Easily Edit DataFrames within JupyterLab

Use Qgrid to simplify data cleaning and track your changes

Pandas is powerful and very flexible, but it does have some weaknesses. One big point of friction I ran into is the inability to double-click on a cell and change its value. Below I will outline the context of my problem, explain how I customized my JupyterLab notebook to improve my workflow, and walk through the code required to create this interface.

# Background

I started learning Python nearly a year ago, after a friend showed me some really interesting analyses he did with Python for his dissertation on the role of genre in popular music <<https://tom-johnson.net/2017/10/09/genre-part-2-genre-tags/>>. After reading some books and going through MOOCs, I needed to find a personal project to *really* digest the concepts. I decided to work on a college football computer poll/model, as I’ve been a college football fan since my freshman year, and my domain knowledge would help me find nonsensical results from coding errors.

My main data source is collegefootballdata.com <<https://collegefootballdata.com>>. The website’s owner gathers a wide range of data from ESPN and other sources, including play-by-play data for each game, drive summaries, matchup information, game location, recruiting rankings, and conference affiliation. It is an invaluable repository for anyone interested in exploring college football data. There is one big issue, however: ESPN’s data sometimes <<http://www.espn.com/college-football/playbyplay?gameId=401012797&wsVar=us~ncf~gamepackage,desktop,en>> have <<http://www.espn.com/college-football/playbyplay?gameId=401019511&wsVar=us~ncf~gamepackage,desktop,en>> problems <<http://www.espn.com/college-football/playbyplay?gameId=401012776&wsVar=us~ncf~gamepackage,desktop,en>>.

One of the main measures I want to derive and model is how a team performs while they are actively trying to win with their best players and best tactics. To do this, I need filter out any garbage time <<https://en.wikipedia.org/wiki/Garbage_time>> plays. I am using college football analytics guru Bill Connelly’s definition <<https://www.footballstudyhall.com/2017/10/20/16507348/college-football-analytics-game-states>> of garbage time, defined as point differentials of 36 in the second quarter, 26 in the third quarter, and 20 in the fourth quarter. However, in order to properly filter out garbage time and get accurate measurements of each team, I need to start with correct statistics.

# Friction in Data Cleaning

I am using CSV files of play-by-play data downloaded from [collegefootballdata.com](http://collegefootballdata.com) as my raw data source. I noticed pretty quickly that there are a number of impossible, mislabeled, and flat-out incorrect plays that I need to fix in order to be satisfied with the results of this project. Additionally, I want to keep a record of which plays I changed to send back to the website so the community as a whole has cleaner data for more accurate analysis. My original workflow for fixing the data was:

* identify and isolate rows that met specific criteria with Pandas
* export the rows to a CSV file and drop them from the DataFrame
* open the CSV in Excel
* find play-by-play data from a different source (usually the school’s athletics website)
* fix the incorrect information
* import the CSV
* merge the two DataFrames

There’s a lot of friction here. Pandas lets you filter data in pretty much any way imaginable, but you need to type all of your criteria. I longed for an interactive way to sort, filter, and edit data directly inside a JupyterLab notebook. I found exactly that when I stumbled across Parul Pandey’s terrific Medium post about enhancing Jupyter notebooks <<https://towardsdatascience.com/bringing-the-best-out-of-jupyter-notebooks-for-data-science-f0871519ca29>>.

# New Workflow

I dove into Qgrid’s documentation and GitHub page as soon as I saw this gif <[https://cdn-images-1.medium.com/max/800/1\*xGvLi1RhdIdSbCA9cPsS2A.gif](https://cdn-images-1.medium.com/max/800/1*xGvLi1RhdIdSbCA9cPsS2A.gif)>. Qgrid’s GitHub page shows off additional functionality you can get with the Qgrid JupyterLab extension and Output widgets from ipywidgets <<https://ipywidgets.readthedocs.io/en/stable/>>.

Qgrid showed me something I didn’t know was possible: the ability to display the entirety of a selected row in a separate output view. The DataFrame I’m working with is 24 columns, one of which contains long strings describing each play. Here’s an example:

Kellen Mond pass complete to Quartney Davis for 17 yds for a TD Timeout TEXAS A&M, clock 00:00 LSU Penalty, Defensive Pass Interference (2 Yards) to the LSU 1 LSU Penalty, Unsportsmanlike Conduct ( Yards) to the LSU 1 TEXAS A&M Penalty, False Start (-4 Yards) to the LSU 5 (Kellen Mond pass to Kendrick Rogers for Two-Point Conversion)

A separate view that shows an entire row would save me a lot of scrolling and keep relevant information in a consistent location in the notebook.

After tackling the APIs for Qgrid and ipywidgets, I created the interface pictured below.

<insert link to picture>

I’d like to point out some key features. The right `Current Row` output view displays the row as it exists when you first select it. I changed the yards\_gained and the play\_text columns in row 530, but the changes are not yet reflected in the output view. They do appear in the output view if I select a different row and reselect row 530.

`Current Row` is capable of displaying more than one row at a time, but it displays the last selected row first. So if you want to display three consecutive rows in the same order, select the rows from the bottom up.

The bottom `Edits` output view shows the coordinates of the edited cell (name of the DataFrame you edited followed by [row name, column name]), the old value, and the new value. This widget keeps track of changes you make in any Qgrid DataFrame in the notebook and reflects this in the coordinates.

The edits are stored in a DataFrame. You can export them to a CSV file by typing a name for the file in the text box and clicking the ‘Export to CSV’ button. The file will export to the directory the notebook is in. Exporting does not automatically clear edits; click the ‘Clear edits’ button to do that.

# Code

The general steps to building these interactive widgets are:

* 1. Instantiate the widgets you want to use from the ipywidgets package
  2. Write functions to execute when certain events happen
  3. Write event handlers that link the functions to events

I’ll walk through the specific implementation of these steps in the current row and edits widgets.

First, start with the proper imports. Pandas display options are used by the `Current Row`. These settings are appropriate for the data I’m working on, customize as necessary for your projects.

<script src="https://gist.github.com/likethebourbon/f6932d45b16bdb4604098089c9b53081.js"></script>

**Step 1**

<script src="https://gist.github.com/likethebourbon/6c143725fde8d461995a008602f21f2c.js"></script>

Create the output widget that will display the currently selected row. `Current Row` is an Output widget with some layout options specified. The `overflow` options allow for scrolling within the widget if necessary. I am not sure if all three options (`overflow\_x`, `overflow\_y`, and `overflow`) are necessary, but after running into bugs and playing around with the settings, this is what gave me the behavior I wanted.

<script src="https://gist.github.com/likethebourbon/3daa18381f57dfa4f82720b54bb08ddd.js"></script>

With Step 1 complete for `Current Row`, let’s do the same with `Edits`. This one is more complicated, as there are five components:

* a DataFrame that stores all of the edits
* an Output widget that displays the edits
* a textbox for entering the name of the file for export
* a button that exports the edits
* and a button that clears the output display and deletes all stored edits

We need to link these components together before bringing Qgrid into the fold, which means following the three general steps listed above on a lower level.

1. Instantiate widgets: Create the DataFrame to hold the edits, and create instances of the four widgets. As you can see, the widgets are all pretty simple.
2. Write functions: the buttons need functions telling them what to do. Each function links the button to other components of this compound widget:
   1. The `export\_edits` function links the button to the text box and the Edits DataFrame
   2. The `clear\_edits` function links the button to the Output widget and the Edits DataFrame
3. Link the functions to events: use the `on\_click()` method to tell the buttons to execute the functions you wrote.

Now all that’s left to do is organize the components and display them. Use `widgets.HBox()` to put the text box and buttons in a row, and `widgets.VBox()` to put that row above the Output widget.

**Step 2**

Now that you’ve made the widgets, you need to tell Qgrid how to use them. Handler functions always take the arguments `event` and `qgrid\_widget`.

<script src="https://gist.github.com/likethebourbon/55610715a4521b9c6ad9e827d8ff506b.js"></script>

Current Row

`output\_area` tells Qgrid where to display the selected row. Using a `with` statement redirects `stdout` to the specified output area. The code block beneath `with` displays the currently selected row by calling the `get\_selected\_df()` method on the `qgrid\_widget` passed to the function. Call `display()` rather than `print()` to present the row with Jupyter’s rich display logic, and transpose the row with `.T` since the Output View is vertical. Finally, since the point of this widget is to display only the current row, clear old rows from the Output widget by calling the `clear\_output()` method. `wait=True` ensures that the Output widget is cleared only when new output is available.

Edits

Like above, specify the `output\_area` and use a `with` statement to redirect `stdout`. The code block adds the variable name of the edited DataFrame to the Qgrid `event` dictionary under the `’data’` key, and uses the values associated with `’data’`, `‘index’`, `‘column’`, `‘old’`, and ‘new’` keys to print a concise summary of the edit to the `edited\_cells` Output widget. After that, those same values are stored in the `edits` DataFrame for easy exporting.

**Step 3**

The last step to pulling this all together is linking these functions to specific events. This is accomplished with Qgrid’s `on()` method, which takes two arguments: the event to link the handler to, and the function that executes when said event happens.

That’s it! Test it out by reading a CSV file into a DataFrame, and instantiating a QgridWidget.

<script src="https://gist.github.com/likethebourbon/540e98620ec544158e734313dacf3233.js"></script>

Feel free to clone my GitHub repo and play around with it. `editing-workflow.ipynb` contains all the code in this post. I also included Week 9 from the 2018 College Football Season as a sample dataset. Comment below with any questions or suggestions!