



---

## Task B3: Data Processing 2

---

Conducted by:

### Group 1

*Team Members*

*Email*

|                     |                               |
|---------------------|-------------------------------|
| Thanh Tam Vo        | 103487596@student.swin.edu.au |
| Phuc Khai Hoan Cao  | 103804739@student.swin.edu.au |
| Tai Minh Huy Nguyen | 104220352@student.swin.edu.au |

March 17, 2024

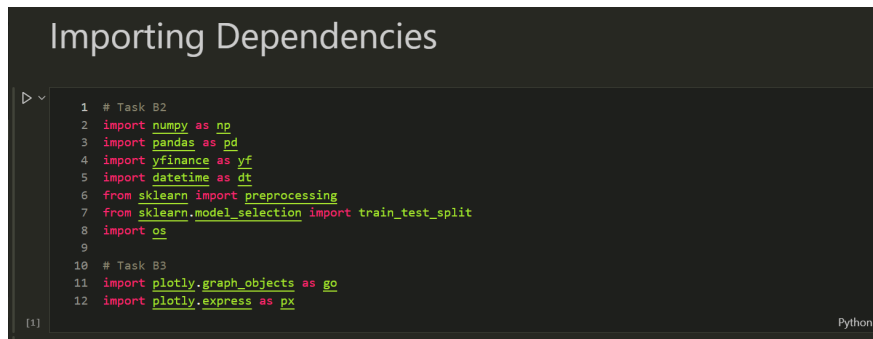
# Table of Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introduction</b>                                    | <b>2</b> |
| <b>2</b> | <b>Importing Dependencies</b>                          | <b>2</b> |
| <b>3</b> | <b>Hyperparameters</b>                                 | <b>2</b> |
| <b>4</b> | <b>Inverse the Dataframe</b>                           | <b>3</b> |
| <b>5</b> | <b>Candlestick chart</b>                               | <b>4</b> |
| 5.1      | Preprocessing Data to fit the trading period . . . . . | 5        |
| 5.2      | Plot the Result . . . . .                              | 6        |
| <b>6</b> | <b>Boxplot Chart</b>                                   | <b>7</b> |

# 1 Introduction

In the Task B3: Data Processing 2, the main goal is writing two function to visualize the financial market with two different techniques: Boxplot chart and Candle Stick chart.

## 2 Importing Dependencies



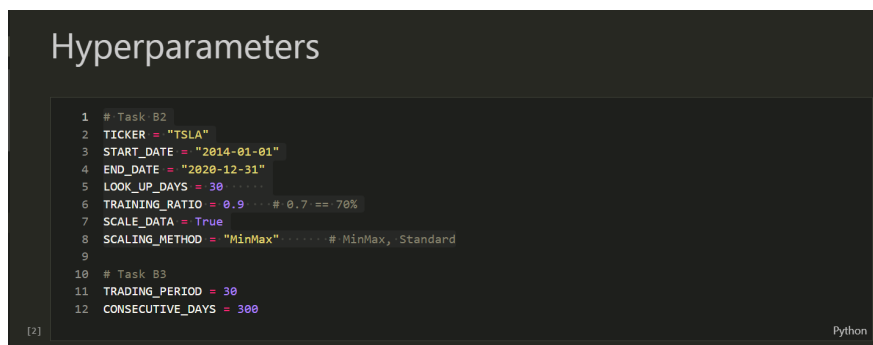
```
1 # Task B2
2 import numpy as np
3 import pandas as pd
4 import yfinance as yf
5 import datetime as dt
6 from sklearn import preprocessing
7 from sklearn.model_selection import train_test_split
8 import os
9
10 # Task B3
11 import plotly.graph_objects as go
12 import plotly.express as px
```

Figure 1: Importing new packages (Line 11 and 12)

There are some new several packages that are need to be imported (Figure 1):

- **plotly.graph\_objects** an open source module to visualize stock data from **plotly**
- **plotly.express** an open source module to visualize statistical data from **plotly**

## 3 Hyperparameters



```
1 # Task B2
2 TICKER = "TSLA"
3 START_DATE = "2014-01-01"
4 END_DATE = "2020-12-31"
5 LOOK_UP_DAYS = 30
6 TRAINING_RATIO = 0.9 # 0.7 == 70%
7 SCALE_DATA = True
8 SCALING_METHOD = "MinMax" # MinMax, Standard
9
10 # Task B3
11 TRADING_PERIOD = 30
12 CONSECUTIVE_DAYS = 300
```

Figure 2: New constants indicating Hyperparameters (Line 11 and 12)

As you can see from Figure 2, there are two new created constants:

- **TRADING\_PERIOD** indicating the trading days to plot the Candle Stick Chart
- **CONSECUTIVE\_DAYS** indicating the consecutive trading days to plot the Boxplot Chart

## 4 Inverse the Dataframe

Firstly, we would like to introduce the **datasetInverser()** function (in Figure 3) to reverse the scaling process applied to the stock market data. This is a necessary step if we want to visualize the original unscaled data.

```
def datasetInverser(scaledStockData, datasetScaler):  
  
    # Getting Column name  
    col_names = scaledStockData.columns  
    # Inversing the dataframe  
    re_scaled_features = datasetScaler.inverse_transform(scaledStockData)  
    re_scaled_stock_data = pd.DataFrame(re_scaled_features, columns = col_names)  
    # Assigning index to the rescaled_data  
    re_scaled_stock_data.index = scaledStockData.index  
  
    ## A Pandas Dataframe  
    return re_scaled_stock_data
```

Figure 3: **datasetInverser()** function

The function above takes two parameters:

- **scaledStockData**: The scaled dataset (Pandas.DataFrame) obtained after preprocessing from B2.
- **datasetScaler**: The scaler object (**MinMaxScaler()** or **StandardScaler()**) used to scale the dataset (also already well-defined and tested in B2)

This function retrieves the column names from the scaled dataset. It then applies the inverse transformation using the **inverse\_transform()** method of the datasetScaler object to get the unscaled features. Then, the unscaled features are then converted back into a DataFrame with the original column names. Finally, the index of the unscaled DataFrame is assigned to match the index of the scaled dataset.

The function returns an unscaled Pandas.DataFrame representing the original stock market data.

## 5 Candlestick chart

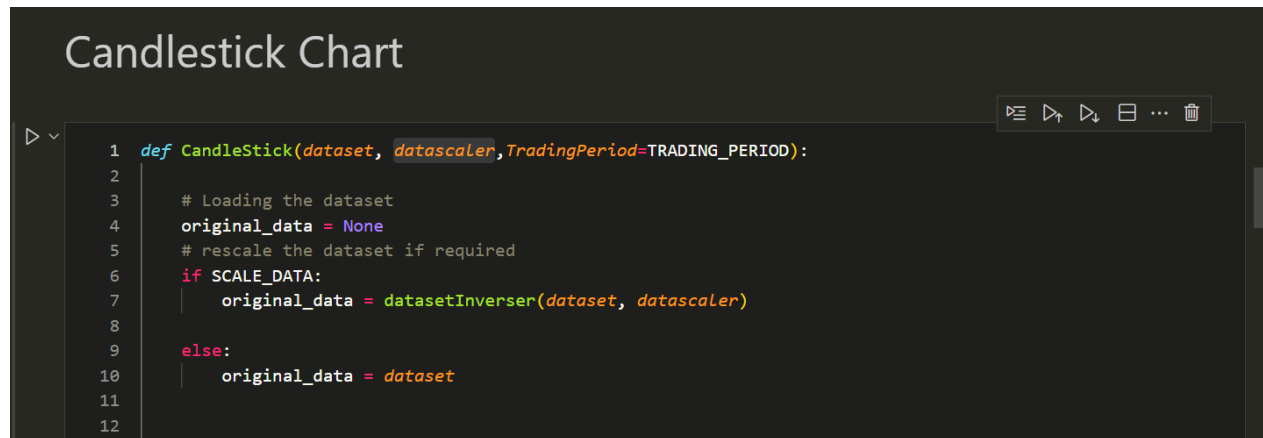


Figure 4: The very first line of the **CandleStick()** function

The **CandleStick()** function generates an interactive candlestick chart to visualize stock price movements over a specified trading period.

The function above takes three parameters:

- **dataset**: The dataset to visualize, which can be either scaled or unscaled based on the value of the **SCALE\_DATA** parameter.
- **datascaler**: The scaler object used for inverse transformation if applicable.
- **TradingPeriod**: The length of each trading period for aggregating data.

From Line 3 to Line 10 (Figure 4), if the data needs to be reversed (**SCALE\_DATA == True**), the function calls the **datasetInverser()** function to obtain the original unscaled data.

## 5.1 Preprocessing Data to fit the trading period

```

12     # Processed Data to fit the Trading Period
13     total_records = len(original_data)
14     Price_Data = { 'Date': [],
15                   'Open': [],
16                   'High': [],
17                   'Low': [],
18                   'Close': [],
19                   'Volume': []
20               }
21
22
23     ## Loop through the DataFrame in batches of TradingPeriod
24     for i in range(0, total_records, TradingPeriod):
25         batch = original_data.iloc[i:i+TradingPeriod]
26
27         Price_Data['Date'].append(batch.index[0])
28         Price_Data['Open'].append(batch['Open'].values[0])
29         Price_Data['High'].append(max(batch['High'].values))
30         Price_Data['Low'].append(min(batch['Low'].values))
31         Price_Data['Close'].append(batch['Close'].values[len(batch) - 1])
32         Price_Data['Volume'].append(sum(batch['Volume'].values))
33
34     # Converting to Pandas Dataframe
35     NewDataFrame = pd.DataFrame(Price_Data)
36     NewDataFrame.set_index('Date', inplace=True)
37

```

Figure 5: Preprocessing Stock Data to fit the trading period

From the line 14 to line 20 of the Figure 5, we initialize an dictionary called PRICE\_DATA before converting it into a Pandas Dataframe. Then, the function will iterate every batch of **TradingPeriod** records and get the appropriate information. Finally, a new Pandas DataFrame is created with the index of Date. This Pandas DataFrame stored the stock data that satisfies the trading period.

In the next subsection, we will show our result after implementing the Candlestick chart.

## 5.2 Plot the Result

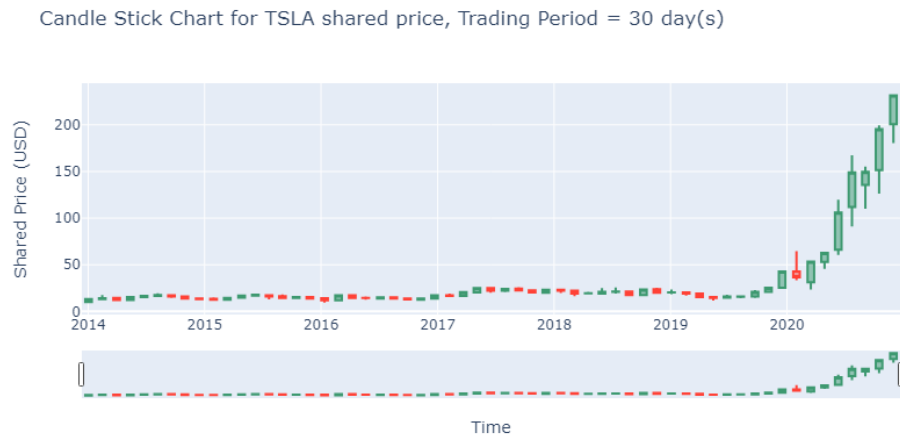


Figure 6: An interactive Candlestick chart by Plotly

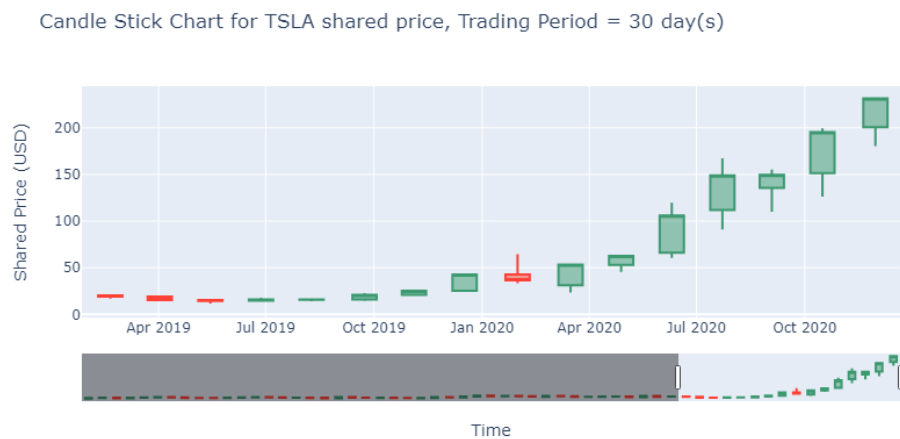


Figure 7: Sliding the bar to see a specific period

## 6 Boxplot Chart

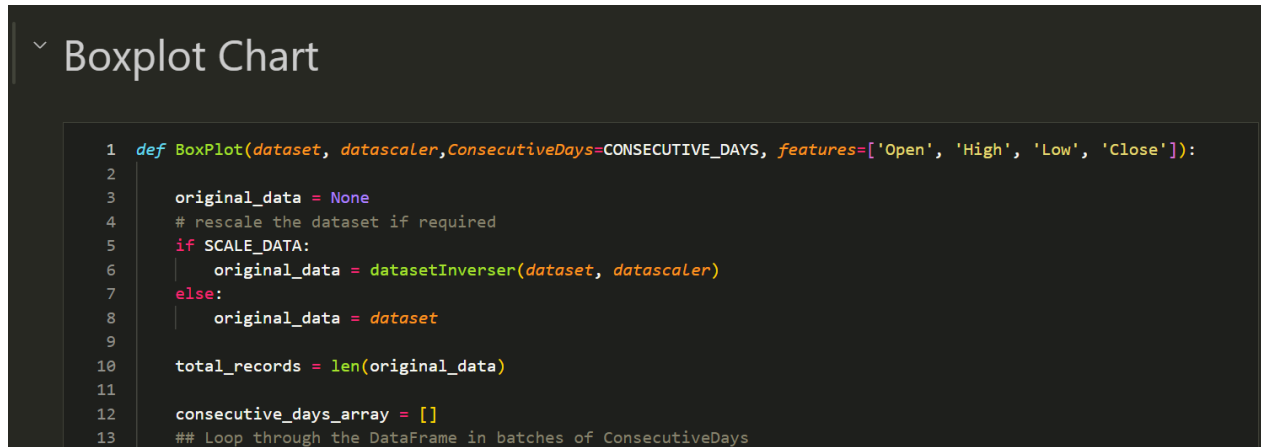


Figure 8: **BoxPlot()** function

The **BoxPlot()** function creates an interactive box plot chart to visualize the distribution of stock prices over consecutive days.

The function above takes four parameters:

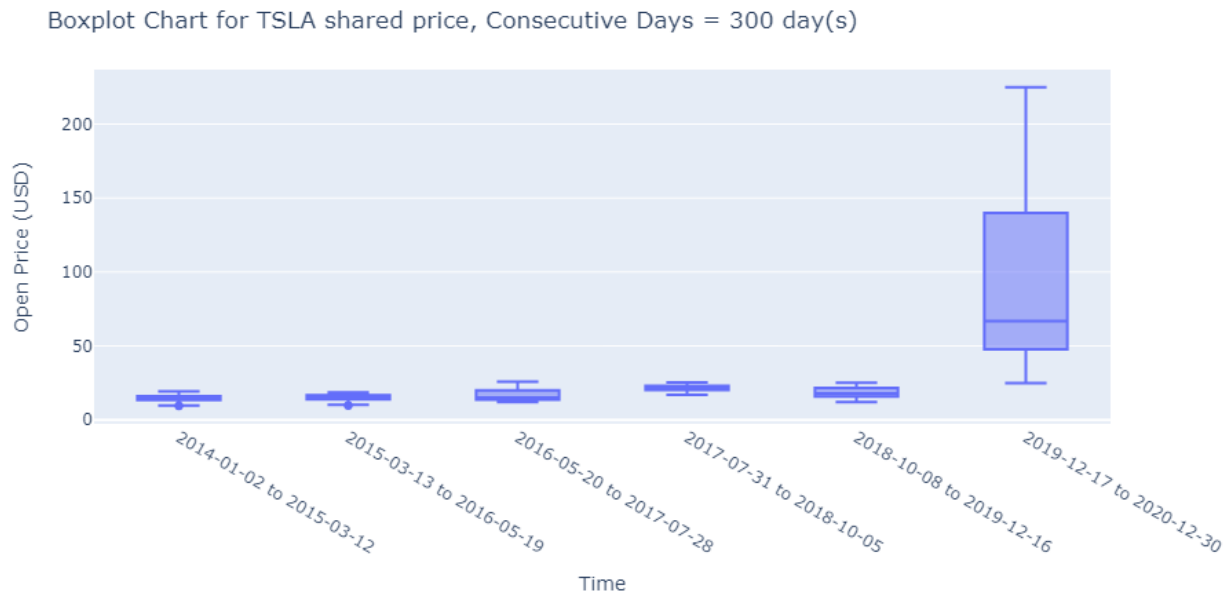
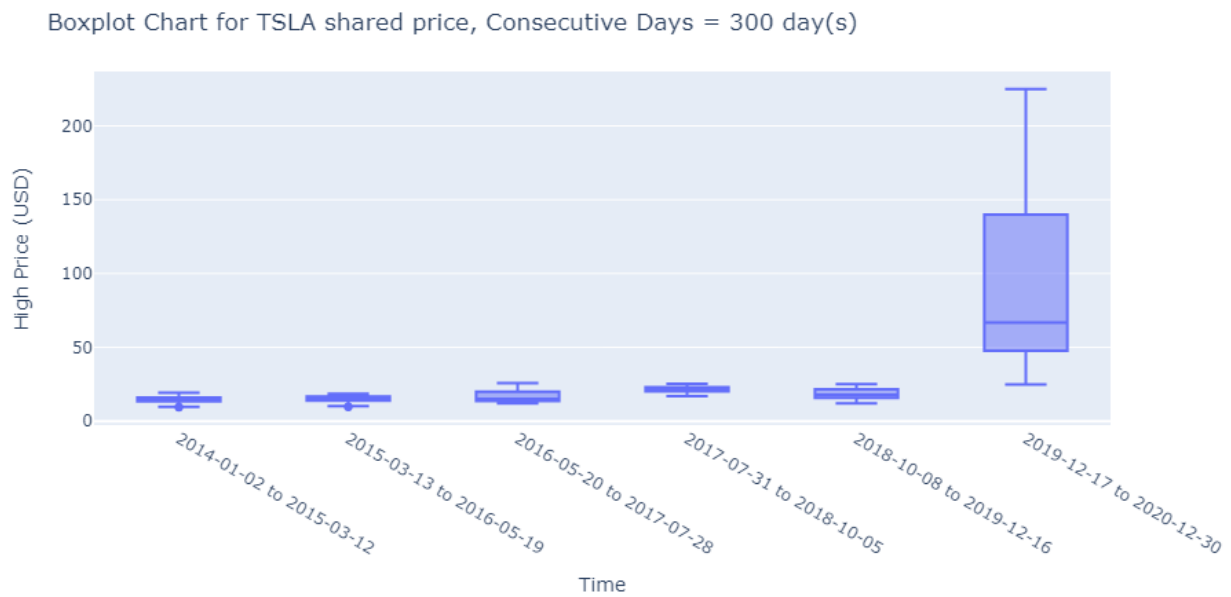
- **dataset**: The dataset to visualize, which can be either scaled or unscaled based on the value of the **SCALE\_DATA** parameter.
- **datasetScaler**: The scaler object (**MinMaxScaler()** or **StandardScaler()**) used to scale the dataset (also already well-defined and tested in B2)
- **ConsecutiveDays**: The number of consecutive days to consider for each box plot.
- **features**: The features (e.g., Open, High, Low, Close) to plot on the box plot chart.

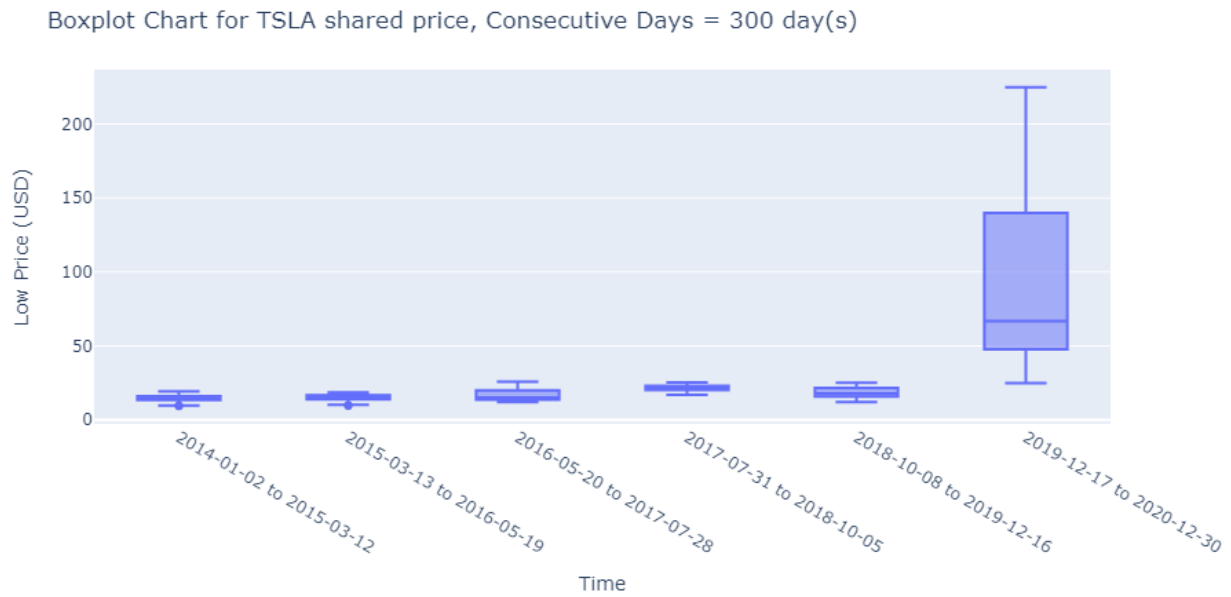
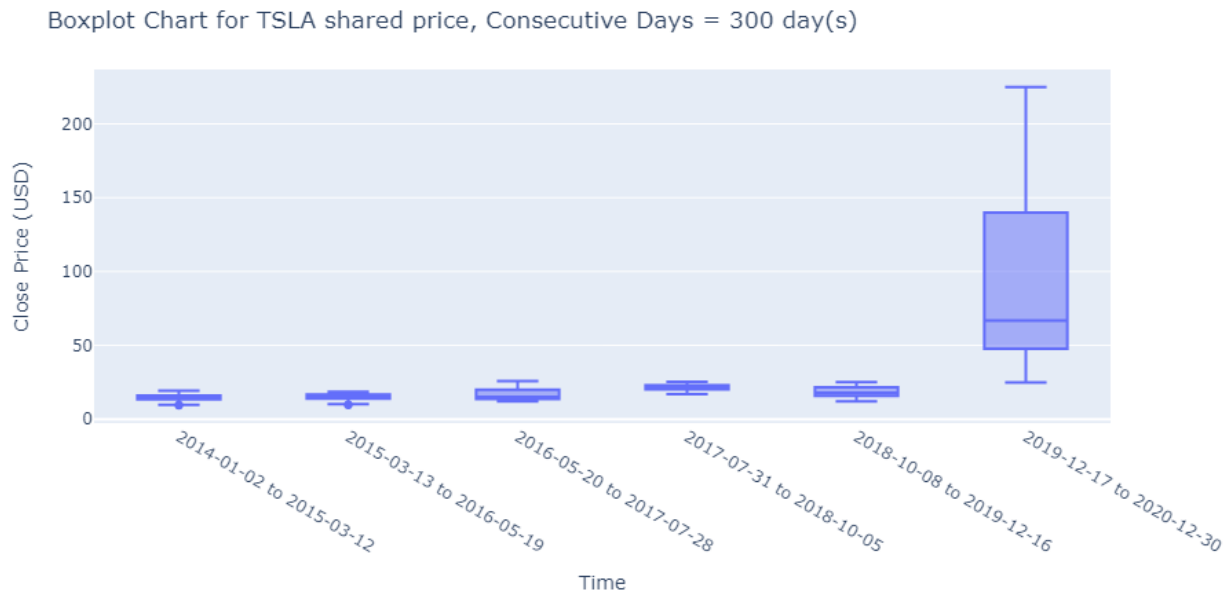
From Line 3 to Line 8 (Figure 8), if the data needs to be reversed (**SCALE\_DATA == True**), the function calls the **datasetInverser()** function to obtain the original unscaled data.

First, the function will loop through the DataFrame in batches of **ConsecutiveDays**. Next, it collects the first date and the last date in the batch and creates a new column called **Consecutive**. Every record in the batch has the same value at **Consecutive** and based on this new column, the function will call Plotly Express to create an interactive box plot chart, visualizing the distribution of each feature over consecutive days.

When we plot the result, there are 4 interactive chart as below:



Figure 9: Boxplot Chart for *Open* FeatureFigure 10: Boxplot Chart for *High* Feature

Figure 11: Boxplot Chart for *Low* FeatureFigure 12: Boxplot Chart for *Close* Feature