*# Getting Started with Streamlit: Building Interactive Dashboards

Streamlit is a Python framework that makes it easy to create interactive web applications for data visualization and analysis. Unlike Jupyter notebooks that run in cells, Streamlit apps run as web applications that can be shared with others.

Key Differences from Jupyter Notebooks

Feature	Jupyter	Streamlit
Display Format	Cells in browser	Full web app in browser
Interactivity	Manual cell execution	Automatic re-run on interaction
Widgets	Limited (ipywidgets)	Rich built-in widgets
Deployment	Need Binder or JupyterHub	Easy with Streamlit Cloud
Sharing	Share .ipynb files	Share live links
Use Case	Exploration & learning	Dashboards & tools

Part 1: Installation and Setup

Installing Streamlit

Streamlit is installed via pip, just like pandas or matplotlib:

```
pip install streamlit
```

Creating Your First App File

Create a new Python file in your project folder. Let's call it `app.py`:

Running Your App

Once you have code in `app.py`, run it with:

```
streamlit run app.py
```

This will:

- Start a local web server

- Open your app in your default browser at `http://localhost:8501`
- Display any changes automatically as you edit the file (hot reload)

Stopping Your App

Press `Ctrl+C` in your terminal to stop the app.

Part 2: Your First Streamlit App

Let's create a simple "Hello World" app. This introduces the basic concepts: setting page config, displaying text, and understanding how Streamlit renders.

What This Code Does

- **`st.set_page_config()`**: Configures the page title (shown in browser tab) and icon (emoji that appears in tab)
- **`st.title()`**: Creates a large heading at the top of the page
- **`st.write()`**: The most flexible text display function - can show text, numbers, DataFrames, charts, and more
- **`st.markdown()`**: Renders markdown formatting (bold, italics, lists, headers)

Copy This Code into `app.py`

```
import streamlit as st

# Set page title and icon
st.set_page_config(page_title="My First App", page_icon="📊")

# Add a title
st.title("Welcome to Streamlit!")

# Add some text
st.write("This is my first Streamlit app.")

# Add markdown with formatting
st.markdown("""
### Why Streamlit?
- **Fast**: Turn data scripts into shareable web apps
- **Easy**: Pure Python, no JavaScript required
- **Flexible**: Supports any visualization library
""")
```

What You'll See

When you run this with `streamlit run app.py`:

- Browser tab shows "My First App" with the 📊 emoji
- Large heading "Welcome to Streamlit!"
- Text greeting
- Formatted bullet list with bold text

Part 3: Loading and Displaying Data

Now let's work with real data. We'll load the Titanic dataset and display it in different ways. This introduces you to working with pandas DataFrames in Streamlit.

What This Code Does

- **`st.subheader()`**: Creates a smaller heading to organize sections
- **`st.metric()`**: Displays a key number in large text (useful for KPIs)
- **`st.dataframe()`**: Shows an interactive, sortable, scrollable table
- **`st.columns()`**: Divides the screen into side-by-side sections for layout

Key Feature: Columns

When you use `st.columns(2)`, you get two columns of equal width. You can then use a `with` statement to add content to each:

```
col1, col2 = st.columns(2)
with col1:
    st.write("This goes in the left column")
with col2:
    st.write("This goes in the right column")
```

Copy This Code into `app.py`

```python
import streamlit as st
import pandas as pd

st.set_page_config(page_title="Titanic Explorer", page_icon="🚢")

st.title("🚢 Titanic Dataset Explorer")
st.markdown("Explore passenger data from the Titanic disaster")

# Load the data from seaborn
df = sns.load_dataset('titanic')

# Display basic information
st.subheader("Dataset Overview")
st.write(f"Total passengers: {len(df)}")
st.write(f"Total features: {len(df.columns)}")

# Show the first few rows
st.subheader("First Few Rows")
st.dataframe(df.head(10))

# Show data types and missing values in two columns
st.subheader("Data Info")
col1, col2 = st.columns(2)

with col1:
    st.write("**Data Types:**")
    st.write(df.dtypes)

with col2:
    st.write("**Missing Values:**")
    st.write(df.isnull().sum())
```

What You'll See

- Overview section with row and column counts
- An interactive table showing first 10 passengers (you can scroll, sort, search)
- Side-by-side columns showing data types and missing value counts

---

Part 4: Adding Interactive Widgets (Filters)

This is where Streamlit becomes powerful! Widgets let users interact with your app. When a user changes a filter, the entire app re-runs with the new values.

Understanding Widgets

Widgets are interactive controls:

- `st.slider()` - A range selector with min/max
- `st.multiselect()` - User can pick multiple options from a list
- `st.checkbox()` - True/False toggle
- `st.selectbox()` - Dropdown with single selection
- `st.text_input()` - User types text

The Sidebar

`st.sidebar` is a special area on the left side of the page. It's perfect for filters because:

- It doesn't interfere with main content
- Users can collapse it on mobile
- It's visually separate from your data

How Filtering Works

When a user selects values with a widget:

1. The variable gets the selected value(s)
2. We use `.isin()` or boolean conditions to filter the DataFrame
3. The filtered DataFrame is displayed
4. The entire app re-runs instantly

Copy This Code into `app.py`

```python
import streamlit as st
import pandas as pd

st.set_page_config(page_title="Titanic Explorer", page_icon="🚢")

st.title("🚢 Titanic Dataset Explorer")
st.markdown("Explore passenger data from the Titanic disaster")

# Load the data
df = sns.load_dataset('titanic')

# ===== INTERACTIVE FILTERS IN SIDEBAR =====
st.sidebar.header("Filters")

# Filter by passenger class (1st, 2nd, or 3rd class)
selected_class = st.sidebar.multiselect(
    "Select Passenger Class:",
    options=sorted(df['pclass'].unique()),  # Get unique class valu
    default=sorted(df['pclass'].unique())   # All classes selected
)

# Filter by age range using a slider
age_min, age_max = st.sidebar.slider(
    "Select age Range:",
    min_value=int(df['age'].min()),
    max_value=int(df['age'].max()),
    value=(int(df['age'].min()), int(df['age'].max()))  # Default t
)

# Filter by gender using radio buttons
selected_sex = st.sidebar.radio(
    "Select Gender:",
    options=["All", "male", "female"],
    index=0  # Default to "All"
)

# Filter by embarkation port using selectbox (dropdown)
selected_port = st.sidebar.selectbox(
    "Embarkation Port:",
    options=["All"] + sorted(df['embarked'].dropna().unique().tolis
    index=0
)

# ===== APPLY FILTERS =====
# Create a filtered DataFrame by applying all conditions
filtered_df = df[
    (df['pclass'].isin(selected_class)) &      # Class must be in s
    (df['age'] >= age_min) &                   # age must be >= min
    (df['age'] <= age_max)                     # age must be <= max
]

# Apply gender filter based on radio selection
```

```
if selected_sex != "All":
    filtered_df = filtered_df[filtered_df['sex'] == selected_sex]

# Apply port filter based on selectbox selection
if selected_port != "All":
    filtered_df = filtered_df[filtered_df['embarked'] == selected_p

# Display the results
st.subheader("Filtered Data")
st.write(f"Showing {len(filtered_df)} of {len(df)} passengers")
st.dataframe(filtered_df)
```

What You'll See

- Left sidebar with five filter controls
- As you change filters, the table updates instantly
- Counter shows how many passengers match your filters

New Widgets Introduced

Radio Buttons (`st.radio`):

- Use when users must pick exactly ONE option
- Better than multiselect when "select all" isn't needed
- Example: Gender filter (All, male, or female)

Selectbox (`st.selectbox`):

- Dropdown menu for single selection
- Great for many options without cluttering the UI
- Example: Embarkation port (saves space compared to radio buttons)

How to Use

Try these interactions:

1. Select only 1st class passengers with multiselect
2. Change the age range slider to 18-65
3. Use radio buttons to select only "male" passengers
4. Pick a specific port from the dropdown
5. Watch the table update with each change!

Part 5: Adding Visualizations

Now let's add charts using matplotlib and seaborn (libraries you've already used in notebooks).

Displaying Charts in Streamlit

- `st.pyplot()` - Shows matplotlib figures
- `st.plotly_chart()` - Shows interactive Plotly charts

Understanding the Visualization Code

Each visualization follows this pattern:

1. Create a matplotlib figure and axes: `fig, ax = plt.subplots()`
2. Draw your chart using pandas/matplotlib: `df.plot(ax=ax)`
3. Display it: `st.pyplot(fig)`

All visualizations automatically respond to your filters!

Copy This Code into `app.py`

```python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

st.set_page_config(page_title="Titanic Explorer", page_icon="🚢")

st.title("🚢 Titanic Dataset Explorer")
st.markdown("Explore passenger data from the Titanic disaster")

# Load the data
df = sns.load_dataset('titanic')

# ===== INTERACTIVE FILTERS =====
st.sidebar.header("Filters")

selected_class = st.sidebar.multiselect(
    "Select Passenger Class:",
    options=sorted(df['pclass'].unique()),
    default=sorted(df['pclass'].unique())
)

age_min, age_max = st.sidebar.slider(
    "Select age Range:",
    min_value=int(df['age'].min()),
    max_value=int(df['age'].max()),
    value=(int(df['age'].min()), int(df['age'].max()))
)

selected_sex = st.sidebar.radio(
    "Select Gender:",
    options=["All", "male", "female"],
    index=0
)

selected_port = st.sidebar.selectbox(
    "Embarkation Port:",
    options=["All"] + sorted(df['embarked'].dropna().unique().tolis
    index=0
)

# Add checkbox to show/hide survivors only
survivors_only = st.sidebar.checkbox(
    "Show Survivors Only",
    value=False
)

# ===== APPLY FILTERS =====
filtered_df = df[
    (df['pclass'].isin(selected_class)) &
    (df['age'] >= age_min) &
    (df['age'] <= age_max)
```

```
]

if selected_sex != "All":
    filtered_df = filtered_df[filtered_df['sex'] == selected_sex]

if selected_port != "All":
    filtered_df = filtered_df[filtered_df['embarked'] == selected_p

if survivors_only:
    filtered_df = filtered_df[filtered_df['survived'] == 1]

# Display the filtered data
st.subheader("Filtered Data")
st.write(f"Showing {len(filtered_df)} of {len(df)} passengers")
st.dataframe(filtered_df, use_container_width=True)

# ===== VISUALIZATIONS IN TWO COLUMNS =====
st.subheader("Visualizations")

col1, col2 = st.columns(2)

# CHART 1: Survival by Passenger Class
with col1:
    st.write("**Survival by Passenger Class**")
    fig, ax = plt.subplots(figsize=(6, 4))

    # Calculate survival rate for each class
    survival_by_class = filtered_df.groupby('pclass')['survived'].m

    # Create bar chart
    survival_by_class.plot(kind='bar', ax=ax, color='steelblue')
    ax.set_ylabel('Survival Rate')
    ax.set_xlabel('Passenger Class')
    ax.set_title('Survival Rate by Class')
    plt.xticks(rotation=0)

    st.pyplot(fig)

# CHART 2: age Distribution
with col2:
    st.write("**age Distribution**")
    fig, ax = plt.subplots(figsize=(6, 4))

    # Create histogram of ages
    filtered_df['age'].hist(bins=20, ax=ax, color='coral', edgecolo
    ax.set_xlabel('age')
    ax.set_ylabel('Count')
    ax.set_title('Passenger age Distribution')

    st.pyplot(fig)

# CHART 3: Survival by Gender (full width)
```

```
# CHART 3: Survival by Gender (full width)
st.write("**Survival by Gender**")
fig, ax = plt.subplots(figsize=(8, 4))

# Create stacked bar chart
survival_by_sex = filtered_df.groupby('sex')['survived'].value_coun
survival_by_sex.plot(kind='bar', ax=ax, color=['#FF6B6B', '#51CF66'
ax.set_ylabel('Count')
ax.set_xlabel('Gender')
ax.set_title('Survival Counts by Gender')
ax.legend(['Did Not Survive', 'Survived'])
plt.xticks(rotation=0)

st.pyplot(fig)
```

What You'll See

- Left column: Bar chart of survival rates by class
- Right column: Histogram showing age distribution
- Bottom: Stacked bar chart of survival by gender
- **All charts update when you change filters!**

Widget Recap

So far we've used:

- **Multiselect**: For passenger class (multiple selections)
- **Slider**: For age range (numeric range)
- **Radio buttons**: For gender (single choice from few options)
- **Selectbox**: For embarkation port (single choice from several options)
- **Checkbox**: For survivors filter (on/off toggle) - Added in this section!

Each widget was chosen for its specific use case - this makes your dashboard intuitive!

Part 6: Adding Summary Metrics (KPIs)

Key Performance Indicators (KPIs) are important numbers displayed prominently. Use `st.metric()` to show them in a professional way.

What Are Metrics?

Metrics are large numbers that answer a specific question:

- "How many passengers are we looking at?" → Total Passengers
- "What percentage survived?" → Survival Rate
- "What was the average age?" → Average age
- "How much did passengers pay?" → Average Fare

Understanding the Code

Each metric needs:

1. A label (what the metric measures)
2. A value (the number to display)
3. Optional: A color indicator (positive/negative change)

Copy This Code into `app.py`

```python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

st.set_page_config(page_title="Titanic Explorer", page_icon="🚢")

st.title("🚢 Titanic Dataset Explorer")
st.markdown("Explore passenger data from the Titanic disaster")

# Load the data
df = sns.load_dataset('titanic')

# ===== INTERACTIVE FILTERS =====
st.sidebar.header("Filters")

selected_class = st.sidebar.multiselect(
    "Select Passenger Class:",
    options=sorted(df['pclass'].unique()),
    default=sorted(df['pclass'].unique())
)

age_min, age_max = st.sidebar.slider(
    "Select age Range:",
    min_value=int(df['age'].min()),
    max_value=int(df['age'].max()),
    value=(int(df['age'].min()), int(df['age'].max()))
)

selected_sex = st.sidebar.radio(
    "Select Gender:",
    options=["All", "male", "female"],
    index=0
)

selected_port = st.sidebar.selectbox(
    "Embarkation Port:",
    options=["All"] + sorted(df['embarked'].dropna().unique().tolis
    index=0
)

survivors_only = st.sidebar.checkbox(
    "Show Survivors Only",
    value=False
)

# Number input for minimum fare
min_fare = st.sidebar.number_input(
    "Minimum Fare ($):",
    min_value=0.0,
    max_value=float(df['fare'].max()),
    value=0.0.
```

```
    step=10.0
)

# ===== APPLY FILTERS =====
filtered_df = df[
    (df['pclass'].isin(selected_class)) &
    (df['age'] >= age_min) &
    (df['age'] <= age_max) &
    (df['fare'] >= min_fare)
]

if selected_sex != "All":
    filtered_df = filtered_df[filtered_df['sex'] == selected_sex]

if selected_port != "All":
    filtered_df = filtered_df[filtered_df['embarked'] == selected_p

if survivors_only:
    filtered_df = filtered_df[filtered_df['survived'] == 1]

# ===== DISPLAY KEY METRICS =====
st.subheader("Key Metrics")

# Create 4 columns for metrics (equally spaced)
metric_col1, metric_col2, metric_col3, metric_col4 = st.columns(4)

# Metric 1: Total Passengers
with metric_col1:
    st.metric(
        label="Total Passengers",
        value=len(filtered_df)
    )

# Metric 2: Survival Rate (as percentage)
with metric_col2:
    survival_rate = (filtered_df['survived'].sum() / len(filtered_d
    st.metric(
        label="Survival Rate",
        value=f"{survival_rate:.1f}%"
    )

# Metric 3: Average age
with metric_col3:
    avg_age = filtered_df['age'].mean()
    st.metric(
        label="Average age",
        value=f"{avg_age:.1f}"
    )

# Metric 4: Average Fare (ticket price)
with metric_col4:
    avg_fare = filtered_df['fare'].mean()
```

```
    avg_fare = filtered_df['fare'].mean()
    st.metric(
        label="Average Fare",
        value=f"${avg_fare:.2f}"
    )

# Display the filtered data
st.subheader("Filtered Data")
st.write(f"Showing {len(filtered_df)} of {len(df)} passengers")
st.dataframe(filtered_df, use_container_width=True)

# ===== VISUALIZATIONS =====
st.subheader("Visualizations")

col1, col2 = st.columns(2)

with col1:
    st.write("**Survival by Passenger Class**")
    fig, ax = plt.subplots(figsize=(6, 4))
    survival_by_class = filtered_df.groupby('pclass')['survived'].m
    survival_by_class.plot(kind='bar', ax=ax, color='steelblue')
    ax.set_ylabel('Survival Rate')
    ax.set_xlabel('Passenger Class')
    plt.xticks(rotation=0)
    st.pyplot(fig)

with col2:
    st.write("**age Distribution**")
    fig, ax = plt.subplots(figsize=(6, 4))
    filtered_df['age'].hist(bins=20, ax=ax, color='coral', edgecolo
    ax.set_xlabel('age')
    ax.set_ylabel('Count')
    st.pyplot(fig)
```

What You'll See

- Four large metric boxes at the top
- Each metric shows a key number
- Metrics update when you change filters
- Metrics appear before detailed data, following dashboard best practices

New Widget: Number Input

We added `st.number_input()` for the minimum fare filter. This widget:

- Lets users type precise numeric values
- Includes increment/decrement buttons
- Has min/max bounds to prevent invalid entries
- Better than a slider when users need exact values

When to use what:

- **Slider**: Visual, quick adjustments, approximate values (age range)
- **Number Input**: Precise entry, specific values (minimum fare of $50.25)

Part 7: Organizing with Tabs and Expanders

As your dashboard grows, organization becomes important. Tabs and expanders help organize content without overwhelming the user.

Understanding Tabs

Tabs let you group related content. Users click to switch between tabs. Great for:

- Separating "Data" vs "Analysis" vs "Settings"
- Keeping related visualizations together
- Reducing initial page scrolling

Understanding Expanders

Expanders hide content by default. Users click to reveal it. Great for:

- Additional details or statistics
- Reference information
- Less frequently used features

Copy This Code into `app.py`

```python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Use layout="wide" to make better use of screen space
st.set_page_config(
    page_title="Titanic Explorer",
    page_icon="🚢",
    layout="wide"
)

st.title("🚢 Titanic Dataset Explorer")
st.markdown("Explore passenger data from the Titanic disaster")

# Load the data
df = sns.load_dataset('titanic')

# ===== INTERACTIVE FILTERS IN SIDEBAR =====
with st.sidebar:
    st.header("Filters")

    selected_class = st.multiselect(
        "Select Passenger Class:",
        options=sorted(df['pclass'].unique()),
        default=sorted(df['pclass'].unique())
    )

    age_min, age_max = st.slider(
        "Select age Range:",
        min_value=int(df['age'].min()),
        max_value=int(df['age'].max()),
        value=(int(df['age'].min()), int(df['age'].max()))
    )

    selected_sex = st.radio(
        "Select Gender:",
        options=["All", "male", "female"],
        index=0
    )

    selected_port = st.selectbox(
        "Embarkation Port:",
        options=["All"] + sorted(df['embarked'].dropna().unique().t
        index=0
    )

    survivors_only = st.checkbox(
        "Show Survivors Only",
        value=False
    )
```

```
    min_fare = st.number_input(
        "Minimum Fare ($):",
        min_value=0.0,
        max_value=float(df['fare'].max()),
        value=0.0,
        step=10.0
    )

    # Text input for searching across text columns
    text_search = st.text_input(
        "Search Text:",
        value="",
        placeholder="Search class, group, port..."
    )

# ===== APPLY FILTERS =====
filtered_df = df[
    (df['pclass'].isin(selected_class)) &
    (df['age'] >= age_min) &
    (df['age'] <= age_max) &
    (df['fare'] >= min_fare)
]

if selected_sex != "All":
    filtered_df = filtered_df[filtered_df['sex'] == selected_sex]

if selected_port != "All":
    filtered_df = filtered_df[filtered_df['embarked'] == selected_p

if survivors_only:
    filtered_df = filtered_df[filtered_df['survived'] == 1]

if text_search:
    # Search across multiple text columns
    mask = (
        filtered_df['who'].astype(str).str.contains(text_search, ca
        filtered_df['class'].astype(str).str.contains(text_search,
        filtered_df['embark_town'].astype(str).str.contains(text_se
        filtered_df['deck'].astype(str).str.contains(text_search, c
    )
    filtered_df = filtered_df[mask]

# ===== DISPLAY KEY METRICS =====
st.subheader("Key Metrics")
metric_col1, metric_col2, metric_col3, metric_col4 = st.columns(4)

with metric_col1:
    st.metric("Total Passengers", len(filtered_df))

with metric_col2:
    survival_rate = (filtered_df['survived'].sum() / len(filtered_d
    st.metric("Survival Rate", f"{survival_rate:.1f}%")
```

```
    st.metric("Survival Rate", f"{survival_rate:.1f}%")

with metric_col3:
    avg_age = filtered_df['age'].mean()
    st.metric("Average age", f"{avg_age:.1f}")

with metric_col4:
    avg_fare = filtered_df['fare'].mean()
    st.metric("Average Fare", f"${avg_fare:.2f}")

# ===== ORGANIZE CONTENT IN TABS =====
tab1, tab2, tab3 = st.tabs(["📊 Data", "📈 Visualizations", "ℹ️ De

# TAB 1: RAW DATA
with tab1:
    st.subheader("Dataset")
    st.write(f"Showing {len(filtered_df)} of {len(df)} passengers")
    st.dataframe(filtered_df, use_container_width=True)

# TAB 2: VISUALIZATIONS
with tab2:
    st.subheader("Visualizations")

    col1, col2 = st.columns(2)

    with col1:
        st.write("**Survival by Passenger Class**")
        fig, ax = plt.subplots(figsize=(6, 4))
        survival_by_class = filtered_df.groupby('pclass')['survived
        survival_by_class.plot(kind='bar', ax=ax, color='steelblue'
        ax.set_ylabel('Survival Rate')
        ax.set_xlabel('Passenger Class')
        plt.xticks(rotation=0)
        st.pyplot(fig)

    with col2:
        st.write("**age Distribution**")
        fig, ax = plt.subplots(figsize=(6, 4))
        filtered_df['age'].hist(bins=20, ax=ax, color='coral', edge
        ax.set_xlabel('age')
        ax.set_ylabel('Count')
        st.pyplot(fig)

    st.write("**Survival by Gender**")
    fig, ax = plt.subplots(figsize=(8, 4))
    survival_by_sex = filtered_df.groupby('sex')['survived'].value_
    survival_by_sex.plot(kind='bar', ax=ax)
    ax.set_ylabel('Count')
    ax.set_xlabel('Gender')
    ax.legend(['Did Not Survive', 'Survived'])
    plt.xticks(rotation=0)
    st.pyplot(fig)
```

```
# TAB 3: ADDITIONAL DETAILS
with tab3:
    st.subheader("Additional Information")

    # Expander 1: Statistics
    with st.expander("📊 View Data Statistics"):
        st.write("**Descriptive Statistics:**")
        st.dataframe(filtered_df.describe())

    # Expander 2: Correlation Matrix
    with st.expander("🔗 View Correlation Matrix"):
        st.write("**Correlation Matrix:**")
        numeric_cols = filtered_df.select_dtypes(include=['int64',
        correlation_matrix = filtered_df[numeric_cols].corr()

        fig, ax = plt.subplots(figsize=(8, 6))
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm'
        st.pyplot(fig)

    # Expander 3: About the dataset
    with st.expander("📖 About This Dataset"):
        st.write("""
        **The Titanic Dataset**

        This dataset contains information about passengers on the RM

        **Column Descriptions:**
        - **survived**: Whether the passenger survived (0 = No, 1 =
        - **pclass**: Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
        - **sex**: Passenger's gender
        - **age**: Age in years
        - **sibsp**: Number of siblings/spouses aboard
        - **parch**: Number of parents/children aboard
        - **fare**: Ticket fare paid in pounds sterling
        - **embarked**: Port of embarkation (C = Cherbourg, Q = Quee
        - **class**: Ticket class as a string
        - **who**: Passenger group (man, woman, child)
        - **adult_male**: Whether the passenger is an adult male
        - **deck**: Cabin deck (many missing)
        - **embark_town**: Town of embarkation
        - **alive**: Survival status as text
        - **alone**: Whether the passenger was alone
        """)
```

What You'll See

- **Data Tab**: Raw filtered data table
- **Visualizations Tab**: All your charts organized together
- **Details Tab**: Statistics, correlations, and dataset info hidden in expanders
- Click expanders to reveal/hide detailed information

New Widget: Text Input

We added `st.text_input()` for searching across multiple text columns. This widget:

- Accepts free-form text entry
- Supports placeholder text to guide users
- Perfect for search/filter functionality
- Searches across 'who' (man/woman/child), 'class', 'embark_town', and 'deck' columns

Text Input Use Cases:

- Search functionality across multiple fields (type "First" to find First class, "woman" to find women, "Southampton" for that port)
- Custom labels or annotations
- User-defined parameters
- Form fields (email, username, etc.)

How the multi-column search works:

```
if text_search:
    mask = (
        filtered_df['who'].astype(str).str.contains(text_search, ca
        filtered_df['class'].astype(str).str.contains(text_search, 
        filtered_df['embark_town'].astype(str).str.contains(text_se
        filtered_df['deck'].astype(str).str.contains(text_search, c
    )
    filtered_df = filtered_df[mask]
```

This creates a boolean mask that checks if the search text appears in ANY of the specified columns.

Key Concepts

- `st.tabs()` creates clickable tabs
- `with tab1:` means all code inside is in that tab
- `st.expander()` creates a collapsible section
- `layout="wide"` uses the full screen width

Widget Progression Summary

Notice how we've gradually added more interactive features:

- **Part 4**: `multiselect` and `slider` (range selection)
- **Part 5**: `radio` (single choice), `selectbox` (dropdown), `checkbox` (toggle)
- **Part 6**: `number_input` (numeric entry)
- **Part 7**: `text_input` (search/filter by text)

Each widget serves a specific purpose:

- **Radio buttons**: When user must pick exactly one option (gender: male/female/all)
- **Selectbox**: Dropdown for single selection with many options (embarkation port)
- **Multiselect**: When user can pick multiple options (passenger classes)
- **Checkbox**: Simple on/off toggle (show survivors only)
- **Slider**: Visual selection of numeric ranges (age range)
- **Number input**: Precise numeric entry (minimum fare)
- **Text input**: Free-form text entry (search by name)

Part 8: Quick Reference - Common Streamlit Components

Text and Display

```
st.title("Large Title")           # Big heading (use once per app)
st.header("Medium Heading")        # Medium heading
st.subheader("Small Heading")      # Small heading
st.write("Text or data")           # Most flexible - works with tex
st.text("Monospace text")          # Text in fixed-width font
st.markdown("# Markdown formatting")  # Supports markdown (bold, it
st.code("print('hello')")          # Display code in a box
```

Data Display

```
st.dataframe(df)                   # Interactive DataFrame (sortabl
st.table(df)                       # Static table (better for small
st.json({"key": "value"})          # Display JSON formatted data
```

Widgets (Interactive Input)

```python
# Button widgets
st.button("Click Me")              # Returns True when clicked
st.checkbox("Check Me")            # Returns True/False
st.toggle("Toggle Me")             # Switch toggle (True/False)

# Selection widgets
st.radio("Pick One", ["A", "B", "C"])          # Radio buttons (sin
st.selectbox("Pick One", ["A", "B", "C"])     # Dropdown menu (sing
st.multiselect("Pick Many", ["A", "B", "C"])  # Multi-select dropdo

# Numeric input widgets
st.slider("Pick Number", 0, 100)               # Slider (single val
st.slider("Pick Range", 0, 100, (20, 80))      # Range slider (two
st.number_input("Enter number", value=0, step=1)  # Number input bo

# Text input widgets
st.text_input("Enter text")                    # Single-line text i
st.text_area("Enter long text")                # Multi-line text in

# Date and time widgets
st.date_input("Pick date")                     # Date picker
st.time_input("Pick time")                     # Time picker

# Other widgets
st.color_picker("Pick color")                  # Color picker
st.file_uploader("Upload file")                # File upload button
```

Layout and Organization

```python
st.columns(3)                      # Create 3 equal-width columns
st.tabs(["Tab1", "Tab2"])          # Create tabs
st.expander("Click to expand")     # Collapsible section
st.container()                     # Invisible container for groupi
st.sidebar                         # Left sidebar area
```

Visualizations

```python
st.pyplot(fig)                     # Matplotlib figure
st.plotly_chart(fig)               # Plotly figure
st.bar_chart(df)                   # Simple bar chart
st.line_chart(df)                  # Simple line chart
st.area_chart(df)                  # Simple area chart
```

Metrics and Status

```
st.metric("Label", "value")          # Large KPI display
st.success("✅ Success message")     # Green success box
st.warning("⚠️ Warning message")     # Yellow warning box
st.error("❌ Error message")         # Red error box
st.info("ℹ️ Info message")           # Blue info box
```

File Operations

```
st.download_button(                  # Let users download files
    label="Download CSV",
    data=csv_string,
    file_name="data.csv"
)
st.file_uploader("Upload file")      # Let users upload files
```

Part 9: Complete Production-Ready Dashboard

This is a full, professional dashboard that combines all the concepts. Use this as a template for your own projects.

Key Features

- **Caching**: `@st.cache_data` loads data only once (much faster)
- **Professional Layout**: Wide mode, custom colors, organized sections
- **Download Functionality**: Users can download filtered data
- **Reset Button**: Clear all filters at once
- **Comprehensive Organization**: Tabs for different data explorations

Copy This Entire Code into `app.py`

```python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# ===== PAGE CONFIGURATION =====
st.set_page_config(
    page_title="Titanic Dashboard",
    page_icon="🚢",
    layout="wide",
    initial_sidebar_state="expanded"
)

# ===== CUSTOM STYLING =====
st.markdown("""
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-s
        }
    </style>
""", unsafe_allow_html=True)

# ===== DATA LOADING WITH CACHING =====
# The @st.cache_data decorator tells Streamlit to only load the data
# and reuse it instead of reloading on every interaction
@st.cache_data
def load_data():
    """Load and return the Titanic dataset"""
    return sns.load_dataset('titanic')

df = load_data()

# ===== PAGE TITLE =====
st.title("🚢 Titanic Dataset Dashboard")
st.markdown("Interactive exploration of passenger survival data fro

# ===== SIDEBAR FILTERS =====
with st.sidebar:
    st.header("🔍 Filter Options")

    # Filter 1: Passenger Class
    selected_class = st.multiselect(
        "Passenger Class",
        options=sorted(df['pclass'].unique()),
        default=sorted(df['pclass'].unique()),
        help="Select which ticket classes to include"
    )

    # Filter 2: age Range
    age_range = st.slider(
        "age Range",
        min_value=int(df['age'].min()),
```

```
        max_value=int(df['age'].max()),
        value=(int(df['age'].min()), int(df['age'].max())),
        help="Drag sliders to select age range"
    )

    # Filter 3: Gender
    selected_sex = st.multiselect(
        "Gender",
        options=df['sex'].unique(),
        default=list(df['sex'].unique()),
        help="Select which genders to include"
    )

    # Reset button to clear all filters
    st.divider()  # Visual separator
    if st.button("🔄 Reset Filters"):
        # This reruns the app with default values
        st.rerun()

# ===== APPLY FILTERS =====
filtered_df = df[
    (df['pclass'].isin(selected_class)) &
    (df['age'] >= age_range[0]) &
    (df['age'] <= age_range[1]) &
    (df['sex'].isin(selected_sex))
]

# ===== KEY METRICS SECTION =====
st.subheader("📊 Key Metrics")

metric_cols = st.columns(4)

with metric_cols[0]:
    st.metric(
        label="Total Passengers",
        value=len(filtered_df),
        help="Number of passengers matching filters"
    )

with metric_cols[1]:
    survival_rate = (filtered_df['survived'].sum() / len(filtered_d
    st.metric(
        label="Survival Rate",
        value=f"{survival_rate:.1f}%",
        help="Percentage of passengers who survived"
    )

with metric_cols[2]:
    avg_age = filtered_df['age'].mean()
    st.metric(
        label="Average age",
        value=f"{avg_age:.1f}"
```

```
        value=f"{avg_age:.1f}",
        help="Mean age of filtered passengers"
    )

with metric_cols[3]:
    avg_fare = filtered_df['fare'].mean()
    st.metric(
        label="Average Fare",
        value=f"${avg_fare:.2f}",
        help="Mean ticket price"
    )

# ===== MAIN CONTENT TABS =====
tab1, tab2, tab3, tab4 = st.tabs(["📈 Overview", "🔍 Data", "📊 An

# ===== TAB 1: OVERVIEW =====
with tab1:
    st.subheader("Data Overview")

    overview_col1, overview_col2 = st.columns(2)

    with overview_col1:
        st.metric("Records Displayed", len(filtered_df))
        st.metric("Total Records", len(df))

    with overview_col2:
        st.metric("Survivors", int(filtered_df['survived'].sum()))
        st.metric("Non-Survivors", len(filtered_df) - int(filtered_

# ===== TAB 2: DATA TABLE =====
with tab2:
    st.subheader("Dataset View")
    st.write(f"Showing {len(filtered_df)} passengers:")
    st.dataframe(filtered_df, use_container_width=True)

    st.divider()

    # Download button for filtered data
    csv = filtered_df.to_csv(index=False)
    st.download_button(
        label="📥 Download as CSV",
        data=csv,
        file_name="titanic_filtered.csv",
        mime="text/csv",
        help="Download the filtered data as a CSV file"
    )

# ===== TAB 3: VISUALIZATIONS =====
with tab3:
    st.subheader("Data Visualizations")

    viz_col1, viz_col2 = st.columns(2)
```

```
    # CHART 1: Survival by Class
    with viz_col1:
        st.write("**Survival Rate by Passenger Class**")
        fig, ax = plt.subplots(figsize=(6, 4))
        survival_by_class = filtered_df.groupby('pclass')['survived
        colors = ['#FF6B6B', '#4ECDC4', '#45B7D1']
        survival_by_class.plot(kind='bar', ax=ax, color=colors)
        ax.set_ylabel('Survival Rate (%)')
        ax.set_xlabel('Class')
        ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
        ax.grid(axis='y', alpha=0.3)
        st.pyplot(fig)

    # CHART 2: age Distribution
    with viz_col2:
        st.write("**age Distribution**")
        fig, ax = plt.subplots(figsize=(6, 4))
        filtered_df['age'].hist(bins=25, ax=ax, color='#95E1D3', ed
        ax.set_xlabel('age')
        ax.set_ylabel('Count')
        ax.grid(axis='y', alpha=0.3)
        st.pyplot(fig)

    # CHART 3: Age by Gender Boxplot (full width)
    st.write("**Age Distribution by Gender**")
    fig, ax = plt.subplots(figsize=(6, 4))
    filtered_df.boxplot(column='age', by='sex', ax=ax, patch_artist
    ax.set_xlabel('Gender')
    ax.set_ylabel('Age')
    ax.set_title('')
    plt.suptitle('')  # Remove default title
    ax.grid(axis='y', alpha=0.3)
    st.pyplot(fig)

# ===== TAB 4: INFORMATION =====
with tab4:
    st.subheader("About")

    with st.expander("📖 Dataset Information"):
        st.write("""
        **The Titanic Dataset**

        This dataset contains information about passengers on the RM
        which sank on April 15, 1912, after hitting an iceberg durir

        The dataset is used extensively for machine learning and dat
        because it has interesting patterns and some missing values
        """)

    with st.expander("📋 Column Descriptions"):
        column_info = {
```

```
            'survived': 'Binary outcome (0 = No, 1 = Yes)',
            'pclass': 'Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)',
            'sex': 'Passenger gender',
            'age': 'Age in years (some missing values)',
            'sibsp': 'Number of siblings/spouses aboard',
            'parch': 'Number of parents/children aboard',
            'fare': 'Ticket fare paid in pounds sterling',
            'embarked': 'Port of embarkation (C, Q, or S)',
            'class': 'Passenger class as a string',
            'who': 'Passenger group (man, woman, child)',
            'adult_male': 'Whether the passenger is an adult male',
            'deck': 'Cabin deck (many missing)',
            'embark_town': 'Town of embarkation',
            'alive': 'Survival status as text',
            'alone': 'Whether the passenger was alone'
        }

        for col, description in column_info.items():
            st.write(f"- **{col}**: {description}")

    with st.expander("📊 Descriptive Statistics"):
        st.write("**Summary Statistics for Numeric Columns:**")
        st.dataframe(filtered_df.describe())

    with st.expander("🔗 Correlation Matrix"):
        st.write("**How numeric columns relate to each other:**")
        numeric_cols = filtered_df.select_dtypes(include=['int64',
        correlation_matrix = filtered_df[numeric_cols].corr()

        fig, ax = plt.subplots(figsize=(8, 6))
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm'
                    center=0, ax=ax, square=True)
        st.pyplot(fig)
```

Key Features Explained

1. **`@st.cache_data`**: Loads data only once, making subsequent interactions instant
2. **`layout="wide"`**: Uses full screen width for better data display
3. **`help=` parameter**: Tooltips that appear when users hover over labels
4. **`st.divider()`**: Visual separator line
5. **`st.rerun()`**: Reruns the entire app (used for reset button)
6. **Color customization**: Uses specific colors for better visualization
7. **`use_container_width=True`**: Makes table take full width
8. **Multiple tabs**: Organizes different types of views

Part 10: Tips and Best Practices

Performance Optimization

When your app loads data or does heavy computation, use caching:

```
@st.cache_data
def load_data():
    """This runs only once, then gets reused"""
    return sns.load_dataset('titanic')  # or pd.read_csv('large_file

@st.cache_data
def expensive_computation(data):
    """Cache results of slow operations"""
    return data.groupby('category').sum()
```

Why caching matters:

- Without it: Every filter change reloads the CSV from disk (slow!)
- With it: Data loads once, filters are instant

Creating Responsive Layouts

Different screen sizes need different layouts:

```
# Good for mobile too
col1, col2, col3 = st.columns(3)

# Wide layouts work better on desktops
if st.checkbox("Use wide layout"):
    st.set_page_config(layout="wide")
```

Handling Errors Gracefully

Always anticipate what could go wrong:

```
try:
    # Your code that might fail
    result = filtered_df['column'].sum()
except KeyError:
    st.error("❌ Column not found in data")
except Exception as e:
    st.error(f"An error occurred: {e}")
```

Providing User Feedback

Let users know what's happening:

```python
# Success messages
st.success("✅ Data loaded successfully!")

# Warnings
st.warning("⚠️ Some data is missing")

# Info messages
st.info("ℹ️ Click the expander for more details")

# Progress indicators
progress_bar = st.progress(0)
for i in range(101):
    progress_bar.progress(i)

# Spinners for long operations
with st.spinner('Loading data...'):
    time.sleep(2)  # Simulating work
st.success("Done!")
```

Organizing Large Apps

For apps with lots of code, consider:

```python
# Page structure (put this at the top)
st.set_page_config(page_title="My App", layout="wide")

# 1. Helper functions
def load_data():
    pass

def process_data(df):
    pass

# 2. Page setup
st.title("My Dashboard")

# 3. Sidebar
with st.sidebar:
    # Filters

# 4. Main content
# All your visualizations and displays
```

Part 11: Deployment

Running Locally

To test your app on your computer:

```
streamlit run app.py
```

The app opens at `http://localhost:8501`

Deploying to Streamlit Cloud (Free!)

Streamlit Community Cloud lets anyone visit your app without running code locally.

Steps to Deploy

1. **Create a GitHub account** (if you don't have one)
2. **Push your code to GitHub**:

   ```
   git init
   git add .
   git commit -m "Initial commit"
   git push origin main
   ```

3. **Create `requirements.txt`**:

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

   This file lists all your Python packages:

   ```
   streamlit==1.28.0
   pandas==2.0.3
   matplotlib==3.7.1
   seaborn==0.12.2
   ```

4. **Go to https://share.streamlit.io**
5. **Click "New app"**
6. **Select your GitHub repository and file**
7. **Click "Deploy"**

That's it! Your app gets a public URL like: `https://my-dashboard-abc123.streamlit.app`

File Structure for Deployment

```
my-app/
├── app.py                    # Your main Streamlit code
├── requirements.txt          # Package dependencies
├── data/                     # Your data files
│   └── titanic.csv
└── README.md                 # Instructions for your app
```

Sharing Your App

Once deployed, you can:

- Share the URL with anyone
- Embed it in a website
- Add to a portfolio
- Use for presentations

Next Steps

Try These Experiments

1. **Change the dataset**: Use `obesity.csv` or `supplements.csv` instead of Titanic
2. **Add more filters**: What other columns could you filter by?
3. **Create new visualizations**: What other charts would be useful?
4. **Combine datasets**: Can you merge two datasets and create a dashboard?

Example: Using Tips Dataset

```
df = sns.load_dataset('tips')

# Show unique values
st.write(df['day'].unique())

# Filter by day
day = st.sidebar.selectbox("Select Day", df['day'].unique())
filtered = df[df['day'] == day]
```

Resources

- **Official Documentation**: https://docs.streamlit.io
- **Streamlit Gallery**: https://streamlit.io/gallery - See amazing examples
- **Streamlit Cheat Sheet**: https://docs.streamlit.io/library/cheatsheet
- **Community Forum**: https://discuss.streamlit.io

Troubleshooting

App Won't Start

```
# Make sure you're in the right directory
pwd

# Check if streamlit is installed
pip list | grep streamlit

# Reinstall if needed
pip install --upgrade streamlit
```

App Runs Slowly

```
# Use caching for data loading
@st.cache_data
def load_data():
    return sns.load_dataset('titanic')  # Loads from seaborn

# Reduce chart resolution
plt.figure(figsize=(6, 4))  # Smaller figures load faster
```

Filters Not Working

```
# Make sure you're using the filtered DataFrame
st.write(len(filtered_df))  # Should show fewer rows

# Check for missing values
st.write(df['age'].isnull().sum())
```

Happy Dashboarding! 🚀

You now have everything you need to create professional data dashboards. Start with Part 2, work through each section, and soon you'll be building interactive tools that make data exploration intuitive and fun!

Remember:

- **Start simple**: Get filters working before adding visualizations
- **Test locally**: Make sure it works on your computer first
- **Share with others**: Deploy and get feedback

- **Iterate**: Keep improving based on what you learn

Good luck! 📊 *