DAILY ASSESSMENT FORMAT

Date:	3rd June 2020	**Name:**	Soundarya NA
Course:	UDEMY	**USN:**	4AL16EC077
Topic:	PYTHON: Application 8: Build a Web based financial graph	**Semester & Section:**	8th - B

FORENOON SESSION DETAILS

Report:

Data visualization is an essential step in quantitative analysis with Python.

There are many tools at our disposal for data visualization, and the topics we will cover in this guide include:

- Matplotlib
- Pandas
- Time Series Visualization
- Seaborn
- Plotly & Dash

Matplotlib:

Matplotlib has established itself as the benchmark for data visualization, and is a robust and reliable tool. It is both easy to use for standard plots and flexible when it comes to more complex plots and customizations. In addition, it is tightly integrated with NumPy and the data structures that it provides. Matplotlib is modeled after MATLAB's plotting capabilities, and creates static image files of almost any plot type.

```
[1] import matplotlib.pyplot as plt

    # allows you to see the plots we create inside the notebook
    %matplotlib inline

    # let's start with a simple example with 2 numpy arrays
    import numpy as np
    x = np.linspace(0,10,10)

[2] y = x**2

[3] x

    array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
            5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.        ])

[4] y

    array([  0.        ,   1.2345679 ,   4.9382716 ,  11.11111111,
            19.75308642,  30.86419753,  44.44444444,  60.49382716,
            79.01234568, 100.        ])
```

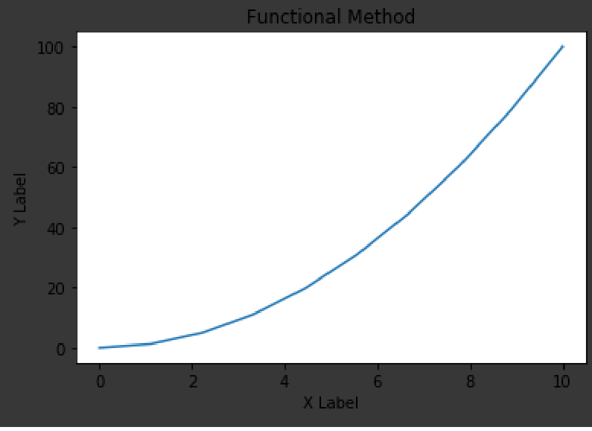
Two different Matplotlib are:

- Functional method
- Object-oriented method

1. **Functional method:**

```
[5] # functional method
    plt.plot(x, y)
    plt.xlabel('X Label')
    plt.ylabel('Y Label')
    plt.title('Functional Method')
```

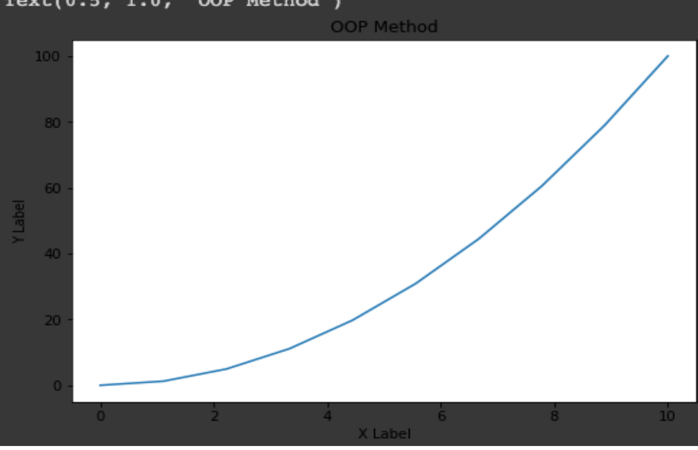
```
Text(0.5, 1.0, 'Functional Method')
```



2. **Object-oriented method:**

```
# create a figure object
fig = plt.figure()

# add axes to the canvas
# left, bottom, width, height
axes = fig.add_axes([0.1,0.1,1,1])

# next we plot on the axes
axes.plot(x,y)
axes.set_xlabel('X Label')
axes.set_ylabel('Y Label')
axes.set_title('OOP Method')
```

```
Text(0.5, 1.0, 'OOP Method')
```



Pandas:

The main purpose of pandas is data analysis, but as we'll see pandas has amazing visualization capabilities.

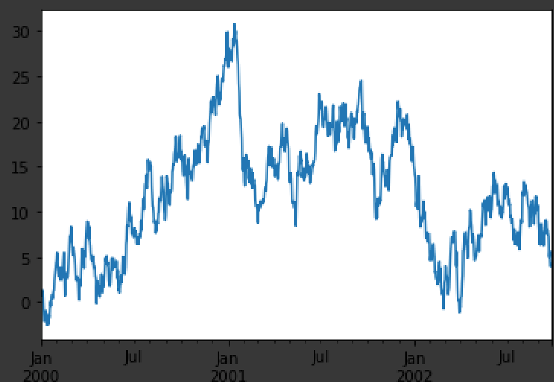
If you set your Data Frame right you can create pretty much any visualization with a single line of code.

Pandas uses matplotlib on the backend through simple. plot calls. The plot method on Series and Data Frame is just a simple wrapper around plt.plot().

Pandas does, however, have a limited scope of plot types, and they are all static.

```
[14] import pandas as pd
     import numpy as np

     # Basic Plotting: plot
     # The plot method on Series and DataFrame is just a simple wrapper around plt.plot()
     ts = pd.Series(np.random.randn(1000),
                    index=pd.date_range('1/1/2000', periods=1000))

     ts = ts.cumsum()
     ts.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9106b4dc18>
```



Time series visualization:

```
[20] from google.colab import files
     uploaded = files.upload()
```

```
Choose Files TSLA.csv
• TSLA.csv(text/csv) - 18760 bytes, last modified: 5/1/2019 - 100% done
Saving TSLA.csv to TSLA.csv
```

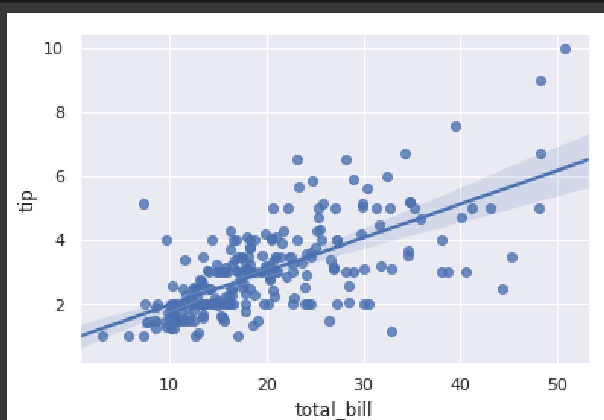
```
[30] # read in our csv
     df = pd.read_csv('TSLA.csv',index_col='Date',parse_dates=True)
     df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2018-05-01	293.510010	300.820007	293.220001	299.920013	299.920013	4625600
2018-05-02	298.570007	306.850006	297.779999	301.149994	301.149994	8970400
2018-05-03	278.790009	288.040009	275.230011	284.450012	284.450012	17352100
2018-05-04	283.000000	296.859985	279.519989	294.089996	294.089996	8569400
2018-05-07	297.500000	305.959991	295.170013	302.769989	302.769989	8678200

Seaborn:

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

```
[31] import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     sns.set(color_codes=True)
     tips = sns.load_dataset("tips")
     sns.regplot(x="total_bill", y="tip", data=tips);
```



Plotly and dash:

All of the plots we've seen so far are static - that is, once you create them you can't interact with the plot in any way.

This is what Plotly solves.

Plotly is both a company and an open source library.

Plotly the company focuses on data visualization for business intelligence, and the open source library is a general data visualization library that specializes in interactive visualizations.

Plotly has libraries for JavaScript, React, R, and Python - but we'll stick with Python in this guide.

Using the plotly python library creates interactive plots as .html files.

Users can interact with these plots (zoom in, select, hover, etc) - but one of the limitations is that these plots can't be connected to changing data sources.

Once the plot is generated, the data is essentially locked-in at that time, and in order to regenerate a plot to see updates you need to re-run the .py script.

This is where Plotly's Dash comes in.

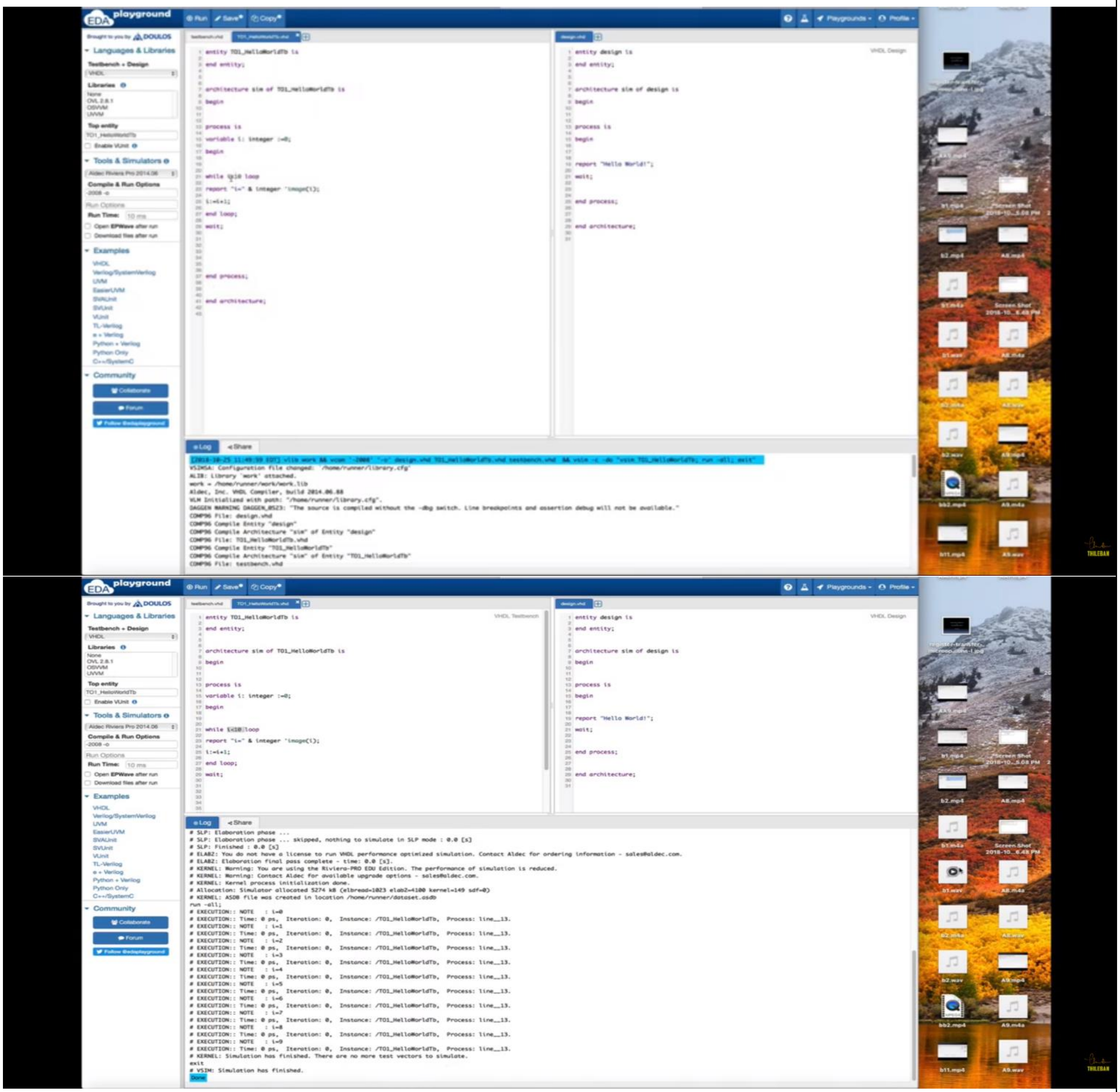
Conclusion:

As we've seen, Python has many data visualization libraries including Matplotlib, Pandas, Seaborn, and Plotly.

Most of these are static visualization libraries, but open-source library Plotly lets you create interactive images, and Dash lets you create dashboard web applications.

Date:	3rd June 2020	**Name:**	Soundarya NA
Course:	Digital Design using HDL	**USN:**	4AL16EC077
Topic:	Hands on EDA tool	**Semester & Section:**	8th - B

Image:



Report:
EDA playground:
Design:

```
/*
  svunitOnSwitch
  The API is defined but that's pretty much it! Use
  the unit tests to help fill the proper implementation.
*/
interface svunitOnSwitch (
  output logic on
);
  initial on = 'hx;
  function logic true();
    // no implementation yet
  endfunction
  function logic false();
    // no implementation yet
  endfunction
  function int return43();
    // no implementation yet
  endfunction
  function void turn_on();
    // no implementation yet
  endfunction
  function void turn_off();
    // no implementation yet
  endfunction
endinterface
```

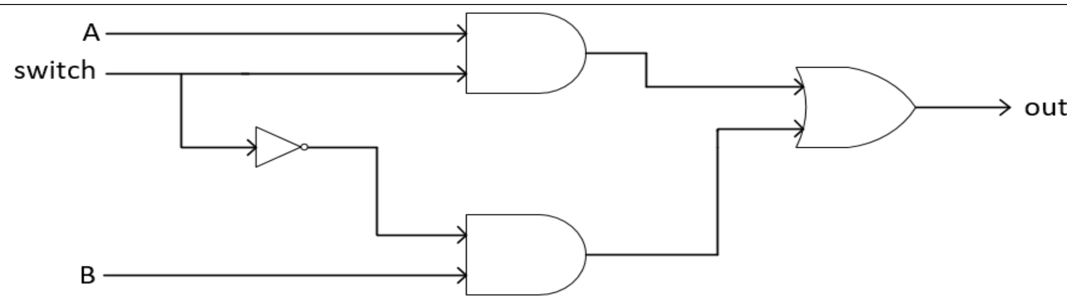
Test bench:

```
/*
```

```
          ------------------------
          SVUnit DEMO INSTRUCTIONS
          ------------------------

  Time to learn the basics of unit testing with SVUnit!
  Here's a simple set of unit tests for an interface called
  svunitOnSwitch. The purpose of the design is to offer a few
  utility functions as well as to operate an on/off output.
  The API of the svunitOnSwitch is as follows:
    output logic on;
    function logic true();
    function logic false();
    function int return43();
    turn_on();
    turn_off();
  The svunitOnSwitch is defined over there --------------->
  The SVTESTs below are the acceptance unit tests that verify
  the functionality of the svunitOnSwitch. If you push run (in
  the top left corner) you'll see all the tests fail b/c none
  of the svunitOnSwitch functionality is implemented. Your
  job is to build a complete svunitOnSwitch, 1 requirement at
  a time, by:

    * examining the requirement defined in the unit test
      (HINT: a unit test is marked by the `SVTEST macro)
    * implementng the corresponding code in the svunitOnSwitch
      (HINT: watch for the "no implementation yet" comment)
    * running the test suite to make sure your implementation
      satisfies the unit test
  When you've gone through all the tests and your entire test
  suite passes, you're done! You've verified the
  svunitOnSwitch and learned the basics of SVUnit!
  The unit tests start a few lines down so scroll down to get
  started.
  Ready... set... go!
  You can do a lot more than test a simple svunitOnSwith with
  SVUnit. When you're ready to test your own design and
  testbench IP visit:
www.AgileSoC.com/svunit
"svunit_defines.svh"
import svunit_pkg::*;
module svunitDemo_unit_test;
  string name = "svunitDemo_ut";
  `SVUNIT_TESTS_BEGIN
  `SVTEST(true_returns_1)
    `FAIL_UNLESS(uut.true() === 1);
```

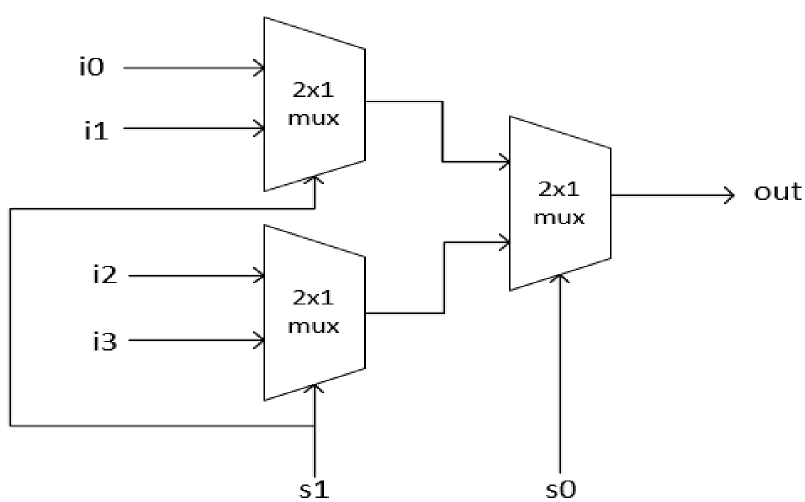
```
  `SVTEST_END
  `SVTEST(false_returns_0)
    `FAIL_UNLESS(uut.false() === 0);
  `SVTEST_END
  `SVTEST(return43)
    `FAIL_UNLESS(uut.return43() === 43);
  `SVTEST_END
  `SVTEST(turn_on)
    uut.turn_on();
    `FAIL_UNLESS(uut.on === 1);
  `SVTEST_END
  `SVTEST(turn_off)
    uut.turn_off();
    `FAIL_UNLESS(uut.on === 0);
  `SVTEST_END
    For more SVUnit, Remember to visit:
        www.AgileSoC.com/svunit
    And try the other SVUnit examples at:
        www.edaplayground.com
  `SVUNIT_TESTS_END
  svunit_testcase svunit_ut;
  svunitOnSwitch uut();
  function void build();
    svunit_ut = new(name);
  endfunction
  task setup();
    svunit_ut.setup();
  endtask
  task teardown();
    svunit_ut.teardown();
  endtask
endmodule
```

Implement 4 to 1 mux using two 2 to 1 mux using structural modelling style:



Verilog code for 2*1 MUX:

```
module mux2x1(out,a,b,s);

input a,b,s;

wire and_1,and_2,s_c;

output out;

not (s_c,s);

and (and_1,a,s_c);

and (and_2,b,s);

or (out,and_1,and_2);

endmodule
```



Verilog code for 4*1 mux:

```
module mux4x2(out,i0,i1,i2,i3,s1,s0);

input i0,i1,i2,i3,s1,s0;

output out;
```

```
wire mux1,mux2;
mux2x1 mux_1(mux1,i0,i1,s1);
mux2x1 mux_2(mux2,i2,i3,s1);
mux2x1 mux_3(out,mux1,mux2,s0);
endmodule
```

Testbench:

```
timescale 1ns/1ns
module mux4x2_tb;
wire t_out;
reg t_a, t_b, t_c, t_d, t_s1, t_s0;
mux4x2 my_4x2_mux( .i0(t_a), .i1(t_b), .i2(t_c), .i3(t_d), .s1(t_s1), .s0(t_s0), .out(t_out) );
initial
begin
// 1
t_a = 1'b1;
t_b = 1'b0;
t_c = 1'b1;
t_d = 1'b1;
t_s0 = 1'b0;
t_s1 = 1'b1;
#5 //2
t_a = 1'b0;
t_b = 1'b1;
t_c = 1'b0;
t_d = 1'b0;
t_s0 = 1'b0;
t_s1 = 1'b1;
#5 //3
t_a = 1'b0;
```

```
t_b = 1'b0;
t_c = 1'b1;
t_d = 1'b0;
t_s0 = 1'b1;
t_s1 = 1'b0;
#5 //4
t_a = 1'b0;
t_b = 1'b0;
t_c = 1'b0;
t_d = 1'b1;
t_s0 = 1'b1;
t_s1 = 1'b1;
#5 //5
t_a = 1'b1;
t_b = 1'b0;
t_c = 1'b0;
t_d = 1'b0;
t_s0 = 1'b0;
t_s1 = 1'b0;
end
endmodule
```

Simulated waveform: