Creation of a Fixed Income Dashboard using Python Dash Library

Environment: JupyterLab in Anaconda

Desired Output

The objective is to have a guide on creating a fixed income market view (go long or short duration, which area of the curve is most attractive) without the need for a Bloomberg Terminal.

In particular, the output will include the following: (1) local GDP growth and inflation, (2) chart of historical yield curve per maturity, (3) Term Spread Z-scores, and (4) Real Yield Z-scores.

Data

Local GDP Growth and Inflation, ***One-Time Extract***

For GDP growth, since I could not find a way to show last column (even with pd.read_excel() and requests), I had to manually download the file and just use Pandas __.read_csv(). The cleaned dataframe output is saved as a csv as well using __.to_csv().

	quarter	year	GDP
0	Q1	2000 - 2001	2.4
1	Q2	2000 - 2001	3.3
2	Q3	2000 - 2001	3.1
3	Q4	2000 - 2001	3.3
4	Q1	2001 - 2002	3.9
...	...	...	...
72	Q1	2018 - 2019	5.7
73	Q2	2018 - 2019	5.4
74	Q3	2018 - 2019	6.3
75	Q4	2018 - 2019	6.7
76	Q1	2019 - 2020	-0.2

For inflation data, I used BeautifulSoup, requests, and Regex to extract the information from html table.

	Year	Month	Inflation
85	2020	2	2.6
86	2020	3	2.5
87	2020	4	2.2
52	2020	5	2.1
53	2020	6	2.5

We perform the same saving of the dataframe stored using __.to_csv().

Unfortunately, as I was about to update for the following month, I noticed the website changed the "td class" associated with the inflation data which changes the pattern I set to search. As such, I put in place another procedure for singular updates.

Local GDP Growth and Inflation, ***Moving Forward***

Since data points are added only quarterly for GDP and monthly for inflation, instead of repeating the entire process again which unfortunately included some manual parts due to problems encountered, we just type down the updated information in a dataframe format then append this to the historical csv created. After which, we just use pd.to_csv() to update the historical dataframe stored.

Philippine Yields, ***One-Time Extract***

Since PH yields (BVAL) are saved as pdf files in the website, I first downloaded these in a local folder using requests and BeautifulSoup.

Before October 2018, PH local yields were PDST-R2 yields and these historical rates are also saved as a pdf which require a one-time download for the creation of our historical PH yield database. For this, I used tabula library since it is a clear table stored in a pdf file. I then saved it as a csv.

I then perform some manual arrangements of both datasets.

DATE	1M	3M	6M	1Y	2Y	3Y	4Y	5Y	7Y	10Y	20Y	25Y
3/19/2007	3.8	3.90	4.400	4.80	5.4000	5.5500	5.8000	6.0000	6.7500	7.3000	8.60	8.875
3/20/2007	3.9	2.95	3.462	4.80	5.3750	5.4700	5.8000	5.9100	6.6000	7.3000	8.60	9
3/21/2007	3.9	4.00	3.400	4.95	5.1750	5.6000	5.8500	5.9650	6.7500	7.3750	8.75	9.1
3/22/2007	3.9	3.95	4.400	4.80	5.2500	5.6000	5.9250	6.0750	6.9000	7.4000	8.70	9.1
3/23/2007	3.9	3.95	4.450	4.65	5.1734	5.4043	5.7147	5.8617	6.6833	7.1428	8.60	9

Philippine Yields, ***Moving Forward***

We only get the dates we need. The code creates the links to the pdf files and if it doesn't exist then it skips for that date, but if it does, then it downloads the pdf and extracts the data into a dataframe. This dataframe containing updated yields then gets appended to the historical dataframe named PHreferencerates.csv.

Term Spreads Z-score Computation

I load the PHreferencerates file and create Spreads columns from there.

```
#Create Spreads
Spread_20Y10Y=df['20Y']-df['10Y']
Spread_10Y7Y=df['10Y']-df['7Y']
Spread_10Y5Y=df['10Y']-df['5Y']
Spread_10Y2Y=df['10Y']-df['2Y']
Spread_7Y5Y=df['7Y']-df['5Y']
Spread_7Y2Y=df['7Y']-df['2Y']
Spread_5Y2Y=df['5Y']-df['2Y']
Spread_2Y1Y=df['2Y']-df['1Y']
```

From each spread, I compute a rolling 1-year and 5-year mean and standard deviation to eventually compute the latest Z-score using (X-mu)/vol.

Computing rolling mean and standard deviation:

```
#Get the rolling average spread
spread_ave_1year=Spreads.rolling(260).mean()[259:]
spread_ave_1year.head()
```

DATE	20-10	10-7	10-5	10-2	7-5	7-2	5-2	2-1
4/14/2008	1.498937	0.320744	0.603073	1.002626	0.282328	0.681882	0.399553	0.499653
4/15/2008	1.497880	0.319270	0.599734	0.998941	0.280464	0.679671	0.399207	0.499472

```
#Get rolling standard deviations
spread_vol_1year=Spreads.rolling(260).std()[259:]
spread_vol_1year.head()
```

DATE	20-10	10-7	10-5	10-2	7-5	7-2	5-2	2-1
4/14/2008	0.285383	0.160707	0.301568	0.348265	0.221330	0.319717	0.185075	0.305258
4/15/2008	0.286630	0.160354	0.298613	0.343773	0.219408	0.317055	0.184782	0.305213

The resulting start date for the 1-year vs. 5-year will be different. To get the difference(X-mu), I cut the initial "X" dataframe, i.e. spreads, such that it has the same starting date as the rolling average.

5 year Z-scores

```
#Get the rolling average spread
spread_ave_5year=Spreads.rolling(260*5).mean()[260*5-1:]
#Get rolling standard deviations
spread_vol_5year=Spreads.rolling(260*5).std()[260*5-1:]
# Get same size
spreads_5year=Spreads[260*5-1:]
```

```
# Get difference (X-mu)
diff_5year=spreads_5year.subtract(spread_ave_5year)
# Divide by vol to get Z
spread_Z_5year=diff_5year.divide(spread_vol_5year)
spread_Z_5year.head()
```

DATE	20-10	10-7	10-5	10-2	7-5	7-2	5-2	2-1
7/5/2012	-1.018218	-1.487481	-1.350080	0.247535	-0.715966	0.785317	1.307876	-0.551919
7/6/2012	-1.120038	-1.223913	-1.109204	0.213592	-0.586818	0.657819	1.088804	-0.497212

Just getting the latest Z-scores to check attractiveness:

```
latest_Z=pd.concat([latest_1yr_Z,latest_5yr_Z],axis=1)
latest_Z
```

	1-year Z-score	5-year Z-score
20-10	0.535533	0.384441
10-7	-0.441274	-0.077886
10-5	-0.198958	-0.372047
10-2	0.453131	-0.445925
7-5	0.088635	-0.357230
7-2	0.799704	-0.436555
5-2	1.060195	-0.244547
2-1	0.406872	-0.460011

Sample Interpretation:

Only the 5-year relative to the 2-year appears attractive with term spread Z-score > 1 for the 1-year horizon.

Real Yields Z-score Computation

After loading inflation and PH yields data, I add a DATE column to inflation data.

```
# add DATE column to inflation
inflation['DATE']=inflation.apply(lambda row: datetime.datetime(int(row.Year),int(row.Month),1),axis=1)
```

```
inflation.tail()
```

	Year	Month	Inflation	DATE
87	2020	4	2.2	2020-04-01
88	2020	5	2.1	2020-05-01
89	2020	6	2.5	2020-06-01
90	2020	7	2.7	2020-07-01
91	2020	7	2.7	2020-07-01

I then merge this with the yields data on the "DATE" column and from there create "real yield" columns as yield minus applicable inflation.

```
combined=pd.merge_asof(rates,inflation,on='DATE')
```

```
# create the real yields dataframe
tenors=['1M','3M','6M','1Y','2Y','3Y','4Y','5Y','7Y','10Y','20Y']
realyields=pd.DataFrame([combined[x]-combined['Inflation'] for x in tenors]).transpose()
realyields.columns=tenors
realyields['DATE']=combined['DATE']
realyields.set_index('DATE',inplace=True)
realyields.head()
```

	1M	3M	6M	1Y	2Y	3Y	4Y	5Y	7Y	10Y	20Y
DATE											
2013-01-02	-2.1	-2.5050	-2.225	-2.0000	-0.0785	0.4500	0.8250	0.9000	1.1000	1.400	2.840
2013-01-03	-2.3	-2.5050	-2.225	-2.0750	-0.4707	0.4250	0.7941	0.8850	1.1000	1.385	2.835
2013-01-04	-2.3	-2.5100	-2.230	-2.2000	-0.4899	0.4000	0.8003	0.8800	1.1250	1.370	2.785
2013-01-07	-2.3	-2.6250	-2.350	-2.0850	-0.5000	0.4000	0.7429	0.8012	1.1250	1.365	2.770
2013-01-08	-2.3	-2.7538	-2.525	-2.0858	-0.6245	0.3437	0.7472	0.8000	1.1745	1.350	2.765

I then save this as PHrealyields.csv.

```
realyields.to_csv("PHrealyields.csv",index=False)
```

I then follow the same process as Term Spreads in computing the Z-scores of real yields.

```
latestZ_realyields=pd.concat([latest1yr_Z_realyield,latest5yr_Z_realyield],axis=1)
latestZ_realyields
```

	Real Yield 1-year Z	Real Yield 5-year Z
1M	-1.92	-1.31
3M	-1.92	-1.27
6M	-1.84	-1.41
1Y	-1.74	-1.39
2Y	-1.62	-1.68
3Y	-1.58	-1.71
4Y	-1.54	-1.95
5Y	-1.51	-1.95
7Y	-1.51	-2.04
10Y	-1.52	-2.16

```
latestZ_realyields.to_csv("latestZ_realyields.csv")
```

Dashboard using Dash Library

I first import all the necessary libraries:

```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import dash_table
import datetime
```

Then I load all the necessary data:

```
df = pd.read_csv (r'C:\Users\j_bia\Documents\projects\FixedIncomeDashboardCreation\PHreferencerates.csv')
```

```
df1=pd.read_csv('latest_Z.csv')
df1.columns=['Spread','1-year Z-score','5-year Z-score']
df1=df1.round(decimals=2)
```

```
realyields=pd.read_csv("latestZ_realyields.csv")
realyields.columns=['Tenor','Real Yield 1-year Z','Real Yield 5-year Z']
```

```
inflation=pd.read_csv("PH_inflation_historical.csv")
inflation['Date']=inflation.apply(lambda row: datetime.datetime(int(row.Year),int(row.Month),1),axis=1)
gdp=pd.read_csv("PH_GDP.csv",index_col=0)
```

To chart GDP properly, I add a date column:

```
def convert_to_date(quarter,year_range):
    #get last 4 characters of year_range
    year=int(year_range[-4:])
    #set month and day for quarters
    if quarter=='Q1':
        month=3
        day=31
    elif quarter=='Q2':
        month=6
        day=30
    elif quarter=='Q3':
        month=9
        day=30
    else:
        month=12
        day=31

    return datetime.datetime(year,month,day)
```

```
gdp['Date']=pd.DataFrame([convert_to_date(gdp['quarter'][x],gdp['year'][x]) for x in range(0,len(gdp))])
gdp.head()
```

	quarter	year	GDP	Date
0	Q1	2000 - 2001	2.4	2001-03-31

Finally, to output all of these in dash:

- Available tenors is stored as a list for our dcc.dropdown later on
- Create base font color for the term spreads with changes depending on the values
- The same is done for real yields but with only 2 conditions.
-

```
## Base: YC, (1) Adding Z-score of spreads, (2) Adding Inflation
app = JupyterDash(__name__)
available_tenors = df.columns[1:]

#create font_color for Term Spreads
font_color1=['rgb(40,40,40)']
labels=list(df1['Spread'])
Zspreadsvals=[list(df1['1-year Z-score']),list(df1['5-year Z-score'])]
for k in range(len(Zspreadsvals)):
    for v in Zspreadsvals[k]:
        if v<=0:
            mix='rgb(200,40,40)'
        elif v< 1:
            mix='rgb(50,50,50)'
        else:
            mix='rgb(35,135,50)'
        font_color1.append(mix)
font_color1=[font_color1[0],font_color1[1:9],font_color1[9:]]

font_color2 = ['rgb(40,40,40)'] +  [['rgb(255,0,0)' if v <= 0 else 'rgb(10,10,10)'
                                     for v in realyields[k]] for k in realyields.columns[1:]]
```

Additional formatting on the Z-score tables along with inputting of values under “cells” portion:

```
# Create Term Spreads Z-table
fig0=go.Figure(data=[go.Table(header=dict(values=list(df1.columns),
                                          fill_color='#FBDDF0',align='center'),
                              cells=dict(values=[labels,Zspreadsvals[0],Zspreadsvals[1]],
                                         fill_color='#F2F2F2',align='center', font=dict(color=font_color1)))])



# Create Real Yields Z-table
fig1=go.Figure(data=[go.Table(header=dict(values=list(realyields.columns),fill_color='#FBDDF0',align='center')
                              cells=dict(values=[realyields['Tenor'],realyields['Real Yield 1-year Z'],
                                                 realyields['Real Yield 5-year Z']],
                                         fill_color='#F2F2F2',align='center',
                                         font=dict(color=font_color2)))])
```

The layout for the graphs where inflation and GDP split the first area in-line each with 50% of the width.

```
app.layout = html.Div([

        html.Div([
            dcc.Graph(
            id='PH_inflation',
            figure=px.line(inflation,x='Date',y='Inflation',title='Philippine Inflation'))],
        style={'width': '50%', 'display': 'inline-block'}),

        html.Div([
            dcc.Graph(
            id='PH GDP',
            figure=px.line(gdp,x='Date',y='GDP',title='Philippine GDP Growth'))],
        style={'width': '50%', 'display': 'inline-block'}),
```

This is followed by the yield curve chart with special “dropdown” option allowing for the viewing of multiple maturities at a time:

```
        html.Div([
            dcc.Dropdown(
                id='yield_curve',
                options=[{'label': i, 'value': i} for i in available_tenors],
                value='10Y',
                multi=True
            ),
            dcc.Graph(id='line_chart')],),
```

Finally, the two Z-score tables each splitting in-line 50% width:

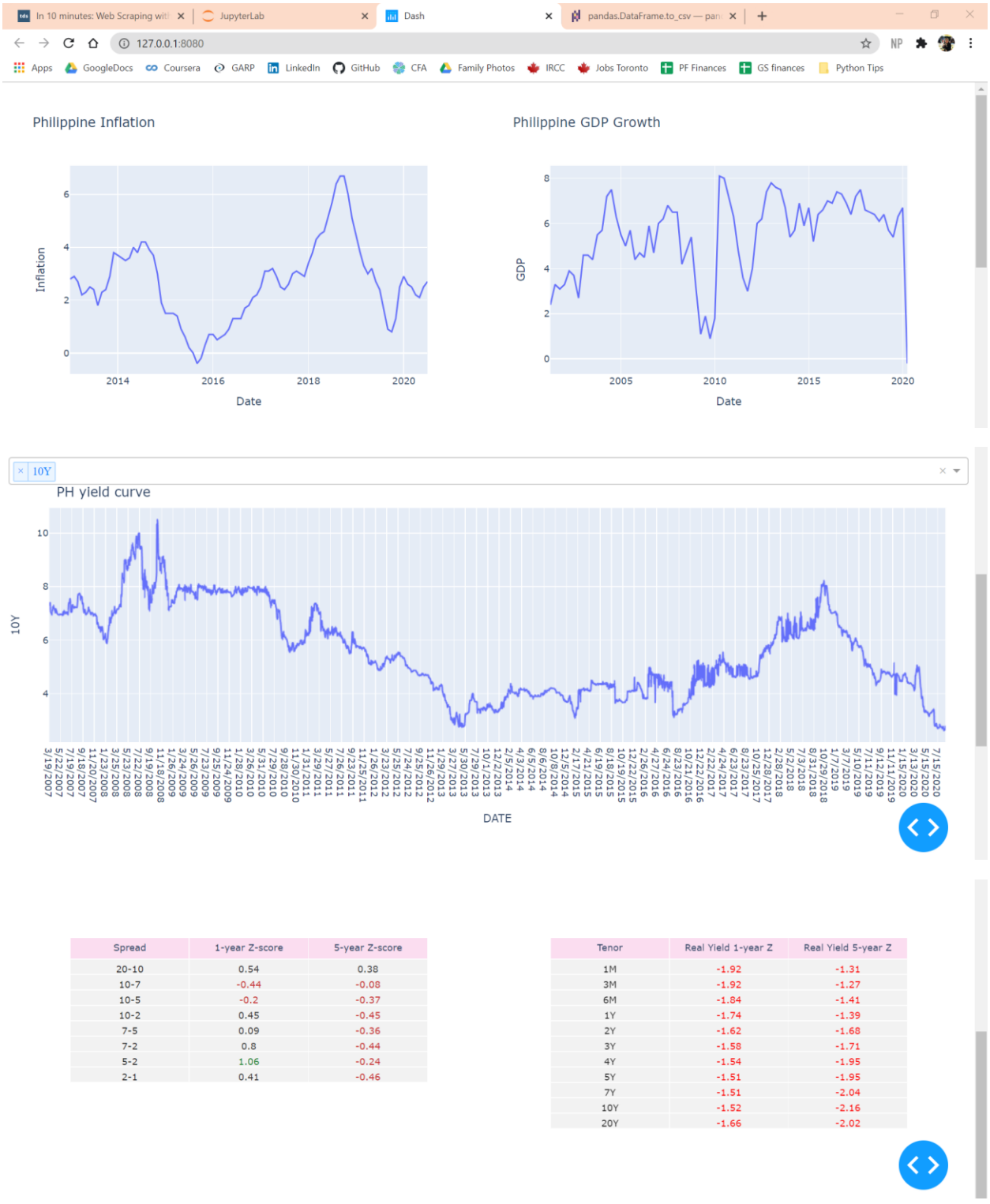
```
        html.Div([
            dcc.Graph(
            id='Spreads_Zscore',
            figure=fig0)],
        style={'width': '50%', 'display': 'inline-block'}),

        html.Div([
            dcc.Graph(
            id='RealYield_Zscore',
            figure=fig1)],
        style={'width': '50%', 'display': 'inline-block'})


])
```

This portion allows the updating of the **line_chart** as identified above depending on input **value**. Finally, some margins are added (left, bottom, top, right):

```
@app.callback(
    Output('line_chart', 'figure'),
    [Input('yield_curve', 'value')])

def update_graph(yaxis_column_name):
    fig = px.line(df, x='DATE',
                  y=yaxis_column_name, title="PH yield curve")

    fig.update_layout(margin={'l': 40, 'b': 40, 't': 30, 'r': 30}, hovermode='closest')

    return fig

app.run_server(mode='external',port=8080)
```

```
Dash app running on http://127.0.0.1:8080/
```

The final dash app appears as below:



Spread	1-year Z-score	5-year Z-score
20-10	0.54	0.38
10-7	-0.44	-0.08
10-5	-0.2	-0.37
10-2	0.45	-0.45
7-5	0.09	-0.36
7-2	0.8	-0.44
5-2	1.06	-0.24
2-1	0.41	-0.46

Tenor	Real Yield 1-year Z	Real Yield 5-year Z
1M	-1.92	-1.31
3M	-1.92	-1.27
6M	-1.84	-1.41
1Y	-1.74	-1.39
2Y	-1.62	-1.68
3Y	-1.58	-1.71
4Y	-1.54	-1.95
5Y	-1.51	-1.95
7Y	-1.51	-2.04
10Y	-1.52	-2.16
20Y	-1.66	-2.02