Streamlit App Development Tutorial

1. Introduction to Streamlit

- Overview of Streamlit and its features.
- Use cases for building interactive data apps.

This is the official website for learning more about Streamlit app.

Recommended Project Structure:

Please ensure that the Streamlit app is set up as a standalone project. It should not be placed as a subfolder within another directory.

Make sure the directory names do not contain spaces. For example, use `nasdaq100_dashboard` instead of `nasdaq100 dashboard`.

Ensure that the file paths for data and images are absolute. For example, for `daq.png`, the path should be like this:

```python
import os
import streamlit as st

# Get the current directory of the script
current_dir = os.path.dirname(__file__)

# Construct the full path to the image file
image_path = os.path.join(current_dir, 'daq.png')

# Display the image
try:
    st.image(image_path)
except FileNotFoundError:
    st.error(f"Image not found at path: {image_path}")
```

```
project_root/
   .streamlit/
      config.toml
   virtual_env/
      activate_this.py
   .gitignore
   README.md
   app.py
   cleaned_data.csv
   company_assets.png
   create_a_virtual_environment.bat
   daq.png
   nasdaq_100_metrics_ratios.csv
```

```
nasdaq_100y.ipynb
requirements.txt
```

.streamlit/

- **config.toml**: Configuration file for Streamlit, an open-source app framework used to create and share data apps.

virtual_env/

- **activate_this.py**: Script to activate the virtual environment, ensuring the project uses the correct dependencies.

.gitignore

- Specifies files and directories that Git should ignore, preventing them from being tracked in version control.

README.md

- Contains information about the project, including an introduction, installation instructions, and usage guidelines.

app.py

- Main Python script for running the application, including helper functions, sidebar, and main panel setup.

cleaned_data.csv

- CSV file with cleaned NASDAQ-100 data, ready for analysis or use within the application.

company_assets.png

- Image file, possibly used for visual representation within the project.

create_a_virtual_environment.bat

- Batch script to create a virtual environment on Windows systems.

daq.png

- Another image file, likely used for visual representation within the project.

nasdaq_100_metrics_ratios.csv

- CSV file containing raw financial metrics and ratios for NASDAQ-100 companies.

nasdaq_100y.ipynb

- Jupyter notebook for exploratory data analysis on historical NASDAQ data.

requirements.txt

- Lists all dependencies needed to run the project, ensuring the correct packages are installed.

How to create a virtual environment in VS Code in window? Please follow along `create_a_virtual_environment.txt` file in the folder.

2. Page Configuration

Set up the app's basic configuration using `st.set_page_config()`.

Parameters: - `page_title`: Title of the app. - `page_icon`: Icon displayed in the browser tab. - `layout`: Layout configuration (wide or centered). - `initial_sidebar_state`: Sidebar state (expanded or collapsed).

```
st.set_page_config(
    page_title="NASDAQ-100 Index",
    page_icon=":line_chart:",
    layout="wide",
    initial_sidebar_state="expanded"
)
```

3. Custom Styling with CSS

Define CSS styles using `st.markdown()` for customizing UI components.

Explanation: - CSS (Cascading Style Sheets): A language used to describe the presentation of a document written in HTML or XML. It allows you to apply styles to web pages, including layout, colors, and fonts. **- `st.markdown()`**: A function in Streamlit that allows you to write Markdown and HTML code. By using this function, you can inject custom CSS styles into your Streamlit app.

Example: Create a styled box for displaying KPIs.

```
st.markdown("""
    <style>
    .box { ... }
    .metric-label { ... }
    .metric-value { ... }
    .metric-delta { ... }
    </style>
""", unsafe_allow_html=True)
```

- **.box**: A CSS class that you can define to style a container or box element. You can add properties like background color, padding, border, etc.
- **.metric-label**: A CSS class for styling the label of a metric, such as font size, color, and weight.
- **.metric-value**: A CSS class for styling the value of a metric, allowing you to customize its appearance.
- **.metric-delta**: A CSS class for styling the delta or change in a metric, which can be useful for indicating increases or decreases.
- **unsafe_allow_html=True**: This parameter allows the use of raw HTML in the `st.markdown()` function. It should be used with caution to avoid security risks like XSS (Cross-Site Scripting) attacks.

4. Building the Sidebar

Use the sidebar for user inputs and filters.

Example: Display an image, add filters for sectors, subsectors, and companies, and upload a file.

```
with st.sidebar:
    st.image(image_path)
    st.write("Filter the data by:")
    uploaded_file = st.file_uploader("Upload a file", type=["csv", "xlsx"])
```

Include user guides and instructions in an expandable section using `st.expander()` and `st.write("""....""")`. I prefer `st.markdown()` and for equations use `r` with the expression enclosed by a dollar sign: `st.markdown(r'''$E = mc^2$''')`. For python value formatting, `st.write(f"""The overall average return on investment when invetsed in NASDAQ-100 over past 5 years is {cagr(df)} %.""")`.

5. Data Filtering

Implement filtering based on user selections (sectors, subsectors, and companies). Use `st.multiselect()` for multiple selections and `st.slider()` for year range. - `st.selectbox()`: Use for single selection from a list of options. - `st.radio()`: Use for single selection from a list of radio buttons. - `st.checkbox()`: Use for boolean selection (True/False). - `st.select_slider()`: Use for selecting a single value or range from a slider with custom options.

Example:

```
selected_sector = st.multiselect('Select sector', sectors)
selected_subsector = st.multiselect('Select subsector', filtered_subsectors)
selected_company = st.multiselect('Select company', filtered_companies)
years = st.slider('Select Year Range', 2017, 2023, (2017, 2023))
selected_option = st.selectbox('Select an option', options)
selected_radio = st.radio('Select one', options)
```

```
selected_checkbox = st.checkbox('Check this box')
selected_slider = st.select_slider('Select a value', options=range(10))
```

6. Displaying Key Performance Indicators (KPIs)

Calculate and display top-performing KPIs in columns.

Use `st.columns()` to create a responsive layout. You can specify the width of the columns, and rows are created automatically as you add more sections.

Example Code

```
import streamlit as st

# Row 1: Equal width columns
col1, col2, col3 = st.columns(3)
with col1:
    st.markdown(f"<div class='box'>KPI 1</div>", unsafe_allow_html=True)
with col2:
    st.markdown(f"<div class='box'>KPI 2</div>", unsafe_allow_html=True)
with col3:
    st.markdown(f"<div class='box'>KPI 3</div>", unsafe_allow_html=True)

# Row 2: Custom width columns
col1, col2, col3 = st.columns([1, 2, 1])
with col1:
    st.markdown(f"<div class='box'>KPI 4</div>", unsafe_allow_html=True)
with col2:
    st.markdown(f"<div class='box'>KPI 5</div>", unsafe_allow_html=True)
with col3:
    st.markdown(f"<div class='box'>KPI 6</div>", unsafe_allow_html=True)
```

Row 1: Equal width columns

- `st.columns(3)` creates three columns of equal width.
- Each column displays a KPI using `st.markdown()`.

Row 2: Custom width columns

- `st.columns([1, 2, 1])` creates three columns with custom widths. The middle column is twice as wide as the other two.
- Each column displays a KPI using `st.markdown()`.

7. Integrate Various Visualization Libraries

Integrate various visualization libraries such as Altair, Plotly, Matplotlib and Seaborn.

st.altair_chart(make_donut(...), use_container_width = True) st.plotly_chart(make_donut(...), use_container_width = True)

With `use_container_width=True`: The plot will automatically adjust its width to fit the container, making it more flexible and responsive. For seaborn and matplotlib plots, thats not necessary.

st.pyplot(fig)

8. File Upload and Progress Indicators

Use `st.file_uploader()` to allow users to upload files.

Implement progress buttons with `st.button()` to handle long-running tasks.

Example Code

File Upload

```python
import streamlit as st

# Allow users to upload files
uploaded_file = st.file_uploader("Upload a file", type=["csv", "xlsx"])

if uploaded_file is not None:
    # Process the uploaded file
    st.write("File uploaded successfully!")
```

File upload from camera

```python
import streamlit as st

# Allow users to upload an image from their camera
uploaded_image = st.camera_input("Take a picture")

if uploaded_image is not None:
    # Process the uploaded image
    st.image(uploaded_image)
    st.write("Image uploaded successfully!")
```

Progress bar

```python
import streamlit as st
import time

# Button to start a long-running task
if st.button('Run Analysis'):
    progress_bar = st.progress(0)
```

```python
for i in range(100):
    # Simulate a long-running task
    time.sleep(0.1)
    progress_bar.progress(i + 1)

st.write("Analysis complete!")
```

9. Deplot on cloud

Use `streamlit run app.py` where `app.py` contains your app to run it locally.

You can host your app for free on the Streamlit app platform. Please make an account here and start deploying. You can link your Streamlit account to your GitHub account and just select the GitHub repo to launch the dashboard.