Assignment 1

IND320

Sigvard Bratlie
https://github.com/sigvardbratlie/ind320
https://ind320-h63n5qj5uc26acyzlq3x39.streamlit.app/

AI-usage
Primarily for content refinement (e.g., spelling, syntax debugging) and generating explanatory text (e.g., for the log section).

Project Log

My work can be divided into four main phases:

Phase 1: Planning and Initial Setup

The project started with a thorough review of the assignment description to get a good understanding of the requirements. I then set up the development environment. A public GitHub repository was created to serve as version control and code-sharing hub.

Next, I wrote a `requirements.in` file listing only the relevant packages, and used **pip-tools** (`pip install pip-tools` + `pip-compile`) to resolve the dependencies. I chose this approach because I use the same conda environment across all my classes, and it contains a lot of extra libraries. Using `pip freeze > requirements.txt` would therefore have pulled in many unnecessary packages.

I also set up a `.gitignore` file to exclude scratch work, lecture notes, and other irrelevant files. At the same time, I linked the repository to my Streamlit Cloud account and deployed a minimal "hello world" app to verify that the CI/CD pipeline worked as intended.

Finally, I downloaded the data from Canvas and prepared a `.streamlit` directory with `config.toml` and `secrets.toml` for later use.

Phase 2: Data Exploration and Analysis in Jupyter Notebook

Once the setup was done, I focused on local data analysis in Jupyter Notebook. I started by loading the provided CSV file into a Pandas DataFrame. An important step here was converting the `time` column into datetime objects and setting it as the index, which is crucial for time-series analysis.

I then explored the dataset with `df.info()` and `df.describe()` to check datatypes, missing values, and overall structure.

For visualization, I first plotted each column individually to see the trends in each variable. The main challenge was combining all columns into a single plot, since they were on different scales (e.g., °C vs. mm). To handle this, I normalized the data so it could be compared meaningfully within one chart.

Phase 3: Building the Interactive Streamlit Application

With the data exploration complete, I moved on to building the Streamlit app. I structured it with a homepage and three subpages inside a `pages` directory, as required. To improve performance, I used Streamlit's caching decorator (`@st.cache_data`), which prevents the app from re-reading the CSV file every time a user interacts with it.

On the second page, I set up a data table and used `st.column_config.LineChartColumn` to add an inline line chart for the first month's data, giving a quick visual summary. The third page was focused on interactive visualization. I added a `st.selectbox` to let users choose between individual columns or all columns at once, and a `st.select_slider` to filter the dataset by date. This meant I had to implement filtering logic on the DataFrame before plotting.

As a small extra feature, I added a selector for data aggregation. Since the raw daily data can look messy when plotted directly, it made sense to allow users to aggregate over longer periods for clearer visualization.

Phase 4: Final Testing and Deployment

Throughout development, I tested components continuously. Jupyter Notebook was useful for checking logic and syntax before transferring it into the main app. In the end, I confirmed that the `requirements.txt` file was correct, and that the app deployed and ran smoothly on Streamlit Cloud.

Final project structure

```
├── .env
├── .gitignore
├── .streamlit
│   ├── config.toml
│   └── secrets.toml
├── 0_home.py
├── assignments
│   ├── assignment1.html
│   ├── assignment1.ipynb
│   └── assignment1.pdf
├── data
│   └── open-meteo-subset.csv
├── pages
```

```
│     ├── 01_Page_1️⃣.py
│     ├── 02_Page_2️⃣.py
│     └── 03_Page_3️⃣.py
├── requirements.in
└── requirements.txt
```

In [6]:
```python
import pandas as pd
import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import math
%matplotlib inline
```

Read the supplied CSV file using Pandas

In [57]:
```python
df = pd.read_csv("open-meteo-subset.csv")
df["time"] = pd.to_datetime(df["time"])
df = df.set_index("time")
```

Print its contents in a relevant way

In [58]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8760 entries, 2020-01-01 00:00:00 to 2020-12-30 23:00:00
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   temperature_2m (°C)     8760 non-null   float64
 1   precipitation (mm)      8760 non-null   float64
 2   wind_speed_10m (m/s)    8760 non-null   float64
 3   wind_gusts_10m (m/s)    8760 non-null   float64
 4   wind_direction_10m (°)  8760 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 410.6 KB
```

In [4]:
```python
df.head()
```

Out[4]:

	time	temperature_2m (°C)	precipitation (mm)	wind_speed_10m (m/s)	wind_gusts_10m (m/s)	w
0	2020-01-01T00:00	-2.2	0.1	9.6	21.3	
1	2020-01-01T01:00	-2.2	0.0	10.6	23.0	
2	2020-01-01T02:00	-2.3	0.0	11.0	23.5	
3	2020-01-01T03:00	-2.3	0.0	10.6	23.3	
4	2020-01-01T04:00	-2.7	0.0	10.6	22.8	

In [5]:
```
df.describe().T
```

Out[5]:

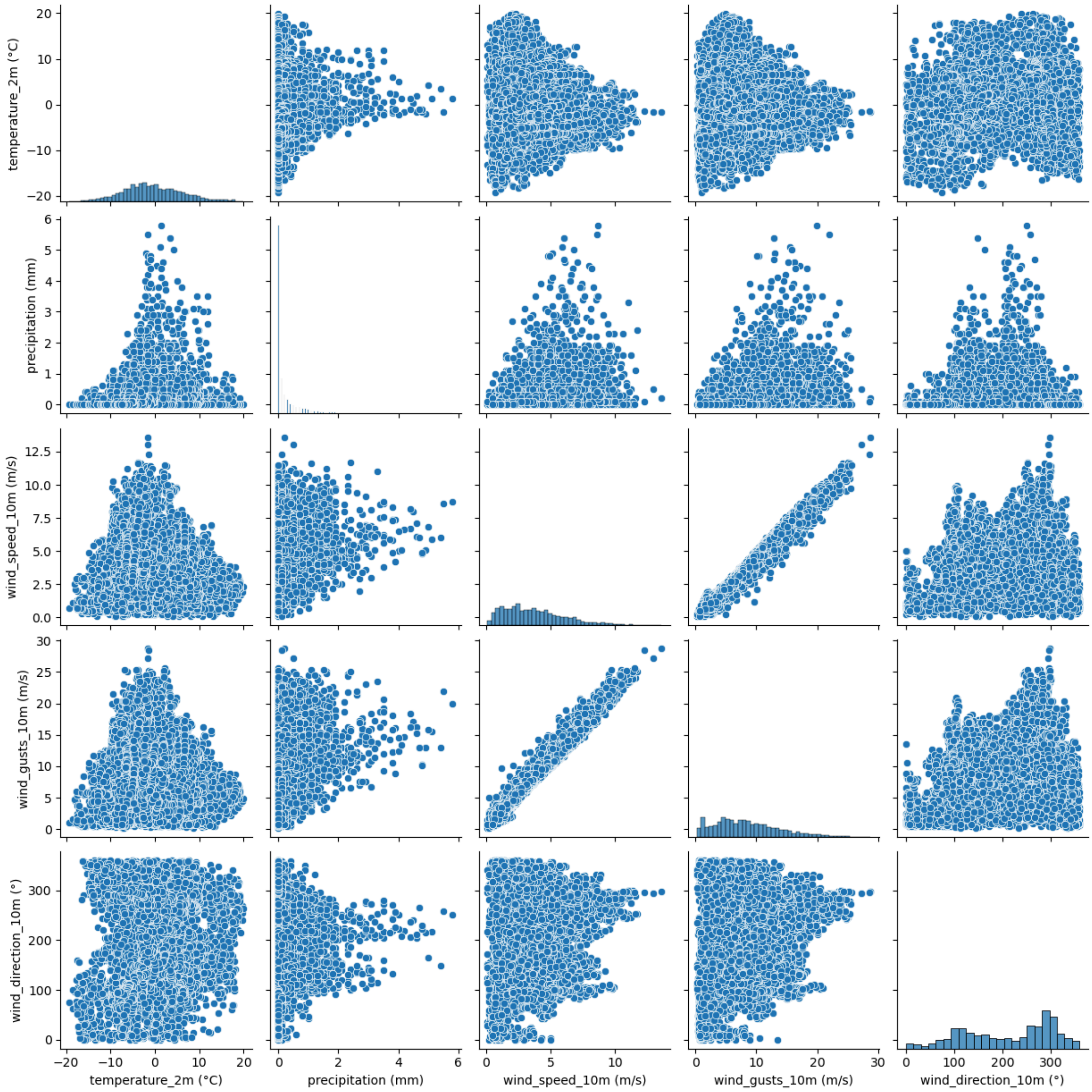
	count	mean	std	min	25%	50%	75%	ma
temperature_2m (°C)	8760.0	-0.394909	6.711903	-19.3	-4.9	-1.0	4.1	19
precipitation (mm)	8760.0	0.222854	0.493747	0.0	0.0	0.0	0.2	5
wind_speed_10m (m/s)	8760.0	3.661689	2.253210	0.1	1.8	3.3	5.1	13
wind_gusts_10m (m/s)	8760.0	8.300719	5.098909	0.2	4.5	7.7	11.5	28
wind_direction_10m (°)	8760.0	212.209589	91.371980	0.0	128.0	238.0	292.0	360

Plot Data

Plot each column separately

In [60]:
```
sns.pairplot(df)
```

Out[60]: <seaborn.axisgrid.PairGrid at 0x339d88f50>

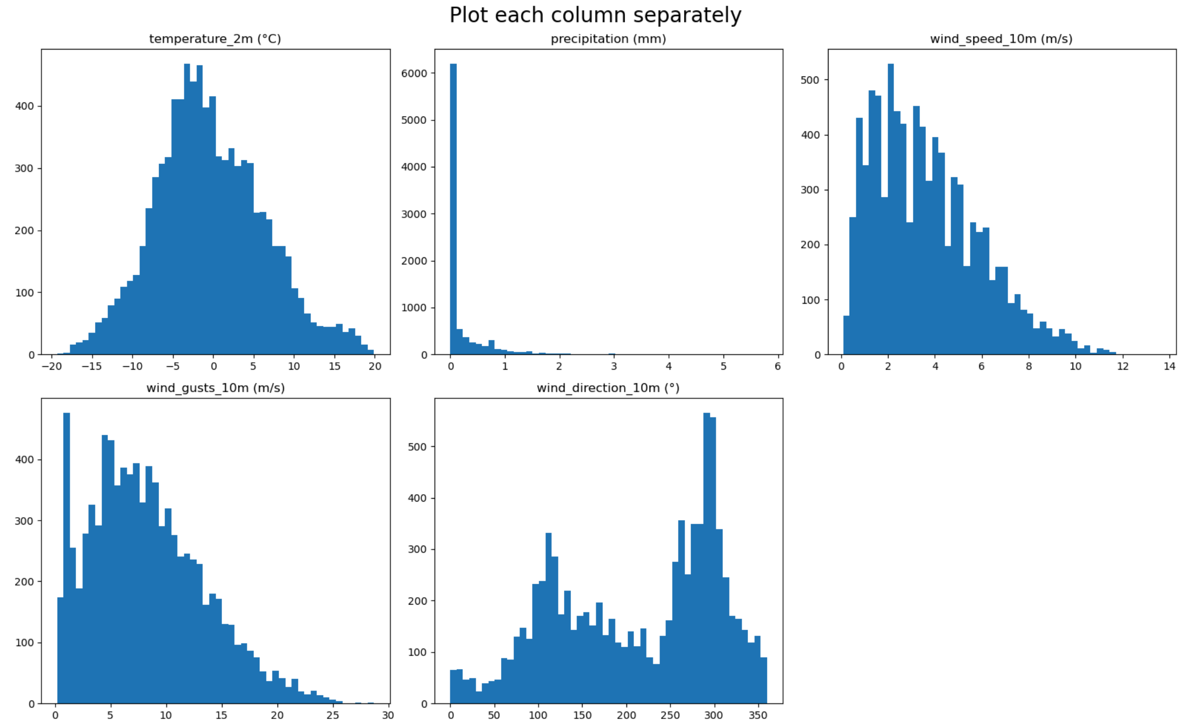


In [135…

```
# Setting a figure with each column in it's own axis
# and looping trough matrix to place data at it's correct locatioon
n_cols = 3
n_rows = math.ceil(len(df.columns) / n_cols)
fig, ax = plt.subplots(n_rows,n_cols, figsize = (16,10))

idx = 0
for i in range(n_rows):
    for j in range(n_cols):
        if idx<len(df.columns): #ensuring no index error
            ax[i,j].hist(df.iloc[:,idx],bins = 50)
            ax[i,j].set_title((df.columns[idx]))
            idx += 1
        else:
            ax[i,j].axis("off") #turning off axis if no data

plt.suptitle("Plot each column separately",fontsize = 20)
plt.tight_layout()
plt.show()
```



Plot all columns together

In [ ]:
```
dfs = (df-df.mean()) / df.std() #normalize the data
dfs.describe().T #Ensure normalization is performed correctly
```

Out[ ]:

	count	mean	std	min	25%	50%	
temperature_2m (°C)	8760.0	-3.893385e-17	1.0	-2.816651	-0.671209	-0.090152	0
precipitation (mm)	8760.0	9.003452e-17	1.0	-0.451352	-0.451352	-0.451352	-0
wind_speed_10m (m/s)	8760.0	1.070681e-16	1.0	-1.580718	-0.826239	-0.160522	0
wind_gusts_10m (m/s)	8760.0	2.027805e-17	1.0	-1.588716	-0.745399	-0.117813	0
wind_direction_10m (°)	8760.0	1.103126e-16	1.0	-2.322480	-0.921613	0.282257	0

In [136…
```
#Since it is not specified what type of plot to use, I will use both of t
fig,ax = plt.subplots(1,2,figsize = (16,6)) #create a figure with to subp
sns.lineplot(data = dfs.resample("ME").mean(),ax = ax[0]) #LINEPLOT: In o
ax[0].set_title("Lineplot aggregatet by month and scaled data")
sns.violinplot(dfs, ax=ax[1])
ax[1].set_title("Violinplot with scaled data")
plt.xticks(rotation = 45, ha= "right") #So that the column names do not c
plt.suptitle("Plot all columns together",fontsize = 20)
;
```

Out[136… ''

Plot all columns together