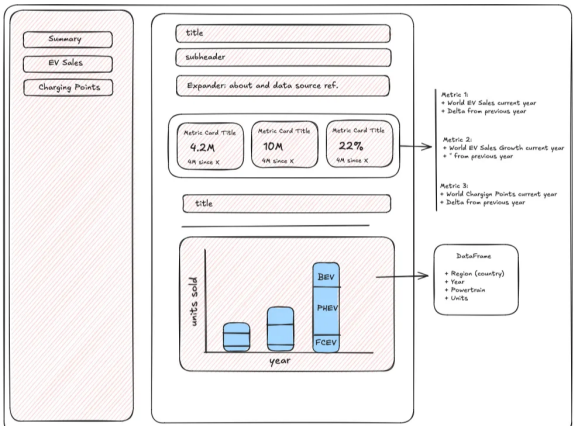
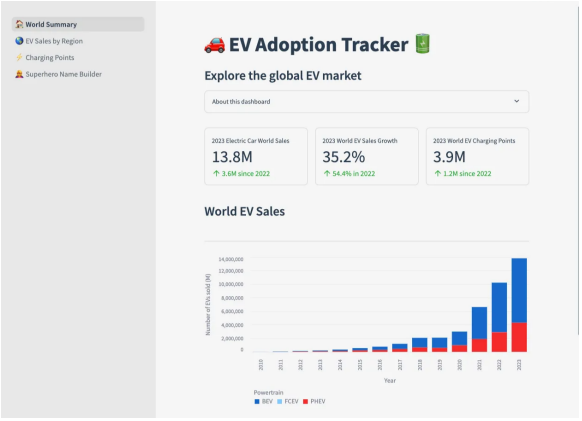
Streamlit Introduction Course

👋 Welcome to the Streamlit Introduction Course!

In this course you will learn how to create and deploy an interactive web application using Streamlit. We will learn the basics of how Streamlit components work, so that you'll be able to continue to explore the framework with ease.

🔨 What we'll be building





1. Set up our project

> To start our project, we will have to set-up a couple of things. Below is a checklist of the needed items. Let's go ahead and do that, it should take too long...

✔️ **Setup to-do list**

- [x] ~~Set-up a Github repository~~
- [ ] Create a new Python environment to work with on your project
- [ ] Build our first Streamlit App, a Superhero Name Creator 🦸

👋

Practice: Create a Github repo

- The first thing we need to do before we start coding is create a repo and clone it locally. Name it `streamlit-course` and make sure you select the Python `.gitignore` template when you create the repo. We will need it when we deploy the application. Go for it!

Python Environments

❓

What is a virtual environment?

- A virtual Python environment is an isolated environment where you can install Python packages and dependencies.
- It is basically a folder with code files that contain functions we will use in our code. For example, when we install Pandas, the code to Pandas functions (such as `pd.read_csv` are copied into our environment folder.

Why do we need an environment?

- To deploy our application (and make it available to the internet), we need to specify to the deploying server what packages our applications needs. When the application get's *built* (process that happens on deployment), those packages are installed in the application *container* using a file called `requirements.txt`

👏 Practice: Set up an environment

1. Create an environment
    a. You can create an environment using the Anaconda Distribution graphical UI or the terminal:

```
conda create -n my-environment
```

2. Activate the environment to use it

```
conda activate my-environment
```

3. Install Pandas and the Streamlit framework

```
conda install streamlit
```

❓ Create a requirements.txt file

- requirements.txt is a text file that contains a list of the packages you have installed in an environment. This file is used to generate a replica of the environment you created. You will need this file to deploy your application later.

1. Install pip in you conda environment

```
conda install pip
```

2. Create the requirements.txt using pip

```
pip freeze > requirements.txt
```

Streamlit

❓

What is it?

Streamlit is an open-source Python framework for data scientists and AI/ML engineers to deliver interactive data apps.

A few references...

https://streamlit.io/

https://docs.streamlit.io/develop/api-reference

https://streamlit.io/gallery

To run a streamlit app, create a new python file `app.py` and add the code below:

```
import streamlit as st

st.title("Hello World!")
```

To start the app, run the command below in your terminal. Make sure you are at the right directory in your terminal.

```
streamlit run app.py
```

A web application should open up on `http://localhost:8501`

Basic Streamlit Components

Copy the page below and paste it on your `app.py` file, run it to explore some basic components.

```
import streamlit as st
import pandas as pd

# 1. Title
st.title("Streamlit Components Demo")

# 2. Header
st.header("This is a Header")

# 3. Subheader
st.subheader("This is a Subheader")

# 4. Text
st.text("Streamlit makes it easy to create web apps for data science.

# 5. Markdown
st.markdown("**Markdown** lets you style text with *italics*, **bold*

# 6. Input Widgets
name = st.text_input("Enter your name:")
st.write(f"Hello, {name}!")

# 7. Slider
age = st.slider("Select your age", 0, 100, 25)
st.write(f"Your age is: {age}")

# 8. Button
if st.button("Click Me"):
    st.write("Button clicked!")

# 9. Checkbox
agree = st.checkbox("I agree to the terms and conditions")
if agree:
    st.write("Thank you for agreeing!")

# 10. Selectbox
option = st.selectbox("Choose an option:", ["Option 1", "Option 2", "
st.write(f"You selected: {option}")

# 11. Bar Chart
st.subheader("Simple Bar Chart")
```

```
data = pd.DataFrame({
    'Category': ['A', 'B', 'C', 'D'],
    'Values': [23, 45, 56, 78]
})
st.bar_chart(data, x="Category", y="Values")
```

👋

Practice: Superhero Name Creator 🦸🦸

Let's Create a Streamlit app that generates a unique superhero name based on the user's inputs (e.g., favorite color, animal, and a random number). Users can also choose a superpower from a dropdown menu and display their superhero profile!

Exercise Steps:

1. **Create Input Component**
   - Use `st.text_input` for their favorite color.
   - Use `st.text_input` for their favorite animal.
   - Use `st.number_input` for a random lucky number.
   - Use `st.selectbox` for selecting a superpower (e.g., flying, invisibility, super strength).
2. **Generate the Superhero Name**:
   - Combine inputs into a unique superhero name.
   - For example: `[Color] [Animal] of [Lucky Number]`
3. **Display the Superhero Profile**:
   - Display the superhero name.
   - Show the chosen superpower.
   - Optionally, display a cool "Superhero Motto."
4. **Bonus Challenge**:
   - Add a button to "Generate Random Superhero Catchphrase" and show a random quote from a predefined list.

Let’s set up the page configuration

Streamlit includes a setting function that allows us to easily set up basic page configuration. Copy and paste the code below at the top of your application file `.py`. Feel free to play around with the options to get a sense of the possible options.

```
# Set up the page
st.set_page_config(
    page_title="EV Adoption Tracker",
    layout="centered", # or wide
    page_icon="🚗", # choose your favorite icon
    initial_sidebar_state="collapsed" # or expanded
)
```

Deploy your app!

When your app is running locally, you will see a button that says “Deploy”. Click on it to make your application public in the internet. You will have to create a free account in Streamlit Cloud.

3. Time to work on a real project....

Project Brief

> We’ve just launched our freelance Data Analyst career, and our first client is a leading European car manufacturer. They’ve approached us to build a public dashboard that will track the yearly global adoption of electric vehicles (EVs). The goal is to help their Head of Sales in Europe determine a focused expansion strategy for EV infrastructure and sales efforts across key markets.
>
> During our initial meeting, the Head of Sales shared the primary data requirements and desired insights, which we’ll implement using the Global EV Data Explorer API from the International Energy Agency (IEA).

🗒️ Client Project Requirements

💡 Proposal

> A good approach to show what our solution proposal would look like is to create a wireframe (a low-detail sketch) of the application front-end layout. How many charts will be displayed? What data will be shown, and how? What filters will it have?

👋

Practice: Draw wireframes using Excalidraw

- Create a wireframe using Excalidraw (amazing tool to quickly sketch anything) on the layout of the proposed solution.
- Let's start with a sketch of the first chart, the requirements are:
  - **Country-Level EV Sales**:
    - Total EV sales by country, broken down historically.
    - Breakdown by powertrain type (Battery Electric Vehicles, Fuel Cell Electric Vehicles, Plug-in Hybrid Electric Vehicles).

> A clear sketch will not only help convince our client of our solution, but it will also help us organise our work and know what to do - like what DataFrames will we need and what functions we need to code to create such DataFrames. before doing that,

👋

Practice continued...

- Now let's do the rest of the charts, add the data tables required as well.

Explore the EV API

👋 Practice!

Now it's time to get explore the datasource provided by the client. Retrieve the data and to a quick exploration to understand the table structure. Then write a function called `load_data()` to retrieve the data in real time in our application.

1. Create a notebook called `utils-dev.ipynb` and explore this API on Electric Vehicle sales data. We will use this notebook to develop all needed functions for our application.
    - EV Sales data url:
        - https://api.iea.org/evs?parameters=EV sales&category=Historical&mode=Cars&csv=true
    - EV Charging Points url:
        - https://api.iea.org/evs?parameters=EV charging points&category=Historical&mode=EV&csv=true
2. Classify the columns as categorical or numerical. List the classes for each categorical column.
3. Write a function to `load_data()` , this function should return to DataFrames:
    a. `ev_sales_df` from this url:
        i. https://api.iea.org/evs?parameters=EV sales&category=Historical&mode=Cars&csv=true
    b. `ev_charging_points_df` from this url:
        i. https://api.iea.org/evs?parameters=EV charging points&category=Historical&mode=EV&csv=true

Continue with the Application...

📖 Adding pages and a sidebar? Easy!

- Streamlit has made it very easy to add pages (and a sidebar!) to our application.
- Create a `pages/` directory at the root of your project, then add pages by creating `.py` files in that directory.

👋 Practice: Add two new pages to your application

Create two new pages:

1. EV Sales by region
2. Charging Points

- You can also add emoticons to the file name, that way they will show up in your sidebar as well!
- Sort the pages however you want by adding a “1_” in front of the file name: `1_filename.py`

👋 Continue with the app!

Now we have all the tools we need to complete our app. Use the sketches below as a guide to build the rest of the application. Remember to use the Streamlit API reference to use the components features properly. Good Luck!

A couple more things...

🐣 Changing the application’s UI theme

`.streamlit/config.toml`

```
[theme]
primaryColor="#2E4053"           # Dark slate gray for primary elemen
backgroundColor="#F5F5F5"        # Light gray for main background
secondaryBackgroundColor="#E8E8E8"  # Slightly darker gray for second
textColor="#2C3E50"             # Dark gray (almost black) for text
font="sans serif"
```

👏 Look up this decorator `@st.cache_data` and use it on your `load_data()` function. What happens?