LABcal: Automated Calendar Analysis for the Education Lab »ExpLAB« at the University Library Mannheim

Final Project in the Data Librarian Certificate Course 2023/24

Thomas Schmidt
University of Mannheim

11 October 2024

Table of contents

Introduction

The ExpLAB[1] at the Mannheim University Library is a space for collaborative and creative work that is available to all members of the university. With flexible furniture, a range of VR glasses, an eye-tracking station and media technology for hybrid lecture scenarios, it has been used in a variety of ways since its opening in 2021 and can be booked for events such as seminars, lectures and workshops. With over 340 events in 2023, it is a very popular multi-purpose space in the library.

[1] https://www.bib.uni-mannheim.de/standorte/explab-schloss-schneckenhof/. Accessed 11 October 2024.

Reservations are managed via a calendar that is hosted in the University Library's NextCloud[2] and can be exported as an `.ics` file (RFC 5545)[3]. For the library's internal statistics, there has long been a desire to automatically analyse these calendar reservations as easily as possible. The focus here is on the total number of participants in events, but also on the question of which university stakeholders (faculties, centers and institutions, student initiatives, etc.) use the ExpLAB, which media technology is used most frequently and where there is room for improvement.

Project outline

For this purpose, the web application "LABcal"[4] based on Python and Streamlit was developed. Streamlit[5] is a framework for creating web applications using Python scripts. A web application was considered as it provides a user-friendly way of interacting with the automated calendar evaluation within a web browser.

The implemented workflow was designed in the following way: The `.ics` calendar is exported and downloaded by a user from NextCloud. The web application is launched and provides a file upload. The `.ics` file is then uploaded, automatically parsed and converted into a Pandas[6] `DataFrame`, which, in conjunction with the Python packages Matplotlib[7] and Seaborn[8], is very well suited for producing high quality data visualisations. The goal of the application was to create a variety of plots that would provide insight into the use of the ExpLAB.

Data structure of calendar entries

Pandas was also considered because the reservations were available in a semi-structured format in the calendar text field `DESCRIPTION`[9], which could be relatively easily converted into a Python `dict` that could then be used as input data for the `DataFrame`.

An example entry in the field `DESCRIPTION` can illustrate the structure of the reservation data:

> Kategorie Workshop: Yoga und Meditation
> Veranstalter Uni: Institut für Sport
> Teilnehmer: bis 10
> Technik: keine, Yoga-Equipment
> Kontakt: mail@example.com
> Catering: nein
> Anmerkung: nein

Various keys, such as `Kategorie Workshop` or `Veranstalter Uni`, are followed by descriptive values like `Yoga und Meditation` or `Institut für Sport`. The keys follow a controlled vocabulary. For example, only `Uni`, `UB`, `Studis`, and `Extern` are permissible as `Veranstalter`. Apart from minor mistakes (e.g., typos), it was assumed that this pre-structured calendar data could be automatically processed to generate a set of data visualisations.

[2] https://github.com/nextcloud. Accessed 11 October 2024.
[3] The NextCloud documentation states, that "The Nextcloud Calendar application only supports iCalendar-compatible .ics-files, defined in RFC 5545." https://docs.nextcloud.com/server/28/user_manual/en/groupware/calendar.html#edit-export-or-delete-a-calendar. Accessed 11 October 2024.
[4] Code repository: https://doi.org/10.5281/zenodo.13919649. Accessed 11 October 2024.
[5] https://streamlit.io/. Accessed 11 October 2024.
[6] https://pandas.pydata.org/. Accessed 11 October 2024.
[7] https://matplotlib.org/. Accessed 11 October 2024.
[8] https://seaborn.pydata.org/. Accessed 11 October 2024.
[9] Cf. RFC 5545 specification at section 3.8.1.5 for more detail: https://datatracker.ietf.org/doc/html/rfc5545#section-3.8.1.5. Accessed 11 October 2024.

Implementation

Work packages

The main work packages included parsing the calendar, the subsequent data cleaning and data visualisation as well as the implementation of the web application.

Data parsing

> *Refer to appendix section "Step 1: Data parsing" for a detailed walkthrough.*

The calendar is parsed using the open source Python package `ics.py`[10]. The `ics.Event` objects serialised as `str` are converted into a Python `dict`, where the typical `.ics` data structure is split into key-value pairs. The resulting `dict` is then transferred to a Pandas `DataFrame`.

Data cleaning

> *Refer to appendix section "Step 2: Data cleaning" for a detailed walkthrough.*

The `DataFrame` is then processed and cleaned. Two functions are used to clean the free text entries in the `DESCRIPTION` field of the calendar, which contain the most important information for the data visualisation: `process_cal.clean_calendar_dataframe()` and `process_cal.process_description()`.

The `clean_calendar_dataframe()` function incrementally cleans various aspects of the `DataFrame`, including removing columns with less than 5% non-null values, rearranging the column layout for easier data access, and, importantly, data cleaning for the `event_desc_list` column, where the information from the calendar's `DESCRIPTION` field is stored in a `list[str]` structure.

Using the `process_description()` function, these list elements are further processed. The function iterates through all list elements and matches certain `str` sequences (e.g., `Kategorie`, `Veranstalter`, and `Teilnehmer`, etc.). When a specific string (e.g., `Kategorie`) is found, a separate function (in this case, `clean_event_category()`) takes over the processing, which consists of a `dict` mapping.

This `dict`, which is used in all mapping functions, has the following structure:

```
patterns = {
    'key1': list[str],
    'key2': list[str],
    ...
}
```

The `dict.keys()` are cleaned `str`, which are returned by the function only if a `str` in the list stored in `dict.values()` matches the `str` passed to the function. In this way, differently written strings, such as `UB Intern` and `Interne Veranstaltung`, are mapped to the same key, creating a consistent matching. In the area of organisers, for example, a uniform `key` can be returned for different terms to create tighter semantic categories for the semi-structured free text entries:

```
'Philosophische Fakultät': ['philosophische', 'phil fak', 'philfak', 'anglistik', 'germanistik'],
```

[10]https://github.com/ics-py/ics-py. Accessed 11 October 2024.

Data visualisation

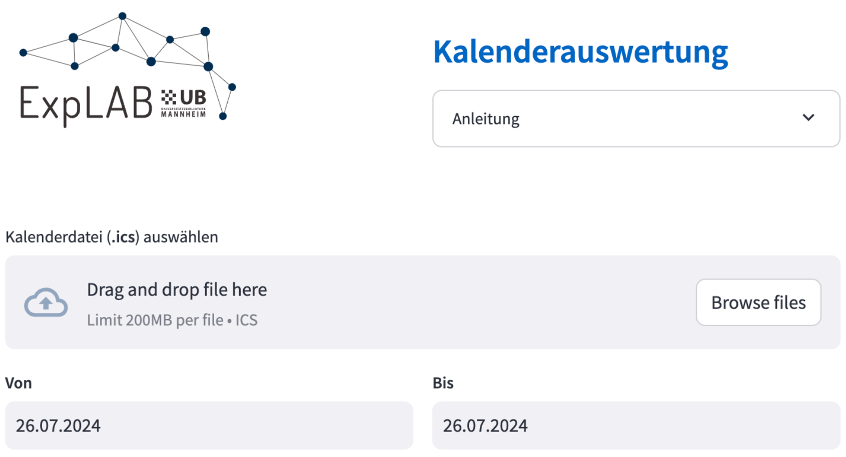
Refer to appendix section "Step 3: Data visualisation" for a detailed walkthrough.

8 different plots can be created using the main plotting function `plot_cal.plot_calendar()`. Setting the `list_func` parameter to `True` will list all available plots:

```
("Available plotting functions: ['organiser', 'organiser_detail', "
 "'event_category', 'event_category_by_organiser', 'equip_details', "
 "'equip_overall', 'participant_stats', 'participants_per_month']")
```

The data visualisations are generated using the Python packages Matplotlib and Seaborn and focus mostly on plotting amounts of categorical (`organiser`, `organiser_detail`, `event_category`, `event_category_by_organiser`, `equip_details`, `equip_overall`) and metric (`participant_stats`, `participants_per_month`) data. Following chapter 4 "Color scales" and chapter 6 "Visualizing amounts" of Claus O. Wilkes' "Fundamentals of Data Visualization"[11], bar charts with 3 different colour maps are used for plotting: *Okabe ito*[12], *ColorBrew qualitative 4 class paired*[13] and *ColorBrew 6 class set2*[14].

Streamlit web application



Kalenderauswertung

Anleitung

Kalenderdatei (.ics) auswählen

Drag and drop file here

Limit 200MB per file • ICS

Browse files

Von

26.07.2024

Bis

26.07.2024

Figure 1: Detail of the Streamlit web application as viewed in a web browser

The Streamlit web application combines the functions of the `labcal` package while providing a simple web interface. The user can provide the `.ics` file via an upload button and can specify the time period for the visualisations via the date input fields `st.session_state.date_start` ("Von") and `st.session_state.date_end` ("Bis").

[11]Wilke, Claus O. (2018). *Fundamentals of Data Visualization.* O'Reilly. Accessed 11 October 2024. https://clauswilke.com/dataviz/.

[12]https://easystats.github.io/see/reference/scale_color_okabeito.html. Accessed 11 October 2024.

[13]https://colorbrewer2.org/?type=qualitative&scheme=Paired&n=4. Accessed 11 October 2024.

[14]https://colorbrewer2.org/?type=qualitative&scheme=Set2&n=6. Accessed 11 October 2024.

The app handles different `streamlit.session_states` (depending on whether a user has clicked the upload button) and displays the resulting plots using Streamlit-specific `tab` elements. In a final step the user can download a `.zip` file containing all plots as `.png` images and a Microsoft Excel `.xlsx` file containing all data used for the plots.

Discussion

LABcal enables the user-friendly evaluation of `.ics` calendars that provide semi-structured event data in the `DESCRIPTION` field. For this reason, LABcal is not a generic solution that can be applied to similar calendars, as the free text field mentioned is maintained with an individual and error-prone structure that cannot be easily applied to other use cases.

In addition, the analysis of the calendar is time consuming as extensive mapping between terms that are written differently but have similar or synonymous meaning needs to be done (see Data Cleaning section). If terms are used during data entry that do not yet exist in the `dict` used for the mapping, no matching can take place, which in turn distorts the results of the data visualisation. The extension of the mapping is therefore complex and must also take into account various mistakes (e.g. incorrect spellings).

In order to make the analysis more robust and generic, the data input would have to be improved. This could be done by using data fields to which a controlled vocabulary can be assigned, for example to standardise information on event categories or organiser details. Although the data entry is already structured, it needs to be reviewed more closely to ensure a robust analysis.

Acknowledgment

OpenAI's gpt4o[15] was used for generating most of the doc strings found in the code as well as for bug fixing.

[15] https://openai.com/index/hello-gpt-4o/. Accessed 11 October 2024.

Appendix and code

Code Repository: https://doi.org/10.5281/zenodo.13919649

Repository layout

```
project_dir /
    assets /
    data /
        dummy_cal.ics           # Calendar with anonymised data
    labcal /                    # LABcal python package
        __init__.py
        plot_cal.py             # Plotting functions
        process_cal.py          # Data processing functions
    app.py                      # LABcal streamlit app
    labCal_notebook.ipynb       # Annotated code notebook (jupyter)
    LICENSE
    README.md
    requirements.txt
```

The calendar `dummy_cal.ics` stored in the `./data` directory contains anonymised event data for the year 2024. It is used as data input in the following Annotated code section as well as the Jupyter notebook `labCal_notebook.ipynb` which contains annotated and executable code.

While the Annotated code section documents and explains each `labcal` processing step in detail, the Jupyter notebook focuses on the more general functions, while maintaining a level of documentation that explains the main processing steps needed to run the Streamlit web application. The notebook is intended as a quick code walkthrough, while the Annotated code section delves deeper into specific aspects of the code.

Refer to the `README.md` for installation instructions.

Annotated code

The following sections explain the LABcal Python code and reproduce the workflow explained in the Implementation chapter (Data parsing, Data cleaning, Data visualisation, Streamlit web application).

For better understanding, individual code sections that are wrapped in the two Python scripts `./labcal/process_cal.py` and `./labcal/plot_cal.py` in more extensive functions are reproduced in individual calls. This mainly concerns the Data parsing and cleaning section. The subsequent parts of the annotated code reproduce the two Python scripts mentioned identically. Refer to the individual functions for more detailed documentation.

Data parsing and cleaning

Import packages

```
import re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from ics import Calendar, Event
from pathlib import Path

# LABcal python package
from labcal import process_cal, plot_cal

# Set seaborn theme
sns.set_theme(style='whitegrid')
```

Step 1: Data parsing

Load and parse the calendar

```
# Set the filepath for the calendar .ics
filepath = './data/dummy_cal.ics'

# Create Path object from filepath string
cal_filepath = Path(filepath)

# Open the file
with open(cal_filepath, 'r', encoding='utf-8') as fin:
    cal_raw = fin.read()

# Create a Calendar object
cal = Calendar(cal_raw)
cal
```

```
<Calendar with 53 events and 0 todo>
```

Print one event from `cal` to get a feel for the data structure

```
from pprint import pprint

for idx, event in enumerate(cal.events):
    if idx == 4:
        pprint(f"{event.serialize()}\n")
        break
```

```
('BEGIN:VEVENT\r\n'
 'CREATED:20240610T082820Z\r\n'
 'SEQUENCE:2\r\n'
 'DTSTAMP:20240610T082846Z\r\n'
 'DESCRIPTION:Kategorie Workshop: Yoga und Meditation \\nVeranstalter Uni: '
 'Institut für Sport\\nTeilnehmer: bis 10\\nTechnik: keine\\, '
 'Yoga-Equipment\\nKontakt:\\nCatering: nein\\n\r\n'
 'DTEND:20240729T104500Z\r\n'
 'LAST-MODIFIED:20240610T082846Z\r\n'
 'DTSTART:20240729T094500Z\r\n'
 'STATUS:CONFIRMED\r\n'
 'SUMMARY:Sport Yoga\r\n'
 'UID:940dbc83-df6a-4bd2-9753-673af96a10c5\r\n'
 'END:VEVENT\n')
```

Create a list from `cal.events`

```python
def clean_event_data(event: list[str]) -> list:
    """
    Clean a split str event and remove various chars and whitespaces.
    """
    clean_event = []
    for line in event:
        event = re.sub(r'\r', '', line)
        # set '¶' as stop char for splitting later
        event = re.sub(r'\\n', '¶', event)
        event = re.sub(r'\\', '', event)
        clean_event.append(event.strip())
    return clean_event


# Create list from Calendar.events and clean resulting event strings
calendar_data = []
for event in cal.events:
    event_lines = event.serialize().split('\n')
    clean_event = clean_event_data(event_lines)
    calendar_data.append(clean_event)

# Print an item of the list
pprint(calendar_data[4])
```

```
['BEGIN:VEVENT',
 'CREATED:20240610T082820Z',
 'SEQUENCE:2',
 'DTSTAMP:20240610T082846Z',
 'DESCRIPTION:Kategorie Workshop: Yoga und Meditation ¶Veranstalter Uni: '
 'Institut für Sport¶Teilnehmer: bis 10¶Technik: keine, '
 'Yoga-Equipment¶Kontakt:¶Catering: nein¶',
 'DTEND:20240729T104500Z',
 'LAST-MODIFIED:20240610T082846Z',
 'DTSTART:20240729T094500Z',
 'STATUS:CONFIRMED',
 'SUMMARY:Sport Yoga',
 'UID:940dbc83-df6a-4bd2-9753-673af96a10c5',
 'END:VEVENT']
```

Create a `dict` from `calendar_data`

```python
def parse_event(event: list[str]) -> dict:
    """
    Parse a serialized Event and return the data as a dict.
    """
    event_dict = {}
    for item in event:
        if ':' in item:
            key, value = item.split(':', 1)
            event_dict[key] = value
    return event_dict


# Process calendar_data and return a dict with all relevant keys for
# each event (BEGIN, CREATED ...)
parsed_events = [parse_event(event) for event in calendar_data]
```

```python
# Create a set of dict.keys() that will be the columns of the DataFrame
columns = set()
for event in parsed_events:
    columns.update(event.keys())
```

```python
# Create a new calendar_dict with those columns as keys
calendar_dict = {}
for col in columns:
    if not col in calendar_dict:
        calendar_dict[col] = []
```

```python
# Populate the dict with the values of parsed_events
for event in parsed_events:
    for key in calendar_dict.keys():
        # If column key of calendar_dict is found in parsed_event
        # append parsed_event value
        if key in event.keys():
            calendar_dict[key].append(event[key])
        # Else append None
        else:
            calendar_dict[key].append(None)
```

Create a `pd.DataFrame`

```python
# Create pd.DataFrame from calendar_dict
df = pd.DataFrame(calendar_dict)
df.head()
```

Inspect the `DataFrame`

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53 entries, 0 to 52
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   UID               53 non-null     object
 1   SEQUENCE          53 non-null     object
 2   DTEND             53 non-null     object
 3   DESCRIPTION       53 non-null     object
 4   LAST-MODIFIED     53 non-null     object
 5   SUMMARY           53 non-null     object
 6   TRANSP            2 non-null      object
 7   BEGIN             53 non-null     object
 8   X-MOZ-LASTACK     1 non-null      object
 9   END               53 non-null     object
 10  ACTION            1 non-null      object
 11  CREATED           53 non-null     object
 12  STATUS            51 non-null     object
 13  X-MOZ-GENERATION  2 non-null      object
 14  TRIGGER           1 non-null      object
 15  DTSTAMP           53 non-null     object
 16  DTSTART           53 non-null     object
dtypes: object(17)
memory usage: 7.2+ KB
```

```
df.shape
```

```
(53, 17)
```

Step 2: Data cleaning

Clean the `DataFrame`

```
# Create a temporary copy of the DataFrame to work with
temp_df = df.copy()
```

```
# Drop columns that have less than 5 % non-null values
threshold = len(temp_df) / 100 * 5
temp_df_processed = temp_df.dropna(axis=1, thresh=threshold)

temp_df_processed.shape
```

```
(53, 12)
```

```python
# Drop columns that do not contain relevant data
columns_to_drop = ['X-MICROSOFT-CDO-BUSYSTATUS',
                   'CLASS',
                   'PRIORITY',
                   'DTSTAMP',
                   'TRANSP',
                   'SEQUENCE',
                   'LAST-MODIFIED',
                   'BEGIN',
                   'END',
                   'STATUS']

for col in temp_df_processed.columns:
    if col in columns_to_drop:
        temp_df_processed = temp_df_processed.drop(columns=[col])

temp_df_processed.shape
```

```
(53, 6)
```

```python
# Split DESCRIPTION column and store data in new column DESCRIPTION_RAW
temp_df_processed['DESCRIPTION_RAW'] = (
    temp_df_processed['DESCRIPTION']
    .str.split('¶')
)
```

```python
# Rename columns for easier data access
temp_df_processed = temp_df_processed.rename(columns={
    'UID': 'id',
    'DTEND': 'event_end',
    'DTSTART': 'event_start',
    'SUMMARY': 'event_title',
    'CREATED': 'created',
    'DESCRIPTION_RAW': 'event_desc_list',
    'DESCRIPTION': 'event_desc'
    })
```

The main event data is now stored in the column `event_desc_list` of the `temp_df_processed`. The following step includes the cleaning of `event_desc_list`.

```python
def process_description(series: list[str], verbose: bool = False) -> pd.Series:
    """
    Processes a list of event description strings, extracting and cleaning various details
    about each event. If `verbose` is True prints all strings that could not be cleaned.

    Args:
        `series (list[str])`: A list of strings containing event description details.

    Returns:
        pd.Series: A pandas Series containing cleaned and extracted event details, which
        includes:
```

```
    `event_category (str)`: The category of the event.
    `event_description (str)`: A detailed description of the event.
    `organiser (str)`: The main organiser of the event.
    `organiser_detail (str)`: Additional details about the organiser.
    `participant_count (str)`: The number of participants.
    `equipment (bool)`: Whether any equipment is needed.
    `equipment_vr (bool)`: Whether VR equipment is needed.
    `equipment_eye_tracking (bool)`: Whether eye tracking equipment is needed.
    `equipment_clevertouch (bool)`: Whether Clevertouch equipment is needed.
    `equipment_large_monitor (bool)`: Whether a large monitor is needed.
    `equipment_design_thinking (bool)`: Whether design thinking equipment is needed.
    `catering (bool)`: Whether catering is required.
    `notes (str)`: Additional notes about the event.
"""
# Variables for storing split and cleaned data
event_category = None
event_description = None
organiser = None
organiser_detail = None
participant_count = None
equipment = None
equipment_vr = None
equipment_eye_tracking = None
equipment_clevertouch = None
equipment_large_monitor = None
equipment_design_thinking = None
catering = None
notes = None

# Internal functions for data splitting and cleaning
def split_item(item):
    split_item = item.split(' ', maxsplit=1)
    return split_item[1].strip() if len(split_item) > 1 else None

def clean_event_category(event_category: str) -> str:
    patterns = {
        'Interne Veranstaltung': ['interne veran', 'ub intern'],
        'Führung': ['veranstaltung/führung'],
        'Lehrveranstaltung': ['seminar'],
        'Workshop': ['vr-einführung']
     }

    for clean_string, keys in patterns.items():
        if any(key in event_category.lower() for key in keys):
            return clean_string

    return event_category

def clean_organiser(organiser: str) -> str:
    patterns = {
        'UB': ['ub'],
        'Uni': ['uni'],
     }

    for clean_string, keys in patterns.items():
        if any(key in organiser.lower() for key in keys):
            return clean_string
    return organiser

def clean_organiser_detail(organiser_detail: str) -> str:
    """
    Main mapping dictionary to map different name variations to the same key.
    """
    patterns = {
        'Institut für Sport': ['sport'],
        'Social Science': ['sowi',
                'social science',
                'powi'],
        'BWL': ['ls bwl',
                'wirtschaftspädag',
                'sales services'],
        'Jura': ['fak jura',
                'rechtswissenschaft'],
        'Wirtschaftsinformatik': ['wirtschaftsinformatik'],
        'Philosophische Fakultät': ['philosophische',
                                    'phil fak',
                                    'philfak',
                                    'anglistik',
                                    'germanistik'],
        'Stud.-Initiative X': ['student group x'],
        'Stud.-Initiative Y': ['student group y'],
        'Stud.-Initiative Z': ['student group z'],
        'Universitäts-IT': ['uni it'],
        'Universitätsbibliothek': ['explab',
                                'ub',
                                'fdz'],
        'Uni Verwaltung': ['verwaltung'],
        'Fachschaftsrat': ['fsr',
                        'fachschaftsr']
    }

    for clean_string, keys in patterns.items():
        if any(re.search(key, organiser_detail.lower()) for key in keys):
            return clean_string

    return organiser_detail

def clean_equipment(equipment_split: list[str]) -> dict[str: bool]:
    patterns = {
        'equipment_vr': ['vr', 'virtual'],
        'equipment_eye_tracking': ['eye', 'tracking'],
        'equipment_clevertouch': ['clever', 'mobiler'],
        'equipment_large_monitor': ['praesentations', 'großer monitor',
                                    'präsentations'],
        'equipment_design_thinking': ['dt', 'design thinking', 'thinking']
    }

    equipment_vr = False
    equipment_eye_tracking = False
    equipment_clevertouch = False
    equipment_large_monitor = False
    equipment_design_thinking = False

    for item in equipment_split:
        for var, keys in patterns.items():
            if any(key in item.lower() for key in keys):
                if var == 'equipment_vr':
                    equipment_vr = True
                elif var == 'equipment_eye_tracking':
                    equipment_eye_tracking = True
                elif var == 'equipment_clevertouch':
                    equipment_clevertouch = True
                elif var ==  'equipment_large_monitor':
                    equipment_large_monitor = True
                elif var == 'equipment_design_thinking':
                    equipment_design_thinking = True

    return {
        'equipment_vr': equipment_vr,
        'equipment_eye_tracking': equipment_eye_tracking,
        'equipment_clevertouch': equipment_clevertouch,
        'equipment_large_monitor': equipment_large_monitor,
        'equipment_design_thinking': equipment_design_thinking
    }

# Check if series is a list
if isinstance(series, list):
    for item in series:

        # Check if item of list is not None and a string
        if pd.notna(item) and isinstance(item, str):

            ### Process "Kategorie" items
            if item.startswith('Kategorie'):
                result = split_item(item)
                if result and re.search(r':|\s', result):
                    event_category_split = re.split(r':|,', result, maxsplit=1)

                    # Check if event_category_split contains > 1 item
                    if len(event_category_split) > 1:
                        event_category = event_category_split[0].strip()
                        event_description = event_category_split[1].strip()

                    # Data cleaning
                    if event_category:
                        event_category = clean_event_category(
                            event_category
                        )

            ### Process "Veranstalter" items
            elif item.startswith('Veranstalter'):
                result = split_item(item)
                if result and re.search(r':|\s', result):
                    organiser_split = re.split(r':', result, maxsplit=1)

                    # Split organiser_split again if it contains != 1 item
                    if len(organiser_split) != 1:
                        organiser = organiser_split[0].strip()
                        organiser_detail = organiser_split[1].strip()

                    if organiser:
                        organiser = clean_organiser(organiser)
            if organiser_detail:
                organiser_detail = clean_organiser_detail(
                    organiser_detail
                )

            else:
                organiser_detail = None

        ### Process "Teilnehmer" items
        elif item.startswith('Teilnehmer'):
            result = split_item(item)
            if result:
                result = result.strip()

                # If '-' in string (e.g. 10-20) match all digits after '-'
                if result and '-' in result:
                    participant_split = result.split('-')
                    participant_count = re.match(
                        r'\d+',
                        participant_split[1].strip()
                    ).group(0)

                # If no '-' try only matching digits
                else:
                    participant_search = re.search(r'\d+', result)
                    if participant_search:
                        participant_count = participant_search.group(0)
                    else:
                        participant_count = None

            # If no result set participant_count to None
            else:
                participant_count = None

        ### Process "Technik" items
        elif item.startswith('Technik'):
            result = split_item(item)
            if result:
                equipment_split = result.split(',')
                equipment_split = [item.strip() for item in equipment_split]
            else:
                equipment = False

            if equipment_split:
                equipment_dict = clean_equipment(equipment_split)
                equipment = True if True in equipment_dict.values() else False
                equipment_vr = equipment_dict['equipment_vr']
                equipment_eye_tracking = equipment_dict['equipment_eye_tracking']
                equipment_clevertouch = equipment_dict['equipment_clevertouch']
                equipment_large_monitor = equipment_dict['equipment_large_monitor']
                equipment_design_thinking = equipment_dict['equipment_design_thinking']


        ### Process "Catering" items
        elif item.startswith('Catering'):
            result = split_item(item)
            if result and 'ja' in result:
```

```
                        catering = True
                    else:
                        catering = False

                ### Process "Anmerkung" items
                elif item.startswith('Anmerkung'):
                    result = split_item(item)
                    if result:
                        notes = result.strip()
                    else:
                        notes = None

    return pd.Series([event_category,
                      event_description,
                      organiser,
                      organiser_detail,
                      participant_count,
                      equipment,
                      equipment_vr,
                      equipment_eye_tracking,
                      equipment_clevertouch,
                      equipment_design_thinking,
                      equipment_large_monitor,
                      catering,
                      notes])


# Clean the data of event_desc_list and create new columns accordingly
temp_df_processed[['event_category', 'event_description', 'organiser', 'organiser_detail',
                   'participant_count', 'equip', 'equip_vr', 'equip_eyetracking',
                   'equip_clevertouch', 'equip_dt', 'equip_monitor', 'catering', 'notes']] = (
    temp_df_processed['event_desc_list'].apply(lambda x: process_description(x))
)
```

```
# Re-order columns for better clarity
new_column_layout = ['id', 'created', 'event_title', 'event_start', 'event_end',
                     'event_category', 'event_description', 'event_desc_list',
                     'event_desc', 'organiser', 'organiser_detail', 'participant_count',
                     'equip', 'equip_clevertouch', 'equip_dt', 'equip_eyetracking',
                     'equip_monitor', 'equip_vr', 'catering', 'notes']
temp_df_processed = temp_df_processed.filter(new_column_layout)
```

```
# Convert columns to appropriate dtypes
column_dtypes = {
    'int': ['participant_count'],
    'date': ['created', 'event_start', 'event_end']
}

for dtype, columns in column_dtypes.items():
    for col in columns:
        if col in temp_df_processed.columns:
            if dtype == 'int':
                temp_df_processed[col] = (
                    temp_df_processed[col]
                    .fillna(0)
                    .astype('int16')
                )
            elif dtype == 'date':
                temp_df_processed[col] = (
                    pd.to_datetime(temp_df_processed[col],
                                   errors='coerce')
                )
        else:
            print(f'{col} not a column. Skipping ...')
```

Inspect the cleaned `DataFrame`

```
df_clean = temp_df_processed.copy()
df_clean.head()
```

Step 3: Data visualisation

The data visualisation is handled by the function `plot_cal.plot_calendar()` that takes several parameters and uses various helper functions.

Helper functions for `plot_cal.plot_calendar()`

```
def set_output_dir(dir_name: str = './output'):
    output_dir = Path(dir_name)
    if not output_dir.exists():
        output_dir.mkdir(parents=True)
    return output_dir


def get_color_mapping(column: str) -> dict:
    """
    Different color maps for plotting
    """
    # Okabe ito color map for categorial data:
    # https://clauswilke.com/dataviz/color-basics.html#ref-Okabe-Ito-CUD
    cmap_okabe_ito = {
        'Workshop': '#E69F00',
        'Lehrveranstaltung': '#56B4E9',
        'Interne Veranstaltung': '#009E73',
        'Veranstaltung': '#F0E442',
        'Führung': '#0072B2',
    }

    # ColorBrew map for qualitative data (colorblind safe):
    # https://colorbrewer2.org/?type=qualitative&scheme=Paired&n=4
    cmap_cbrew = {
        'Uni': '#a6cee3',
        'UB': '#1f78b4',
        'Extern': '#b2df8a',
        'Studis': '#33a02c'
    }
```

```python
    # ColorBrew map for qualitative data:
    # https://colorbrewer2.org/?type=qualitative&scheme=Set2&n=6
    cmap_cbrew_equip = {
        'equip_monitor': '#66c2a5',
        'equip_clevertouch': '#fc8d62',
        'equip_vr': '#8da0cb',
        'equip_eyetracking': '#e78ac3',
        'equip_dt': '#a6d854',
        'equip': '#ffd92f'
    }

    if column == 'event_category':
        return cmap_okabe_ito
    elif column == 'organiser':
        return cmap_cbrew
    elif column in ['equip']:
        return cmap_cbrew_equip


def create_img_title(title: str):
    return re.sub(r'[\s\W]+', '_', title.lower())[:-1]


def parse_and_verify_dates(df: pd.DataFrame,
                           start_date: str,
                           end_date: str) -> tuple[pd.Timestamp, str, pd.Timestamp, str]:
    """
    Converts the input start and end dates to timezone-aware datetime objects and their
    corresponding string representations. Adjusts the dates to the first of the month
    for start date and the last of the month for end date if they don't exist in the DataFrame.

    Parameters:
    start_date (str): The start date in the format 'DD.MM.YYYY'.
    end_date (str): The end date in the format 'DD.MM.YYYY'.

    Returns:
    tuple: A tuple containing:
        - start_date (pd.Timestamp): The start date as a timezone-aware datetime object.
        - start_date_str (str): The start date as a formatted string 'DD.MM.YYYY'.
        - end_date (pd.Timestamp): The end date as a timezone-aware datetime object.
        - end_date_str (str): The end date as a formatted string 'DD.MM.YYYY'.
    """
    # Timezone aware datetime objects are needed; otherwise DataFrame filtering would result
    # in a TypeError
    start_date = pd.to_datetime(start_date, dayfirst=True).tz_localize('GMT')
    end_date = pd.to_datetime(end_date, dayfirst=True).tz_localize('GMT')

    # Adjust start date to the first of the month and end date to the last of the month
    start_date = start_date.replace(day=1)
    end_date = end_date + pd.offsets.MonthEnd(0)

    # Define datetime strings for easier plotting
    start_date_str = start_date.strftime('%d.%m.%Y')
    end_date_str = end_date.strftime('%d.%m.%Y')

    # Check if dates exist in the DataFrame
    if df is not None and not df.empty:
        df_start_date = df['event_start'].min()
        df_end_date = df['event_end'].max()

        # Adjust DataFrame start date to the first of the month
        df_start_date = df_start_date.replace(day=1)

        # Adjust DataFrame end date to the last of the month
        df_end_date = df_end_date + pd.offsets.MonthEnd(0)

        if start_date < df_start_date:
            start_date = df_start_date
            start_date_str = start_date.strftime('%d.%m.%Y')

        if end_date > df_end_date:
            end_date = df_end_date
            end_date_str = end_date.strftime('%d.%m.%Y')

    return start_date, start_date_str, end_date, end_date_str
```

```
# Set output folder for plots (default: './output')
output_dir = set_output_dir(dir_name='./output')
```

Main plotting function `plot_cal.plot_calendar()`

```
from pprint import pprint

def plot_calendar(df: pd.DataFrame = None,
                  start_date: str = '01.01.2024',
                  end_date: str  = '01.02.2024',
                  func: str = 'organiser',
                  top_k: int = 10,
                  streamlit: bool = False,
                  list_func: bool = False):
    """
    Plots various types of visualisations based on the input event data within a
    specified date range.

    Parameters:
    -----------
    df : pd.DataFrame, optional
        DataFrame containing event data with necessary columns depending on the
        selected `func`.
        Default is None.
    start_date : str, optional
        The start date for filtering events in the format 'DD.MM.YYYY'.
        Default is '01.01.2024'.
    end_date : str, optional
        The end date for filtering events in the format 'DD.MM.YYYY'.
        Default is '01.02.2024'.
    func : str, optional
        The type of plot to generate. Must be one of the following:
        ['organiser', 'organiser_detail', 'event_category', 'event_category_by_organiser',
         'equip_details', 'equip_overall', 'participant_stats', 'participants_per_month'].
        Default is 'organiser'.
    top_k : int, optional
        The number of top categories to display in the plot for certain functions.
        Default is 10.
    streamlit : bool, optional
        If True, the plot will be displayed using Streamlit's `st.pyplot()`.
        Default is False.
    list_func : bool, optional
        If True, prints the available plotting functions and returns without generating a plot.
        Default is False.

    Returns:
    --------
    None or str
        If `streamlit` is True, returns the file path of the saved plot image.
        Otherwise, displays the plot using matplotlib.

    Raises:
    -------
    ValueError
        If `func` is not one of the available plotting functions.

    Notes:
    ------
    This function generates different types of plots based on the selected `func` parameter:
    - 'organiser': Plots events by organiser.
    - 'organiser_detail': Plots top `top_k` events by organiser details.
    - 'event_category': Plots events by event category.
    - 'event_category_by_organiser': Plots event categories grouped by organisers.
    - 'equip_details': Plots usage of different equipment types in events.
    - 'equip_overall': Plots overall tool usage in events.
    - 'participant_stats': Plots total participants by event category.
    - 'participants_per_month': Plots total participants per month.

    Example:
    --------
    >>> plot_calendar(df=events_df, start_date='01.01.2024',
                      end_date='31.01.2024', func='organiser')
    """
    # Define available plotting functions
    all_funcs = ['organiser',
                 'organiser_detail',
                 'event_category',
                 'event_category_by_organiser',
                 'equip_details',
                 'equip_overall',
                 'participant_stats',
                 'participants_per_month']

    if list_func:
        pprint(f"Available plotting functions: {all_funcs}")
        return

    if not func in all_funcs:
        print(f"Plotting function {func} not available. Check available functions \
        by passing the 'list_func' parameter.")
        return

    # Parse dates and check if they exist in the DataFrame
    start_date, start_date_str, end_date, end_date_str = parse_and_verify_dates(df,
                                                                              start_date,
                                                                              end_date)
    # Define color map
    color_mapping = get_color_mapping('organiser')

    # Plot data for different functions
    if func == 'organiser_detail':
        # Filter data
        temp_df = df[(df['event_start'] >= start_date) & (df['event_end'] <= end_date)].copy()
        result_counts = temp_df['organiser_detail'].value_counts().head(top_k)
        result_counts = result_counts.sort_values(ascending=True)

        # Get colors from map for key in dict
        bar_colors = [color_mapping.get(df[df['organiser_detail'] == detail]['organiser'].iloc[0],
                                        '#333333') for detail in result_counts.index]

        # Define labels
        title = f"""Anzahl Veranstaltungen nach Veranstalterdetails (Top {top_k})
({start_date_str} bis {end_date_str}) | n={result_counts.values.sum()}"""
        xlabel = 'Anzahl Veranstaltungen'
        ylabel = 'Veranstalterdetails'

    elif func == 'organiser':
        result_counts = (
            df[(df['event_start'] >= start_date) & (df['event_end'] <= end_date)]
            ['organiser']
            .value_counts()
            )
        bar_colors = [color_mapping.get(cat, '#333333') for cat in result_counts.index]
        title = f"""Anzahl Veranstaltungen nach Veranstaltern
({start_date_str} bis {end_date_str}) | n={result_counts.values.sum()}"""
        xlabel = 'Veranstalter'
        ylabel = 'Anzahl Veranstaltungen'

    elif func == 'event_category':
        result_counts = (
            df[(df['event_start'] >= start_date) & (df['event_end'] <= end_date)]
            ['event_category']
            .value_counts()
            )
        color_mapping = get_color_mapping('event_category')
        bar_colors = [color_mapping.get(cat, '#333333') for cat in result_counts.index]
        title = f"""Anzahl Veranstaltungen nach Veranstaltungstyp
({start_date_str} bis {end_date_str}) | n={result_counts.values.sum()}"""
        xlabel = 'Veranstaltungstyp'
        ylabel = 'Anzahl Veranstaltungen'

    elif func == 'event_category_by_organiser':
        # Filter data
        temp_df = df[(df['event_start'] >= start_date) & (df['event_end'] <= end_date)].copy()
        result_counts = (
            temp_df
            .groupby(['organiser', 'event_category'])
            .size()
            .unstack(fill_value=0)
            )
        result_counts['total_events'] = result_counts.sum(axis=1)
        result_counts = result_counts.sort_values(by='total_events', ascending=False)
        result_counts = result_counts.drop('total_events', axis=1)
        # Define color mapping
        color_mapping = get_color_mapping(column='event_category')
        bar_colors = [color_mapping.get(cat, '#333333') for cat in result_counts.columns]

        # Define labels
        title = f"""Anzahl Veranstaltungen nach Veranstaltungstyp und Veranstalter
({start_date_str} bis {end_date_str}) | n={result_counts.values.sum()}"""
        xlabel = 'Veranstalter'
        ylabel = 'Anzahl Veranstaltungen'

    elif func == 'equip_details':
        # Filter data
        equip_cols = {
            'equip_clevertouch': 'Clevertouch',
            'equip_dt': 'Design Thinking',
            'equip_eyetracking': 'Eye Tracking',
            'equip_monitor': 'Präsentationsmonitor',
            'equip_vr': 'VR'
        }

        result_counts = (
            df
            [(df['event_start'] >= start_date) & (df['event_end'] <= end_date)]
            [list(equip_cols.keys())]
            .sum()
            .sort_values(ascending=False)
            )

        # Define color mapping
        color_mapping = get_color_mapping('equip')
        bar_colors = [color_mapping.get(key, '#333333') for key in color_mapping]

        # Define labels
        title = f"""Toolnutzung in Veranstaltungen nach Tooltyp
({start_date_str} bis {end_date_str}) | n={result_counts.values.sum()}‡"""
        ylabel = 'Anzahl Veranstaltungen'
        xlabel = """Tooltyp\n
‡ Nutzung mehrerer Tools pro Veranstaltung möglich"""

    elif func == 'equip_overall':
        result_counts = (
            df[(df['event_start'] >= start_date) & (df['event_end'] <= end_date)]
            ['equip']
            .value_counts()
            )
        bar_colors = ['#e5c494', '#b3b3b3']
        title = f"""Toolnutzung in Veranstaltungen
({start_date_str} bis {end_date_str}) | n={result_counts.values.sum()}"""
        ylabel = 'Anzahl Veranstaltungen'
        xlabel = 'Toolnutzung'

    elif func == 'participant_stats':
        result_counts = (
            df[(df['event_start'] >= start_date) & (df['event_end'] <= end_date)]
            .groupby(['event_category'])['participant_count']
            .sum()
            .sort_values(ascending=False)
        )
        color_mapping = get_color_mapping('event_category')
        bar_colors = [color_mapping.get(key, '#333333') for key in result_counts.index]
        title = f"""Anzahl Teilnehmende nach Veranstaltungstyp
({start_date_str} bis {end_date_str})‡ | n={result_counts.values.sum()}"""
        xlabel = f"""Veranstaltungstyp

‡ Teilnehmendenzahl nach Angabe der Veranstalter vor Durchführung einer Veranstaltung"""
        ylabel = 'Anzahl Teilnehmende'

    elif func == 'participants_per_month':
        result_counts = (
            df.groupby([df['event_start'].dt.year, df['event_start'].dt.month])
            ['participant_count']
            .sum()
            )
        bar_colors = ['#e5c494']
        title = f"""Anzahl Teilnehmende pro Monat
({start_date_str} bis {end_date_str})‡ | n={result_counts.values.sum()}"""
        xlabel = f"""Monat

‡ Teilnehmendenzahl nach Angabe der Veranstalter vor Durchführung einer Veranstaltung"""
        ylabel = 'Anzahl Teilnehmende'

    # Plot data
    if func in ['organiser', 'event_category', 'equip_details', 'participant_stats',
                'participants_per_month']:
        plt.figure(figsize=(10, 8))
        plt.grid(True, which='both', linestyle='-', linewidth=0.2)

        bars = plt.bar(list(range(1, len(result_counts) + 1)),
                       result_counts.values,
                       color=bar_colors)
        for bar in bars:
            bar_height = bar.get_height()
            plt.text(bar.get_x() + bar.get_width()/2,
                     bar_height,
                     int(bar_height),
                     ha='center',
                     va='bottom')

        # xticks
        if func in ['equip_details']:
            plt.xticks(ticks=list(range(1, len(result_counts) + 1)),
                       labels=[equip_cols[key] for key in result_counts.index],
                       rotation=0,
                       ha='center')
        elif func in ['participant_stats', 'event_category', 'organiser']:
            plt.xticks(ticks=list(range(1, len(result_counts) + 1)),
                       labels=result_counts.index,
                       rotation=0,
                       ha='center')
        else:
            plt.xticks(ticks=list(range(1, len(result_counts) + 1)),
                       labels=result_counts.index,
                       rotation=45,
                       ha='right')
    elif func in ['organiser_detail']:
        plt.figure(figsize=(12, 8))
        plt.grid(True, which='both', linestyle='-', linewidth=0.2)
        bars = plt.barh(list(range(1, len(result_counts) + 1)),
                        result_counts.values,
                        color=bar_colors)
        for bar in bars:
            bar_width = bar.get_width()
            plt.text(bar_width + 0.1, bar.get_y() + bar.get_height() / 2,
                     int(bar_width),
                     ha='left',
                     va='center')
        plt.yticks(ticks=list(range(1, len(result_counts) + 1)),
                          labels=result_counts.index)
        plt.ylim(0.3, len(result_counts) + 0.7)

    elif func in ['event_category_by_organiser']:
        ax = result_counts.plot(kind='bar', stacked=False, figsize=(12, 8),
                                color=bar_colors)
        plt.grid(True, which='both', linestyle='-', linewidth=0.2)
        for container in ax.containers:
            ax.bar_label(container)
        plt.xticks(rotation=0, ha='center')

    elif func in ['equip_overall']:
        plt.figure(figsize=(6, 9))
        plt.grid(True, which='both', linestyle='-', linewidth=0.2)
        bars = plt.bar(list(range(1, len(result_counts) + 1)),
                       result_counts.values,
                       color=bar_colors)
        for bar in bars:
            bar_height = bar.get_height()
            plt.text(bar.get_x() + bar.get_width()/2, bar_height, int(bar_height),
                     ha='center', va='bottom')
        plt.xticks(ticks=list(range(1, len(result_counts) + 1)),
                   labels=['Toolnutzung', 'Keine Toolnutzung'],
                   rotation=0,
                   ha='center')
        plt.xlim(0.5, len(result_counts) + 0.5)

    # Plot labels
    plt.title(title, weight='bold', fontsize=13, pad=10)
    plt.xlabel(xlabel, weight='bold', labelpad=25)
    plt.ylabel(ylabel, weight='bold', labelpad=10)

    # Plot legends
    if func == 'organiser_detail':
        handles = [plt.Rectangle((0,0),1,1, color=color_mapping[key]) for key in color_mapping]
        labels = color_mapping.keys()
        plt.legend(handles, labels, title="Veranstalter", loc='lower right')

    elif func in ['event_category_by_organiser']:
        plt.legend(title='Veranstaltungstyp')

    # Tight layout
    plt.tight_layout()

    # Save and show plot
    img_title = create_img_title(title)
    plt.savefig(f"{str(output_dir.joinpath(img_title))}", dpi=300, bbox_inches='tight')

    # If running the function via the streamlit app use st.pyplot()
    if streamlit:
        st.pyplot(plt)
        return f"{output_dir.joinpath(img_title)}.png"

    plt.show()
```

The function `plot_calender()` plots several visualisations via the `func` parameter:

```
# List all available plotting functions
plot_calendar(list_func=True)
```

```
("Available plotting functions: ['organiser', 'organiser_detail', "
 "'event_category', 'event_category_by_organiser', 'equip_details', "
 "'equip_overall', 'participant_stats', 'participants_per_month']")
```

```
plot_funcs = ['organiser', 'organiser_detail', 'event_category',
              'event_category_by_organiser', 'equip_details', 'equip_overall',
              'participant_stats', 'participants_per_month']

for f in plot_funcs:
    plot_calendar(df_clean,
                  start_date='01.01.2024',
                  end_date='31.12.2024',
                  func=f,
                  top_k=20)
```

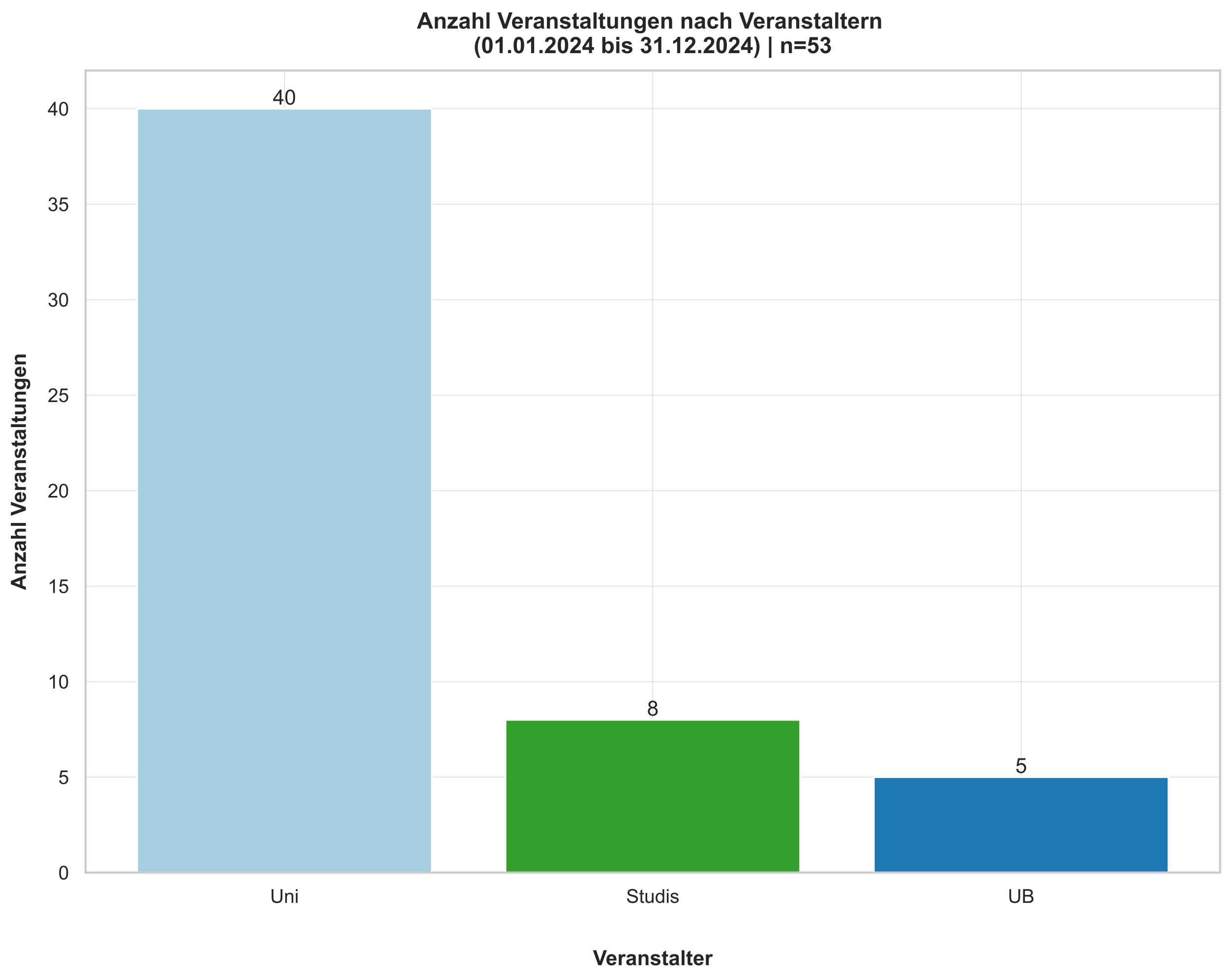


Figure 2: Bar plot using `plot_calendar()` with `func='organiser'`.

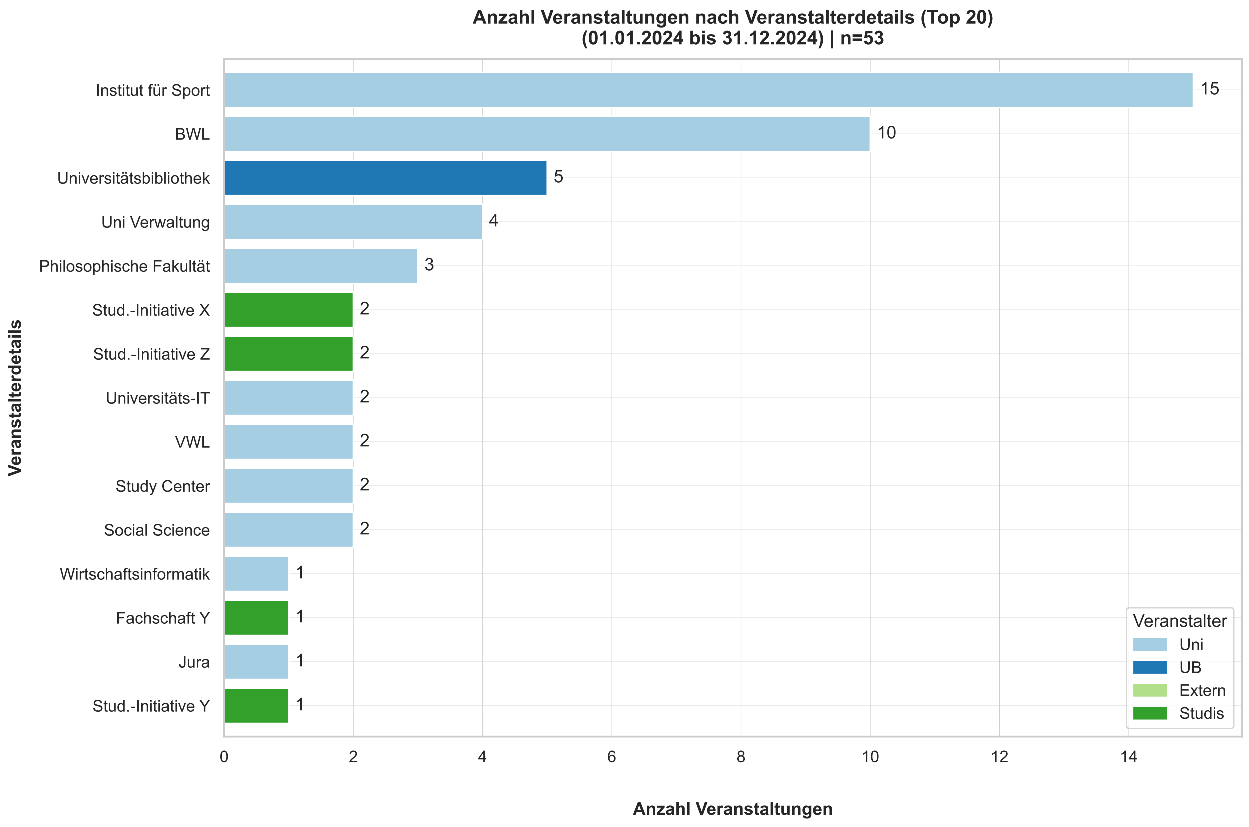


Figure 3: Bar plot using `plot_calendar()` with `func='organiser_detail'`.

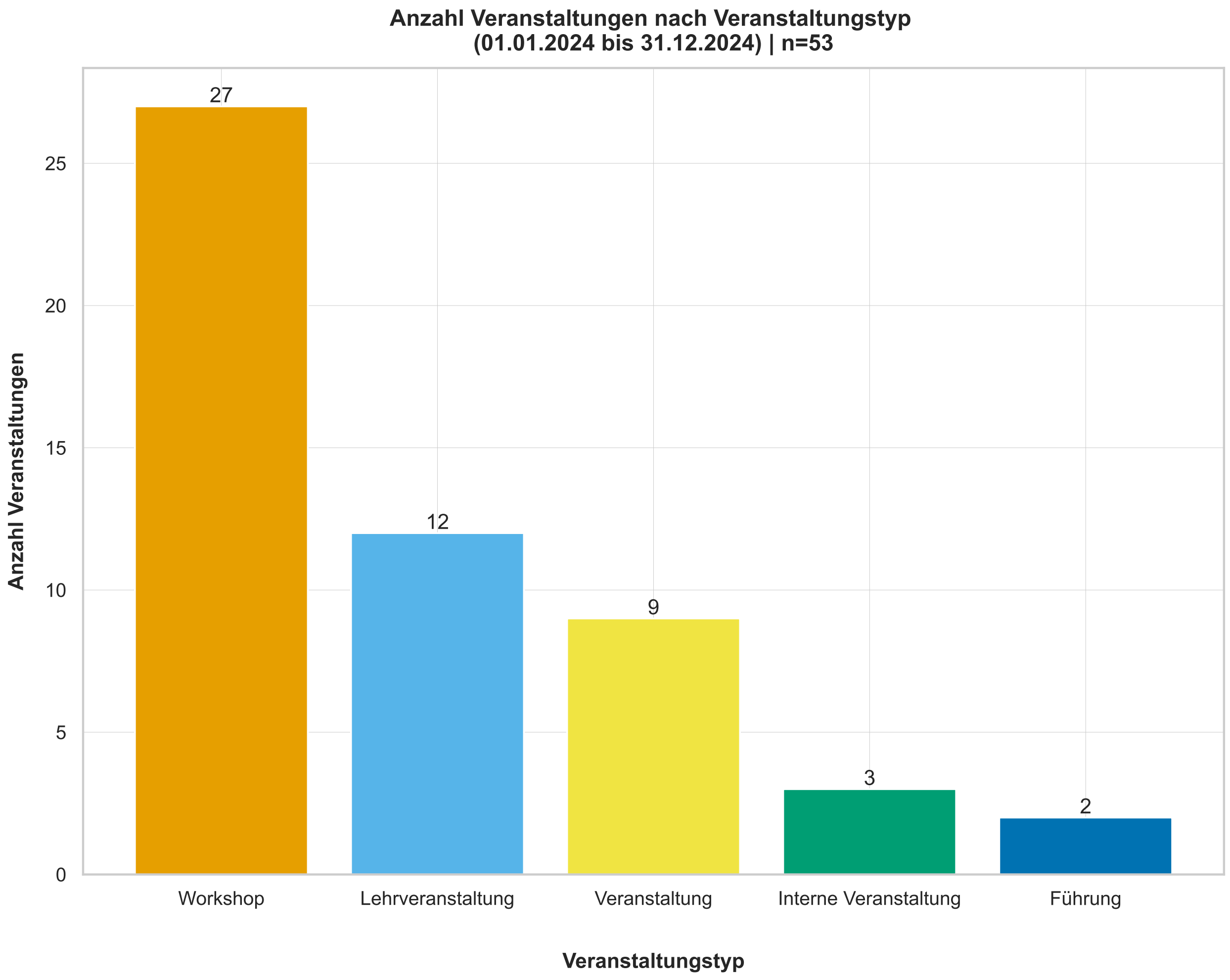


Figure 4: Bar plot using `plot_calendar()` with `func='event_category'`.

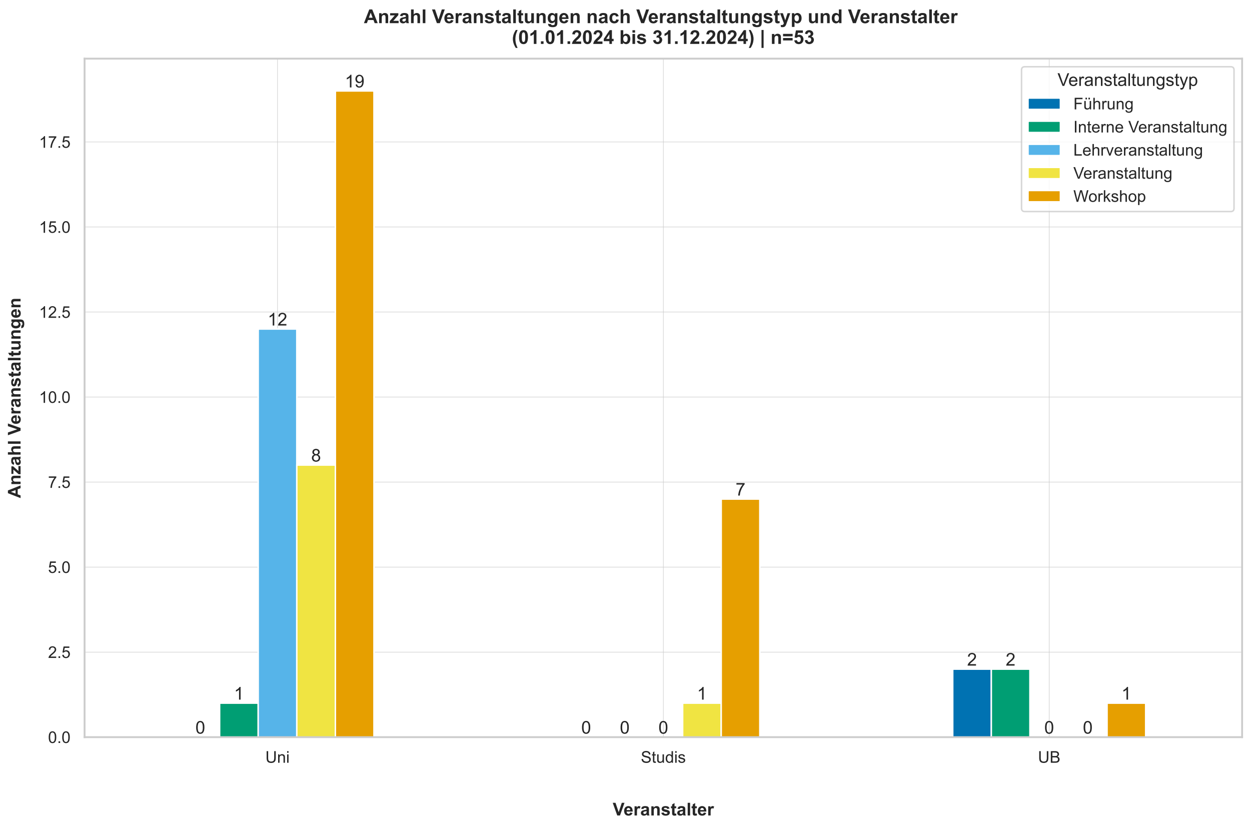


Figure 5: Bar plot using `plot_calendar()` with `func=='event_category_by_organiser'`.

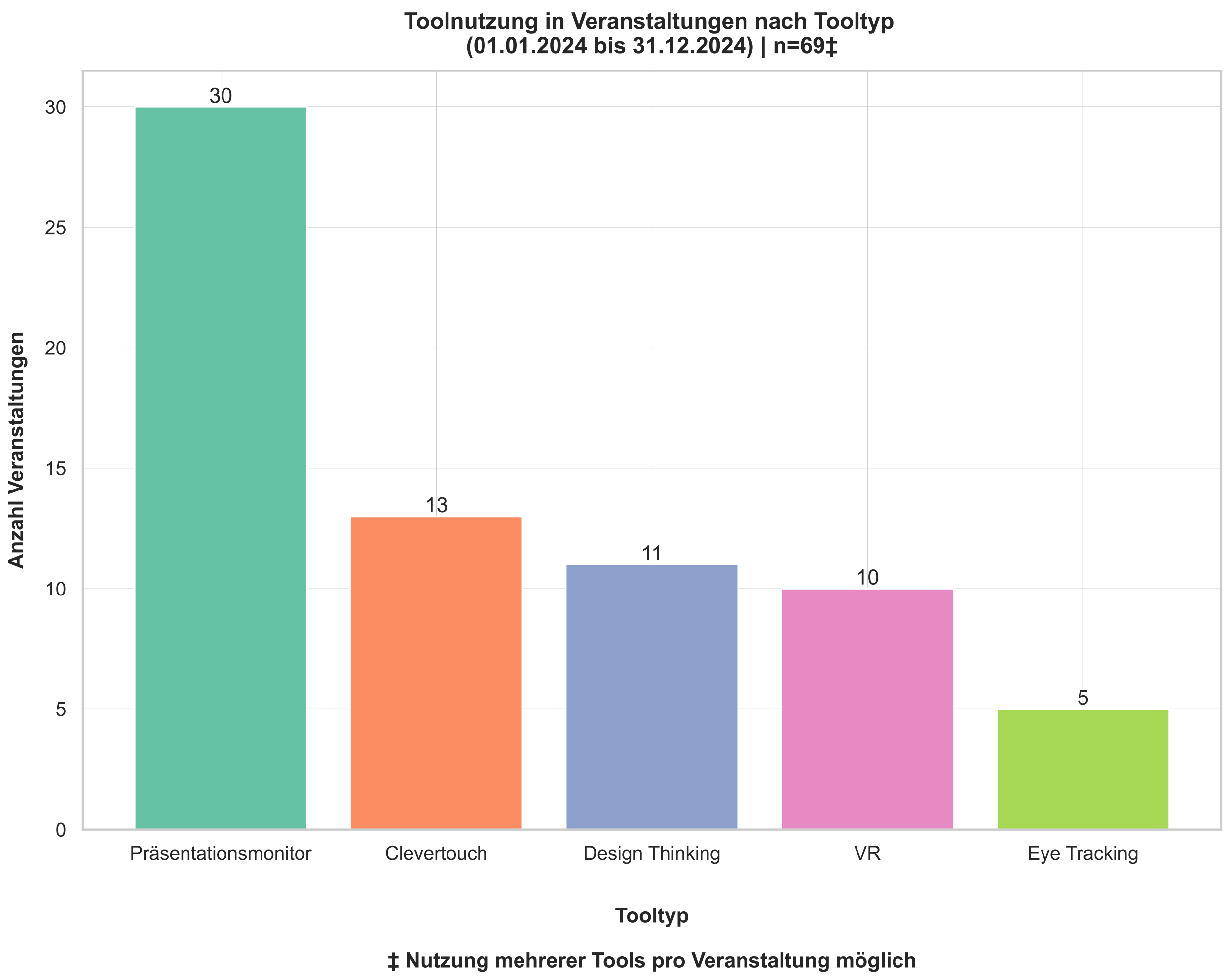


Figure 6: Bar plot using `plot_calendar()` with `func='equip_details'`.

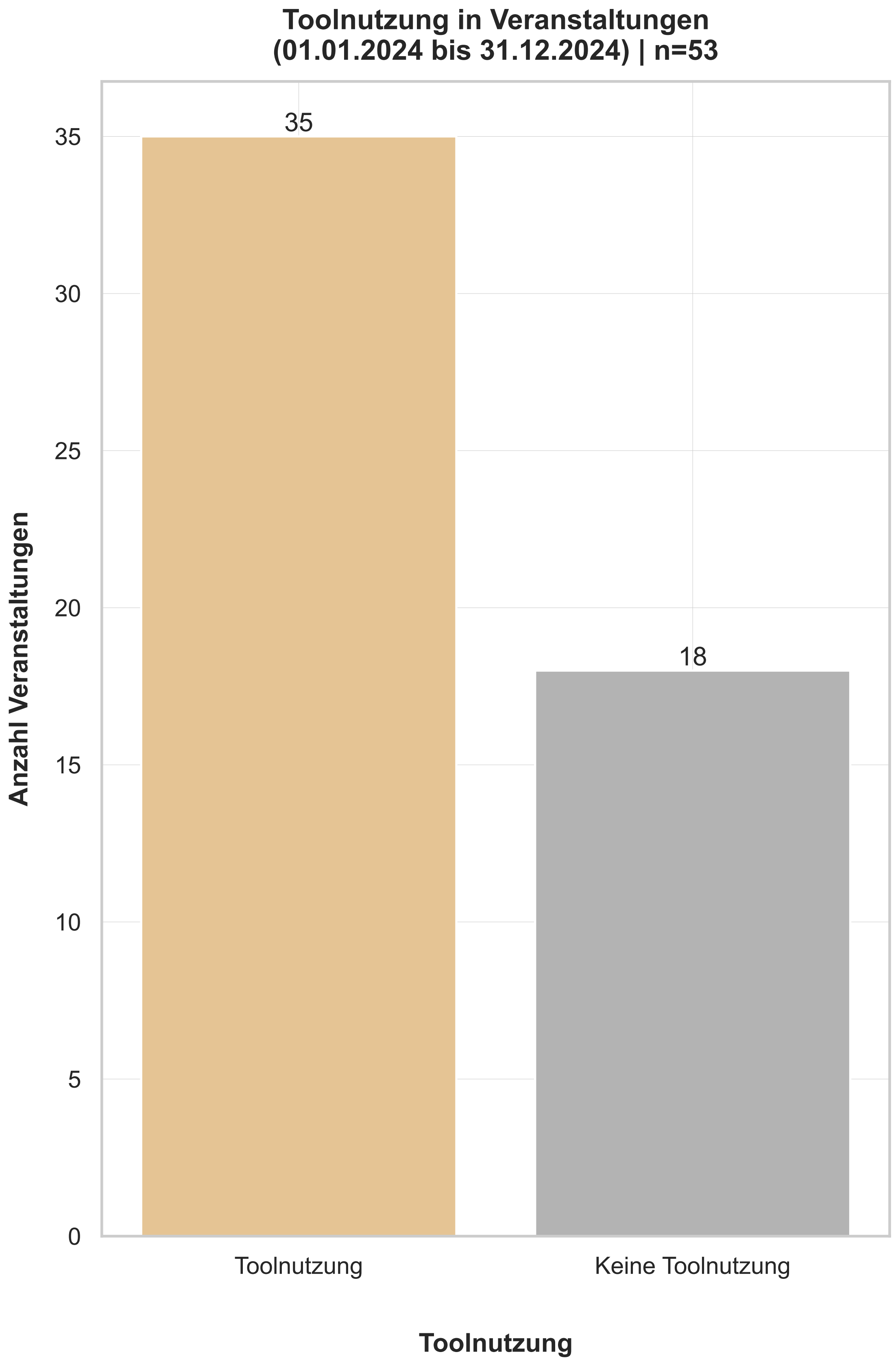


Figure 7: Bar plot using `plot_calendar()` with `func='equip_overall'`.

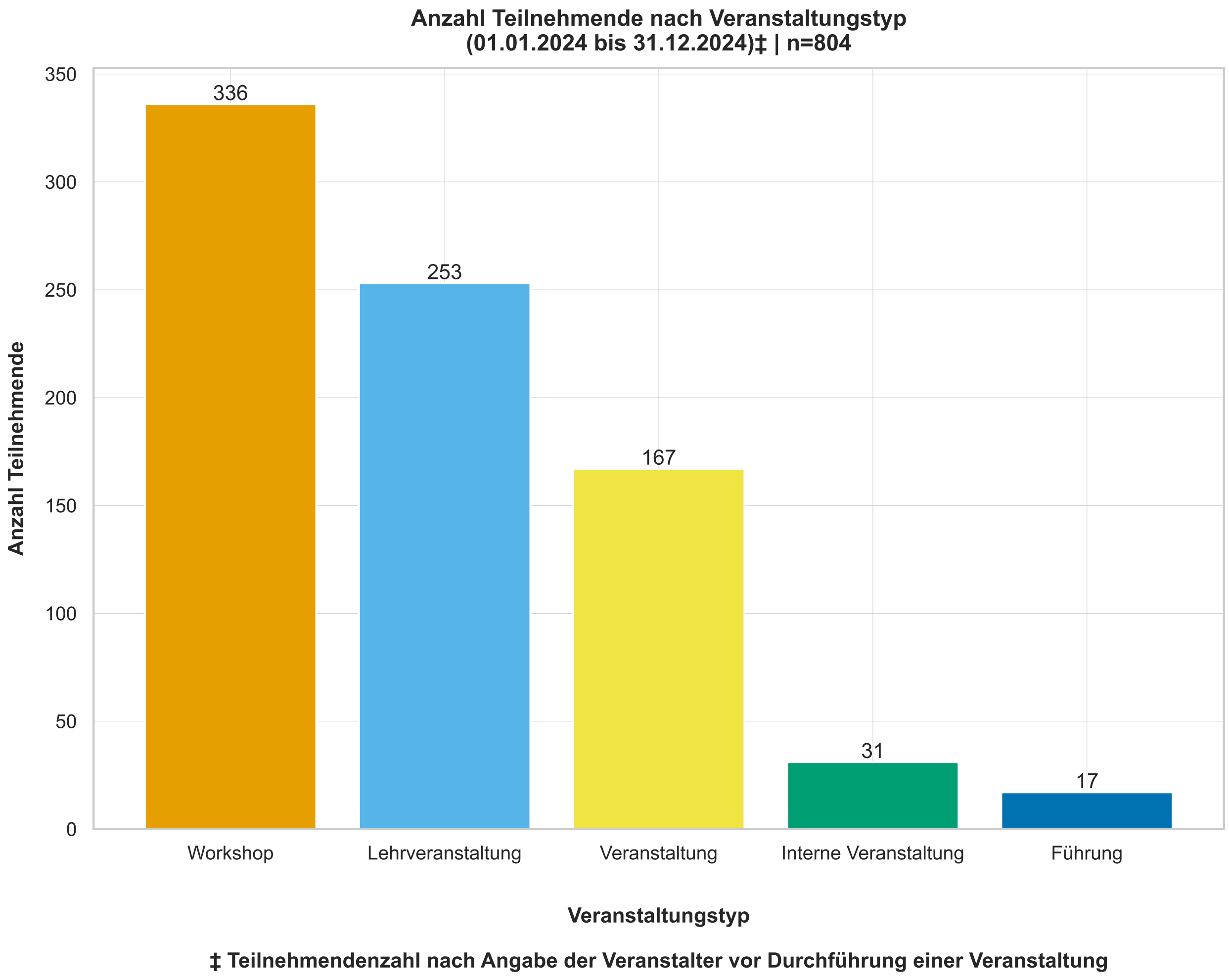


Figure 8: Bar plot using `plot_calendar()` with `func='participant_stats'`.

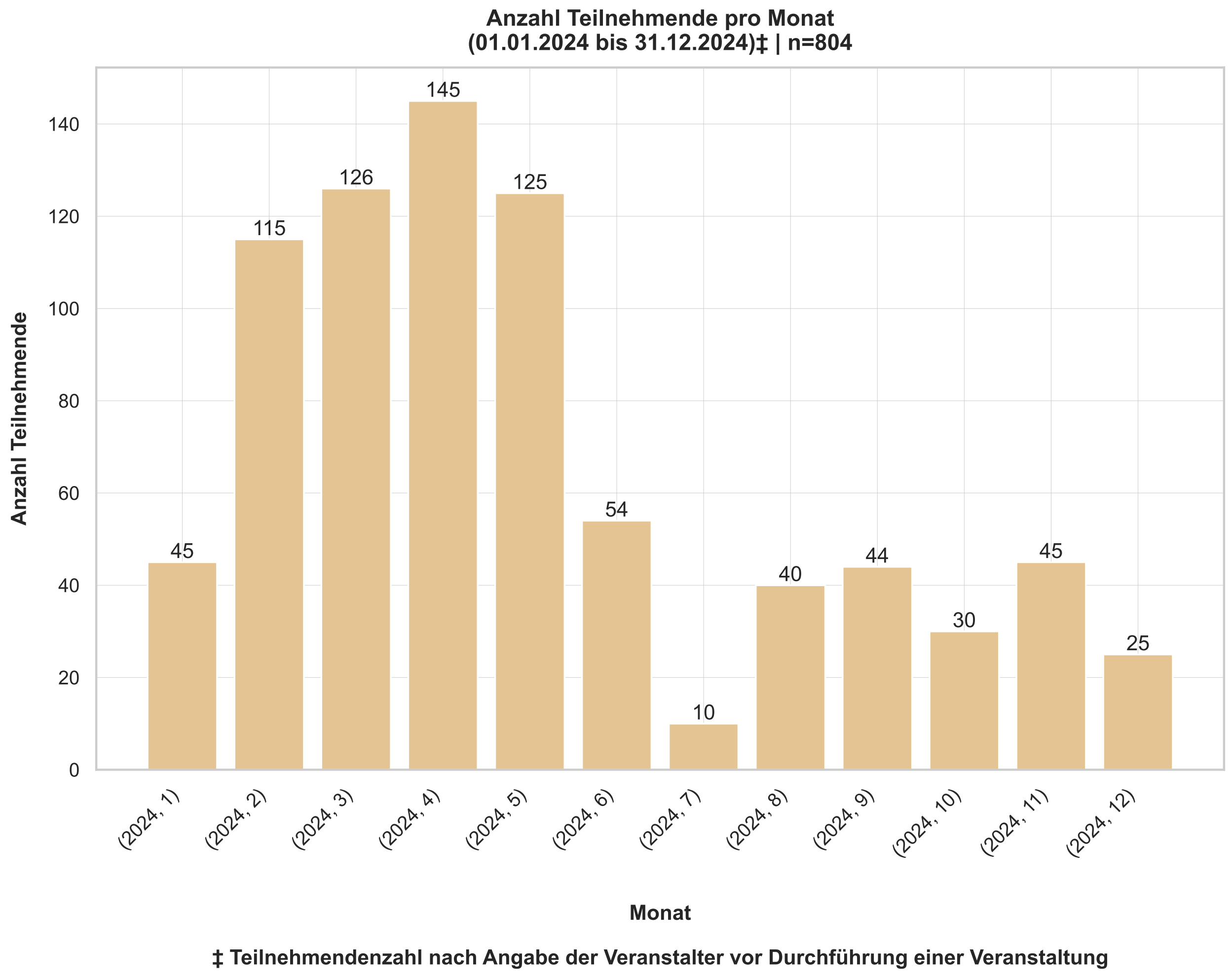


Figure 9: Bar plot using `plot_calendar()` with `func='participants_per_month'`.

Step 4: Streamlit web application

The streamlit web application combines all functionality mentioned above and is started from the commmand line using the streamlit specific `run` command:

```
$ streamlit run app.py
```