CS 340 Dashboard Sample Walkthrough

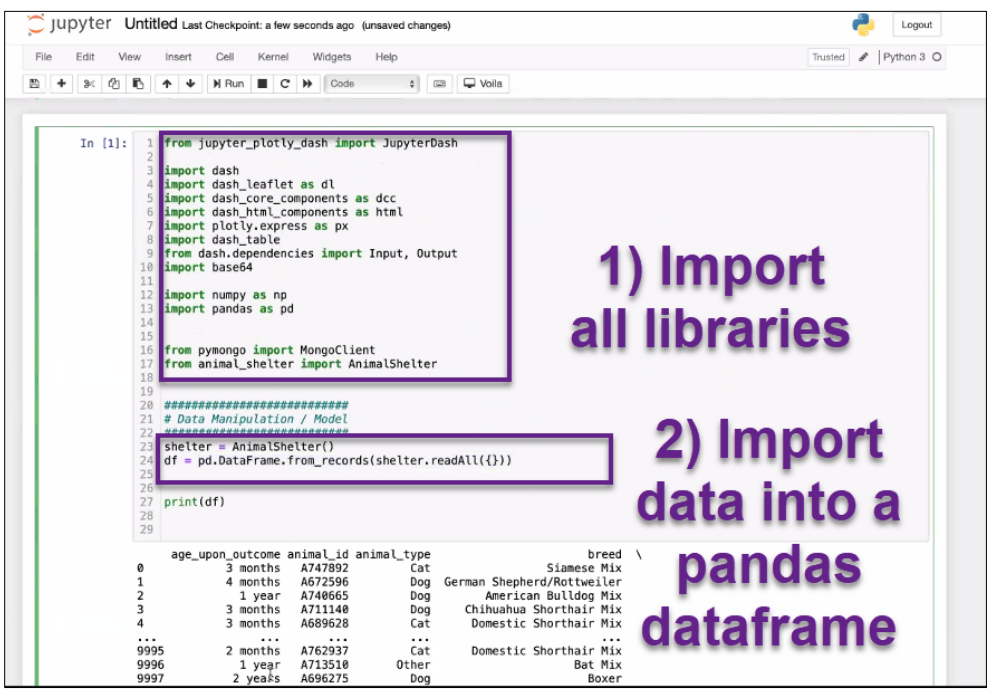
There are many different ways to use the Dash framework. This walkthrough will take you step-by-step through the process of creating a dashboard with data in MongoDB and a CRUD Python module. This dashboard will be similar to the one you will need to create in your assignments and projects.

Important note: The line numbers in the screenshots may vary slightly from the sample code.

Importing Data From MongoDB Using CRUD Python Module

As always, before writing your code, make sure to import all of the libraries that you will need in your project. These import statements are included in the starter notebooks for your assignments, but normally you would find them in the documentation for any framework or libraries you would like to use.

We first use the CRUD Python module to import data from MongoDB into a Pandas dataframe. Pandas has good "glue" functionality to translate data between MongoDB and other libraries. We then run the notebook to test and make sure that the data imported correctly. Notice that in the output underneath the cell, we can see records from the data set. The formatting is not easy to read and it is plain text. However, this important test ensures that our CRUD Python module is working properly. It is important to build out large projects in small pieces at a time, testing as we go to ensure that each component is working before incorporating the next element.



Note: In this walkthrough, the CRUD Python module is titled **animal_shelter** and imported as **AnimalShelter.** When writing your own code, you will need to use the title of *your* CRUD Python module.

Simple Dashboard Layout

Now that we know our data is importing correctly, we can begin using the Dash framework to create the web application, or client-side code. First, we will start by building the **layout** of the dashboard in the app.layout section of the sample code. The layout allows us to organize the different components that make up our dashboard, such as tables, graphs, headers, text, and so on. You will learn more about how to build Dash layouts and the DataTable component in the module readings.

In the sample code, pictured below, you will see code for a basic layout that includes a title and a data table. The Dash framework has a built-in data table component, titled "DataTable," which will format the data in a much more user-friendly manner. Note that the DataTable component uses our Pandas data frame to populate the table: data=df.to_dict("records"). For now, the table is populating the whole data set, because that is what we imported in the previous step.

```
27 ##########################
28 # Dashboard Layout / View
29 ##########################
30 app = JupyterDash('Dash DataTable Only')
31
32 app.layout = html.Div([
33
34     html.Center(html.B(html.H1('SNHU CS-340 Dashboard'))),
35     html.Br(),
36     dash_table.DataTable(
37         id='datatable-interactivity',
38         columns=[
39             {"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns
40         ],
41         data=df.to_dict('records'),
42         editable=False,
43         filter_action="native",
44         sort_action="native",
45         sort_mode="multi",
46         column_selectable=False,
47         row_selectable=False,
48         row_deletable=False,
49         selected_columns=[],
50         selected_rows=[],
51         page_action="native",
52         page_current= 0,
53         page_size= 10,
54     ),
55     html.Br(),
56     html.Hr(),
57
58 ])
```

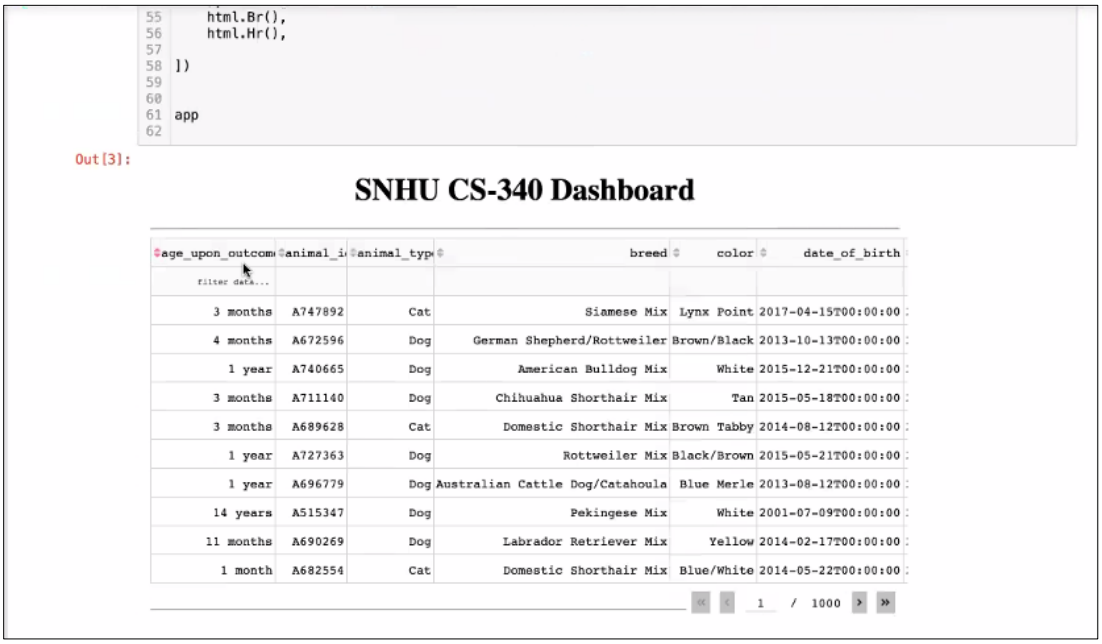
Title

Imports data

Data Table

Note: This code is a very simple example based on samples from the Dash DataTable documentation. You will notice that different sorting and filtering options have been enabled. Learn more about these features in the Dash documentation and experiment with different ways of setting up your data table or layout.

Now, we will test our code for the data table by running the application. In the next image, we have added the line "app" to the end of the code and have run the cell in Jupyter. Notice how the header and title for our dashboard are at the top with the data table underneath, just like in the app.layout code. Already, this data table is neater and much easier to read!

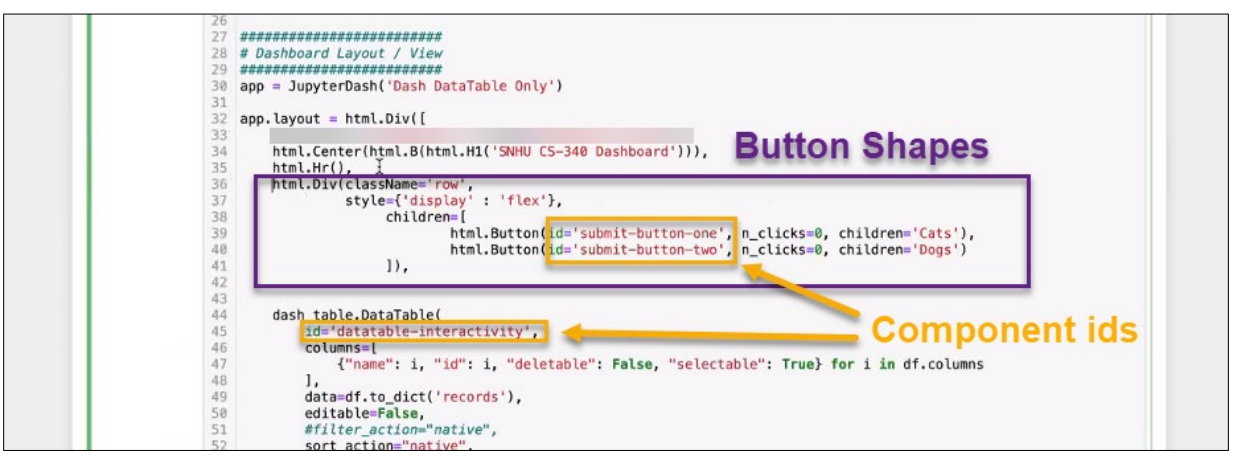


Adding Interactive Filter Options: Layout

Our next task will be to add some filtering options to allow the client to easily filter the data set to find cats or dogs. First, let's add the code for these filtering options to the layout. The Dash Core Components documentation describes several built-in elements that we could use: buttons, radio items, a drop-down menu, and so on. In this example, we have chosen to use the **button** element.

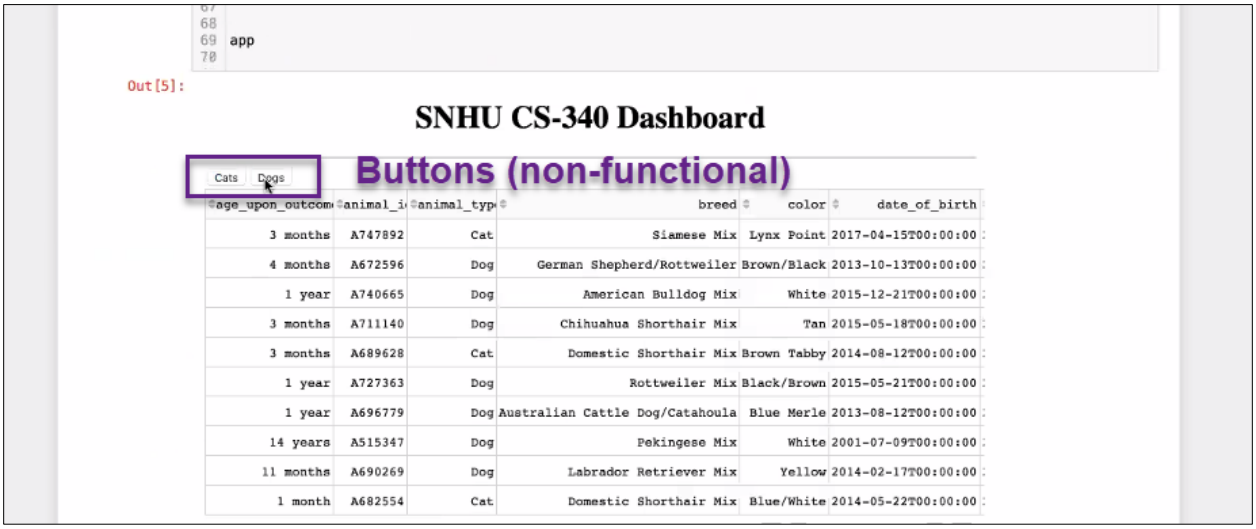
In the app.layout code pictured below, you can see new lines added for the button, based on the documentation. In this example, lines 39 and 40 show the code for the two buttons that we are adding, an initialized value of 0 for the number of clicks, and labels for the buttons. The additional new lines of code place the buttons in one row, so that they are side by side instead of stacked vertically. The code lines also add optional styling to the buttons.

Notice that we need to specify an "id" for each of the buttons. There is an "id" for the DataTable element as well. Pay special attention to these "id" elements, as they will be important for the callbacks in the next step.



Running this code will produce the following result. Notice that the buttons are included in the dashboard layout just as we have structured them in the code: under the table's title but before the table. The buttons are also in a single, horizontal row instead of stacked one on top of the other. We could move these buttons to a different part of the dashboard by moving those lines of code in the app.layout section.

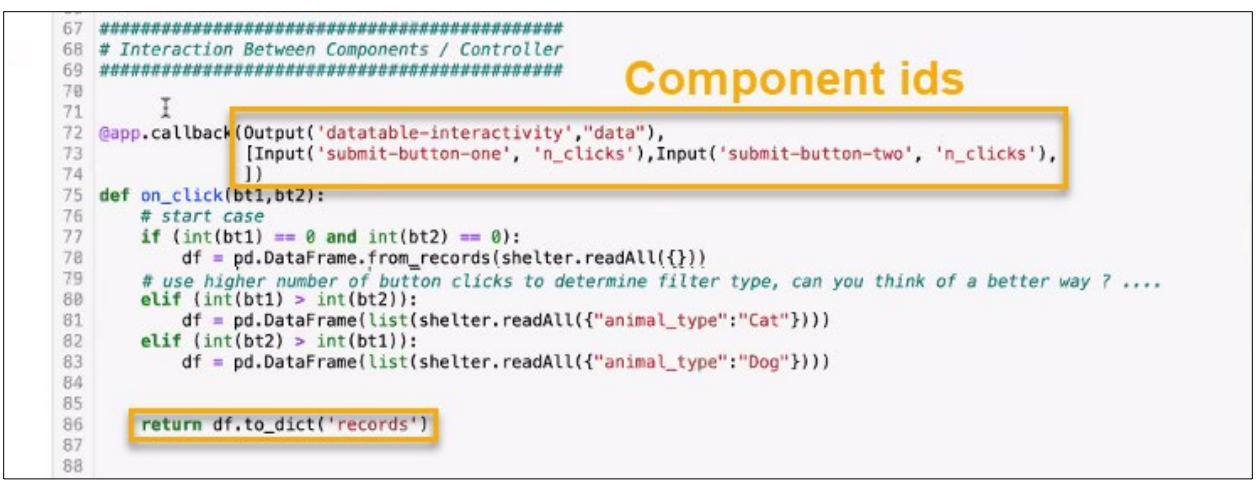
At this point, if we were to click on these buttons, they would **not** have any functionality. To make the buttons interact with the table, we need to add in the appropriate callbacks.



age_upon_outcom	animal_i	animal_typ	breed	color	date_of_birth
3 months	A747892	Cat	Siamese Mix	Lynx Point	2017-04-15T00:00:00
4 months	A672596	Dog	German Shepherd/Rottweiler	Brown/Black	2013-10-13T00:00:00
1 year	A740665	Dog	American Bulldog Mix	White	2015-12-21T00:00:00
3 months	A711140	Dog	Chihuahua Shorthair Mix	Tan	2015-05-18T00:00:00
3 months	A689628	Cat	Domestic Shorthair Mix	Brown Tabby	2014-08-12T00:00:00
1 year	A727363	Dog	Rottweiler Mix	Black/Brown	2015-05-21T00:00:00
1 year	A696779	Dog	Australian Cattle Dog/Catahoula	Blue Merle	2013-08-12T00:00:00
14 years	A515347	Dog	Pekingese Mix	White	2001-07-09T00:00:00
11 months	A690269	Dog	Labrador Retriever Mix	Yellow	2014-02-17T00:00:00
1 month	A682554	Cat	Domestic Shorthair Mix	Blue/White	2014-05-22T00:00:00

Adding Interactive Filter Options: Callbacks

The documentation for the Dash Core Components includes samples of code for the callbacks. In the callback structure, there is an "output" id that matches the id of the data table element from the layout. Then there are two "input" ids that match the ids of the two button elements from the layout. This indicates that this callback will take input from the button elements and output it to the data table. Notice that the "return" for the "on_click" function in the code pictured below is "df.to_dict('records')", which corresponds to the "data=df.to_dict('records')" line for the DataTable component. This makes it so that the results of the callback will return the correct data type for the data table.




```
67 ##########################################
68 # Interaction Between Components / Controller
69 ##########################################
70
71
72 @app.callback(Output('datatable-interactivity',"data"),
73                [Input('submit-button-one', 'n_clicks'),Input('submit-button-two', 'n_clicks'),
74                ])
75 def on_click(bt1,bt2):
76     # start case
77     if (int(bt1) == 0 and int(bt2) == 0):
78         df = pd.DataFrame.from_records(shelter.readAll({}))
79     # use higher number of button clicks to determine filter type, can you think of a better way ? ....
80     elif (int(bt1) > int(bt2)):
81         df = pd.DataFrame(list(shelter.readAll({"animal_type":"Cat"})))
82     elif (int(bt2) > int(bt1)):
83         df = pd.DataFrame(list(shelter.readAll({"animal_type":"Dog"})))
84
85
86     return df.to_dict('records')
87
88
```

You will have to determine the appropriate logic based on the element that you choose. This button component uses “clicks.” The way this logic is structured, there is an initial state for the callback: *What happens if nothing is pressed*? In this case, the table will just display all records, based on the command “df = pd.DataFrame.from_records(shelter.readAll({}))”.

Note: “readAll” was the name of the read method for the “shelter_animals” CRUD Python module. You will need to use the name of the method from your CRUD Python module.

```
67 ##############################################
68 # Interaction Between Components / Controller
69 ##############################################
70
71
72 @app.callback(Output('datatable-interactivity',"data"),
73                [Input('submit-button-one', 'n_clicks'),Input('submit-button-two', 'n_clicks'),
74                ])
75 def on_click(bt1,bt2):
76     # start case
77     if (int(bt1) == 0 and int(bt2) == 0):
78         df = pd.DataFrame.from_records(shelter.readAll({}))
79     # use higher number of button clicks to determine filter type, can you think of a better way ? ....
80     elif (int(bt1) > int(bt2)):
81         df = pd.DataFrame(list(shelter.readAll({"animal_type":"Cat"})))
82     elif (int(bt2) > int(bt1)):
83         df = pd.DataFrame(list(shelter.readAll({"animal_type":"Dog"})))
84
85
86     return df.to_dict('records')
87
88
```

Component ids

(Inital state: no clicks)

There are different filtering queries written for each of the two buttons, Cat and Dog.

1. df = pd.DataFrame(list(shelter.readAll({"animal_type":"Cat"}))
2. df = pd.DataFrame(list(shelter.readAll({"animal_type":"Dog"}))

```
67 ##############################################
68 # Interaction Between Components / Controller
69 ##############################################
70
71
72 @app.callback(Output('datatable-interactivity',"data"),
73                [Input('submit-button-one', 'n_clicks'),Input('submit-button-two', 'n_clicks'),
74                ])
75 def on_click(bt1,bt2):
76     # start case
77     if (int(bt1) == 0 and int(bt2) == 0):
78         df = pd.DataFrame.from_records(shelter.readAll({}))
79     # use higher number of button clicks to determine filter type, can you think of a better way ? ....
80     elif (int(bt1) > int(bt2)):
81         df = pd.DataFrame(list(shelter.readAll({"animal_type":"Cat"})))
82     elif (int(bt2) > int(bt1)):
83         df = pd.DataFrame(list(shelter.readAll({"animal_type":"Dog"})))
84
85
86     return df.to_dict('records')
87
88
```

Component ids

(State changes)

Notice that the logic is set up so that these actions “trigger” when one button is clicked more than another. This is not the only approach, and may include some risks. If the Cat button is clicked once, how many times will the Dog button have to be clicked to filter the data table to show dogs? Can you think of another way to set up the logic for these buttons? How would you set up buttons for more than two filtering options?

When the app is run again, the buttons are now functional. Clicking the buttons should filter the data set. Based on the current button logic, from the initial state, clicking the Cats button once will filter the table to show only cats. To filter the table to show dogs, the Dogs button will need to be clicked twice.

```
app
```

Out[6]:

SNHU CS-340 Dashboard

Click "Cats" button once

Cats Dogs

age_upon_outcom	animal_i	animal_typ	breed	color	date_of_birth
3 months	A747892	Cat	Siamese Mix	Lynx Point	2017-04-15T00:00:00	2017-08-0
3 months	A689628	Cat	Domestic Shorthair Mix	Brown Tabby	2014-08-12T00:00:00	2014-11-1
1 month	A682554	Cat	Domestic Shorthair Mix	Blue/White	2014-05-22T00:00:00	2014-06-3
2 months	A730724	Cat	Domestic Shorthair Mix	Tortie	2016-06-09T00:00:00	2016-08-1
1 month	A705018	Cat	Domestic Shorthair Mix	Black	2015-05-21T00:00:00	2015-07-1
1 weeks	A728214	Cat	Domestic Shorthair Mix	Gray Tabby	2016-05-19T00:00:00	2016-06-0
1 month	A709302	Cat	Domestic Shorthair Mix	Blue Tabby	2015-06-30T00:00:00	2015-08-1
2 months	A713268	Cat	Domestic Shorthair Mix	Black/White	2015-09-13T00:00:00	2015-11-1
1 week	A706525	Cat	Domestic Shorthair Mix	Brown Tabby	2015-06-23T00:00:00	2015-06-3
4 months	A731775	Cat	Domestic Shorthair Mix	Calico	2016-03-12T00:00:00	2016-08-0

« ‹ 1 / 371 › »

```
app
```

Out[6]:

SNHU CS-340 Dashboard

Cats Dogs Click "Dogs" button twice

age_upon_outcom	animal_i	animal_typ	breed	color	date_of_birth
4 months	A672596	Dog	German Shepherd/Rottweiler	Brown/Black	2013-10-13T00:00:00
1 year	A740665	Dog	American Bulldog Mix	White	2015-12-21T00:00:00
3 months	A711140	Dog	Chihuahua Shorthair Mix	Tan	2015-05-18T00:00:00
1 year	A727363	Dog	Rottweiler Mix	Black/Brown	2015-05-21T00:00:00
1 year	A696779	Dog	Australian Cattle Dog/Catahoula	Blue Merle	2013-08-12T00:00:00
14 years	A515347	Dog	Pekingese Mix	White	2001-07-09T00:00:00
11 months	A690269	Dog	Labrador Retriever Mix	Yellow	2014-02-17T00:00:00
3 years	A698651	Dog	Dogo Argentino Mix	White/Black	2012-03-15T00:00:00
1 year	A650694	Dog	Pit Bull/Blue Lacy	Blue/White	2012-05-31T00:00:00
8 months	A764967	Dog	Pbgv Mix	Tan/White	2017-05-10T00:00:00

« ‹ 1 / 569 › »