



GWALSH@IMADA.SDU.DK

DS808/DM878 Visualization

EXERCISE CLASS 1

GARETH WALSH, PHD FELLOW



Syddansk Universitet



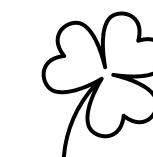


GWALSH@IMADA.SDU.DK



A bit about me

- Final-year **PhD Fellow** at the Centre for Visual Data Science
- Led the design of the frontend and visualization module for the EDIDP R&D Project **CUIIS**.
- **Authored publications** on topics such as:
 - Geospatial-Temporal Visualizations for Military Operations.
 - Military Diving Incident Reporting systems.
 - Diver Tracking and Health Monitoring tools.
 - NATO Symbology for Subsea Operations.
- **TA** for this course last year, and helped 5 groups receive publication at EuroVis 2024.
- Spent the last five months at **DG DEFIS** in Brussels, learning about EU Defence Industry Policy and working on EU defence innovation.
- And, of course, I'm from **Ireland**.





GWALSH@IMADA.SDU.DK



A Survey of Geospatial-Temporal Visualizations for Military Operations

G. Walsh¹*, N. S. Andersen¹, N. Stoiakov² and S. Janicek³
¹Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark
²Department of Computing, Bulgarian Defence Institute, Sofia, Bulgaria

Keywords: Military Operations, Command and Control, Situational Awareness, Geospatial-Temporal Visualization

Abstract: European defence funding has surpassed 20 billion euro per year for the first time, with a renewed interest in military operations and decision support systems. Such systems are critical for military command and control support systems. Overcoming the many challenges associated with the research and development of geospatial-temporal visualizations for military operations is a key area of interest. This paper reviews the domain, as there is ample scope for applied research. No other recent surveys examine the use and deployment of Information Visualization (IV) and Visual Analytics (VA) tools in the military domain. As such, the survey is timely and provides a much-needed synthesis of the field. The paper also highlights the potential of IV and VA for military decision-support systems, specifically focusing on geospatial-temporal visualization aspects. Considering the above, the paper concludes by discussing the future research directions for IV and VA in military operations. They may offer to military decision-support systems through the lens of the Military Operations Process model. The paper also highlights the potential of IV and VA for military decision-support systems. The paper's outcome and main contribution is thus the formulation of a design space and analysis of emerging trends, promising research topics to identify gaps, opportunities, and guidelines for future potential research.

* indicates author to whom correspondence should be addressed. ¹ denotes military decision-support systems can enhance military commanders' decision-making abilities and ability to act.

1. INTRODUCTION

The Russian invasion of Ukraine has increased the risk of a conflict between NATO member states and Russia. In 2022, the European Defence Agency has stated, "Europe's defence spending is projected to grow by 1.5% in 2022, fueling the increased need to develop corresponding decision support systems. The most necessary intervening variable to handle both conventional and unconventional future warfare conflicts" [20]. The European Defence Agency has stated that military forces have re-emphasized and re-prioritized communication and collaboration with allies and partners, with a desire to create common open-source, integrated, and interoperable systems for military operations which easily enable cross-platform interactions. Future development of such emerging military technologies will be driven by the needs of the general public as well as the general public. An example of such emerging technologies is the development of a network which began as a modest research initiative to link three early packet networks in an academic framework (Kahn et al., 1979).

[DOI: https://doi.org/10.4236/ojs.2022.1205471](https://doi.org/10.4236/ojs.2022.1205471)

[DOI: https://doi.org/10.4236/ojs.2022.1205472](https://doi.org/10.4236/ojs.2022.1205472)

[DOI: https://doi.org/10.4236/ojs.2022.1205473](https://doi.org/10.4236/ojs.2022.1205473)

[DOI: https://doi.org/10.4236/ojs.2022.1205474](https://doi.org/10.4236/ojs.2022.1205474)

[DOI: https://doi.org/10.4236/ojs.2022.1205475](https://doi.org/10.4236/ojs.2022.1205475)

[DOI: https://doi.org/10.4236/ojs.2022.1205476](https://doi.org/10.4236/ojs.2022.1205476)

[DOI: https://doi.org/10.4236/ojs.2022.1205477](https://doi.org/10.4236/ojs.2022.1205477)

[DOI: https://doi.org/10.4236/ojs.2022.1205478](https://doi.org/10.4236/ojs.2022.1205478)

[DOI: https://doi.org/10.4236/ojs.2022.1205479](https://doi.org/10.4236/ojs.2022.1205479)

[DOI: https://doi.org/10.4236/ojs.2022.1205480](https://doi.org/10.4236/ojs.2022.1205480)

[DOI: https://doi.org/10.4236/ojs.2022.1205481](https://doi.org/10.4236/ojs.2022.1205481)

[DOI: https://doi.org/10.4236/ojs.2022.1205482](https://doi.org/10.4236/ojs.2022.1205482)

[DOI: https://doi.org/10.4236/ojs.2022.1205483](https://doi.org/10.4236/ojs.2022.1205483)

[DOI: https://doi.org/10.4236/ojs.2022.1205484](https://doi.org/10.4236/ojs.2022.1205484)

[DOI: https://doi.org/10.4236/ojs.2022.1205485](https://doi.org/10.4236/ojs.2022.1205485)

[DOI: https://doi.org/10.4236/ojs.2022.1205486](https://doi.org/10.4236/ojs.2022.1205486)

[DOI: https://doi.org/10.4236/ojs.2022.1205487](https://doi.org/10.4236/ojs.2022.1205487)

[DOI: https://doi.org/10.4236/ojs.2022.1205488](https://doi.org/10.4236/ojs.2022.1205488)

[DOI: https://doi.org/10.4236/ojs.2022.1205489](https://doi.org/10.4236/ojs.2022.1205489)

[DOI: https://doi.org/10.4236/ojs.2022.1205490](https://doi.org/10.4236/ojs.2022.1205490)

[DOI: https://doi.org/10.4236/ojs.2022.1205491](https://doi.org/10.4236/ojs.2022.1205491)

[DOI: https://doi.org/10.4236/ojs.2022.1205492](https://doi.org/10.4236/ojs.2022.1205492)

[DOI: https://doi.org/10.4236/ojs.2022.1205493](https://doi.org/10.4236/ojs.2022.1205493)

[DOI: https://doi.org/10.4236/ojs.2022.1205494](https://doi.org/10.4236/ojs.2022.1205494)

[DOI: https://doi.org/10.4236/ojs.2022.1205495](https://doi.org/10.4236/ojs.2022.1205495)

[DOI: https://doi.org/10.4236/ojs.2022.1205496](https://doi.org/10.4236/ojs.2022.1205496)

[DOI: https://doi.org/10.4236/ojs.2022.1205497](https://doi.org/10.4236/ojs.2022.1205497)

[DOI: https://doi.org/10.4236/ojs.2022.1205498](https://doi.org/10.4236/ojs.2022.1205498)

[DOI: https://doi.org/10.4236/ojs.2022.1205499](https://doi.org/10.4236/ojs.2022.1205499)

[DOI: https://doi.org/10.4236/ojs.2022.1205500](https://doi.org/10.4236/ojs.2022.1205500)

[DOI: https://doi.org/10.4236/ojs.2022.1205501](https://doi.org/10.4236/ojs.2022.1205501)

[DOI: https://doi.org/10.4236/ojs.2022.1205502](https://doi.org/10.4236/ojs.2022.1205502)

[DOI: https://doi.org/10.4236/ojs.2022.1205503](https://doi.org/10.4236/ojs.2022.1205503)

[DOI: https://doi.org/10.4236/ojs.2022.1205504](https://doi.org/10.4236/ojs.2022.1205504)

[DOI: https://doi.org/10.4236/ojs.2022.1205505](https://doi.org/10.4236/ojs.2022.1205505)

[DOI: https://doi.org/10.4236/ojs.2022.1205506](https://doi.org/10.4236/ojs.2022.1205506)

[DOI: https://doi.org/10.4236/ojs.2022.1205507](https://doi.org/10.4236/ojs.2022.1205507)

[DOI: https://doi.org/10.4236/ojs.2022.1205508](https://doi.org/10.4236/ojs.2022.1205508)

[DOI: https://doi.org/10.4236/ojs.2022.1205509](https://doi.org/10.4236/ojs.2022.1205509)

[DOI: https://doi.org/10.4236/ojs.2022.1205510](https://doi.org/10.4236/ojs.2022.1205510)

[DOI: https://doi.org/10.4236/ojs.2022.1205511](https://doi.org/10.4236/ojs.2022.1205511)

[DOI: https://doi.org/10.4236/ojs.2022.1205512](https://doi.org/10.4236/ojs.2022.1205512)

[DOI: https://doi.org/10.4236/ojs.2022.1205513](https://doi.org/10.4236/ojs.2022.1205513)

[DOI: https://doi.org/10.4236/ojs.2022.1205514](https://doi.org/10.4236/ojs.2022.1205514)

[DOI: https://doi.org/10.4236/ojs.2022.1205515](https://doi.org/10.4236/ojs.2022.1205515)

[DOI: https://doi.org/10.4236/ojs.2022.1205516](https://doi.org/10.4236/ojs.2022.1205516)

[DOI: https://doi.org/10.4236/ojs.2022.1205517](https://doi.org/10.4236/ojs.2022.1205517)

[DOI: https://doi.org/10.4236/ojs.2022.1205518](https://doi.org/10.4236/ojs.2022.1205518)

[DOI: https://doi.org/10.4236/ojs.2022.1205519](https://doi.org/10.4236/ojs.2022.1205519)

[DOI: https://doi.org/10.4236/ojs.2022.1205520](https://doi.org/10.4236/ojs.2022.1205520)

[DOI: https://doi.org/10.4236/ojs.2022.1205521](https://doi.org/10.4236/ojs.2022.1205521)

[DOI: https://doi.org/10.4236/ojs.2022.1205522](https://doi.org/10.4236/ojs.2022.1205522)

[DOI: https://doi.org/10.4236/ojs.2022.1205523](https://doi.org/10.4236/ojs.2022.1205523)

[DOI: https://doi.org/10.4236/ojs.2022.1205524](https://doi.org/10.4236/ojs.2022.1205524)

[DOI: https://doi.org/10.4236/ojs.2022.1205525](https://doi.org/10.4236/ojs.2022.1205525)

[DOI: https://doi.org/10.4236/ojs.2022.1205526](https://doi.org/10.4236/ojs.2022.1205526)

[DOI: https://doi.org/10.4236/ojs.2022.1205527](https://doi.org/10.4236/ojs.2022.1205527)

[DOI: https://doi.org/10.4236/ojs.2022.1205528](https://doi.org/10.4236/ojs.2022.1205528)

[DOI: https://doi.org/10.4236/ojs.2022.1205529](https://doi.org/10.4236/ojs.2022.1205529)

[DOI: https://doi.org/10.4236/ojs.2022.1205530](https://doi.org/10.4236/ojs.2022.1205530)

[DOI: https://doi.org/10.4236/ojs.2022.1205531](https://doi.org/10.4236/ojs.2022.1205531)

[DOI: https://doi.org/10.4236/ojs.2022.1205532](https://doi.org/10.4236/ojs.2022.1205532)

[DOI: https://doi.org/10.4236/ojs.2022.1205533](https://doi.org/10.4236/ojs.2022.1205533)

[DOI: https://doi.org/10.4236/ojs.2022.1205534](https://doi.org/10.4236/ojs.2022.1205534)

[DOI: https://doi.org/10.4236/ojs.2022.1205535](https://doi.org/10.4236/ojs.2022.1205535)

[DOI: https://doi.org/10.4236/ojs.2022.1205536](https://doi.org/10.4236/ojs.2022.1205536)

[DOI: https://doi.org/10.4236/ojs.2022.1205537](https://doi.org/10.4236/ojs.2022.1205537)

[DOI: https://doi.org/10.4236/ojs.2022.1205538](https://doi.org/10.4236/ojs.2022.1205538)

[DOI: https://doi.org/10.4236/ojs.2022.1205539](https://doi.org/10.4236/ojs.2022.1205539)

[DOI: https://doi.org/10.4236/ojs.2022.1205540](https://doi.org/10.4236/ojs.2022.1205540)

[DOI: https://doi.org/10.4236/ojs.2022.1205541](https://doi.org/10.4236/ojs.2022.1205541)

[DOI: https://doi.org/10.4236/ojs.2022.1205542](https://doi.org/10.4236/ojs.2022.1205542)

[DOI: https://doi.org/10.4236/ojs.2022.1205543](https://doi.org/10.4236/ojs.2022.1205543)

[DOI: https://doi.org/10.4236/ojs.2022.1205544](https://doi.org/10.4236/ojs.2022.1205544)

[DOI: https://doi.org/10.4236/ojs.2022.1205545](https://doi.org/10.4236/ojs.2022.1205545)

[DOI: https://doi.org/10.4236/ojs.2022.1205546](https://doi.org/10.4236/ojs.2022.1205546)

[DOI: https://doi.org/10.4236/ojs.2022.1205547](https://doi.org/10.4236/ojs.2022.1205547)

[DOI: https://doi.org/10.4236/ojs.2022.1205548](https://doi.org/10.4236/ojs.2022.1205548)

[DOI: https://doi.org/10.4236/ojs.2022.1205549](https://doi.org/10.4236/ojs.2022.1205549)

[DOI: https://doi.org/10.4236/ojs.2022.1205550](https://doi.org/10.4236/ojs.2022.1205550)

[DOI: https://doi.org/10.4236/ojs.2022.1205551](https://doi.org/10.4236/ojs.2022.1205551)

[DOI: https://doi.org/10.4236/ojs.2022.1205552](https://doi.org/10.4236/ojs.2022.1205552)

[DOI: https://doi.org/10.4236/ojs.2022.1205553](https://doi.org/10.4236/ojs.2022.1205553)

[DOI: https://doi.org/10.4236/ojs.2022.1205554](https://doi.org/10.4236/ojs.2022.1205554)

[DOI: https://doi.org/10.4236/ojs.2022.1205555](https://doi.org/10.4236/ojs.2022.1205555)

[DOI: https://doi.org/10.4236/ojs.2022.1205556](https://doi.org/10.4236/ojs.2022.1205556)

[DOI: https://doi.org/10.4236/ojs.2022.1205557](https://doi.org/10.4236/ojs.2022.1205557)

[DOI: https://doi.org/10.4236/ojs.2022.1205558](https://doi.org/10.4236/ojs.2022.1205558)

[DOI: https://doi.org/10.4236/ojs.2022.1205559](https://doi.org/10.4236/ojs.2022.1205559)

[DOI: https://doi.org/10.4236/ojs.2022.1205560](https://doi.org/10.4236/ojs.2022.1205560)

[DOI: https://doi.org/10.4236/ojs.2022.1205561](https://doi.org/10.4236/ojs.2022.1205561)

[DOI: https://doi.org/10.4236/ojs.2022.1205562](https://doi.org/10.4236/ojs.2022.1205562)

[DOI: https://doi.org/10.4236/ojs.2022.1205563](https://doi.org/10.4236/ojs.2022.1205563)

[DOI: https://doi.org/10.4236/ojs.2022.1205564](https://doi.org/10.4236/ojs.2022.1205564)

[DOI: https://doi.org/10.4236/ojs.2022.1205565](https://doi.org/10.4236/ojs.2022.1205565)

[DOI: https://doi.org/10.4236/ojs.2022.1205566](https://doi.org/10.4236/ojs.2022.1205566)

[DOI: https://doi.org/10.4236/ojs.2022.1205567](https://doi.org/10.4236/ojs.2022.1205567)

[DOI: https://doi.org/10.4236/ojs.2022.1205568](https://doi.org/10.4236/ojs.2022.1205568)

[DOI: https://doi.org/10.4236/ojs.2022.1205569](https://doi.org/10.4236/ojs.2022.1205569)

[DOI: https://doi.org/10.4236/ojs.2022.1205570](https://doi.org/10.4236/ojs.2022.1205570)

[DOI: https://doi.org/10.4236/ojs.2022.1205571](https://doi.org/10.4236/ojs.2022.1205571)

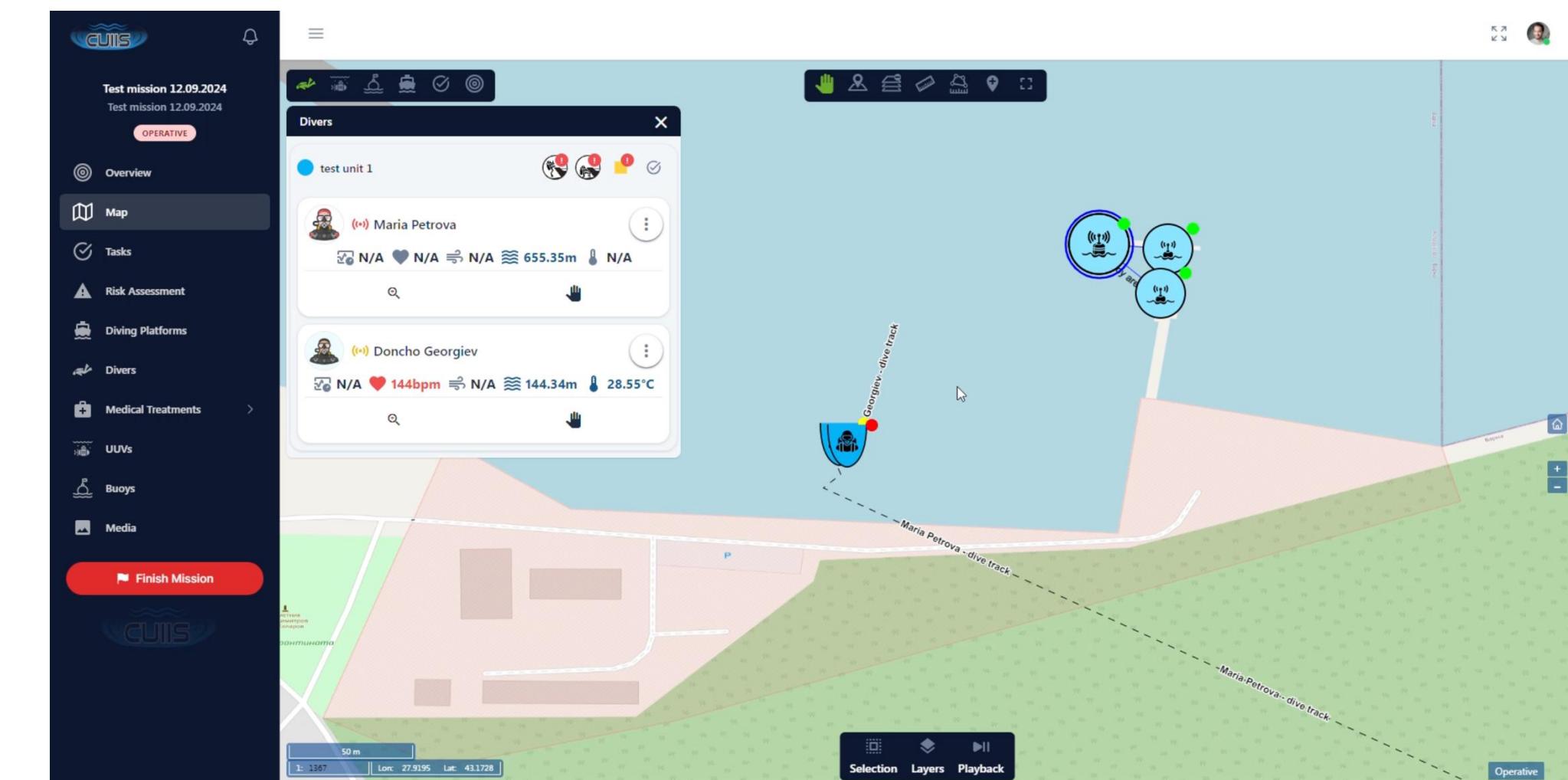
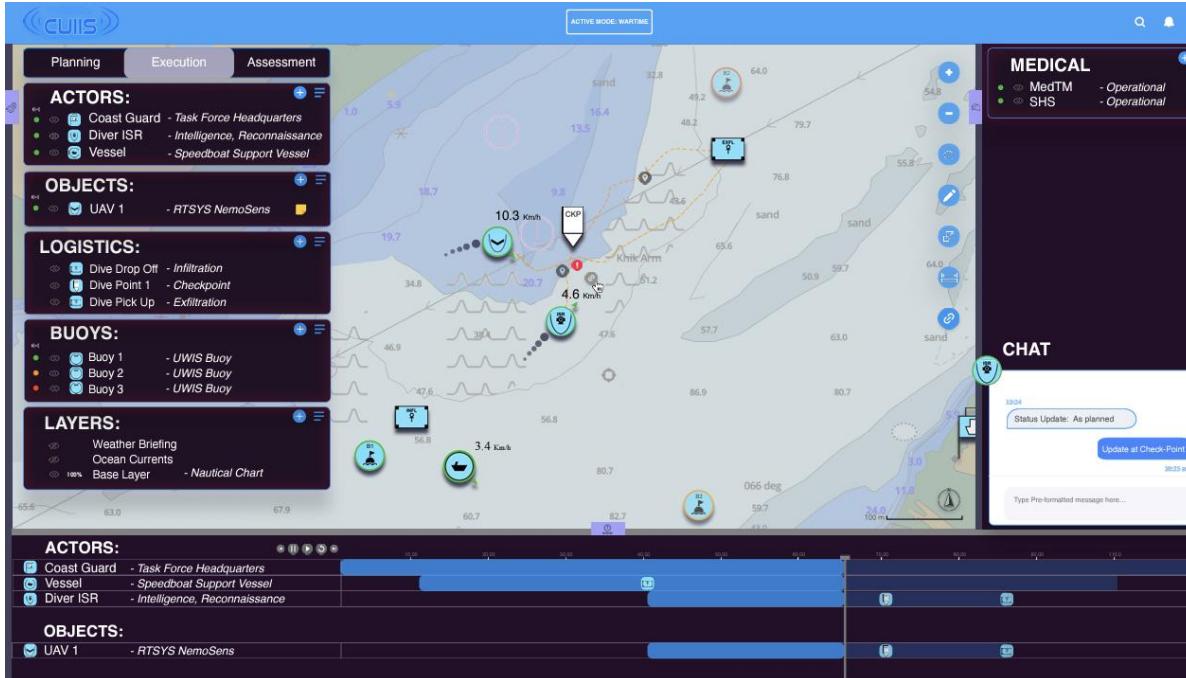
[DOI: https://doi.org/10.4236/ojs.2022.1205572](https://doi.org/10.4236/ojs.2022.1205572)

[DOI: https://doi.org/10.4236/ojs.2022.1205573](https://doi.org/10.4236/ojs.2022.1205573)

[DOI: https://doi.org/10.4236/ojs.2022.12055](https://doi.org/10.4236/ojs.2022.1205574)

GWALSH@IMADA.SDU.DK

On the CUIIS Project



SDU

Syddansk Universitet



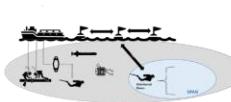
For my Vis Poster

VISUAL ANALYSIS OF MILITARY DIVING INCIDENT REPORTS

G. Walsh, N. S. Andersen, J. Kusnick, E. B. Sørensen, and S. Jänicke



1 Motivation & Background:



1) Military diving operations are inherently high-risk and complex, with the potential for accidents due to diverse tasks, advanced equipment, and varying operational contexts.

A MDIR database system should capture specialized tasks, advanced equipment data, environmental conditions, and geospatial/temporal aspects for comprehensive analysis.

The goal of such a MDIR VIS tool is to assist in the training and identification of safety hazards relating to equipment, human error and environmental impact.

This is done by aligning with models such as RAG, ALARMA and the Military Operations Process - to improve existing systems based on previous incidents.

2 Methodology:

Criteria	Proposed MDIR System	NICS	JDRS
Data Collection and Integration	Moderate	Moderate	Moderate
Data Visualization	Excellent	Moderate	Moderate
User Experience	User friendly	Non-existent	Medium
Scalability and Adaptability	Moderate	Moderate	Limited
Data Accuracy and Validation	Moderate	Moderate	Moderate
Training and Improvement Focus	Strong	Weak	Weak

Comparative Evaluation of MDIR Systems

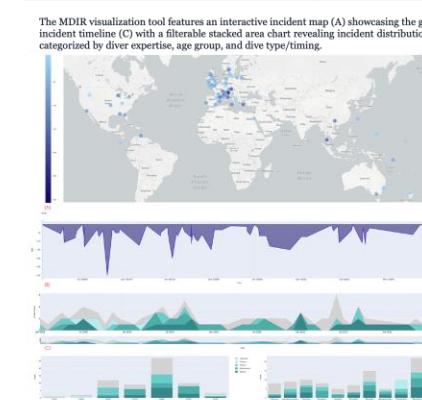


DJRS System



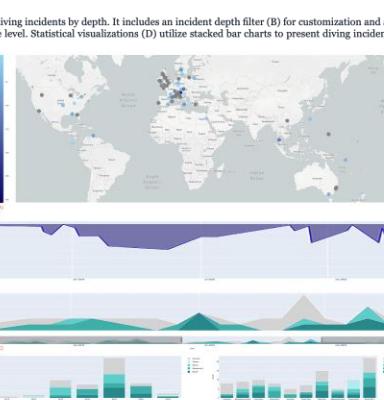
NICS System

3 MDIR Prototype:



The MDIR visualization tool features an interactive incident map (A) showcasing the spatial distribution of diving incidents by depth. It includes an incident depth filter (B) for customization and an incident timeline (C) with a filterable stacked area chart revealing incident distribution over time by experience level. Statistical visualizations (D) utilize stacked bar charts to present diving incidents categorized by diver expertise, age group, and dive type/timing.

4 Prototype Interactions:



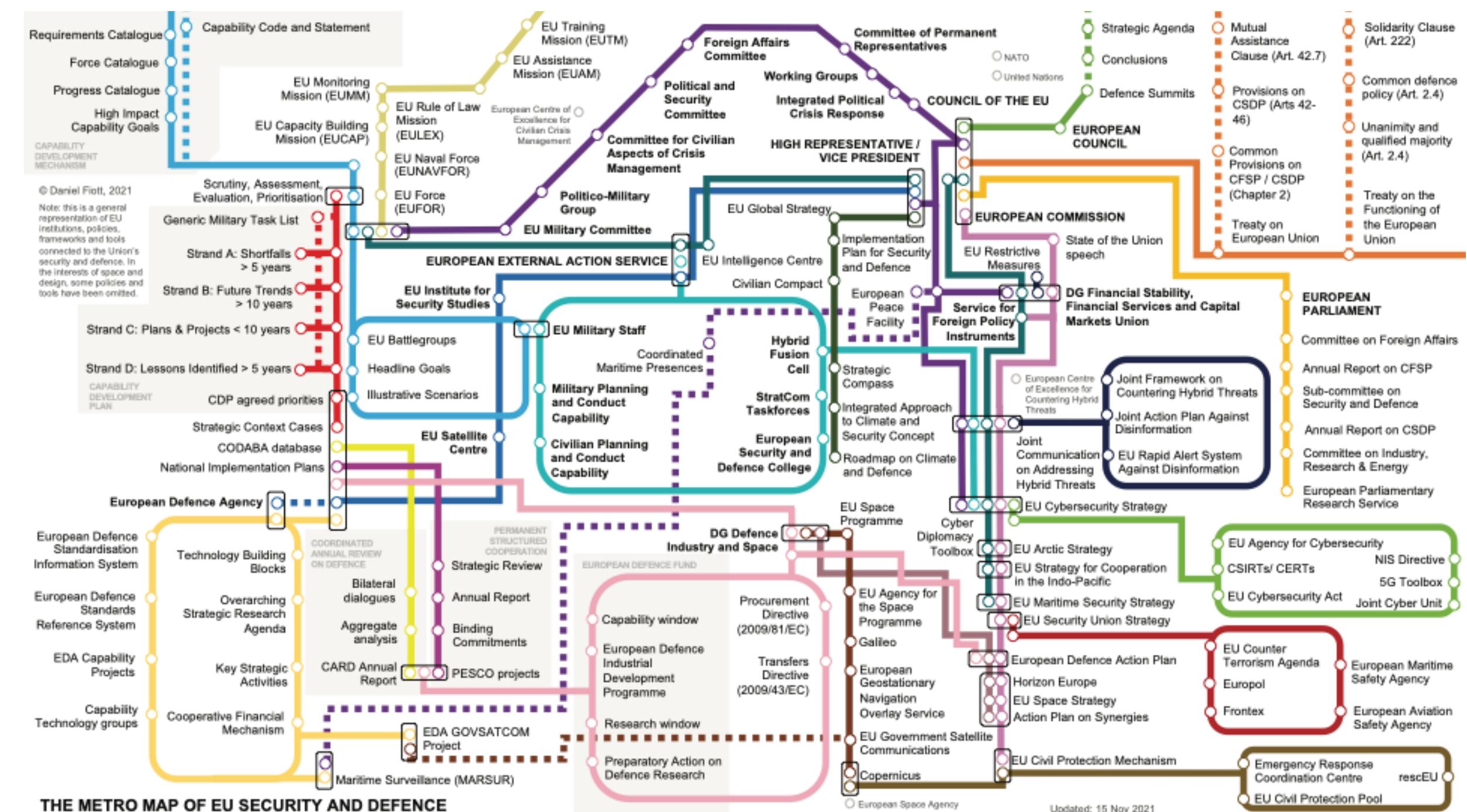


GWALSH@IMADA.SDU.DK

Making Sense of EU Defence Structures: A Policy-Focused Visualization Tool for Exploring EU Defence Entities and Procurement Pathways



Current work:



So, what's this course all about?

- Yes, it's about exams, grades, and building technical skills, but in my experience, it offers a lot more than that.
- Gives you the chance to **explore** a topic that you're genuinely interested in and turn that into a **practical, real-world visualization tool**
 - And some fun experiences in-between
- Examples from Last Year; students got to connect with domain experts and experts in the field

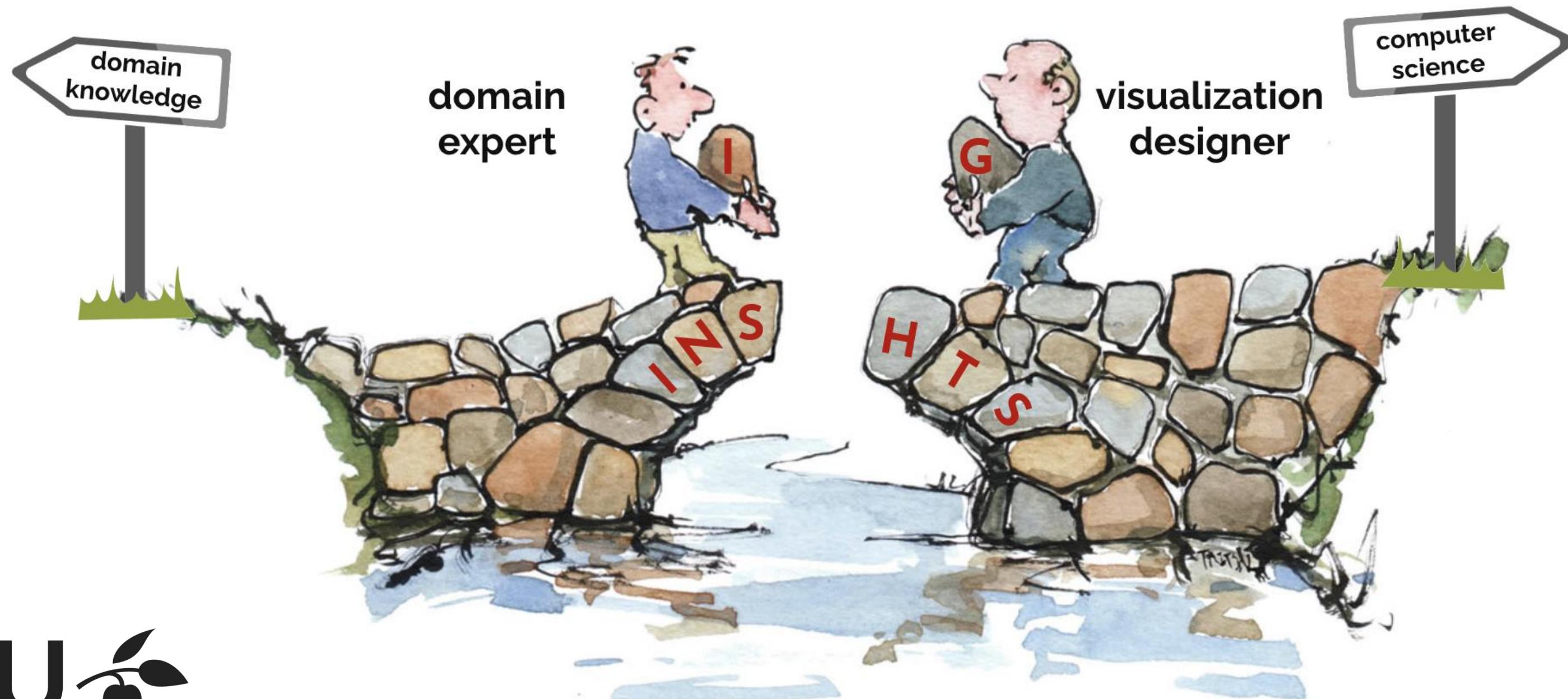




GWALSH@IMADA.SDU.DK

Building Bridges with Visualization

visualization enables data analytics





1) Idea Generation & Dataset



2) Sketching



3) Iteration



4) Dashboard Implementation



5) Report & Poster Session

So, what do we have to do?

Come up with a topic or area you are **passionate about**

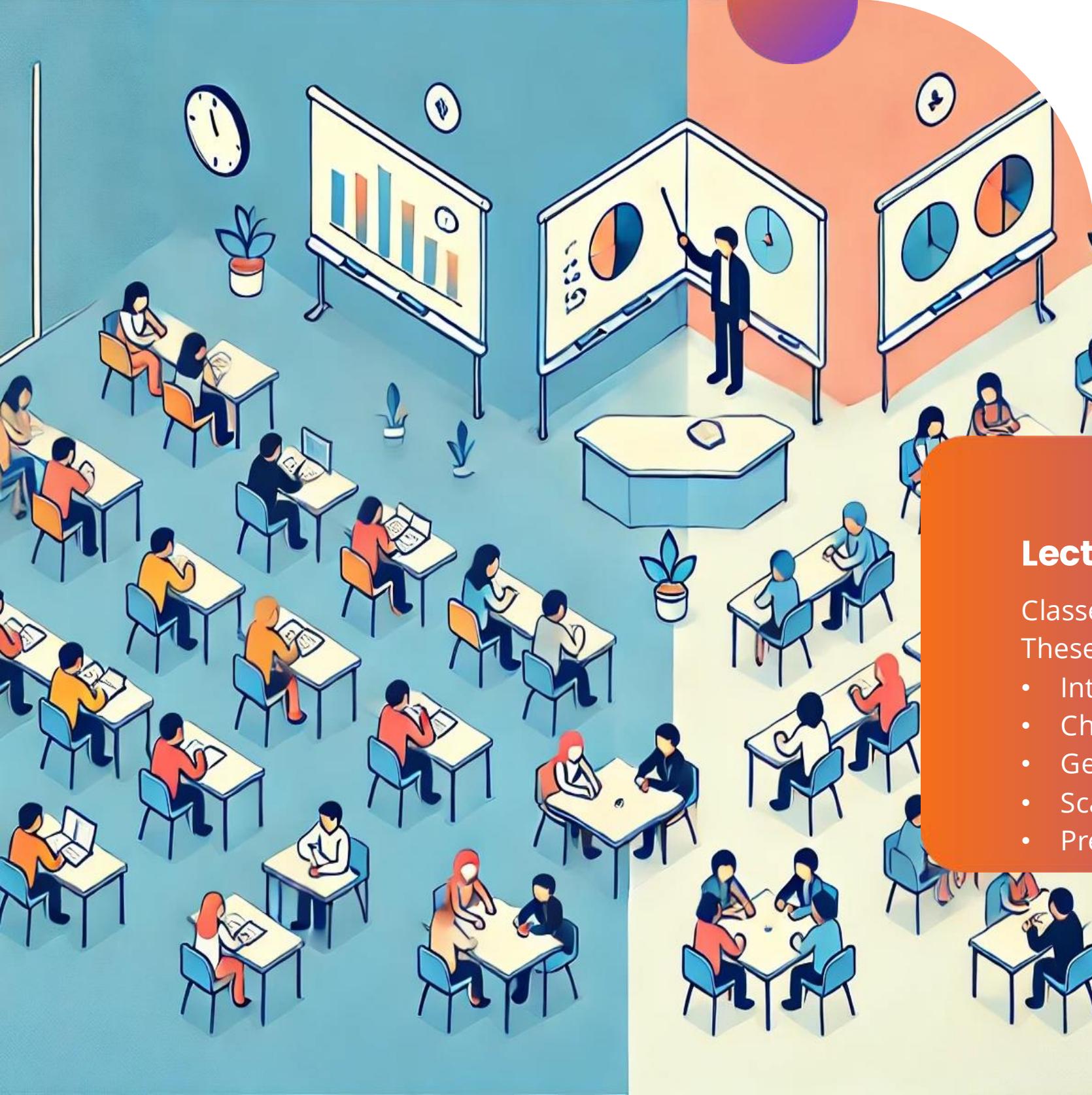
1. **Find a dataset** you think would be interesting to "story-tell" the data
2. Based on the data, **sketch-out** how you might do this
3. Start **iterating with visualizations** using Dash + Plotly
4. Continue iterating into an **interactive Dashboard with linked visualizations**
5. Write your project **report and present at the poster session**



Report & Poster Session

How will I be assessed?

- 1. A written scientific short paper (2-4 pages) describing the visualization project**
- 2. A (short) demonstration video of the visualization as supplemental material to the paper**
- 3. A shared presentation of the visualization project with an oral group discussion**
- 4. Short individual sessions with questions on theoretical contents of the course**



Structure + Agenda

Lectures

- Classes 3-4 will be a lecture style format.
These will cover:
- Intro to Dash/Plotly + Anaconda (D3)
 - Charts in Plotly + Examples
 - GeoJson + Examples
 - Scatter Matrix + Network Graph
 - Previous Student work + Examples

Group Feedback Sessions

- Classes 4-6 will be more group feedback sessions on your project.
- These will mostly be done in 10min slots
 - and I would prefer if you send material beforehand.



Dash

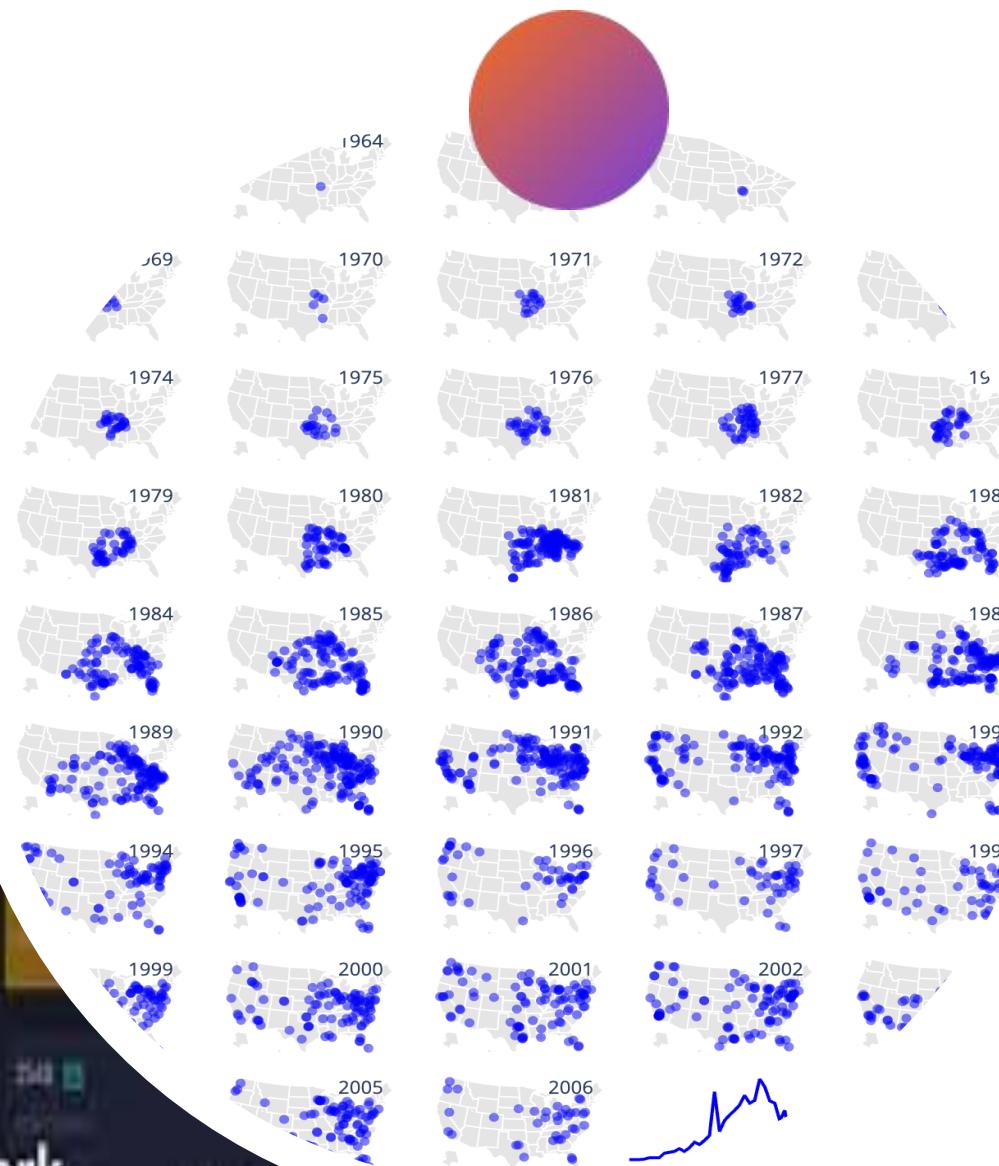
Flask
backend server

Plotly
charting library

React
interactive components

Dash

GWALSH@IMADA.SDU.DK



**Dashboards using
Python Dash Framework**

***These tools will help you
make your Dashboard***

What is Dash/Plotly? And how will it help you?

Dash is a **Python framework** for building analytical **web applications**. Dash helps in building responsive web dashboards that is good to look at and is very fast without the need to understand complex front-end frameworks or languages such as HTML, CSS, JavaScript.

Plotly is the underlying **visualization library** that Dash uses to **create its interactive graphs and charts**. While Dash is the framework that helps you build full web applications, Plotly is responsible for creating the actual visualizations (like line charts, scatter plots, heatmaps, and more) within those applications.

You use Plotly to define your charts, and then Dash allows you to embed those charts into a web application with added functionality (like user interaction, layout, and deployment).

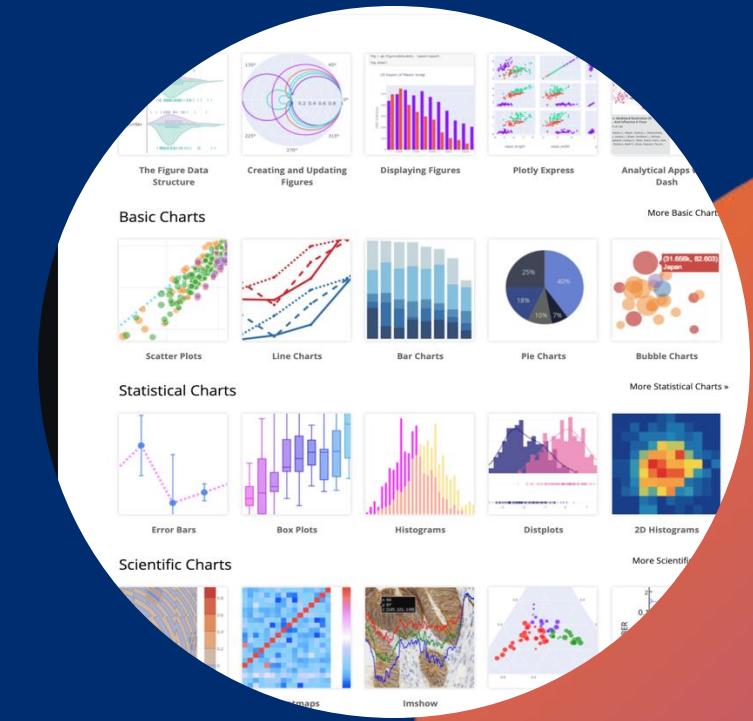
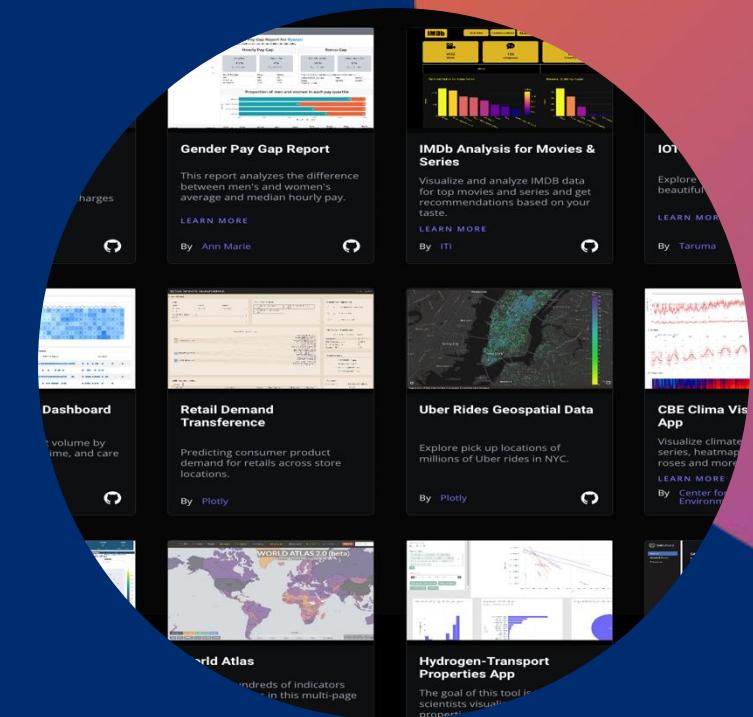
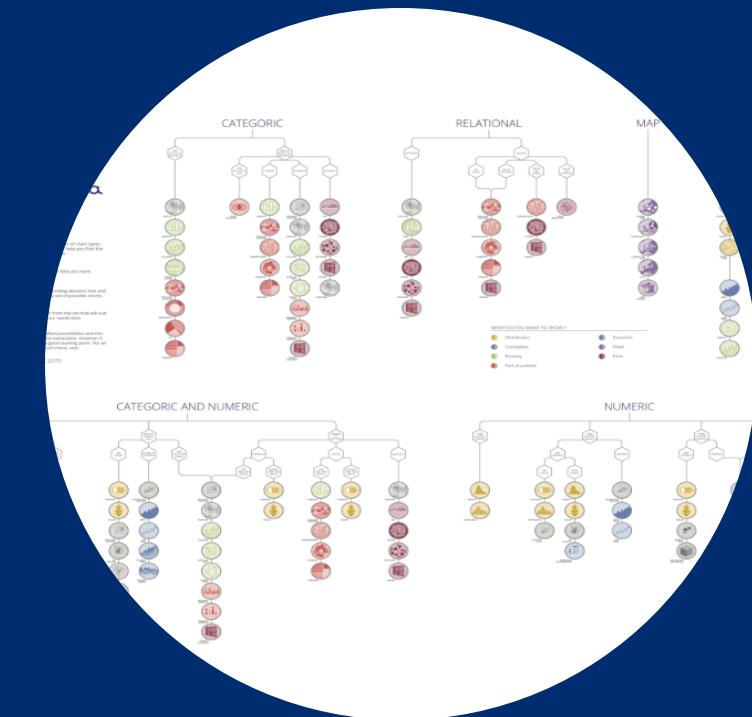
Dash = Web App Framework, and Plotly = Visualization Library used within Dash to create charts.



GWALSH@IMADA.SDU.DK

Resources to help you

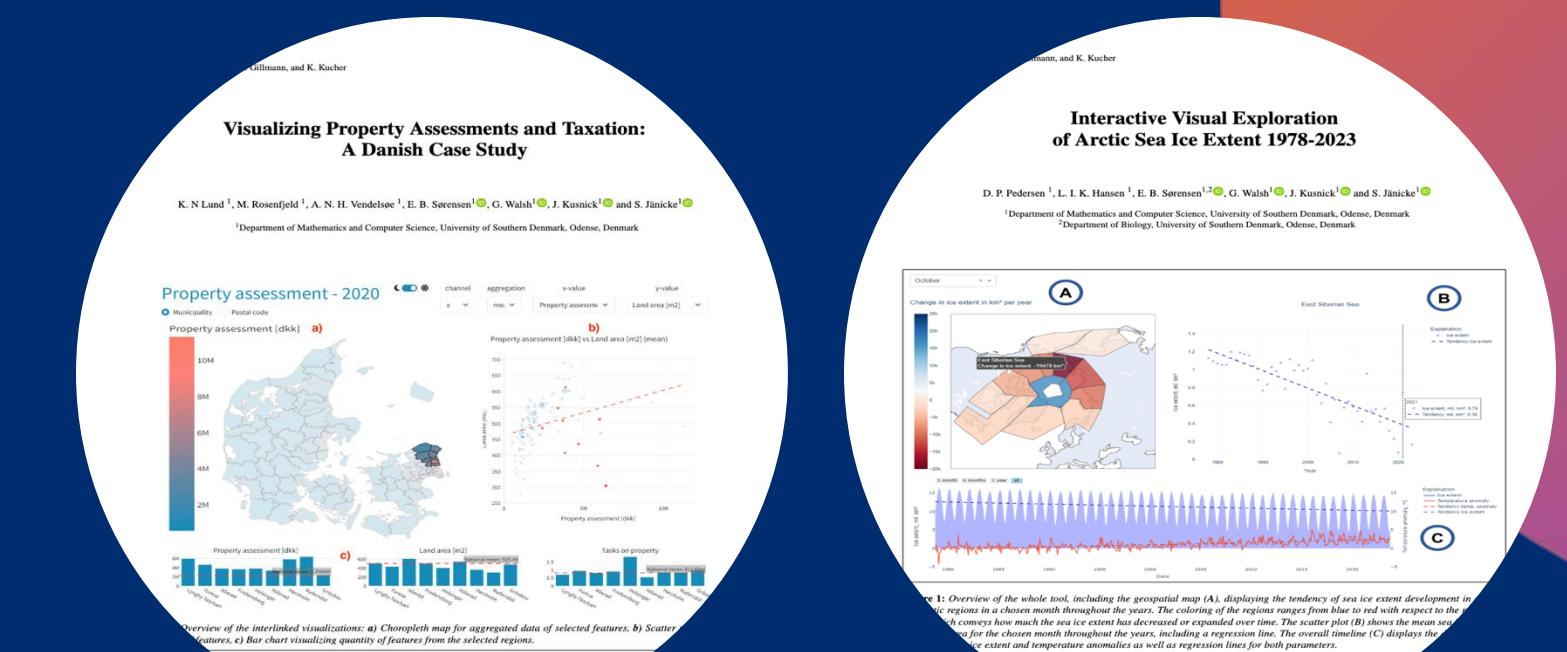
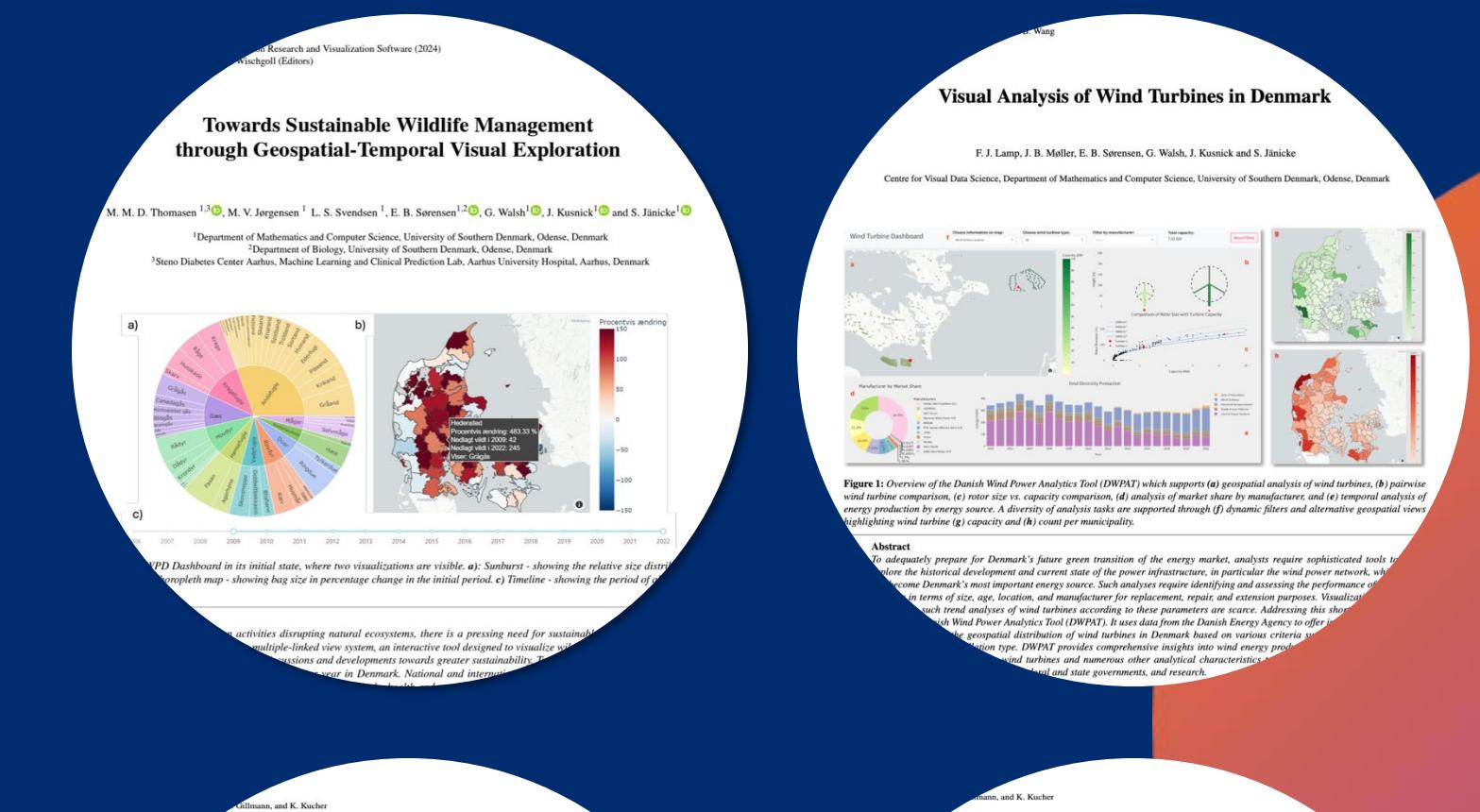
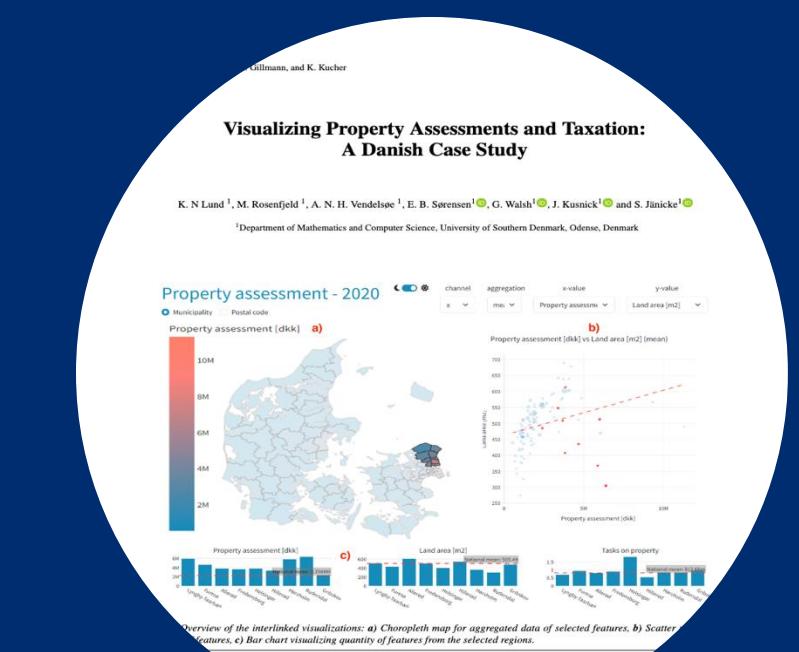
- Code Café
- Plotly Documentation: <https://plotly.com/python/> (a powerful tool for creating interactive graphs)
- <https://plotly.com/examples/dashboards/> (A collection of dashboard examples)
- Charming Data:
<https://www.youtube.com/@CharmingData/featured>
- D3 Graph Gallery: <https://d3-graph-gallery.com/>
- Awesome Dash GitHub Repository:
<https://github.com/ucg8j/awesome-dash>
- <https://dash.gallery/Portal/>
- From Data to Vis Poster
- <https://github.com/plotly/dash-sample-apps/tree/main/apps>





And Previous examples

- Interactive Visual Exploration of Arctic Sea Ice Extent 1978-2023
- Towards Sustainable Wildlife Management through Geospatial-Temporal Visual Exploration
- Visualizing Property Assessments and Taxation: A Danish Case Study
- Visual Analysis of Wind Turbines in Denmark



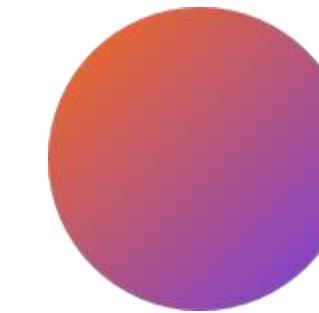


Other helpful tips to get started



Choosing a Project Topic

Pick a topic that **excites you**. Make sure there's enough **accessible data** to extract meaningful insights. Narrow your project to one clear question or narrative.



Where to Find Datasets

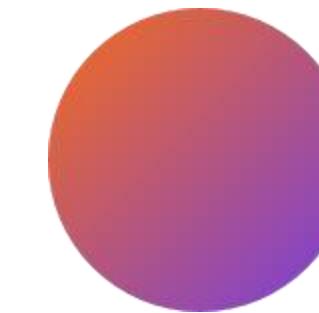
Kaggle Datasets - offers a vast collection of public datasets across many categories.

Google Dataset Search - is a powerful tool to explore millions of datasets across various domains.



Check Data Points and Data Cleaning

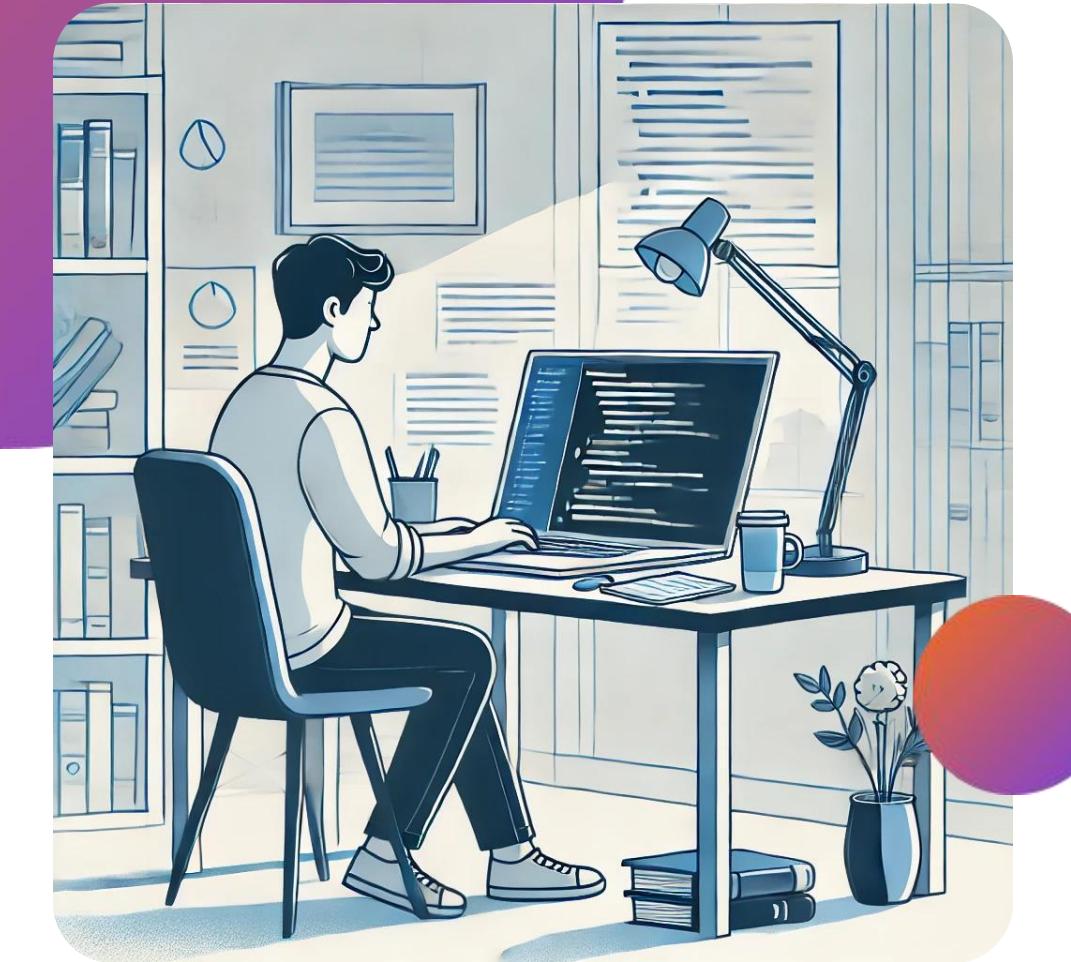
Check the DB includes the specific data points you need for your project. Also **consider what data cleaning or preprocessing might be required**. Many datasets need to be cleaned or transformed before they can be used in dashboards



Get to Know your Dataset

Check data is current and correct. Handle any missing values. Look out for duplicates or inconsistent formatting. Assess outliers and think about how you might visualize them. Ensure labels and IDs are correct.

Ways to work



As a Group

- Between 2-5 people
- Distribution of tasks

Individually

- Allows more personal autonomy
- Increased workload/ less feedback



GWALSH@IMADA.SDU.DK

QUESTIONS

E N D O F P A R T 1

Coming back to Building Web Apps with Dash

What Makes Up Dash?

Flask (Backend Server):
Handles the **server-side** of your app

Plotly (Charting Library):
Creates interactive visualizations like graphs

React (Interactive UI Components):
Front-end interactions (like dropdowns, buttons)



Basic Dash App Structure (How to Build a Dash App)

Imports:

```
import dash
import dash_html_components as html
import dash_core_components as dcc
```

Creating the App:

```
app = dash.Dash(__name__)
```

App Layout
(What your user sees):

```
app.layout = html.Div([
    dcc Dropdown(),  #
    dcc Graph()      #
])
```

Callbacks
(Making the app interactive):

```
@app.callback()
def update_graph(...):
    return ...
```

```
if __name__ == '__main__':
    app.run_server()
```

Run the App:



GWALSH@IMADA.SDU.DK



Why use Plotly/Dash?

All-in-One Solution:

Dash combines the power of data visualization with web app functionality—**all in Python**. No need for extra tools like HTML or JavaScript.

Interactive Data Exploration:

With Plotly, you can create highly interactive graphs that let users explore data by **zooming, filtering, and hovering for details**. This makes your dashboard much more engaging than static charts.



Why Use Spyder and Anaconda for Plotly Dash Dashboards?



Easy Setup:

Anaconda makes installing Dash, Plotly, and all dependencies quick and straightforward.

Integrated Development:

Spyder provides an all-in-one workspace to write, test, and debug your Dash code

Spyder's variable explorer and plotting pane:

Allows you to preview and interact with your data.

Environment Management:

Conda simplifies managing multiple environments, keeping your projects organized and isolated.





GWALSH@IMADA.SDU.DK

Spyder and Anaconda Plotly Dash Installation Guide



Syddansk Universitet





GWALSH@IMADA.SDU.DK

<https://www.anaconda.com/products/individual>

Download Installer



Products ▾

Pricing

Solutions ▾

Resources ▾

Blog

Company ▾

Get Started



Individual Edition

Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data

science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

SDU



Syddansk Universitet

Anaconda Individual Edition

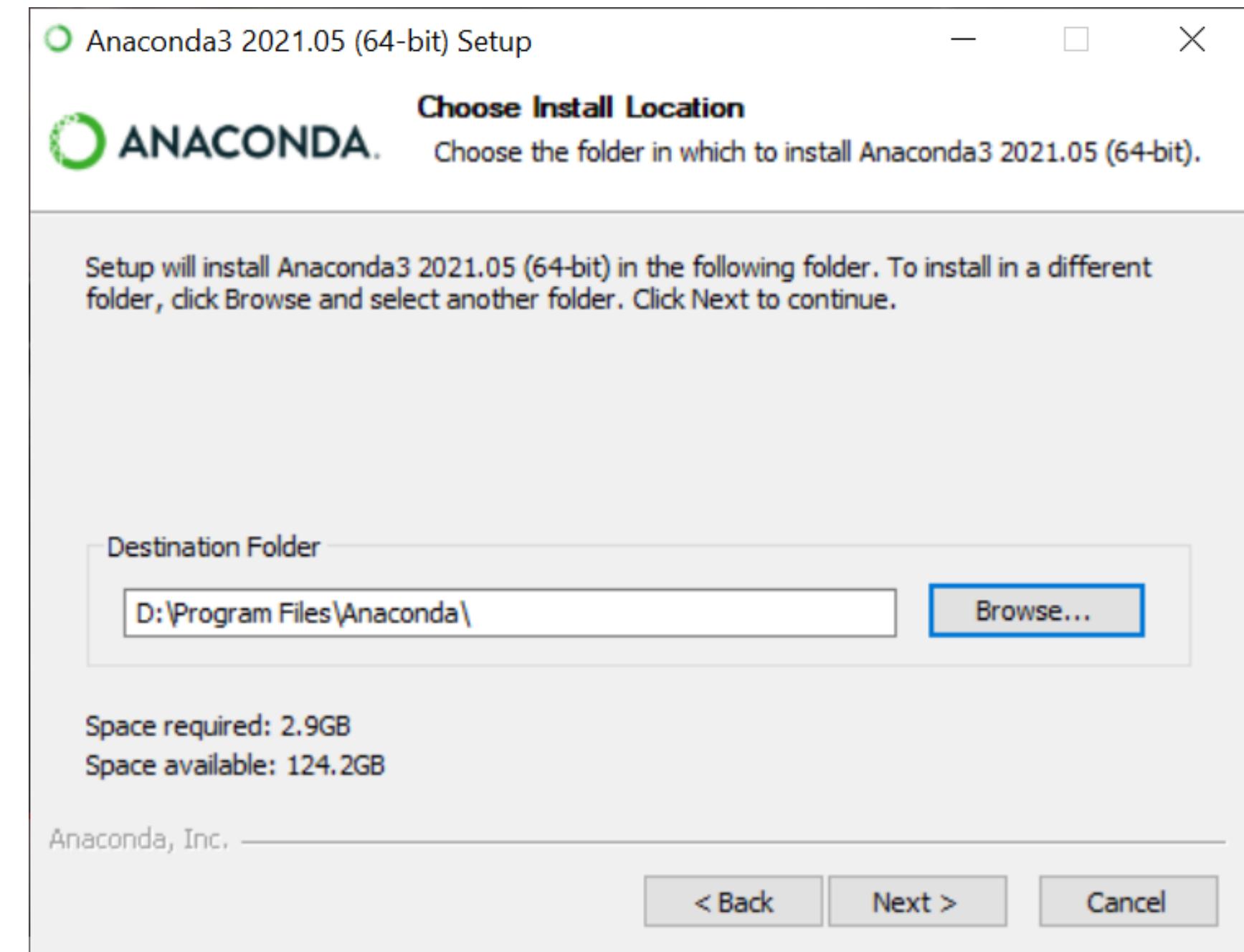
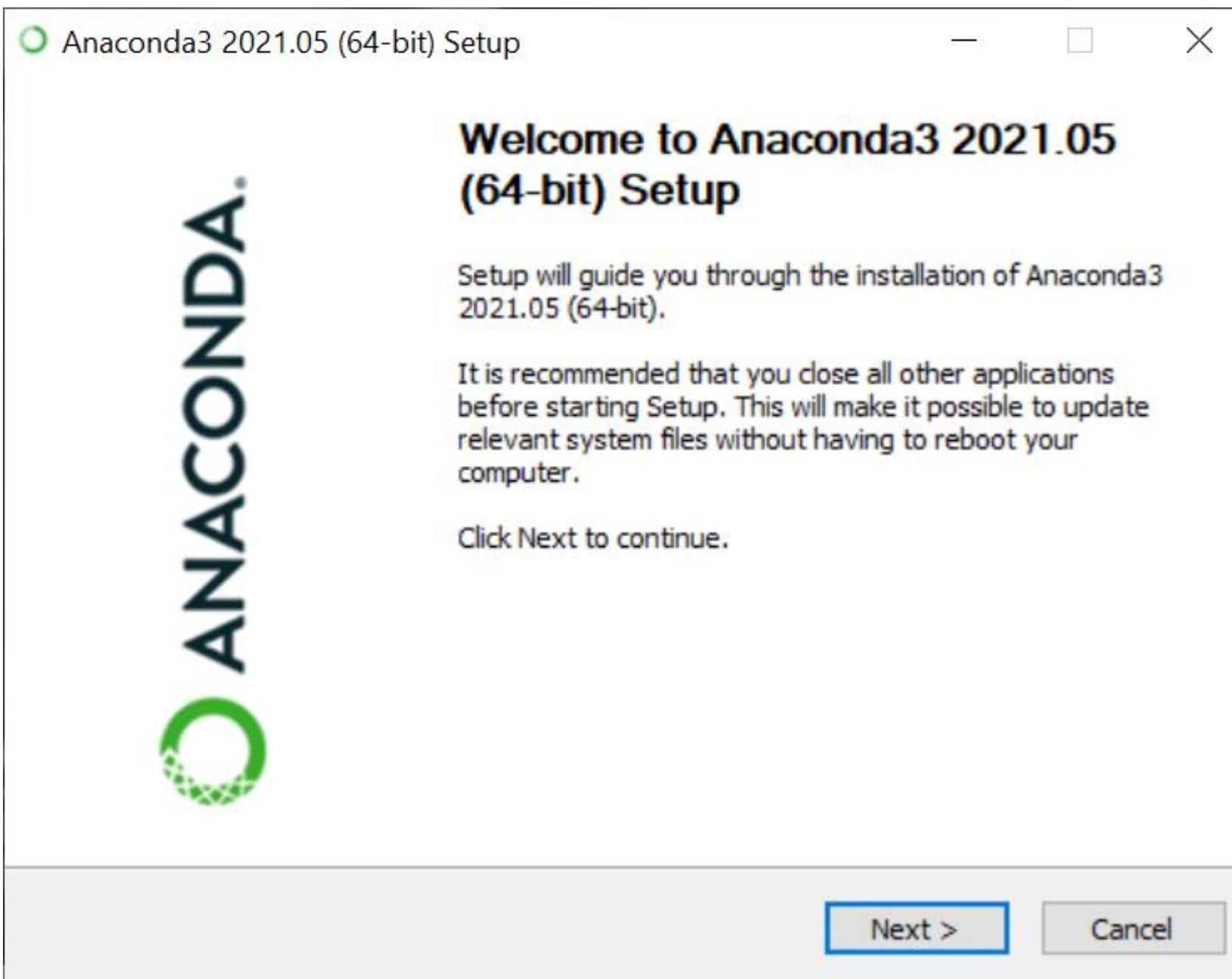
[Download](#)

For MacOS
Python 3.8 • 64-Bit Graphical Installer • 440 MB

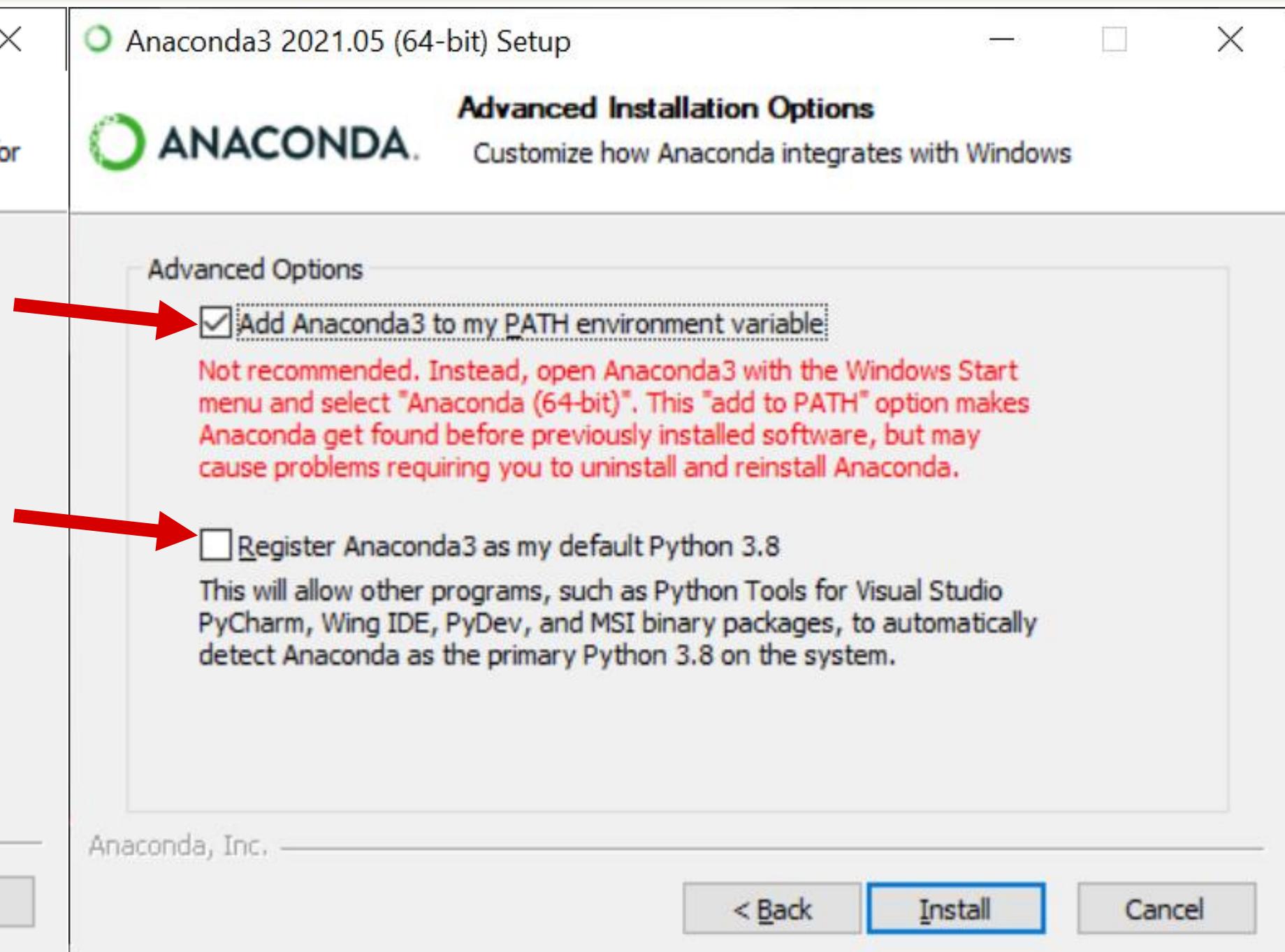
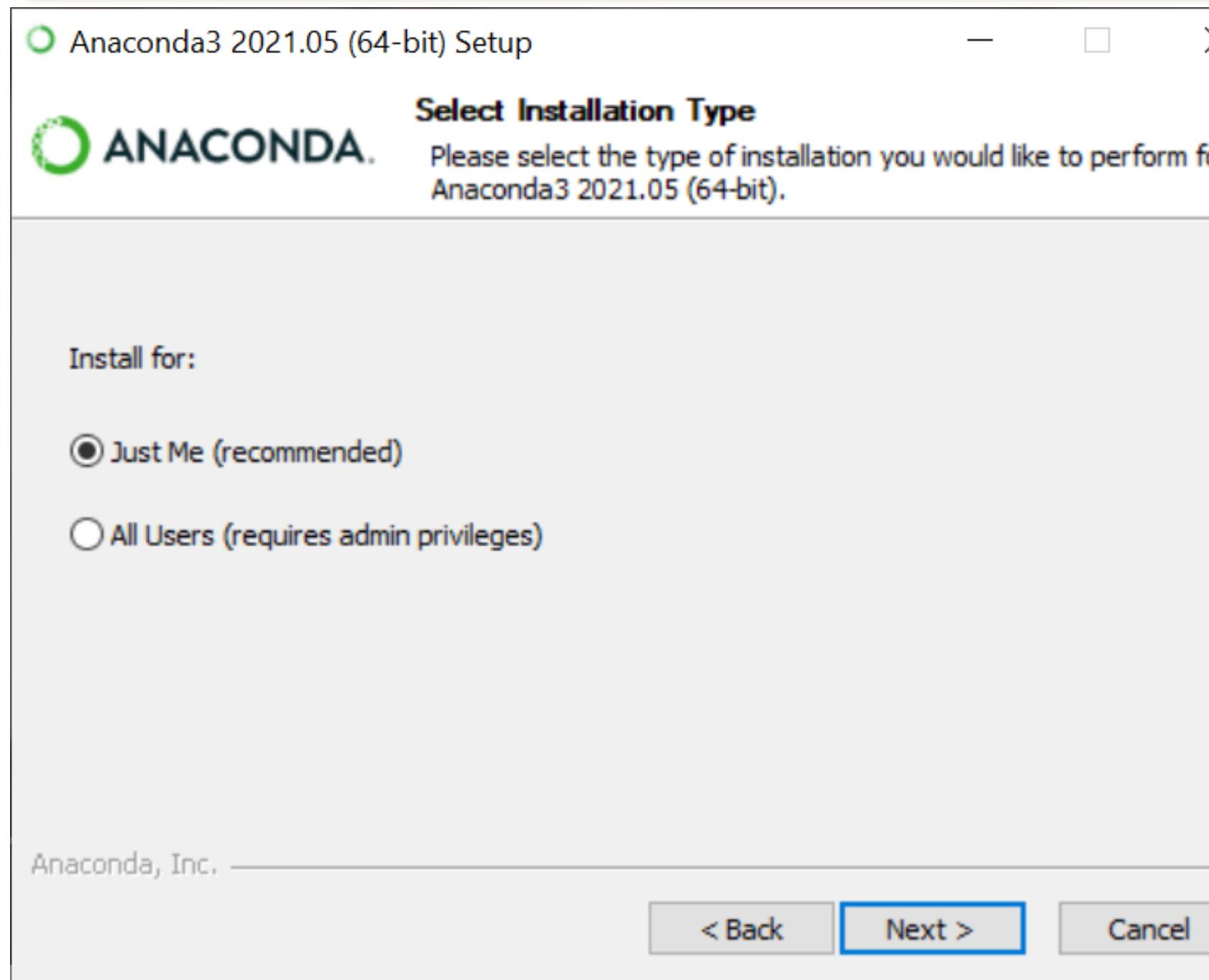
[Get Additional Installers](#)

| |

Anaconda Installation for Windows



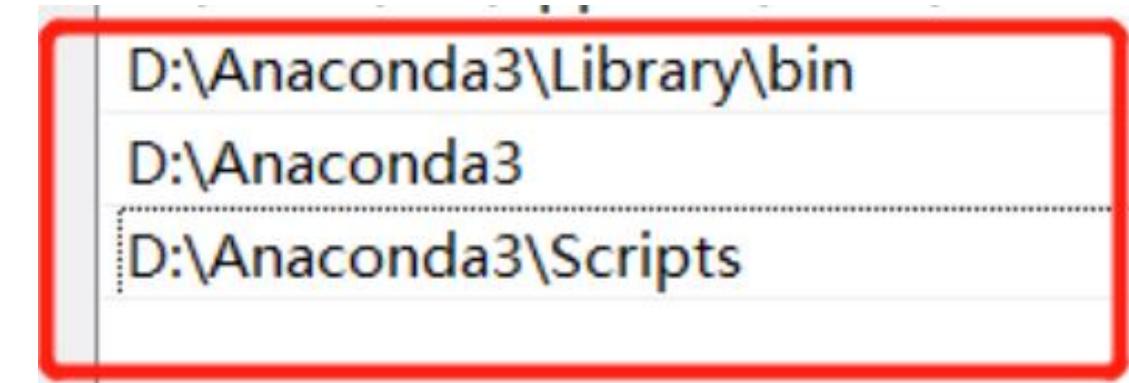
Anaconda Installation for Windows



Anaconda Installation for Windows

Setup environment (if you don't check the environment option)

- Find the **system properties**
- Click “**Advanced**”
- Choose “**Environment Variables**”
- Under the “**user variables**” edit “**Path**”
- Add three new path about anaconda (it should depend on your installation path)



Verifying your installation

Windows:

Click Start, search or select Anaconda Navigator from the menu.



Click Start, search, or select Anaconda Prompt from the menu.

Type `conda --v` and it will return the version of conda

Mac:

Cmd+Space to open Spotlight Search and type “Navigator” to open the program.



Cmd+Space to open Spotlight Search and type “Terminal” to open the terminal program.

Type `conda --v` and will return the version of conda on the screen.

Anaconda Navigator

The screenshot shows the Anaconda Navigator application window. The main area displays a grid of data science tools:

Tool	Description	Version	Action Buttons
Datalore	Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.	3.0.14	Launch
IBM Watson Studio Cloud	IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.	3.0.14	Launch
JupyterLab	An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.	3.0.14	Launch
Notebook	Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.	6.3.0	Launch
PyCharm Professional	A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.	2021.1.2	Launch
Qt Console	PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.	5.0.3	Launch
Spyder	Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features.	4.2.5	Launch
VS Code	Streamlined code editor with support for development operations like debugging, task running and version control.	1.61.0	Launch
Glueviz	Multidimensional data visualization across files. Explore relationships within and among related datasets.	1.0.0	Install
Orange 3	Component-based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.	3.26.0	Install
RStudio	A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.	1.1.456	Install

The sidebar on the left includes links for Home, Environments, Learning, and Community, along with a section for ANACONDA.NUCLEUS featuring a "Join Now" button and links for Discover premium data science content, Documentation, and Anaconda Blog. Social media icons for Twitter, YouTube, and GitHub are also present at the bottom of the sidebar.

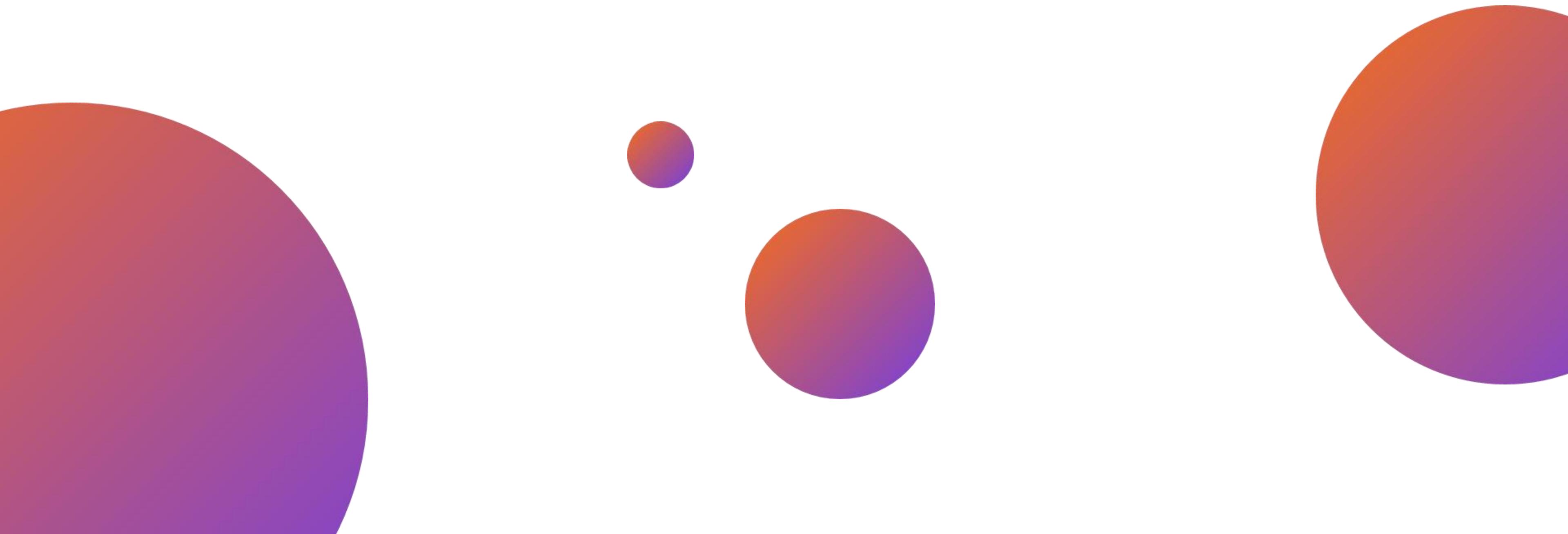


Plotly Installation

```
conda install -c plotly plotly=5.11.0
```

or

```
pip install plotly==5.11.0
```





Dash Installation

```
conda install -c conda-forge dash
```

or

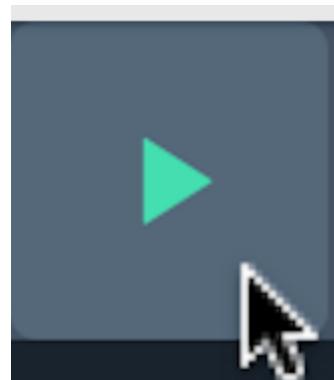
```
pip install dash
```



GWALSH@IMADA.SDU.DK

Test it!

- Open iris.py (see resources on itsLearning) with Spyder
- Hit the ‘run’ button!



- Server should run on port on your system:

Or run `python iris.py` in terminal app



```
Dash is running on http://127.0.0.1:8080/
 * Serving Flask app "iris" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```

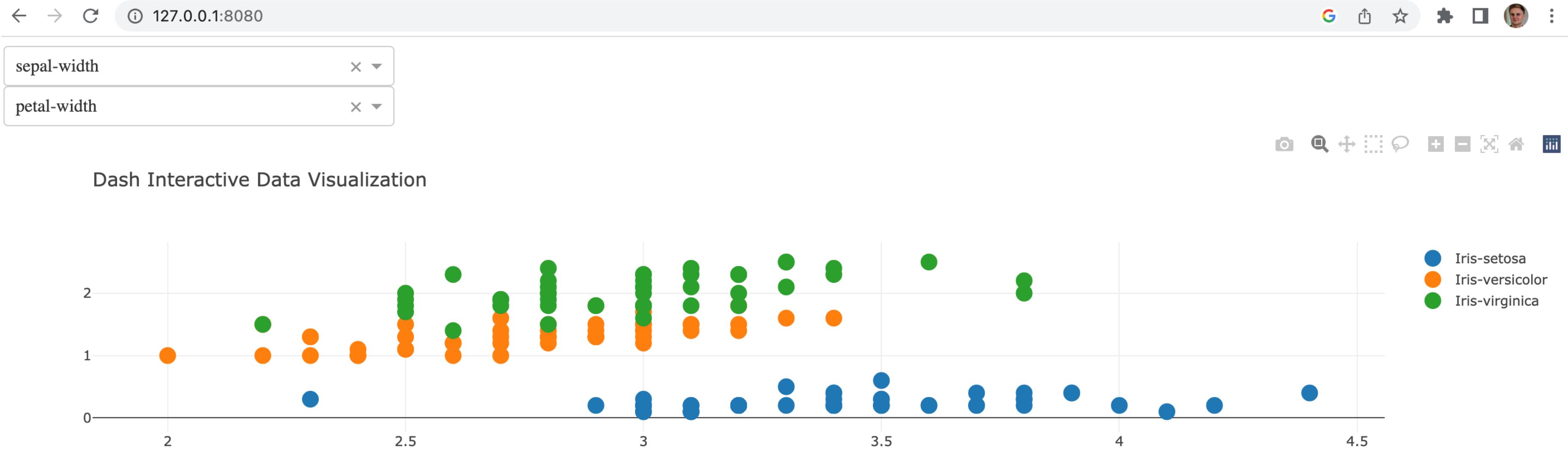


Syddansk Universitet



GWALSH@IMADA.SDU.DK

Iris.py



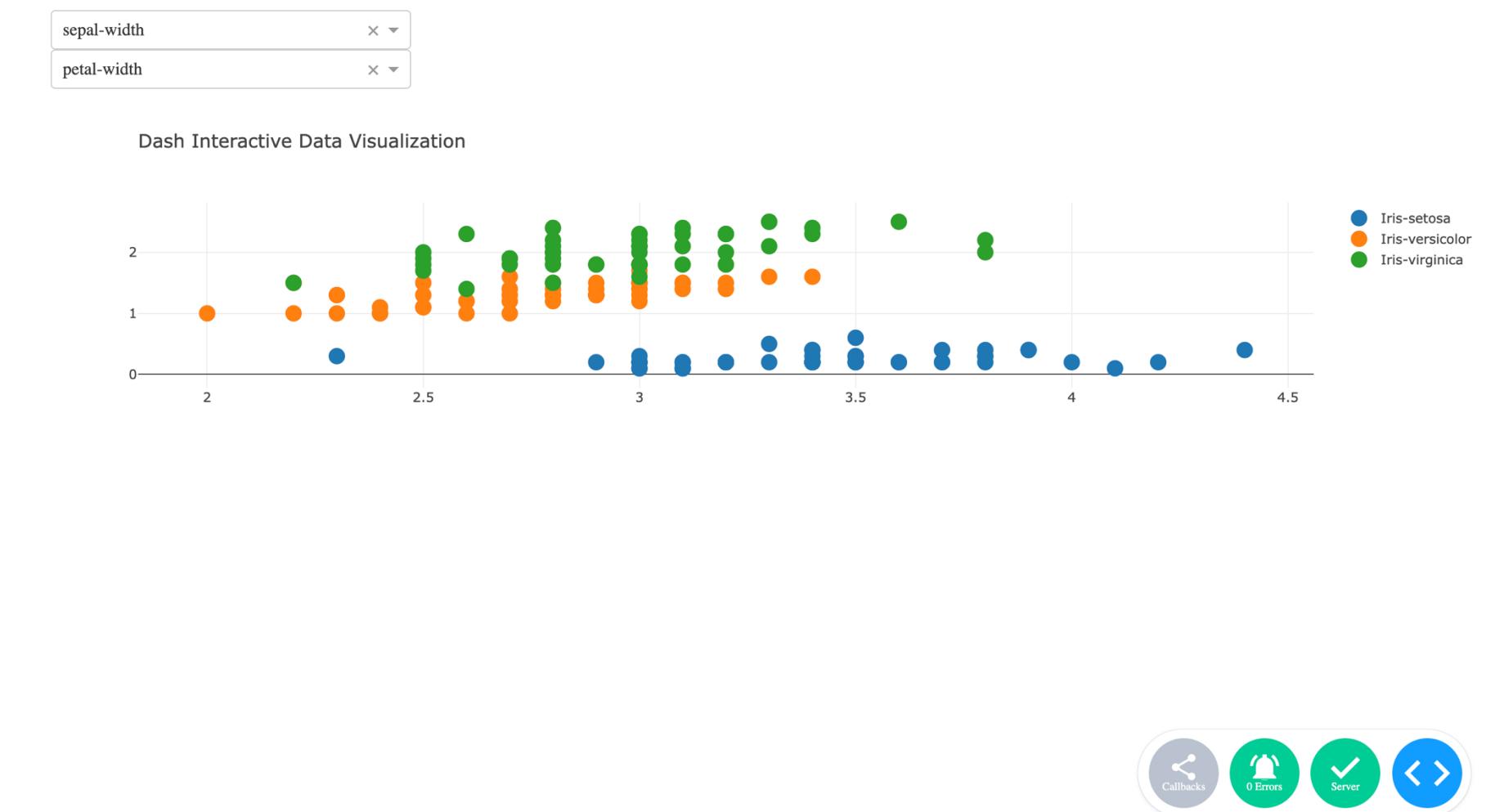
Syddansk Universitet



GWALSH@IMADA.SDU.DK

Turn on de-bugging mode

```
if __name__ == '__main__':
    app.run_server(debug=True, port=8080)
```



Syddansk Universitet



Explanation of iris.py

- 1) Import Libraries
- 2) Set up the dash application
- 3) Establish a list with commonly used attributes in the data
- 4) Read the data from a URL using pandas

```
# coding=utf8
import random
import pandas as pd
import dash
from dash.dependencies import Input, Output
from dash import dcc, html
import plotly.graph_objs as go

app = dash.Dash(__name__)

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', names=names)
```



GWALSH@IMADA.SDU.DK

Explanation of iris.py

Create the app layout:

- 1) Create a div for 2 dropdown options to select x & y values
- 2) Create another div for the graph

```
app.layout = html.Div([
    html.Div([
        dcc Dropdown(
            id='ddl_x',
            options=[{'label': i, 'value': i} for i in names],
            value='sepal-width',
            style={'width': '50%'}
        ),
        dcc Dropdown(
            id='ddl_y',
            options=[{'label': i, 'value': i} for i in names],
            value='petal-width',
            style={'width': '50%'}
        ),
    ], style={'width': '100%', 'display': 'inline-block'}),
    html.Div([
        dcc Graph(id='graph1')
    ], style={'width': '100%', 'display': 'inline-block'})
])
```



Syddansk Universitet



Explanation of iris.py

Adding a Callback in Dash:

@app.callback() Decorator:

- This defines the callback function, which dynamically updates the dashboard when the input values change.
- **Output:** Specifies what part of the dashboard should be updated. In this case, it updates the figure of the component with ID graph1.
- **Inputs:** The inputs specify which elements trigger the update. Here, we have two dropdowns (ddl_x and ddl_y), and their values are the inputs.

update_output() Function:

- This function takes the input values (from the dropdowns) and returns an updated **Plotly figure**.
- **Data Filtering:** It filters the dataset based on the input values (x-axis and y-axis), creating a scatter plot for each unique class.
- **Layout:** Sets the layout properties of the graph, such as **height**, **hovermode**, and **title**.

```
@app.callback(
    Output(component_id='graph1', component_property='figure'),
    [
        Input(component_id='ddl_x', component_property='value'),
        Input(component_id='ddl_y', component_property='value')
    ]
)
def update_output(ddl_x_value, ddl_y_value):
    figure={
        'data': [
            go.Scatter(
                x=data[data['class'] == cls][ddl_x_value],
                y=data[data['class'] == cls][ddl_y_value],
                mode='markers',
                marker={'size': 15},
                name=cls
            ) for cls in data['class'].unique()
        ],
        'layout':
            go.Layout(
                height= 350,
                hovermode= 'closest',
                title=go.layout.Title(text='Dash Interactive Data Visualization', xref='paper', x=0)
            )
    }
    return figure

if __name__ == '__main__':
    app.run_server(debug=True, port=8080)
```



GWALSH@IMADA.SDU.DK

What is a Callback and Why Use them?

What is a Callback?

- A **callback** is a function that automatically gets triggered when certain **inputs** change, updating specific parts of the dashboard.
- In Dash, callbacks connect user actions (like selecting from a dropdown) to the app's visual output.

Why is it Useful?

- **Interactivity:** Callbacks allow your dashboard to respond instantly to user inputs, updating charts, graphs, or text dynamically.
- **Real-Time Data Updates:** When new data is selected or filters are changed, the graph or chart updates in **real-time** without needing to reload the page.
- **Customization:** You can use callbacks to create highly customizable dashboards where users control the displayed data.



GWALSH@IMADA.SDU.DK

How to Create a Callback

@app.callback() Decorator:

Defines the **output** (what gets updated) and **input** (what triggers the update).

Example:

```
@app.callback( Output('graph_id', 'figure'),  
[Input('dropdown_id', 'value')] )
```

Callback Function:

The function takes the **input values** and returns the updated **figure** or component.

Example:

```
def update_output(selected_value): figure =  
go.Figure(...) return figure
```



Syddansk Universitet



GWALSH@IMADA.SDU.DK

- htmlTutorial.html
- htmlTutorial2.html

HTML & CSS



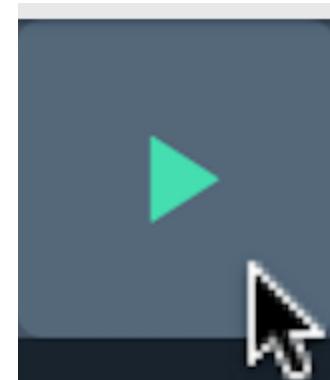
Syddansk Universitet



GWALSH@IMADA.SDU.DK

Volcanos

- Open volcanos.py (see resources on itsLearning) with Spyder
- Hit the ‘run’ button!



- Server should run on port on your system:

Or run `python volcanos.py` in terminal app



```
Dash is running on http://127.0.0.1:8080/
 * Serving Flask app "volcanos" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```



GWALSH@IMADA.SDU.DK

Volcanos.py Explanation

Import Libraries :

- pandas is used for data manipulation and analysis.
- dash and its components (dcc, html) are used for creating the layout and interactivity of the web app.
- plotly.express is used for creating interactive plots.
- Input and Output are used for callback functions in Dash, which update the app's components.

Initialize Dash App:

Read the data:

from a CSV file named 'volcanos.csv' into a pandas Data Frame

```
import pandas as pd
from dash.dependencies import Input, Output
import dash
from dash import dcc, html
import plotly.express as px

app = dash.Dash(__name__)

data = pd.read_csv('volcanos.csv', encoding='utf-8')
```



Syddansk Universitet



GWLASH@IMADA.SDU.DK

Volcanos.py Explanation

Prepare Dropdown Options:

- Extract unique volcano types and rock types from the data, and create options for dropdown menus.
- An additional option 'All Volcano Types' and 'All Rock Types' are appended to the list.

```
types = data['primary_volcano_type'].unique()
type_options = [{'label': i, 'value': i} for i in types]
type_options.append({'label': 'All Volcano Types', 'value': 'all'})

rocks = data['major_rock_1'].unique()
rock_options = [{'label': i, 'value': i} for i in rocks]
rock_options.append({'label': 'All Rock Types', 'value': 'all'})
```

Time Slider Range:

- Defines the minimum and maximum values for the time slider. The values range from -10500 to 2100.

```
mintime = -10500
maxtime = 2100
```



Syddansk Universitet

Volcanos.py Explanation

```
app.layout = html.Div([
    html.H1(children="The World's Volcanos",
           style = {'textAlign': 'center', 'font-family' : 'Roboto'}),
    html.Div([
        dcc Dropdown(
            id='volcano_types',
            options=type_options,
            value='all',
            style={'width': '50%', 'display': 'inline-block'}
        ),
        dcc Dropdown(
            id='volcano_rocks',
            options=rock_options,
            value='all',
            style={'width': '50%', 'display': 'inline-block'}
        )
    ],style={'width': '100%', 'display': 'inline-block'}),
    html.Div([
        html.Div([
            dcc.Graph(id='volcano-map')
        ],style={'width': '46%', 'display': 'inline-block', 'vertical-align': 'top', 'margin': '2%'}),
        html.Div([
            html.Div(id='miframe')
        ],style={'width': '46%', 'display': 'inline-block', 'vertical-align': 'top', 'margin': '2%'})
    ]),
    html.Div([
        dcc.RangeSlider(
            id='time-slider',
            min=mintime,
            max=maxtime,
            step=100,
            value=[mintime,maxtime],
            marks={i: str(i) for i in range(mintime, maxtime, 500)})
    ])
])
```

Create Layout of the App:

- The app.layout defines the HTML structure of the web app.
- It includes a title, two dropdown menus for filtering by volcano type and rock type, a map for visualization, an iframe to display Wikipedia content, and a time slider.

Volcanos.py Explanation

Callback with input components: volcano type, rock type, and time.

Function starts with the entire dataset and applies filters based on the input values:

If a specific volcano type is selected (not 'all'), it filters the data to only include volcanoes of that type.

Similarly, if a specific rock type is selected, it further filters the data.

Lastly, it filters the data based on the selected time range from the time slider.

Plotly.express is then used to create a scatter map. This map displays the locations of the filtered volcanoes using latitude and longitude data.

The hover information includes the volcano's name.. The zoom level is set to 0, and the map height is set to 1000 pixels.

The function returns the figure (fig), which automatically updates the 'figure' property of the 'volcano-map' component in the app layout.

```
@app.callback(
    Output(component_id='volcano-map', component_property='figure'),
    [
        Input(component_id='volcano_types', component_property='value'),
        Input(component_id='volcano_rocks', component_property='value'),
        Input(component_id='time-slider', component_property='value')
    ]
)
def update_output(volcano_type, volcano_rock, time):
    mydata = data
    if volcano_type != 'all':
        mydata = data[data['primary_volcano_type'] == volcano_type]
    if volcano_rock != 'all':
        mydata = mydata[mydata['major_rock_1'] == volcano_rock]
    if time != [mintime,maxtime]:
        mydata = mydata[mydata['last_eruption_year'] >= time[0]]
        mydata = mydata[mydata['last_eruption_year'] <= time[1]]
    fig = px.scatter_mapbox(data_frame=mydata,
                           lat="latitude",
                           lon="longitude",
                           hover_name="volcano_name",
                           hover_data=["primary_volcano_type","tectonic_settings"],
                           color="primary_volcano_type",
                           size=[1 for i in mydata['volcano_number']],
                           size_max=10,
                           zoom=0,
                           height=1000)
    fig.update_layout(mapbox_style="open-street-map")
    fig.update_layout(margin={"r":0,"t":0,"l":20,"b":0})
    return fig
```

Volcanos.py Explanation

Output here is the 'children' property of the component with id 'miframe'. In Dash, the 'children' property often refers to the content inside a component.

Input is the 'clickData' property of the 'volcano-map' component. This property contains information about the point on the map that the user clicks.

update_wiki takes one parameter, click_data. function then checks if click_data is not None. If true, it constructs a new URL using the volcano's name from the click data.

Finally, the function returns an html.Iframe component with the constructed URL as its source

```
@app.callback(Output('miframe', 'children'),
              [Input('volcano-map', 'clickData')])
def update_wiki(click_data):
    url = "https://en.wikipedia.org/wiki/Volcano"
    if click_data != None:
        url = "https://en.wikipedia.org/wiki/"+click_data['points'][0]['hovertext'].replace(" ", "_")
    return [
        html.Iframe(src=url, style={'width': '100%', 'height': '1000px', 'display': 'inline-block'})
    ]
if __name__ == '__main__':
    app.run_server(debug=True, port=8080)
```



GWALSH@IMADA.SDU.DK

Thank You

E N D O F P A R T 2