Problem set 2: Hands on with sustainability models

This problem set will have you work through several sustainability models. To turn this in, please submit a PDF of your completed notebook and upload to the assignment on canvas. There are lots of ways to convert a completed Jupyter Notebook to PDF, including the Export feature in VS Code in the "…" options menu above this notebook. The most robust way is to use Quarto (https://quarto.org/docs/getting-started.html), which I had you install as a VS Code Extension. The easiest way to do that is open up a Command Prompt/Terminal in the folder where your notebook is and type `quarto render problem_set_2.ipynb --to pdf`. This will create a PDF in the same folder as your notebook.

Required imports

```
import os, sys
from IPython.display import display, HTML
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import dicelib  # https://github.com/mptouzel/PyDICE

import ipywidgets as widgets  # interactive display

plt.style.use(
    "https://raw.githubusercontent.com/ClimateMatchAcademy/course-content/main/cma.mplstyl
)

%matplotlib inline
sns.set_style("ticks", {"axes.grid": False})
params = {"lines.linewidth": "3"}
plt.rcParams.update(params)
```

```
display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
<IPython.core.display.HTML object>
```

Helper Functions

```
def plot_future_returns(gamma, random_seed):
    fig, ax = plt.subplots(1, 2, figsize=(8, 4))
    np.random.seed(random_seed)
    undiscounted_utility_time_series = np.random.rand(time_steps)
    ax[0].plot(undiscounted_utility_time_series)

    discounted_utility_time_series = undiscounted_utility_time_series * np.power(
        gamma, np.arange(time_steps)
    )
    ax[0].plot(discounted_utility_time_series)

    cumulsum_discounted_utility_time_series = np.cumsum(discounted_utility_time_series)
    ax[1].plot(
        cumulsum_discounted_utility_time_series * (1 - gamma),
        color="C1",
        label=r"discounted on $1/(1-\gamma)=$"
        + "\n"
        + r"$"
        + str(round(1 / (1 - gamma)))
        + "$-step horizon",
    )
    cumulsum_undiscounted_utility_time_series = np.cumsum(
        undiscounted_utility_time_series
    )
    ax[1].plot(
        cumulsum_undiscounted_utility_time_series
        / cumulsum_undiscounted_utility_time_series[-1],
        label="undiscounted",
        color="C0",
    )
    ax[1].axvline(1 / (1 - gamma), ls="--", color="k")

    ax[0].set_ylabel("utility at step t")
    ax[0].set_xlim(0, time_steps)
    ax[0].set_xlabel("time steps into the future")
    ax[1].legend(frameon=False)
    ax[1].set_ylabel("future return (normalized)")
    ax[1].set_xlabel("time steps into the future")
    ax[1].set_xlim(0, time_steps)
    fig.tight_layout()
```

Question 1

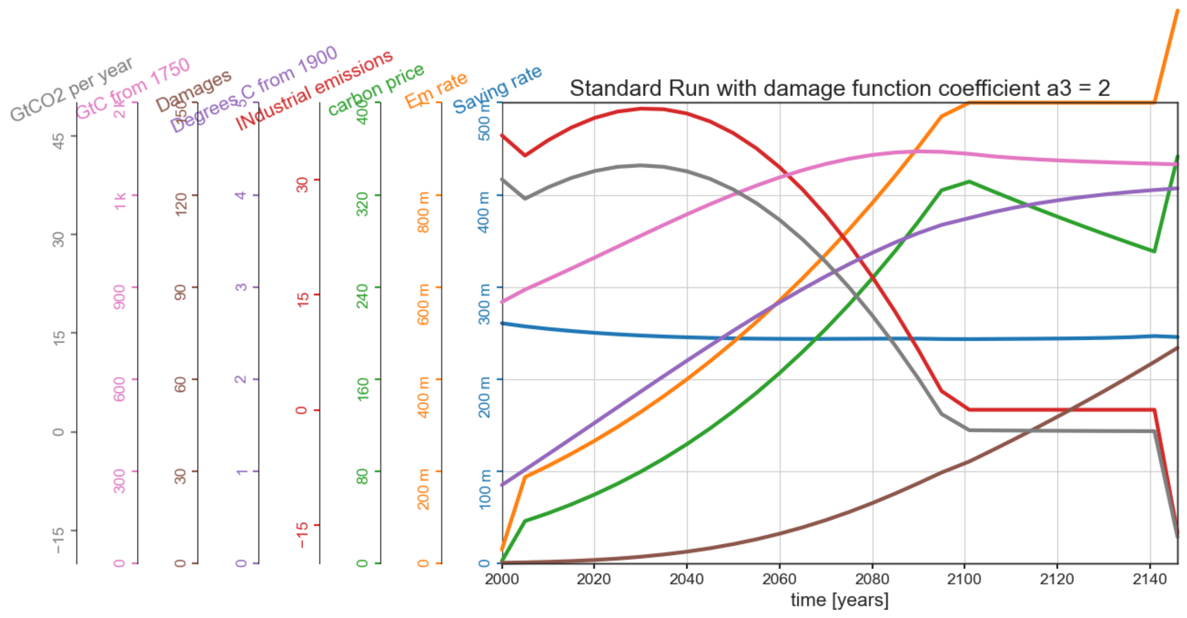
Write a for loop that tests the DICE model to see how the total damages in 2100 change when there is a value of 2 versus 3 for the damage function coefficient. Output a figure for each of the two values and then report out the actual value of total damage

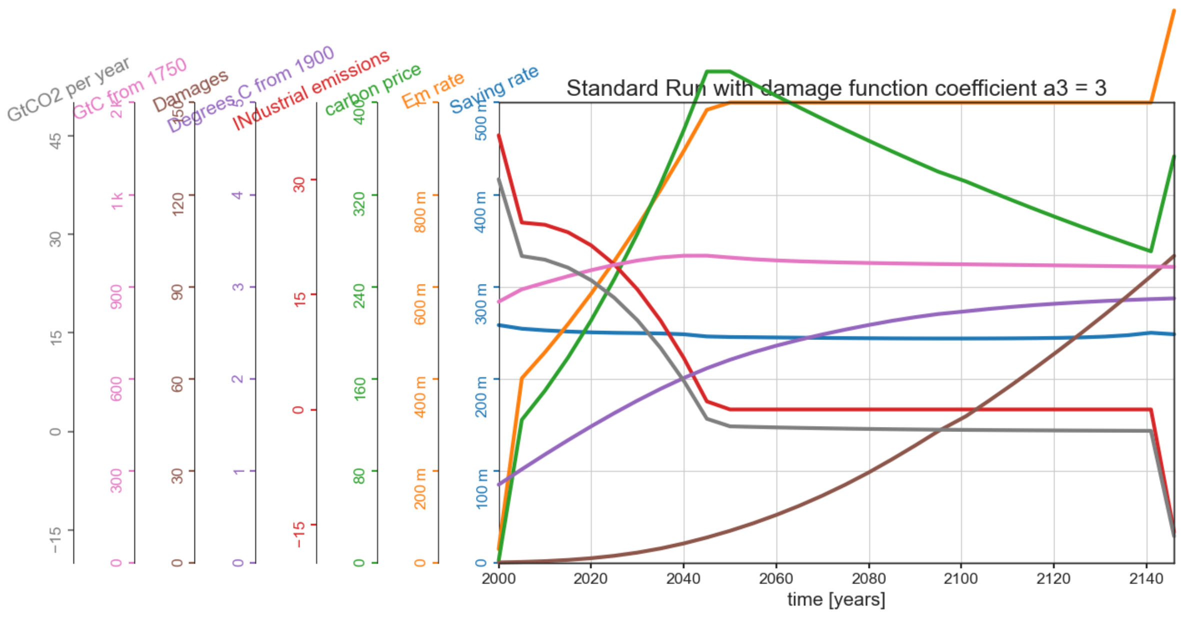
Reminder on some key python terms:

To iterate ver a list, you use a for loop:

```
for i in [2,3]:
    # Create a new DICE model object
    dice_std = dicelib.DICE()
    dice_std.init_parameters(a3 = i) #where a3 is the coefficient in the damage function
    dice_std.init_variables()
    controls_start_std, controls_bounds_std = dice_std.get_control_bounds_and_startvalue()
    dice_std.optimize_controls(controls_start_std, controls_bounds_std);
    dice_std.roll_out(dice_std.optimal_controls)
    dice_std.plot_run(f"Standard Run with damage function coefficient a3 = {i}")
    total_damages = dice_std.DAMAGES[-1] #years range from 2000 to 2100
    print("Total damages in 2100 with damage function coefficient {i}:", total_damages)
```

```
/Users/prayashpathak/mambaforge/envs/8222env2/lib/python3.10/site-packages/scipy/optimize/_o
  warnings.warn("Values in x were outside bounds during a "

Optimization terminated successfully    (Exit mode 0)
            Current function value: -4517.318954614702
            Iterations: 95
            Function evaluations: 19230
            Gradient evaluations: 95
Total damages in 2100 with damage function coefficient {i}: 118.14792732303067
Optimization terminated successfully    (Exit mode 0)
            Current function value: -4416.94398539428
            Iterations: 88
            Function evaluations: 17821
            Gradient evaluations: 88
Total damages in 2100 with damage function coefficient {i}: 0.0018714379716670666
```





Answer: Comparing the two different values of damage fucntion coefficient, we see carbon price and em rate reach their peak much earlier in second case.

Actual value of total damage: 118.15 trillions 2010 USD per year (for a3=2), 0.00187 trillions 2010 USD per year (for a3 = 3)

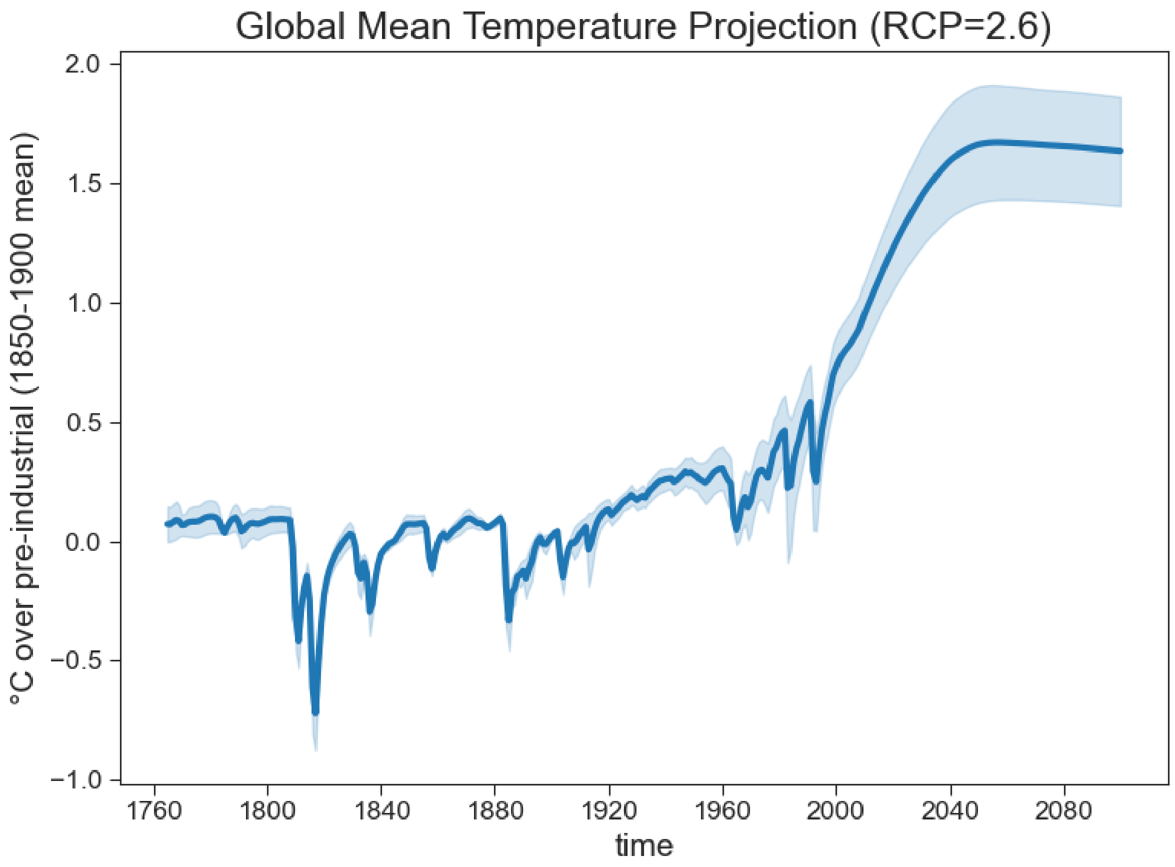
Question 2

Run the MAGICC model for the 4 main RCPs (2.6, 4.5, 7.0, 8.5). Plot the temperature of each pathway on (a single or seperate) graphs. Report out what is the expected temperature in 2100 for each RCP.

```
import pymagicc
import matplotlib.pyplot as plt
```

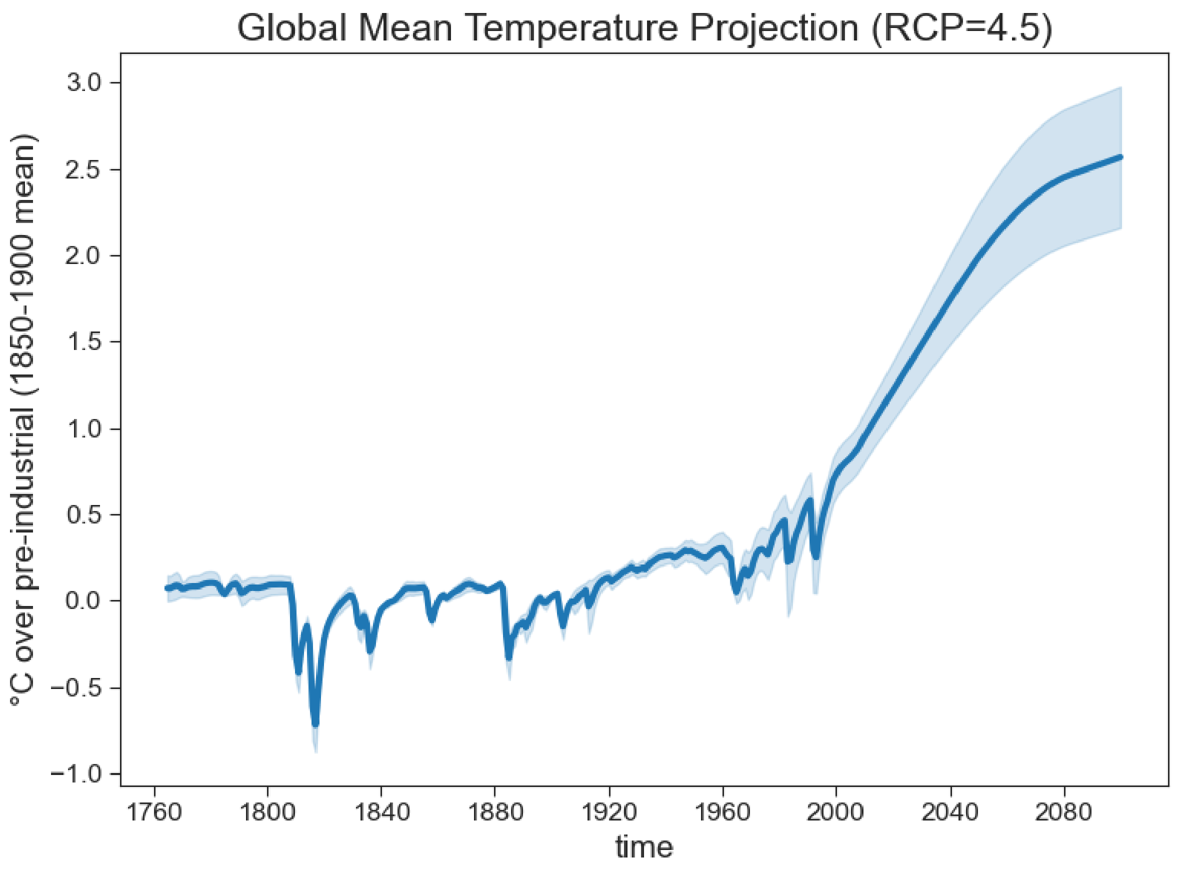
```
#doing it individuallly for the four scenarios first
#RCP 2.6
from pymagicc.scenarios import rcp26
results = pymagicc.run(rcp26)

temperature_rel_to_1850_1900 = (
    results
    .filter(variable="Surface Temperature")
    .relative_to_ref_period_mean(year=range(1850, 1900 + 1))
)
temperature_rel_to_1850_1900.lineplot()
plt.title("Global Mean Temperature Projection (RCP=2.6)")
plt.ylabel("°C over pre-industrial (1850-1900 mean)");
plt.legend().remove()
plt.show()
```

```
/Users/prayashpathak/mambaforge/envs/8222env2/lib/python3.10/site-packages/scmdata/plotting.p

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  ax = sns.lineplot(data=plt_df, **kwargs)
```



```
#RCP 4.5
from pymagicc.scenarios import rcp45
results = pymagicc.run(rcp45)

temperature_rel_to_1850_1900 = (
    results
    .filter(variable="Surface Temperature")
    .relative_to_ref_period_mean(year=range(1850, 1900 + 1))
)
temperature_rel_to_1850_1900.lineplot()
plt.title("Global Mean Temperature Projection (RCP=4.5)")
plt.ylabel("°C over pre-industrial (1850-1900 mean)");
plt.legend().remove()
plt.show()
```

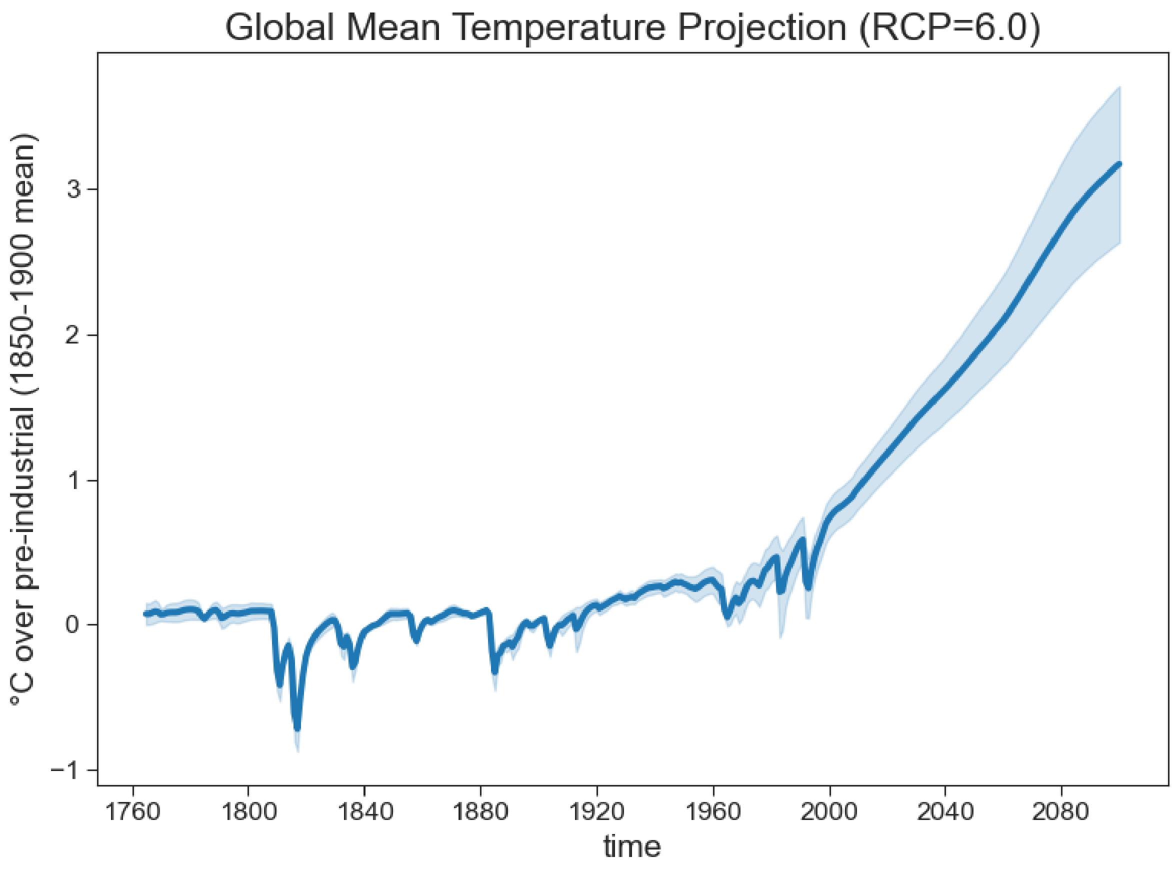
```
/Users/prayashpathak/mambaforge/envs/8222env2/lib/python3.10/site-packages/scmdata/plotting.p

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  ax = sns.lineplot(data=plt_df, **kwargs)
```



```
#RCP 6.0
from pymagicc.scenarios import rcp60
results = pymagicc.run(rcp60)

temperature_rel_to_1850_1900 = (
    results
    .filter(variable="Surface Temperature")
    .relative_to_ref_period_mean(year=range(1850, 1900 + 1))
)
temperature_rel_to_1850_1900.lineplot()
plt.title("Global Mean Temperature Projection (RCP=6.0)")
plt.ylabel("°C over pre-industrial (1850-1900 mean)");
plt.legend().remove()
plt.show()
```

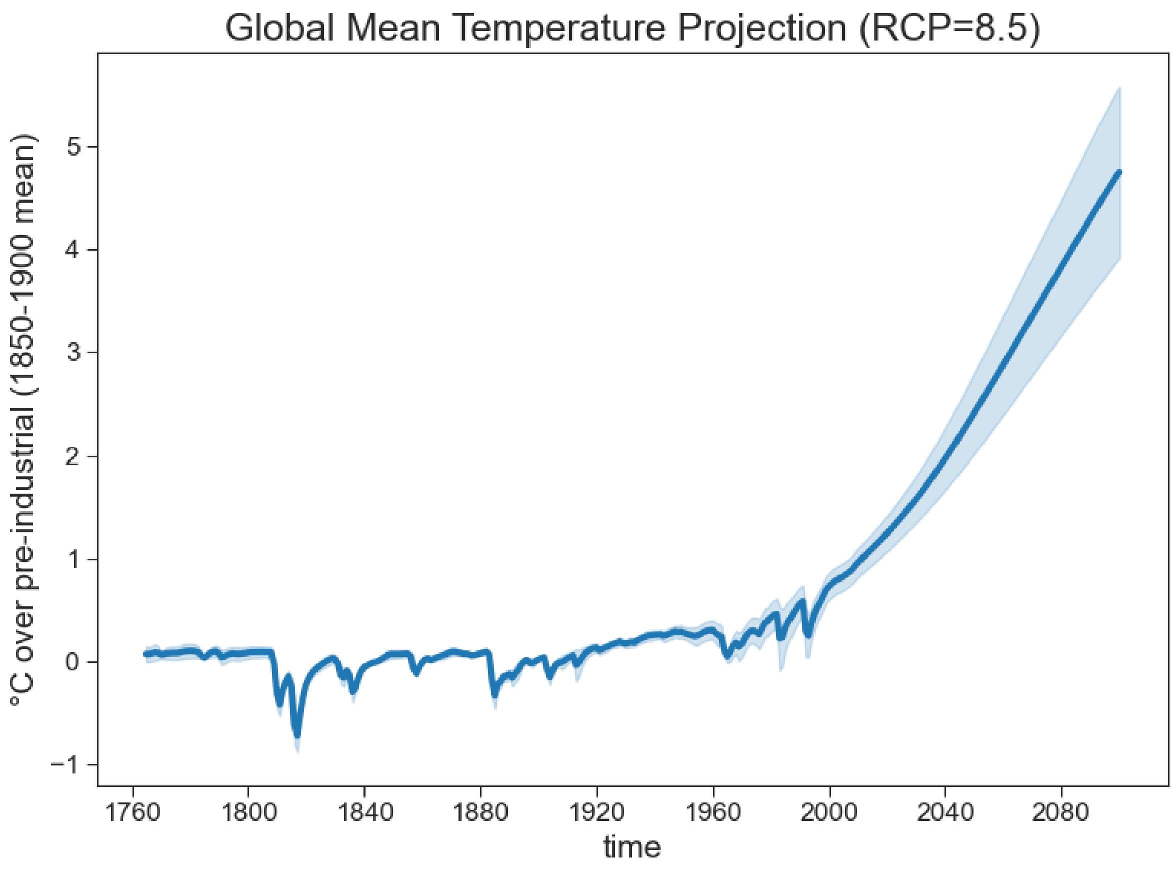
```
/Users/prayashpathak/mambaforge/envs/8222env2/lib/python3.10/site-packages/scmdata/plotting.
```

```
The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  ax = sns.lineplot(data=plt_df, **kwargs)
```



```python
#RCP 8.5
from pymagicc.scenarios import rcp85
results = pymagicc.run(rcp85)

temperature_rel_to_1850_1900 = (
    results
    .filter(variable="Surface Temperature")
    .relative_to_ref_period_mean(year=range(1850, 1900 + 1))
)
temperature_rel_to_1850_1900.lineplot()
plt.title("Global Mean Temperature Projection (RCP=8.5)")
plt.ylabel("°C over pre-industrial (1850-1900 mean)");
plt.legend().remove()
plt.show()
```

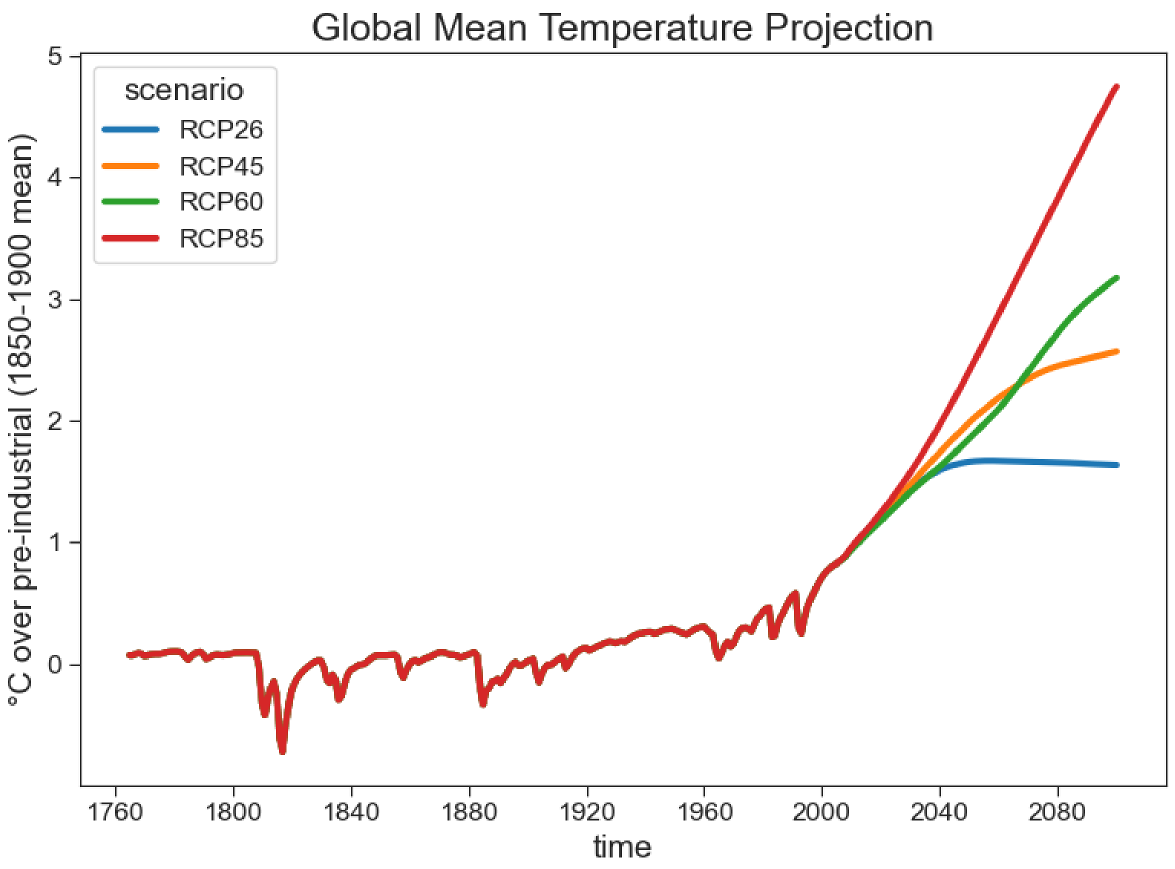
```
/Users/prayashpathak/mambaforge/envs/8222env2/lib/python3.10/site-packages/scmdata/plotting.p

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  ax = sns.lineplot(data=plt_df, **kwargs)
```



```
#Now plotting them all in single graph
import matplotlib.pyplot as plt
import pymagicc
import scmdata
from pymagicc import rcps
```

```python
# Initialize an empty list to store results
results = []

# Run MAGICC model for each RCP
for scen in rcps.groupby("scenario"):
    results.append(pymagicc.run(scen))

# Concatenate the results into a single DataFrame
results = scmdata.run_append(results)

# Filter the temperature variable for the World region and relative to the reference perio
temperature_rel_to_1850_1900 = (
    results
    .filter(variable="Surface Temperature", region="World")
    .relative_to_ref_period_mean(year=range(1850, 1900 + 1))
)

# Plot the temperature for each RCP
temperature_rel_to_1850_1900.lineplot()
plt.title("Global Mean Temperature Projection")
plt.ylabel("°C over pre-industrial (1850-1900 mean)")

# Show the plot
plt.show()
```

```
/Users/prayashpathak/mambaforge/envs/8222env2/lib/python3.10/site-packages/scmdata/plotting.

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  ax = sns.lineplot(data=plt_df, **kwargs)
```



Expected temperature in 2010 fopr each RCP:

RCP 2.6 : 1.8 RCP 4.5 : 2.6 RCP 6.0 : 3.2 RCP 8.5 : 4.8

Question 3

The DICE version that we ran above is the 2016 version of Nordhaus' model. Recently, Nordhaus and team released a new version, documented in Barrage and Nordhaus (2023). Read the paper and compare it to what was in the 2016 version and summarize in 2-3 paragraphs what the main differences are with emphasis on explaining the technical differences (i.e., what coefficient changed and what were its new and old values). You may want to look through the code the Lint Barrange and William Nordhaus put online, which can be found here: https://yale.app.box.com/s/whlqcr7gtzdm4nxnrfhvap2hlzebuvvm

The first change that they mention is decreasing the pure rate of social time preference from 0.015 to 0.010, and increasing the elasticity of the marginal utility of consumption from 1.45 to 1.5 attributing the reason to reflect the decline in market interest rate over recent years.

Moving on to the damage function, the authors mention almost two times increase in the damage coefficient as compared to 2016 model to account for the increase in social cost of carbon. Previously, 3 degree celsius warming was equivalent to damages loss estimated at 1.62% of output which now has been increased to 3.12%.

Likewise, the authors mention two minor revisions to the abatement cost - one of which is the inclusion of emissions other than energy CO2. Another major change, is the revision in its treatment of GHG emissions. Basically, the 2016 version only controlled for industrial CO2 emission, which is now updated to include all abatable emissions (land emission, methane, CFCs). Likewise, carbon tax has been added as an additional control variable in the optimization. As for carbon cycle, the authors introduce DFAIR model as a major structural revision.