SWINBURNE
UNIVERSITY OF
TECHNOLOGY

Intelligent System

Assignment 1 - Option B

Week 3 Report

Xuan Tuan Minh Nguyen - 103819212

Environment Setup

1. Create a new environment using Conda CLI

There are several different procedures to create an environment, but the given procedure below will **encapsulate** all of the **required steps** to create **a safe and clean environment** using Conda.

- Navigate to the Github repository that contains the source code for both P1 and V0.1.
- Once navigated, download the source code by clicking on **Code → Download ZIP** or use the following command in the **CLI (Terminal):**

```
git clone https://github.com/cobeo2004/cos30018.git
```

- Once the source code is successfully cloned (downloaded), navigate to the Week 3/v0.2 folder and execute the file conda-config.sh using the following command:

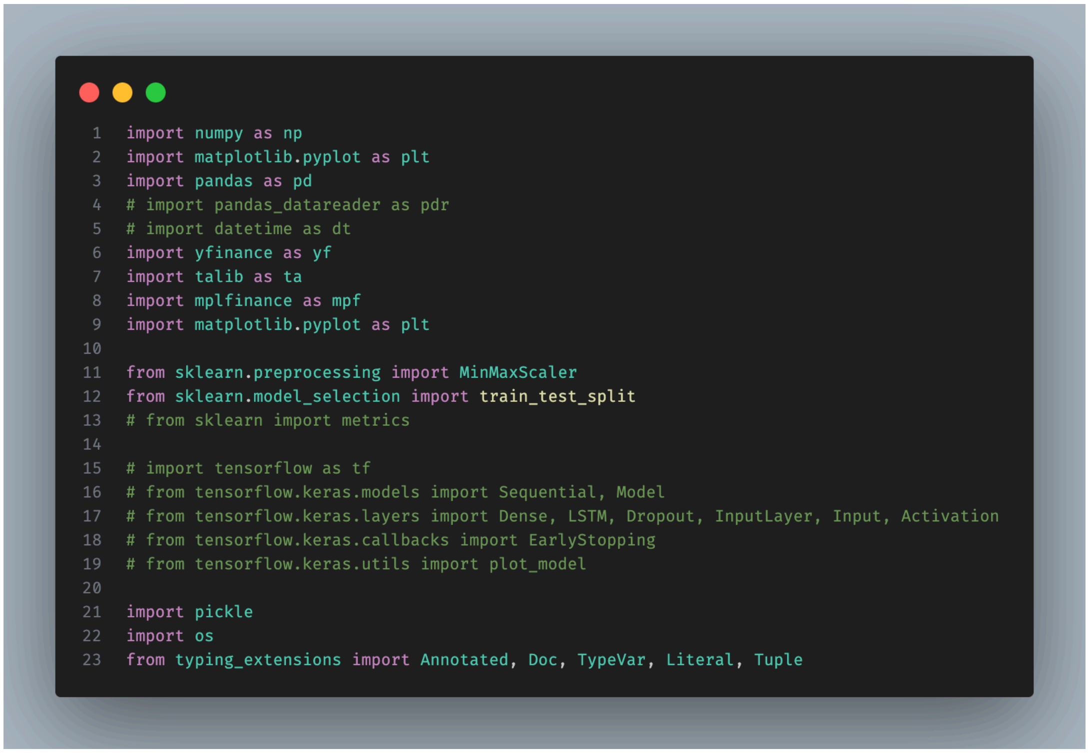
```
bash conda-config.sh
```

- The given file config.sh will execute the following procedure:
  - Generate an environment with a pre-defined name (you can change the name if you want to) in Python 3.10.9 by using the command: **conda create -n cos30018_env_w3_v0.2 python=3.10.9**
  - Activate the created environment using: **conda activate cos30018_env_w3_v0.2.**
  - Check and validate if the conda environment is successfully initialized by running **conda info –envs** for listing conda environments and see which environment that we are in and current Python version using **python –version.**

2. Installing required dependencies

Once the **environment** is **successfully initialized**, we can start **installing** the **dependencies (libraries)** that are **required by the program**. There are multiple pathways to install dependencies in Python, but the **most popular steps** are:

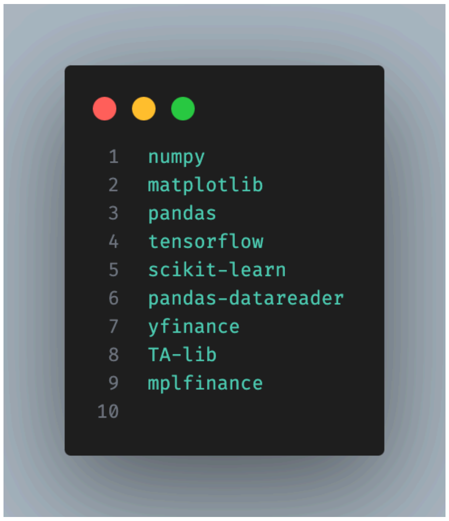
- Scan through the code to find out the required dependencies; for example, consider the file stock_prediction.py. We could see that there are quite a few required dependencies, such as: numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance TA-lib. However, there will be a new library called mplfinance that helps us to efficiently create a beautiful and easy to analyze **candlestick chart** without having to manually set up.



```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# import pandas_datareader as pdr
# import datetime as dt
import yfinance as yf
import talib as ta
import mplfinance as mpf
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# from sklearn import metrics

# import tensorflow as tf
# from tensorflow.keras.models import Sequential, Model
# from tensorflow.keras.layers import Dense, LSTM, Dropout, InputLayer, Input, Activation
# from tensorflow.keras.callbacks import EarlyStopping
# from tensorflow.keras.utils import plot_model

import pickle
import os
from typing_extensions import Annotated, Doc, TypeVar, Literal, Tuple
```

Figure 1: Required Dependencies in stock_prediction.ipynb

- Once dependencies are scanned, use the following command to install the dependencies: pip install numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance TA-lib mplfinance.
- Another step is to list all required libraries into a requirements.txt file, and using the following command to install the required dependencies: pip install -U -r requirements.txt.



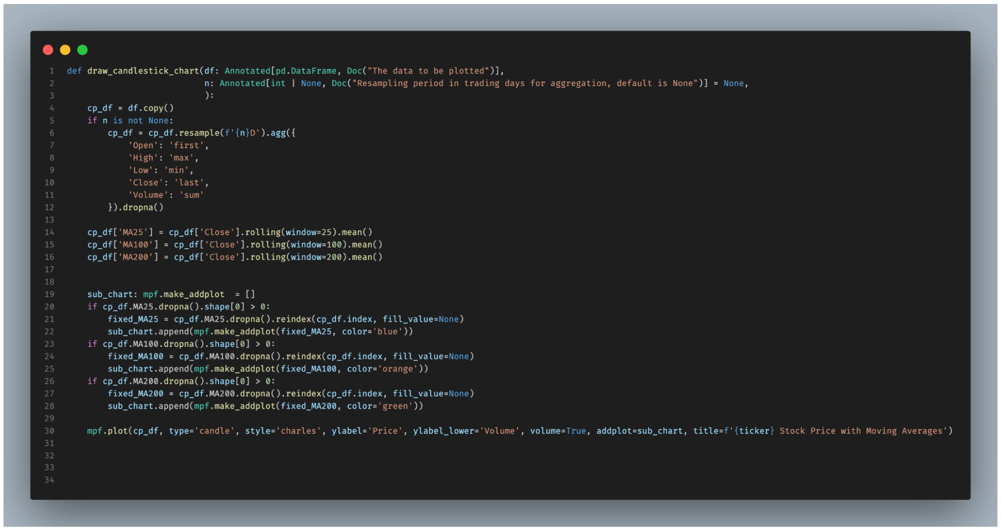
```
numpy
matplotlib
pandas
tensorflow
scikit-learn
pandas-datareader
yfinance
TA-lib
mplfinance

```

Figure 2: Example of requirements.txt

Understanding the Data Processing II

1. Function to generate candlestick chart

Codebase



```python
def draw_candlestick_chart(df: Annotated[pd.DataFrame, Doc("The data to be plotted")],
                           n: Annotated[int | None, Doc("Resampling period in trading days for aggregation, default is None")] = None,
                           ):
    cp_df = df.copy()
    if n is not None:
        cp_df = cp_df.resample(f'{n}D').agg({
            'Open': 'first',
            'High': 'max',
            'Low': 'min',
            'Close': 'last',
            'Volume': 'sum'
        }).dropna()

    cp_df['MA25'] = cp_df['Close'].rolling(window=25).mean()
    cp_df['MA100'] = cp_df['Close'].rolling(window=100).mean()
    cp_df['MA200'] = cp_df['Close'].rolling(window=200).mean()


    sub_chart: mpf.make_addplot  = []
    if cp_df.MA25.dropna().shape[0] > 0:
        fixed_MA25 = cp_df.MA25.dropna().reindex(cp_df.index, fill_value=None)
        sub_chart.append(mpf.make_addplot(fixed_MA25, color='blue'))
    if cp_df.MA100.dropna().shape[0] > 0:
        fixed_MA100 = cp_df.MA100.dropna().reindex(cp_df.index, fill_value=None)
        sub_chart.append(mpf.make_addplot(fixed_MA100, color='orange'))
    if cp_df.MA200.dropna().shape[0] > 0:
        fixed_MA200 = cp_df.MA200.dropna().reindex(cp_df.index, fill_value=None)
        sub_chart.append(mpf.make_addplot(fixed_MA200, color='green'))

    mpf.plot(cp_df, type='candle', style='charles', ylabel='Price', ylabel_lower='Volume', volume=True, addplot=sub_chart, title=f'{ticker} Stock Price with Moving Averages')
```

Figure 3: Codebase for drawing ***candlestick chart***

Parameters

- df: pandas.DataFrame: The dataset in the type of pandas.DataFrame that is used for drawing the candlestick chart.
- n: int | None: The period that is used for resampling trading days, by default it is set to None.

Functionalities

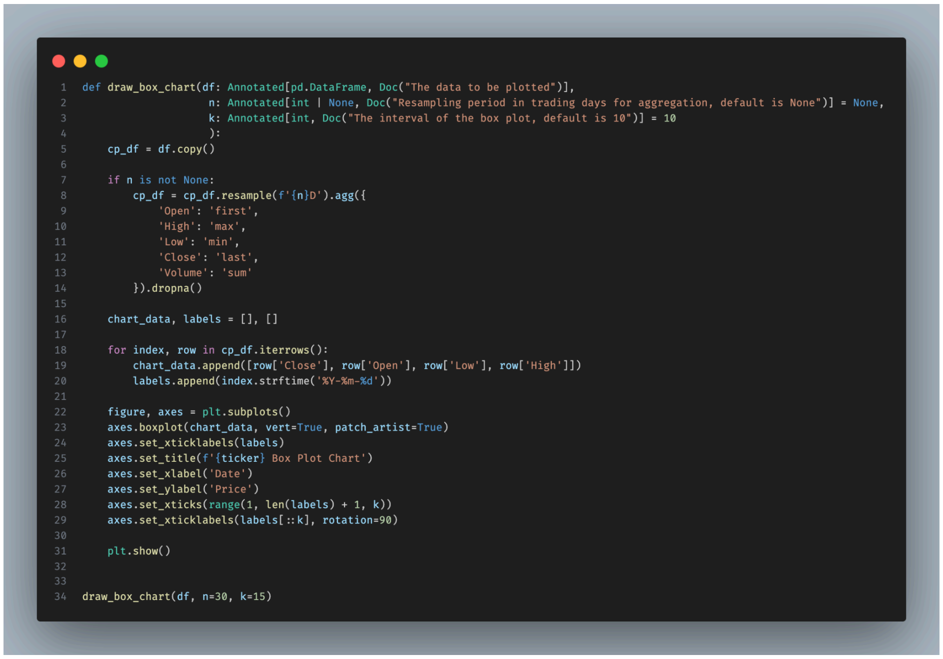
- In short, the draw_candlestick_chart() will take in the **dataframe** and create a copy, once the copy is created then it will resample the data by the n trading days if n is not None. It then adds in essential **technical analysis strategies** as **additional values** to display on the chart and draw the **candlestick chart** using mplfinance. The following steps display the **procedure** of the draw_candlestick_chart():
    - Dataframe copy: When the **dataframe** is passed in as an argument, this function will start **copying** the dataframe using the copy() function provided by pandas. This step is critically essential, as it will prevent any **unwanted changes** that could be executed to the **real dataframe**.
    - Data resample: The draw_candlestick_chart() will then check if the n parameter is **not None**, if the n parameter is **not None** then the function will start aggregating the data to find the **maximum value of high price**, **minimum value of low price**, **first open price value**,

last close price value and **the total of volumes** within each resampling n period.

- Technical Analysis Strategies: Once the resampling procedure is finished, it will add three moving averages (MA) for the 'Close' price to the copied dataframe, which are 25 days, 100 days and 200 days moving average.
- Create Sub-Plot: The function also creates an array of make_addplot()instances, which will be used to display the moving averages (25 days, 100 days and 200 days) alongside the candlestick chart. However, before the chart is being appended to the array, it must be pre-processed by drop NaN values and reindexed for assuring safe data.
- Display the chart: Once the moving averages sub-plot procedure is finished, the function will utilize plot() function from mplfinance to display the candlestick chart. As shown in the codebase, this function will take in the dataframe that we have processed before, with some required parameters, which could be list as the type of the chart, the title of the chart, the style of the chart, the label name, the total volume of the data and the sub-plot that we have created before.

2. Function to generate box chart

Codebase



```python
def draw_box_chart(df: Annotated[pd.DataFrame, Doc("The data to be plotted")],
                   n: Annotated[int | None, Doc("Resampling period in trading days for aggregation, default is None")] = None,
                   k: Annotated[int, Doc("The interval of the box plot, default is 10")] = 10
                   ):
    cp_df = df.copy()

    if n is not None:
        cp_df = cp_df.resample(f'{n}D').agg({
            'Open': 'first',
            'High': 'max',
            'Low': 'min',
            'Close': 'last',
            'Volume': 'sum'
        }).dropna()

    chart_data, labels = [], []

    for index, row in cp_df.iterrows():
        chart_data.append([row['Close'], row['Open'], row['Low'], row['High']])
        labels.append(index.strftime('%Y-%m-%d'))

    figure, axes = plt.subplots()
    axes.boxplot(chart_data, vert=True, patch_artist=True)
    axes.set_xticklabels(labels)
    axes.set_title(f'{ticker} Box Plot Chart')
    axes.set_xlabel('Date')
    axes.set_ylabel('Price')
    axes.set_xticks(range(1, len(labels) + 1, k))
    axes.set_xticklabels(labels[::k], rotation=90)

    plt.show()


draw_box_chart(df, n=30, k=15)
```

*Figure 4: Codebase for drawing **box chart***

Parameters

- df: pandas.DataFrame: The dataset in the type of pandas.DataFrame that is used for drawing the box chart.
- n: int | None: The period that is used for resampling trading days, by default it is set to None.
- k: int: The interval between each day that will be used to display in the box plot.

Functionalities

- The process of resampling data and the process of prepare data for the box plot is partially the same as the draw_candlestick_chart(), however, the key difference that separates the draw_box_chart() function is the way that it creates the sub-chart and the libraries that it uses for creating the chart. The below steps show the overall procedure of how the draw_box_chart()function work:
    - **Dataframe copy**: When the **dataframe** is passed in as an argument, this function will start **copying** the dataframe using the copy() function provided by pandas. This step is critically essential, as it will prevent any **unwanted changes** that could be executed to the **real dataframe**.
    - **Data resample**: The draw_candlestick_chart() will then check if the n parameter is **not None**, if the n parameter is **not None** then the function will start aggregating the data to find the **maximum value of high price**, **minimum value of low price**, **first open price value**, **last close price value** and **the total of volumes** within each resampling n period.
    - **Data preparation**: After the **data resample** is finished, the function will start preparing data for displaying to the plot. First, the two arrays of chart_data and labels are created. The chart_data contains the c**lose, open, low and high price** for each data point and the labels will contain the corresponding date for each data point.
    - **Chart display**: After preparing the data, the function will start **creating and configuring** the chart. First, the function will use the subplots() function provided by matplotlib to **create a figure and axes**. It will then **initialize and create** the box chart using the boxplot() function (which is also provided by matplotlib). The prepared chart_data array will be used as the **main data** to **draw the box plot** with different parameters are set to customize the appearance of the chart, which could be listed as: vertical alignment, patch artist, label , title, label for x and y axis, x-tick interval, the label of the x-tick and the rotation degree of the chart. The k parameter that is defined in the function will be used for determining

the interval for displaying the labels at which days. Finally, the show() function of matplotlib is called to display the charts.

Deploying and Testing the Codebase

1. Result

- All of the charts are tested on the raw datasets that is directly downloaded from yfinance (as defined as data variables in the codebase).

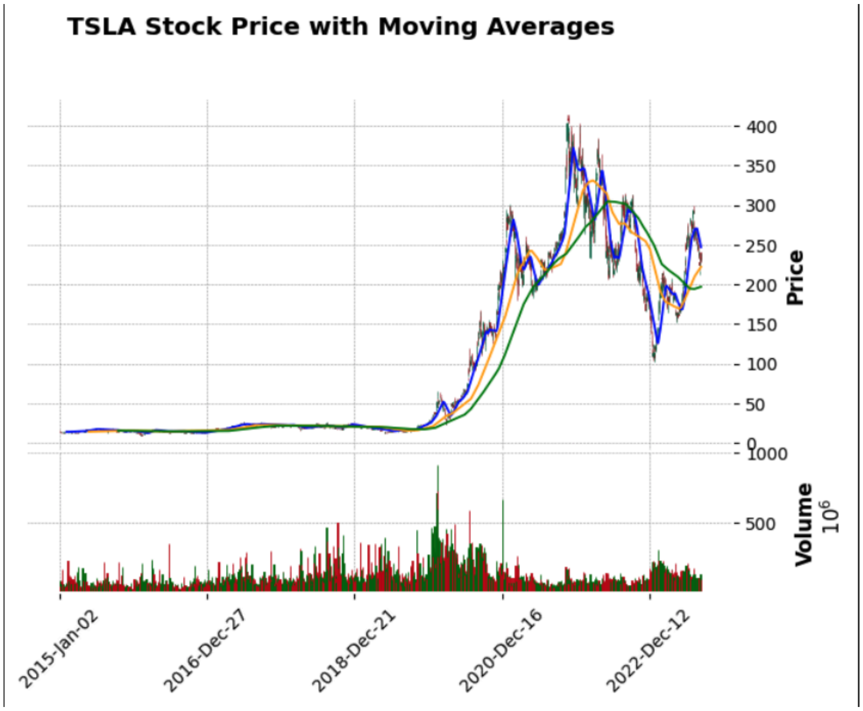


Figure 5: Candlestick chart with n equals None

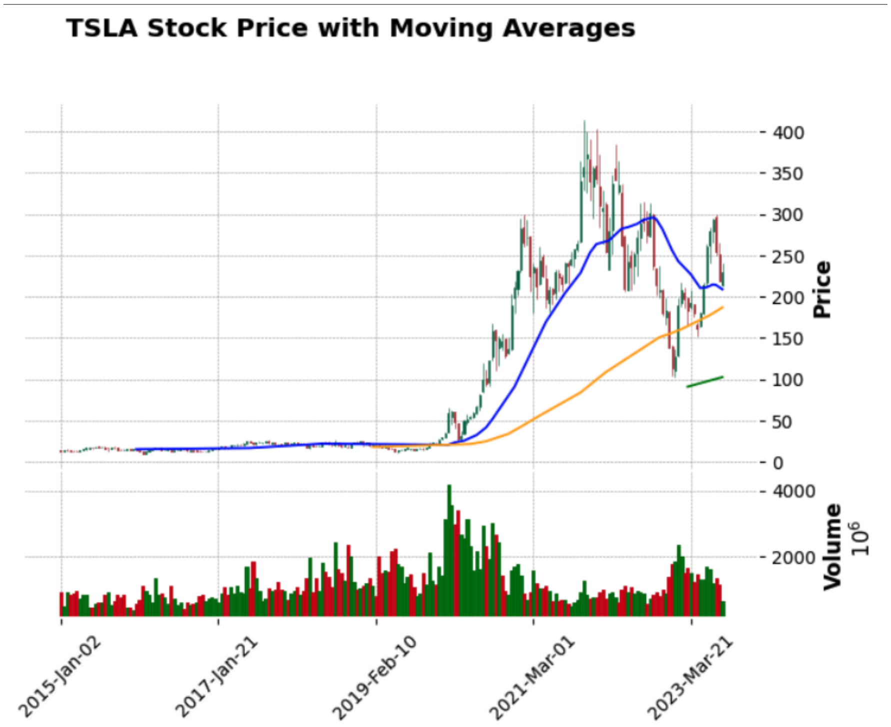


Figure 6: Candlestick chart with n equals 15

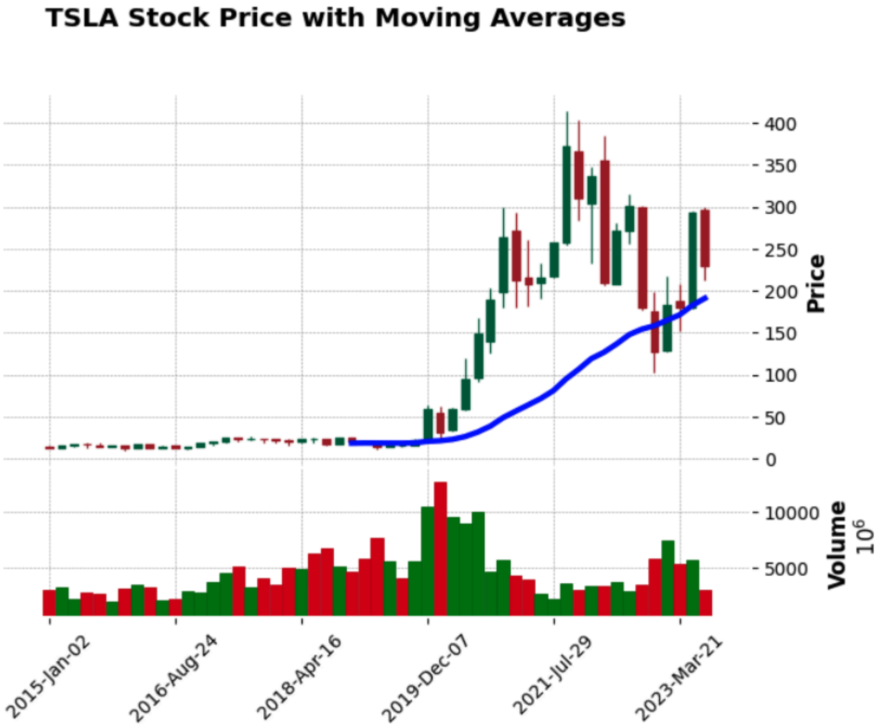


Figure 7: Candlestick chart with n equals 60

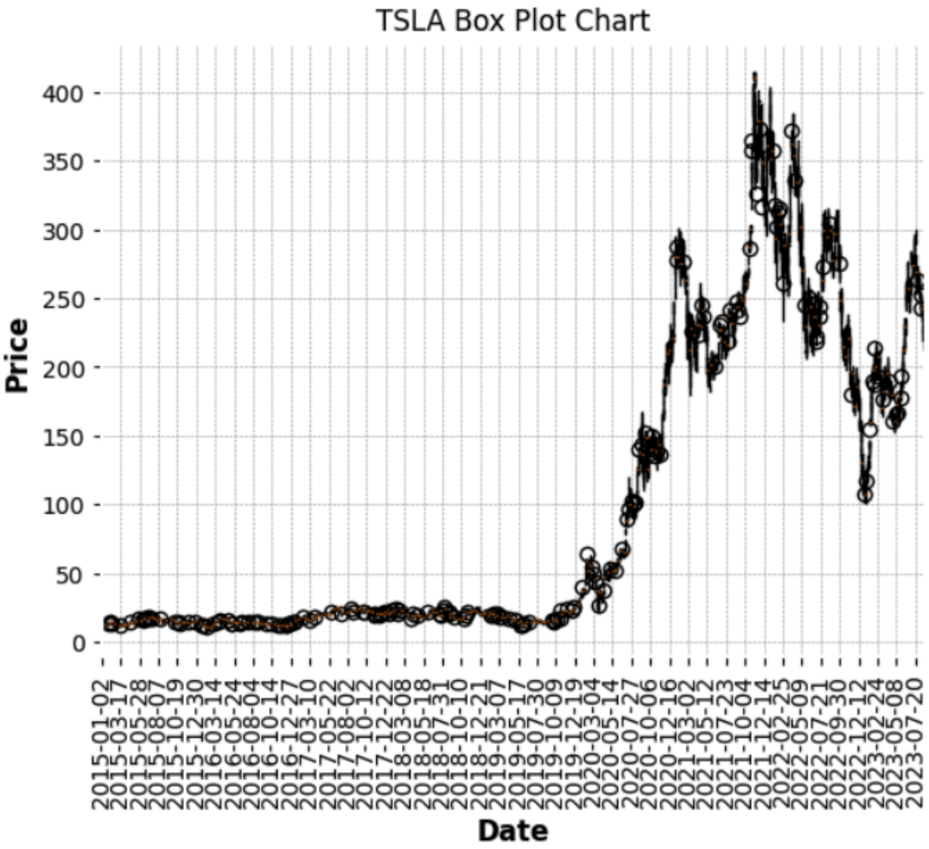


Figure 8: Box chart with n equals None and k equals 50

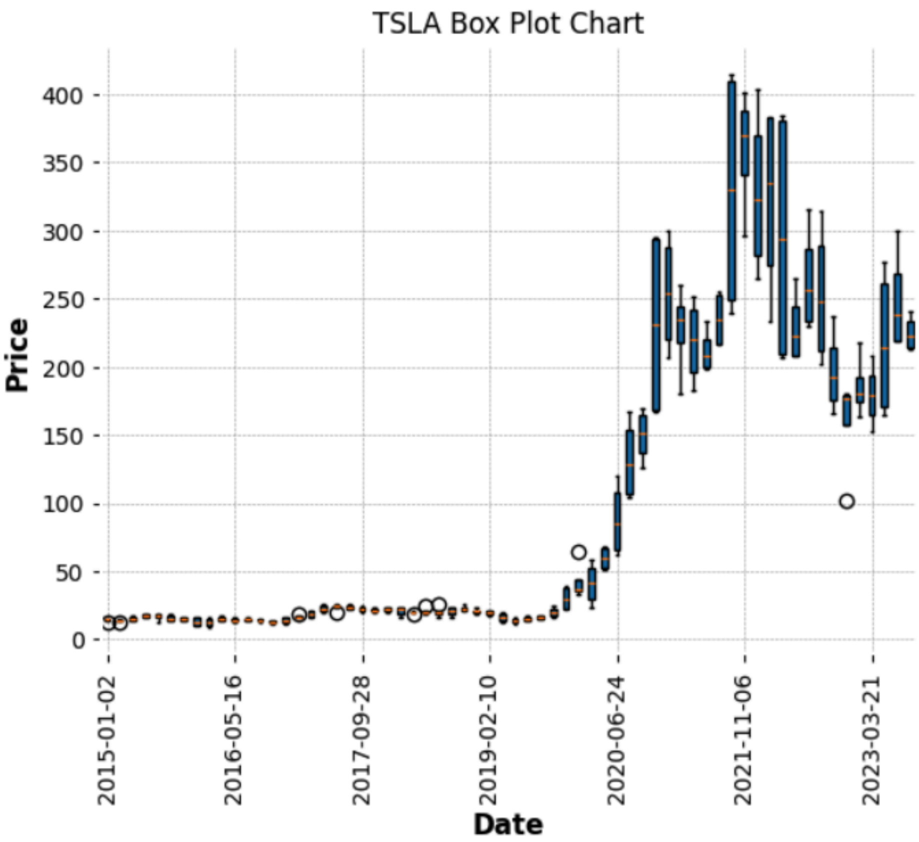


Figure 9: Box chart with n equals 50 and k equals 10

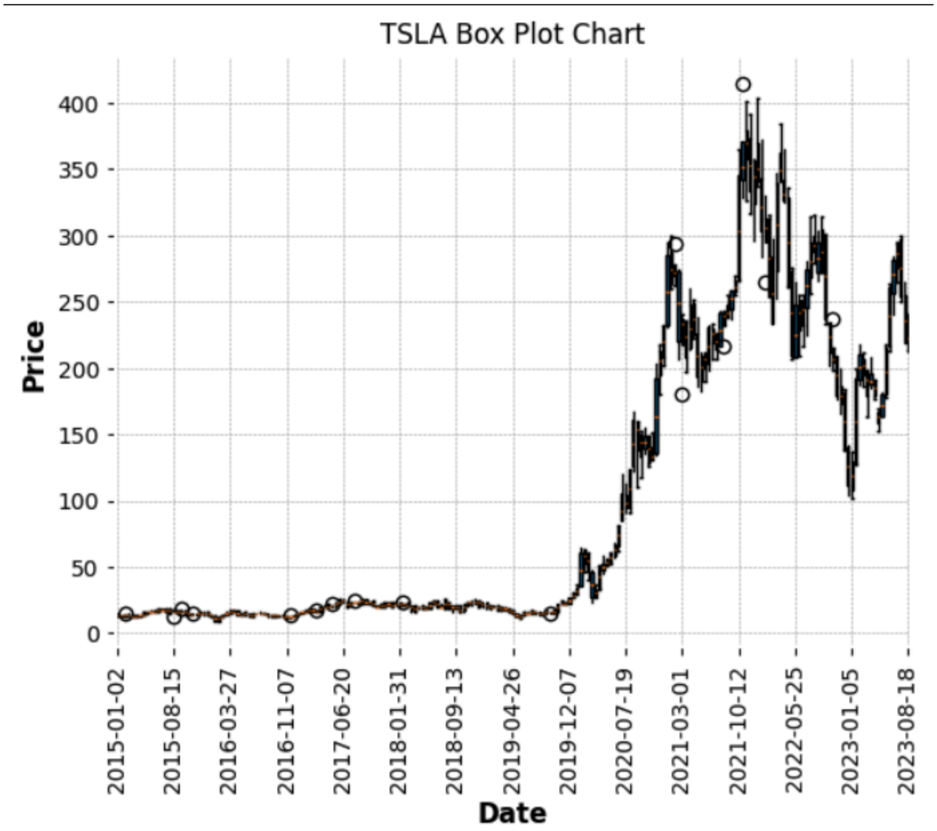


Figure 10: Box chart with n and k equals 15

2. Understanding the relations between n and k

- When n starts to:
  + Decrease: Result in more **data points** (crowded boxes in the box chart). Thus increase the **x-axis** which is not ideal when we resampling the data over a smaller window.
  + Increase: Result in less **data points** (less boxes in the box chart). Thus decreasing the amount of data **x-axis** which is ideal when we resampling the data over a smaller window.
- When k starts to:
  + Decrease: Result in more x-axis labels being displayed. Thus increase the crowd of labels in the x-axis.
  + Increase: Result in less x-axis labels being displayed. Thus decrease the crowd of labels in the x-axis.
- In short, we may want to increase our k to prevent the labels being too crowded in the x-axis if the n parameter is decreased. On the other hand, if the n parameter is increased, then we should decreasing the value of k to show a more detailed labels.