Streamlit App Tutorial

Below is a concise tutorial on how to build a Streamlit application from scratch, illustrating each of the 12 essential Streamlit commands. This tutorial references the key ideas found in the “12 Essential Commands for Streamlit” article, along with some practical tips for creating an engaging user interface. We’ll also show some optional ways you could incorporate a “March Madness” or other thematic angle into your app.

---

1. Introduction

What Is Streamlit?

Streamlit is an open-source Python library that makes it easy to build interactive web apps for data science and machine learning projects. With just a few lines of Python, you can create custom dashboards, visualization tools, and forms, and share them immediately with collaborators.

Why Use Streamlit?

- **Fast Prototyping**: Quickly turn data scripts into shareable web apps.
- **Minimal Boilerplate**: No HTML, CSS, or JavaScript required.
- **Live Previews**: See changes instantly when developing your app.
- **Flexible**: Integrates easily with data manipulation libraries (e.g., pandas, NumPy) and popular plotting libraries (e.g., matplotlib, Altair, Plotly).

---

2. Installation and Setup

1. **Create (optional) and activate a virtual environment:**

```bash
python -m venv venv
source venv/bin/activate  # On macOS/Linux
```

```
venv\Scripts\activate     # On Windows
```

2. **Install Streamlit**:

   ```bash
   pip install streamlit
   ```

3. **Create an `app.py` (or any filename) in your project folder**:

   ```bash
   touch app.py
   ```

---

3. Building a Basic Streamlit App

In this tutorial, we'll build a simple app that demonstrates the 12 essential commands discussed below.

Open `app.py` in your editor and start by importing the library:

```python
import streamlit as st
import pandas as pd
import numpy as np
from PIL import Image  # for loading images
import time  # to simulate long-running tasks for progress bar
```

Now, step-by-step, we'll incorporate each of the 12 commands. Feel free to adapt these snippets to your own dataset or "March Madness" theme.

---

1. `st.write`

`st.write()` is one of the most versatile commands: it can display plain text, tables, plots, and more.

```python
# 1. Using st.write
st.title("Welcome to Our Streamlit Tutorial")
st.write("Streamlit is an awesome tool for building data web apps in **Python**.")

# You can also display pandas DataFrames directly:
df = pd.DataFrame({
    "Team": ["Team A", "Team B", "Team C", "Team D"],
    "Wins": [25, 22, 18, 27],
    "Losses": [5, 8, 12, 3]
})
st.write("### Sample DataFrame:", df)
```

> **Tip:** You can use variants like `st.title`, `st.header`, `st.subheader`, `st.code`, and `st.latex` for more specialized text display.

---

2. `st.markdown`

If you're comfortable with Markdown, you can use `st.markdown` to display richly formatted text, images, and more.

```python
# 2. Using st.markdown
st.markdown("Here is a **Markdown** text with [a link](https://www.tdwi.org/).")
```

Markdown is particularly useful for adding bold text, headers, bullets, or embedding images directly with Markdown syntax.

---

3. `st.dataframe`

`st.dataframe()` makes it easy to render data tables with optional width, height, and styling adjustments.

```python
```

```
# 3. Using st.dataframe
st.dataframe(df, width=700, height=200)
```

> If you want a static table (without interactive controls), you can use `st.table()` instead.

---

4. `st.metric`

For quick stats or KPIs, you can use `st.metric`. It displays a label, a main numeric value, and an optional delta.

```python
# 4. Using st.metric
st.metric(label="Win Rate of Team A", value="80%", delta="+5% from last month")
```

You can line up multiple metrics horizontally using `st.columns`.

---

5. `st.line_chart`

If you have numeric data, `st.line_chart` can create a quick interactive chart.

```python
# 5. Using st.line_chart
chart_data = pd.DataFrame(
    np.random.randn(20, 3),
    columns=['Team A', 'Team B', 'Team C']
)
st.line_chart(chart_data)
```

> Streamlit also supports other charting commands like `st.bar_chart`, `st.area_chart`, or more advanced `st.altair_chart`, `st.plotly_chart`, etc.

---

6. `st.pyplot`

You can integrate any matplotlib visualization into Streamlit with `st.pyplot`.

```python
# 6. Using st.pyplot
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [10, 20, 5, 30], marker='o')
ax.set_title("Simple Matplotlib Chart")
st.pyplot(fig)
```

---

7. `st.text_input`

To gather input from users, you can create a text box:

```python
# 7. Using st.text_input
user_input = st.text_input("Enter your favorite team", "Team A")
st.write("You entered:", user_input)
```

---

8. `st.selectbox`

`st.selectbox` displays a dropdown menu for selecting options.

```python
# 8. Using st.selectbox
selected_option = st.selectbox("Pick a stat to view", ["Wins", "Losses", "Win
Rate"])
st.write("You selected:", selected_option)
```

> Other input widgets you could explore include `st.button`, `st.slider`, `st.radio`,

> `st.checkbox`, etc.

---

9. `st.image`

To display images (e.g., your March Madness bracket), open them with PIL and pass them to `st.image`.

```python
# 9. Using st.image
# Suppose you have an image named "march_madness_logo.png" in your project folder:
# image = Image.open("march_madness_logo.png")
# st.image(image, caption="March Madness")
st.write("Add an image here when you have one.")
```

---

10. `st.progress`

If your app runs a long process, show a progress bar to keep users informed.

```python
# 10. Using st.progress
with st.spinner("Simulating a long process..."):
    my_bar = st.progress(0)
    for percent_complete in range(100):
        time.sleep(0.01)  # simulate work
        my_bar.progress(percent_complete + 1)
st.success("Process is complete!")
```

---

11. `st.sidebar`

Create a sidebar for navigation or filtering options. Everything under the `with st.sidebar` block appears in the sidebar.

```python
# 11. Using st.sidebar
with st.sidebar:
    st.header("Sidebar Options")
    sidebar_choice = st.radio("Select an action:", ["Show Data", "Show Charts"])
```

Then you can conditionally display things in the main area depending on the user's choice.

---

12. `st.cache` (or the newer `st.cache_data` / `st.cache_resource`)

Caching helps speed up performance by memoizing function outputs, especially useful for large datasets or expensive computations. (Note that in Streamlit newer versions, `st.cache_data` is often recommended, but the concept remains similar.)

```python
# 12. Using st.cache (or st.cache_data in newer versions)
@st.cache
def load_data():
    # pretend this function reads a large CSV or does complex computation
    data = pd.DataFrame(np.random.randn(1000, 3), columns=["Team A", "Team B", "Team
C"])
    return data

if st.button("Load Large Dataset"):
    big_df = load_data()
    st.write(big_df.head())
```

---

4. Running the App

1. Navigate to the folder containing your `app.py`.
2. Run:

```bash
```

```
streamlit run app.py
```

3. Streamlit will spin up a local web server. Look for a URL in your terminal that typically starts with `http://localhost:8501/`. Open that link in your browser to see your app.

---

5. Putting It All Together

Below is a minimal, combined sample of what your `app.py` might look like (feel free to copy-paste and adapt as you wish):

```python
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time

st.title("TDWI March Madness Streamlit App")
st.markdown("""
Welcome to the **March Madness** data analytics app!
Here we'll demonstrate 12 essential Streamlit commands.
""")

# Sample DataFrame
df = pd.DataFrame({
    "Team": ["Team A", "Team B", "Team C", "Team D"],
    "Wins": [25, 22, 18, 27],
    "Losses": [5, 8, 12, 3]
})

# Display DataFrame
st.write("### Team Performance:", df)
st.dataframe(df)

# Metrics
st.metric("Team A Win Rate", "83%", "+10% from last week")
```

```
# Chart
chart_data = pd.DataFrame(np.random.randn(20, 3), columns=['Team A', 'Team B', 'Team
C'])
st.line_chart(chart_data)

# Matplotlib example
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [10, 20, 5, 30], marker='o')
ax.set_title("Simple Matplotlib Chart")
st.pyplot(fig)

# Text input
user_input = st.text_input("Enter your favorite team:", "Team A")
st.write("You entered:", user_input)

# Selectbox
selected_stat = st.selectbox("Select stat to view:", ["Wins", "Losses"])
st.write("You selected:", selected_stat)

# Image placeholder
st.write("### March Madness Logo")
st.write("(*Add an image with `st.image` if you have one*)")

# Progress simulation
with st.spinner("Processing..."):
    my_bar = st.progress(0)
    for i in range(100):
        time.sleep(0.01)
        my_bar.progress(i+1)
st.success("Done!")

# Sidebar
with st.sidebar:
    st.header("Sidebar Settings")
    st.radio("Choose Option", ["Overview", "Detailed Stats"])

@st.cache
def load_data(num_rows=1000):
    return pd.DataFrame(np.random.randn(num_rows, 3), columns=["Team A", "Team B",
"Team C"])

if st.button("Load Large Data"):
    large_df = load_data()
    st.write(large_df.head())
```

---

6. Best Practices and Next Steps

1. **Use `st.cache` (or the newer caching commands) wisely** to reduce the load time when loading big files or training models.
2. **Organize your layout** with `st.columns`, `st.container`, or `st.expander` to keep your interface clean.
3. **Split logic** into multiple pages or files if your app becomes large (Streamlit supports multipage apps).
4. **Deploy your app** on Streamlit Cloud or other hosting services (Heroku, AWS, etc.) to share with anyone, anywhere.

Conclusion

You now know how to use the 12 essential Streamlit commands to build an interactive, data-driven web app. Whether you're creating a **March Madness** bracket analyzer, a data science demo, or a business intelligence dashboard, Streamlit's simplicity and power allow you to go from concept to production-ready app quickly.

- **Further Reading**:
  - Streamlit Documentation
  - Streamlit Cheat Sheet
  - Official Gallery for real-world examples

Happy Streamlit-ing! If you found this tutorial helpful, feel free to share it or reach out with questions. Enjoy building your next data web app!