GWALSH@IMADA.SDU.DK

DS808/DM878

Visualization

EXERCISE CLASS 1

GARETH WALSH, PHD FELLOW

SDU
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**.

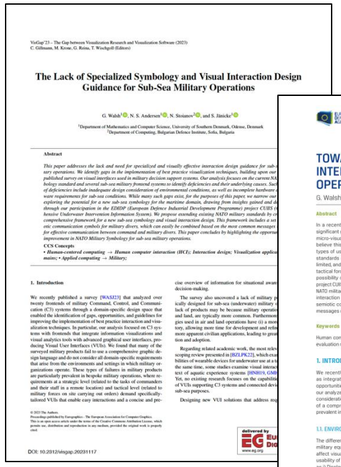
SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

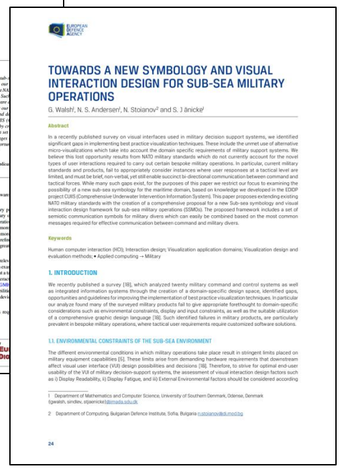
About my research

A Survey of Geospatial-Temporal Visualizations for Military Operations

Visualizing Military Operations: Extended Geospatial-Temporal Survey

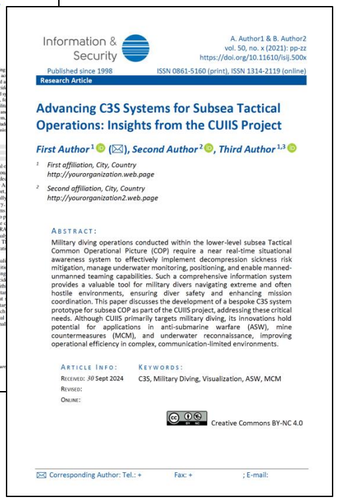


The Lack of Specialized Symbology and Visual Interaction Design Guidance for Sub-Sea Military Operations



TOWARDS A NEW SYMBOLOGY AND VISUAL INTERACTION DESIGN FOR SUB-SEA MILITARY OPERATIONS

Visual Analysis of Military Diving Incident Reports



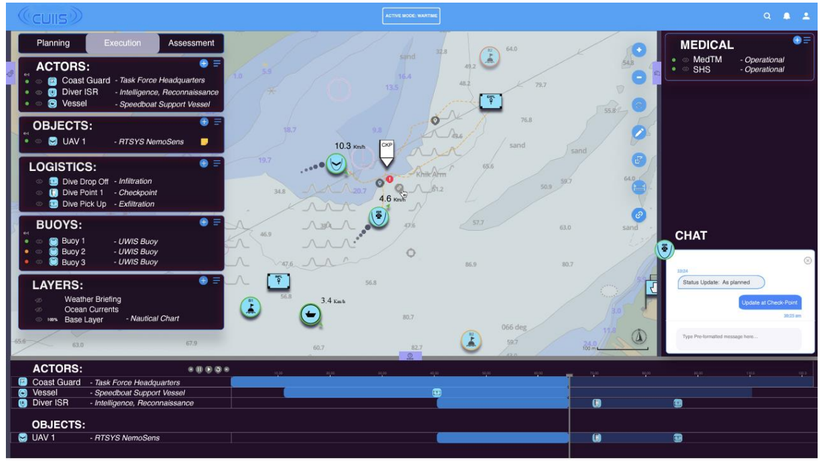
Advancing C3S Systems for Subsea Tactical Operations: Insights from the CUIIS Project

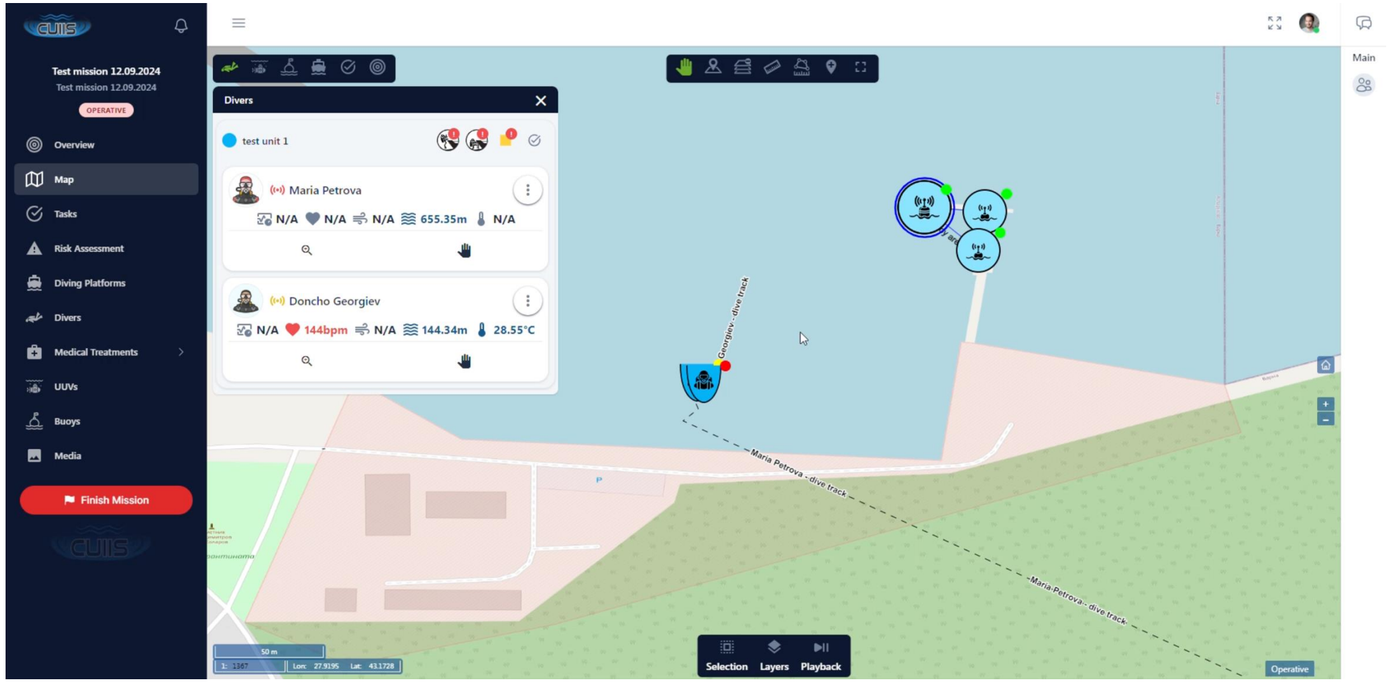
SDU

Syddansk Universitet

GWALSH@IMADA.SDU.DK

On the CUIIS Project





Syddansk Universitet

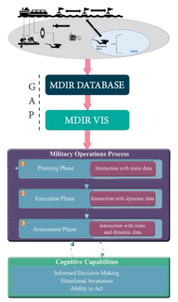
GWALSH@IMADA.SDU.DK

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 to align with models such as RAG, ALARMA and the Military Operations Process - to improve existing systems based on previous incidents.

2) In pursuing an enhanced the MDIR system, we conducted a comparative analysis of existing systems, namely NICS and JDRS, against a set of defined requirment criteria.

This evaluation helped to pinpoint opportunities for improvement in the MDIR system. This revealed shortcomings in terms of task-specific analysis, environmental context integration, and the emphasis on training and improvement in both the NICS and JDRS systems.

With this in mind, we developed a prototype for the MDIR system using Dash and Plotly, utilizing the DeepLife reBreather fatality database. This prototype is designed to address the identified deficiencies and align with the specified requirements for efficient data analysis, integration, and visualization of military diving incidents.

2 Methodology:

Criteria	Proposed MDIR System	NICS	JDRS
Data Collection and Integration	Moderate	Moderate	Moderate
Data Visualization	Excellent	Moderate	Moderate
User Interface and Experience	User-Friendly	Non-Intuitive	Moderate
Scalability and Adaptability	Moderate	Moderate	Limited
Data Accuracy and Validation	Moderate	Moderate	Moderate
Real-Time Data	N/A	N/A	N/A
Task-Specific Analysis	Moderate	Basic	Basic
Environmental Context Integration	Moderate	Basic	Basic
Training and Improvement Focus	Strong	Weak	Weak

Comparative Evaluation of MDIR Systems

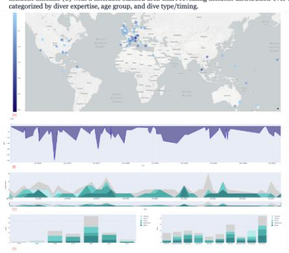
Database

DJRS System

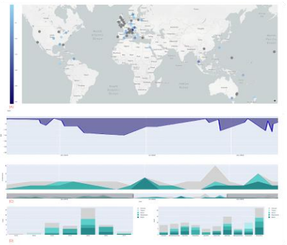
NICS System

3 MDIR Prototype:

The MDIR visualization tool features an interactive incident map (A) showcasing the geospatial 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 categorised by diver expertise, age group, and dive type/timing.



4 Prototype Interactions:



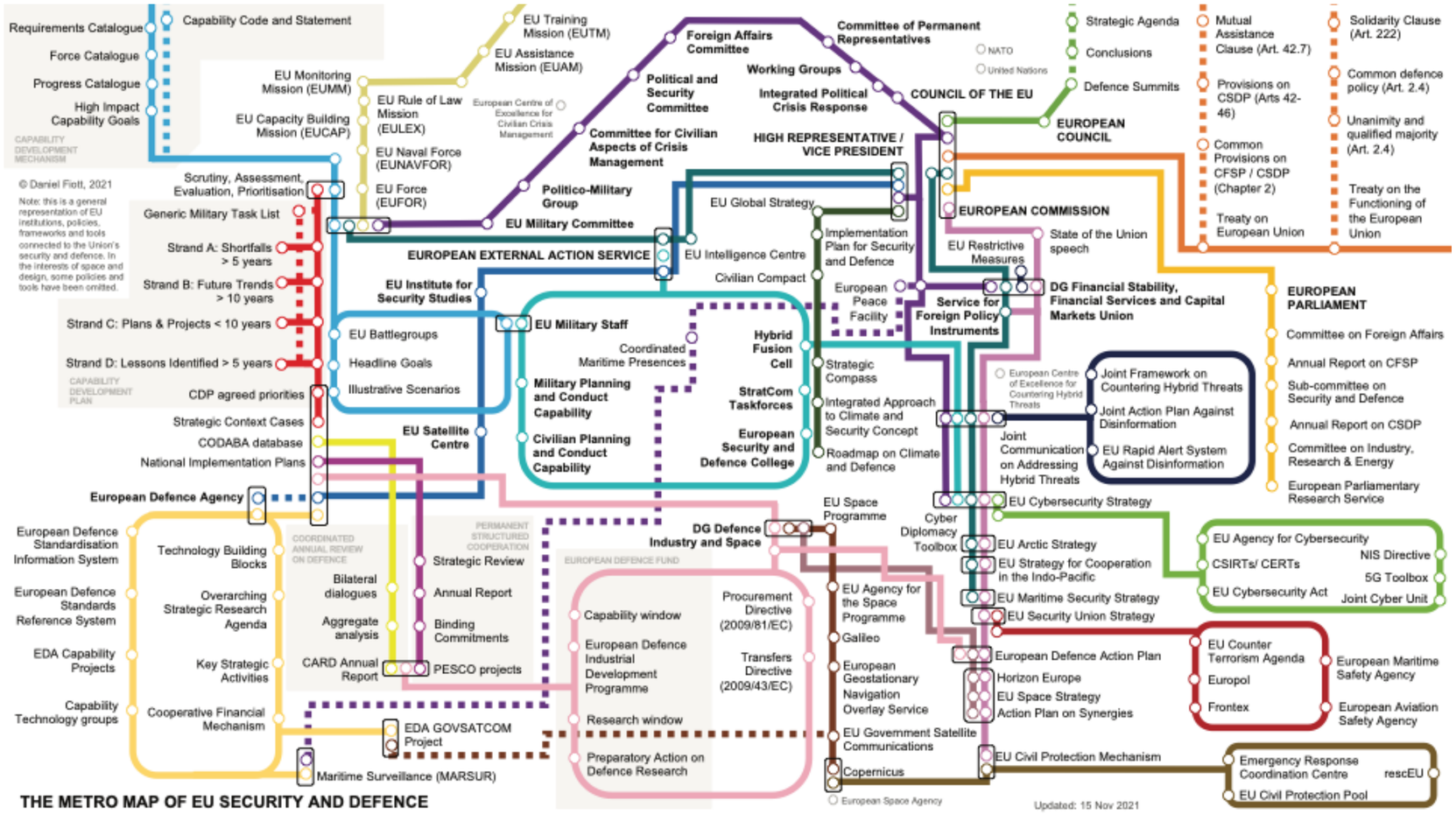
SDU CUIIS Co-funded by the European Union

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

Current work:

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



SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

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

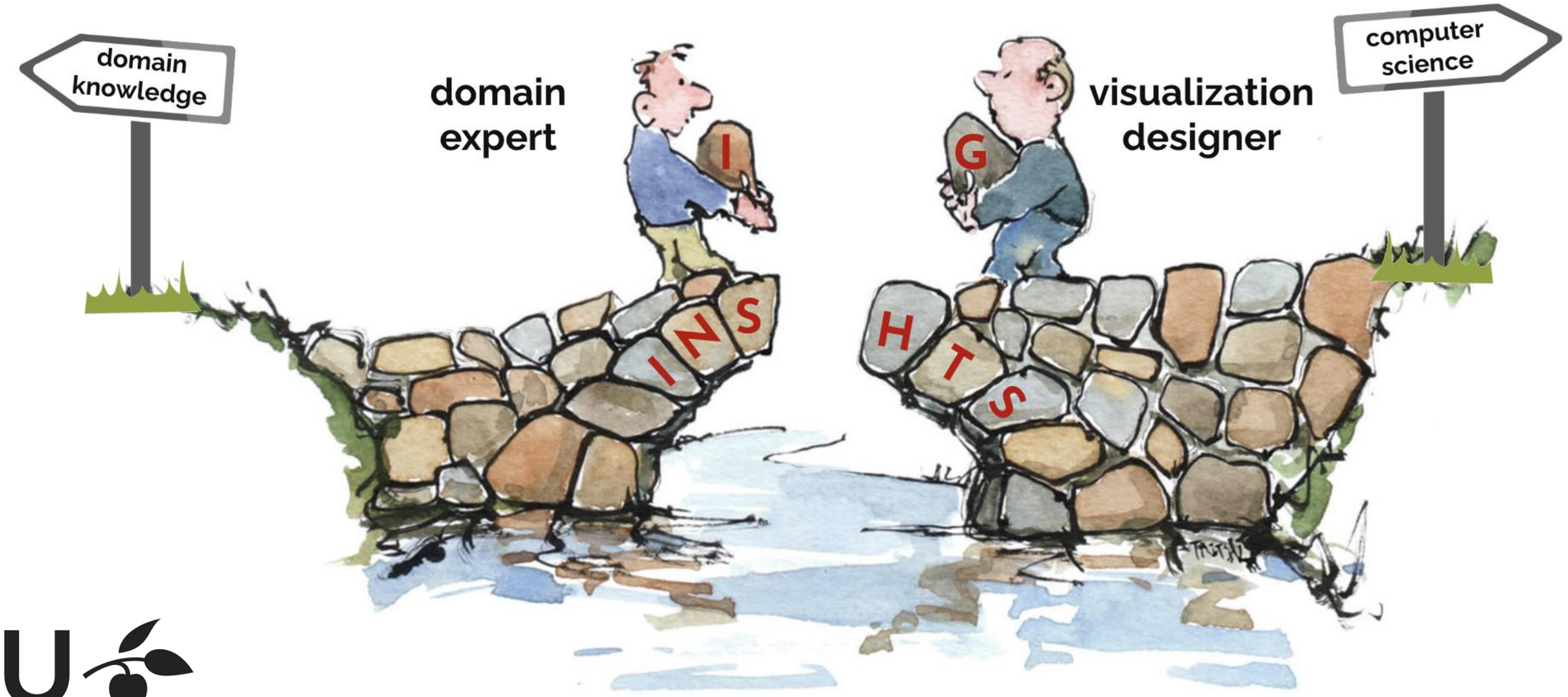
- Examples from Last Year; students got to connect with domain experts and experts in the field

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

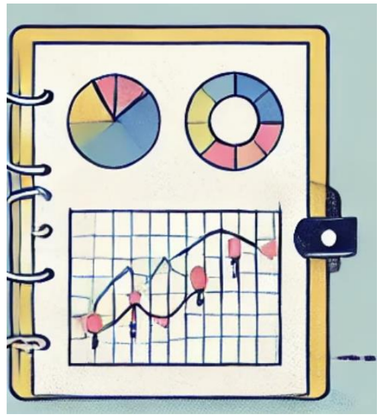
Building Bridges with Visualization

visualization enables data analytics

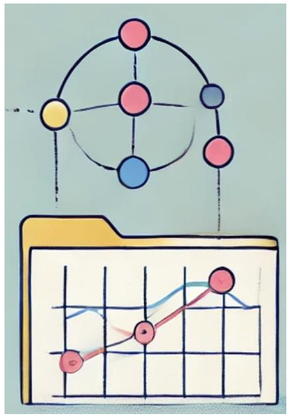


SDU
Syddansk Universitet

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

GWALSH@IMADA.SDU.DK

Structure + Agenda

Lectures

Classes 3-4 will be a lecture style format. These will cover:

- Intro to Dash/Ploty + Anaconda (D3)
- Charts in Ploty + 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.

GWALSH@IMADA.SDU.DK

Flask + plotly + React = Dash

Flask
backend server

Plotly
charting library

React
interactive components

Dash

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

These tools will help you make your Dashboard

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.

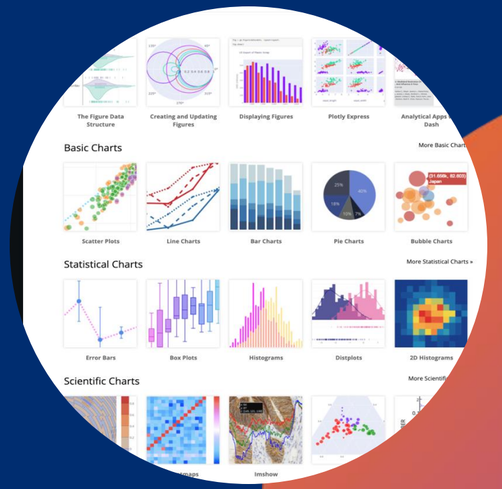
You use Plotly to define your charts, and then Dash allows you to embed those charts into a web application with added functionality (like user interaction, layout, and deployment).

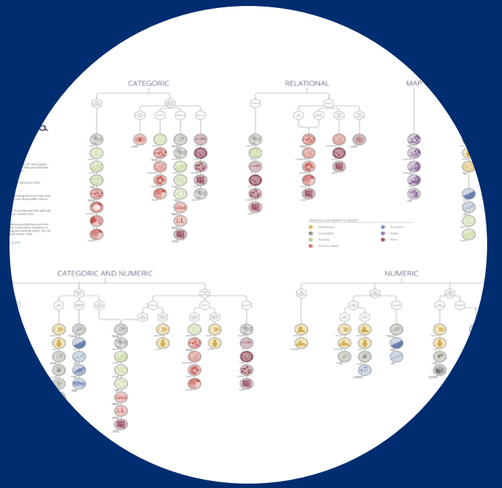
Dash = Web App Framework, and Plotly = Visualization Library used within Dash to create charts.

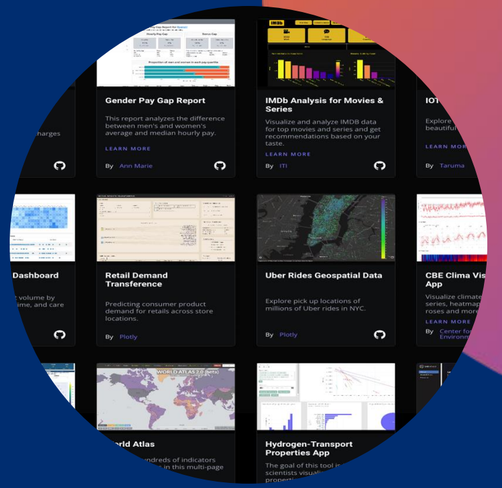
GWALSH@IMADA.SDU.DK

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





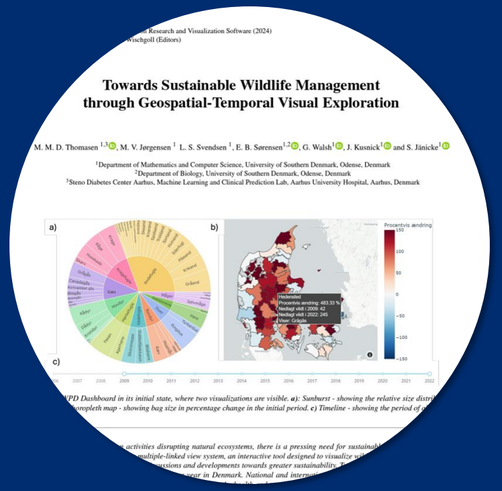


Syddansk Universitet

GWALSH@IMADA.SDU.DK

And Previous examples

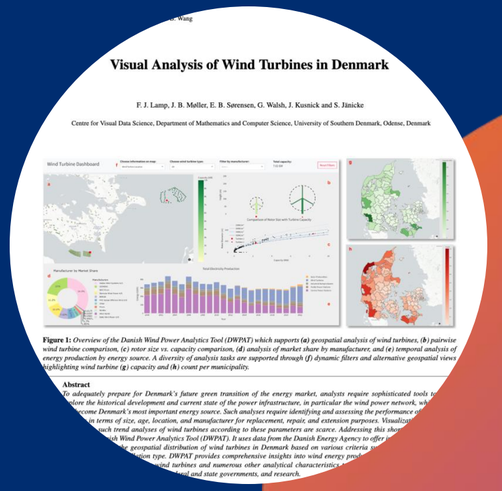
- Interactive Visual Exploration of Arctic Sea Ice Extent 1978-2023
- Towards Sustainable Wildlife Management through Geospatial-Temporal Visual Exploration
- Visualizing Property Assessments and Taxation: A Danish Case Study
- Visual Analysis of Wind Turbines in Denmark


Research and Visualization Software (2024)
Wischgoll (Editors)

Towards Sustainable Wildlife Management through Geospatial-Temporal Visual Exploration

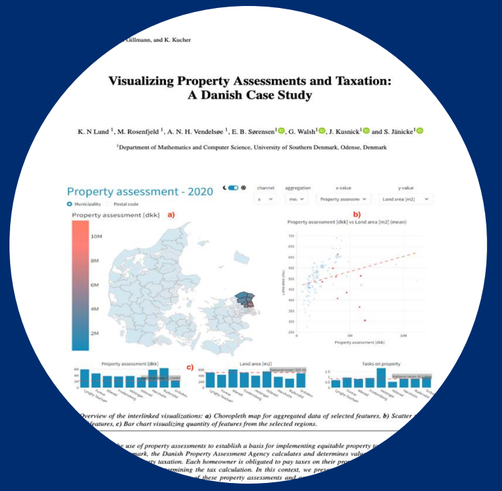
M. M. D. Thomasen [1,3], M. V. Jørgensen [1], L. S. Svendsen [1], E. B. Sørensen [1,2], G. Walsh [1], J. Kusnick [1] and S. Jänicke [1]

[1]Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark
[2]Department of Biology, University of Southern Denmark, Odense, Denmark
[3]Steno Diabetes Center Aarhus, Machine Learning and Clinical Prediction Lab, Aarhus University Hospital, Aarhus, Denmark


Visual Analysis of Wind Turbines in Denmark

F. J. Lamp, J. B. Møller, E. B. Sørensen, G. Walsh, J. Kusnick and S. Jänicke

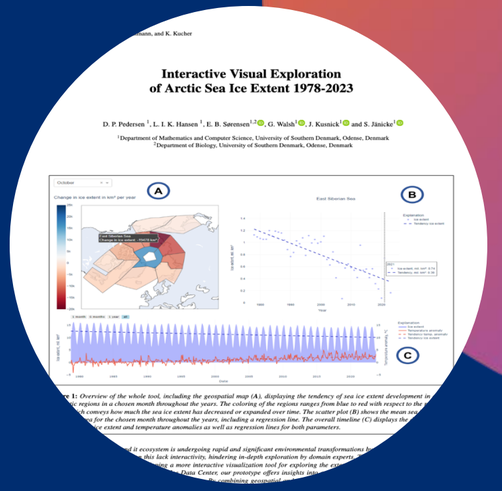
Centre for Visual Data Science, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark


Gillmann, and K. Kocher

Visualizing Property Assessments and Taxation: A Danish Case Study

K. N Lund [1], M. Rosenfjeld [1], A. N. H. Vendelbøe [1], E. B. Sørensen [1], G. Walsh [1], J. Kusnick [1] and S. Jänicke [1]

[1]Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark


Interactive Visual Exploration of Arctic Sea Ice Extent 1978-2023

D. P. Pedersen [1], L. I. K. Hansen [1], E. B. Sørensen [1,2], G. Walsh [1], J. Kusnick [1] and S. Jänicke [1]

[1]Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark
[2]Department of Biology, University of Southern Denmark, Odense, Denmark

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

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

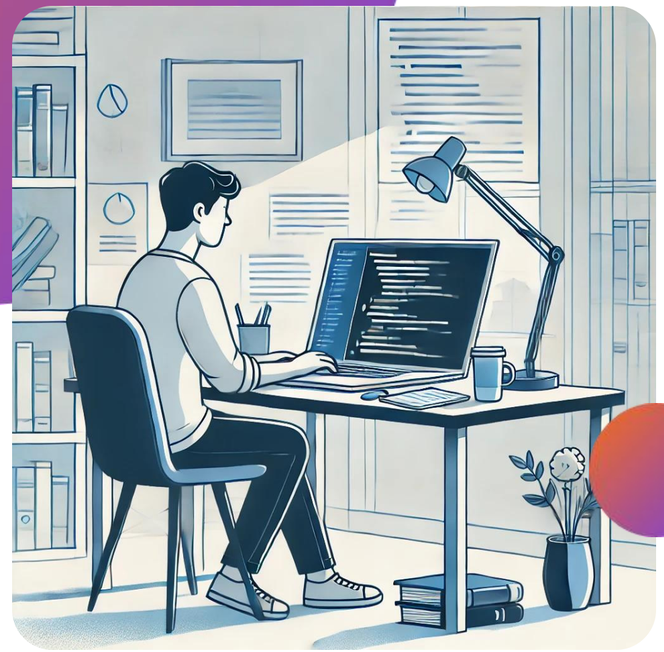
Check the DB includes the specific data points you need for your project. Also **consider what data cleaning or preprocessing might be required**. Many datasets need to be cleaned or transformed before they can be used in dashboards

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.

GWALSH@IMADA.SDU.DK

Ways to work



As a Group

Between 2-5 people
- Distribution of tasks

Individually

Allows more personal autonomy
- Increased workload/ less feedback

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

QUESTIONS

END OF PART 1

GWALSH@IMADA.SDU.DK

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)

GWALSH@IMADA.SDU.DK

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):

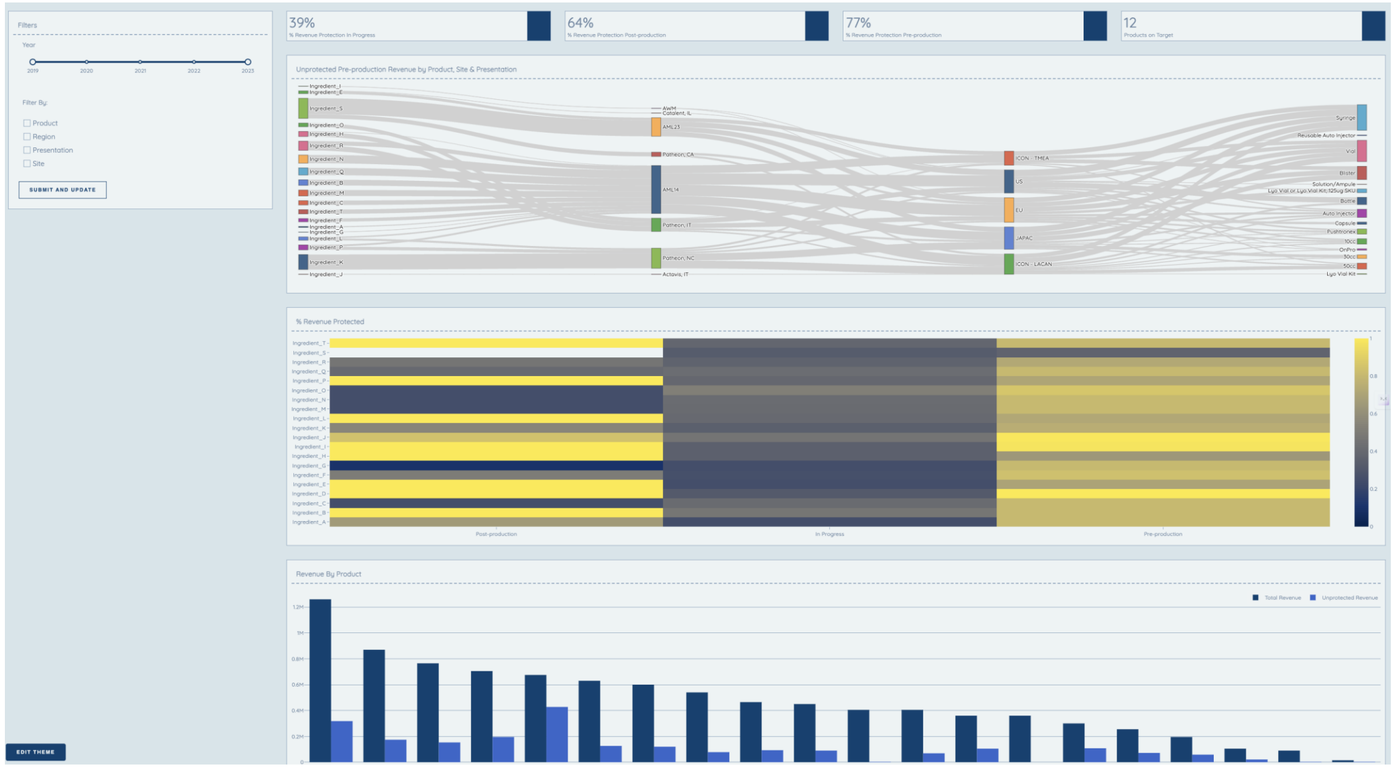
```
@app.callback()
def update_graph(...):
    return ...
```

Run the App:

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

SDU
Syddansk Universitet

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 detail**s. This makes your dashboard much more engaging than static charts.

GWALSH@IMADA.SDU.DK

Why Use Spyder and Anaconda for Plotly Dash Dashboards?

ANACONDA

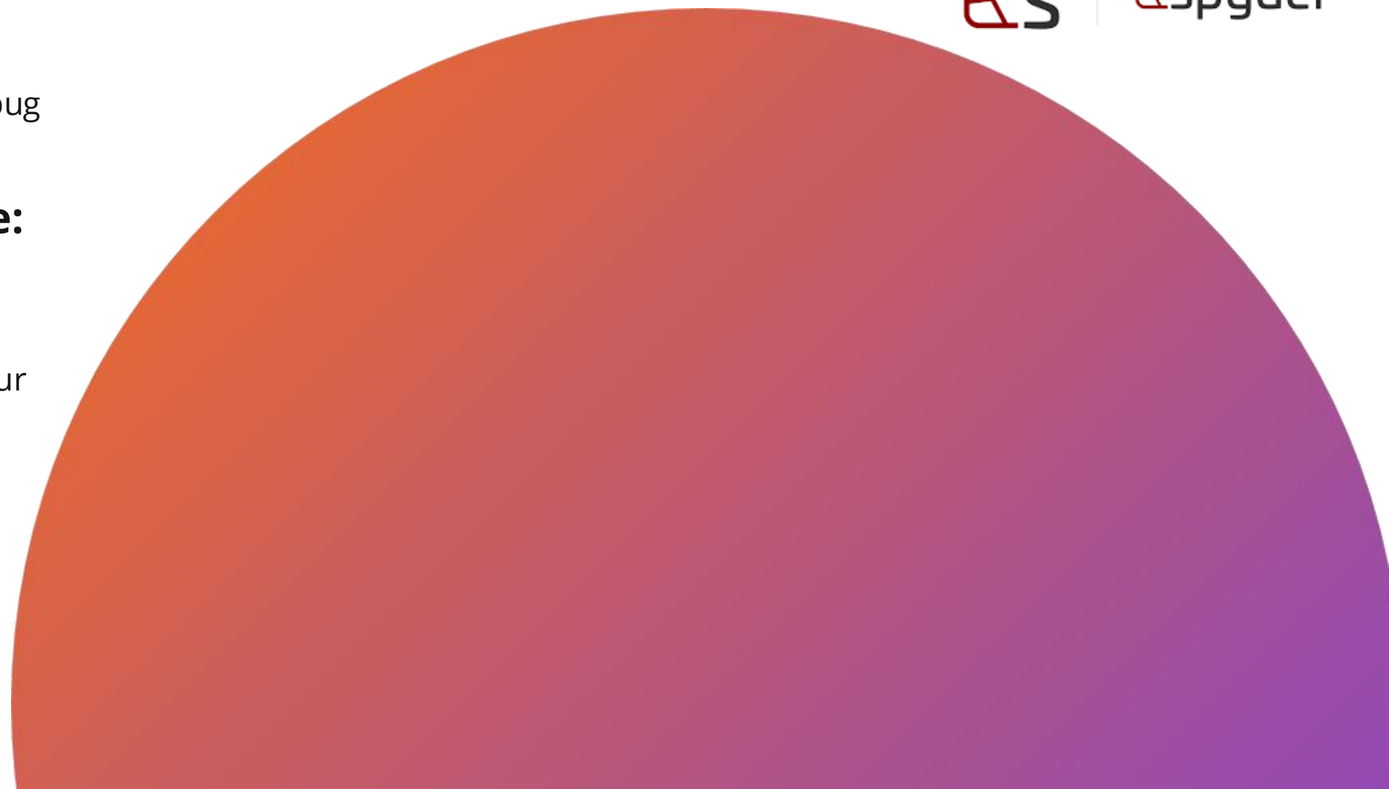
spyder

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.



SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

Spyder and Anaconda
Plotly Dash Installation Guide

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

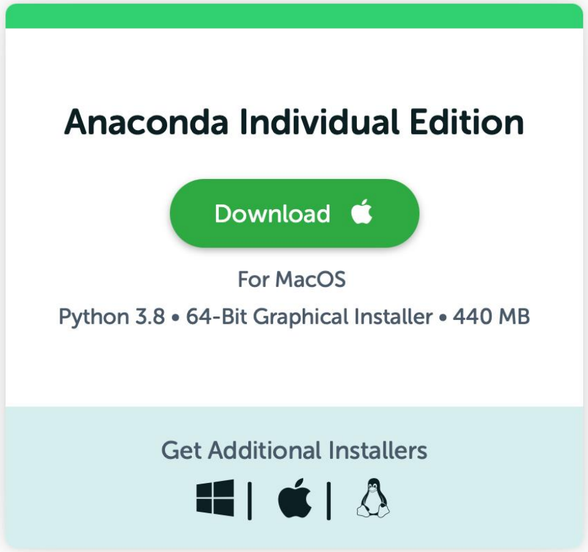
https://www.anaconda.com/products/individual

Download Installer

Individual Edition

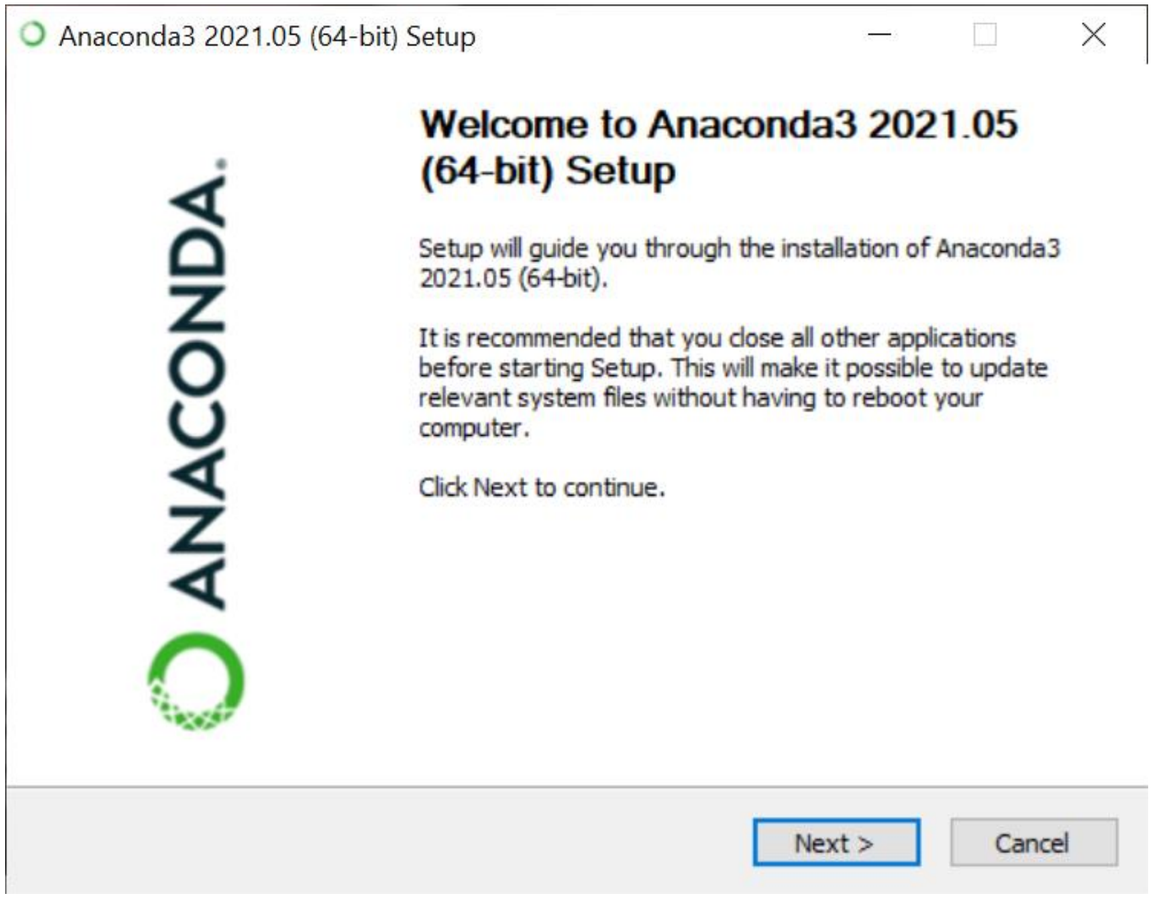
Your data science toolkit

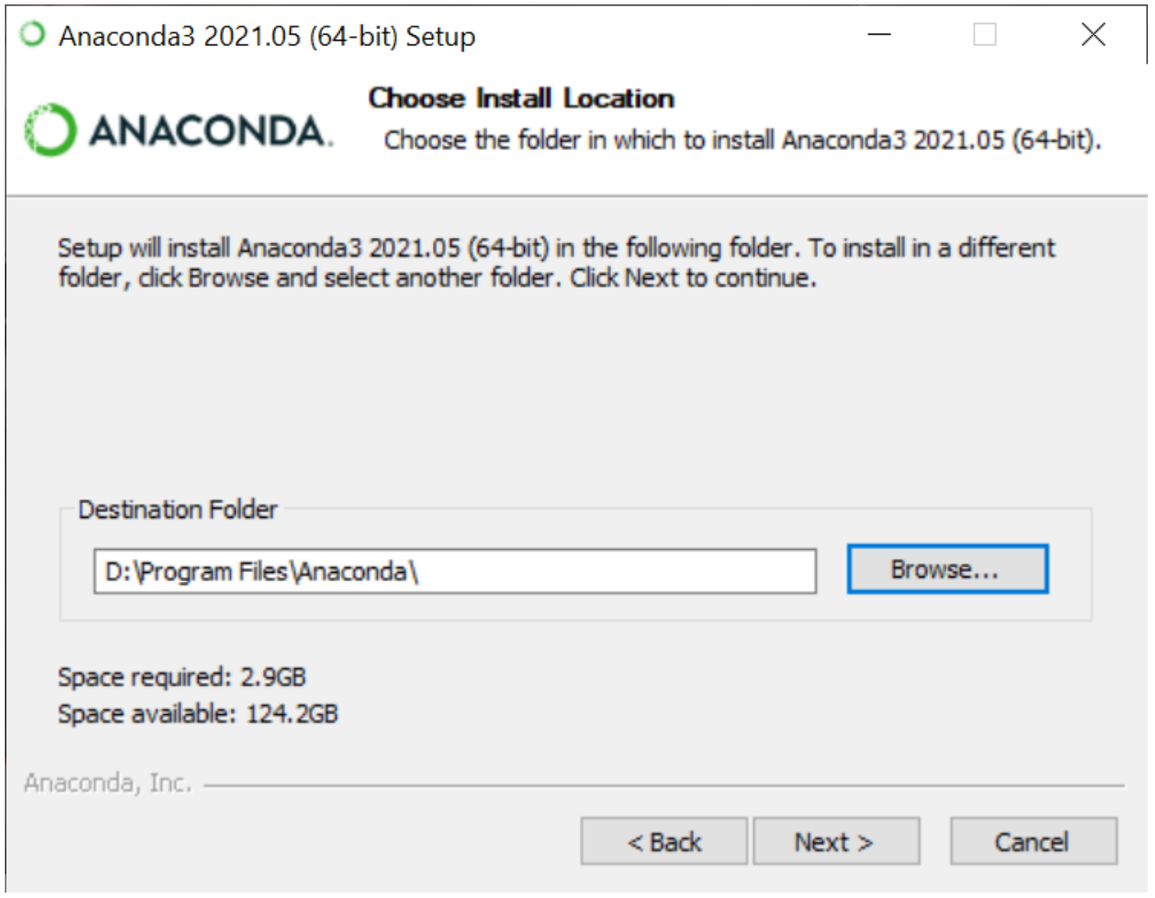
With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.



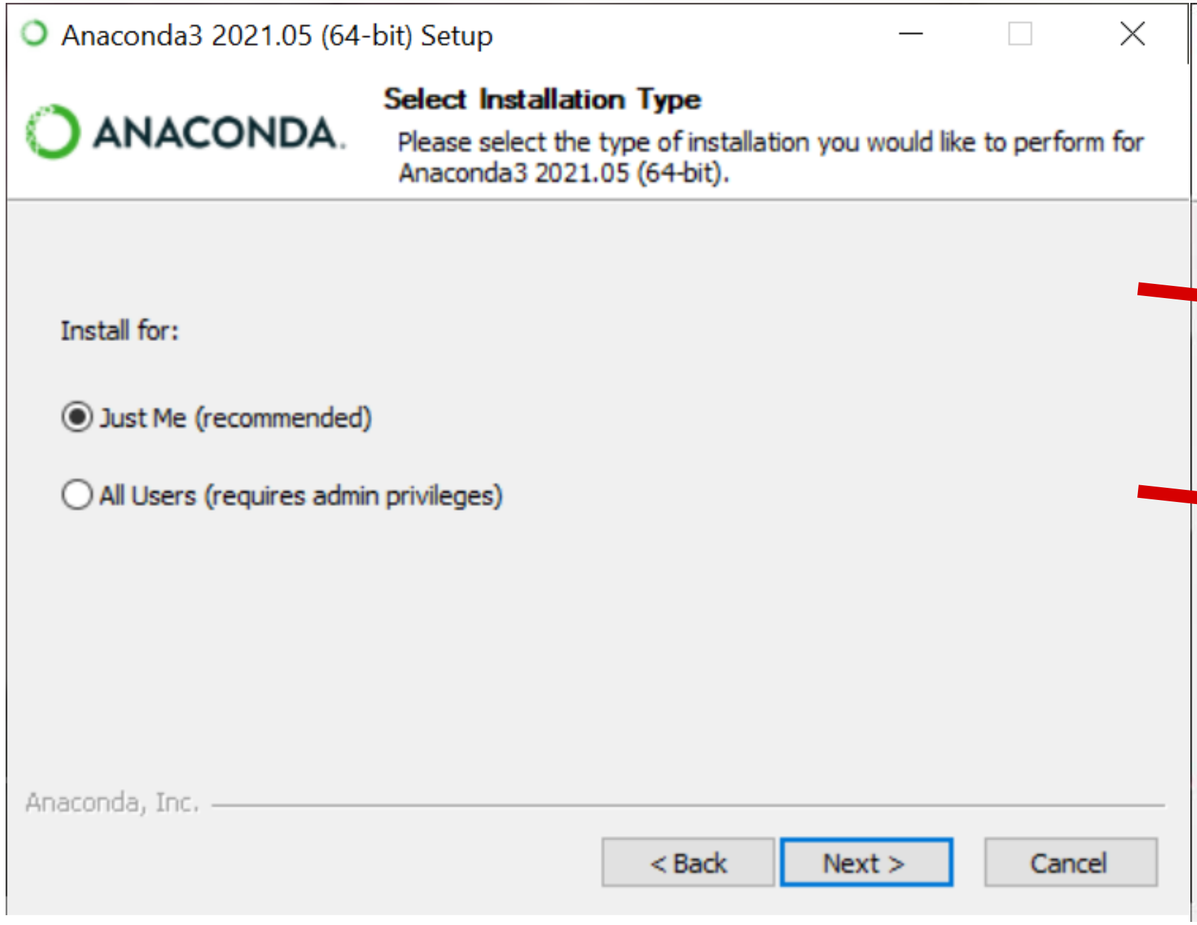
SDU
Syddansk Universitet

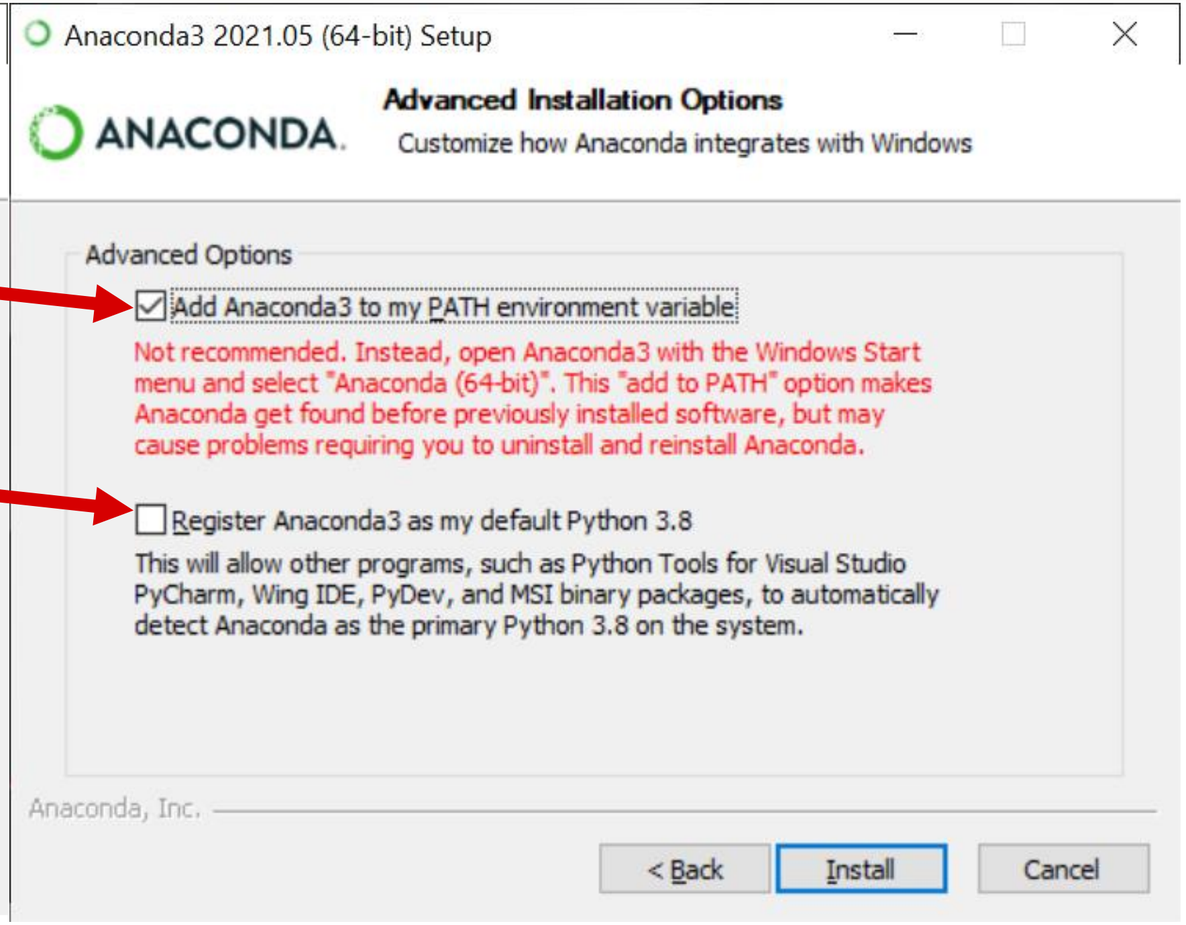
Anaconda Installation for Windows





Anaconda Installation for Windows

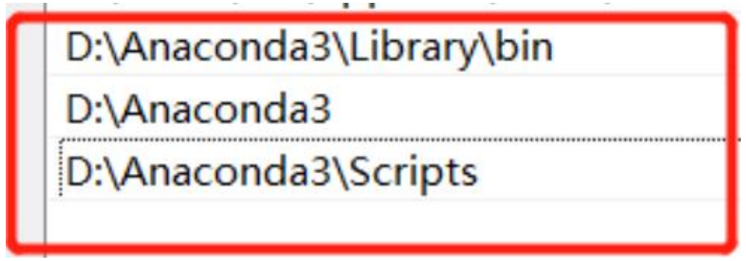




Anaconda Installation for Windows

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.

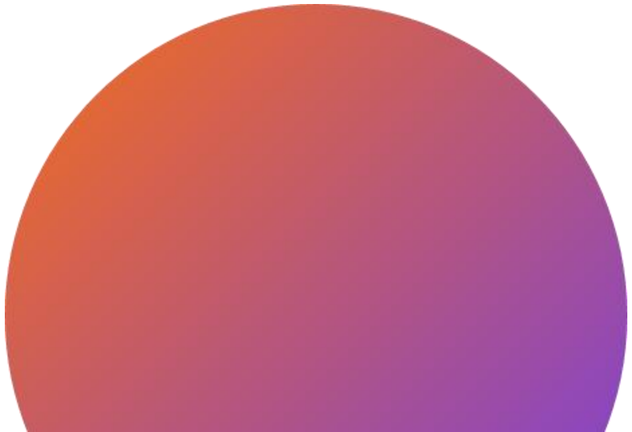
Type `conda --v` and it will return the version of conda

Mac:

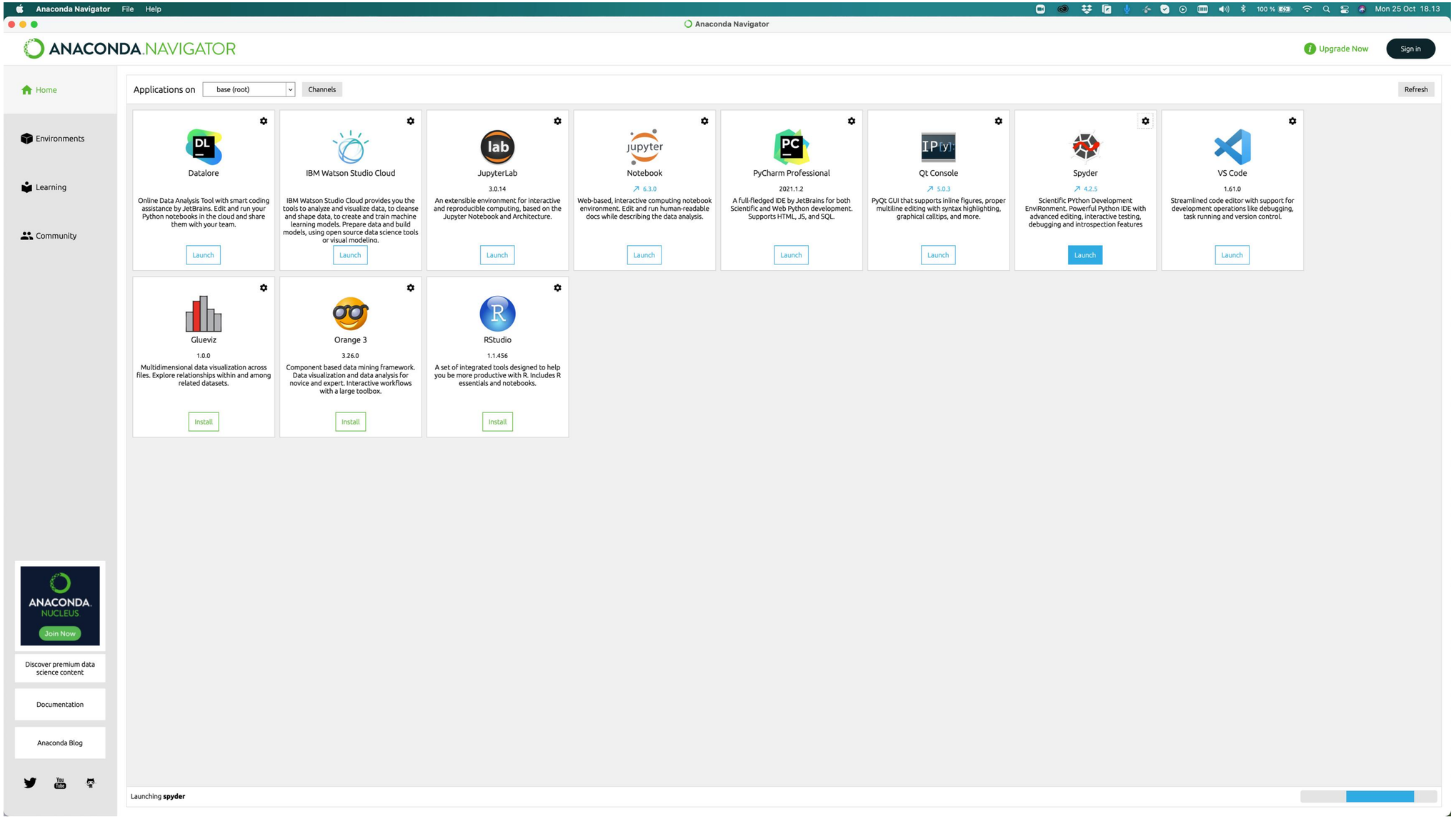
Cmd+Space to open Spotlight Search and type “Navigator” to open the program.

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



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!

Or run `python iris.py` in terminal app

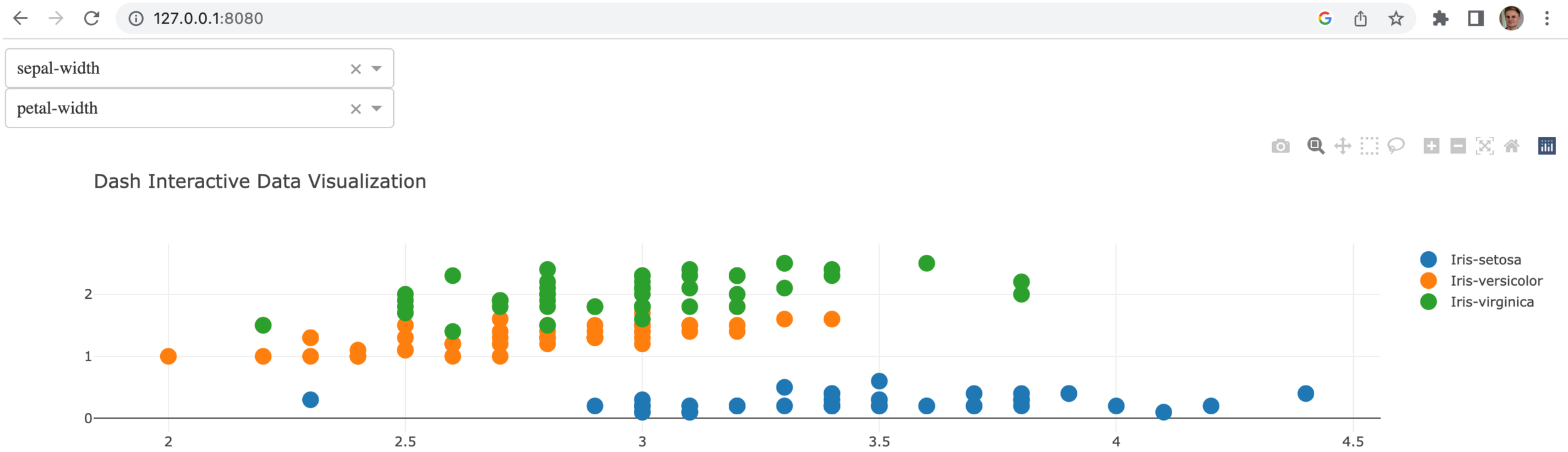

- Server should run on port on your system:

```
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
```

GWALSH@IMADA.SDU.DK

Iris.py

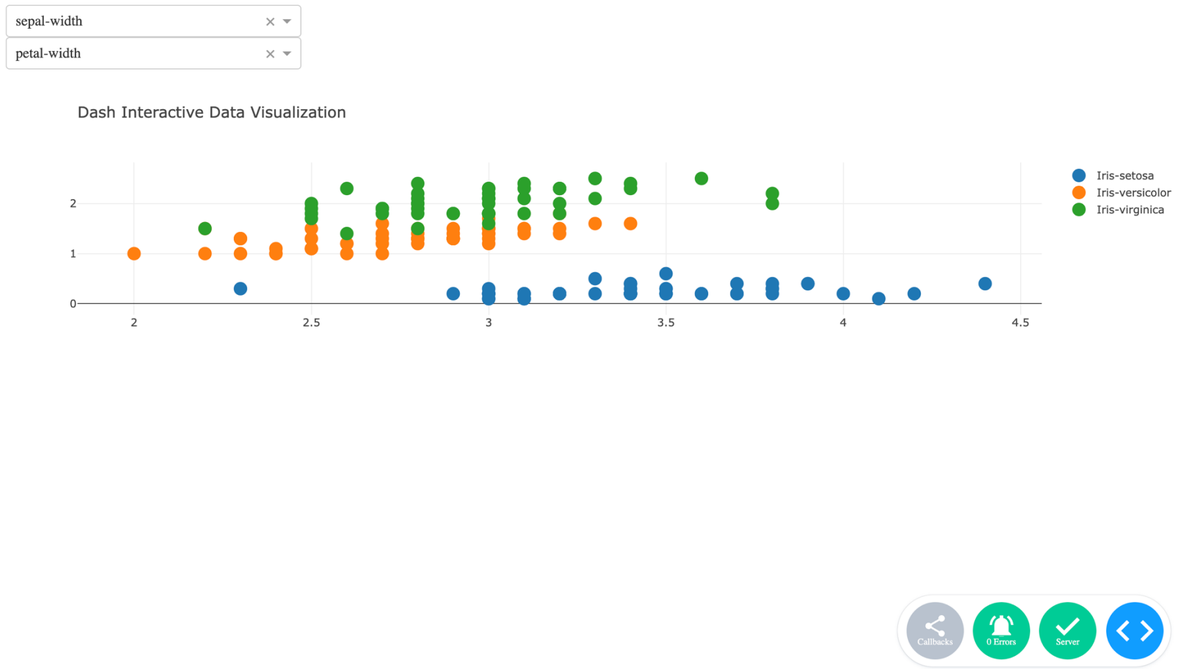


Syddansk Universitet

GWALSH@IMADA.SDU.DK

Turn on de-bugging mode

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



SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

Explanation of iris.py

1) Import Libaries
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'})
    ]
)
```

GWALSH@IMADA.SDU.DK

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**.

```python
@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)
```

SDU
Syddansk Universitet

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

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

HTML & CSS

- htmlTutorial.html
- htmlTutorial2.html

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

Volcanos

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

Or run `python volcanos.py` in terminal app


- Server should run on port on your system:

```
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 Libaries :

- 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.

```
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')
```

Initialize Dash App:

Read the data:

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

SDU
Syddansk Universitet

GWALSH@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
```

SDU
Syddansk Universitet

GWALSH@IMADA.SDU.DK

Volcanos.py Explanation

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.

```
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)})
                ])
])
```

Syddansk Universitet

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.

```python
@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-ma 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

END OF PART 2