PX Index in Context of Other European Indices

Soňa Obůrková

Description

Docuementation of the Dashboard includes summary of why these data, target grop, dashboard type, description of each graph, dashboard testing, unexpected graphic situations but primarily **description of functions, their usage, outputs from these functions and examples of their use on data**

Data

Data on indices closing price and currency exchange rates as given in table below were downloaded from TradingView.com. Countries were grouped by the condition of entering the EU to make countries better visible in the dashboard

	Countries typology	**Country**	**Index symbol**	**Stock exchange symbol**
Main index		- Czech Republic	‘PX’	‘PSECZ’
Other indices	*East European stock exchanges indices corresponding with 5th, 6th and 7th EU enlargement*	- Hungary - Romania - Bulgaria - Croatia - Poland	‘BUX’ 'BET' 'BTE' 'CRE' ‘WIG’	‘BET’ ‘BVB' 'VIE' 'VIE' 'GPW'
	EU founding countries + 1st EU enlargement	- The Netherlands - France - Germany - Italy - UK	'AEX' 'CAC' 'DAX' 'FTMIB' 'UKX'	'EURONEXT', ‘EURONEXT' 'XETR' 'TVC' 'FTSE'
	EU 3rd and 4th enlargement,	- Spain - Austria - Sweden	'IBC' 'ATX' 'OMXS30'	'BME' 'VIE' 'OMXSTO'
	Supranational indices and Switzerland	- Scandinavian countries - Europe's leading blue-chip index for the Eurozone - Switzerland	'OMXN40' 'SX5E' 'SMI',	'OMXNORDIC' 'TVC' 'SIX'
Exchange rates			'EURCZK' 'EURHUF' 'EURRON' 'EURPLN' 'EURGBP'	'OANDA' 'OANDA' 'OANDA' 'OANDA' 'OANDA'

Why these data

I am interested in financial sector and would like to make the diploma thesis on on impact of developed financial markets on emerging financial markets with focus on the Czech Republic. This dashboard serves as my analysis of the position of the Czech PX Index in context of developed and emerging indices from other European stock exchanges.

Target group

Those interested in country stock exchange indices, emerging financial markets, Czech stock exchange and its PX index

Dashboard type

Analytical

Description of each graph

Side bar filters

Currency: Button enable converting currencies from EUR (default) into CZK. This applies on line and distribution graphs available in the content part of the dashboard

Year: Year range slider enables to pick years of interest. Applicable on all graphs in the dashboard.

Bar chart: Pearson coefficient of correlation between PX Index and other indices available in the bar chart. The value of the coefficient is available outside of each bar

PX Index Time series: Four graphs available in this figure. The first one shows 'open' and 'close' price of the PX Index in EUR. Under the dropdown one can select graph named 'High' which is the highest price of the index on that day, compared with the average highest value in given period of time. 'Low' shows the lowest price of the PX Index on that day compared with the average lowest value in given period of time. 'High and low' graph combines 'High' and 'Low' graphs into one

PX Distribution plot: Distribution of PX Index closing price. The graph shows curve and the rug. The curve shows normalized frequency of the index and the rug shows the density of the normalized frequency

All Indices Time Series: Line graph of PX Index and Indices from other European stock exchanges in one graph. Beside applying side bar currency and year filters, it is possible to use slider under the graph or graph button filters to visualize data by the last month, last six months, year to date, one year and all available data (default). Also there are buttons to visualize linear data (default) or log data. Indexes were grouped by color: 1. **black** – PX Index, 2. **blue** – East European stock exchanges indices corresponding with 5th, 6th and 7th EU enlargement. 3. **orange** – stock exchanges indices corresponding with EU founding countries + 1st EU enlargement, 4. **green** - stock exchanges indices corresponding with EU 3rd and 4th enlargement, 5. **red** – other indices - supranational European indices and Switzerland.

Dashboard testing

Dashboard testing was carried out thoroughly and bugs were fixed . There should not be any issues at the moment.

Unexpected graphic situations	May be caused in case new dataset is downloaded from TradingView.com in case input parameters used in ‘get_data’ function changed (new parameters included, another removed).

Documentation of functions used in the code

List of functions	get_data, cleaning, update_bar_chart_output, update_main_index_chart_output, update_all_line1_plot, update_dist_plot
Other functions	go.Figure go.Bar go.Scatter create_distplot add_traces update_traces update_layout corr

get_data

Description

Function that downloads desired historical data on the main index, i.e. Czech stock exchange PX index and selected European stock exchange indices from TradingView.com.

Function **returns** downloaded DataFrame that contains columns '**date**' which is the time series when main index is available on Trading View, '**symbol**' of the main index, its '**open**', '**high**' and '**low**' prices as well as close prices of main and other indices and exchange rates.

Usage

```
get_data(main_index, tickers, exchanges)
```

Arguments

'main_index', 'tickers', 'exchanges'

main_index	list of symbols of the main Czech stock exchange index and its stock exchange symbol
tickers	list of symbols of main indices of selected European stock exchanges and currency exchange rates.
exchanges	list of symbles of stock exchanges. Position of the stock exchange symbol corresponds with respective index/exchange rate defined in argument 'tickers'. Argument 'exchanges'

Example

```
main_index = ["PX","PSECZ"]

tickers = ["BUX",'BET','BTE','CRE','WIG','AEX','CAC','DAX','FTMIB','UKX','IBC','ATX','OMXS30','OMXN40','SX5E','SMI',
           'EURCZK','EURHUF','EURRON','EURPLN','EURGBP']

exchanges = ["BET",'BVB','VIE','VIE','GPW','EURONEXT','EURONEXT','XETR','TVC','FTSE','BME','VIE','OMXSTO',
             'OMXNORDIC','TVC','SIX',
             'OANDA','OANDA','SAXO','OANDA','OANDA']

def get_data(main_index, tickers, exchanges):
    try:
        data = pd.DataFrame(tv.get_hist(symbol=main_index[0],exchange=main_index[1],interval=Interval.in_daily,n_bars=7255))
            .rename(columns={"close": main_index[0]})
        data = data.drop(data.columns[5], axis=1)
        data.index = data.index.date

        for ticker, exchange in zip(tickers, exchanges):
            df = tv.get_hist(ticker, exchange, interval=Interval.in_daily, n_bars=7500)
            df = pd.DataFrame(df['close']).rename(columns={'close': ticker})
            df.index = df.index.date  # Změna datetime indexu na date index
            data = pd.merge(data, df, how='left', left_index=True, right_index=True)

        data = data.reset_index()
        data.rename(columns={'index': 'date'}, inplace=True)
        data['date'] = pd.to_datetime(data['date']).dt.strftime('%Y-%m-%d')

        return data
    except:
        print(ticker, exchange)
```

cleaning

Description

DataFrame obtained with the function '**get_data()**' is loaded into the functions. Afterwards the function checks first and end dates in the DataFrame with '**start_date**' and '**end_date**' given to the function as arguments. Creates DataFrame limited by given '**start_date**' and '**end_date'**. Drops rows having number of NaN rows to be greater than 1/3 of total rows number. Resets index and checks if the first and last date of the cleaned DataFrame are same as '**start_date**' and '**end_date**' eventually by how many days were moved. In the end all financial values are changed into EUR.

Function returns number of '**col_to_delete**' (number of deleted columns), '**cleaned_start_date**' (start date of the cleaned DataFrame), '**cleaned_end_date**' (end date of the cleaned DataFrame) and '**clean_df**' that has following columns 'date', columns related to the Czech stock exchange 'symbol', 'open', 'low', 'high' and 'PX' (closing price), closing prices of other indices e.g. 'BUX', 'BET', 'BTE' and exchange rates e.g. 'EURCZK', 'EURPLN', 'EURRON'

Usage

cleaning(df, start_date, end_date)

Arguments

'df', 'start_date', 'end_date'

df — DataFrame obtained from 'get_data()' function

start_date	Preferred first date of the clean DataFrame
end_date	Preferred last date of the clean DataFrame

Example

```
def cleaning(df, start_date, end_date):

    # Checsk if start date is before the first date of the main index or end date is after the last date of the main index
    if start_date < df.date.min() or end_date > df.date.max() or start_date > end_date:
        print("start date is before the first date of the main index or end date is after the last date of the main index")
    else:
        start_date = pd.to_datetime(start_date).strftime('%Y-%m-%d')
        end_date = pd.to_datetime(end_date).strftime('%Y-%m-%d')

    # Create DataFrame for given start_date and end_date as set in the function
        filter_df = df[df["date"] >= start_date]
        filter_df = filter_df[filter_df["date"] <= end_date]
    # Count number of colums having minimal one NaN value
        nan_counts = filter_df.isna().sum()
        rows = filter_df.shape[0]
    # Identify columns that have more than 1/3 of NaN values - these will be removed
        col_to_delete = nan_counts[nan_counts > rows/3].index
    # Drop columns that were identified to be removed
        clean_df = filter_df.drop(col_to_delete, axis=1)
    # Drop NaN rows
        clean_df = clean_df.dropna()
    # Reset index
    clean_df = clean_df.reset_index(drop=True)

    # Confirm, if start_date was moved
    if start_date < clean_df['date'].iloc[0]:
        start_date_diff = pd.to_datetime(clean_df['date'].iloc[0]) - pd.to_datetime(start_date)
        cleaned_start_date = f'Start date was moved by {start_date_diff} on: {clean_df["date"].iloc[0]}'
    else:
        cleaned_start_date = f'No change in start date: {start_date}'

    # Confirm, if end_date was moved
    if end_date < clean_df['date'].iloc[-1]:
        end_date_diff = pd.to_datetime(clean_df['date'].iloc[-1]) - pd.to_datetime(end_date)
        cleaned_end_date = f'End date was moved by {end_date_diff} on: {clean_df["date"].iloc[-1]}'
    else:
        cleaned_end_date = f'No change in end date: {end_date}'

    # Convert all 'clean_df' dataset into EUR
    clean_df.iloc[:,2:6] = clean_df.iloc[:,2:6].div(clean_df['EURCZK'], axis=0)
    clean_df['BUX'] = clean_df['BUX'] / clean_df['EURHUF']
    clean_df['BET'] = clean_df['BET'] / clean_df['EURRON']
    clean_df['WIG'] = clean_df['WIG'] / clean_df['EURPLN']
    clean_df['UKX'] = clean_df['UKX'] / clean_df['EURGBP']
```

update_bar_chart_output

Description	Callback function associated with updating figure of a bar chart based on values selected from a range slider. First the range of DataFrame is updated based on start and end dates selected from a range slider in the dashboard. Afterwards the Pearson coefficient of correlation between PX Index and other indices is calculated. Last but not least the plot is updated through '**add_traces(go.Bar())**', '**updated_traces()**' and '**update_layout()**' functions. The function returns the variable '**fig_bar_updated**' that updates the bar chart
Usage	update_bar_chart_output(selected_years)

Arguments	‘selected_years‘
selected_years	List of two items start and end dates selected from a range slider in the dashboard

Example

On line 260 or later

```
# Callbacks bar chart
@app.callback(
    Output('bar-graph', 'figure'),
    Input('year-range-slider', 'value')
)

def update_bar_chart_output(selected_years):
    fig_bar_updated = go.Figure()

    # Filter clean_df based on selected years
    filtered_bar_df = clean_df[(clean_df['year'] >= selected_years[0]) & (clean_df['year'] <= selected_years[1])]
    filtered_bar_df = filtered_bar_df.iloc[:,5:22]
    filtered_bar_cols = filtered_bar_df.columns[1:]

    filtered_corr_values = [filtered_bar_df['PX'].corr(filtered_bar_df.iloc[:,1:][columns]) for columns in filtered_bar_cols]
    filtered_corr_values = [round(num, 2) for num in filtered_corr_values]

    fig_bar_updated.add_traces(go.Bar(y=filtered_bar_cols,x=filtered_corr_values,orientation='h',text=filtered_corr_values,textposition='outside'))
    fig_bar_updated.update_traces(marker_color='#3283FE', marker_line_color='rgb(0,0,0)', marker_line_width=1, opacity=0.8)
    fig_bar_updated.update_layout(width=250,height=320,autosize=False,margin=dict(l=30, r=10, t=10, b=10),paper_bgcolor='rgba(0,0,0,0)',template="plotly
                xaxis=dict(title='Coefficient of correlation',titlefont_size=12,tickfont_size=11, range=[0, 1.2],dtick=0.2,tickwidth=1),
                yaxis=dict(range=[-1, len(filtered_bar_cols)],tickfont_size=10,autorange="reversed"))

    return fig_bar_updated
```

update_main_index_chart_output

Description	Callback function associated with updating figures of PX Index line charts based on values selected from a range slider and currency selector. First the range of DataFrame is updated based on start and end dates selected from a range slider in the dashboard. If there is the change in the selected currency, the new converted values are calculated and the currency on the yaxis will be changed. Afterwards all four plots in the figure are updated through ‘**add_traces(go.Scatter())**’ and ‘**update_layout()**’ functions. The function returns the variable ‘**fig_line_updated**’ that updates line charts in all four figures
Usage	update_main_index_chart_output(selected_years, selected_currency)
Arguments	‘selected_years‘, ‘selected_currency‘
selected_years	List of two items start and end dates selected from a range slider in the dashboard
selected_currency	Result of radio buttons enabling selecting CZK or EUR

Example

On line 290 or later

```
@app.callback(
    Output('main-index-line-chart', 'figure'),
    Input('year-range-slider', 'value'),
    Input('currency-radio', 'value')
)
# Use the selected_years, and currency in your data processing
def update_main_index_chart_output(selected_years, selected_currency):
    fig_line_updated = go.Figure()

    # Filter clean_df based on selected years
    filtered_main_line_df = clean_df[(clean_df['year'] >= selected_years[0]) & (clean_df['year'] <= selected_years[1])]
    exchange_rate_col = filtered_main_line_df['EURCZK']
    filtered_main_line_date_col = filtered_main_line_df['date']
    filtered_main_line_to_convert_df = filtered_main_line_df.iloc[:,2:6]

    if selected_currency == 'CZK_to_EUR':
        # Convert from CZK to EUR
        converted_values_df = pd.concat([filtered_main_line_date_col, filtered_main_line_to_convert_df], axis=1)
        currency = 'EUR'
    elif selected_currency == 'EUR_to_CZK':
        # Convert from EUR to CZK
        converted_values_df = filtered_main_line_to_convert_df.multiply(exchange_rate_col, axis=0)
        converted_values_df = pd.concat([filtered_main_line_date_col, converted_values_df], axis=1)
        currency = 'CZK'
    else:
        converted_values = []

    # Add Traces
    fig_line_updated.add_trace(go.Scatter(x=list(converted_values_df.date),y=list(converted_values_df.high),name="High",line=dict(color="#1CBE4F")))
    fig_line_updated.add_trace(go.Scatter(x=list(converted_values_df.date),y=[converted_values_df.high.mean()] * len(converted_values_df),
                                          name="High Average",visible=False,line=dict(color="#1CBE4F", dash="dash")))   #33CFA5
    fig_line_updated.add_trace(go.Scatter(x=list(converted_values_df.date),y=list(converted_values_df.low),name="Low",line=dict(color="#C4451C")))
    fig_line_updated.add_trace(go.Scatter(x=list(converted_values_df.date),y=[converted_values_df.low.mean()] * len(converted_values_df.index),
                                          name="Low Average",visible=False,line=dict(color="#C4451C", dash="dash")))   #F06A6A

    # Add Buttons and update plot and its area
    fig_line_updated.update_layout(
        updatemenus=[dict(active=0,buttons=list([
            dict(label="None",method="update",args=[{"visible": [True, False, True, False]},{"title": "PX Index","annotations": []}]),
            dict(label="High",method="update",args=[{"visible": [True, True, False, False]},{"title": "PX Index High","annotations": []}]),
            dict(label="Low",method="update",args=[{"visible": [False, False, True, True]},{"title": "PX Index Low","annotations": []}]),
            dict(label="High and Low",method="update",args=[{"visible": [True, True, True, True]},{"title": "PX Index","annotations": []}]),
            ]),direction="down",showactive=True,xanchor="left",yanchor="top",x=-0.02,y=1.2)],
        width=450,height=350,autosize=False,template="plotly_white",title_text="PX Index Time Series",legend=dict(orientation="h",
                                          yanchor="bottom",y=1.02,xanchor="right",x=1),
        xaxis_title='Year',yaxis_title=currency,)

    return fig_line_updated
```

update_all_line_plot

Description

Callback function associated with updating line charts in All Index Time series figure based on values selected from a range slider and currency selector. First the range of DataFrame is updated based on start and end dates selected from a range slider in the dashboard. If there is the change in the selected currency, the new converted values are calculated and the currency on the yaxis will be changed. Afterwards all sixteen traces in the figure are updated through '**add_traces(go.Scatter())**' and '**update_layout()**' functions.

The function returns the variable '**fig_all_line_updated**' that updates the line chart

Usage

update_all_line_plot(selected_years,selected_currency)

Arguments	‘selected_years‘, ‘selected_currency‘
selected_years	List of two items start and end dates selected from a range slider in the dashboard
selected_currency	Result of radio buttons enabling selecting CZK or EUR

Example

On line 330 or later

```
def update_all_line_plot(selected_years,selected_currency):
    fig_all_line_updated = go.Figure()

    filtered_all_line_df = clean_df[(clean_df['year'] >= selected_years[0]) & (clean_df['year'] <= selected_years[1])]
    exchange_rate_col = filtered_all_line_df['EURCZK']
    filtered_all_line_date_col = filtered_all_line_df['date']
    filtered_all_line_to_convert_df = filtered_all_line_df.iloc[:,5:22]

    if selected_currency == 'CZK_to_EUR':
        # Convert from CZK to EUR
        converted_values_all_df = pd.concat([filtered_all_line_date_col, filtered_all_line_to_convert_df], axis=1)
        currency = 'EUR'
    elif selected_currency == 'EUR_to_CZK':
        # Convert from EUR to CZK
        converted_values_all_df = filtered_all_line_to_convert_df.multiply(exchange_rate_col, axis=0)
        converted_values_all_df = pd.concat([filtered_all_line_date_col, converted_values_all_df], axis=1)
        currency = 'CZK'
    else:
        converted_values = []

    # Main index
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['PX'], name='PX', line=dict(color='#325A9B', wi
    # East European Indices - fifth and sixth wave - "BUX","BET","BTE","CRE","WIG",
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['BUX'], name='BUX', line=dict(color='#3283FE', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['BET'], name='BET', line=dict(color='#3283FE', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['BTE'], name='BTE', line=dict(color='#3283FE', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['WIG'], name='WIG', line=dict(color='#3283FE', 
    # EU Founders + first wave of extensions - 'AEX','CAC','DAX','FTMIB','UKX',
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['AEX'], name='AEX', line=dict(color='#FEAF16', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['CAC'], name='CAC', line=dict(color='#FEAF16', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['DAX'], name='DAX', line=dict(color='#FEAF16', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['FTMIB'], name='FTMIB', line=dict(color='#FEAF1
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['UKX'], name='UKX', line=dict(color='#FEAF16', 
    # Third and fourth waves of extensions - 'IBC','ATX','OMXS30',
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['IBC'], name='IBC', line=dict(color='#1CBE4F', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['ATX'], name='ATX', line=dict(color='#1CBE4F', 
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['OMXS30'], name='OMXS30', line=dict(color='#1CBE
    # Supranational indices - 'OMXN40','SX5E','SMI',
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['OMXN40'], name='OMXN40', line=dict(color='#C44
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['SX5E'], name='SX5E', line=dict(color='#C4451C'
    fig_all_line_updated.add_trace(go.Scatter(x=filtered_all_line_date_col, y=converted_values_all_df['SMI'], name='SMI', line=dict(color='#C4451C', 

    # Edit the layout
    fig_all_line_updated.update_layout(title='All Indices Time Series',xaxis_title='Year',yaxis_title=currency,width=800,height=350,template="plotly_
    fig_all_line_updated.update_layout(
        updatemenus=[dict(visible=True,type="buttons",
             buttons=[
                dict(label="Linear",method="relayout",args=[{"yaxis.type": "linear"}]),
                dict(label="Log",method="relayout",args=[{"yaxis.type": "log"}]),],
                direction="right",showactive=True,xanchor="center",yanchor="top",y=1.3,x=0.9)]);
    # Add range slider
    fig_all_line_updated.update_layout(xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,label="1m",step="month",stepmode="backward"),
                dict(count=6,label="6m",step="month",stepmode="backward"),
                dict(count=1,label="YTD",step="year",stepmode="todate"),
                dict(count=1,label="1y",step="year",stepmode="backward"),
                dict(step="all")])),
        rangeslider=dict(visible=True),type="date"))

    return fig_all_line_updated
```

update_dist_plot

Description	Callback function associated with updating distribution plot based on values selected from a range slider and currency selector. First the range of DataFrame is updated based on start and end dates selected from a range slider in the dashboard. If there is the change in the selected currency, the new

converted values are calculated and the currency on the yaxis will be changed. Afterwards the distribution plot in the figure is updated through '**create_distplot()**' and '**update_layout()**' functions.

The function returns the variable '**fig_dist_pdated**' that updates the distribution plot

Usage update_dist_plot(selected_years,selected_currency)

Arguments 'selected_years', 'selected_currency'

selected_years	List of two items start and end dates selected from a range slider in the dashboard
selected_currency	Result of radio buttons enabling selecting CZK or EUR

Example

On line 400 or later

```
def update_dist_plot(selected_years, selected_currency):
    # Filter clean_df based on selected years
    filtered_dist_df = clean_df[(clean_df['year'] >= selected_years[0]) & (clean_df['year'] <= selected_years[1])]

    if selected_currency == 'CZK_to_EUR':
        # Convert from CZK to EUR
        converted_hist = [filtered_dist_df['PX']]
        currency = 'EUR'
    elif selected_currency == 'EUR_to_CZK':
        # Convert from EUR to CZK
        converted_hist = [filtered_dist_df['PX'] * filtered_dist_df['EURCZK']]
        currency = 'CZK'
    else:
        converted_values = []

    #hist_df = [clean_df['PX']]
    #group_labels = ['PX Index']
    #colors = ['#3283FE']

    fig_dist_pdated = ff.create_distplot(converted_hist, group_labels, show_hist=False, colors=colors)
    fig_dist_pdated.update_layout(title_text='PX Index Distribution',xaxis_title=currency,width=450,height=350,autosize=True,template="plotly_white",
        legend=dict(orientation="h",yanchor="bottom",y=1.02,xanchor="right",x=1))

    return fig_dist_pdated
```

Other functions used to create graphs and links on their documentations

go.Figure

plotly.graph_objects.Figure
https://plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html

go.Bar

plotly.graph_objects.Bar
https://plotly.com/python-api-reference/generated/plotly.graph_objects.Bar.html

go.Scatter

plotly.graph_objects.Scatter
https://plotly.com/python-api-reference/generated/plotly.graph_objects.Scatter.html

create_distplot

plotly.figure_factory.create_distplot
https://plotly.github.io/plotly.py-docs/generated/plotly.figure_factory.create_distplot.html

add_traces

plotly.graph_objects.Figure
https://plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html

update_traces

plotly.graph_objects.Figure
https://plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html

update_layout

plotly.graph_objects.Figure
https://plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html

corr

pandas.DataFrame.corr
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html