# Milestone Project #1 - Stock Ticker Dashboard

The goal of this project is to develop a dashboard that allows users to select one or more stocks, a start and end date, and have the closing stock prices displayed as a time series.

Rather than attempt to code it all at once, we’ll break it down into manageable (and testable) benchmarks.

The sequence of files will be:

**StockTicker1.py** - perform imports, set up a graph with static data, ensure that we can lay everything out on the screen

**StockTicker2.py** - add an input box and a basic callback to display the input value (the ticker) on the graph.

**StockTicker3.py** - Ensure that we can read data off the web using pandas datareader

**StockTicker4.py** - add datepickers to select start and end dates and apply them to the callback

**StockTicker5.py** - take advantage of Dash State, and hold all API calls until a Submit button is pressed

**StockTicker6.py** - replace the input box with a multiple dropdown list of choices. Pass multiple stocks as traces on the same graph.

### StockTicker1.py

Create a new file, and build a dashboard with a very simple graph. Don’t worry about assigning IDs at this time.   
We’ll address callbacks in the next section.

***# perform the basic imports***

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html

***# launch the application***

app = dash.Dash()

***# Create a Div to contain basic headers, an input box, and our graph***

app.layout = html.Div([  
 html.H1('Stock Ticker Dashboard'),  
 html.H3('Enter a stock symbol:'),  
 dcc.Input(

id ='my\_ticker\_symbol'   
 value='TSLA' *# sets a default value* ),  
 dcc.Graph(id='my\_graph',

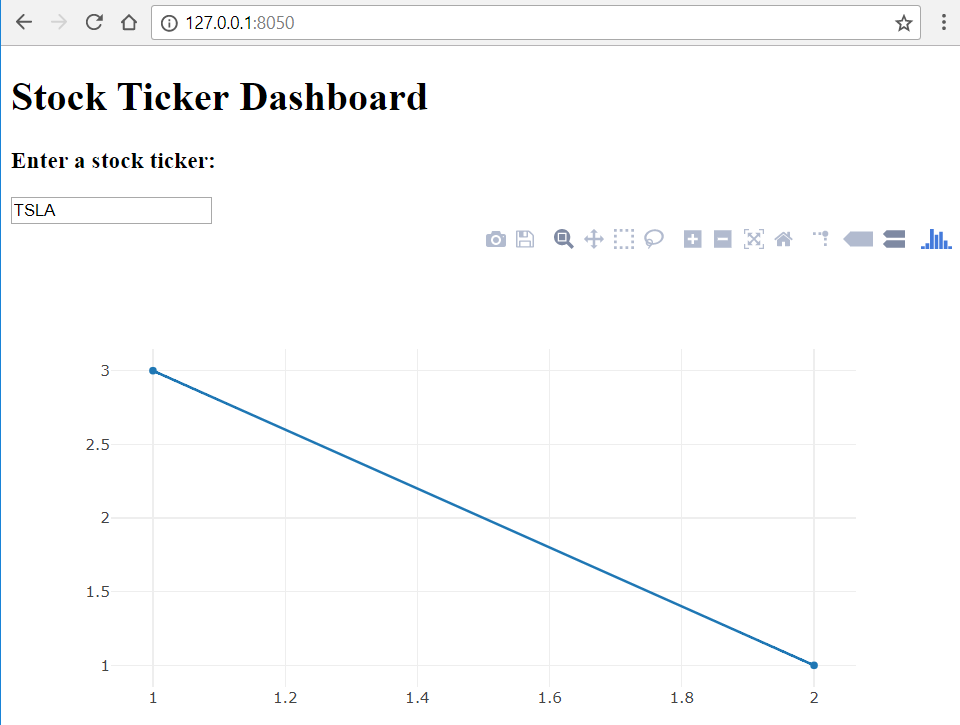
figure={  
 'data': [  
 {'x': [1,2], 'y': [3,1]}  
 ]  
 }  
 )  
])

***# Add the server clause***

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server()

Run the script, open a browser to <http://127.0.0.1:8050/> , and you should see this:



Great! We know our layout works. Now it’s time to add a callback, and see if we can add some interactivity.

### StockTicker2.py

Copy StockTicker1.py and name the new file StockTicker2.py. Here we’re only going to add a callback to our dashboard, and have the text entered into the Input box appear as our graph’s title.

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html

***# Add an import for Input/Output***

from dash.dependencies import Input, Output

app = dash.Dash()

app.layout = html.Div([  
 html.H1('Stock Ticker Dashboard'),  
 html.H3('Enter a stock symbol:'),  
 dcc.Input(

***# Add an ID to the input box***

id='my\_ticker\_symbol’,  
 value='TSLA'

),  
 dcc.Graph(

***# Add an ID to the graph***  
 id='my\_graph',  
 figure={  
 'data': [  
 {'x': [1,2], 'y': [3,1]}  
 ],  
 }  
 )  
])

***# Add a callback function***

@app.callback(  
 Output('my\_graph', 'figure'),  
 [Input('my\_ticker\_symbol', 'value')])  
def update\_graph(stock\_ticker):  
 fig = {  
 'data': [  
 {'x': [1,2], 'y': [3,1]}  
 ],  
 'layout': {'title':stock\_ticker}  
 }  
 return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server()

What happens here is we return the exact same figure, except that we add a **layout** with a title that automatically updates with the contents of the Input box. Run the script to make sure it works!

### StockTicker3.py

Copy StockTicker2.py and name the new file StockTicker3.py. Here we’re going to import *pandas\_datareader* and try to obtain stock data off the web.

NOTE: APIs are constantly changing. We use IEX in this example, as Google and Yahoo have recently been deprecated.   
For the latest information on supported sites, visit <https://pandas-datareader.readthedocs.io/en/latest/index.html>

In order to use IEX, we also have to pass start and end dates to the API. We’ll import the *datetime* module for this.

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
from dash.dependencies import Input, Output  
***# Add an import for pandas\_datareader and datetime***

import pandas\_datareader.data as web

from datetime import datetime

app = dash.Dash()  
  
app.layout = html.Div([  
 html.H1('Stock Ticker Dashboard'),  
 html.H3('Enter a stock symbol:'),  
 dcc.Input(  
 id='my\_ticker\_symbol',  
 value='TSLA'  
 ),  
 dcc.Graph(  
 id='my\_graph',  
 figure={  
 'data': [  
 {'x': [1,2], 'y': [3,1]}  
 ]  
 }  
 )  
])  
@app.callback(  
 Output('my\_graph', 'figure'),  
 [Input('my\_ticker\_symbol', 'value')])  
def update\_graph(stock\_ticker):

***# Use datareader and datetime to define a DataFrame***

start = datetime(2017, 1, 1)

end = datetime(2017, 12, 31)

df = web.DataReader(stock\_ticker,'iex',start,end)

***# Change the output data***  
 fig = {  
 'data': [  
 {'x': df.index, 'y': df.close}  
 ],  
 'layout': {'title':stock\_ticker}  
 }  
 return fig  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run\_server()

Run the script, and now you should see a year’s worth of stock closing prices!

NOTE: We can provide other datetime examples here. For instance, to get the last 90 days stock activity:

from datetime import date, timedelta

start = datetime.today()-timedelta(days=90)  
end = datetime.today()

Next, let’s add DatePicker components that allow the user to set start and end dates for the stock data. We can either use two **DatePickerSingle** elements, or one **DatePickerRange**. To understand how they work, visit <https://dash.plot.ly/dash-core-components/datepickersingle> and <https://dash.plot.ly/dash-core-components/datepickerrange>

### StockTicker4.py

Copy StockTicker3.py and name the new file StockTicker4.py. We’re adding a DatePickerRange component, and passing its input values to our callback.

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
from dash.dependencies import Input, Output  
import pandas\_datareader.data as web  
from datetime import datetime  
  
app = dash.Dash()  
  
app.layout = html.Div([  
 html.H1('Stock Ticker Dashboard'),  
 html.Div([  
 ***# add styles to enlarge the input box and make room for DatePickerRange***

html.H3('Enter a stock symbol:', style={'paddingRight':'30px'}),  
 dcc.Input(  
 id='my\_ticker\_symbol',  
 value='TSLA',  
 style={'fontSize':24, 'width':75}  
 )  
 ], style={'display':'inline-block', 'verticalAlign':'top'}),

***# add a Div to contain the DatePickerRange***  
 html.Div([  
 html.H3('Select start and end dates:'),  
 dcc.DatePickerRange(  
 id='my\_date\_picker',  
 Min\_date\_allowed = datetime(2015, 1, 1),  
 Max\_date\_allowed = datetime.today(),  
 Start\_date = datetime(2018, 1, 1),  
 End\_date = datetime.today()  
 )  
 ], style={'display':'inline-block'}),  
 dcc.Graph(  
 id='my\_graph',  
 figure={  
 'data': [  
 {'x': [1,2], 'y': [3,1]}  
 ]  
 }  
 )  
])

@app.callback(

***# add inputs from the DatePickerRange component***  
 Output('my\_graph', 'figure'),  
 [Input('my\_ticker\_symbol', 'value'),  
 Input('my\_date\_picker', 'start\_date'),  
 Input('my\_date\_picker', 'end\_date')])  
def update\_graph(stock\_ticker, start\_date, end\_date):  
 start = datetime.strptime(start\_date[:10], '%Y-%m-%d')  
 end = datetime.strptime(end\_date[:10], '%Y-%m-%d')  
 df = web.DataReader(stock\_ticker,'iex',start,end)  
 fig = {  
 'data': [  
 {'x': df.index, 'y': df.close}  
 ],  
 'layout': {'title':stock\_ticker}  
 }  
 return fig  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run\_server()

Remember, even though **dcc.DatePickerRange** can use datetime objects for start and end dates, when we perform a callback, the web page is submitting dates as strings, not as datetime objects. For this we use   
start\_date = datetime.**strptime**(start\_date**[:10]**, '%Y-%m-%d')  
The [:10] slice is because we only want the YYYY-MM-DD portion of the input string.

Later, if we want to display start\_date in a longer text format, we can use

start\_date\_string = start\_date.**strftime**('%B %d, %Y')

For more information on string formatting of datetime objects visit <https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior>

NOTE: As much as I’d like to use **date** objects instead of **datetime**, the **strptime** method is only available on datetime objects. Hence the need for slice notation...

Great job! At this point we’ve constructed a dashboard that opens with default values, polls the web for financial data and displays it in a graph, then gives the user the option of calling a different stock symbol, and setting the starting and ending dates for the graph.

Before we address the challenge of expanding the stock symbol input to allow multiple stocks to appear on the same graph, let’s take advantage of Dash State, and add a Submit button.

### StockTicker5.py

Copy StockTicker4.py and name the new file StockTicker5.py. You may have noticed in earlier versions that the graph would try to paint as soon as a letter was typed into the input box, and for every letter thereafter. To reduce network traffic, we want to enter all of our stock symbol and date settings first, and *then* submit the API call.   
In this section we’ll add a Submit button and move all our previous Inputs into Dash State properties.

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
from dash.dependencies import Input, Output, State ***# add State to imports***import pandas\_datareader.data as web  
from datetime import datetime  
  
app = dash.Dash()  
  
app.layout = html.Div([  
 html.H1('Stock Ticker Dashboard'),  
 html.Div([  
 html.H3('Enter a stock symbol:', style={'paddingRight':'30px'}),  
 dcc.Input(  
 id='my\_ticker\_symbol',  
 value='TSLA', # sets a default value  
 style={'fontSize':24, 'width':75}  
 )  
 ], style={'display':'inline-block', 'verticalAlign':'top'}),  
 html.Div([  
 html.H3('Select start and end dates:'),  
 dcc.DatePickerRange(  
 id='my\_date\_picker',  
 min\_date\_allowed=datetime(2015, 1, 1),  
 max\_date\_allowed=datetime.today(),  
 start\_date=datetime(2018, 1, 1),  
 end\_date=datetime.today()  
 )  
 ], style={'display':'inline-block'}),

***# add a Button element***  
 html.Div([  
 html.Button(  
 id='submit-button',  
 n\_clicks=0,  
 children='Submit',  
 style={'fontSize':24, 'marginLeft':'30px'}  
 ),  
 ], style={'display':'inline-block'}),  
 dcc.Graph(  
 id='my\_graph',  
 figure={  
 'data': [  
 {'x': [1,2], 'y': [3,1]}  
 ]  
 }  
 )  
])  
@app.callback(  
 Output('my\_graph', 'figure'),

***# add a button input, and move previous inputs to State***  
 [Input('submit-button', 'n\_clicks')],  
 [State('my\_ticker\_symbol', 'value'),  
 State('my\_date\_picker', 'start\_date'),  
 State('my\_date\_picker', 'end\_date')])

***# pass n\_clicks into the output function***

def update\_graph(n\_clicks, stock\_ticker, start\_date, end\_date):  
 start = datetime.strptime(start\_date[:10], '%Y-%m-%d')  
 end = datetime.strptime(end\_date[:10], '%Y-%m-%d')  
 df = web.DataReader(stock\_ticker,'iex',start,end)  
 fig = {  
 'data': [  
 {'x': df.index, 'y': df.close}  
 ],  
 'layout': {'title':stock\_ticker}  
 }  
 return fig  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run\_server()

That’s it! Run the script and you should be able to make multiple changes without affecting the graph until Submit is clicked!

### StockTicker6.py

Copy StockTicker5.py and name the new file StockTicker6.py. In this next section we want to permit multiple stock selections, and have them all display on the same graph.

You have a lot options for this. You can restrict users to a small set of radio buttons. You can set up several input boxes. You can use a Dropdown that permits multiple selections.

We chose to use a Dropdown list only because what’s displayed in the list can be different than the value passed to Input (at this time we can’t do that with Input boxes). This lets us select “AAPL Apple Computer” and have just “AAPL” pass behind the scenes.

We obtained a csv file of NASDAQ listed companies with stock symbols. We whittled the 3299 records down to 256 companies with relatively high market capitalization as of 4/6/18.  
Data source: <http://www.nasdaq.com/screening/companies-by-industry.aspx?exchange=NYSE&render=download>

INTERMEDIATE SCRIPTS:   
**StockTicker6a.py** offers a Multi-Value Dropdown with 3 hardwired options, where only the first value is considered for the API call.

**StockTicker6b.py** creates a trace from each ticker symbol by building dataframes and putting relevant information into each trace. It successfully charts multiple traces.

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
from dash.dependencies import Input, Output, State  
import pandas\_datareader.data as web  
from datetime import datetime  
***# import pandas***

import pandas as pd  
  
app = dash.Dash()  
***# read a .csv file, make a dataframe, and build a list of Dropdown options***  
nsdq = pd.read\_csv('../data/NASDAQcompanylist.csv')  
nsdq.set\_index('Symbol', inplace=True)  
options = []  
for tic in nsdq.index:  
 options.append({'label':'{} {}'.format(tic,nsdq.loc[tic]['Name']), 'value':tic})  
  
app.layout = html.Div([  
 html.H1('Stock Ticker Dashboard'),  
 html.Div([  
 html.H3('Select stock symbols:', style={'paddingRight':'30px'}),

***# replace dcc.Input with dcc.Options, set options=options***  
 dcc.Dropdown(  
 id='my\_ticker\_symbol',  
 options=options,  
 value=['TSLA'],  
 multi=True  
 )  
 ***# widen the Div to fit multiple inputs***

], style={'display':'inline-block', 'verticalAlign':'top', 'width':'30%'}),  
 html.Div([  
 html.H3('Select start and end dates:'),  
 dcc.DatePickerRange(  
 id='my\_date\_picker',  
 min\_date\_allowed=datetime(2015, 1, 1),  
 max\_date\_allowed=datetime.today(),  
 start\_date=datetime(2018, 1, 1),  
 end\_date=datetime.today()  
 )  
 ], style={'display':'inline-block'}),  
 html.Div([  
 html.Button(  
 id='submit-button',  
 n\_clicks=0,  
 children='Submit',  
 style={'fontSize':24, 'marginLeft':'30px'}  
 ),  
 ], style={'display':'inline-block'}),  
 dcc.Graph(  
 id='my\_graph',  
 figure={  
 'data': [  
 {'x': [1,2], 'y': [3,1]}  
 ]  
 }  
 )  
])  
@app.callback(  
 Output('my\_graph', 'figure'),  
 [Input('submit-button', 'n\_clicks')],  
 [State('my\_ticker\_symbol', 'value'),  
 State('my\_date\_picker', 'start\_date'),  
 State('my\_date\_picker', 'end\_date')])  
def update\_graph(n\_clicks, stock\_ticker, start\_date, end\_date):  
 start = datetime.strptime(start\_date[:10], '%Y-%m-%d')  
 end = datetime.strptime(end\_date[:10], '%Y-%m-%d')

***# since stock\_ticker is now a list of symbols, create a list of traces***  
 traces = []  
 for tic in stock\_ticker:  
 df = web.DataReader(tic,'iex',start,end)  
 traces.append({'x':df.index, 'y': df.close, 'name':tic})  
 fig = {

***# set data equal to traces***  
 'data': traces,

***# use string formatting to include all symbols in the chart title***  
 'layout': {'title':', '.join(stock\_ticker)+' Closing Prices'}  
 }  
 return fig  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run\_server()

Run the script, and you’ll notice that the Dropdown has TSLA preselected. To the right of it, if you place your cursor inside the box you can see the list of all 256 companies. Even better, if you start to type letters in the box, only those entries that match appear. Type “**go**” to see this happen.

We’re done! We have a stock closing price dashboard that accepts multiple selections, a range of dates, makes use of Dash State, and does it all in less than 100 lines of code. The end result should look something like this:



Feel free to customize your dashboard by changing the layout and adding more style