Google
Summer of Code

TARDIS!

Exploring supernovae made easy

Personal Information

Name: Swayam Shah
Email: swayamshah66@gmail.com
GitHub: Sonu0305
LinkedIn: swayamshahh
Resume: Swayam Shah
Current Country: India
Time Zone: Indian Standard Time (GMT + 5:30)

Background Information

Why You?

I'm currently pursuing a *B.Tech* in *Computer Engineering* at *VJTI*, where I've developed my skills in languages like `C++` and `Python`. I thrive on **problem-solving**, regularly solving challenges on LeetCode, which has sharpened my analytical abilities. As an **active** open-source contributor, I've built a strong track record with TARDIS, making healthy **contributions** to both the tardis and stardis repositories. This experience has **familiarized** me with the codebase, mentors, and org admins, giving me a solid foundation to tackle the visualization module project. My technical expertise, passion for coding, and commitment to TARDIS make me an ideal fit to deliver impactful, user-focused solutions through this **GSoC opportunity**.

Past Contributions

- ➔ **TARDIS**:
  a. *PRs*: 2922, 2923, 2932, 2938, 2946, 2951, 2956, 2959, First Objective
  b. *Issues*: 2590, 2755, 2765, 2903, 2937, 2950, 2955, 2960, 2979

- ➔ **STARDIS:**
  a. *PRs*: 239, 241, 244, 250
  b. *Issues*: 238, 240, 243

- ➔ **CARSUS:**
  a. *PRs*: 445

Why Us?

The TARDIS visualization module project **aligns perfectly** with my passion for scientific computing and interactive data exploration. As a **contributor** to TARDIS since late 2024, I've grown to love its mission to advance supernova research. This project excites me because it **matches my skills** in Python, Panel, and visualization tools while serving a **vital purpose** for users. Migrating widgets like CustomAbundanceWidget, ShellInfoWidget, LineInfoWidget, and GrotrianWidget to Panel and embedding them on GitHub Pages will empower astronomers with accessible, interactive tools to analyze simulations efficiently. I'm **committed** to TARDIS, finding joy in tackling its challenges, and I aspire to remain a **permanent contributor**, enhancing its tools and community beyond GSoC.

Project Information

Rewrite the TARDIS visualisation module using Panel

Brief project summary

The **TARDIS** project offers a suite of visualization tools and widgets that enable users to interactively explore supernova simulations within Jupyter Notebooks. However, these tools currently rely on dependencies like `IPython` and `Qgrid`, which are **incompatible** with Sphinx documentation. This limitation prevents the integration of interactive visualizations directly into the TARDIS website, and reduces their accessibility to users who do not run Jupyter locally. To address this, this project proposes **rewriting** the **TARDIS visualization module using Panel**, a Python library designed for creating interactive dashboards.

I have **already completed** the first objective of this project: implementing a **Panel-based** visualization of hierarchical datasets using two interlinked tables. In this implementation, the first table displays shell data from TARDIS' SimulationShellInfo, and selecting a row **updates** the second table to show corresponding element abundances. This **achievement** validates the potential of Panel for this migration and provides a solid starting point for further development. Further development in this project involves: **fully migrating** all existing visualization modules to Panel, **eliminating** any reliance **on** `qgrid`/`qgridnext`/`IPython`, **embedding** the Panel-based tools into the **TARDIS documentation**, writing tests for the changes, and lastly, updating the documentation.

In essence, this project will leverage Panel to transform TARDIS's visualization capabilities into a web-accessible platform. By building on my initial **success** with the interlinked tables, I aim to deliver a cohesive, interactive, and **well-documented** solution that showcases the power of Python-based scientific visualization tools.

Detailed project description

This project aims to fully migrate these visualization modules to use **Panel**, a versatile Python library for building interactive dashboards, **eliminating** problematic **dependencies** like `Qgrid`/`Qgridnext` and enabling embeddable, **user-friendly** widgets in the documentation.

As mentioned above, I have completed the **First Objective PR** for this project idea, so I would like to **dive deeper** into what changes I exactly incorporated to migrate to panel and ensure it works as well as before. The major changes for migrating **SimulationShellInfo** widget to panel are made to the files: *tardis\tardis\visualization\widgets\shell_info.py* & *tardis\tardis\visualization\widgets\util.py*, following is the detailed description of my changes in these files:

Note: **Panel version** which I have used is **1.6.1**

```
(tardis) swayam-shah@Swayam:/mnt/c/Users/admin/Desktop/tardis$ conda list panel
# packages in environment at /home/swayam-shah/miniconda3:
#
```

```
# Name                      Version                    Build  Channel
panel                       1.6.1               pyhd8ed1ab_0    conda-forge
```

1. **Main** helper function `create_table_widget_shell_info()`:

Documentation Reference: panel.widgets.Tabulator

```
    tabulator_options = {
        'layout': 'fit_data_fill',
        'pagination': None,
        'selectable': 1,
    }
```

Modified to use panel.widgets.Tabulator along with its available appropriate options

```
    num_rows = data.shape[0]
    if num_rows > 20:
        tabulator_options['height'] = 550
    else:
        tabulator_options['height'] = None
```

Ensuring the visibility of tables, by making it scrollable if height exceeds a certain value set to ~20 rows

```
custom_css = """
.tabulator-header, .tabulator-col-title {
    height: auto; min-height: 40px; white-space:
normal; overflow: visible !important; text-overflow:
clip;
}
.tabulator-col-title { padding: 4px; }
.tabulator-tableholder { overflow-y: auto !important; }
"""
```

Added custom-css styles to retain the perfect appearance of table widgets

```
    return pn.widgets.Tabulator(
        data,
        **tabulator_options,
        widths=widths,
        stylesheets=[
            ":host {--mdc-ripple-color:
transparent;}",
            Custom_css ] )
```

Returns the panel based widgets using Tabulator along with its well defined parameters: data (shell-info), table-widths, styles;

2. `class TableSummaryLabel: _create()`

Documentation Reference: panel.Row & panel.pane.HTML
Link to My Code: Code
Description of Changes:
Used the **Row Layout** which allows arranging multiple panel objects in a **horizontal container**, and the layout contains the **HTML pane** which allows rendering HTML in a panel

3. `class ShellInfoWidget: __init__()`

Link to My Code: Code
Description of Changes:
.df to ***.value*** was necessary because self.shells_table **transitioned** from a qgrid widget (with *.df*) to a panel widget (with *.value*)

Similarly, this change is implemented for the ion_count (Code) & level_count (Code) table, too.

4. `class ShellInfoWidget: update_element_count_table()`

Documentation Reference: .selection
Link to My Code: Code
Description of Changes:
The qgrid_widget **parameter** was removed since **panel** relies on direct property access rather than an explicit widget reference, and event["new"] was replaced with self.shells_table.selection to leverage **panel's selection** list instead of qgrid's event dictionary.

Similarly, this change is implemented for the update method for ion_count (Code) & level_count (Code) tables with their **respective** nested updation logic.

5. `class ShellInfoWidget: display()`

Documentation Reference: panel.Tabulator(attributes) & param.watch & panel.Row & panel.Column
Link to My Code: Code
Description of Changes:
1. **panel's Tabulator** expects width as pixel values (integers) for precise control in web layouts.
2. panel uses ***param.watch*** for reactive event binding on widget properties like selection

3. Used **panel's Row** for flexible web layout component with CSS styles as a dict
4. Used **panel's Column** since it provides vertical stacking for web layouts.

Above provided **code-snippets** & **code-links** (from my First Objective PR) include the migration of *shell_info* widget to panel along with signifying why and how panel was used and **why it is helpful** at respective places in the function.

I would further like to focus on the remaining visualization modules, out of which I want to propose **potential** changes for migration of the **LineInfoWidget** module to panel.

1. `class LineInfoWidget: __init__()` in `line_info.py`

Documentation Reference: panel.widgets.ToggleGroup

Before:	**After:**

`self.filter_mode_buttons = ipw.ToggleButtons(`
 	
`options=self.FILTER_MODES_DESC,`
`index=0`
`)`
	`self.filter_mode_buttons = pn.widgets.ToggleGroup(`
	
	`options=self.FILTER_MODES_DESC,`
	`value=self.FILTER_MODES_DESC[0],`
	`button_type='primary'`
	`)`

Documentation Reference: panel.widgets.Select

Before:	**After:**

`self.group_mode_dropdown = ipw.Dropdown(`
 	
`options=self.GROUP_MODES_DESC,`
`index=0`
`)`
	`self.group_mode_dropdown = pn.widgets.Select(`
	
	`options=self.GROUP_MODES_DESC,`
	`value=self.GROUP_MODES_DESC[0]`
	`)`

2. `class LineInfoWidget: get_species_interactions()` in `line_info.py`

Before:

```
            else:  # No species could be selected in specified
wavelength_range
```

```
                # qgrid cannot show empty dataframe properly,
                # so create one row with empty strings
                selected_species_symbols = [""]
                fractional_species_interactions = pd.Series([""])

        else:  # wavelength_range is None
            selected_species_symbols = [""]
            fractional_species_interactions = pd.Series([""])

        fractional_species_interactions.index = pd.Index(
            selected_species_symbols, name="Species"
        )
        fractional_species_interactions.name = "Fraction of packets
interacting"
```

After:

```
fractional_species_interactions.index = pd.Index(
                selected_species_symbols, name="Species"
            )
            fractional_species_interactions.name = "Fraction of
packets interacting"
            return
fractional_species_interactions.sort_values(ascending=False).to_frame
()

        else:  # No species could be selected in specified
wavelength_range
            return pd.DataFrame(
                columns=["Species", "Fraction of packets
interacting"]
            )

    else:  # wavelength_range is None
        return pd.DataFrame(
            columns=["Species", "Fraction of
packets interacting"]
        )
```

panel's Tabulator renders empty DataFrames correctly, hence simplifying the logic, whereas qgrid required a non-empty DataFrame for display

Similarly, the above mentioned **workaround** can be simplified even in the `class LineInfoWidget: get_last_line_counts()`.
Likewise, I plan to **systematically** migrate the entire TARDIS visualization module to Panel by addressing individual functions and components in an **organized** manner. The proposed changes above don't encompass the full module but demonstrate specific migration steps, serving as examples of how other modules and updates **will be approached**.

Embedding Widgets

To tackle the most **challenging** task about embedding the interactive widget & tools on the **TARDIS docs**, I would like to try to implement this feature by using either panel's Embedding into sphinx documentation or panel's functionality of converting panel applications. I prefer to go with **approach 1**, since it is the exact application of what TARDIS wants to set up for their users. **Approach 2** is more inclined towards deploying dashboards and similar applications.

Approach 1:
Documentation Reference: nbsite.pyodide

Key Benefits:
1. Seamless Integration into Sphinx Documentation
2. Configurable Options i.e. customizing the '`nbsite_pyodide_conf`' dictionary.

To begin, firstly we install the `nbsite` through `pyviz` channel into our environment.

```
conda install -c pyviz nbsite
```

Next, we add the '`nbsite.pyodide`' to the extensions list in our Sphinx docs *conf.py.*

```
extensions += [
    ...,
    'nbsite.pyodide'
]
```

This step enables the directive for use in the reStructuredText (.rst) files that we use in our documentation.

Additionally, we can also customize the behavior through the '`nbsite_pyodide_conf`' dictionary, for example:

```
nbsite_pyodide_conf = {
    "PYODIDE_URL":
"https://cdn.jsdelivr.net/pyodide/v0.19.0/full/pyodide.js",
    "autodetect_deps": True,
    "enable_pwa": True,
    "requirements": ['panel', 'numpy', 'matplotlib'],
    "scripts": ["custom_scripts."],
    "setup_code": ''' '''
}
```

a. *PYODIDE_URL*: URL to fetch Pyodide from
b. *autodetect_deps*: Automatically detects dependencies in executed code
c. *enable_pwa*: Adds web manifest and service worker for PWA support
d. *requirements*: Default requirements to include, e.g., panel, matplotlib, numpy, etc.
e. *scripts*: Scripts to add on first Pyodide cell execution
f. *setup_code*: Python code to run when initializing Pyodide runtime

Next, in our .rst files where want pyodide to be used, we use it this way, for example:

```
.. pyodide::

    import panel as pn
    import numpy as np
    x = np.linspace(0, 10, 100)
    y = np.sin(x)
    plot = pn.pane.Matplotlib(plt.figure(figsize=(5, 4)))
    plot.object = plt.plot(x, y)
    plot
```

This directive (`.. pyodide::`) leverages Pyodide's ability to run Python in the browser, ensuring no backend server is needed, aligning with the goal of static hosting on platforms like GitHub Pages

Approach 2:
Reference Article: Dynamic GitHub Pages – Panel (pyodide-worker)
Documentation Reference: panel convert

Key Benefits:
1. Static hosting on GitHub Pages.

2. Retains interactivity (e.g., table row selections, updates).

For example:

1. Convert to HTML

```
panel convert shell_info_app.py --to pyodide-worker --out
docs/visualization
```

- **a.** *panel convert*: script conversion panel package command
- **b.** *shell_info_app.py*: script to be converted
- **c.** *--to pyodide-worker*: generates an HTML file and a JS file containing a Web Worker that runs in a separate thread
- **d.** -- *out docs*: output folder for the 2 files (HTML and JavaScript) generated

2. The Github Pages will automatically deploy the files to existing docs, and verify the interactivity for the widgets.

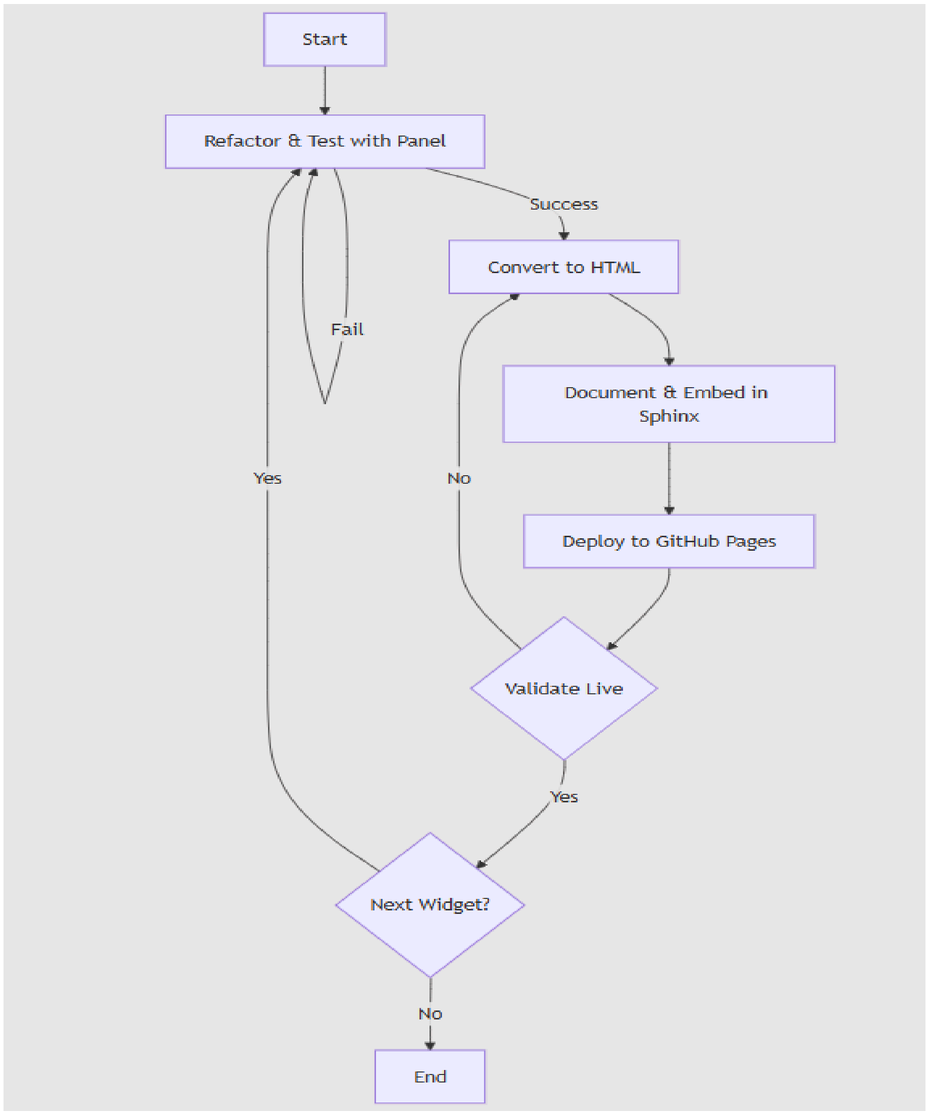
Deliverables

1. **Migration** of TARDIS Visualization Module to Panel:
   Goal: Refactor TARDIS visualization model to use Panel exclusively.

2. **Interactive Widgets** Hosted on **GitHub Pages**:
   Goal: Convert and deploy interactive visualization tools as standalone HTML files using panel convert, hosted on tardis-sn.github.io.

3. **Dependency Cleanup**:
   Goal: Eliminate all uses of Qgrid/Qgridnext from the TARDIS visualization codebase.

Development Workflow



Development Schedule

Up till May 8	**Proposal accepted or rejected** ☑ **Familiarise** myself **more** with the codebase ☑ Discuss the project idea in **detail** with the mentors
May 8 - June 1	**Community Bonding Period** ☑ Set up the latest TARDIS development environment on my machine ☑ Prepare a **detailed plan**: prioritize widgets & tools, identify **challenges**

Week 1 & 2	**Coding Officially Begins!** ☑ Migrate **ShellInfoWidget** to panel and test it (ensure shell data interactivity remains intact) ☑ Convert the migrated **ShellInfoWidget** to HTML using panel convert ☑ Deploy and **test it** on the TARDIS docs (GitHub pages)
Week 3 & 4	☑ Migrate **LineInfoWidget** to panel and test it (ensure the spectrum interactivity remains intact) ☑ Write/Update documentation for the migrated **ShellInfoWidget** (along with any screenshots if required) ☑ Convert **LineInfoWidget**, deploy to GitHub Pages and **test it** on the TARDIS docs.
Week 5	**Buffer Period** ☑ **Fix any issues** from previous work, or complete any task mentioned before if not completed ☑ **Review** and talk about the progress with mentors to adjust the **remaining** work
Week 6 & 7	☑ Migrate **CustomAbundanceWidget** to panel and test it (ensure the table & other interactive features remain intact) ☑ Convert **CustomAbundanceWidget** and deploy to GitHub pages, also **test it** on TARDIS docs ☑ Start writing tests for the work done (i.e. test table updates, edge cases, selection updates, etc.)
Week 8 & 9	☑ Migrate **GrotrianWidget** to panel and test it (ensure the energy level interactive features remain intact) ☑ Convert **GrotrianWidget** and deploy to GitHub pages, also **test it** on TARDIS docs ☑ Complete the remaining tests and ensure they work well with the GitHub Actions
Week 10	**Buffer Period** ☑ **Resolve any bugs** which are remaining to be fixed ☑ Analyse any dependency on Qgrid/Qgridnext which is **yet to be eliminated** ☑ Analyse the whole TARDIS visualization module to **check** whether any part requires migration ☑ **Confirm** deliverables **with mentors**

Week 11 & 12	☑ **Finalize** all the sections of the TARDIS docs, and complete any documentation & tests remaining ☑ **Organise** the GitHub Pages deployment for **intuitive** user experience ☑ Prepare **final submission**: code, docs, test reports, and the **final blog post**
Until Pens Down & Post GsoC	☑ Address last-minute **mentor feedback** ☑ Wrap up the Final Submission **well before deadline** ☑ Continue to develop & improve the new support for interactive widgets for users ☑ **Continue contributing** to TARDIS & the project in general

My Availability & Other Commitments

1. Once coding begins, I will work dedicatedly for at least **4-5 hours** on **weekdays** (and **6-8 hours** on **weekends**) until the completion of the project. After the submission of my proposal (on or before **April 8**), I will start **researching** more about the widgets and start noting down important aspects of it.
2. During the start of the community bonding period, I have my end-semester **examinations** which will start from **May 5 until May 17**. However, I promise to begin my work as early as possible to be on track. After the examinations, I shall put in **more time** to make up for any lost time or lag.
3. I have a **2-month** long **vacation** after **May 17** and no other commitments in hand, so I will be able to devote ample time to the project.