extended_case_4

May 13, 2020

1 Analyzing an automated Bitcoin trading strategy

1.1 Introduction

Business Context. As part of a crypto trading desk, you've been tasked with analyzing an automated Bitcoin trading strategy. The creator of the strategy has made historical data of the strategy available to you. The trades are on leveraged trading platforms: Bitmex & Deribit. Each trade can be long (buy Bitcoin) or short (sell Bitcoin). The firm is interested in visualizing the returns offered by the strategy over selected periods of time and comparing it to the returns of just passively holding Bitcoin. They are also interested in seeing individual trade data as well as visualizing returns for each individual trade. Finally, they would like to see how the strategy does, if the degree of leverage (essentially a way of borrowing money to increase risk) is increased.

Business Problem. Your task is to **make an interactive dashboard that your firm can use to visualize the strategy's trades and performance in the manners they indicated above.** The dashboard should be usable via a web browser like Chrome on the public Internet.

Analytical Context. In the current case, we will be using Dash by Plotly to develop the dashboard. This time, we will be using a file `aggr.csv` which was made available by the creators of the trading strategy. You should put together everything in an `app.py` file as we go along. This time, however, we will be hosting the data in an AWS RDS instance and running the app's relevant code from an AWS EC2 instance.

1.2 Checking the data source

We will be reading data from the file `aggr.csv` which was made available by the creators of the strategy. Let's load it into a `pandas` dataframe and look at the columns available to us for use:

[1]:
```
import pandas as pd

df = pd.read_csv('aggr.csv', parse_dates=['Entry time'])
print(df.head(5))
```

```
   Number Trade type          Entry time                    Exposure  \
0     505      Short 2019-09-16 12:30:00  2 days 11 hours 30 minutes
1     504       Long 2019-09-15 22:00:00         14 hours 30 minutes
2     503      Short 2019-09-09 08:00:00             6 days 14 hours
3     502       Long 2019-09-07 15:00:00              1 day 17 hours
```

```
4     501      Short 2019-09-06 20:00:00                    19 hours

   Entry balance  Exit balance     Profit  Pnl (incl fees) Exchange  Margin  \
0     1499.70810    1497.45854  -2.249562           -0.15   Bitmex       1
1     1523.08014    1499.23112 -23.849025           -1.57   Bitmex       1
2     1548.75311    1522.55620 -26.196910           -1.69   Bitmex       1
3     1594.40783    1547.82139 -46.586446           -2.92   Bitmex       1
4     1604.98017    1594.19207 -10.788103           -0.67   Bitmex       1

   BTC Price
0    10262.5
1    10304.4
2    10310.0
3    10484.3
4    10306.9
```

The available data columns are as follows:

1. **Number**: The serial number of the trade
2. **Trade type**: Whether the trade was a "Long" or a "Short"
3. **Entry time**: The entry time of the trade
4. **Exposure**: The length of the trade before exiting
5. **Entry balance**: BTC balance at entry
6. **Exit balance**: BTC balance at exit
7. **Profit**: Profit (in BTC) for the trade (can be positive or negative)
8. **Pnl (incl fees)**: Profit or loss of trade in %
9. **Exchange**: Exchange the trade was executed on
10. **Margin**: Margin or leverage used in the trade (hereforth, margin/leverage will be used interchangably)
11. **BTC Price**: Closing price of BTC on entry date

1.3 Planning the dashboard

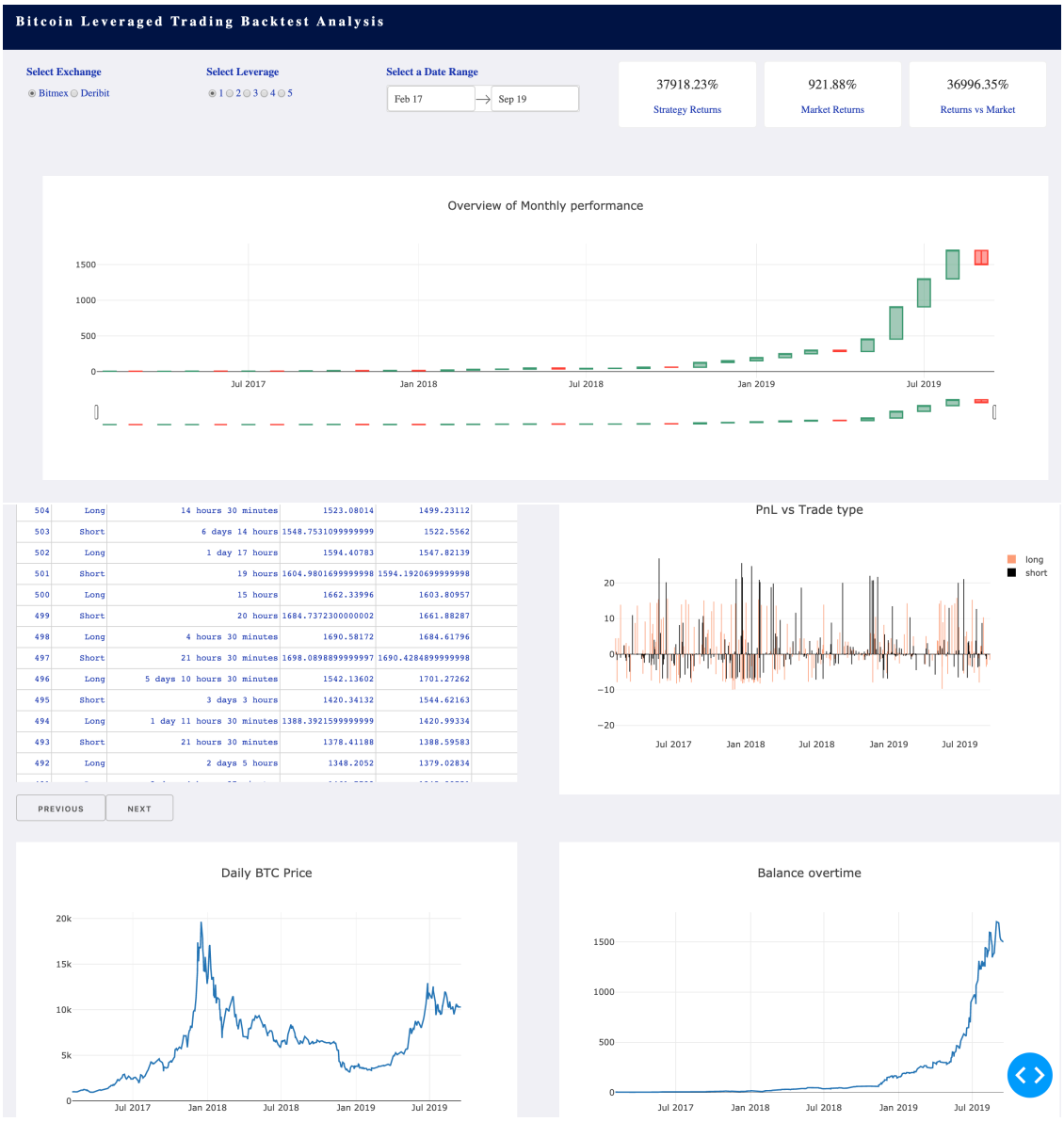
Let's go over the requirements that our firm has to determine what ought to be on our dashboard.

1. Our firm wants to know the returns of the strategy over time relative to just holding Bitcoin. This suggests that we ought to have some text boxes which show the percentage returns for each, as well as the differential.
2. The firm wants to look at selected time periods with varying degrees of leverage. This means we should have custom selectors that allow us to pick date ranges and leverage levels.
3. Since the firm wants to visualize the performance, we should have a few (probably line) graphs showing the evolution of returns for the strategy and for Bitcoin.
4. Finally, the firm wants to look at individual trade data and visualize returns per trade. This suggests that we should have a table of trades as well as a bar chart showing the return percentage per trade across time. Since we are going to be analyzing data for one exchange with a specific value of leverage, these will form the selectors in our dashboard.

Since there are two exchanges, we'll also add a selector for which exchange.

1.3.1 Ideal end state of the dashboard

Given the requirements we have scoped above, the following seems like a reasonable design for the final dashboard:



1.4 Setting up the Dash app

Let's get started building the above design. Copy the following into an `app.py` file:

```
import pandas as pd
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objects as go
```

```
df = pd.read_csv('aggr.csv', parse_dates=['Entry time'])

app = dash.Dash(__name__, external_stylesheets=['https://codepen.io/uditagarwal/pen/oNvwKNP.cs

if __name__ == "__main__":
    app.run_server(debug=True)
```

We will start by adding the selectors to the dashboard. The first part of the layout will be the title of our analysis. The class names of the divs are based on the styling information specified in the CSS file which is imported during setup of the app. Add the following to your `app.py` file:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    )]
)
```

1.5 Adding the selectors

1.5.1 Exchange selector

We will now add the exchange selector with ID `exchange-select`, which will be a radio button based on unique values in the `Exchange` column in our dataset. The Dash component for this is `dcc.RadioItems()`, which has an `options` parameter to specify the options that will be rendered on the page. For more information, see here: https://dash.plot.ly/dash-core-components/radioitems.

Note that we will render all of our selectors in a twelve-column row named `twelve column card` which appears horizontally at the top of the dashboard:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    ),
    html.Div(
        className="row app-body",
        children=[
            html.Div(
                className="twelve columns card",
                children=[
                    html.Div(
                        className="padding row",
                        children=[
```

```
                            html.Div(
                                className="two columns card",
                                children=[
                                    html.H6("Select Exchange",),
                                    dcc.RadioItems(
                                        id="exchange-select",
                                        options=[
                                            {'label': label, 'value': label} for label in df['
                                        ],
                                        value='Bitmex',
                                        labelStyle={'display': 'inline-block'}
                                    )
                                ]
                            )
                        ]
                )
            ])
    ])
])
```

1.5.2 Exercise 1:

Add the leverage selector with ID `leverage-select`, which will also be a radio selection box based on unique values in the `Margin` column of our dataset. Add it next to the exchange selector as shown in the design. Place it inside a div named `two columns card 2`.

Answer. Our recommended solution is shown below:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    ),
    html.Div(
        className="row app-body",
        children=[
            html.Div(
                className="twelve columns card",
                children=[
                    html.Div(
                        className="padding row",
                        children=[
                            html.Div(
                                className="two columns card",
                                children=[
                                    html.H6("Select Exchange",),
```

```
                                    dcc.RadioItems(
                                        id="exchange-select",
                                        options=[
                                            {'label': label, 'value': label} for label in df['
                                        ],
                                        value='Bitmex',
                                        labelStyle={'display': 'inline-block'}
                                    )
                                ]
                            ),
                            # Leverage Selector
                            html.Div(
                                className="two columns card",
                                children=[
                                    html.H6("Select Leverage"),
                                    dcc.RadioItems(
                                        id="leverage-select",
                                        options=[
                                            {'label': str(label), 'value': str(label)} for labe
                                        ],
                                        value='1',
                                        labelStyle={'display': 'inline-block'}
                                    ),
                                ]
                            ),
                        ]
                )
            ])
    ])
])
```

1.5.3 Exercise 2:

Add the date range selector with ID `date-range-select`. The start date and end date of the selector is going to be the minimum and maximum of `Entry time`, respectively. Set the `display_format` parameter to `MMM YY`. Place it inside a div named `three columns card`.

Answer. Our recommended solution is shown below:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    ),
    html.Div(
        className="row app-body",
```

```
children=[
    html.Div(
        className="twelve columns",
        children=[
            html.Div(
                className="padding row",
                children=[
                    html.Div(
                        className="two columns card",
                        children=[
                            html.H6("Select Exchange",),
                            dcc.RadioItems(
                                id="exchange-select",
                                options=[
                                    {'label': label, 'value': label} for label in df['
                                ],
                                value='Bitmex',
                                labelStyle={'display': 'inline-block'}
                            )
                        ]
                    ),
                    # Leverage Selector
                    html.Div(
                        className="two columns card",
                        children=[
                            html.H6("Select Leverage"),
                            dcc.RadioItems(
                                id="leverage-select",
                                options=[
                                    {'label': str(label), 'value': str(label)} for labe
                                ],
                                value='1',
                                labelStyle={'display': 'inline-block'}
                            ),
                        ]
                    ),
                    html.Div(
                        className="three columns card",
                        children=[
                            html.H6("Select a Date Range"),
                            dcc.DatePickerRange(
                                id="date-range-select",
                                display_format="MMM YY",
                                start_date=df['Entry time'].min(),
                                end_date=df['Entry time'].max()
                            ),
                        ]
                    ),
```

```
                    ]
                )
            ])
    ])
])
```

1.6 Adding returns text boxes

We will be adding three text values for "Strategy Returns", "Market Returns", and "Strategy vs. Market Returns" to the `twelve columns card`. These will give us an overview of the performance of the automated trading strategy. The values will change based on the selections of time period, exchange, and leverage:

1. **Strategy Returns**: The returns offered by the strategy
2. **Market Returns**: The returns offered by investing in BTCUSD directly
3. **Strategy vs. Market**: Returns of the strategy vs. returns of the market

Let's add a div for each of these to the layout. The layout of these is based on the `pretty_container` CSS class which renders them as boxes next to our selectors:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    ),
    html.Div(
        className="row app-body",
        children=[
            html.Div(
                className="twelve columns card",
                children=[
                    html.Div(
                        className="padding row",
                        children=[
                            html.Div(
                                className="two columns card",
                                children=[
                                    html.H6("Select Exchange",),
                                    dcc.RadioItems(
                                        id="exchange-select",
                                        options=[
                                            {'label': label, 'value': label} for label in df['E
                                        ],
                                        value='Bitmex',
                                        labelStyle={'display': 'inline-block'}
                                    )
```

```
    ]
),
# Leverage Selector
html.Div(
    className="two columns card",
    children=[
        html.H6("Select Leverage"),
        dcc.RadioItems(
            id="leverage-select",
            options=[
                {'label': str(label), 'value': str(label)} for labe
            ],
            value='1',
            labelStyle={'display': 'inline-block'}
        ),
    ]
),
html.Div(
    className="three columns card",
    children=[
        html.H6("Select a Date Range"),
        dcc.DatePickerRange(
            id="date-range",
            display_format="MMM YY",
            start_date=df['Entry time'].min(),
            end_date=df['Entry time'].max()
        ),
    ]
),
html.Div(
    id="strat-returns-div",
    className="two columns indicator pretty_container",
    children=[
        html.P(id="strat-returns", className="indicator_value"),
        html.P('Strategy Returns', className="twelve columns indica
    ]
),
html.Div(
    id="market-returns-div",
    className="two columns indicator pretty_container",
    children=[
        html.P(id="market-returns", className="indicator_value"),
        html.P('Market Returns', className="twelve columns indicato
    ]
),
html.Div(
    id="strat-vs-market-div",
    className="two columns indicator pretty_container",
```

```
                                children=[
                                    html.P(id="strat-vs-market", className="indicator_value"),
                                    html.P('Strategy vs. Market Returns', className="twelve col
                                ]
                            ),
                        ]
                    )
                ])
            ])
        ])
```

1.6.1 Exercise 3:

Add a callback function which updates the start and end date values of the date range selector based on the exchange selected (since each exchange contains data on different time periods). Please type code for the callback in the cell below in addition to having it in your `app.py` file.

Answer. Our recoommended solution is shown below:

[ ]:
```
# NOTE: Do NOT run this cell!!! It is for instructional purposes only - it will␣
↪NOT work!
@app.callback(
    [
        dash.dependencies.Output('date-range', 'start_date'),
        dash.dependencies.Output('date-range', 'end_date'),
    ],
    (dash.dependencies.Input('exchange-select', 'value'),)
)
def update_daterange(value):
    # Update the start and end date on the Chart based on the exchange selected
    dff = df[df['Exchange'] == value]
    return (dff['Entry time'].min(), dff['Entry time'].max())
```

1.7 Adding a monthly overview candlestick plot

In order to shown the performance of the strategy on a month-by-month basis, we use a monthly candlestick chart which shows the start and end value of the strategy portfolio after making all the trades for that month. It's fairly typical for financial trade information to be represented in this type of chart. The x-axis is going to be the year-month combination and the candles will be based on balances at the start and end of each month. See here for a reference guide on candlestick charts: https://plot.ly/python/candlestick-charts/.

We will add a new div to our layout for this graph with id `monthly-chart`:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
```

```
        ],
        className='study-browser-banner row'
),
html.Div(
    className="row app-body",
    children=[
        html.Div(
            className="twelve columns card",
            children=[
                html.Div(
                    className="padding row",
                    children=[
                        html.Div(
                            className="two columns card",
                            children=[
                                html.H6("Select Exchange",),
                                dcc.RadioItems(
                                    id="exchange-select",
                                    options=[
                                        {'label': label, 'value': label} for label in df['
                                    ],
                                    value='Bitmex',
                                    labelStyle={'display': 'inline-block'}
                                )
                            ]
                        ),
                        # Leverage Selector
                        html.Div(
                            className="two columns card",
                            children=[
                                html.H6("Select Leverage"),
                                dcc.RadioItems(
                                    id="leverage-select",
                                    options=[
                                        {'label': str(label), 'value': str(label)} for labe
                                    ],
                                    value='1',
                                    labelStyle={'display': 'inline-block'}
                                ),
                            ]
                        ),
                        html.Div(
                            className="three columns card",
                            children=[
                                html.H6("Select a Date Range"),
                                dcc.DatePickerRange(
                                    id="date-range",
                                    display_format="MMM YY",
```

```
                                    start_date=df['Entry time'].min(),
                                    end_date=df['Entry time'].max()
                                ),
                            ]
                        ),
                        html.Div(
                            id="strat-returns-div",
                            className="two columns indicator pretty_container",
                            children=[
                                html.P(id="strat-returns", className="indicator_value"),
                                html.P('Strategy Returns', className="twelve columns indica
                            ]
                        ),
                        html.Div(
                            id="market-returns-div",
                            className="two columns indicator pretty_container",
                            children=[
                                html.P(id="market-returns", className="indicator_value"),
                                html.P('Market Returns', className="twelve columns indicato
                            ]
                        ),
                        html.Div(
                            id="strat-vs-market-div",
                            className="two columns indicator pretty_container",
                            children=[
                                html.P(id="strat-vs-market", className="indicator_value"),
                                html.P('Strategy vs. Market Returns', className="twelve col
                            ]
                        ),
                    ]
                )
            ]),
            html.Div(
                id="monthly-chart-div",
                children=[
                    dcc.Graph(
                        id="monthly-chart",
                        figure={
                            'data': []
                        }
                    )
                ]
            ),
        ])
    ])
```

1.7.1 Exercise 4:

To ease the construction of all our callback functions, create a helper function `filter_df()` that will be re-used through the app. Our DataFrame will need to be filtered frequently based on the selection for `Exchange`, `Margin`, `start_date`, and `end_date`. Please type code for the function in the cell below in addition to having it in your `app.py` file.

Answer. Our recommended solution is shown below:

[6]:
```
def filter_df(df, exchange, leverage, start_date, end_date):
    dff = df[(df['Exchange'] == exchange) & (df['Entry time'] > start_date) &␣
↪(df['Entry time'] < end_date) & (df['Margin'] == int(leverage))]
    dff.sort_values(by='Entry time', ascending=False)
    dff['YearMonth'] = pd.to_datetime(dff['Entry time'].map(lambda x: "{}-{}".
↪format(x.year, x.month)))
    return dff
```

1.7.2 Exercise 5:

Write a callback function that uses the filtered dataset to calculate the monthly returns of our strategy and create the candlestick chart. The candlestick chart uses the open, close, high, and low values to plot monthly candlesticks. Since we don't have the high and low value for each month, we will just default the low and high values to the entry and exit values (i.e. there will be no "wicks" to the candles). Please type code in the cell below in addition to having it in your `app.py` file.

Answer. Our recommended solution is shown below:

[8]:
```
# NOTE: Do NOT run this cell!!! It is for instructional purposes only - it will␣
↪NOT work!
def calc_returns_over_month(dff):
    out = []

    for name, group in dff.groupby('YearMonth'):
        exit_balance = group.head(1)['Exit balance'].values[0]
        entry_balance = group.tail(1)['Entry balance'].values[0]
        monthly_return = (exit_balance*100 / entry_balance)-100
        out.append({
            'month': name,
            'entry': entry_balance,
            'exit': exit_balance,
            'monthly_return': monthly_return
        })
    return out, btc_returns, strat_returns, strat_vs_market

@app.callback(
    [
        dash.dependencies.Output('monthly-chart', 'figure'),
    ],
```

```
    (
        dash.dependencies.Input('exchange-select', 'value'),
        dash.dependencies.Input('leverage-select', 'value'),
        dash.dependencies.Input('date-range', 'start_date'),
        dash.dependencies.Input('date-range', 'end_date'),

    )
)
def update_monthly(exchange, leverage, start_date, end_date):
    dff = filter_df(df, exchange, leverage, start_date, end_date) # Filter the␣
 ↪dataframe
    data = calc_returns_over_month(dff) # Calc monthly returns
    return {
        'data': [
            go.Candlestick(
                open=[each['entry'] for each in data],
                close=[each['exit'] for each in data],
                x=[each['month'] for each in data],
                low=[each['entry'] for each in data],
                high=[each['exit'] for each in data]
            )
        ],
        'layout': {
            'title': 'Overview of Monthly performance'
        }
    }
```

```
      ␣
↪---------------------------------------------------------------------------

       NonExistentIdException                    Traceback (most recent call␣
↪last)

       <ipython-input-8-27096563ac68> in <module>
         7         dash.dependencies.Input('leverage-select', 'value'),
         8         dash.dependencies.Input('date-range', 'start_date'),
   ----> 9         dash.dependencies.Input('date-range', 'end_date'),
        10
        11     )


       ~/.virtualenvs/correlationone/lib/python3.6/site-packages/dash/dash.py␣
↪in callback(self, output, inputs, state)
      1150     # pylint: disable=dangerous-default-value
      1151     def callback(self, output, inputs=[], state=[]):
   -> 1152         self._validate_callback(output, inputs, state)
```

```
      1153
      1154         callback_id = _create_callback_id(output)


       ~/.virtualenvs/correlationone/lib/python3.6/site-packages/dash/dash.py␣
↪in _validate_callback(self, output, inputs, state)
       865                                you can suppress this exception by␣
↪setting
       866                                `suppress_callback_exceptions=True`.
   --> 867                            ''').format(arg_id, all_ids))
       868
       869                        component = (


       NonExistentIdException:
                                  Attempting to assign a callback to the
                                  component with the id "monthly-chart" but no
                                  components with id "monthly-chart" exist in the
                                  app's layout.


                                  Here is a list of IDs in layout:
   ['exchange-select']


                                  If you are assigning callbacks to components
                                  that are generated by other callbacks
                                  (and therefore not in the initial layout), then
                                  you can suppress this exception by setting
                                  `suppress_callback_exceptions=True`.
```

1.8 Multi-output callback functions

Since we can calculate the text values for ”Strategy Returns”, ”Market Returns”, and ”Strategy vs. Market Returns” with the same callback, lets add them as output to our `update_monthly()` function too. The output of the callback function is going to be multiple tuples which connect back to the corresponding value in the output list:

```
def calc_returns_over_month(dff):
    out = []

    for name, group in dff.groupby('YearMonth'):
        exit_balance = group.head(1)['Exit balance'].values[0]
        entry_balance = group.tail(1)['Entry balance'].values[0]
        monthly_return = (exit_balance*100 / entry_balance)-100
```

```
        out.append({
            'month': name,
            'entry': entry_balance,
            'exit': exit_balance,
            'monthly_return': monthly_return
        })
    return out


def calc_btc_returns(dff):
    btc_start_value = dff.tail(1)['BTC Price'].values[0]
    btc_end_value = dff.head(1)['BTC Price'].values[0]
    btc_returns = (btc_end_value * 100/ btc_start_value)-100
    return btc_returns

def calc_strat_returns(dff):
    start_value = dff.tail(1)['Exit balance'].values[0]
    end_value = dff.head(1)['Entry balance'].values[0]
    returns = (end_value * 100/ start_value)-100
    return returns

@app.callback(
    [
        dash.dependencies.Output('monthly-chart', 'figure'),
        dash.dependencies.Output('market-returns', 'children'),
        dash.dependencies.Output('strat-returns', 'children'),
        dash.dependencies.Output('market-vs-returns', 'children'),
    ],
    (
        dash.dependencies.Input('exchange-select', 'value'),
        dash.dependencies.Input('leverage-select', 'value'),
        dash.dependencies.Input('date-range', 'start_date'),
        dash.dependencies.Input('date-range', 'end_date'),

    )
)
def update_monthly(exchange, leverage, start_date, end_date):
    dff = filter_df(df, exchange, leverage, start_date, end_date)
    data = calc_returns_over_month(dff)
    btc_returns = calc_btc_returns(dff)
    strat_returns = calc_strat_returns(dff)
    strat_vs_market = strat_returns - btc_returns

    return {
        'data': [
            go.Candlestick(
                open=[each['entry'] for each in data],
                close=[each['exit'] for each in data],
```

```
                x=[each['month'] for each in data],
                low=[each['entry'] for each in data],
                high=[each['exit'] for each in data]
            )
        ],
        'layout': {
            'title': 'Overview of Monthly performance'
        }
    }, f'{btc_returns:0.2f}%', f'{strat_returns:0.2f}%', f'{strat_vs_market:0.2f}%'
```

1.9 Adding a data table to our plot

Recall that we wish to occasionally analyze specific trades over a given time period. A data table containing information per trade is suited for this. A reference guide for data tables in Dash is here: https://dash.plot.ly/datatable.

To add a data table, we need to import the `dash_table` library. Let's add the table to our layout by setting a new div and placing it beneath the monthly plot:

```
import dash_table

app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    ),
    html.Div(
        className="row app-body",
        children=[
            html.Div(
                className="twelve columns card",
                children=[
                    html.Div(
                        className="padding row",
                        children=[
                            html.Div(
                                className="two columns card",
                                children=[
                                    html.H6("Select Exchange",),
                                    dcc.RadioItems(
                                        id="exchange-select",
                                        options=[
                                            {'label': label, 'value': label} for label in df['E
                                        ],
                                        value='Bitmex',
                                        labelStyle={'display': 'inline-block'}
```

```
        )
    ]
),
# Leverage Selector
html.Div(
    className="two columns card",
    children=[
        html.H6("Select Leverage"),
        dcc.RadioItems(
            id="leverage-select",
            options=[
                {'label': str(label), 'value': str(label)} for labe
            ],
            value='1',
            labelStyle={'display': 'inline-block'}
        ),
    ]
),
html.Div(
    className="three columns card",
    children=[
        html.H6("Select a Date Range"),
        dcc.DatePickerRange(
            id="date-range",
            display_format="MMM YY",
            start_date=df['Entry time'].min(),
            end_date=df['Entry time'].max()
        ),
    ]
),
html.Div(
    id="strat-returns-div",
    className="two columns indicator pretty_container",
    children=[
        html.P(id="strat-returns", className="indicator_value"),
        html.P('Strategy Returns', className="twelve columns indica
    ]
),
html.Div(
    id="market-returns-div",
    className="two columns indicator pretty_container",
    children=[
        html.P(id="market-returns", className="indicator_value"),
        html.P('Market Returns', className="twelve columns indicato
    ]
),
html.Div(
    id="strat-vs-market-div",
```

```
                                    className="two columns indicator pretty_container",
                                    children=[
                                        html.P(id="strat-vs-market", className="indicator_value"),
                                        html.P('Strategy vs. Market Returns', className="twelve col
                                    ]
                                ),
                            ]
                        )
                ]),
                html.Div(
                    className="twelve columns card",
                    children=[
                        dcc.Graph(
                            id="monthly-chart",
                            figure={
                                'data': []
                            }
                        )
                    ]
                ),
                html.Div(
                        className="padding row",
                        children=[
                            html.Div(
                                className="six columns card",
                                children=[
                                    dash_table.DataTable(
                                        id='table',
                                        columns=[
                                            {'name': 'Number', 'id': 'Number'},
                                            {'name': 'Trade type', 'id': 'Trade type'},
                                            {'name': 'Exposure', 'id': 'Exposure'},
                                            {'name': 'Entry balance', 'id': 'Entry balance'},
                                            {'name': 'Exit balance', 'id': 'Exit balance'},
                                            {'name': 'Pnl (incl fees)', 'id': 'Pnl (incl fees)'},
                                        ],
                                        style_cell={'width': '50px'},
                                        style_table={
                                            'maxHeight': '450px',
                                            'overflowY': 'scroll'
                                        },
                                    )
                                ]
                            ),
                        ]
                    ),
            ])
        ])
```

We pass the columns as a list of dictionaries, with `name` and `id` as the keys. Since we will be plotting the trade type, exposure, entry balance, etc. we pass these as the columns to our data table. The styling of the data table is important because we only want the height of the table to be 450px so that it doesn't overflow on the dashboard. Setting the `style_table` attribute with `maxHeight` and `overflowY` allows us to control for this. The table data will be rendered using the following callback function:

[ ]:
```
# NOTE: Do NOT run this cell!!! It is for instructional purposes only - it will␣
↪NOT work!
@app.callback(
    dash.dependencies.Output('table', 'data'),
    (
        dash.dependencies.Input('exchange-select', 'value'),
        dash.dependencies.Input('leverage-select', 'value'),
        dash.dependencies.Input('date-range', 'start_date'),
        dash.dependencies.Input('date-range', 'end_date'),
    )
)
def update_table(exchange, leverage, start_date, end_date):
    dff = filter_df(df, exchange, leverage, start_date, end_date)
    return dff.to_dict('records')
```

1.10 Bar chart for visualizing trades

Let's now add the bar chart to help visualize our trades. This can help us see the distribution of long vs. short trades as well as the most profitable trades over the selected period. Let's add this to our layout next to the data table, with the ID `pnl-types`:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    ),
    html.Div(
        className="row app-body",
        children=[
            html.Div(
                className="twelve columns card",
                children=[
                    html.Div(
                        className="padding row",
                        children=[
                            html.Div(
                                className="two columns card",
                                children=[
```

```
        html.H6("Select Exchange",),
        dcc.RadioItems(
            id="exchange-select",
            options=[
                {'label': label, 'value': label} for label in df['
            ],
            value='Bitmex',
            labelStyle={'display': 'inline-block'}
        )
    ]
),
# Leverage Selector
html.Div(
    className="two columns card",
    children=[
        html.H6("Select Leverage"),
        dcc.RadioItems(
            id="leverage-select",
            options=[
                {'label': str(label), 'value': str(label)} for labe
            ],
            value='1',
            labelStyle={'display': 'inline-block'}
        ),
    ]
),
html.Div(
    className="three columns card",
    children=[
        html.H6("Select a Date Range"),
        dcc.DatePickerRange(
            id="date-range",
            display_format="MMM YY",
            start_date=df['Entry time'].min(),
            end_date=df['Entry time'].max()
        ),
    ]
),
html.Div(
    id="strat-returns-div",
    className="two columns indicator pretty_container",
    children=[
        html.P(id="strat-returns", className="indicator_value"),
        html.P('Strategy Returns', className="twelve columns indica
    ]
),
html.Div(
    id="market-returns-div",
```

```
                        className="two columns indicator pretty_container",
                        children=[
                            html.P(id="market-returns", className="indicator_value"),
                            html.P('Market Returns', className="twelve columns indicato
                        ]
                    ),
                    html.Div(
                        id="strat-vs-market-div",
                        className="two columns indicator pretty_container",
                        children=[
                            html.P(id="strat-vs-market", className="indicator_value"),
                            html.P('Strategy vs. Market Returns', className="twelve col
                        ]
                    ),
                ]
        )
]),
html.Div(
    className="twelve columns card",
    children=[
        dcc.Graph(
            id="monthly-chart",
            figure={
                'data': []
            }
        )
    ]
),
html.Div(
        className="padding row",
        children=[
            html.Div(
                className="six columns card",
                children=[
                    dash_table.DataTable(
                        id='table',
                        columns=[
                            {'name': 'Number', 'id': 'Number'},
                            {'name': 'Trade type', 'id': 'Trade type'},
                            {'name': 'Exposure', 'id': 'Exposure'},
                            {'name': 'Entry balance', 'id': 'Entry balance'},
                            {'name': 'Exit balance', 'id': 'Exit balance'},
                            {'name': 'Pnl (incl fees)', 'id': 'Pnl (incl fees)'},
                        ],
                        style_cell={'width': '50px'},
                        style_table={
                            'maxHeight': '450px',
                            'overflowY': 'scroll'
```

```
                            },
                        )
                    ]
                ),
                dcc.Graph(
                    id="pnl-types",
                    className="six columns card",
                    figure={}
                )
            ]
        ),
    ])
])
```

1.10.1 Exercise 6:

Add a callback function to load the data we are looking to visualize into a bar chart. The x-axis should be the entry time for that trade, and the y-axis should be the profit or loss for that trade (incl fees) in percentage terms. Please type code in the cell below in addition to having it in your `app.py` file.

Answer. Our recommended solution is shown below:

[ ]:
```
# NOTE: Do NOT run this cell!!! It is for instructional purposes only - it will␣
↪NOT work!
@app.callback(
    dash.dependencies.Output('pnl-types', 'figure'),
    (
        dash.dependencies.Input('exchange-select', 'value'),
        dash.dependencies.Input('leverage-select', 'value'),
        dash.dependencies.Input('date-range', 'start_date'),
        dash.dependencies.Input('date-range', 'end_date'),

    )
)
def update_pnl_types(exchange, leverage, start_date, end_date):
    dff = filter_df(df, exchange, leverage, start_date, end_date)
    dff_long = dff[dff['Trade type']=='Long']
    dff_short = dff[dff['Trade type']=='Short']
    return {
        'data': [
            go.Bar(
                x=dff_long['Entry time'],
                y=dff_long['Pnl (incl fees)'],
                name='long',
                marker_color='lightsalmon'
            ),
```

```
            go.Bar(
                x=dff_short['Entry time'],
                y=dff_short['Pnl (incl fees)'],
                marker_color='black',
                name='short'
            )
        ],
        'layout': {
            'title': 'PnL vs Trade type'
        }
    }
```

1.11 Topping it off

We will now add our two last charts to the layout. One will track the overall value of the portfolio over time, and the other will plot the value of BTCUSD over that same period. This will help us visualize trends in BTC markets as well as the overall value of our balance. Let's add two charts to the layout at the bottom:

```
app.layout = html.Div(children=[
    html.Div(
            children=[
                html.H2(children="Bitcoin Leveraged Trading Backtest Analysis", className='h2-t
            ],
            className='study-browser-banner row'
    ),
    html.Div(
        className="row app-body",
        children=[
            html.Div(
                className="twelve columns card",
                children=[
                    html.Div(
                        className="padding row",
                        children=[
                            html.Div(
                                className="two columns card",
                                children=[
                                    html.H6("Select Exchange",),
                                    dcc.RadioItems(
                                        id="exchange-select",
                                        options=[
                                            {'label': label, 'value': label} for label in df['E
                                        ],
                                        value='Bitmex',
                                        labelStyle={'display': 'inline-block'}
                                    )
```

```
    ]
),
# Leverage Selector
html.Div(
    className="two columns card",
    children=[
        html.H6("Select Leverage"),
        dcc.RadioItems(
            id="leverage-select",
            options=[
                {'label': str(label), 'value': str(label)} for labe
            ],
            value='1',
            labelStyle={'display': 'inline-block'}
        ),
    ]
),
html.Div(
    className="three columns card",
    children=[
        html.H6("Select a Date Range"),
        dcc.DatePickerRange(
            id="date-range",
            display_format="MMM YY",
            start_date=df['Entry time'].min(),
            end_date=df['Entry time'].max()
        ),
    ]
),
html.Div(
    id="strat-returns-div",
    className="two columns indicator pretty_container",
    children=[
        html.P(id="strat-returns", className="indicator_value"),
        html.P('Strategy Returns', className="twelve columns indica
    ]
),
html.Div(
    id="market-returns-div",
    className="two columns indicator pretty_container",
    children=[
        html.P(id="market-returns", className="indicator_value"),
        html.P('Market Returns', className="twelve columns indicato
    ]
),
html.Div(
    id="strat-vs-market-div",
    className="two columns indicator pretty_container",
```

```
                        children=[
                            html.P(id="strat-vs-market", className="indicator_value"),
                            html.P('Strategy vs. Market Returns', className="twelve col
                        ]
                    ),
                ]
        )
]),
html.Div(
    className="twelve columns card",
    children=[
        dcc.Graph(
            id="monthly-chart",
            figure={
                'data': []
            }
        )
    ]
),
html.Div(
        className="padding row",
        children=[
            html.Div(
                className="six columns card",
                children=[
                    dash_table.DataTable(
                        id='table',
                        columns=[
                            {'name': 'Number', 'id': 'Number'},
                            {'name': 'Trade type', 'id': 'Trade type'},
                            {'name': 'Exposure', 'id': 'Exposure'},
                            {'name': 'Entry balance', 'id': 'Entry balance'},
                            {'name': 'Exit balance', 'id': 'Exit balance'},
                            {'name': 'Pnl (incl fees)', 'id': 'Pnl (incl fees)'},
                        ],
                        style_cell={'width': '50px'},
                        style_table={
                            'maxHeight': '450px',
                            'overflowY': 'scroll'
                        },
                    )
                ]
            ),
            dcc.Graph(
                id="pnl-types",
                className="six columns card",
                figure={}
            )
```

```
                ]
            ),
            html.Div(
                className="padding row",
                children=[
                    dcc.Graph(
                        id="daily-btc",
                        className="six columns card",
                        figure={}
                    ),
                    dcc.Graph(
                        id="balance",
                        className="six columns card",
                        figure={}
                    )
                ]
            )
        ]
    )
])
```

1.11.1 Exercise 7:

Write two callback functions, one that returns a line chart of the price of BTC over the selected period, and one that returns a line chart of the portfolio balance over the same period. Please type code in the cell below in addition to having it in your `app.py` file.

Answer. Our recommended solution is shown below:

[ ]:
```
# NOTE: Do NOT run this cell!!! It is for instructional purposes only - it will␣
↪NOT work!
@app.callback(
    dash.dependencies.Output('daily-btc', 'figure'),
    (
        dash.dependencies.Input('exchange-select', 'value'),
        dash.dependencies.Input('leverage-select', 'value'),
        dash.dependencies.Input('date-range', 'start_date'),
        dash.dependencies.Input('date-range', 'end_date'),
    )
)
def update_btc_price(exchange, leverage, start_date, end_date):
    dff = filter_df(df, exchange, leverage, start_date, end_date)
    return {
        'data': [
            go.Scatter(
                x=dff['Entry time'],
                y=dff['BTC Price']
```

```
            )
        ],
        'layout': {
            'title': 'Daily BTC Price'
        }
    }


@app.callback(
    dash.dependencies.Output('balance', 'figure'),
    (
        dash.dependencies.Input('exchange-select', 'value'),
        dash.dependencies.Input('leverage-select', 'value'),
        dash.dependencies.Input('date-range', 'start_date'),
        dash.dependencies.Input('date-range', 'end_date'),
        dash.dependencies.Input('monthly-chart', 'selectedData')
    )
)
def update_balance(exchange, leverage, start_date, end_date, data):
    dff = filter_df(df, exchange, leverage, start_date, end_date)
    return {
        'data': [
            go.Scatter(
                x=dff['Entry time'],
                y=dff['Exit balance']
            )
        ],
        'layout': {
            'title': 'Balance overtime'
        }
    }
```

1.12 Deploying our application

Now, let's deploy our application on the Web. We will be using an AWS RDS PostgreSQL instances as our data source. We are going to run this app on an EC2 instance which is going to be available and reachable over the Internet.

1.12.1 Setting up the RDS PostgreSQL instance

All of the code in our application is going to remain the same, except that we are going to switch the data source. In the previous case, we worked with a local SQL data. The same approach can be used to connect to an RDS instance by using a different connection string. We need to make this RDS instance publicly accessible, so that we can connect to it from our local machines.

NOTE: This approach is highly insecure and only for the purpose of this case. A

database instance should only ever be allowed access from the IP of the source application: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.Postg

After the instance has been created, load the data from the `aggr.csv` file into a database called `strategy`, which contains the table `trades`. The names of the columns should exactly match the names of the columns in the CSV file.

1.12.2 Replacing the data source

After the database is set up and loaded with the requisite data, we can modify our code to read data from the RDS instance instead. The connection string passed to `create_engine()` is the same for all PostgreSQL instances.

The format is:

postgresql://username:password@host/db_name

Replace these values (username, password, host, db_name) with the configuration of your own RDS instance. Running the `app.py` file should then work exactly the same way. If there are any issues, check that your IP is whitelisted to access the RDS instance.

For example:

```
from sqlalchemy import create_engine

engine = create_engine('postgresql://test:test@rds-url/trades')
df = pd.read_sql("SELECT * from trades", engine.connect(), parse_dates=('OCCURRED_ON_DATE',))
```

1.12.3 Deploying the app to an EC2 instance

We are now going to deploy the code to an EC2 instance. Go ahead and use whatever flavor of deployment you are most comfortable in.

After you've setup a new instance, make sure it's in the same VPC as the RDS database so that they can talk to each other.

1.12.4 Getting our app files ready

We need to get the `app.py` file on our server and run it with `python3 app.py`. This runs a debug server, which for the purpose of this case we will be exposing to the Internet. In enterprise-level production systems, deploying an app in production requires a few more steps to ensure security and scalability: https://dash.plot.ly/deployment.

We are going to create a git repository locally and on Github where the code will be hosted. We also need to get the `requirements.txt` file ready to install all the required libraries on the server. Go to the terminal and run these commands:

```
pip freeze > requirements.txt
git init
git add -a
```

```
git commit -m "Added app.py file"
git remote add origin githuburl
git push origin master --set-upstream
```

This is going to create a `requirements.txt` file based on the libraries being used, initialize a `git` repository, and commit our files to it. We then push this to our remote github repository.

Please note that the step above can be done without the use of git

1.12.5 SSH-ing into our EC2 instance

Let's now SSH into the EC2 instance and install `git` and other libraries that we are going to use. Remember to use `python3` in the shell because that's how we built our apps:

```
sudo apt install git python3
sudo apt-get install postgresql libpq-dev postgresql-client postgresql-client-common
git clone github-url
cd name_of_directory
pip3 install -r requirements.txt
python3 app.py
```

If all the commands completed successfully, you should see a server running prompt much like the one on your local machine. The one step left is to allow the EC2 security group to be world accessible. In your AWS sonsole, make port 8050 accessible from 0.0.0.0, which will let anyone connect to the server: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/authorizing-access-to-an-instance.html

Finally, open up a browser and launch the URL http://instance-ip:8050, and your app should be accessible from over the Internet.