Actuaries Climate Index Documentation

KANE Dabakh

July 23rd, 2024

Contents

1 Introduction **3**

2 Installation **3**
2.1 Steps to Install the ACI Project . . . 3
2.1.1 Install Poetry . . . 3
2.1.2 Install the Required Dependencies . . . 3
2.1.3 Verify the Installation . . . 4
2.1.4 Run Jupyter Notebook (Optional) . . . 4
2.2 Important Notes . . . 4

3 Project Structure **4**
3.1 Root Directory . . . 4
3.2 aci/ Directory . . . 5
3.3 components/ Directory . . . 6
3.4 data/ Directory . . . 6
3.5 required_data/ Directory . . . 6
3.6 tests/ Directory . . . 7
3.7 Project Directory Structure Diagram . . . 7

4 Usage **8**
4.1 Downloading Data via the Copernicus API . . . 8
4.2 Downloading and Processing Sea Level Data . . . 8
4.3 Processing Temperature Data . . . 9
4.4 Processing Precipitation Data . . . 9
4.5 Processing Drought Data . . . 10
4.6 Processing Wind Data . . . 10
4.7 Processing Sea Level Data . . . 11
4.8 Running a Complete Pipeline . . . 11
4.9 Visualizing the Results . . . 12
4.10 Running Unit Tests . . . 12

5 Components Documentation **12**
5.1 Component Class . . . 12
5.2 TemperatureComponent Class . . . 13
5.3 PrecipitationComponent Class . . . 14

5.4 DroughtComponent Class 15
5.5 WindComponent Class 16
5.6 SeaLevelComponent Class 17
5.7 ActuariesClimateIndex Class 18

6 Calculation **19**
6.1 Mathematical Models and Algorithms 19
6.1.1 Component Calculations 19

7 Use Case Examples **21**
7.1 Example 1: Analysis of Precipitation Data 21
7.2 Example 2: Analysis of Wind Data 22
7.3 Example 3: Analysis of Drought Data 22
7.4 Example 4: Analysis of Sea Level Data 22

1 Introduction

Climate change represents an increasingly significant challenge for modern societies, affecting not only the natural environment but also infrastructures, the economy, and the quality of life. Among the many manifestations of climate change are rising average temperatures, intensifying extreme weather events such as floods, droughts, and storms, as well as rising sea levels. These transformations have direct consequences on various sectors, including agriculture, water management, energy, and, crucially, the insurance industry.

In response to these challenges, actuaries play a crucial role in assessing climate risks and developing appropriate management strategies. It is in this context that the Actuaries Climate Index™ (ACI) was developed—a tool designed to measure changes in climate extremes over time, particularly in the United States and Canada. The ACI provides an objective and quantifiable measure of climate risks, enabling actuaries, policymakers, and the general public to better understand climate trends and adopt suitable measures.

The ACI is based on the analysis of several key climate variables, such as temperature, precipitation, wind speed, and sea level. These variables are integrated into a mathematical model that calculates anomalies relative to a defined reference period. The result is a composite index reflecting the intensity of climate extremes in a given region. By providing a robust and transparent tool for climate risk assessment, the ACI helps strengthen societal resilience to the impacts of climate change.

This document provides a detailed description of the components of the ACI, as well as the methods used to calculate and interpret the index. We also explore the technical aspects related to the implementation of the model, focusing on the use of climate data, the calculation of anomalies, and the standardization of results.

2 Installation

To ensure the proper functionality of the Actuaries Climate Index™ (ACI) project, it is essential to set up a compatible Python environment with the necessary dependencies. Given the specific versions of some packages required by the project, particularly due to compatibility issues between recent versions of `xarray` and `numpy`, we recommend using `poetry` to create a dedicated environment for the project. This environment will isolate the dependencies and prevent conflicts with other projects.

2.1 Steps to Install the ACI Project

2.1.1 Install Poetry

```
pipx install poetry
```

2.1.2 Install the Required Dependencies

With the environment activated, install the necessary packages using the `poetry` package manager. The ”pyproject.toml” file lists all the dependencies to in stall:

```
poetry install
```

2.1.3 Verify the Installation

After installation, verify that the packages are correctly installed by running Python and importing the necessary libraries:

```
python
```

Inside the Python interpreter, try importing the libraries:

```
import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
```

If no errors occur, your environment is correctly set up.

2.1.4 Run Jupyter Notebook (Optional)

If you plan to use Jupyter Notebook for working with the project, you can start it by running:

```
jupyter notebook
```

This command will open a Jupyter Notebook interface in your web browser, allowing you to interactively work with the ACI project.

2.2 Important Notes

- **Compatibility:** This setup ensures that you avoid known compatibility issues between `xarray` and `numpy`. The specified versions are critical for the proper functioning of the ACI project.
- **Environment Isolation:** Using a `poetry` environment ensures that the installed packages do not interfere with other Python projects on your system. It also makes it easier to manage dependencies specific to the ACI project.

3 Project Structure

The Actuaries Climate Index™ (ACI) project is organized into several directories and files, each serving a specific purpose in the development, execution, and maintenance of the project. This structure is designed to separate the source code, data, documentation, and tests, facilitating collaborative development and efficient maintenance.

3.1 Root Directory

The root directory of the ACI project contains the following key components:

- **aci/**: This directory contains the core Python scripts and modules essential for calculating the Actuaries Climate Index™. The scripts here handle data processing, integration of various climate components, and the computation of the index.
  - **aci.py**: The central script that coordinates the integration of different components to calculate the ACI.

  - **components/**: A subdirectory that houses modular Python scripts, each dedicated to handling a specific climate variable such as temperature, precipitation, wind, and sea level.
  - **request_copernicus_data.py** and **request_sealevel_data.py**: These scripts are responsible for downloading and processing climate data from external sources like the Copernicus Climate Data Store and sea level databases.

- **data/**: This directory stores all data files used by the project, structured into subdirectories to organize the raw input data and required data files for processing.
  - **required_data/**: Contains essential data files necessary for the project's operations, and is the folder by default in which extracted data from Copernicus and PSMSL is stored.
  - **tests_data/**: This subdirectory holds test data that mimics the structure of the actual data but on a smaller scale, used for testing the project's components.

- **documentation/**: This directory contains the LaTeX source files needed to generate the project's documentation, as well as the compiled PDF and auxiliary files.

- **tests/**: The tests directory includes all unit tests for the project's various components. Each Python file here is dedicated to testing a specific module, ensuring the correctness of the implementation.

- **README.md**: A markdown file providing an overview of the project, along with basic instructions on how to set up and use the ACI project.

- **pyproject.toml**: A toml file listing all the dependencies required to run the project, intended for use with poetry to set up a compatible environment.

- **poetry.lock**: A file created with the command "poetry install" which is the result from dependencies version management.

3.2 aci/ Directory

The **aci/** directory is the core of the project, containing the main script (`aci.py`) that integrates the various components to compute the Actuaries Climate Index™. Within this directory, the **components/** subdirectory holds modular scripts for handling different climate variables. For instance:

- **aci.py**: Acts as the central hub for processing and integrating data from all components to generate the ACI.

- **components/**: A collection of scripts, each dedicated to a specific climate variable. These include:
  - **component.py**: The base class from which all other components inherit.
  - **drought.py**: Manages data related to drought conditions.
  - **precipitation.py**: Handles precipitation data processing.
  - **sealevel.py**: Manages sea level data and processing.

  - **temperature.py**: Processes temperature data.
  - **wind.py**: Deals with wind data, including wind power calculations.

- **request_copernicus_data.py** and **request_sealevel_data.py**: These scripts are used for downloading and processing data from external sources such as Copernicus Climate Data Store and sea level databases, ensuring the project has up-to-date climate data.

3.3 components/ Directory

The **components/** directory is crucial as it houses the modular scripts for each climate component. Each script focuses on a different aspect of climate data:

- **component.py**: Serves as the foundational class from which other component classes inherit methods and attributes.
- **drought.py**: Specializes in the processing and analysis of drought data.
- **precipitation.py**: Focuses on handling and analyzing precipitation data.
- **sealevel.py**: Deals with the processing of sea level data.
- **temperature.py**: Manages temperature data, including the calculation of anomalies and extremes.
- **wind.py**: Handles wind data, including calculations related to wind power and thresholds.

3.4 data/ Directory

The **data/** directory is dedicated to storing all data files used by the project. This directory is organized into subdirectories for clarity:

- **required_data/**: This subdirectory contains the critical data files necessary for the project's operations, such as gridded country data and NetCDF files for various climate variables.
- **tests_data/**: Stores test data that is structured similarly to the actual data but on a smaller scale. This data is used to test the functionality of the project without processing large datasets.

3.5 required_data/ Directory

Within the **required_data/** directory, essential data files are stored. These include:

- **countries_gridded_0.1deg_v0.1_FRo.nc**: A NetCDF file containing gridded climate data specific to France.
- **psmsl_data.csv**: This dataset provides informations about all tide gauges referenced by the Permanent Service for Mean Sea Level (PSMSL). It is used to extract all data available on their website for a specified country.

3.6 tests/ Directory

The **tests/** directory contains unit tests that are critical for ensuring the functionality and reliability of the project's components. Each test script is dedicated to a specific module:

- **test_drought.py**: Contains unit tests for the drought component, verifying its correctness.
- **test_precipitation.py**: Tests the functionality of the precipitation component.
- **test_sealevel.py**: Ensures that the sea level component processes data correctly.
- **test_temperature.py**: Contains tests for the temperature component, ensuring accurate data processing.
- **test_wind.py**: Verifies the calculations related to wind data within the project.
- **test_aci.py**: Contains tests that validate the overall ACI calculation process, integrating all components.

3.7 Project Directory Structure Diagram

To provide a visual understanding of the project's organization, below is a diagram representing the directory structure:

```
aci_project/
|
+-- aci/
|   +-- aci.py
|   +-- utils.py
|   +-- request_copernicus_data.py
|   +-- request_sealevel_data.py
|   +-- components/
|       +-- component.py
|       +-- drought.py
|       +-- precipitation.py
|       +-- sealevel.py
|       +-- temperature.py
|       +-- wind.py
+-- data/
|   +-- required_data/
|   |   +-- psmsl_data.csv
|   +-- tests_data/
+-- docs/
|   +-- conf.py
|   +-- example_aci.ipynb
|   +-- index.rst
|   +-- make.bat
|   +-- Makefile
|   +-- documentation.pdf
+-- tests/
|   +-- test_drought.py
```

```
|   +-- test_precipitation.py
|   +-- test_sealevel.py
|   +-- test_temperature.py
|   +-- test_wind.py
+-- environment.yml
+-- requirements.txt
+-- setup.py
+-- README.md
```

4 Usage

This section describes how to use the Actuaries Climate Index™ (ACI) project to perform climate data analyses, download necessary data, and calculate the climate index. The examples provided cover the use of the main components, including data downloading and processing.

4.1 Downloading Data via the Copernicus API

To download climate data such as ERA5 data using the Copernicus API, you can use the 'request_copernicus_data.py' script. Here's how to request data:

```python
from aci.request_copernicus_data import Era5var

# Define the area, coordinates, year range, and variable to download
area = 'France'
coordinates = [51.0, -5.0, 41.0, 9.0]  # North, West, South, East
years = '1980-1981'
variable_name = '2m_temperature'

# Initialize the data request for ERA5
era5 = Era5var(area, coordinates, years, variable_name, monthly=True)

# Request and download the data
era5.request_data()

# The data will be saved in the specified directory as NetCDF files.
```

This will download ERA5 data for the specified area and time period, saving it as NetCDF files. In order to download, you first need to register a API key on their website (`cds.climate.copernicus.eu`) and follow the instructions to place the key.

4.2 Downloading and Processing Sea Level Data

To download and process sea level data using the Permanent Service for Mean Sea Level (PSMSL) data, use the 'request_sealevel_data.py' script as follows:

```python
from aci.request_sealevel_data import download_and_extract_data,
    ↪ load_dataframe, copy_and_rename_files_by_country

# Download and extract the PSMSL data
download_and_extract_data()

# Load the DataFrame containing metadata about the sea level data
```

```
df = load_dataframe()

# Copy and rename files for a specific country (e.g., France)
country_abbrev = 'FRA'
copy_and_rename_files_by_country(country_abbrev, df)

# The sea level data will now be available in the `../data/
    ↪ sealevel_data_FRA/` directory.
```

4.3 Processing Temperature Data

To process temperature data and calculate temperature-based climate indices:

```
from aci.components.temperature import TemperatureComponent

# Example 1: Initialize with a single file and no mask
temperature10_component = TemperatureComponent(
    'data/required_data/temperature_1960-1970.nc'
    percentile=10,
    extremum='min',
    above_thresholds=False
)

# Example 2: Initialize with a directory and an optional mask
temperature90_component = TemperatureComponent(
    temperature_data_path = 'data/required_data/temperature_data/',
    mask_path = 'data/required_data/mask.nc',
    percentile=90,
    extremum='max',
    above_thresholds=True
)

# Calculate the T90 index (90th percentile exceedance)
t90_values = temperature90_component.calculate_component(
    reference_period('1961-01-01', '1990-12-31'))

# Calculate the T10 index (10th percentile exceedance)
t10_values = temperature10_component.calculate_component(
    reference_period=('1961-01-01', '1990-12-31'))

print(f"T90 Values: {t90_values}")
print(f"T10 Values: {t10_values}")
```

4.4 Processing Precipitation Data

To process precipitation data and calculate related indices:

```
from aci.components.precipitation import PrecipitationComponent

# Example 1: Initialize with a single file and no mask
precipitation_component = PrecipitationComponent(
    'data/required_data/precipitation_1960-1970.nc'
)

# Example 2: Initialize with a directory and an optional mask
precipitation_component = PrecipitationComponent(
```

```
    'data/required_data/precipitation_data/',
    'data/required_data/mask.nc'
)

# Calculate the anomaly of maximum monthly precipitation
monthly_max_anomaly = precipitation_component.calculate_component(
     reference_period=('1961-01-01', '1990-12-31'),
     var_name = 'tp',
     window_size = 5
)

print(f"Monthly Max Anomaly: {monthly_max_anomaly}")
```

4.5 Processing Drought Data

To process precipitation data and calculate related indices:

```
from aci.components.precipitation import PrecipitationComponent

# Example 1: Initialize with a single file and no mask
drought_component = DroughtComponent(
    'data/required_data/precipitation_1960-1970.nc'
)

# Example 2: Initialize with a directory and an optional mask
drought_component = DroughtComponent(
    'data/required_data/precipitation_data/',
    'data/required_data/mask.nc'
)

# Calculate the anomaly of maximum monthly precipitation
consecutive_dry_days = drought_component.calculate_component(
     reference_period=('1961-01-01', '1990-12-31')
)

print(f"Consecutive dry days : {consecutive_dry_days}")
```

4.6 Processing Wind Data

To process wind data and calculate wind-related indices:

```
from aci.components.wind import WindComponent

# Example 1: Initialize with single files for u10, v10, and no mask
wind_component = WindComponent(
    'data/required_data/wind_u10_1960-1970.nc',
    'data/required_data/wind_v10_1960-1970.nc'
)

# Example 2: Initialize with directories for u10, v10, and an optional
    ↪ mask
wind_component = WindComponent(
    'data/required_data/wind_u10_data/',
    'data/required_data/wind_v10_data/',
    'data/required_data/mask.nc'
)
```

```
15
16 # Calculate wind exceedance frequency
17 wind_exceedance = wind_component.calculate_component(
18     reference_period=('1961-01-01', '1990-12-31'))
19
20 print(f"Wind Exceedance Frequency: {wind_exceedance}")
```

4.7 Processing Sea Level Data

To process sea level data using the SeaLevelComponent:

```
1  from aci.components.sealevel import SeaLevelComponent
2
3  # Define study and reference periods
4  study_period = ('1980-01-01', '2020-12-31')
5  reference_period = ('1961-01-01', '1990-12-31')
6
7  # Initialize the SeaLevelComponent with the country abbreviation and
       ↪ periods
8  sea_level_component = SeaLevelComponent('FRA', study_period,
       ↪ reference_period)
9
10 # Process the sea level data
11 processed_sea_level_data = sea_level_component.process()
12
13 # View the processed sea level data
14 print(processed_sea_level_data)
```

4.8 Running a Complete Pipeline

To run a complete pipeline, integrating multiple components and calculating the Actuaries Climate Index:

```
1  from aci.aci import ActuariesClimateIndex
2
3  # Example 1: Initialize the Actuaries Climate Index with single file
       ↪ paths and no mask
4  aci = ActuariesClimateIndex(
5      temperature_data_path='data/required_data/temperature_1960-1970.nc',
6      precipitation_data_path='data/required_data/precipitation_1960-1970.
           ↪ nc',
7      wind_u10_data_path='data/required_data/wind_u10_1960-1970.nc',
8      wind_v10_data_path='data/required_data/wind_v10_1960-1970.nc',
9      country_abbrev='FRA',
10     study_period=('1980-01-01', '2020-12-31'),
11     reference_period=('1961-01-01', '1990-12-31')
12 )
13
14 # Example 2: Initialize the Actuaries Climate Index with directories and
       ↪  an optional mask
15 aci = ActuariesClimateIndex(
16     temperature_data_path='data/required_data/temperature_data/',
17     precipitation_data_path='data/required_data/precipitation_data/',
18     wind_u10_data_path='data/required_data/wind_u10_data/',
19     wind_v10_data_path='data/required_data/wind_v10_data/',
20     country_abbrev='FRA',
```

```
    mask_path='data/required_data/mask.nc',
    study_period=('1980-01-01', '2020-12-31'),
    reference_period=('1961-01-01', '1990-12-31')
)

# Calculate the overall ACI index
aci_index = aci.calculate_aci()

print(f"Actuaries Climate Index: {aci_index}")
```

4.9 Visualizing the Results

To visualize the results using Matplotlib:

```
import matplotlib.pyplot as plt

# Visualize the T90 index
t90_values.plot()
plt.title("T90 Index (90th Percentile Exceedance)")
plt.show()

# Visualize the Actuaries Climate Index
plt.plot(aci_index['ACI'])
plt.title("Actuaries Climate Index")
plt.xlabel("Time")
plt.ylabel("Index")
plt.show()
```

4.10 Running Unit Tests

To verify the accuracy of the ACI project, unit tests can be run with the following command:

```
poetry run make test
```

This command will execute all tests defined in the 'tests/' directory and display the results, confirming whether the different components of the project are functioning correctly.

To check the code coverage :

```
poetry run make coverage
```

5 Components Documentation

5.1 Component Class

File: component.py
Description: Serves as the base class for various climate data components, providing essential functionalities for data management and parallel processing.

Methods:

- `__init__(self, data`$_path, mask_path, var\_name$) : $Constructs the component with specified data$
  `Parameters:`

- $\texttt{data}_{p}ath(str) : pathtotheDatasetcontainingtheprimaryclimatedata.$
- `apply_mask(self, var_name, threshold=0.8)`: Applies a geographical or conditional mask to the data, which is essential for focusing analysis on specific regions or conditions.
  Parameters:
  - `var_name (str)`: The variable within the dataset to which the mask is applied.
  - `threshold (float)`: Threshold value above which data is retained.

  `standardize_metric(self, metric, reference_period, area)`: Standardizes a metric using historical data, crucial for trend analysis and anomaly detection in climate studies.
  Parameters:
  - `metric (xarray.DataArray)`: The climate metric to standardize.
  - `reference_period (tuple)`: The start and end dates defining the historical period for normalization.
  - `area (bool)`: Whether to average the results over a geographical area.

  `calculate_rolling_sum(self, var_name, window_size)`: Computes a rolling sum over a specified window size, which is essential for analyzing trends over time.
  Parameters:
  - `var_name (str)`: The variable name to calculate the rolling sum.
  - `window_size (int)`: The number of intervals over which to calculate the sum.

5.2 TemperatureComponent Class

File: temperature.py
Description: Manages temperature data, including the processing of daily extremes and the computation of temperature-based indices such as T90 and T10.

Methods:

- `__init__(self, temperature_data_path, mask_data_path, percentile, extremum, above_thresholds)`: Initializes the TemperatureComponent with paths to temperature data and a mask.
  Parameters:
  - `temperature_data_path (str)`: Path to the dataset containing temperature data.
  - `mask_data_path (str)`: Path to the dataset used for applying geographical masks.
  - `percentile (float)`: percentile chosen for the thresholds.
  - `extremum (str)`: specifies whether to find 'min' or 'max' temperature.
  - `above_thresholds (bool)`: if True counts the values above the percentile, if False under the thresholds.
- `temp_extremum(self, extremum, period)`: Calculates daily minimum or maximum temperatures based on specified time periods (day or night), which is crucial for assessing temperature extremes.
  Parameters:

- `extremum (str)`: Specifies whether to find 'min' or 'max' temperature.
- `period (str)`: Specifies the time period, 'day' or 'night'.

- `calculate`$_p ercentiles(self, n, reference\_period, part\_of\_day)$: $Computespercentilesfortempe$
  `Parameters:`

  - `n (int)`: The percentile to compute (e.g., 90 for the 90th percentile).
  - `reference_period (tuple)`: The start and end dates for the period over which to calculate percentiles.
  - `part_of_day (str)`: 'day' or 'night' to specify when the percentiles should be calculated.

5.3 PrecipitationComponent Class

File: precipitation.py

The `PrecipitationComponent` class extends the `Component` base class and specializes in processing precipitation data. It offers methods to calculate monthly and seasonal maximum precipitation values and their anomalies. This class is crucial for understanding precipitation trends, which are significant for studying climate change impacts, especially in terms of hydrological extremes such as floods and droughts.

- **Constructor:** `__init__(self, precipitation_path, mask_path)`
  - **Parameters:**
    - `precipitation_path (str)`: Path to the dataset containing precipitation data.
    - `mask_path (str)`: Path to the dataset containing mask data for filtering the geographic area of interest.
  - **Description:** This constructor initializes the component with the specified datasets, loading the precipitation data and applying a geographical mask if provided. This setup is essential for subsequent calculations that rely on this filtered data.

- **Method:** `calculate`$_m aximum_p recipitation_o ver_w indow(self, var\_name, window\_size, season)$

  - **Parameters:**
    - `var_name (str)`: Variable name in the precipitation dataset to calculate the maximum for.
    - `window_size (int)`: The size of the rolling window in days over which the maximum will be calculated.
  - **Returns:** xarray.DataArray containing the maximum precipitation values for each month.
  - **Description:** This method calculates the maximum precipitation values over a specified rolling window for each month. This is particularly useful for identifying the most intense precipitation events within each month, which can be critical for flood risk assessment.

Method: `monthly_max_anomaly(self, var_name, window_size, reference_period, area=None)`

- **Parameters:**
  - `var_name (str)`: Variable name in the precipitation dataset.
  - `window_size (int)`: Window size in days for calculating maximum precipitation.
  - `reference_period (tuple)`: Start and end dates of the reference period for anomaly calculation.
  - `area (bool, optional)`: If provided, calculates the area-averaged anomaly across the specified geographic area.
- **Returns:** xarray.DataArray of monthly maximum precipitation anomalies.
- **Description:** Calculates anomalies in monthly maximum precipitation relative to a historical baseline defined by the reference period. This method is essential for detecting changes or shifts in precipitation intensity, providing insights into how precipitation extremes are evolving in the context of climate change.

5.4 DroughtComponent Class

File: drought.py

The `DroughtComponent` class extends the `Component` base class and is specifically designed to manage and analyze data related to drought conditions. This class is pivotal for assessing the impact and frequency of drought events, which are significant for agricultural planning, water resource management, and environmental conservation.

- **Constructor:** `__init__(self, precipitation_path, mask_path)`
  - **Parameters:**
    - `precipitation_path (str)`: Path to the dataset containing precipitation data.
    - `mask_path (str)`: Path to the dataset containing mask data, used to filter the geographic area of interest.
  - **Description:** Initializes the `DroughtComponent` with specific datasets for precipitation and geographical masking, setting up the essential data structures for further analysis of drought conditions.
- **Method:** `max_consecutive_dry_days(self)`
  - **Returns:** `xarray.DataArray` representing the maximum number of consecutive dry days within each year, crucial for identifying prolonged drought periods.
  - **Description:** Calculates the maximum consecutive days with minimal or no precipitation, providing key data for drought severity assessments. This method is essential for understanding the temporal distribution and intensity of drought conditions, which can impact water resource management and agricultural productivity.
- **Method:** `drought_interpolate(self, max_days_drought_per_year)`
  - **Parameters:**

    * `max_days_drought_per_year (xarray.DataArray)`: DataArray containing the maximum number of consecutive dry days per year.
  - **Returns:** `xarray.DataArray` representing interpolated monthly values based on annual drought data. This method ensures a smooth transition between yearly data points, filling in monthly values where only annual data is available.
  - **Description:** Interpolates monthly drought data based on annual maximum consecutive dry days. It handles the transition between years by interpolating monthly values and fills in missing data for the last available year using the data from the preceding year.

- **Method:** `std_max_consecutive_dry_days(self, reference_period, area=None)`
  - **Parameters:**
    * `reference_period (tuple)`: The start and end dates of the reference period for normalization.
    * `area (bool, optional)`: If True, calculates the area-averaged standardized metric. Default is None.
  - **Returns:** `xarray.DataArray` of standardized values for the maximum consecutive dry days, adjusted for the reference period. This standardized metric is important for comparing drought severity across different times and locations.
  - **Description:** Standardizes the maximum consecutive dry days using historical data from the reference period to provide a normalized measure of drought severity. This standardized metric is crucial for detecting changes in drought patterns over time and assessing the impacts of climate change on drought frequency and intensity.

5.5 WindComponent Class

File: wind.py

The `WindComponent` class extends the `Component` base class and specializes in managing and analyzing wind data. It is pivotal for calculating wind power potential and identifying extreme wind events, which are crucial for both energy resource management and risk assessment in storm-prone areas.

- **Constructor:** `__init__(self, u10_path, v10_path, mask_path)`
  - **Parameters:**
    * `u10_path (str)`: Path to the dataset containing wind u-component data.
    * `v10_path (str)`: Path to the dataset containing wind v-component data.
    * `mask_path (str)`: Path to the dataset containing mask data for filtering the geographic area of interest.
  - **Description:** This constructor initializes the wind component with the specified datasets for the u and v components of wind, setting the stage for complex vector calculations required for wind power and other analyses.

- **Method:** `wind_power(self, reference_period=None)`
  - **Parameters:**

* `reference_period (tuple, optional)`: If provided, calculates daily wind power for the specified reference period, which is crucial for analyzing historical data or specific time frames.

– **Returns:** xarray.DataArray representing the daily mean wind power, crucial for assessments of wind energy potential.

– **Description:** Calculates daily wind power using the formula $\frac{1}{2}\rho v^3$ where ρ is the air density and v is the wind speed. This method is integral for determining the energy production capabilities of wind turbines and for assessing potential wind damage during extreme events.

- **Method:** `wind_thresholds(self, reference_period)`

  – **Parameters:**

    * `reference_period (tuple)`: The start and end dates of the reference period for calculating wind power thresholds.

  – **Returns:** xarray.DataArray of wind power thresholds based on the 90th percentile, used for categorizing wind events as normal or extreme.

  – **Description:** This method establishes thresholds for wind power that categorize days as having normal or extreme wind conditions based on historical data. These thresholds are essential for understanding and predicting storm damage potential and for planning in wind-sensitive industries like insurance and construction.

5.6 SeaLevelComponent Class

File: sealevel.py

The `SeaLevelComponent` is designed to handle sea level data, a critical aspect of climate studies, especially in the context of global warming and its impact on rising sea levels. This class is crucial for analyzing changes in sea levels over time, providing essential data for coastal management and flood risk assessment.

- **Constructor:** `__init__(self, country_abbrev, study_period, reference_period)`

  – **Parameters:**

    * `country_abbrev (str)`: Abbreviation of the country for which sea level data is to be analyzed.
    * `study_period (tuple)`: Start and end dates for the period under study.
    * `reference_period (tuple)`: Start and end dates for the historical period used for normalization.

  – **Description:** Constructs the necessary attributes for the Sea Level Component, setting up the environment to load, process, and analyze sea level data specific to a given country.

- **Method:** `load_data(self)`

  – **Returns:** pandas.DataFrame containing concatenated data from all relevant files in the specified directory.

  - **Description:** Loads sea level data from files stored in a designated directory, combining them into a comprehensive dataset. This method is essential for preparing the data for further processing and analysis.

- **Method:** `correct_date_format(self, data)`
  - **Parameters:**
    * `data (pd.DataFrame)`: DataFrame containing the original sea level data with dates in float format.
  - **Returns:** pandas.DataFrame with corrected date formats.
  - **Description:** Converts the date format from a float representation (e.g., 1999.9583 for December 1999) to a standard YYYY-MM-DD format, facilitating easier analysis and visualization of the data.

- **Method:** `clean_data(self, data)`
  - **Parameters:**
    * `data (pd.DataFrame)`: DataFrame to be cleaned.
  - **Returns:** pandas.DataFrame with NaNs replacing any sentinel values that represent missing or erroneous data.
  - **Description:** Cleans the dataset by replacing known sentinel values (e.g., -99999) with NaN to prevent them from affecting the subsequent analyses. This is a crucial step to ensure the accuracy of the sea level trends and statistics derived from the data.

- **Method:** `process(self)`
  - **Returns:** pandas.DataFrame that has been fully processed and standardized, ready for detailed analysis or reporting.
  - **Description:** Conducts a comprehensive processing of the sea level data which includes loading, date correction, cleaning, and standardizing based on reference periods. This method is essential for preparing the data for in-depth analysis and for generating actionable insights regarding sea level changes.

5.7 ActuariesClimateIndex Class

File: aci.py

The `ActuariesClimateIndex` class serves as the central node for calculating the ACI, integrating data from multiple climate variables. It synthesizes outputs from temperature, precipitation, wind, drought, and sea level components to provide a holistic view of climate extremes.

- **Constructor:** `__init__(self, temperature_data_path, precipitation_data_path, wind_u10_data_path, wind_v10_data_path, country_abbrev, mask_data_path, study_period, reference_period)`
  - **Parameters:**
    * `temperature_data_path (str)`: Path to the temperature data file.

* `precipitation_data_path (str)`: Path to the precipitation data file.
* `wind_u10_data_path (str)`: Path to the wind u-component data file.
* `wind_v10_data_path (str)`: Path to the wind v-component data file.
* `country_abbrev (str)`: Abbreviation of the country for which the ACI is being calculated.
* `mask_data_path (str)`: Path to the mask data file, used for spatial filtering of data.
* `study_period (tuple)`: Tuple containing the start and end dates of the study period.
* `reference_period (tuple)`: Tuple containing the start and end dates of the reference period used for normalization.

– **Description:** This constructor sets up the environment and initializes all necessary parameters and paths to facilitate the calculation of the ACI.

- **Method:** `calculate_aci(self, factor=None)`

  – **Parameters:**

  * `factor (float, optional)`: A multiplier used in the computation to adjust the influence of certain components, defaulting to 1.

  – **Returns:** pandas.DataFrame containing the calculated ACI values.

  – **Description:** This method integrates anomalies from all climate components to compute the ACI. Each component contributes to the final index, reflecting the aggregated effect of climate extremes as specified by the methodology. The calculation considers anomalies relative to a baseline period, adjusted by the provided factor for each component to emphasize or de-emphasize certain effects.

6 Calculation

The Actuaries Climate Index (ACI) aggregates various climate indicators into a single measure that reflects the intensity and frequency of extreme climate events over time. Each component of the ACI—temperature, precipitation, wind, drought, and sea level—is processed to compute anomalies and standardized metrics that contribute to the final index. This section describes the mathematical models and algorithms used in these calculations.

6.1 Mathematical Models and Algorithms

6.1.1 Component Calculations

Each component of the ACI undergoes specific preprocessing, calculations, and standardizations to quantify climate extremes. These processes are implemented in Python classes, each tailored to handle different aspects of climate data.

Temperature Component The `TemperatureComponent` class calculates the T90 and T10 indices, which represent the exceedance over the 90th percentile and the exceedance below the 10th percentile of temperatures, respectively. These indices are vital for assessing the frequency of extreme temperatures.

- **Methodology:**
  1. Data is first masked to focus on relevant geographical areas.
  2. Daily maximum and minimum temperatures are calculated.
  3. The 90th and 10th percentiles are determined based on a historical reference period.
  4. The exceedance days are counted where temperatures surpass these thresholds.
- **Formulas:**

$$T90 = \frac{\text{Number of days above 90th percentile}}{\text{Total number of days}}$$

$$T10 = \frac{\text{Number of days below 10th percentile}}{\text{Total number of days}}$$

Precipitation Component The `PrecipitationComponent` focuses on anomalies in precipitation, calculating monthly and seasonal maximums which are crucial for identifying significant rainfall events that could lead to flooding.

- **Methodology:**
  1. Precipitation data is aggregated monthly and seasonally.
  2. Maximum precipitation levels are computed for these periods.
  3. Anomalies are calculated against a historical reference period to identify significant deviations.
- **Formulas:**

$$\text{Precipitation Anomaly} = \frac{\text{Observed maximum} - \text{Historical mean maximum}}{\text{Standard deviation of historical maximum}}$$

Wind Component The `WindComponent` assesses wind power potential and the frequency of exceedance over set wind speed thresholds, important for evaluating potential wind damage during storms.

- **Methodology:**
  1. Wind speeds are calculated using vector components (u and v).
  2. Daily average wind power is derived from the wind speed.
  3. Thresholds for extreme wind events are established based on the 90th percentile of historical data.
- **Formulas:**

$$\text{Wind Power} = \frac{1}{2}\rho v^3$$

where ρ is air density, and v is wind speed.

Drought Component The `DroughtComponent` calculates the maximum number of consecutive dry days, a critical metric for drought assessment.

- **Methodology:**
  1. Daily precipitation is monitored to identify days with minimal or no rainfall.
  2. Consecutive dry days are counted and tracked over time.
  3. The longest dry spell for each period is compared against historical norms.

Sea Level Component The `SeaLevelComponent` processes sea level data to evaluate mean sea level changes and anomalies, key indicators of long-term sea level trends affecting coastal areas.

- **Methodology:**
  1. Sea level measurements are collected and corrected for local datum discrepancies.
  2. Monthly and annual mean sea levels are calculated.
  3. Anomalies are determined relative to a long-term historical mean to identify trends.
- **Formulas:**

$$\text{Sea Level Anomaly} = \text{Observed mean} - \text{Historical mean}$$

7 Use Case Examples

In this section, we present concrete examples of how to use the different components of our project to analyze specific climate data. The data used in these examples are from the Paris region, downloaded from the Copernicus server. Each example is accompanied by an explanation of the code used and the generated graphs to illustrate the results obtained.

7.1 Example 1: Analysis of Precipitation Data

This first example demonstrates how to use the precipitation component to analyze precipitation data over a given period. The goal is to visualize the maximum monthly anomalies in precipitation and smooth them using a moving average.

```
val = precip_component.monthly_max_anomaly("tp", 5, ('1960-01-01', '
    ↪ 1964-12-31'), True)
val.rolling(time=10, center=True).mean().plot()
```

This code extracts the maximum monthly precipitation anomalies for the period from 1960 to 1964 and applies a 10-month moving average to smooth the data. The graph below shows the results.

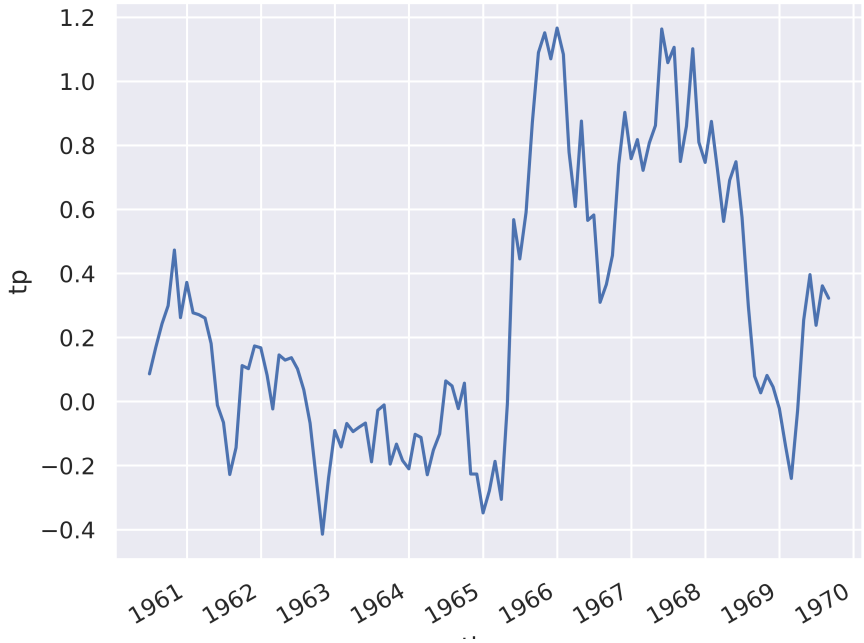

Figure 1: Maximum precipitation anomalies with a 10-month moving average

7.2 Example 2: Analysis of Wind Data

In this example, we use the wind component to analyze wind data and calculate the standardized wind exceedance frequency over a given period.

```
val = wind_component.std_wind_exceedance_frequency(('1960-01-01', '
    ↪ 1964-12-31'), True)
val.rolling(time=12, center=True).mean().plot()
```

This code calculates the frequency at which wind speeds exceed a certain threshold over the period from 1960 to 1964, and smooths the results with a 12-month moving average. The generated graph shows the wind trends.

7.3 Example 3: Analysis of Drought Data

This third example illustrates the use of the drought component to analyze drought periods, particularly consecutive days without precipitation.

```
val = drought_component.std_max_consecutive_dry_days(('1960-01-01', '
    ↪ 1964-12-31'), True)
val.rolling(time=12, center=True).mean().plot()
```

The code above calculates the maximum number of consecutive dry days over a five-year period, with data smoothing by a 12-month moving average.

7.4 Example 4: Analysis of Sea Level Data

Finally, we show how to analyze sea level data using the corresponding component. This example highlights the average sea level evolution over a given period.

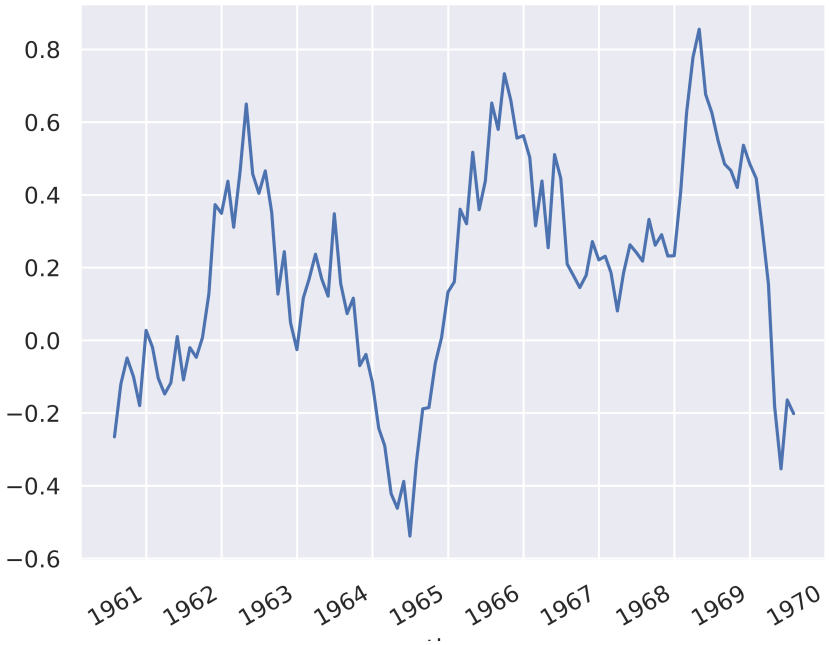

Figure 2: Wind exceedance frequency with a 12-month moving average

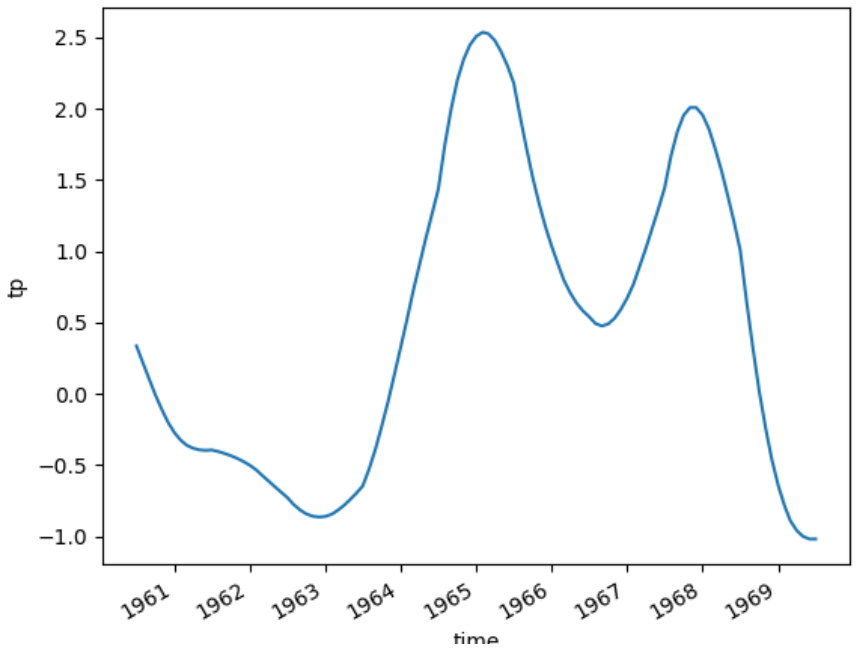

Figure 3: Maximum consecutive dry days with a 12-month moving average

```
a = niv_mer.mean(axis=1)
a.rolling(window=60, min_periods=12, center=True).mean().plot()
```

This code calculates the mean sea level and applies smoothing with a 60-month moving average. The resulting graph shows the variations in sea level over time.

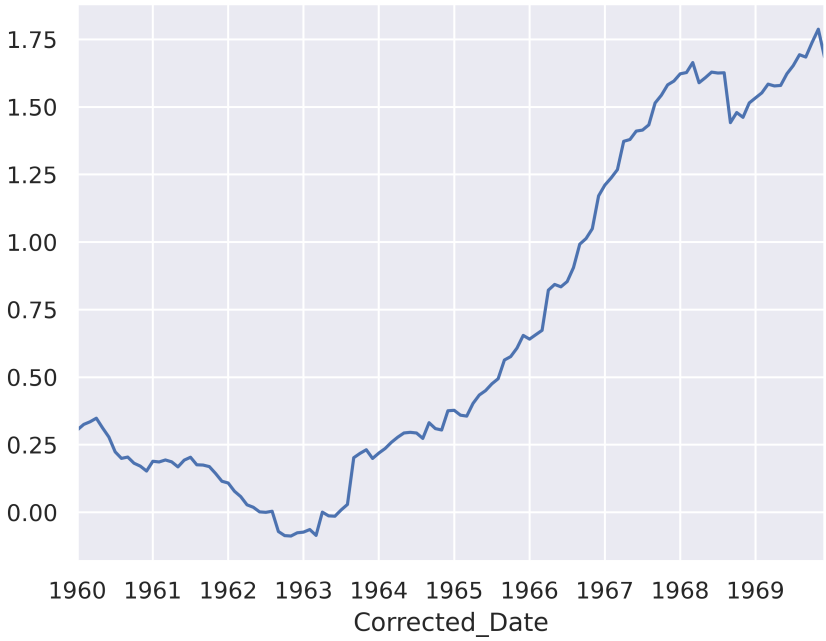

Figure 4: Sea level evolution with a 60-month moving average

Conclusion

These examples demonstrate how to use the developed components to analyze different aspects of climate data, using analysis techniques such as maximum anomalies, exceedance frequencies, and moving averages. Each component can be adapted to specific periods or other types of data depending on the needs of the analysis.