Exploratory script for carbon cycle analysis

on the Tegel Project dataset.

the dataset is represented in the input file 'tp_config.yml'

How to:

Input: Place the input file ('tp_config.yml' by default) in the same directory as the code.
In this dataset, for each construction system, masses per building part are given for each material use case (each building part being present once per building!).

Input-structure: The structure and level0 keywords from the yml file are referenced in the code and musn't be changed independently. However, adding lines next to existing ones in a similar matter doesn't require changes in code.

Output: Create a folder 'export' in the same directory as the code. Table and Graphic files are being written to this folder.

Terminal: No interaction with or prompt in the Terminal needed. The print statements are more than nice-to-have. Feel free to intuitively use and extend them for so-called print-based debugging.

calculations: The whole idea consists of calculating everything on the very level that is given in the input file, so to take every single material that is used in a part (etc.) of a building, or an area or what might be of interest in the end. And with this single material, its (sub)fraction of the final emissions, carbon storage etc.
This emancipates the depth of analysis of the calculations. The opposite case would be if you calculated only the emissions for e.g. a building and then can't go back to compare the contributions of the building parts (or materials).
Per material use, the following is being calculated:

co2e_per_building = mass_per_building * co2coeff
total_emissions = co2e_per_building * number_of_buildings

c_storage_per_building = mass_per_building * biom_fraction * carbon_ratio
total_c_storage = c_storage_per_building * number_of_buildings

The main part of this jupyther notebook has the following sections:

1. Preconditions and Loading Input Config
2. Initialisation and Matching Constants
3. Actual Calculations

4. Data Aggregation and Export
5. Plotting and Graph Export

Comparison of the Output with previous work

Beforehand, an excel Sheet was used to calculate and sum the respective inputs.
The following emissions per building were calculated:

Construction System:	Timber Frame	Mass Timber	Light Weight Timber	Concrete & Steel	Brick
t CO2e (production):	293.642	242.061	235.071	2681.329	2070.139

For comparison: The following code produces these numbers (names were partly updated but correspond):
Please note that also many more aggregations of the output data are accesible.

Construction System:	Timber Frame	Mass Timber	Light Weight Timber	Reinf. Concrete	Brick-based
t CO2e (production):	293.642	242.442	235.071	2683.130	2071.941

The discrepancies present result from malconfigurations of the excel sheet and inconsistent use of emission coefficients for the same material:
e.g. reinforcement (0.68 or 0.683) or clt (0.1299 or 0.1298)
To avoid this, make sure that when copying from an excel sheet, the actual values are being copied to the configuration file, not only the displayed numbers (in their perhaps limiting decimal formatting). Also material names should only be referred to one single coefficient at a tim - if there is two, that there are two materials with different names (only exception: the coefficient is given as a range, e.g. with minimal and maximal values).

Sidenotes:

- *consistent naming:* corresponding specific files named with prefix 'tp_'

- At this stage, the input data and the code don't match the uncertainty of literature on topics as emission coefficients or carbon content of materials. Representing this dimension of uncertainty (e.g. as error categories min, mean, max) requires adaptation of the code on a basic level. This raises the question of whether to use one df for all, one per dimension or one per attribute of interest (e.g. emissions per building). Recommendation goes towards having all data present in one df, making it universally accessible but slightly more difficult to group by and pivot at will. Multi-indexing is promising for that task yet it is quite a logic to itself.

- future to do: should warn if mutliple entries with same material name, as only the first one will be imported (dicts enforce unique keys)

- adding material use efficiency (discard from processing and refining)

- and transport emissions
  - from place of material extraction to place of fabrication (including discard mass)
  - and from place of fabrication to the construction area (only mass used)
  - (and maybe emissions from discarding itself)

The following code cell is an recursive function (consisting of several functions) that can handle deeply nested yaml input and returns those as pandas dataframes in a list. It is not important to get behind its functioning to follow the rest of this code and might as well be externalised.

In [ ]:
```
from typing import Any, Dict, List
import pandas as pd
import yaml

# function and helper-function to dynamicly extract nested dict paths to list of li

def _into_rows(nested_dict: Dict[str, Any], path: List[Any], rows: List[List[Any]])
    for key, value in nested_dict.items():
        innerpath = path + [key]
        if isinstance(value, dict):
            _into_rows(value, innerpath, rows)
        else:
            rows.append(innerpath + [value])

def _nested_dict_into_rows(nested_dict: Dict[str, Any]) -> List[List[Any]]:
    rows = []
    _into_rows(nested_dict, [], rows)
    return rows

def yaml_to_dict_of_dfs(nested_dict: Dict[str, Any]) -> Dict[Any, Any]:
    full_out = {}
    for name, sub_config in nested_dict.items():
        if isinstance(sub_config, dict):
            full_out[name] = pd.DataFrame(_nested_dict_into_rows(sub_config))
            # print(f"Added: a dataframe with {name=}")
        else: #recently added
            full_out[name] = {name: sub_config}
            # print(f"Added: a key with {name=}")
    return full_out
```

Main

1 Preconditions

packages, input file name and loading data in

In [ ]:
```
# packages used in code:
import pandas as pd
import numpy as np
import yaml
# used for graphics:
import seaborn as sns
import matplotlib.pyplot as plt
# possibly externalised self-written script
# from function_flatten_yml import yaml_to_dict_of_dfs

# display setting for decimals in dataframes:
pd.set_option('display.float_format', '{:.3f}'.format) # optional

# configuration file load-in:
filepath = 'tg_config.yml'

with open(filepath, "r") as file:
    data_dict = yaml.safe_load(file) # contains a list of dicts
    print(f'The content of the YAML-file {filepath} \n is now accessible as "data_d
    data_items = yaml_to_dict_of_dfs(data_dict) # contains a dict of dataframes
    print(f'The content of the YAML-file {filepath} \n is now accessible as "data_i
```

```
The content of the YAML-file tg_config.yml
 is now accessible as "data_dict"
 containing a list of dicts

The content of the YAML-file tg_config.yml
 is now accessible as "data_items"
 containing a dict of dataframes and keys
```

2.1 Initialising the dataframes with column headers

(Column headers are not given in the input yml file) Number of buildings is not given, but calculated by division of total building area by area per building.

In [ ]:
```
data_items['Material_use'].columns = ['construction_sys', 'building_part', 'materia
data_items['Material_infos_co2coeff'].columns = ['material','co2coeff']
data_items['Material_infos_biom_fraction'].columns = ['material', 'biom_fraction']

# future to do: should warn if mutliple entries with same material name, as only th

number_of_buildings = round(data_dict['area_parameters']['total'] / data_dict['area
```

2.2 Adding given relevant info to main dataframe

CO2 coefficients and information about the fraction of biomass in given materials are added to the dataframe, being matched by material names.

In [ ]:
```
full_df = (data_items['Material_use']
           .merge(data_items['Material_infos_co2coeff'], on='material', how='left'
           .merge(data_items['Material_infos_biom_fraction'], on='material', how='
```

3 Calculations

Adding:

- total_masses to full_df (optional, but completes the logic of the dataset)
- co2e_per_building and adding total_emissions to full_df
- c_storage_per_building and adding c_storage_per_building to_full df

In [ ]:

```
# emissions
full_df["co2e_per_building"] = (
          full_df['mass_per_building'] # kg / building
        * full_df['co2coeff'] # kg CO2equ / kg material
        / 1000) # t CO2equ / kg CO2equ

full_df["total_emissions"] = (
          number_of_buildings # number of buildings
        * full_df['co2e_per_building']) # kg CO2equ / building

# storage
full_df["c_storage_per_building"] = (
          full_df['mass_per_building'] # kg / building
        * full_df['biom_fraction'] # kg biomass / kg material
        * data_dict['Material_infos_carbon_ratio'] # unit C / unit biomass
        / 1000) # t C / kg C

full_df["total_c_storage"] = (
          number_of_buildings # number of buildings
        * full_df['c_storage_per_building']) # kg CO2equ / building

# total masses (just for completion)
full_df["total_masses"] = (
          full_df['mass_per_building'] # kg / building
        * number_of_buildings) # number of buildings
```

In [ ]:

```
# Taking a peek:
full_df.head()
```

Out[ ]:

	construction_sys	building_part	material	mass_per_building	co2coeff	biom_1
0	sys1_timber_frame	wall	gypsum_fibreboard	14562.544	1.970	
1	sys1_timber_frame	wall	osb	17470.446	0.285	
2	sys1_timber_frame	wall	wooden_frame_6-24	43322.154	0.078	
3	sys1_timber_frame	wall	cellulose_insulation	17841.047	0.240	
4	sys1_timber_frame	wall	wood_fibre_insulation_board	23640.381	0.724	

4 DataFrame Grouping and Export

Grouping the Dataframes to be summed up to construction systems (and building parts).

In [ ]:

```
cols = ['mass_per_building', 'co2e_per_building', 'c_storage_per_building', 'total_
sum_aggr = {f'{col}': "sum" for col in cols}
df_out = full_df.loc[:,['construction_sys', 'building_part', 'mass_per_building', '

df_out_systems_parts = df_out.groupby(['construction_sys', 'building_part']).agg(su
df_out_systems = df_out.groupby(['construction_sys']).agg(sum_aggr).reset_index()

# for easy debugging:
# data_check = full_df[['construction_sys', 'building_part' ,'mass_per_building', '
# print(data_check)
```

Out[ ]:

	construction_sys	mass_per_building	co2e_per_building	c_storage_per_building	total_m
0	sys1_timber_frame	641273.643	293.631	19666.998	10709269
1	sys2_mass_timber	948118.051	242.442	37852.899	15833571
2	sys3_light_weight_timber	809595.341	235.071	30293.857	13520242
3	sys4_reinforced_concrete	6309512.210	2683.130	0.000	105368853
4	sys5_brick_based	5042379.451	2071.941	0.000	84207736

In the following, the main dataframes of interest are being written including masses, emissions and storage:

- Values per material. Full Dataset, no sums being aggregated.
- Total values per construction systems
- Values per construction systems split into the categories of building parts (wall, ceiling, roof)

Saving to an subfolder 'export'.

In [ ]:

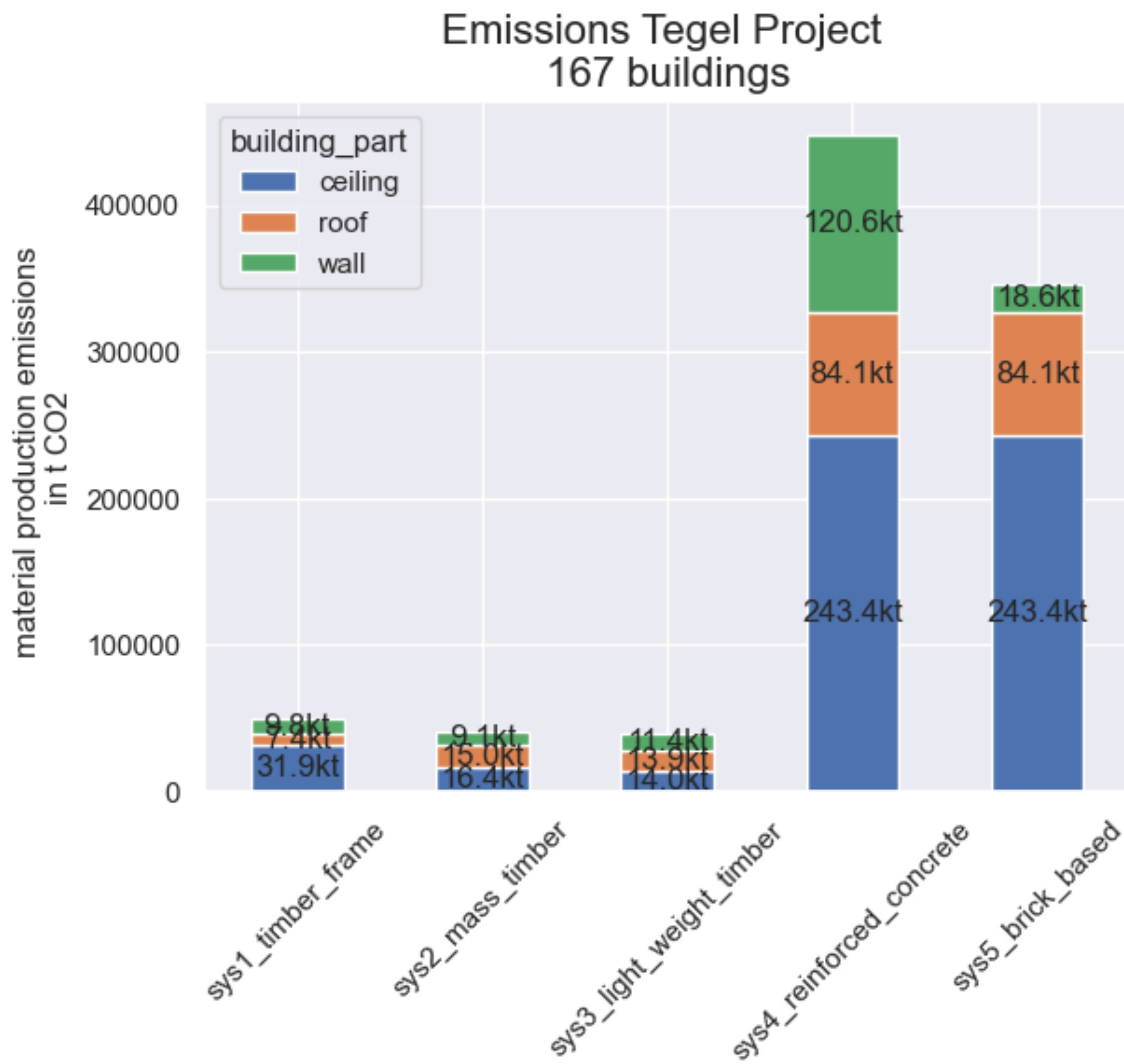
```
df_out.to_csv(r'./export/tg_subtotals.csv', sep=',', encoding='utf-8', index=False,
df_out_systems.to_csv(r'./export/tg_total_per_csys.csv', sep=',', encoding='utf-8',
df_out_systems_parts.to_csv(r'./export/tg_per_csys_and_part.csv', sep=',', encoding

# for tab-seperated use sep= '\t'
# na_rep= can be changed as needed (e.g. to NA, NaN, 0)
```

5 Graphs and Graph Export

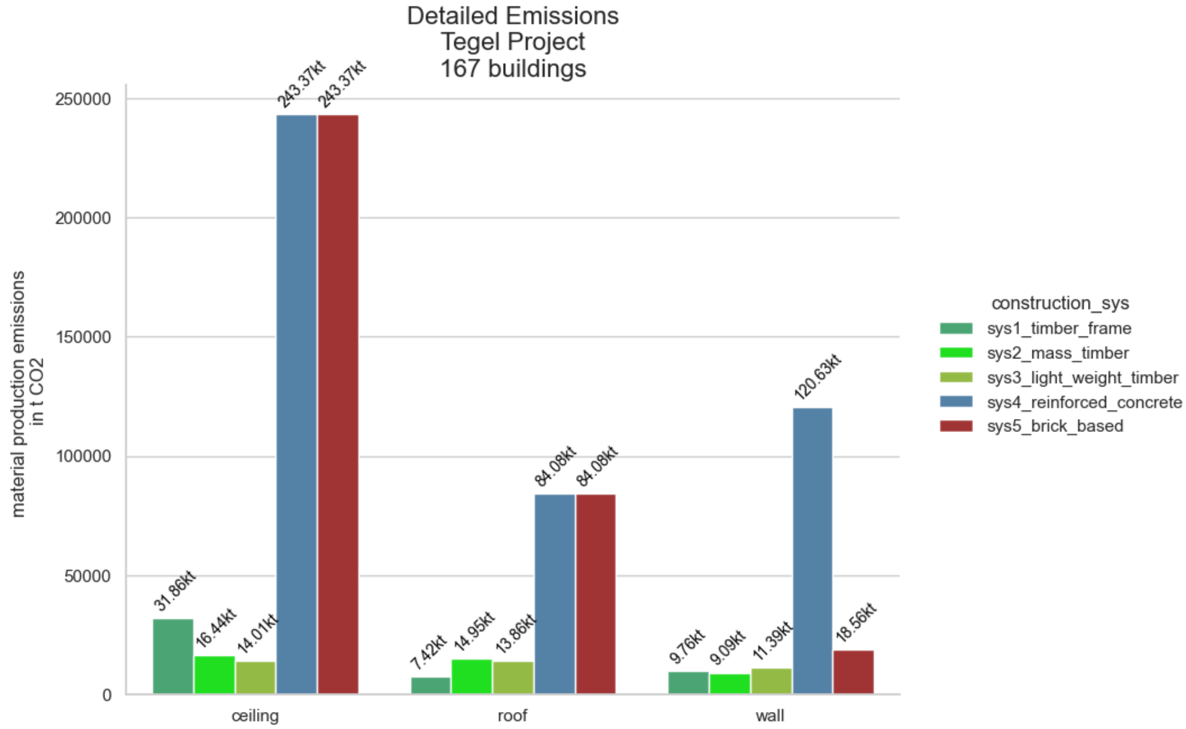
This section seeks to visualise the emissions:

In [ ]:

```
df_graph1 = (df_out_systems_parts
             .groupby(['construction_sys', 'building_part']).agg({'total_emissions':
             .reset_index()
             .pivot(index='construction_sys', columns='building_part', values='total
             )

# from matplotlib import cm
# cmap = cm.get_cmap('Spectral') # example of a colormap that could be used
ax = df_graph1.plot(kind= 'bar', stacked=True,
        # colormap=cmap
        grid=True
)

for c in ax.containers: # value labels
    # Optional: if the segment is small or 0, customize the labels
    labels = ['{0:.0f}kt'.format(v.get_height()/1000) if v.get_height() > 0 else ''
    # remove the labels parameter if it's not needed for customized labels
    ax.bar_label(c, labels=labels, label_type='center') #, fmt='%0.2f' could be use

plt.title('Emissions Tegel Project \n167 buildings', fontsize=16) # add overall tit
plt.xlabel('') # add axis titles
plt.ylabel('material production emissions \nin t CO2')
plt.xticks(rotation=45) # rotate x-axis labels
# EXPORT (before printing! ...else the saved graph is empty)

plt.savefig(r'./export/tg_emissions_per_csys_and_part.png', bbox_inches='tight', dp

plt.show()
```

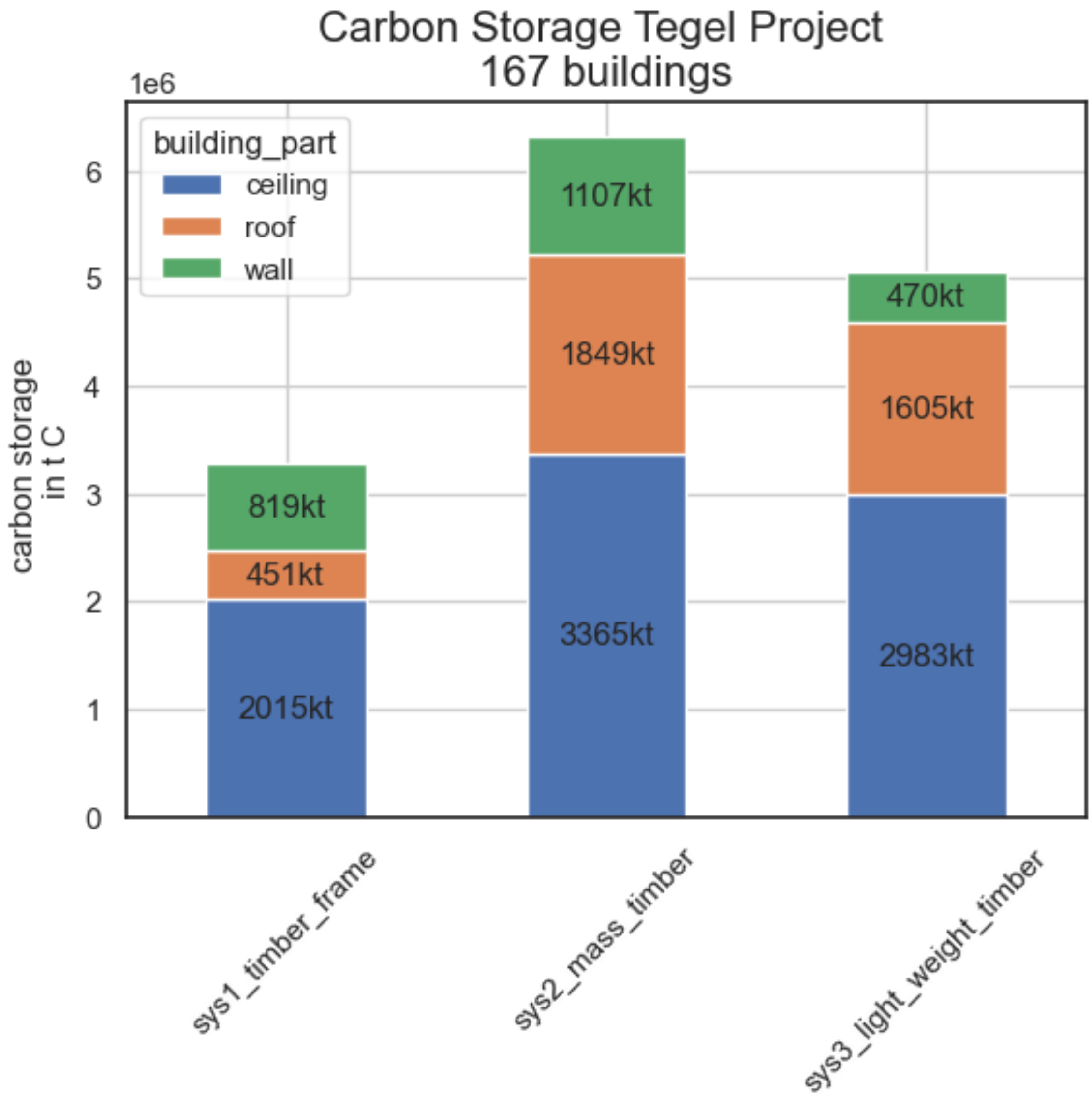

Emissions Tegel Project
167 buildings
building_part
ceiling
roof
wall
material production emissions
in t CO2
400000
300000
200000
100000
0
120.6kt
84.1kt
243.4kt
18.6kt
84.1kt
243.4kt
9.8kt
7.4kt
31.9kt
9.1kt
15.0kt
16.4kt
11.4kt
13.9kt
14.0kt
sys1_timber_frame
sys2_mass_timber
sys3_light_weight_timber
sys4_reinforced_concrete
sys5_brick_based

In [ ]:
```
mypalette = {"sys1_timber_frame":"mediumseagreen",
           "sys2_mass_timber":"lime",
           "sys3_light_weight_timber":"yellowgreen",
           "sys4_reinforced_concrete": "steelblue",
           "sys5_brick_based": "firebrick"}

g =sns.catplot(x='building_part', y='total_emissions', hue='construction_sys', data
    kind='bar',
    palette=mypalette)

g.fig.set_size_inches(12,8)
g.fig.subplots_adjust(top=0.81,right=0.78)

ax = g.facet_axis(0,0)
for p in ax.patches:
    ax.text(p.get_x(),
            p.get_height() + 3000,
           '{0:.2f}kt'.format(p.get_height()/1000),   #Used to format it K represen
            color='black',
            rotation=45,
            size='small')

sns.set(style='darkgrid')
plt.title('Detailed Emissions\nTegel Project\n167 Buildings', fontsize=16) # add ov
plt.xlabel('') # add axis titles
plt.ylabel('material production emissions \nin t CO2')
plt.xticks(rotation='horizontal') # rotate x-axis labels

# EXPORT (before printing! ...else the saved graph is empty)
plt.savefig(r'./export/tg_detailed_emissions_unstacked.png', bbox_inches='tight', d

plt.show()
```



This section seeks to visualise carbon storage:

In [ ]:
```
df_graph2 = (df_out_systems_parts
            .groupby(['construction_sys', 'building_part']).agg({'total_c_storage':
            .reset_index()
            .pivot(index='construction_sys', columns='building_part', values='total
            )
df_graph2_filtered = df_graph2[(df_graph2['ceiling'] != 0) & (df_graph2['roof'] !=

# from matplotlib import cm
# cmap = cm.get_cmap('Spectral') # example of a colormap that could be used

ax = df_graph2_filtered.plot(kind= 'bar', stacked=True,
        label= 'plot_storage_per_csys_and_part',
        # colormap=cmap,
        grid=True)

for c in ax.containers: # value labels
    # Optional: if the segment is small or 0, customize the labels
    labels = ['{0:.0f}kt'.format(v.get_height()/1000) if v.get_height() > 0 else ''
    # remove the labels parameter if it's not needed for customized labels
    ax.bar_label(c, labels=labels, label_type='center') #, fmt='%0.2f' could be use

plt.title('Carbon Storage Tegel Project \n167 buildings', fontsize=16) # add overal
plt.xlabel('') # add axis titles
plt.ylabel('carbon storage \nin t C')
plt.xticks(rotation=45) # rotate x-axis labels

# EXPORT (before printing! ...else the saved graph is empty)
plt.savefig(r'./export/tg_storage_per_csys_and_part.png', bbox_inches='tight', dpi=

plt.show()
```


Carbon Storage Tegel Project
167 buildings
1e6
building_part
ceiling
roof
wall
carbon storage
in t C
0
1
2
3
4
5
6
2015kt
451kt
819kt
3365kt
1849kt
1107kt
2983kt
1605kt
470kt
sys1_timber_frame
sys2_mass_timber
sys3_light_weight_timber