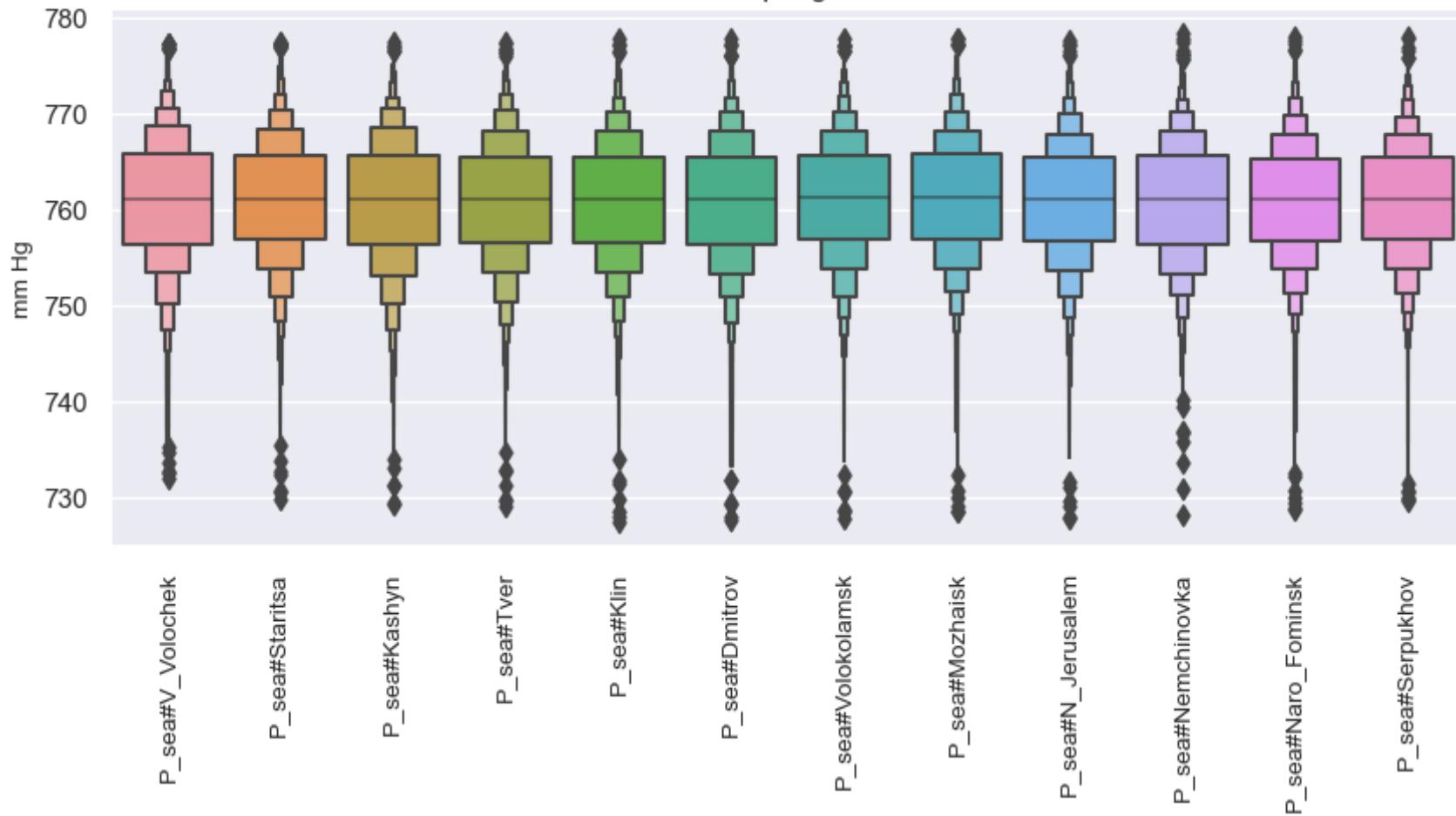
### Визуализируем архив давления на уровне моря (P\_sea) по сезонам

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

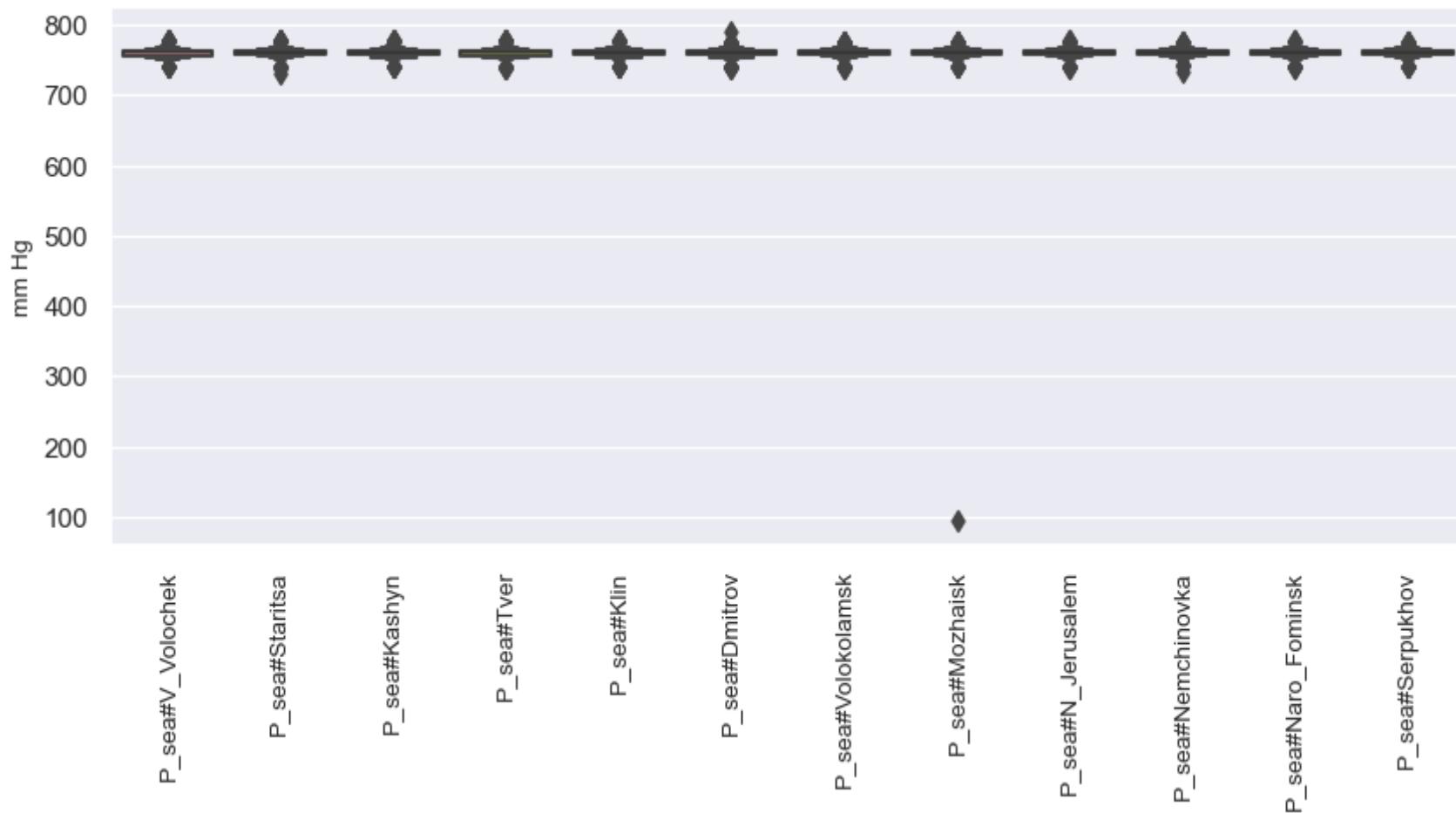
- Зима (ниже 0°): В среднем длится с 5 ноября по 4 апреля
- Весна (от 0° до +10°): В среднем длится с 5 апреля по 18 мая
- Лето (выше +10°): В среднем длится с 19 мая по 14-15 сентября
- Осень (от +10° до 0°): В среднем длится с 14-15 сентября по 4 ноября

```
In [34]: # В цикле выведем графики давления по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp41).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp41[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('мм Hg', size=10)
    dummy = g.set_title(f'Распределение значений P_sea в разрезе метеостанций:\n{season_name}')
plt.show()
```

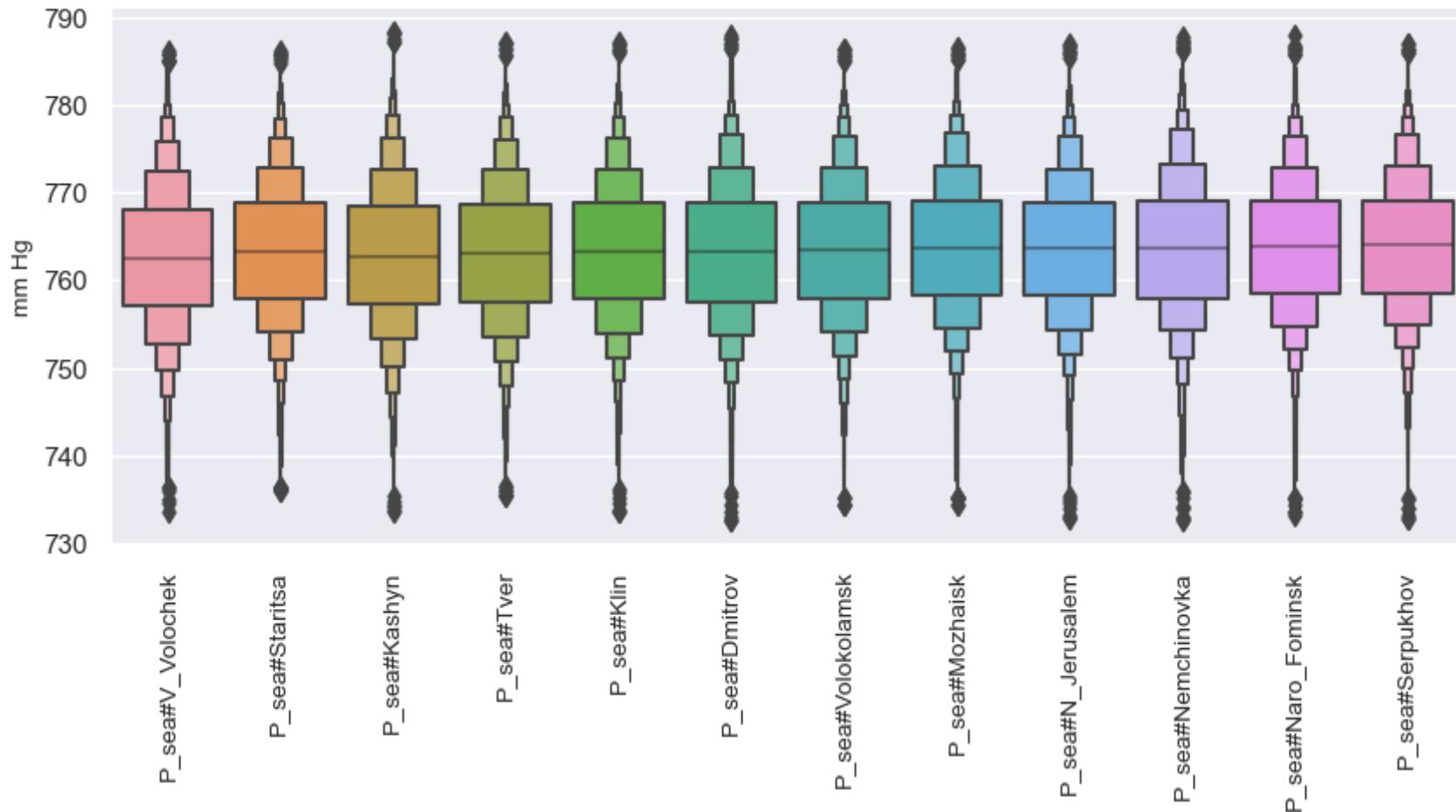
Распределение значений P\_sea в разрезе метеостанций:  
spring



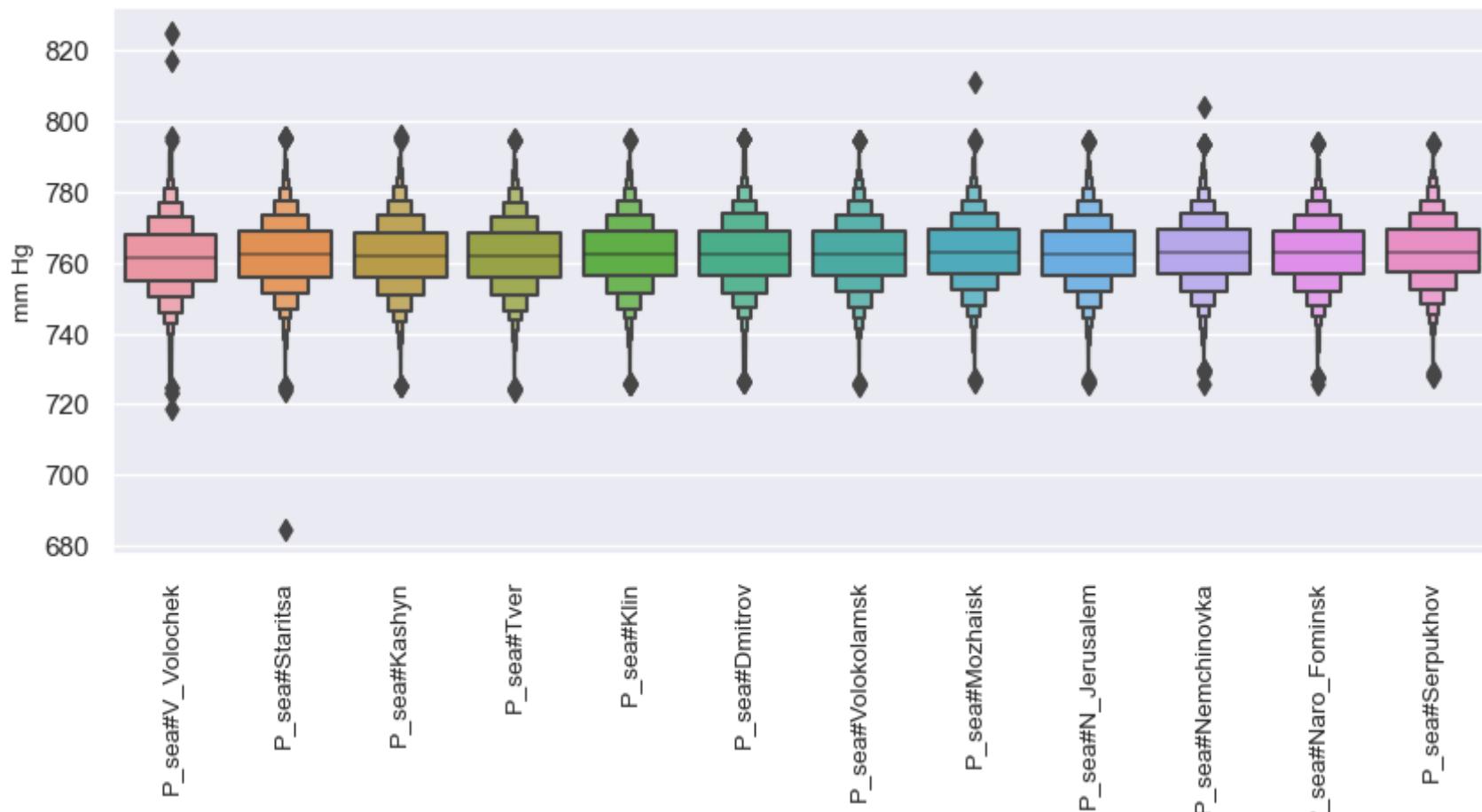
Распределение значений P\_sea в разрезе метеостанций:  
summer



Распределение значений P\_sea в разрезе метеостанций:  
autumn



## Распределение значений P\_sea в разрезе метеостанций: winter



Даже беглого взгляда на графики достаточно, чтобы понять, что в данных содержится ошибки, выразившиеся в нереальных значениях давления.

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

```
In [35]: print(f'Минимальное значение: {np.nanmin(df_tmp41)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp41)},\n'
      f'Средняя: {np.nanmean(df_tmp41)},\n'
      f'Медиана: {np.nanmedian(df_tmp41)})'
```

Минимальное значение: 93.8,  
Максимальное значение: 824.6,  
Средняя: 761.657022051166,  
Медиана: 761.5

**Попытаемся восстановить пропущенные значения в df\_tmp41 расчётными значениями по барометрической формуле, используя данные об атмосферном давлении на уровне метеостанции**

Не исключено, что при этом мы перенесём в df\_tmp41 ошибки, имеющиеся в значениях атмосферного давления на уровне метеостанции, но мы их сможем идентифицировать при анализе выбросов в поле метеостанций и во временном окне, что проделаем ниже, в соответствующем разделе.

*Выясним количество пропущенных значений в df\_tmp41*

```
In [36]: np.isnan(df_tmp41).sum().sum()
```

```
Out[36]: 56323
```

*Определим их координаты в DF и заменим их на расчётные значения из значений атмосферного давления на уровне метеостанции*

```
In [37]: for col in df_tmp41.columns: # По столбцам в df_tmp41
    station = col[len(PARAMETER41)+1:] # определяем название метеостанции из названия столбца
    for moment in df_tmp41.index: # по моментам наблюдения
        if np.isnan(df_tmp41.at[moment, col]): # Если в текущей ячейке находится NaN
            # Присваиваем текущей ячейке результат функции P_to_P0
            # (приведение давления на уровне метеостанции к давлению на уровне моря):
            df_tmp41.at[moment, col] =
                P_to_P0(pHg=reference_param_extractor(station=station, # получаем референсное значение для давления
                                                       param_='P_station',
                                                       dict_archive_=dict_df_locations,
                                                       idx_station_=moment),
                        h=df_station_dists[df_station_dists.station == station] # получаем значение высоты станции
                        .height.values[0],
                        t=reference_param_extractor(station=station, # получаем референсное значение температуры
                                                       param_='T',
                                                       dict_archive_=dict_df_locations,
                                                       idx_station_=moment))
        )
    else:
        pass
```

```
In [38]: np.isnan(df_tmp41).sum().sum() # Проверяем оставшее количество NaN
```

```
Out[38]: 56159
```

Таким образом удалось заменить только около 160 значений

### Проверим соответствие давления на уровне метеостанции давлению на уровне моря по всему DF

Определим расхождения между давлением на уровне моря и расчётым давлением, полученным по барометрической формуле от давления на уровне станции

Это целесообразно сделать в самом начале рассмотрения показателя P\_sea, чтобы исключить влияние некорректных значений на поиск выбросов.

Определим критерий для некорректных значений в 1,0 mmHg и более (Предположим, что использовалась барометрическая формула, сама имеющая некоторую погрешность, пусть и незначительную для малых высот, плюс могла возникнуть ошибка округления при расчётах)

```
In [39]: criterium = 1 # критерий для определения допустимой погрешности
# расчётного значения давления на уровне станции от архивного

# В цикле по столбцам df_tmp41
# - определим промежуточный DF для расчётов;
# - заполним его значениями, необходимыми для приведения давления к уровню моря,
# - расчитаем значение давления на уровне моря по значению на уровне станции,
# - рассчитаем абсолютное отклонение расчётных значений от архивных для давления на уровне моря,
# - определим соответствие архивных значений критерию ошибки
# Исходим из идентичности индексов всех датафреймов в архивах

list_error_at = [] # определим список координат ошибочных значений в df_tmp41
for col in df_tmp41.columns:
    df_p0 = pd.DataFrame(None, index=df_tmp41.index) # создадим пустой DF
    station_name = col[len(PARAMETER41)+1:] # выделяем название метеостанции из названия столбца
    height = df_station_dists[df_station_dists.station == station_name].height.values[0] # находим высоту метеостанции

    df_p0 = (df_p0
              .assign(station=station_name,
                     P0=df_tmp41.loc[:,PARAMETER41+'#'+station_name], # давление на уровне моря
                     P1=dict_df_parameters['df_P_station'][f'{P_station}'+col[len(PARAMETER41):]]), # давление на уровне станции
```

```

T=dict_df_parameters['df_T'][f'{ 'T'+col[len(PARAMETER41):]}"]], # Температура воздуха
h=height, # высота над уровнем моря
P0_calc=lambda x: P_to_P0(pHg_=x.P1, h_=x.h, t_=x["T"]), # Расчётная величина давления на уровне моря
P0_delta=lambda x: abs(x.P0 - x.P0_calc), # абсолютное отклонение расчётной величины от архивной
P0_error=lambda x: x.P0_delta >= criterium, # критерий ошибки для отклонения расчётной величины, mmHg
)
)
#
# Расширяем список координат ошибочных значений в df_tmp41:
# Если абсолютное отклонение расчётного значения от архивного больше или равно установленного критерия:
# прибавляем список кортежей моментов наблюдения и метеостанций по которым значения превышают допустимую погрешность
list_error_at = list_error_at + (df_p0
    .apply(
        lambda x: (x.name, station_name) if x.P0_delta >= criterium else np.nan,
        axis=1)
    .dropna()
    .tolist()
)
)

print(f'Количество аномальных отклонений расчётного давления на уровне моря от архивного '
      f'для метеостанции {station_name} = {df_p0.P0_error.sum()}'')
#list_error_at

```

Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции V\_Volochev = 137  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Staritsa = 70  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Kashyn = 14  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Tver = 49726  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Klin = 76  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Dmitrov = 77  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Volokolamsk = 11  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Mozhaisk = 74  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции N\_Jerusalem = 77  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Nemchinovka = 67  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Naro\_Fominsk = 60  
Количество аномальных отклонений расчётного давления на уровне моря от архивного для метеостанции Serpukhov = 1529

Удаляем некорректные значения

```
In [40]: for error in list_error_at:
    df_tmp41.at[error[0], PARAMETER41 + '#' + error[1]] = np.nan
```

## Найдем выбросы в поле метеостанций.

Параметры:

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

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

df_tmp41 = df_tmp41.assign(field_out=df_tmp41.apply(lambda x: field_outliers(row=x,
                                                               method='sigma',
                                                               criterium=3,
                                                               IDW=True,
                                                               param=PARAMETER41,
                                                               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: 00:04:43
```

```
In [42]: df_tmp41.dropna(subset=["field_out"]).sample(5, random_state=56)
```

Out[42]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2019-07-08 21:00:00	751.0	751.6	751.7	NaN	751.8	751.7	751.8	751.9	751.9
2016-11-28 06:00:00	748.0	746.3	746.9	NaN	746.2	746.3	746.3	746.3	746.3
2006-01-05 00:00:00	NaN	783.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013-09-23 15:00:00	739.6	738.4	741.3	NaN	738.9	739.7	737.8	738.8	738.8
2014-03-19 15:00:00	746.7	745.9	748.6	NaN	746.2	746.9	746.2	746.5	746.5

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 [43]:

```
# 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_tmp41.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: 18

Проверим максимальное количество выбросов в поле метеостанций на момент наблюдения.
```

In [44]:

```
# Находим максимальное количество выбросов в строке поля метеостанций
max_field_outliers = df_tmp41.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 [45]: # Определяем столбец error_at (помним, что в field_out у нас всегда БОЛЕЕ 1 выброса),  
# значит вторым элементом кортежа в error_at будет СПИСОК станций, по которым найдены выбросы  
  
df_tmp41 = df_tmp41.assign(error_at =  
                           df_tmp41.  
                           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_=PARAMETER41)  
                                 ) if isinstance(x.field_out, list) else np.nan,  
                                 axis=1  
                           )  
                           )
```

```
In [46]: df_tmp41.dropna(subset=["error_at"]).sample(5, random_state=56)
```

Out[46]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusal
<b>2019-07-08 21:00:00</b>	751.0	751.6	751.7	NaN	751.8	751.7	751.8	751.9	751.
<b>2016-11-28 06:00:00</b>	748.0	746.3	746.9	NaN	746.2	746.3	746.3	746.3	746
<b>2006-01-05 00:00:00</b>	NaN	783.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2013-09-23 15:00:00</b>	739.6	738.4	741.3	NaN	738.9	739.7	737.8	738.8	738
<b>2014-03-19 15:00:00</b>	746.7	745.9	748.6	NaN	746.2	746.9	746.2	746.5	746

**◀ ▶**

Для подтверждения, являются ли отобранные значения давления на уровне моря ошибками, найдём выбросы во временном ряду

Для анализа выбросов во временном окне следует иметь в виду, что атмосферное давление не подвержено столь большим суточным колебаниям, как температура, и изменения атмосферного давления носят гораздо более плавный характер. Поэтому целесообразно изучить выбросы во временном окне с минимальными интервалами (то есть 3 часа) в течение небольшого периода.

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 +/- 12 часов ('12H'),
- включим в подсчёт все моменты наблюдения (equalhours=False).

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 [47]: start_time = time.time() # для замера времени выполнения кода

# Нельзя удалять NaN в поле field_out непосредственно в df_tmp41 (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_tmp41.index, dtype='object', name="time_out_24H")

for elem in df_tmp41.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], PARAMETER41+'#'+station)

        # Вызываем функцию time_outliers с обозначенными выше параметрами
        val_func = time_outliers(df=df_tmp41,
                                  # В качестве серии row_ передаём серию из одного значения: на момент наблюдения,
                                  # где индекс серии соответствует названию столбца (idxs[1])
                                  row_=df_tmp41.loc[idxs[0]][df_tmp41.loc[idxs[0]].index == idxs[1]],
                                  td_symbol_='H',
                                  td_quant_='12',
                                  equal_hours_=False,
                                  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: 00:00:01

Присоединим полученную серию к df\_tmp41

```

In [48]: # Индекс серии совпадает с индексом всех архивов, а значения в серии упорядочены по этому индексу.
# Поэтому произведём объединение по индексу
df_tmp41 = (df_tmp41
            .merge(ser_time_out, left_index=True, right_index=True) # производим объединение
            )

```

```

In [49]: # Выводим случайные строки из df_tmp41, удалив NaN в столбце time_out_24H
df_tmp41.dropna(subset=["time_out_24H"]).sample(5, random_state=56)

```

	P_sea#V_Volochev	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
<b>2019-07-08 21:00:00</b>	751.0	751.6	751.7	NaN	751.8	751.7	751.8	751.9	751.9
<b>2016-11-28 06:00:00</b>	748.0	746.3	746.9	NaN	746.2	746.3	746.3	746.3	746.3
<b>2006-01-05 00:00:00</b>	NaN	783.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
<b>2013-09-23 15:00:00</b>	739.6	738.4	741.3	NaN	738.9	739.7	737.8	738.8	738.8
<b>2014-03-19 15:00:00</b>	746.7	745.9	748.6	NaN	746.2	746.9	746.2	746.5	746.5

Заменим пустые списки в столбце time\_out\_24H на NaN

```
In [50]: df_tmp41.time_out_24H = df_tmp41.time_out_24H.apply(lambda x: np.nan if x == [] else x)
```

```
In [51]: df_tmp41.dropna(subset=["time_out_24H"]).sample(5, random_state=56)
```

Out[51]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2014-06-05 18:00:00	764.7	764.5	765.1	NaN	764.5	764.7	764.9	764.6	751.1
2012-02-28 09:00:00	762.3	762.8	761.5	NaN	761.5	761.4	761.8	761.8	761.1
2017-04-11 18:00:00	749.2	750.9	750.9	NaN	751.4	751.1	751.6	751.6	751.1
2019-03-21 12:00:00	763.6	764.5	763.5	NaN	764.3	764.2	764.3	765.1	764.1
2015-06-18 12:00:00	761.6	762.0	761.9	NaN	762.1	762.1	762.1	762.1	753.1

## Применимость критериев ошибочных значений

Рассмотрим полученные данные об аномальных значениях.

```
In [52]: # 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_tmp41 = (df_tmp41  
    .assign(double_error_at =  
        df_tmp41.dropna(subset=["time_out_24H"])  
        .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_24H', # используем выбросы во временном окне
```

```
        param_=PARAMETER41)
    ) if isinstance(x.field_out, list) else np.nan,
    axis=1
)
)
```

```
In [53]: df_tmp41.dropna(subset=["double_error_at"])
```

Out[53]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2020-06-18 12:00:00	762.2	762.1	764.2	762.7	762.4	762.4	762.1	762.1	762.1
2019-11-25 03:00:00	773.0	774.2	774.3	NaN	775.0	775.3	774.9	775.6	771.1
2019-06-04 09:00:00	767.6	768.6	768.6	757.5	769.0	769.1	768.9	769.2	769.2
2019-03-21 12:00:00	763.6	764.5	763.5	NaN	764.3	764.2	764.3	765.1	764.1
2018-10-13 03:00:00	772.2	773.0	770.9	NaN	772.4	771.9	773.3	774.1	774.1
2018-06-16 18:00:00	764.5	764.5	765.0	NaN	764.4	764.5	764.3	764.3	764.3
2017-06-15 00:00:00	751.5	751.2	747.6	NaN	749.4	748.6	750.1	93.8	749.1
2017-04-11 18:00:00	749.2	750.9	750.9	NaN	751.4	751.1	751.6	751.6	751.6
2016-12-14 03:00:00	760.9	762.4	761.7	NaN	763.2	763.4	763.6	810.8	763.6
2016-09-12 00:00:00	764.2	764.8	765.3	NaN	765.1	765.3	765.0	765.3	764.2
2015-06-18 12:00:00	761.6	762.0	761.9	NaN	762.1	762.1	762.1	762.1	753.1



P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2009-07-29 15:00:00	764.3	764.5	755.8	NaN	763.6	763.0	763.7	763.5
2009-07-18 21:00:00	759.8	759.9	759.0	NaN	760.1	760.0	760.2	760.0
2009-06-30 09:00:00	770.8	770.6	770.5	NaN	769.8	769.7	770.0	769.8
2009-02-20 21:00:00	774.1	774.5	774.5	NaN	773.9	774.0	773.7	773.6
2008-09-30 15:00:00	754.7	771.1	754.7	NaN	756.3	756.0	756.7	757.3
2008-04-28 15:00:00	766.3	766.4	767.0	NaN	766.6	766.6	766.7	770.0
2007-01-28 09:00:00	745.4	747.2	745.7	NaN	746.4	746.0	746.9	747.3
2006-08-24 15:00:00	760.4	752.6	754.2	753.0	753.8	754.4	753.1	753.4
2006-02-16 09:00:00	769.7	769.9	770.8	NaN	770.2	770.8	770.2	770.6
2005-09-21 21:00:00	769.0	769.5	769.3	NaN	770.0	770.2	770.2	770.5

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 [54]: df_tmp41 = df_tmp41.assign(  
    cross_check = df_tmp41  
        .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 [55]: df_tmp41.dropna(subset=["cross_check"]).sample(7, random_state=56)
```

Out[55]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2014-06-05 18:00:00	764.7	764.5	765.1	NaN	764.5	764.7	764.9	764.6	754.5
2012-02-28 09:00:00	762.3	762.8	761.5	NaN	761.5	761.4	761.8	761.8	761.5
2017-04-11 18:00:00	749.2	750.9	750.9	NaN	751.4	751.1	751.6	751.6	751.5
2019-03-21 12:00:00	763.6	764.5	763.5	NaN	764.3	764.2	764.3	765.1	764.5
2015-06-18 12:00:00	761.6	762.0	761.9	NaN	762.1	762.1	762.1	762.1	755.5
2005-09-21 21:00:00	769.0	769.5	769.3	NaN	770.0	770.2	770.2	770.5	770.0
2012-10-09 21:00:00	752.9	753.5	753.5	NaN	753.2	753.1	753.6	753.5	752.5



In [56]:

```
# Подсчитаем количество выбросов в поле метеостанций
count_field_out = (df_tmp41
    .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_tmp41
    .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_tmp41
    .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 +/- 12 часов 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: 236,

из них:

Подтверждается аномалиями в промежутке +/- 12 часов по всем моментам наблюдения: 32

Проверка: пересечение множеств выбросов в поле метеостанций и во временном окне насчитывает 32 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 +/- 12 часов (выбивается из общей тенденции изменения давления)

### **Удалим (заменим на NaN) все ошибочные значения**

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

In [57]: df\_tmp41.dropna(subset=['cross\_check'])

Out[57]:	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2020-06-18 12:00:00	762.2	762.1	764.2	762.7	762.4	762.4	762.1	762.1	762.1
2019-11-25 03:00:00	773.0	774.2	774.3	NaN	775.0	775.3	774.9	775.6	771.1
2019-06-04 09:00:00	767.6	768.6	768.6	757.5	769.0	769.1	768.9	769.2	769.2
2019-03-21 12:00:00	763.6	764.5	763.5	NaN	764.3	764.2	764.3	765.1	764.1
2018-10-13 03:00:00	772.2	773.0	770.9	NaN	772.4	771.9	773.3	774.1	772.1
2018-06-16 18:00:00	764.5	764.5	765.0	NaN	764.4	764.5	764.3	764.3	764.1
2017-06-15 00:00:00	751.5	751.2	747.6	NaN	749.4	748.6	750.1	93.8	749.1
2017-04-11 18:00:00	749.2	750.9	750.9	NaN	751.4	751.1	751.6	751.6	751.1
2016-12-14 03:00:00	760.9	762.4	761.7	NaN	763.2	763.4	763.6	810.8	763.1
2016-09-12 00:00:00	764.2	764.8	765.3	NaN	765.1	765.3	765.0	765.3	764.1
2015-06-18 12:00:00	761.6	762.0	761.9	NaN	762.1	762.1	762.1	762.1	753.1



P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2009-07-29 15:00:00	764.3	764.5	755.8	NaN	763.6	763.0	763.7	763.5
2009-07-18 21:00:00	759.8	759.9	759.0	NaN	760.1	760.0	760.2	760.0
2009-06-30 09:00:00	770.8	770.6	770.5	NaN	769.8	769.7	770.0	769.8
2009-02-20 21:00:00	774.1	774.5	774.5	NaN	773.9	774.0	773.7	773.6
2008-09-30 15:00:00	754.7	771.1	754.7	NaN	756.3	756.0	756.7	757.3
2008-04-28 15:00:00	766.3	766.4	767.0	NaN	766.6	766.6	766.7	770.0
2007-01-28 09:00:00	745.4	747.2	745.7	NaN	746.4	746.0	746.9	747.3
2006-08-24 15:00:00	760.4	752.6	754.2	753.0	753.8	754.4	753.1	753.4
2006-02-16 09:00:00	769.7	769.9	770.8	NaN	770.2	770.8	770.2	770.6
2005-09-21 21:00:00	769.0	769.5	769.3	NaN	770.0	770.2	770.2	770.5

```
In [58]: # Создадим список координат ячеек, содержащих значения, определённые как ошибки
list_error_coords = df_tmp41.dropna(subset=['double_error_at']).double_error_at.tolist()
# list_error_coords
```

```
In [59]: # Восстановим исходное состояние df_tmp1
df_tmp41 = dict_df_parameters[param_df_name].copy(deep=True)
```

```
In [60]: 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
        # Преобразуем координату столбца и присваиваем ячейке NaN
        df_tmp41.at[error_coords[0], PARAMETER41+'#' + station] = np.nan
for error in list_error_at: # по списку координат выявленных выше значений, аномально не совпадающих с расчётными
    df_tmp41.at[error[0], PARAMETER41 + '#' + error[1]] = np.nan
# df_tmp41
```

```
In [61]: # Проверим, все ли ошибки заменены на NaN
# Количество нeNaN значений в df_tmp41 по координатам, указанным в списках list_error_coords и list_error_at

counter_notna1 = 0 # счётчик нeNaN значений
for error_coords in list_error_coords: # по списку координат ошибочных значений
    for station in error_coords[1]: # перечню метеостанций в каждом списке координат ошибочных значений для станций
        # подсчитаем количество нeNaN значений и прибавим их к счётчику нeNaN значений.
        counter_notna1 += np.sum(pd.notna(df_tmp41.at[error_coords[0], PARAMETER41+'#' + station]))
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna1}')

counter_notna2 = 0 # счётчик нeNaN значений
for error in list_error_at: # по списку координат выявленных выше значений, аномально не совпадающих с расчётными
    counter_notna1 += np.sum(pd.notna(df_tmp41.at[error[0], PARAMETER41 + '#' + error[1]]))
print(f'По результатам удаления выявленных несоответствий архивных и расчётных величин осталось значений: '
      f'{counter_notna2}')
# df_tmp41
```

По результатам удаления выявленных аномальных выбросов осталось значений: 0

По результатам удаления выявленных несоответствий архивных и расчётных величин осталось значений: 0

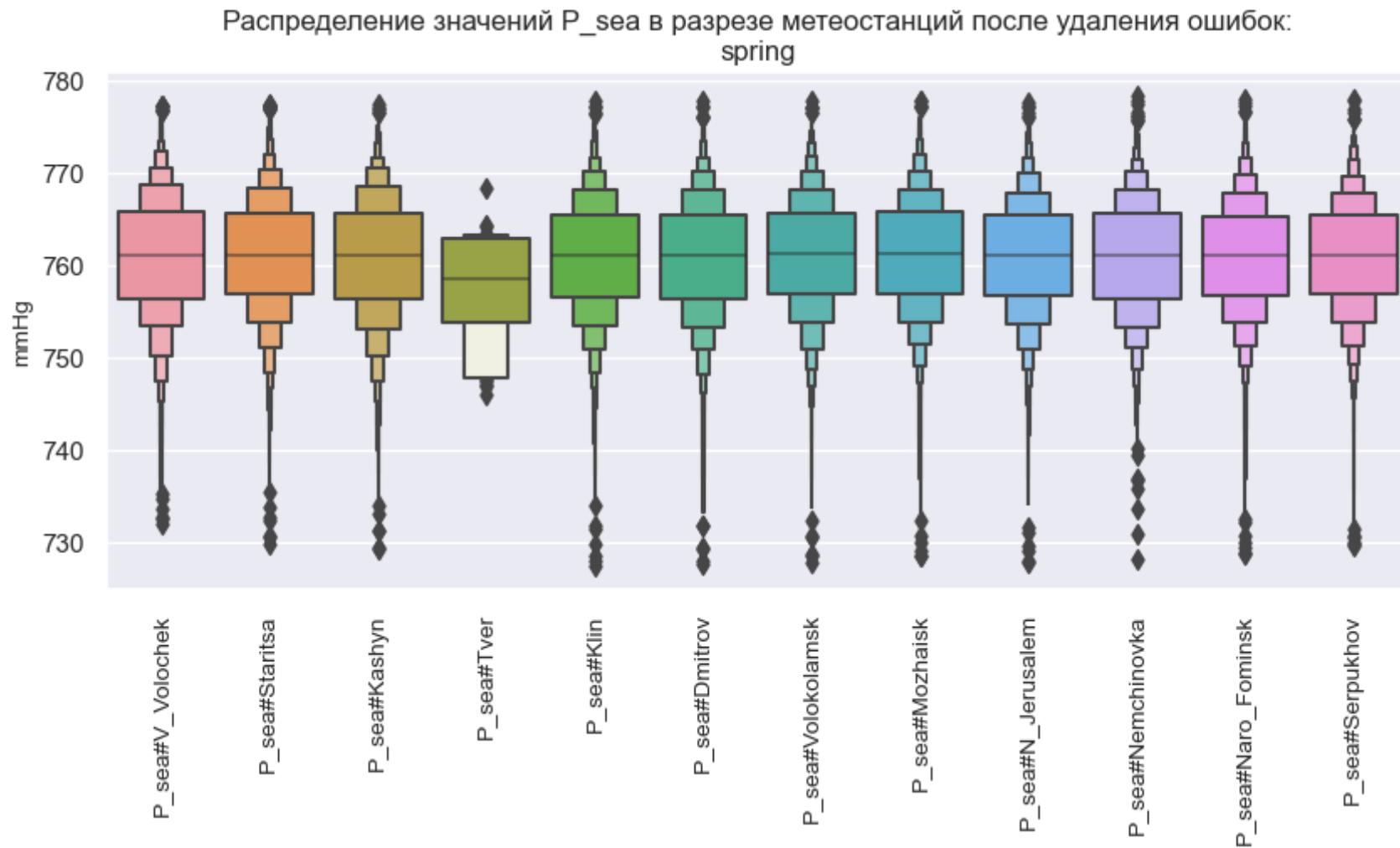
### **Визуализируем архив атмосферного давления на уровне метеостанции (P\_sea) по сезонам после удаления ошибок**

```
In [62]: # В цикле выведем графики давления на уровне моря по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp41).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp41[season_mask],
```

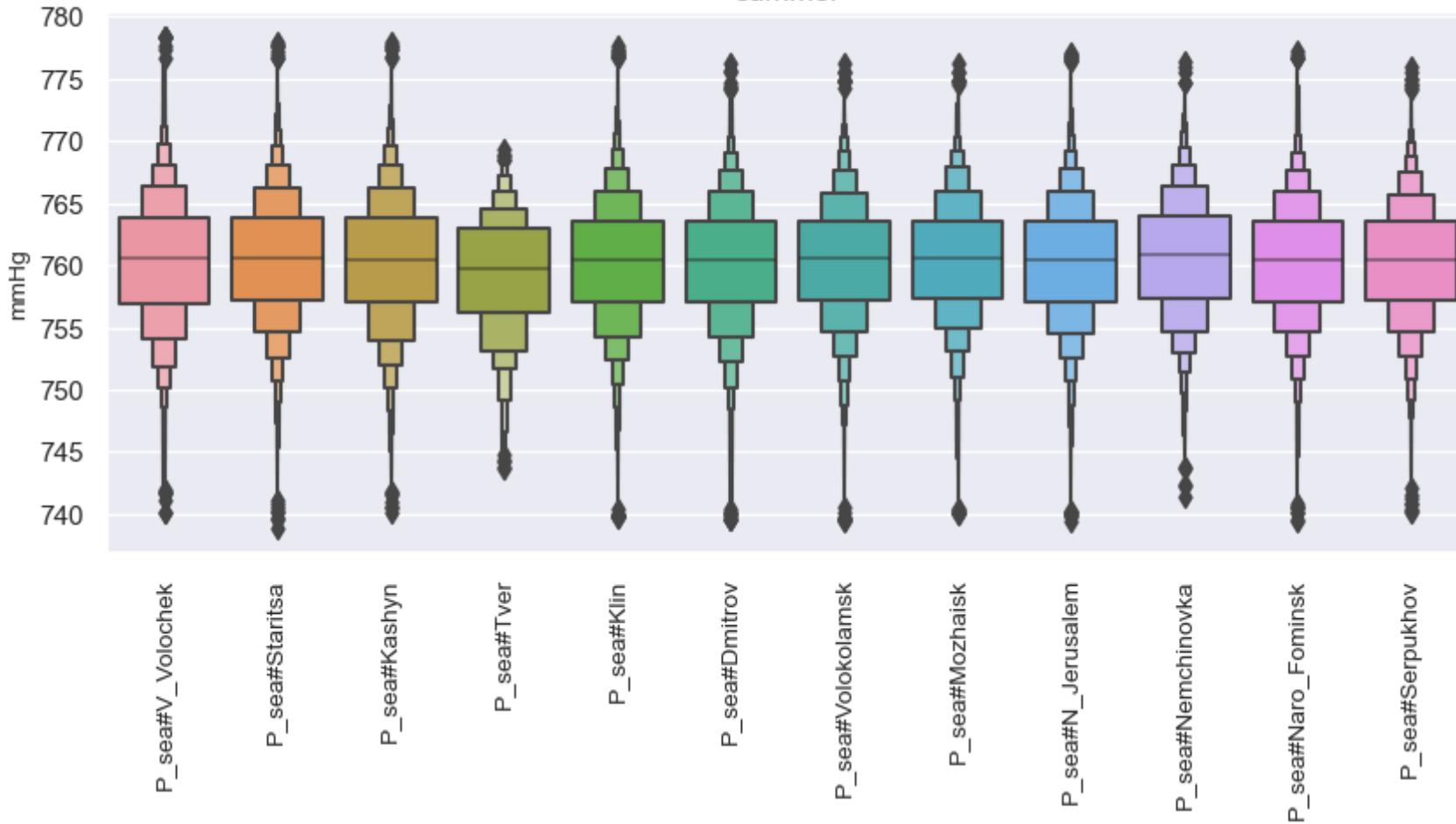
```

        ax=ax)
dummy = plt.xticks(rotation=90, size=10)
dummy = g.set_ylabel('mmHg', size=10)
dummy = g.set_title(f'Распределение значений P_sea в разрезе метеостанций после удаления ошибок:\n{season_name}')
plt.show()

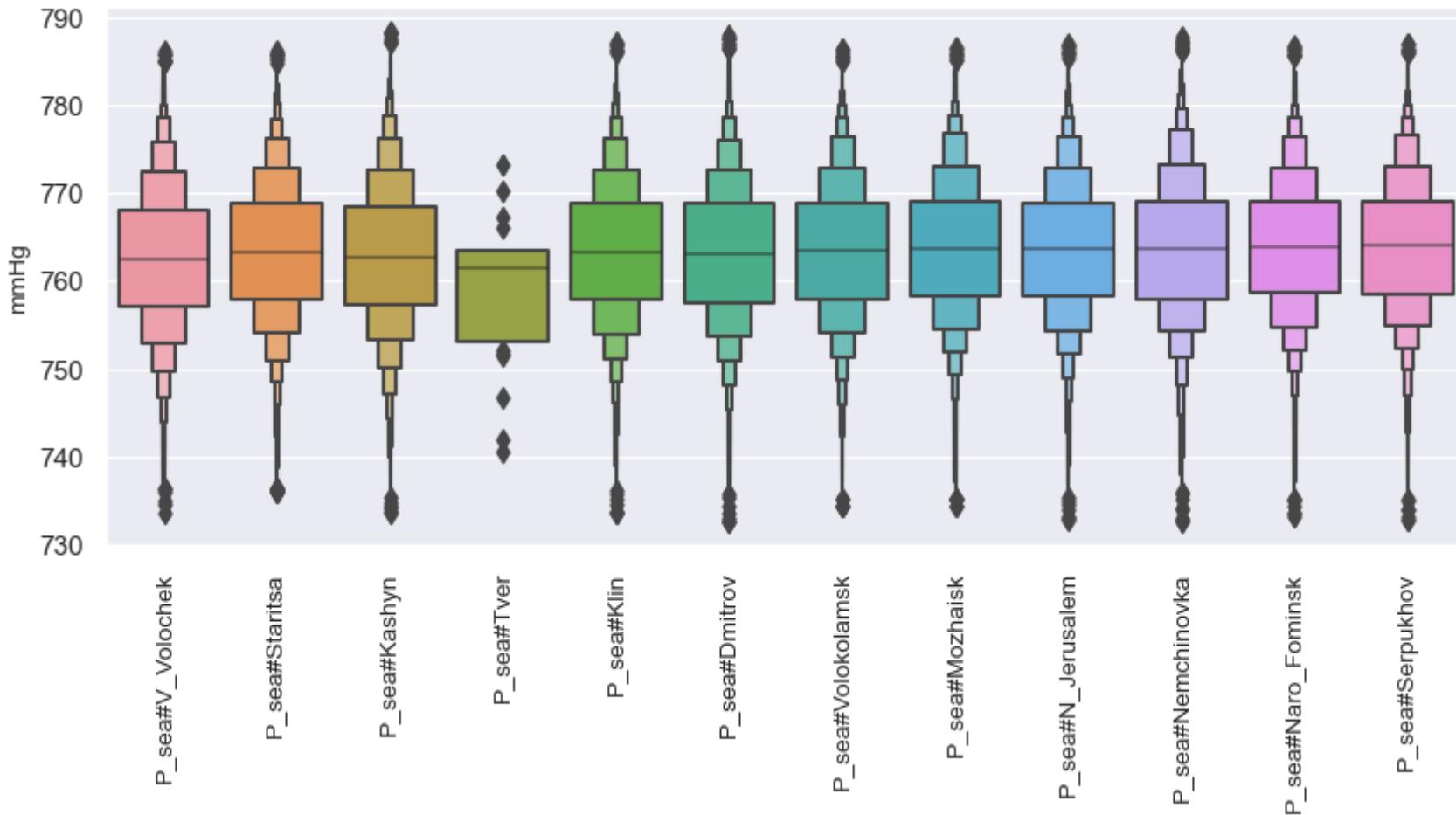
```



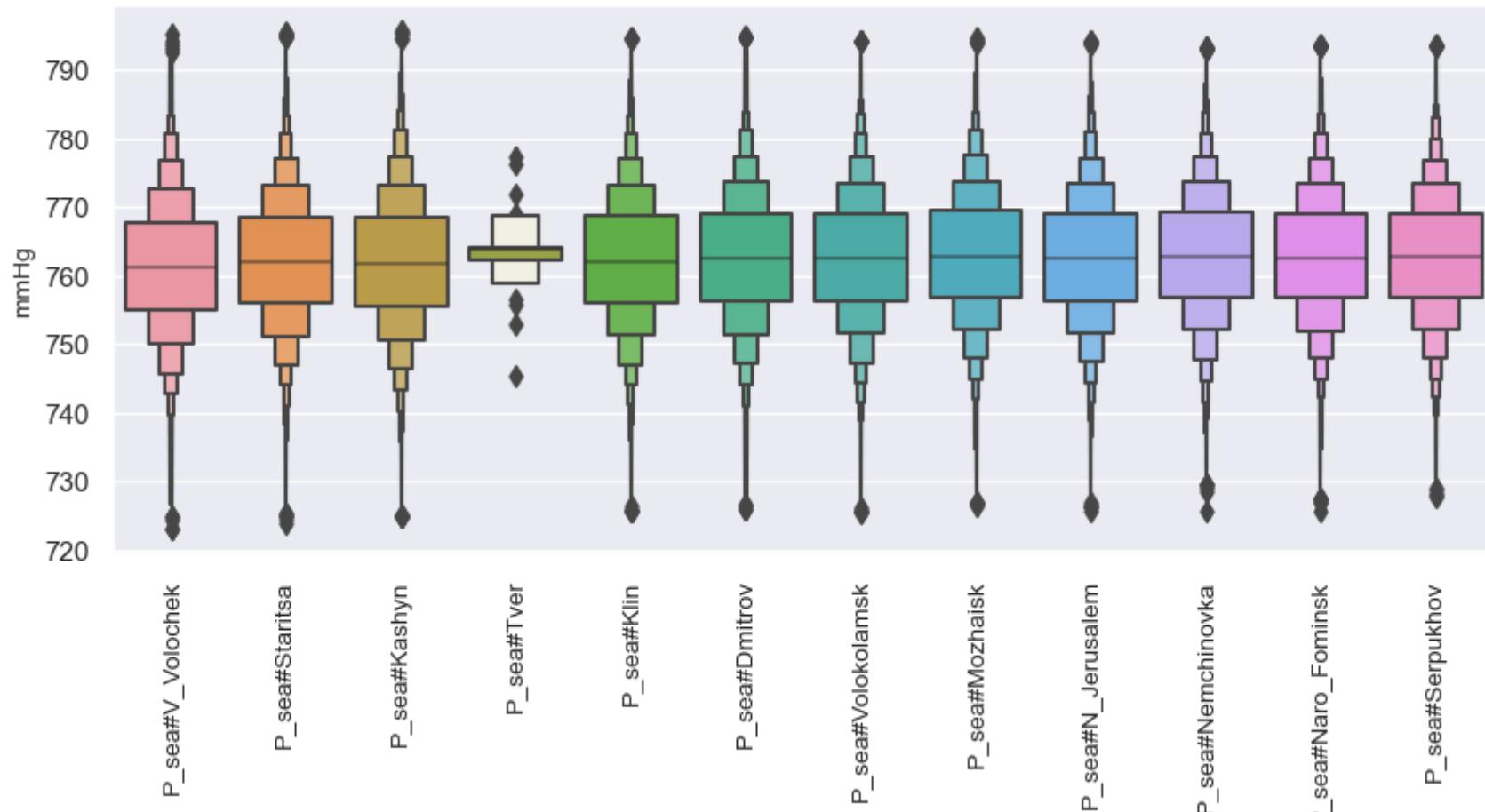
Распределение значений P\_sea в разрезе метеостанций после удаления ошибок:  
summer



Распределение значений P\_sea в разрезе метеостанций после удаления ошибок:  
autumn



Распределение значений P\_sea 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 [63]: print(f'Минимальное значение: {np.nanmin(df_tmp41)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp41)},\n'
      f'Средняя: {np.nanmean(df_tmp41)},\n'
      f'Медиана: {np.nanmedian(df_tmp41)}')
```

Минимальное значение: 723.0,  
Максимальное значение: 795.6,  
Средняя: 761.6664206885525,  
Медиана: 761.5

### Сохраним очищенные данные в файл параметров

```
In [64]: # # 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_'+PARAMETER41 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER41].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER41}.csv'  
)  
print('DONE!')  
  
df_P_sea.csv -> DONE!
```

Теперь распределение значений атмосферного давления на уровне моря находится в физически разумных пределах.

### 4.1.2. Восстановление "сплошных" NaN методом средней между соседними моментами наблюдения для показателя атмосферного давления P\_sea

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

```
In [65]: # Подсчитаем количество строк со "сплошными" NaN в df_tmp41  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp41.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp41.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 162

Очевидно такие строки нужно заполнить до пространственной экстраполяции. Представляется, что лучшим способом заполнения пропущенных строк будет средняя между двумя соседними моментами наблюдения

```
In [66]: # Создадим список DateTime индексов строк со сплошными NaN  
# Выбираем из датафрейма строки, где количество NaN равно количеству столбцов - то есть сплошные NaN  
list_time_total_nans = df_tmp41[df_tmp41.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp41.keys())].index.tolist()
```

```
In [67]: # По списку  
for idx in list_time_total_nans[:-1]: # кроме самого раннегоTimeStamp, для которого нельзя вычислить start_date  
    # определяем начало и конец временного интервала  
    start_time = idx - pd.Timedelta('3H')  
    end_time = idx + pd.Timedelta('3H')  
    while sum(df_tmp41.loc[start_time].notna()) == 0: # Пока ряд от start_time тоже состоит из сплошных NaN  
        start_time = start_time - pd.Timedelta('3H') # Уменьшаем start_time на 3 часа  
    while sum(df_tmp41.loc[end_time].notna()) == 0: # Пока ряд от end_time тоже состоит из сплошных NaN  
        end_time = end_time + pd.Timedelta('3H') # Увеличиваем end_time на 3 часа  
  
    # по ряду сплошных NaN заменяем NaN на средние значения их соседних двух рядов  
    for label in df_tmp41.loc[idx].index:  
        df_tmp41.at[idx, label] = np.mean([df_tmp41.at[start_time, label], df_tmp41.at[end_time, label]])
```

```
In [68]: # Подсчитаем количество строк со "сплошными" NaN в df_tmp1  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp41.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp41.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 1

Всё верно, это самая начальная строка. Она должна остаться

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

```
In [69]: # Проверим, все ли ошибки заменены на NaN  
# Количество нeNaN значений в df_tmp41 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]: # перечень 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_tmp41.at[error_coords[0], PARAMETER41+'#' + station]))
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna1}')

counter_notna2 = 0 # счётчик неNaN значений
for error in list_error_at: # по списку координат выявленных выше значений, аномально не совпадающих с расчётными
    counter_notna1 += np.sum(pd.notna(df_tmp41.at[error[0], PARAMETER41 + '#' + error[1]]))
print(f'По результатам удаления выявленных несоответствий архивных и расчётных величин осталось значений: '
      f'{counter_notna2}')
# df_tmp41

```

По результатам удаления выявленных аномальных выбросов осталось значений: 0

По результатам удаления выявленных несоответствий архивных и расчётных величин осталось значений: 0

Получившееся значение показывает количество исправленных ошибочных значений, там где был применён метод восстановления сплошных NaN. В данном случае, исправлений индивидуальных ошибочных значений при заполнении сплошных NaN не произошло.

### 4.1.3. Подбор модели для восстановления пропущенных и удалённых значений показателя атмосферного давления на уровне станции, а также для моделирования значений для искомой точки.

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

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

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

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

Ожидаемые выходные данные:

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

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

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

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

Создадим набор данных для обучения и валидации модели IDW.

1. Используем данные в `df_tmp41`.
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 [70]: # Создаём датафрейм для валидации модели IDW
df_test = df_tmp41.dropna(how='any')
```

```
In [71]: df_test.info()
df_test.shape

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 408 entries, 2022-06-06 09:00:00 to 2007-12-05 21:00:00
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   P_sea#V_Volochek    408 non-null   float64
 1   P_sea#Staritsa      408 non-null   float64
 2   P_sea#Kashyn        408 non-null   float64
 3   P_sea#Tver          408 non-null   float64
 4   P_sea#Klin          408 non-null   float64
 5   P_sea#Dmitrov       408 non-null   float64
 6   P_sea#Volokolamsk   408 non-null   float64
 7   P_sea#Mozhaisk      408 non-null   float64
 8   P_sea#N_Jerusalem    408 non-null   float64
 9   P_sea#Nemchinovka   408 non-null   float64
 10  P_sea#Naro_Fominsk  408 non-null   float64
 11  P_sea#Serpukhov     408 non-null   float64
dtypes: float64(12)
memory usage: 41.4 KB
```

```
Out[71]: (408, 12)
```

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

```
In [72]: # 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[72]: 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])
```

```
Out[72]: 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])
```

```
Out[72]: array([[ 0,  5],
 [ 1,  4],
 [ 2,  0],
 [ 3,  2],
 [ 4, 11],
 [ 5, 10],
 [ 6,  9],
 [ 7, 11],
 [ 8,  7],
 [ 9,  6],
 [ 10,  4],
 [ 11,  9],
 [ 12, 10],
 [ 13,  7],
 [ 14,  1],
 [ 15,  8],
 [ 16,  2],
 [ 17, 11],
 [ 18, 11],
 [ 19,  0],
 [ 20,  5],
 [ 21,  6],
 [ 22,  1],
 [ 23,  9],
 [ 24, 10],
 [ 25,  5],
 [ 26,  5],
 [ 27,  2],
 [ 28,  9],
 [ 29,  3],
 [ 30,  5],
 [ 31,  9],
 [ 32,  2],
 [ 33, 10],
 [ 34,  1],
 [ 35,  0],
 [ 36,  4],
 [ 37,  6],
 [ 38,  2],
 [ 39,  0],
 [ 40,  8],
 [ 41,  6],
 [ 42,  4],
 [ 43,  0],
```

```
[ 44,   1],  
[ 45,   11],  
[ 46,   1],  
[ 47,   3],  
[ 48,   9],  
[ 49,   6],  
[ 50,  10],  
[ 51,   2],  
[ 52,   1],  
[ 53,   3],  
[ 54,   8],  
[ 55,   8],  
[ 56,   3],  
[ 57,   2],  
[ 58,   0],  
[ 59,   3],  
[ 60,   0],  
[ 61,   5],  
[ 62,   8],  
[ 63,   0],  
[ 64,   8],  
[ 65,   6],  
[ 66,  10],  
[ 67,   0],  
[ 68,   9],  
[ 69,   4],  
[ 70,  11],  
[ 71,  10],  
[ 72,   6],  
[ 73,   0],  
[ 74,  10],  
[ 75,   2],  
[ 76,   3],  
[ 77,   7],  
[ 78,   1],  
[ 79,  11],  
[ 80,  11],  
[ 81,   6],  
[ 82,   5],  
[ 83,   0],  
[ 84,  11],  
[ 85,   6],  
[ 86,   8],  
[ 87,   2],
```

```
[ 88,   8],  
[ 89,   7],  
[ 90,   6],  
[ 91,   0],  
[ 92,   3],  
[ 93,   4],  
[ 94,   2],  
[ 95,   7],  
[ 96,   2],  
[ 97,   4],  
[ 98,   5],  
[ 99,   0],  
[100,   7],  
[101,  10],  
[102,   1],  
[103,   1],  
[104,   8],  
[105,   6],  
[106,   7],  
[107,   4],  
[108,   1],  
[109,   7],  
[110,   8],  
[111,   4],  
[112,  11],  
[113,   3],  
[114,   6],  
[115,   2],  
[116,  10],  
[117,   5],  
[118,   0],  
[119,   8],  
[120,   0],  
[121,   5],  
[122,   3],  
[123,   7],  
[124,   8],  
[125,   7],  
[126,   0],  
[127,  10],  
[128,  10],  
[129,   9],  
[130,  11],  
[131,   1],
```

```
[132,  3],  
[133,  5],  
[134,  3],  
[135,  2],  
[136,  5],  
[137,  1],  
[138,  5],  
[139,  1],  
[140, 11],  
[141, 10],  
[142,  4],  
[143,  3],  
[144,  9],  
[145,  7],  
[146,  6],  
[147,  7],  
[148,  7],  
[149,  0],  
[150,  4],  
[151,  8],  
[152,  9],  
[153,  0],  
[154,  1],  
[155, 11],  
[156,  1],  
[157,  3],  
[158,  6],  
[159,  7],  
[160,  7],  
[161,  5],  
[162,  2],  
[163,  6],  
[164,  0],  
[165,  7],  
[166, 11],  
[167,  7],  
[168,  9],  
[169,  8],  
[170,  3],  
[171,  3],  
[172,  8],  
[173,  8],  
[174,  8],  
[175,  9],
```

```
[176,  11],  
[177,   4],  
[178,  11],  
[179,   5],  
[180,  10],  
[181,   1],  
[182,   5],  
[183,  10],  
[184,   8],  
[185,  10],  
[186,   2],  
[187,   0],  
[188,   8],  
[189,   7],  
[190,   1],  
[191,   7],  
[192,   6],  
[193,   6],  
[194,   9],  
[195,   3],  
[196,   2],  
[197,   5],  
[198,   5],  
[199,   4],  
[200,   9],  
[201,   3],  
[202,   2],  
[203,   2],  
[204,   9],  
[205,  10],  
[206,   3],  
[207,   0],  
[208,   4],  
[209,  11],  
[210,   5],  
[211,   8],  
[212,   3],  
[213,   5],  
[214,   0],  
[215,   0],  
[216,   9],  
[217,  11],  
[218,   0],  
[219,   8],
```

```
[220, 11],  
[221, 9],  
[222, 5],  
[223, 0],  
[224, 5],  
[225, 3],  
[226, 5],  
[227, 6],  
[228, 2],  
[229, 4],  
[230, 0],  
[231, 8],  
[232, 8],  
[233, 6],  
[234, 7],  
[235, 0],  
[236, 1],  
[237, 1],  
[238, 5],  
[239, 6],  
[240, 4],  
[241, 4],  
[242, 9],  
[243, 8],  
[244, 5],  
[245, 1],  
[246, 1],  
[247, 9],  
[248, 10],  
[249, 6],  
[250, 9],  
[251, 0],  
[252, 6],  
[253, 3],  
[254, 11],  
[255, 11],  
[256, 11],  
[257, 9],  
[258, 9],  
[259, 7],  
[260, 3],  
[261, 6],  
[262, 0],  
[263, 3],
```

```
[264,  4],  
[265,  5],  
[266,  7],  
[267,  2],  
[268, 10],  
[269,  4],  
[270,  8],  
[271,  2],  
[272, 10],  
[273,  6],  
[274,  6],  
[275, 10],  
[276,  5],  
[277,  2],  
[278, 11],  
[279,  0],  
[280,  5],  
[281,  7],  
[282,  3],  
[283,  3],  
[284, 10],  
[285,  9],  
[286,  0],  
[287, 10],  
[288,  0],  
[289,  9],  
[290,  0],  
[291, 11],  
[292,  3],  
[293,  2],  
[294,  4],  
[295,  0],  
[296,  7],  
[297, 10],  
[298, 10],  
[299,  7],  
[300,  1],  
[301,  4],  
[302, 10],  
[303,  3],  
[304,  0],  
[305,  4],  
[306, 10],  
[307,  7],
```

```
[308, 11],  
[309, 6],  
[310, 8],  
[311, 2],  
[312, 0],  
[313, 8],  
[314, 3],  
[315, 1],  
[316, 9],  
[317, 5],  
[318, 0],  
[319, 4],  
[320, 9],  
[321, 7],  
[322, 0],  
[323, 6],  
[324, 11],  
[325, 1],  
[326, 7],  
[327, 4],  
[328, 5],  
[329, 6],  
[330, 7],  
[331, 8],  
[332, 4],  
[333, 4],  
[334, 3],  
[335, 4],  
[336, 6],  
[337, 0],  
[338, 5],  
[339, 5],  
[340, 6],  
[341, 9],  
[342, 9],  
[343, 8],  
[344, 7],  
[345, 1],  
[346, 8],  
[347, 10],  
[348, 10],  
[349, 9],  
[350, 8],  
[351, 7],
```

```
[352,  4],  
[353, 10],  
[354,  8],  
[355,  1],  
[356,  4],  
[357,  5],  
[358,  7],  
[359,  8],  
[360,  5],  
[361, 10],  
[362,  7],  
[363, 10],  
[364, 10],  
[365,  8],  
[366,  4],  
[367,  2],  
[368,  7],  
[369, 10],  
[370,  7],  
[371,  8],  
[372,  7],  
[373, 10],  
[374,  9],  
[375,  7],  
[376,  6],  
[377,  8],  
[378,  2],  
[379,  2],  
[380,  9],  
[381,  7],  
[382,  4],  
[383,  7],  
[384,  3],  
[385,  3],  
[386,  4],  
[387, 10],  
[388,  9],  
[389,  5],  
[390,  3],  
[391, 11],  
[392,  8],  
[393,  4],  
[394,  6],  
[395,  2],
```

```
[396, 8],  
[397, 6],  
[398, 1],  
[399, 9],  
[400, 4],  
[401, 11],  
[402, 7],  
[403, 0],  
[404, 1],  
[405, 3],  
[406, 8],  
[407, 7]])
```

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 [73]: # 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[73]: (244, 2)
```

```
Out[73]: (244,)
```

```
Out[73]: (164, 2)
```

```
Out[73]: (164,)
```

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

```
In [74]: start_time = time.time() # для замера времени выполнения кода

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

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

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

    # Создаём y_predict_idw_tr по индексам в массиве x_train
    # используем функцию inverse_distance_avg
    # Проходим по массиву и считываем из df_test данные по координатам, выводим результатом списком
    y_predict_idw_tr = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param_=PARAMETER41,
                             station_=df_test.keys()[y][len(PARAMETER41)+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 = 1.8646778  
Средняя абсолютная ошибка (MAE) IDW = 0.3549706  
Средний квадрат ошибки (MSE) IDW = 0.2360800  
Средняя квадратическая ошибка (RMSE) IDW = 0.4858807  
Коэффициент детерминации (R2) IDW = 0.9910243

Elapsed time=00:00:03

Текущие значения: iterations=2, power=3.5,  
Максимальная ошибка (MAX\_E) IDW = 1.8326701  
Средняя абсолютная ошибка (MAE) IDW = 0.3501268  
Средний квадрат ошибки (MSE) IDW = 0.2292618  
Средняя квадратическая ошибка (RMSE) IDW = 0.4788129  
Коэффициент детерминации (R2) IDW = 0.9912835

Elapsed time=00:00:04

Текущие значения: iterations=3, power=3.75,  
Максимальная ошибка (MAX\_E) IDW = 1.8024811  
Средняя абсолютная ошибка (MAE) IDW = 0.3463250  
Средний квадрат ошибки (MSE) IDW = 0.2234731  
Средняя квадратическая ошибка (RMSE) IDW = 0.4727294  
Коэффициент детерминации (R2) IDW = 0.9915036

Elapsed time=00:00:05

Текущие значения: iterations=4, power=4.0,  
Максимальная ошибка (MAX\_E) IDW = 1.7741210  
Средняя абсолютная ошибка (MAE) IDW = 0.3435924  
Средний квадрат ошибки (MSE) IDW = 0.2185877  
Средняя квадратическая ошибка (RMSE) IDW = 0.4675337  
Коэффициент детерминации (R2) IDW = 0.9916893

Elapsed time=00:00:06

Текущие значения: iterations=5, power=4.25,  
Максимальная ошибка (MAX\_E) IDW = 1.7475753  
Средняя абсолютная ошибка (MAE) IDW = 0.3414782  
Средний квадрат ошибки (MSE) IDW = 0.2144921  
Средняя квадратическая ошибка (RMSE) IDW = 0.4631330  
Коэффициент детерминации (R2) IDW = 0.9918451

Elapsed time=00:00:08

Текущие значения: iterations=6, power=4.5,

Максимальная ошибка (MAX\_E) IDW = 1.7228081  
Средняя абсолютная ошибка (MAE) IDW = 0.3398988  
Средний квадрат ошибки (MSE) IDW = 0.2110846  
Средняя квадратическая ошибка (RMSE) IDW = 0.4594394  
Коэффициент детерминации (R2) IDW = 0.9919746

Elapsed time=00:00:09  
Текущие значения: iterations=7, power=4.75,  
Максимальная ошибка (MAX\_E) IDW = 1.6997660  
Средняя абсолютная ошибка (MAE) IDW = 0.3386899  
Средний квадрат ошибки (MSE) IDW = 0.2082743  
Средняя квадратическая ошибка (RMSE) IDW = 0.4563708  
Коэффициент детерминации (R2) IDW = 0.9920815

Elapsed time=00:00:10  
Текущие значения: iterations=8, power=5.0,  
Максимальная ошибка (MAX\_E) IDW = 1.6783812  
Средняя абсолютная ошибка (MAE) IDW = 0.3377245  
Средний квадрат ошибки (MSE) IDW = 0.2059811  
Средняя квадратическая ошибка (RMSE) IDW = 0.4538514  
Коэффициент детерминации (R2) IDW = 0.9921686

Elapsed time=00:00:11  
Текущие значения: iterations=9, power=5.25,  
Максимальная ошибка (MAX\_E) IDW = 1.6585758  
Средняя абсолютная ошибка (MAE) IDW = 0.3369303  
Средний квадрат ошибки (MSE) IDW = 0.2041341  
Средняя квадратическая ошибка (RMSE) IDW = 0.4518120  
Коэффициент детерминации (R2) IDW = 0.9922389

Elapsed time=00:00:13  
Текущие значения: iterations=10, power=5.5,  
Максимальная ошибка (MAX\_E) IDW = 1.6402642  
Средняя абсолютная ошибка (MAE) IDW = 0.3363474  
Средний квадрат ошибки (MSE) IDW = 0.2026712  
Средняя квадратическая ошибка (RMSE) IDW = 0.4501902  
Коэффициент детерминации (R2) IDW = 0.9922945

Elapsed time=00:00:15  
Текущие значения: iterations=11, power=5.75,  
Максимальная ошибка (MAX\_E) IDW = 1.6233563  
Средняя абсолютная ошибка (MAE) IDW = 0.3359159  
Средний квадрат ошибки (MSE) IDW = 0.2015383  
Средняя квадратическая ошибка (RMSE) IDW = 0.4489301

Коэффициент детерминации (R2) IDW = 0.9923376

Elapsed time=00:00:17

Текущие значения: iterations=12, power=6.0,  
Максимальная ошибка (MAX\_E) IDW = 1.6077602  
Средняя абсолютная ошибка (MAE) IDW = 0.3355891  
Средний квадрат ошибки (MSE) IDW = 0.2006881  
Средняя квадратическая ошибка (RMSE) IDW = 0.4479823  
Коэффициент детерминации (R2) IDW = 0.9923699

Elapsed time=00:00:19

Текущие значения: iterations=13, power=6.25,  
Максимальная ошибка (MAX\_E) IDW = 1.5933835  
Средняя абсолютная ошибка (MAE) IDW = 0.3353510  
Средний квадрат ошибки (MSE) IDW = 0.2000798  
Средняя квадратическая ошибка (RMSE) IDW = 0.4473028  
Коэффициент детерминации (R2) IDW = 0.9923930

Elapsed time=00:00:20

Текущие значения: iterations=14, power=6.5,  
Максимальная ошибка (MAX\_E) IDW = 1.5801355  
Средняя абсолютная ошибка (MAE) IDW = 0.3352243  
Средний квадрат ошибки (MSE) IDW = 0.1996779  
Средняя квадратическая ошибка (RMSE) IDW = 0.4468533  
Коэффициент детерминации (R2) IDW = 0.9924083

Elapsed time=00:00:22

Текущие значения: iterations=15, power=6.75,  
Максимальная ошибка (MAX\_E) IDW = 1.5679284  
Средняя абсолютная ошибка (MAE) IDW = 0.3352657  
Средний квадрат ошибки (MSE) IDW = 0.1994515  
Средняя квадратическая ошибка (RMSE) IDW = 0.4465999  
Коэффициент детерминации (R2) IDW = 0.9924169

Elapsed time=00:00:24

Текущие значения: iterations=16, power=7.0,  
Максимальная ошибка (MAX\_E) IDW = 1.5566778  
Средняя абсолютная ошибка (MAE) IDW = 0.3354995  
Средний квадрат ошибки (MSE) IDW = 0.1993742  
Средняя квадратическая ошибка (RMSE) IDW = 0.4465134  
Коэффициент детерминации (R2) IDW = 0.9924198

Elapsed time=00:00:25

Текущие значения: iterations=17, power=7.25,

Максимальная ошибка (MAX\_E) IDW = 1.5463038  
Средняя абсолютная ошибка (MAE) IDW = 0.3357780  
Средний квадрат ошибки (MSE) IDW = 0.1994230  
Средняя квадратическая ошибка (RMSE) IDW = 0.4465681  
Коэффициент детерминации (R2) IDW = 0.9924180

ЛУЧШИЕ значения: power=7.0,  
Максимальная ошибка (MAX\_E) IDW = 1.5566778  
Средняя абсолютная ошибка (MAE) IDW = 0.3354995  
Средний квадрат ошибки (MSE) IDW = 0.1993742  
Средняя квадратическая ошибка (RMSE) IDW = 0.4465134  
Коэффициент детерминации (R2) IDW = 0.9924198

Несколько запусков кода выше, с различными параметрами степени (от 1 до 10), показали, что, судя по R2, лучшим значением степени на обучающем массиве является 8,0 Оно даёт лучшие метрики качества.

Применим эту степень для валидационного массива.

## Валидационный датасет

```
In [75]: # 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=PARAMETER41,
                          station=df_test.keys()[y][len(PARAMETER41)+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 = 1.1375727  
Средняя абсолютная ошибка (MAE) IDW = 0.3431941  
Средний квадрат ошибки (MSE) IDW = 0.1909757  
Средняя квадратическая ошибка (RMSE) IDW = 0.4370076  
Коэффициент детерминации (R2) IDW = 0.9927798

## ВЫВОД

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

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

Опробуем работу модели кригинга на уже сформированных тренинговой и валидационной выборках

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

```
In [76]: # 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[76]:

LoE      NaN

station	LoE	NaN
<b>V_Volochek</b>	34.566667	57.583333
<b>Staritsa</b>	34.933333	56.500000
<b>Kashyn</b>	37.583333	57.350000
<b>Tver</b>	35.922000	56.857300
<b>Klin</b>	36.716667	56.333333
<b>Dmitrov</b>	37.533333	56.366667
<b>Volokolamsk</b>	35.933333	56.016700
<b>Mozhaisk</b>	36.000000	55.516700
<b>N_Jerusalem</b>	36.816667	55.900000
<b>Nemchinovka</b>	37.350000	55.716667
<b>Naro_Fominsk</b>	36.700000	55.383333
<b>Serpukhov</b>	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 [77]: # Намеренно оставим закомментированные части кода, они могут использоваться для отладки
# start_time = time.time() # для замера времени выполнения кода
# counter = 0
y_predict_kriging_tr = [] # список предиктов для x_test
```

```

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

```

```

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

##
# V=V_West
##
```

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

```

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

```

```

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

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

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

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

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

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

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

```

```

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

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

```

```

In [78]: 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: 2  
Это составляет 0.8197% от обучающего массива данных

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

```
In [79]: # 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 [80]: 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 = 1.9027724, Kriging - IDW = 0.3564686  
Средняя абсолютная ошибка (MAE) Kriging = 0.3145193, Kriging - IDW = -0.0212587  
Средний квадрат ошибки (MSE) Kriging = 0.2028455, Kriging - IDW = 0.0034225  
Средняя квадратическая ошибка (RMSE) Kriging = 0.4503837, Kriging - IDW = 0.0038157  
Коэффициент детерминации (R2) Kriging = 0.9922980, Kriging - IDW = -0.0001200

## Валидационный датасет

Посмотрим, как модель будет отрабатывать на валидационной выборке.

```
In [81]: start_time = time.time() # для замера времени выполнения кода
# counter = 0
y_predict_kriging_vld = [] # список предиктов для x_test

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

```

#         break
#     else:
#         counter +=1
##  

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

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

```

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

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

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

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

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

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

y_predict_kriging_vld = np.array(y_predict_kriging_vld)

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

```

In [82]:

```

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

```

```

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

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

```

Elapsed time=00:00:03  
Значения метрик:  
R2=0.9925186, MAX\_E=1.5133890622685158, MAE=0.3274705, MSE=0.19788423855679174, RMSE=0.4448418

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ  
Количество значений, которые не удалось предсказать: 0  
Это составляет 0.0000% от валидационного массива данных

In [83]:

```

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 [84]:

```

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 = 1.5133891, Kriging - IDW = 0.3758164  
Средняя абсолютная ошибка (MAE) Kriging = 0.3274705, Kriging - IDW = -0.0157236  
Средний квадрат ошибки (MSE) Kriging = 0.1978842, Kriging - IDW = 0.0069086  
Средняя квадратическая ошибка (RMSE) Kriging = 0.4448418, Kriging - IDW = 0.0078342  
Коэффициент детерминации (R2) Kriging = 0.9925186, Kriging - IDW = -0.0002612

## ВЫВОД

В данном случае модель кригинга дала практически 100% результат, и не смогла предсказать только 1 значение на тренировочном 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.

### 4.1.3.3. 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*

В случае имеющихся обучающего и валидационного датасетов модель кригинга не смогла предсказать только одно значение на обучающем датасете и предсказала все значения на валидационном датасете. Проверять совместную работу двух моделей не имеет смысла.

```
In [85]: # # 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 срепень для IDW

```
In [86]: # 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_=PARAMETER41,
#                             station_=df_test.keys()[y][Len(PARAMETER41)+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}\n'
#           )

```

Опробуем новое значение лучшей степени для IDW на сокращённом валидационном датасете

In [87]:

```

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

# y_predict_idw_vld_shrunk = [
#     inverse_distance_avg(row_=df_test.iloc[x],
#                           param_=PARAMETER41,
#                           station_=df_test.keys()[y][Len(PARAMETER41)+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}\n'
#     )

```

## 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 [88]: # # Восстановим y_predict для лучшего R2 IDW
# y_predict_idw_tr_shrunk = [
#     inverse_distance_avg(row_=df_test.iloc[x],
#                           param_=PARAMETER41,
#                           station_=df_test.keys()[y][Len(PARAMETER41)+1:],
#                           df_dists_= df_station_dists,
#                           power_=best_power_idw_tr_shrunk)
#     for x, y in x_train_idw_shrunk
# ]

```

```
In [89]: # # последовательно соединим датасеты для кrigинга (там, где есть предсказания) и для IDW
# x_train_2 = np.append(x_train[~np.isnan(y_predict_kriging_tr)], x_train_idw_shrunk)
# y_train_2 = np.append(y_train_kriging_shrunk, y_train_idw_shrunk)
# y_predict_tr_2 = np.append(y_predict_kriging_tr_shrunk, y_predict_idw_tr_shrunk)
# x_test_2 = np.append(x_test[~np.isnan(y_predict_kriging_vld)], x_test_idw_shrunk)
# y_test_2 = np.append(y_test_kriging_shrunk, y_test_idw_shrunk)
# y_predict_vld_2 = np.append(y_predict_kriging_vld_shrunk, y_predict_idw_vld_shrunk)
```

```
In [90]: # # 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 [91]: # 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}'
```

## ВЫВОД

Стоит отметить, что модель IDW дала очень близкие результаты с моделью кригинга. Средняя абсолютная ошибка для кригинга выглядит немного лучшими, остальные метрики - немного хуже, чем у IDW (хотя речь идет о различиях в 3, 4, а иногда и в 5 знаке после запятой). Ниже используем совместно обе модели для предсказания отсутствующих значений.

### 4.1.4. Применение выбранных моделей для исправления, восстановления данных и получения предиктов показателя атмосферного давления на уровне метеостанции P\_sea для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

```
In [92]: # Добавим в df_tmp41 столбцы для будущих значений для Чашниково и центральной точки, заполним их NaN
# - это необходимо, чтобы вычислить значения для архивов
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER31#Chashnikovo и PARAMETER31#Rfrnce_point
df_tmp41 = (df_tmp41.
```

```

        assign(col_name1 = np.nan,
               col_name2 = np.nan).
        rename(columns={"col_name1": PARAMETER41+"#"+'Chashnikovo',
                       "col_name2": PARAMETER41+"#"+'Rfrnce_point'})
    )
df_tmp41.sample(3, random_state=56)

```

Out[92]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
<b>2014-02-22 03:00:00</b>	766.4	766.9	768.4	NaN	768.1	768.6	767.2	767.4	767.6
<b>2015-05-16 03:00:00</b>	749.2	748.5	745.7	NaN	745.8	745.0	746.8	747.0	747.1
<b>2020-06-18 18:00:00</b>	761.2	760.9	763.1	NaN	761.4	761.7	760.6	761.3	760.9

In [93]:

```

# Добавим в архив параметров P_sea столбец для Чашниково и будущей центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER41#Chashnikovo и PARAMETER41#Rfrnce_point
dict_df_parameters['df_'+PARAMETER41] = (dict_df_parameters['df_'+PARAMETER41].
                                           assign(col_name1 = np.nan,
                                                 col_name2 = np.nan).
                                           rename(columns={"col_name1": PARAMETER41+"#"+'Chashnikovo',
                                                          "col_name2": PARAMETER41+"#"+'Rfrnce_point'}))
dict_df_parameters['df_'+PARAMETER41].sample(3, random_state=56)

```

Out[93]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
<b>2014-02-22 03:00:00</b>	766.4	766.9	768.4	767.8	768.1	768.6	767.2	767.4	767.6
<b>2015-05-16 03:00:00</b>	749.2	748.5	745.7	747.1	745.8	745.0	746.8	747.0	749.1
<b>2020-06-18 18:00:00</b>	761.2	760.9	763.1	761.7	761.4	761.7	760.6	761.3	760.9



In [94]: # Добавим в архив метеостанций df Чашниково и df для центральной точки,  
# Создадим в нём столбец с назначением PARAMETER41

```
dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER41})
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER41})
                                         )

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

Out[94]:

	T	T_min	T_max	P_sea
<b>2014-02-22 03:00:00</b>	-4.156558	-4.156558	-2.109643	NaN
<b>2015-05-16 03:00:00</b>	9.181730	9.181730	10.434571	NaN
<b>2020-06-18 18:00:00</b>	27.439052	25.366130	30.241632	NaN

Out[94]:

	T	T_min	T_max	P_sea
2014-02-22 03:00:00	-3.589183	-3.794602	-2.700734	NaN
2015-05-16 03:00:00	7.789650	7.789650	8.797603	NaN
2020-06-18 18:00:00	29.404954	25.572651	29.941094	NaN

## Перенесение уже исправленных сплошных NaN в архивы

In [95]:

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER41] = df_tmp41.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[:, PARAMETER41] = df_tmp41.loc[:, PARAMETER41 + '#' + name_df[3:]]
```

In [96]:

```
# Подсчитаем количество строк со "сплошными" NaN dict_df_parameters['df_'+PARAMETER41]
# построчно подсчитаем суммуой количество NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(
    dict_df_parameters['df_'+PARAMETER41]
    .apply(lambda x: sum(x.isna()), axis=1)==len(dict_df_parameters['df_'+PARAMETER41].keys()))
)

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

Количество строк со сплошными NaN равно 1

In [97]:

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp41
# построчно подсчитаем суммуой количество NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(
    df_tmp41
    .apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp41.keys()))
)

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

Количество строк со сплошными NaN равно 1

## 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.

```
In [98]: start_time = time.time() # для замера времени выполнения кода

# Подсчитаем количество NaN в df_tmp41 и присвоим их переменной old_nan_count
# np.isnan(df_tmp41).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
new_nan_count = np.sum(np.isnan(df_tmp41).sum()) # результат всегда будет >= 0
# Определим начальное значение для переменной для обновления количества оставшихся NaN
old_nan_count = new_nan_count
counter = 0 # определим счётчик

# заменим значения в архивах, на исправленные и вычисленные из данных в df_tmp41
# используем функцию row_nan_kriging_correct
# функция настроена таким образом, что при возникновении ошибки, связной с недостаточностью данных,
# осуществляется выход из функции без возврата каких бы то ни было значений.
while (old_nan_count != new_nan_count) or (counter == 0):
    counter += 1
    print (f'\nИтерация № {counter}: осталось NaN: {new_nan_count}')

    # Построчно применяем функцию обработки NaN
    df_tmp41.apply(lambda x: row_nan_kriging_correct(row=x,
                                                       name_param_=PARAMETER41
                                                       ),
                  axis=1
                  )
    old_nan_count = new_nan_count # сохраняем прежнее количество NaN в old_nan_count
    new_nan_count = np.sum(np.isnan(df_tmp41).sum()) # подсчитаем оставшиеся количество NaN

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

Итерация № 1: осталось NaN: 208283

```
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 2: осталось NaN: 5261  
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 3: осталось NaN: 2105  
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 4: осталось NaN: 1681
```

```
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 5: осталось NaN: 1565  
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 6: осталось NaN: 1522  
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 7: осталось NaN: 1516
```

```
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 8: осталось NaN: 1505  
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 9: осталось NaN: 1503  
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 10: осталось NaN: 1502
```

```
Out[98]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Elapsed time=00:23:34
```

Подсчитаем конечное количество NaN

```
In [99]: # np.isnan(df_tmp41).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp41).sum())
```

```
Out[99]: 1502
```

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

```
In [100...]: # Произведём вычисление отсутствующих значений в df_tmp41 через модель IDW.  
# Заменим соответствующие значения в архивах на вновь вычисленные.  
# используем функцию row_nan_idw_correct  
  
start_time = time.time() # для замера времени выполнения кода  
  
# Построчно применяем функцию обработки NaN  
df_tmp41.apply(lambda x: row_nan_idw_correct(row_=x,  
                                              name_param_=PARAMETER41,  
                                              power_=best_power_idw_tr),  
               axis=1  
)  
  
chk_time = time.time()  
elapsed_time = chk_time - start_time
```

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

Out[100]:

```
2022-06-09 21:00:00      None
2022-06-09 18:00:00      None
2022-06-09 15:00:00      None
2022-06-09 12:00:00      None
2022-06-09 09:00:00      None
...
2005-02-01 12:00:00      None
2005-02-01 09:00:00      None
2005-02-01 06:00:00      None
2005-02-01 03:00:00      None
2005-02-01 00:00:00      None
Length: 50704, dtype: object
Elapsed time=00:00:16
```

Подсчитаем окончательное количество NaN

In [101...]

```
# np.isnan(df_tmp41).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
np.sum(np.isnan(df_tmp41).sum())
```

Out[101]:

```
14
```

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

Выведем, рандомные строки из df\_tmp41

In [102...]

```
df_tmp41.sample(7)
```

Out[102]:

	P_sea#V_Volochek	P_sea#Staritsa	P_sea#Kashyn	P_sea#Tver	P_sea#Klin	P_sea#Dmitrov	P_sea#Volokolamsk	P_sea#Mozhaisk	P_sea#N_Jerusalem
2018-07-30 03:00:00	769.6	768.8	769.8	768.916324	768.9	768.800000	768.100000	768.100000	768.100000
2015-10-09 03:00:00	763.1	763.2	760.2	762.293076	761.6	761.094656	762.408361	762.492198	762.492198
2021-07-23 18:00:00	755.8	756.1	754.0	755.388878	755.2	755.100000	755.400000	755.700000	755.700000
2020-02-23 18:00:00	742.4	744.6	743.5	744.367661	745.1	745.200000	745.700000	746.800000	746.800000
2019-11-20 09:00:00	777.9	778.0	780.6	778.493184	779.0	779.800000	778.200000	778.200000	778.200000
2015-08-31 18:00:00	762.4	762.3	760.9	761.942088	761.6	761.161375	761.825904	761.796121	761.796121
2020-07-03 06:00:00	757.0	758.2	757.9	757.969586	758.5	758.500000	758.300000	759.100000	759.100000

#### 4.1.5. Визуализация графиков давления P\_sea для условной метеостанции Чашниково

In [103...]

```
# Построим список годов для графиков
list_years = df_tmp41.index.year.unique().tolist()
```

In [104...]

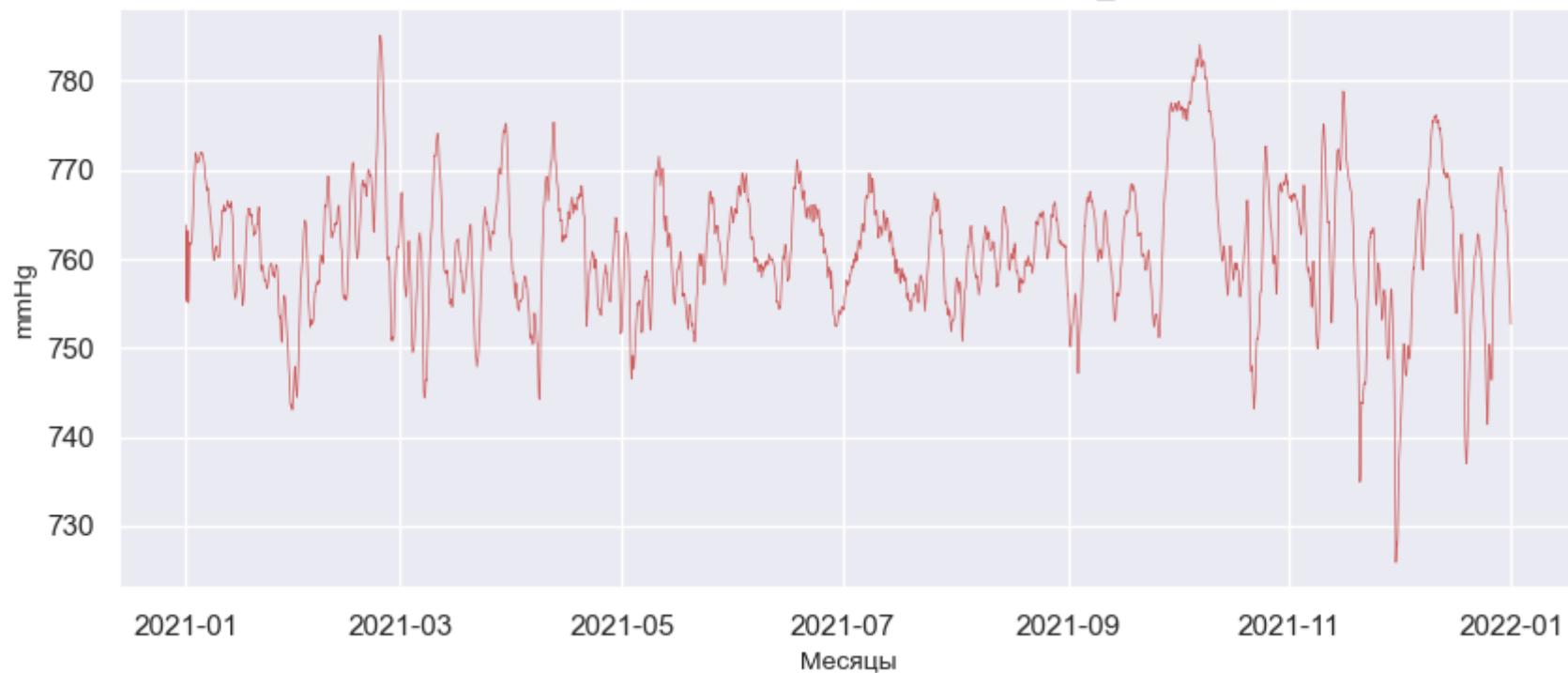
```
for year in list_years:
    data1 = dict_df_parameters['df_P_sea'][PARAMETER41+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_P_sea'].index.year == year)]\n\n    fig, ax = plt.subplots(figsize=(10, 4))\n    g1 = sns.lineplot(data=data1,
```

```
        color='indianred',
        linewidth=0.5,
        ax=ax)

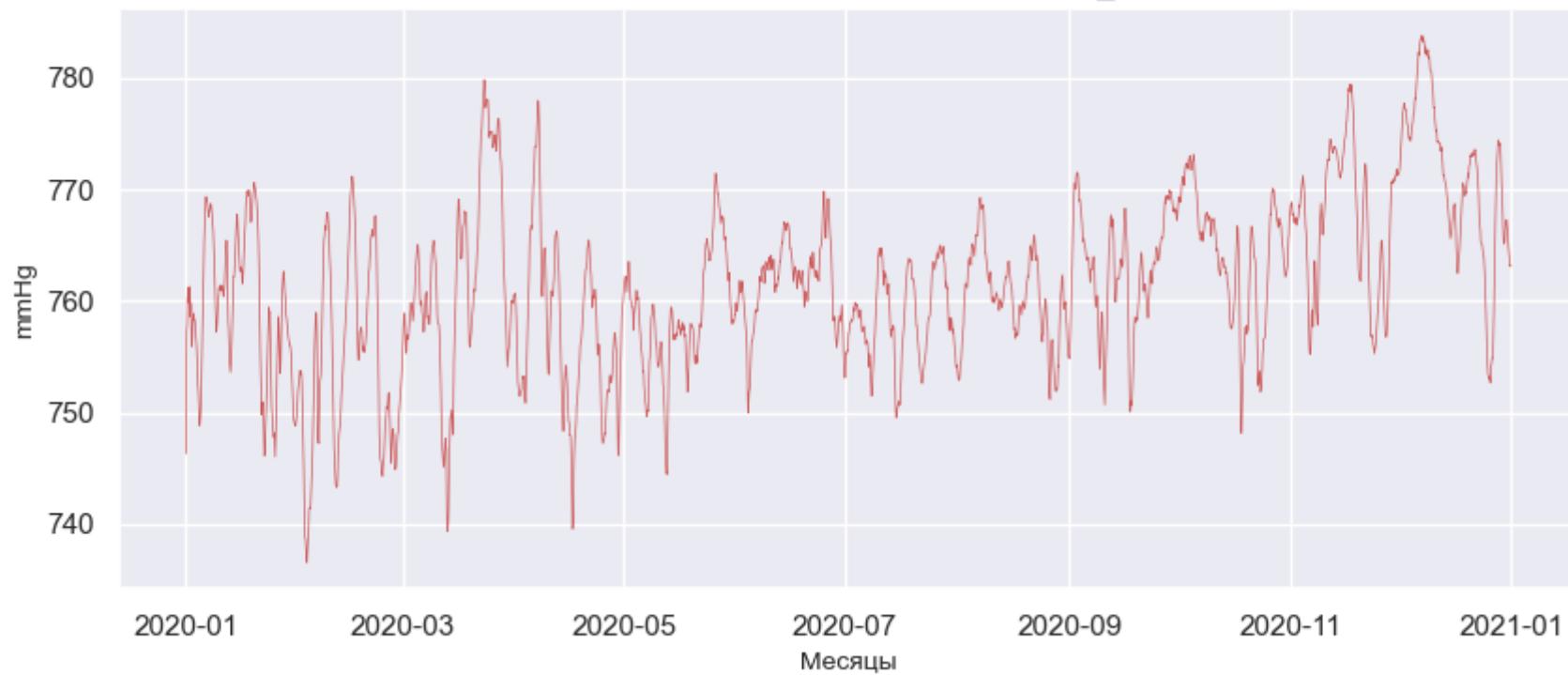
dummy = ax.set_ylabel('mmHg', size=10)
dummy = ax.set_xlabel('Месяцы', size=10)
dummy = plt.title(f'Чашниково: Ежедневная динамика параметра {PARAMETER41}, {year} год')
plt.show()
```



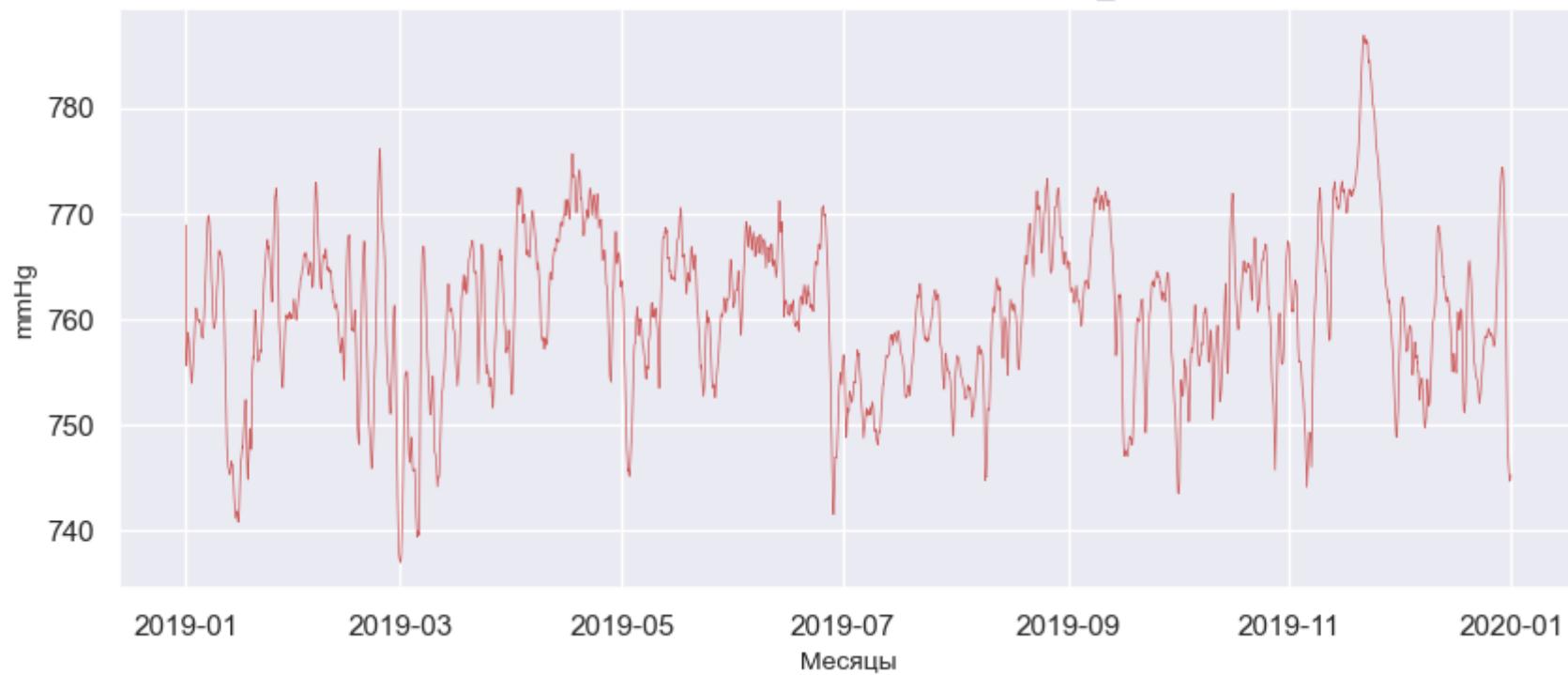
Чашниково: Ежедневная динамика параметра P\_sea, 2021 год



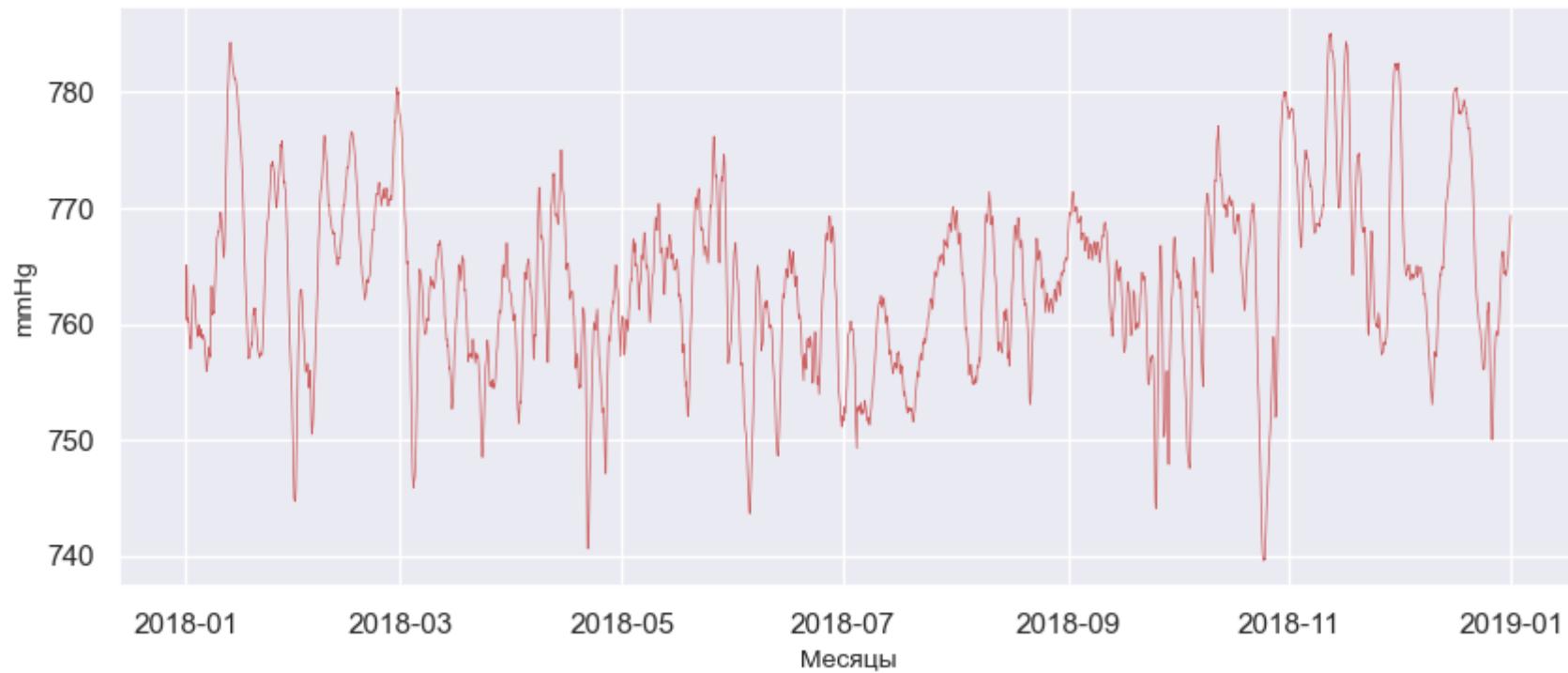
Чашниково: Ежедневная динамика параметра P\_sea, 2020 год



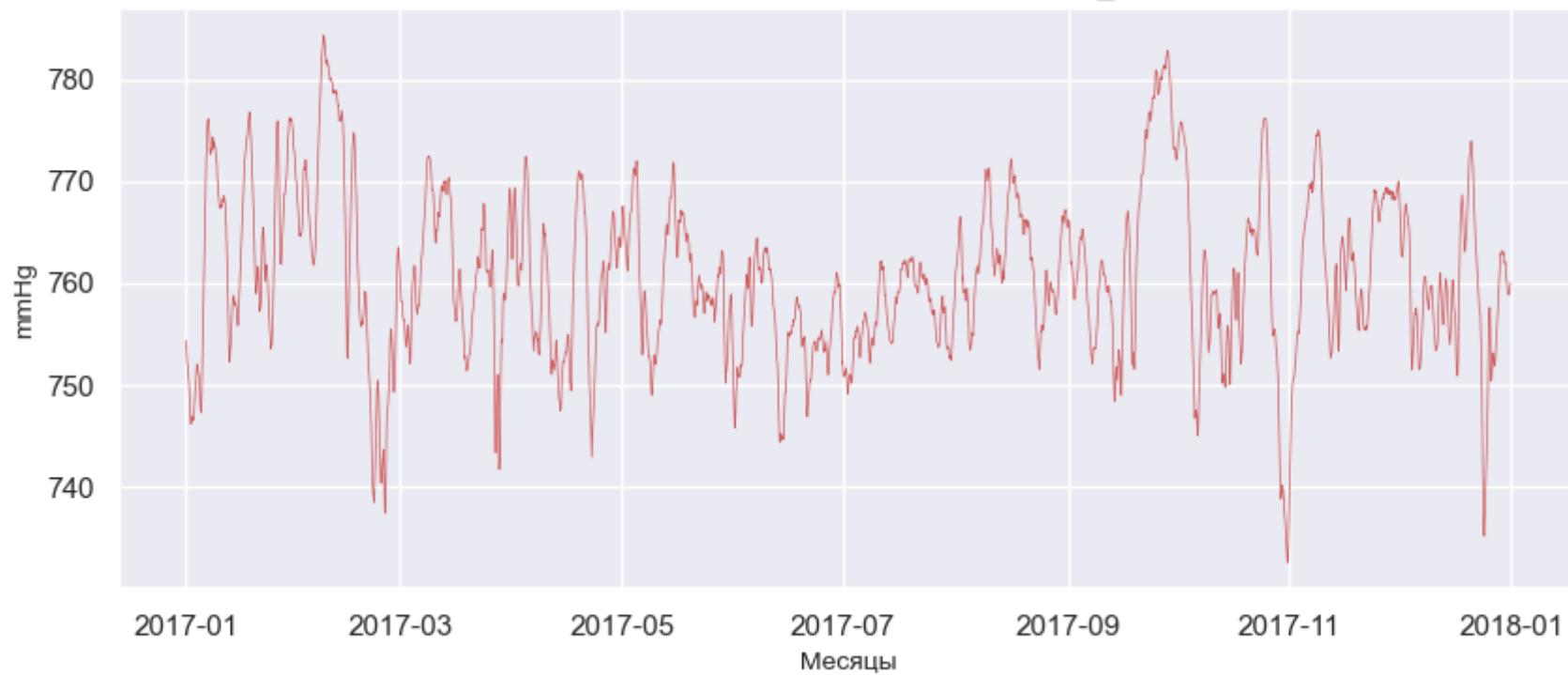
Чашниково: Ежедневная динамика параметра P\_sea, 2019 год



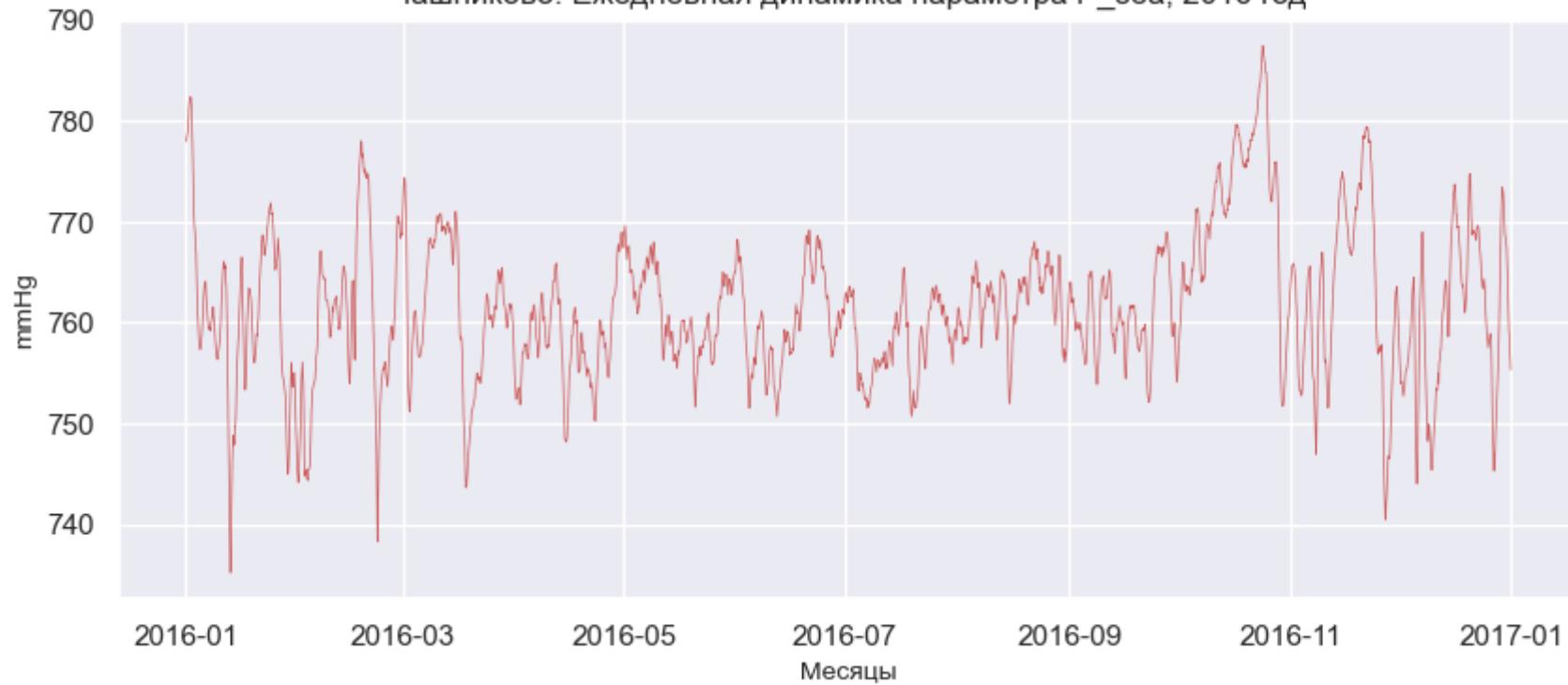
Чашниково: Ежедневная динамика параметра P\_sea, 2018 год



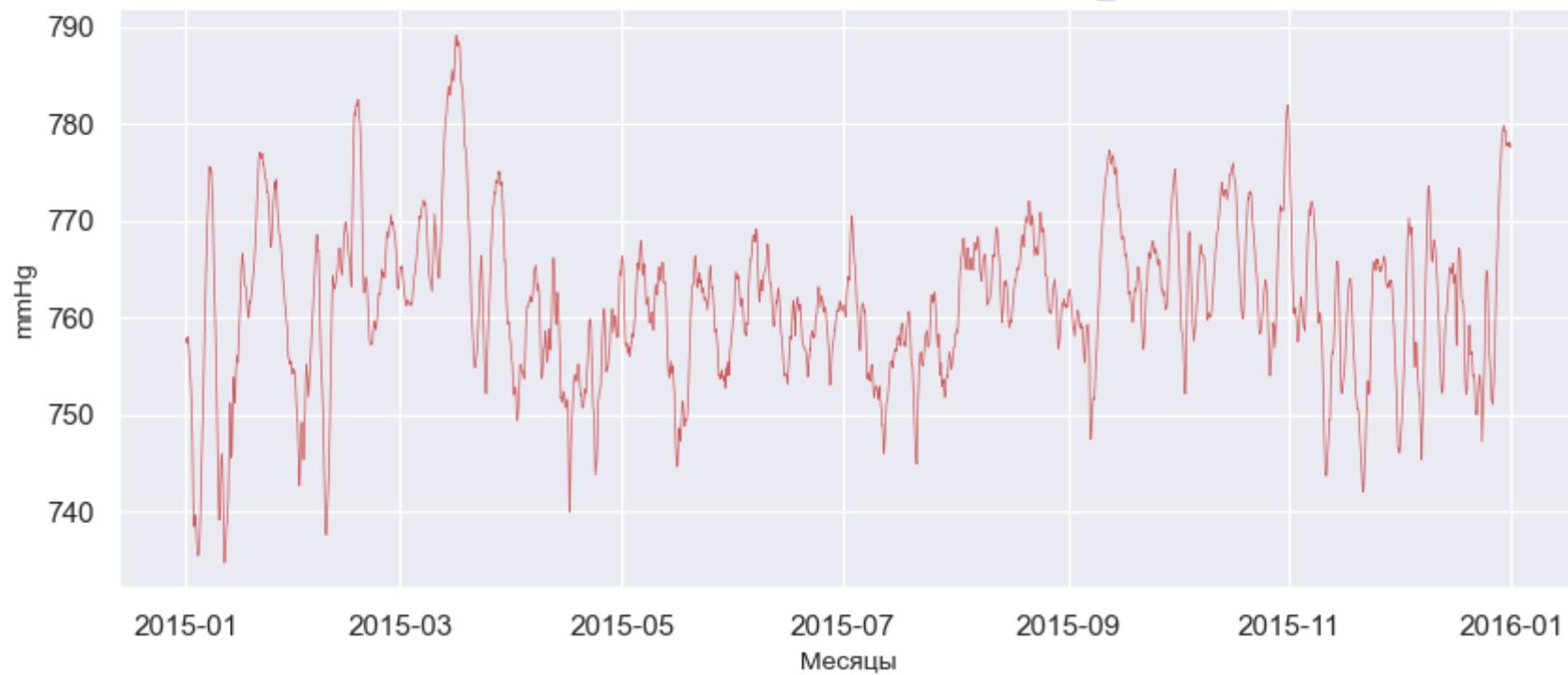
Чашниково: Ежедневная динамика параметра P\_sea, 2017 год



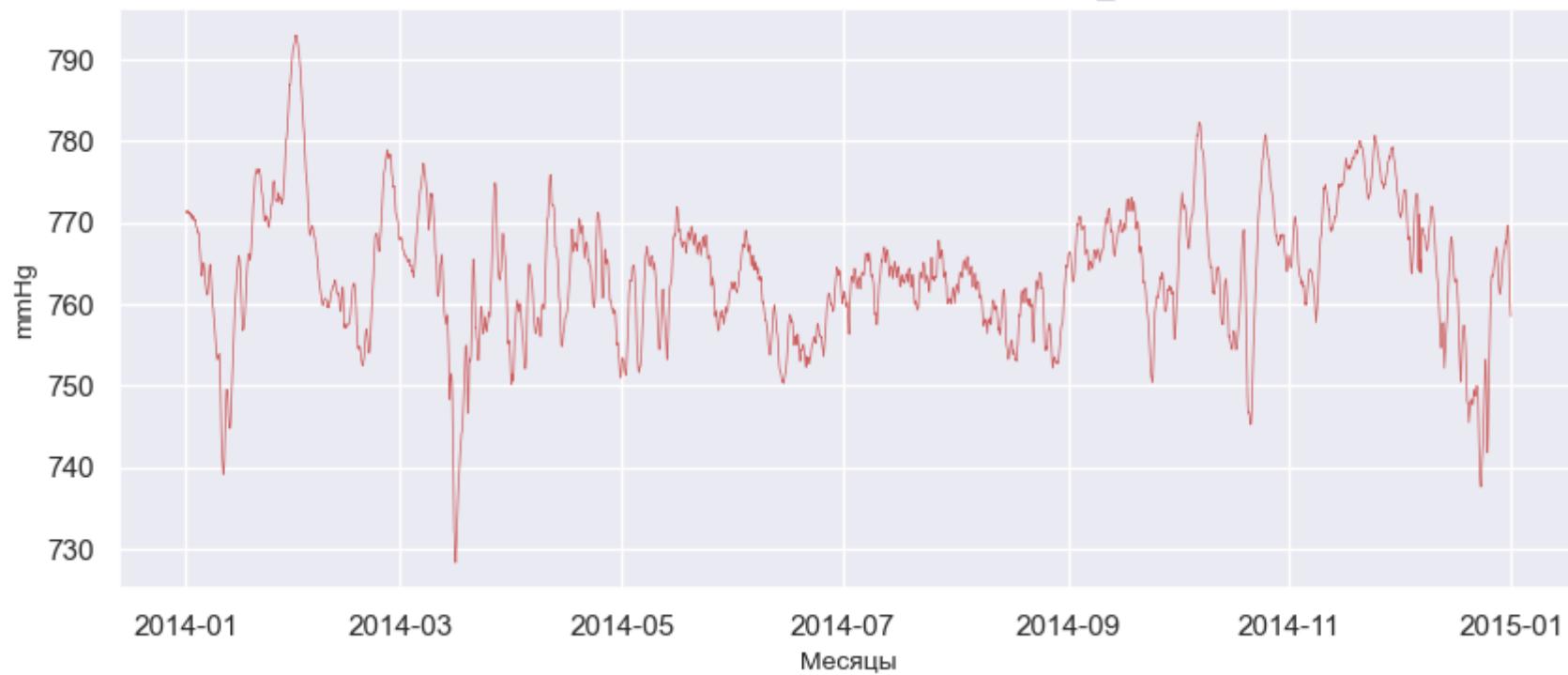
Чашниково: Ежедневная динамика параметра P\_sea, 2016 год



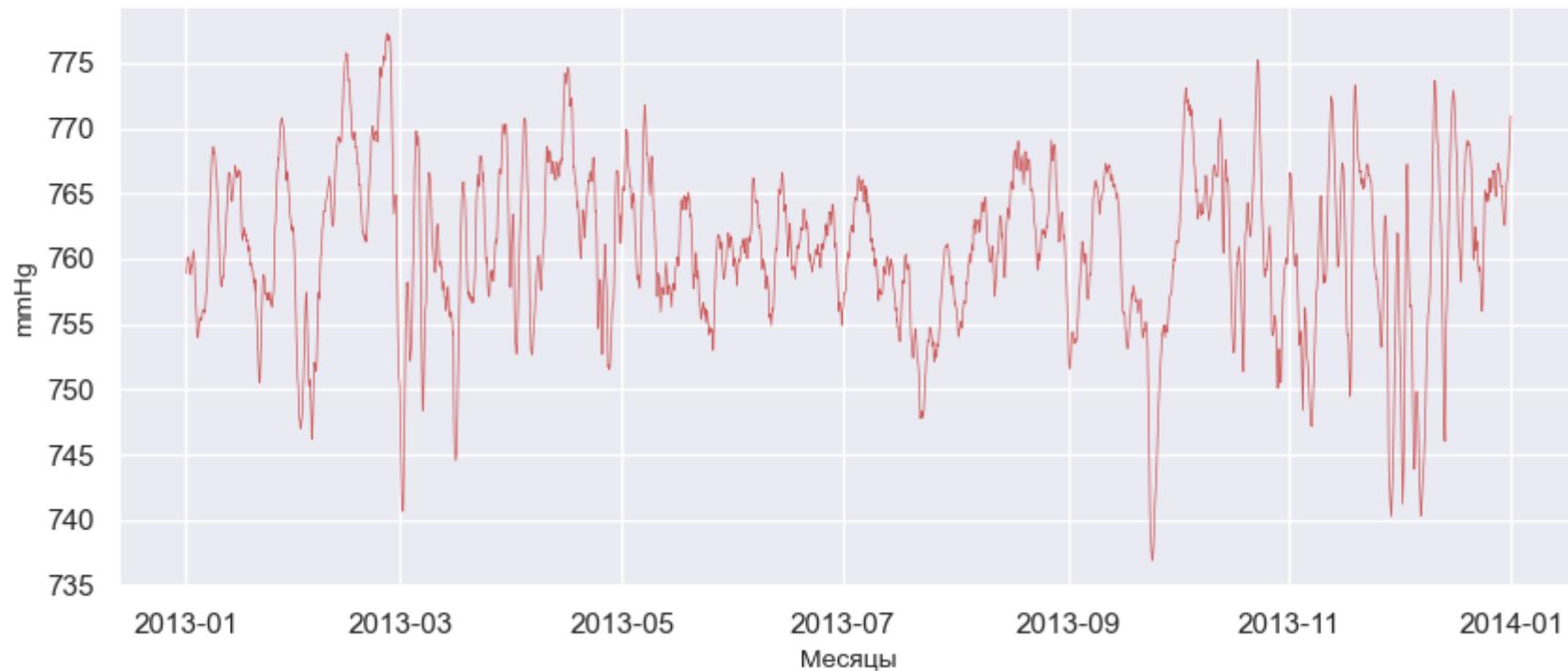
### Чашниково: Ежедневная динамика параметра P\_sea, 2015 год



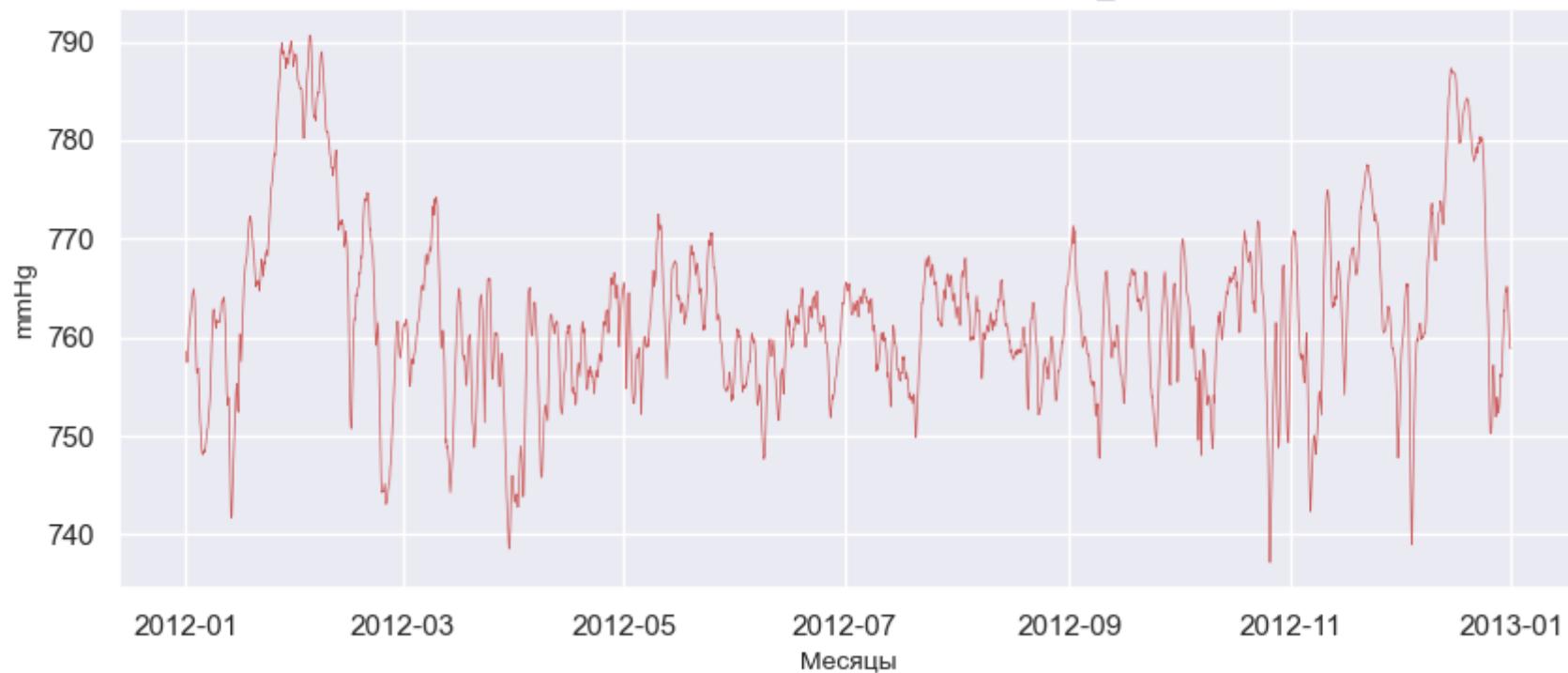
Чашниково: Ежедневная динамика параметра P\_sea, 2014 год



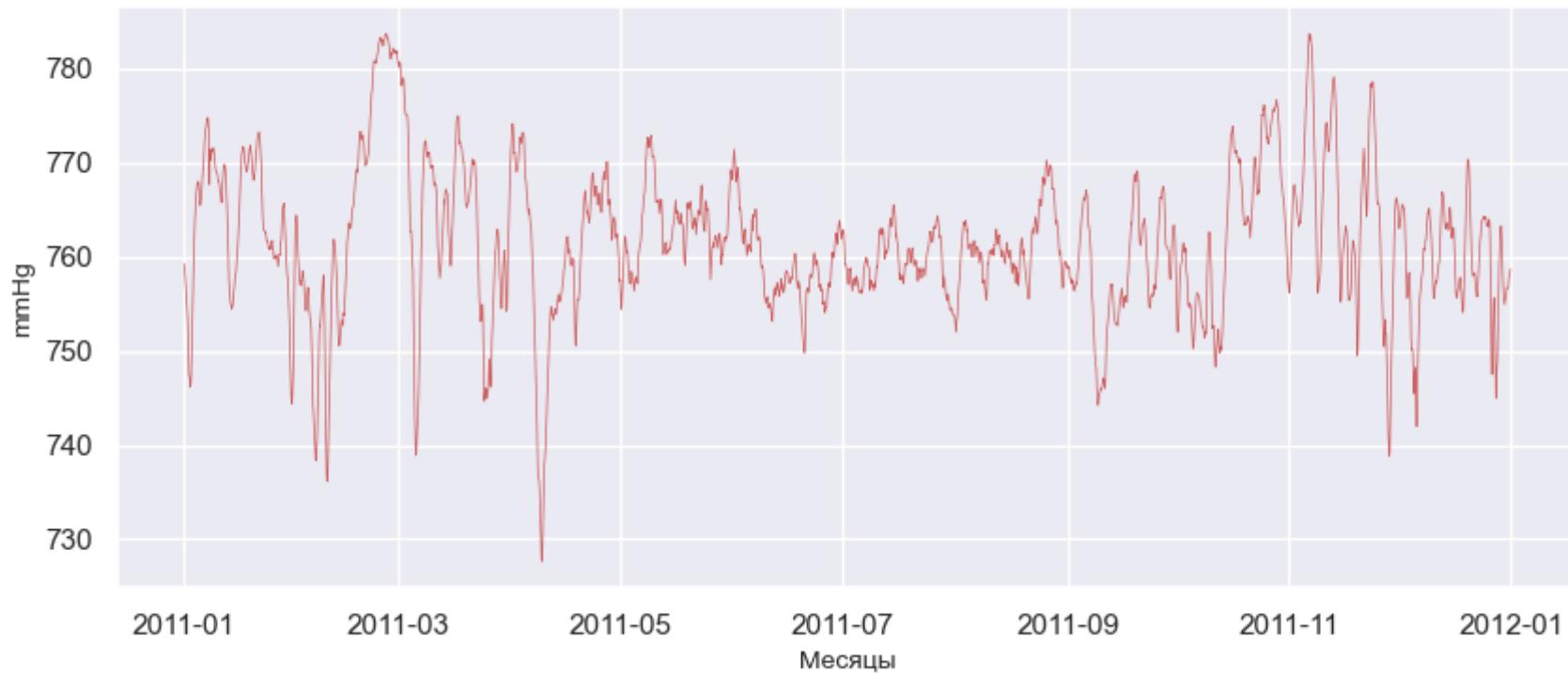
### Чашниково: Ежедневная динамика параметра P\_sea, 2013 год



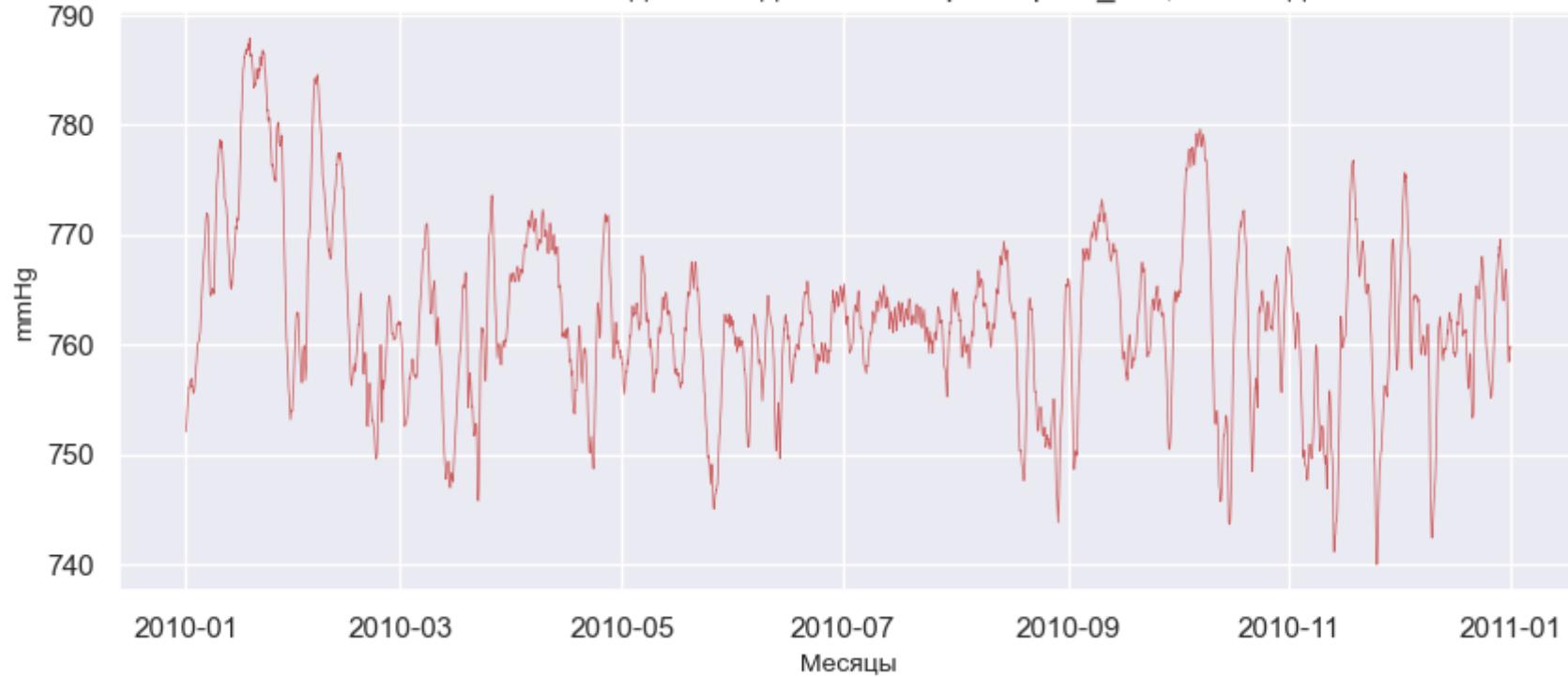
Чашниково: Ежедневная динамика параметра P\_sea, 2012 год



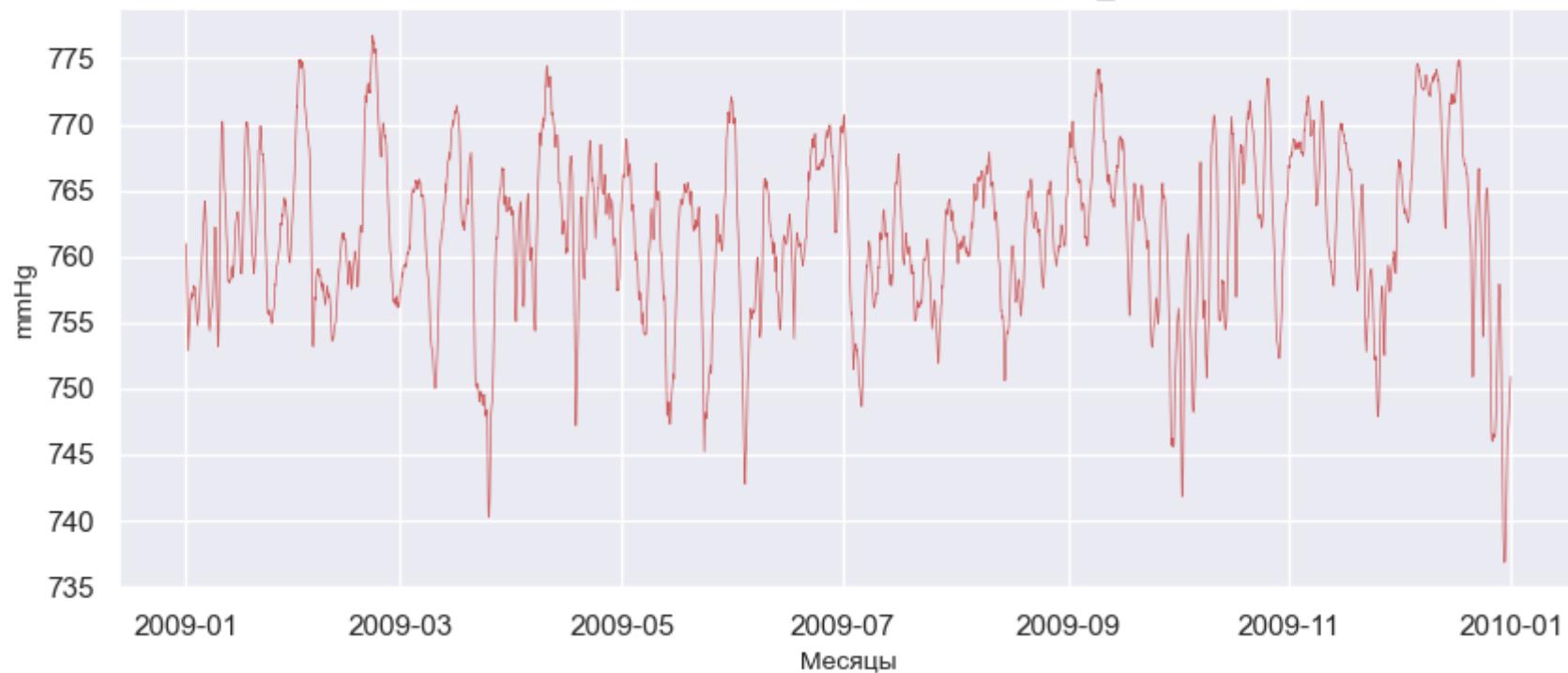
Чашниково: Ежедневная динамика параметра P\_sea, 2011 год



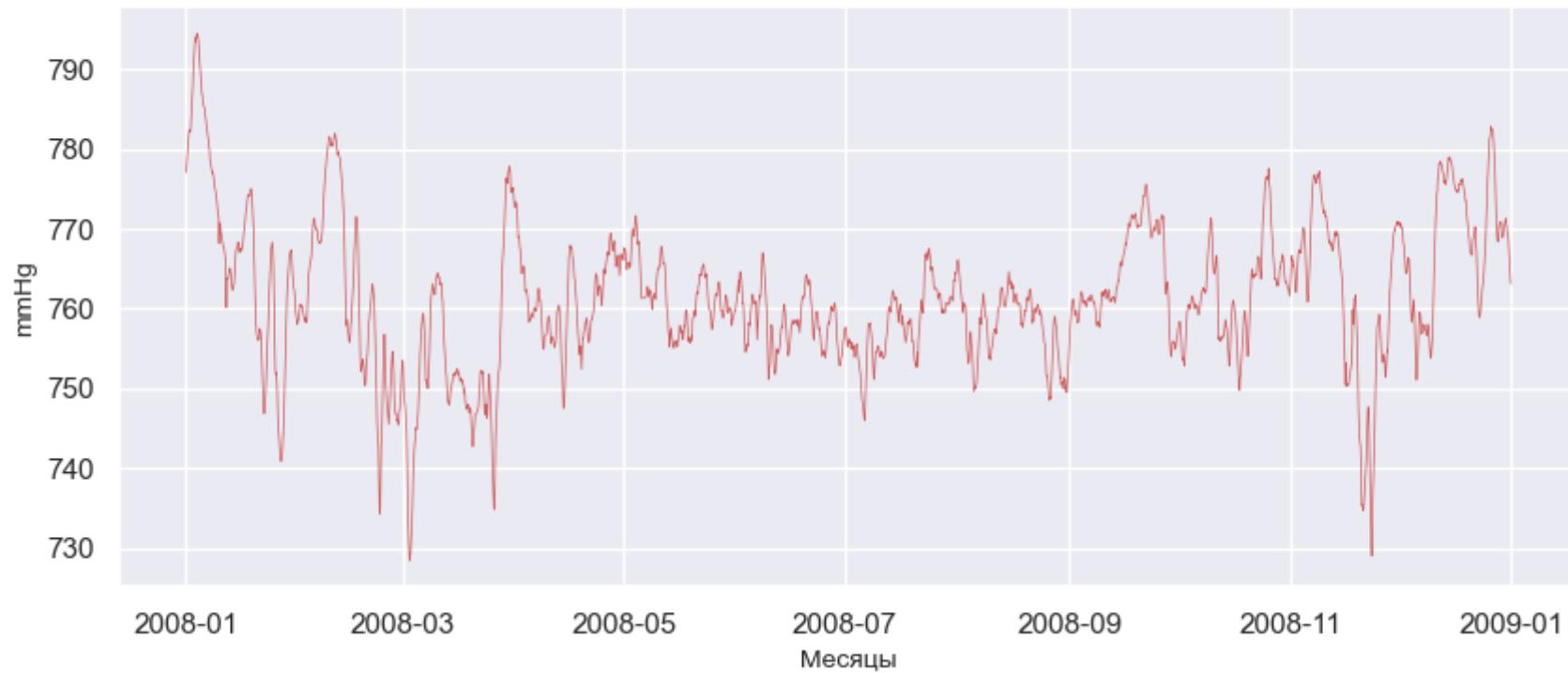
Чашниково: Ежедневная динамика параметра P\_sea, 2010 год



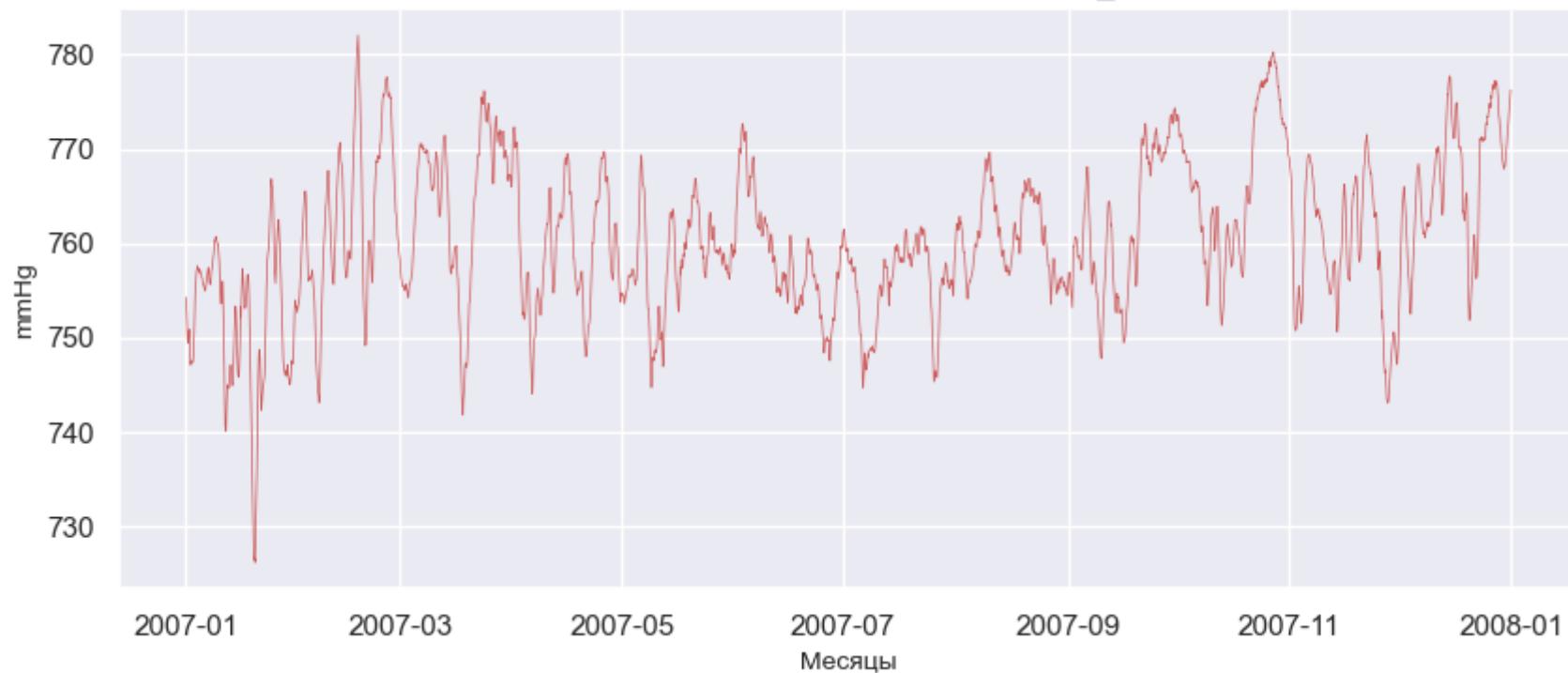
Чашниково: Ежедневная динамика параметра P\_sea, 2009 год



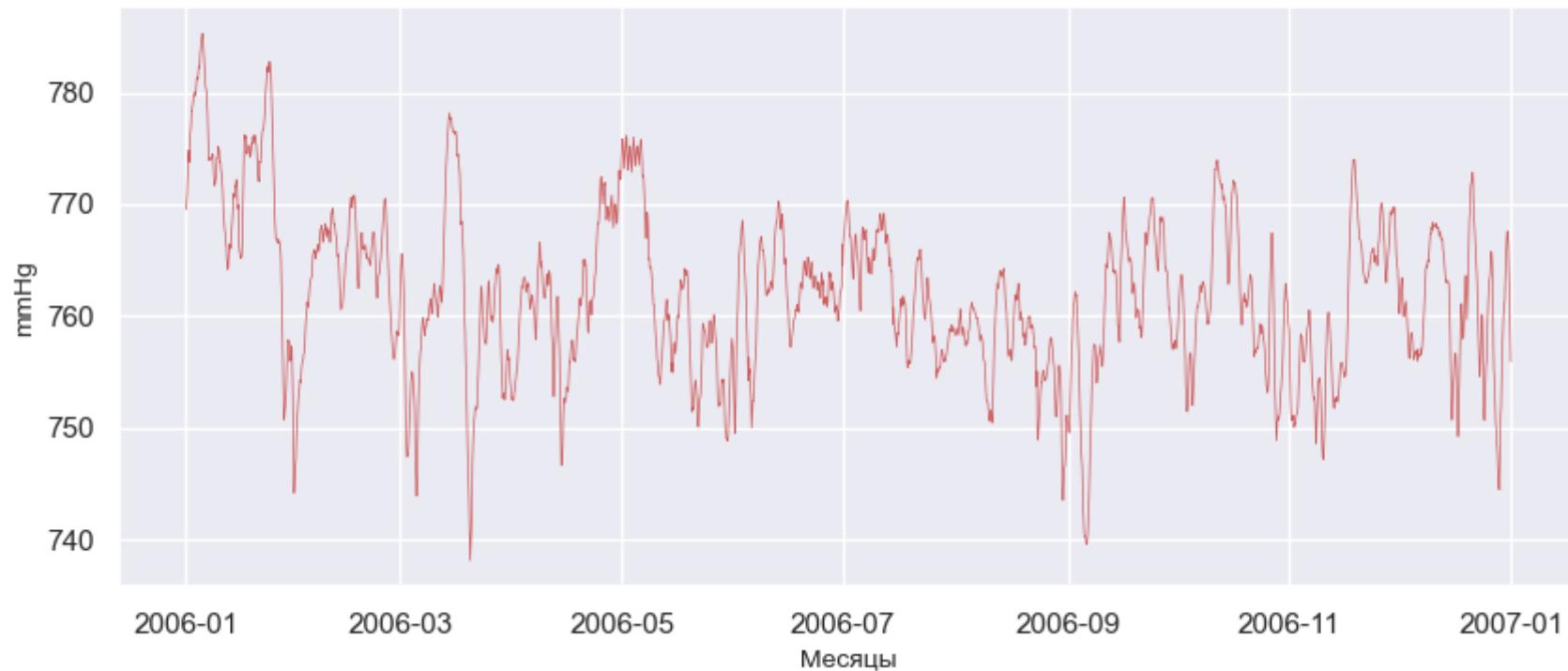
Чашниково: Ежедневная динамика параметра P\_sea, 2008 год



Чашниково: Ежедневная динамика параметра P\_sea, 2007 год



Чашниково: Ежедневная динамика параметра P\_sea, 2006 год



### Чашниково: Ежедневная динамика параметра P\_sea, 2005 год



#### 4.1.6. Сохранение полученных данных в файлы

In [105...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER41}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER41}'  
  
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_'+PARAMETER41 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER41].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER41}.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\_P\_sea.csv -> DONE!

## 4.2. Атмосферное давление: P\_station (атмосферное давление на уровне метеостанции)

Показатель атмосферного давления на уровне метеостанции является основанием для расчёта значения атмосферного давления на уровне моря. Для сравнения давления в различных точках показатель давления на уровне данной точки использовать нельзя, так как он зависит еще и от высоты над уровнем моря. Поэтому и некорректные значения, и выбросы показателя давления на уровне станции корректно находить через давление на уровне моря.

### 4.2.1. Поиск и удаление ошибок показателя давления P\_station

Для данного раздела обозначим константу названия параметра

In [106...]

```
PARAMETER42 = 'P_station'
```

### Создаём временный DF для работы с параметром P\_station

In [107...]

```
param_df_name = f'df_{PARAMETER42}' # преобразуем полученное значение в df_PARAMETER42 - ключ словаря dict_df_parameters
# Создадим временный df
df_tmp42 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp42.sample(5, random_state=56)
```

Out[107]:

	P_station#V_Volochek	P_station#Staritsa	P_station#Kashyn	P_station#Tver	P_station#Klin	P_station#Dmitrov	P_station#Volokolamsk	P_station
2014-02-22 03:00:00	750.2	749.1	755.0	753.8	752.0	751.3		748.4
2015-05-16 03:00:00	734.2	731.8	733.4	734.1	731.0	729.2		729.3
2020-06-18 18:00:00	746.9	745.2	751.1	749.3	747.3	746.5		743.8
2019-12-02 21:00:00	740.1	739.9	744.1	743.6	742.4	741.5		739.7
2008-06-05 18:00:00	NaN	745.4	NaN	747.7	746.4	744.9		744.2

Определим последовательность действий, для поиска ошибок показателя Атмосферное давление на уровне станции P\_station.

1. Проверяем соответствие давления на уровне метеостанции давлению на уровне моря. Архив давления на уровне моря у нас уже исправлен и не должен содержать ошибок и NaN.
2. Определяем несоответствия, как ошибки и удаляем их.

**Визуализируем архив давления на уровне станции (P\_station) по сезонам**

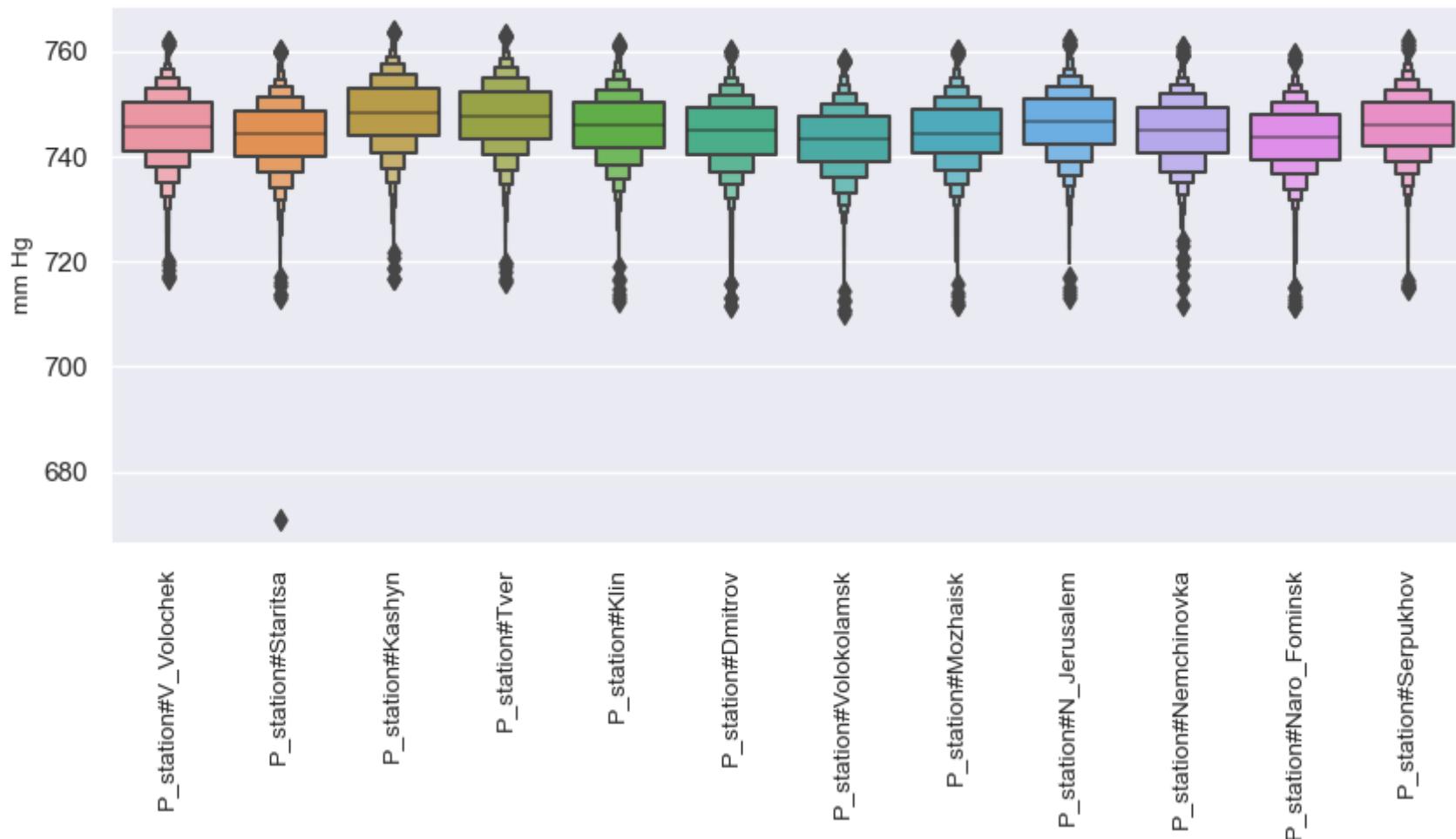
Исходя из данных о климате Московской области, временные границы сезонов определены следующим образом.

- 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

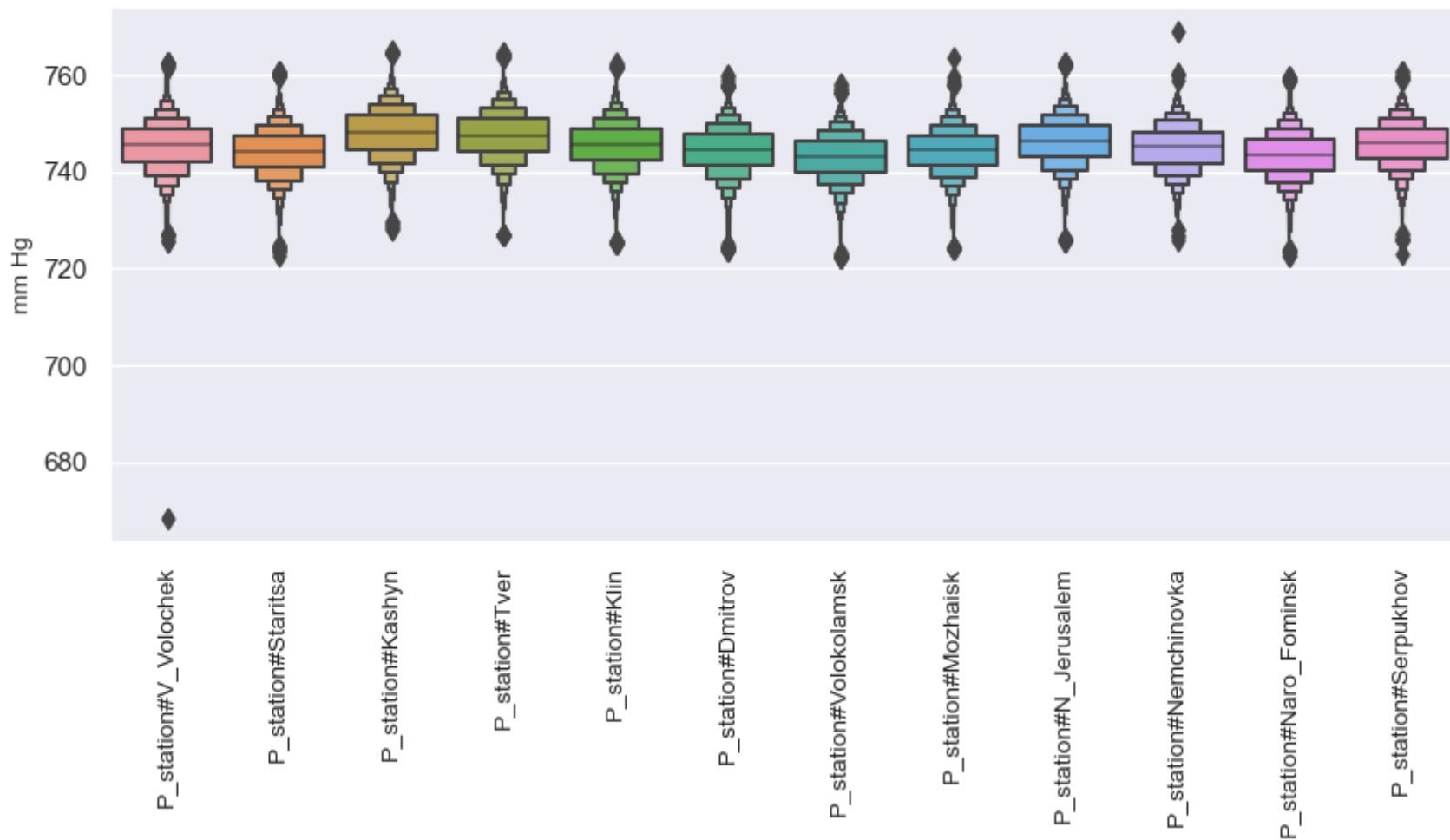
In [108...]

```
# 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_tmp42).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp42[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('мм Hg', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER42} в разрезе метеостанций:\n'
                        f'{season_name}')
plt.show()
```

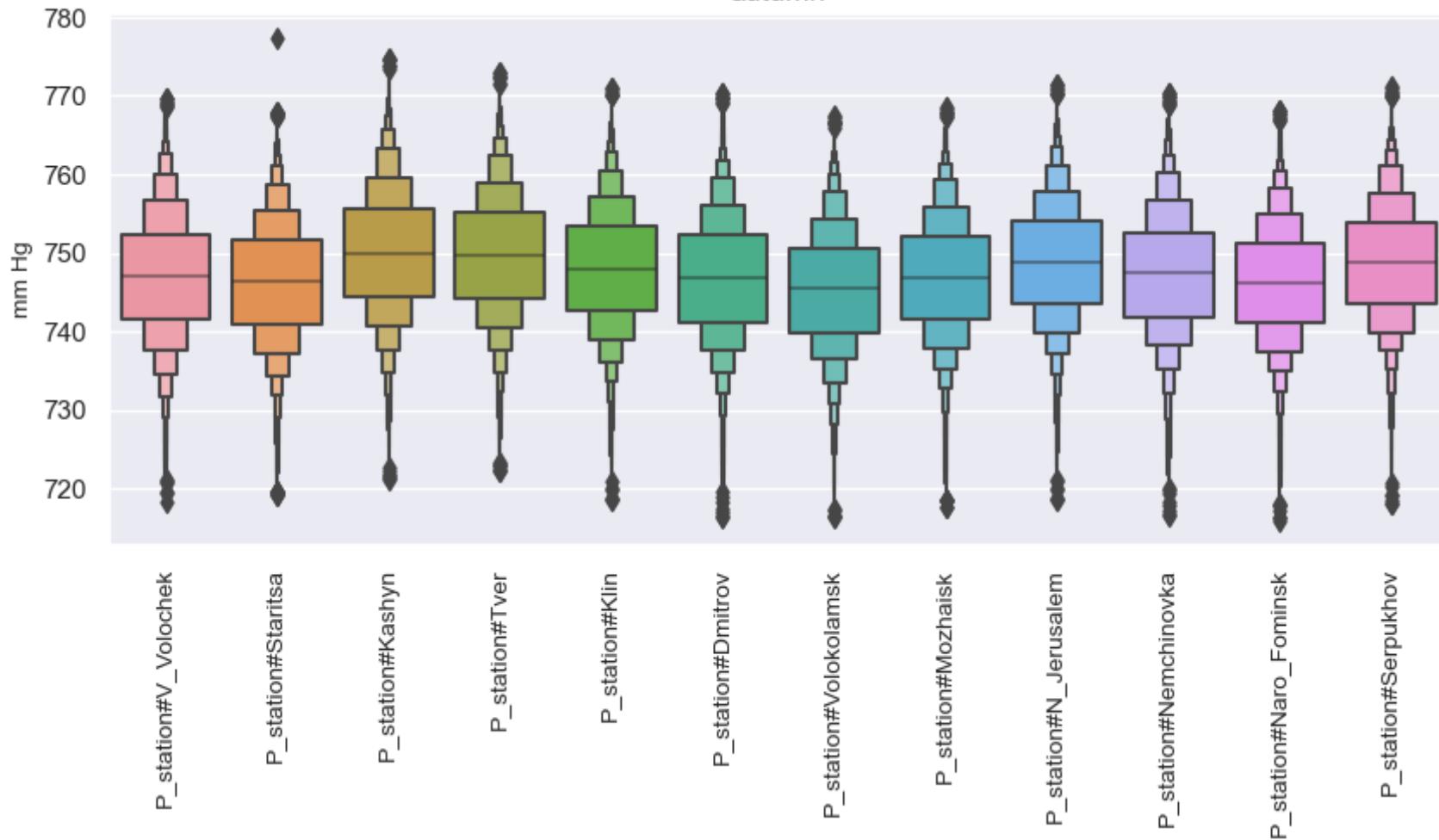
Распределение значений P\_station в разрезе метеостанций:  
spring



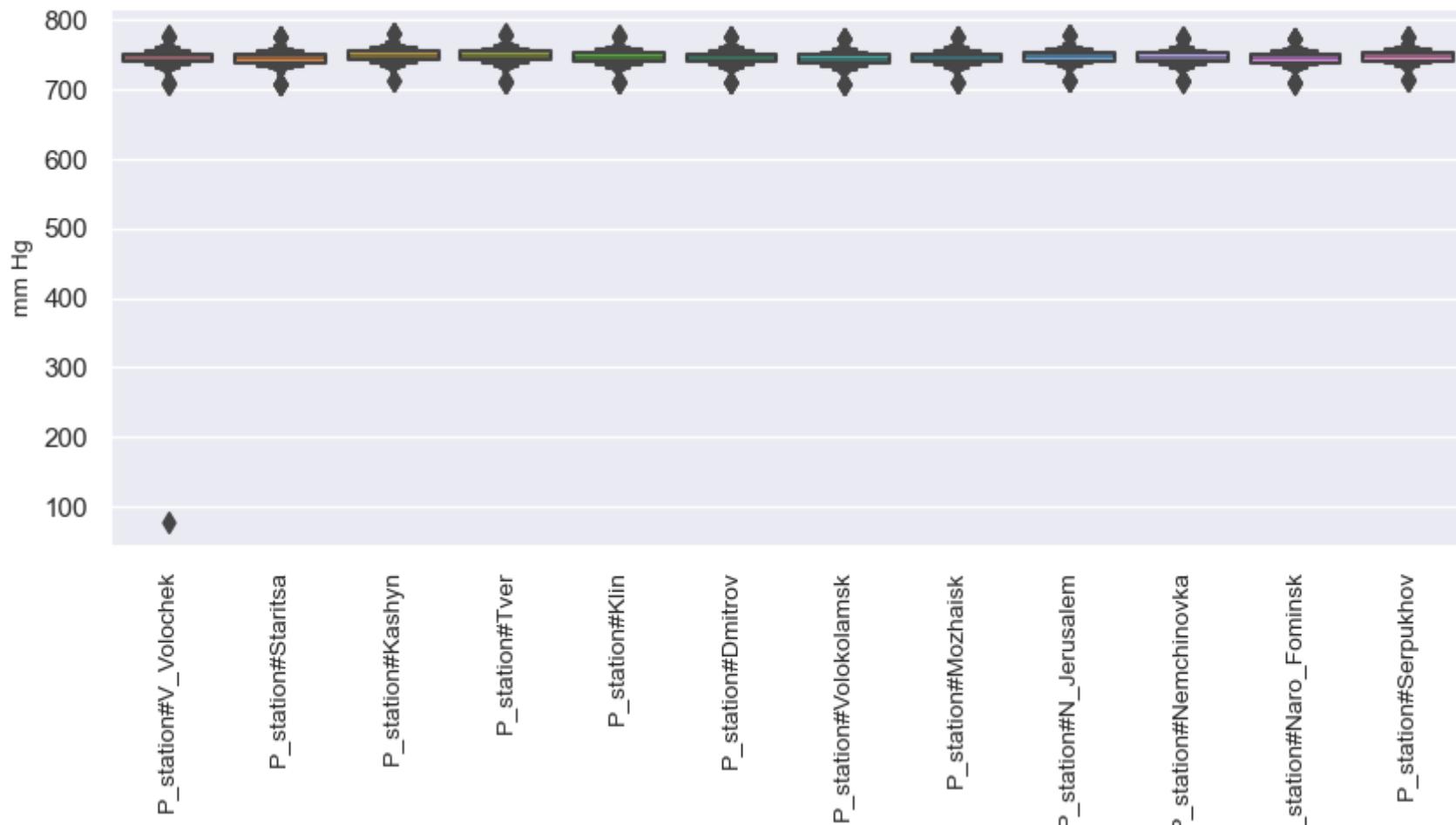
Распределение значений P\_station в разрезе метеостанций:  
summer



Распределение значений P\_station в разрезе метеостанций:  
autumn



### Распределение значений P\_station в разрезе метеостанций: winter



Даже беглого взгляда на графики достаточно, чтобы понять, что в данных содержится ошибки, выразившиеся в нереальных значениях давления.

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

In [109...]

```
print(f'Минимальное значение: {np.nanmin(df_tmp42)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp42)},\n'
```

```
f'Средняя: {np.nanmean(df_tmp42)},\n'
f'Медиана: {np.nanmedian(df_tmp42)}')
```

Минимальное значение: 77.9,  
Максимальное значение: 781.1,  
Средняя: 745.8412276777972,  
Медиана: 745.8

### Проверим соответствие давления на уровне метеостанции давлению на уровне моря

Определим расхождения между архивным значением давления на уровне метеостанции и расчётым давлением, полученным по барометрической формуле от давления на уровне моря. Сохраним координаты ошибочных значений.

Определим критерий для некорректных значений в 1,0 mmHg (Предположим, что использовалась барометрическая формула, сама имеющая некоторую погрешность пусть и для малых высот, плюс могла возникнуть ошибка округления при расчётах)

In [110...]

```
criterium = 1 # критерий для определения аномального отклонения
# расчётного значения давления на уровне станции от архивного

# В цикле по столбцам df_tmp42
# - определим промежуточный DF для расчётов;
# - заполним его значениями, необходимыми для приведения давления к уровню моря,
# - расчитаем давление на уровне моря по значению на уровне станции ,
# - рассчитаем абсолютное отклонение расчётных значений от архивных,
# - определим критерий ошибки и соответствие ему архивных значений
# Исходим из идентичности индексов всех датафреймов в архивах

list_error_at = [] # определим список координат ошибочных значений в df_tmp42
for col in df_tmp42.columns:
    df_p1 = pd.DataFrame(None, index=df_tmp42.index) # создадим пустой DF
    station_name = col[len(PARAMETER42)+1:] # выделяем название метеостанции из названия столбца
    height = df_station_dists[df_station_dists.station == station_name].height.values[0] # находим высоту метеостанции

    df_p1 = (df_p1
        .assign(station=station_name,
               P0=df_tmp41.loc[:,PARAMETER41+'#'+station_name], # давление на уровне моря
               P1=df_tmp42.loc[:,PARAMETER42+'#'+station_name], # давление на уровне станции
               T=dict_df_parameters['df_T'][f"{'T'+col[len(PARAMETER42):]}"], # Температура воздуха
               h=height, # высота над уровнем моря
               P1_calc=lambda x: P0_to_P(pHg0_=x.P0, h_=x.h, t_=x["T"])), # Расчётная величина давления на уровне моря
               P1_delta=lambda x: abs(x.P1 - x.P1_calc), # абсолютное отклонение расчётной величины от архивной
               P1_error=lambda x: x.P1_delta >= criterium, # критерий ошибки для отклонения расчётной величины mmHg
            )
```

```

        )
# Расширяем список координат ошибочных значений в df_tmp42:
# Если абсолютное отклонение расчётного значения от архивного больше или равно установленного критерия
list_error_at = list_error_at + (df_p1
    .apply(
        lambda x: (x.name, station_name) if x.P1_delta >= criterium else np.nan,
        axis=1)
    .dropna()
    .tolist()
)

print(f'Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного '
      f'для метеостанции {station_name} = {df_p1.P1_error.sum()}')


#list_error_at

```

Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции V\_Volochev = 81  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Staritsa = 35  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Kashyn = 5  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Tver = 35112  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Klin = 38  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Dmitrov = 24  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Volokolamsk = 3  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Mozhaisk = 42  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции N\_Jerusalem = 36  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Nemchinovka = 32  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Naro\_Fominsk = 25  
 Количество аномальных отклонений расчётного давления на уровне метеостанции от архивного для метеостанции Serpukhov = 638

Заменим ошибочные значения на NaN

```
In [111...]: for error in list_error_at:
    df_tmp42.at[error[0], PARAMETER42 + '#' + error[1]] = np.nan
```

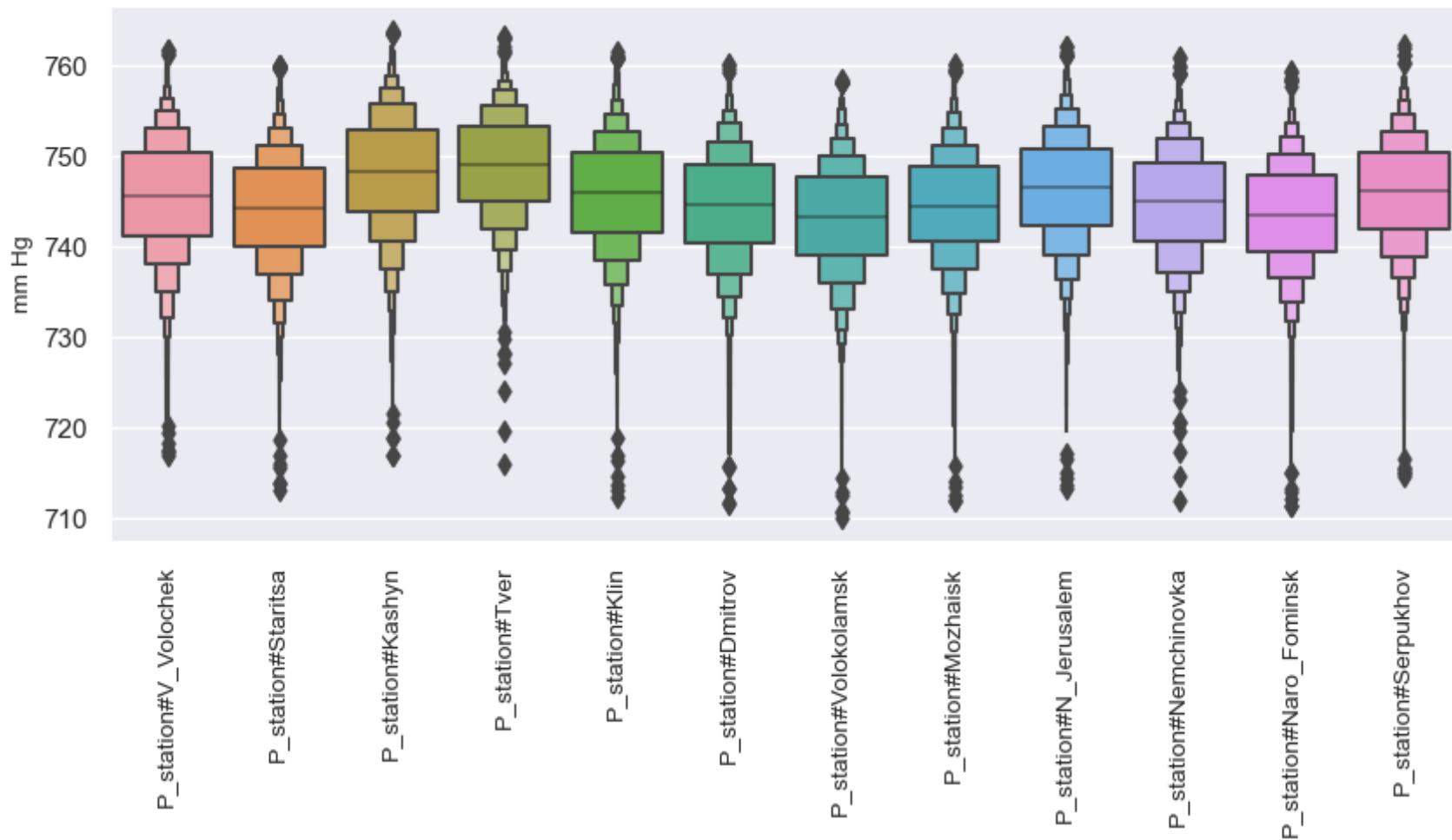
Так как атмосферное давление, приведённое к уровню моря, и атмосферное давление на уровне метеостанции жёстко связаны между собой барометрической формулой, у нас нет необходимости искать выбросы в значениях показателя P\_station. После проверки соответствия архивных и расчётных значений все некорректные значения для P\_station уже удалены. В свою очередь, P\_sea уже очищен от аномалий и ошибок ранее, и ошибочные значения в нём уже исправлены.

**Повторно, после удаления всех ошибочных значений, визуализируем архив давления на уровне станции (P\_station) по сезонам**

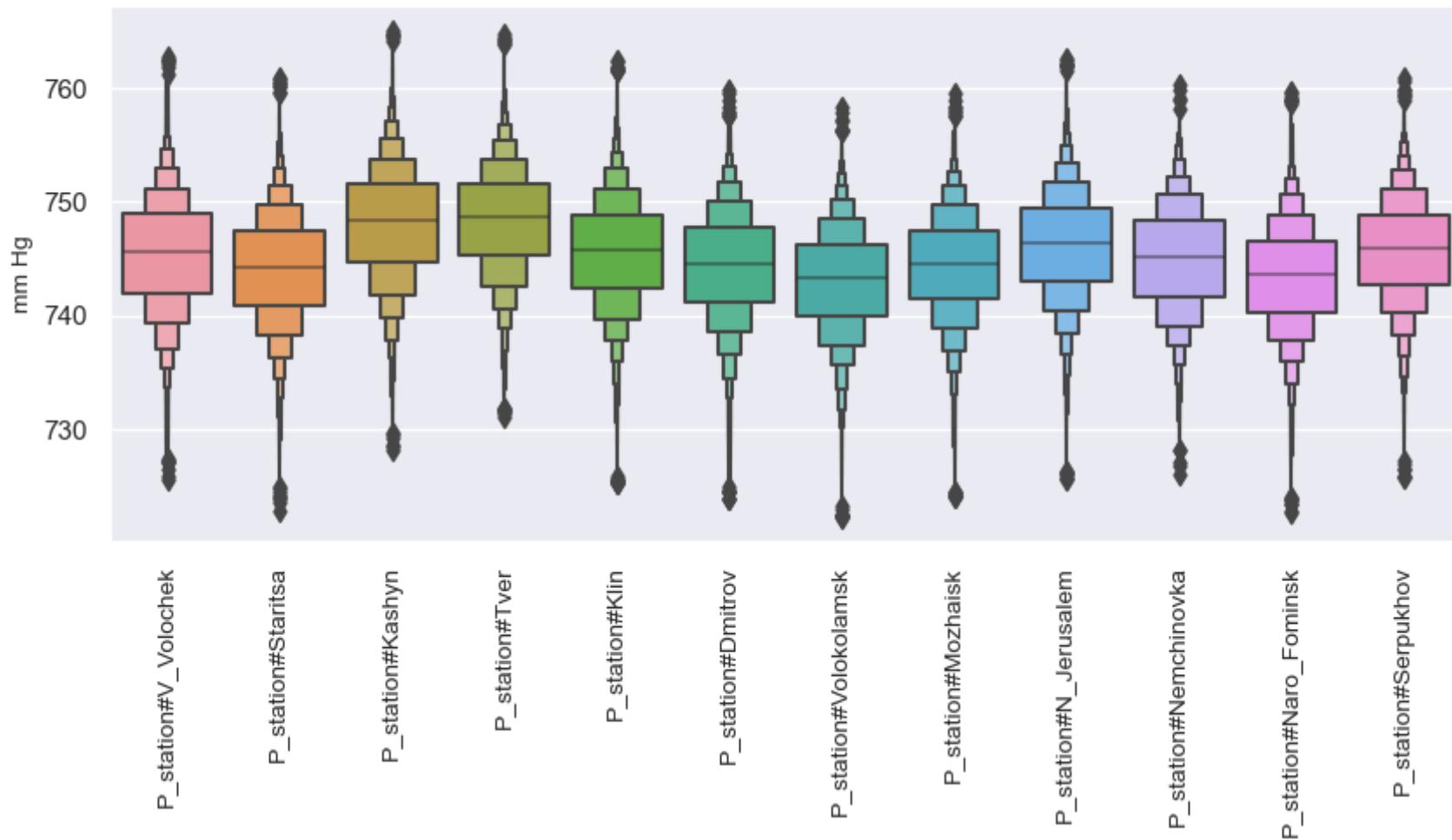
In [112...]

```
# 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_tmp42).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp42[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('мм Hg', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER42} в разрезе метеостанций после удаления ошибок:\n'{season_name})
plt.show()
```

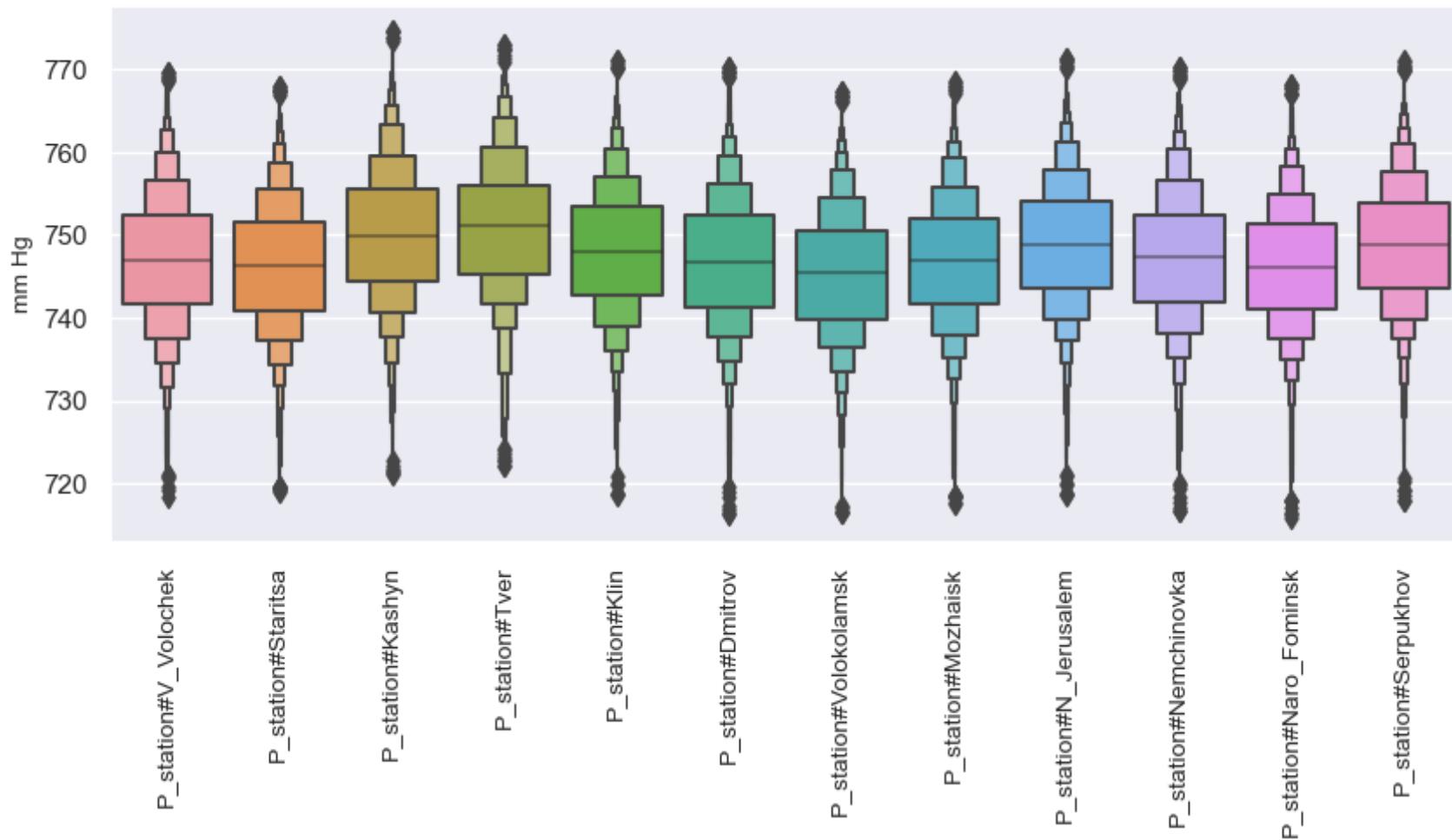
Распределение значений P\_station в разрезе метеостанций после удаления ошибок:  
spring



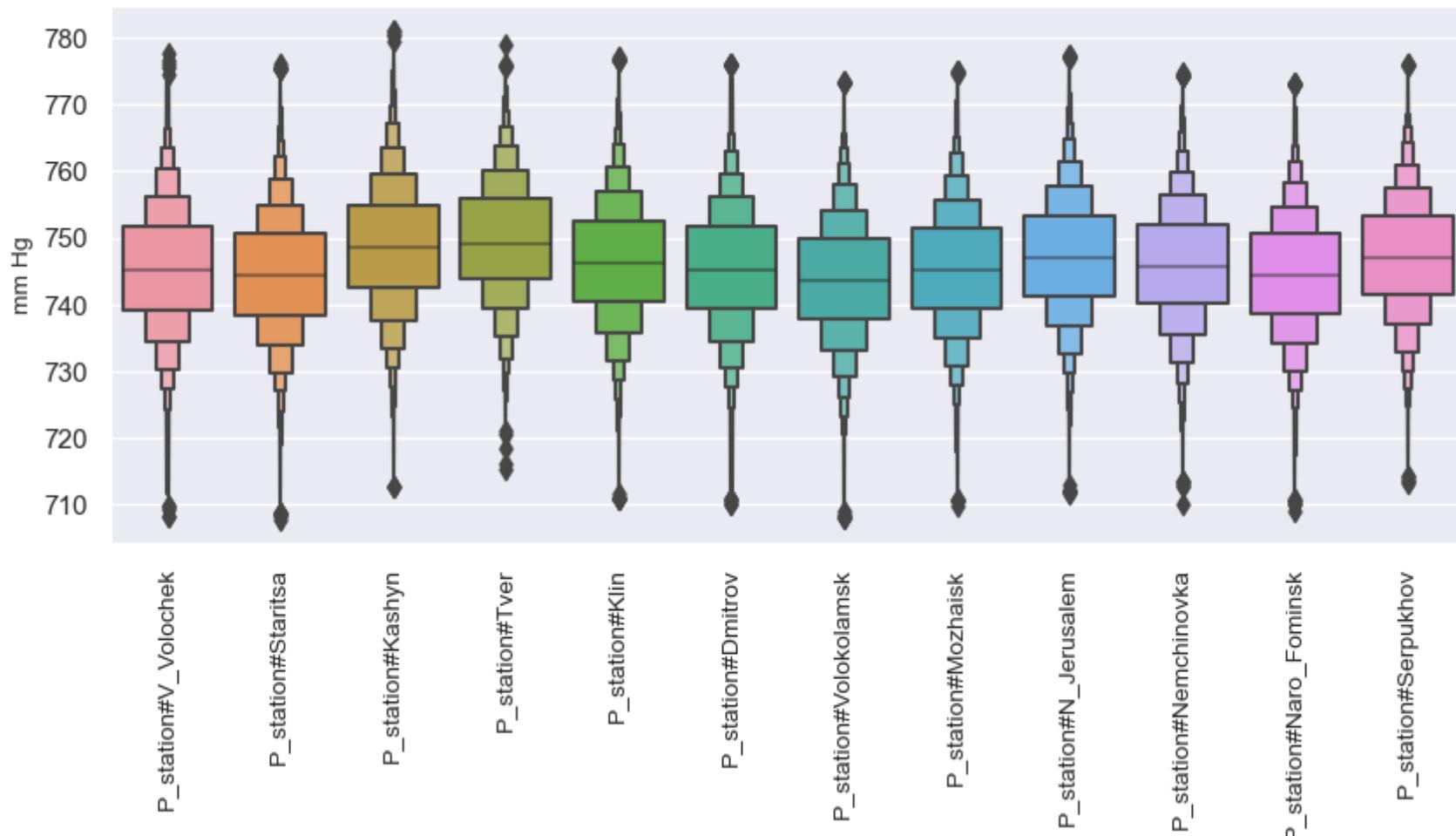
Распределение значений P\_station в разрезе метеостанций после удаления ошибок:  
summer



Распределение значений P\_station в разрезе метеостанций после удаления ошибок:  
autumn



## Распределение значений P\_station в разрезе метеостанций после удаления ошибок: winter



Проверим, не осталось ли ошибок в части чрезмерно низких и чрезмерно высоких значений атмосферного давления.

In [113]:

```
print(f'Среднеквадратическое отклонение значений атмосферного давления между станциями на момент наблюдения, mmHg:\n'
      f'Для уровня менее 720 mmHg, минимум = {df_tmp41[df_tmp41 < 730].dropna(how="all").agg("std", axis=1).min()}\n'
      f'Для уровня менее 720 mmHg, максимум = {df_tmp41[df_tmp41 < 730].dropna(how="all").agg("std", axis=1).max()}\n'
      f'Для уровня более 770 mmHg, минимум = {df_tmp41[df_tmp41 > 770].dropna(how="all").agg("std", axis=1).min()}\n'
      f'Для уровня более 770 mmHg, максимум = {df_tmp41[df_tmp41 > 770].dropna(how="all").agg("std", axis=1).max()}\n'
)
```

Среднеквадратическое отклонение значений атмосферного давления между станциями на момент наблюдения, mmHg:  
Для уровня менее 720 mmHg, минимум = 0.0  
Для уровня менее 720 mmHg, максимум = 2.0266601666415567  
Для уровня более 770 mmHg, минимум = 0.0  
Для уровня более 770 mmHg, максимум = 2.5958976021977924

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

```
In [114...]: print(f'Минимальное значение: {np.nanmin(df_tmp42)}\n'
      f'Максимальное значение: {np.nanmax(df_tmp42)}\n'
      f'Средняя: {np.nanmean(df_tmp42)}\n'
      f'Медиана: {np.nanmedian(df_tmp42)}')
```

Минимальное значение: 707.8,  
Максимальное значение: 781.1,  
Средняя: 745.724926413966,  
Медиана: 745.7

Исходя из значений стандартного отклонения, даже предельные значения атмосферного давления не выглядят как выбросы.

### Сохраним очищенные данные в файл параметров

```
In [115...]: # # 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_'+PARAMETER42 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER42].to_csv(
    path_or_buf=f'{clean_path}df_{PARAMETER42}.csv'
)
print('DONE!')
```

df\_P\_station.csv -> DONE!

#### 4.2.2. Замена удалённых и пропущенных значений показателя P\_station расчётными (с применением барометрической формулы), расчёт показателя атмосферного давления на уровне метеостанции P\_station для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

In [116]:

```
# Добавим в df_tmp42 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 PARAMETER42#Chashnikovo и PARAMETER42#Rfrnce_point
df_tmp42 = (df_tmp42.
             assign(col_name1 = np.nan,
                   col_name2 = np.nan).
             rename(columns={"col_name1": PARAMETER42+'#'+ 'Chashnikovo',
                            "col_name2": PARAMETER42+'#'+ 'Rfrnce_point'}))
         )

df_tmp42.sample(3, random_state=56)
```

Out[116]:

	P_station#V_Volochek	P_station#Staritsa	P_station#Kashyn	P_station#Tver	P_station#Klin	P_station#Dmitrov	P_station#Volokolamsk	P_station#
2014-02-22 03:00:00	750.2	749.1	755.0	753.8	752.0	751.3	748.4	
2015-05-16 03:00:00	734.2	731.8	733.4	NaN	731.0	729.2	729.3	
2020-06-18 18:00:00	746.9	745.2	751.1	749.3	747.3	746.5	743.8	

In [117]:

```
# Добавим в архив параметров P_sea столбец для Чашниково и будущей центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER42#Chashnikovo и PARAMETER42#Rfrnce_point
dict_df_parameters['df_'+PARAMETER42] = (dict_df_parameters['df_'+PARAMETER42].
                                             assign(col_name1 = np.nan,
                                                   col_name2 = np.nan).
                                             rename(columns={"col_name1": PARAMETER42+'#'+ 'Chashnikovo',
```

```

        "col_name2": PARAMETER42+'#'+Rfrnce_point' }
    )
)
dict_df_parameters['df_'+PARAMETER42].sample(3, random_state=56)

```

Out[117]:

	P_station#V_Volochek	P_station#Staritsa	P_station#Kashyn	P_station#Tver	P_station#Klin	P_station#Dmitrov	P_station#Volokolamsk	P_station#
2014-02-22 03:00:00	750.2	749.1	755.0	753.8	752.0	751.3	748.4	
2015-05-16 03:00:00	734.2	731.8	733.4	734.1	731.0	729.2	729.3	
2020-06-18 18:00:00	746.9	745.2	751.1	749.3	747.3	746.5	743.8	

In [118...]

```

# Добавим в архив метеостанций df Чашниково и df для центральной точки,
# Создадим в нём столбец с названием PARAMETER42

```

```

dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER42}
                                                 )
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER42}
                                                 )
                                         )
dict_df_locations['df_Chashnikovo'].sample(3, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(3, random_state=56)

```

Out[118]:

	T	T_min	T_max	P_sea	P_station
2014-02-22 03:00:00	-4.156558	-4.156558	-2.109643	768.336709	NaN
2015-05-16 03:00:00	9.181730	9.181730	10.434571	745.095535	NaN
2020-06-18 18:00:00	27.439052	25.366130	30.241632	761.480701	NaN

Out[118]:

	T	T_min	T_max	P_sea	P_station
2014-02-22 03:00:00	-3.589183	-3.794602	-2.700734	767.373725	NaN
2015-05-16 03:00:00	7.789650	7.789650	8.797603	746.708428	NaN
2020-06-18 18:00:00	29.404954	25.572651	29.941094	760.796222	NaN

## Заполнение NaN расчётными значениями по барометрической формуле

Подсчитаем количество NaN

In [119...]

```
# np.isnan(df_tmp42).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp42).sum())
```

Out[119]:

198210

Составим массив координат значений NaN в df\_tmp42

In [120...]

```
arr_idx_nan = np.where(np.isnan(df_tmp42)) # Кортеж из двух одномерных массивов координат  
arr_idx_nan = np.stack((arr_idx_nan[0], arr_idx_nan[1]), axis = 1) # массив из парных координат
```

По координатам значений NaN, заменим в df\_tmp42 NaNы на расчётные значения

In [121...]

```
start_time = time.time() # для замера времени выполнения кода  
  
for idxs in arr_idx_nan: # по индексам в массиве индексов NaNов  
  
    station_name = df_tmp42.columns[idxs[1]][len(PARAMETER42)+1:] # выделяем название метеостанции из названия столбца  
    height = df_station_dists[df_station_dists.station == station_name].height.values[0] # находим высоту метеостанции  
    temperature = dict_df_parameters['df_T'].iat[idxs[0], idxs[1]] # находим Определяем температуру воздуха  
    pHg0 = df_tmp41.iat[idxs[0], idxs[1]] # находим давление на уровне станции
```

```
P1 = P0_to_P(pHg0_=pHg0, h_=height, t_=temperature) # вычисляем по барометрической формуле давление на уровне моря

df_tmp42.iat[idxs[0], idxs[1]] = P1 # записываем значение в df_tmp42

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')

print(f'Осталось NaN: {np.sum(np.isnan(df_tmp42).sum())}')
```

Elapsed time=00:02:02

Осталось NaN: 14

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

### Перезапись архивов в части показателя P\_station

In [122...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER42] = df_tmp42.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[:, PARAMETER42] = df_tmp42.loc[:, PARAMETER42 + '#' + name_df[3:]]

# Выведем случайные ряды из архивов
dict_df_parameters['df_'+PARAMETER42].sample(7, random_state=56)
dict_df_locations['df_Chashnikovo'].sample(7, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(7, random_state=56)
```

Out[122]:

	P_station#V_Volochek	P_station#Staritsa	P_station#Kashyn	P_station#Tver	P_station#Klin	P_station#Dmitrov	P_station#Volokolamsk	P_station#Leningrad
<b>2014-02-22 03:00:00</b>	750.200000	749.1	755.000000	753.800000	752.0	751.3	748.4	748.4
<b>2015-05-16 03:00:00</b>	734.200000	731.8	733.400000	735.361137	731.0	729.2	729.3	729.3
<b>2020-06-18 18:00:00</b>	746.900000	745.2	751.100000	749.300000	747.3	746.5	743.8	743.8
<b>2019-12-02 21:00:00</b>	740.100000	739.9	744.100000	744.868341	742.4	741.5	739.7	739.7
<b>2008-06-05 18:00:00</b>	747.539666	745.4	748.645338	749.807257	746.4	744.9	744.2	744.2
<b>2016-10-20 06:00:00</b>	759.800000	758.3	762.800000	762.200000	760.1	758.8	757.2	757.2
<b>2006-09-11 03:00:00</b>	748.400000	747.2	747.700000	750.708005	746.8	744.7	745.0	745.0

Out[122]:

	T	T_min	T_max	P_sea	P_station
<b>2014-02-22 03:00:00</b>	-4.156558	-4.156558	-2.109643	768.336709	747.597791
<b>2015-05-16 03:00:00</b>	9.181730	9.181730	10.434571	745.095535	725.921747
<b>2020-06-18 18:00:00</b>	27.439052	25.366130	30.241632	761.480701	743.060948
<b>2019-12-02 21:00:00</b>	-3.004683	-4.229056	-2.673631	758.773486	738.378842
<b>2008-06-05 18:00:00</b>	16.565949	12.255838	16.565949	760.858054	741.771366
<b>2016-10-20 06:00:00</b>	0.117681	0.074269	0.916309	775.972222	755.350419
<b>2006-09-11 03:00:00</b>	9.855422	9.855422	11.886722	761.694029	742.139162

Out[122]:

	T	T_min	T_max	P_sea	P_station
<b>2014-02-22 03:00:00</b>	-3.589183	-3.794602	-2.700734	767.373725	754.041734
<b>2015-05-16 03:00:00</b>	7.789650	7.789650	8.797603	746.708428	734.256500
<b>2020-06-18 18:00:00</b>	29.404954	25.572651	29.941094	760.796222	749.008692
<b>2019-12-02 21:00:00</b>	-2.977448	-3.796107	-2.513436	758.016264	744.876404
<b>2008-06-05 18:00:00</b>	17.613615	11.754644	17.613615	761.523291	749.249895
<b>2016-10-20 06:00:00</b>	-0.382597	-0.369287	0.885072	775.916024	762.592727
<b>2006-09-11 03:00:00</b>	9.181940	9.181940	11.628949	762.828050	750.169523

Выведем, рандомные строки из df\_tmp42

In [123...]

```
df_tmp42.sample(7)
```

Out[123]:

	P_station#V_Volochek	P_station#Staritsa	P_station#Kashyn	P_station#Tver	P_station#Klin	P_station#Dmitrov	P_station#Volokolamsk	P_station#Chashnikovo
2020-08-06 18:00:00	752.900000	751.3	754.300000	754.600000	752.2	750.7	749.3	
2006-06-05 21:00:00	736.300000	734.4	738.100000	738.869257	736.5	735.2	733.3	
2014-02-06 21:00:00	743.000000	742.3	748.500000	747.627409	745.3	744.7	742.3	
2008-12-07 21:00:00	741.400000	741.2	744.500000	745.701241	742.2	741.1	740.3	
2021-06-06 15:00:00	745.900000	743.8	748.300000	747.700000	745.7	744.2	742.6	
2020-02-11 18:00:00	725.200000	724.8	729.800000	729.594581	727.4	726.9	724.4	
2008-10-04 12:00:00	744.794971	742.7	747.979329	747.793897	745.0	744.4	742.4	

In [124...]

```
for year in list_years:
    # Возьмём параллельно данные о давлении на уровне станции и на уровне моря
    data1 = dict_df_parameters['df_P_sea'][PARAMETER41+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_P_sea'].index.year == year)]\n\n    data2 = dict_df_parameters['df_P_station'][PARAMETER42+"#"+"Chashnikovo"]\n        [(dict_df_parameters['df_P_station'].index.year == year)]\n\n    fig, ax = plt.subplots(figsize=(10, 4))
```

```
g1 = sns.lineplot(data=data1,
                   color='indianred',
                   linewidth=0.5,
                   ax=ax)

g1 = sns.lineplot(data=data2,
                   color='forestgreen',
                   linewidth=0.5,
                   ax=ax)
# Добавляем легенду
green_line = mlines.Line2D([], [], color='forestgreen', label=PARAMETER42, linewidth=0.5)
red_line = mlines.Line2D([], [], color='indianred', label=PARAMETER41, linewidth=0.5)
dummy = ax.legend(handles=[green_line, red_line])

dummy = ax.set_ylabel('ммHg', size=10)
dummy = ax.set_xlabel('Месяцы', size=10)
dummy = plt.title(f'Чашниково: Ежедневная динамика параметров {PARAMETER42} и {PARAMETER41}, {year} год')
plt.show()
```

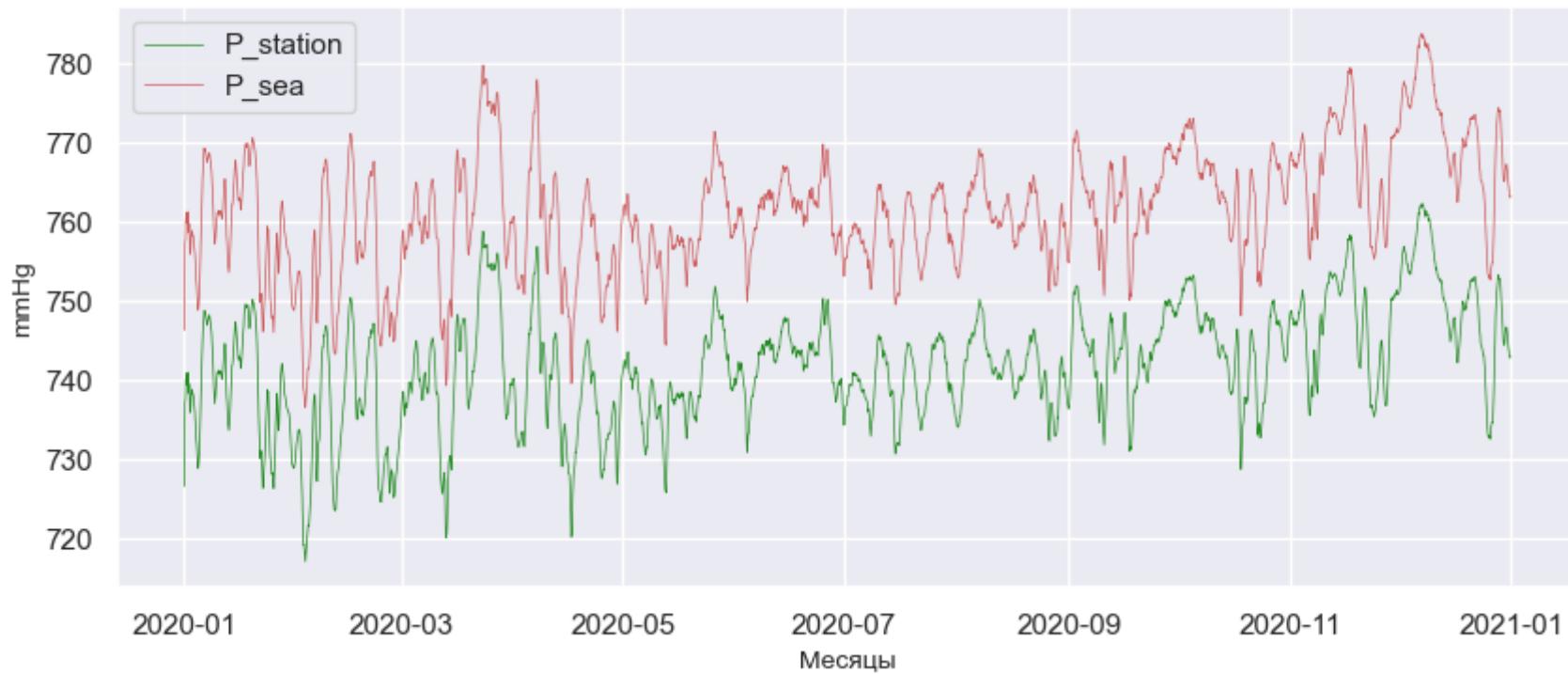
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2022 год



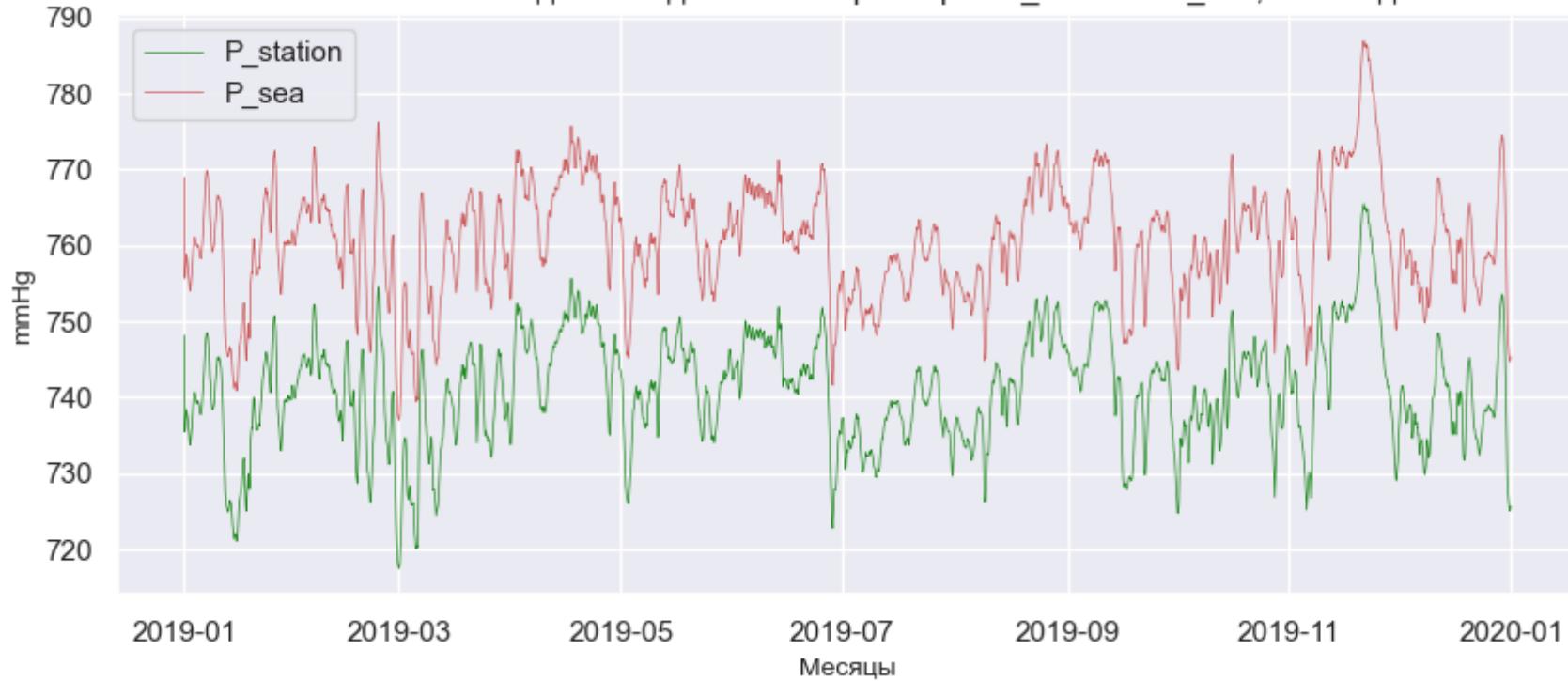
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2021 год



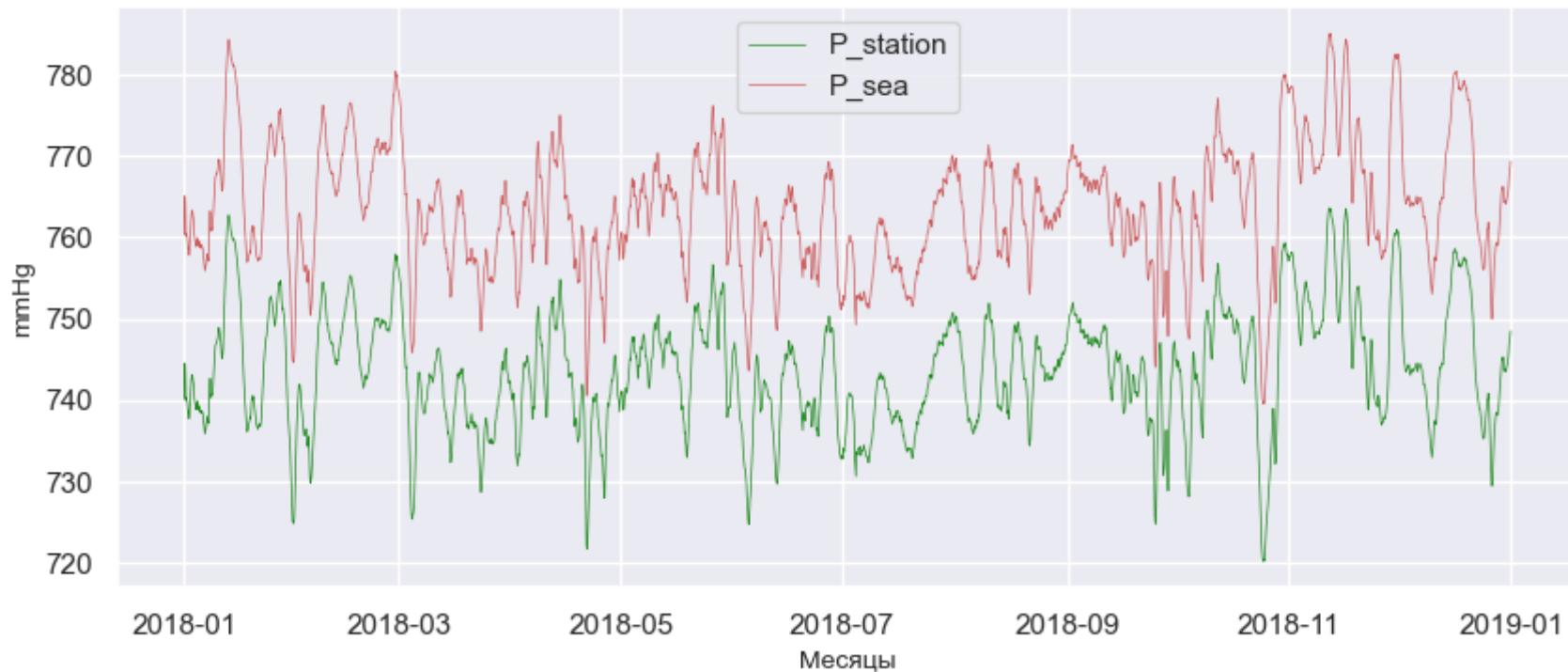
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2020 год



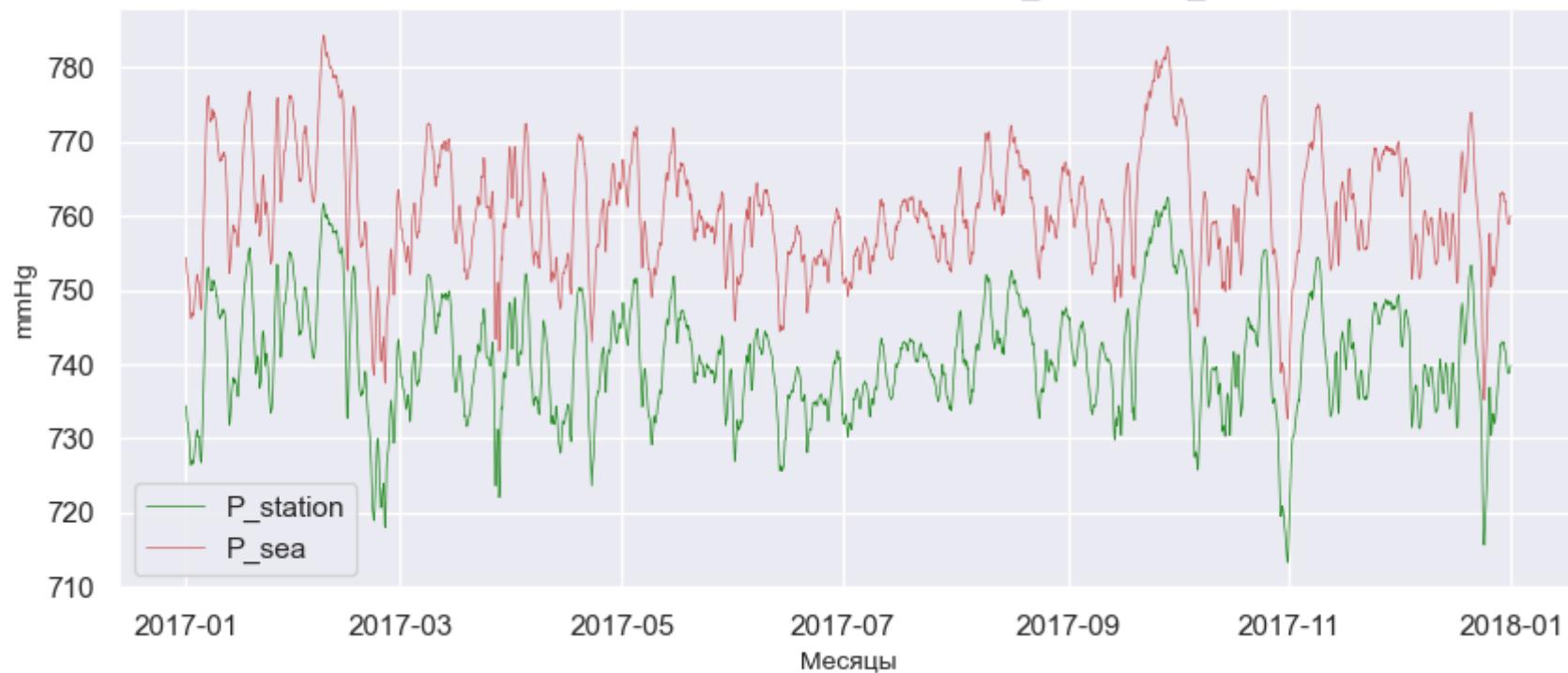
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2019 год



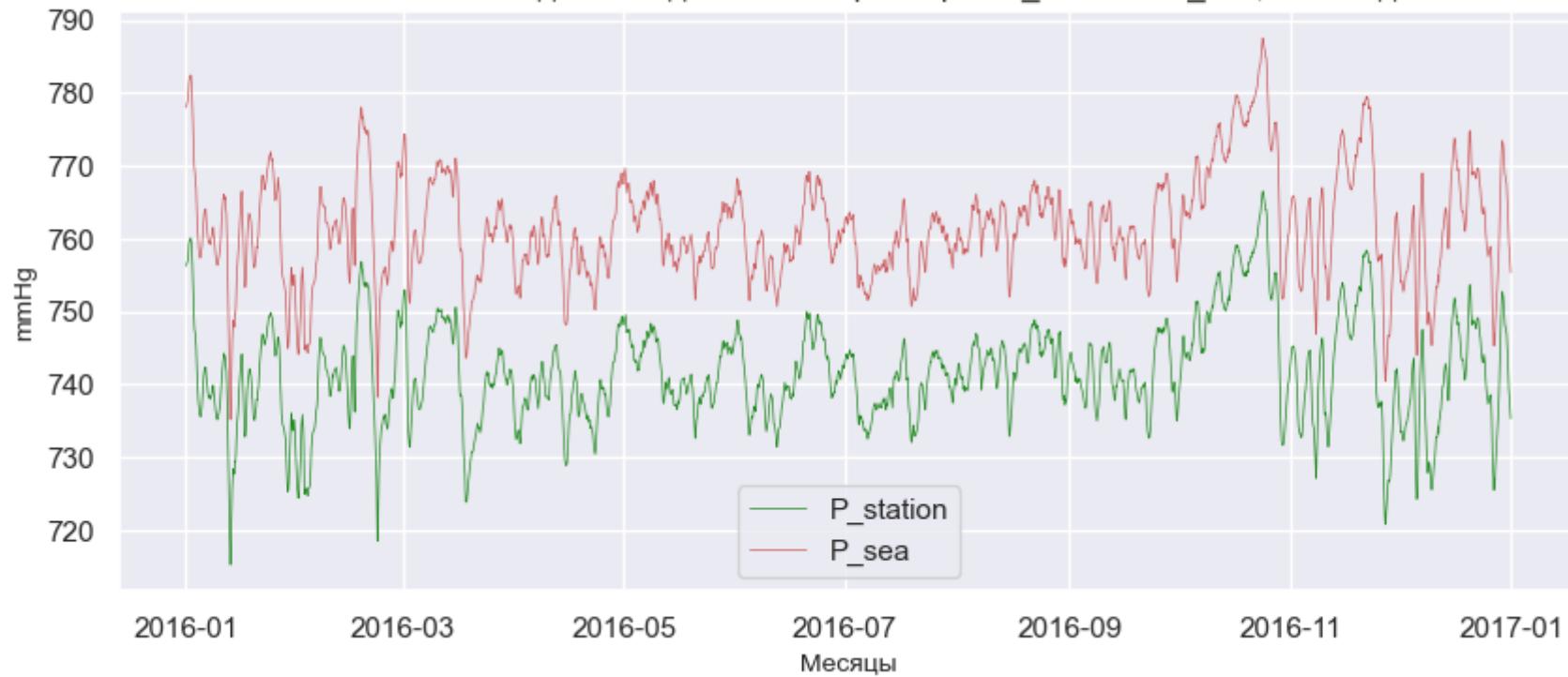
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2018 год



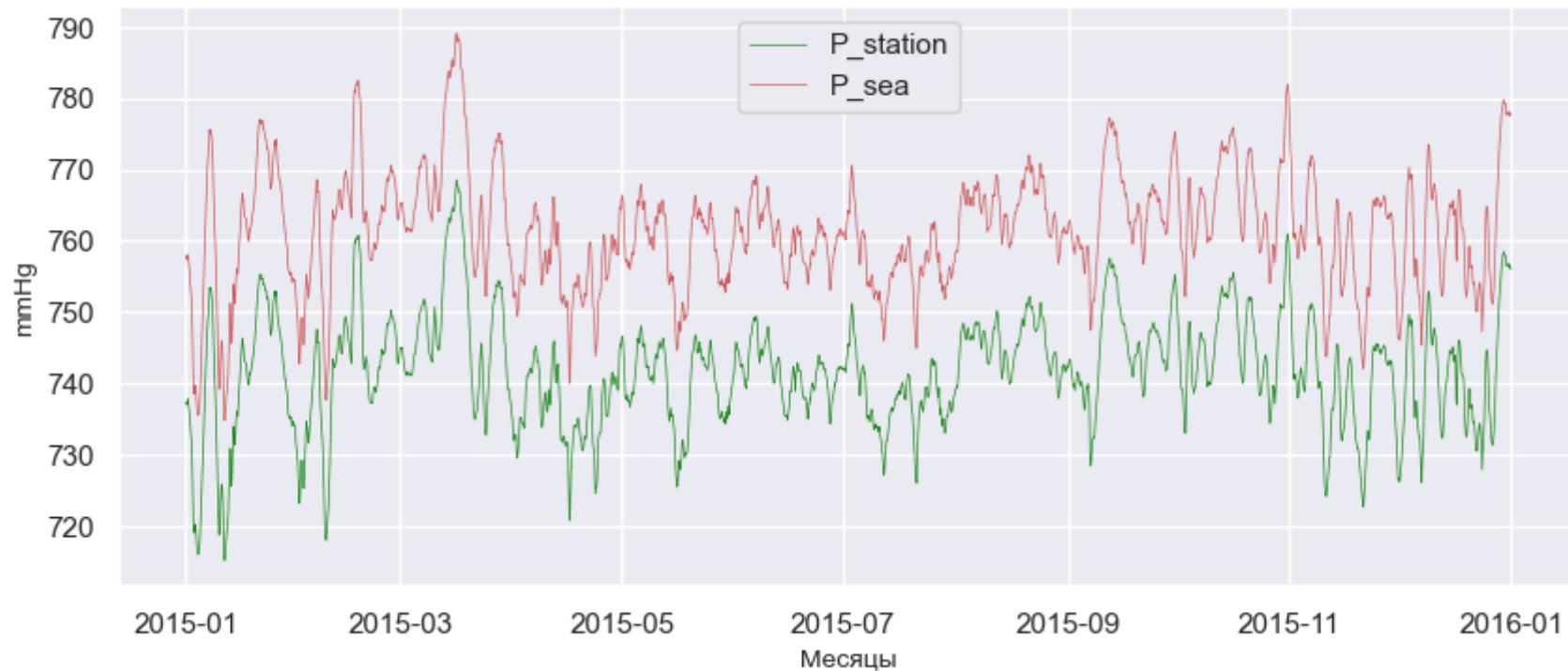
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2017 год



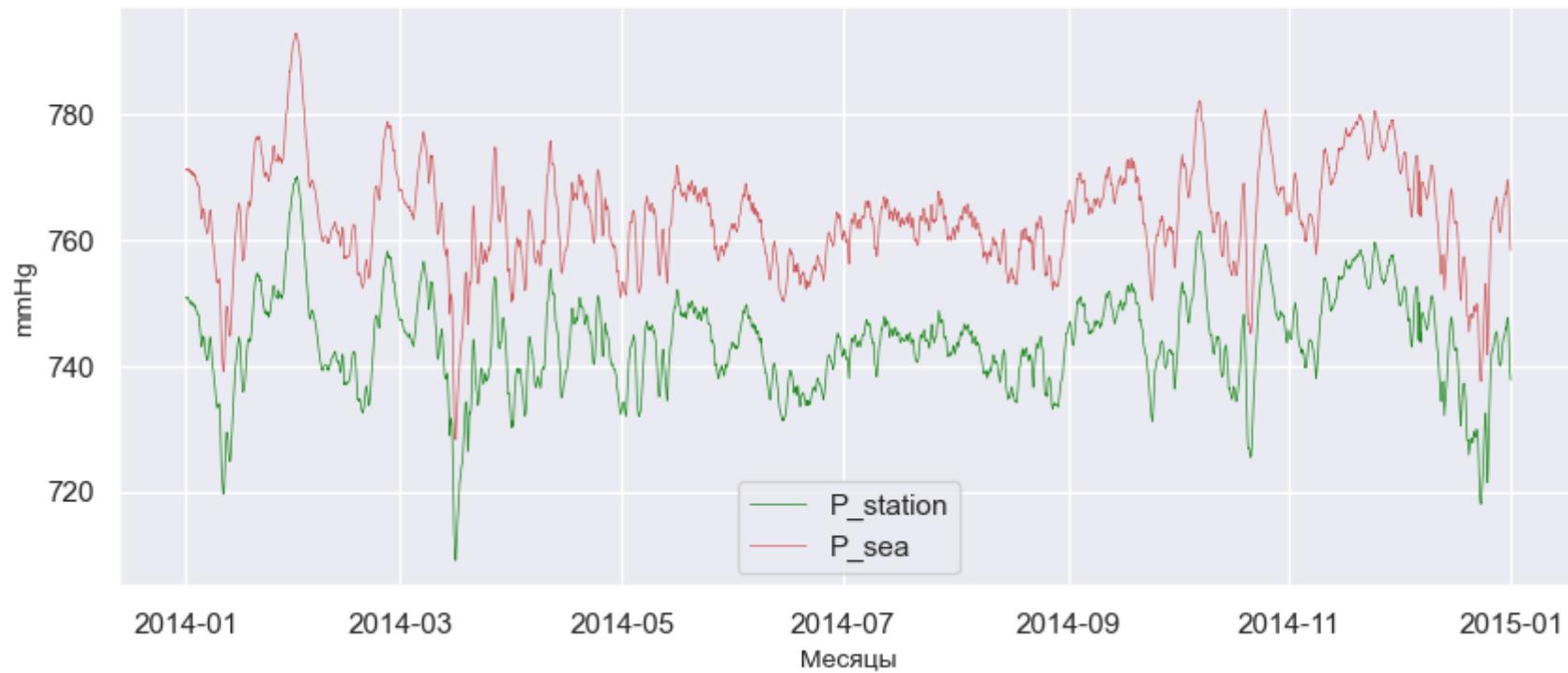
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2016 год



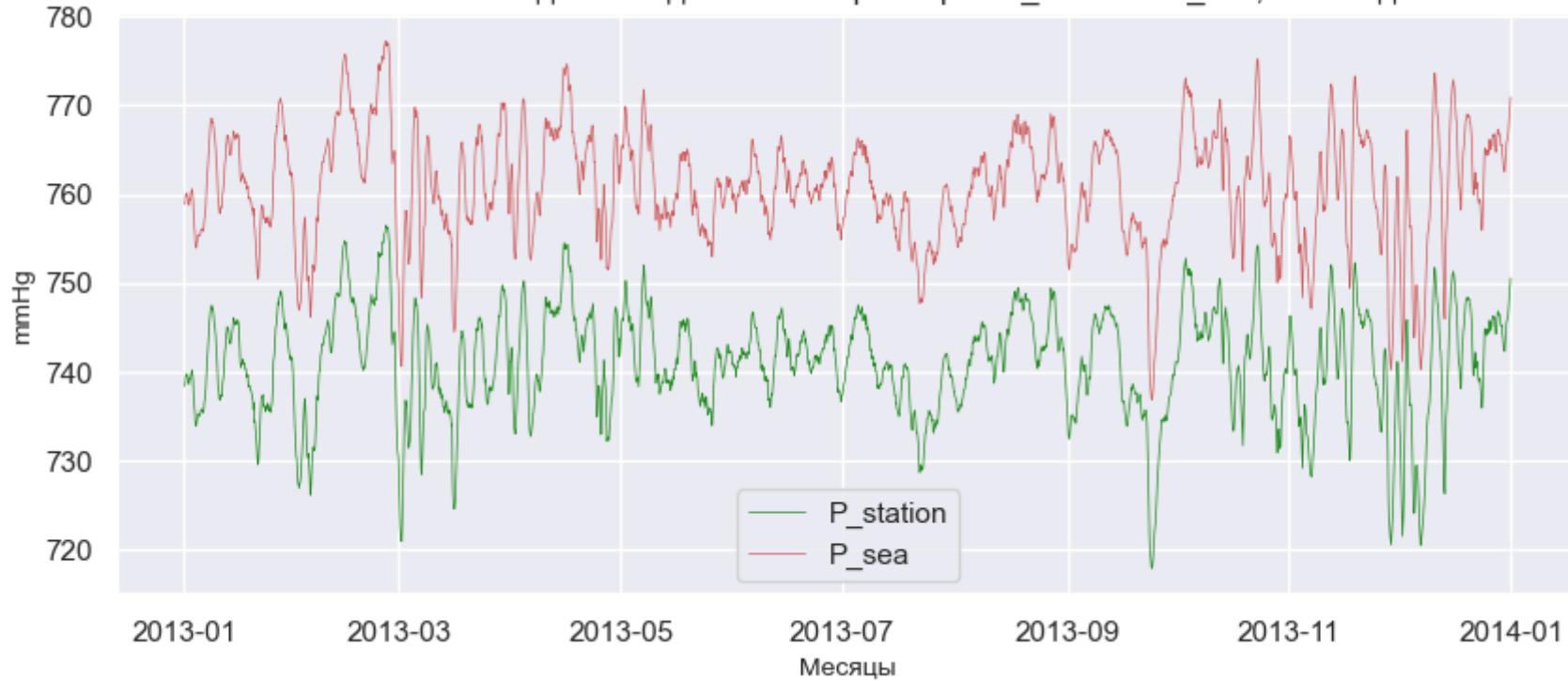
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2015 год



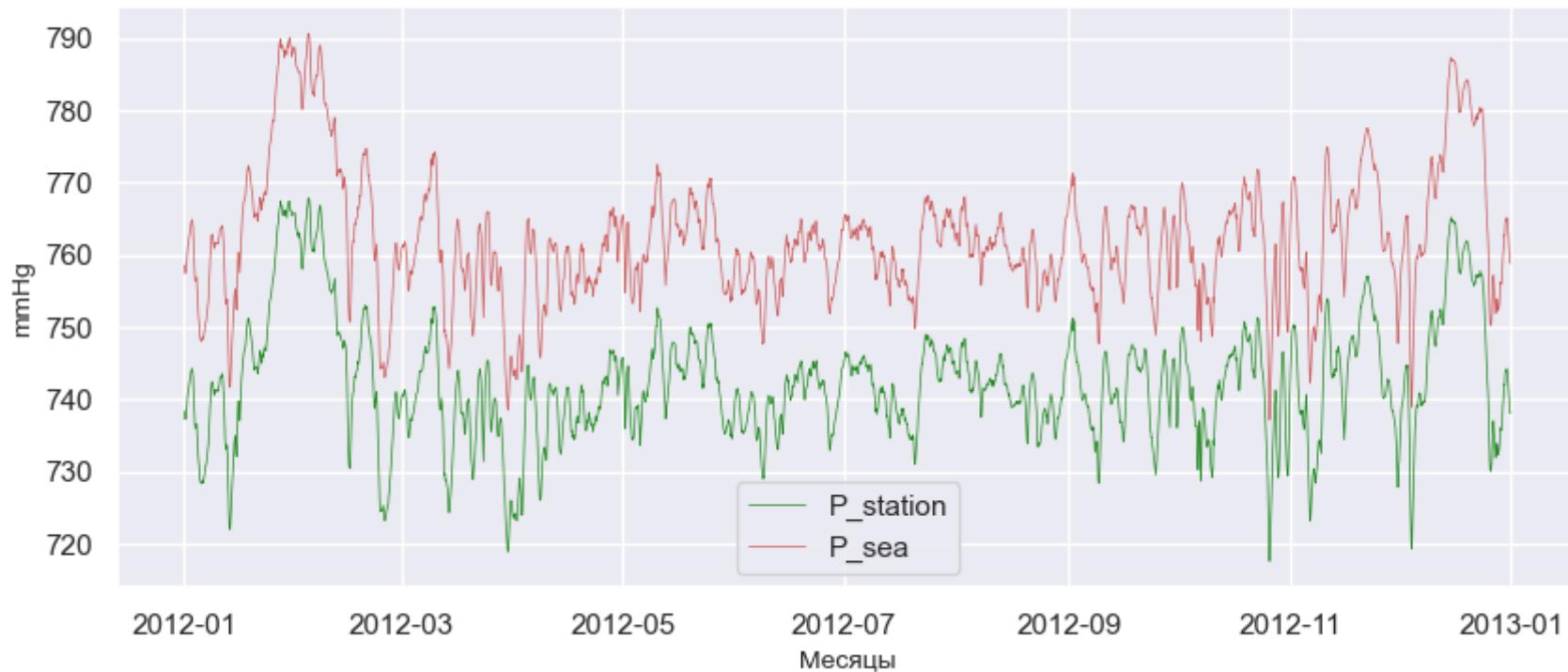
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2014 год



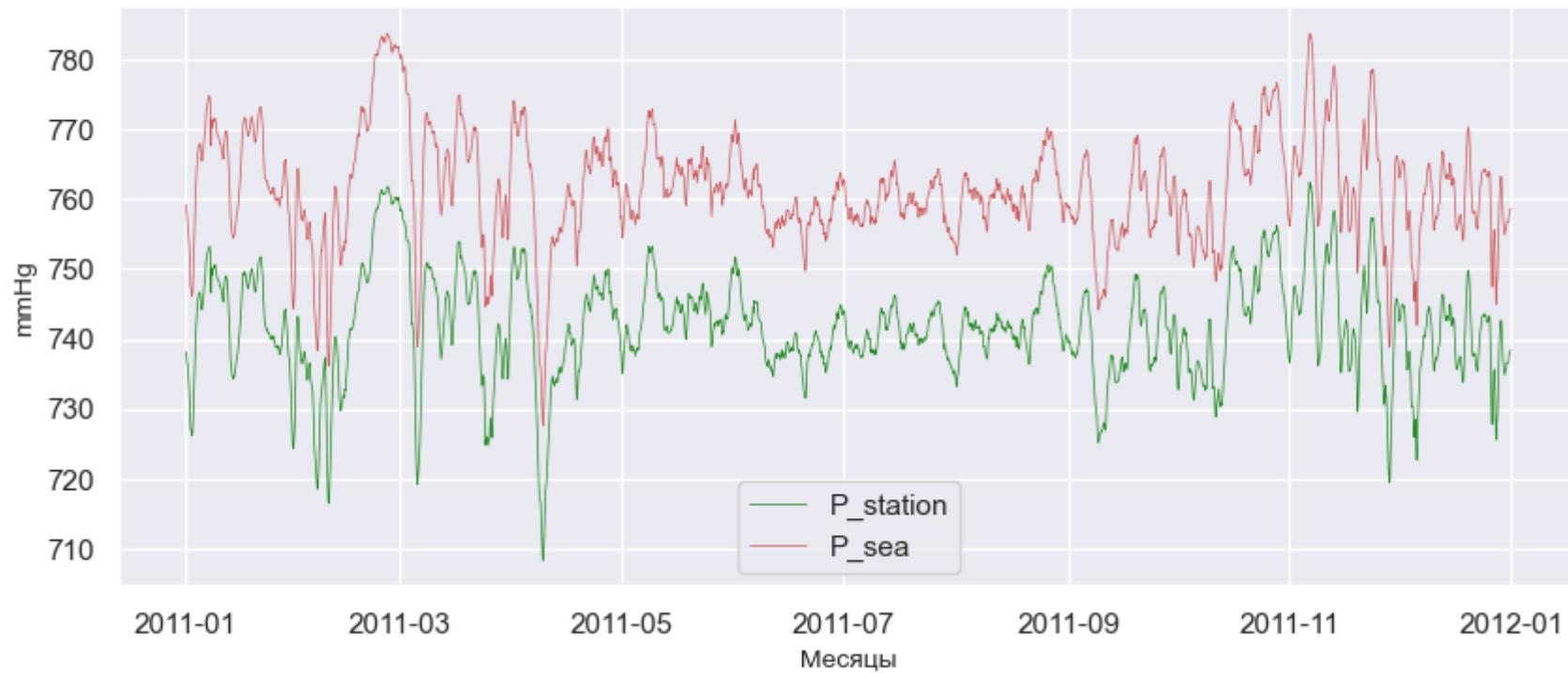
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2013 год



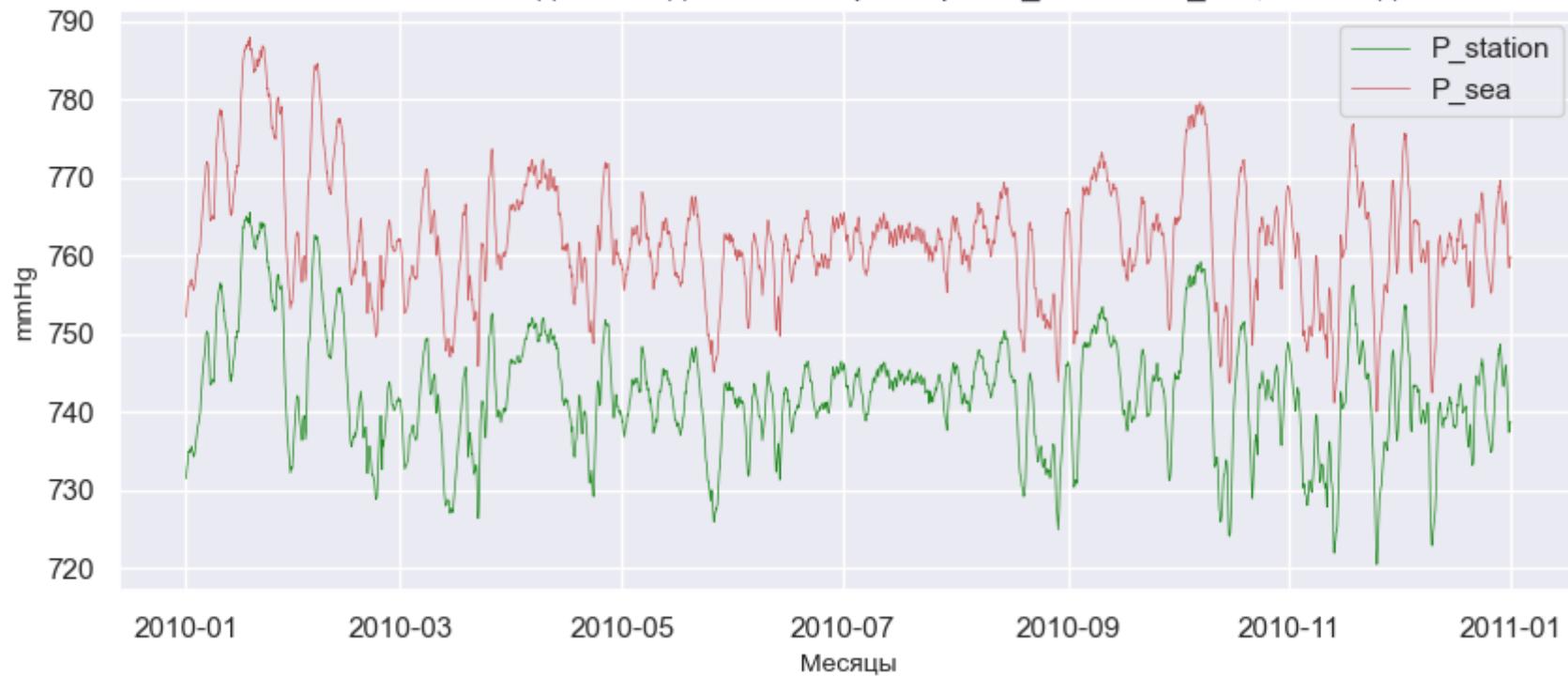
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2012 год



Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2011 год



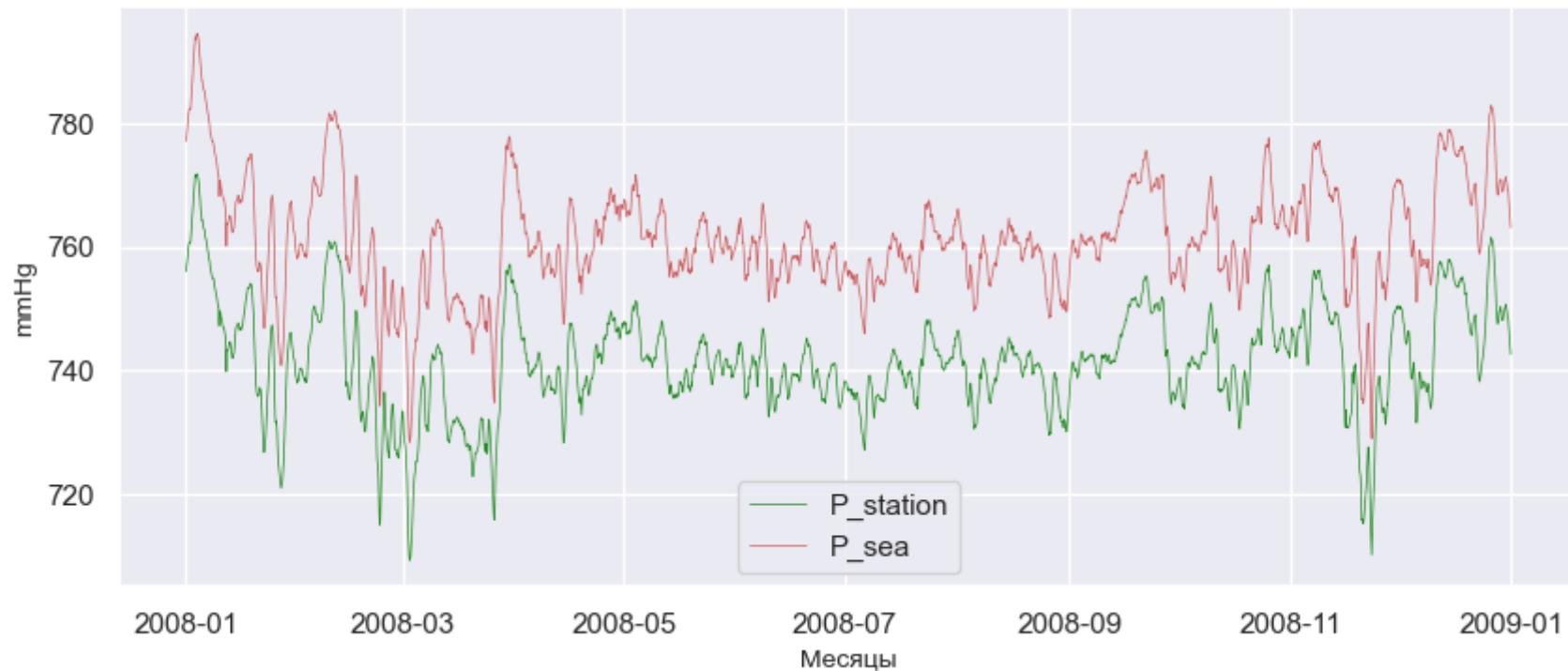
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2010 год



Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2009 год



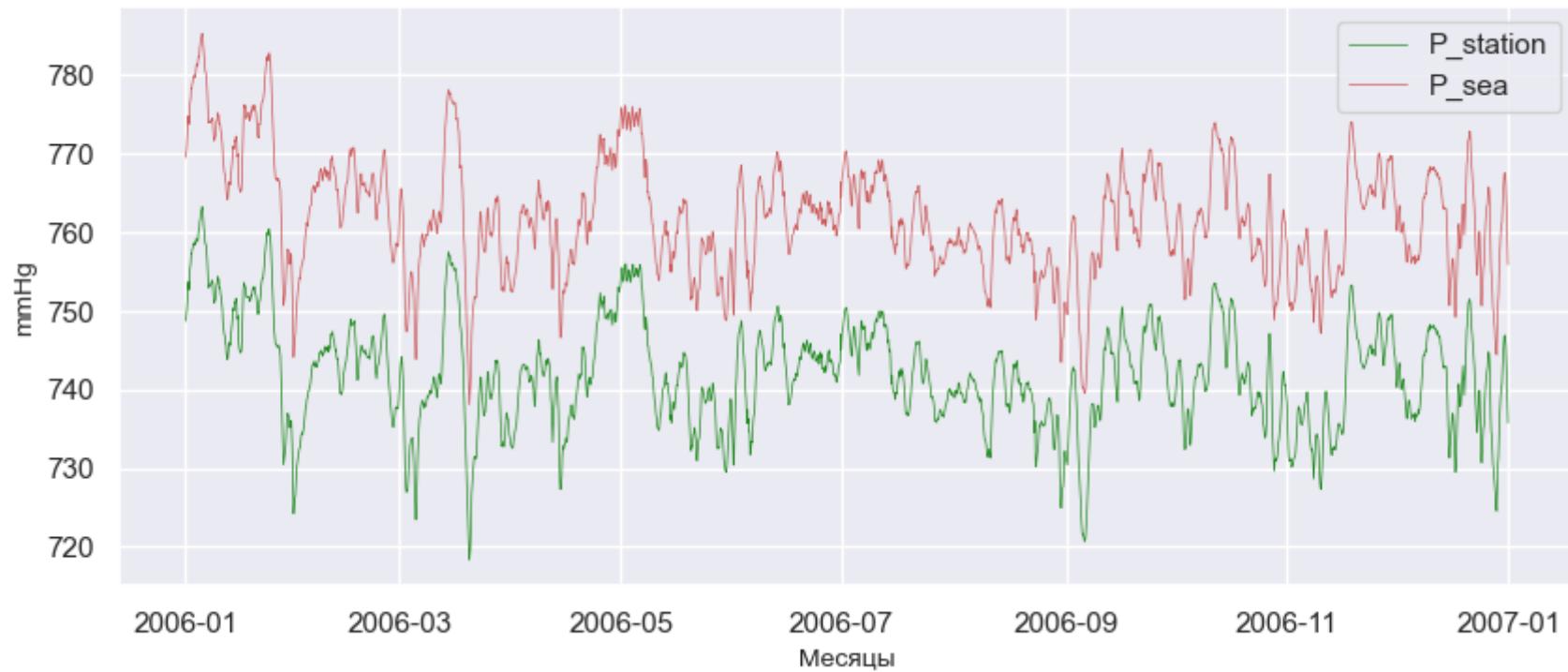
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2008 год



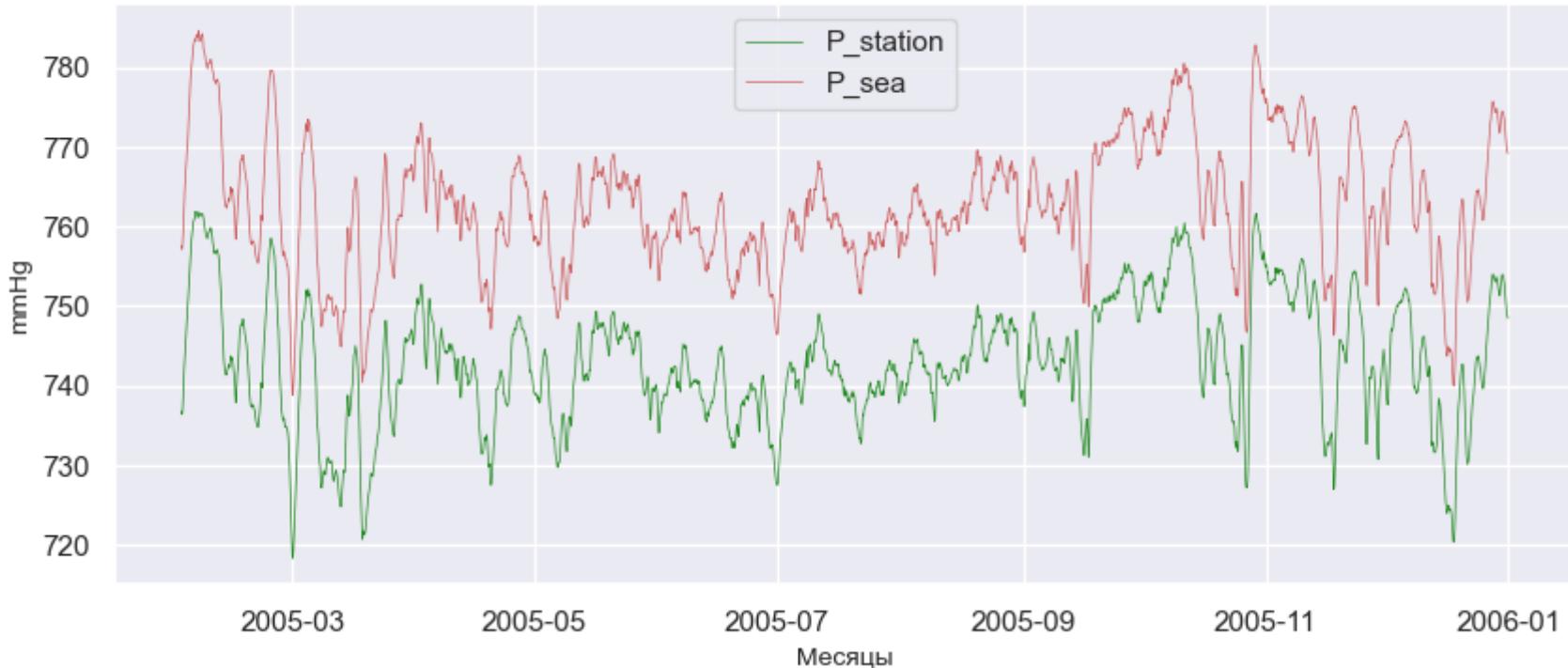
Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2007 год



Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2006 год



### Чашниково: Ежедневная динамика параметров P\_station и P\_sea, 2005 год



#### 4.2.4. Сохранение полученных данных в файлы

In [125...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER42}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER42}'  
  
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_'+PARAMETER42 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER42].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER42}.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_P_station.csv -> DONE!

```

## 4.3. Атмосферное давление: P\_drift (барическая тенденция)

Параметр барической тенденции P\_drift является вычисляемым между наблюдениями для каждой отдельной станции. Он не имеет значения для анализа выбросов и моделирования в искомой точки локации пос. Чашниково. Его можно просто вычислить. Тем не менее, для выявления возможных ошибок в наших расчётах полезно посмотреть, насколько барическая тенденция в исправленных значениях давления отличается от изначальной барической тенденции в архивах метеостанций.

### 4.3.1. Поиск и удаление ошибок показателя давления P\_drift

In [126...]

PARAMETER43 = 'P\_drift'

## Создаём временный DF для работы с параметром P\_drift

In [127...]

```
param_df_name = f'df_{PARAMETER43}' # преобразуем полученное значение в df_PARAMETER43 - ключ словаря dict_df_parameters
# Создадим временный df
df_tmp43 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp43.sample(5, random_state=56)
```

Out[127]:

	P_drift#V_Volochek	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#
2014-02-22 03:00:00	0.1	0.4	0.6	0.6	0.6	0.9	0.6	0.6	0.8
2015-05-16 03:00:00	-0.8	-1.0	6.3	-1.0	-0.8	-1.2	-0.8	-0.8	-0.4
2020-06-18 18:00:00	-0.5	-0.8	-0.5	-0.7	-0.7	NaN	NaN	NaN	NaN
2019-12-02 21:00:00	0.2	-0.3	0.0	-0.2	-0.4	NaN	NaN	NaN	NaN
2008-06-05 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Подсчитаем случаи, когда учтённая барическая тенденция отличается от разности исправленных значений давления за 3 часа

In [128...]

```
error_count = np.sum(np.array(df_tmp42.iloc[:, :-2] - df_tmp42.iloc[:, :-2].shift(-1)) != np.array(df_tmp43))

cells_count = df_tmp43.shape[0] * df_tmp43.shape[1]
print(f'В датафрейм всего {cells_count} значений\n'
      f'В том числе, {error_count} ошибочных значений, то есть {error_count / cells_count:.2%}')
    )
```

В датафрейм всего 608448 значений

В том числе, 567156 ошибочных значений, то есть 93.21%

Конечно, расхождения в значениях могут составлять и десятичные доли. Учитывая количество несоответствий, сразу перейдём к их исправлению и заполнению значений для локаций пос. Чашниково и центра поля метеостанций.

#### 4.3.2. Замена отсутствующих и несоответствующих значений барической тенденции P\_drift расчётными значениями : разница P\_station за 3 часа, расчёт показателя барической тенденции P\_drift для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

In [129...]

```
# Добавим в df_tmp43 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 PARAMETER43#Chashnikovo и PARAMETER43#Rfrnce_point
df_tmp43 = (df_tmp43.
             assign(col_name1 = np.nan,
                   col_name2 = np.nan).
             rename(columns={"col_name1": PARAMETER43+"#"+'Chashnikovo',
                           "col_name2": PARAMETER43+"#"+'Rfrnce_point'}))
)

df_tmp43.sample(3, random_state=56)
```

Out[129]:

	P_drift#V_Volochek	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#
2014-02-22 03:00:00	0.1	0.4	0.6	0.6	0.6	0.9	0.6	0.6	0.8
2015-05-16 03:00:00	-0.8	-1.0	6.3	-1.0	-0.8	-1.2	-0.8	-0.8	-0.4
2020-06-18 18:00:00	-0.5	-0.8	-0.5	-0.7	-0.7	NaN	NaN	NaN	NaN

In [130...]

```
# Добавим в архив параметров P_sea столбец для Чашниково и будущей центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями pr.nan
# Переименуем столбец PARAMETER43#Chashnikovo и PARAMETER43#Rfrnce_point
dict_df_parameters['df_'+PARAMETER43] = (dict_df_parameters['df_'+PARAMETER43].
```

```

        assign(col_name1 = np.nan,
               col_name2 = np.nan).
        rename(columns={"col_name1": PARAMETER43+'#'+ 'Chashnikovo',
                       "col_name2": PARAMETER43+'#'+ 'Rfrnce_point'}
                  )
    )

dict_df_parameters['df_'+PARAMETER43].sample(3, random_state=56)

```

Out[130]:

	P_drift#V_Volochek	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#
2014-02-22 03:00:00	0.1	0.4	0.6	0.6	0.6	0.9	0.6	0.8	
2015-05-16 03:00:00	-0.8	-1.0	6.3	-1.0	-0.8	-1.2	-0.8	-0.4	
2020-06-18 18:00:00	-0.5	-0.8	-0.5	-0.7	-0.7	NaN	NaN	NaN	

In [131...]

```
# Добавим в архив метеостанций df Чашниково и df для центральной точки,
# Создадим в нём столбец с названием PARAMETER43
```

```

dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER43}
                                                 )
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER43}
                                                 )
                                         )

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

```

Out[131]:

	T	T_min	T_max	P_sea	P_station	P_drift
<b>2014-02-22 03:00:00</b>	-4.156558	-4.156558	-2.109643	768.336709	747.597791	NaN
<b>2015-05-16 03:00:00</b>	9.181730	9.181730	10.434571	745.095535	725.921747	NaN
<b>2020-06-18 18:00:00</b>	27.439052	25.366130	30.241632	761.480701	743.060948	NaN

Out[131]:

	T	T_min	T_max	P_sea	P_station	P_drift
<b>2014-02-22 03:00:00</b>	-3.589183	-3.794602	-2.700734	767.373725	754.041734	NaN
<b>2015-05-16 03:00:00</b>	7.789650	7.789650	8.797603	746.708428	734.256500	NaN
<b>2020-06-18 18:00:00</b>	29.404954	25.572651	29.941094	760.796222	749.008692	NaN

### Пересчёт барической тенденции, исходя из исправленных значений давления на уровне станции

In [132...]

```
# Сохраним индекс столбцов для df_tmp43
idx_cols43 = df_tmp43.columns
# Переопределим значения для df_tmp43 (это приведёт к замене названий столбцов)
df_tmp43 = (df_tmp42 - df_tmp42.shift(-1))
```

In [133...]

```
df_tmp43.columns = list(idx_cols43) # восстановим названия столбцов из сохранённого индекса столбцов
df_tmp43.sample(5, random_state=56)
```

Out[133]:

	P_drift#V_Volochek	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#
<b>2014-02-22 03:00:00</b>	0.100000	0.4	0.600000	0.600000	0.6	0.9	0.6	0.8	
<b>2015-05-16 03:00:00</b>	-0.800000	-1.0	-1.200000	-0.931922	-0.8	-1.2	-0.8	-0.4	
<b>2020-06-18 18:00:00</b>	-0.500000	-0.8	-0.500000	-0.700000	-0.7	-0.5	-0.7	-0.9	
<b>2019-12-02 21:00:00</b>	0.200000	-0.3	0.000000	-0.197698	-0.4	-0.4	-0.5	-0.5	
<b>2008-06-05 18:00:00</b>	0.139666	-1.0	-1.354662	-0.986819	-0.7	-0.5	-0.6	-0.5	



### Перезапись архивов в части показателя P\_drift

In [134...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER43] = df_tmp43.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[:, PARAMETER43] = df_tmp43.loc[:, PARAMETER43 + '#' + name_df[3:]]

# Выведем случайные ряды из архивов
dict_df_parameters['df_'+PARAMETER43].sample(7, random_state=56)
dict_df_locations['df_Chashnikovo'].sample(7, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(7, random_state=56)
```

Out[134]:	P_drift#V_Volochek	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#Lomonosov
<b>2014-02-22 03:00:00</b>	0.100000	0.4	0.600000	0.600000	0.6	0.9	0.6	0.8	
<b>2015-05-16 03:00:00</b>	-0.800000	-1.0	-1.200000	-0.931922	-0.8	-1.2	-0.8	-0.4	
<b>2020-06-18 18:00:00</b>	-0.500000	-0.8	-0.500000	-0.700000	-0.7	-0.5	-0.7	-0.9	
<b>2019-12-02 21:00:00</b>	0.200000	-0.3	0.000000	-0.197698	-0.4	-0.4	-0.5	-0.5	
<b>2008-06-05 18:00:00</b>	0.139666	-1.0	-1.354662	-0.986819	-0.7	-0.5	-0.6	-0.5	
<b>2016-10-20 06:00:00</b>	-0.200000	-0.1	-0.200000	-1.272800	-0.2	-0.3	-0.2	-0.2	
<b>2006-09-11 03:00:00</b>	0.089440	1.0	0.120969	0.971137	0.9	1.3	0.9	1.0	

Out[134]:	T	T_min	T_max	P_sea	P_station	P_drift
<b>2014-02-22 03:00:00</b>	-4.156558	-4.156558	-2.109643	768.336709	747.597791	0.812775
<b>2015-05-16 03:00:00</b>	9.181730	9.181730	10.434571	745.095535	725.921747	-1.043037
<b>2020-06-18 18:00:00</b>	27.439052	25.366130	30.241632	761.480701	743.060948	-0.492627
<b>2019-12-02 21:00:00</b>	-3.004683	-4.229056	-2.673631	758.773486	738.378842	-0.577660
<b>2008-06-05 18:00:00</b>	16.565949	12.255838	16.565949	760.858054	741.771366	-0.440787
<b>2016-10-20 06:00:00</b>	0.117681	0.074269	0.916309	775.972222	755.350419	-0.276710
<b>2006-09-11 03:00:00</b>	9.855422	9.855422	11.886722	761.694029	742.139162	1.174445

Out[134]:

	T	T_min	T_max	P_sea	P_station	P_drift
<b>2014-02-22 03:00:00</b>	-3.589183	-3.794602	-2.700734	767.373725	754.041734	0.616282
<b>2015-05-16 03:00:00</b>	7.789650	7.789650	8.797603	746.708428	734.256500	-0.826706
<b>2020-06-18 18:00:00</b>	29.404954	25.572651	29.941094	760.796222	749.008692	-0.723724
<b>2019-12-02 21:00:00</b>	-2.977448	-3.796107	-2.513436	758.016264	744.876404	-0.402065
<b>2008-06-05 18:00:00</b>	17.613615	11.754644	17.613615	761.523291	749.249895	-0.789045
<b>2016-10-20 06:00:00</b>	-0.382597	-0.369287	0.885072	775.916024	762.592727	-0.296078
<b>2006-09-11 03:00:00</b>	9.181940	9.181940	11.628949	762.828050	750.169523	0.987633

Выведем, рандомные строки из df\_tmp42

In [135...]

```
df_tmp43.sample(7)
```

Out[135]:

	P_drift#V_Volochek	P_drift#Staritsa	P_drift#Kashyn	P_drift#Tver	P_drift#Klin	P_drift#Dmitrov	P_drift#Volokolamsk	P_drift#Mozhaisk	P_drift#Lomonosov
<b>2011-01-21 09:00:00</b>	-0.732028	0.2	1.217702	0.455685	0.5	0.2	0.2	0.1	
<b>2019-11-11 06:00:00</b>	-0.800000	-0.8	-0.600000	-1.376529	-0.4	-1.1	-0.7	-0.7	-0.7
<b>2010-04-22 00:00:00</b>	-2.600000	-1.9	-1.700000	-1.800000	-1.3	-1.3	-1.2	-1.1	
<b>2013-01-18 21:00:00</b>	-0.500000	-0.2	-0.300000	0.740188	-0.1	-0.1	0.0	0.1	
<b>2008-09-07 21:00:00</b>	-2.048198	-0.3	-0.534784	1.330085	-0.1	-0.2	-0.2	0.0	0.0
<b>2011-01-31 21:00:00</b>	-1.800000	-1.1	-0.600000	-0.648200	-0.1	-0.2	0.0	-0.1	
<b>2018-03-07 06:00:00</b>	-0.700000	-0.9	-1.000000	-0.900000	-0.9	-0.9	-0.8	-1.5	

In [136...]

```
# # 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)
```

### Сохраним очищенные данные в файл параметров

```
# Запишем текущие данные в файл
print('df_'+PARAMETER43 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER43].to_csv(
    path_or_buf=f'{clean_path}df_{PARAMETER43}.csv'
)
print('DONE!')
```

df\_P\_drift.csv -> DONE!

### 4.3.3. Визуализация графиков барической тенденции P\_drift для условной метеостанции Чашниково

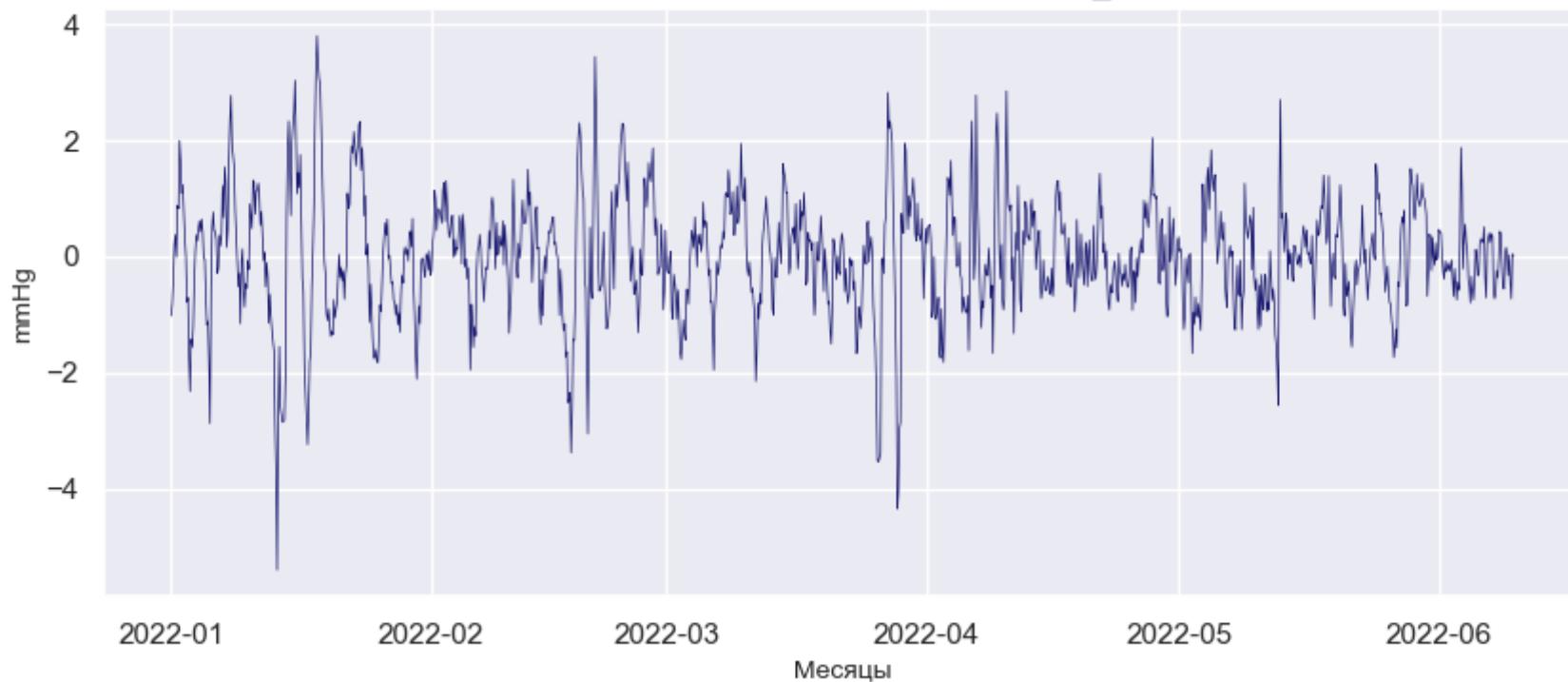
In [137...]

```
for year in list_years:
    # Возьмём параллельно данные о давлении на уровне станции и на уровне моря
    data1 = dict_df_parameters['df_P_drift'][PARAMETER43+"#"+"Chashnikovo"]\n    [(dict_df_parameters['df_P_drift'].index.year == year)]

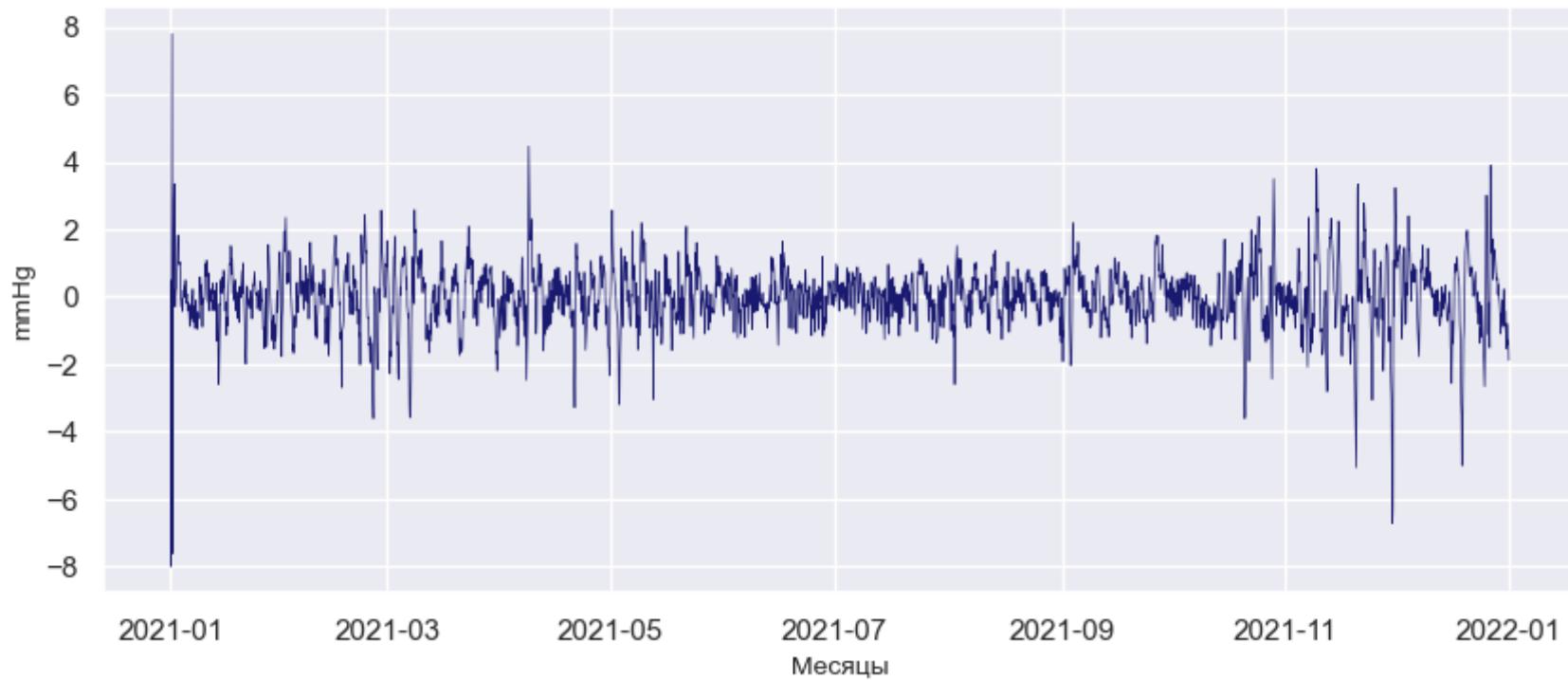
    fig, ax = plt.subplots(figsize=(10, 4))
    g1 = sns.lineplot(data=data1,
                      color='midnightblue',
                      linewidth=0.5,
                      ax=ax)

    dummy = ax.set_ylabel('mmHg', size=10)
    dummy = ax.set_xlabel('Месяцы', size=10)
    dummy = plt.title(f'Чашниково: Ежедневная динамика параметра {PARAMETER43}, {year} год')
    plt.show()
```

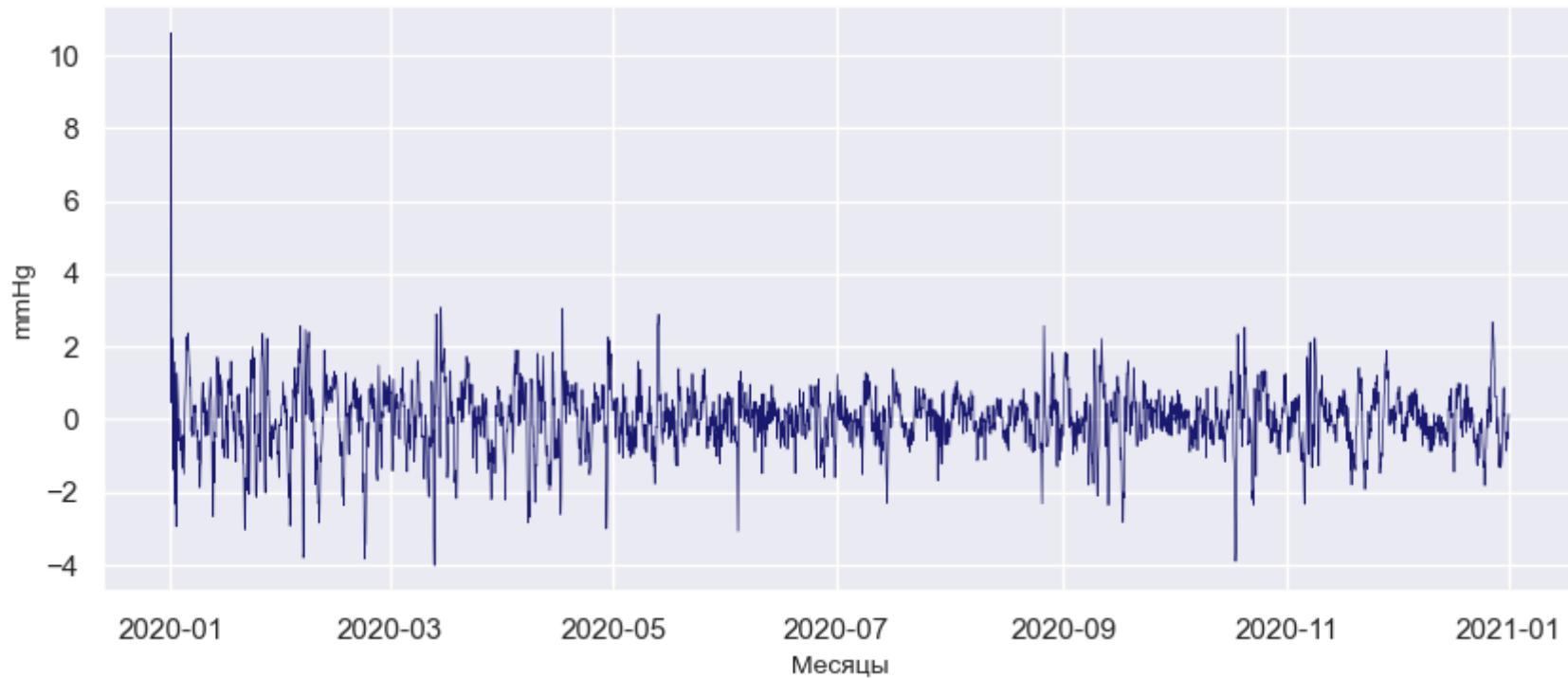
### Чашниково: Ежедневная динамика параметра P\_drift, 2022 год



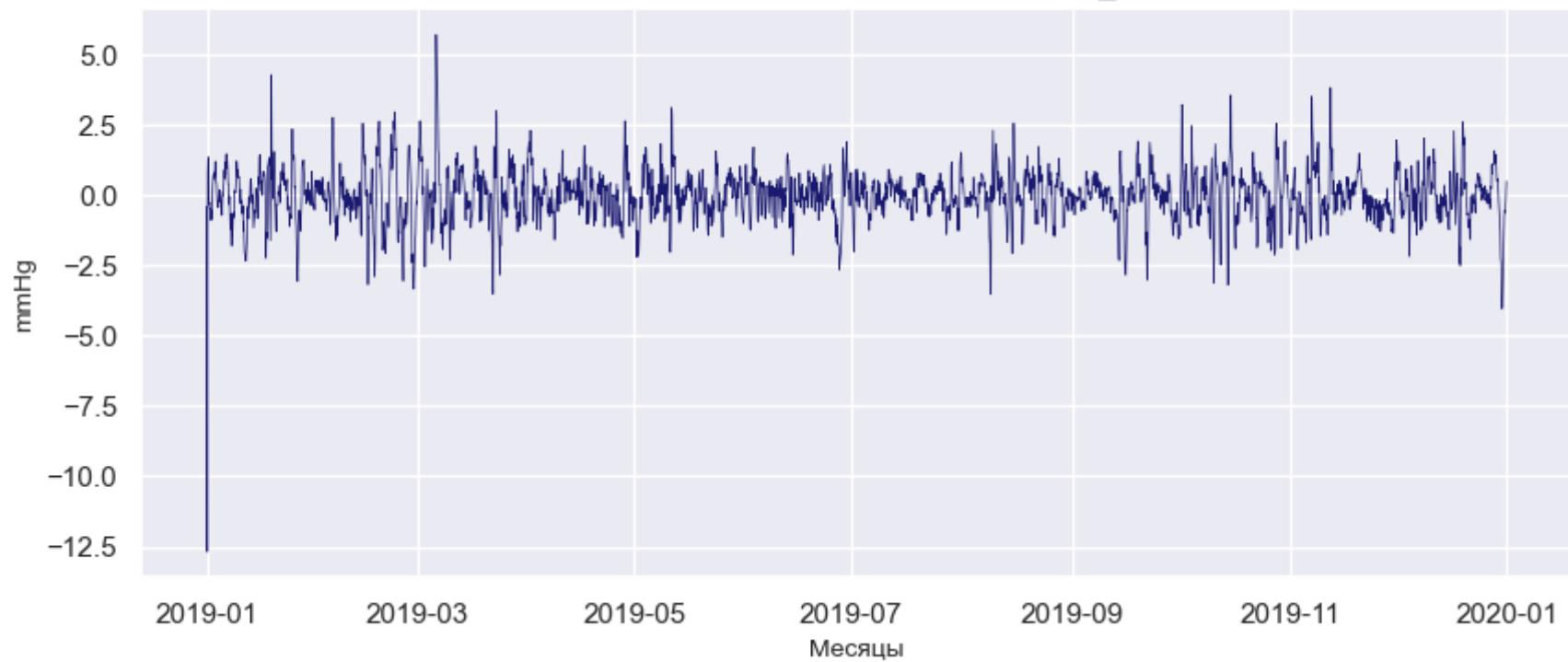
Чашниково: Ежедневная динамика параметра P\_drift, 2021 год



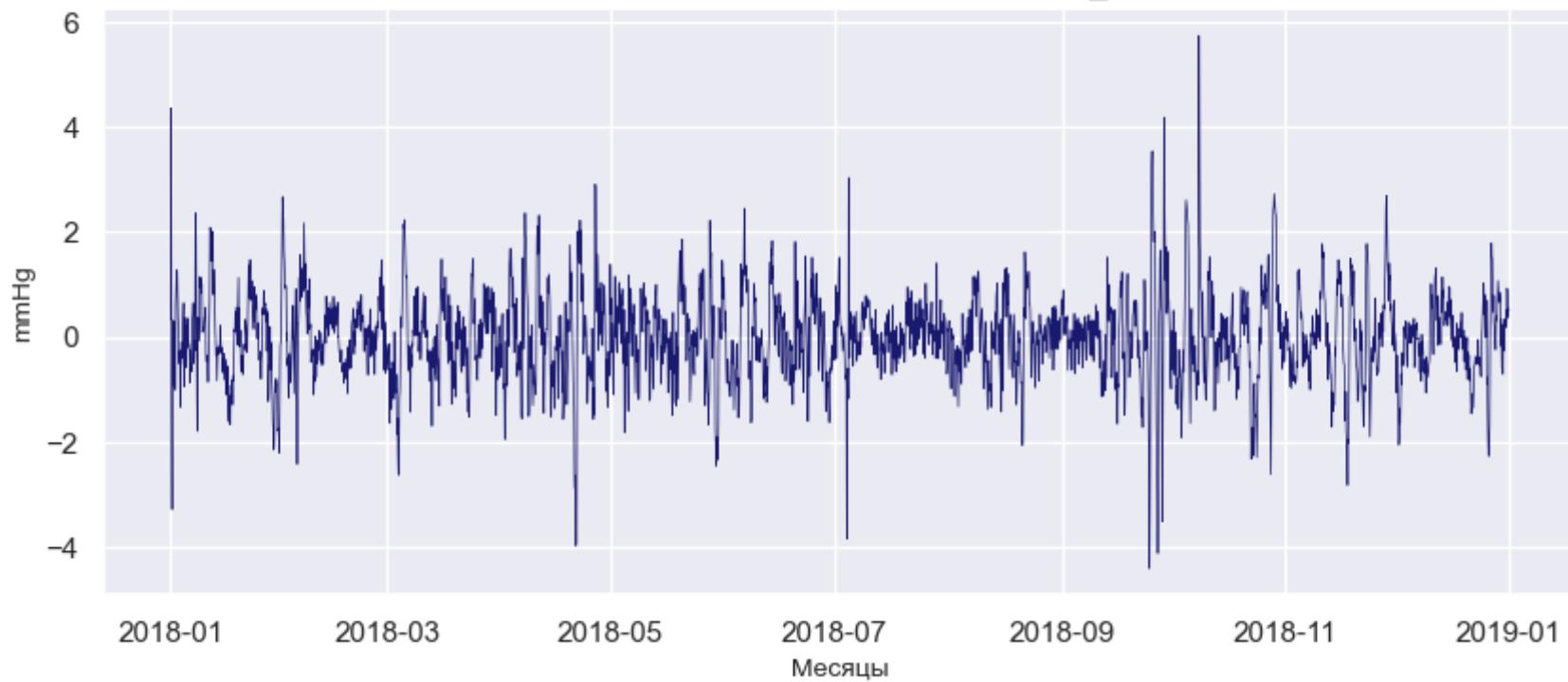
Чашниково: Ежедневная динамика параметра P\_drift, 2020 год



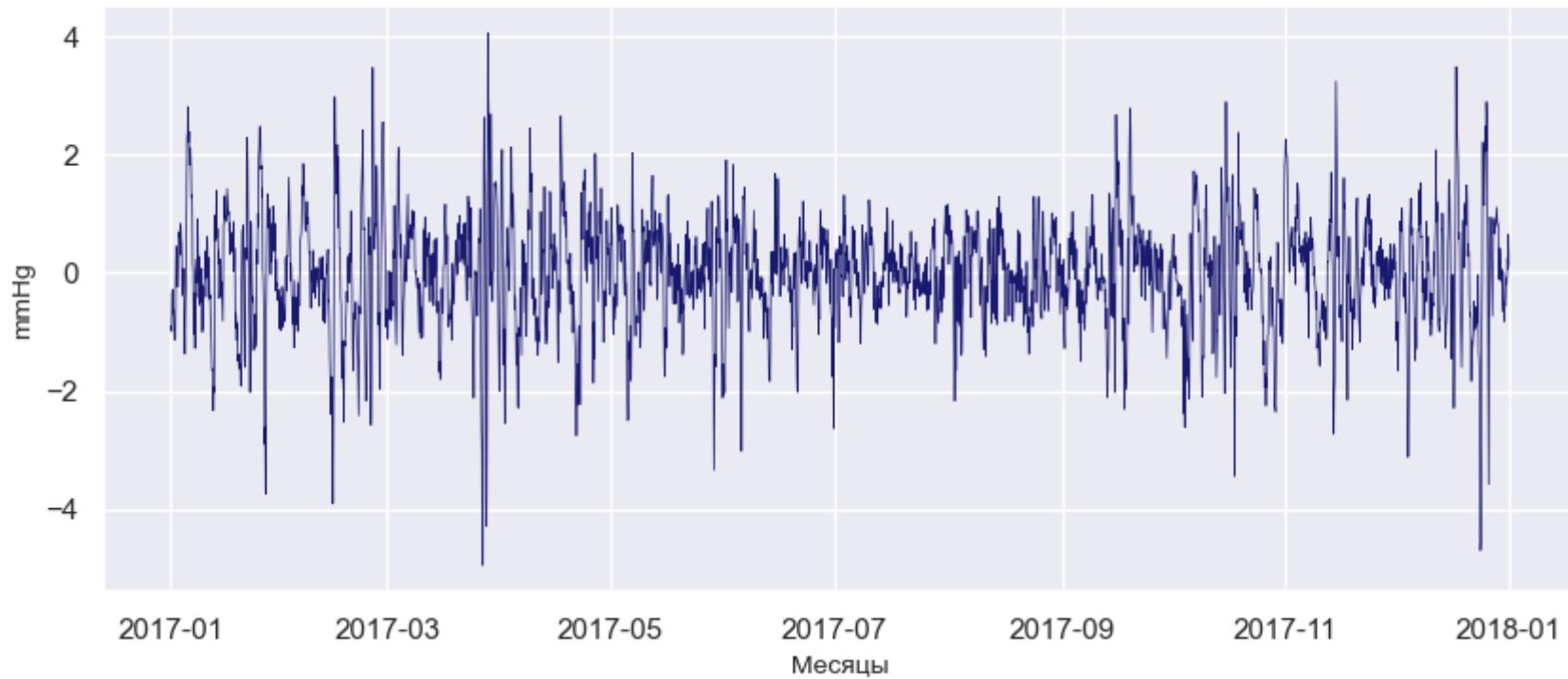
Чашниково: Ежедневная динамика параметра P\_drift, 2019 год



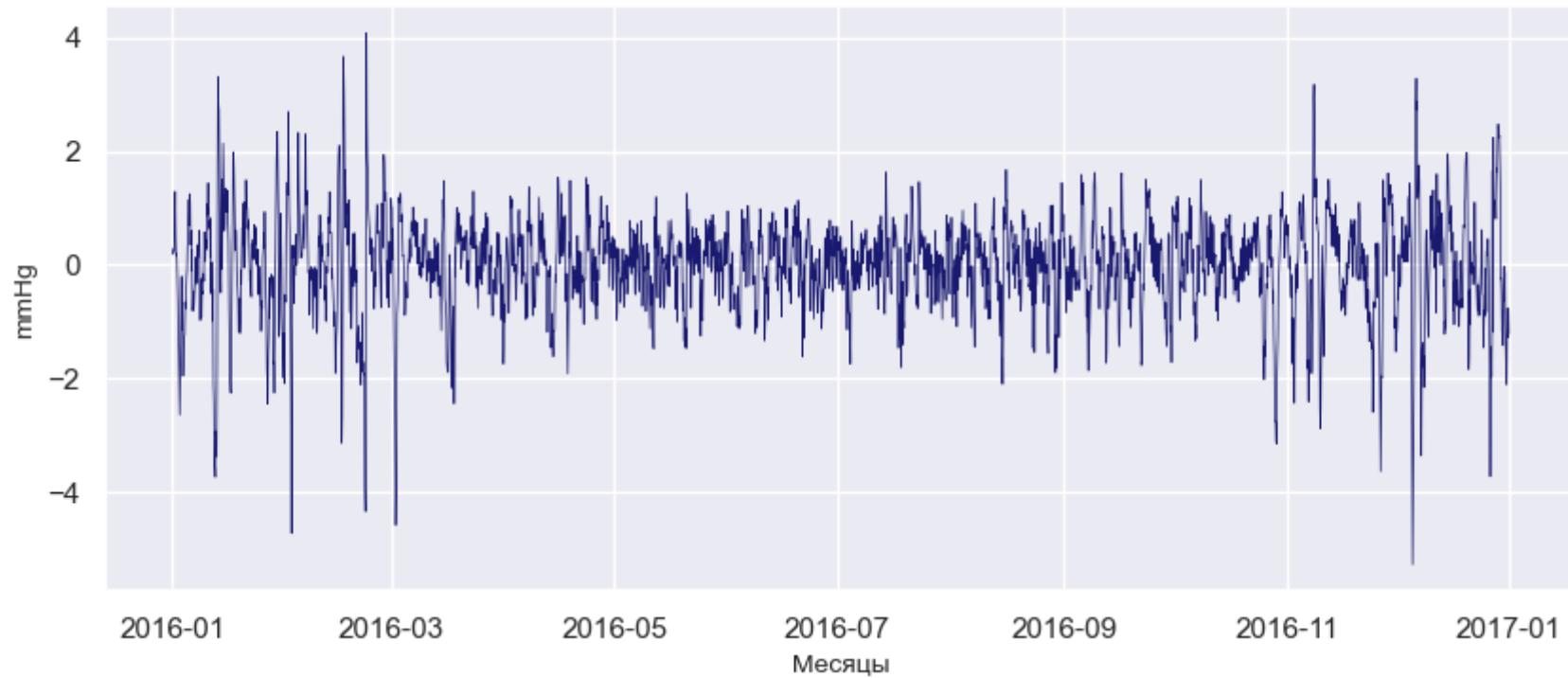
Чашниково: Ежедневная динамика параметра P\_drift, 2018 год



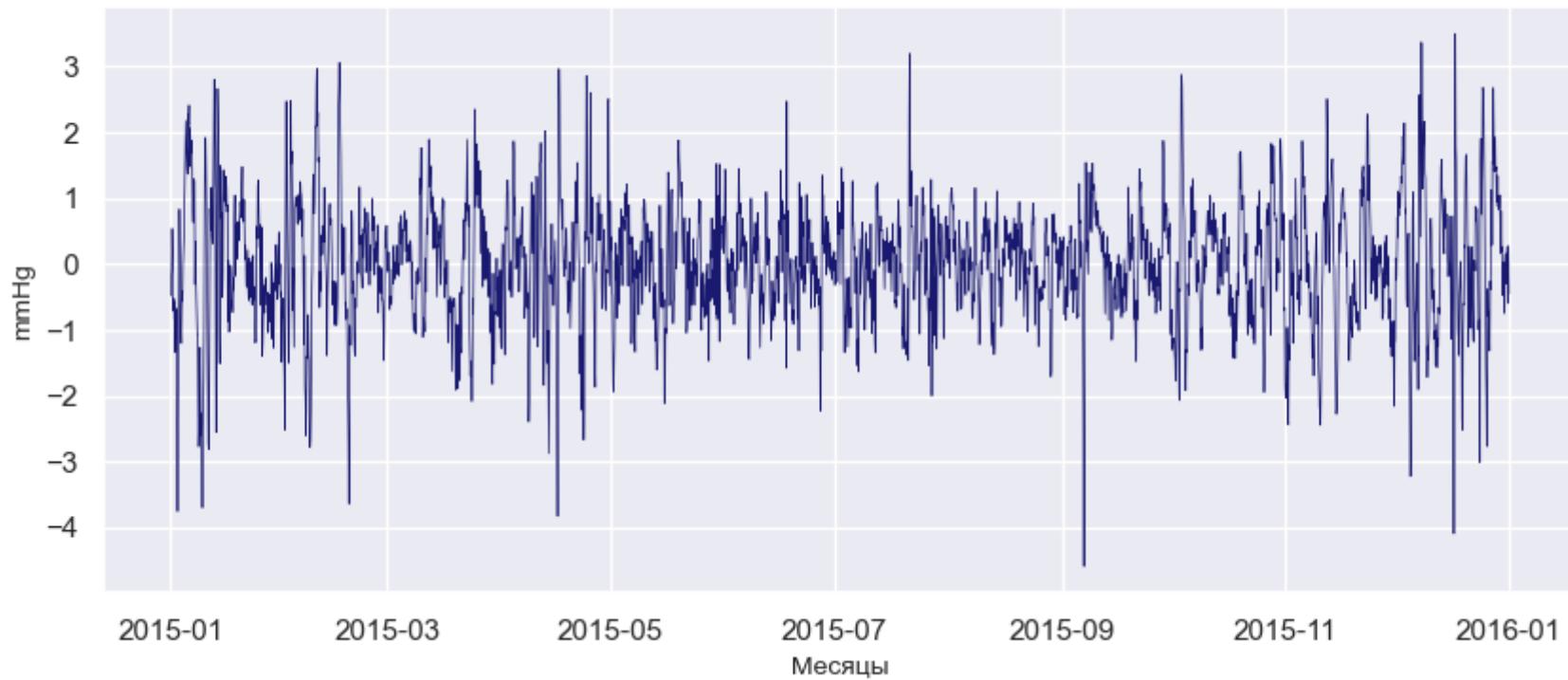
### Чашниково: Ежедневная динамика параметра P\_drift, 2017 год



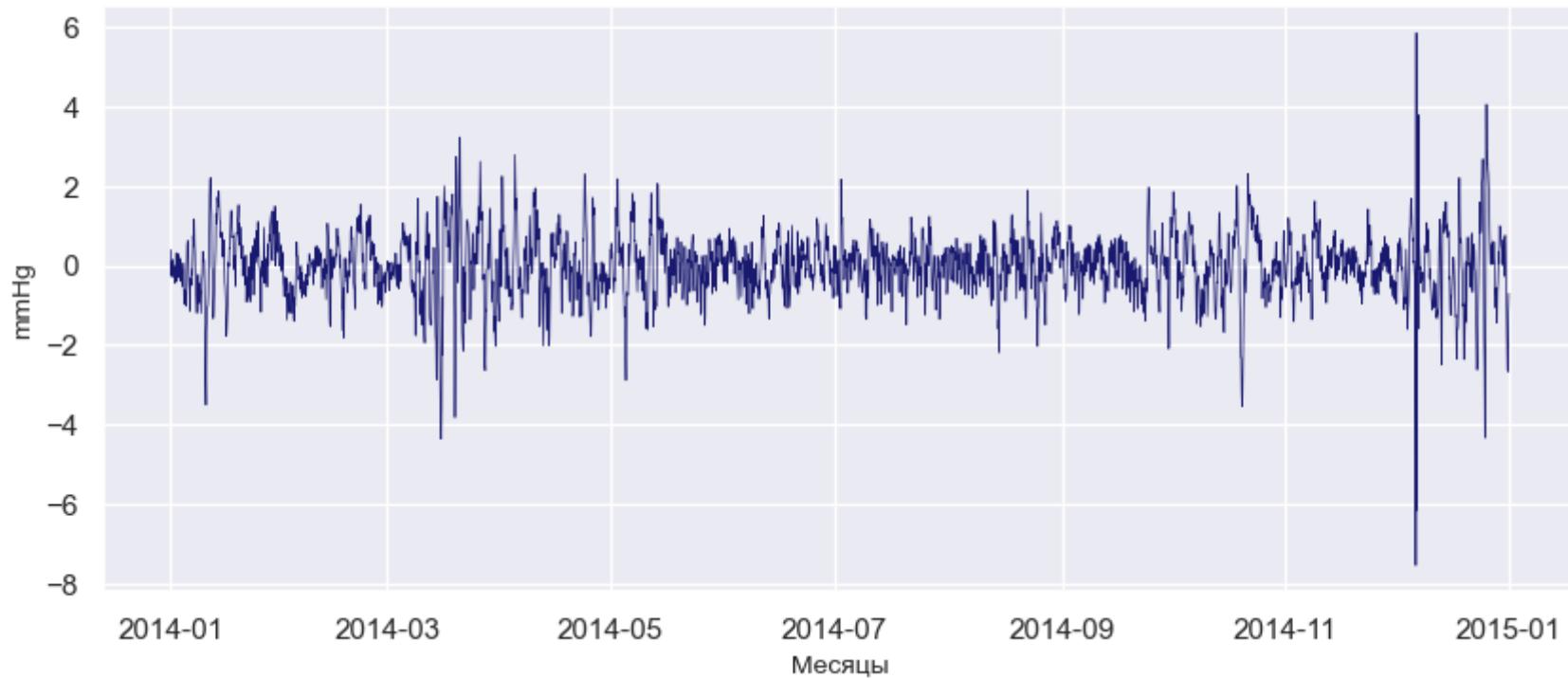
Чашниково: Ежедневная динамика параметра P\_drift, 2016 год



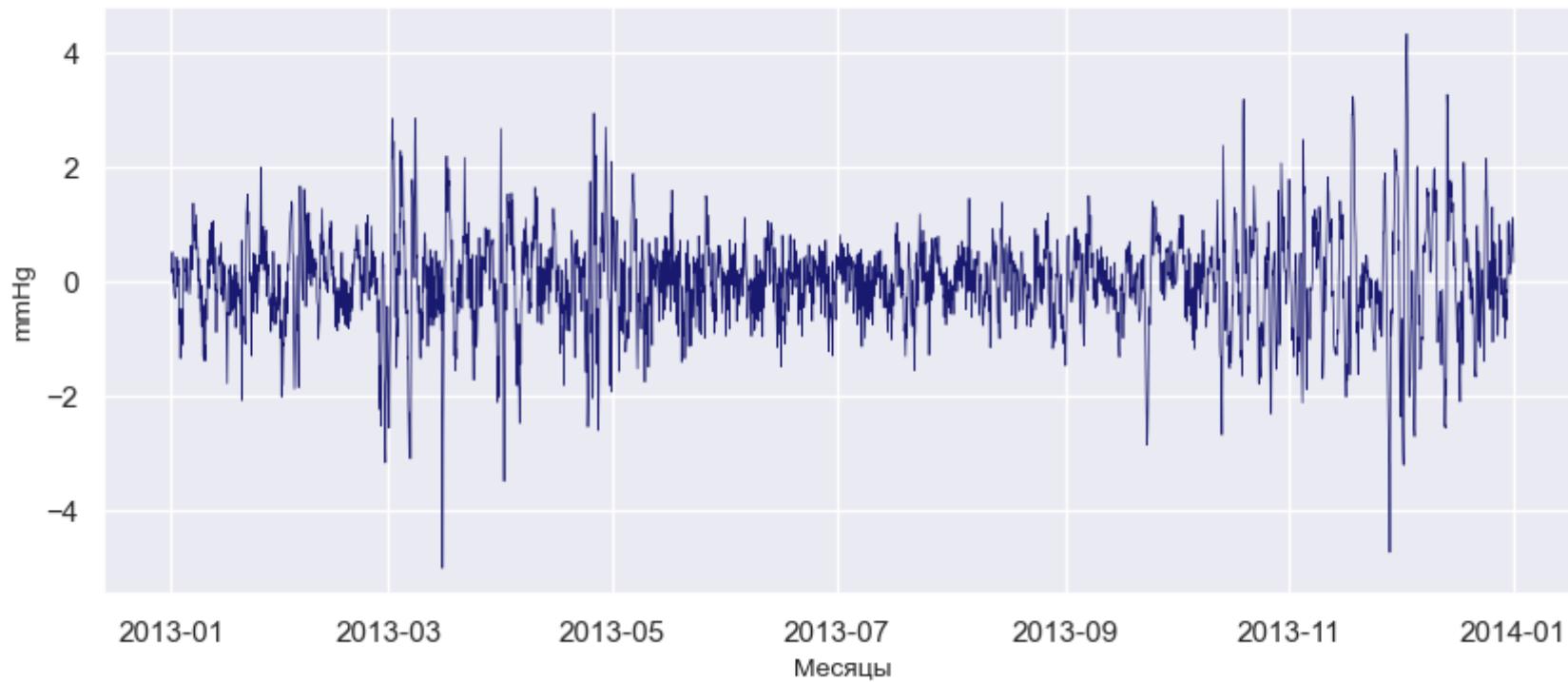
Чашниково: Ежедневная динамика параметра P\_drift, 2015 год



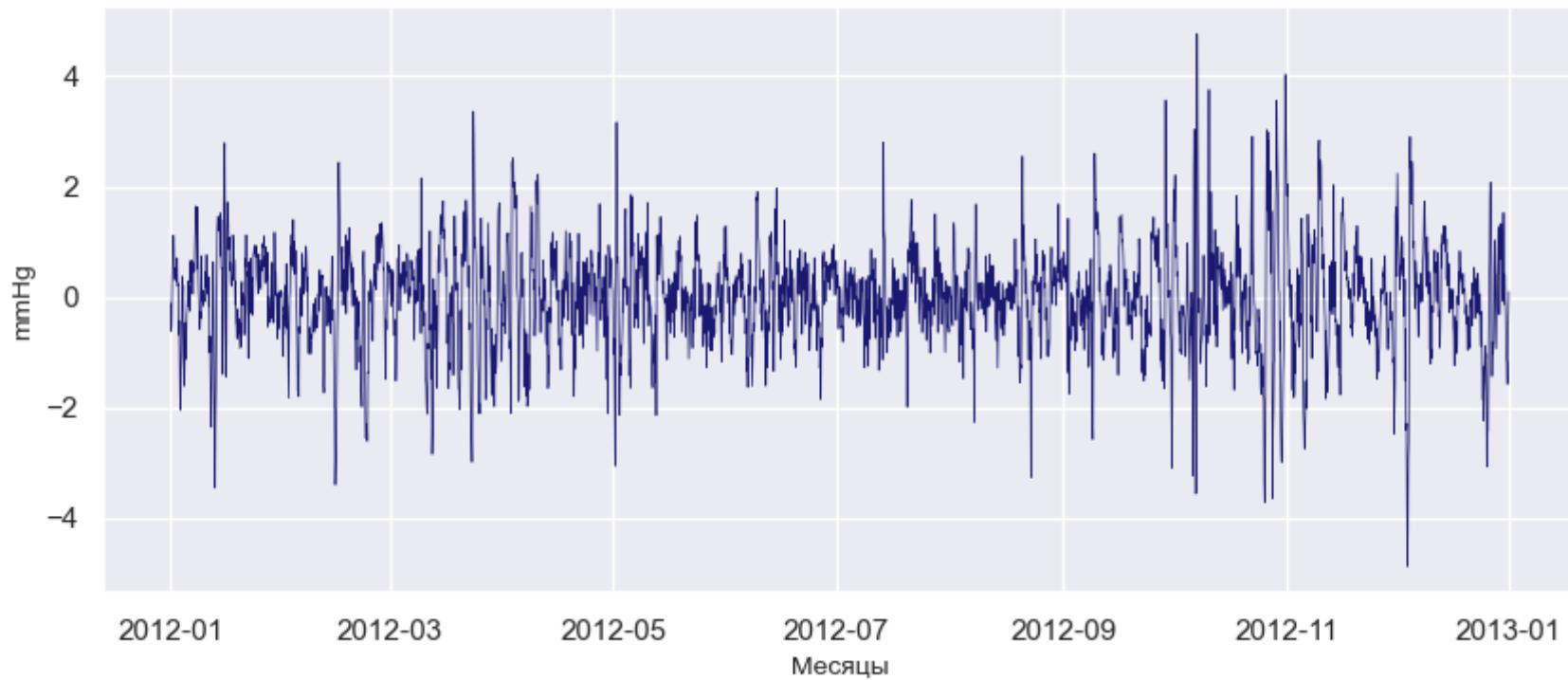
Чашниково: Ежедневная динамика параметра P\_drift, 2014 год



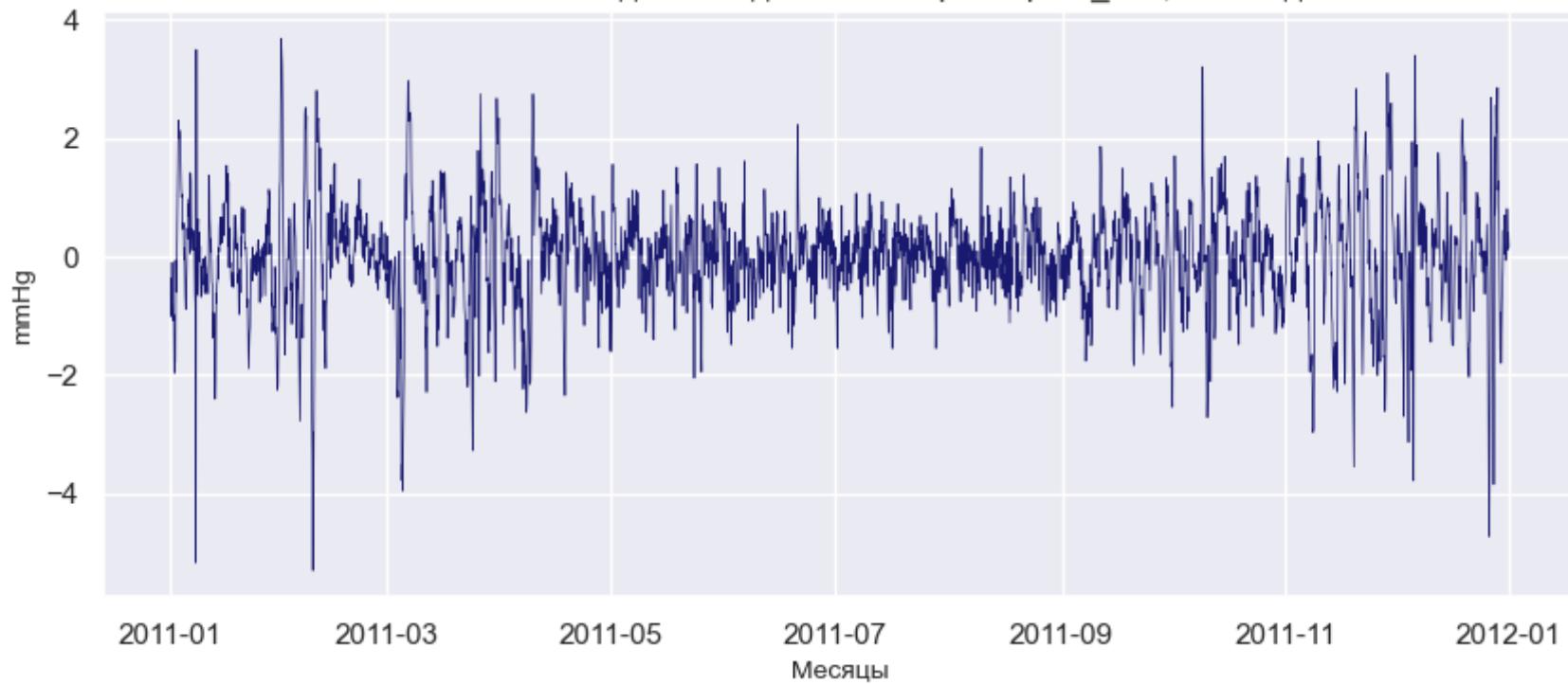
Чашниково: Ежедневная динамика параметра P\_drift, 2013 год



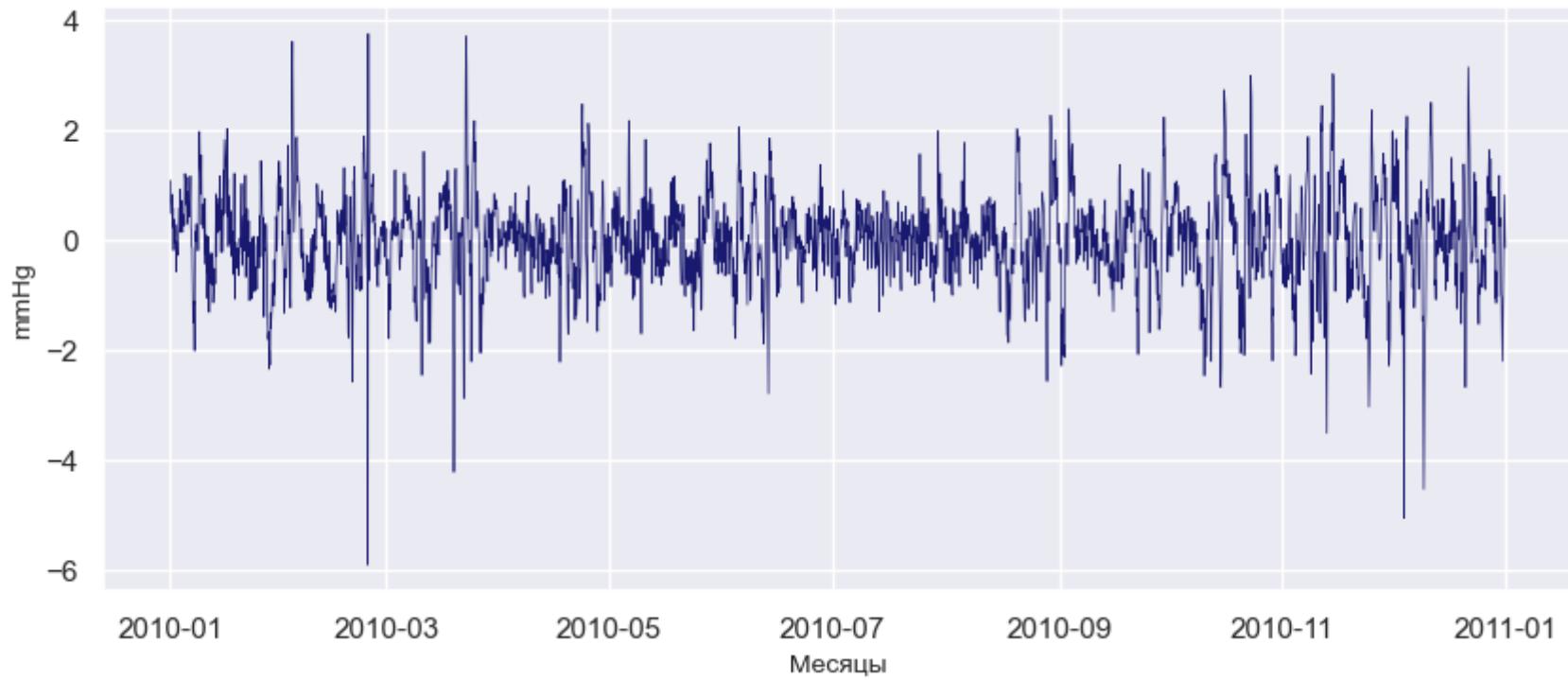
Чашниково: Ежедневная динамика параметра P\_drift, 2012 год



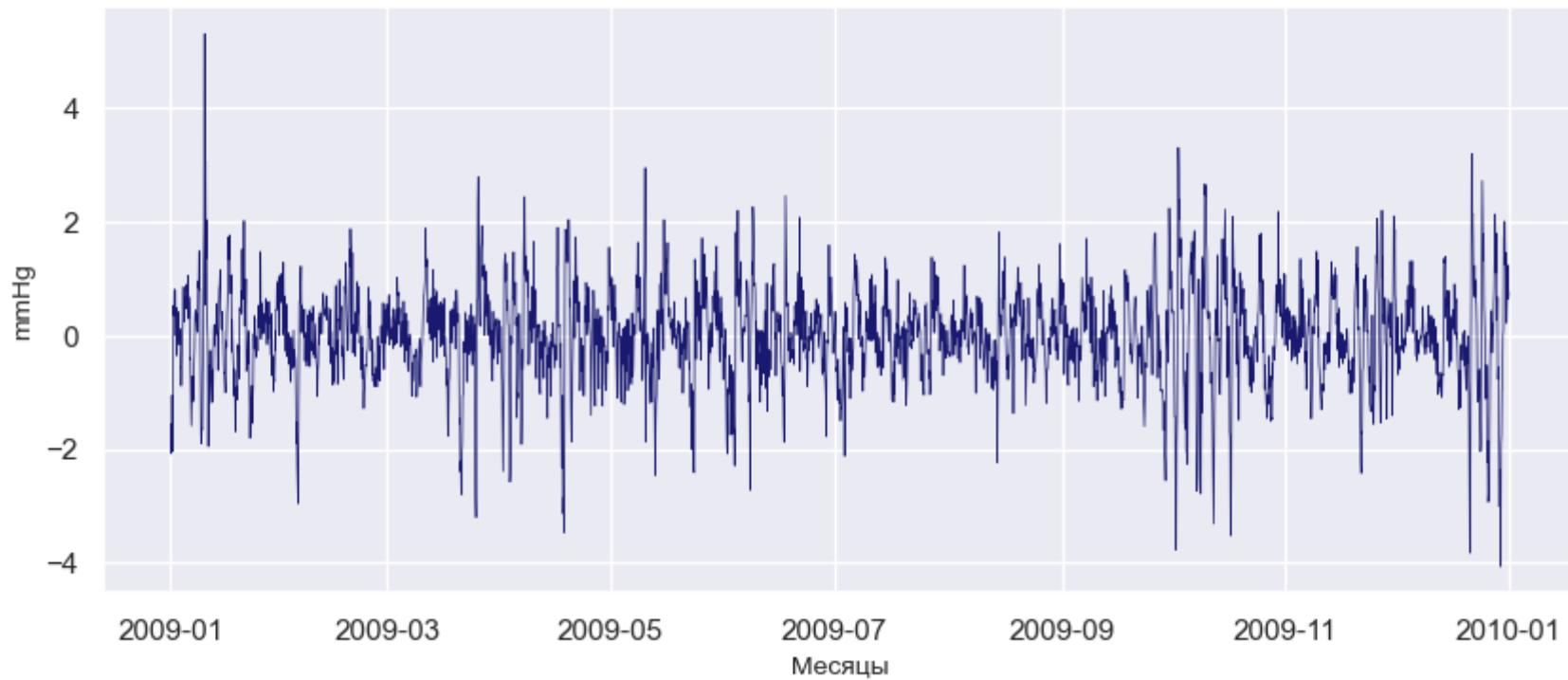
Чашниково: Ежедневная динамика параметра P\_drift, 2011 год



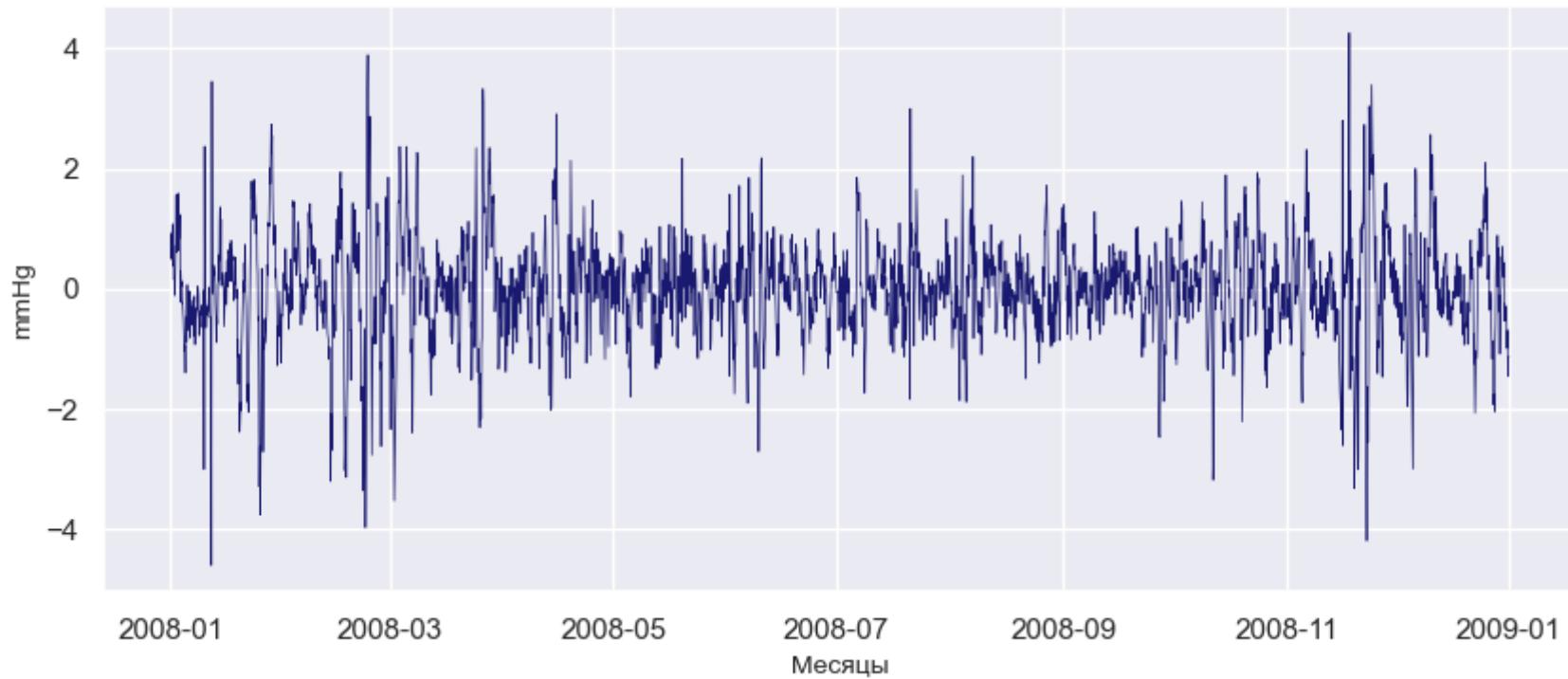
Чашниково: Ежедневная динамика параметра P\_drift, 2010 год



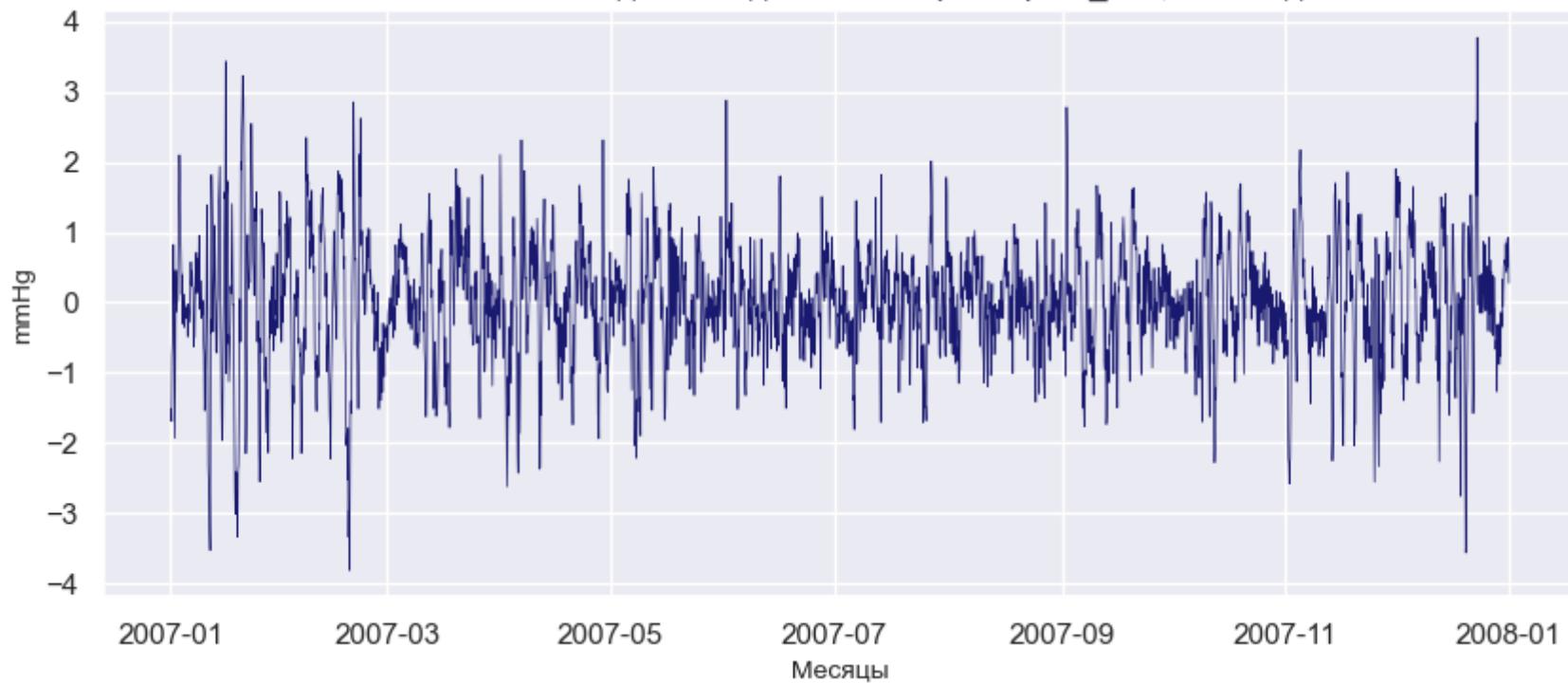
Чашниково: Ежедневная динамика параметра P\_drift, 2009 год



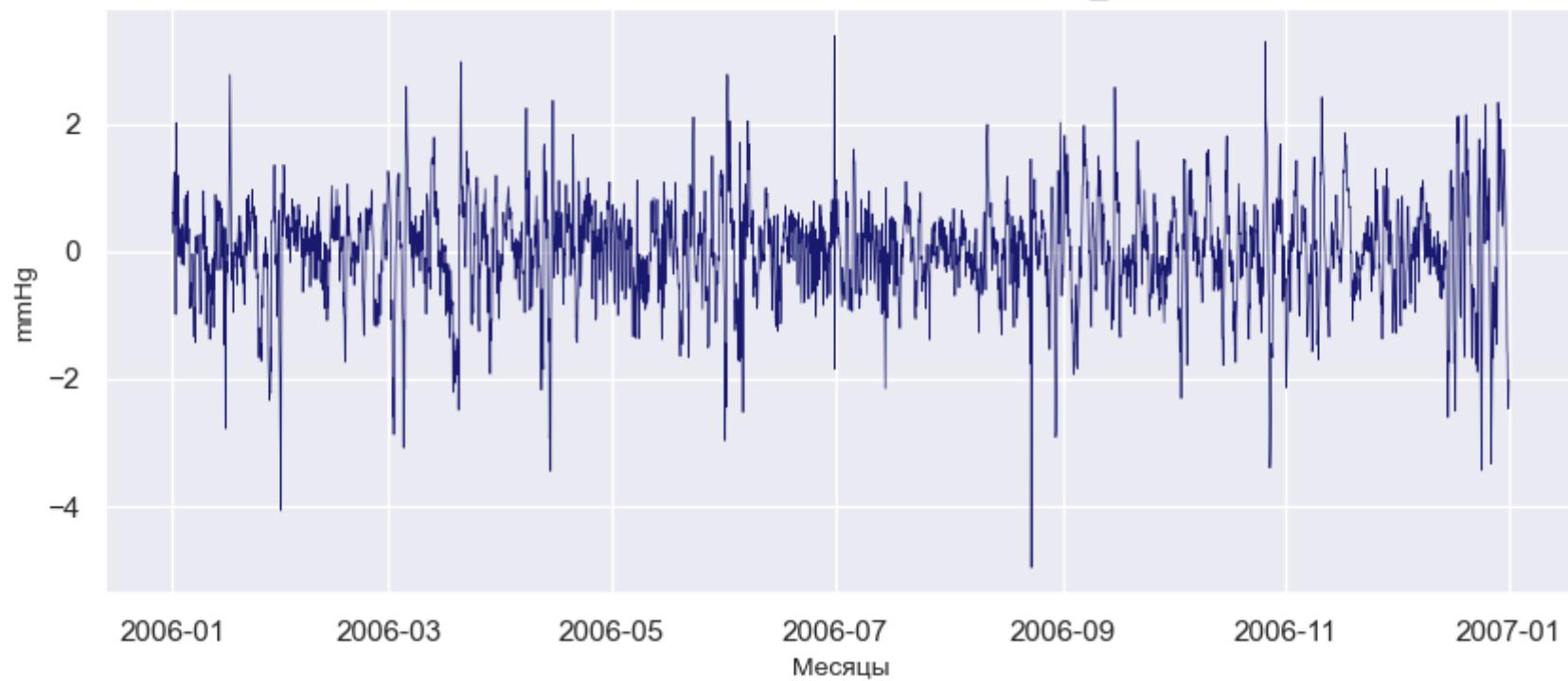
Чашниково: Ежедневная динамика параметра P\_drift, 2008 год



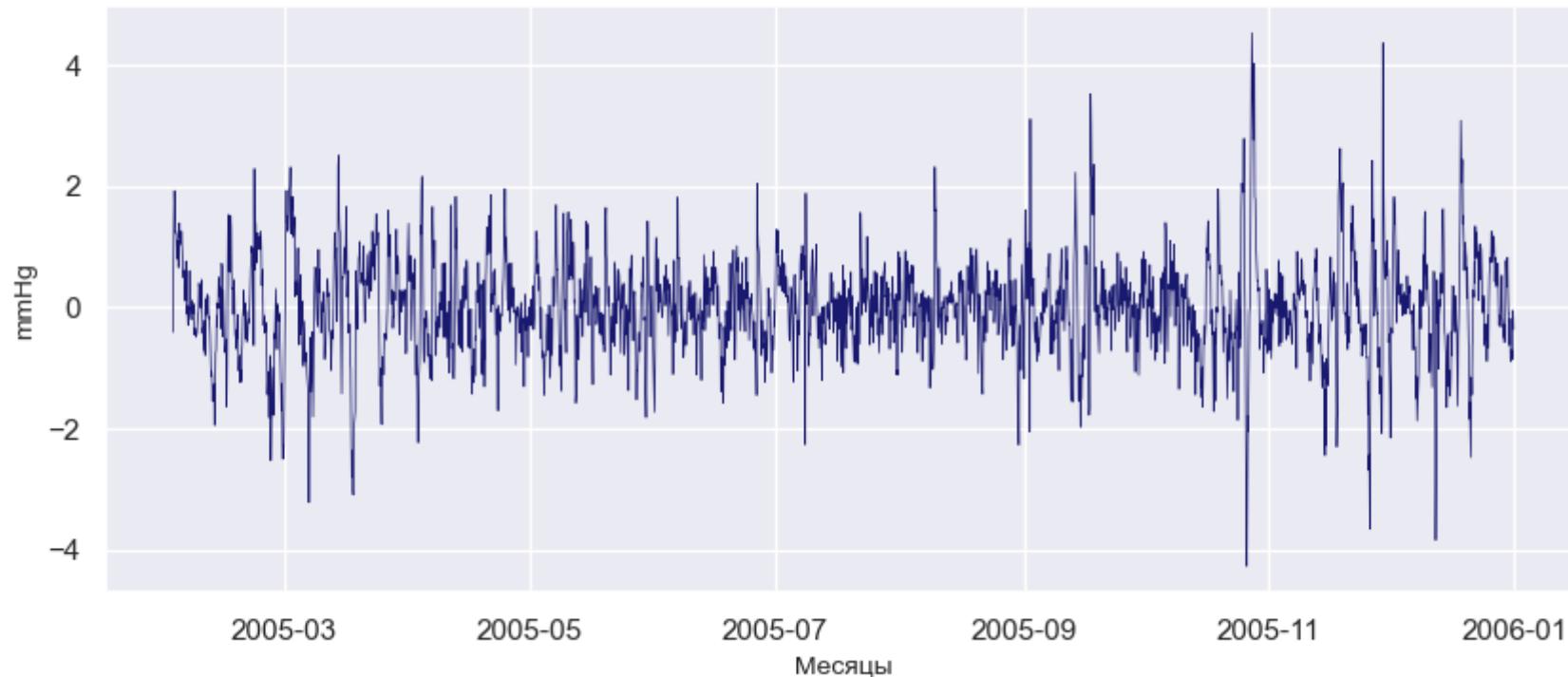
Чашниково: Ежедневная динамика параметра P\_drift, 2007 год



Чашниково: Ежедневная динамика параметра P\_drift, 2006 год



### Чашниково: Ежедневная динамика параметра P\_drift, 2005 год



#### 4.3.4. Сохранение полученных данных в файлы

In [138...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER43}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER43}'  
  
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_'+PARAMETER43 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER43].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER43}.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\_P\_drift.csv -> DONE!

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

Определим функцию рассчёта температуры точки росы

```
In [139...]: def dew_point_calculator(temp_=20, humid_=50):
    """
    Функция расчёта точки росы по температуре и относительной влажности
    Принимает:
    - значение температуры в градусах С,

```

```
- значение относительной влажности воздуха в %
Возвращает:
- температуру точки росы в градусах С
"""

#     print(temp_, humid_)
gamma_ = (17.27 * temp_) / ( 237.71 + temp_) + log(humid_/100)
dew_temp_ = (237.71 * gamma_) / (17.27 - gamma_)
return dew_temp_
```

## 5.1. Концентрация водяных паров в воздухе: относительная влажность воздуха (Humid)

### 5.1.1. Поиск и удаление ошибок показателя относительной влажности воздуха (Humid)

Для данного раздела обозначим константу названия параметра

In [140...]: PARAMETER51 = 'Humid'

**Создаём временный DF для работы с параметром Humid**

In [141...]: param\_df\_name = f'df\_{PARAMETER51}' # преобразуем полученное значение в df\_PARAMETER51 - ключ словаря dict\_df\_parameters  
# Создадим временный df  
df\_tmp51 = dict\_df\_parameters[param\_df\_name].copy(deep=True)  
df\_tmp51.sample(5, random\_state=56)

Out[141]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#...
<b>2014-02-22 03:00:00</b>	75.0	75.0	73.0	75.0	79.0	77.0	74.0	77.0	
<b>2015-05-16 03:00:00</b>	93.0	93.0	96.0	94.0	95.0	96.0	94.0	92.0	
<b>2020-06-18 18:00:00</b>	47.0	46.0	64.0	48.0	62.0	61.0	35.0	42.0	
<b>2019-12-02 21:00:00</b>	88.0	91.0	88.0	81.0	88.0	96.0	90.0	83.0	
<b>2008-06-05 18:00:00</b>	Nan	40.0	Nan	42.0	43.0	43.0	47.0	40.0	



Определим последовательность действий, для поиска ошибок показателя влажности

Изменения влажности могут иметь резкие колебания: они могут быть связаны с выпадением осадков и последующим испарением влаги с земной поверхности, а также с колебаниями температуры воздуха и атмосферного давления. Локальное выпадение осадков может сопровождаться резким увеличением влажности, как в поле метеостанций, так и во временном окне.

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

### Визуализируем архив относительной влажности воздуха (Humid) по сезонам

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

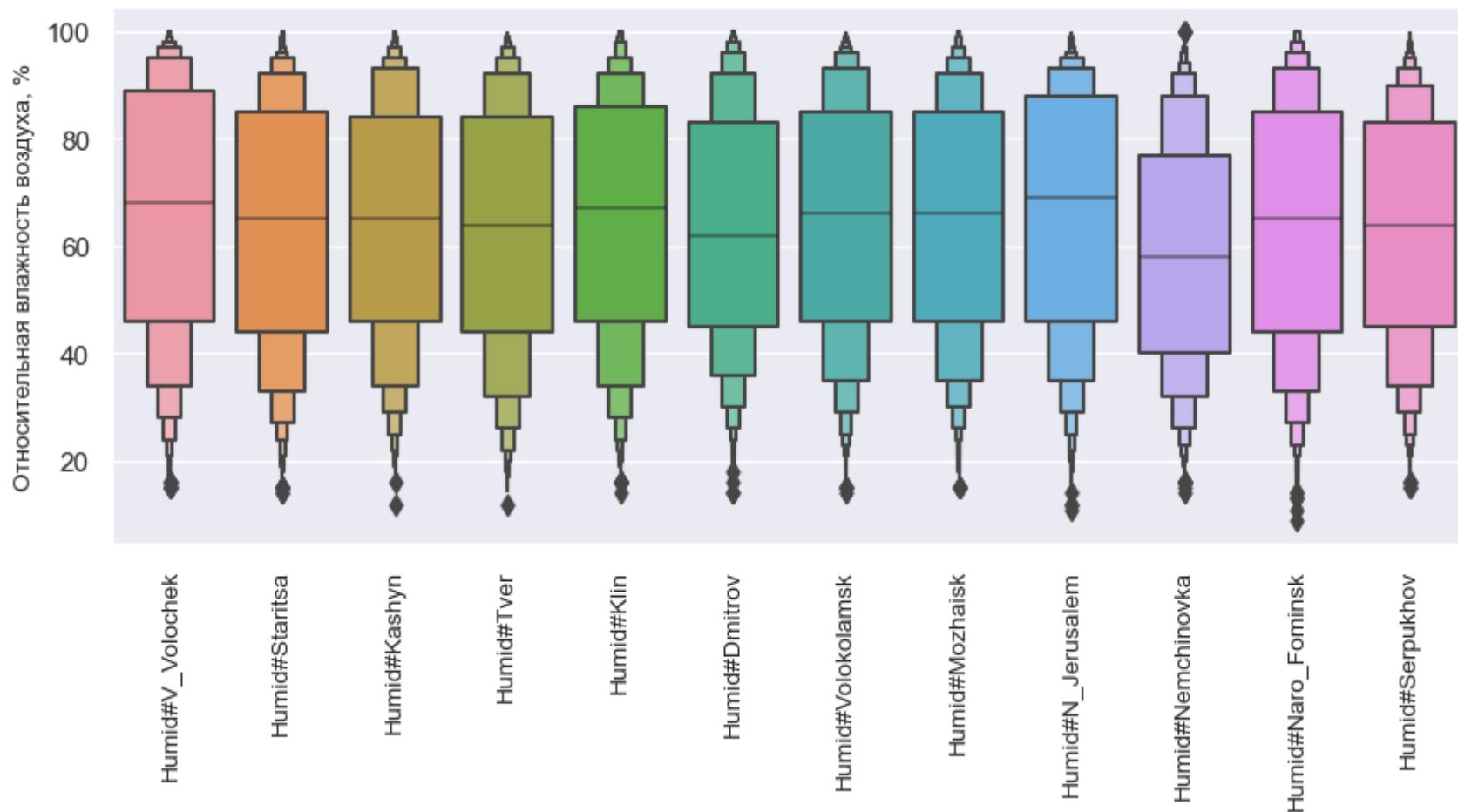
- Зима (ниже 0°): В среднем длится с 5 ноября по 4 апреля
- Весна (от 0° до +10°): В среднем длится с 5 апреля по 18 мая

- 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

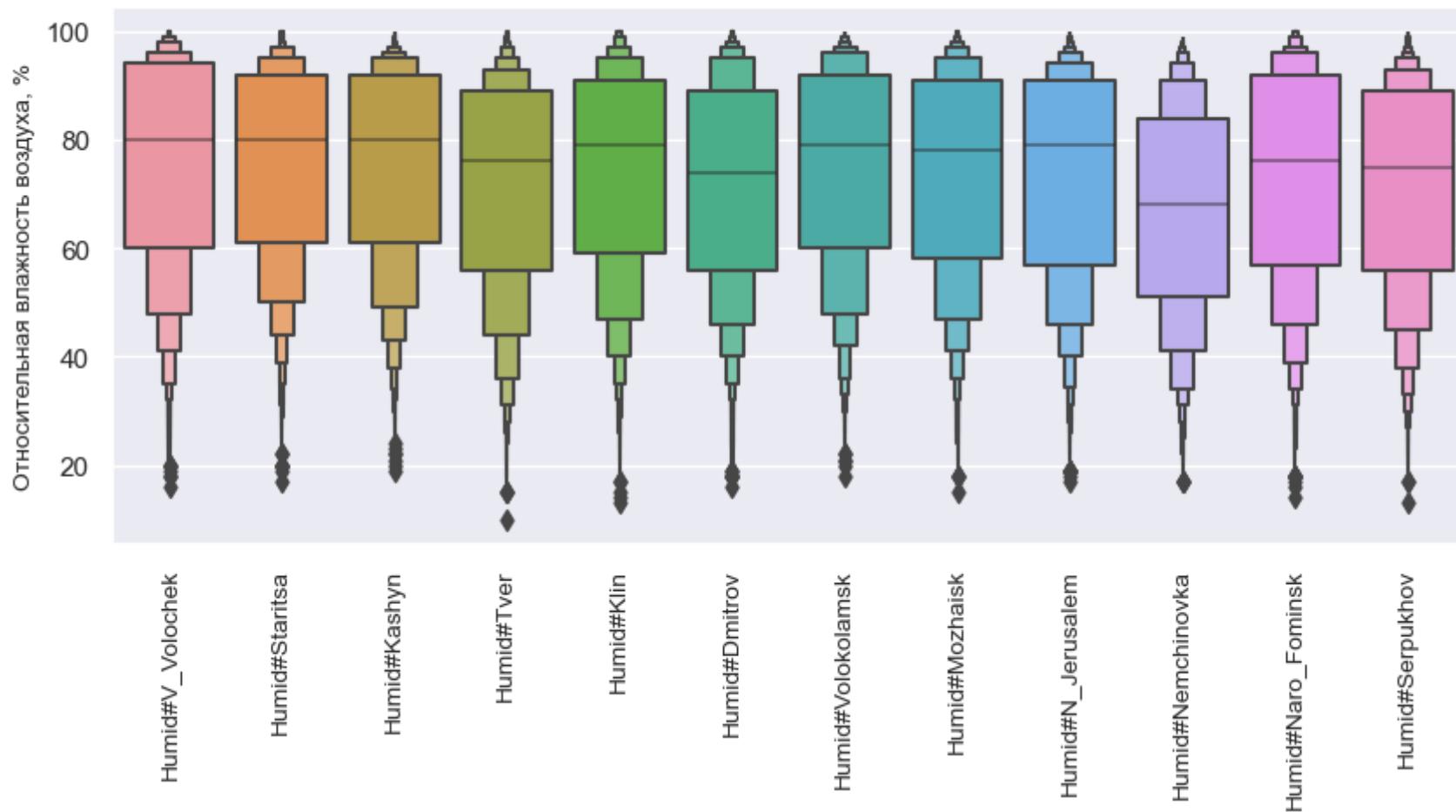
In [142...]

```
# В цикле выведем графики относительной влажности воздуха по метеостанциям в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp51).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp51[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Относительная влажность воздуха, %', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER51} в разрезе метеостанций:\n'
                        f'{season_name}')
plt.show()
```

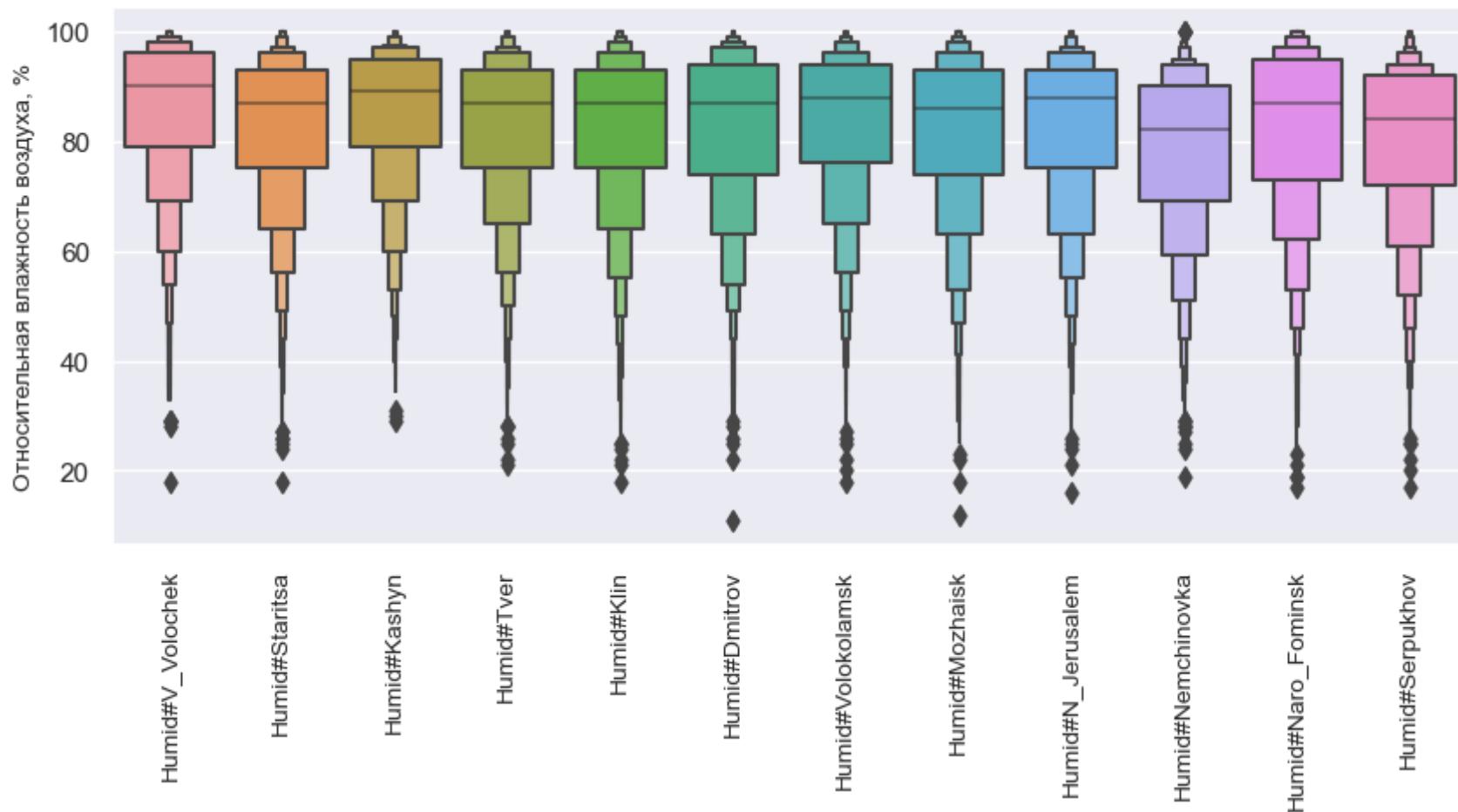
Распределение значений Humid в разрезе метеостанций:  
spring



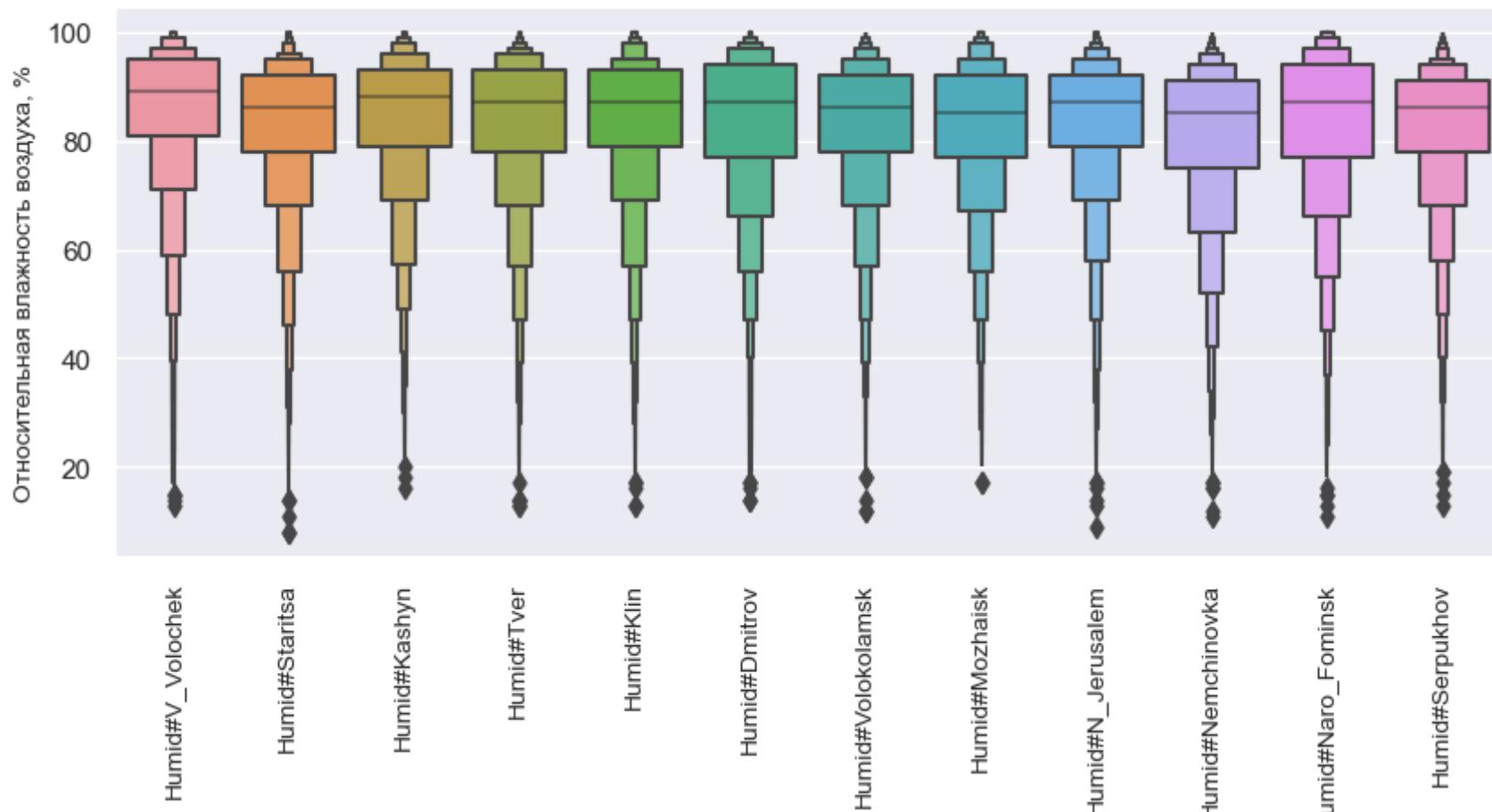
Распределение значений Humid в разрезе метеостанций:  
summer



Распределение значений Humid в разрезе метеостанций:  
autumn



### Распределение значений Humid в разрезе метеостанций: winter



Визуально сложно определить наличие аномальных значений.

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

In [143...]

```
print(f'Минимальное значение: {np.nanmin(df_tmp51)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp51)},\n'
      f'Средняя: {np.nanmean(df_tmp51)},\n'
      f'Медиана: {np.nanmedian(df_tmp51)}')
```

Минимальное значение: 8.0,  
Максимальное значение: 100.0,  
Средняя: 77.42531881251958,  
Медиана: 84.0

### Проверим соответствие влажности воздуха значениям температуры точки росы по всему DF

Рассчитаем температуру точки росы из имеющихся данных о влажности и температуре. Если расчётные значения не будут совпадать с архивными, пометим соответствующие значения влажности как ошибки.

Температура точки росы всегда является расчётным значением. Поэтому несовпадение расчёта и архива покажет нам ошибки в исходных данных.

Это целесообразно сделать в самом начале рассмотрения показателя Humid, чтобы исключить влияние некорректных значений на поиск выбросов.

Определим критерий для некорректных значений в 0.5 градуса по Цельсию и более

```
In [144...]: criterium = 0.5 # критерий для определения допустимой погрешности
# расчётного значения температуры точки росы от архивного

# В цикле по столбцам df_tmp51
# - определим промежуточный DF для расчётов;
# - заполним его значениями, необходимыми для приведения расчёта температуры точки росы,
# - расчитаем температуру точки росы,
# - рассчитаем абсолютное отклонение расчётных значений от архивных,
# - определим соответствие архивных значений критерию ошибки
# Исходим из идентичности индексов всех датафреймов в архивах

list_error_at = [] # определим список координат ошибочных значений в df_tmp51
for col in df_tmp51.columns:
    df_dew_p = pd.DataFrame(None, index=df_tmp51.index) # создадим пустой DF
    station_name = col[len(PARAMETER51)+1:] # выделяем название метеостанции из названия столбца
    height = df_station_dists[df_station_dists.station == station_name].height.values[0] # находим высоту метеостанции

    df_dew_p = (
        df_dew_p
        .assign(
            station=station_name,
            humid=df_tmp51.loc[:,PARAMETER51+'#'+station_name], # архивное значение относительной влажности
            T=dict_df_parameters['df_T'][f"{'T'}{col[len(PARAMETER51):]}"], # Температура воздуха (архив)
```

```

        dew_p=dict_df_parameters['df_Dew_point'][f"{'Dew_point'+col[len(PARAMETER51):]}"] # Точка росы (архив)
    )
)

df_dew_p = (
    df_dew_p
    .assign(dew_p_c=df_dew_p.apply(lambda x: dew_point_calculator(temp_=x['T'], humid_=x.humid), axis=1)
           , # Расчётная величина точки росы
           dp_delta=lambda x: abs(x.dew_p_c - x.dew_p), # абсолютное отклонение расчётной величины от архивной
           dp_error=lambda x: x.dp_delta >= criterium # критерий ошибки для отклонения расчётной величины, t с
    )
)

# Расширяем список координат ошибочных значений в df_tmp51:
# Если абсолютное отклонение расчётного значения от архивного больше или равно установленного критерия:
# прибавляем список кортежей моментов наблюдения и метеостанций по которым значения превышают допустимую погрешность
list_error_at = list_error_at + (df_dew_p
                                  .apply(
                                      lambda x: (x.name, station_name) if x.dp_delta >= criterium else np.nan,
                                      axis=1)
                                  .dropna()
                                  .tolist()
                                )

print(f'Количество аномальных отклонений расчётной температуры точки росы от архивного значения '
      f'для метеостанции {station_name} = {df_dew_p.dp_error.sum()}')


#list_error_at

```

Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции V\_Volochek = 434  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Staritsa = 15  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Kashyn = 225  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Tver = 19  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Klin = 5  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Dmitrov = 13  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Volokolamsk = 3  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Mozhaisk = 2  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции N\_Jerusalem = 9  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Nemchinovka = 40  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Naro\_Fominsk = 12  
Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Serpukhov = 360

Удаляем некорректные значения

In [145...]

```
for error in list_error_at:  
    df_tmp51.at[error[0], PARAMETER41 + '#' + error[1]] = np.nan
```

## Найдем выбросы в поле метеостанций.

Параметры:

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

In [146...]

```
df_tmp51.head()
```

Out[146]:

	Humid#V_Volochev	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#...
2022-06-09 21:00:00	76.0	92.0	89.0	92.0	NaN	NaN	80.0	NaN	NaN
2022-06-09 18:00:00	65.0	89.0	71.0	63.0	NaN	NaN	34.0	NaN	NaN
2022-06-09 15:00:00	61.0	45.0	40.0	34.0	NaN	NaN	32.0	NaN	NaN
2022-06-09 12:00:00	59.0	41.0	36.0	38.0	NaN	NaN	36.0	NaN	NaN
2022-06-09 09:00:00	56.0	56.0	52.0	48.0	41.0	NaN	41.0	NaN	NaN

In [147...]

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

```
df_tmp51 = df_tmp51.assign(field_out=df_tmp51.apply(lambda x: field_outliers(row_=x,
```

```

        method_='sigma',
        criterium_=3,
        IDW_=True,
        param_=PARAMETER51,
        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: 00:07:31

In [148]: df\_tmp51.dropna(subset=["field\_out"]).sample(5, random\_state=56)

Out[148]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#
2018-01-21 21:00:00	92.0	92.0	95.0	90.0	90.0	99.0	90.0	90.0	90.0
2015-04-30 15:00:00	34.0	19.0	26.0	22.0	28.0	40.0	28.0	47.0	
2017-09-02 12:00:00	92.0	55.0	68.0	56.0	57.0	59.0	61.0	57.0	
2013-01-01 00:00:00	100.0	97.0	98.0	99.0	98.0	99.0	99.0	98.0	
2018-02-21 06:00:00	89.0	86.0	87.0	85.0	85.0	95.0	87.0	87.0	

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 [149...]

```
# 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_tmp51.apply(lambda x : True if (len(x[:-1].dropna()) == 1) else False, axis = 1).dropna().sum()  
print(f'Количество случаев единичных значений в рядах моментов наблюдений: {single_values_count}')
```

Количество случаев единичных значений в рядах моментов наблюдений: 9

Проверим максимальное количество выбросов в поле метеостанций на момент наблюдения.

In [150...]

```
# Находим максимальное количество выбросов в строке поля метеостанций  
max_field_outliers = df_tmp51.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 [151...]

```
# 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_tmp51 = df_tmp51.assign(error_at =  
    df_tmp51.  
    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_=PARAMETER51)  
           ) if isinstance(x.field_out, list) else np.nan,  
           axis=1  
         )  
    )
```

In [152...]

```
df_tmp51.dropna(subset=['error_at']).sample(5, random_state=56)
```

Out[152]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2018-01-21 21:00:00	92.0	92.0	95.0	90.0	90.0	99.0	90.0	90.0	90.0
2015-04-30 15:00:00	34.0	19.0	26.0	22.0	28.0	40.0	28.0	47.0	
2017-09-02 12:00:00	92.0	55.0	68.0	56.0	57.0	59.0	61.0	57.0	
2013-01-01 00:00:00	100.0	97.0	98.0	99.0	98.0	99.0	99.0	98.0	
2018-02-21 06:00:00	89.0	86.0	87.0	85.0	85.0	95.0	87.0	87.0	

◀ ▶ Для подтверждения, являются ли отобранные значения давления на уровне моря ошибками, найдём выбросы во временном ряду

Среди выбросов в поле метеостанций найдем выбросы во временном окне.

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

- используем границы нормального распределения,
- не включаем проверяемое значение в подсчёт средней и сигмы,
- определим границы доверительного интервала в 97%,
- определим временное окно в +/- 3 дня ('7D'),
- включим в подсчёт только моменты наблюдения на данный час (equalhours=True).

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

- используем границы нормального распределения,
- не включаем проверяемое значение в подсчёт средней и сигмы,

- определим границы доверительного интервала в 97%,
- определим временное окно в +/- 24 часов ('48H'),
- включим в подсчёт все моменты наблюдения (equalhours=False).

In [153...]

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

# Нельзя удалять NaN в поле field_out непосредственно в df_tmp51 (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_out7D = pd.Series(index = df_tmp51.index, dtype='object', name="time_out_7D")

for elem in df_tmp51.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_out7D.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], PARAMETER51+'#'+station)

    # Вызываем функцию time_outliers с обозначенными выше параметрами
    val_func = time_outliers(df=df_tmp51,
                             # В качестве серии row_ передаём серию из одного значения: на момент наблюдения,
                             # где индекс серии соответствует названию столбца (idxs[1])
                             row_=df_tmp51.loc[idxs[0]][df_tmp51.loc[idxs[0]].index == idxs[1]],
                             td_symbol_='D',
                             td_quant_='3',
                             equal_hours_=True,
                             method_='norm',
                             criterium_=0.97,
                             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_out7D.at[idxs[0]] # Получаем список, находящийся в серии по индексу
        list_out = list_out + val_func # Расширяем его за счёт вывода функции (контактнаяция)
        ser_time_out7D.at[idxs[0]] = list_out # Записываем в серию обновлённый список

#         ser_time_out7D.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: 00:00:08

In [154...]

```

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

# Нельзя удалять NaN в поле field_out непосредственно в df_tmp51 (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_out48H = pd.Series(index = df_tmp51.index, dtype='object', name="time_out_48H")

for elem in df_tmp51.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_out48H.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], PARAMETER51+'#'+station)

    # Вызываем функцию time_outliers с обозначенными выше параметрами
    val_func = time_outliers(df_=df_tmp51,
                             # В качестве серии row_ передаём серию из одного значения: на момент наблюдения,
                             # где индекс серии соответствует названию столбца (idxs[1])
                             row_=df_tmp51.loc[idxs[0]][df_tmp51.loc[idxs[0]].index == idxs[1]],
                             td_symbol_='H',
                             td_quant_='24',
                             equal_hours_=False,
                             method_='norm',
                             criterium_=0.97,
                             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_out48H.at[idxs[0]] # Получаем список, находящийся в серии по индексу
        list_out = list_out + val_func # Расширяем его за счёт вывода функции (контактация)
        ser_time_out48H.at[idxs[0]] = list_out # Записываем в серию обновлённый список

```

```
#         ser_time_out7D.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: 00:00:04

Присоединим полученные серии к df\_tmp51

```
In [155...]  
# Индекс серии совпадает с индексом всех архивов, а значения в серии упорядочены по этому индексу.  
# Поэтому произведём объединение по индексу  
df_tmp51 = (df_tmp51  
            .merge(ser_time_out7D, left_index=True, right_index=True) # производим объединение  
            )  
df_tmp51 = (df_tmp51  
            .merge(ser_time_out48H, left_index=True, right_index=True)  
            )
```

```
In [156...]  
df_tmp51.head()
```

Out[156]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2022-06-09 21:00:00	76.0	92.0	89.0	92.0	NaN	NaN	80.0	NaN	NaN
2022-06-09 18:00:00	65.0	89.0	71.0	63.0	NaN	NaN	34.0	NaN	NaN
2022-06-09 15:00:00	61.0	45.0	40.0	34.0	NaN	NaN	32.0	NaN	NaN
2022-06-09 12:00:00	59.0	41.0	36.0	38.0	NaN	NaN	36.0	NaN	NaN
2022-06-09 09:00:00	56.0	56.0	52.0	48.0	41.0	NaN	41.0	NaN	NaN

◀ ▶

Заменим пустые списки в столбце time\_out\_48H и time\_out7D на NaN

In [157...]

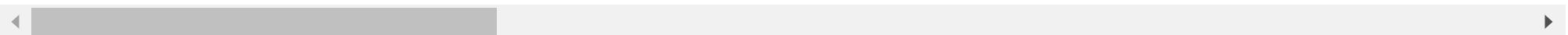
```
df_tmp51.time_out_48H = df_tmp51.time_out_48H.apply(lambda x: np.nan if x == [] else x)
df_tmp51.time_out_7D = df_tmp51.time_out_7D.apply(lambda x: np.nan if x == [] else x)
```

In [158...]

```
# Выводим рандомные строки из df_tmp51, удалив NaN в столбцах time_out48H и time_out7D
df_tmp51.dropna(subset=['time_out_7D', 'time_out_48H'], how='all').sample(5, random_state=56)
```

Out[158]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2021-03-31 12:00:00</b>	61.0	85.0	97.0	90.0	99.0	95.0	95.0	97.0	97.0
<b>2011-03-27 03:00:00</b>	87.0	93.0	80.0	86.0	83.0	80.0	88.0	76.0	76.0
<b>2017-08-05 15:00:00</b>	57.0	48.0	54.0	47.0	44.0	45.0	50.0	48.0	48.0
<b>2017-04-26 06:00:00</b>	83.0	52.0	71.0	56.0	51.0	54.0	53.0	53.0	53.0
<b>2010-02-18 06:00:00</b>	90.0	85.0	85.0	85.0	90.0	90.0	90.0	90.0	90.0



## Применимость критериев ошибочных значений

Рассмотрим полученные данные об аномальных значениях.

In [159...]

```
# Определяем столбец error_48H_at и error_7D_at
# Вторым элементом кортежа в error_at является СПИСОК станций, по которым найдены выбросы,
df_tmp51 = (df_tmp51
    .assign(error_48H_at =
        df_tmp51.dropna(subset=["time_out_48H"])
        .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_48H', # используем выбросы во временном окне
                    param_=PARAMETER51)
               ) if isinstance(x.field_out, list) else np.nan,
               axis=1
            ))
```

```
        )
    .assign(error_7D_at =
        df_tmp51.dropna(subset=["time_out_7D"])
        .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_7D', # используем выбросы во временном окне
                    param_=PARAMETER51)
                 ) if isinstance(x.field_out, list) else np.nan,
               axis=1
            )
        )
    )
)
```

```
In [160... df_tmp51.dropna(subset=["error_48H_at", "error_7D_at"], how='all')
```

Out[160]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2022-06-04 18:00:00</b>	70.0	46.0	44.0	35.0	40.0	41.0	40.0	48.0	48.0
<b>2022-05-08 12:00:00</b>	88.0	51.0	45.0	39.0	47.0	50.0	46.0	43.0	43.0
<b>2022-05-03 15:00:00</b>	83.0	31.0	36.0	29.0	34.0	34.0	34.0	35.0	35.0
<b>2022-04-29 15:00:00</b>	82.0	45.0	34.0	33.0	34.0	31.0	33.0	36.0	36.0
<b>2022-04-29 06:00:00</b>	63.0	88.0	99.0	99.0	100.0	98.0	99.0	98.0	98.0
<b>2022-04-29 03:00:00</b>	63.0	94.0	93.0	99.0	100.0	96.0	99.0	99.0	99.0
<b>2022-04-18 18:00:00</b>	29.0	29.0	24.0	22.0	29.0	24.0	26.0	36.0	36.0
<b>2022-02-27 21:00:00</b>	76.0	67.0	90.0	75.0	75.0	77.0	67.0	76.0	76.0
<b>2021-12-01 09:00:00</b>	89.0	96.0	93.0	97.0	100.0	96.0	94.0	82.0	82.0
<b>2021-12-01 06:00:00</b>	90.0	97.0	93.0	99.0	100.0	95.0	93.0	86.0	86.0
<b>2021-11-25 12:00:00</b>	99.0	93.0	70.0	94.0	94.0	95.0	95.0	91.0	91.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2021-11-21 21:00:00	76.0	97.0	95.0	93.0	100.0	87.0	97.0	98.0	98.0
2021-08-31 09:00:00	99.0	95.0	97.0	97.0	100.0	97.0	93.0	93.0	93.0
2021-08-04 03:00:00	91.0	96.0	89.0	88.0	97.0	84.0	96.0	90.0	90.0
2021-06-13 09:00:00	71.0	69.0	96.0	60.0	68.0	72.0	64.0	61.0	61.0
2021-06-12 00:00:00	97.0	96.0	76.0	89.0	100.0	94.0	97.0	100.0	100.0
2021-05-22 18:00:00	89.0	39.0	34.0	29.0	33.0	32.0	32.0	32.0	32.0
2021-03-31 12:00:00	61.0	85.0	97.0	90.0	99.0	95.0	95.0	97.0	97.0
2021-03-24 06:00:00	64.0	89.0	92.0	90.0	96.0	94.0	90.0	96.0	96.0
2021-02-14 06:00:00	64.0	83.0	85.0	81.0	90.0	86.0	83.0	79.0	79.0
2021-02-09 06:00:00	80.0	82.0	76.0	74.0	83.0	85.0	84.0	84.0	84.0
2020-10-12 15:00:00	71.0	92.0	89.0	82.0	96.0	95.0	95.0	90.0	90.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2020-08-23 18:00:00</b>	97.0	64.0	60.0	49.0	52.0	48.0	51.0	50.0	50.0
<b>2020-08-07 12:00:00</b>	55.0	66.0	52.0	10.0	62.0	54.0	58.0	61.0	61.0
<b>2020-07-23 15:00:00</b>	47.0	49.0	52.0	43.0	60.0	97.0	49.0	49.0	49.0
<b>2020-07-12 00:00:00</b>	60.0	96.0	90.0	88.0	99.0	90.0	99.0	97.0	97.0
<b>2020-07-08 03:00:00</b>	93.0	99.0	95.0	94.0	100.0	98.0	99.0	100.0	100.0
<b>2020-07-08 00:00:00</b>	93.0	98.0	94.0	93.0	100.0	98.0	99.0	100.0	100.0
<b>2020-06-23 03:00:00</b>	95.0	86.0	95.0	74.0	100.0	52.0	96.0	97.0	97.0
<b>2020-05-29 21:00:00</b>	79.0	93.0	91.0	90.0	100.0	99.0	97.0	98.0	98.0
<b>2020-05-29 15:00:00</b>	63.0	89.0	90.0	87.0	100.0	98.0	92.0	95.0	95.0
<b>2020-05-23 06:00:00</b>	70.0	92.0	93.0	93.0	100.0	97.0	95.0	96.0	96.0
<b>2020-05-23 03:00:00</b>	68.0	93.0	95.0	93.0	100.0	97.0	95.0	96.0	96.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2020-05-07 18:00:00	74.0	87.0	91.0	90.0	90.0	94.0	91.0	90.0	90.0
2020-05-01 12:00:00	36.0	42.0	44.0	43.0	36.0	39.0	39.0	49.0	49.0
2020-04-13 06:00:00	71.0	53.0	57.0	48.0	53.0	52.0	52.0	55.0	55.0
2020-03-19 12:00:00	57.0	45.0	86.0	50.0	45.0	47.0	37.0	49.0	49.0
2020-01-12 06:00:00	65.0	91.0	96.0	89.0	94.0	94.0	92.0	98.0	98.0
2019-11-25 15:00:00	71.0	73.0	79.0	72.0	74.0	73.0	71.0	66.0	66.0
2019-11-07 18:00:00	94.0	96.0	94.0	93.0	92.0	98.0	95.0	97.0	97.0
2019-11-07 12:00:00	85.0	96.0	88.0	93.0	88.0	89.0	93.0	97.0	97.0
2019-10-26 12:00:00	66.0	65.0	71.0	67.0	69.0	75.0	69.0	73.0	73.0
2019-09-21 15:00:00	80.0	59.0	49.0	47.0	45.0	56.0	44.0	48.0	48.0
2019-09-04 21:00:00	94.0	90.0	91.0	92.0	91.0	81.0	94.0	97.0	97.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2019-08-09 18:00:00</b>	93.0	85.0	93.0	91.0	86.0	89.0	88.0	80.0	80.0
<b>2019-06-27 15:00:00</b>	75.0	87.0	51.0	79.0	91.0	88.0	95.0	99.0	99.0
<b>2019-06-09 09:00:00</b>	54.0	56.0	81.0	50.0	54.0	54.0	54.0	50.0	50.0
<b>2019-05-30 09:00:00</b>	89.0	60.0	64.0	57.0	58.0	60.0	62.0	61.0	61.0
<b>2019-05-09 12:00:00</b>	90.0	93.0	84.0	92.0	89.0	91.0	94.0	18.0	18.0
<b>2019-04-28 12:00:00</b>	33.0	30.0	34.0	31.0	33.0	35.0	33.0	45.0	45.0
<b>2019-03-04 15:00:00</b>	77.0	90.0	60.0	86.0	86.0	90.0	89.0	86.0	86.0
<b>2018-12-02 06:00:00</b>	82.0	87.0	61.0	82.0	83.0	95.0	88.0	90.0	90.0
<b>2018-11-17 21:00:00</b>	97.0	93.0	98.0	95.0	93.0	98.0	90.0	80.0	80.0
<b>2018-11-13 09:00:00</b>	71.0	57.0	77.0	69.0	73.0	71.0	75.0	74.0	74.0
<b>2018-10-01 18:00:00</b>	52.0	51.0	64.0	48.0	51.0	56.0	50.0	51.0	51.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#
2018-08-30 06:00:00	96.0	94.0	90.0	92.0	91.0	67.0	94.0	99.0	
2018-08-04 18:00:00	47.0	51.0	86.0	42.0	49.0	60.0	44.0	47.0	
2018-07-18 12:00:00	20.0	89.0	63.0	62.0	70.0	75.0	78.0	70.0	
2018-07-05 15:00:00	50.0	56.0	83.0	63.0	61.0	57.0	48.0	55.0	
2018-06-02 09:00:00	53.0	45.0	82.0	44.0	48.0	49.0	47.0	42.0	
2018-05-27 18:00:00	37.0	40.0	72.0	34.0	40.0	40.0	38.0	39.0	
2018-05-20 06:00:00	90.0	93.0	79.0	94.0	94.0	99.0	95.0	96.0	
2018-05-05 15:00:00	83.0	43.0	41.0	38.0	36.0	38.0	33.0	34.0	
2018-05-05 12:00:00	90.0	54.0	49.0	51.0	56.0	55.0	38.0	43.0	
2018-05-02 12:00:00	68.0	44.0	90.0	46.0	38.0	42.0	35.0	34.0	
2018-04-10 15:00:00	89.0	42.0	54.0	36.0	32.0	32.0	31.0	26.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2018-04-06 18:00:00</b>	95.0	51.0	51.0	42.0	34.0	49.0	41.0	37.0	
<b>2018-03-24 09:00:00</b>	94.0	72.0	74.0	68.0	65.0	70.0	67.0	80.0	
<b>2017-09-14 09:00:00</b>	89.0	46.0	82.0	83.0	76.0	77.0	83.0	79.0	
<b>2017-09-05 00:00:00</b>	89.0	87.0	89.0	82.0	88.0	90.0	84.0	86.0	
<b>2017-08-24 00:00:00</b>	92.0	88.0	67.0	92.0	93.0	91.0	92.0	94.0	
<b>2017-08-07 12:00:00</b>	56.0	49.0	61.0	47.0	53.0	54.0	51.0	57.0	
<b>2017-08-05 15:00:00</b>	57.0	48.0	54.0	47.0	44.0	45.0	50.0	48.0	
<b>2017-07-22 18:00:00</b>	49.0	48.0	47.0	34.0	44.0	46.0	40.0	53.0	
<b>2017-07-22 15:00:00</b>	42.0	46.0	43.0	43.0	40.0	53.0	45.0	53.0	
<b>2017-07-21 18:00:00</b>	55.0	51.0	75.0	56.0	46.0	56.0	47.0	50.0	
<b>2017-04-26 06:00:00</b>	83.0	52.0	71.0	56.0	51.0	54.0	53.0	53.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2017-04-17 03:00:00	56.0	87.0	92.0	94.0	94.0	99.0	96.0	90.0	90.0
2017-04-06 03:00:00	64.0	83.0	85.0	78.0	80.0	85.0	86.0	84.0	84.0
2017-03-26 15:00:00	72.0	44.0	45.0	44.0	32.0	34.0	32.0	30.0	30.0
2016-12-17 18:00:00	99.0	92.0	96.0	94.0	94.0	97.0	96.0	96.0	96.0
2016-12-17 06:00:00	99.0	91.0	94.0	93.0	89.0	22.0	93.0	95.0	95.0
2016-12-01 09:00:00	93.0	86.0	89.0	86.0	85.0	21.0	88.0	87.0	87.0
2016-11-02 03:00:00	94.0	90.0	96.0	94.0	91.0	89.0	93.0	91.0	91.0
2016-10-08 12:00:00	97.0	95.0	95.0	94.0	93.0	94.0	95.0	77.0	77.0
2016-09-28 09:00:00	72.0	81.0	87.0	88.0	88.0	90.0	88.0	85.0	85.0
2016-09-20 15:00:00	64.0	83.0	74.0	83.0	88.0	89.0	92.0	84.0	84.0
2016-08-22 15:00:00	61.0	55.0	61.0	56.0	57.0	58.0	68.0	59.0	59.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2016-08-19 21:00:00</b>	95.0	94.0	84.0	66.0	91.0	87.0	94.0	92.0	92.0
<b>2015-07-03 06:00:00</b>	97.0	88.0	87.0	84.0	85.0	78.0	92.0	89.0	89.0
<b>2015-07-01 12:00:00</b>	52.0	79.0	43.0	32.0	47.0	40.0	37.0	48.0	48.0
<b>2015-06-27 09:00:00</b>	95.0	92.0	94.0	96.0	94.0	99.0	96.0	96.0	96.0
<b>2015-06-08 09:00:00</b>	53.0	51.0	50.0	42.0	42.0	38.0	44.0	47.0	47.0
<b>2015-06-04 18:00:00</b>	62.0	38.0	38.0	32.0	36.0	36.0	35.0	41.0	41.0
<b>2015-05-17 12:00:00</b>	89.0	86.0	56.0	89.0	89.0	81.0	90.0	93.0	93.0
<b>2015-01-20 18:00:00</b>	84.0	84.0	87.0	90.0	90.0	68.0	91.0	85.0	85.0
<b>2015-01-10 12:00:00</b>	96.0	94.0	92.0	91.0	91.0	93.0	96.0	94.0	94.0
<b>2015-01-06 00:00:00</b>	76.0	77.0	77.0	77.0	76.0	77.0	78.0	72.0	72.0
<b>2014-12-27 03:00:00</b>	17.0	89.0	89.0	87.0	88.0	92.0	93.0	90.0	90.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#
2014-11-26 15:00:00	92.0	92.0	87.0	88.0	91.0	95.0	96.0	95.0	
2014-11-26 12:00:00	91.0	92.0	89.0	89.0	93.0	95.0	96.0	95.0	
2014-10-29 12:00:00	29.0	26.0	44.0	26.0	22.0	28.0	20.0	18.0	
2014-09-04 06:00:00	95.0	98.0	97.0	94.0	93.0	99.0	96.0	96.0	
2014-08-31 06:00:00	87.0	96.0	93.0	95.0	95.0	97.0	99.0	97.0	
2014-07-18 18:00:00	89.0	70.0	61.0	53.0	50.0	57.0	42.0	43.0	
2014-07-05 12:00:00	59.0	52.0	71.0	51.0	44.0	51.0	44.0	55.0	
2014-07-02 12:00:00	84.0	50.0	41.0	30.0	24.0	31.0	30.0	38.0	
2014-06-30 09:00:00	96.0	62.0	71.0	56.0	59.0	59.0	62.0	57.0	
2014-05-13 18:00:00	58.0	54.0	77.0	46.0	50.0	53.0	54.0	54.0	
2014-05-10 15:00:00	30.0	33.0	35.0	29.0	37.0	36.0	42.0	37.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2014-04-10 15:00:00</b>	28.0	29.0	31.0	28.0	29.0	31.0	31.0	25.0	
<b>2014-04-08 15:00:00</b>	30.0	29.0	32.0	27.0	31.0	34.0	27.0	34.0	
<b>2014-02-14 09:00:00</b>	99.0	100.0	100.0	98.0	99.0	100.0	100.0	99.0	
<b>2013-12-19 06:00:00</b>	83.0	81.0	79.0	76.0	77.0	84.0	79.0	83.0	
<b>2013-12-12 21:00:00</b>	99.0	96.0	100.0	95.0	95.0	100.0	97.0	96.0	
<b>2013-12-12 12:00:00</b>	99.0	99.0	100.0	96.0	96.0	97.0	98.0	96.0	
<b>2013-12-05 15:00:00</b>	96.0	89.0	94.0	92.0	94.0	96.0	92.0	89.0	
<b>2013-12-01 06:00:00</b>	96.0	92.0	92.0	90.0	90.0	95.0	94.0	95.0	
<b>2013-11-26 15:00:00</b>	87.0	99.0	100.0	96.0	99.0	100.0	99.0	99.0	
<b>2013-11-25 15:00:00</b>	98.0	100.0	99.0	96.0	95.0	98.0	97.0	96.0	
<b>2013-09-20 21:00:00</b>	88.0	97.0	94.0	94.0	94.0	94.0	95.0	95.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2013-09-20 18:00:00	77.0	90.0	95.0	91.0	94.0	95.0	94.0	93.0	92.0
2013-09-08 12:00:00	49.0	71.0	93.0	77.0	85.0	81.0	94.0	85.0	86.0
2013-09-01 21:00:00	97.0	95.0	96.0	93.0	95.0	97.0	96.0	94.0	93.0
2013-08-30 03:00:00	99.0	99.0	100.0	78.0	97.0	99.0	98.0	95.0	94.0
2013-08-23 21:00:00	96.0	92.0	89.0	88.0	96.0	97.0	96.0	88.0	87.0
2013-08-18 03:00:00	99.0	98.0	98.0	56.0	95.0	94.0	97.0	96.0	95.0
2013-08-15 18:00:00	54.0	47.0	54.0	37.0	44.0	50.0	39.0	46.0	45.0
2013-08-11 12:00:00	94.0	65.0	57.0	52.0	51.0	57.0	51.0	50.0	49.0
2013-07-25 06:00:00	95.0	96.0	98.0	75.0	95.0	98.0	97.0	97.0	96.0
2013-07-05 06:00:00	99.0	93.0	98.0	38.0	96.0	89.0	97.0	90.0	89.0
2013-06-15 15:00:00	66.0	46.0	45.0	34.0	40.0	37.0	43.0	41.0	40.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2013-05-10 15:00:00</b>	79.0	23.0	26.0	24.0	25.0	23.0	26.0	23.0	23.0
<b>2013-05-07 03:00:00</b>	100.0	92.0	97.0	91.0	97.0	95.0	96.0	91.0	91.0
<b>2013-04-26 03:00:00</b>	94.0	96.0	94.0	93.0	94.0	95.0	90.0	92.0	92.0
<b>2013-03-25 15:00:00</b>	23.0	72.0	90.0	62.0	81.0	84.0	91.0	74.0	74.0
<b>2013-03-25 12:00:00</b>	28.0	77.0	86.0	73.0	79.0	80.0	90.0	83.0	83.0
<b>2013-03-25 09:00:00</b>	37.0	81.0	90.0	79.0	88.0	83.0	91.0	87.0	87.0
<b>2013-03-25 06:00:00</b>	47.0	78.0	92.0	85.0	88.0	87.0	92.0	82.0	82.0
<b>2013-03-25 03:00:00</b>	50.0	74.0	90.0	80.0	90.0	86.0	92.0	81.0	81.0
<b>2013-03-21 09:00:00</b>	65.0	87.0	81.0	76.0	89.0	85.0	90.0	85.0	85.0
<b>2013-03-18 15:00:00</b>	75.0	75.0	77.0	74.0	86.0	85.0	84.0	81.0	81.0
<b>2013-03-15 18:00:00</b>	71.0	96.0	95.0	93.0	96.0	98.0	96.0	98.0	98.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#
<b>2013-03-15 15:00:00</b>	69.0	93.0	88.0	88.0	94.0	96.0	96.0	97.0	
<b>2013-02-03 21:00:00</b>	84.0	100.0	100.0	97.0	96.0	97.0	99.0	97.0	
<b>2013-02-03 18:00:00</b>	82.0	94.0	100.0	97.0	95.0	97.0	99.0	96.0	
<b>2013-01-23 15:00:00</b>	93.0	95.0	93.0	91.0	94.0	90.0	93.0	88.0	
<b>2012-10-12 03:00:00</b>	99.0	100.0	100.0	96.0	98.0	99.0	99.0	98.0	
<b>2012-10-09 21:00:00</b>	67.0	88.0	86.0	92.0	92.0	92.0	96.0	94.0	
<b>2012-10-07 09:00:00</b>	99.0	95.0	97.0	99.0	95.0	97.0	97.0	91.0	
<b>2012-09-28 21:00:00</b>	95.0	95.0	97.0	97.0	95.0	91.0	96.0	87.0	
<b>2012-09-23 00:00:00</b>	99.0	96.0	96.0	97.0	96.0	96.0	94.0	94.0	
<b>2012-09-21 21:00:00</b>	99.0	97.0	89.0	95.0	90.0	93.0	96.0	92.0	
<b>2012-09-09 06:00:00</b>	93.0	94.0	97.0	97.0	97.0	93.0	97.0	93.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2012-08-11 06:00:00</b>	96.0	99.0	94.0	88.0	97.0	76.0	100.0	93.0	93.0
<b>2012-08-08 06:00:00</b>	93.0	97.0	90.0	94.0	90.0	82.0	97.0	92.0	92.0
<b>2012-07-13 03:00:00</b>	98.0	100.0	90.0	94.0	95.0	63.0	98.0	94.0	94.0
<b>2012-07-11 21:00:00</b>	86.0	93.0	89.0	93.0	93.0	88.0	87.0	74.0	74.0
<b>2012-07-10 03:00:00</b>	98.0	98.0	92.0	94.0	95.0	89.0	98.0	91.0	91.0
<b>2012-06-24 09:00:00</b>	83.0	82.0	97.0	85.0	87.0	93.0	82.0	82.0	82.0
<b>2012-06-24 00:00:00</b>	96.0	97.0	64.0	94.0	94.0	81.0	98.0	91.0	91.0
<b>2012-06-13 03:00:00</b>	52.0	94.0	91.0	94.0	96.0	91.0	92.0	93.0	93.0
<b>2012-06-10 03:00:00</b>	95.0	91.0	92.0	91.0	95.0	80.0	96.0	87.0	87.0
<b>2012-06-08 03:00:00</b>	77.0	96.0	93.0	91.0	97.0	97.0	97.0	93.0	93.0
<b>2012-05-24 18:00:00</b>	57.0	36.0	43.0	38.0	41.0	39.0	39.0	34.0	34.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2012-05-24 15:00:00	53.0	39.0	42.0	39.0	39.0	41.0	41.0	37.0	37.0
2012-05-21 15:00:00	80.0	29.0	50.0	30.0	28.0	36.0	34.0	26.0	26.0
2012-05-08 15:00:00	65.0	85.0	78.0	80.0	84.0	96.0	87.0	75.0	75.0
2012-04-15 09:00:00	74.0	83.0	76.0	97.0	74.0	86.0	88.0	85.0	85.0
2012-04-06 06:00:00	69.0	97.0	95.0	94.0	93.0	94.0	97.0	96.0	96.0
2012-04-06 03:00:00	69.0	96.0	90.0	85.0	91.0	91.0	96.0	95.0	95.0
2012-03-05 18:00:00	74.0	70.0	68.0	69.0	73.0	70.0	68.0	79.0	79.0
2012-03-03 12:00:00	98.0	68.0	79.0	71.0	61.0	73.0	67.0	72.0	72.0
2012-01-27 21:00:00	90.0	85.0	70.0	85.0	81.0	80.0	90.0	85.0	85.0
2012-01-08 21:00:00	95.0	95.0	92.0	94.0	95.0	95.0	95.0	96.0	96.0
2011-12-05 09:00:00	94.0	84.0	95.0	89.0	93.0	90.0	87.0	87.0	87.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2011-11-25 12:00:00</b>	90.0	89.0	81.0	93.0	89.0	89.0	92.0	90.0	90.0
<b>2011-11-20 03:00:00</b>	92.0	92.0	78.0	96.0	92.0	88.0	94.0	88.0	88.0
<b>2011-10-29 03:00:00</b>	87.0	95.0	94.0	96.0	94.0	98.0	98.0	94.0	94.0
<b>2011-10-28 15:00:00</b>	66.0	82.0	86.0	92.0	92.0	82.0	92.0	87.0	87.0
<b>2011-10-12 12:00:00</b>	82.0	88.0	69.0	84.0	83.0	82.0	88.0	89.0	89.0
<b>2011-10-10 15:00:00</b>	80.0	94.0	93.0	93.0	97.0	93.0	97.0	97.0	97.0
<b>2011-07-16 15:00:00</b>	82.0	52.0	49.0	45.0	34.0	44.0	44.0	40.0	40.0
<b>2011-07-12 03:00:00</b>	97.0	96.0	94.0	89.0	96.0	59.0	93.0	93.0	93.0
<b>2011-05-24 03:00:00</b>	88.0	86.0	73.0	88.0	88.0	84.0	91.0	89.0	89.0
<b>2011-03-29 12:00:00</b>	53.0	51.0	46.0	47.0	44.0	50.0	42.0	51.0	51.0
<b>2011-03-27 03:00:00</b>	87.0	93.0	80.0	86.0	83.0	80.0	88.0	76.0	76.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
2011-03-14 18:00:00	23.0	65.0	60.0	67.0	62.0	63.0	65.0	66.0	66.0
2011-03-14 15:00:00	19.0	55.0	49.0	61.0	53.0	55.0	60.0	57.0	57.0
2011-03-14 12:00:00	26.0	62.0	58.0	66.0	57.0	61.0	65.0	65.0	65.0
2011-02-18 21:00:00	85.0	85.0	79.0	85.0	80.0	79.0	79.0	70.0	70.0
2011-01-27 09:00:00	90.0	95.0	90.0	90.0	90.0	90.0	90.0	80.0	80.0
2010-11-04 15:00:00	64.0	88.0	96.0	90.0	94.0	97.0	97.0	86.0	86.0
2010-10-09 09:00:00	98.0	90.0	65.0	92.0	98.0	95.0	100.0	96.0	96.0
2010-09-29 06:00:00	97.0	97.0	93.0	95.0	97.0	92.0	99.0	95.0	95.0
2010-09-21 21:00:00	88.0	90.0	90.0	89.0	91.0	57.0	88.0	88.0	88.0
2010-06-23 15:00:00	79.0	44.0	33.0	33.0	23.0	29.0	22.0	21.0	21.0
2010-05-26 21:00:00	61.0	91.0	93.0	96.0	95.0	93.0	97.0	92.0	92.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2010-05-26 15:00:00</b>	50.0	92.0	79.0	82.0	89.0	92.0	89.0	91.0	91.0
<b>2010-04-12 12:00:00</b>	39.0	45.0	38.0	42.0	43.0	47.0	46.0	96.0	96.0
<b>2010-03-20 12:00:00</b>	70.0	94.0	94.0	98.0	94.0	90.0	94.0	93.0	93.0
<b>2010-02-24 18:00:00</b>	61.0	88.0	98.0	94.0	92.0	94.0	96.0	92.0	92.0
<b>2010-02-18 06:00:00</b>	90.0	85.0	85.0	85.0	90.0	90.0	90.0	90.0	90.0
<b>2010-02-16 12:00:00</b>	90.0	81.0	70.0	85.0	85.0	90.0	90.0	85.0	85.0
<b>2009-12-13 21:00:00</b>	90.0	87.0	73.0	93.0	87.0	91.0	89.0	89.0	89.0
<b>2009-11-10 09:00:00</b>	95.0	94.0	84.0	96.0	NaN	94.0	96.0	98.0	98.0
<b>2009-11-08 21:00:00</b>	97.0	95.0	97.0	99.0	93.0	97.0	99.0	97.0	97.0
<b>2009-11-08 09:00:00</b>	97.0	97.0	96.0	99.0	97.0	94.0	99.0	97.0	97.0
<b>2009-10-27 18:00:00</b>	NaN	95.0	NaN	99.0	97.0	95.0	99.0	97.0	97.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#
<b>2009-05-29 03:00:00</b>	50.0	97.0	89.0	87.0	94.0	97.0	96.0	95.0	
<b>2009-05-13 15:00:00</b>	97.0	97.0	40.0	97.0	87.0	88.0	92.0	94.0	
<b>2009-05-12 09:00:00</b>	76.0	87.0	86.0	89.0	86.0	73.0	88.0	74.0	
<b>2009-05-08 15:00:00</b>	37.0	36.0	38.0	33.0	39.0	38.0	35.0	35.0	
<b>2009-04-12 15:00:00</b>	78.0	26.0	34.0	25.0	23.0	42.0	27.0	33.0	
<b>2009-04-06 09:00:00</b>	96.0	93.0	77.0	98.0	95.0	93.0	100.0	90.0	
<b>2009-02-21 03:00:00</b>	91.0	93.0	85.0	94.0	95.0	90.0	92.0	90.0	
<b>2009-02-19 21:00:00</b>	89.0	84.0	83.0	94.0	95.0	91.0	93.0	85.0	
<b>2009-01-24 03:00:00</b>	94.0	96.0	94.0	98.0	98.0	96.0	98.0	90.0	
<b>2008-12-28 09:00:00</b>	77.0	89.0	90.0	94.0	91.0	47.0	93.0	84.0	
<b>2008-12-14 21:00:00</b>	98.0	56.0	92.0	97.0	88.0	87.0	93.0	91.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2008-12-09 21:00:00</b>	93.0	95.0	93.0	96.0	98.0	92.0	100.0	94.0	94.0
<b>2008-12-09 03:00:00</b>	75.0	92.0	92.0	93.0	97.0	92.0	99.0	94.0	94.0
<b>2008-12-08 21:00:00</b>	74.0	85.0	94.0	94.0	94.0	92.0	98.0	96.0	96.0
<b>2008-11-19 03:00:00</b>	88.0	90.0	91.0	92.0	92.0	88.0	93.0	82.0	82.0
<b>2008-10-20 09:00:00</b>	79.0	93.0	97.0	97.0	97.0	94.0	95.0	94.0	94.0
<b>2008-08-25 21:00:00</b>	87.0	86.0	92.0	69.0	91.0	90.0	97.0	94.0	94.0
<b>2008-07-10 15:00:00</b>	46.0	75.0	47.0	52.0	50.0	51.0	47.0	40.0	40.0
<b>2008-06-29 03:00:00</b>	96.0	97.0	76.0	96.0	97.0	85.0	97.0	94.0	94.0
<b>2008-05-14 03:00:00</b>	59.0	94.0	80.0	91.0	92.0	93.0	94.0	85.0	85.0
<b>2008-04-18 03:00:00</b>	98.0	98.0	54.0	97.0	92.0	84.0	94.0	97.0	97.0
<b>2008-04-03 15:00:00</b>	22.0	20.0	21.0	39.0	26.0	37.0	30.0	33.0	33.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#
<b>2008-02-01 09:00:00</b>	79.0	88.0	90.0	93.0	98.0	98.0	97.0	94.0	
<b>2008-01-28 09:00:00</b>	80.0	80.0	84.0	86.0	96.0	88.0	92.0	83.0	
<b>2008-01-08 09:00:00</b>	85.0	85.0	80.0	90.0	95.0	85.0	90.0	80.0	
<b>2008-01-07 21:00:00</b>	70.0	85.0	80.0	85.0	95.0	85.0	90.0	85.0	
<b>2008-01-03 09:00:00</b>	85.0	90.0	85.0	90.0	95.0	80.0	90.0	85.0	
<b>2007-11-01 09:00:00</b>	85.0	90.0	97.0	94.0	93.0	94.0	95.0	97.0	
<b>2007-10-22 21:00:00</b>	99.0	97.0	99.0	99.0	99.0	94.0	95.0	91.0	
<b>2007-10-22 09:00:00</b>	99.0	99.0	97.0	97.0	99.0	92.0	99.0	94.0	
<b>2007-10-20 15:00:00</b>	72.0	93.0	92.0	97.0	97.0	95.0	97.0	97.0	
<b>2007-10-07 15:00:00</b>	97.0	95.0	69.0	96.0	97.0	89.0	96.0	91.0	
<b>2007-09-13 09:00:00</b>	95.0	94.0	94.0	97.0	92.0	91.0	97.0	94.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2007-09-10 03:00:00</b>	80.0	95.0	94.0	94.0	98.0	91.0	97.0	94.0	94.0
<b>2007-09-08 03:00:00</b>	97.0	95.0	95.0	93.0	98.0	94.0	93.0	91.0	91.0
<b>2007-09-03 09:00:00</b>	72.0	69.0	78.0	69.0	66.0	69.0	66.0	70.0	70.0
<b>2007-07-30 21:00:00</b>	77.0	77.0	81.0	85.0	82.0	86.0	82.0	47.0	47.0
<b>2007-07-30 03:00:00</b>	96.0	96.0	83.0	94.0	96.0	93.0	97.0	96.0	96.0
<b>2007-03-31 15:00:00</b>	34.0	28.0	54.0	35.0	33.0	38.0	25.0	31.0	31.0
<b>2007-03-25 09:00:00</b>	73.0	65.0	76.0	76.0	76.0	76.0	85.0	73.0	73.0
<b>2007-02-15 03:00:00</b>	90.0	91.0	93.0	95.0	99.0	92.0	96.0	94.0	94.0
<b>2007-01-10 15:00:00</b>	73.0	86.0	92.0	90.0	94.0	93.0	93.0	88.0	88.0
<b>2006-12-21 15:00:00</b>	96.0	96.0	85.0	96.0	96.0	41.0	93.0	90.0	90.0
<b>2006-11-10 09:00:00</b>	81.0	95.0	90.0	96.0	98.0	98.0	100.0	96.0	96.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2006-11-07 03:00:00</b>	80.0	90.0	93.0	95.0	93.0	91.0	94.0	55.0	
<b>2006-10-22 03:00:00</b>	99.0	95.0	96.0	99.0	97.0	95.0	99.0	93.0	
<b>2006-10-22 00:00:00</b>	Nan	95.0	Nan	98.0	99.0	97.0	99.0	95.0	
<b>2006-10-09 03:00:00</b>	97.0	96.0	95.0	96.0	92.0	79.0	99.0	92.0	
<b>2006-08-10 21:00:00</b>	99.0	95.0	97.0	98.0	95.0	91.0	97.0	90.0	
<b>2006-08-03 21:00:00</b>	98.0	98.0	90.0	98.0	97.0	99.0	99.0	93.0	
<b>2006-07-17 03:00:00</b>	88.0	92.0	91.0	93.0	96.0	84.0	96.0	86.0	
<b>2006-06-28 15:00:00</b>	96.0	55.0	54.0	53.0	47.0	49.0	52.0	46.0	
<b>2006-05-15 15:00:00</b>	46.0	42.0	40.0	44.0	34.0	41.0	38.0	54.0	
<b>2005-11-19 21:00:00</b>	72.0	91.0	85.0	90.0	94.0	84.0	90.0	87.0	
<b>2005-10-09 15:00:00</b>	Nan	46.0	49.0	47.0	47.0	46.0	55.0	44.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#...
2005-09-21 09:00:00	NaN	NaN	NaN	83.0	NaN	NaN	NaN	NaN	NaN
2005-09-05 03:00:00	NaN	97.0	96.0	96.0	95.0	97.0	99.0	97.0	
2005-09-03 00:00:00	93.0	96.0	93.0	48.0	93.0	99.0	100.0	95.0	

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, error\_48H\_at, error\_7D\_at и выведем его в отдельный столбец cross\_check

```
In [161...]: df_tmp51 = df_tmp51.assign(
    cross_check = df_tmp51
        .dropna(subset=["error_at", "error_48H_at", "error_7D_at"])
        .apply(
            lambda x: list(set(x.error_at[1]) & set(x.error_48H_at[1]) & set(x.error_7D_at[1])),
            axis=1
        )
)
```

```
In [162...]: df_tmp51.dropna(subset=["cross_check"]).sample(7, random_state=56)
```

Out[162]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2017-09-05 00:00:00</b>	89.0	87.0	89.0	82.0	88.0	90.0	84.0	86.0	86.0
<b>2019-11-25 15:00:00</b>	71.0	73.0	79.0	72.0	74.0	73.0	71.0	66.0	66.0
<b>2011-02-18 21:00:00</b>	85.0	85.0	79.0	85.0	80.0	79.0	79.0	70.0	70.0
<b>2013-09-20 18:00:00</b>	77.0	90.0	95.0	91.0	94.0	95.0	94.0	93.0	93.0
<b>2013-12-01 06:00:00</b>	96.0	92.0	92.0	90.0	90.0	95.0	94.0	95.0	95.0
<b>2009-02-21 03:00:00</b>	91.0	93.0	85.0	94.0	95.0	90.0	92.0	90.0	90.0
<b>2011-01-27 09:00:00</b>	90.0	95.0	90.0	90.0	90.0	90.0	90.0	80.0	80.0

In [163...]

```
# Подсчитаем количество выбросов в поле метеостанций
count_field_out = (df_tmp51
    .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_out24H = (df_tmp51
    .error_48H_at # по столбцу error_48H_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_out7D = (df_tmp51
    .error_7D_at # по столбцу error_7D_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_tmp51
    .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 +/- 24 часа по всем моментам наблюдения: '
      f'{count_time_out24H}\n'
      f'Подтверждается аномалиями в промежутке +/- 3 дня на один и тот же час наблюдения: '
      f'{count_time_out7D}\n'
      f'Проверка: пересечение множеств выбросов в поле метеостанций и во временных окнах насчитывает '
      f'{count_cross_out} элемент(ов)')

```

В поле метеостанций выявлено аномальных значений: 1178,  
из них:  
Подтверждается аномалиями в промежутке +/- 24 часа по всем моментам наблюдения: 80  
Подтверждается аномалиями в промежутке +/- 3 дня на один и тот же час наблюдения: 230  
Проверка: пересечение множеств выбросов в поле метеостанций и во временных окнах насчитывает 43 элемент(ов)

### **Определим конечный критерий ошибочных значений:**

1. Аномальное значение выявлено в поле метеостанций И ПРИ ЭТОМ
  - A. Аномальное значение выявлено во временном ряду в промежутке +/- 24 часа по всем моментам наблюдения И
  - B. Аномальное значение выявлено во временном ряду в промежутке +/- 3 дня по одному и тому же часу наблюдения

### **Удалим (заменим на NaN) все ошибочные значения**

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

```
In [164]: df_tmp51.dropna(subset=['cross_check'])
```

Out[164]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2021-02-09 06:00:00</b>	80.0	82.0	76.0	74.0	83.0	85.0	84.0	84.0	84.0
<b>2020-08-07 12:00:00</b>	55.0	66.0	52.0	10.0	62.0	54.0	58.0	61.0	61.0
<b>2020-01-12 06:00:00</b>	65.0	91.0	96.0	89.0	94.0	94.0	92.0	98.0	98.0
<b>2019-11-25 15:00:00</b>	71.0	73.0	79.0	72.0	74.0	73.0	71.0	66.0	66.0
<b>2019-11-07 12:00:00</b>	85.0	96.0	88.0	93.0	88.0	89.0	93.0	97.0	97.0
<b>2018-11-17 21:00:00</b>	97.0	93.0	98.0	95.0	93.0	98.0	90.0	80.0	80.0
<b>2018-07-18 12:00:00</b>	20.0	89.0	63.0	62.0	70.0	75.0	78.0	70.0	70.0
<b>2017-09-05 00:00:00</b>	89.0	87.0	89.0	82.0	88.0	90.0	84.0	86.0	86.0
<b>2016-12-17 18:00:00</b>	99.0	92.0	96.0	94.0	94.0	97.0	96.0	96.0	96.0
<b>2016-12-17 06:00:00</b>	99.0	91.0	94.0	93.0	89.0	22.0	93.0	95.0	95.0
<b>2016-12-01 09:00:00</b>	93.0	86.0	89.0	86.0	85.0	21.0	88.0	87.0	87.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2015-06-27 09:00:00</b>	95.0	92.0	94.0	96.0	94.0	99.0	96.0	96.0	96.0
<b>2014-12-27 03:00:00</b>	17.0	89.0	89.0	87.0	88.0	92.0	93.0	90.0	90.0
<b>2014-04-10 15:00:00</b>	28.0	29.0	31.0	28.0	29.0	31.0	31.0	25.0	25.0
<b>2014-02-14 09:00:00</b>	99.0	100.0	100.0	98.0	99.0	100.0	100.0	99.0	99.0
<b>2013-12-19 06:00:00</b>	83.0	81.0	79.0	76.0	77.0	84.0	79.0	83.0	83.0
<b>2013-12-05 15:00:00</b>	96.0	89.0	94.0	92.0	94.0	96.0	92.0	89.0	89.0
<b>2013-12-01 06:00:00</b>	96.0	92.0	92.0	90.0	90.0	95.0	94.0	95.0	95.0
<b>2013-11-25 15:00:00</b>	98.0	100.0	99.0	96.0	95.0	98.0	97.0	96.0	96.0
<b>2013-09-20 18:00:00</b>	77.0	90.0	95.0	91.0	94.0	95.0	94.0	93.0	93.0
<b>2012-10-09 21:00:00</b>	67.0	88.0	86.0	92.0	92.0	92.0	96.0	94.0	94.0
<b>2012-06-24 09:00:00</b>	83.0	82.0	97.0	85.0	87.0	93.0	82.0	82.0	82.0

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2012-06-13 03:00:00</b>	52.0	94.0	91.0	94.0	96.0	91.0	92.0	93.0	
<b>2012-04-15 09:00:00</b>	74.0	83.0	76.0	97.0	74.0	86.0	88.0	85.0	
<b>2011-12-05 09:00:00</b>	94.0	84.0	95.0	89.0	93.0	90.0	87.0	87.0	
<b>2011-10-28 15:00:00</b>	66.0	82.0	86.0	92.0	92.0	82.0	92.0	87.0	
<b>2011-03-14 15:00:00</b>	19.0	55.0	49.0	61.0	53.0	55.0	60.0	57.0	
<b>2011-02-18 21:00:00</b>	85.0	85.0	79.0	85.0	80.0	79.0	79.0	70.0	
<b>2011-01-27 09:00:00</b>	90.0	95.0	90.0	90.0	90.0	90.0	90.0	80.0	
<b>2010-02-18 06:00:00</b>	90.0	85.0	85.0	85.0	90.0	90.0	90.0	90.0	
<b>2009-11-08 09:00:00</b>	97.0	97.0	96.0	99.0	97.0	94.0	99.0	97.0	
<b>2009-10-27 18:00:00</b>	Nan	95.0	Nan	99.0	97.0	95.0	99.0	97.0	
<b>2009-02-21 03:00:00</b>	91.0	93.0	85.0	94.0	95.0	90.0	92.0	90.0	

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Ostankino
<b>2009-02-19 21:00:00</b>	89.0	84.0	83.0	94.0	95.0	91.0	93.0	85.0	85.0
<b>2008-12-28 09:00:00</b>	77.0	89.0	90.0	94.0	91.0	47.0	93.0	84.0	84.0
<b>2008-12-14 21:00:00</b>	98.0	56.0	92.0	97.0	88.0	87.0	93.0	91.0	91.0
<b>2008-12-08 21:00:00</b>	74.0	85.0	94.0	94.0	94.0	92.0	98.0	96.0	96.0
<b>2008-01-08 09:00:00</b>	85.0	85.0	80.0	90.0	95.0	85.0	90.0	80.0	80.0
<b>2007-07-30 21:00:00</b>	77.0	77.0	81.0	85.0	82.0	86.0	82.0	47.0	47.0
<b>2006-12-21 15:00:00</b>	96.0	96.0	85.0	96.0	96.0	41.0	93.0	90.0	90.0
<b>2006-11-07 03:00:00</b>	80.0	90.0	93.0	95.0	93.0	91.0	94.0	55.0	55.0
<b>2006-10-09 03:00:00</b>	97.0	96.0	95.0	96.0	92.0	79.0	99.0	92.0	92.0
<b>2005-09-03 00:00:00</b>	93.0	96.0	93.0	48.0	93.0	99.0	100.0	95.0	95.0

In [165]:

```
# Создадим список координат ячеек, содержащих значения, определённые как ошибки
list_error_coords = df_tmp51.dropna(subset=['cross_check']).error_at.tolist()
# list_error_coords
```

In [166...]

```
# Восстановим исходное состояние df_tmp1
df_tmp51 = dict_df_parameters[param_df_name].copy(deep=True)
```

In [167...]

```
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
        # Преобразуем координату столбца и присваиваем ячейке NaN
        df_tmp51.at[error_coords[0], PARAMETER51+'#'+ station] = np.nan
for error in list_error_at: # по списку координат выявленных выше значений, аномально не совпадающих с расчётными
    df_tmp51.at[error[0], PARAMETER51 + '#' + error[1]] = np.nan
# df_tmp51
```

In [168...]

```
# Проверим, все ли ошибки заменены на NaN
# Количество нeNaN значений в df_tmp51 по координатам, указанным в списках list_error_coords и list_error_at

counter_notna1 = 0 # счётчик нeNaN значений
for error_coords in list_error_coords: # по списку координат ошибочных значений
    for station in error_coords[1]: # перечню метеостанций в каждом списке координат ошибочных значений для станций
        # подсчитаем количество нeNaN значений и прибавим их к счётчику нeNaN значений.
        counter_notna1 += np.sum(pd.notna(df_tmp51.at[error_coords[0], PARAMETER51+'#'+ station]))
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna1}')

# df_tmp51
```

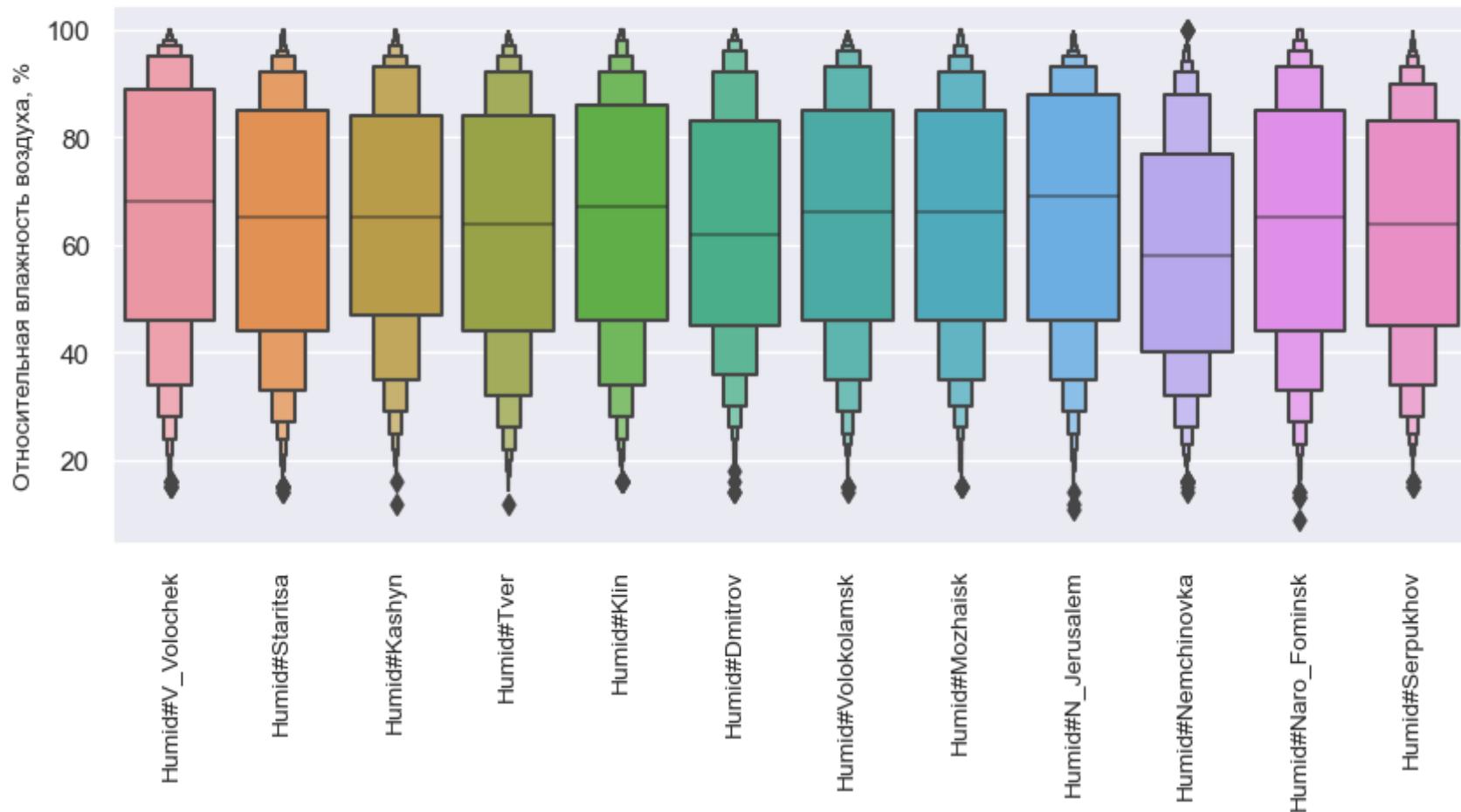
По результатам удаления выявленных аномальных выбросов осталось значений: 0

### Визуализируем архив относительной влажности воздуха (Humid) по сезонам после удаления ошибок

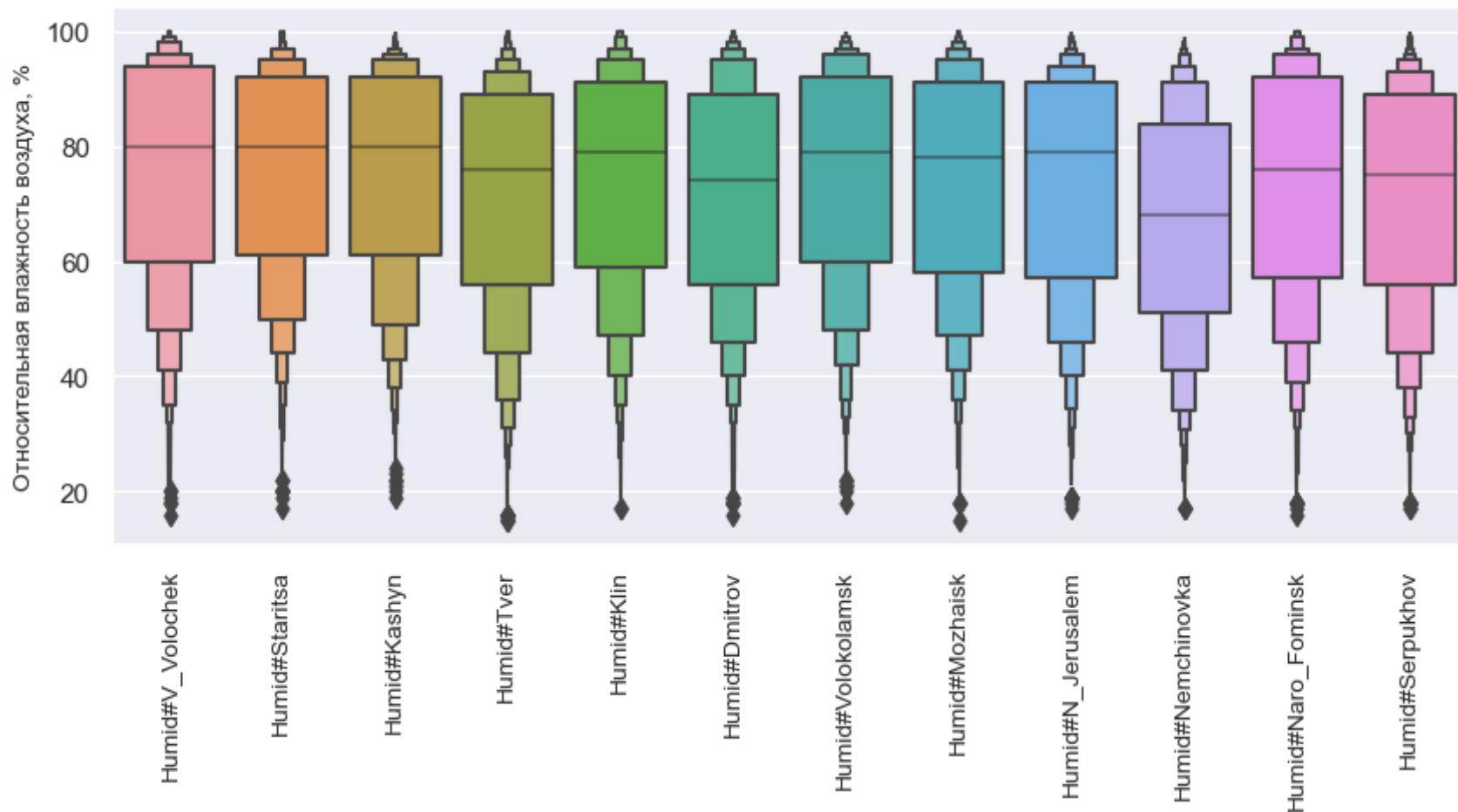
In [169...]

```
# В цикле выведем графики относительной влажности воздуха по метеостанциями в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp51).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp51[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Относительная влажность воздуха, %', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER51} в разрезе метеостанций после удаления ошибок:\n{season_name}')
plt.show()
```

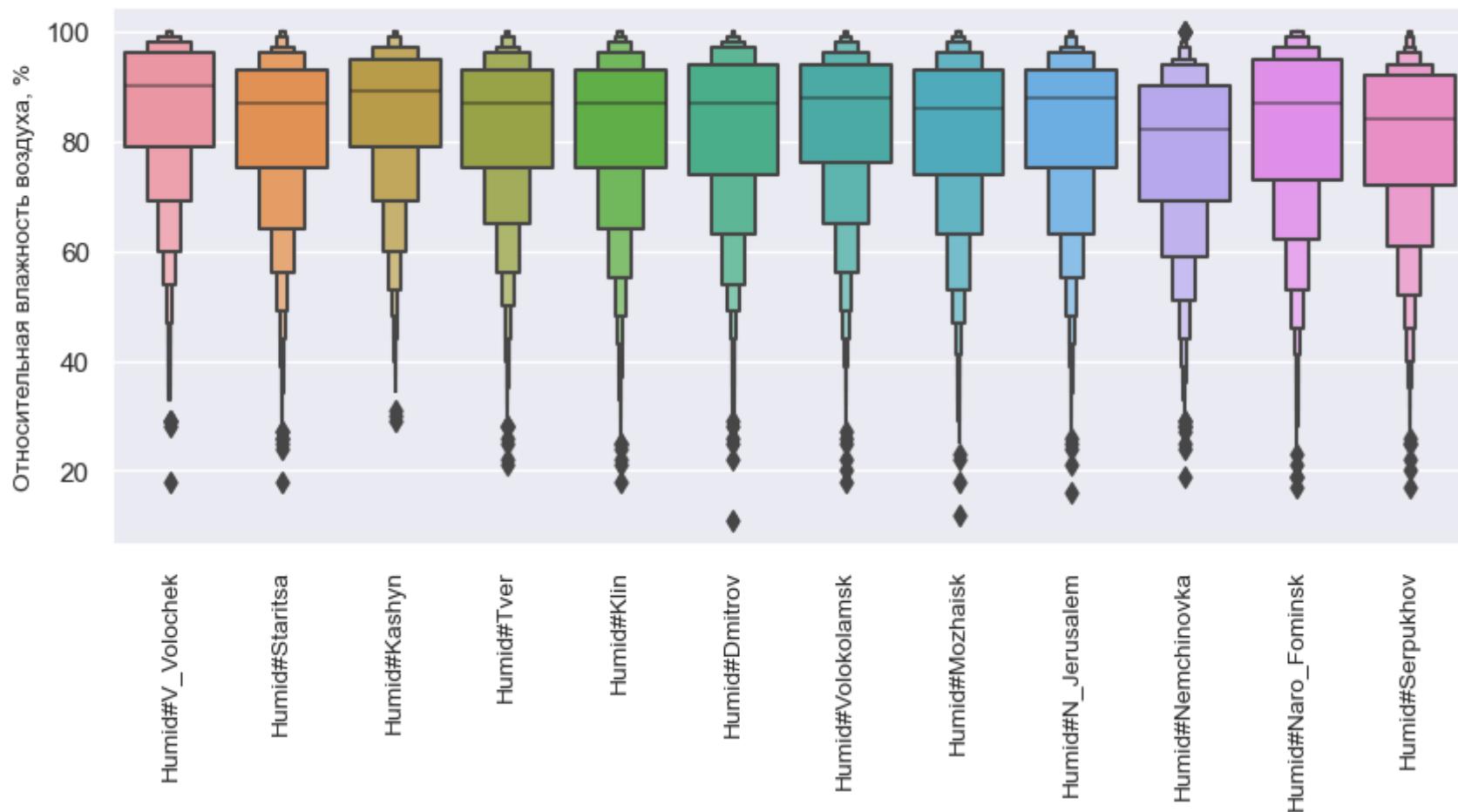
Распределение значений Humid в разрезе метеостанций после удаления ошибок:  
spring



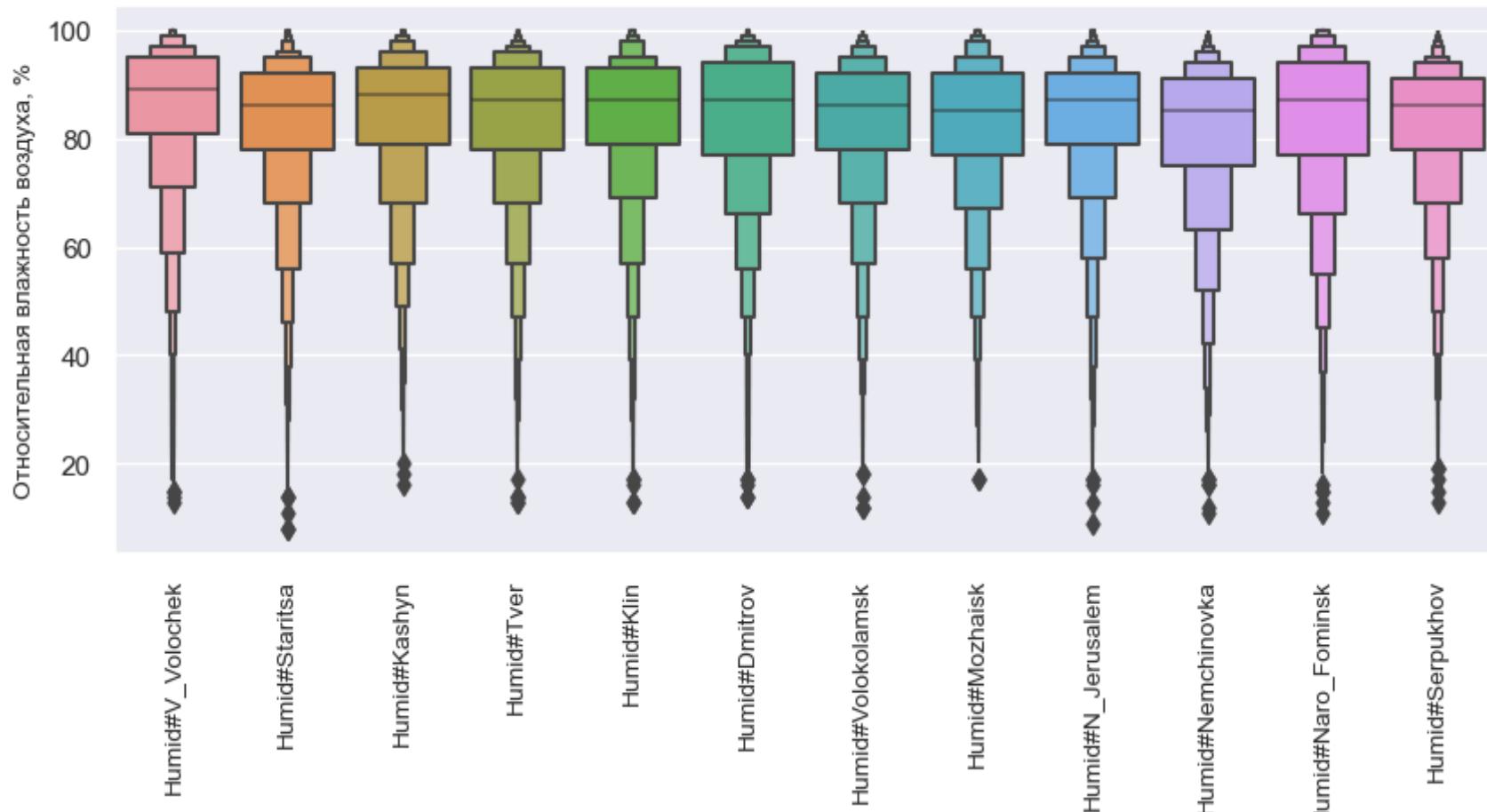
Распределение значений Humid в разрезе метеостанций после удаления ошибок:  
summer



Распределение значений Humid в разрезе метеостанций после удаления ошибок:  
autumn



## Распределение значений Humid 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 [170]: print(f'Минимальное значение: {np.nanmin(df_tmp51)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp51)},\n'
      f'Средняя: {np.nanmean(df_tmp51)},\n'
      f'Медиана: {np.nanmedian(df_tmp51)})')
```

Минимальное значение: 8.0,  
Максимальное значение: 100.0,  
Средняя: 77.42200389504082,  
Медиана: 84.0

## Сохраним очищенные данные в файл параметров

In [171...]

```
# # 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_'+PARAMETER51 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER51].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER51}.csv'  
)  
print('DONE!')  
  
df_Humid.csv -> DONE!
```

## 5.1.2. Восстановление "сплошных" NaN методом средней между соседними моментами наблюдения для показателя относительной влажности воздуха (Humid)

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

In [172...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp51  
# построчно подсчитаем сумму количеству NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp51.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp51.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 164

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

In [173...]

```
# Создадим список DateTime индексов строк со сплошными NaN  
# Выбираем из датафрейма строки, где количество NaN равно количеству столбцов - то есть сплошные NaN  
list_time_total_nans = df_tmp51[df_tmp51.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp51.keys())].index.tolist()
```

In [174...]

```
# По списку  
for idx in list_time_total_nans[:-1]: # кроме самого раннегоTimeStamp, для которого нельзя вычислить start_date  
    # определяем начало и конец временного интервала  
    start_time = idx - pd.Timedelta('3H')  
    end_time = idx + pd.Timedelta('3H')  
    while sum(df_tmp51.loc[start_time].notna()) == 0: # Пока ряд от start_time тоже состоит из сплошных NaN  
        start_time = start_time - pd.Timedelta('3H') # Уменьшаем start_time на 3 часа  
    while sum(df_tmp51.loc[end_time].notna()) == 0: # Пока ряд от end_time тоже состоит из сплошных NaN  
        end_time = end_time + pd.Timedelta('3H') # Увеличиваем end_time на 3 часа  
  
    # по ряду сплошных NaN заменяем NaN на средние значения их соседних двух рядов  
    for label in df_tmp51.loc[idx].index:  
        df_tmp51.at[idx, label] = np.mean([df_tmp51.at[start_time, label], df_tmp51.at[end_time, label]])
```

In [175...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp51  
# построчно подсчитаем сумму количества NaN,  
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количество таких строк  
all_nans_count = sum(df_tmp51.apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp51.keys()))  
print(f'Количество строк со сплошными NaN равно {all_nans_count}')
```

Количество строк со сплошными NaN равно 1

Всё верно, это самая начальная строка. Она должна остаться

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

In [176...]

```
# Проверим, все ли ошибки заменены на NaN  
# Количество нeNaN значений в df_tmp51 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]: # перечень 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_tmp51.at[error_coords[0], PARAMETER51+'#'+ station]))
print(f'По результатам удаления выявленных аномальных выбросов осталось значений: {counter_notna1}')

# df_tmp51
```

По результатам удаления выявленных аномальных выбросов осталось значений: 0

Получившееся значение показывает количество исправленных ошибочных значений, там где был применён метод восстановления сплошных NaN. В данном случае, исправлений индивидуальных ошибочных значений при заполнении сплошных NaN не произошло.

### 5.1.3. Подбор модели для восстановления пропущенных и удалённых значений показателя относительной влажности воздуха, а также для моделирования значений для искомой точки.

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

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

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

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

Ожидаемые выходные данные:

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

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

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

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

Создадим набор данных для обучения и валидации модели IDW.

1. Используем данные в df\_tmp41.
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 [177]: # Создаём датафрейм для валидации модели IDW
df_test = df_tmp51.dropna(how='any')
```

```
In [178]: df_test.info()
df_test.shape

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 27339 entries, 2022-06-09 03:00:00 to 2007-09-26 15:00:00
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  -- 
 0   Humid#V_Volochek    27339 non-null   float64
 1   Humid#Staritsa      27339 non-null   float64
 2   Humid#Kashyn        27339 non-null   float64
 3   Humid#Tver          27339 non-null   float64
 4   Humid#Klin          27339 non-null   float64
 5   Humid#Dmitrov       27339 non-null   float64
 6   Humid#Volokolamsk   27339 non-null   float64
 7   Humid#Mozhaisk      27339 non-null   float64
 8   Humid#N_Jerusalem   27339 non-null   float64
 9   Humid#Nemchinovka   27339 non-null   float64
 10  Humid#Naro_Fominsk 27339 non-null   float64
 11  Humid#Serpukhov     27339 non-null   float64
dtypes: float64(12)
memory usage: 2.7 MB
(27339, 12)
```

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

In [179...]

```
# 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[179]: array([ 0, 1, 2, ..., 27336, 27337, 27338])

Out[179]: array([5, 4, 0, ..., 6, 9, 4])

Out[179]: array([[ 0, 5],
 [ 1, 4],
 [ 2, 0],
 ...,
 [27336, 6],
 [27337, 9],
 [27338, 4]])

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 [180...]

```
# 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[180]: (16403, 2)
Out[180]: (16403,)
Out[180]: (10936, 2)
Out[180]: (10936,)
```

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

```
In [181... 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_=PARAMETER41,
                             station_=df_test.keys()[y][len(PARAMETER41)+1:],
                             df_dists_= df_station_dists,
                             power_=power)
        for x, y in x_train
    ]

    # Расчитываем метрики качества
    max_e_idw_tr = max_error(y_train, y_predict_idw_tr)
    mae_idw_tr = mean_absolute_error(y_train, y_predict_idw_tr)
```

```
mse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr)
rmse_idw_tr = mean_squared_error(y_train, y_predict_idw_tr, squared=False)
r2_idw_tr = r2_score(y_train, y_predict_idw_tr)

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

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

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

ОБУЧАЮЩИЙ ДАТАСЕТ, IDW:

```
Elapsed time=00:01:23
Текущие значения: iterations=1, power=3.25,
Максимальная ошибка (MAX_E) IDW = 43.1379167
Средняя абсолютная ошибка (MAE) IDW = 4.7535068
Средний квадрат ошибки (MSE) IDW = 45.4952294
Средняя квадратическая ошибка (RMSE) IDW = 6.7450151
Коэффициент детерминации (R2) IDW = 0.8727398
```

```
Elapsed time=00:02:47
Текущие значения: iterations=2, power=3.5,
Максимальная ошибка (MAX_E) IDW = 43.1782840
Средняя абсолютная ошибка (MAE) IDW = 4.7556996
Средний квадрат ошибки (MSE) IDW = 45.5295025
Средняя квадратическая ошибка (RMSE) IDW = 6.7475553
Коэффициент детерминации (R2) IDW = 0.8726439
```

ЛУЧШИЕ значения: power=3.25,  
Максимальная ошибка (MAX\_E) IDW = 43.1379167  
Средняя абсолютная ошибка (MAE) IDW = 4.7535068  
Средний квадрат ошибки (MSE) IDW = 45.4952294  
Средняя квадратическая ошибка (RMSE) IDW = 6.7450151  
Коэффициент детерминации (R2) IDW = 0.8727398

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

Применим эту степень для валидационного массива.

## Валидационный датасет

In [182...]

```
# 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=PARAMETER41,
                          station=df_test.keys()[y][len(PARAMETER41)+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 = 54.7698331  
 Средняя абсолютная ошибка (MAE) IDW = 4.8246866  
 Средний квадрат ошибки (MSE) IDW = 46.8684308  
 Средняя квадратическая ошибка (RMSE) IDW = 6.8460522  
 Коэффициент детерминации (R2) IDW = 0.8697805

## ВЫВОД

Модель средней, взвешенной по обратным расстояниям, даёт хорошие, но не лучшие метрики качества.

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

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

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

In [183...]

```
# 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[183]:

LoE      NaN

station	LoE	NaN
<b>V_Volochek</b>	34.566667	57.583333
<b>Staritsa</b>	34.933333	56.500000
<b>Kashyn</b>	37.583333	57.350000
<b>Tver</b>	35.922000	56.857300
<b>Klin</b>	36.716667	56.333333
<b>Dmitrov</b>	37.533333	56.366667
<b>Volokolamsk</b>	35.933333	56.016700
<b>Mozhaisk</b>	36.000000	55.516700
<b>N_Jerusalem</b>	36.816667	55.900000
<b>Nemchinovka</b>	37.350000	55.716667
<b>Naro_Fominsk</b>	36.700000	55.383333
<b>Serpukhov</b>	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 [184...]

```
# 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 [185...]:
if np.isnan(np.array(y_predict_kriging_tr)).sum() == 0:
    max_e_kriging_tr = max_error(y_train, y_predict_kriging_tr)
    mae_kriging_tr = mean_absolute_error(y_train, y_predict_kriging_tr)
    mse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr)
    rmse_kriging_tr = mean_squared_error(y_train, y_predict_kriging_tr, squared=False)
    r2_kriging_tr = r2_score(y_train, y_predict_kriging_tr)
else:
    max_e_kriging_tr = np.nan
    mae_kriging_tr = np.nan
    mse_kriging_tr = np.nan
    rmse_kriging_tr = np.nan
    r2_kriging_tr = np.nan

print('ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ')
print(f'Elapsed time={time_formatted}\n'
      f'Значения метрик:\n'
      f'R2={r2_kriging_tr:.7f}, MAX_E={max_e_kriging_tr}, MAE={mae_kriging_tr:.7f}, MSE={mse_kriging_tr}, '
      f'RMSE={rmse_kriging_tr:.7f}\n'
      )
print(f'Количество значений, которые не удалось предсказать: {pd.isna([y_predict_kriging_tr]).sum()}\n'
      f'Это составляет {pd.isna([y_predict_kriging_tr]).sum()/len(y_predict_kriging_tr):.4%} от обучающего массива данных')

```

ОБУЧАЮЩИЙ ДАТАСЕТ, КРИГИНГ  
Elapsed time=00:05:55  
Значения метрик:  
R2=nan, MAX\_E=nan, MAE=nan, MSE=nan, RMSE=nan

Количество значений, которые не удалось предсказать: 1252  
Это составляет 7.6328% от обучающего массива данных

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

In [186...]

```
# 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 [187...]

```
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 = 47.9244897, Kriging - IDW = 4.7462057  
Средняя абсолютная ошибка (MAE) Kriging = 4.7483893, Kriging - IDW = -0.0073103  
Средний квадрат ошибки (MSE) Kriging = 44.9657681, Kriging - IDW = -0.5637344  
Средняя квадратическая ошибка (RMSE) Kriging = 6.7056520, Kriging - IDW = -0.0419033  
Коэффициент детерминации (R2) Kriging = 0.8754627, Kriging - IDW = 0.0028188

## Валидационный датасет

Посмотрим, как модель будет отрабатывать на валидационной выборке.

In [188...]

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

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

```

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

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

```

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

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

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

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

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

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

y_predict_kriging_vld = np.array(y_predict_kriging_vld)

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

```

In [189...]

```

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

```

```

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

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

```

Elapsed time=00:03:41

Значения метрик:

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

ВАЛИДАЦИОННЫЙ ДАТАСЕТ, КРИГИНГ

Количество значений, которые не удалось предсказать: 883

Это составляет 8.0743% от валидационного массива данных

In [190...]

```

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 [191...]

```

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 = 57.7976771, Kriging - IDW = 3.0278440  
Средняя абсолютная ошибка (MAE) Kriging = 4.7550585, Kriging - IDW = -0.0696281  
Средний квадрат ошибки (MSE) Kriging = 45.6763321, Kriging - IDW = -1.1920986  
Средняя квадратическая ошибка (RMSE) Kriging = 6.7584268, Kriging - IDW = -0.0876255  
Коэффициент детерминации (R2) Kriging = 0.8740892, Kriging - IDW = 0.0043088

## ВЫВОД

Модель кригинга в данном случае не может дать 100% результат, и не может предсказать все значения. Это происходит из-за того, что 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. Тем не менее, модель кригинга показывает лучшие метрики (кроме максимальной ошибки) по сравнению с IDW, как на обучающем, так и на валидационном датасетах.

### 5.1.3.3. 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

В случае имеющихся обучающего и валидационного датасетов модель кригинга не смогла предсказать только одно значение на обучающем датасете и предсказала все значения на валидационном датасете. Проверять совместную работу двух моделей не имеет смысла.

In [192...]

```
# 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 срепень для IDW

In [193...]

```
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 результат списком
    y_predict_idw_tr_shrunk = [
        inverse_distance_avg(row=df_test.iloc[x],
                             param_=PARAMETER41,
                             station_=df_test.keys()[y][len(PARAMETER41)+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=00:00:05

Текущие значения: iterations=1, power=2.0,  
Максимальная ошибка (MAX\_E) IDW = 32.5513883  
Средняя абсолютная ошибка (MAE) IDW = 4.6272916  
Средний квадрат ошибки (MSE) IDW = 43.7158362  
Средняя квадратическая ошибка (RMSE) IDW = 6.6117952  
Коэффициент детерминации (R2) IDW = 0.8503131

Elapsed time=00:00:11

Текущие значения: iterations=2, power=2.25,  
Максимальная ошибка (MAX\_E) IDW = 32.8713288  
Средняя абсолютная ошибка (MAE) IDW = 4.6217025  
Средний квадрат ошибки (MSE) IDW = 43.6436758  
Средняя квадратическая ошибка (RMSE) IDW = 6.6063360  
Коэффициент детерминации (R2) IDW = 0.8505602

Elapsed time=00:00:17

Текущие значения: iterations=3, power=2.5,  
Максимальная ошибка (MAX\_E) IDW = 33.1738234  
Средняя абсолютная ошибка (MAE) IDW = 4.6197311  
Средний квадрат ошибки (MSE) IDW = 43.6361041  
Средняя квадратическая ошибка (RMSE) IDW = 6.6057629  
Коэффициент детерминации (R2) IDW = 0.8505861

Elapsed time=00:00:22

Текущие значения: iterations=4, power=2.75,  
Максимальная ошибка (MAX\_E) IDW = 33.4590903  
Средняя абсолютная ошибка (MAE) IDW = 4.6225505  
Средний квадрат ошибки (MSE) IDW = 43.6890461  
Средняя квадратическая ошибка (RMSE) IDW = 6.6097690  
Коэффициент детерминации (R2) IDW = 0.8504048

ЛУЧШИЕ значения: power=2.5,

Максимальная ошибка (MAX\_E) IDW = 33.1738234  
Средняя абсолютная ошибка (MAE) IDW = 4.6197311  
Средний квадрат ошибки (MSE) IDW = 43.6361041  
Средняя квадратическая ошибка (RMSE) IDW = 6.6057629  
Коэффициент детерминации (R2) IDW = 0.8505861

*Опробуем новое значение лучшей степени для IDW на сокращённом валидационном датасете*

```
In [194...]
# 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_=PARAMETER41,
        station_=df_test.keys()[y][len(PARAMETER41)+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 = 47.5691601  
Средняя абсолютная ошибка (MAE) IDW = 4.7822967  
Средний квадрат ошибки (MSE) IDW = 48.7791290  
Средняя квадратическая ошибка (RMSE) IDW = 6.9842057  
Коэффициент детерминации (R2) IDW = 0.8428833

### 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 [195...]
# Восстановим y_predict для лучшего R2 IDW
y_predict_idw_tr_shrunk = [
    inverse_distance_avg(row=df_test.iloc[x],
        param_=PARAMETER41,
```

```
station_=df_test.keys()[y][len(PARAMETER41)+1:],
df_dists_= df_station_dists,
power_=best_power_idw_tr_shrunk)
for x, y in x_train_idw_shrunk
]
```

In [196...]

```
# 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 [197...]

```
# 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 [198...]

```
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 = 47.9244897  
Средняя абсолютная ошибка (MAE) IDW = 4.7385692  
Средний квадрат ошибки (MSE) IDW = 44.8642782  
Средняя квадратическая ошибка (RMSE) IDW = 6.6980802  
Коэффициент детерминации (R2) IDW = 0.8745047

ОБЩИЙ ВАЛИДАЦИОННЫЙ ДАТАСЕТ, Кригинг + IDW:

Максимальная ошибка (MAX\_E) IDW = 57.7976771  
Средняя абсолютная ошибка (MAE) IDW = 4.7572578  
Средний квадрат ошибки (MSE) IDW = 45.9268597  
Средняя квадратическая ошибка (RMSE) IDW = 6.7769359  
Коэффициент детерминации (R2) IDW = 0.8723965

Для сравнения: ВАЛИДАЦИОННЫЙ ДАТАСЕТ, только IDW:

Максимальная ошибка (MAX\_E) IDW = 54.7698331  
Средняя абсолютная ошибка (MAE) IDW = 4.8246866  
Средний квадрат ошибки (MSE) IDW = 46.8684308  
Средняя квадратическая ошибка (RMSE) IDW = 6.8460522  
Коэффициент детерминации (R2) IDW = 0.8697805

## ВЫВОД

И модель IDW, и модель кригинга дают в целом хорошие, но не лучшие результаты. Совместное применение двух моделей позволяет предсказать значения там, где их не может предсказать модель кригинга, и даёт метрики качества лучше, чем применение исключительно модели IDW

### 5.1.4. Применение выбранных моделей для исправления, восстановления данных и получения предиктов показателя относительной влажности воздуха (Humid) для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

In [199...]

```
# Добавим в df_tmp51 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 PARAMETER51#Chashnikovo и PARAMETER51#Rfrnce_point
df_tmp51 = (df_tmp51.
             assign(col_name1 = np.nan,
                   col_name2 = np.nan).
             rename(columns={"col_name1": PARAMETER51+'#'+ 'Chashnikovo',
                            "col_name2": PARAMETER51+'#'+ 'Rfrnce_point'}))
         )

df_tmp51.sample(3, random_state=56)

```

Out[199]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#...
2014-02-22 03:00:00	75.0	75.0	73.0	75.0	79.0	77.0	74.0	77.0	77.0
2015-05-16 03:00:00	93.0	93.0	96.0	94.0	95.0	96.0	94.0	92.0	92.0
2020-06-18 18:00:00	47.0	46.0	64.0	48.0	62.0	61.0	35.0	42.0	42.0

In [200...]

```

# Добавим в архив параметров P_sea столбец для Чашниково и будущей центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER51#Chashnikovo и PARAMETER51#Rfrnce_point
dict_df_parameters['df_'+PARAMETER51] = (dict_df_parameters['df_'+PARAMETER51].
                                         assign(col_name1 = np.nan,
                                               col_name2 = np.nan).
                                         rename(columns={"col_name1": PARAMETER51+'#'+ 'Chashnikovo',
                                                        "col_name2": PARAMETER51+'#'+ 'Rfrnce_point'}))
                                         )

dict_df_parameters['df_'+PARAMETER51].sample(3, random_state=56)

```

Out[200]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid
<b>2014-02-22 03:00:00</b>	75.0	75.0	73.0	75.0	79.0	77.0	74.0	77.0	
<b>2015-05-16 03:00:00</b>	93.0	93.0	96.0	94.0	95.0	96.0	94.0	92.0	
<b>2020-06-18 18:00:00</b>	47.0	46.0	64.0	48.0	62.0	61.0	35.0	42.0	

In [201...]:

```
# Добавим в архив метеостанций df Чашниково и df для центральной точки,
# Создадим в нём столбец с называнием PARAMETER51
```

```
dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER51})
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER51})
                                         )

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

Out[201]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid
<b>2014-02-22 03:00:00</b>	-4.156558	-4.156558	-2.109643	768.336709	747.597791	0.812775	NaN
<b>2015-05-16 03:00:00</b>	9.181730	9.181730	10.434571	745.095535	725.921747	-1.043037	NaN
<b>2020-06-18 18:00:00</b>	27.439052	25.366130	30.241632	761.480701	743.060948	-0.492627	NaN

Out[201]:

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

## Перенесение уже исправленных сплошных NaN в архивы

In [202...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER51] = df_tmp51.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[:, PARAMETER51] = df_tmp51.loc[:, PARAMETER51 + '#' + name_df[3:]]
```

In [203...]

```
# Подсчитаем количество строк со "сплошными" NaN dict_df_parameters['df_'+PARAMETER51]
# 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_'+PARAMETER51]
    .apply(lambda x: sum(x.isna()), axis=1)==len(dict_df_parameters['df_'+PARAMETER51].keys()))
)

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

Количество строк со сплошными NaN равно 1

In [204...]

```
# Подсчитаем количество строк со "сплошными" NaN в df_tmp51
# построчно подсчитаем сумму количества NaN,
# и если оно количеству столбцов в DF (boolean), подсчитаем через сумму количества таких строк
all_nans_count = sum(
    df_tmp51
    .apply(lambda x: sum(x.isna()), axis=1)==len(df_tmp51.keys()))
)

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

Количество строк со сплошными NaN равно 1

## 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.

In [205...]

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

# Подсчитаем количество NaN в df_tmp51 и присвоим их переменной old_nan_count
# np.isnan(df_tmp51).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
new_nan_count = np.sum(np.isnan(df_tmp51).sum()) # результат всегда будет >= 0
# Определим начальное значение для переменной для обновления количества оставшихся NaN
old_nan_count = new_nan_count
counter = 0 # определим счётчик

# заменим значения в архивах, на исправленные и вычисленные из данных в df_tmp51
# используем функцию row_nan_kriging_correct
# функция настроена таким образом, что при возникновении ошибки, связной с недостаточностью данных,
# осуществляется выход из функции без возврата каких бы то ни было значений.
while (old_nan_count != new_nan_count) or (counter == 0):
    counter += 1
    print (f'\nИтерация № {counter}: осталось NaN: {new_nan_count}')

    # Построчно применяем функцию обработки NaN
    df_tmp51.apply(lambda x: row_nan_kriging_correct(row=x,
                                                       name_param_=PARAMETER51
                                                       ),
                  axis=1
                  )
    old_nan_count = new_nan_count # сохраняем прежнее количество NaN в old_nan_count
    new_nan_count = np.sum(np.isnan(df_tmp51).sum()) # подсчитаем оставшиеся количество NaN

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

Итерация № 1: осталось NaN: 157314

```
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 2: осталось NaN: 10145  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 3: осталось NaN: 3234  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 4: осталось NaN: 2485
```

```
Out[205]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 5: осталось NaN: 2240  
Out[205]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 6: осталось NaN: 2110  
Out[205]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 7: осталось NaN: 2055
```

```
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 8: осталось NaN: 2021  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 9: осталось NaN: 1994  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 10: осталось NaN: 1981
```

```
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 11: осталось NaN: 1974  
  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 12: осталось NaN: 1972  
  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 13: осталось NaN: 1968
```

```
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 14: осталось NaN: 1962  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 15: осталось NaN: 1958  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 16: осталось NaN: 1955
```

```
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 17: осталось NaN: 1953  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 18: осталось NaN: 1952  
Out[205]: 2022-06-09 21:00:00    None  
           2022-06-09 18:00:00    None  
           2022-06-09 15:00:00    None  
           2022-06-09 12:00:00    None  
           2022-06-09 09:00:00    None  
           ...  
           2005-02-01 12:00:00    None  
           2005-02-01 09:00:00    None  
           2005-02-01 06:00:00    None  
           2005-02-01 03:00:00    None  
           2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Итерация № 19: осталось NaN: 1951
```

```
Out[205]: 2022-06-09 21:00:00    None  
2022-06-09 18:00:00    None  
2022-06-09 15:00:00    None  
2022-06-09 12:00:00    None  
2022-06-09 09:00:00    None  
...  
2005-02-01 12:00:00    None  
2005-02-01 09:00:00    None  
2005-02-01 06:00:00    None  
2005-02-01 03:00:00    None  
2005-02-01 00:00:00    None  
Length: 50704, dtype: object  
Elapsed time=00:21:56
```

Подсчитаем конечное количество NaN

```
In [206]: # np.isnan(df_tmp51).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp51).sum())
```

```
Out[206]: 1951
```

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

```
In [207]: # Произведём вычисление отсутствующих значений в df_tmp51 через модель IDW.  
# Заменим соответствующие значения в архивах на вновь вычисленные.  
# используем функцию row_nan_idw_correct  
  
start_time = time.time() # для замера времени выполнения кода  
  
# Построчно применяем функцию обработки NaN  
df_tmp51.apply(lambda x: row_nan_idw_correct(row_=x,  
                                              name_param_=PARAMETER51,  
                                              power_=best_power_idw_tr),  
               axis=1  
)  
  
chk_time = time.time()  
elapsed_time = chk_time - start_time
```

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

Out[207]:

```
2022-06-09 21:00:00      None
2022-06-09 18:00:00      None
2022-06-09 15:00:00      None
2022-06-09 12:00:00      None
2022-06-09 09:00:00      None
...
2005-02-01 12:00:00      None
2005-02-01 09:00:00      None
2005-02-01 06:00:00      None
2005-02-01 03:00:00      None
2005-02-01 00:00:00      None
Length: 50704, dtype: object
Elapsed time=00:00:20
```

Подсчитаем окончательное количество NaN

In [208...]

```
# np.isnan(df_tmp51).sum() выдаёт количество NaN по каждому столбцу.
# Поэтому дополнительно просуммируем полученные по столбцам значения
np.sum(np.isnan(df_tmp51).sum())
```

Out[208]:

```
14
```

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

Выведем, рандомные строки из df\_tmp51

In [209...]

```
df_tmp51.sample(7)
```

Out[209]:

	Humid#V_Volochek	Humid#Staritsa	Humid#Kashyn	Humid#Tver	Humid#Klin	Humid#Dmitrov	Humid#Volokolamsk	Humid#Mozhaisk	Humid#Chashnikovo
<b>2011-06-26 09:00:00</b>	98.000000	96.0	64.000000	93.0	95.0	63.0	95.0	92.0	92.0
<b>2009-10-30 03:00:00</b>	74.000000	74.0	86.000000	90.0	65.0	80.0	85.0	96.0	96.0
<b>2015-05-15 06:00:00</b>	99.000000	96.0	95.000000	91.0	88.0	81.0	93.0	93.0	93.0
<b>2005-07-06 18:00:00</b>	79.856489	94.0	51.719463	67.0	46.0	48.0	46.0	43.0	43.0
<b>2007-04-15 15:00:00</b>	35.000000	31.0	29.000000	36.0	33.0	37.0	33.0	39.0	39.0
<b>2010-07-05 09:00:00</b>	64.000000	68.0	62.000000	60.0	80.0	54.0	58.0	56.0	56.0
<b>2011-03-25 00:00:00</b>	89.000000	78.0	56.000000	60.0	58.0	51.0	54.0	65.0	65.0

◀ ▶

## 5.1.5. Визуализация графиков относительной влажности Humid для условной метеостанции Чашниково

In [210...]

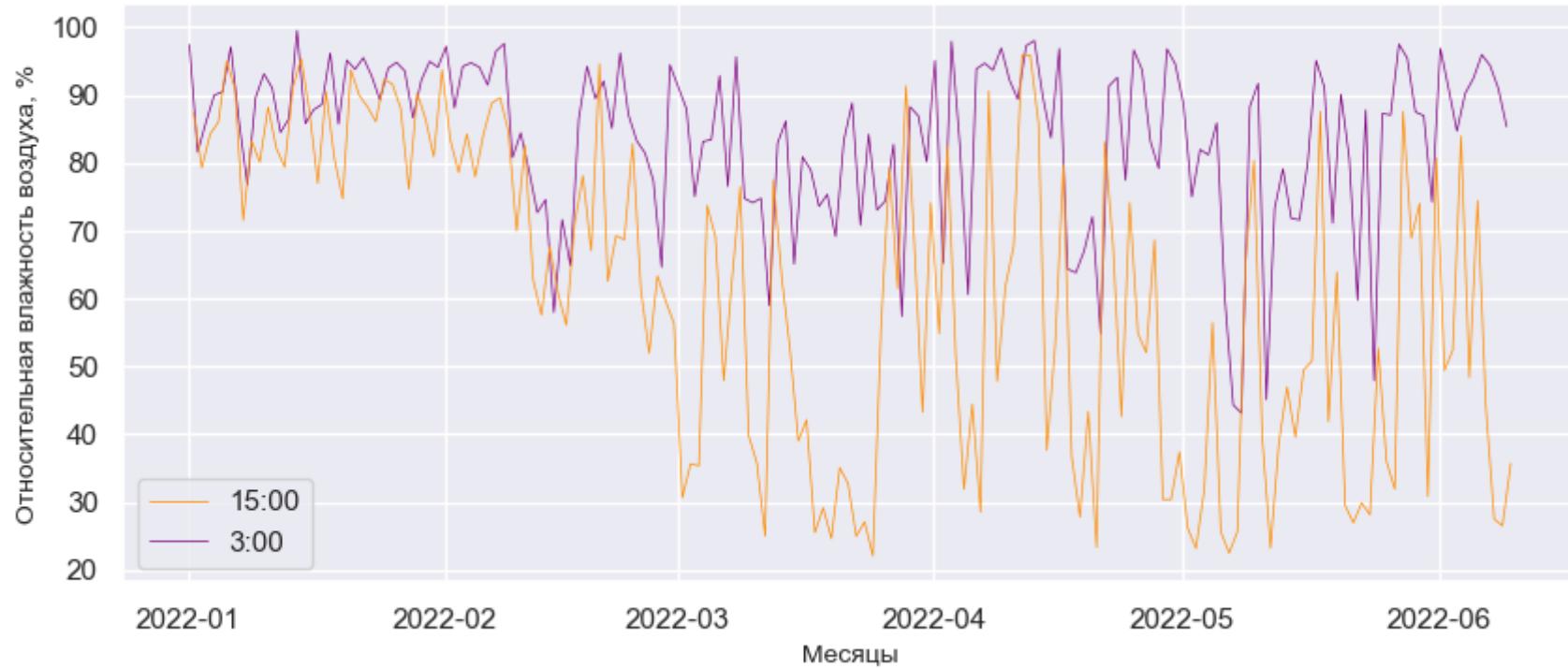
```
# Построим список годов для графиков
list_years = df_tmp51.index.year.unique().tolist()
```

In [211...]

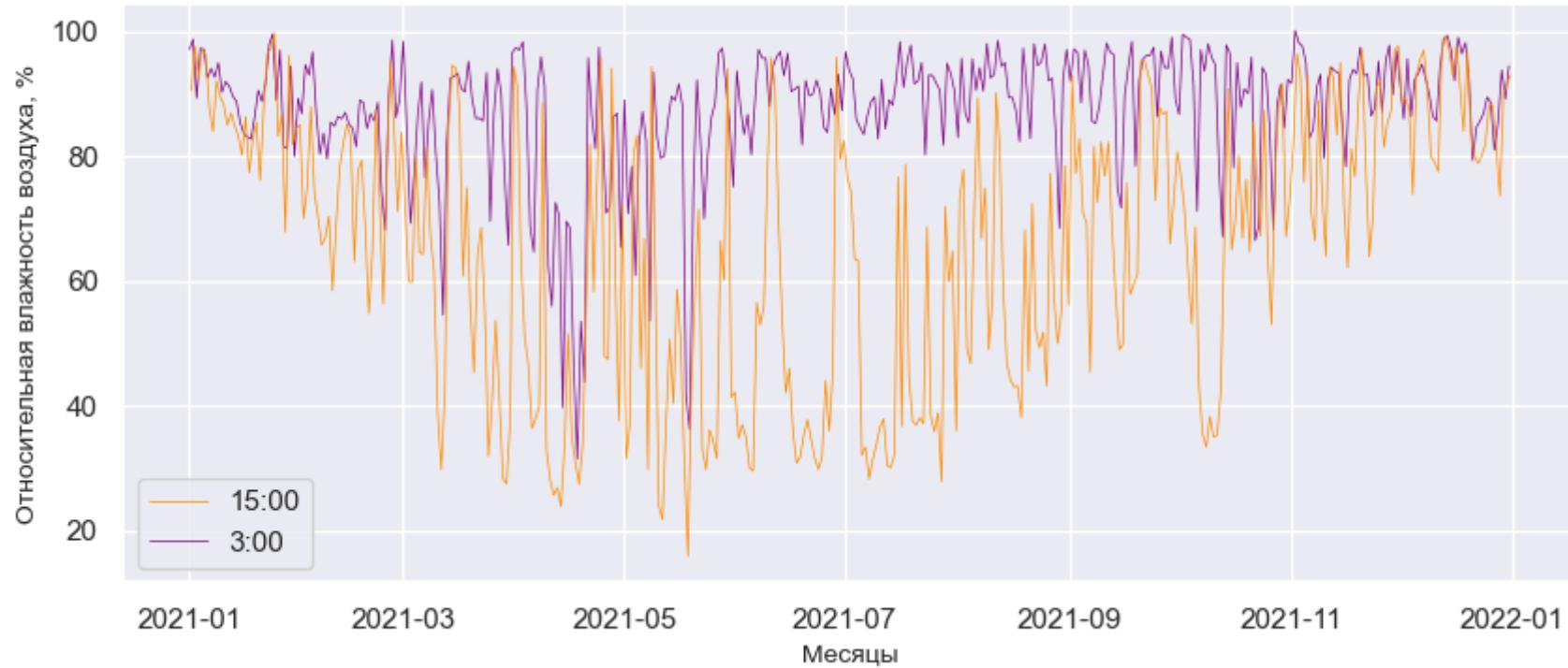
```
# Определим часы момента наблюдения для построения графиков
plot_hour1 = 3
plot_hour2 = 15
# Строим график в цикле для каждого года
for year in list_years:
```

```
data1 = dict_df_parameters['df_'+PARAMETER51][PARAMETER51+"#"+"Chashnikovo"]\n[(dict_df_parameters['df_'+PARAMETER51].index.year == year) &\n (dict_df_parameters['df_'+PARAMETER51].index.hour == plot_hour1)]\n\ndata2 = dict_df_parameters['df_'+PARAMETER51][PARAMETER51+"#"+"Chashnikovo"]\n[(dict_df_parameters['df_'+PARAMETER51].index.year == year) &\n (dict_df_parameters['df_'+PARAMETER51].index.hour == plot_hour2)]\n\nfig, ax = plt.subplots(figsize=(10, 4))\ng1 = sns.lineplot(data=data1,\n                  color='purple',\n                  linewidth=0.5,\n                  ax=ax)\ng2 = sns.lineplot(data=data2,\n                  color='darkorange',\n                  linewidth=0.5,\n                  ax=ax)\n\n# Добавляем легенду\nblue_line = mlines.Line2D([], [], color='purple', label=f'{plot_hour1}:00', linewidth=0.5)\nred_line = mlines.Line2D([], [], color='darkorange', label=f'{plot_hour2}:00', linewidth=0.5)\ndummy = ax.legend(handles=[red_line, blue_line])\n\ndummy = ax.set_ylabel('Относительная влажность воздуха, %', size=10)\ndummy = ax.set_xlabel('Месяцы', size=10)\ndummy = plt.title(f'Чашниково: Ежедневная динамика параметра {PARAMETER51} '\n                  f'на {plot_hour1} и {plot_hour2} часов, {year} год')\nplt.show()
```

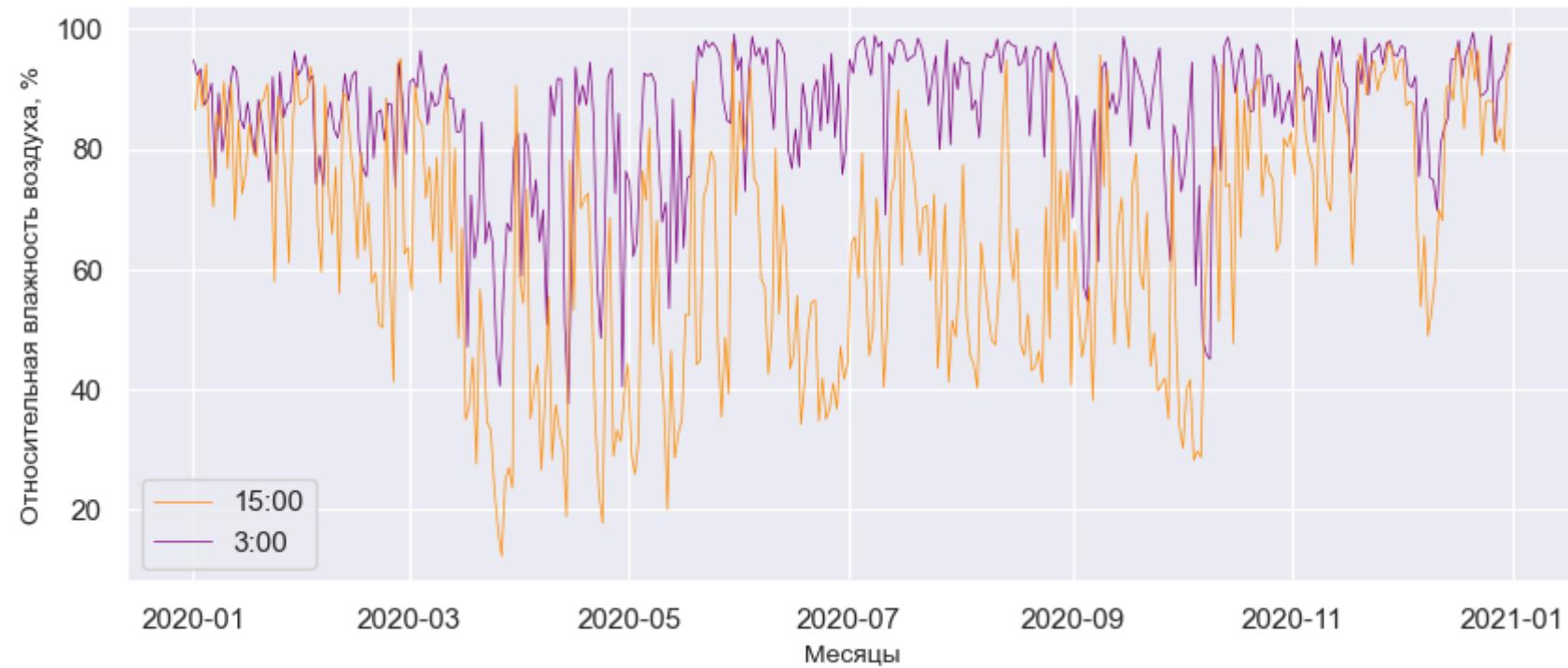
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2022 год



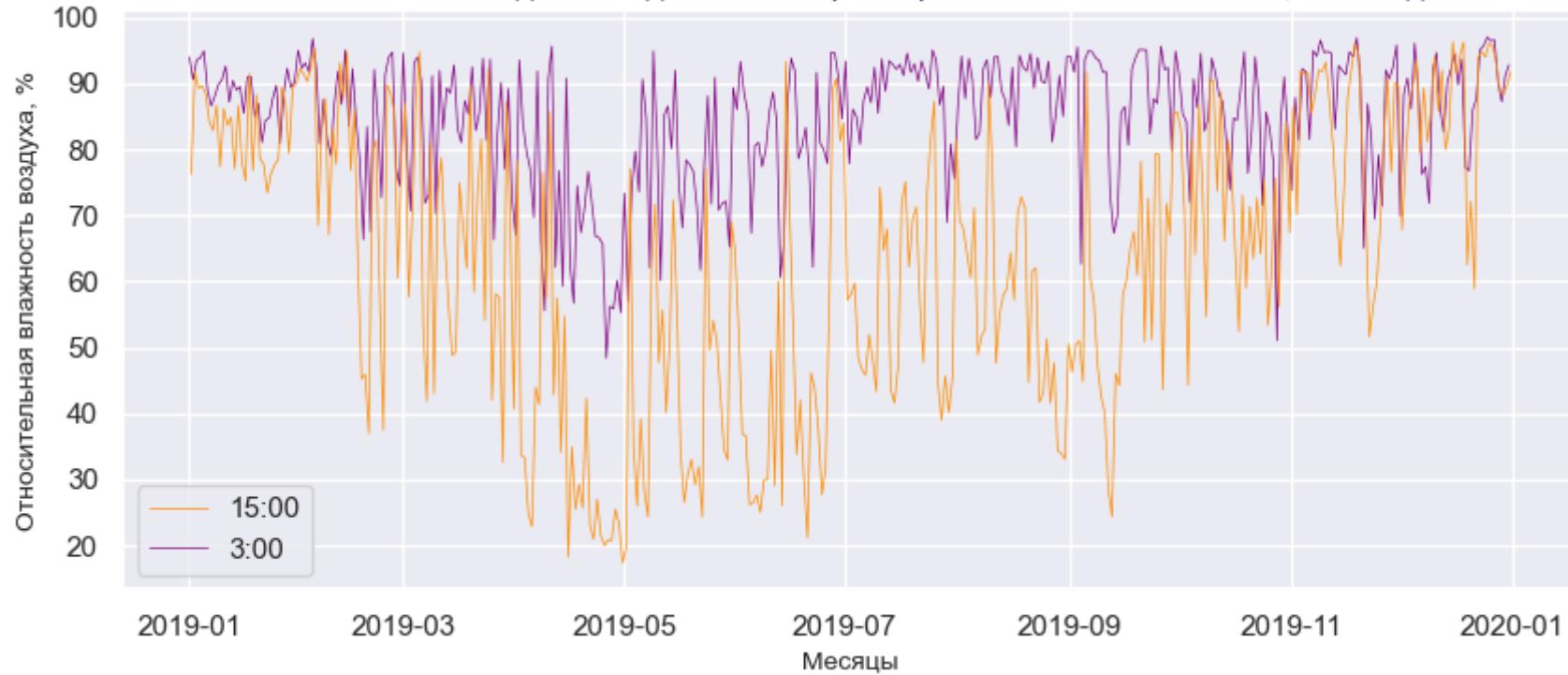
### Чашниково: Ежедневная динамика параметра Нумид на 3 и 15 часов, 2021 год



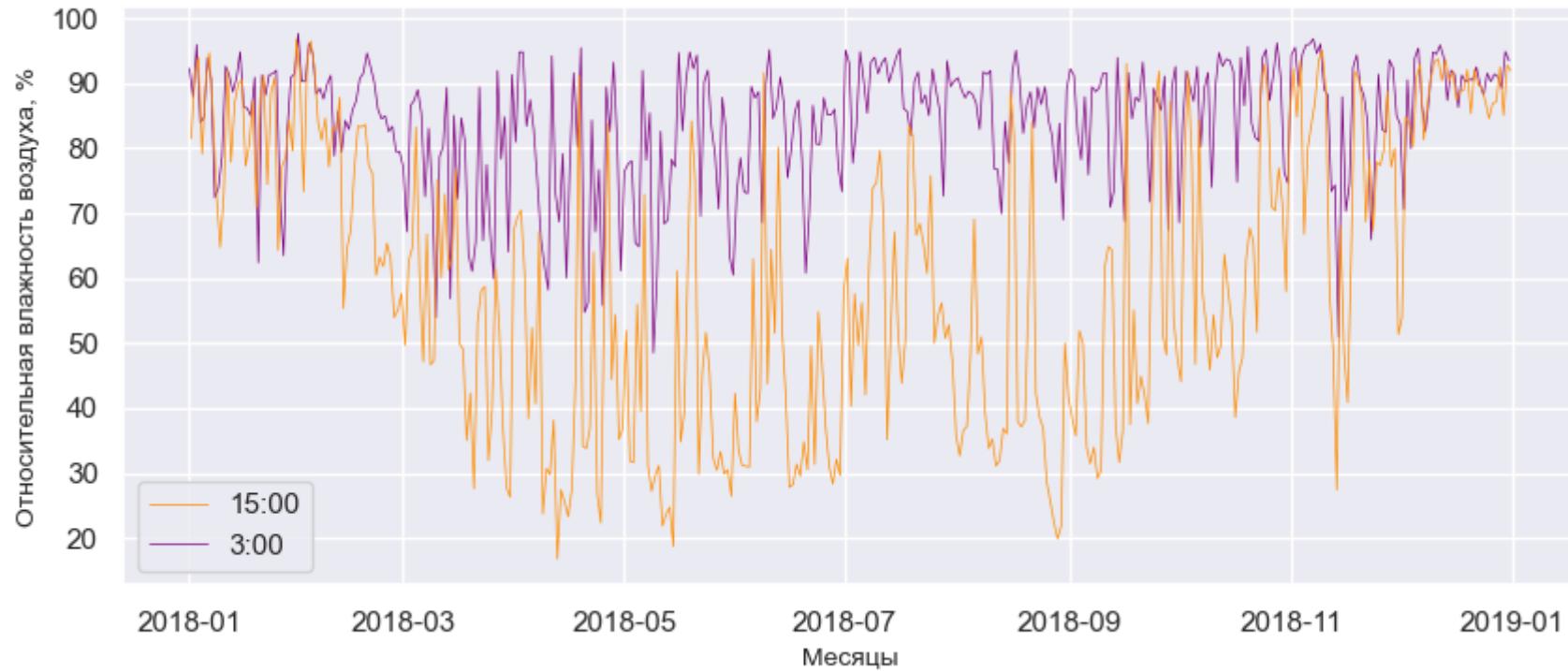
### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2020 год



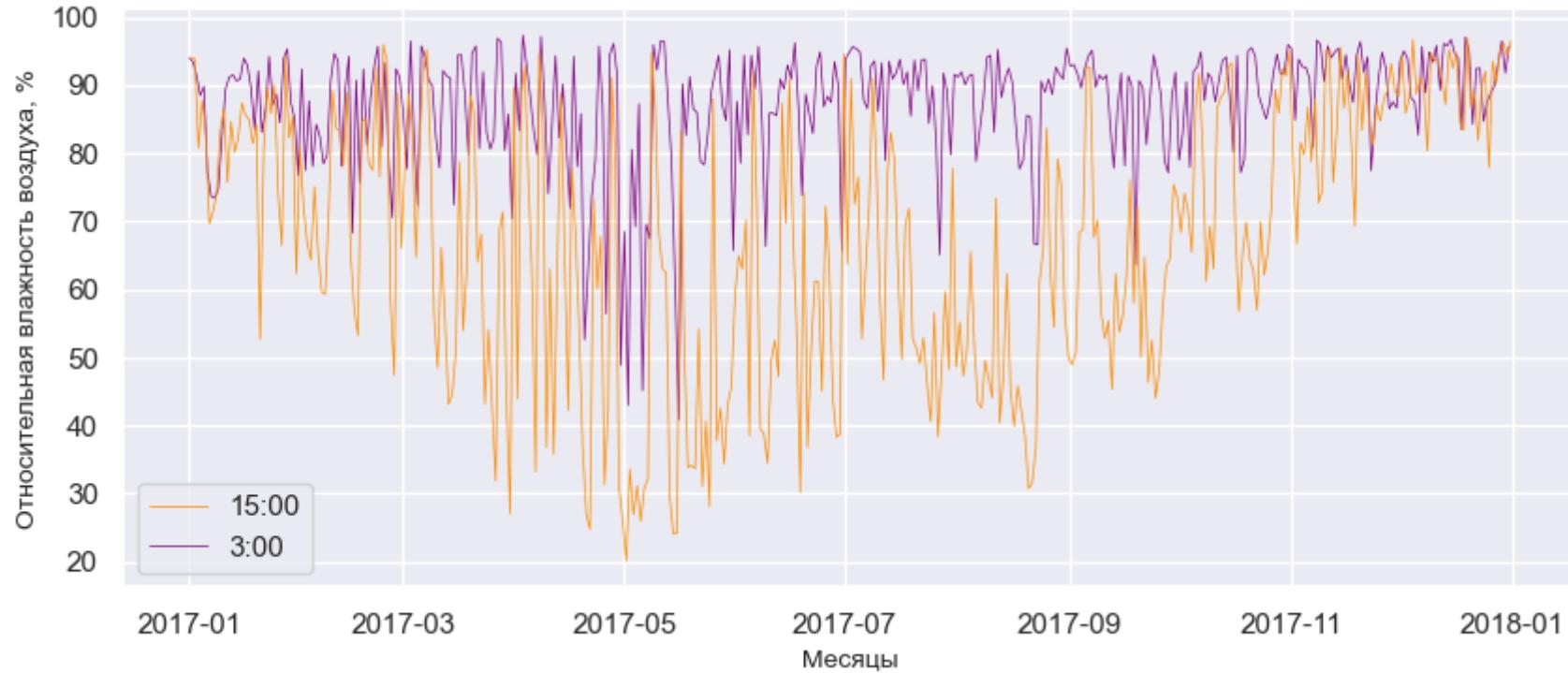
### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2019 год



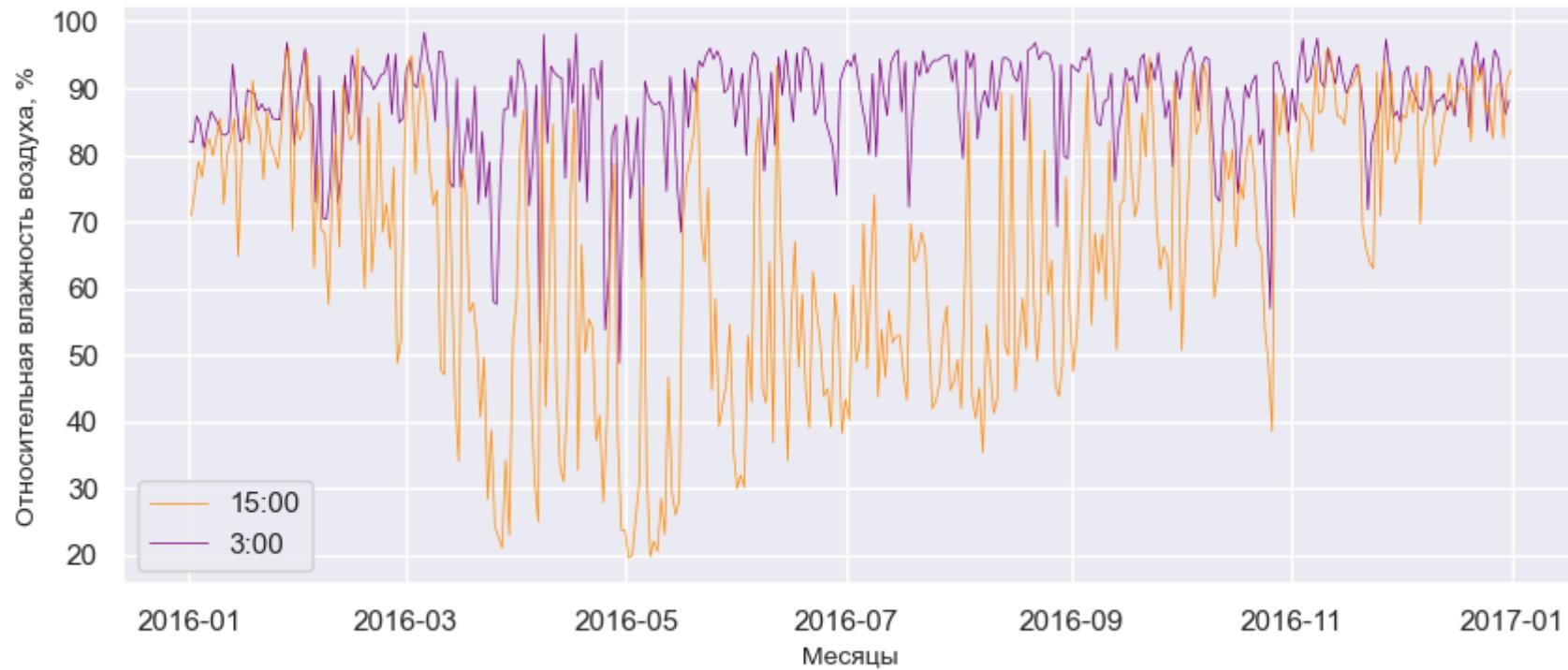
### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2018 год



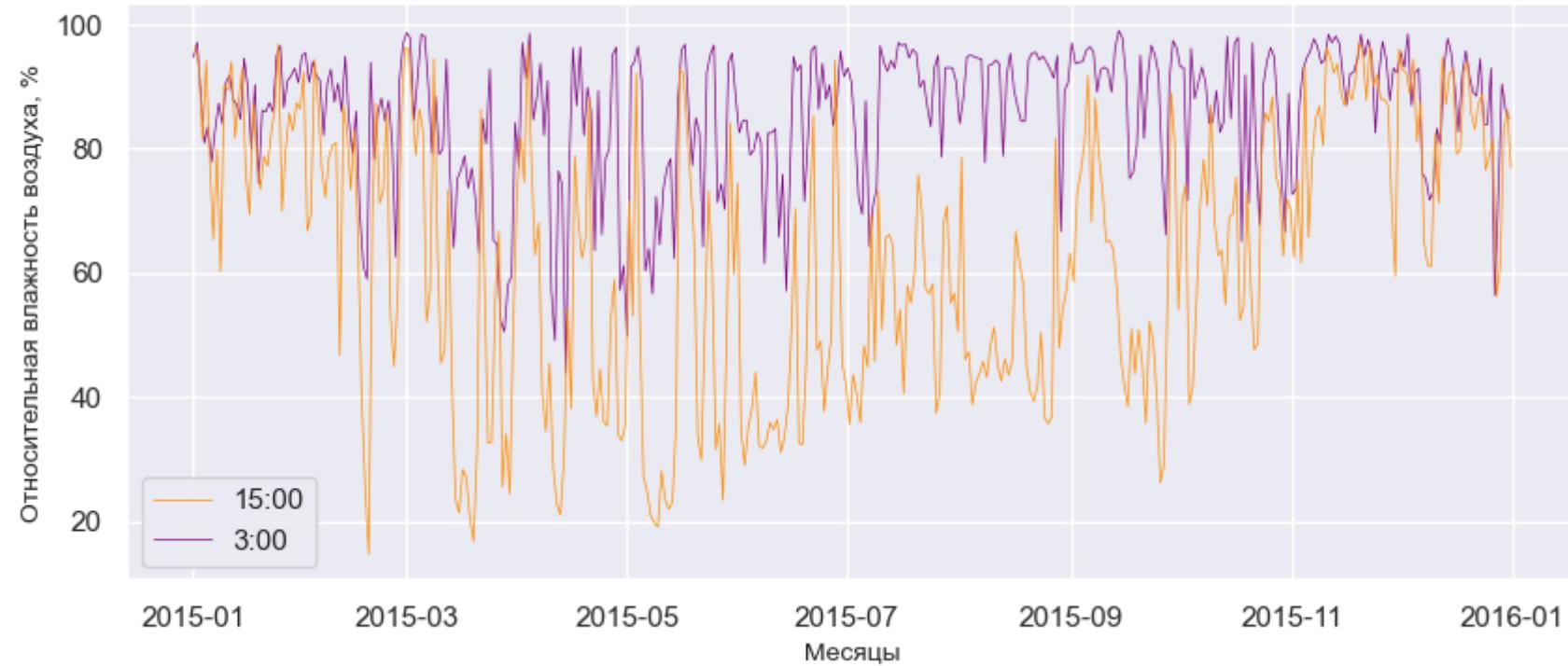
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2017 год



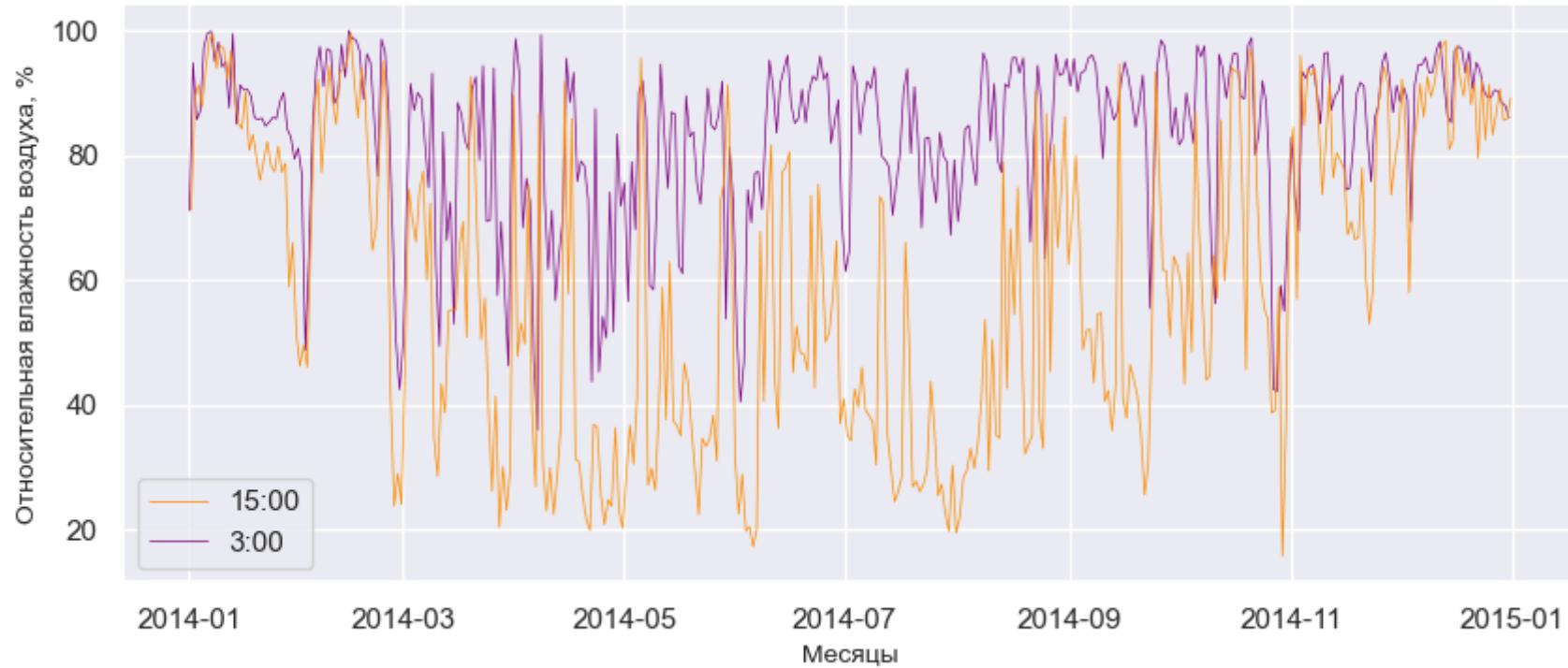
### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2016 год



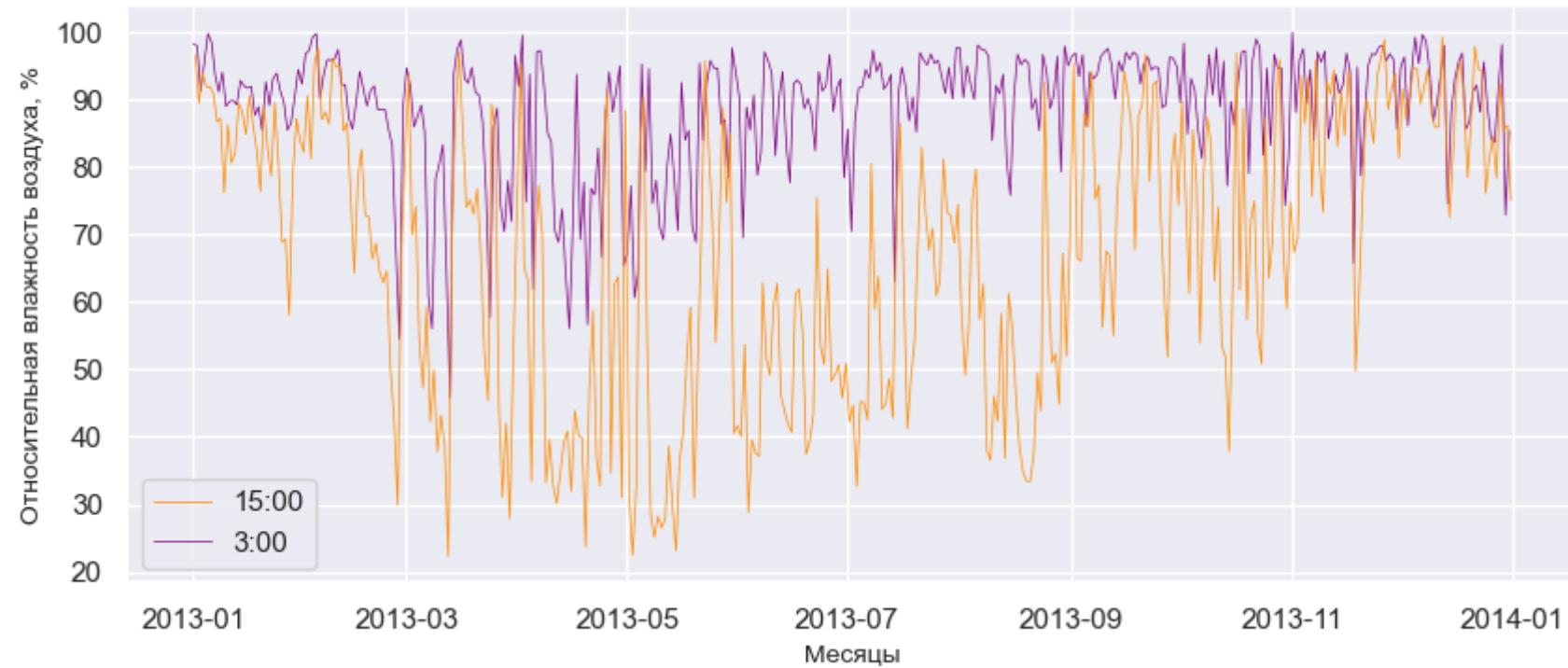
### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2015 год



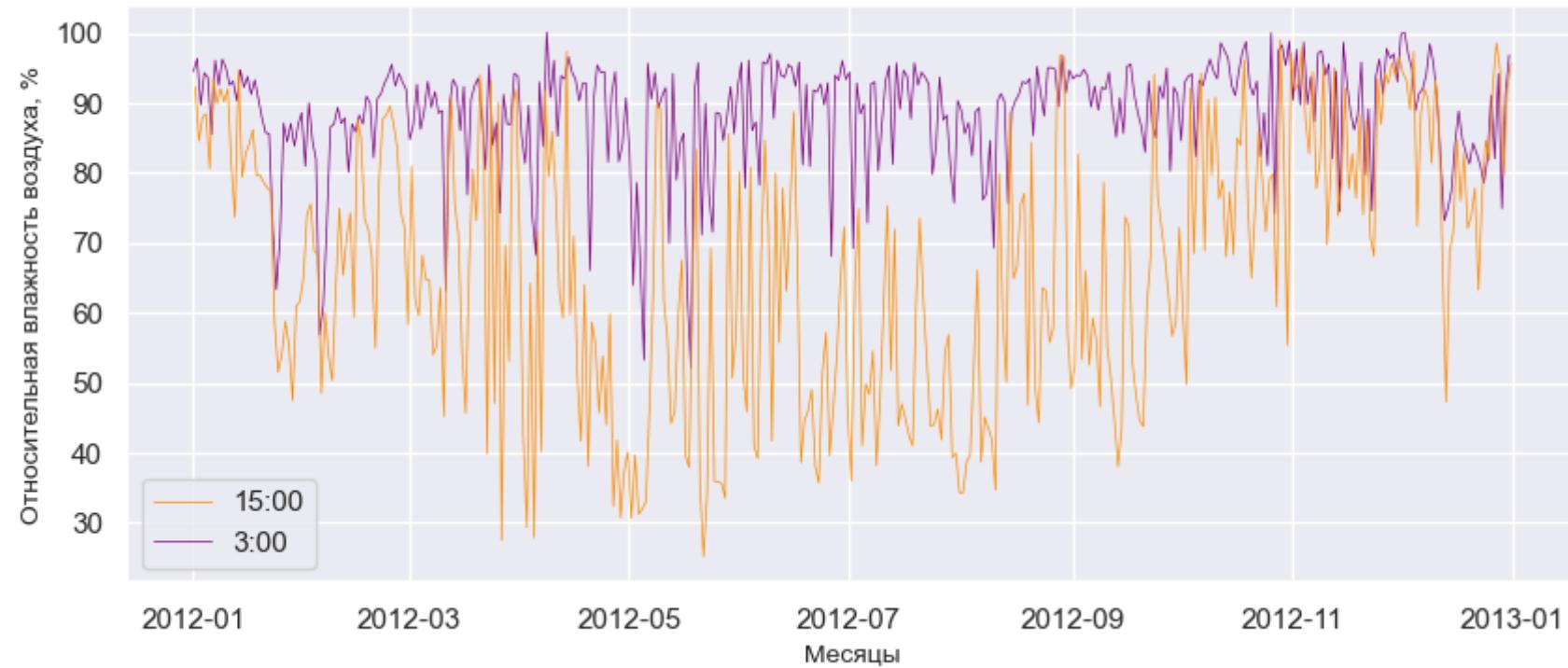
### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2014 год



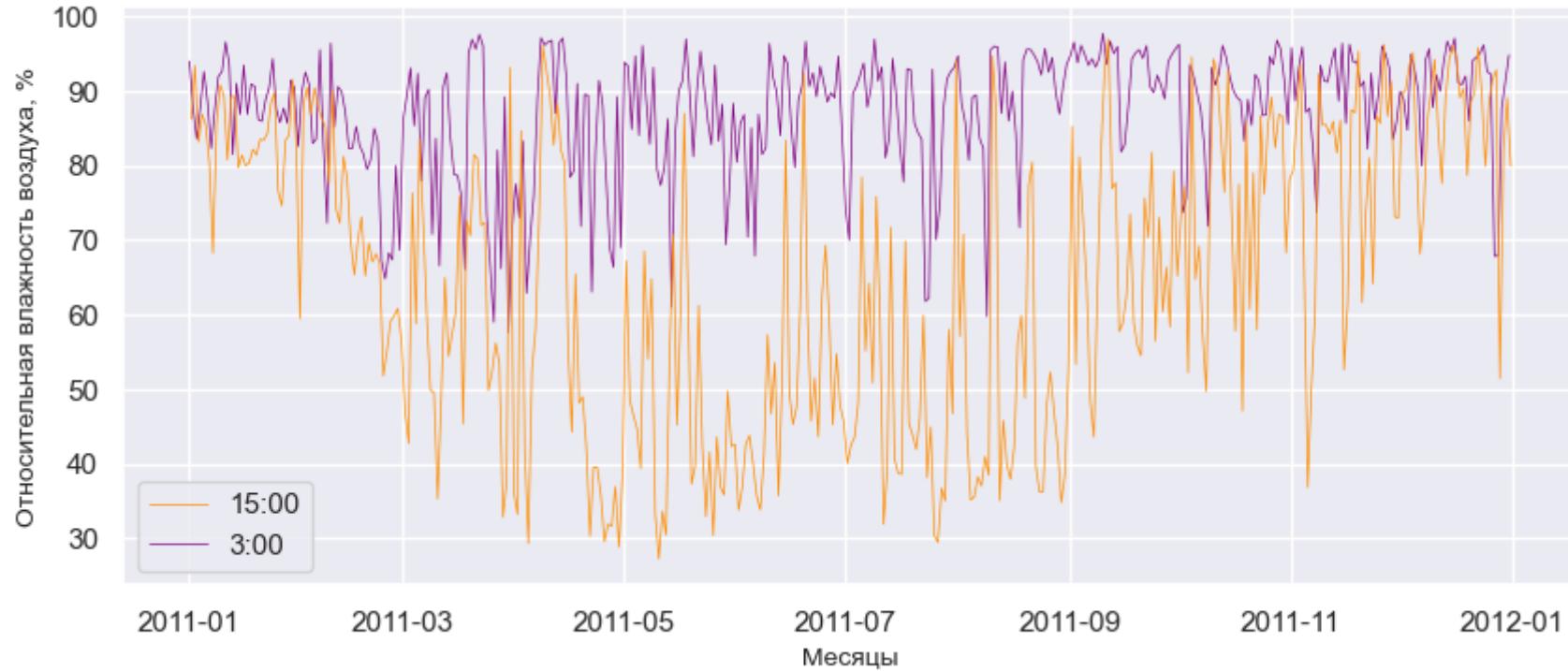
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2013 год



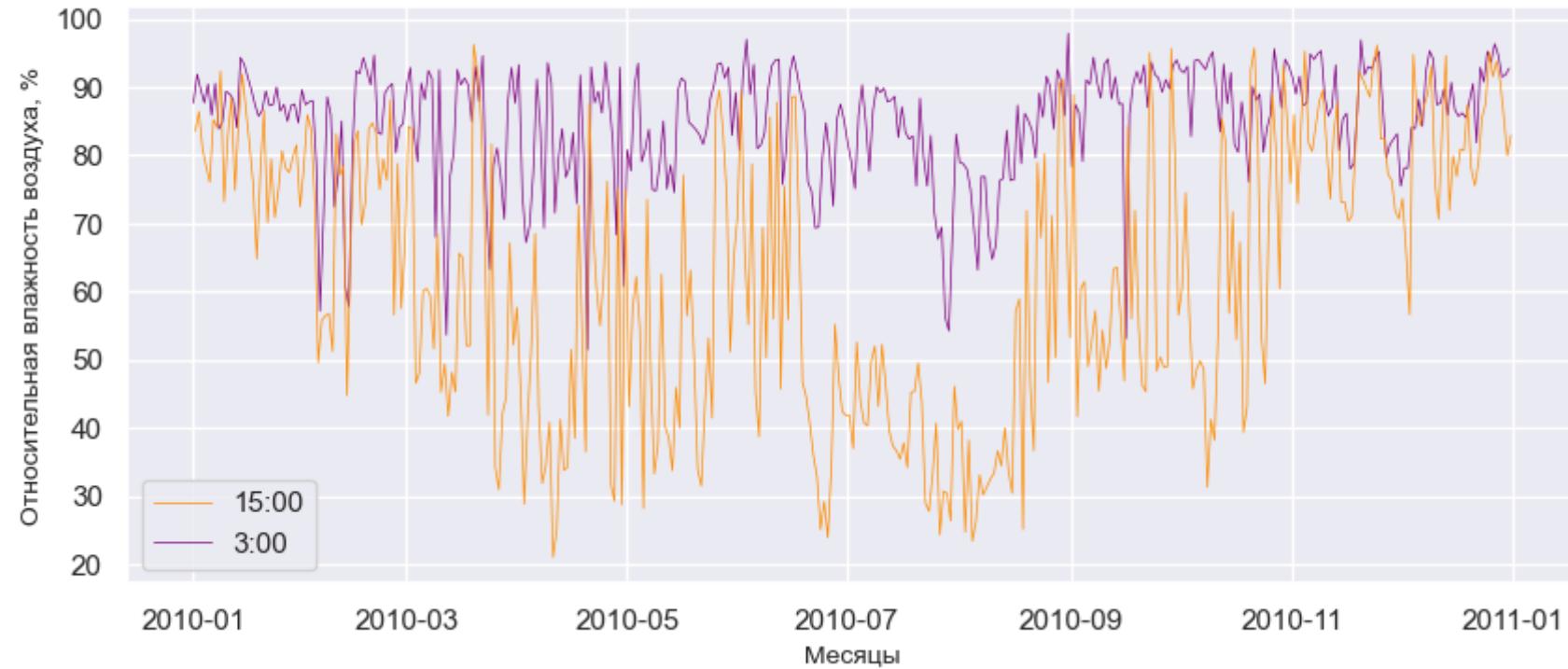
### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2012 год



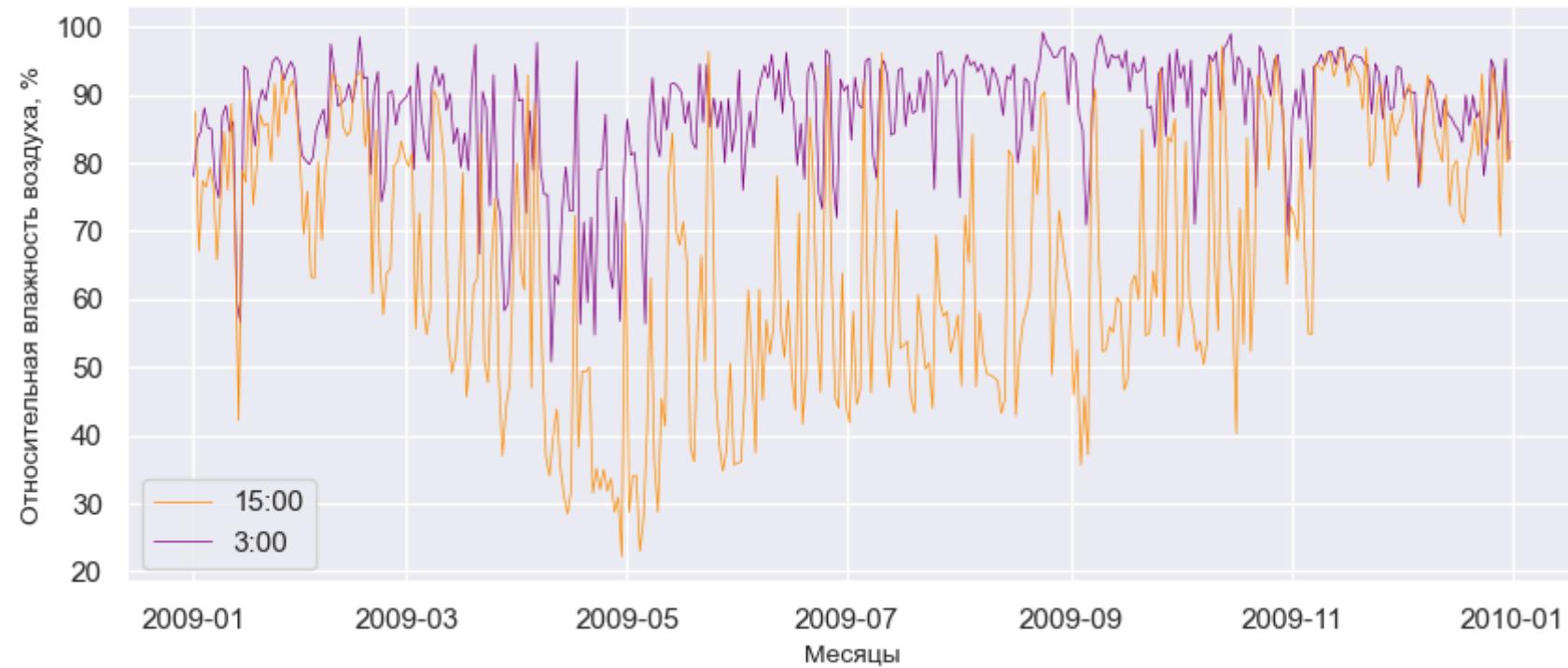
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2011 год



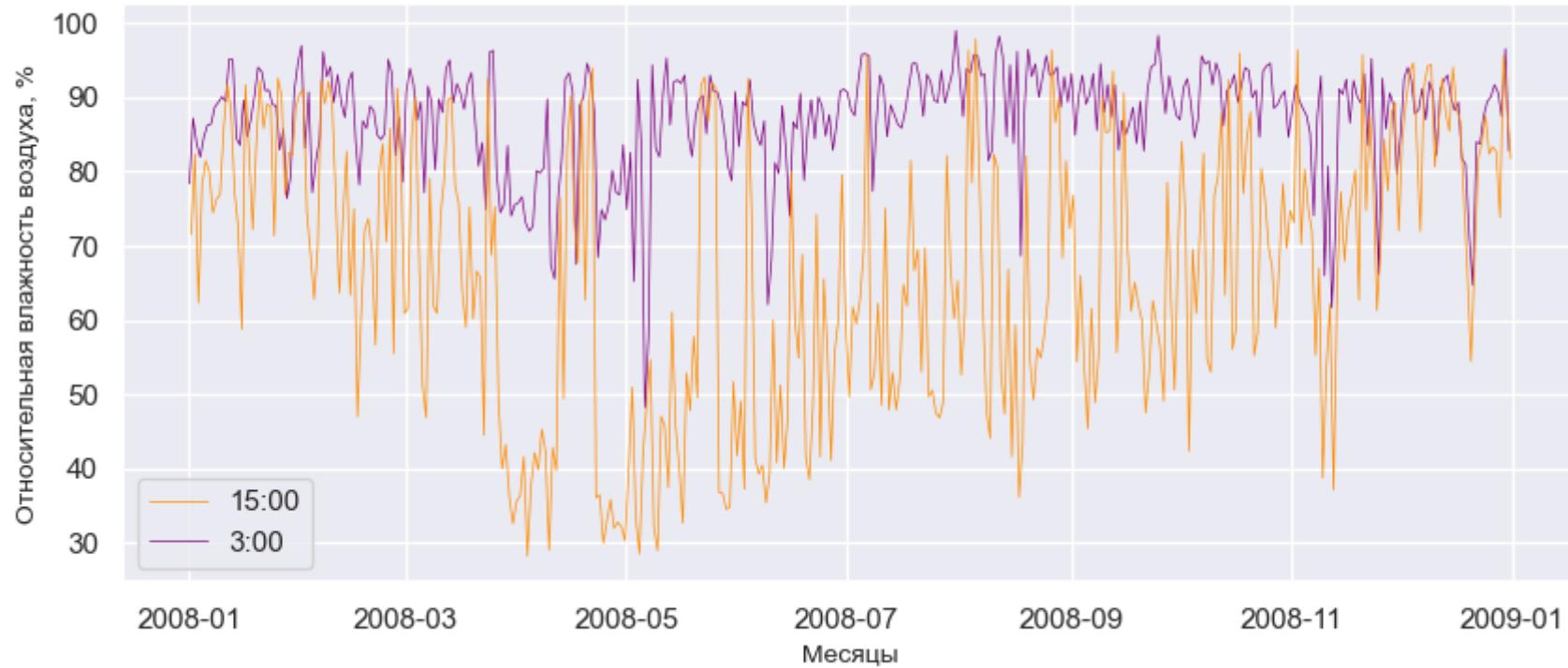
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2010 год



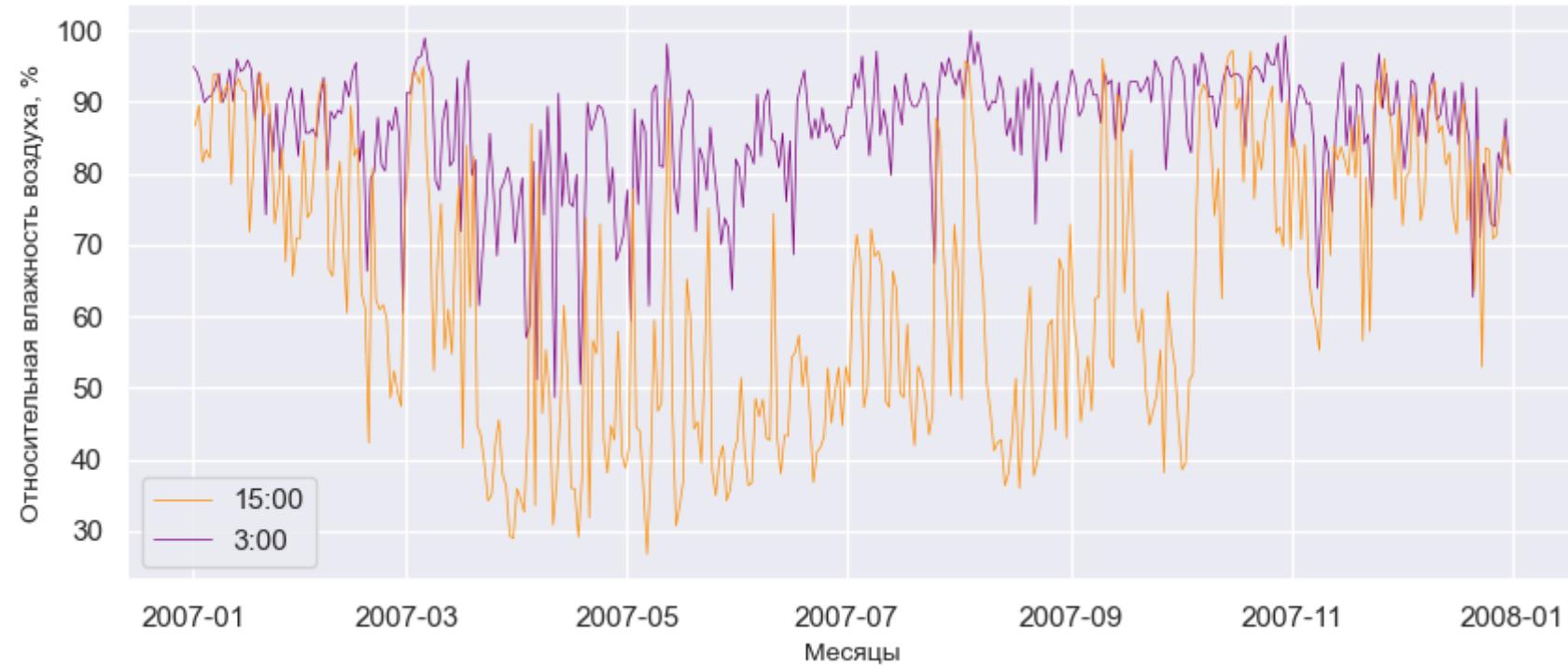
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2009 год



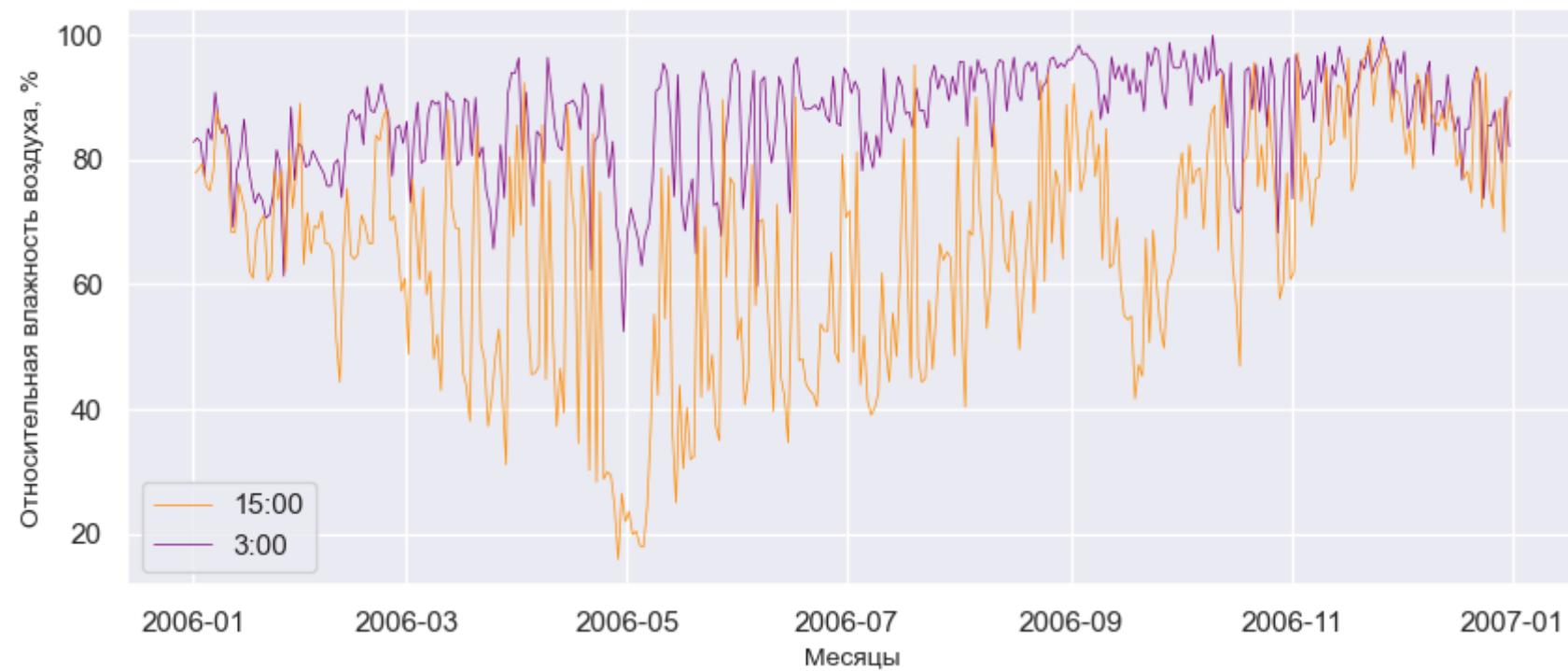
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2008 год



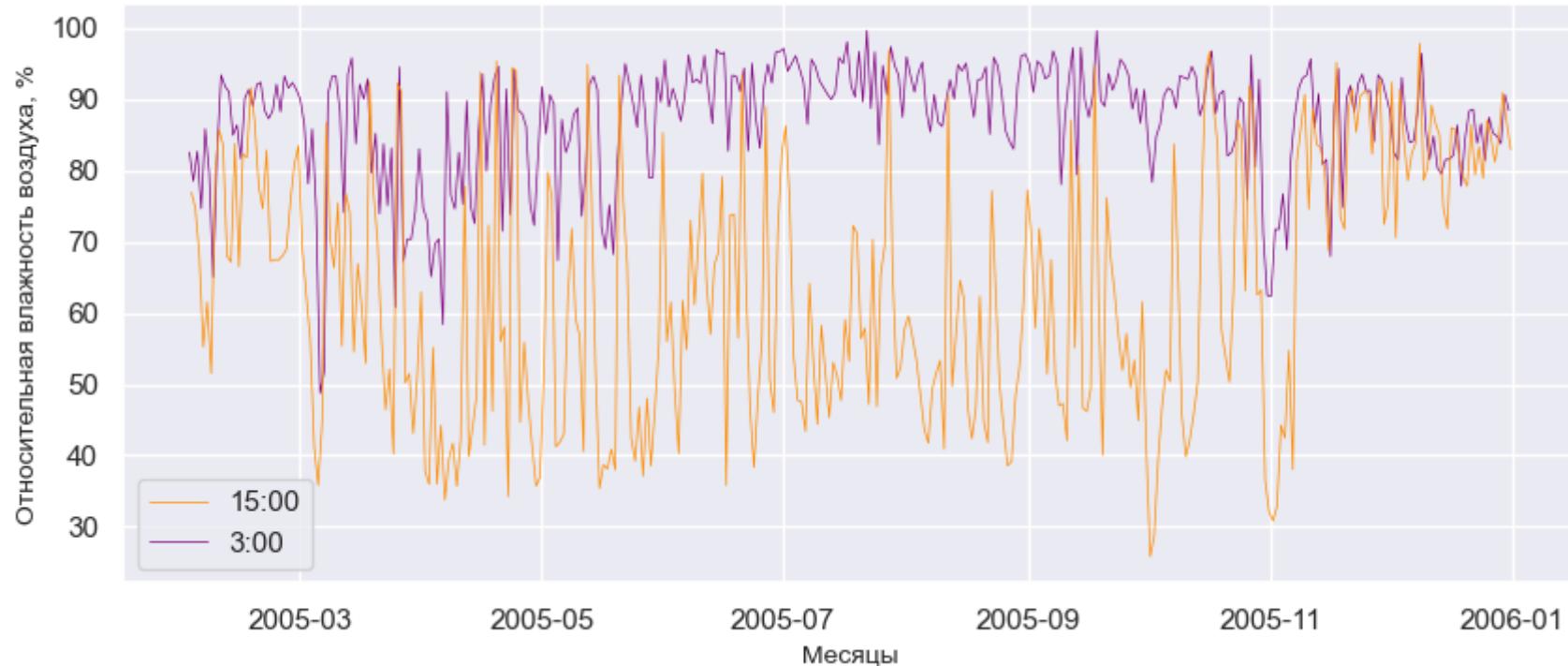
Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2007 год



Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2006 год



### Чашниково: Ежедневная динамика параметра Humid на 3 и 15 часов, 2005 год



#### 5.1.6. Сохранение полученных данных в файлы

In [212...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER51}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER51}'  
  
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_'+PARAMETER51 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER51].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER51}.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_Humid.csv -> DONE!
```

## 5.2. Температура точки росы: Dew\_point

Показатель температуры точки росы является сугубо расчётным. Нет смысла искать в нём выбросы и моделировать его значения - они вычисляются из значений температуры и влажности воздуха, которые уже обработаны.

### 5.2.1. Поиск и удаление ошибок показателя давления Dew\_point

Для данного раздела обозначим константу названия параметра

```
In [213...]: PARAMETER52 = 'Dew_point'
```

**Создаём временный DF для работы с параметром Dew\_point**

```
In [214...]: param_df_name = f'df_{PARAMETER52}' # преобразуем полученное значение в df_PARAMETER52 - ключ словаря dict_df_parameters
# Создадим временный df
df_tmp52 = dict_df_parameters[param_df_name].copy(deep=True)
df_tmp52.sample(5, random_state=56)
```

Out[214]:

	Dew_point#V_Volochek	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
2014-02-22 03:00:00	-7.3	-6.7	-7.2	-8.1	-8.1	-8.3	-6.9
2015-05-16 03:00:00	7.5	7.0	10.2	7.1	7.5	10.3	6.8
2020-06-18 18:00:00	16.3	16.0	17.4	16.2	19.8	19.2	NaN
2019-12-02 21:00:00	-4.0	-4.5	-4.6	-5.0	-4.9	-4.2	-4.6
2008-06-05 18:00:00	NaN	4.8	NaN	5.9	4.6	3.6	5.8

Определим последовательность действий, для поиска ошибок показателя температуры точки росы Dew\_point.

1. Проверяем соответствие значения температуры точки росы архивным значениям температуры и влажности воздуха. Архивы температуры и относительной влажности у нас уже исправлены и не должны содержать ошибок и NaN.
2. Определяем несоответствия как ошибки, и удаляем их.

### Визуализируем архив температуры точки росы (Dew\_point) по сезонам

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

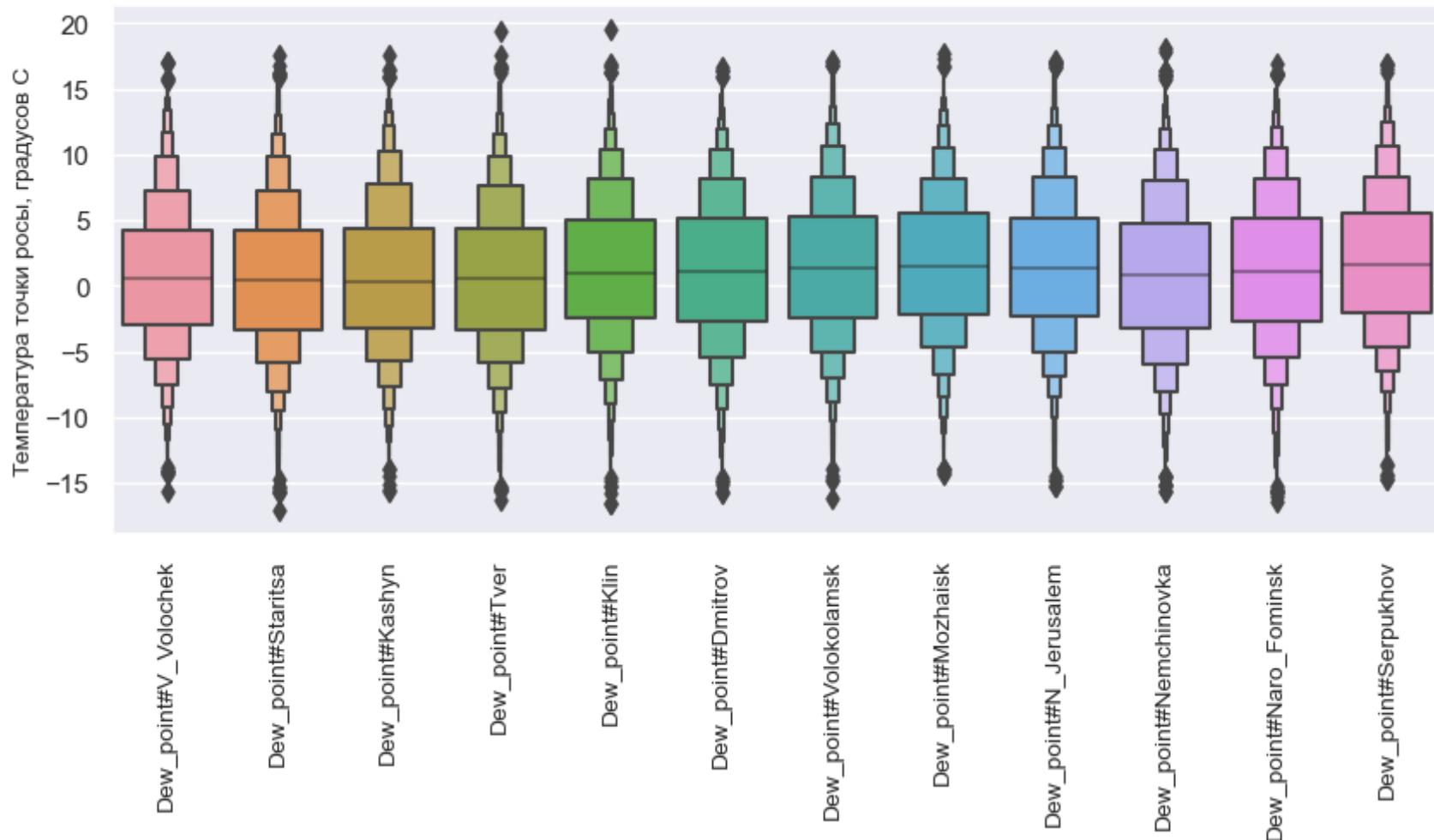
- Зима (ниже 0°): В среднем длится с 5 ноября по 4 апреля
- Весна (от 0° до +10°): В среднем длится с 5 апреля по 18 мая

- 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

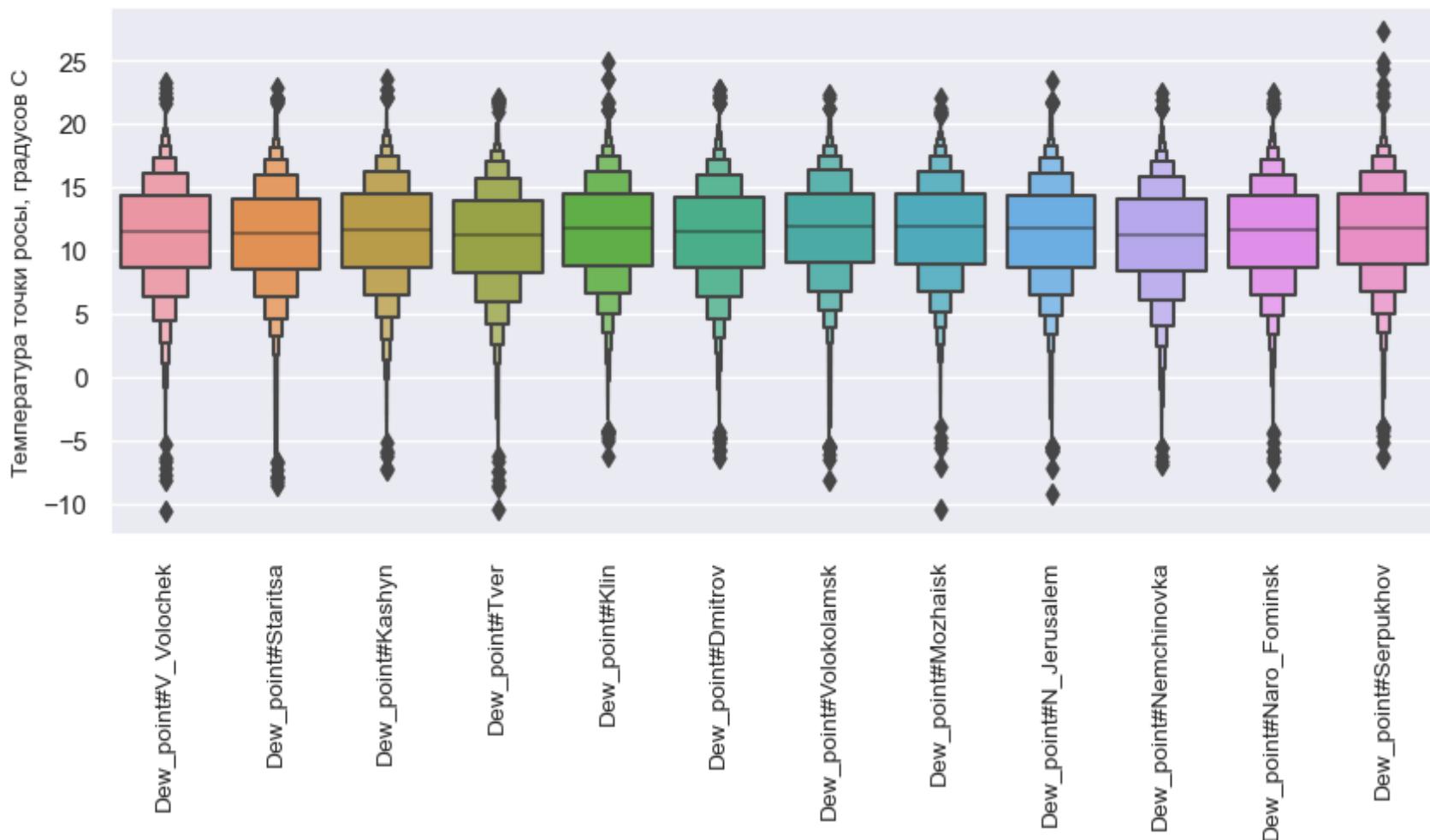
In [215...]

```
# В цикле выведем графики температуры точки росы в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp52).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp52[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Температура точки росы, градусов С', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER52} в разрезе метеостанций:\n'
                        f'{season_name}')
plt.show()
```

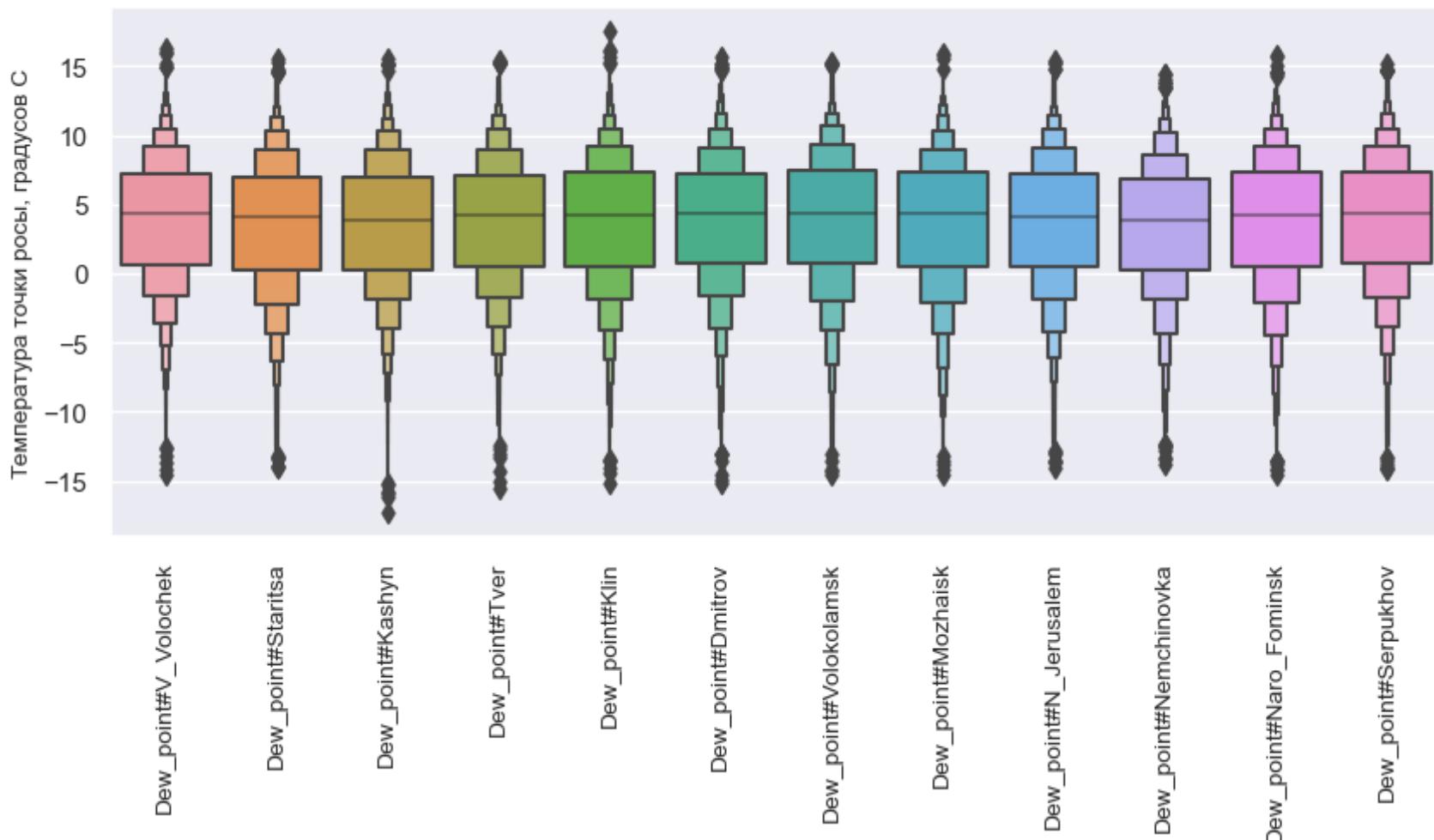
Распределение значений Dew\_point в разрезе метеостанций:  
spring



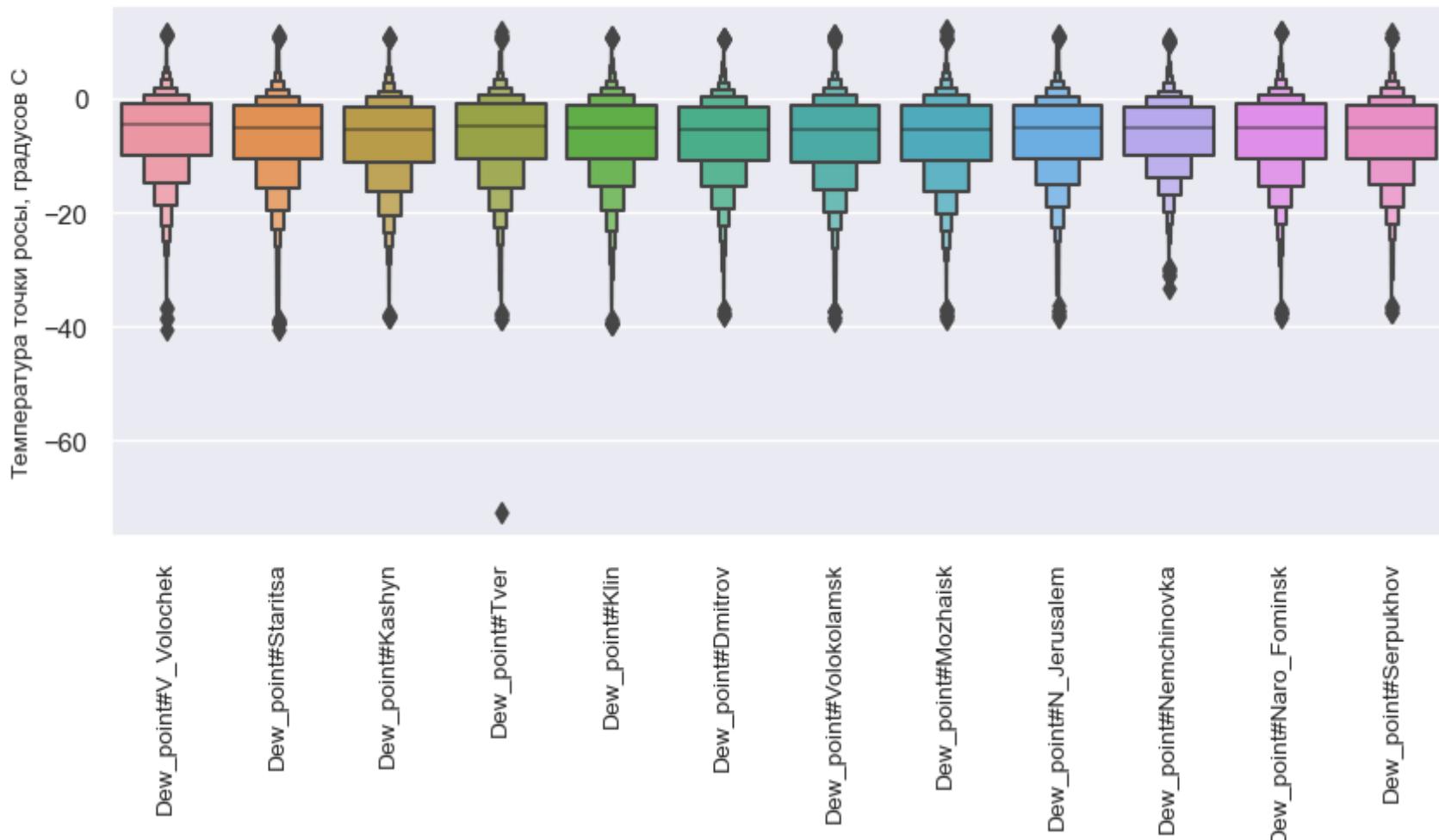
Распределение значений Dew\_point в разрезе метеостанций:  
summer



Распределение значений Dew\_point в разрезе метеостанций:  
autumn



### Распределение значений Dew\_point в разрезе метеостанций: winter



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

In [216]:

```
print(f'Минимальное значение: {np.nanmin(df_tmp52)},\n'
      f'Максимальное значение: {np.nanmax(df_tmp52)},\n'
      f'Средняя: {np.nanmean(df_tmp52)},\n'
      f'Медиана: {np.nanmedian(df_tmp52)}')
```

Минимальное значение: -72.7,  
Максимальное значение: 27.3,  
Средняя: 1.5611522435087837,  
Медиана: 1.5

### Проверим соответствие температуры точки росы значениям температуры воздуха и относительной влажности

Определим расхождения между архивным значением температуры точки росы и расчётым значением, полученным по формуле расчёта температуры точек росы. Сохраним координаты ошибочных значений.

Определим критерий для некорректных значений в 0,5 %

```
In [217...]: criterium = 0.5 # критерий для определения допустимой погрешности
# расчётного значения температуры точки росы от архивного

# В цикле по столбцам df_tmp52
# - определим промежуточный DF для расчётов;
# - заполним его значениями, необходимыми для приведения расчёта температуры точки росы,
# - расчитаем температуру точки росы,
# - рассчитаем абсолютное отклонение расчётных значений от архивных,
# - определим соответствие архивных значений критерию ошибки
# Исходим из идентичности индексов всех датафреймов в архивах

list_error_at = [] # определим список координат ошибочных значений в df_tmp52
for col in df_tmp52.columns:
    df_dew_p = pd.DataFrame(None, index=df_tmp52.index) # создадим пустой DF
    station_name = col[len(PARAMETER52)+1:] # выделяем название метеостанции из названия столбца
    height = df_station_dists[df_station_dists.station == station_name].height.values[0] # находим высоту метеостанции

    df_dew_p = (
        df_dew_p
        .assign(
            station=station_name,
            humid=df_tmp51.loc[:,PARAMETER51+'#'+station_name], # архивное значение относительной влажности
            T=dict_df_parameters['df_T'][f"{'T'+col[len(PARAMETER52):]}"], # Температура воздуха (архив)
            dew_p=dict_df_parameters['df_Dew_point'][f"{'Dew_point'+col[len(PARAMETER52):]}"] # Точка росы (архив)
        )
    )

    df_dew_p = (
        df_dew_p
        .assign(dew_p_c=(df_dew_p.apply(lambda x: dew_point_calculator(temp_x['T'], humid=x.humid), axis=1)
                    ), # Расчётная величина точки росы
    )

    if np.abs(df_dew_p[col].values - df_tmp52[col].values) > criterium:
        list_error_at.append((df_dew_p.index, col))
```

```

        dp_delta=lambda x: abs(x.dew_p_c - x.dew_p), # абсолютное отклонение расчётной величины от архивной
        dp_error=lambda x: x.dp_delta >= criterium # критерий ошибки для отклонения расчётной величины, t с
    )
)

# Расширяем список координат ошибочных значений в df_tmp52:
# Если абсолютное отклонение расчётного значения от архивного больше или равно установленному критерия:
# прибавляем список кортежей моментов наблюдения и метеостанций по которым значения превышают допустимую погрешность
list_error_at = list_error_at + (df_dew_p
    .apply(
        lambda x: (x.name, station_name) if x.dp_delta >= criterium else np.nan,
        axis=1)
    .dropna()
    .tolist()
)

print(f'Количество аномальных отклонений расчётной температуры точки росы от архивного значения '
      f'для метеостанции {station_name} = {df_dew_p.dp_error.sum()}'')
#List_error_at

```

Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции V\_Volochev = 405  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Staritsa = 33  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Kashyn = 217  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Tver = 42  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Klin = 16  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Dmitrov = 20  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Volokolamsk = 6  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Mozhaisk = 12  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции N\_Jerusalem = 15  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Nemchinovka = 42  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Naro\_Fominsk = 12  
 Количество аномальных отклонений расчётной температуры точки росы от архивного значения для метеостанции Serpukhov = 364

Заменим ошибочные значения на NaN

```
In [218...]: for error in list_error_at:
    df_tmp52.at[error[0], PARAMETER52 + '#' + error[1]] = np.nan
```

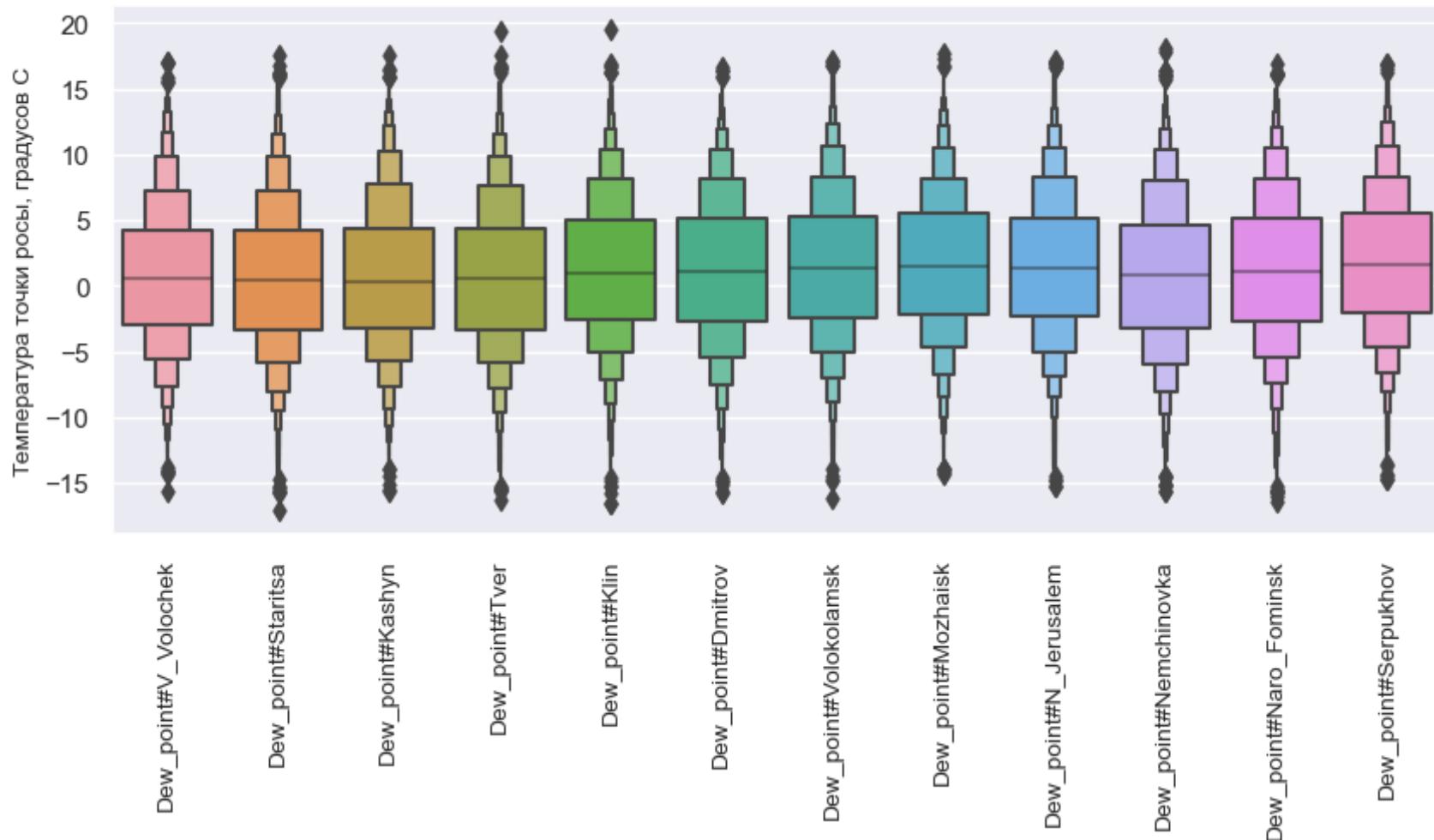
Так как температура точки росы является сугубо расчётным показателем из значений температуры воздуха и относительной влажности, у нас нет необходимости искать выбросы в значениях показателя P\_station. После проверки соответствия архивных и расчётных значений

все некорректные значения для Dew\_point уже удалены. В свою очередь, T и Humid уже очищены от аномалий и ошибок ранее, и ошибочные значения в них уже исправлены.

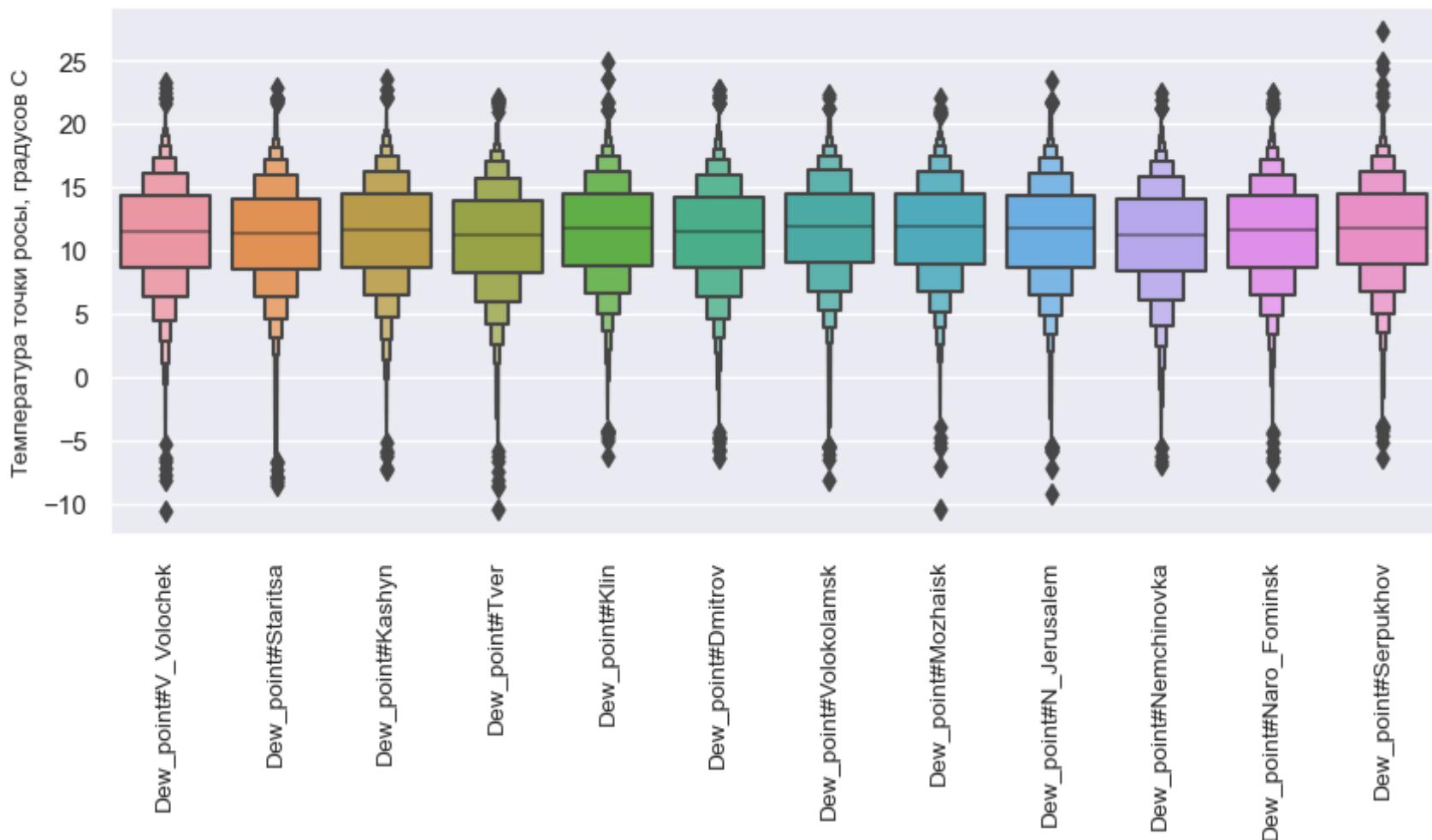
### Повторно, после удаления всех ошибочных значений, визуализируем архив давления на уровне станции (Dew\_point) по сезонам

```
In [219...]: # В цикле выведем графики температуры точки росы по метеостанциям в зависимости от сезона
# используем функцию создания масок климатических сезонов
for season_name, season_mask in season_masks(df_tmp52).items():
    fig, ax = plt.subplots(figsize=(10, 4))
    g = sns.boxenplot(data=df_tmp52[season_mask],
                       ax=ax)
    dummy = plt.xticks(rotation=90, size=10)
    dummy = g.set_ylabel('Температура точки росы, градусов С', size=10)
    dummy = g.set_title(f'Распределение значений {PARAMETER52} в разрезе метеостанций после удаления ошибок:\n'{season_name}')
plt.show()
```

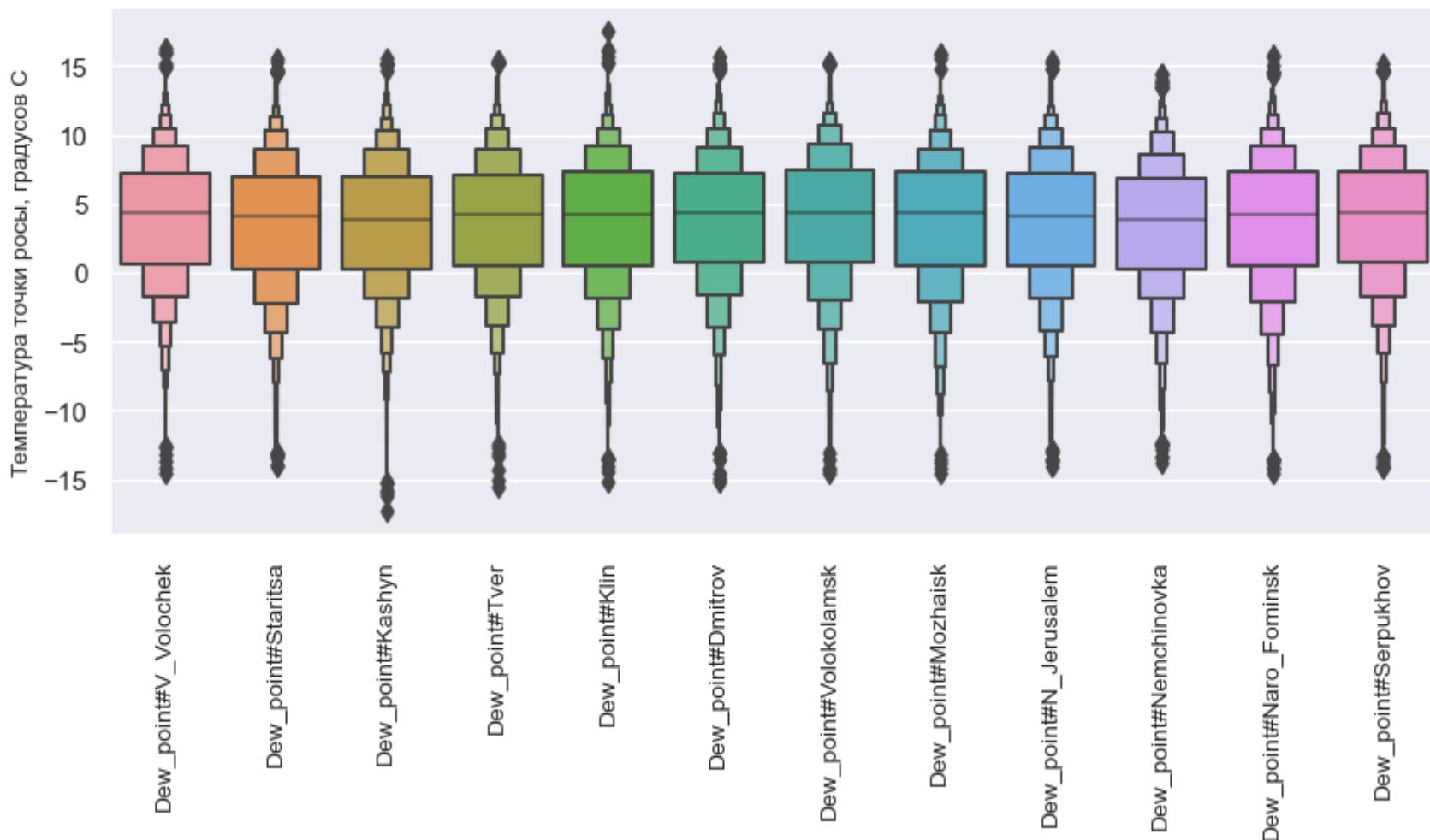
Распределение значений Dew\_point в разрезе метеостанций после удаления ошибок:  
spring



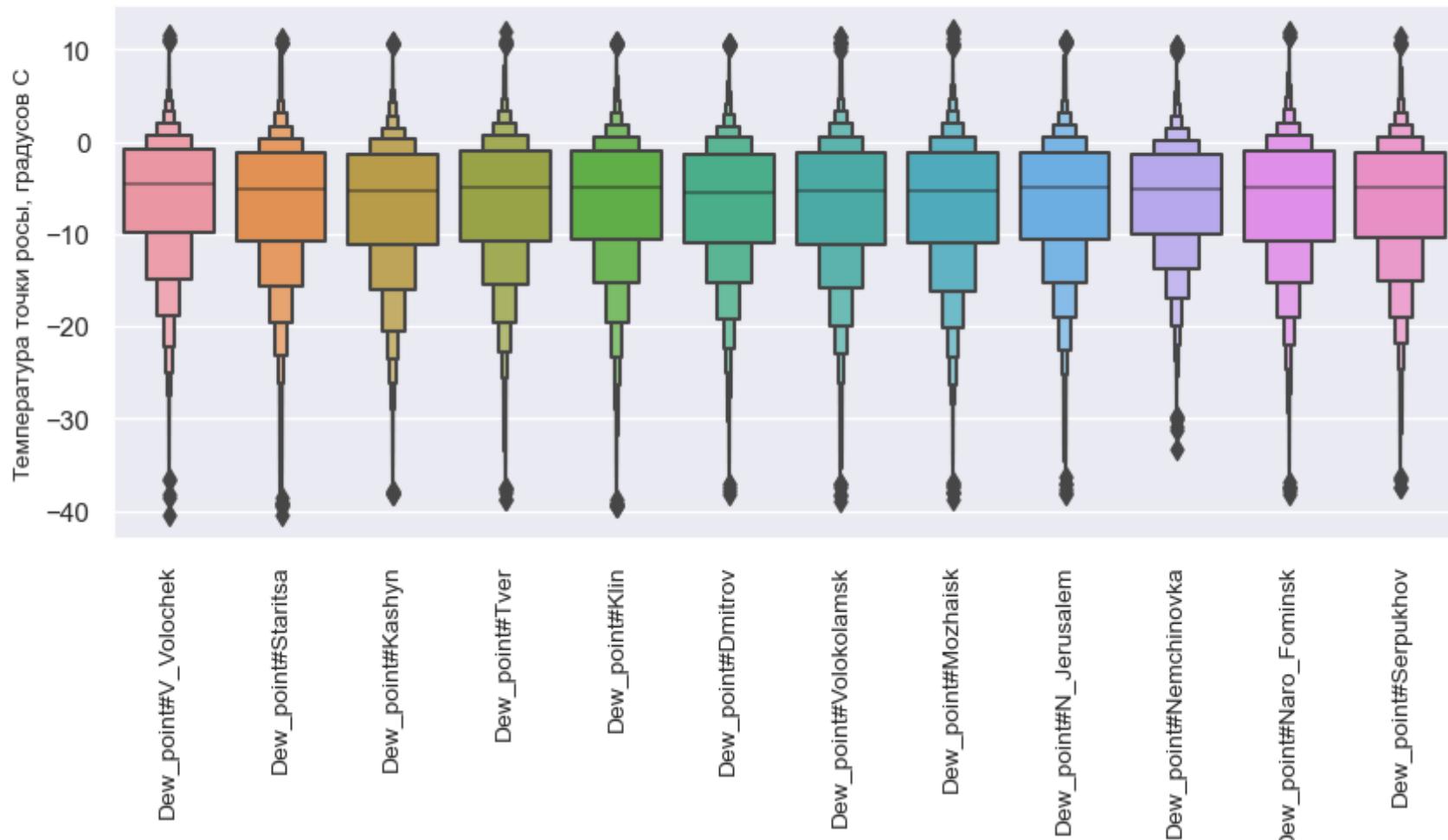
Распределение значений Dew\_point в разрезе метеостанций после удаления ошибок:  
summer



Распределение значений Dew\_point в разрезе метеостанций после удаления ошибок:  
autumn



### Распределение значений Dew\_point в разрезе метеостанций после удаления ошибок: winter



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

```
In [220]:  
print(f'Минимальное значение: {np.nanmin(df_tmp51)},\n'  
      f'Максимальное значение: {np.nanmax(df_tmp51)},\n'  
      f'Средняя: {np.nanmean(df_tmp51)},\n'  
      f'Медиана: {np.nanmedian(df_tmp51)}')
```

Минимальное значение: 8.0,  
Максимальное значение: 101.37720825752231,  
Средняя: 77.44346673243616,  
Медиана: 83.38725163487183

### Сохраним очищенные данные в файл параметров

In [221...]

```
# # 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_'+PARAMETER52 + '.csv ->', end=' ')  
dict_df_parameters['df_'+PARAMETER52].to_csv(  
    path_or_buf=f'{clean_path}df_{PARAMETER52}.csv'  
)  
print('DONE!')  
  
df_Dew_point.csv -> DONE!
```

### 5.2.2. Замена удалённых и пропущенных значений показателя Dew\_point расчётными, расчёт показателя относительной влажности воздуха Dew\_point для Агробиостанции МГУ в пос. Чашниково и для центральной точки поля метеостанций; фиксация исправлений в архивах

In [222...]

```
# Добавим в df_tmp52 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 PARAMETER52#Chashnikovo и PARAMETER52#Rfrnce_point  
df_tmp52 = (df_tmp52.  
    assign(col_name1 = np.nan,  
          col_name2 = np.nan).  
    rename(columns={"col_name1": PARAMETER52+'#'+ 'Chashnikovo',  
              "col_name2": PARAMETER52+'#'+ 'Rfrnce_point'}))  
)
```

```
df_tmp52.sample(3, random_state=56)
```

Out[222]:

	Dew_point#V_Volochek	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
2014-02-22 03:00:00	-7.3	-6.7	-7.2	-8.1	-8.1	-8.3	-6.9
2015-05-16 03:00:00	7.5	7.0	10.2	7.1	7.5	10.3	6.8
2020-06-18 18:00:00	16.3	16.0	17.4	16.2	19.8	19.2	NaN

In [223...]

```
# Добавим в архив параметров P_sea столбец для Чашниково и будущей центральной точки, заполним их NaN
# Создадим столбцы col_name со значениями np.nan
# Переименуем столбец PARAMETER52#Chashnikovo и PARAMETER52#Rfrnce_point
dict_df_parameters['df_'+PARAMETER52] = (dict_df_parameters['df_'+PARAMETER52].
                                             assign(col_name1 = np.nan,
                                                   col_name2 = np.nan).
                                             rename(columns={"col_name1": PARAMETER52+'#'+ 'Chashnikovo',
                                                               "col_name2": PARAMETER52+'#'+ 'Rfrnce_point'})
                                             )
dict_df_parameters['df_'+PARAMETER52].sample(3, random_state=56)
```

Out[223]:

	Dew_point#V_Volochev	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
<b>2014-02-22 03:00:00</b>	-7.3	-6.7	-7.2	-8.1	-8.1	-8.3	-6.9
<b>2015-05-16 03:00:00</b>	7.5	7.0	10.2	7.1	7.5	10.3	6.8
<b>2020-06-18 18:00:00</b>	16.3	16.0	17.4	16.2	19.8	19.2	NaN

In [224...]

```
# Добавим в архив метеостанций df Чашниково и df для центральной точки,
# Создадим в нём столбец с называнием PARAMETER52
```

```
dict_df_locations['df_Chashnikovo'] = (dict_df_locations['df_Chashnikovo']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER52})
                                         )
dict_df_locations['df_Rfrnce_point'] = (dict_df_locations['df_Rfrnce_point']
                                         .assign(col_name = np.nan)
                                         .rename(columns={"col_name": PARAMETER52})
                                         )

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

Out[224]:

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

Out[224]:

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

## Заполнение NaN расчётными значениями

Подсчитаем количество NaN

In [225...]

```
# np.isnan(df_tmp52).sum() выдаёт количество NaN по каждому столбцу.  
# Поэтому дополнительно просуммируем полученные по столбцам значения  
np.sum(np.isnan(df_tmp52).sum())
```

Out[225]: 176587

Составим массив координат значений NaN в df\_tmp52

In [226...]

```
arr_idx_nan = np.where(np.isnan(df_tmp52)) # Кортеж из двух одномерных массивов координат  
arr_idx_nan = np.stack((arr_idx_nan[0], arr_idx_nan[1]), axis = 1) # массив из парных координат
```

По координатам значений NaN, заменим в df\_tmp52 NaNы на расчётные значения

In [227...]

```
start_time = time.time() # для замера времени выполнения кода  
  
for idxs in arr_idx_nan: # по индексам в массиве индексов NaNов  
  
    station_name = df_tmp52.columns[idxs[1]][len(PARAMETER52)+1:] # выделяем название метеостанции из названия столбца  
    temperature = dict_df_parameters['df_T'].iat[idxs[0], idxs[1]] # находим температуру воздуха  
    humid = df_tmp51.iat[idxs[0], idxs[1]] # находим отн.влажность  
    dew_point = dew_point_calculator(temp=temperature, humid=humid) # вычисляем по барометрической формуле давление на уровне  
  
    df_tmp52.iat[idxs[0], idxs[1]] = dew_point # записываем значение в df_tmp52  
  
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')
```

```
print(f'Осталось NaN: {np.sum(np.isnan(df_tmp52).sum())}')
```

Elapsed time=00:00:17

Осталось NaN: 14

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

### Перезапись архивов в части показателя Dew\_point

In [228...]

```
# Перенесём уже исправленные значения в dict_df_parameters, оставшиеся NaN исправим ниже
dict_df_parameters['df_'+PARAMETER52] = df_tmp52.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[:, PARAMETER52] = df_tmp52.loc[:, PARAMETER52 + '#' + name_df[3:]]

# Выведем случайные ряды из архивов
dict_df_parameters['df_'+PARAMETER52].sample(7, random_state=56)
dict_df_locations['df_Chashnikovo'].sample(7, random_state=56)
dict_df_locations['df_Rfrnce_point'].sample(7, random_state=56)
```

Out[228]:

	Dew_point#V_Volochek	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
<b>2014-02-22 03:00:00</b>	-7.300000	-6.7	-7.20000	-8.1	-8.1	-8.3	-6.900000
<b>2015-05-16 03:00:00</b>	7.500000	7.0	10.20000	7.1	7.5	10.3	6.800000
<b>2020-06-18 18:00:00</b>	16.300000	16.0	17.40000	16.2	19.8	19.2	12.934336
<b>2019-12-02 21:00:00</b>	-4.000000	-4.5	-4.60000	-5.0	-4.9	-4.2	-4.600000
<b>2008-06-05 18:00:00</b>	4.983832	4.8	4.23301	5.9	4.6	3.6	5.800000
<b>2016-10-20 06:00:00</b>	-2.800000	-2.9	-2.40000	-2.8	-2.0	-2.3	-3.100000
<b>2006-09-11 03:00:00</b>	7.600000	7.0	8.20000	7.3	6.7	8.4	7.000000

Out[228]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point
<b>2014-02-22 03:00:00</b>	-4.156558	-4.156558	-2.109643	768.336709	747.597791	0.812775	76.635021	-7.639754
<b>2015-05-16 03:00:00</b>	9.181730	9.181730	10.434571	745.095535	725.921747	-1.043037	95.837940	8.552121
<b>2020-06-18 18:00:00</b>	27.439052	25.366130	30.241632	761.480701	743.060948	-0.492627	58.118042	18.459938
<b>2019-12-02 21:00:00</b>	-3.004683	-4.229056	-2.673631	758.773486	738.378842	-0.577660	86.998573	-4.858836
<b>2008-06-05 18:00:00</b>	16.565949	12.255838	16.565949	760.858054	741.771366	-0.440787	40.000000	2.909744
<b>2016-10-20 06:00:00</b>	0.117681	0.074269	0.916309	775.972222	755.350419	-0.276710	86.022825	-1.938759
<b>2006-09-11 03:00:00</b>	9.855422	9.855422	11.886722	761.694029	742.139162	1.174445	87.412725	7.863153

Out[228]:

	T	T_min	T_max	P_sea	P_station	P_drift	Humid	Dew_point
<b>2014-02-22 03:00:00</b>	-3.589183	-3.794602	-2.700734	767.373725	754.041734	0.616282	75.006151	-7.367199
<b>2015-05-16 03:00:00</b>	7.789650	7.789650	8.797603	746.708428	734.256500	-0.826706	94.058645	6.893682
<b>2020-06-18 18:00:00</b>	29.404954	25.572651	29.941094	760.796222	749.008692	-0.723724	41.950321	15.114705
<b>2019-12-02 21:00:00</b>	-2.977448	-3.796107	-2.513436	758.016264	744.876404	-0.402065	90.000000	-4.383095
<b>2008-06-05 18:00:00</b>	17.613615	11.754644	17.613615	761.523291	749.249895	-0.789045	47.000000	6.161840
<b>2016-10-20 06:00:00</b>	-0.382597	-0.369287	0.885072	775.916024	762.592727	-0.296078	83.523874	-2.827280
<b>2006-09-11 03:00:00</b>	9.181940	9.181940	11.628949	762.828050	750.169523	0.987633	85.899940	6.945645

Выведем, рандомные строки из df\_tmp42

In [229...]

```
df_tmp52.sample(7)
```

Out[229]:

	Dew_point#V_Volochek	Dew_point#Staritsa	Dew_point#Kashyn	Dew_point#Tver	Dew_point#Klin	Dew_point#Dmitrov	Dew_point#Volokolamsk
<b>2017-11-21 06:00:00</b>	-0.800000	-1.5	-1.60000	-1.7	-1.8	-1.200000	-2.650713
<b>2006-09-09 21:00:00</b>	9.200000	9.4	9.20000	9.8	8.6	9.500000	8.800000
<b>2013-11-03 12:00:00</b>	5.700000	5.7	4.80000	5.1	5.0	5.438911	5.727715
<b>2022-03-06 06:00:00</b>	-10.900000	-10.8	-15.50000	-11.2	-13.3	-10.400000	-15.200000
<b>2009-07-04 00:00:00</b>	5.407713	5.9	8.37268	4.1	7.7	9.500000	8.900000
<b>2018-07-09 18:00:00</b>	13.800000	13.4	15.00000	13.4	14.0	15.400000	15.937976
<b>2016-12-29 06:00:00</b>	-5.000000	-5.4	-5.00000	-4.6	-2.5	-1.900000	-2.890977

In [230...]

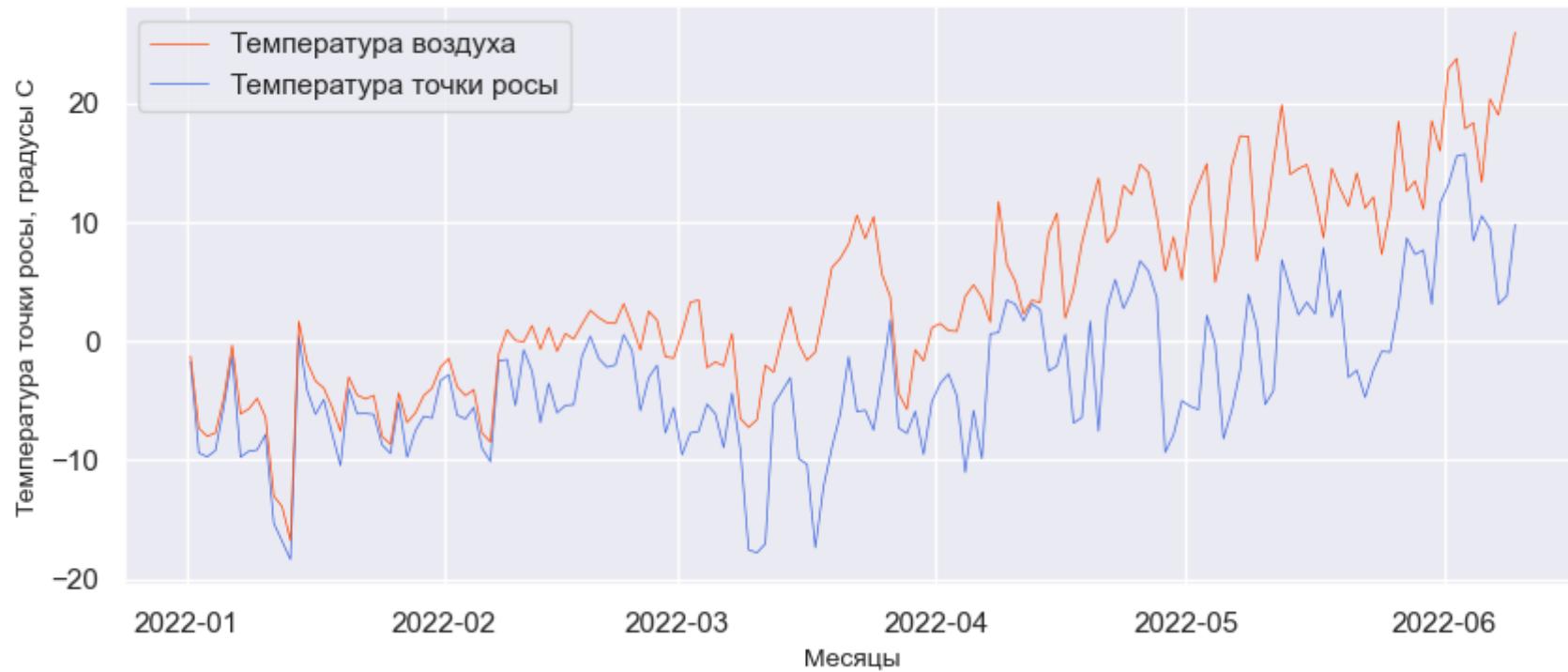
```
# Определим часы момента наблюдения для построения графиков
plot_hour1 = 12

# Строим график в цикле для каждого года
for year in list_years:
    data1 = dict_df_parameters['df_'+PARAMETER52][PARAMETER52+"#"+"Chashnikovo"]\
        [(dict_df_parameters['df_'+PARAMETER52].index.year == year) &
         (dict_df_parameters['df_'+PARAMETER52].index.hour == plot_hour1)]
```

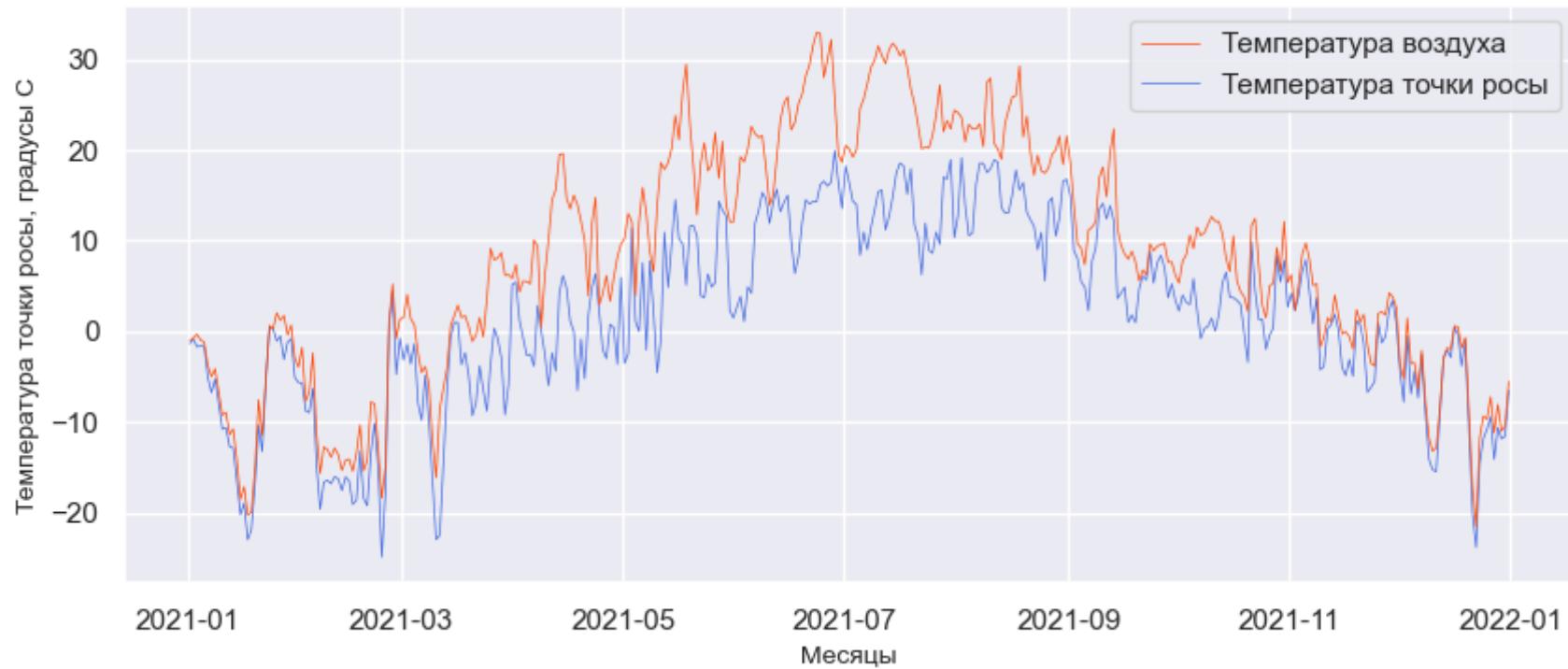
## 5.2.3. Визуализация графиков относительной влажности воздуха для условной метеостанции Чашниково

```
data2 = dict_df_parameters['df_T'][T#"+"Chashnikovo"]\n[(dict_df_parameters['df_T'].index.year == year) &\n (dict_df_parameters['df_T'].index.hour == plot_hour1)]\n\nfig, ax = plt.subplots(figsize=(10, 4))\ng1 = sns.lineplot(data=data1,\n                    color='royalblue',\n                    linewidth=0.5,\n                    ax=ax)\ng2 = sns.lineplot(data=data2,\n                    color='orangered',\n                    linewidth=0.5,\n                    ax=ax)\n\n# Добавляем легенду\nblue_line = mlines.Line2D([], [], color='royalblue', label=f'Температура точки росы', linewidth=0.5)\nred_line = mlines.Line2D([], [], color='orangered', label=f'Температура воздуха', linewidth=0.5)\ndummy = ax.legend(handles=[red_line, blue_line])\n\ndummy = ax.set_ylabel('Температура точки росы, градусы С', size=10)\ndummy = ax.set_xlabel('Месяцы', size=10)\ndummy = plt.title(f'Чашниково: Ежедневная динамика параметра {PARAMETER52} и температуры воздуха '\nf'на {plot_hour1} часов, {year} год')\nplt.show()
```

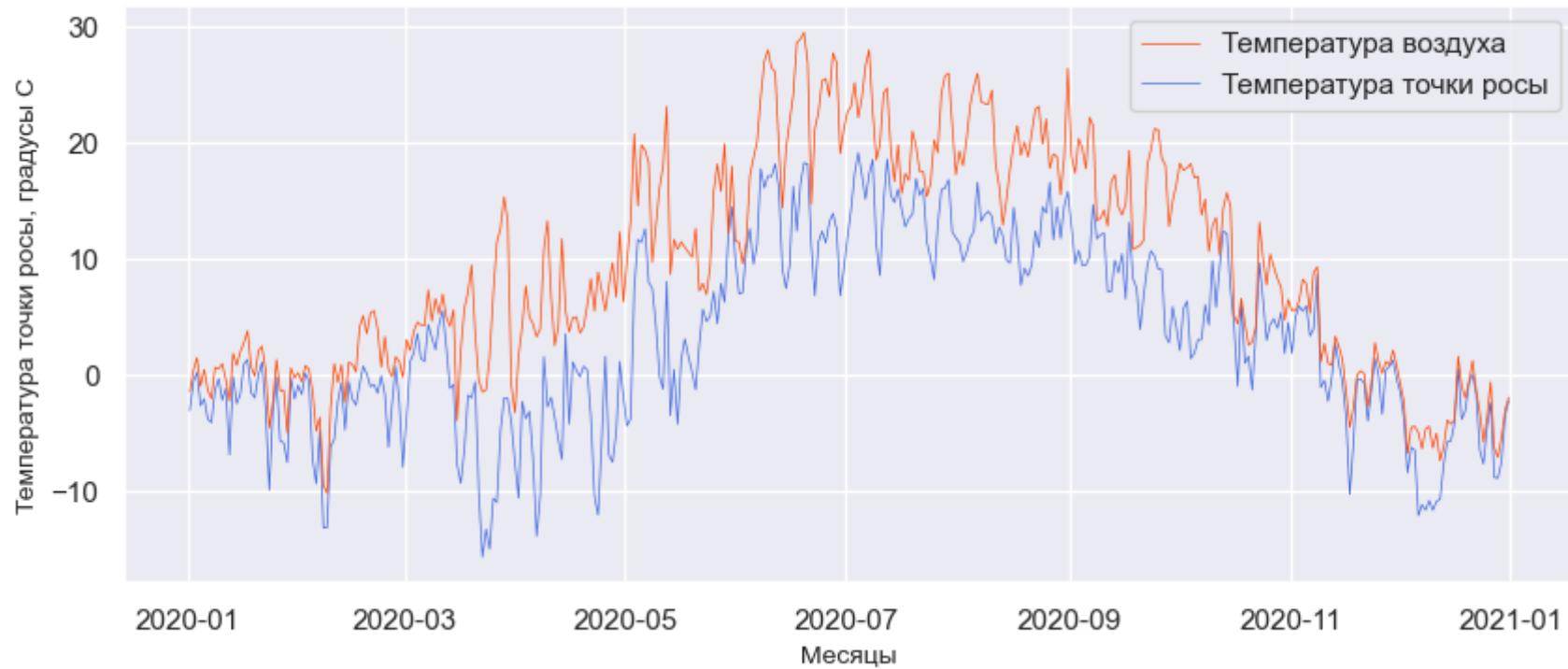
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2022 год



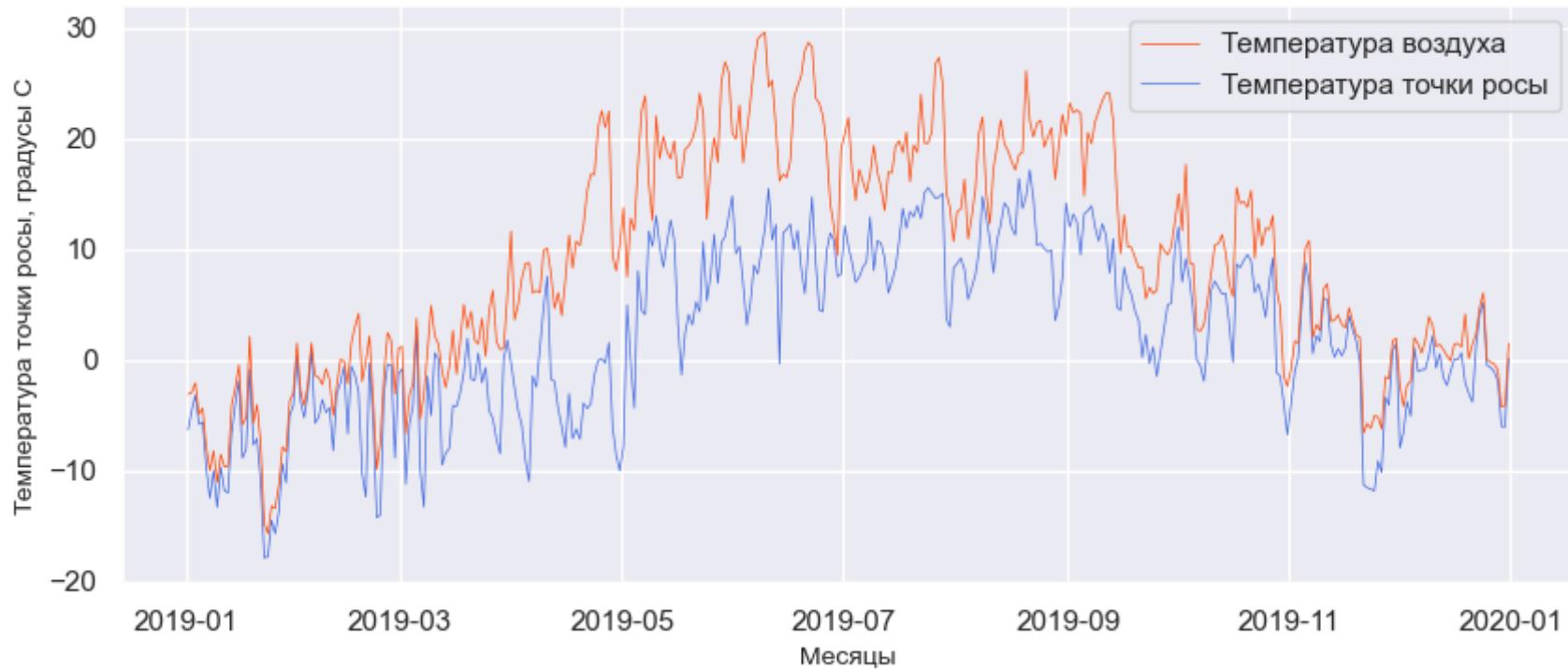
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2021 год



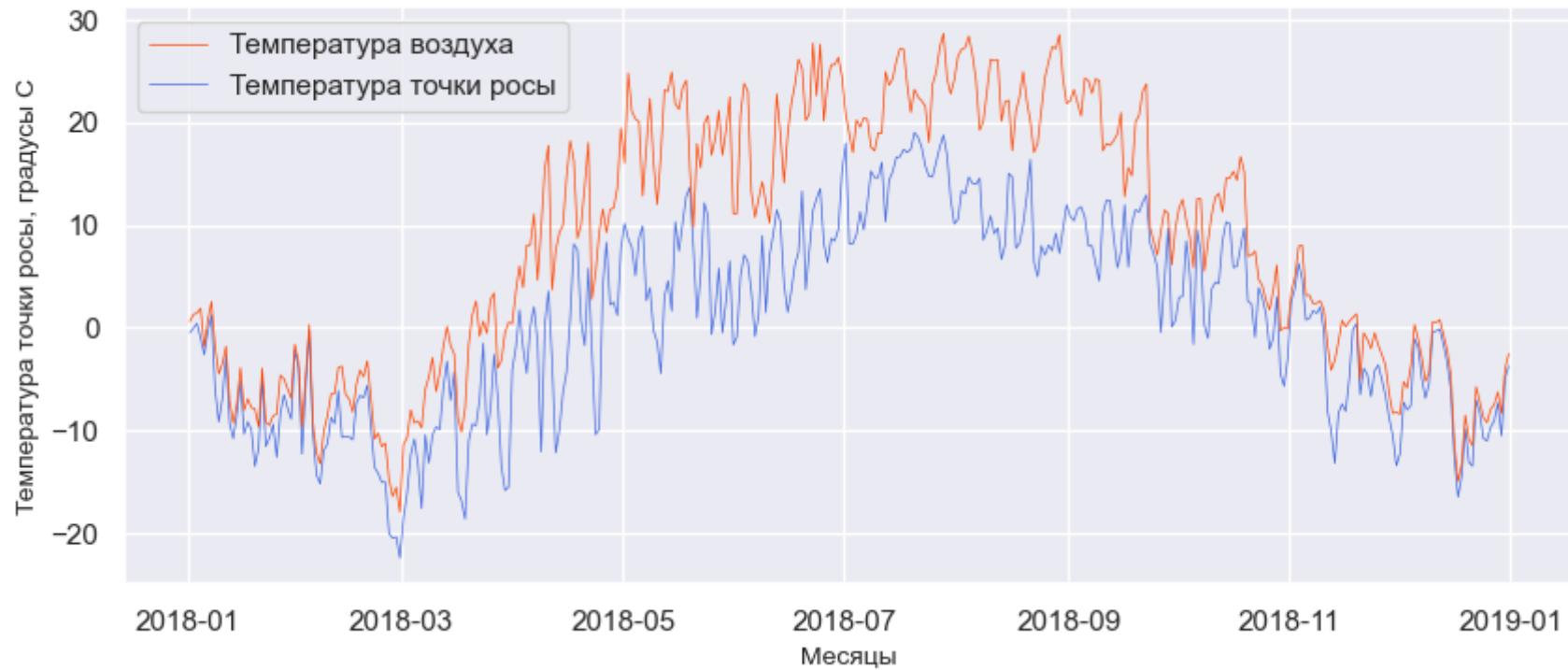
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2020 год



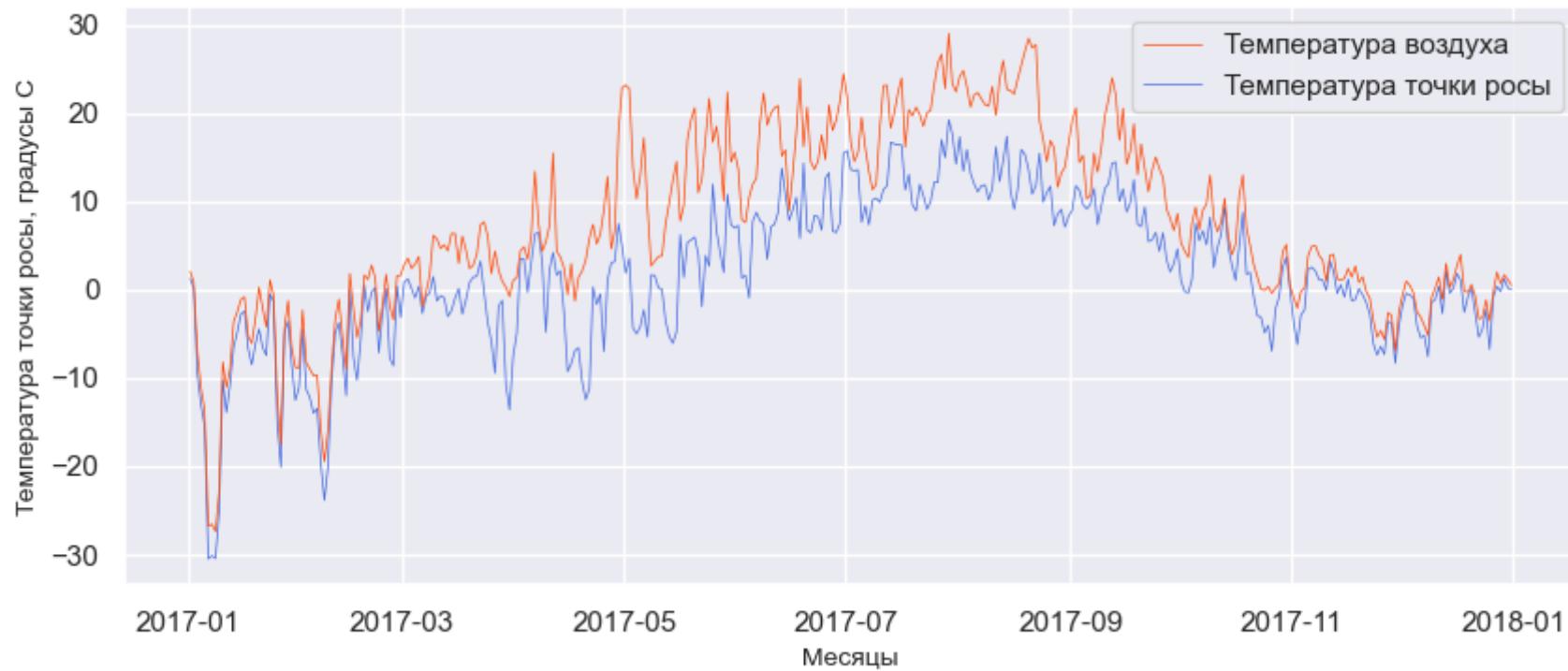
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2019 год



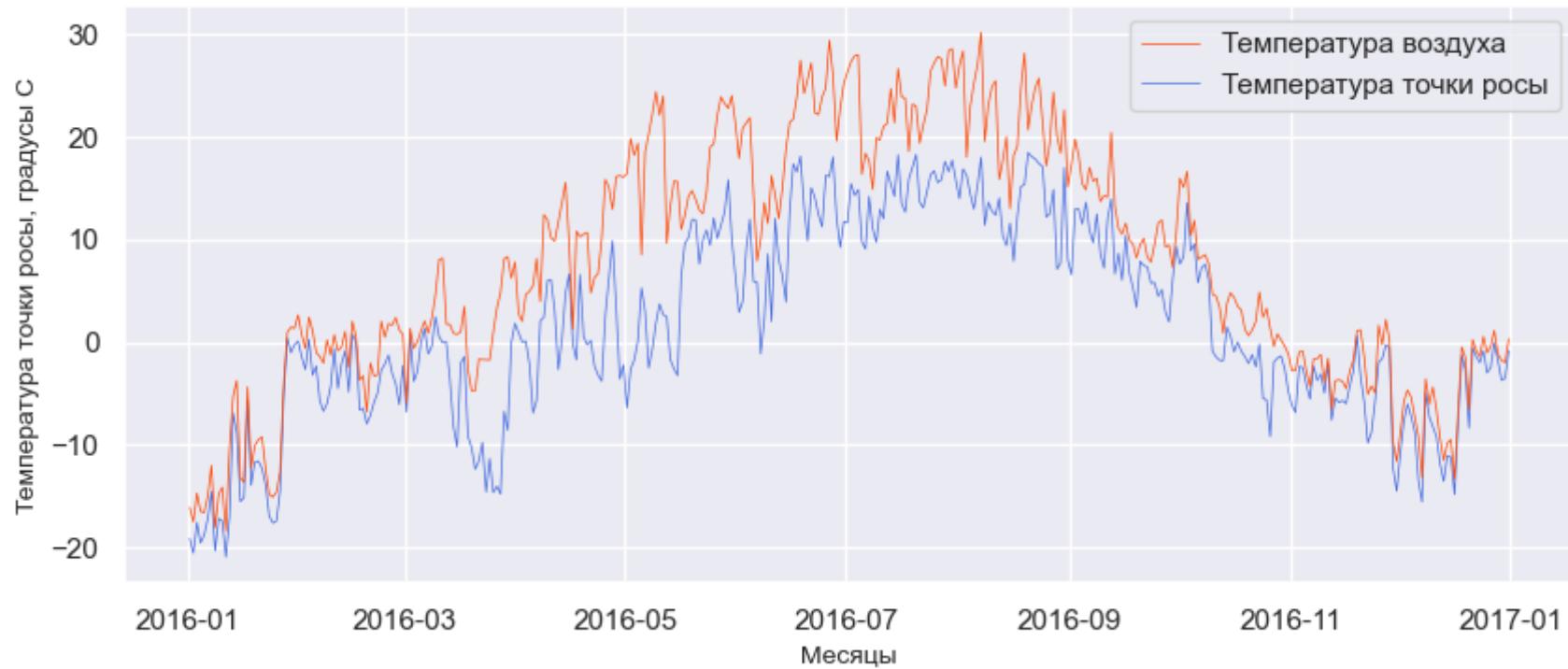
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2018 год



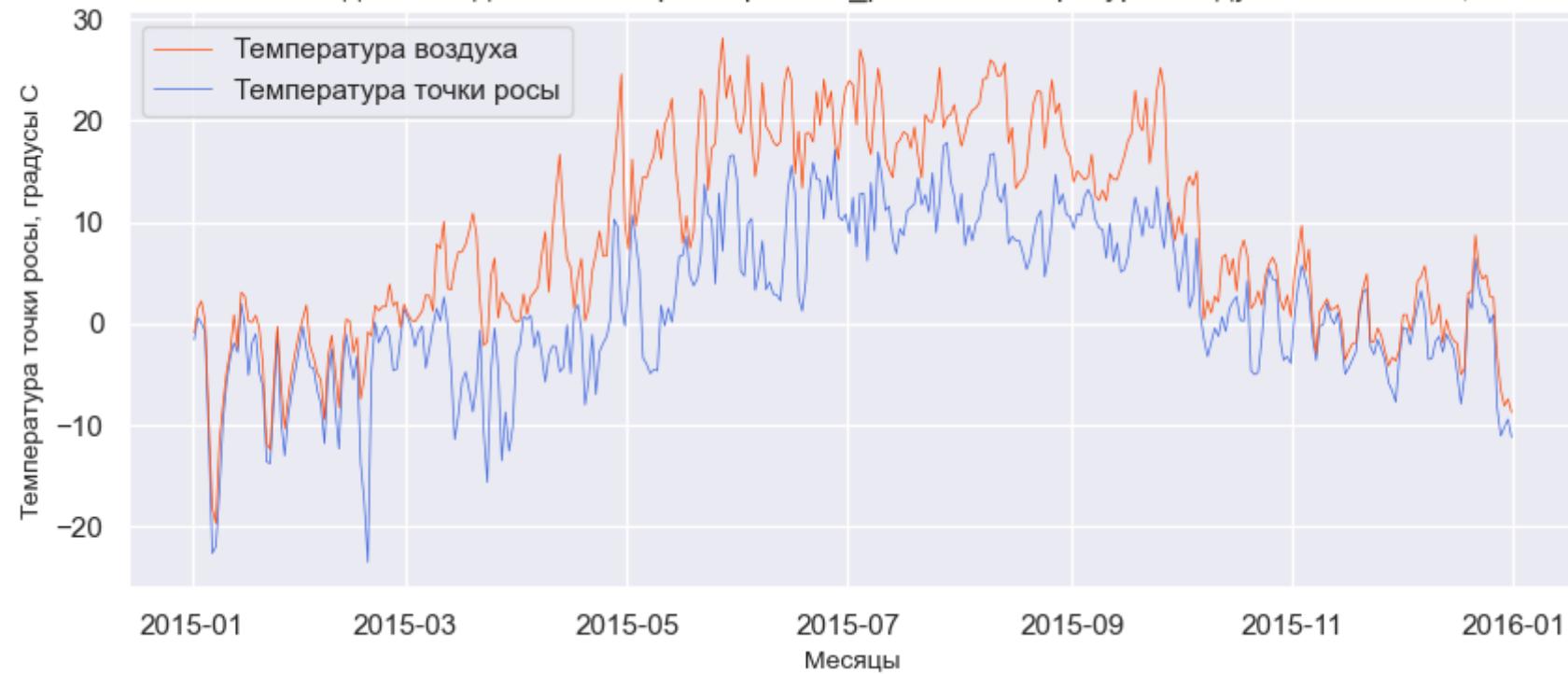
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2017 год



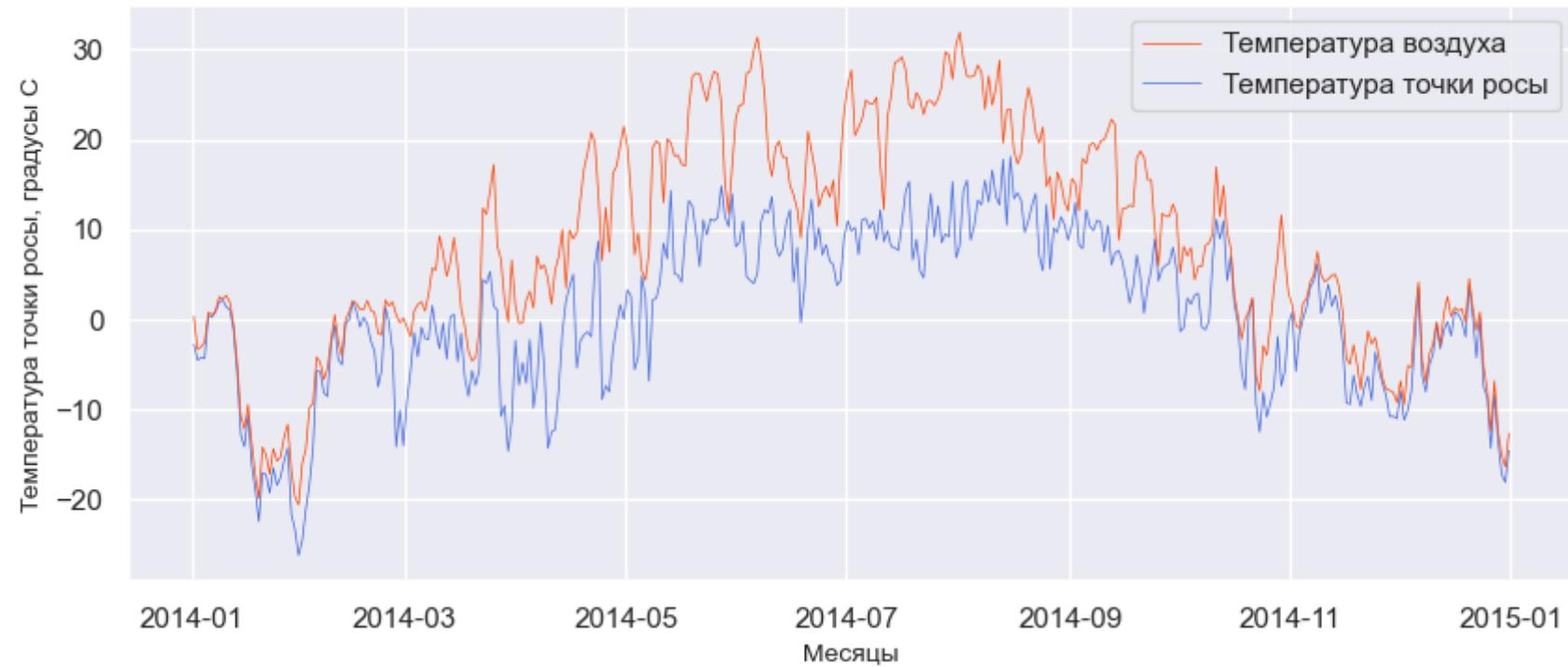
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2016 год



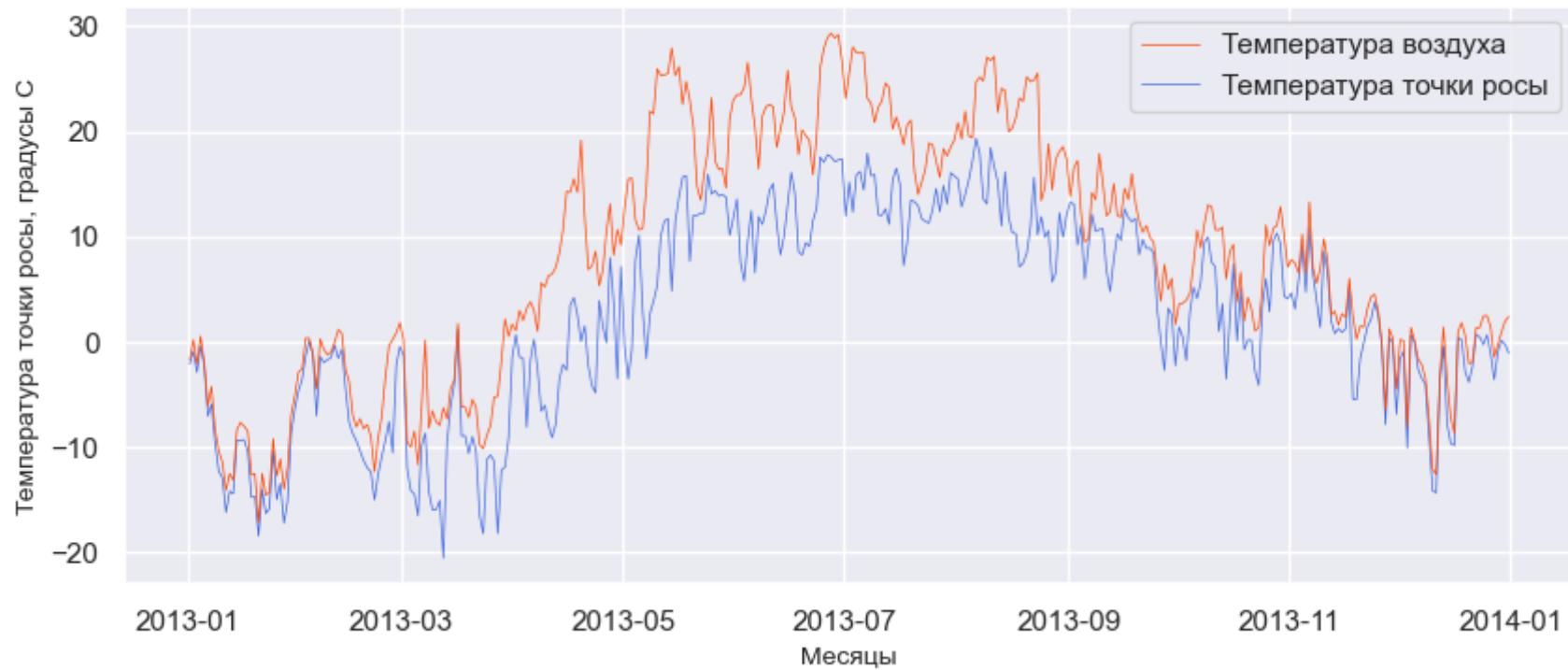
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2015 год



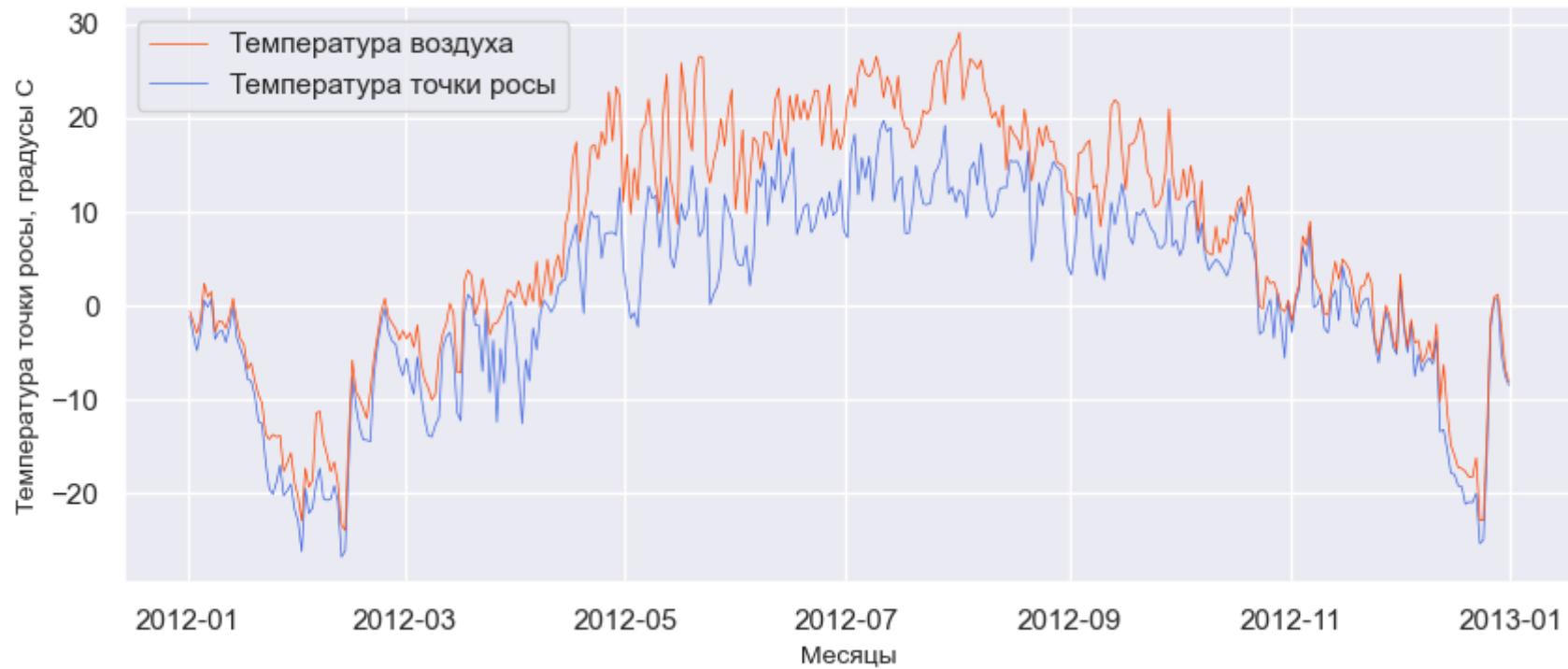
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2014 год



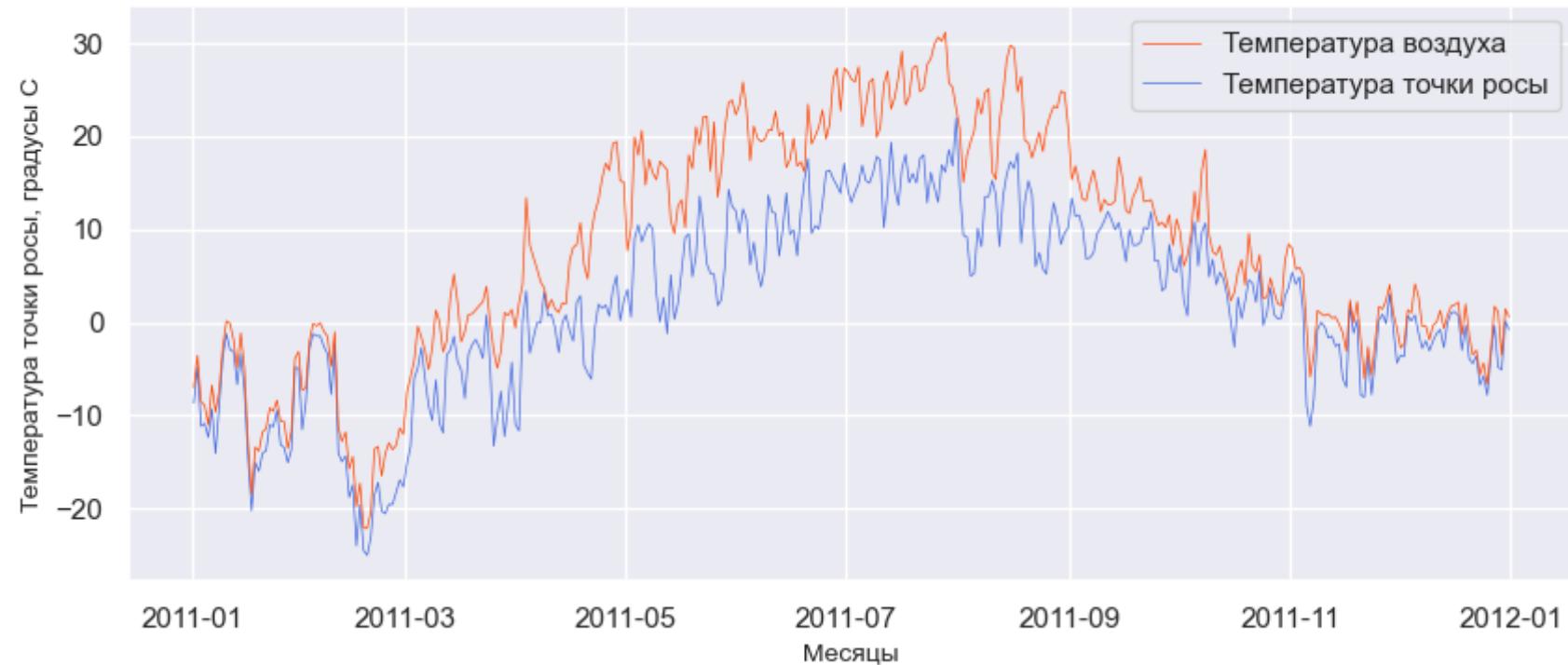
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2013 год



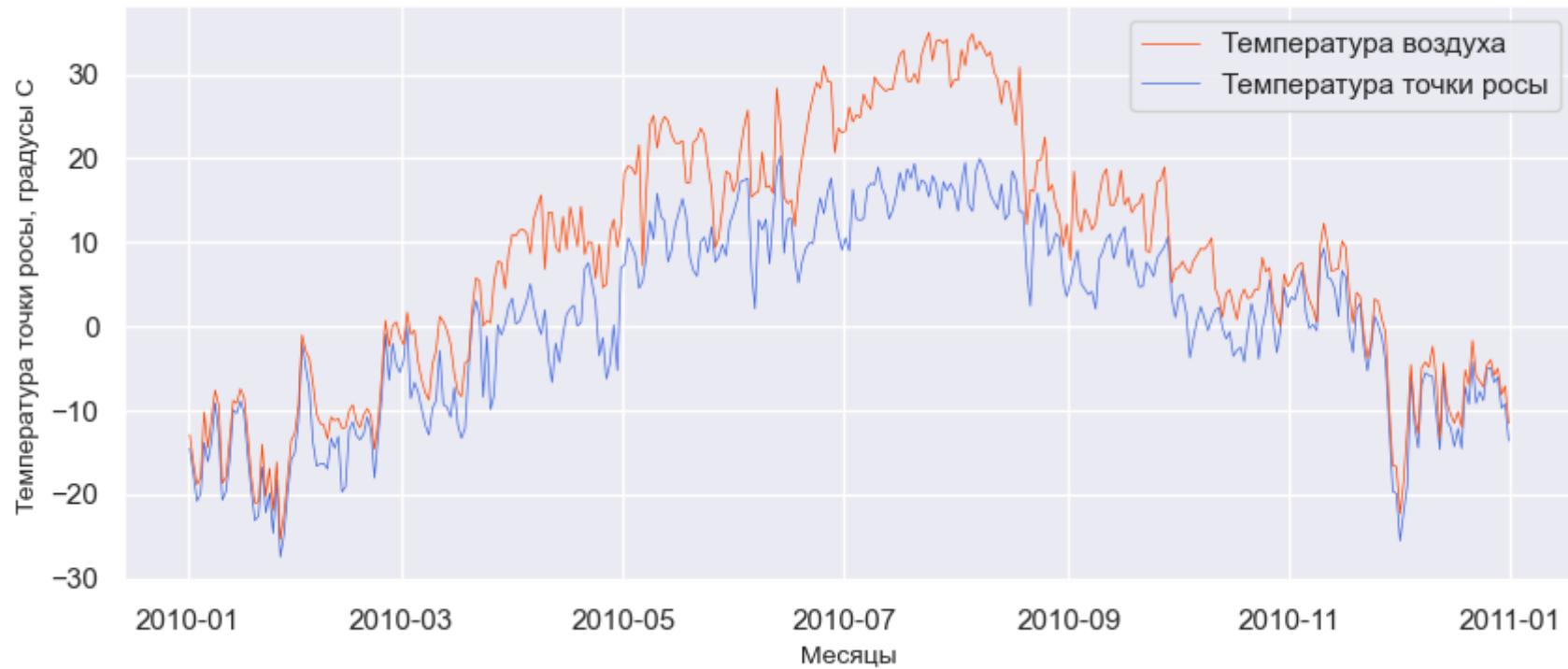
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2012 год



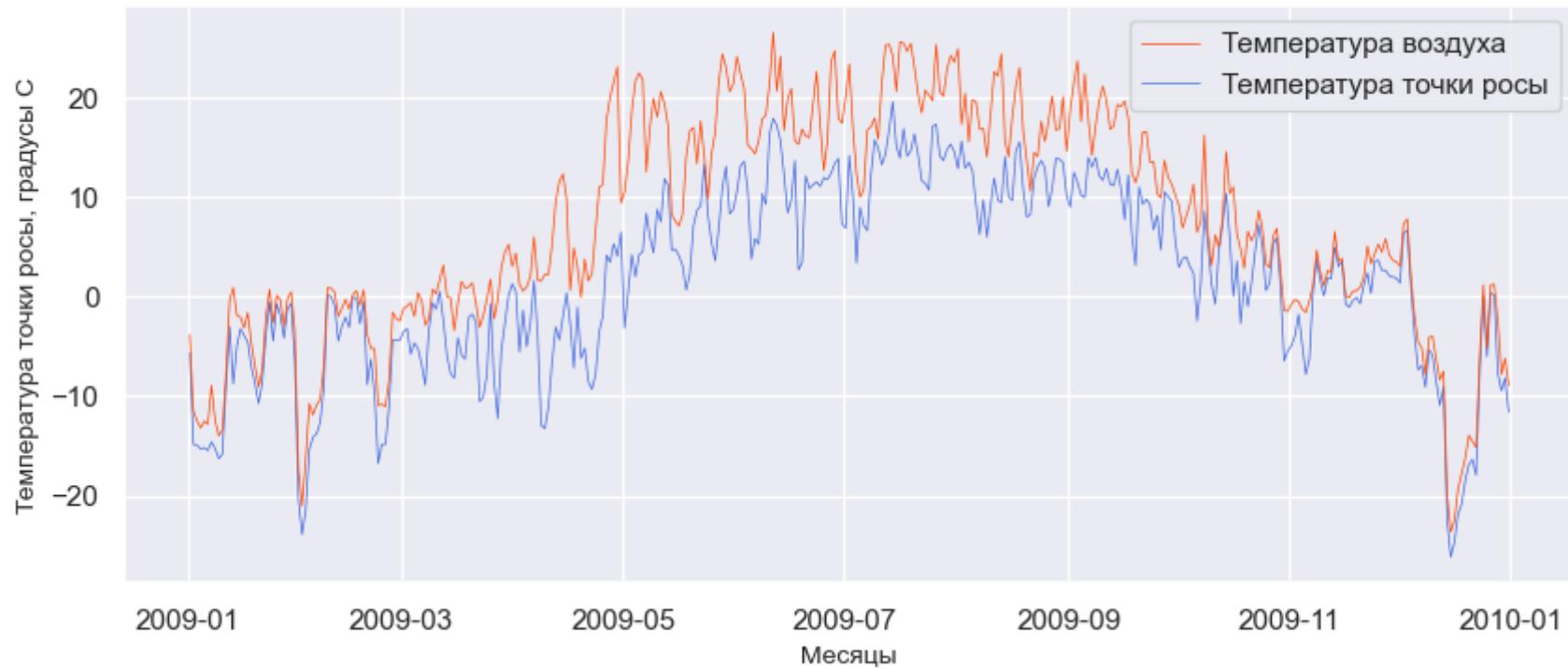
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2011 год



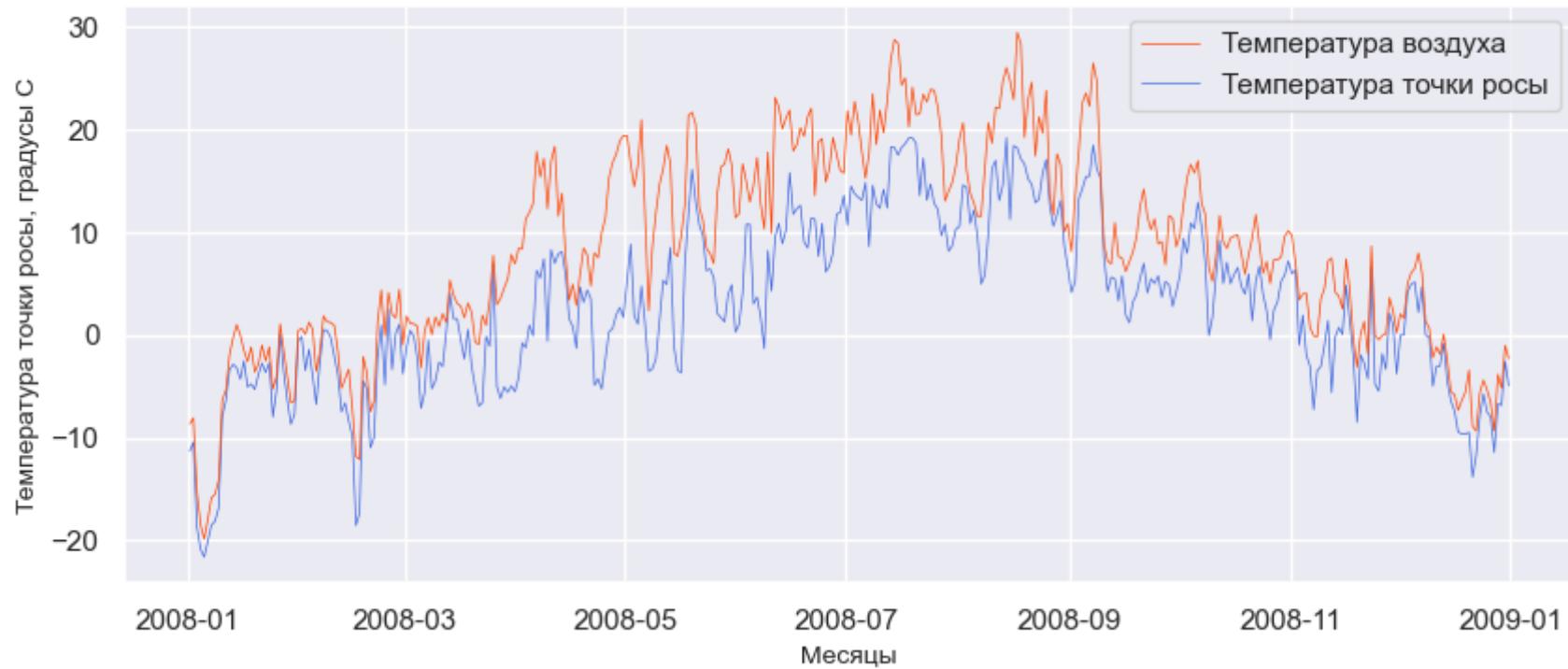
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2010 год



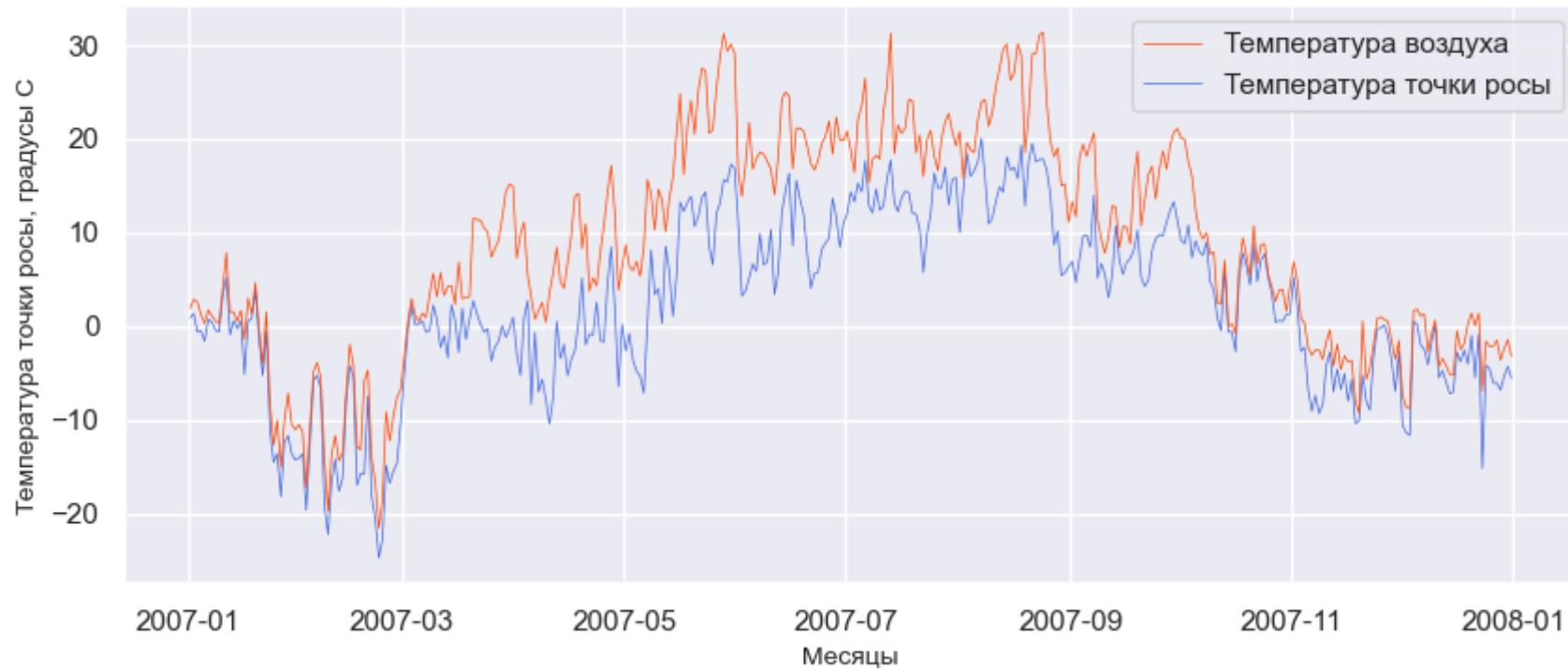
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2009 год



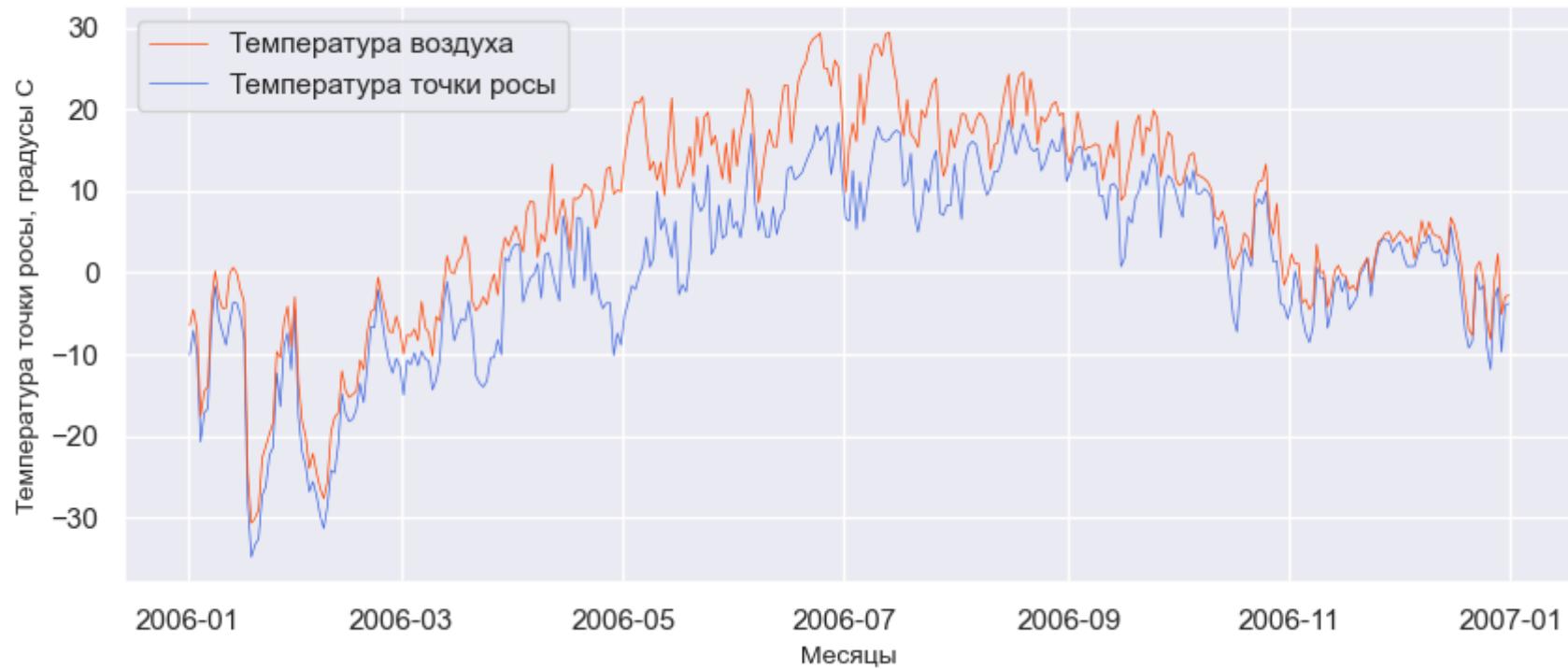
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2008 год



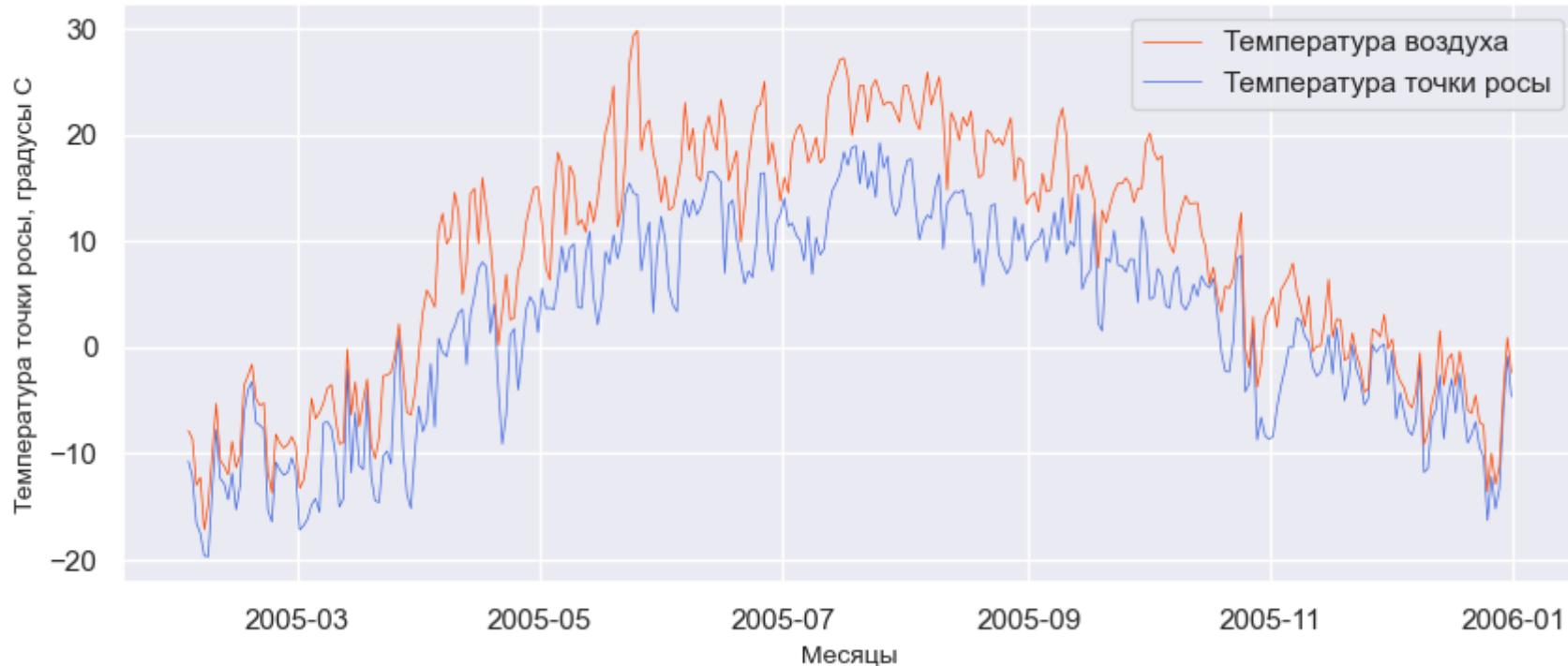
Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2007 год



Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2006 год



### Чашниково: Ежедневная динамика параметра Dew\_point и температуры воздуха на 12 часов, 2005 год



#### 5.2.4. Сохранение полученных данных в файлы

In [231...]

```
# # Определённые выше пути к файлам данных:  
# path  
# raw_path1  
# raw_path2  
  
# Создадим новые значения директорий  
predict_path1 = f'{path}predict/{PARAMETER52}/locations/'  
predict_path2 = f'{path}predict/{PARAMETER52}'  
  
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_'+PARAMETER52 + '.csv ->', end=' ')
dict_df_parameters['df_'+PARAMETER52].to_csv(
    path_or_buf=f'{predict_path2}df_{PARAMETER52}.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\_Dew\_point.csv -> DONE!

#### **ПРОДОЛЖЕНИЕ В ТЕТЕРАДИ 4**

## Разное

Вернём значения отображения к настройкам по умолчанию

```
In [232...]: 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 [233...]: InteractiveShell.ast_node_interactivity = "last_expr" # Отображение результата только последней строки кода
```

```
In [ ]:
```

