Exploratory script for carbon cycle analysis

on the global scenario dataset.

from:

Churkina, G., Organschi, A., Reyer, C.P.O. et al. Buildings as a global carbon sink. Nat Sustain 3, 269–276 (2020).
https://doi.org/10.1038/s41893-019-0462-4

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

How to:

The Main part of this Jupyther Notebook has the following sections:

1. Preconditions and Loading Input
2. Initialisation and Matching Constants
3. Actual Calculatins
4. Data Aggregation and Export
5. Plotting and Graph Export

Input: Place the input file ('gs_config.yml' by default) in the same directory as the code.

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. Certain inputs are given in error dimensions which must match by string.

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 print-based debugging if needed.

calculations: The whole idea consists of calculating everything on the very level that is given for material usages in the input file. 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 (and error dimension), the following is being calculated:

mass_per_captia = mass_per_area * area_per_capita
co2_per_captia = mass_per_captia * co2_coeff
c_storage_per_captia = mass_per_captia * carbon_ratio

The total amounts are being calculated in transformation scenarios by multiplication with the fraction of population growth attributed to the respective construction system.

Sidenotes:

- *consistent naming:* corresponding specific files named with prefix 'gs_'
- In the requirements.txt file of the local environment, all sub-packages and their version specifications are listed. Also VS-Code extensions are listed.
- future to do:
  - should warn if mutliple entries with same material name, as only the first one will be imported (dicts enforce unique keys)
  - grouping for graph and table exports might be given as promts or input(?)
  - coefficients: min mean max are given for overall emissions; maybe better rerun with transport coeff, total coeff, etc.?
  - sometimes preprocessing of materials depending on category can be interesting - it also influences emissions based on e.g. transport
  - Packeage Streamlit was recommended to be suited to one day melt this into an appliation.

Externalised code blocks:

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 [ ]:
```python
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)) # con
            # print(f"Added: a dataframe with {name=}")
        else:
            full_out[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 [ ]:
```
# dependencies 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 = 'gs_config.yml'

with open(filepath, "r") as file:
    data_dict = yaml.safe_load(file) # contains a list of dicts
    data_items = yaml_to_dict_of_dfs(data_dict) # converts deep dicts to dataframes
    print(data_items['Name'], 'Dataset', '\nfrom YAML-file:', filepath, '\n')
    # print(f'The content of the YAML-file is accessible as "data_dict"\n containin
    # print(f'The content of the YAML-file is accessible as "data_items"\n containi
    print('Data load-in done.')
```

```
Global Transformation Scenarios Dataset
from YAML-file: gs_config.yml

Data load-in done.
```

2.1 Initialising the dataframes with column headers

(Column headers are not given in the input yml file)

In [ ]:
```
regrowing_scenario = data_dict['Regrowing_scenarios']

df_material_use = data_items['Material_intensities']
df_material_use.columns = ['building_type', 'building_part', 'construction_sys', 'm
# df_material_use = data_items['Material_intensities'].rename(columns = ['building_

df_living_standard = pd.DataFrame.from_dict((data_dict)['Living_standard']).T.reset
        columns={'index': 'building_type',
                 'min':'area_per_capita_min',
                 'mean':'area_per_capita_mean',
                 'max':'area_per_capita_max'})

df_coeff = pd.DataFrame.from_dict((data_dict)['Material_infos_co2coeff']).T.reset_i
        columns={'index': 'material',
                 'max': 'co2_coeff_max',
                 'min': 'co2_coeff_min',
                 'mean': 'co2_coeff_mean'})

df_carbon_ratios = pd.DataFrame.from_dict(data_dict['Material_infos_carbon']).T.res
        columns={'index': 'material'}).drop(columns= 'category').replace({None: 0})

error_vals = data_items['Error_dimensions'] # categoricals reused all over the data

print('Initialised dataframes with headers.')
```

```
Initialised dataframes with headers.
```

2.2 Adding given relevant info to main dataframe

In [ ]:
```
full_df = (df_material_use
           .merge(df_living_standard, on='building_type', how='left')
           .merge(df_coeff, on='material', how='left')
           .merge(df_carbon_ratios, on='material', how='left'))
```

In [ ]:
```
for pr_name, pr_value in regrowing_scenario.items(): # sys 3 is timber based, sys 4
  # pr_condition = ['sys3']
  # not_pr_condition = ['sys4']

  full_df[f'{pr_name}_prvalue'] = None
  # full_df.loc[full_df.construction_sys.isin(pr_condition),f'{pr_name}_prvalue'] =
  # full_df.loc[full_df.construction_sys.isin(not_pr_condition),f'{pr_name}_prvalue

  full_df.loc[full_df['construction_sys'] == 'sys3',f'{pr_name}_prvalue'] = pr_valu
  full_df.loc[full_df['construction_sys'] == 'sys4',f'{pr_name}_prvalue'] = (1-pr_v
```

In [ ]:
```
print('Main dataframe established:\nConstants are matched with material usage.')
```

```
Main dataframe established:
Constants are matched with material usage.
```

3 Calculations

3.1 Basic Calculations

Scenario independent calculations. Adding:

- mass_per_captia to full_df
- co2_per_captia to full_df
- carbon_per_captia to full df

In [ ]:
```
# adding mass_per_captia to full df
for column in error_vals:
    full_df[f"mass_per_captia_{column}"] = (
          full_df[f'area_per_capita_{column}']
        * full_df['mass_per_area'])

# adding co2_per_captia to full df
for column in error_vals:
    full_df[f"co2_per_captia_{column}"] = (
          full_df[f'mass_per_captia_{column}']
        * (full_df[f'co2_coeff_{column}']/1000)) #given in ton co2 equ per ton, nee

# adding carbon_per_captia to full df
for column in error_vals:
    full_df[f"c_storage_per_captia_{column}"] = (
          full_df[f'mass_per_captia_{column}']
        * full_df['carbon_ratio'])
```

3.2 Scenario Calculations: Emissions

For the scenarios, only a part of the dataset is going to be compared: Construction system 3 (regrowing, timber-based) and construction system 4 (conventional, reinforced concrete-based). The two other construction systems are fractions of construction system 4 and will be omitted (either by dropping those rows with NA or those with sys1 & sys2)

In [ ]:
```
full_df1 = full_df.dropna().reset_index(drop=True)
full_df1
```

Out[ ]:

	building_type	building_part	construction_sys	material	mass_per_area	area_per_capita_m
0	residential	primary_structure	sys3	timber	193.610	8.5(
1	residential	primary_structure	sys4	steel	46.190	8.5(
2	residential	primary_structure	sys4	concrete	251.920	8.5(
3	residential	enclosure	sys3	timber	27.680	8.5(
4	residential	enclosure	sys3	wood_fiber	10.020	8.5(
5	residential	enclosure	sys4	steel	5.640	8.5(
6	residential	enclosure	sys4	fiberglass	0.860	8.5(
7	residential	enclosure	sys4	gypsum	6.000	8.5(
8	residential	enclosure	sys4	polystyrene	0.770	8.5(
9	commercial	primary_structure	sys3	timber	209.400	0.7(
10	commercial	primary_structure	sys4	steel	82.700	0.7(
11	commercial	primary_structure	sys4	concrete	608.440	0.7(
12	commercial	enclosure	sys3	timber	59.930	0.7(
13	commercial	enclosure	sys3	wood_fiber	20.770	0.7(
14	commercial	enclosure	sys4	steel	10.470	0.7(
15	commercial	enclosure	sys4	fiberglass	1.820	0.7(
16	commercial	enclosure	sys4	gypsum	13.230	0.7(
17	commercial	enclosure	sys4	polystyrene	1.590	0.7(

18 rows × 25 columns

In [ ]:
```
# this block does two things: add the emissions per scenario to the main df and
# extract the data of total emissions per scenario on the go

df_graph_emissions = pd.DataFrame(columns=['scenario', 'error_dim', 'total_emission

for pr_name in regrowing_scenario.keys():
  for column in error_vals:
      full_df1[f"{pr_name}_co2_{column}"] = (
        full_df1[f'co2_per_captia_{column}']
        * full_df1[f'{pr_name}_prvalue']
        * data_items['Population_diff'])
      entry =pd.DataFrame({
        'scenario': [pr_name],
        'error_dim': [column],
        'total_emissions': [full_df1[f"{pr_name}_co2_{column}"].sum()]})
      df_graph_emissions = pd.concat([df_graph_emissions, entry], ignore_index=True
df_graph_emissions
```

Out[ ]:

	scenario	error_dim	total_emissions
0	b_as_usual	min	2.141
1	b_as_usual	mean	14.316
2	b_as_usual	max	67.000
3	scenario10%	min	2.033
4	scenario10%	mean	13.650
5	scenario10%	max	63.596
6	scenario50%	min	1.577
7	scenario50%	mean	10.849
8	scenario50%	max	49.263
9	scenario90%	min	1.121
10	scenario90%	mean	8.048
11	scenario90%	max	34.931

3.3 Scenario Calculations: Storage

In [ ]:

```
# this block does two things: add the storage per scenario to the main df and
# extract the data of total storage per scenario on the go

df_graph_storage = pd.DataFrame(columns=['scenario', 'error_dim', 'storage'])

for pr_name in regrowing_scenario.keys():
  for column in error_vals:
      full_df1[f"{pr_name}_c_storage_{column}"] = (
        full_df1[f'c_storage_per_captia_{column}'] / 1000 # converting to tons
      * full_df1[f'{pr_name}_prvalue']
      * data_items['Population_diff'])
      entry =pd.DataFrame({
        'scenario': [pr_name],
        'error_dim': [column],
        'storage': [full_df1[f"{pr_name}_c_storage_{column}"].sum()]})
      df_graph_storage = pd.concat([df_graph_storage, entry], ignore_index=True)
df_graph_storage
```

Out[ ]:

	scenario	error_dim	storage
0	b_as_usual	min	0.012
1	b_as_usual	mean	0.043
2	b_as_usual	max	0.113
3	scenario10%	min	0.249
4	scenario10%	mean	0.855
5	scenario10%	max	2.256
6	scenario50%	min	1.247
7	scenario50%	mean	4.277
8	scenario50%	max	11.281
9	scenario90%	min	2.245
10	scenario90%	mean	7.699
11	scenario90%	max	20.306

4 DataFrame Grouping and Export

Grouping the Dataframes to be summed up to scenarios.

In [ ]:

```
agg_dict = {f'{scenario}_c_storage_{error}': "sum" for scenario in regrowing_scenar
agg_dict.update({f'{scenario}_co2_{error}': "sum" for scenario in regrowing_scenari

df_out = (full_df1.groupby(['building_part', 'construction_sys'])
        .agg(agg_dict)
        .T.reset_index())

df_out_e = df_out.assign(list_col=lambda df_out: df_out[('index', '')].str.split('_
                    scenario=lambda df_out: df_out.list_col.str[0],
                    error=lambda df_out: df_out.list_col.str[1]
                    ).drop(columns=[('list_col',''), ('index', '')])

df_out_c = df_out.assign(list_col=lambda df_out: df_out[('index', '')].str.split('_
                    scenario=lambda df_out: df_out.list_col.str[0],
                    error=lambda df_out: df_out.list_col.str[1]
                    ).drop(columns=[('list_col',''), ('index', '')])
```

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

- Values per material usage. Full dataset, no sums being aggregated. Scenarios and error dimensions being represented horizontally.
- Totals for emissions and storage.
- ~~Values per construction systems.~~
- ~~Values per construction systems per building part (enclosure, primary_structure).~~
- ~~Values per construction systems per building part (enclosure, primary_structure) and building type (residential, commercial).~~

Saving to an subfolder 'export'.

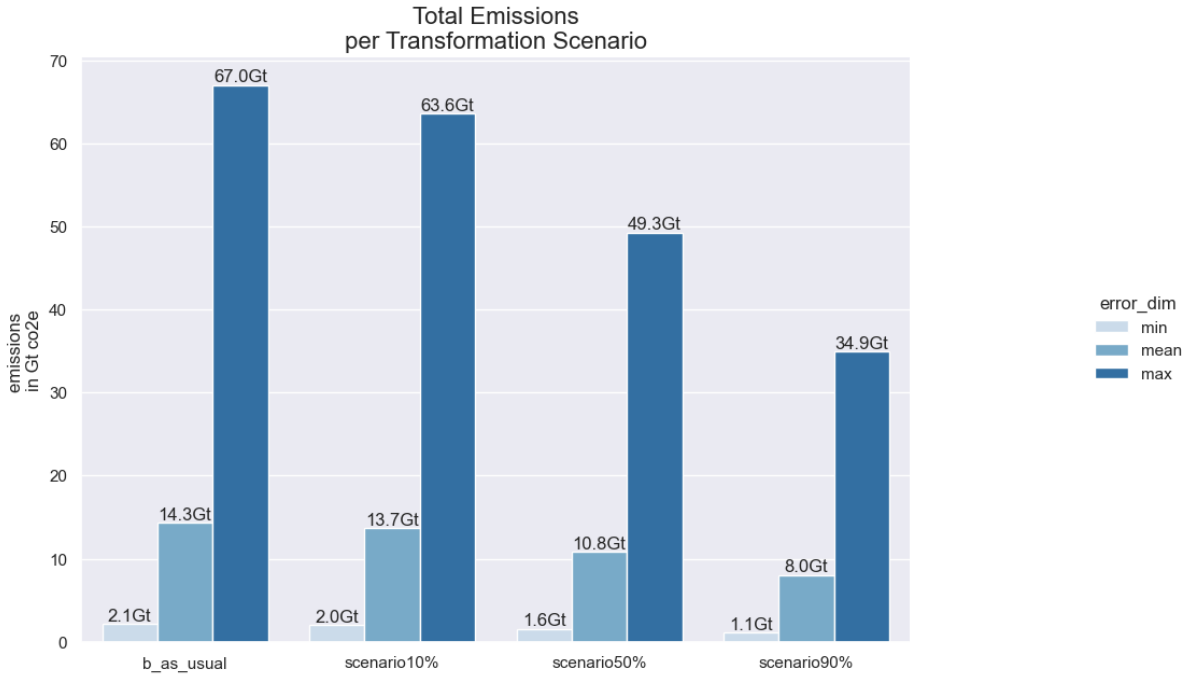
In [ ]:
```
full_df1.to_csv(r'./export/gs_full_dataframe.csv', sep=',', encoding='utf-8', index
df_out_c.to_csv(r'./export/gs_total_cstorage.csv', sep=',', encoding='utf-8', index
df_out_e.to_csv(r'./export/gs_total_emissions.csv', sep=',', encoding='utf-8', inde

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

5 Graphs and Graph Export

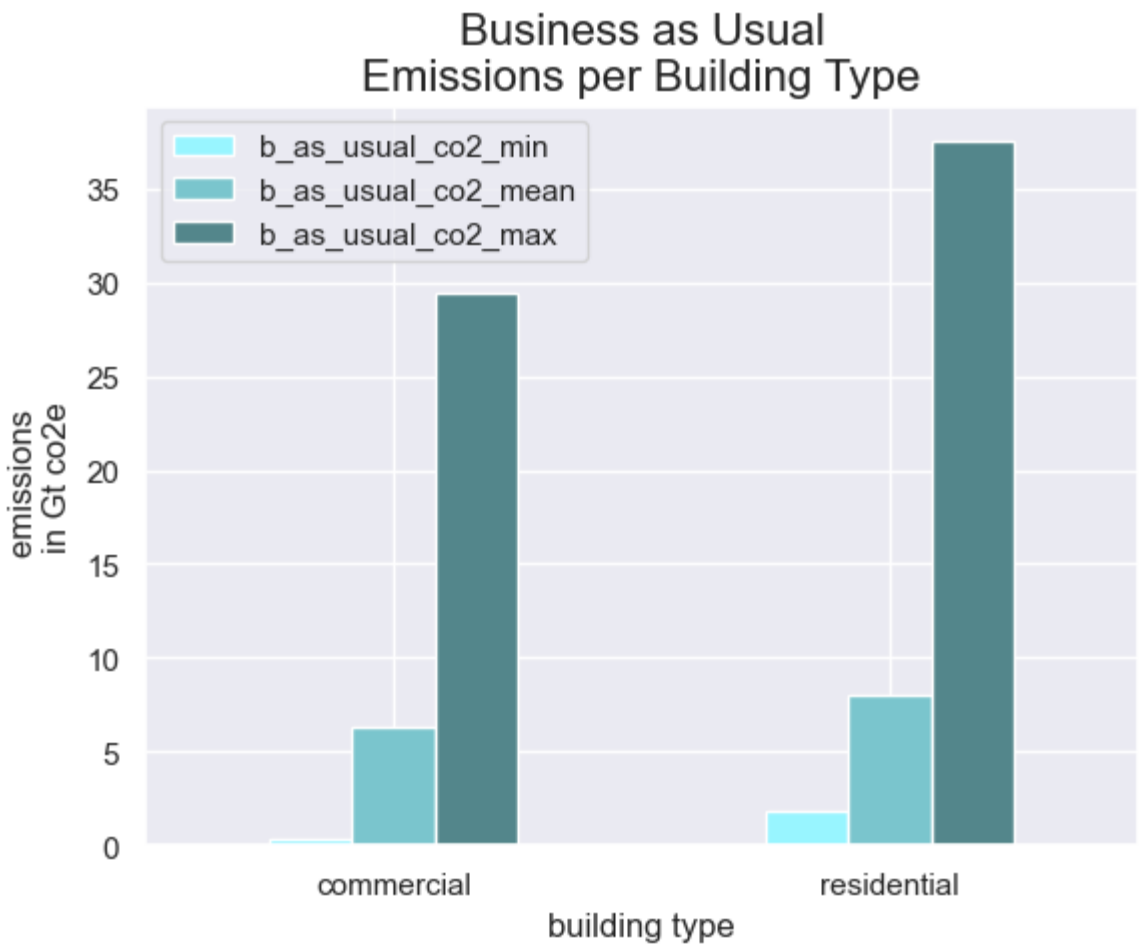
5.1 Emission Plots

This section seeks to visualise the emissions:

In [ ]:
```
g =sns.catplot(x='scenario', y='total_emissions', hue='error_dim', data=df_graph_em
g.fig.set_size_inches(12,8)
g.fig.subplots_adjust(top=0.81,right=0.78)
ax = g.facet_axis(0, 0)
for c in ax.containers:
    labels = [f'{(v.get_height()):.1f}Gt' for v in c]
    ax.bar_label(c, labels=labels, label_type='edge', rotation=0)

sns.set(style='darkgrid')
plt.title('Total Emissions\nper Transformation Scenario', fontsize=16) # add overal
plt.xlabel('') # add axis titles
plt.ylabel('emissions \nin Gt co2e')
plt.xticks(rotation=0) # rotate x-axis labels

# EXPORT (before printing! ...else the saved graph is empty)
plt.savefig(r'./export/gs_emissions_unstacked.png', bbox_inches='tight', dpi=300)

plt.show()

# future to do: visualise share of timber in bars
```



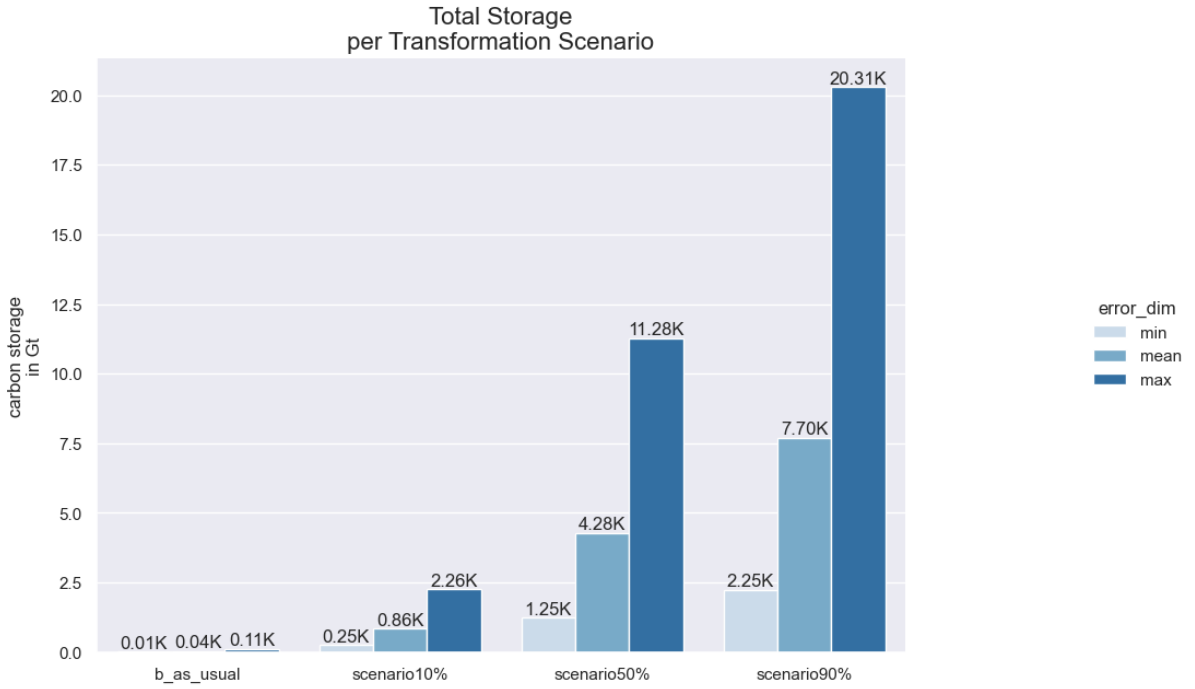
In [ ]:
```
# graph
g = (full_df1.groupby('building_type')[[f'b_as_usual_co2_{col}' for col in error_va
    .plot.bar(stacked=False, grid=True, color=['#98f5ff', '#7ac5cd', '#53868b'])
)
sns.set(style='darkgrid')
# df_graph_only.set_index('construction_system').plot(kind='bar', stacked=True) # ,
plt.title('Business as Usual\nEmissions per Building Type', fontsize=16) # add over
plt.xlabel('building type') # add axis titles
plt.ylabel('emissions \nin Gt co2e')
plt.xticks(rotation=0) # rotate x-axis labels

# EXPORT (before printing! ...else the saved graph is empty)
plt.savefig(r'./export/gs_emissions_b-as-usual_building-types.png', bbox_inches='ti

plt.show()
```



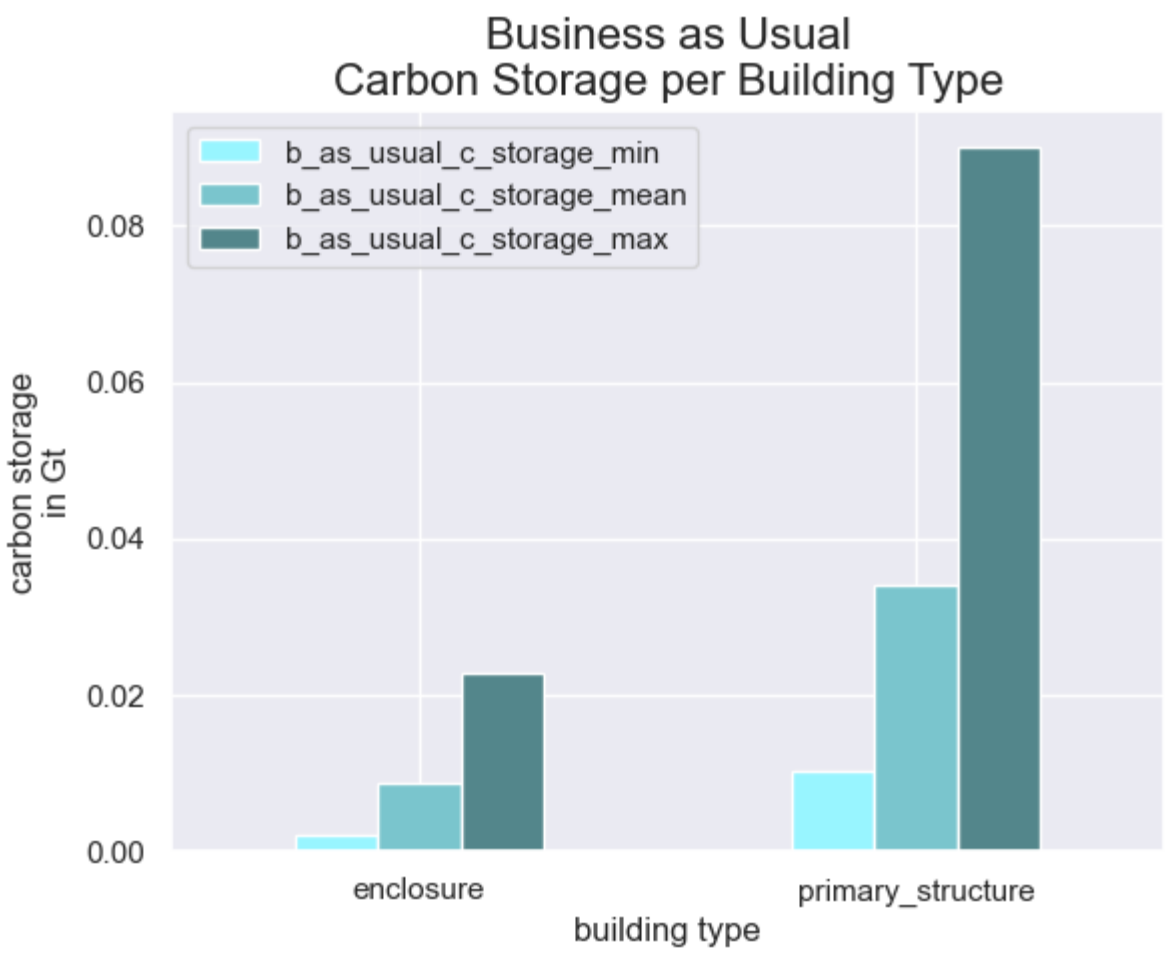
5.2 Carbon Storage Plots

This section seeks to visualise the storage:

In [ ]:
```
# graph
g =sns.catplot(x='scenario', y='storage', hue='error_dim', data=df_graph_storage, k
g.fig.set_size_inches(12,8)
g.fig.subplots_adjust(top=0.81,right=0.78)

ax = g.facet_axis(0, 0)
for c in ax.containers:
    labels = [f'{(v.get_height()):.2f}K' for v in c]
    ax.bar_label(c, labels=labels, label_type='edge', rotation=0)

sns.set(style='darkgrid')
plt.title('Total Storage\nper Transformation Scenario', fontsize=16) # add overall
plt.xlabel('') # add axis titles
plt.ylabel('carbon storage\nin Gt') # add axis titles
plt.xticks(rotation=0) # rotate x-axis labels

# EXPORT (before printing! ...else the saved graph is empty)
plt.savefig(r'./export/gs_cstorage_unstacked.png', bbox_inches='tight', dpi=300)

plt.show()
```



In [ ]:
```
# graph
g= (full_df1.groupby('building_part')[[f'b_as_usual_c_storage_{col}' for col in err
    .plot.bar(stacked=False, grid=True, color=['#98f5ff', '#7ac5cd', '#53868b'])

)
plt.title('Business as Usual\nCarbon Storage per Building Type', fontsize=16) # add
plt.xlabel('building type') # add axis titles
plt.ylabel('carbon storage\nin Gt') # add axis titles
plt.xticks(rotation=0) # rotate x-axis labels

# EXPORT (before printing! ...else the saved graph is empty)
plt.savefig(r'./export/gs_cstorage_b-as-usual_building-types.png', bbox_inches='tig
plt.show()
```



Taking a peek:

In [ ]:
```
full_df1.iloc[0]
```

Out[ ]:

building_type	residential
building_part	primary_structure
construction_sys	sys3
material	timber
mass_per_area	193.610
area_per_capita_min	8.500
area_per_capita_mean	21.500
area_per_capita_max	56.600
co2_coeff_min	0.200
co2_coeff_mean	0.440
co2_coeff_max	0.720
carbonratio	0.500
b_as_usual_prvalue	0.005
scenario10%_prvalue	0.100
scenario50%_prvalue	0.500
scenario90%_prvalue	0.900
mass_per_captia_min	1645.685
mass_per_captia_mean	4162.615
mass_per_captia_max	10958.326
co2_per_captia_min	0.329
co2_per_captia_mean	1.832
co2_per_captia_max	7.890
c_storage_per_captia_min	822.843
c_storage_per_captia_mean	2081.308
c_storage_per_captia_max	5479.163
b_as_usual_co2_min	0.004
b_as_usual_co2_mean	0.021
b_as_usual_co2_max	0.091
scenario10%_co2_min	0.076
scenario10%_co2_mean	0.421
scenario10%_co2_max	1.815
scenario50%_co2_min	0.379
scenario50%_co2_mean	2.106
scenario50%_co2_max	9.073
scenario90%_co2_min	0.681
scenario90%_co2_mean	3.791
scenario90%_co2_max	16.332
b_as_usual_c_storage_min	0.009
b_as_usual_c_storage_mean	0.024
b_as_usual_c_storage_max	0.063
scenario10%_c_storage_min	0.189
scenario10%_c_storage_mean	0.479
scenario10%_c_storage_max	1.260
scenario50%_c_storage_min	0.946
scenario50%_c_storage_mean	2.394
scenario50%_c_storage_max	6.301
scenario90%_c_storage_min	1.703
scenario90%_c_storage_mean	4.308
scenario90%_c_storage_max	11.342

Name: 0, dtype: object