phuse·share
connect·share·advance

Single Day Event

Interactive Swimmer Plot Application for Oncology Response Data using Python and Streamlit

Jagadish Katam,
Princeps Technologies

Disclaimer

- The views and opinions expressed in this presentation are those of the speakers and do not reflect opinions of any other person or organization.

Agenda

- Application Overview
- Python Modules
- Data
- Custom Python Functions
- Introduction to Streamlit
- Building the Web Application
- Demo
- Conclusion
- References

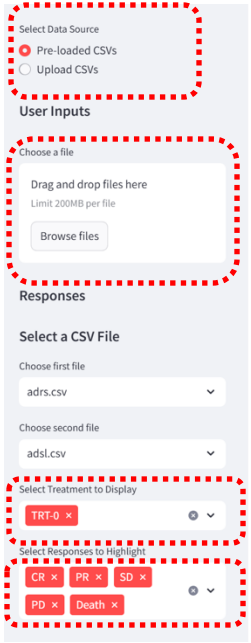
Application Overview

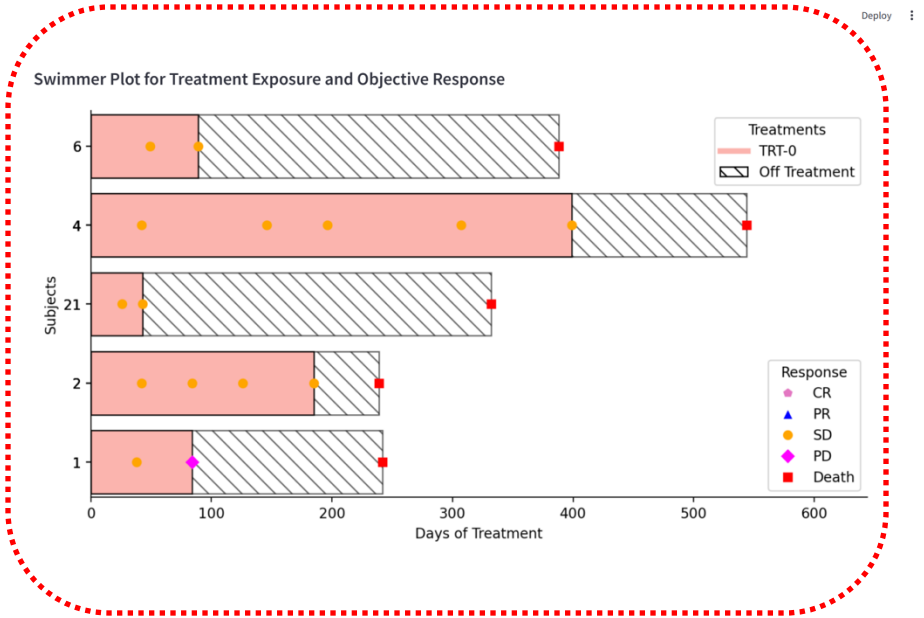
Radio button to select the options

External File or SAS datasets upload option

Check the uploaded files

Select from the drop down for treatment and responses





Actual plot based on the option we choose from the treatment and responses from the sidebar panel

Python Libraries

- **pyreadstat:** For reading SAS/SPSS files, such as .xpt and .sas7bdat formats.
- **pandas (pd):** Utilized for efficient data manipulation through DataFrames.
- **matplotlib.pyplot (plt):** Used for creating static visualizations, including bar and swimmer plots.
- **Line2D, Patch:** Essential for creating custom legend elements within plots.
- **streamlit (st):** Enables the building of interactive web applications directly from Python scripts.
- **tempfile, os:** Employed for handling temporary files and managing file paths.

Data

This section outlines how clinical trial datasets (ADRS and ADSL) are prepared for analysis and visualization, particularly for swimmer plots.

- Import ADRS and ADSL SAS datasets using pyreadstat.
- Filter ADRS data frame for PARAMCD = 'OVRLRESP' (overall response).
- Extract death-related info from ADSL where DTHFL = ‘Y’, then append death records to main data frame with label 'Death’.
- Returns processed data frame and list of unique treatments.

USUBJID	TRT01P	TRT01PN	AVISIT	AVALC	ADY	DTHDY	max_ady
0	TRT-1	1	CYCLE 2 DAY 15-21	SD	38.0	NaN	84.0
0	TRT-1	1	CYCLE 4 DAY 15-21	PD	84.0	NaN	84.0
0	TRT-1	1	NaN	Death	242.0	158.0	84.0
1	TRT-1	1	CYCLE 2 DAY 15-21	SD	42.0	NaN	185.0
1	TRT-1	1	CYCLE 4 DAY 15-21	SD	84.0	NaN	185.0
...	...	...	...	...	...	...	...
107	TRT-8	8	WEEK 6	SD	43.0	NaN	194.0
107	TRT-8	8	WEEK 12	SD	100.0	NaN	194.0
108	TRT-3	3	WEEK 6	SD	44.0	NaN	44.0
109	TRT-13	13	CYCLE 2 DAY 15-21	PD	35.0	NaN	35.0
110	TRT-13	13	CYCLE 2 DAY 15-21	SD	36.0	NaN	36.0

Custom Python Function (1)

- Function:
  preprocess_data_df(data1, data2)

- Arguments:

  data1: Clinical response dataset (e.g., ADRS)
  data2: Subject-level dataset (e.g., ADSL)

- This function is designed to merge and preprocess clinical trial datasets (ADRS and ADSL) for swimmer plot visualization, handling death information and computing treatment duration.

```python
def preprocess_data_df(data1, data2):
    adrs, meta = pyreadstat.read_sas7bdat(f'C:/Users/jagad/OneDrive/Documents/python/test/{data1}.csv')
    # adrs, meta = pyreadstat.read_sas7bdat(temp1.name)
    #adrs = data1.copy()
    adrs['USUBJID'] = adrs['USUBJID'].astype(str)
    # df_sub_list = adrs['USUBJID'].unique()

    # print(df_sub_list)

    adsl, meta = pyreadstat.read_sas7bdat(f'C:/Users/jagad/OneDrive/Documents/python/test/{data2}.csv')
    # adsl, meta = pyreadstat.read_sas7bdat(temp2.name)
    # adsl = data2.copy()
    adsl['USUBJID'] = adsl['USUBJID'].astype(str)
    # Step 1: Filter for specific PARAMCD
    df = adrs[adrs['PARAMCD'] == 'OVRLRESP']

    # Step 2: Keep relevant columns
    columns_to_keep = ['USUBJID', 'AVISIT', 'AVISITN', 'AVALC', 'ADY', 'TRT01P','PARAMCD']
    df = df[columns_to_keep]
```

⋮

```python
# Step 11: Add a prefix to TRT01P
df['TRT01P'] = 'TRT-' + df['TRT01P_numeric'].fillna('').astype(str)

df['DTHDY'] = df['DTHDY'] - df['max_ady']

# Step 12: Extract unique treatment groups
unique_records = df[['TRT01P', 'TRT01P_numeric']].drop_duplicates()
unique_trt = unique_records['TRT01P'].unique()

return df, list(unique_trt)
```

Custom Python Function (2)

```
def preprocess_data(data1, data2):

    # Save the uploaded files to temporary files
with tempfile.NamedTemporaryFile(delete=False, suffix=".sas7bdat") as temp1, \
    tempfile.NamedTemporaryFile(delete=False, suffix=".sas7bdat") as temp2:

    temp1.write(data1.read())
    temp2.write(data2.read())
    temp1.flush()
    temp2.flush()

adrs, meta = pyreadstat.read_sas7bdat(temp1.name)
df_sub_list = adrs['USUBJID'].unique()

print(df_sub_list)

isinstance(df_sub_list, list)

adsl, meta = pyreadstat.read_sas7bdat(temp2.name)

# Step 1: Filter for specific PARAMCD
df = adrs[adrs['PARAMCD'] == 'OVRLRESP']

# Step 2: Keep relevant columns
columns_to_keep = ['USUBJID', 'AVISIT', 'AVISITN', 'AVALC', 'ADY', 'TRT01P']
df = df[columns_to_keep]

# Step 3: Calculate the maximum ADY for each subject
new_data = (
    df.groupby(["USUBJID"], as_index=False)
    .agg(max_ady=("ADY", "max"))
)
df = pd.merge(df, new_data, on="USUBJID", how="left", indicator=False)

# Step 4: Prepare adsl for merging
adsl_columns = ['USUBJID', 'DTHDT', 'DTHDY', 'DTHFL', 'TRT01P']
adsl = adsl[adsl_columns]
adsl['ADY'] = adsl['DTHDY']
adsl = adsl[adsl['DTHFL'] == 'Y']
adsl['AVALC'] = 'Death'

# Step 5: Combine adsl with the main dataframe
df = pd.concat([df, adsl], ignore_index=True)
```

- Function:
  preprocess_data(data1, data2)
- Arguments:

  data1: Uploaded .sas7bdat file for response data (e.g., ADRS)

  data2: Uploaded .sas7bdat file for subject-level data (e.g., ADSL)
- It filters for key response variables, merges death information, and calculates each subject's treatment duration (e.g., maximum ADY). It then prepares the data for visualization by encoding subject IDs and treatment groups into numeric formats.

```
adsl_columns2 = ['USUBJID', 'DTHDY']
adsl2 = adsl[adsl_columns2]
adsl2 = adsl2.rename(columns={'DTHDY': 'DTHDY2'})

df = pd.merge(
    df,
    adsl2,  # Select relevant columns from ADSL
    on='USUBJID',
    how='left'
)

# Step 6: Forward fill max_ady values
df['max_ady'] = df.groupby('USUBJID')['max_ady'].ffill()

# Step 7: Extract numeric portion of USUBJID and sort by USUBJID and ADY
df['USUBJID'] = df['USUBJID'].str.split(r'[\-]').str[3].str[2:5]
df = df.sort_values(by=['USUBJID', 'ADY'], ascending=[True, True])

# Step 8: Assign numeric codes for USUBJID
df['USUBJID_numeric'] = pd.Categorical(df['USUBJID']).codes

# Step 9: Sort and encode treatment groups
df = df.sort_values(by=['TRT01P'], ascending=True)
df['TRT01P_numeric'] = pd.Categorical(df['TRT01P']).codes

# Step 10: Filter out rows with missing max_ady values
df = df[df['max_ady'].notna()]

# Step 11: Add a prefix to TRT01P
df['TRT01P'] = 'TRT-' + df['TRT01P_numeric'].fillna('').astype(str)

df['DTHDY'] = df['DTHDY'] - df['max_ady']

# Step 12: Extract unique treatment groups
unique_records = df[['TRT01P', 'TRT01P_numeric']].drop_duplicates()
unique_trt = unique_records['TRT01P'].unique()

return df, list(unique_trt)
```

tempfile module

- When files are uploaded through a web application like Streamlit, they are initially stored as in-memory, file-like objects.
- the `pyreadstat.read_sas7bdat()` function requires a direct file path, not an in-memory object.
- The `tempfile` module addresses this by creating actual temporary files on disk, typically with a `.sas7bdat` extension.
- It writes the uploaded in-memory file data (e.g., `data1`, `data2`) into these newly created temporary files.
- Crucially, it provides the file paths (e.g., `temp1.name`, `temp2.name`), which are then used by `pyreadstat`.
- It keeps the file after closing, so pyreadstat can read it using its filename.
- This persistence prevents immediate deletion, which would otherwise occur, making the file unavailable for reading.

Step	What happens
Upload data1	Stored in memory
temp1.write(data1.read())	Saves to disk as temp file
pyreadstat.read_sas7bdat(temp1.name)	Reads from that saved file

Custom Plot Python Function (3)

- Function:

  plot_avalc_symbols(data, selected_trt, selected_avalc)

- Arguments:

  data: data frame containing preprocessed clinical trial data.

  selected_trt: List of the treatment group selected by the user

  selected_avalc: List of response values to select for plotting as markers

- It generates dynamic swimmer plots to visualize patient treatment duration, response, and off-treatment periods in oncology data.

```python
def plot_avalc_symbols(data, selected_trt, selected_avalc):
    # Define mappings for AVALC to markers and colors
    avalc_markers = {
        'Death': 's',
        'CR': 'p',  # Star filled
        'PR': '^',  # Triangle filled
        'SD': 'o',  # Circle filled
        'PD': 'D',  # Diamond filled

    }

    avalc_colors = {
        'CR': '#e377c2',
        'PR': 'blue',
        'SD': 'orange',
        'PD': 'magenta',
        'Death': 'red',
    }

    # Create the bar chart
    plt.figure(figsize=(10, 20))

    data = data[data['TRT01P'].isin(selected_trt)]

    # Example dataset (replace with your data's column of treatment groups)
    unique_treatments = data['TRT01P'].unique()

    # Define a set of colors (expand or change as needed)
    color_palette = plt.cm.Pastel1.colors  # Use a colormap (e.g., Pastel1, Set2, etc.)
    num_colors = len(color_palette)

    # Generate the treatment_colors dictionary dynamically
    treatment_colors = {
    treatment: color_palette[i % num_colors]  # Cycle through the color palette if treatments exceed colors
    for i, treatment in enumerate(unique_treatments)
    }

    # Assign colors based on the treatment group
    data['color'] = data['TRT01P'].map(treatment_colors)

    xupper = max(data['max_ady'].max(), data['DTHDY2'].max(skipna=True))

    # Dynamically adjust height based on the number of subjects
    num_subjects = data['USUBJID'].nunique()
    height = max(5, num_subjects * 0.5)  # Minimum height of 5, scaling factor of 0.5
```

matplotlib module

- **matplotlib.pyplot** (as **plt**)**:** Used to generate static, interactive, and animated plots.
- **st.pyplot(plt.gcf()):** This is essential for integrating your matplotlib plot into a Streamlit application. This displays the current matplotlib figure (`plt.gcf()`) in a Streamlit app.
- **plt.show():** This function is used in Jupyter notebook or script, to display a plot.

```
    legend_elements2.append(
    Patch(facecolor='white', edgecolor='black', hatch='\\\\', label='Off Treatment')
    )


# Update the y-axis ticks to show original USUBJID values
plt.yticks(data['USUBJID_numeric'], data['USUBJID'])

# Add labels and title
plt.xlabel('Days of Treatment')
plt.ylabel('Subjects')
# plt.title('Swimmer Plot for Treatment Exposure')
plt.margins(x=0, y=0.01)
ax = plt.gca()
legend1 = ax.legend(handles=legend_elements1, title='Response', loc='lower right')
ax.legend(handles=legend_elements2, title='Treatments', loc='upper right')

ax.add_artist(legend1)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.xlim(0, xupper+100)

# Display the plot in Streamlit
st.pyplot(plt.gcf())
```

Introduction to Streamlit

- Streamlit is a powerful Python library designed for building interactive web applications with ease.
- It offers similar functionality to R Shiny, enabling rapid development of data applications.
- Leverages a wide variety of existing Python packages and libraries.
- Allows you to publish web applications without the need to deploy a separate web server.
- Utilize Streamlit Cloud (**https://streamlit.io/cloud**) for convenient deployment and hosting.

Streamlit Installation

- Install Python on your PC
- Run following pip command
  - pip install streamlit

Function	Purpose
st.sidebar.radio()	Radio button to choose data source
st.sidebar.markdown()	Sidebar text for headers/spacing
st.sidebar.file_uploader()	Upload files via sidebar
st.sidebar.selectbox()	Dropdown to select pre-loaded files
st.markdown()	Display markdown in main app
st.sidebar.multiselect()	Select multiple options (e.g., treatment arms)
st.spinner()	Show loading spinner
st.warning()	Show warning message
st.container()	Group layout elements
st.info()	Show info message
st.pyplot()	Show matplotlib plot

Building the Web Application - Widgets

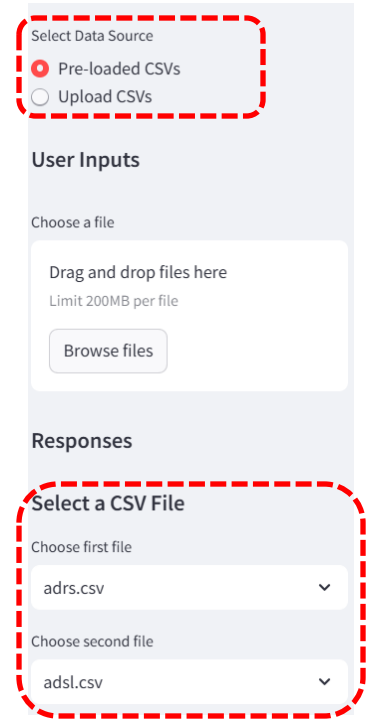
```
data_source = st.sidebar.radio("Select Data Source", ["Pre-loaded CSVs", "Upload CSVs"])
```

```
st.sidebar.markdown("## User Inputs")
st.sidebar.markdown("\n")
```

```
upload = st.sidebar.file_uploader('Choose a file', accept_multiple_files=True)
```

```
st.sidebar.markdown("\n ## Responses")
```

```
# Sidebar: Select CSV file
st.sidebar.markdown("## Select a CSV File")
selected_csv1 = st.sidebar.selectbox("Choose first file", csv_files, index=0)
selected_csv2 = st.sidebar.selectbox("Choose second file", csv_files, index=1)
```



Other Widgets

```
selected_trt = st.sidebar.multiselect("Select Treatment to Display", options=unique_trt, default=unique_trt)
```

```
# Filter the data based on selected treatments
df2 = processed_df[processed_df['TRT01P'].isin(selected_trt)]
```

```
# Recompute USUBJID_numeric dynamically based on the filtered data
df2['USUBJID_numeric'] = pd.Categorical(df2['USUBJID']).codes
```

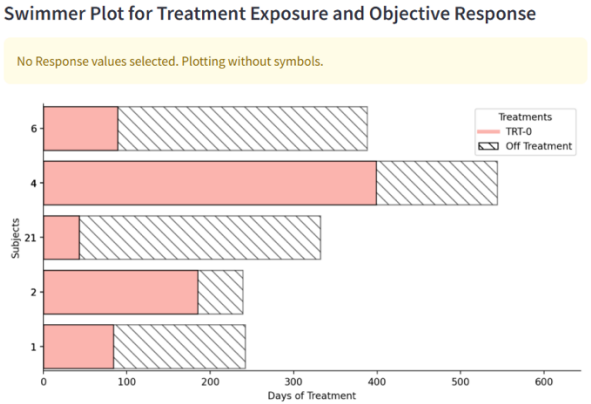
```
# Pass the updated DataFrame to the plot functions
selected_trt = df2['TRT01P'].unique()
selected_avalc = st.sidebar.multiselect("Select Responses to Highlight", options=['CR', 'PR', 'SD', 'PD', 'Death'], default=[])
```

Other Widgets

```
if selected_avalc:
    with st.spinner("Generating plot with symbols..."):

        plot_avalc_symbols(df2, selected_trt, selected_avalc)
else:
    st.warning("No Response values selected. Plotting without symbols.")
    with st.spinner("Generating plot..."):

        plot_(df2,selected_trt)
```



Steps to Deploy the App

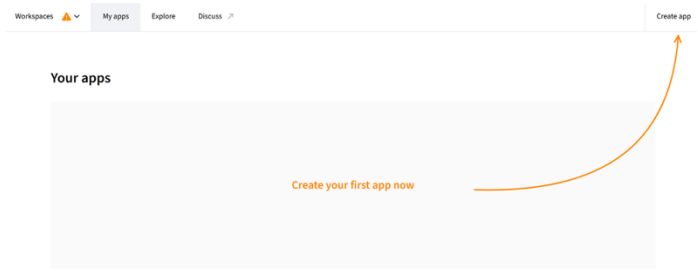
Create accounts- Streamlit cloud & GitHub

Create a Streamlit web application program on your GitHub repository

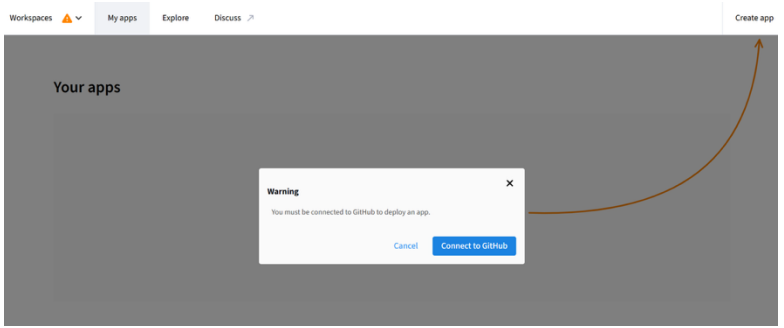
Deploy web application program on your GitHub repository

Steps to Deploy the App

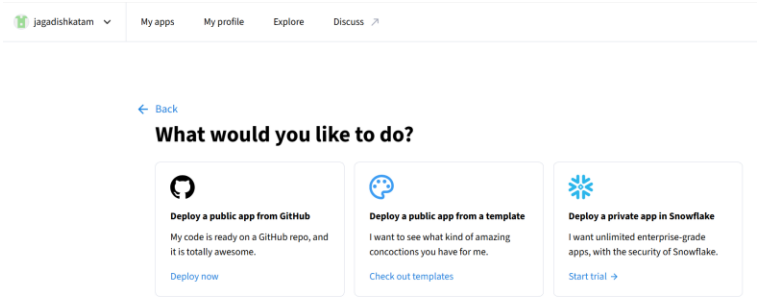
1



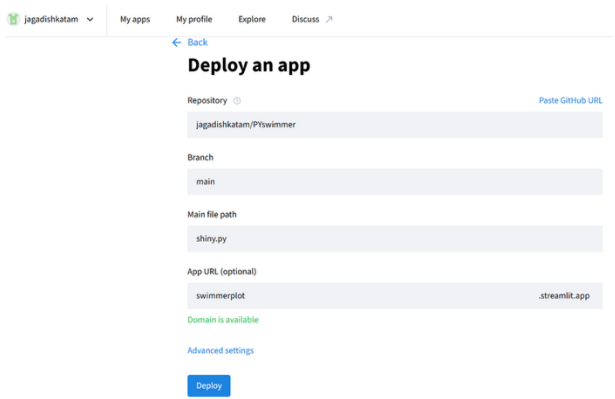
1



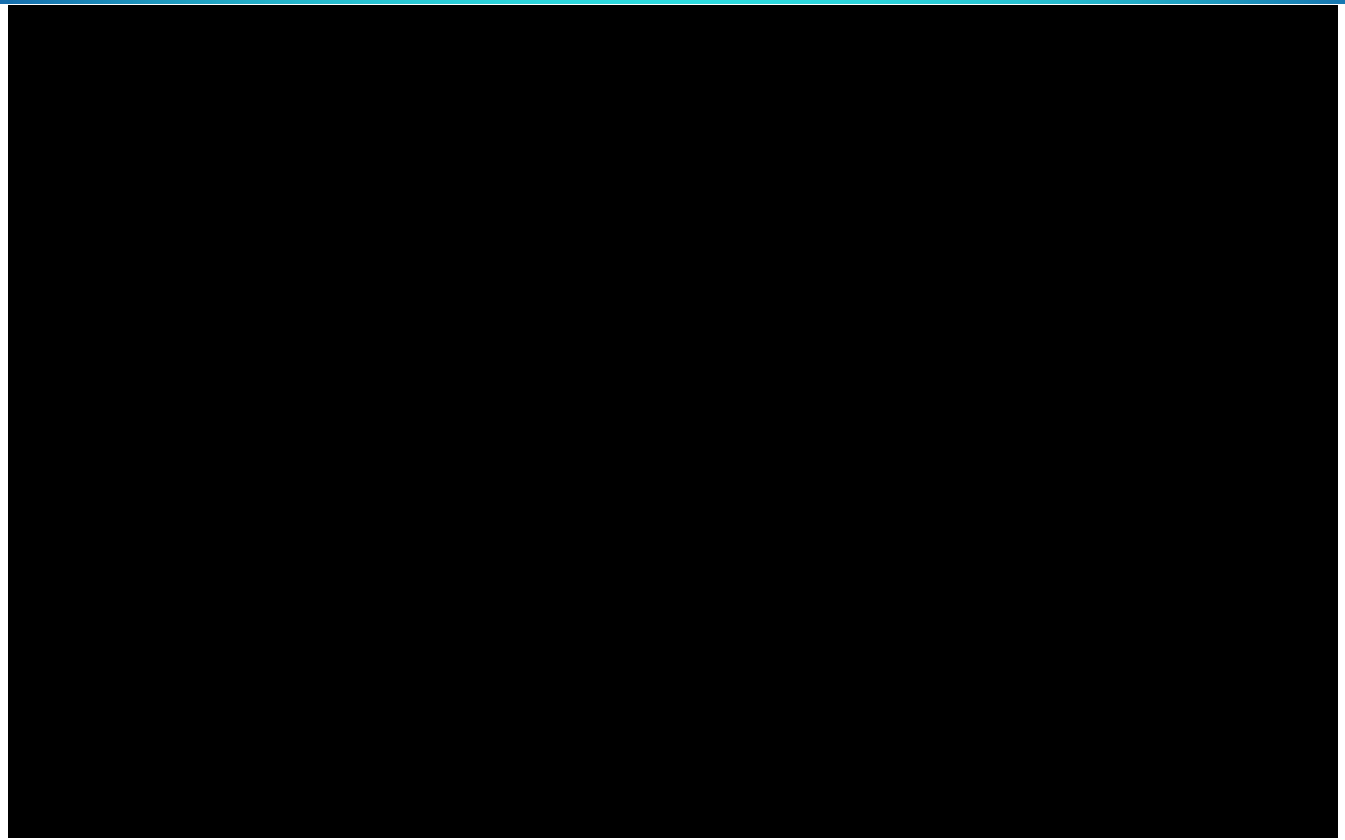
2



3



Demo



Conclusion

- Streamlit is a powerful Python library for creating interactive web applications.
- Unlike R Shiny, Streamlit integrates user interface definition and data processing within the same codebase.
- Streamlit web applications can be published in just three easy steps.
- When parameters are modified using widgets, the application automatically re-runs and regenerates the output.
- The Streamlit application provides a flexible and interactive way to visualize oncology response data, aiding in clinical trial analysis.

References

- Streamlit Official Website (https://streamlit.io)
- Sakaue, T. (2025). *Creating interactive web applications with Python: Streamlit*. Presented at PharmaSUG Japan 2025 (Paper 07). Chugai Pharmaceutical Co., Ltd. Retrieved from https://pharmasug.org/wp-content/uploads/2025/04/PharmaSUG-Japan-2025-07.pdf
- Matplotlib: Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55
- Gutierrez, S. (2023). Data science applications using Streamlit: Develop and deploy interactive web apps for data analysis and machine learning. Apress.
- Zinoviev, D. (2022). Data visualization with Python and Matplotlib. Franklin, Beedle & Associates.
- McKinney, W. (2022). Python for data analysis: Data wrangling with Pandas, NumPy, and Jupyter (3rd ed.). O'Reilly Media.

Thank you

Scan to access the app

phuse

The Global Healthcare Data Science Community

Contact Channels
UK +44 1843 609600
office@phuse.global
phuse.global

Social Media
@phusetwitta
/phusebook
/phusetube
/company/phuse