DataFrameViewer: Getting Started Guide

Jack Gillett

August 19, 2018

1 Introduction

Pandas, numpy, scikit-learn and a small family of other libraries have become the 'killer app' for scientific computing within python, with pandas dataframes as the basic building blocks for most of modern data science.

One limitation is that dataframes can be large and unwieldy, and aside from viewing the top few rows and columns using `head()` it can be difficult to quickly see what your data looks like. DataFrameViewer is a data exploration and visualisation tool that aims to provide a GUI representation of an underlying dataframe, and allow application of filters, pivots and sorts in the GUI layer to quickly determine the underlying structure of your dataset.

2 Getting Started

DataFrameViewer is really easy to get started with:

1. Clone the repo from Github

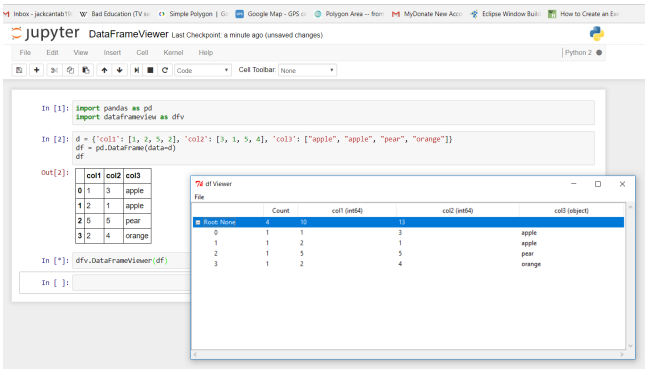

Figure 1: A very simple dataframe, visualised!

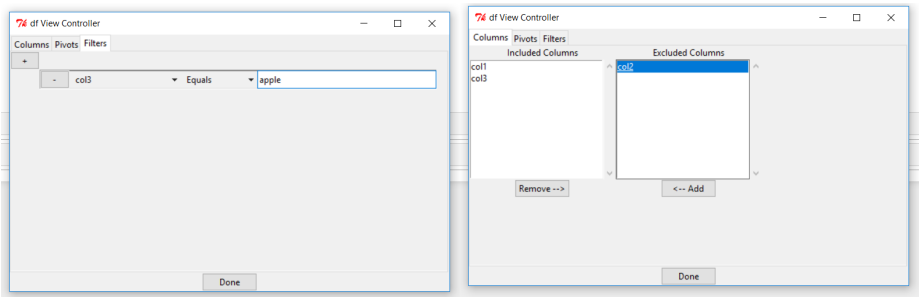

Figure 2: The window controlling the filterset to be applied to the dataset and the visible columns in the view

2. In cmd, navigate to the folder containing setup.py and install via `$ python setup.py install`
3. The module can now be installled via pip, `$ pip install dataframeview`
4. Import the module into your code, using `import dataframeview as dfv`
5. Define a Pandas dataframe as usual
6. Call `dfv.DataFrameViewer(df)` function on the dataframe as in Fig 1, and bingo!

As well as viewing the whole of the dataframe, you can control the visibility of columns and add filters and pivots via the GUI in the ViewController option in the File menu, to rapidly explore the data. By clicking on the column titles once or twice the viewer will automatically sort or reverse sort on that column, which is an incredibly powerful tool for finding unexpected Null values in your dataset.

Fig 2 shows the procedure for filtering out rows from the view based on the values in particular columns and how to control the columns visible when viewing larger datasets. The resulting view from the simple table we saw below once these changes are made is shown in Fig. 3.

3 Adding View Elements Programatically

In addition to filtering, a very important concept when dealing with tabular data is the pivot or groupby operation, which splits the data into several subsets on the basis of the values of one of their features. This allows us to rapidly visualise the quantity of data in each subset, and by viewing the aggregate of the other features of the underlying rows we can rapidly identify high-level differences in

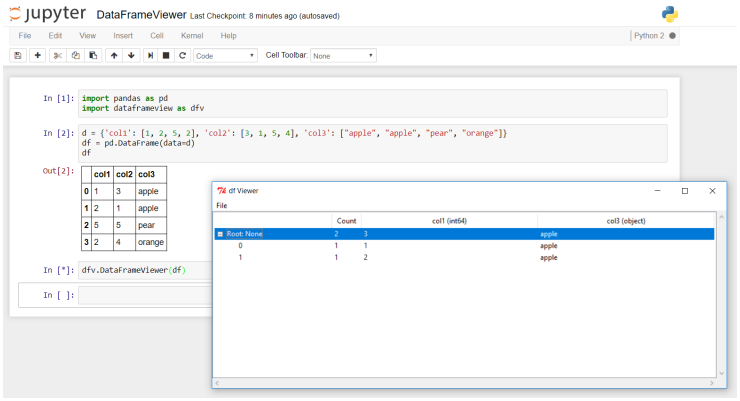

Figure 3: The first dataset we saw, once the filter and column visibility changes shown above are applied

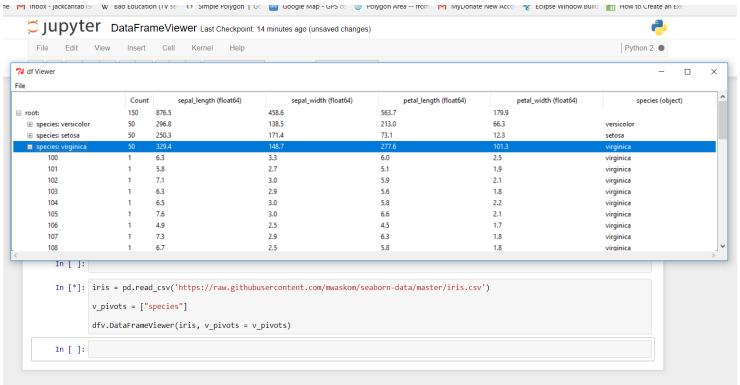

Figure 4: The Iris dataset, pivoted by the species type

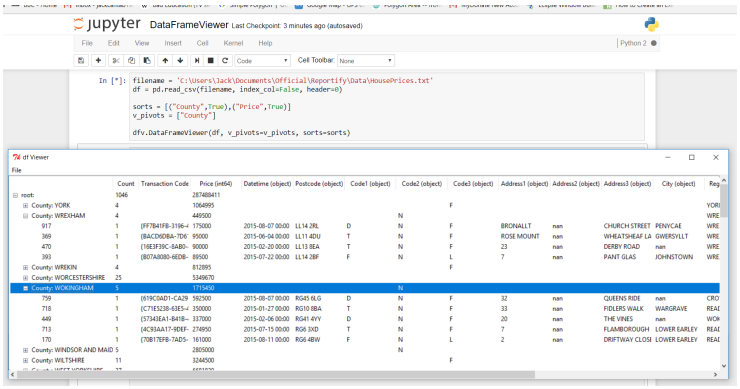

Figure 5: A very simple dataframe visualised

the underlying data for each subset. Fig. 4 shows a simple example of how this is achieved by DataFrameView.

Of course, to be useful for real datasets and in real programming environments you will often want to control the view settings programmatically. The example above shows demonstrates this, pivoting on species to aggregate all of the sub-rows of a given species into buckets

You can also control the sorting behaviour programmatically - in Fig. 5 we pivot another dataset by the county field, and then reverse sort by county and sort by price ascending for records with the same county programmatically. Note that the list of sorts requires tuples, specifying the direction of the sort as well as the column. If you wish to sort by multiple columns, you'll need to do this programmatically at present (a sort tab is coming soon to the view controller).

4 Coming Soon...

1. more filter types for different datatypes (eg. 'string contains')
2. controllable column aggregations (averages, weighted sums, elements -¿ list)
3. a nicer-looking GUI
4. selectable copy-paste
5. save/load views, save/load view+dataframe
6. export to excel/email/text
7. better performance for large datasets

8. horizontal pivoting!
9. much, much more...