

# MASTER - Notebook 2.i

Matteo Grazioso 884055

```
In [ ]: # Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates
from datetime import datetime
import json
import warnings
warnings.filterwarnings('ignore')

import myfunctions as mf # Custom functions
```

```
In [ ]: # Display all columns and all rows
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
In [ ]: # The file contains the data of the validation of tickets in the city of public transport of Venice.
# The file has been created by the Notebook 1.ipynb

# Import the data into a dataframe of a txt file
path = 'data/processed/dataset_cleaned_validazioni.txt'
# path = 'data/processed/dataset_cleaned_esportazioneCompleta.txt'

df = pd.read_csv(path, header=0, sep='\t')

# Save the name of the file in a variable for future use extracting the name of the file from the path
file_name = path.split('_')[-1].split('.')[0]

# Display the first 5 rows of the dataframe
df.head()
```

```
# Convert the column 'DATA' to datetime format
df['DATA'] = pd.to_datetime(df['DATA'], format='%Y-%m-%d')
```

```
In [ ]: df.head()
```

Out [ ]:		DATA	ORA	DATA_VALIDAZIONE	SERIALE	FERMATA	DESCRIZIONE	TITOLO	TICKET_CODE	DESCRIZIONE_TITOLO
	0	2022-05-13	00:00:00	2022-05-13 00:00:00	65676291870913797	5089	FERROVIA	11149	4	7GG-TPL 43,60-COMVE16,40
	1	2022-05-13	00:00:00	2022-05-13 00:00:00	36141384536591364	5032	FERROVIA	11107	2	48H-TPL 24,90-COMVE5,10
	2	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856606063108	5031	P.LE ROMA	11108	3	72H-TPL 33,40-COMVE6,60
	3	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856474364932	506	VENEZIA	11261	1	DAILY PASS VENEZIA - AVM
	4	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856606062852	5031	P.LE ROMA	11108	3	72H-TPL 33,40-COMVE6,60

```
In [ ]: df.tail()
```

Out [ ]:		DATA	ORA	DATA_VALIDAZIONE	SERIALE	FERMATA	DESCRIZIONE	TITOLO	TICKET_CODE	DESCRIZIONE_TI'
	4427556	2022-07-15	02:27:00	2022-07-15 02:27:00	37271982183271940	4525	SANTA MARIA	11261	1	DAILY PASS VENE
	4427557	2022-07-15	02:27:00	2022-07-15 02:27:00	37271982183274756	4525	SANTA MARIA	11261	1	DAILY PASS VENE
	4427558	2022-07-15	04:33:00	2022-07-15 04:33:00	36088514819663876	5030	P.LE ROMA	5	7	75'-TPL 6,64-COMVI
	4427559	2022-07-15	05:06:00	2022-07-15 05:06:00	40832955551087108	509	VENEZIA	12101	7	BIGL.AUT.75'MESTRE/I
	4427560	2022-07-15	05:13:00	2022-07-15 05:13:00	40832947760207876	509	VENEZIA	12101	7	BIGL.AUT.75'MESTRE/I

---

## Focus on specific types of tickets or ticket codes

```
In [ ]: def to_datetime(date: np.datetime64) -> datetime:
        """
        Converts a numpy datetime64 object to a python datetime object
        Input:
            date - a numpy datetime64 object
        Output:
            DATE - a python datetime object

        Credit: Brian Blaylock on GitHub Gist https://gist.github.com/blaylockbk/1677b446bc741ee2db3e943ab7e4cabd
        """
        # timestamp = ((date - np.datetime64('1970-01-01T00:00:00')) / np.timedelta64(1, 's'))
        # return datetime.utcfromtimestamp(timestamp)
        # date = pd.to_datetime(date)
        date = datetime.strptime(date, '%Y-%m-%d')
        return date

In [ ]: # Focus on a specific day
def number_of_tickets_per_day(df_d: pd.DataFrame, target_ticket_code_or_type: str, is_ticket_code: bool) -> None:
    """
    This function plots the number of validations of the specified ticket code for each day.
    :param df: the dataframe
    :param target_ticket_code: the ticket code
    :param is_ticket_code: a boolean that specifies if the target is a ticket code or a ticket type
    :return: None
    """
    # Group the dataframe by date and hour and count the number of validations of the specified ticket code
    df_d = df_d.groupby('DATA').count()['SERIALE'].reset_index()
    df_d['cumulative_sum'] = df_d.groupby('DATA')['SERIALE'].cumsum()

    # Plot the cumulative sum of the number of validations of the target ticket code or type for each day
    plt.figure(figsize=(20, 10))
    plt.plot(df_d['DATA'], df_d['cumulative_sum'])

    if is_ticket_code:
        descr = mf.get_ticket_code_description(target_ticket_code_or_type)
        plt.title('Cumulative sum of the number of validations of the ticket code "{}" - "{}" for each day'.format(
```

```

else:
    plt.title('Cumulative sum of the number of validations of the ticket type "{}" for each day'.format(target_
plt.xlabel('Date - days', fontsize=15)
plt.ylabel('Cumulative sum', fontsize=15)

# Calculate the step of the y-axis
step = int(df_d['cumulative_sum'].max()/10)
# Manage the y-axis
plt.yticks(ticks=np.arange(0, df_d['cumulative_sum'].max()+step, step))

# Manage the x-axis:
# rotate the labels of the x-axis
plt.xticks(rotation=45)

# Add a point for each day
plt.scatter(df_d['DATA'], df_d['cumulative_sum'], color='darkblue', s=20)

# Add the date in the plot
for i, txt in enumerate(df_d['DATA']):
    # Set the date format in %Y-%m-%d
    txt = txt.strftime('%m-%d')
    # Print the date for each point every 10 days or if it is the first or the last point
    # or if it is the point with the highest number of validations or if it is the point with the lowest number
    if i % 10 == 0 or i == 0 or i == len(df_d['DATA'])-1 or i == df_d['SERIALE'].idxmax() or i == df_d['SERIALE
        plt.annotate(txt, (df_d['DATA'][i], df_d['cumulative_sum'][i]), fontsize=10)

# Add a grid
plt.grid()

# Highlight the day with the highest number of validations and the day with the lowest number of validations
max = df_d[df_d['SERIALE'] == df_d['SERIALE'].max()]
plt.scatter(max['DATA'], max['cumulative_sum'], color='red', s=50)
min = df_d[df_d['SERIALE'] == df_d['SERIALE'].min()]
plt.scatter(min['DATA'], min['cumulative_sum'], color='green', s=50)

# Add the legend
plt.legend(['Number of validations', 'Day with the highest number of validations', 'Day with the lowest number

plt.show()

```

```

# Focus on a specific day: min and max number of tickets per day
def min_max_number_of_tickets_per_day(df_d: pd.DataFrame, target_ticket_code_or_type: str, is_ticket_code: bool) ->
    """
        This function prints the day with the highest and the day with the lowest number of validations of the spec
        :param df: the dataframe
        :param target_ticket_code_or_type: the ticket code or ticket type
        :param is_ticket_code: a boolean that specifies if the target is a ticket code or a ticket type
        :return: None
    """

    # Group the dataframe by date and hour and count the number of validations of the specified ticket code
    df_d = df_d.groupby('DATA').count()['SERIALE'].reset_index()
    max = df_d[df_d['SERIALE'] == df_d['SERIALE'].max()]
    min = df_d[df_d['SERIALE'] == df_d['SERIALE'].min()]

    # Composing the string to print the result, specifying:
    # - if the target is a ticket code or a ticket type,
    # - converting the date to string format '%Y-%m-%d' and
    # - if is a ticket code, getting the description of the ticket code
    if is_ticket_code:
        # Get the description of the ticket code
        descr = ' - ' + mf.get_ticket_code_description(target_ticket_code_or_type) + ''
    else:
        descr = ''

    s = 'The {} number of validations of the {} "{}" + descr + ' was {} with {} validations'
    # Print the result
    for i in range(0, 2):
        if i == 0:
            if is_ticket_code:
                print(s.format('highest', 'ticket code', target_ticket_code_or_type, max['DATA'].values[0].astype('d
            else:
                print(s.format('highest', 'ticket type', target_ticket_code_or_type, max['DATA'].values[0].astype('d
        else:
            if is_ticket_code:
                print(s.format('lowest', 'ticket code', target_ticket_code_or_type, min['DATA'].values[0].astype('d
            else:
                print(s.format('lowest', 'ticket type', target_ticket_code_or_type, min['DATA'].values[0].astype('d

# Focus on a specific month
def barplot_number_of_tickets_per_month(df_m: pd.DataFrame, target_ticket_code_or_type: str, is_ticket_code: bool)

```

```

"""
    This function plots the number of validations of the specified ticket code or ticket type for each month.
    :param df: the dataframe
    :param target_ticket_code_or_type: the ticket code or ticket type
    :param is_ticket_code: a boolean that specifies if the target is a ticket code or a ticket type
    :return: None
"""

# Set all the values of the date on the first day of the month
df_m['DATA'] = df_m['DATA'].dt.strftime('%Y-%m-01')

# Group the dataframe by date and count the number of validations of the specified ticket code
df_m = df_m.groupby('DATA').count()['SERIALE'].reset_index(drop=False)

# Convert the date to string format '%Y/%m'
df_m['DATA'] = df_m['DATA'].astype('datetime64[M]').astype(str)

# Remove the last 3 characters of the date (the day) to have the date in the format '%Y/%m'
for i in range(0, len(df_m['DATA'])):
    df_m['DATA'][i] = df_m['DATA'][i][::-3]

# Plot the cumulative sum of the number of validations of the target ticket code or type for each month
plt.figure(figsize=(20, 10))
plt.bar(df_m['DATA'], df_m['SERIALE'])

# If the target is a ticket code, get the description of the ticket code, otherwise focus on the ticket type
if is_ticket_code:
    descr = mf.get_ticket_code_description(target_ticket_code_or_type)
    plt.title('Cumulative sum of the number of validations of the ticket code "{}" - "{}" for each month'.format(
        target_ticket_code_or_type, descr))
else:
    plt.title('Cumulative sum of the number of validations of the ticket type "{}" for each month'.format(target_ticket_code_or_type))
plt.xlabel('Date - months', fontsize=15)
plt.ylabel('Cumulative sum', fontsize=15)

# Calculate the step of the y-axis
step = int(df_m['SERIALE'].max()/10)
# Manage the y-axis
plt.yticks(ticks=np.arange(0, df_m['SERIALE'].max()+step, step))

# Manage the x-axis: rotate the labels of the x-axis
# Print the x-axis labels as '%Y/%m', so remove the day from the date and print only the year and the month of

```

```

# for i in range(0, len(df_m['DATA'])):
    #df_m['DATA'][i] = df_m['DATA'][i][::-3]

plt.xticks(rotation=45, ticks=df_m['DATA'])

# Add the value of the number of validations of the target ticket code or type for each month
for i in range(0, len(df_m['DATA'])):
    plt.text(x=df_m['DATA'][i], y=df_m['SERIALE'][i], s=df_m['SERIALE'][i], ha='center', va='bottom', fontsize=

# Add a grid
plt.grid(axis='y', linestyle='--')

plt.show()

def min_max_number_of_tickets_per_month(df_m: pd.DataFrame, target_ticket_code_or_type: str, is_ticket_code: bool)
    """
        This function prints the month with the highest and the month with the lowest number of validations of the
        :param df: the dataframe
        :param target_ticket_code_or_type: the ticket code or ticket type
        :param is_ticket_code: a boolean that specifies if the target is a ticket code or a ticket type
        :return: None
    """
    # Convert DATA to datetime
    df_m['DATA'] = pd.to_datetime(df_m['DATA'], format='%Y/%m/%d')
    # Set all the values of the date on the first day of the month
    df_m['DATA'] = df_m['DATA'].dt.strftime('%Y-%m-01')

    # Group the dataframe by date and hour and count the number of validations of the specified ticket code
    df_m = df_m.groupby('DATA').count()['SERIALE'].reset_index()

    max = df_m[df_m['SERIALE'] == df_m['SERIALE'].max()]
    min = df_m[df_m['SERIALE'] == df_m['SERIALE'].min()]

    # Composing the string to print the result, specifying:
    # - if the target is a ticket code or a ticket type,
    # - converting the date to string format '%Y-%m-%d' and
    # - if is a ticket code, getting the description of the ticket code
    if is_ticket_code:
        # Get the description of the ticket code
        descr = ' - ' + mf.get_ticket_code_description(target_ticket_code_or_type) + ' '

```

```

else:
    descr = ''

s = 'The {} number of validations of the {} "{}" + descr + ' was the month {} with {} validations.'
for i in range(0, 2):
    if i == 0:
        if is_ticket_code:
            print(s.format('highest', 'ticket code', target_ticket_code_or_type, max['DATA'].values[0][:3], ma
        else:
            print(s.format('highest', 'ticket type', target_ticket_code_or_type, max['DATA'].values[0][:3], ma
    else:
        if is_ticket_code:
            print(s.format('lowest', 'ticket code', target_ticket_code_or_type, min['DATA'].values[0][:3], min
        else:
            print(s.format('lowest', 'ticket type', target_ticket_code_or_type, min['DATA'].values[0][:3], min

```

```

In [ ]: # Focus on all TICKET_CODES
def focus_on_all_ticket_codes(df_tca: pd.DataFrame, dictionary: str) -> None:
    """
    This function focuses on the different ticket codes.
    :param df_tca: the dataframe
    :param dictionary: the dictionary with the ticket codes and their description
    :return: None
    """
    # Open the dictionary
    dict_ticket_code = mf.open_dict_ticket_codes()

    for ticket_code in dict_ticket_code.keys():
        # Select only the rows of the specified ticket code
        df_tca_sup = mf.focus_on_ticket_code(df_tca, ticket_code)
        # If the dataframe is empty, skip the ticket code but launch a warning
        if df_tca_sup.shape[0] == 0:
            print('WARNING: There are no validations of the ticket code "{}".format(ticket_code))
        else:
            # If the dataframe is not empty, focus on the ticket code
            df_tc = mf.focus_on_ticket_code(df_tca_sup, ticket_code)
            number_of_tickets_per_day(df_tc, ticket_code, is_ticket_code=True)
            min_max_number_of_tickets_per_day(df_tc, ticket_code, is_ticket_code=True)
            barplot_number_of_tickets_per_month(df_tc, ticket_code, is_ticket_code=True)
            min_max_number_of_tickets_per_month(df_tc, ticket_code, is_ticket_code=True)

```

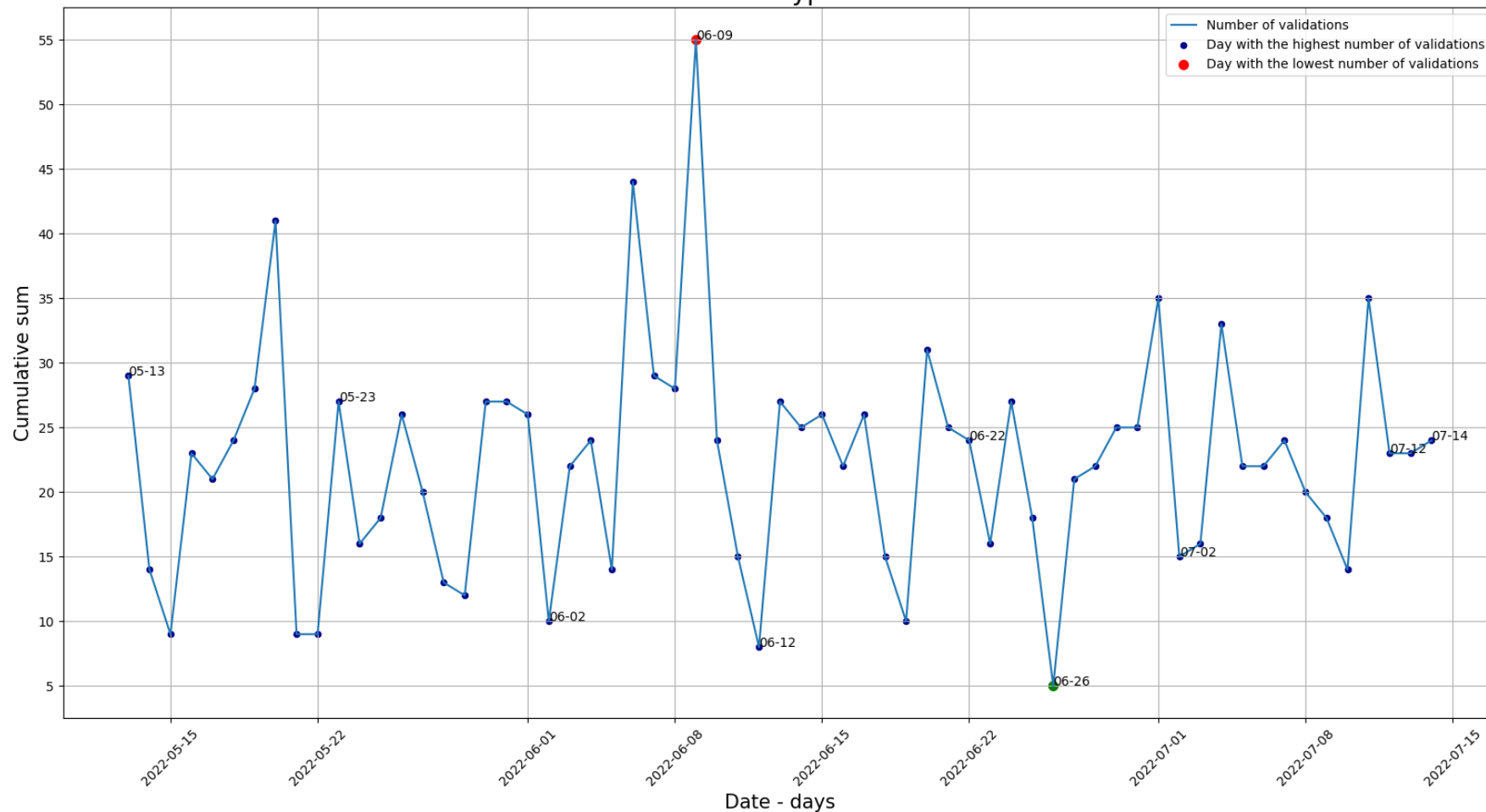


## Focus on the type of ticket named **\*abbonamento 30 gg.PeopleMover\***

The ticket is valid for 30 days and allows you to use the PeopleMover service.

```
In [ ]: target_ticket = "abbonamento 30 gg.PeopleMover".upper()
# target_ticket = "480RE ONLINE AEROBUS CS".upper()
df_PM = mf.focus_on_ticket_type(df, target_ticket)
if df_PM.shape[0] == 0:
    print('WARNING: There are no validations of the ticket type "{}".format(target_ticket))
else:
    number_of_tickets_per_day(df_PM, target_ticket, is_ticket_code=False)
    min_max_number_of_tickets_per_day(df_PM, target_ticket, is_ticket_code=False)
    barplot_number_of_tickets_per_month(df_PM, target_ticket, is_ticket_code=False)
    min_max_number_of_tickets_per_month(df_PM, target_ticket, is_ticket_code=False)
```

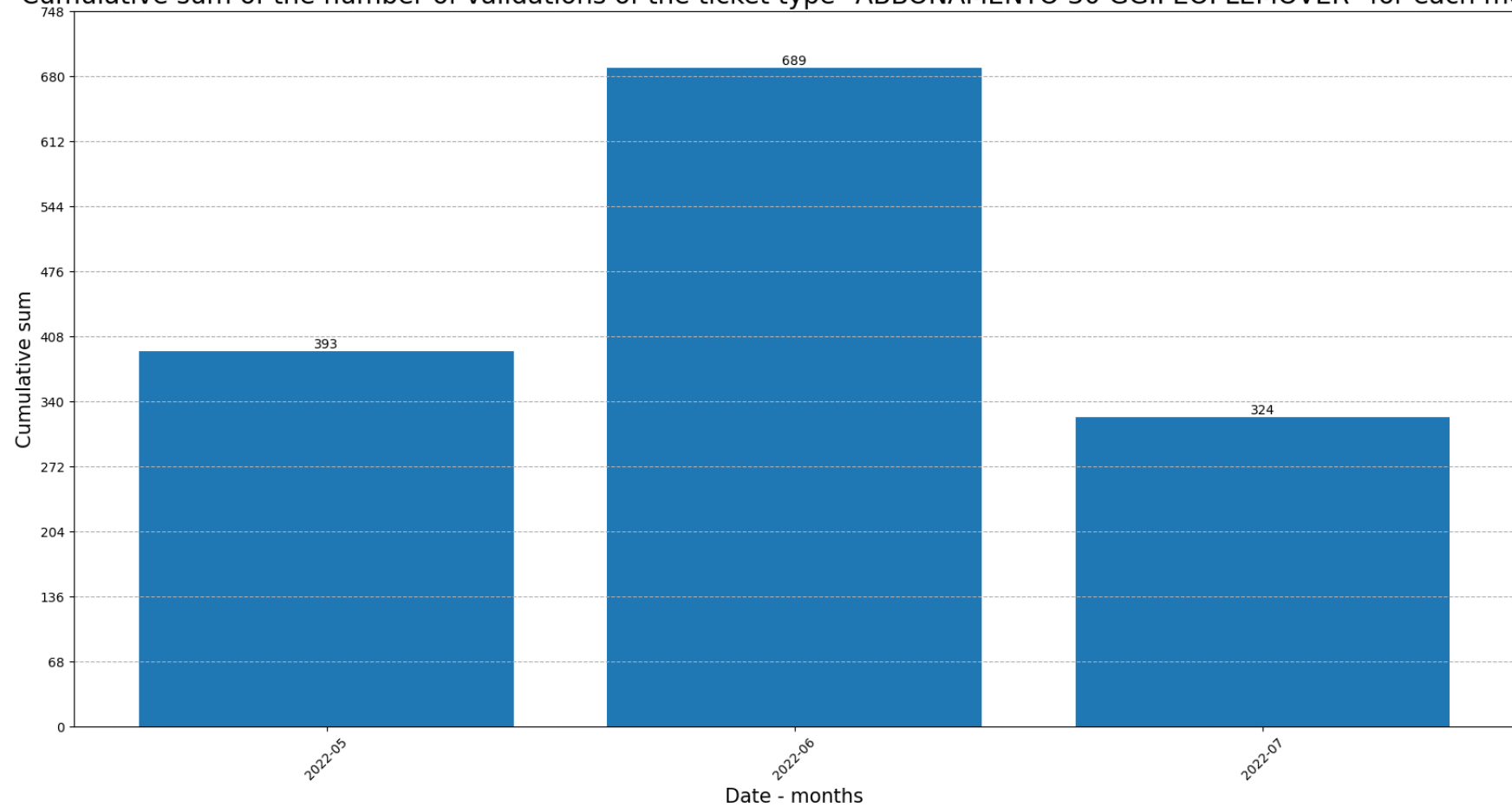
Cumulative sum of the number of validations of the ticket type "ABBONAMENTO 30 GG.PEOPLEMOVER" for each day



The highest number of validations of the ticket type "ABBONAMENTO 30 GG.PEOPLEMOVER" was 2022-06-09 with 55 validations

The lowest number of validations of the ticket type "ABBONAMENTO 30 GG.PEOPLEMOVER" was 2022-06-26 with 5 validations

Cumulative sum of the number of validations of the ticket type "ABBONAMENTO 30 GG.PEOPLEMOVER" for each month



The highest number of validations of the ticket type "ABBONAMENTO 30 GG.PEOPLEMOVER" was the month 2022-06 with 689 validations.

The lowest number of validations of the ticket type "ABBONAMENTO 30 GG.PEOPLEMOVER" was the month 2022-07 with 324 validations.

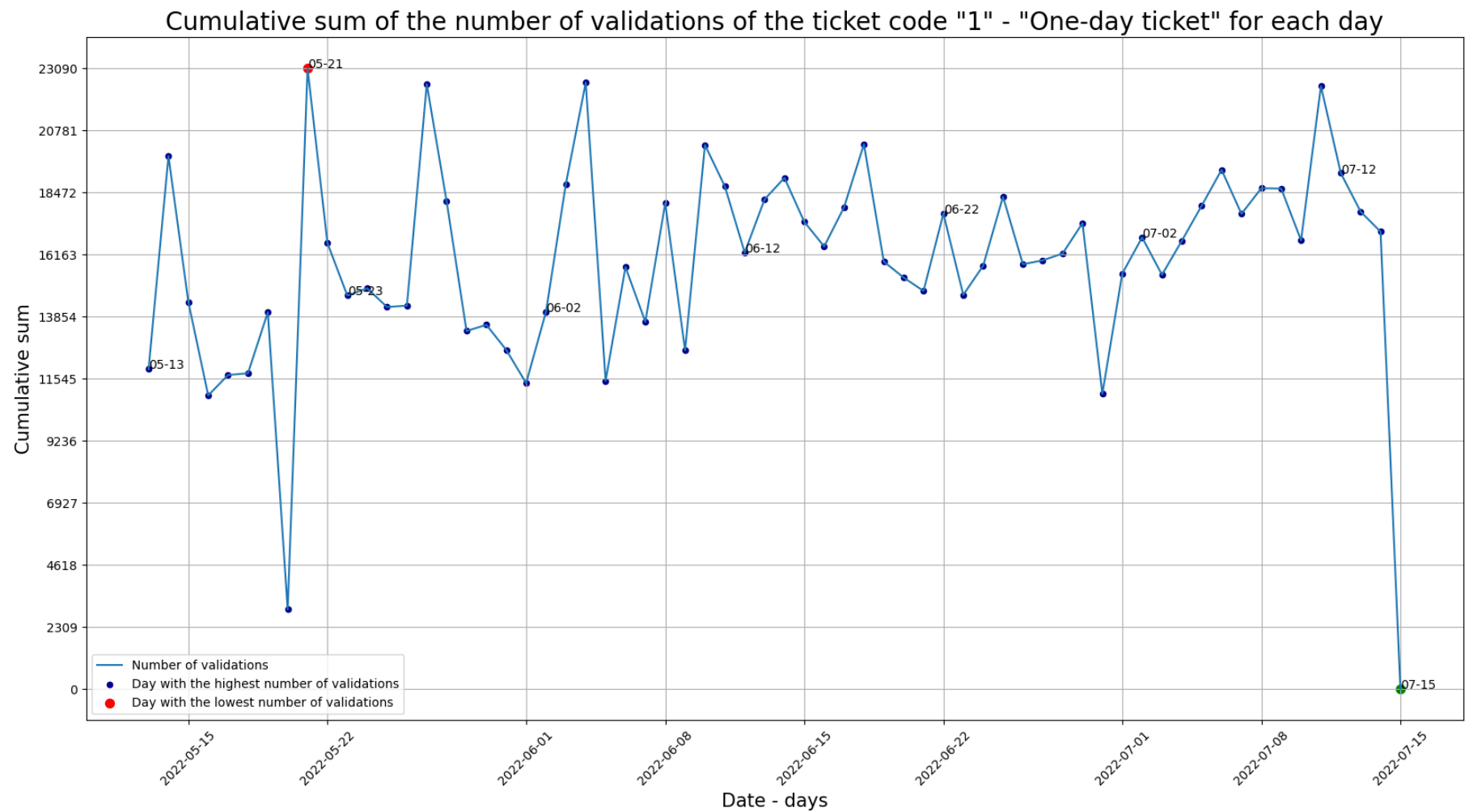
## Focus on ticket codes

```
In [ ]: df.head()
```

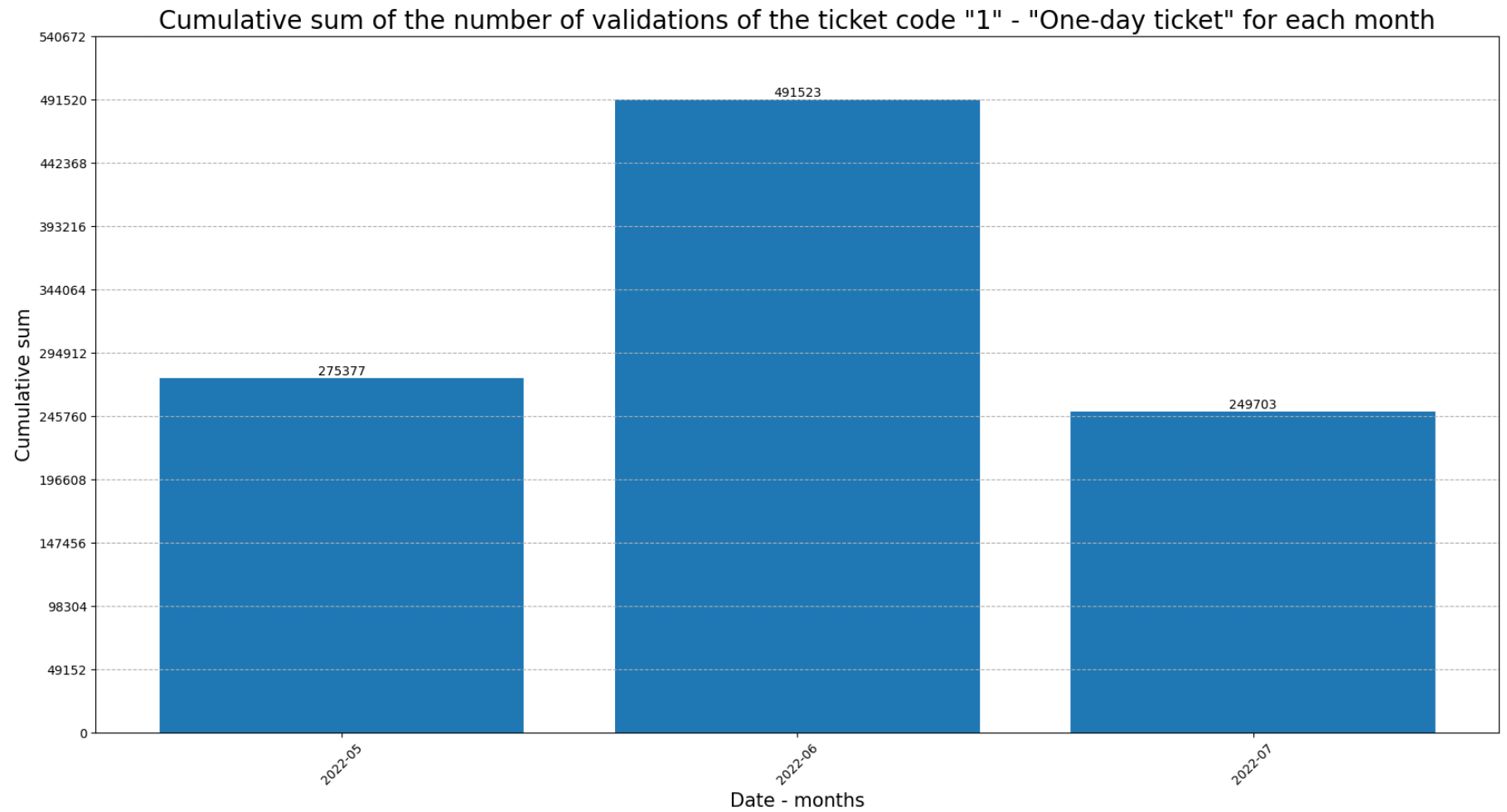
Out [ ]:

	DATA	ORA	DATA_VALIDAZIONE	SERIALE	FERMATA	DESCRIZIONE	TITOLO	TICKET_CODE	DESCRIZIONE_TITOLO
0	2022-05-13	00:00:00	2022-05-13 00:00:00	65676291870913797	5089	FERROVIA	11149	4	7GG-TPL 43,60- COMVE16,40
1	2022-05-13	00:00:00	2022-05-13 00:00:00	36141384536591364	5032	FERROVIA	11107	2	48H-TPL 24,90- COMVE5,10
2	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856606063108	5031	P.LE ROMA	11108	3	72H-TPL 33,40- COMVE6,60
3	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856474364932	506	VENEZIA	11261	1	DAILY PASS VENEZIA - AVM
4	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856606062852	5031	P.LE ROMA	11108	3	72H-TPL 33,40- COMVE6,60

```
In [ ]: # The TICKET_CODES are in the dictionary "dict_ticket_codes.json", created in Notebook 1
        focus_on_all_ticket_codes(df, 'data/dictionaries/dict_ticket_codes.json')
```

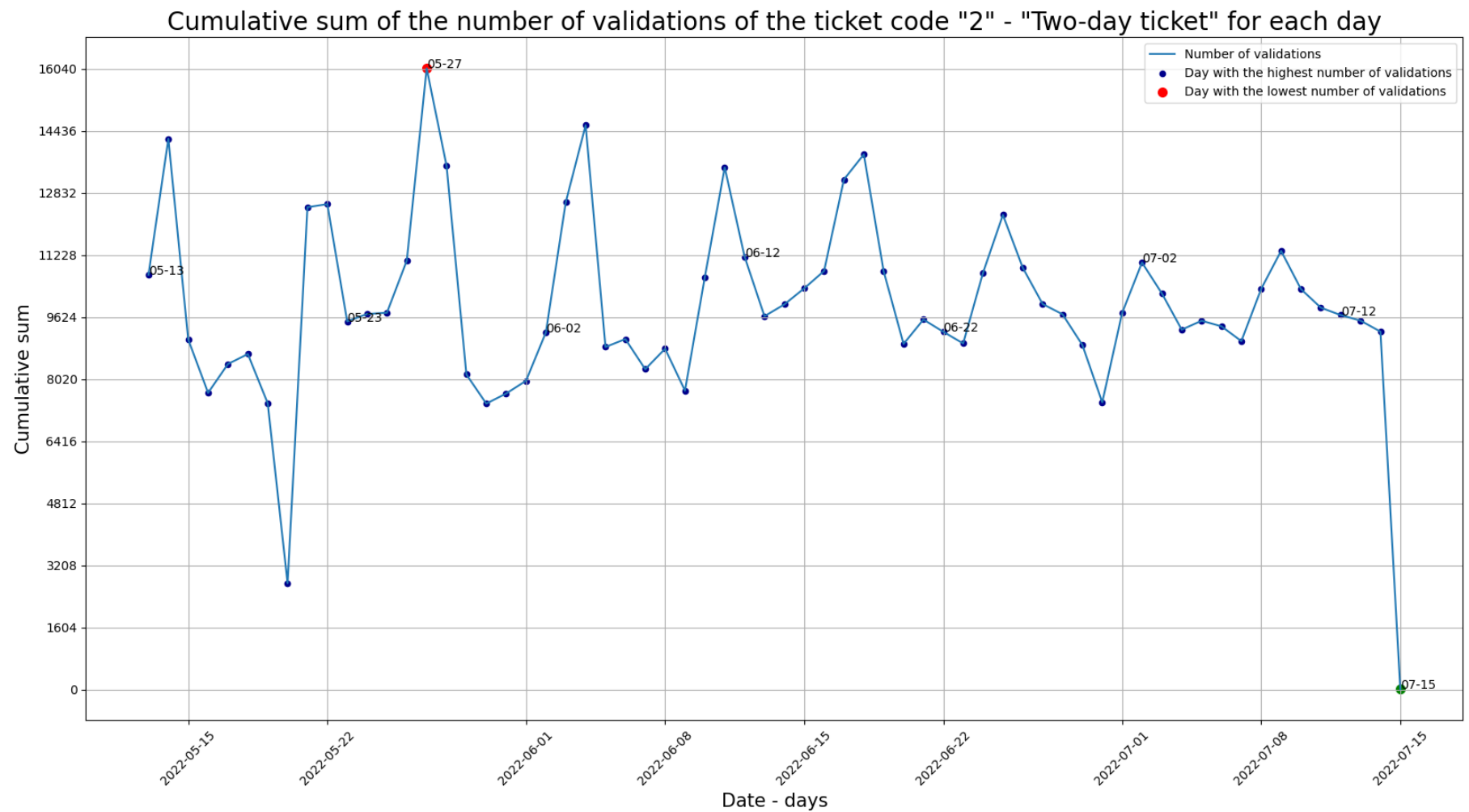


The highest number of validations of the ticket code "1" – "One-day ticket" was 2022-05-21 with 23091 validations  
The lowest number of validations of the ticket code "1" – "One-day ticket" was 2022-07-15 with 7 validations

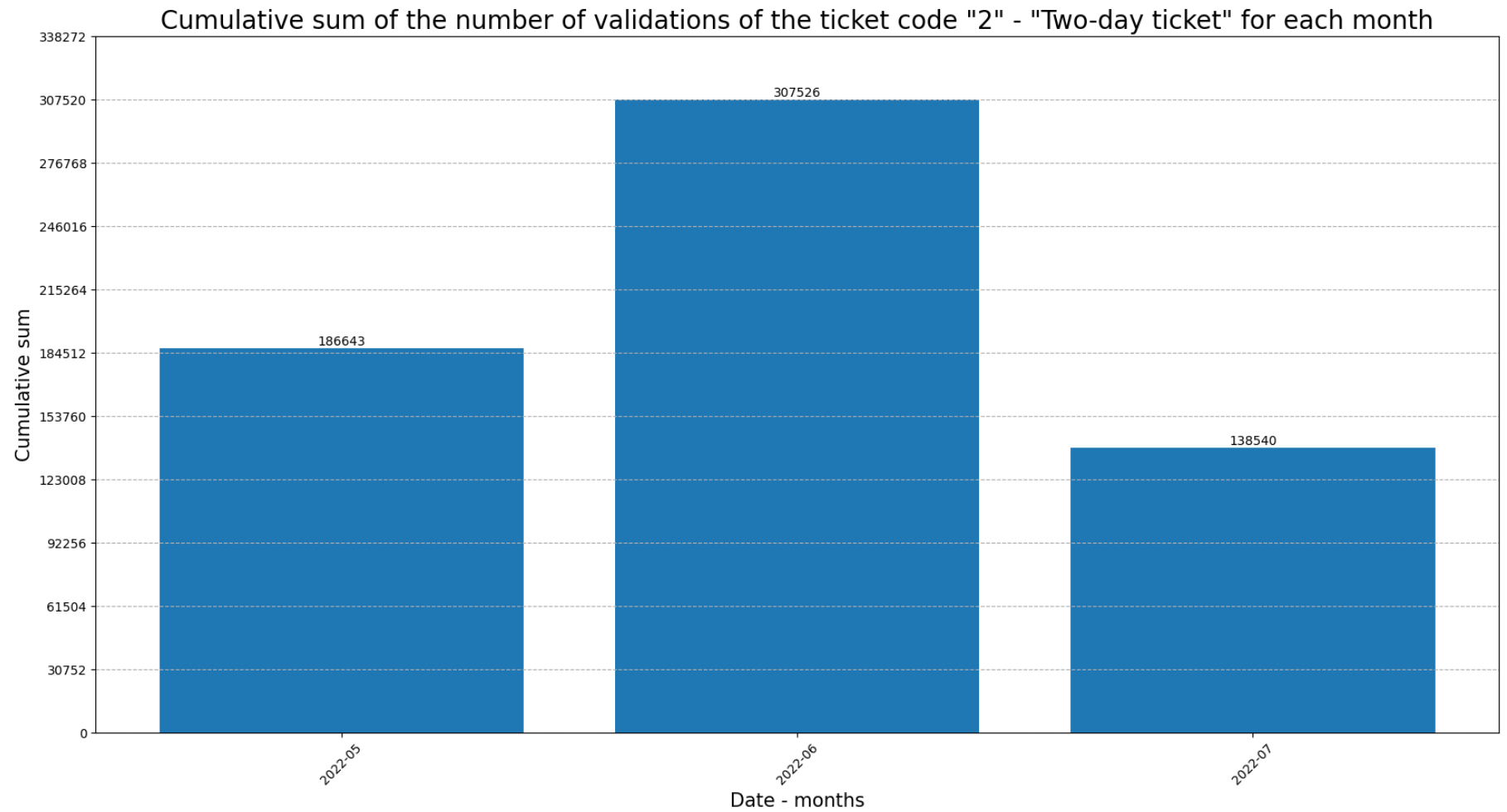


The highest number of validations of the ticket code "1" - "One-day ticket" was the month 2022-06 with 491523 validations.

The lowest number of validations of the ticket code "1" - "One-day ticket" was the month 2022-07 with 249703 validations.



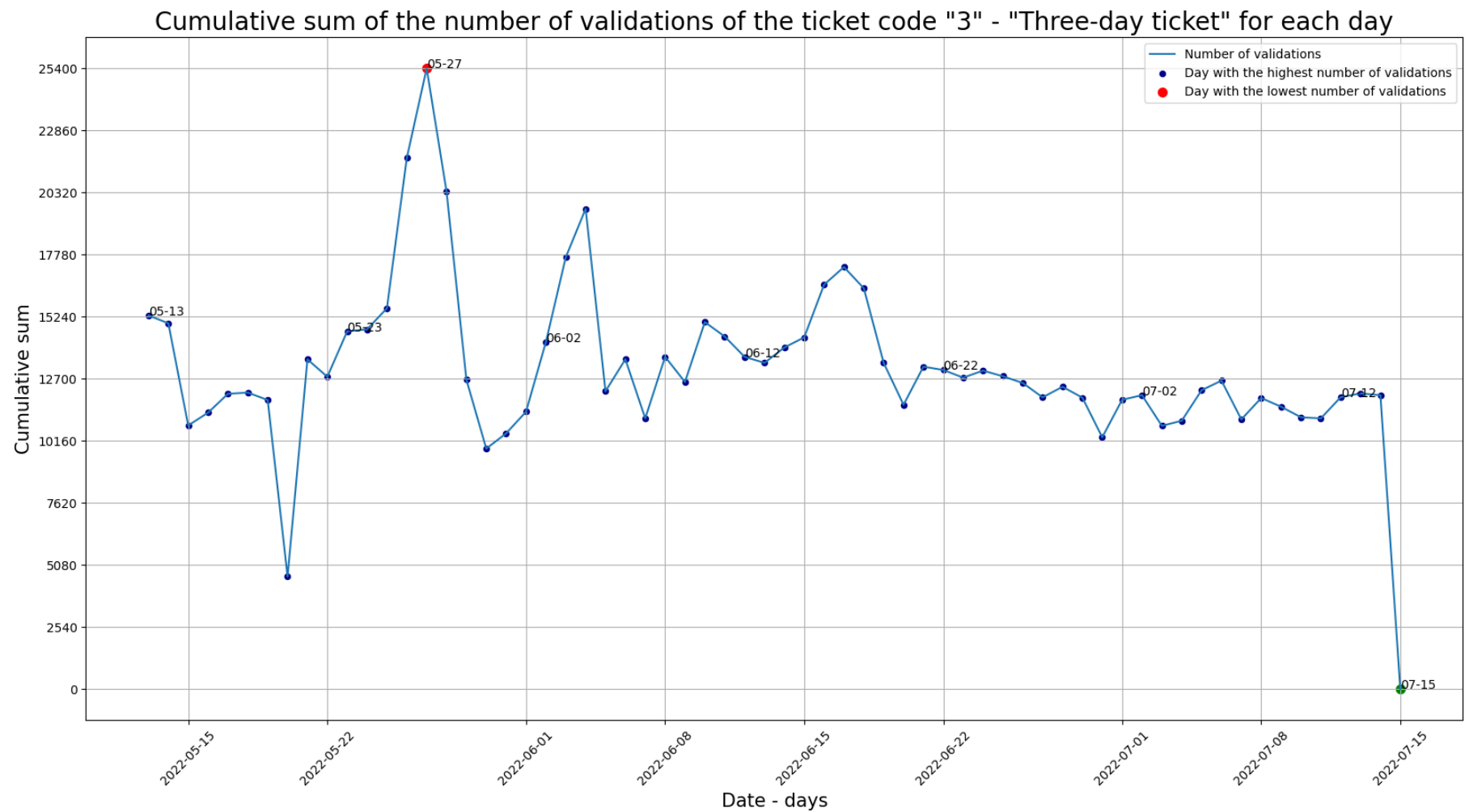
The highest number of validations of the ticket code "2" – "Two-day ticket" was 2022-05-27 with 16049 validations  
The lowest number of validations of the ticket code "2" – "Two-day ticket" was 2022-07-15 with 6 validations



The highest number of validations of the ticket code "2" - "Two-day ticket" was the month 2022-06 with 307526 validations.

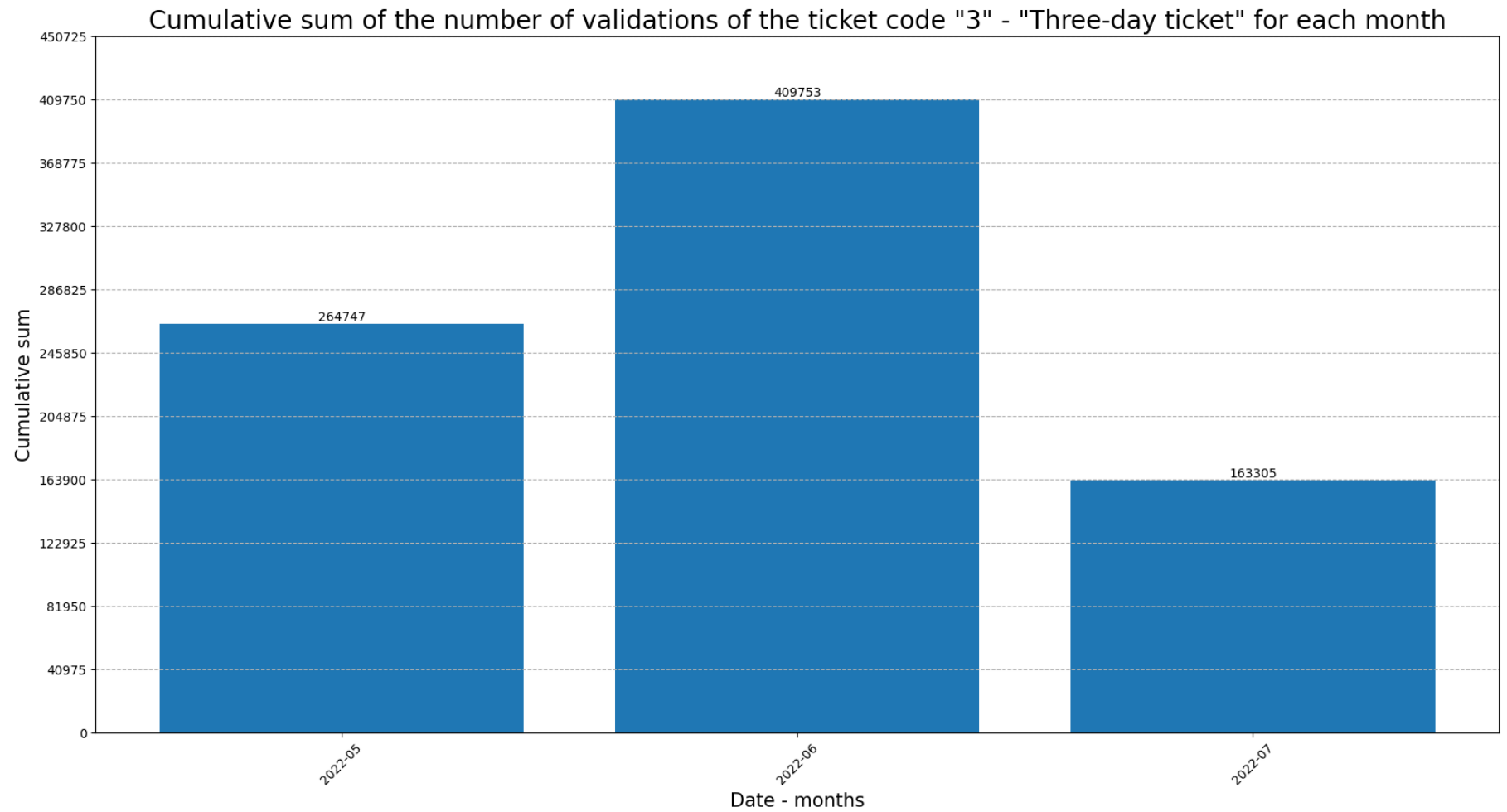
The lowest number of validations of the ticket code "2" - "Two-day ticket" was the month 2022-07 with 138540 validations.





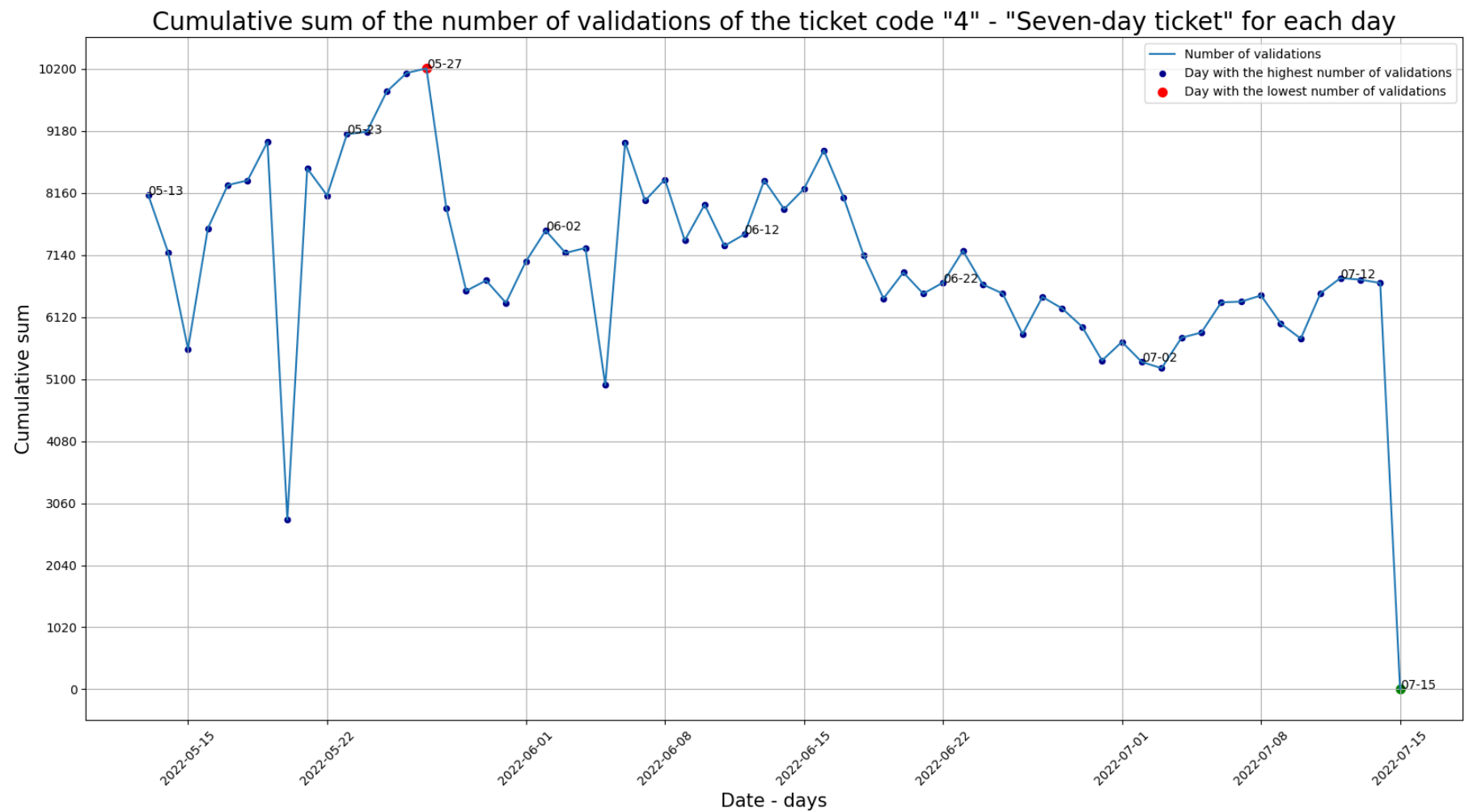
The highest number of validations of the ticket code "3" - "Three-day ticket" was 2022-05-27 with 25400 validations

The lowest number of validations of the ticket code "3" - "Three-day ticket" was 2022-07-15 with 6 validations



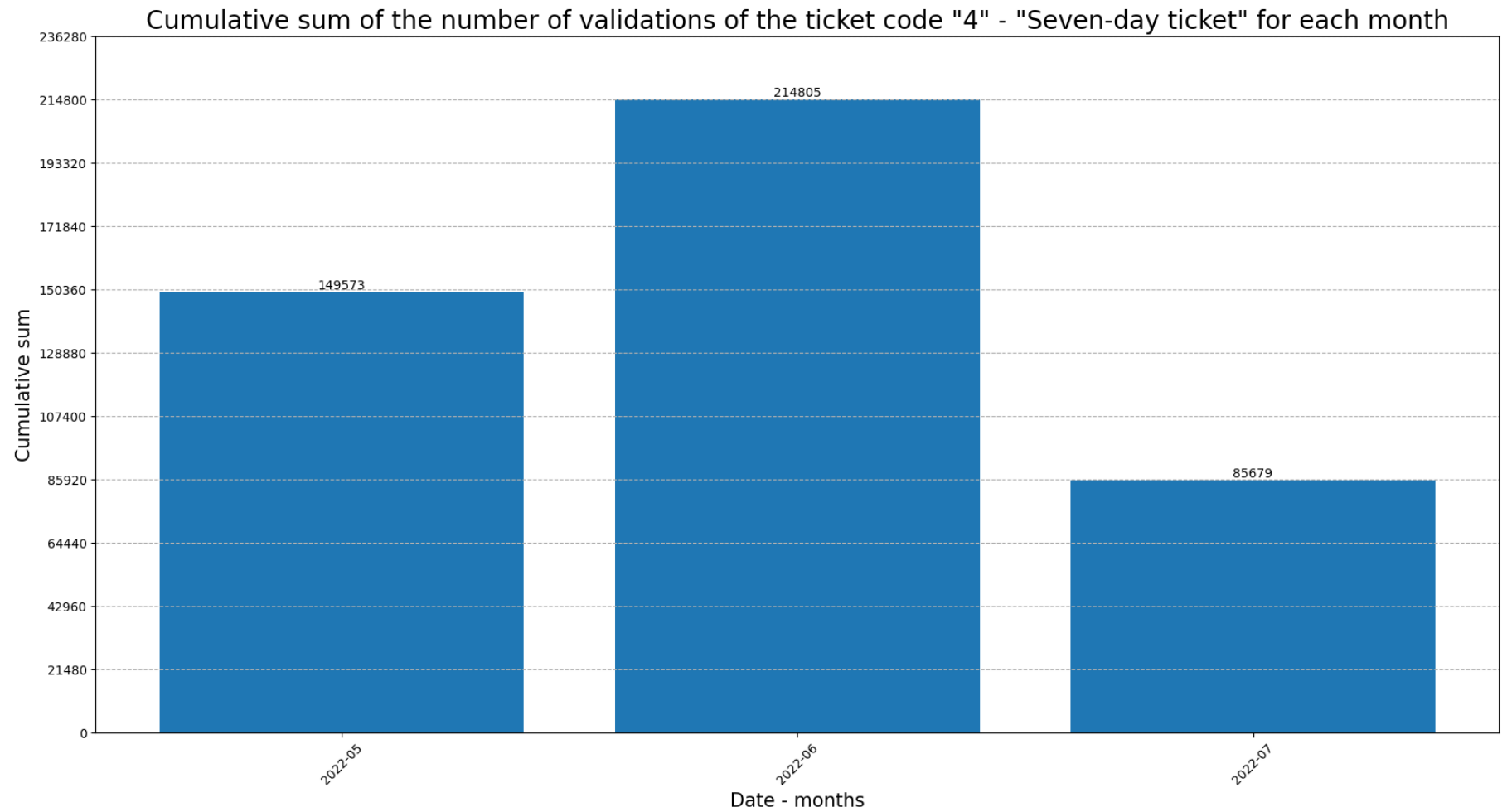
The highest number of validations of the ticket code "3" - "Three-day ticket" was the month 2022-06 with 409753 validations.

The lowest number of validations of the ticket code "3" - "Three-day ticket" was the month 2022-07 with 163305 validations.



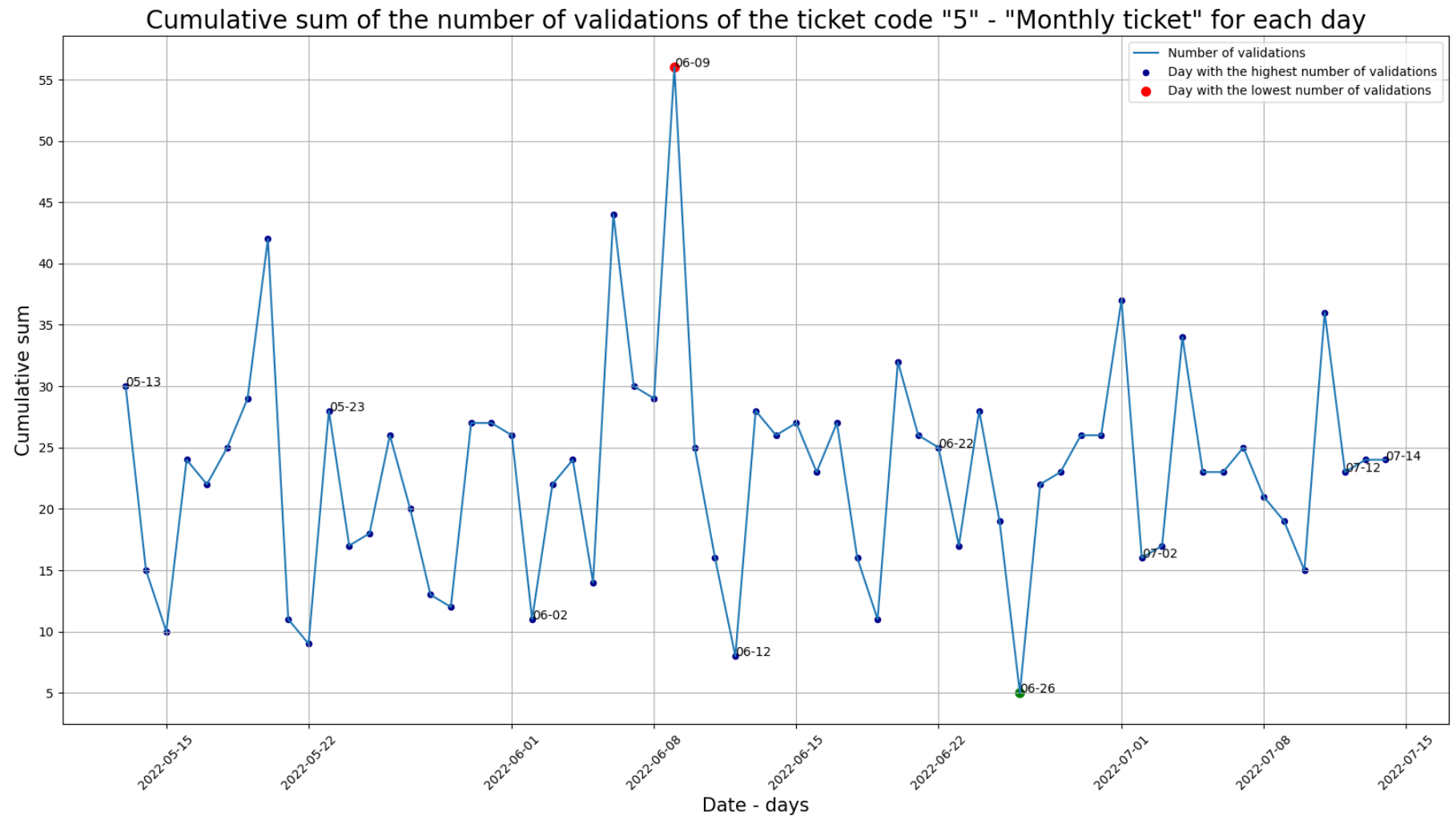
The highest number of validations of the ticket code "4" – "Seven-day ticket" was 2022-05-27 with 10206 validation s

The lowest number of validations of the ticket code "4" – "Seven-day ticket" was 2022-07-15 with 3 validations

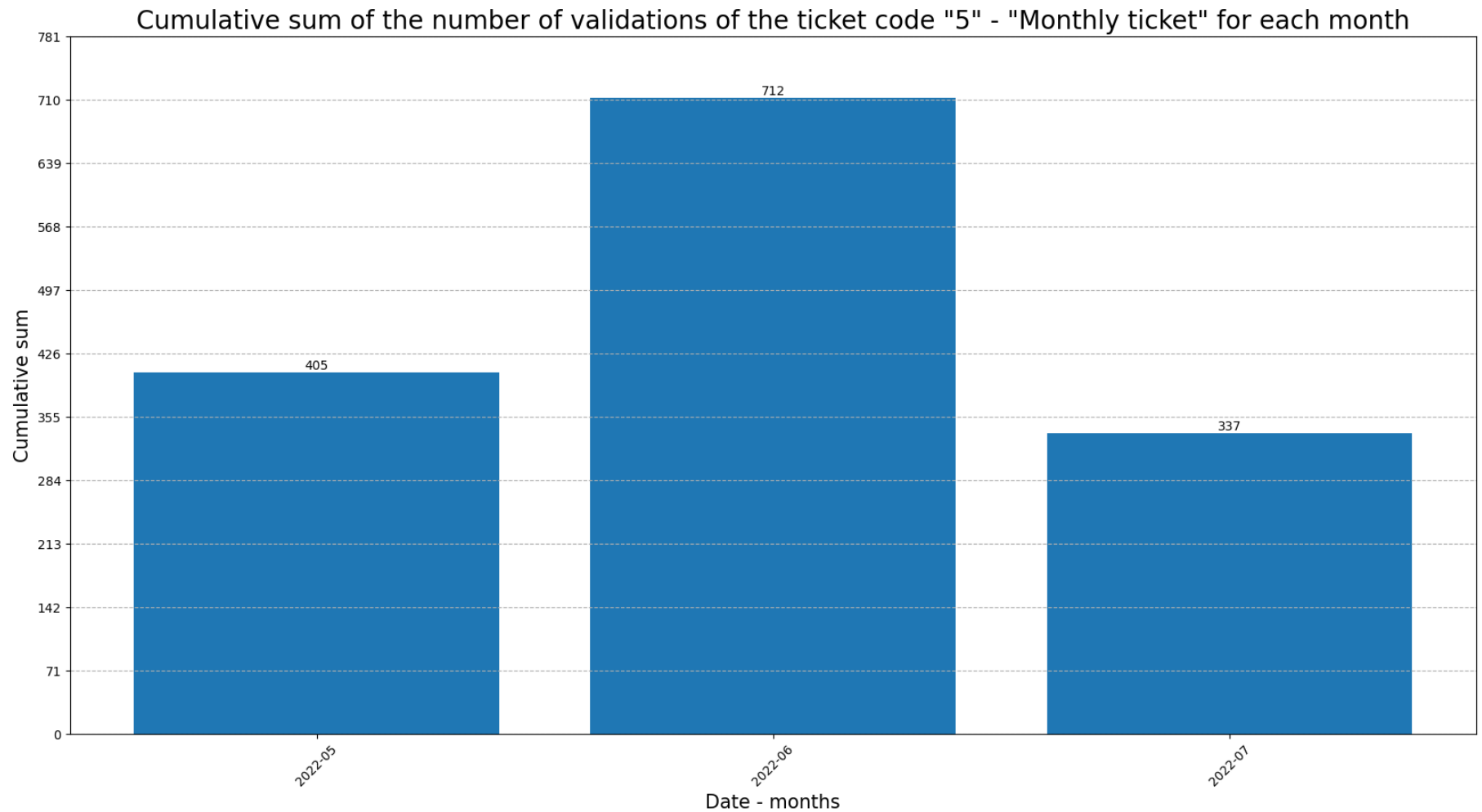


The highest number of validations of the ticket code "4" – "Seven-day ticket" was the month 2022-06 with 214805 validations.

The lowest number of validations of the ticket code "4" – "Seven-day ticket" was the month 2022-07 with 85679 validations.



The highest number of validations of the ticket code "5" - "Monthly ticket" was 2022-06-09 with 56 validations  
The lowest number of validations of the ticket code "5" - "Monthly ticket" was 2022-06-26 with 5 validations



The highest number of validations of the ticket code "5" - "Monthly ticket" was the month 2022-06 with 712 validations.

The lowest number of validations of the ticket code "5" - "Monthly ticket" was the month 2022-07 with 337 validations.

WARNING: There are no validations of the ticket code "5-STUD"

WARNING: There are no validations of the ticket code "5-RET"

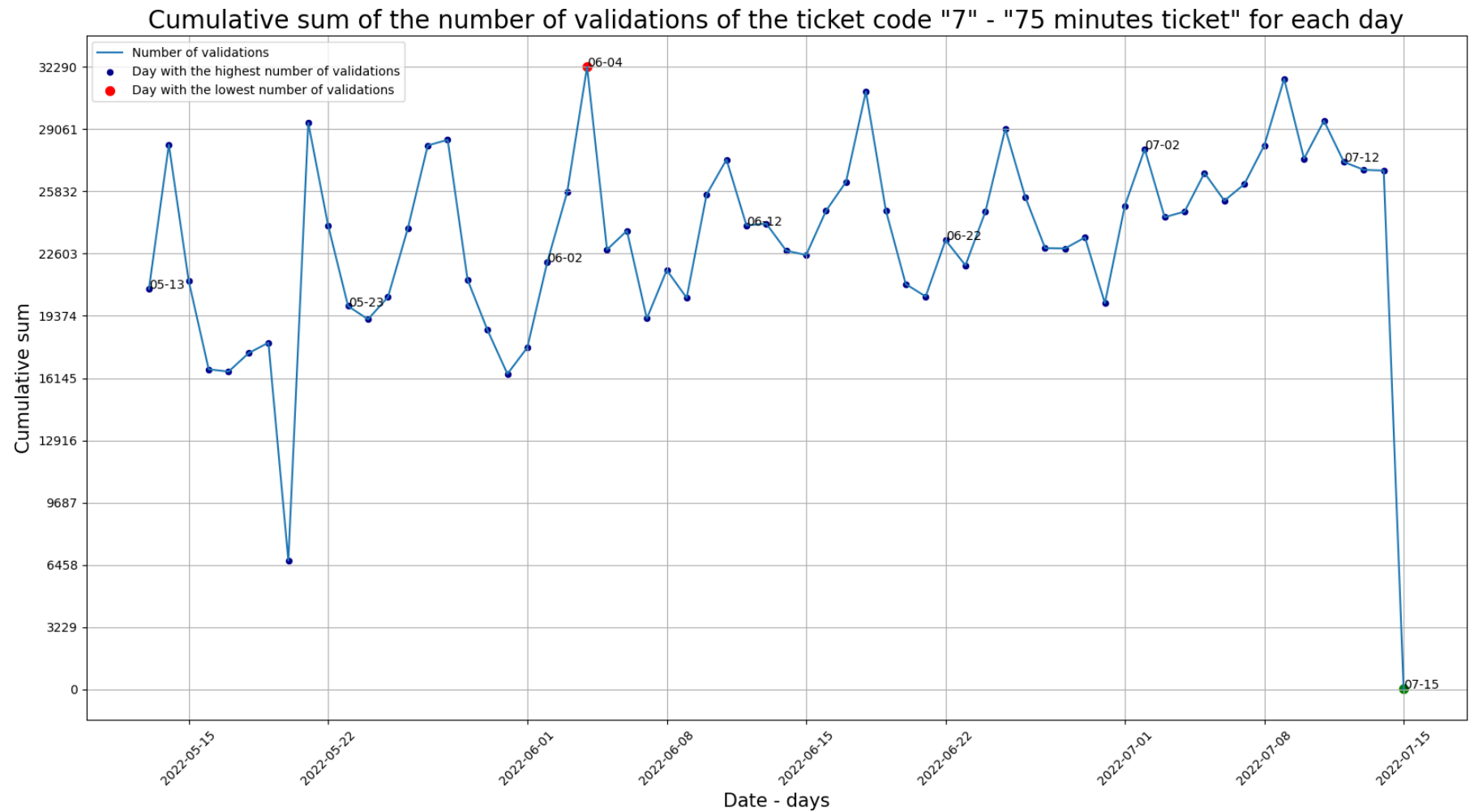
WARNING: There are no validations of the ticket code "5-WKRS"

WARNING: There are no validations of the ticket code "6"

WARNING: There are no validations of the ticket code "6-STUD"

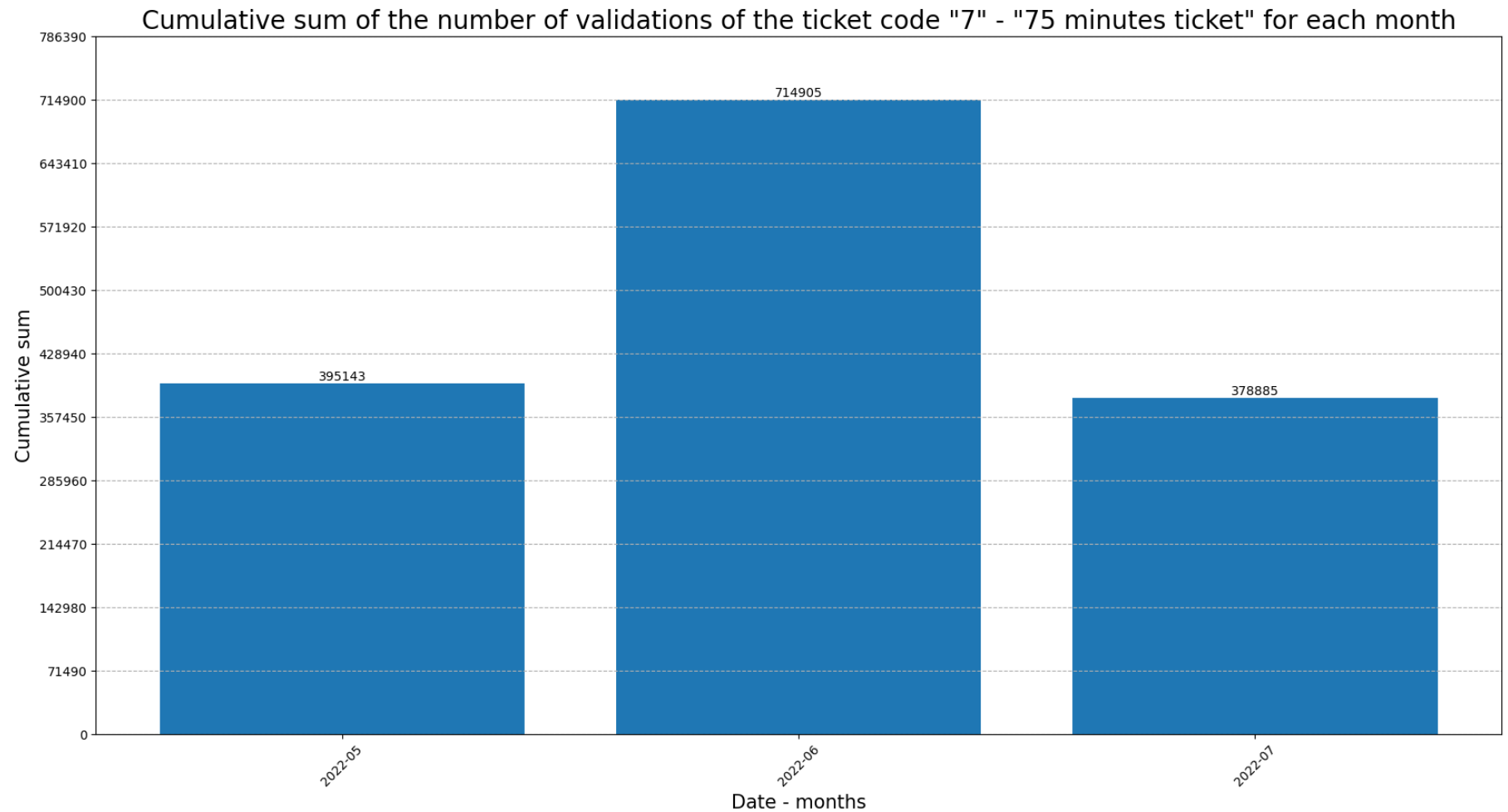
WARNING: There are no validations of the ticket code "6-RET"

WARNING: There are no validations of the ticket code "6-WKRS"



The highest number of validations of the ticket code "7" – "75 minutes ticket" was 2022-06-04 with 32294 validations

The lowest number of validations of the ticket code "7" – "75 minutes ticket" was 2022-07-15 with 19 validations



The highest number of validations of the ticket code "7" - "75 minutes ticket" was the month 2022-06 with 714905 validations.

The lowest number of validations of the ticket code "7" - "75 minutes ticket" was the month 2022-07 with 378885 validations.

WARNING: There are no validations of the ticket code "8"

**Focus on the ticket code \*1\***

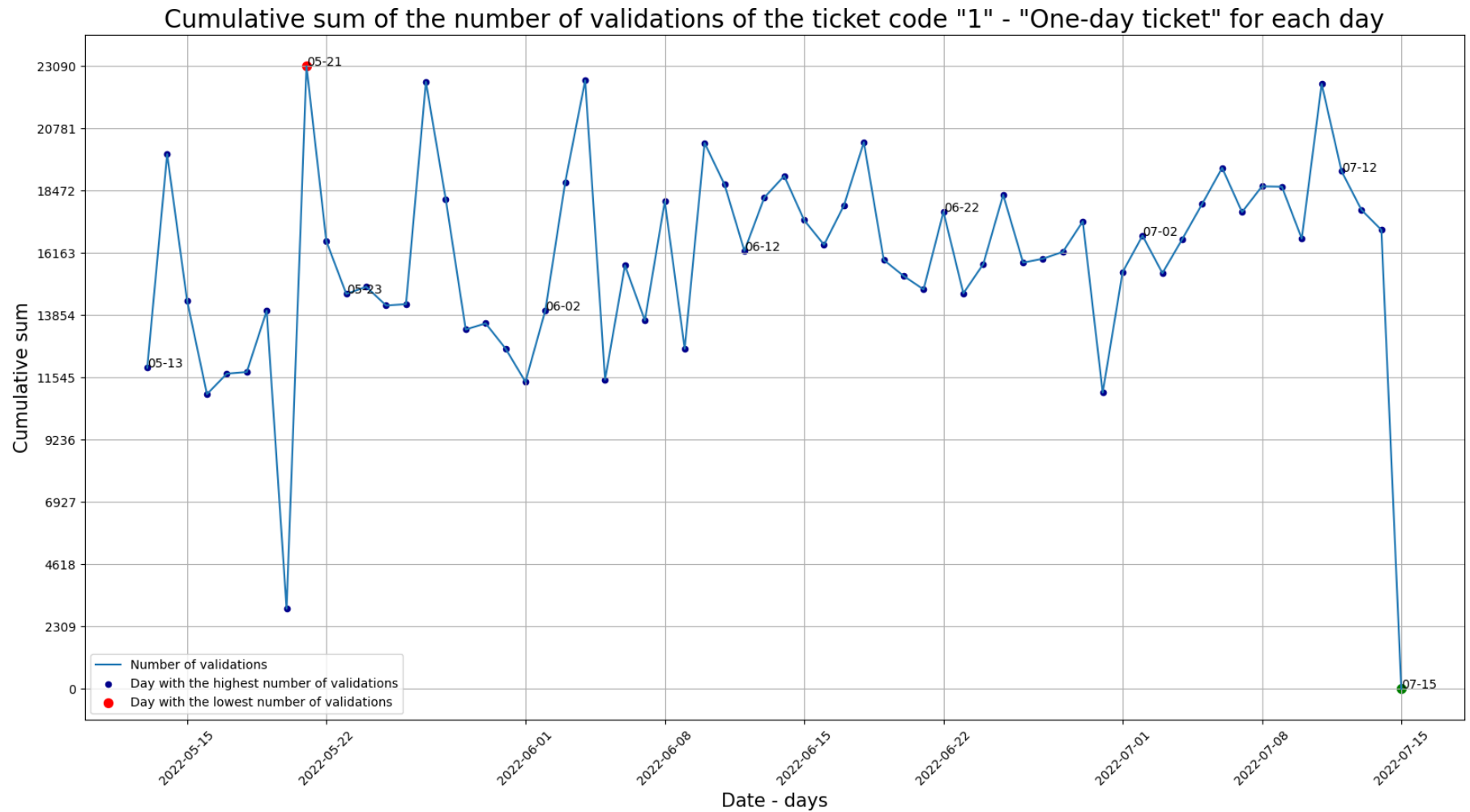
```
In [ ]: # Focus on TICKET_CODE = 1
        target_ticket_code = '1'
```



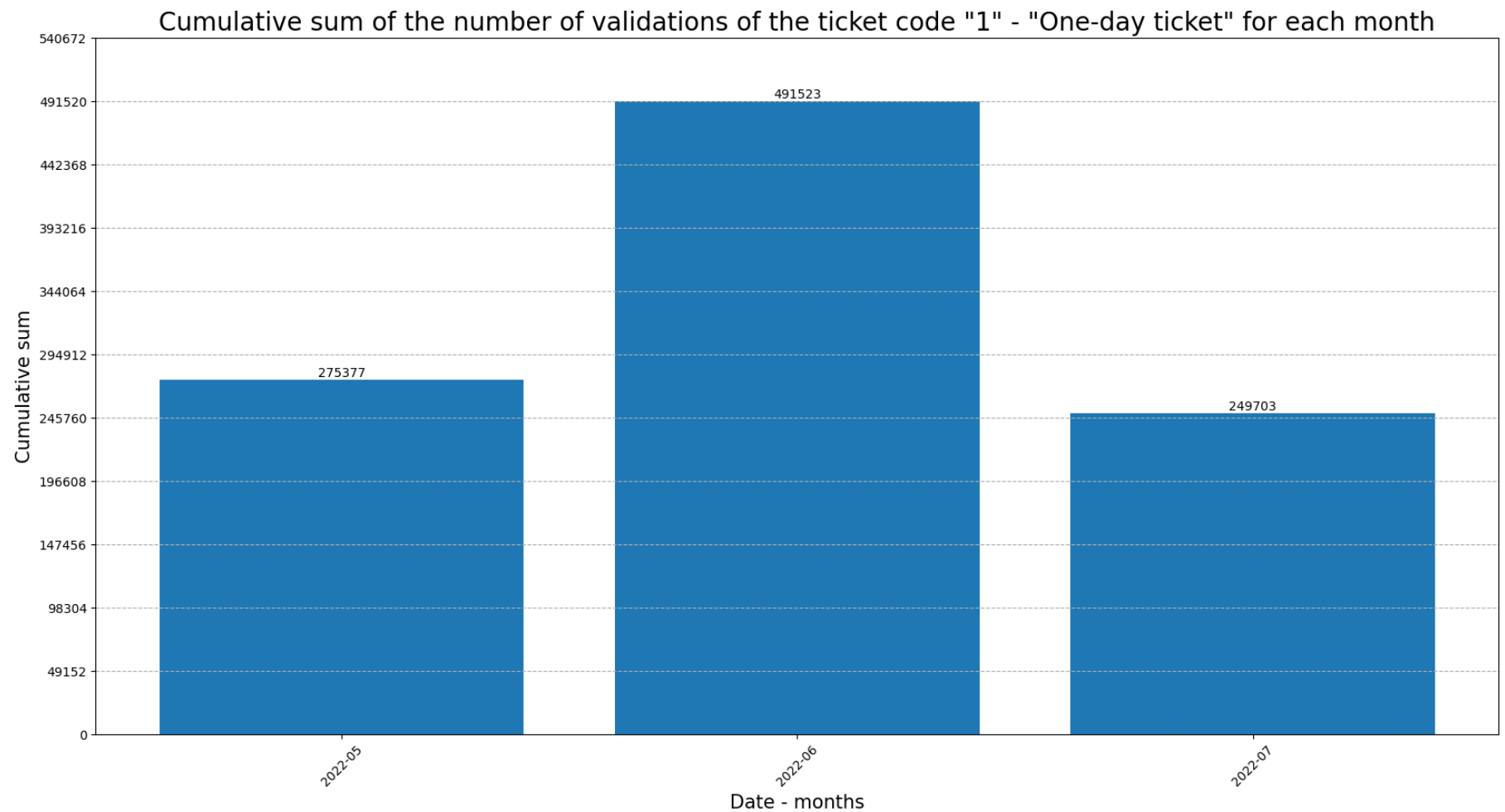
```

df_tc1 = mf.focus_on_ticket_code(df, target_ticket_code)
if df_tc1.shape[0] == 0:
    print('WARNING: There are no validations of the ticket code "{}".format(target_ticket_code))
else:
    number_of_tickets_per_day(df_tc1, target_ticket_code, is_ticket_code=True)
    min_max_number_of_tickets_per_day(df_tc1, target_ticket_code, is_ticket_code=True)
    barplot_number_of_tickets_per_month(df_tc1, target_ticket_code, is_ticket_code=True)

```



The highest number of validations of the ticket code "1" - "One-day ticket" was 2022-05-21 with 23091 validations  
The lowest number of validations of the ticket code "1" - "One-day ticket" was 2022-07-15 with 7 validations



## Focus on SERIALE

The SERIALEs represent the different users.

```
In [ ]: def focus_on_serial(df_s: pd.DataFrame, serial: str):  
        """  
        This function focuses on the specified SERIAL.
```

```

        :param df: the dataframe
        :param serial: the SERIAL
        :return: the dataframe focused on the specified SERIAL
    """
    return df_s[df_s['SERIALE'] == serial]

# Focus on serial over time (all the validations of the SERIAL)
def focus_on_serial_over_time(df_s: pd.DataFrame, serial: str):
    """
        This function focuses on the specified SERIAL over time.
        :param df: the dataframe
        :param serial: the SERIAL of the user
        :return: None
    """
    # Select only the rows of the specified SERIAL
    df_sup = focus_on_serial(df_s, serial)
    # If the dataframe is empty, skip the SERIAL but launch a warning
    if df_sup.shape[0] == 0:
        print('WARNING: There are no validations of the SERIAL "{}".format(serial))
    else:
        # If the dataframe is not empty, focus on the SERIAL
        df_s = focus_on_serial(df_sup, serial)
        # Convert DATA to datetime
        df_s['DATA'] = pd.to_datetime(df_s['DATA'], format='%Y/%m/%d')
        # Set all the values of the date on the first day of the month
        df_s['DATA'] = df_s['DATA'].dt.strftime('%Y-%m-%d')
        # Group the dataframe by date and hour and count the number of validations of the specified ticket code
        df_s = df_s.groupby('DATA').count()['SERIALE'].reset_index()
        # Plot the number of validations per day
        plt.figure(figsize=(20, 10))
        plt.plot(df_s['DATA'], df_s['SERIALE'])
        # Add points to the plot
        plt.scatter(df_s['DATA'], df_s['SERIALE'], color='blue')
        plt.title('Number of validations of the SERIAL "{}" over time'.format(serial))
        plt.xlabel('Date')
        plt.ylabel('Number of validations')
        plt.xticks(rotation=45, ticks=df_s['DATA'])

    # Manage the y-axis
    if df_s['SERIALE'].max() < 10:

```

```

plt.yticks(ticks=range(0, df_s['SERIALE'].max() + 6))
else:
    plt.yticks(ticks=range(0, df_s['SERIALE'].max() + 6, 2))

# Highlight the day with the highest number of validations and the day with the lowest number of validation
max = df_s[df_s['SERIALE'] == df_s['SERIALE'].max()]
plt.scatter(max['DATA'], max['SERIALE'], color='red', s=50)
min = df_s[df_s['SERIALE'] == df_s['SERIALE'].min()]
plt.scatter(min['DATA'], min['SERIALE'], color='green', s=50)

# Add the legend
plt.legend(['Number of validations', 'Day with the highest number of validations', 'Day with the lowest num

# Add the grid
plt.grid()

plt.show()

```

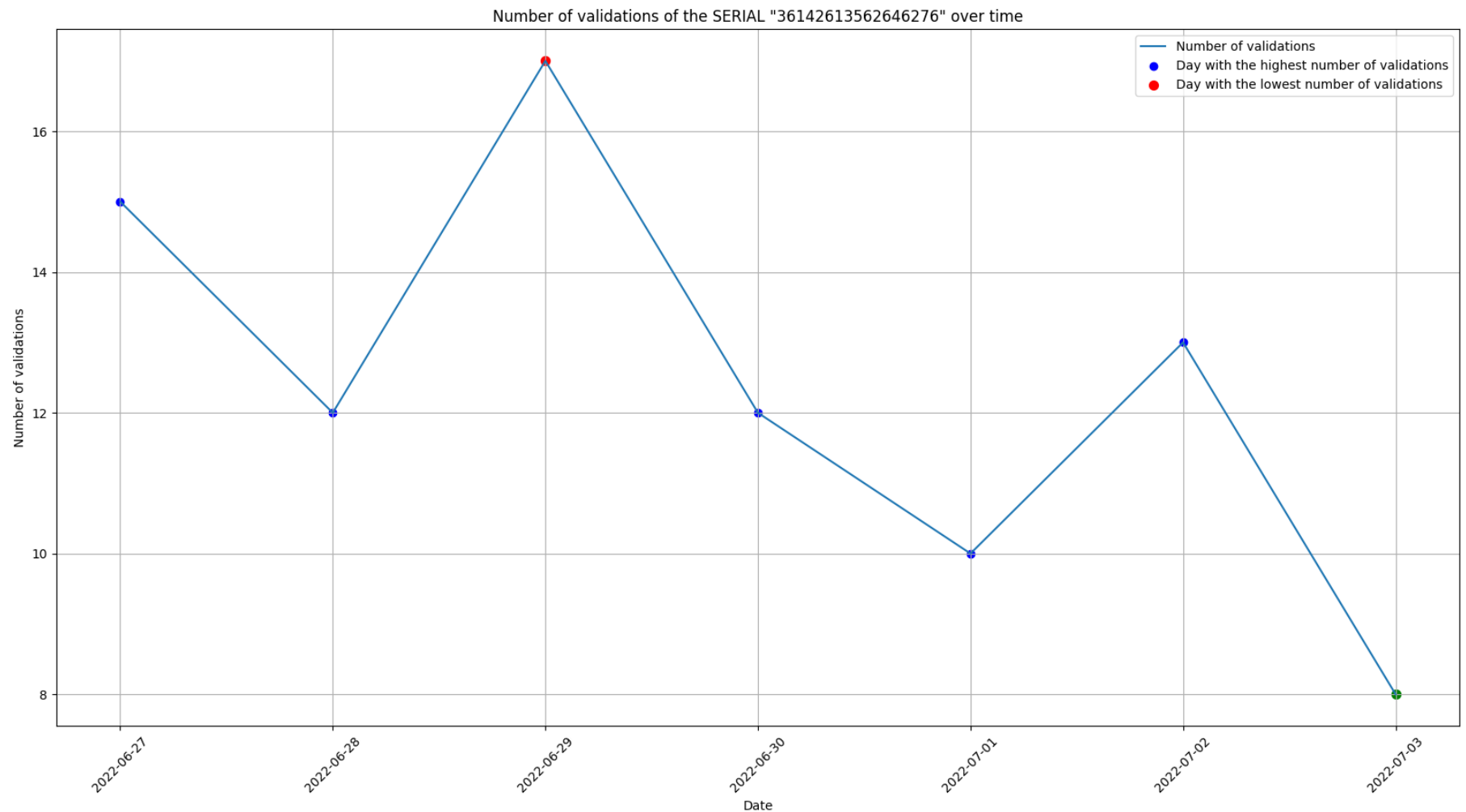
## Focus on a specified SERIALE

```

In [ ]: # Select the serial of the user with the highest number of validations
# Ask in input the index of the serial to focus on
index = int(input('Insert the index of the serial to focus on: '))

if index >= len(df['SERIALE'].value_counts()):
    print('WARNING: The index is not valid. The index must be less than {}'.format(len(df['SERIALE'].value_counts()))
else:
    serial = df['SERIALE'].value_counts().index[index]
    df_s = focus_on_serial(df, serial)
    focus_on_serial_over_time(df_s, serial)

```



## Focus on the Carnival of Venice period

```
In [ ]: def focus_on_carnival_period(df_c: pd.DataFrame):  
        .....
```

This function focuses on the Carnival of Venice period (4th of February 2023 – 21st of February 2023), and compares the number of validations of the different ticket codes with the number of validations of the ticket codes in the previous and next period (18 days before the Carnival of Venice and 18 days after the Carnival of Venice).

```

:param df: the dataframe
:return: None
"""

# Focus on the period 4th of February 2023 – 21st of February 2023 (the period of the Carnival of Venice)
df_carnival = mf.focus_on_period(df, '2023-02-04', '2023-02-21')

# How many days are there in the Carnival of Venice?
print('There are {} days in the Carnival of Venice'.format(df_carnival['DATA'].nunique()))

# Search if there is an increase in the number of validations of the ticket code 1 during the Carnival of Venice
# Create a dataframe of 18 days before the Carnival of Venice and 18 days after the Carnival of Venice
df_no_carnival_prev = mf.focus_on_period(df, '2023-01-17', '2023-02-03')
df_no_carnival_next = mf.focus_on_period(df, '2023-02-22', '2023-03-12')

# For each ticket code, find the average number of validations and compare it with the average number of validations
# Use the ticket code contained in the dictionary
ticket_codes = mf.open_dict_ticket_codes()

for ticket_code in ticket_codes:

    # Get the description of the ticket code
    description = ticket_codes[ticket_code]

    # Focus on the ticket code
    df_tc_prev = mf.focus_on_ticket_code(df_no_carnival_prev, ticket_code)
    df_tc_next = mf.focus_on_ticket_code(df_no_carnival_next, ticket_code)
    # If the dataframe is empty, skip the ticket code but launch a warning
    if df_tc_prev.shape[0] == 0 or df_tc_next.shape[0] == 0 and ticket_code != '8':
        print('WARNING: There are no validations of the ticket code "{}"'.format(ticket_code))
    else:
        # If the dataframe is not empty, focus on the ticket code
        df_tc_prev = mf.focus_on_ticket_code(df_no_carnival_prev, ticket_code)
        df_tc_next = mf.focus_on_ticket_code(df_no_carnival_next, ticket_code)
        # Convert DATA to datetime
        df_tc_prev['DATA'] = pd.to_datetime(df_tc_prev['DATA'], format='%Y/%m/%d')
        df_tc_next['DATA'] = pd.to_datetime(df_tc_next['DATA'], format='%Y/%m/%d')
        # Set all the values of DATA to the same format
        df_tc_prev['DATA'] = df_tc_prev['DATA'].dt.strftime('%Y-%m-%d')
        df_tc_next['DATA'] = df_tc_next['DATA'].dt.strftime('%Y-%m-%d')
        # Group the dataframe by date and hour and count the number of validations of the specified ticket code

```

```

df_tc_prev = df_tc_prev.groupby('DATA').count()['TICKET_CODE'].reset_index()
df_tc_next = df_tc_next.groupby('DATA').count()['TICKET_CODE'].reset_index()
# Find the average number of validations of the ticket code
avg_no_carnival_prev = df_tc_prev['TICKET_CODE'].mean()
avg_no_carnival_next = df_tc_next['TICKET_CODE'].mean()

# Focus on the period of the Carnival of Venice
df_tc_carnival = mf.focus_on_ticket_code(df_carnival, ticket_code)
if df_tc_carnival.shape[0] == 0:
    print('WARNING: There are no validations of the ticket code "{}" during the Carnival of Venice'.format(ticket_code))
else:
    # Convert DATA to datetime
    df_tc_carnival['DATA'] = pd.to_datetime(df_tc_carnival['DATA'], format='%Y/%m/%d')
    # Set all the values of DATA to the same format
    df_tc_carnival['DATA'] = df_tc_carnival['DATA'].dt.strftime('%Y-%m-%d')
    # Group the dataframe by date and hour and count the number of validations of the specified ticket
    df_tc_carnival = df_tc_carnival.groupby('DATA').count()['TICKET_CODE'].reset_index()
    # Find the average number of validations of the ticket code
    avg_carnival = df_tc_carnival['TICKET_CODE'].mean()

# Compare the average number of validations of the ticket code during the Carnival of Venice with the previous and next periods
if avg_carnival > avg_no_carnival_prev and avg_carnival > avg_no_carnival_next:
    print('The average number of validations of the ticket code "{} - {}" during the Carnival of Venice is higher than the previous and next periods'.format(ticket_code, ticket_code))
elif avg_carnival > avg_no_carnival_prev and avg_carnival < avg_no_carnival_next:
    print('The average number of validations of the ticket code "{} - {}" during the Carnival of Venice is higher than the previous period but lower than the next period'.format(ticket_code, ticket_code))
elif avg_carnival < avg_no_carnival_prev and avg_carnival > avg_no_carnival_next:
    print('The average number of validations of the ticket code "{} - {}" during the Carnival of Venice is lower than the previous period but higher than the next period'.format(ticket_code, ticket_code))
elif avg_carnival < avg_no_carnival_prev and avg_carnival < avg_no_carnival_next:
    print('The average number of validations of the ticket code "{} - {}" during the Carnival of Venice is lower than both the previous and next periods'.format(ticket_code, ticket_code))
else:
    print('The average number of validations of the ticket code "{} - {}" during the Carnival of Venice is equal to the previous and next periods'.format(ticket_code, ticket_code))

# Represent these information in a plot
plt.figure(figsize=(20, 10))
plt.plot(df_tc_prev['DATA'], df_tc_prev['TICKET_CODE'], label='Previous period', color='blue')
plt.plot(df_tc_carnival['DATA'], df_tc_carnival['TICKET_CODE'], label='Carnival of Venice', color='red')
plt.plot(df_tc_next['DATA'], df_tc_next['TICKET_CODE'], label='Next period', color='green')

```

```

plt.title('Average number of validations of the ticket code {} - "{}" during the Carnival of Venice')
plt.xlabel('Date')
plt.ylabel('Number of validations')

# Highlight the period of the Carnival of Venice in the plot (2023-02-04, '2023-02-21')
plt.axvspan('2023-02-04', '2023-02-22', alpha=0.5, color='lightgrey')

# Color the x-axis labels according to the period (previous: blue, carnival: orange, next: green)
for label in plt.gca().get_xticklabels():
    if label.get_text() < '2023-02-04':
        label.set_color('blue')
    elif label.get_text() > '2023-02-21':
        label.set_color('green')
    else:
        label.set_color('orange')

# If label is on weekend, set it to bold and underline
if datetime.strptime(label.get_text(), '%Y-%m-%d').weekday() >= 5:
    label.set_weight('bold')
    label.set_style('italic')
# Add vertical lines to separate the weekends
# If weekend is not already present in the legend, add it
if 'Weekend' not in plt.gca().get_legend_handles_labels()[1]:
    plt.axvline(x=label.get_text(), color='black', linestyle='dotted', linewidth=2, label='Weekend')
else:
    plt.axvline(x=label.get_text(), color='black', linestyle='dotted', linewidth=2)

# For each period, add a line that represents the average number of validations of the ticket code
# Draw these lines only in the corresponding period
plt.axhline(y=avg_no_carnival_prev, color='blue', linestyle='--', xmin=0.05, xmax=0.341, label='Avg. no. prev.')
plt.axhline(y=avg_carnival, color='orange', linestyle='--', xmin=0.35, xmax=0.641, label='Avg. num. carnival')
plt.axhline(y=avg_no_carnival_next, color='green', linestyle='--', xmin=0.65, xmax=0.96, label='Avg. no. next.')

# For each period, add the points of the maximum and minimum number of validations of the ticket code
# Draw these points only in the corresponding period
plt.plot(df_tc_prev['DATA'].iloc[df_tc_prev['TICKET_CODE'].idxmax()], df_tc_prev['TICKET_CODE'].max)
plt.plot(df_tc_prev['DATA'].iloc[df_tc_prev['TICKET_CODE'].idxmin()], df_tc_prev['TICKET_CODE'].min)
plt.plot(df_tc_carnival['DATA'].iloc[df_tc_carnival['TICKET_CODE'].idxmax()], df_tc_carnival['TICKET_CODE'].max)
plt.plot(df_tc_carnival['DATA'].iloc[df_tc_carnival['TICKET_CODE'].idxmin()], df_tc_carnival['TICKET_CODE'].min)

```



```

plt.plot(df_tc_next['DATA'].iloc[df_tc_next['TICKET_CODE'].idxmax()], df_tc_next['TICKET_CODE'].max)
plt.plot(df_tc_next['DATA'].iloc[df_tc_next['TICKET_CODE'].idxmin()], df_tc_next['TICKET_CODE'].min)

# Add a grid
plt.grid()

# Add a text for each period (previous, during, next) under the x-axis
plt.text(0.2, -0.2, 'Period before the Carnival', transform=plt.gca().transAxes, color='blue', horizontalalignment='left')
plt.text(0.5, -0.2, 'Carnival of Venice', transform=plt.gca().transAxes, color='orange', horizontalalignment='center')
plt.text(0.8, -0.2, 'Period after the Carnival', transform=plt.gca().transAxes, color='green', horizontalalignment='right')

# Make the plot more readable
plt.xticks(rotation=90)
plt.tight_layout()

# Add a legend
plt.legend()

plt.show()

```

```

In [ ]: # Focus on the period of the Carnival of Venice (2023-02-04, '2023-02-21') and
# comparison with the previous and the next period
# Execute this cell only if the variable file_name is 'esportazioneCompleta'
if file_name == 'esportazioneCompleta':
    focus_on_carnival_period(df)
else:
    print('The analysis regarding the Carnival of Venice is available only on the dataset "esportazioneCompleta".')

```

The analysis regarding the Carnival of Venice is available only on the dataset "esportazioneCompleta".