Python libraries and their usage in visualizing meteorological data

Viktoriia Khisanova

Haaga-Helia University of Applied Sciences

Bachelor's Thesis

2022

Bachelor of Business Administration

Abstract

Author(s)
Viktoriia Khisanova

Degree
Bachelor of Business Administration

Report/Thesis Title
Python libraries and their usage in visualizing meteorological data

Number of pages and appendix pages
51+10

The thesis is dedicated to the research of different Python libraries, exploring their functionalities and limitations with the focus on their general use and in meteorological area. As an outcome of this research, the list of the libraries used in meteorology will be presented with comparison by functionalities as well as a small demo of visualized meteorological data using one or more libraries.

The theoretical part includes research on Python as a programming language, the history and practices of data visualization in general and of meteorological data in particular. The empirical part consists of exploring the Python's libraries for visualization in general and in meteorological data. The practical part shows how meteorological data can be visualized using one or more of the libraries.

The outcomes of the thesis can be used by those who are generally interested in Python libraries and want to start using them for visualization as well as those who want to learn more about meteorological data visualization.

Keywords
Python, visualization, library, meteorological data, data analysis.

Contents

1 Introduction 4

2 Theoretical framework 6

2.1 Python: overview and its role in data analysis 6

2.2 Data visualization: history and practices 9

2.3 Data visualization in meteorology: history and practices 14

2.4 Theoretical framework summary 19

3 Carrying out the study 20

3.1 Usage of Python's libraries 20

3.2 Python libraries for visualizing meteorological data 26

3.2.1 MetPy 26

3.2.2 Alarconpy 29

3.2.3 Iris 31

3.2.4 CliMetLab 33

3.2.5 GeoPandas 34

3.3 Hands-on examples 36

3.3.1 CliMetLab 36

3.3.2 GeoPandas 39

3.4 Carrying out the study summary 43

4 Discussion 45

References 48

Appendices 52

Appendix 1. CliMetLab source code and output 52

Appendix 2. GeoPandas source code and output 54

1 Introduction

Programming language Python offers endless possibilities for developers, data scientists, digital artists. Being first released in 1991 by Dutch developer Guido van Rossum, there are two major versions of the language with over 20 sub-versions. Consistently, Python is on top of the “most used”, “to learn” and “most wanted” lists, and 2022 is no exception. According to Berkeley, Python is the second in-demand programming language (Berkeley Extension 2022). It also takes the first position in the “most wanted to learn” for fifth year in the Stackoverflow’s massive developers survey results (Stackoverflow 2021).

One of the key features of Python’s is a simple syntax which makes this language relatively easy to learn and understand when reading. This also increases the coding speed. However, its simplicity does not limit the functionality and possibilities.

Being one of the preferred languages among data scientists, Python has many libraries that simplify the process of working with data while expanding the opportunities. Data visualization is one of the angles of data science and is used to get a visual summary of data. With Python, obtaining valuable insights and catching knowledge from data is not only easy but also highly flexible, interactive and customizable (Gilbert Tanner 2018).

Visualization is an integral part of meteorological data analysis. This data is used not only for showing the current state of such weather indicators as the temperature or wind direction but to predict the future conditions and to make meteorology-related decisions. As the demand for such information increases, so does the complexity of the data which contributes to the development and change in the way of data analysis and visualization.

Research methods

The scientific methods applied in this thesis are mainly represented by qualitative research and comparative analysis of the relevant scientific papers and journals and books.

Goals and objectives

The key objectives of the thesis are as follows:

- collecting the theoretical basis on Python data analysis libraries and their usage in visualizing meteorological data
- highlighting the features of meteorological data visualization

- presenting a demo of visualized meteorological data using one or more libraries.

This work is aimed at developing a strong basis and understanding of Python libraries for further appliance in professional career. Extensive usage of Python's features enables more efficient and accurate results of data analysis. Python allows for solving the problem of automating data collection and data processing, speeds up data analysis, and allows the implementation of new approaches to analysis at work.

Since data analysis is a professional interest of the author and she has been doing data visualization for a year and a half using Business Intelligence tools, the next stage of growth is to learn to program in the field of data analysis and visualization. To be more precise, expanding knowledge and skills in the field of data visualization by exploring the possibilities of Python libraries for visualizing data. The author also finds interesting the visualization of meteorological data in particular, so another goal of this thesis is to study the features of meteorological data visualization and apply this knowledge in practice.

Project scope

This thesis is aimed to deliver the qualitative research results on Python's visualization libraries by their usage in visualizing meteorological data as well as delivering a demo of meteorological data visualization. This does not include drilling through sub-areas of meteorology but instead this topic will be studied in general. Using one or more libraries for visualizing meteorological data does not mean showing the best visualization practices but rather testing and exploring the features. Although data visualization as a research area is not the main objective of this work, its history and practices will be described in the theoretical framework in order to get deeper understanding of from where the current visualization practices came.

2 Theoretical framework

2.1 Python: overview and its role in data analysis

Python is an interpreted, high-level object-oriented programming language with dynamic typing, automatic memory management and convenient high-level data structures such as dictionaries (hash tables), lists, tuples. Supports classes, modules, exception handling, and multithreading. Python has a simple and expressive syntax. The language supports several programming paradigms: structural, object-oriented, functional and aspect-oriented (IT-Black 2017). The benefits of Python, namely “BDFL’s guiding principles for Python’s design” were written in the form of 20 aphorisms by Tim Peters and called “The Zen of Python” (Python 2004).


The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Image 1. The Zen of Python (peps.python.org 2022).

The history of Python started in late 1980s when Guido van Rossum first thought of creating his own programming language. The work began in 1989, and the intention was so that Python will be a descendant of the programming language ABC, in the development of which Guido was also involved. The very first piece of code was published in 1991 (Artima 2003).

A peculiarity of this programming language is that it is completely open to its users: it is free, has no restrictions on the commercial use of products written in Python, and programmers can modify the language without informing its developer (Techtarget 2021). Open-source development allows to involve a large number of engineers and enthusiasts to collaboration, implement ideas and increase the popularity of the programming language.

Python is a cost-effective solution when users factor in the free comprehensive standard library and Python interpreter. It is highly adaptable. Users can easily do edit-test-debug cycles without the requirement for a compilation step. Software engineers frequently choose to code in Python as

they find that it helps them be more productive (Techtarget 2021).

Python's use cases are diverse and numerous which is primarily caused by its simple syntax and wide choice of libraries:

- data science
- machine learning
- web development
- financial analysis
- desktop applications
- business applications
- scripting and utility software (Codecademy 2022).

Python is often used in Data Science because, firstly, compared to other languages, the code for complex tasks in Python is simpler. And secondly, there are many powerful application libraries for solving various problems: primary data processing and analysis, natural language processing and visualization. Python libraries are collections of additional modular code components, sharpened for certain subtle tasks. To manage them, specific skills are needed, having mastered which it is possible to make programming in Python much more efficient. Third-party library modules are needed to extend the functionality of the Python standard library. With the set of features that is offered, users can work on large projects in stages and solve various complex tasks.

In addition to the advantages described above, there are more:

- Comparatively small amounts of code are required to solve complex problems (Figure 1). Algorithms from the Data Science domains are quite complex. Therefore, for their implementation, it is desirable to use a programming language that allows to express the ideas of the developer briefly and concisely. Python, thanks to its syntax and clear code structure, is great for solving such problems. It helps programmers to create compact and powerful programs.

- Cross-platform. Python programs can run on a variety of platforms. In particular - on Windows, Linux, macOS. Code written for a particular platform can run on other platforms without modification.

- Large community. A huge community has formed around Python. There are many online platforms where developers discuss their problems and help each other in solving them as

well as brainstorming ideas on how to improve the language in general.

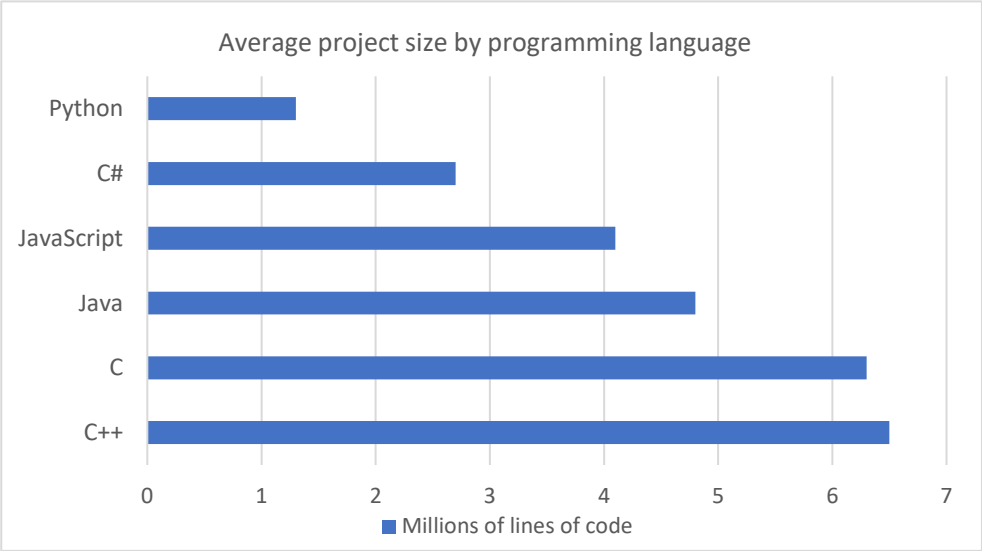


Figure 1. Average project size by language (adapted from Sands 2012).

All the mentioned above is also applicable to Data Analysis in Python. This programming language is functional on all the stages: requirement understanding, data collection, data cleaning, data analysis and processing, result interpretation (Figure 2) (Butwall, Ranka & Shah 2019, 20). All these steps are performed in order to get useful information from raw data. The process of transforming data into information and knowledge includes such operations as sorting, aggregating, filtering, and sometimes machine learning algorithms for discovering patterns.

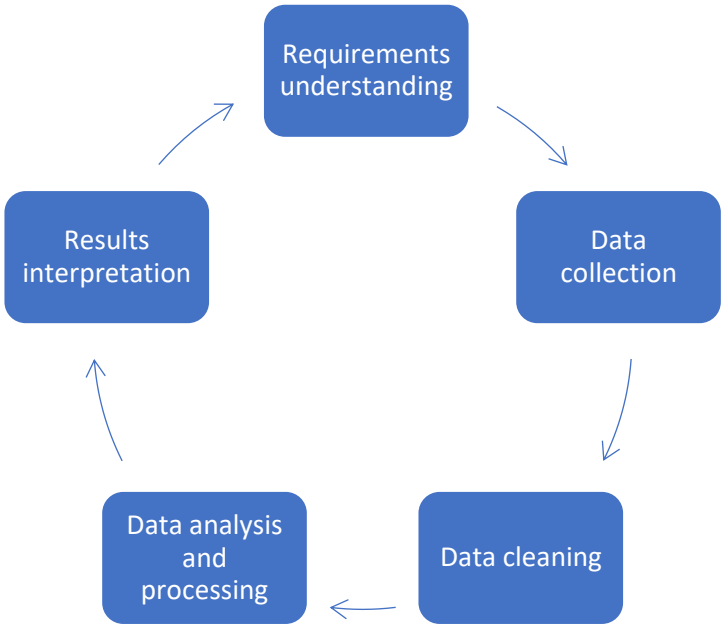


Figure 2. Data Analysis process cycle (adapted from Butwall, Ranka & Shah 2012).

Due to improved library support, Python makes an alternative for data manipulation to such

languages as R or MATLAB, as it is more capable at data set construction. Python is used as a single language for building data-centric applications. (McKinney 2013, 2.) It represents an end-to-end solution that makes access to database and web-based servers as well as data management and processing and statistical computation smoother (Sheppard 2021, 2). One more benefit is ease of using multi-processing on large datasets while reducing the time required to analyze them. In addition to that, Python supports parallel processing – doing more tasks while reducing the overall processing time (Mueller & Massaron 2019, 23-24).

2.2 Data visualization: history and practices

Having its beginning in the mid-1990s, the Milestones Project, started by Michael Friendly, is now an open database consisting of “300 significant milestone events, 400 images, and 350 references to original sources, together with a Google map of authors and a milestones calendar of births, deaths, and important events in this history”, - Michael Friendly and Howard Wainer say in their fundamental work called “A History of Data Visualization & Graphic Communication” (2021). Called statistical historiography, the approach is to use statistical and graphical methods to investigate “historical issues and issues in the history of data visualization itself” (Friendly & Wainer 2021, 3-4).

From ancient times, events and concepts were and are described in the form of words, pictures, and numbers. The oldest one of these is words, which was developed about 100000 years ago, and was used to deliver a concrete signal to others. It took way too much time for the first inscription to appear. 3100 BCE is dated as the first emergence of the clay tablets pictographs (Image 2). The lettering was used not to capture a message but to record numbers. The first witnessed numbers are dated from 30000 years ago (Image 3), and they were used to count in a form of a simple notch. (Friendly & Wainer 2021, 10-12.)

Image 2. Cuneiform tablet: administrative account with entries concerning malt and barley groats, ca. 3100–2900 B.C. (metmuseum.org 2022)

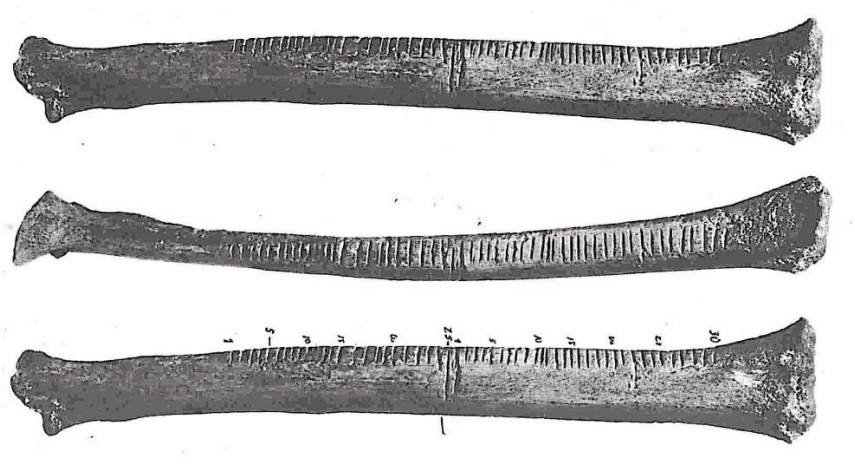


Image 3. Paleolithic Wolf Bone Tally Stick. (kartsci.org 2022)

With these small steps, that took hundreds of thousands of years, humanity came to the first preserved table in the 2nd century in Egypt. That table was a navigation tool that structured astronomical information by being organized by attributes of alignment, white spaces, vertical and horizontal lines of time rules, or, simply put, by columns and rows. (Few 2007, 2-3.)

Over the years, demand for recording data grew and new ways – more accurate, understandable, and efficient – needed to be implemented. As the grouping of numbers became bigger and bigger, the Romans have come up with “C” and “M” that represented 100 and 1000 respectively. But not only numbers became bigger, data became more voluminous and meaningful and, accordingly, more complex for recording. (Hillery 2020.)

One of the most impressive ways of communicating extremely complex maths and narratives is the knotted strings of the Inkas’ (Hillery 2021). Those strings are called knipus, and they are still very hard to decode. Scientists face tens of thousands of knots created “by different people, for different purposes and in different regions of the empire”, - Manuel Medrano (an anthropology research assistant at Harvard University) and Gary Urton (Dumbarton Oaks Professor of Pre-Columbian Studies in the Department of Anthropology at Harvard University) say in their article “The khipu code: the knotty mystery of the Inkas’ 3D records” (Aeon 2018.).

1786 is dated by the release of the work of William Playfair “The Commercial and Political Atlas”, and he is considered the originator of line, bar, area, and pie charts (Hillery 2021). He owes the invention of the charts to his brother, who was a mathematician and geologist and taught his brother that lines can be used to represent everything that can be stated numerically (Gutenberg Ebook 2005). In this particular book, William Playfair presented 43 variants of the time series line graph (Figure 3) and a single bar chart (Figure 4) (Wainer & Spence 2005, 8-9).

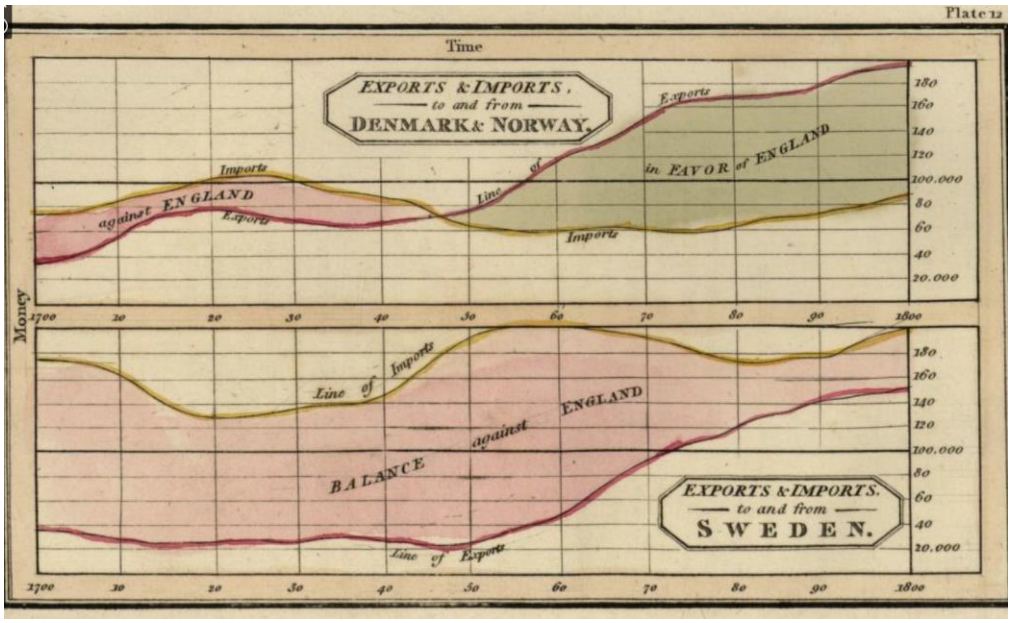


Figure 3. Time series line graph from Playfair's book that shows exports and imports to and from Denmark, Norway and Sweden (archive.org 2022).

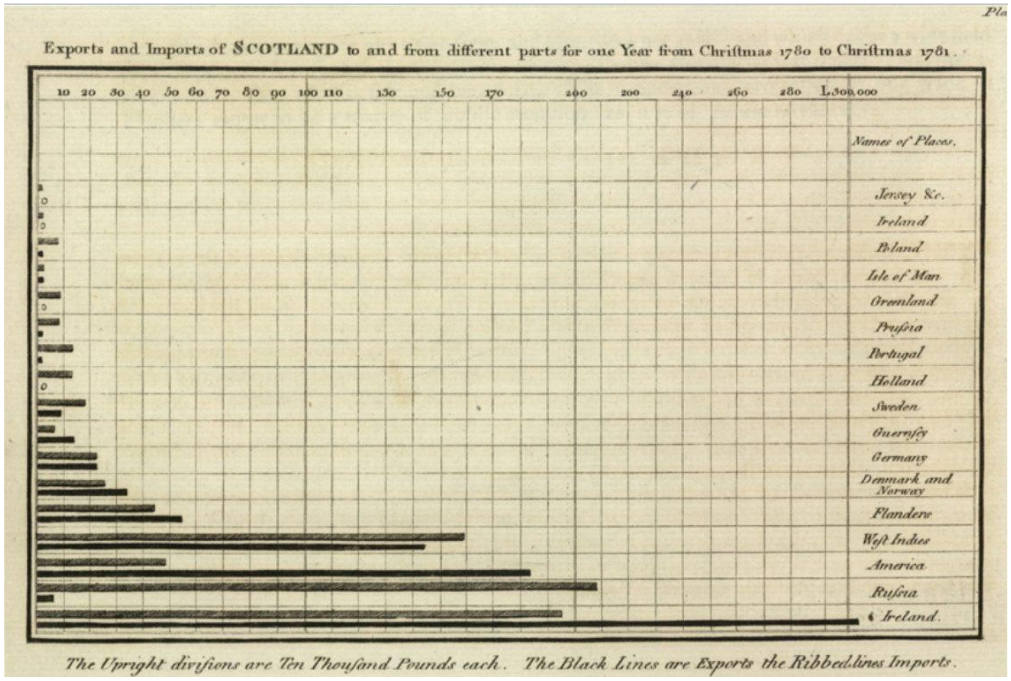


Figure 4. Bar chart from Playfair's book that shows the relative importance of Scotland's trading partners (archive.org 2022).

The next era of visualization belongs to the French civil engineer Charles J Minard who is considered contributing to the information graphics in civil engineering and statistics. He is also famous by showing numerical data on geographic maps which is called "thematic map". His graphical representation of numerical data on a map illustrating Napoleon's devastating losses

sustained during the Russian campaign of 1812 is what made him most famous (Figure 5). (Hillery 2021.)

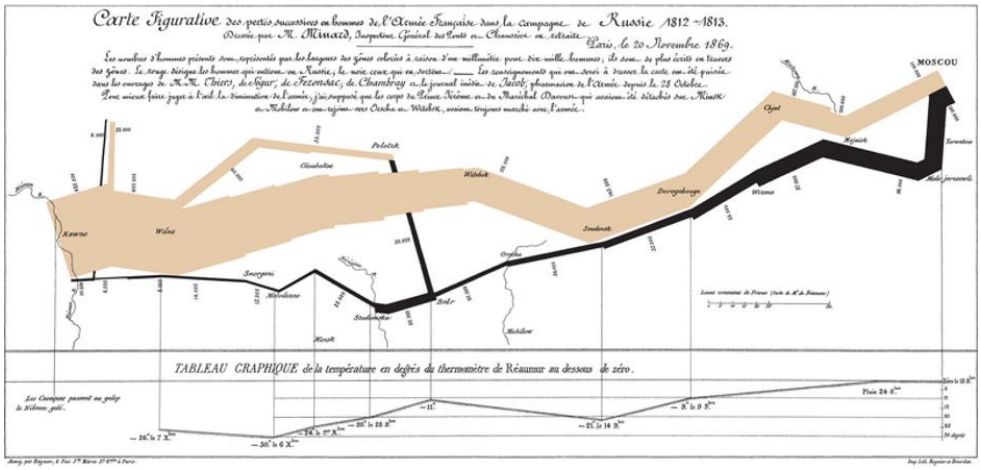


Figure 5. Flow Map of Napoleon's Russian Campaign of 1812 by Charles Minard (datavizblog.com 2022).

In the middle of 19th century, an English statistician Florence Nightingale presented her descriptive graph of the reasons British soldiers' deaths during the Crimean war (Figure 6). It was investigated that majority of 55% dead warriors died due to diseases caused by extremely bad living conditions. As a result, Nightingale started a political campaign to carry out a sanitary reform in the army. The Coxcombs, over 1000-page report, contained an enormous amount of evidence in favor of her case. (Hillery 2021.)

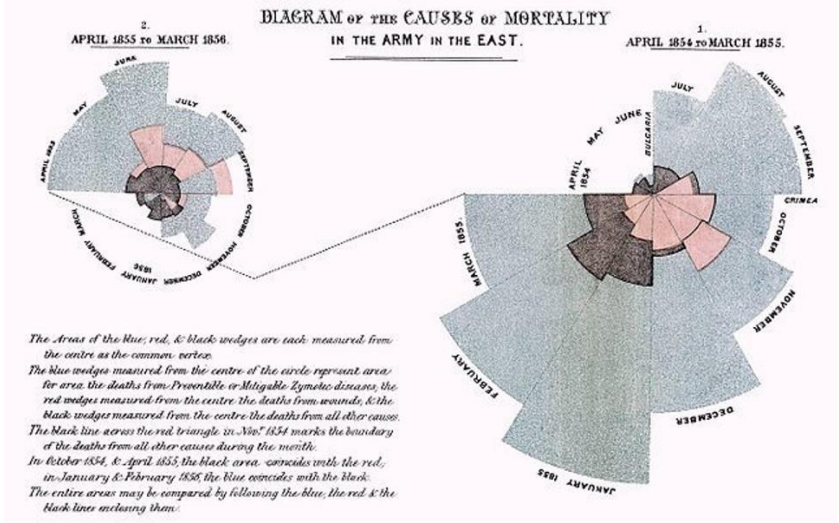


Figure 6. The causes of British soldiers' deaths during the Crimean war (blog.panoply.io 2022).

The Paris Exposition of 1900 is remembered by the graphic visualizations of Black Americans post-slavery created by W. E. B. Du Bois. This massive work is represented by 60 hand-painted 22" x 28" poster-sized drawings and 500 photographs. (Hillery 2021.) Although bar charts made up

the majority of the exhibit's graphics, Du Bois also created a number of original charts. The term “Du Bois Spiral” (Figure 8) was most famously coined by renowned data visualization specialist Elijah Meeks. The Du Bois spiral takes disproportional data and angles (or wraps) them to highlight the inequities in the data, much like a stacked bar chart does. Each acute angle draws attention to the category shift, and the greatest number is so wildly out of proportion that it coils the bar into a spiral. (Forrest 2019.)

Figure 7. Du Bois’s charts from the 1900 Paris Exhibition (tableau.com 08 October 2022)

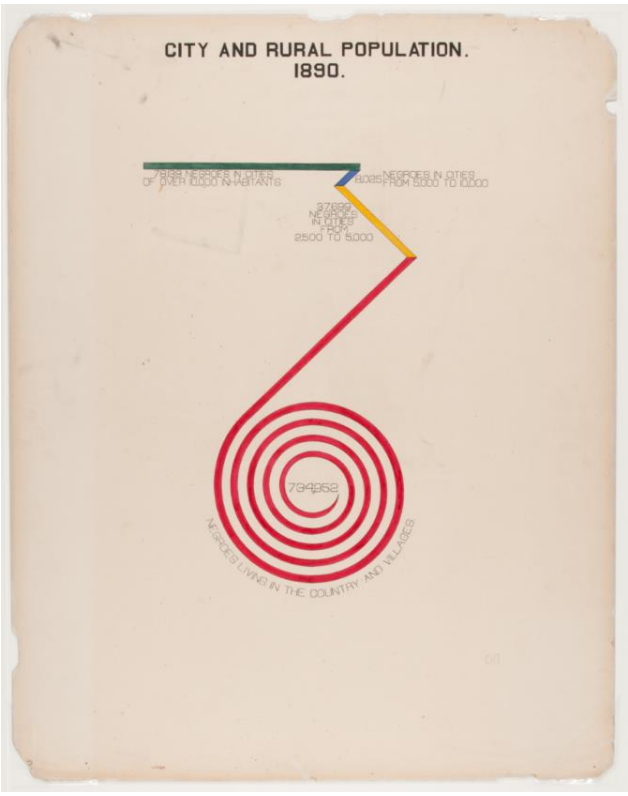


Figure 8. Du Bois Spiral (go.distance.ncsu.edu 2022).

The last quarter of the 20th century is the new era in data visualization as software tools start to appear. Although it is difficult to list all the main events of this period, Michel Friendly marks the following: “the development of highly interactive statistical computing systems, new paradigms of direct manipulation for visual data analysis, new methods for visualizing high-dimensional data, vastly increased computer processing speed and capacity, access to massive data problems and real-time streaming data” (2007).

In the 21st century business intelligence tools have become widespread and popular, thanks to which work with data has passed from the hands of technically and scientifically equipped people to the hands of business professionals. Organizations can use it to find insights and answers to various problems which allows to make data-driven decisions.

Programming has become a crucial technique in data analysis due to the increase in data generation. The automation and reduction of human error are made possible by programming-based approaches as opposed to spreadsheet-based technologies, such as Microsoft Excel. Consider contrasting the usage of Microsoft Excel with the R programming language to create a plot that displays the amount of data generated over time. Just around 10 lines of code in the R programming language would need to be entered. On the other hand, Microsoft Excel would need about the same number of clicks to create the identical plot. It would likely take about the same amount of time to create one plot. But it makes a difference if multiple similar plots are needed. In this case, the programming approach allows for more consistent plotting. More importantly, the code can easily see the parameters used for plotting. (Ouyang 2020.)

2.3 Data visualization in meteorology: history and practices

Modern meteorological visualization is comprehensive, simple, interactive. It is not limited to the air temperature only but is complimented by wind and wind gusts, air pressure, humidity, precipitation, visibility and cloud cover, snow depth, sunset and sunrise, auroras and space weather, and so on (based on Finnish Meteorological Institute’s forecast: en.ilmatieteenlaitos.fi). This data is shown hourly and is updated regularly, as often as changes in data are captured.

The history of meteorology starts from very ancient times. Egyptians believed in sky in a religious manner and performed “rain-making” rituals, the Babylonian astronomer-priests developed astrometeorology and predicted weather by searching the clouds’ color, moon position, etc. One of the “seven wise men” of antiquity, a Greek, Thales of Miletus, as early as in 585 B.C., predicted a solar eclipse. Thales’ followers – Anaximander and Anaximenes – found that thunder appeared because of air smashing against clouds. Another Greek philosopher, who was interested in meteorology, Anaxagoras, explained correlation between decreasing temperatures and increasing altitudes. Natural philosophers’ studies in meteorology were made without using any instruments,

so their research was rather qualitative. (Frisinger 1983, 2-10.)

Number of people interested in meteorology increased in the 19th century, so this area started to evolve quicky and research in this area became more distinct, divided by empirical, theoretical, and practical parts. Weather forecasting as a profession appeared in the 1870s. (Nebeker 1995.)

Before technological advances gave their benefits to the meteorological data visualization area, drawing and paintings have been the way weather was recorded. Up to 1400s, the attention of masters of visuals was on religion, which explained by weather being mild and therefore insignificant. But from 1400, more visual works depicted snow scenes started to appear (Image 4). (Keeling 2010, 126-127.) Further, this area only developed, and artists began to capture more and more different weather conditions and its components: the sky, the change of seasons, sunsets and sunrises.

Image 4. The Hunters in the Snow by Pieter Bruegel (bruegel2018.at 2022).

For quite a long-time weather forecasting was not considered as an everyday need but invention of such technologies as telegraph, barometer, thermometer changed the situation and led to the birth of the National Weather Service, the Meteorological Department of the Board of Trade of the United Kingdom, or the Met Office. The Met Office started publishing weather forecasts for the public by 1861. By 1870, the government of the Unites States started paying more attention to weather forecasting, and the International Meteorological Organization was created in 1873. (Teague & Gallicchio 2017, 22-25.)

The first daily weather chart (Figure 9), created by polymath and meteorologist Sir Francis Galton, was published in The Times in 1875. Thanks to him, a new era in the development of meteorological data visualization began and weather maps became an integral part of all newspapers. Galton also created icons associated with weather conditions (Figure 10). (Keeling 2010, 129-130). Galton also developed several novel and sophisticated mechanical devices for recording weather information (Figure 11) (galton.org).

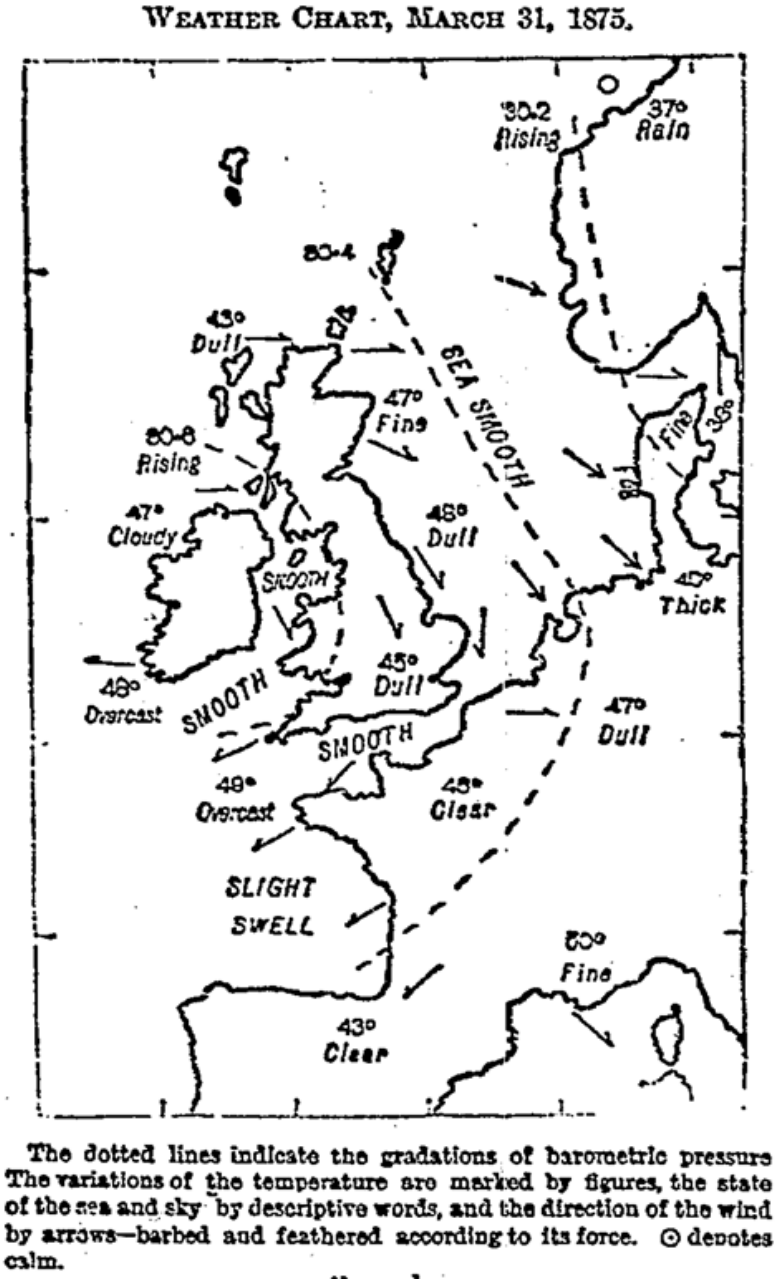


Figure 9. Galton's first weather map published in The Times in 1875 (galton.org 2022).

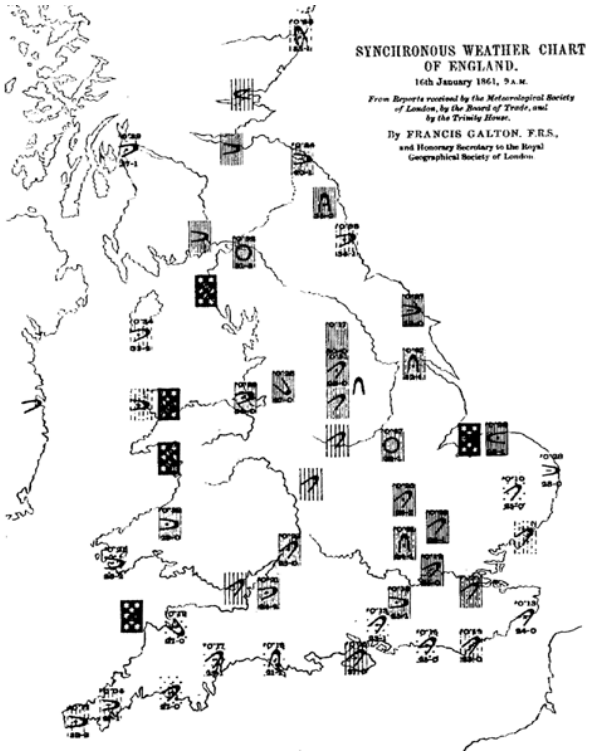


Figure 10. Galton's weather map with icons indicating weather conditions (galton.org 2022).

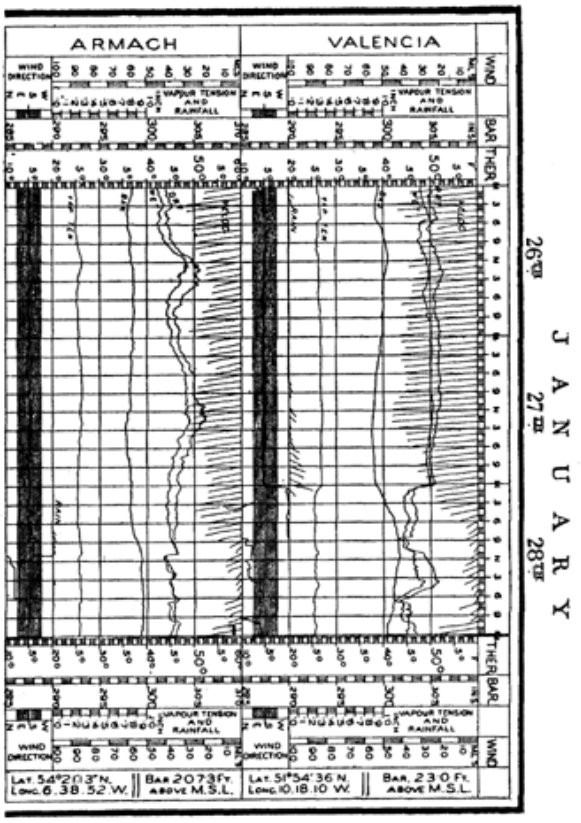


Figure 11. Information recorded by the Galton's instrument (galton.org 2022).

Meteorology was no longer explained by gods or based on guesswork by 1950. It took many years this area to develop and the governments to start paying attention but when the stars aligned – industrialization, theories and mathematical processes – the new ways of analyzing data as well as its representation became important and spread. (Teague & Gallicchio 2017, 28.)

The 1950s are notable for the rapid growth of technology, the emergence of television as a household item, and the computer as an essential business tool. The number of television stations only increased, and with them the desire for high ratings, which led to the development of ways to convey weather information: first in black and white, and then in color and with more technical graphics. Computers began to appear in weather services around the world, and together with advances in mathematics and technology, weather predictions were vastly superior to anything that had gone before. (Teague & Gallicchio 2017, 40.)

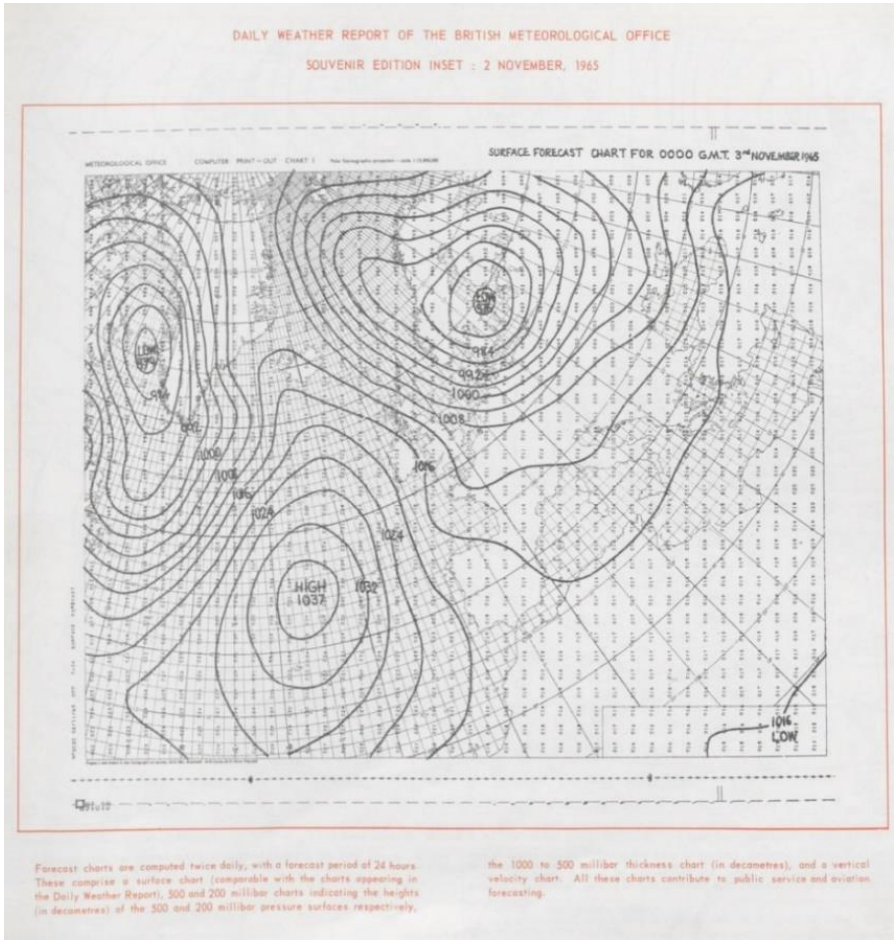


Figure 12. The first forecast issued using a computer by the Met Office on the 2nd of November 1965 (metoffice.gov.uk 2022).

The next years are highlighted by evolution of radar and satellites, which made the data gathered more accurate as well as weather forecasting.

Computer technology has greatly improved the systematic numerical processing of data. Moreover, computers themselves have evolved rapidly, making them more powerful for collecting big data and performing calculations. (Teague & Gallicchio 2017, 48.)

The modern history of meteorology visualization is characterized by computers becoming more widely available and cellphones and smartphones being an integral part of people's life. Personal computers allowed to share information faster which affected the way meteorological data was created, presented and distributed, and software and Internet development made scientists' precision even better. In addition to that the World Wide Web opened the whole new area of potential in broadcasting and distribution of weather information and news. (Teague & Gallicchio 2017, 55-56.)

2.4 Theoretical framework summary

In this chapter, the overview of the Python programming language was given which is shortly described as an object-oriented language, easy to learn, open-source, highly readable. The areas where Python is used are numerous and the focus of the theoretical part was concentrated on the Data Analysis. Library support, capacious code, wide community, and ease of integration of other programming languages and of multi-processing are some of the reasons Python is popular among data analysts.

The history of data visualization was discussed from ancient to modern times. Since the times of BC, people tend to visualize data and save the results of their research for further use. Starting from cuneiform tablets and tally sticks, data visualization transformed into a scientific area with variety of ways to realize: advanced technologies for technicians and scientists and business intelligence tools for almost any possible user.

The history of meteorological data visualization received a powerful impetus to development with the advent of the first devices for reading weather data. The next important event was recognition of the importance and necessity of weather monitoring by governmental institutions. This, in return, boosted technologies that made data more accurate. Not to mention television, computers and available Internet.

In the next chapter, data visualization libraries will be discussed and the widely used libraries in meteorology will be compared by features.

3 Carrying out the study

3.1 Usage of Python's libraries

In the traditional understanding, a library is book storage or a book collection. Library in programming is built around the same logic and is represented by precompiled codes for later use in specific areas. Libraries usage makes programming more accessible and faster due to having specific modules of codes, depending on needs, prewritten. The range of libraries is so wide so that it is most likely that majority of programming goals may be achieved with the help of them. The list of areas is as follows but not limited by:

- data science
- machine learning
- deep learning
- image manipulation
- scientific computing
- natural language processing.

NumPy is one of the most popular libraries of Python which is used for scientific computation. It was created in 2005 based on Numeric and Numarray libraries. Its widespread use is due to the presence of built-in mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more. These functions enable faster computation, and they support multidimensional data and large matrices.

One of the real examples of how NumPy works and how powerful it can be is its contribution to getting the first image of a black hole. This was achieved by the following capabilities of the library:

- it enables processing large sets of data; daily volume of data involved in the project was approx. 350 Terabytes
- it makes working with complex data concise; data from 8 telescopes was correlated
- it analyses data quickly and precisely (NumPy 2022).

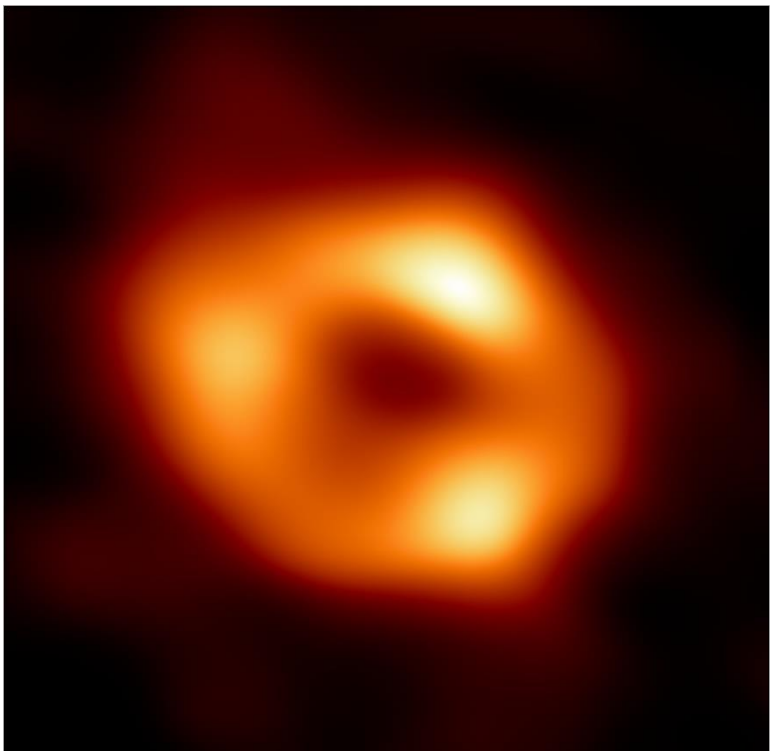


Image 5. First image of the black hole at the centre of the Milky Way (eventhorizontelescope.org 2022).

While NumPy is used for data evaluation in data science, such libraries as Pandas, Intake and others help extract, transform, and load data.

Pandas' history starts from 2008, when the library started to be developing. In 2009, the library becomes open sourced. Today, it is known as a package allowing manipulation of "relational" or "labeled" data by offering data structures and operations. It works well with different kinds of data: tabular, ordered and unordered time series data, arbitrary matrix data, and more (Pandas 2022). One of the simplest examples of using Pandas is analyzing historical data of sales: what month is profitable, who are main customers, what product was sold the most, etc.

Intake is officially described as a "a common platform to smooth the progression of data from developers and providers to users" (2018). The library has various components but due to its modular nature, those can be used as needed – altogether and/or separate. One of the components is a set of data loaders, and it enables turning any type of data into a data structure by one-call investigation or loading locally or remotely. A cataloging system allows to list data sources and their metadata and parameters, indicating which driver each should load. Looking for catalogs and navigation in them, investigation of data sources, and plotting predefined visuals are permitted by GUI layer (Intake 2018).

Exploratory data analysis – a method that uses data visualization techniques in order to get insightful summary out of a dataset – is performed in Python with Seaborn, Matplotlib, Plotly, and

others.

Matplotlib is a visualization library that makes 2D plots of arrays. Ideology of this library is explained by its creator John D. Hunter as giving an ability to create a simple plot with just a few, or even one, commands (John D. Hunter 2008). One day John was specifically looking for a tool that will help him to create publication quality plots, which did not happen, and that is how the history of Matplotlib started. Today this library makes possible to create interactive figures with zooming, panning and updating features, with unique style, and exported to different file formats. (Matplotlib 2022.) Some of the examples of visuals are presented in the figures 13-16 below.

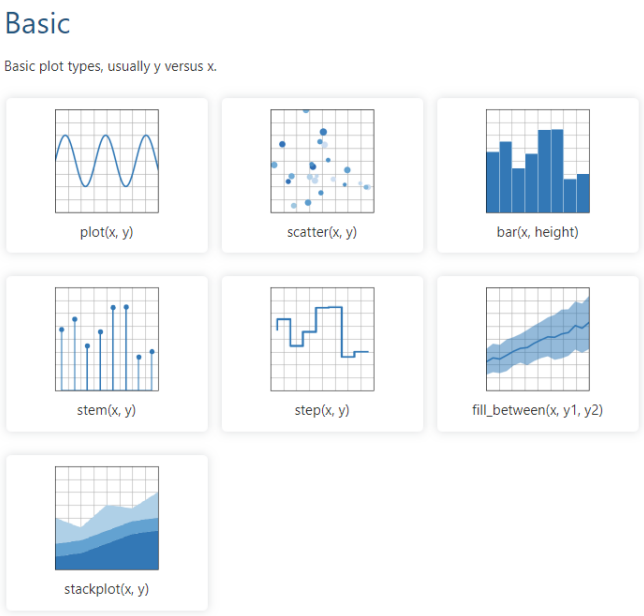


Figure 13. Basic plot types (matplotlib.org 2022).

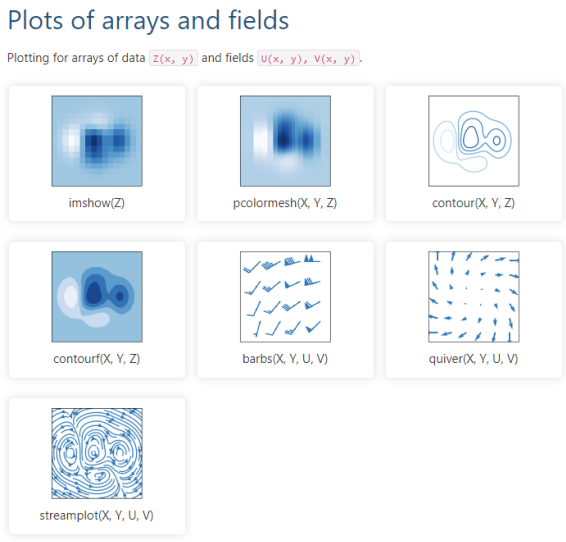


Figure 14. Plots of arrays and fields visualization (matplotlib.org 2022).

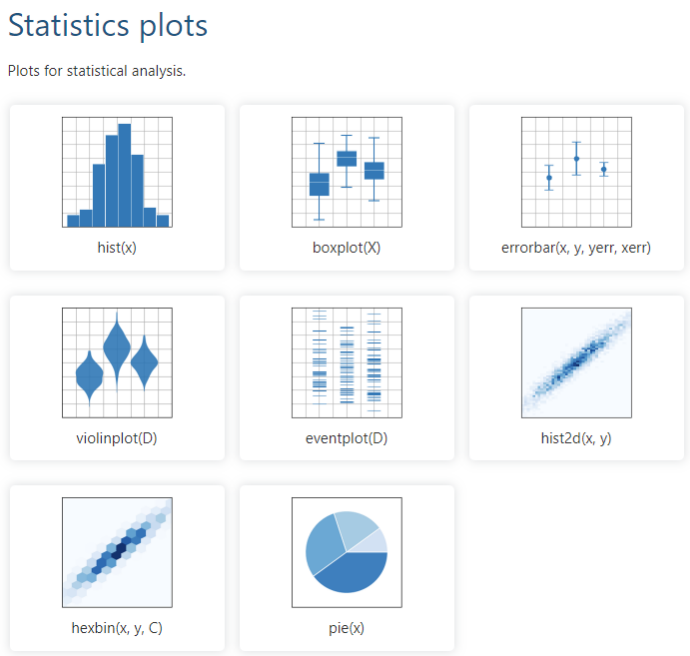


Figure 15. Statistics plots visualization (matplotlib.org 2022).

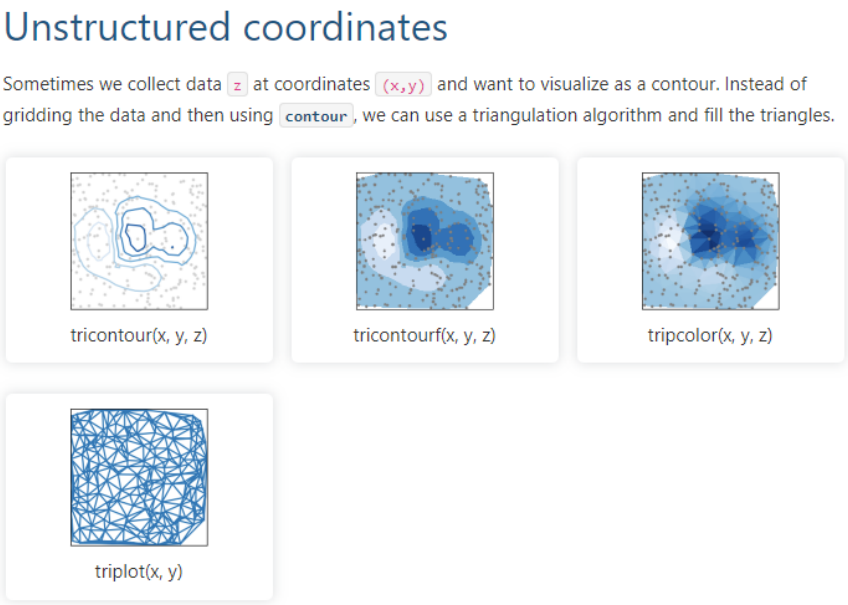


Figure 16. Unstructured coordinates visualization (matplotlib.org 2022).

Seaborn – a library that is based on Matplotlib – “provides a high-level interface for drawing attractive and informative statistical graphics” (Seaborn 2022). Seaborn is written on top of the Matplotlib library, but offers a higher-level interface, which means that the boilerplate code that forms the appearance of the plots is hidden inside. It only takes one or two lines of code to create a chart more than suitable for quick analysis of data, and if it is needed to prepare a chart for presentation or publication, Seaborn also provides access to low-level settings. Another important

feature of the Seaborn library is its data preprocessing mechanism, which is possible due to tight integration with the Pandas library. Data can be passed directly in DataFrame objects, and Seaborn itself will do all the necessary work: aggregation (selection of subsets), statistical processing (for example, calculation of confidence intervals) and visual highlighting of the results. In the same way that Python allows to focus on the task and not the code that solves it, Seaborn allows to focus on the important nuances of the visual representation of data, and not the code with which these nuances need to be drawn and highlighted. Perhaps the only disadvantage of Seaborn is that the resulting graphs are not interactive: no possibility to hover a mouse over a point and see the value corresponding to it (PyProg Semyon Lukashevskiy 2018-2022).

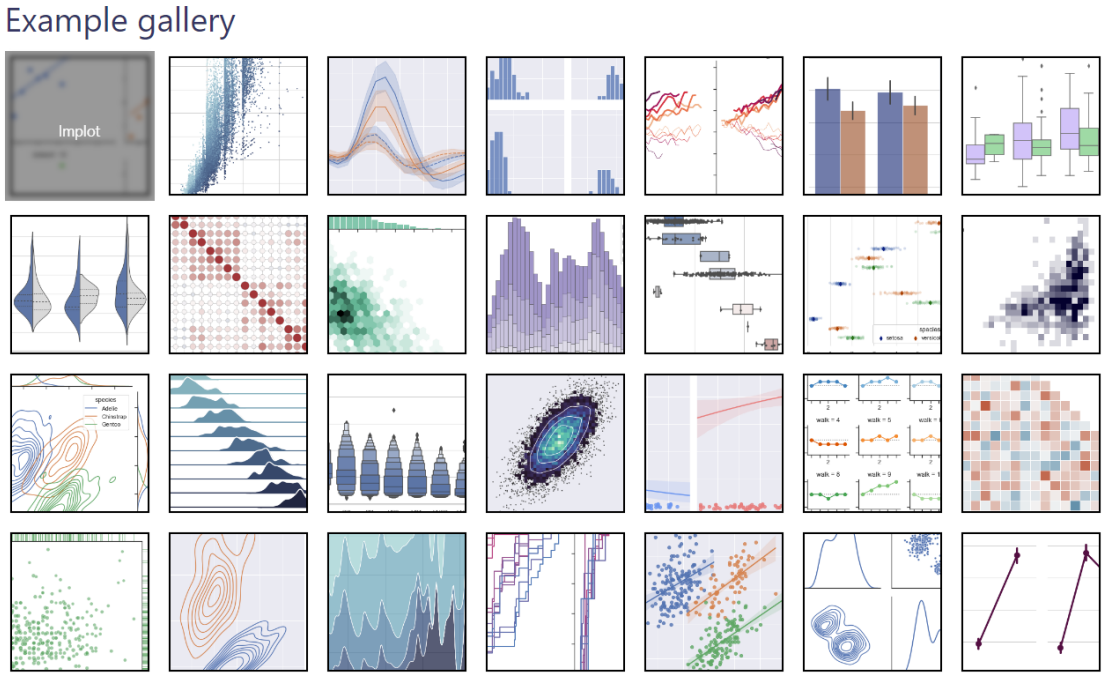


Figure 17. Examples of visuals created using Seaborn (seaborn.pydata.org 2022).

Plotly – an open-source graphing library – works and is built not only for using in Python but also in R, Julia, JavaScript (Python's Plotly is based on the JavaScript's library), MATLAB, ggplot2, F#, and Dash. It makes interactive visualizations (of over 40 different types) of publication-quality. Its key advantage over seaborn and matplotlib is the convenience of building complex interactive visualizations – full-fledged mini-applications that make the result of work more acceptable to the end user. Visuals can be saved as a single HTML file, be part of Python-built web apps using Dash or be displayed in Jupyter notebooks (Plotly 2022).

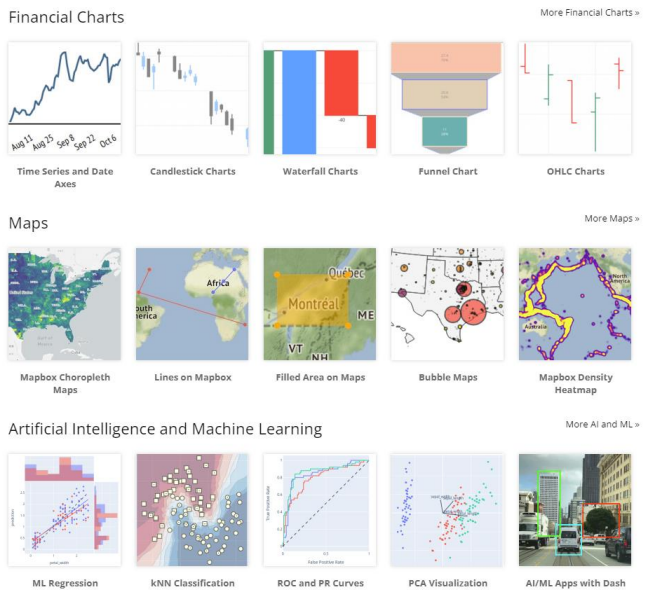


Figure 18. Examples of visuals created using Plotly (plotly.com 2022).

Of course, the list of Pithon's data visualization libraries is not limited to those listed above. There are also such as Ggplot, Altair, Bokeh, Pygal, Geoplotlib, Folium, Gleam, and others. It is important to note that data visualization libraries do not work as a standalone data analysis solution. To make a visual analysis, it is must to first load the data, perform the necessary transformations, check whether the individual pieces of data work correctly and together, and only after these steps visualization will work correctly and show reliable results. A simple example of how different libraries can interact with each other: first, Pandas is used to unload the data, then Matplotlib is used to read the data, Numpy is used to check the data, and then visualization occurs using Plotly (Figure 19).

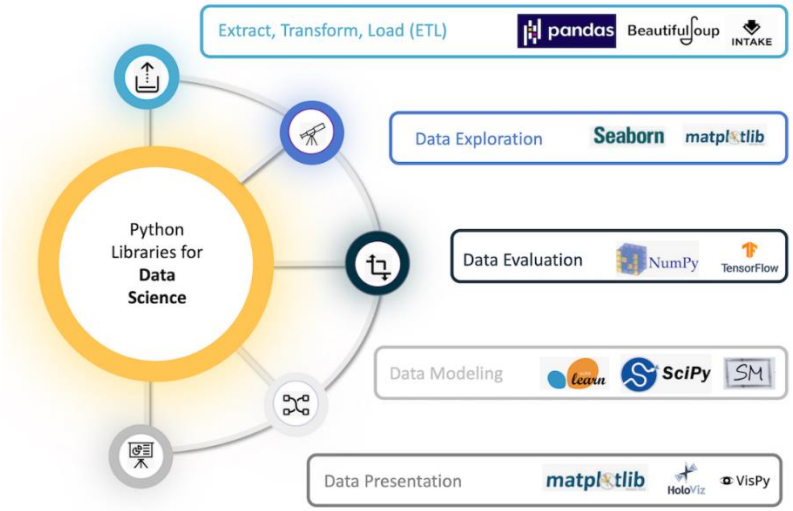


Figure 19. Different Python’s libraries to work with data (numpy.org 2022).

3.2 Python libraries for visualizing meteorological data

Meteorological data can be presented in different ways: bars, charts, maps, and others. Some of the most important metrics in meteorology are air temperature, humidity, atmospheric pressure, wind, rainfall; these factors can be presented together in different combinations, or separately (Hakštok & Mihajlović 2014, 423).

Python has hundreds of thousands of different libraries, and it is not known for certain how many of them are for visualization. In this chapter, such libraries as MetPy, Alarconpy, Iris, and CliMetLab will be covered.

3.2.1 MetPy

MetPy is a collection of tools for reading, visualizing, and performing calculations with weather data. This package was built on top of the interdisciplinary scientific Python software stack – NumPy, SciPy, Matplotlib, xarray, Dask, and Cartopy. These libraries ensure numeric arrays for fast computation, scientific algorithms, publication-quality plotting, and data structures and algorithms for tabular data, parallel computing, and geospatial data processing, respectively. This combination of libraries eases the process of building domain-specific tools by eliminating the need to make core functions from the beginning (May, Goebbert, Thielen, Leeman, Camron, Bruick, Bruning, Manser, Arms & Marsh 2022, 2-6).

Creation of this toolset started in 2008 with intention to replace legacy tools such as GEMPAK – an analysis, display, and product generation package for meteorological data – by bringing the best features of it to Python (May & Leeman 28 November 2017; Unidata Data Services and Tools for Geoscience, UCAR Community Programs 2022). The main areas of MetPy are:

- plotting: meteorologists create certain types of plots, some of which are hard to plot in most plotting software (Figure 20)
- reading data files: MetPy supports different file formats that researchers may encounter when analysing meteorological data
- calculations: meteorology-specific calculations are usually done manually by scientists while MetPy has in-built tested calculations the number of which is only growing (UCAR/Unidata 2020).

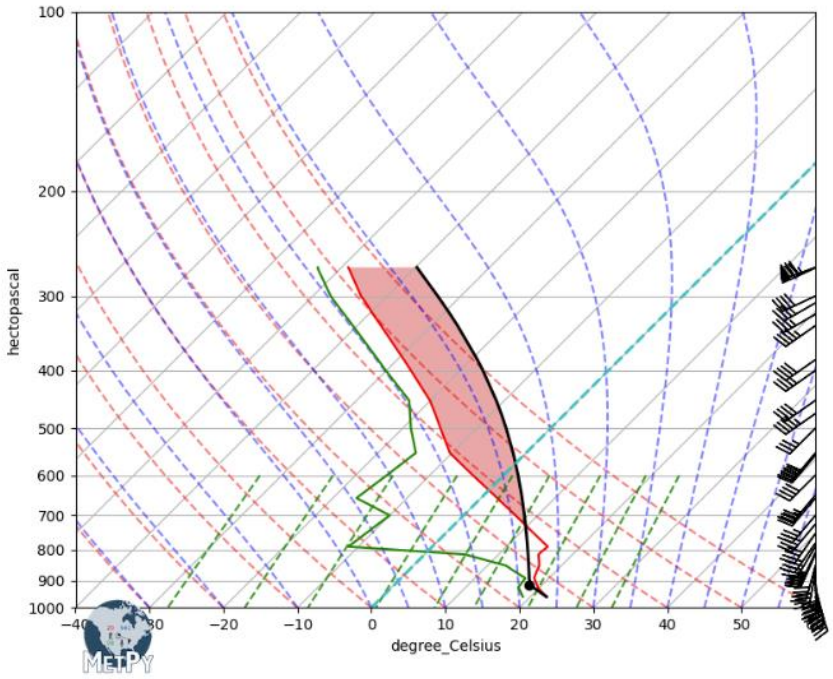


Figure 20. Skew T-log p plot created using MetPy (unidata.github.io 2022).

Today's MetPy is highly efficient but for many years replacing GEMPAK with Python was limited by inability to plot surface and upper-air observation data using a station model. This limitation has been overcome: station plots, layering of different elements together, and domain-specific fonts are available in the library enabling observations of atmospheric phenomena in complex figures (Figure 21) (May et. al. 2022).

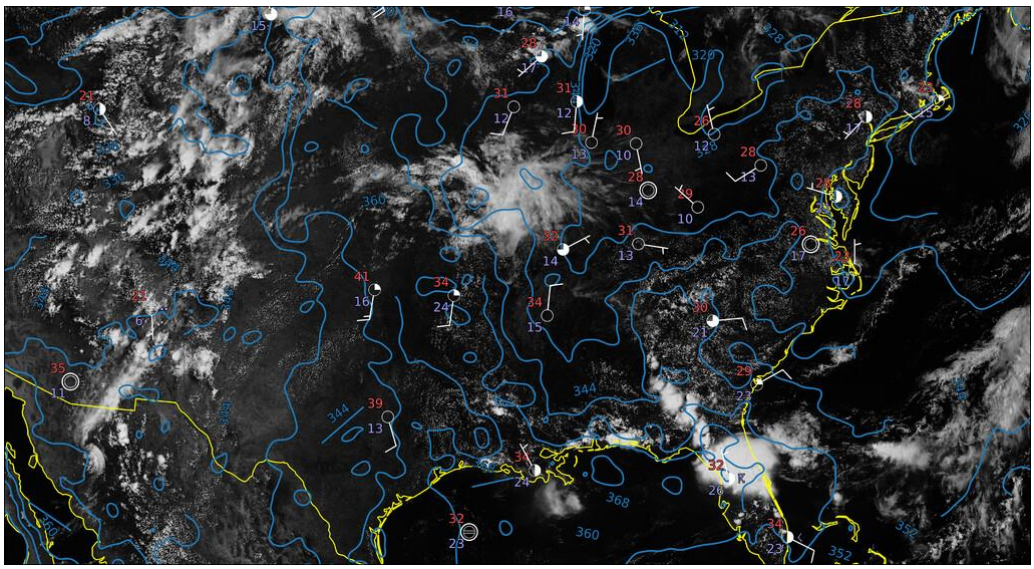


Figure 21. Bulletin of the American Meteorological Society 103, 10; 10.1175/BAMS-D-21-0125.1: "Map of the continental United States with background GOES-16 channel 02 imagery valid 1926 UTC 27 Jun 2022, overlayed with contours of equivalent potential temperature calculated from Real-Time Mesoscale Analysis

(RTMA) output and station models of surface observations from a collection of surface observation METARs" (journals.ametsoc.org 2022).

The calculation package allows to perform calculations in the range from single values to one-dimensional and multidimensional arrays. Also, the library has a built-in data interpolation function, including Cressman and Barnes-style interpolations traditionally used in meteorology.

MetPy's coherent interface does not require knowledge of the Python stack to create visualizations. Interpretation of a small number of plot and data attributes into a plot with a "sensible" default presentation is allowed by the declarative syntax in order to make the code much shorter (traditional syntax is almost twice longer) (Figure 22) (May et. al. 2022).

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt

level = 300 * units.hPa

ds_subset = ds[
    [
        "Geopotential_height_isobaric",
        "wind_speed",
        "u-component_of_wind_isobaric",
        "v-component_of_wind_isobaric",
    ]
].metpy.sel(isobaric3=level)

plot_crs = ccrs.LambertConformal(
    central_latitude=40, central_longitude=-100,
standard_parallels=[30, 60]
)

data_crs = ds_subset["metpy_crs"].metpy.cartopy_crs

fig = plt.figure(figsize=(15, 15))
ax = fig.add_subplot(projection=plot_crs)

c = ax.contour(
    ds_subset["lon"],
    ds_subset["lat"],
    ds_subset["Geopotential_height_isobaric"],
    levels=list(range(0, 10000, 120)),
    transform=data_crs,
    colors="k",
)

c.clabel(inline=1, fmt="%.0f", inline_spacing=8,
use_clabeltext=True)

cf = ax.contourf(
    ds_subset["lon"],
    ds_subset["lat"],

 ds_subset["wind_speed"].metpy.convert_units("knots"),
    levels=list(range(10, 201, 20)),
    cmap="BuPu",
    transform=data_crs,
)

fig.colorbar(cf, orientation="horizontal", pad=0,
aspect=50)

x_slice = slice(None, None, 3)
y_slice = slice(None, None, 3)

ds_subset = ds_subset.sel(lon=x_slice, lat=y_slice)

ax.barbs(
    ds_subset["lon"],
    ds_subset["lat"],
    ds_subset["u-
component_of_wind_isobaric"].metpy.convert_units("knots"
).data,
    ds_subset["v-
component_of_wind_isobaric"].metpy.convert_units("knots"
).data,
    pivot="middle",
    transform=data_crs,
)

ax.set_extent((-125, -74, 20, 55), ccrs.PlateCarree())

ax.add_feature(cfeature.BORDERS)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.STATES)

ax.set_title(f"{level:~P} Heights and Wind Speed at
{dt_string}")
```

```
from metpy.plots import (
    BarbPlot,
    ContourPlot,
    FilledContourPlot,
    MapPanel,
    PanelContainer,
)

contour = ContourPlot()
contour.data = ds
contour.field = "Geopotential_height_isobaric"
contour.level = 300 * units.hPa
contour.contours = list(range(0, 10000, 120))
contour.clabels = True

cfill = FilledContourPlot()
cfill.data = ds
cfill.field = "wind_speed"
cfill.level = 300 * units.hPa
cfill.contours = list(range(10, 201, 20))
cfill.colormap = "BuPu"
cfill.colorbar = "horizontal"
cfill.plot_units = "knot"

barbs = BarbPlot()
barbs.data = ds
barbs.field = ["u-component_of_wind_isobaric", "v-
component_of_wind_isobaric"]
barbs.level = 300 * units.hPa
barbs.skip = (3, 3)
barbs.plot_units = "knot"

panel = MapPanel()
panel.area = [-125, -74, 20, 55]
panel.projection = "lcc"
panel.layers = ["states", "coastline", "borders"]
panel.title = f"{cfill.level:~P} Heights and Wind
Speed at {dt_string}"
panel.plots = [cfill, contour, barbs]

pc = PanelContainer()
pc.size = (15, 15)
pc.panels = [panel]
```

Figure 22. Left: Figure and map creation using Matplotlib and Cartopy. Right: MetPy's declarative plotting syntax (journals.ametsoc.org 2022).

3.2.2 Alarconpy

Alarconpy is another package consisting of different libraries for meteorological data visualization:

- NumPy: scientific computing
- MetPy: meteorological data analysis with plots, calculations, file I/O
- Cartopy: geospatial data processing
- SciPy: numerical integration, interpolation, optimization, linear algebra, statisctics
- NetCDF4: read and write files in netCDF (Network Common Data Form for storing multidimensional scientific data) 3 and 4 formats, create readable files by HDF5 (Hierarchical Data Format supporting large, complex data) clients
- Matplotlib: data visualization
- Time: time-related functions (Perez-Alarcon, Fernandez-Alvarez 2021, 2-3).

Creation of geographic maps suing Cartopy inside Alarconpy is much faster by a shorter code (Figure 23). A map can be visualized in one line of practical code, while direct Cartopy's syntax requires using at least 10 lines of code (Perez-Alarcon, Fernandez-Alvarez 2021, 3-4).

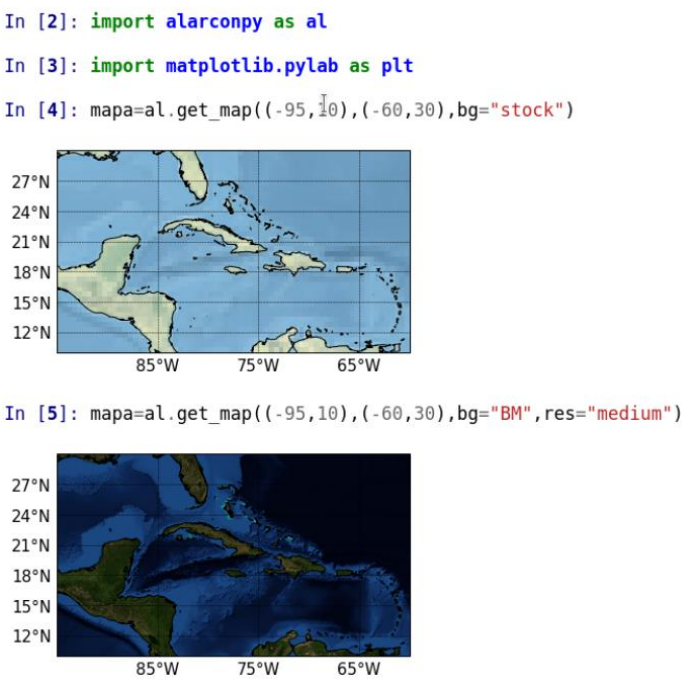


Figure 23. Alarconpy's code and output with different backgrounds: default Cartopy in the top, Blue Marble in the bottom (apalarcon.github.io 2022).

Unix Time format – 1 January 1970 00:00:00 UTC – is one of the most common time formats contained in meteorological data sets. The package consists of a function for time formatting from the Unix Time format to the Julian's as well as calculating the total hours between two dates.

Forecast as a prediction of weather is done by assistance of interpolation. SciPy's and MetPy's presence in the Alarconpy package allows to use such interpolation methods as linear, nearest, cubic, rbf, natural_neighbor, barnes, and cressman (Perez-Alarcon, Fernandez-Alvarez 2021, 4).

Unit conversion in acceleration, angle, area, the moment of inertia, density, length, mass, temperature, and velocity is done in Alarconpy. Different meteorological variables are shown using color palettes in correspondence with the Cuban Institute of Meteorology (Figure 24) (Perez-Alarcon, Fernandez-Alvarez 2021, 5).

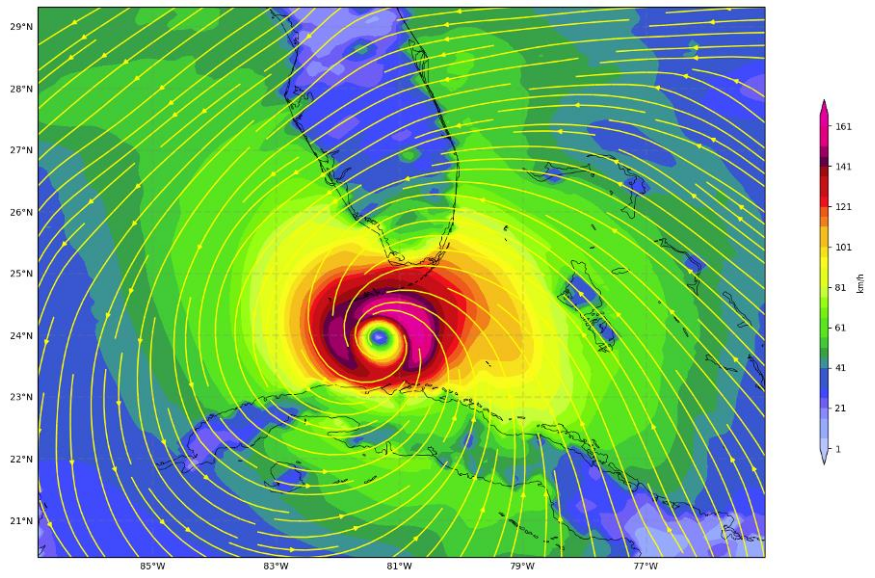


Figure 24. The Hurricane Irma's stream flow created using Alarconpy (apalarcon.github.io 2022).

The description of the radial wind pattern of tropical cyclones is a built-in function in the Python Alarconpy package. In addition, it is possible to study the risks associated with the impact of a tropical cyclone on certain geographical areas (Perez-Alarcon, Fernandez-Alvarez 2021, 6).

Meteorological data are often stored in large volumes and their analysis is a complex multi-step process. The package's built-in function to determine the indexes of a particular string of characters in a list of strings makes it possible, for example, to find entries for Hurricane Irma in the National Hurricane Center's database (Perez-Alarcon, Fernandez-Alvarez 2021, 7).

The work on this package is still ongoing as its authors – Albenis Pérez-Alarcón and José Carlos Fernández-Alvarez – started working on Alarconpy in 2021. The package is openly available on github.

3.2.3 Iris

Iris – a package for analyzing and visualizing meteorological and oceanographic data – is part of the open tools' collaborative called SciTools which produces open-source tools for Earth scientists. All the packages are open-sourced, and the direction of development is decided by the community (SciTools, 2022). Iris, whose data model is based on the Climate and Forecast Metadata Conventions, – geoscience data description conventions designed to facilitate the processing and exchange of data files – is a format-independent interface for efficient data manipulation. Originally developed by the UK Met Office, Iris simplifies the processing of file formats such as NetCDF, GRIB and others, which was previously done manually by writing code from scratch. Since there are many file formats, scientists used to have to use separate packages for each type of format, which led to rewriting the same or very similar code. In addition, weather data operations such as interpolation, merging, subset extraction, etc. are very similar, so Iris was designed to combine these operations and provide a data format independent package (Coding club, github.io).

As are the packages already discussed in this chapter, Iris is based on the Python's libraries such as Matplotlib and Cartopy, which enable visualization interface. In addition, this package can scale from small single-machine processes to multi-core clusters and high-performance computing thanks to NumPy and Dask. The data is represented by using the following features of Iris:

- unit conversion
- subsetting and extraction
- merge and concatenate
- aggregations and reductions
- interpolation and regridding
- operator overloads (SciTools-Iris, 2022).

Some of the examples of visualizations made with Iris are Polar stereographic plot (Figure 25), Coloring anomaly data with logarithmic scaling (Figure 26), Ionosphere space weather (Figure 27).

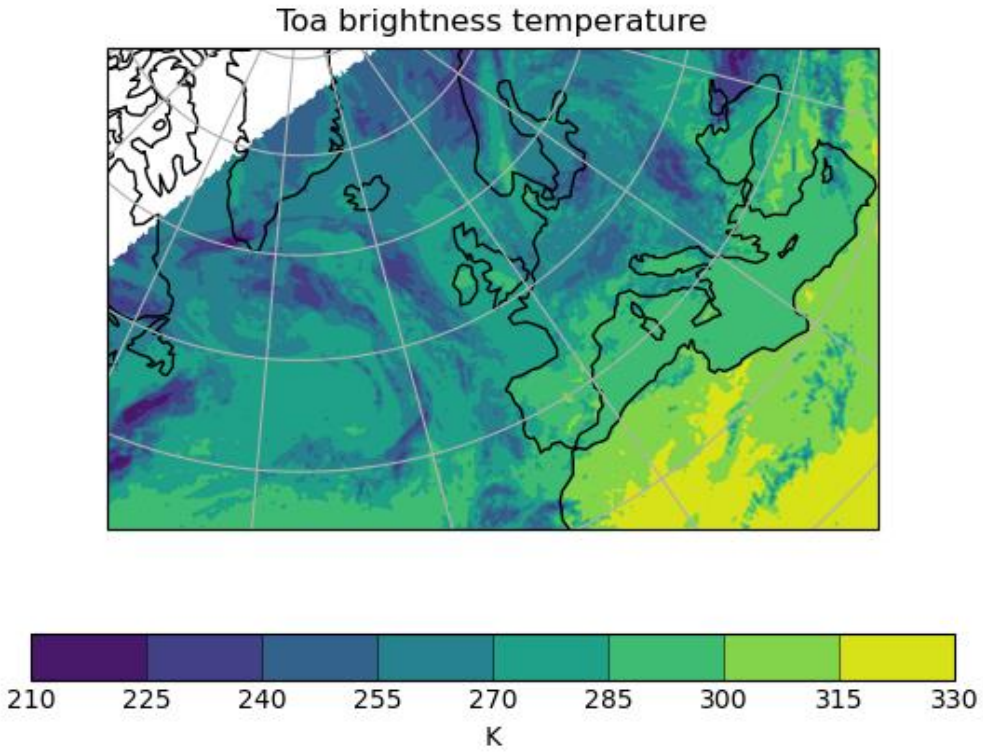


Figure 25. Polar stereographic plot (scitools-iris.readthedocs.io 2022).

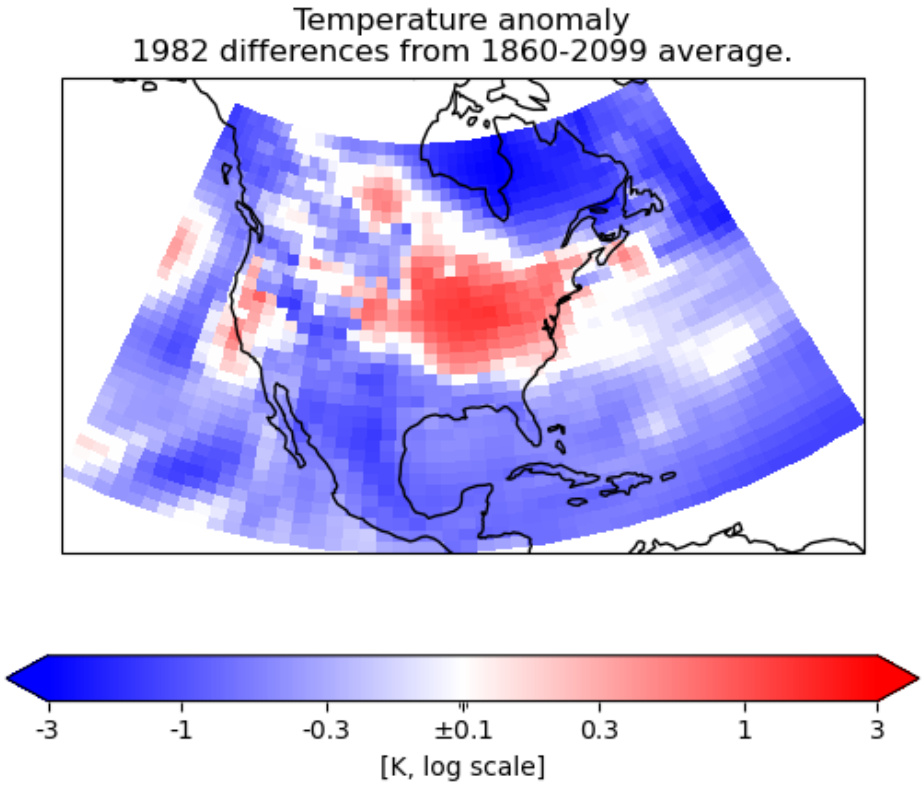


Figure 26. Coloring anomaly data with logarithmic scaling (scitools-iris.readthedocs.io 2022).

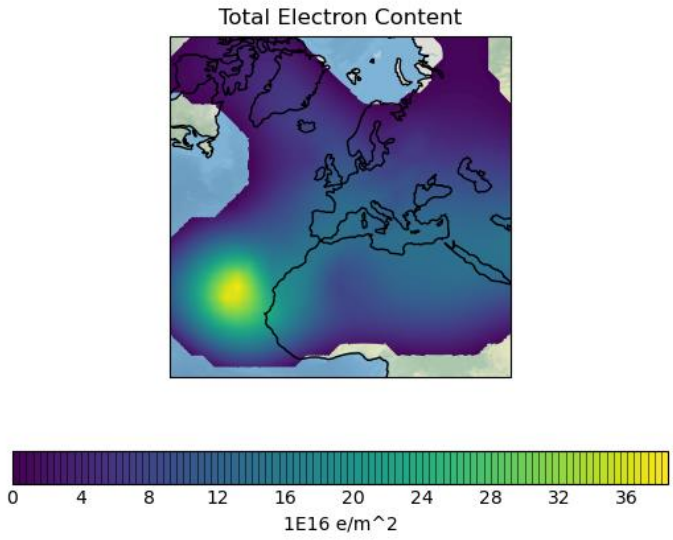


Figure 27. Ionosphere Space Weather (scitools-iris.readthedocs.io 2022).

3.2.4 CliMetLab

CliMetLab is a Python package whose main goal is to significantly reduce boilerplate code by providing high-level, unified access to weather and climate datasets. Loading, caching, and transforming data in NumPy, Pandas or xarray is done automatically with further transfer to scientific packages such as SciPy and TensorFlow. Simplification of the construction of 2D maps is possible due to the automatic selection of the most appropriate styles for the given data (CliMetLab 2022).

To achieve the goal, which is to ease access to climate and meteorological datasets, CliMetLab introduces the concepts of Data source and Dataset, where Data sources are various access methods (Figure 28).

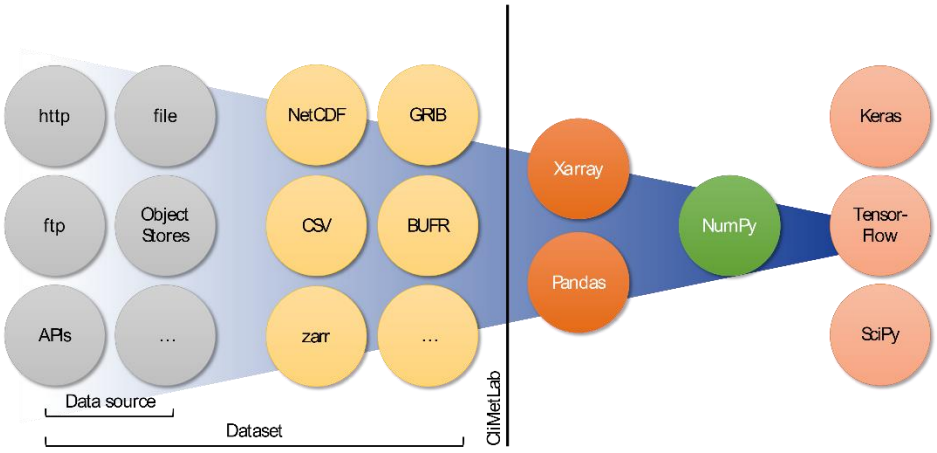
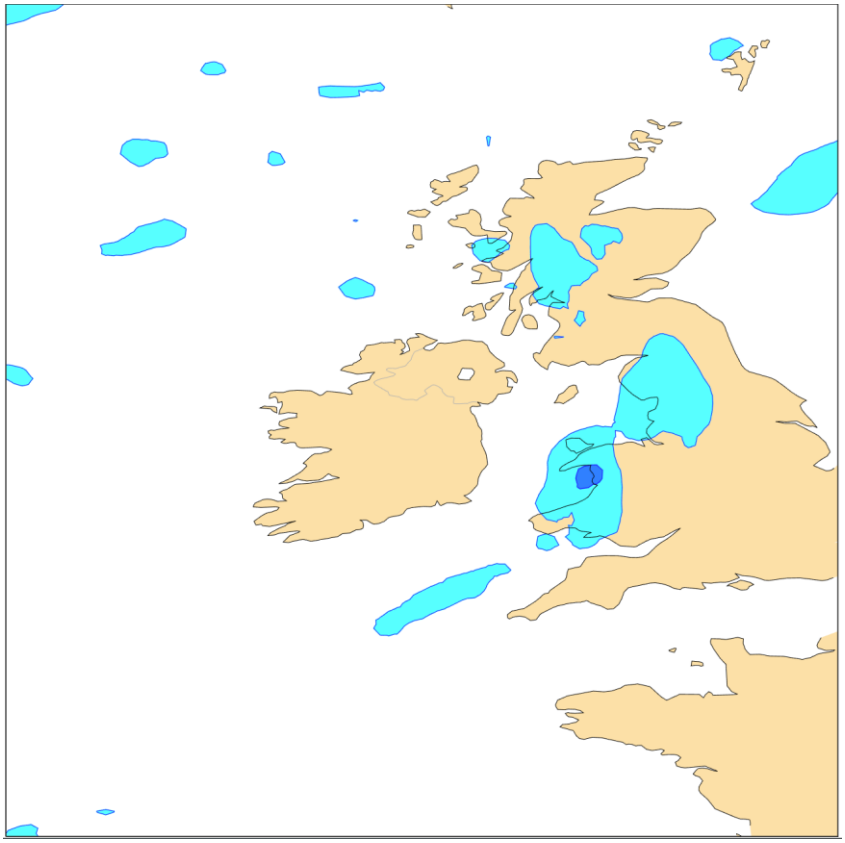


Figure 28. The concept of CliMetLab (climetlab.readthedoc.io 2022).

Very high-level map plotting facilities are available in CliMetLab which by default are selected automatically based on a dataset (Figure 29). Users can also control the mapping with their own choices (CliMetLab 2022).



```
import climetlab as clm

data = clm.load_dataset("some-dataset")
cml.plot_map(data)
```

Figure 29. Automatically generated map using CliMetLab and the code required to create it (climetlab.readthedocs.io 2022).

3.2.5 GeoPandas

An open-source library called GeoPandas adds support for geospatial data to the Pandas library. Information that defines objects, events or other features in relation to a location is known as geospatial data.

The library's history began in 2013 at the SciPy Conference, when Kelsey Jordahl created the project. In July 2014, the initial version was made available. Joris Van den Bossche received the lead and maintenance two years later. GeoPandas got two Small Development Grants to aid in its development in 2020 after being a NumFOCUS Affiliated Project.

The Pandas, Shapely, Fiona, and Pyproj packages are prerequisites for GeoPandas. Such spatial operations on geometric types that produce new spatial data (for example, a roadmap) from defined input data are permitted by the datatypes extension for Pandas. Shapely makes geometric operations possible, Fiona ensures file access, and Matplotlib is responsible for plotting. This library contains two data objects:

- GeoSeries based on the Pandas Series object
- GeoDataFrame based on the Pandas DataFrame object (GeoPandas, 2022).

From the name of the data objects, they are used to process spatial data. Their derivation from Pandas data objects allows the use of the same properties for selecting data, such as .loc and .iloc, where the former defines columns and rows by their labels and the latter by their integer position values. (Packt Hub, Kumaraswamy 2018.)

Plotting maps is the core feature of GeoPandas. On top of this, many additional elements can be imposed that reveal the essence of the functionality possibilities. For example, below is a visualization of captured fireballs (Figure 30). Three libraries were used to draw this picture: Pandas for reading files, Matplotlib for plotting and GeoPandas for spatial visualization (Fizell 2022).

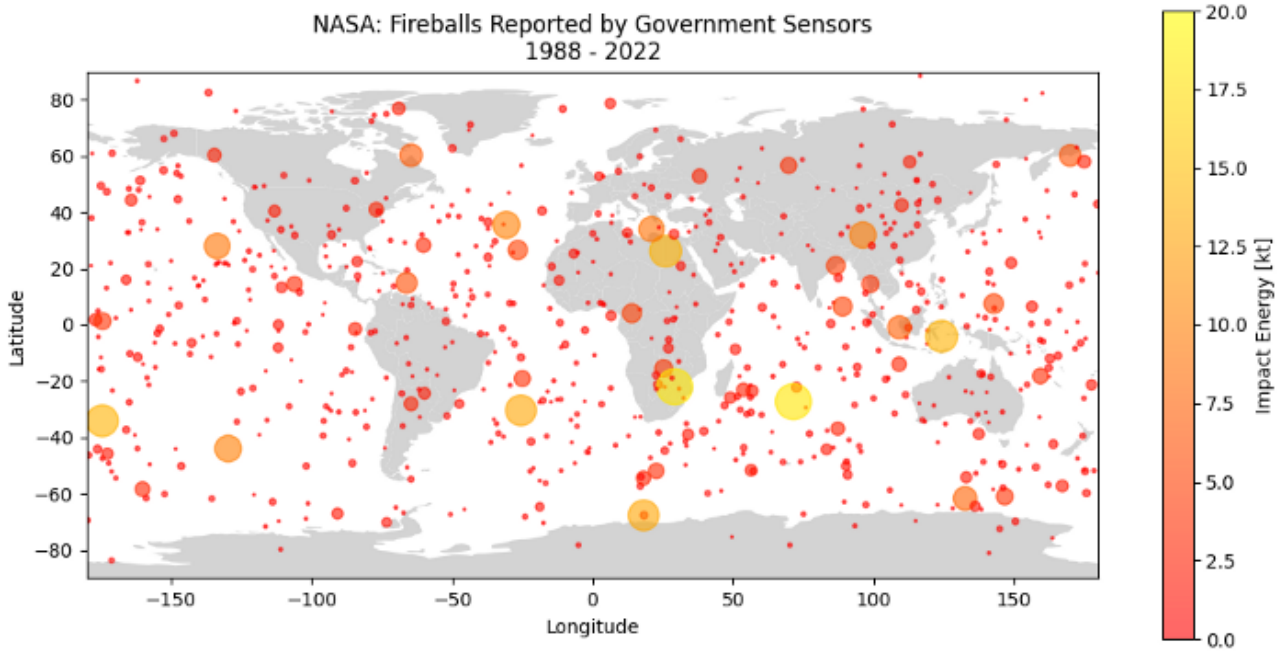


Figure 30. Example of the GeoPandas-based output illustrating the statistics of captured fireballs (Fizell 2022).

3.3 Hands-on examples

For the practical part, Google Colaboratory was used as an editing tool as it can be run in a browser and does not require installation of any software and additional components. The visual output of the executed code that is represented in this chapter by smaller pieces, step-by-step, is available in the appendices.

3.3.1 CliMetLab

As the first example, CliMetLab shows a visualization of hurricane's track. CliMetLab has in-built hurricane database that can be easily accessed. All the actions are performed based on the examples from the official documentation's website.

The first step of the visualization process using Google Colab is to install CliMetLab and Matplotlib packages. Matplotlib is needed to plot a map that will show the track of a hurricane.

```
!pip install --quiet climetlab matplotlib
```

Next, importing the package.

```
import climetlab as cml
```

When the environment is set up, the database can be accessed. The dataset must be loaded first (ehe parentheses indicate the name of the database and its separate part, which will be used).

```
pacific = cml.load_dataset("hurricane-database", bassin="pacific")
```

Getting the Pandas frame is the next step. Hurricane data is non-gridded, and accessing it using CliMetLab requires Pandas, and this will also allow to specify the date and area later.

```
df = pacific.to_pandas()
df.bassin.unique()
```

To check specific data, which in this case is hurricanes with intensity of 5, in a tabular format, the below code must be executed:

```
df[(df.category == 5)]
```

This piece of code helps retrieve a table of hurricane data with an intensity of 5, namely hurricane basin, amount, year, name, time, type, status, longitude, latitude, category, and pressure (check Appendix 1 for an output).

The full code to get access to the database and retrieve data in a tabular format using Google Colab is as follows:

```
!pip install --quiet climetlab matplotlib

import climetlab as cml

pacific = cml.load_dataset("hurricane-database", bassin="pacific")

df = pacific.to_pandas()
df.bassin.unique()
df[(df.category == 5)]
```

The idea is to show the hurricane's track. The Walaka hurricane, that happened in Hawaii in September 2018, was chosen as an example. To get the records of it, the following code was executed (check Appendix 1 for the output):

```
walaka = df[(df.name == "walaka") & (df.year == 2018)]
walaka
```

To get a picture of how the hurricane moved, the plot.map function must be used:

```
cml.plot_map(walaka)
```

The final picture of a map with default styles is as follows.

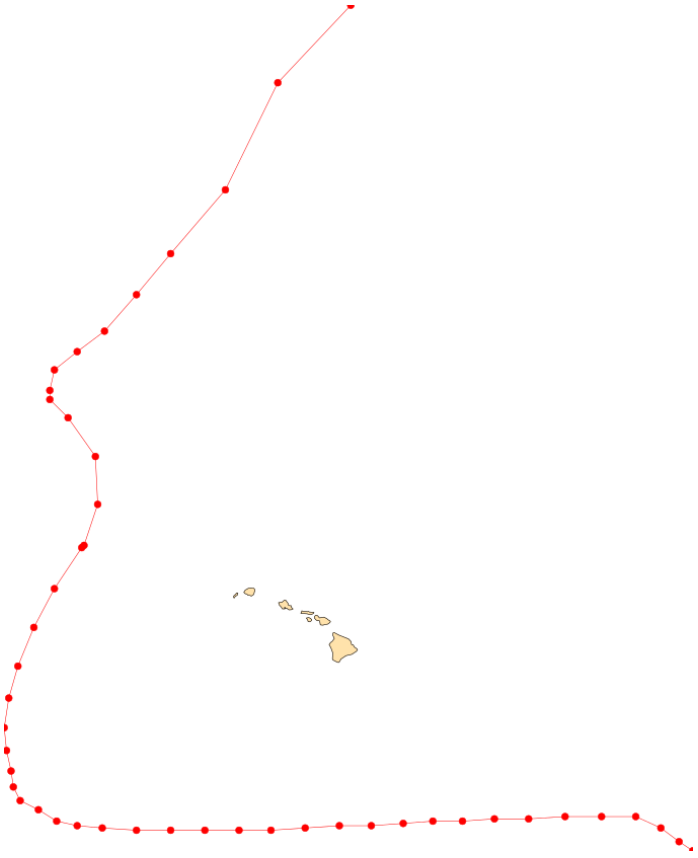


Figure 31. The final plot showing the hurricane Walaka's path over Hawaii.

Writing just three lines of visualization-related code (see below) allows building visualizations in CliMetLab easily and quickly. It helps make very first steps in exploring meteorological data visualization in Python painlessly. The code can be modified to perform more complex actions, as for example, showing different hurricanes of one year on one picture using different colors.

```
!pip install --quiet climetlab matplotlib

import climetlab as cml

pacific = cml.load_dataset("hurricane-database", bassin="pacific")

#df = pacific.to_pandas()
#df.bassin.unique()
#df[(df.category == 5)]

walaka = df[(df.name == "walaka") & (df.year == 2018)]
cml.plot_map(walaka)
```

As CliMetLab is still under development, some of the features do not work properly or are not implemented yet. For example, there are certain limitations in styling:

- Background and foreground options are limited to default and land-sea

```
import climetlab.plotting
# List of possible backgrounds/foregrounds
for p in climetlab.plotting.layers():
   print(p)
```

```
default-background
default-foreground
land-sea
```

Figure 32. List of possible backgrounds and foregrounds in CliMetLab.

- Plotting styles are either default or no-style with a couple of options

```
import climetlab.plotting
# List of possible styles
for p in climetlab.plotting.styles():
  print(p)
```

```
cyclone-track
default-style-fields
default-style-observations
land-sea-mask
no-style
orography
rainbow-markers
```

Figure 33. List of possible styles in CliMetLab.

– Plotting projections is the most comprehensive part of plotting styles in this library.

```
import climetlab.plotting
# List of possible projections
for p in climetlab.plotting.projections():
    print(p)
```

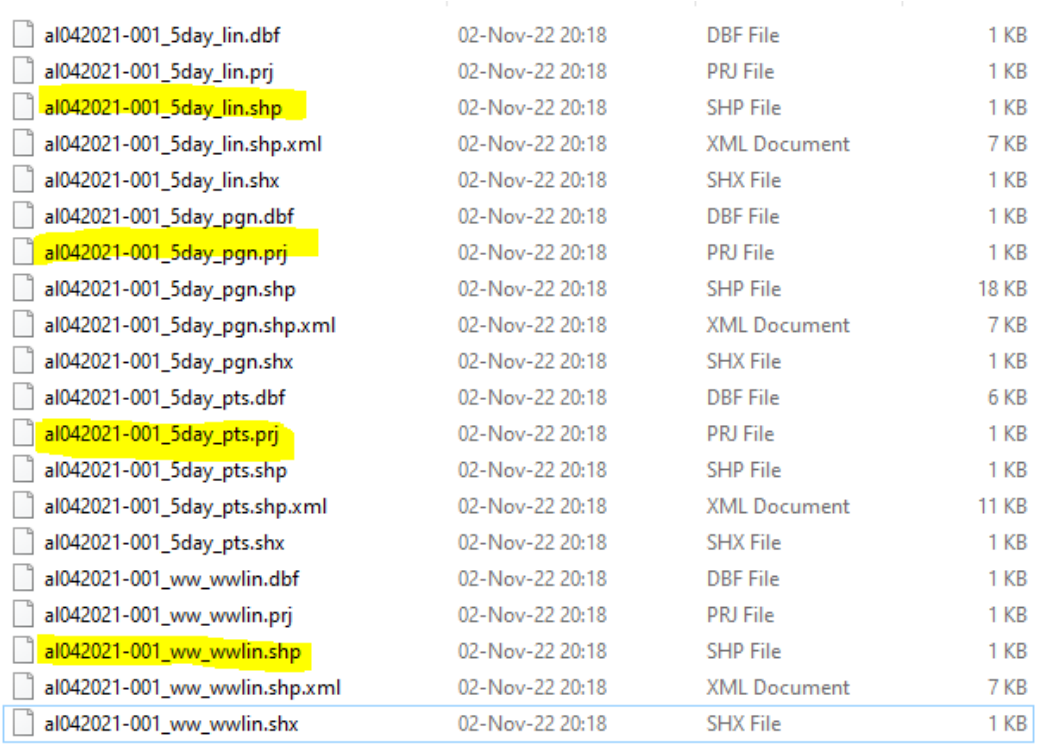
```
africa
asia
bonne
collignon
euro-atlantic
europe
europe-cylindrical
global
goode
mercator
mollweide
north-america
north-america1
north-atlantic
north-hemisphere
polar-north
robinson
south-america
south-atlantic
south-hemisphere
south-pacific
tropics-east
tropics-west
web-mercator
```

Figure 34. List of possible projections in CliMetLab.

3.3.2 GeoPandas

The following practice was realized based on the workshop by Unidata – a “community of education and research institutions with the common goal of sharing geoscience data and the tools to access and visualize that data” (Unidata, 2022). The workshop is openly available on YouTube and is performed by John Leeman – a software engineer at Unidata.

In this example, the open archived data of The Tropical Storm Danny that happened in 2021 stored on the website of the National Hurricane Center and Central Pacific Hurricane Center was used (https://www.nhc.noaa.gov/gis/archive_forecast.php?year=2021). The dataset consists of 20 files of different formats (Figure 35). The highlighted documents were used in the visualization. The first highlighted document shows a projected line of a storm, the second is a path and intensity of a storm, the third is a five-day-point forecast, and the last one is a watch and warning (a watch covers a broad region, a warning is issued by a local meteorologist for a smaller area) coastline.



al042021-001_5day_lin.dbf	02-Nov-22 20:18	DBF File	1 KB
al042021-001_5day_lin.prj	02-Nov-22 20:18	PRJ File	1 KB
al042021-001_5day_lin.shp	02-Nov-22 20:18	SHP File	1 KB
al042021-001_5day_lin.shp.xml	02-Nov-22 20:18	XML Document	7 KB
al042021-001_5day_lin.shx	02-Nov-22 20:18	SHX File	1 KB
al042021-001_5day_pgn.dbf	02-Nov-22 20:18	DBF File	1 KB
al042021-001_5day_pgn.prj	02-Nov-22 20:18	PRJ File	1 KB
al042021-001_5day_pgn.shp	02-Nov-22 20:18	SHP File	18 KB
al042021-001_5day_pgn.shp.xml	02-Nov-22 20:18	XML Document	7 KB
al042021-001_5day_pgn.shx	02-Nov-22 20:18	SHX File	1 KB
al042021-001_5day_pts.dbf	02-Nov-22 20:18	DBF File	6 KB
al042021-001_5day_pts.prj	02-Nov-22 20:18	PRJ File	1 KB
al042021-001_5day_pts.shp	02-Nov-22 20:18	SHP File	1 KB
al042021-001_5day_pts.shp.xml	02-Nov-22 20:18	XML Document	11 KB
al042021-001_5day_pts.shx	02-Nov-22 20:18	SHX File	1 KB
al042021-001_ww_wwlin.dbf	02-Nov-22 20:18	DBF File	1 KB
al042021-001_ww_wwlin.prj	02-Nov-22 20:18	PRJ File	1 KB
al042021-001_ww_wwlin.shp	02-Nov-22 20:18	SHP File	1 KB
al042021-001_ww_wwlin.shp.xml	02-Nov-22 20:18	XML Document	7 KB
al042021-001_ww_wwlin.shx	02-Nov-22 20:18	SHX File	1 KB

Figure 35. The documents that were downloaded from the open database and used to build a visualization of a storm using GeoPandas.

As in the previous example, first step is to install packages into Google Colaboratory.

```
!pip install matplotlib
!pip install cartopy
!pip install geopandas

#the binary package should not be used in cartopy => when running visu
alization
#using default cartopy's settings, there is an error
#below actions help to solve the issue
!pip uninstall shapely
!pip install shapely --no-binary shapely
```

Next is importing the libraries.

```
import geopandas
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import shapely

#magic command that makes the plots generated by matplotlib
#show into the IPython shell and not in a separate output window:
%matplotlib inline
```

Then is reading the files.

```
gdf = geopandas dataframe
track_line_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_lin.shp')
cone_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pgn.shp')
points_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pts.shp')
ww_gdf = geopandas.read_file('/content/sample2/al042021-
001A_ww_wwlin.shp')
```

The next step is to set the form of the map using Cartopy classes which are explained as the comments in the following code (check Appendix 2 for the output).

```
#cartopy classes:
#Lambert Conformal for conic projection
#Plate Carree for equirectangular projection
map_crs = ccrs.LambertConformal(central_latitude=35, central_longitude
=-100,
                                standard_parallels=(30,60))
data_crs = ccrs.PlateCarree()
```

Then, creating a figure and setting the boundaries of it.

```
#setting the size of the figure
fig = plt.figure(figsize=(12,10))

#creating a figure and multiple axes
ax = plt.subplot(1, 1, 1, projection=map_crs)

#first two are longitude, last two are latitude
#to adjust the map's extents
ax.set_extent([-90, -75, 20, 40])
```

To add the shapes of coastlines, lakes, states, ocean and land, the Cartopy's class *feature* is used and specific constants inside of it (check Appendix 2 for the output).

```
#adding shapes:
ax.add_feature(cfeature.COASTLINE.with_scale('50m'))
ax.add_feature(cfeature.LAKES.with_scale('50m'))
ax.add_feature(cfeature.STATES.with_scale('50m'))
ax.add_feature(cfeature.OCEAN.with_scale('50m'))
ax.add_feature(cfeature.LAND.with_scale('50m'))
```

To add geometry of a cone for showing the storm's path and shape of a track line of the storm,

Cartopy's *geometries* class was used.

```
#using the attached documents to add shapes
ax.add_geometries(cone_gdf['geometry'], crs=data_crs,
                  facecolor='white',
                  edgecolor='black', linewidth=0.25, alpha=0.4)

ax.add_geometries(track_line_gdf['geometry'], crs=data_crs,
                  facecolor='none',
                  edgecolor='black', linewidth=4)
```

Adding points, which will show the wind speed, to the line that shows a track of the storm is performed using the *scatter* class. The points of the wind are colored so that the harder the wind, the warmer the color (check Appendix 2 for the output).

```
#longtitude is X coordinate, lattitude is Y coordinate
#c = points_gdf: color series that shows the intensity of the wind
sc= ax.scatter(points_gdf['LON'], points_gdf['LAT'],
                  transform=data_crs,
                  c=points_gdf['MAXWIND'])

#color bar of the wind speed
plt.colorbar(sc)
```

To complete the map, it is necessary to show whether a watch- or warning-type storm it was. To add a visualization of a watch and warning, it is needed to create a dictionary with assigned values to keys, where the key is a type of watch or warning, and the value is a corresponding color. After the dictionary is created, for loop is used to go through the values, and when the corresponding value was chosen, the colored line will appear along the coastline (figure 35 is the output).

```
#create dictionary and assign color(value) to each element(key):
#TWA = tropical storm watch
#HWA = hurricane watch
#TWR = tropical storm warning
#HWR = hurricane warning
ww_colors = {'TWA': 'gold',
             'HWA': 'pink',
             'TWR': 'tab:blue',
             'HWR': 'tab:red'}

#TCWW = watch or warning of tropical storm or hurricane
for ww_type in ww_colors.keys():
  ww_subset = ww_gdf[ww_gdf['TCWW']==ww_type]
  ax.add_geometries(ww_subset['geometry'], facecolor='none',
                    edgecolor=ww_colors[ww_type], crs=data_crs,
                    linewidth=10)
```

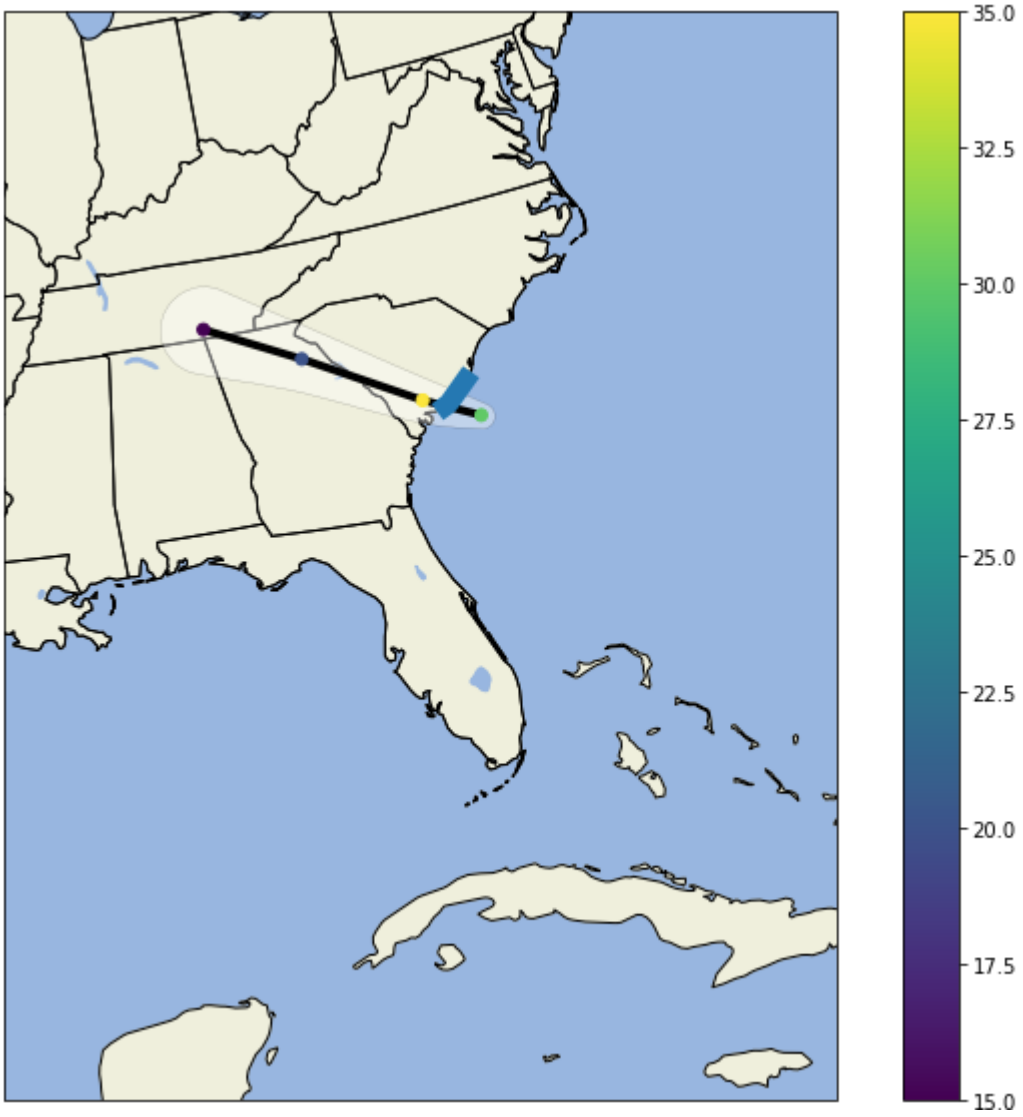


Figure 36. The result of the GeoPandas workshop.

As shown in the screenshot above, the type of warning that was associated with Tropical Storm Danny was a tropical storm warning (the bold blue line along the coast). In the final output, there is a color bar representing the wind speed, the direction of the storm with the wind speed points, the type of warning, and the cone – how wide the wind spread.

3.4 Carrying out the study summary

This chapter has been considered from three sides, each of which complements the previous one. At first, attention was paid to Python libraries in general, in what areas they are used, how popular they are in research, and what benefits they bring to developers and researchers. In subchapter 3.1, libraries such as:

- NumPy, which is one of the most popular libraries for scientific computing and one of the most striking examples of its use is the first picture of a black hole obtained in 2019
- Pandas, a Python library for processing and parsing structured data
- Intake, a lightweight set of tools for loading and sharing data in data science projects
- Matplotlib, a library for visualizing data in 2D format
- Seaborn, a Matplotlib-based data visualization library that allows to create interactive plots
- Plotly, a graphing library for interactive data visualization.

All these libraries share their open source nature, which allows anyone who wants to help in their development to contribute.

Next, the libraries that are used to visualize meteorological data were considered. Among them are MetPy, Alarconpy, Iris, CliMetLab, and GeoPandas. It is more correct to call them packages because in order to get the most out of all their capabilities, it is needed to connect to some other libraries. Accordingly, a set of these libraries is a package. This is necessary because the process of data visualization is multi-stage and begins with the search for a dataset, to which it is needed to connect, read, extract, make the necessary changes, and then create visualizations.

In the third part of this chapter, two examples of the hurricane and tropical storm data visualization using CliMetLab and GeoPandas were presented. The first Python package made it possible to create a graph depicting the direction of Hurricane Walaka in just a couple of steps, thanks to the built-in database. However, this simplicity comes with some limitations in terms of the number of styles that can be applied to a map. GeoPandas provides the ability to apply different styles, ranging from colors to geometric elements. The GeoPandas example was more complex than the CliMetLab's, and the final result has more details. The visualization was built on the basis of the same database that was used in the CliMetLab example, only in this case, it was necessary to download a set of documents and not connect to the database directly.

4 Discussion

The purpose of this work was to study the Python libraries and how they help simplify the process of visualizing data in general and meteorological data in particular. The direction for the start was chosen to study the history and practices of visualization, and how Python is used in data analysis field. To do this, the theoretical part was divided into three subchapters, covering topics such as the Python programming language and its usage in data analysis, the history and practices of data visualization, as well as the history and practices of visualization in meteorology. These sections facilitated the creation of a basis for further diving into the Python libraries and a practical part with the results. Two subchapters on the history and practices of visualization helped to deepen an understanding of the need and usefulness of graphical representation of data, as well as what assisted this vector of development in this field. Among other everyday reasons, such as simply the desire for a visual representation of phenomena and the search for a simpler way to explain them, one of the most key factors in the leap in the development of meteorological data visualization was technological progress and the advent of the computer.

The practical part of this thesis was devoted to the study of Python libraries and the demonstration of results using one or more of the libraries described in the chapter. To begin with, libraries used to simplify the data visualization process and libraries for weather visualization directly were studied. The first step was necessary in order to form an idea of the functionality of the programming language and its libraries. The initial intention was to study data visualization libraries, but in the course of the work it turned out that it would be more correct to call them packages or libraries without specifying the area, since creating images and plots requires the use of several libraries for processing data, reading and transforming it, and for visualization as the last step of the process. The demonstrated sets of meteorological data visualization libraries (packages, in one word) differ from each other not only in some functions but also in the degree of development and use. The result of the work is a demonstrated mini-demo visualization of data related to a hurricane and a tropical storm, made using CliMetLab and GeoPandas.

Achieved objectives

All the objectives that were set at the beginning of the research were met:

1. collecting the theoretical basis on Python data analysis libraries and their usage in visualizing meteorological data:

- Python’s role in data analysis was studied. Due to its simplicity, wide community, libraries and extensions support, and most importantly, applicability to all the stages of the data analysis, this programming language is popular among data analysts. To collect information

related to this phenomenon, such books as “Python for Data Analysis” by W. McKinney and online resources such as Stackoverflow and Berkeley were studied. It was quite easy to find the materials related to Python in general but how it relates to Data Analysis was a bit challenging and required more time. There are lots of books on how to analyze data using Python which are manuals but not so many books or articles to read the history of data analysis in Python nor what is it all about in general.

- Python’s libraries for visualizing meteorological data were studied. This was mostly done by first finding the overview of the libraries with ratings, and then reading the documentation on separate libraries. There are articles from scientific journals on how certain libraries are used for certain meteorological visualization. However, not enough sources of information of this type were found with an overview of libraries for visualizing meteorological data and therefore it was also needed to read articles on Medium and Towards Data Science to form an idea. Based on that the list of packages was chosen to further review from the official documentation websites.

2. highlighting the features of meteorological data visualization:

- history and practices of data visualization in general and in meteorology was learned to get an understanding of the current visualization practices. This was presented in the second chapter of the thesis. Even though the topic is wide, it was decided to catch the key points from history that affected the current state of visualization practices. The main resources of information for this part were the books and articles of Michael Friendly and Howard Wainer.

- data formats and specifics of meteorological data visualizations were studied in the third chapter of the thesis. This refers to the meteorological data visualization packages and libraries explained in 3.2 based on the documentation available online. It was expected that there will be more information available on functionality in more detail and in a descriptive manner rather than code pieces examples. Therefore, the feature explanations were limited to the documentation-provided materials. Nevertheless, the main functions and characteristics were presented.

3. presenting a demo of visualized meteorological data using one or more libraries:

- two practical visual examples showing (1) the hurricane’s track and (2) the tropical storm’s track, intensity, and type of warning were presented in the third chapter of the thesis. As this thesis was aimed at beginners in Python programming and data visualization, the examples given were based on the official documentation (for CliMetLab) and on the

workshop available on YouTube (for GeoPandas). The author explained every step and included the comments in the source code. The author is also a beginner in Python, which is why the examples were not complex but rather introductory.

Research results

A study of Python libraries revealed a data visualization feature and changed the direction of the thesis slightly. Initially, it was planned to get an overview of the Python visualization libraries alone, but research showed that the visualization libraries always work in connection with other packages, libraries, or extensions in order to get reliable results. That is why not only visualization libraries were discussed in the first section of the second chapter but also data science related as a whole. The complementation of different libraries is also demonstrated in the practical part where the figure of the tropical storm's path was plotted using several libraries which in combination are called a package.

Challenges

The first challenge was to find comprehensive articles on Python libraries with explained examples and a wider overview. Most of the information is stored on the official documentation's websites and represents an instrument to get started. Often, such additional information as a history of a library or real-life examples was missing which led to gaps in some parts of the thesis.

The second challenge was to build visualizations using chosen Python libraries without having experience in it. The represented examples in this work were based on official documentation and on openly available workshops from professionals.

Timeline

The thesis work was mostly performed between September and November 2022.

References

About GeoPandas. GeoPandas. URL: https://geopandas.org/en/stable/about.html Accessed: 03 November 2022.

Artima 2003. The Making of Python. A Conversation with Guido van Rossum, Part I. URL: artima - The Making of Python Accessed: 20 September 2022.

Berkeley Extension 2022. 11 Most In-Demand Programming Languages in 2022. URL: https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages/ Accessed: 18 September 2022.

Butwall, M., Ranka, P. & Shah, S. 2019. Python in Field of Data Science: A Review. International Journal of Computer Applications (0975-8887), Volume 178 – No. 49, September 2019. URL: https://www.ijcaonline.org/archives/volume178/number49/butwall-2019-ijca-919404.pdf Accessed: 15 September 2022.

CF Metadata 2022. CF Metadata Conventions. URL: http://cfconventions.org/ Accessed: 30 October 2022.

Chen, C., Härdle, W. & Unwin, A. Handbook of Data Visualization. 1st ed. Springer. Berlin.

Codecademy 2022. What Is Python Used For? URL: https://www.codecademy.com/resources/blog/what-is-python-used-for/ Accessed: 20 September 2022.

Coding club. Analysing earth science and climate data with Iris. Manipulate multi-dimensional climate data from common file formats in python. Github. URL: https://ourcodingclub.github.io/tutorials/iris-python-data-vis/ Accessed: 30 October 2022.

Event Horizon Telescope 2022. Astronomers Reveal First Image of the Black Hole at the Heart of Our Galaxy. URL: https://eventhorizontelescope.org/blog/astronomers-reveal-first-image-black-hole-heart-our-galaxy Accessed: 21 October 2022.

Examples 2022. CliMetLab. URL: https://climetlab.readthedocs.io/en/latest/examples.html Accessed: 30 October 2022.

Examples. GeoPandas. URL: https://geopandas.org/en/stable/gallery/index.html Accessed: 03 November 2022.

Few, S. 2007. Data Visualization – Past, Present, and Future. Cognos Innovation Center for Performance Management. URL: https://www.perceptualedge.com/articles/Whitepapers/Data_Visualization.pdf Accessed: 08 October 2022.

Fizell, Z. 2022. Easiest Way to Plot on a World Map with Pandas and GeoPandas. Towards Data Science. URL: https://towardsdatascience.com/easiest-way-to-plot-on-a-world-map-with-pandas-and-geopandas-325f6024949f Accessed: 03 November 2022.

Forrest, J. 2019. How W.E.B. Du Bois used data visualization to confront prejudice in the early 20th century. Tableau. URL: https://www.tableau.com/about/blog/2019/2/how-web-du-bois-used-data-visualization-confront-prejudice-early-20th-century Accessed: 08 October 2022.

Friendly, M. & Wainer, H. 2021. A History of Data Visualization & Graphic Communication. Harvard University Press.

Friendly, M. 2007. A Brief History of Data Visualization. Handbook of Data Visualization: C. Chen, W. Härdle, A. Unwin (Editors). Springer.

Frisinger, H. H. 1983. The History of Meteorology: to 1800. 2nd ed. Historical monograph series. American Meteorological Society.

Gilbert Tanner 2018. Introduction to Data Visualization in Python. URL: Introduction to Data Visualization in Python (gilberttanner.com) Accessed: 17 September 2022.

Hakštok, I. & Mihajlović, Ž. 2014. Visualization of Meteorological Data. 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). URL: https://ieeexplore.ieee.org/abstract/document/6859600 Accessed: 23 October 2022.

Hillery, A. 2020 The Evolution of Data Visualization. URL: https://chartio.com/blog/the-evolution-of-data-visualization/ Accessed: 08 October 2022.

Hillery, A. 2021. History of Data Visualization: Pivotal Moments Analysts Can Learn From. Panoply blog a SQream company. URL: https://blog.panoply.io/history-of-data-visualization Accessed: 08 October 2022.

Intake 2018. URL: https://intake.readthedocs.io/en/latest/ Accessed: 21 October 2022.

Iris 2022. SciTools – Iris. URL: https://scitools-iris.readthedocs.io/en/latest/index.html Accessed: 30 October 2022.

IT Black 2017. История языка Python. URL: https://it-black.ru/istoriya-yazyka-python/ Accessed: 20 September 2022.

Keeling, S. J. 2010. Visualization of the weather – past and present. Meteorological Applications. Wiley InterScience. URL: https://rmets.onlinelibrary.wiley.com/doi/epdf/10.1002/met.208 Accessed: 09 October 2022.

Kumaraswamy, A. 2018. Top 7 libraries for geospatial analysis. Packt Hub. URL: https://hub.packtpub.com/libraries-for-geospatial-analysis/ Accessed: 03 November 2022.

May, R. & Leeman, J. 28 November 2017. UCAR/Unidata. MetPy: Community-driven Meteorological Analysis Tools in Python
UCAR/Unidata 2020. Introduction to MetPy. Unidata Python Training. URL: https://unidata.github.io/python-training/workshop/Metpy_Introduction/introduction-to-metpy/ Accessed: 26 October 2022.

May, R., Goebbert, K., Thielen, J., Leeman, J., Camron, M., Bruick, Z., Bruning, C., Manser, R., Arms, S. & Marsh, P. 2022. MetPy: A meteorological Python library for data analysis and visualization, Bulletin of the American Meteorological Society. URL: https://journals.ametsoc.org/view/journals/bams/aop/BAMS-D-21-0125.1/BAMS-D-21-0125.1.xml Accessed: 25 October 2022.

McKinney, W. 2013. Python for Data Analysis. 1st ed. O'Reilly. Sebastopol.

Medrano, M. & Urton, G. 2018. The khipu code: the knotty mystery of the Inkas' 3D records. Aeon. URL: https://aeon.co/ideas/the-khipu-code-the-knotty-mystery-of-the-inkas-3d-records Accessed: 08 October 2022.

Mueller, J.P. & Massaron, L. 2019. Python for Data Science. 2nd ed. For dummies. A Wiley Brand.

Nebeker, F. 1995. Calculating the Weather: Meteorology in the 20th Century. Academic Press, Inc.

NHC GIS Archive - Tropical Cyclone Advisory Forecast. National Hurricane Center and Central Pacific Hurricane Center. URL: https://www.nhc.noaa.gov/gis/archive_forecast.php?year=2021 Accessed: 02 November 2022.

NumPy 2022. Case study: first image of a black hole. URL: https://numpy.org/case-studies/blackhole-image/ Accessed: 21 October 2022.

NumPy 2022. URL: https://numpy.org/ Accessed 21 October 2022.

Ouyang, J. F. 2020 Data Visualization using COVID-19 Statistics. Github.io. URL: https://jfouyang.github.io/covid19dataviz/dataviz.html Accessed: 08 October 2022.

Overview 2022. CliMetLab. URL: https://climetlab.readthedocs.io/en/latest/overview.html Accessed: 30 October 2022.

Pandas 2022. About pandas. URL: https://pandas.pydata.org/about/ Accessed: 21 October 2022.

Pérez-Alarcón, A. & Fernández-Alvarez, J. C. 2021. Alarconpy: a Python Package for Meteorologists. Technical Report. URL: https://www.researchgate.net/profile/Albenis-Perez-Alarcon/publication/350663375_Alarconpy_a_Python_Package_for_Meteorologists/links/606c72d4299bf13f5d5f7966/Alarconpy-a-Python-Package-for-Meteorologists.pdf Accessed: 30 October 2022.

Playfair, W. An Inquiry into the Permanent Causes of the Decline and Fall of Powerful and Wealthy Nations. 2005. Project Gutenberg EBook. URL: https://www.gutenberg.org/files/16575/16575-h/16575-h.htm Accessed: 08 October 2022.

Plotly 2022. Getting Started with Plotly in Python. URL: https://plotly.com/python/getting-started/ Accessed: 21 October 2022.

Plotly 2022. Plotly Open Source Graphing Library for Python. URL: https://plotly.com/python/ Accessed: 21 October 2022.

PyProg Semyon Lukashevskiy 2022. Seaborn введение. URL: https://pyprog.pro/sns/sns_1_introduction.html Accessed: 21 October 2022.

Python 2004. Python Enhancement Proposals. URL: https://peps.python.org/pep-0020/ Accessed: 20 September 2022.

Sands, R 2012. Open source by the numbers. URL: https://www.slideshare.net/blackducksoftware/open-source-by-the-numbers/ Accessed: 07 October 2022.

Schiavone, J. A. & Papathomas T. V. 1990 Visualizaing Meteorological Data. Bulletin of the American Meteorological Society. URL: https://journals.ametsoc.org/view/journals/bams/71/7/1520-0477_1990_071_1012_vmd_2_0_co_2.xml?tab_body=pdf Accessed 09 October 2022.

Sheppard, K. 2021. Introduction to Python for Econometrics, Statistics and Data Analysis. 5th ed.

University of Oxford. URL:
https://www.kevinsheppard.com/files/teaching/python/notes/python_introduction_2021.pdf
Accessed: 22 September 2022.

Stackoverflow 2021. Developer Survey. URL:
https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-language-want
Accessed: 18 September 2022.

Techtarget 2021. Python. URL: https://www.techtarget.com/whatis/definition/Python/ Accessed:
20 September 2022.

Unidata 2 Septmeber 2019. MetPy Mondays #95 - Hurricane Dorian Mapping. YouTube. URL:
https://www.youtube.com/watch?v=L3WV7dK1Ld0&list=PLQut5OXpV-
0ir4IdlISt1iEZKTwFBa7kO&index=86 Accessed: 2 November 2022.

Wainer, H & Spence, I. 2005. The Commercial and Political Atlas and Statistical Breviary.
Cambridge University Press. Golden Cup.

Appendices

Appendix 1. CliMetLab source code and output

1. Input:

```
!pip install --quiet climetlab matplotlib
import climetlab as cm

pacific = cml.load_dataset("hurricane-database", bassin="pacific")

df = pacific.to_pandas()
df.bassin.unique()

df[(df.category == 5)]
```

Output:

	bassin	number	year	name	time	type	status	lat	lon	knots	category	pressure
1527	CP	2	1959	patsy	1959-09-06 18:00:00		HU	22.2	-178.5	150.0	5	NaN
5241	EP	1	1973	ava	1973-06-07 00:00:00		HU	12.1	-108.8	140.0	5	NaN
5242	EP	1	1973	ava	1973-06-07 06:00:00		HU	12.2	-110.3	140.0	5	NaN
5243	EP	1	1973	ava	1973-06-07 12:00:00		HU	12.4	-111.8	140.0	5	NaN
5244	EP	1	1973	ava	1973-06-07 18:00:00		HU	12.6	-113.3	140.0	5	928.0
...	...	...	...	...	...	...	...	...	...	...	...	...
28160	EP	14	2018	lane	2018-08-22 00:00:00		HU	14.3	-153.6	140.0	5	929.0
28161	EP	14	2018	lane	2018-08-22 06:00:00		HU	14.5	-154.3	140.0	5	926.0
28424	CP	1	2018	walaka	2018-10-02 00:00:00		HU	12.9	-169.6	140.0	5	921.0
28425	CP	1	2018	walaka	2018-10-02 06:00:00		HU	13.5	-169.9	140.0	5	921.0
28545	EP	24	2018	willa	2018-10-22 06:00:00		HU	17.9	-107.1	140.0	5	925.0

81 rows × 12 columns

Figure 1. The output of the executed code shown above.

2. Input:

```
!pip install --quiet climetlab matplotlib
import climetlab as cml

pacific = cml.load_dataset("hurricane-database", bassin="pacific")

df = pacific.to_pandas()
df.bassin.unique()

walaka = df[(df.name == "walaka") & (df.year == 2018)]
walaka
```

Output:

	bassin	number	year	name	time	type	status	lat	lon	knots	category	pressure
28401	CP	1	2018	walaka	2018-09-26 06:00:00		DB	10.7	-140.1	25.0	1	1008.0
28402	CP	1	2018	walaka	2018-09-26 12:00:00		DB	11.1	-140.7	25.0	1	1008.0
28403	CP	1	2018	walaka	2018-09-26 18:00:00		DB	11.7	-141.5	25.0	1	1008.0
28404	CP	1	2018	walaka	2018-09-27 00:00:00		DB	12.2	-142.6	25.0	1	1008.0
28405	CP	1	2018	walaka	2018-09-27 06:00:00		DB	12.2	-144.1	25.0	1	1008.0
28406	CP	1	2018	walaka	2018-09-27 12:00:00		LO	12.2	-145.7	25.0	1	1008.0
28407	CP	1	2018	walaka	2018-09-27 18:00:00		LO	12.1	-147.3	25.0	1	1008.0
28408	CP	1	2018	walaka	2018-09-28 00:00:00		LO	12.1	-148.8	25.0	1	1007.0
28409	CP	1	2018	walaka	2018-09-28 06:00:00		LO	12.0	-150.2	25.0	1	1007.0
28410	CP	1	2018	walaka	2018-09-28 12:00:00		LO	12.0	-151.5	25.0	1	1007.0
28411	CP	1	2018	walaka	2018-09-28 18:00:00		LO	11.9	-152.8	25.0	1	1007.0
28412	CP	1	2018	walaka	2018-09-29 00:00:00		LO	11.8	-154.2	25.0	1	1007.0
28413	CP	1	2018	walaka	2018-09-29 06:00:00		LO	11.8	-155.6	25.0	1	1007.0
28414	CP	1	2018	walaka	2018-09-29 12:00:00		TD	11.7	-157.1	30.0	1	1006.0

Figure 2. The output of the executed code shown above.

Appendix 2. GeoPandas source code and output

1. Input:

```
!pip install matplotlib
!pip install cartopy
!pip install geopandas

#the binary package should not be used in cartopy => when running visualizat
ion
#using default cartopy's settings, there is an error
#below actions help to solve the issue
!pip uninstall shapely
!pip install shapely --no-binary shapely

import geopandas
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import shapely

#magic command that makes the plots generated by matplotlib
#show into the IPython shell and not in a separate output window:
%matplotlib inline

#gdf = geopandas dataframe
track_line_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_lin.shp')
cone_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pgn.shp')
points_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pts.shp')
ww_gdf = geopandas.read_file('/content/sample2/al042021-001A_ww_wwlin.shp')

#cartopy classes:
#Lambert Conformal for conic projection
#Plate Carree for equirectangular projection
map_crs = ccrs.LambertConformal(central_latitude=35, central_longitude=-100,
                                standard_parallels=(30,60))
data_crs = ccrs.PlateCarree()

#setting the size of the figure
fig = plt.figure(figsize=(12,10))

#creating a figure and multiple axes
ax = plt.subplot(1, 1, 1, projection=map_crs)

#first two are longitude, last two are latitude
#to adjust the map's extents
```

```
ax.set_extent([-90, -75, 20, 40])
```

Output:

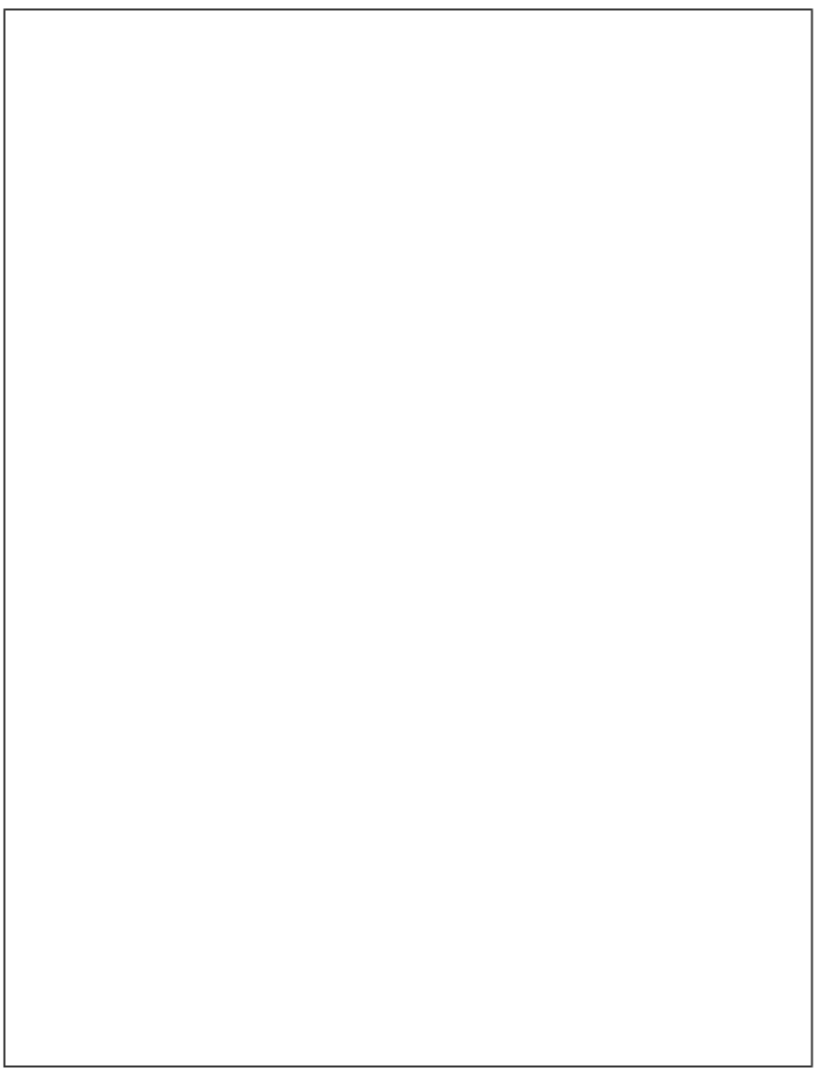


Figure 3. The output of the executed code shown above.

2. Input:

```
!pip install matplotlib
!pip install cartopy
!pip install geopandas

#the binary package should not be used in cartopy => when running visualizat
ion
#using default cartopy's settings, there is an error
#below actions help to solve the issue
!pip uninstall shapely
!pip install shapely --no-binary shapely
```

```
import geopandas
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import shapely

#magic command that makes the plots generated by matplotlib
#show into the IPython shell and not in a separate output window:
%matplotlib inline


gdf = geopandas dataframe
track_line_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_lin.shp')
cone_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pgn.shp')
points_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pts.shp')
ww_gdf = geopandas.read_file('/content/sample2/al042021-001A_ww_wwlin.shp')

#cartopy classes:
#Lambert Conformal for conic projection
#Plate Carree for equirectangular projection
map_crs = ccrs.LambertConformal(central_latitude=35, central_longitude=-100,
                                standard_parallels=(30,60))
data_crs = ccrs.PlateCarree()

#setting the size of the figure
fig = plt.figure(figsize=(12,10))

#creating a figure and multiple axes
ax = plt.subplot(1, 1, 1, projection=map_crs)

#first two are longitude, last two are latitude
#to adjust the map's extents
ax.set_extent([-90, -75, 20, 40])
#adding shapes:
ax.add_feature(cfeature.COASTLINE.with_scale('50m'))
ax.add_feature(cfeature.LAKES.with_scale('50m'))
ax.add_feature(cfeature.STATES.with_scale('50m'))
ax.add_feature(cfeature.OCEAN.with_scale('50m'))
ax.add_feature(cfeature.LAND.with_scale('50m'))
```

Output:

```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7fa69f68f7d0>
```

Figure 4. The output of the executed code shown above.

3. Input:

```
!pip install matplotlib


!pip install cartopy
!pip install geopandas

#the binary package should not be used in cartopy => when running visualizat
ion
#using default cartopy's settings, there is an error
#below actions help to solve the issue
!pip uninstall shapely
!pip install shapely --no-binary shapely
```

```
import geopandas
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import shapely

#magic command that makes the plots generated by matplotlib
#show into the IPython shell and not in a separate output window:
%matplotlib inline


gdf = geopandas dataframe
track_line_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_lin.shp')
cone_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pgn.shp')
points_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pts.shp')
ww_gdf = geopandas.read_file('/content/sample2/al042021-001A_ww_wwlin.shp')

#cartopy classes:
#Lambert Conformal for conic projection
#Plate Carree for equirectangular projection
map_crs = ccrs.LambertConformal(central_latitude=35, central_longitude=-100,
                                standard_parallels=(30,60))
data_crs = ccrs.PlateCarree()

#setting the size of the figure
fig = plt.figure(figsize=(12,10))

#creating a figure and multiple axes
ax = plt.subplot(1, 1, 1, projection=map_crs)

#first two are longitude, last two are latitude
#to adjust the map's extents
ax.set_extent([-90, -75, 20, 40])
#adding shapes:
ax.add_feature(cfeature.COASTLINE.with_scale('50m'))
ax.add_feature(cfeature.LAKES.with_scale('50m'))
ax.add_feature(cfeature.STATES.with_scale('50m'))
ax.add_feature(cfeature.OCEAN.with_scale('50m'))
ax.add_feature(cfeature.LAND.with_scale('50m'))

#using the attached documents to add shapes
ax.add_geometries(cone_gdf['geometry'], crs=data_crs, facecolor='white',
                  edgecolor='black', linewidth=0.25, alpha=0.4)
```

```
ax.add_geometries(track_line_gdf['geometry'], crs=data_crs, facecolor='none'
,
                  edgecolor='black', linewidth=4)
```

Output:

```
<cartopy.mpl.feature_artist.FeatureArtist at 0x7fa69ed13cd0>
```

Figure 5. The output of the executed code shown above.

4. Input:

```
!pip install matplotlib
!pip install cartopy
!pip install geopandas

#the binary package should not be used in cartopy => when running visualizat
ion
#using default cartopy's settings, there is an error
```

```
#below actions help to solve the issue
!pip uninstall shapely
!pip install shapely --no-binary shapely


import geopandas
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import shapely

#magic command that makes the plots generated by matplotlib
#show into the IPython shell and not in a separate output window:
%matplotlib inline


gdf = geopandas dataframe
track_line_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_lin.shp')
cone_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pgn.shp')
points_gdf = geopandas.read_file('/content/sample2/al042021-
001A_5day_pts.shp')
ww_gdf = geopandas.read_file('/content/sample2/al042021-001A_ww_wwlin.shp')

#cartopy classes:
#Lambert Conformal for conic projection
#Plate Carree for equirectangular projection
map_crs = ccrs.LambertConformal(central_latitude=35, central_longitude=-100,
                                standard_parallels=(30,60))
data_crs = ccrs.PlateCarree()

#setting the size of the figure
fig = plt.figure(figsize=(12,10))

#creating a figure and multiple axes
ax = plt.subplot(1, 1, 1, projection=map_crs)

#first two are longitude, last two are latitude
#to adjust the map's extents
ax.set_extent([-90, -75, 20, 40])
#adding shapes:
ax.add_feature(cfeature.COASTLINE.with_scale('50m'))
ax.add_feature(cfeature.LAKES.with_scale('50m'))
ax.add_feature(cfeature.STATES.with_scale('50m'))
ax.add_feature(cfeature.OCEAN.with_scale('50m'))
ax.add_feature(cfeature.LAND.with_scale('50m'))

#using the attached documents to add shapes
```

```
ax.add_geometries(cone_gdf['geometry'], crs=data_crs, facecolor='white',
                  edgecolor='black', linewidth=0.25, alpha=0.4)

ax.add_geometries(track_line_gdf['geometry'], crs=data_crs, facecolor='none'
,
                  edgecolor='black', linewidth=4)
#longtitude is X coordinate, lattitude is Y coordinate
#c = points_gdf: color series that shows the intensity of the wind
sc= ax.scatter(points_gdf['LON'], points_gdf['LAT'], transform=data_crs,
           c=points_gdf['MAXWIND'])

#color bar of the wind speed
plt.colorbar(sc)
```

Output:

```
<matplotlib.colorbar.Colorbar at 0x7fa69ec97590>
```

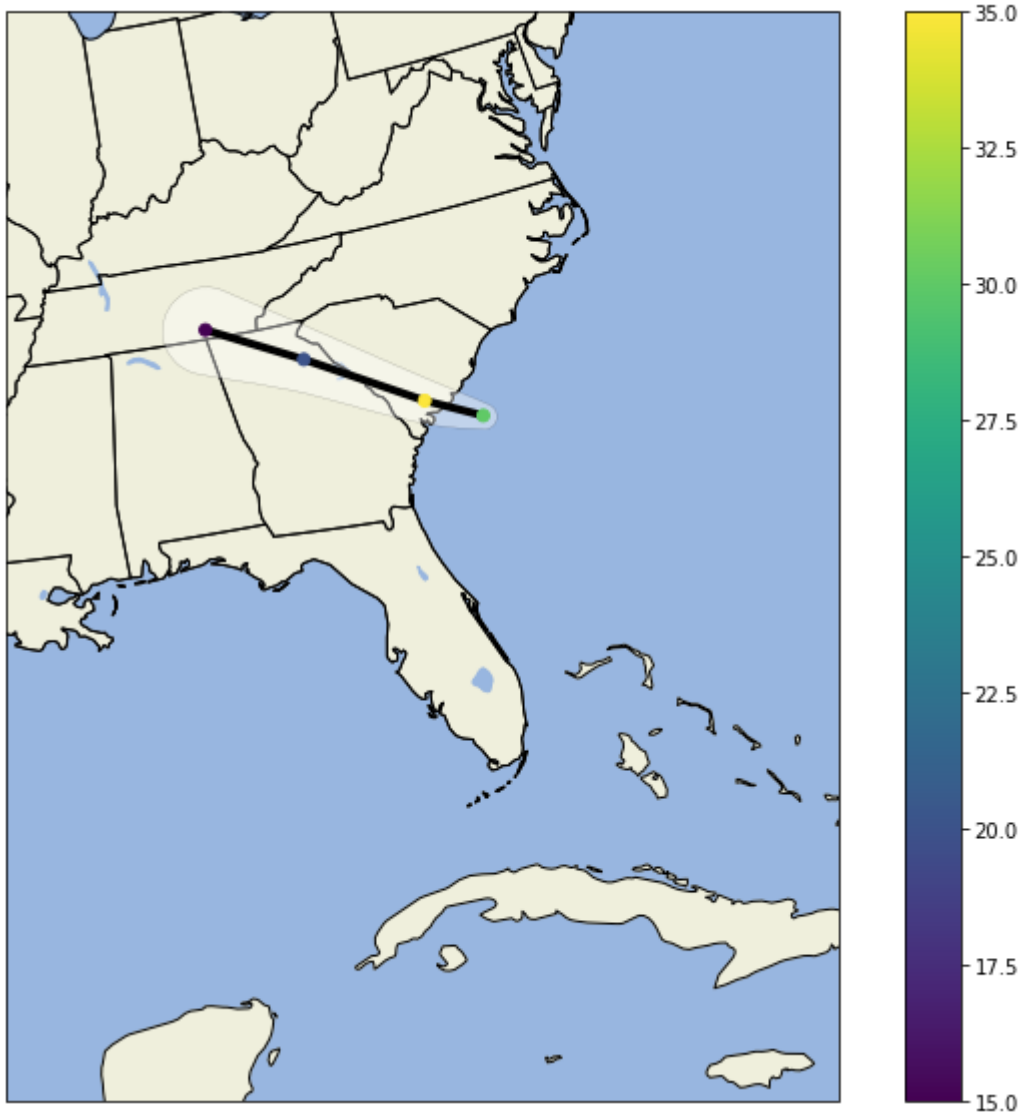


Figure 6. The output of the executed code shown above.