Contents

I. Introduction

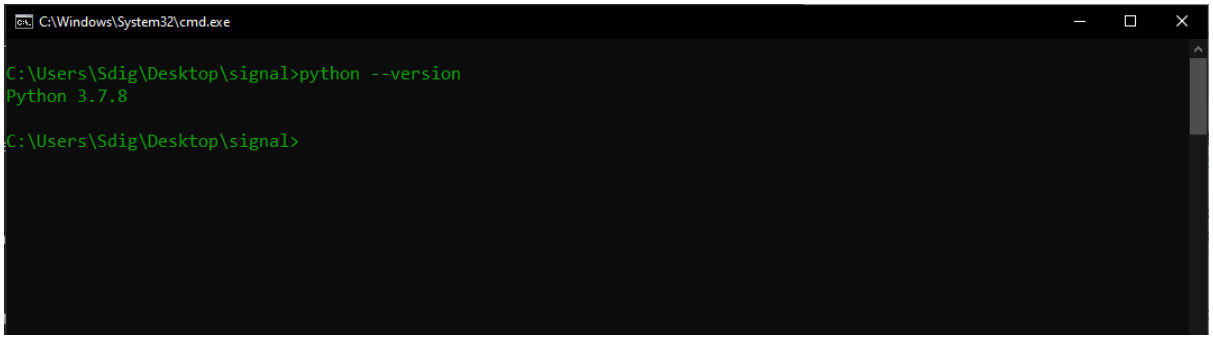
Welcome to the documentation of “Stock Signal” developed by Digitarab Consulting. This product enables user to perform analytics on a given stock API” Alpha Vantage”. This documentation contains information about the project and its development as well as the software’s feature and usage.

II. Requirements

In this section we will talk about the necessary requirements in each operating system.

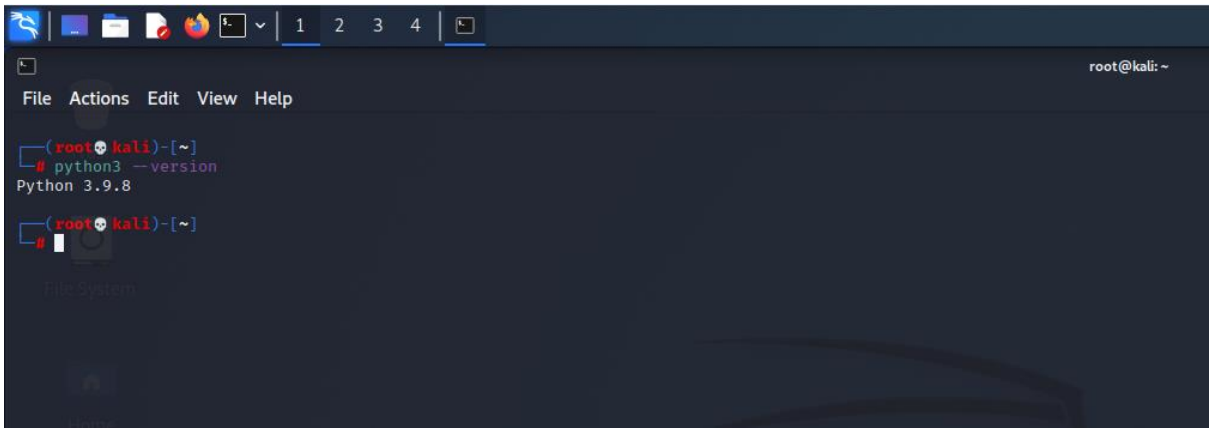
A. Windows

First check python is properly installed in your current machine. Open your terminal and tape this command “python –version” as it indicates the figure below. The minimum required version of python is 3.7 or above. If you haven’t installed python on your machine, please click this link which redirects you to the download page.



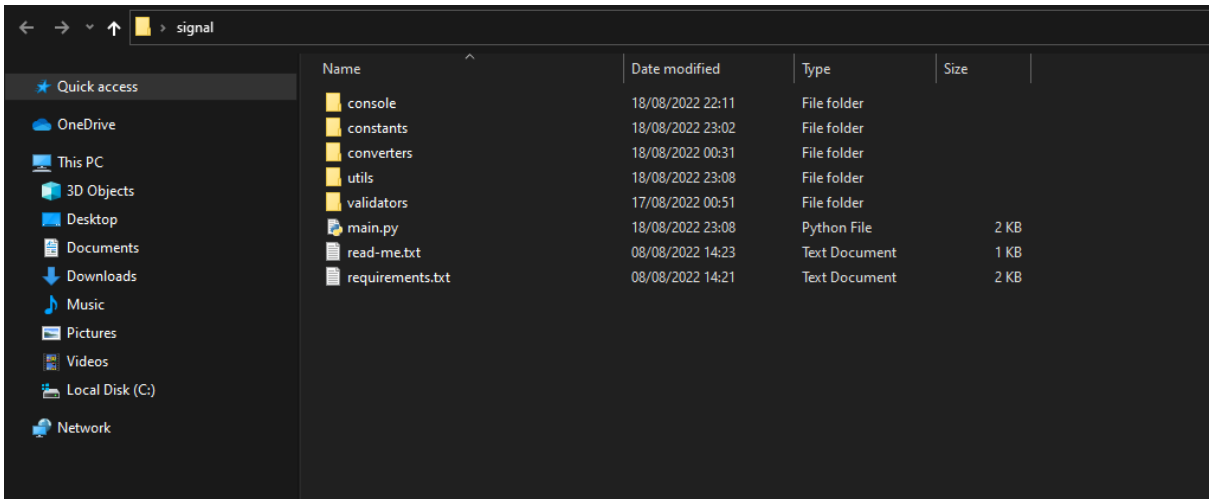
B. Linux

For Linux users, python comes with any distribution of Linux. To check if it’s properly installed in your current machine, open your terminal and tape this command “python3 –version”. If you haven’t got the message in your terminal as it indicates the figure below. You should run the first command “sudo apt-get update” then run the second command “sudo apt-get install python3.7”.

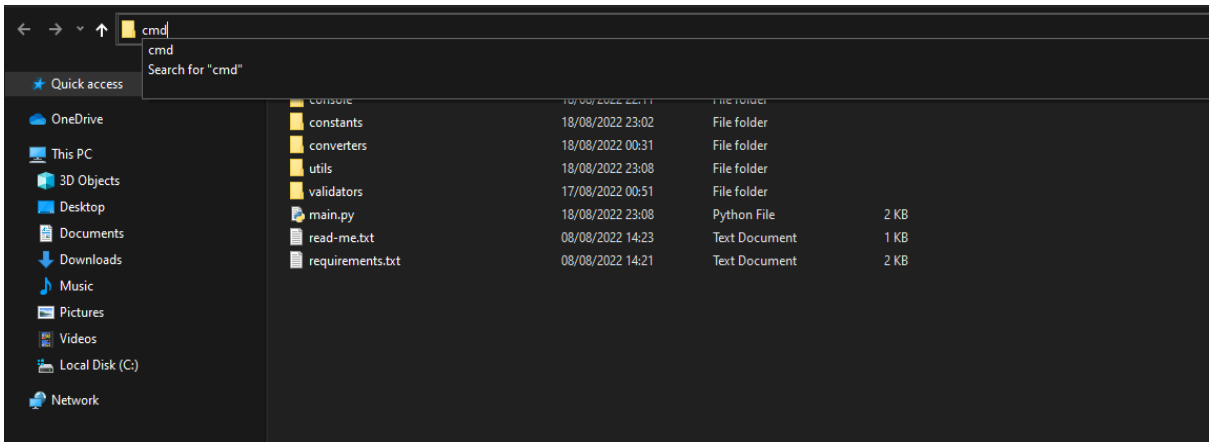


III. Project Installation (Window users)

Download the project folder, you can drag and drop it on your desktop as it indicates the figure below.

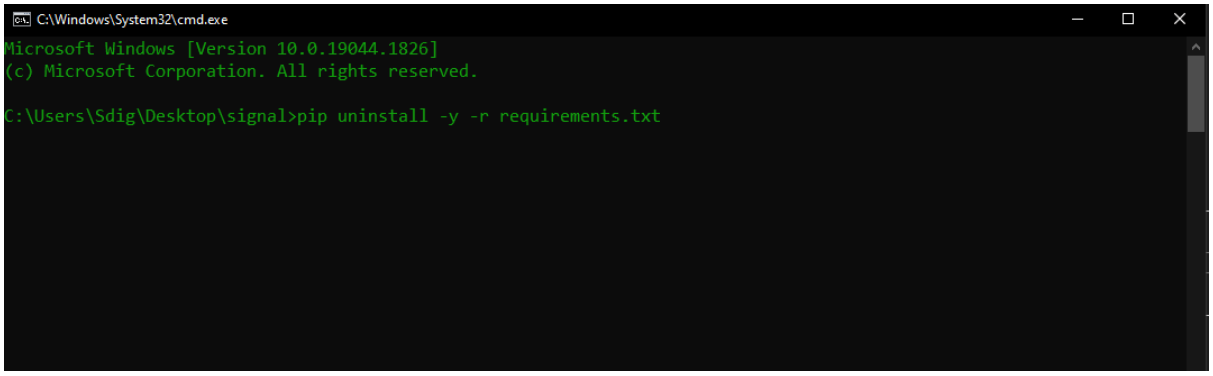


In the current project directory, select the bar in the top and tape the command “cmd”, then press enter keyboard as it indicates the figure below.

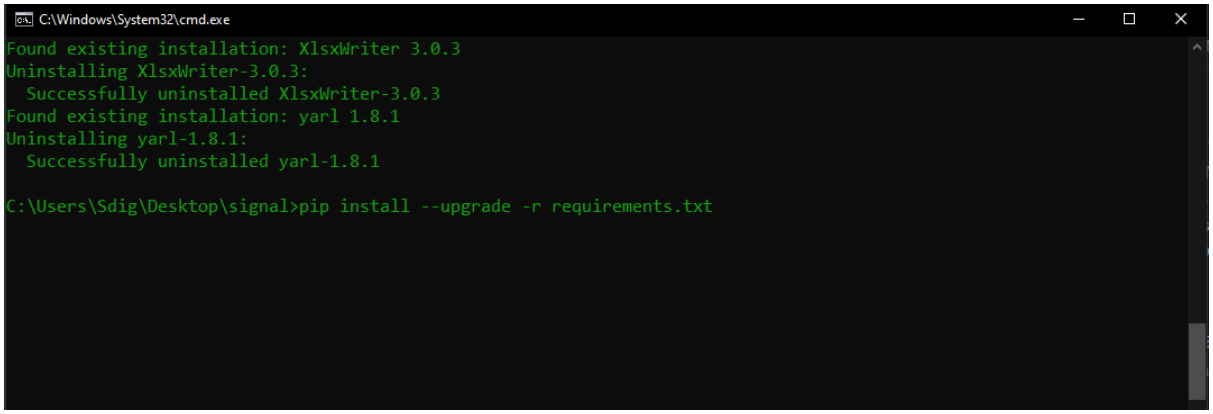


A black screen will appear as your main terminal as it indicates the figure bellow.

In the current project directory, there is a file called “read-me.txt”. Open the file, copy the first command and paste it in the terminal and press “enter”. This command will uninstall the previous installed version of packages in the “requirements.txt” as it indicates the figure below.

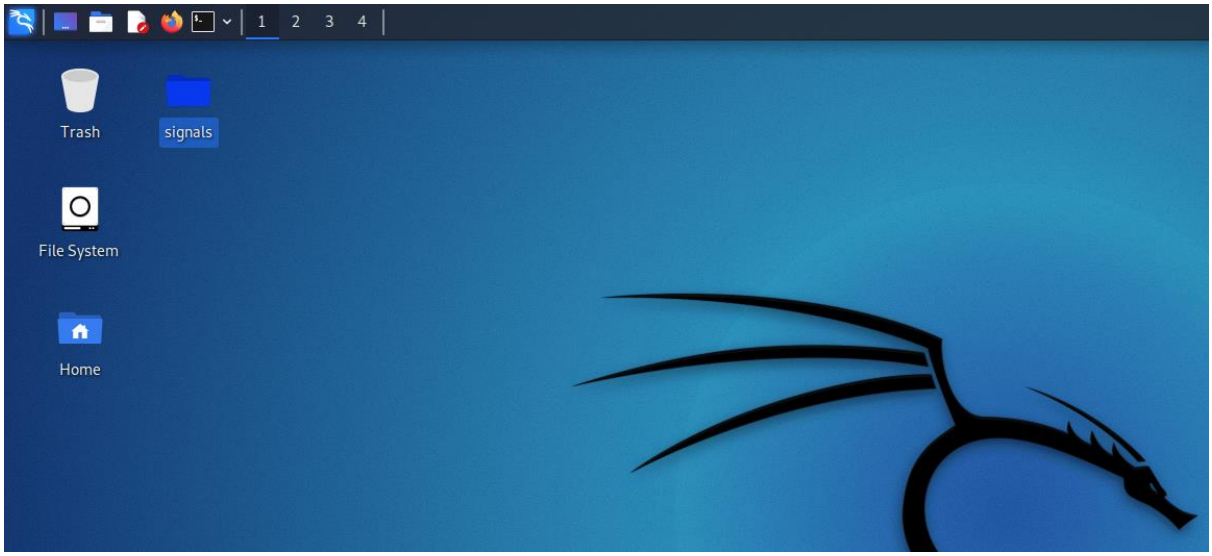


When the execution of the last command is complete, copy the second command and paste it in the terminal and press enter. This command will install the missing required dependencies as it indicates the figure below.

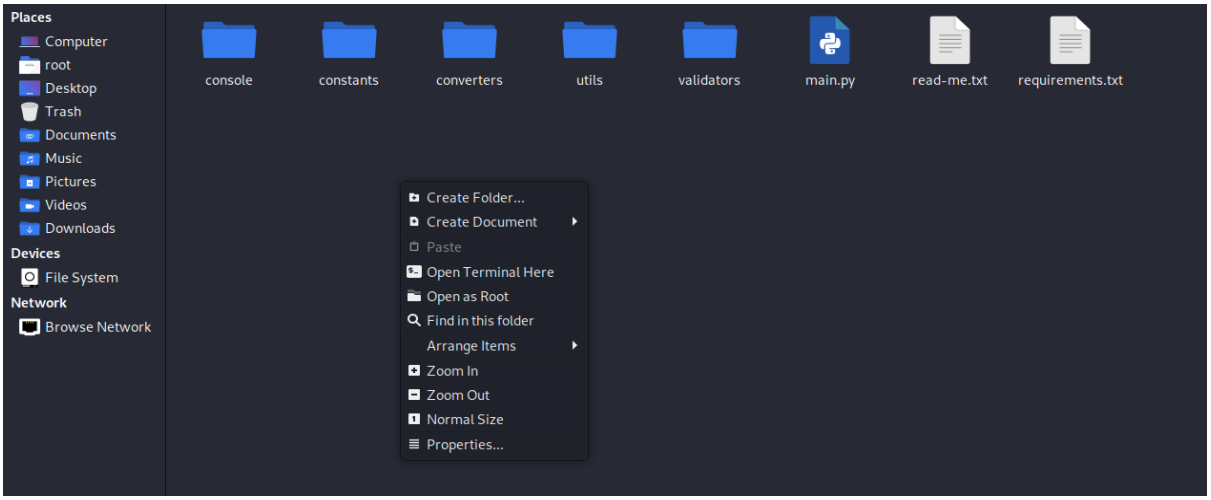


IV. Project Installation (Linux users)

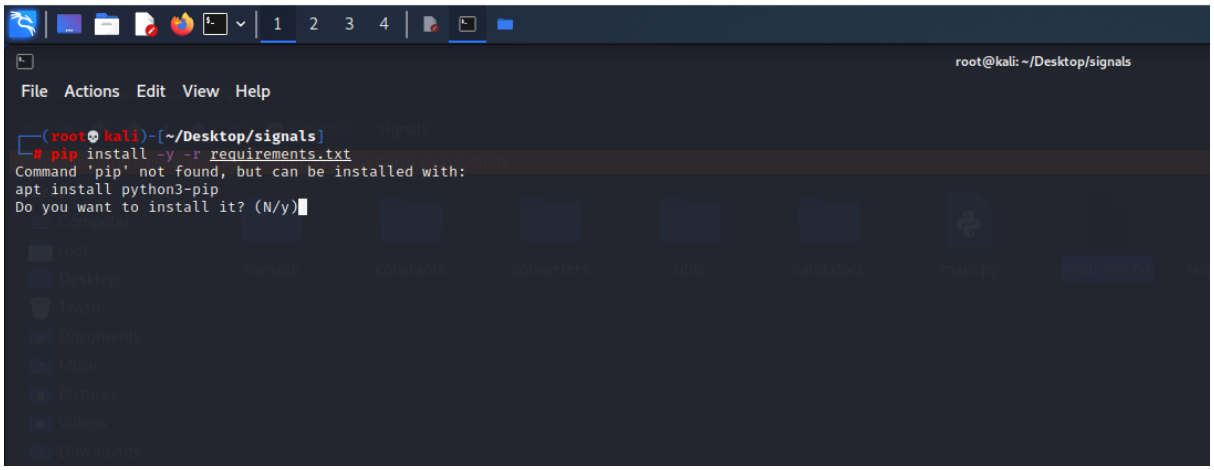
Download the project folder, you can drag and drop it on your desktop as it indicates the figure below.



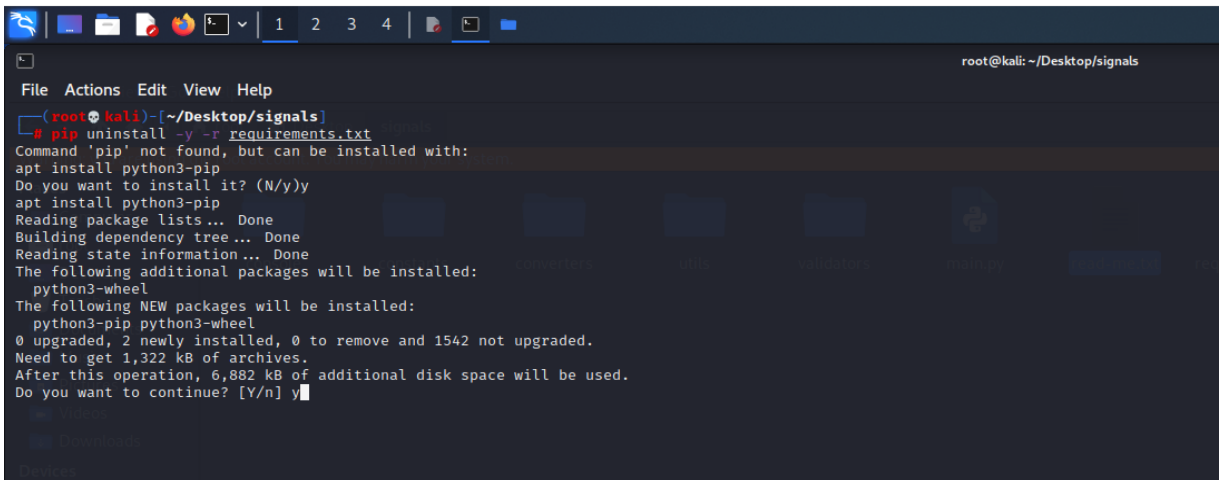
Open the project directory and press the right side of the mouse. This menu will appear. Then choose “Open Terminal Here” option as it demonstrates the figure below.



In your terminal write the first command “pip uninstall -y -r requirements.txt” and press enter as it demonstrates the figure below.



After the completion of the first command, type the final command “pip install -y -r requirements.txt” and press enter as it demonstrates the figure below.



V. Features

In the current section, we will talk about the different features which the script provides.

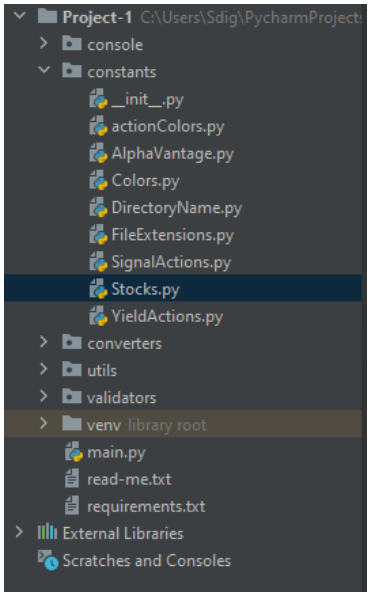
A. Introduction

The python script offers 6 main features.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

__author__ = "Digitarab"
__copyright__ = "Copyright 2022, Digitarab"
__license__ = "GPL"
__version__ = "1.0.0"
__maintainer__ = "Digitarab"
__status__ = "Production"

from alpha_vantage.timeseries import TimeSeries
from converters.dfConverter import convertDfToExcel
from utils.alphaVantageUtil import getAlphaVantageDataStock, initializeAlphaVantage
from utils.calculateIndicatorsUtil import calculateIndicators
from utils.defineActionIndicatorUtil import defineActionIndicators
from utils.signalActionsChartUtil import drawSignalActionsChart
from utils.yieldSignalUtil import calculateYieldSignalAction

if __name__ == '__main__':
    try:
        # Get alpha vantage instance
        ts: TimeSeries = initializeAlphaVantage()
        # Read stock symbol and check if it is available
        data, meta_data = getAlphaVantageDataStock(timeSeries=ts)
        # Calculate indicators white-indicator, green-indicator, grey-indicator, rose-indicator
        df = calculateIndicators(data=data)
        # Define action indicators (open-buy, close-buy, open-sell, close-sell)
        df = defineActionIndicators(df=df)
        # Calculate yields for every signal actions
        df = calculateYieldSignalAction(df=df)
        # Convert data frame to excel
        convertDfToExcel(df=df, metaData=meta_data)
        # Draw indicator fields (white-indicator, green-indicator, grey-indicator, rose-indicator) + (open-buy, close-buy, open-sell, close-sell)
        drawSignalActionsChart(df=df, metaData=meta_data)
    except Exception as ex:
        print(ex)
        exit(1)

```

B. Extract data from alpha vantage API

Before calling alpha vantage API, you should provide your own key in “Stocks.py''. This file is located in the constants folder as it demonstrates the figure below. If you don’t provide the key an exception will be thrown.




```python
# -*- coding: utf-8 -*-
from enum import Enum, unique


@unique
class Stocks(Enum):
    API_KEY = ''
    TESLA = 'TSLA'
    MICROSOFT = 'MSFT'
    APPLE = 'AAPL'
    AMAZON = 'AMZN'
    NVIDIA = 'NVDA'
    VISA = 'V'
    FACEBOOK = 'META'
```

When the API key is provided, the “initializeAlphaVantage'' function is executed. The user is asked to write the stock symbol in the terminal. Then, an API call will be made to check if the stock symbol is correct or not. If it’s wrong, the user is obliged to rewrite the stock symbol until he writes a valid one as it demonstrates the figure below.

```
Run: main
C:\Users\Sdig\PycharmProjects\Project-1\venv\Scripts\python.exe C:/Users/Sdig/PycharmProjects/Project-1/main.py
-> Please enter a symbol to see if it's found in alpha vantage database: something_wrong
-> Launching search process for something_wrong data. Please wait ...
-> Symbol not found: Invalid API call. Please retry or visit the documentation (https://www.alphavantage.co/documentation/) for TIME_SERIES_INTRADAY.!
-> Please enter a symbol to see if it's found in alpha vantage database:
```

When the user provides the correct stock symbol, the “getAlphaVantageDataStock” function is executed to get data from the response call. It returns the result data as the first variable and metadata as the second variable as it shows in the figure below.

```python
# -*- coding: utf-8 -*-
from alpha_vantage.timeseries import TimeSeries

from console.readFromConsole import readStockSymbol
from constants.Stocks import Stocks
from validators.apiKeyValidator import isKeyProvided


# Get alphaVantage instance
def initializeAlphaVantage():
    try:
        isKeyProvided()
        return TimeSeries(key=Stocks.API_KEY.value, output_format='pandas')
    except Exception as ex:
        raise Exception(ex)


# Get alpha - vantage data stock by giving a symbol
def getAlphaVantageDataStock(timeSeries: TimeSeries) -> list:
    while True:
        try:
            symbol = readStockSymbol()
            print('-> Launching search process for {} data. Please wait ...'.format(symbol))
            data, meta_data = timeSeries.get_intraday(symbol=symbol, interval='1min', outputsize='full')
            print('-> Success ...')

            return [data, meta_data]
        except Exception as ex:
            print("-> Symbol not found: {}!".format(ex))
```

C. Calculate stock indicators

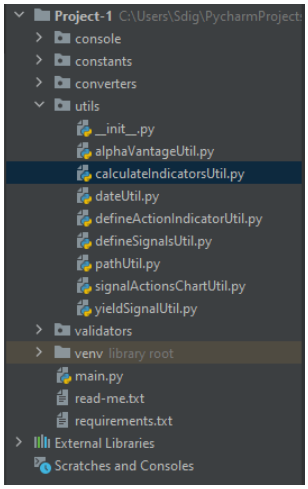
After getting data from Alpha Vantage API, “calculateIndicators” function will run. It accepts the first return of the “getAlphaVantageDataStock '' function and returns a data frame. It calculates five indicators as it demonstrates the figure below:

- Grey Indicator: calculates the moving average of 20 observations.
- White Indicator: close value.
- Standard deviation Indicator: calculate the standard deviation of 20 close value.

- Rose Indicator: uses this formula (moving average + 2 * standard deviation).
- Green Indicator: uses this formula (moving average - 2 * standard deviation).

```python
# -*- coding: utf-8 -*-
from typing import Optional

import pandas as pd
from pandas import DataFrame
from constants.AlphaVantage import AlphaVantageDataFields


# Calculate a bunch of indicators (green-indicator, white-indicator, green-indicator, grey-indicator)
from constants.Stocks import StockIndicators


def calculateIndicators(data: list) -> Optional[DataFrame]:
    try:
        df = pd.DataFrame(data) \
            .sort_values(by=[AlphaVantageDataFields.DATE.value], ascending=True)

        # Calculate gey indicator (Moving average)
        df = calculateGreyIndicator(df)

        # Calculate white indicator (Close value)
        df = calculateWhiteIndicator(df)

        # Calculate standard deviation value
        df = calculateStandardDeviation(df)

        # Calculate rose indicator (Moving average + 2 * standardDeviation)
        df = calculateRoseIndicator(df)

        # Calculate green indicator (Moving average - 2 * standardDeviation)
        df = calculateGreenIndicator(df)

        return df
    except Exception as ex:
        print("-> Unable to calculate indicators: {}!".format(ex))
        exit(1)
```

This implementation of this feature is located in “calculateIndicatorsUtil” under “utils” directory. We add constant indicators’ names as Enum to refer to them around the project. They are located in the “Stocks.py '' file under “constants” folder as it demonstrates the figures below.




```python
@unique
class StockIndicators(Enum):
    MOVING_AVERAGE = 20
    STANDARD_DEVIATION_INDICATOR = 'standard_deviation_indicator'
    ROSE_INDICATOR = 'rose_indicator'
    WHITE_INDICATOR = 'white_indicator'
    GREEN_INDICATOR = 'green_indicator'
    GREY_INDICATOR = 'grey_indicator'

```

D. Define Action indicators

After calculating stock indicators, it's time to define actions (open-buy, close-buy, open-sell, close-sell). These actions are calculated while running "defineActionIndicators" function, it accepts a data frame as param and returns an updated data frame. This implementation of this feature is located in "calculateActionIndicatorsUtil" under "utils" directory. We add constant action indicators' names as Enum to refer them around the project. They are located in "Stocks.py" file. The actions are defined by following these conditions:

- Open-Buy: It's true when WHITE INDICATOR < GREEN INDICATOR.
- Close-Buy: It's true when WHITE INDICATOR >= GREY INDICATOR.
- Open-Sell: It's true when WHITE INDICATOR > ROSE INDICATOR.
- Close-Sell: It's true when WHITE INDICATOR <= GREY INDICATOR.

```python
# -*- coding: utf-8 -*-
from typing import Optional
from typing import Tuple, List, Union, Any

import numpy as np
from pandas import DataFrame
from pandas import Series


# Define a bunch of action indicators (open-buy, close-buy, open-sell, close-sell)
from constants.Stocks import StockIndicators, StockActions


def defineActionIndicators(df: Optional[DataFrame]) -> Optional[DataFrame]:
    try:
        # Define open-sell indicator
        df = defineOpenSellActionIndicator(df=df)

        # Define close-sell indicator
        df = defineCloseSellActionIndicator(df=df)

        # Define close-buy indicator
        df = defineOpenBuyActionIndicator(df=df)

        # Define close-buy indicator
        df = defineCloseBuyActionIndicator(df=df)

        # Fill first nth rows of indicators with 0. It depends on the value of moving average
        df.loc[df[StockIndicators.STANDARD_DEVIATION_INDICATOR.value].isna(), [el.value for el in StockActions]] = 0

        return df
    except Exception as ex:
        print("-> Unable to define action indicators: {}!".format(ex))
        exit(1)
```

E. Calculate yield per signal

After defining stock indicators, it's time to filter correct signals from the wrong ones.

Example:

- It's impossible to define close buy action when there is no open-buy action before.
- It's impossible to define close sell action when there is no open-sell action before.

So, we implement "filterSignalsAction" function to filter the defined actions. It accepts data frame as param and returns a list of valid "buy-action" as first param and list of valid "sell-actions" as second param. This function is implemented in "defineSignalsUtil.py" under the "utils" folder as it demonstrates the figure below.

Example:

For buy-actions: [[20,21],24], [50,50], 58]]

[20,21] = 20 indicates the first open index of first open-buy, 21 indicates the last open-buy index, 24 indicates the index of close-buy. Same is applied for sell-actions.

```python
# -*- coding: utf-8 -*-
from typing import Optional
from typing import Tuple, List, Union, Any
from pandas import DataFrame, Series
from constants.Stocks import StockIndicators, StockActions


# Filter signal actions
def filterSignalsAction(df: Optional[DataFrame]) -> Tuple[List[List[Union[List[int], List[List[int]], int]]], List[List[Union[List[int], List[List[int]], int]]]]:
    try:
        # Calculating firstIndex and lastIndex of valid buy operation
        buySignal = filterOpenCloseBuySignal(df=df)

        # Calculating firstIndex and lastIndex of valid sell operation
        sellSignal = filterOpenCloseSellSignal(df=df)

        return buySignal, sellSignal
    except Exception as ex:
        print("-> Unable to filter signal actions: {}!".format(ex))
        exit(1)

```

Then it's time for the yield calculation. We Applied these formulas:

- For Buy-Signal: (Price Open Buy/Price Close Buy) – 1.
- For Sell-Signal: - ((Price Open Sell/Price Close Sell) – 1).

We took in consideration if there are multiple open buy indexes before close-buy action. Same is applied for sell-signals. The calculations' logic is implemented in "yieldSignalUtil.py" under the "utils" folder.

```python
# -*- coding: utf-8 -*-
from typing import Optional, List, Union
from pandas import DataFrame
from constants.AlphaVantage import AlphaVantageDataFields
from constants.SignalActions import SignalActions
from constants.Stocks import StockActions, StockIndicators
from constants.YieldActions import YieldActions
from utils.defineSignalsUtil import filterSignalsAction


# Define signal actions
def calculateYieldSignalAction(df: Optional[DataFrame]) -> Optional[DataFrame]:
    try:
        # Get coherent signal actions
        buySignal, sellSignal = filterSignalsAction(df=df)

        df = yieldPreconfigure(df=df)
        df = calculateBuyActionYield(df=df, buySignal=buySignal)
        df = calculateSellActionYield(df=df, sellSignal=sellSignal)

        return df
    except Exception as ex:
        print("-> Unable to calculate yield for signal actions: {}!".format(ex))
        exit(1)
```

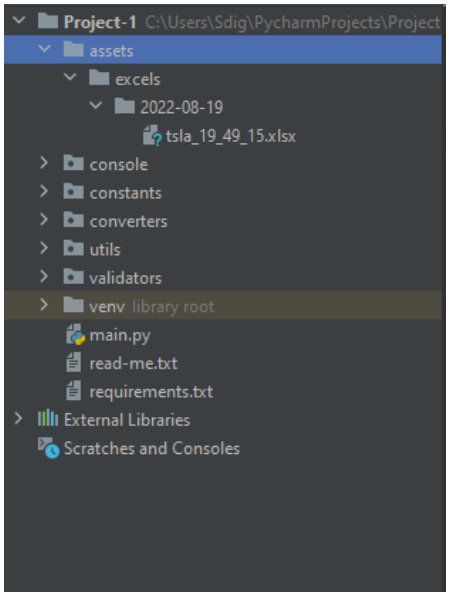
F. Convert data frame to excel

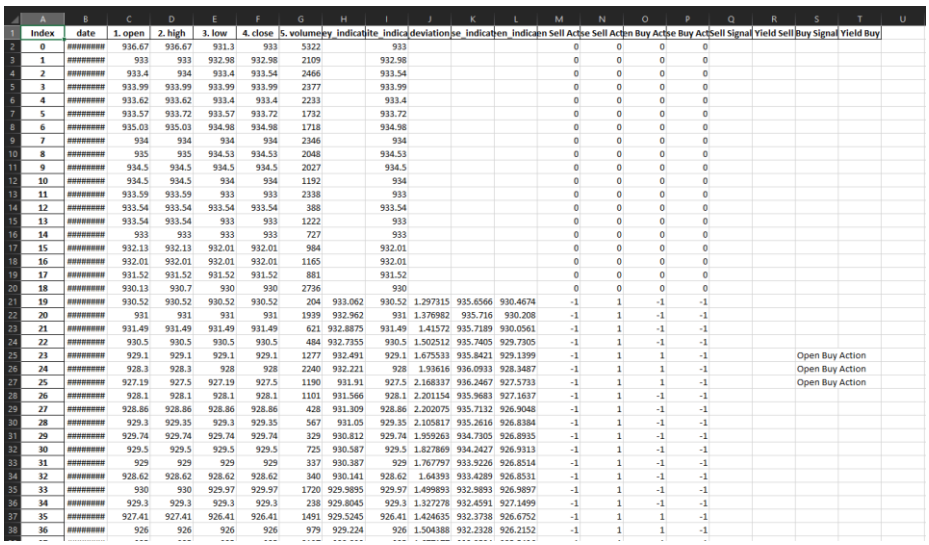
After filtering and calculating yields for each action. It's time to convert result data to excel. "convertDfToExcel" function in "dfConverter.py" file under "converters" folder is responsible for converting process. It took data frame as first param and metadata as second param. A menu of available columns in data frame appears. The user has to write down the index of elements. If he wants to select all fields he should write "All" in console. To finish selecting, he should write "Stop" as it indicates the figure below.

Note: The symbols are case insensitive.

```
-> Please enter a symbol to see if it's found in alpha vantage database: tsla
-> Launching search process for tsla data. Please wait ...
-> Success ...
-> Displaying available fields ...
1 -- date
2 -- 1. open
3 -- 2. high
4 -- 3. low
5 -- 4. close
6 -- 5. volume
7 -- grey_indicator
8 -- white_indicator
9 -- standard_deviation_indicator
10 -- rose_indicator
11 -- green_indicator
12 -- Open Sell Action
13 -- Close Sell Action
14 -- Open Buy Action
15 -- Close Buy Action
16 -- Sell Signal
17 -- Yield Sell
18 -- Buy Signal
19 -- Yield Buy
-> Please enter key 'Stop' to terminate reading field indexes - 'All' key to select all fields
-> Enter your choice: |
```

After selecting fields, it comes the saving process. We took this convention to organise folder structure (excel files are located under "assets"/" excels"/" current date" folder which is automatically created with an excel file "stocksymbol_hours_minutes_seconds".xlsx). This protocol is used to prevent name incoherence as it indicates the figures below.





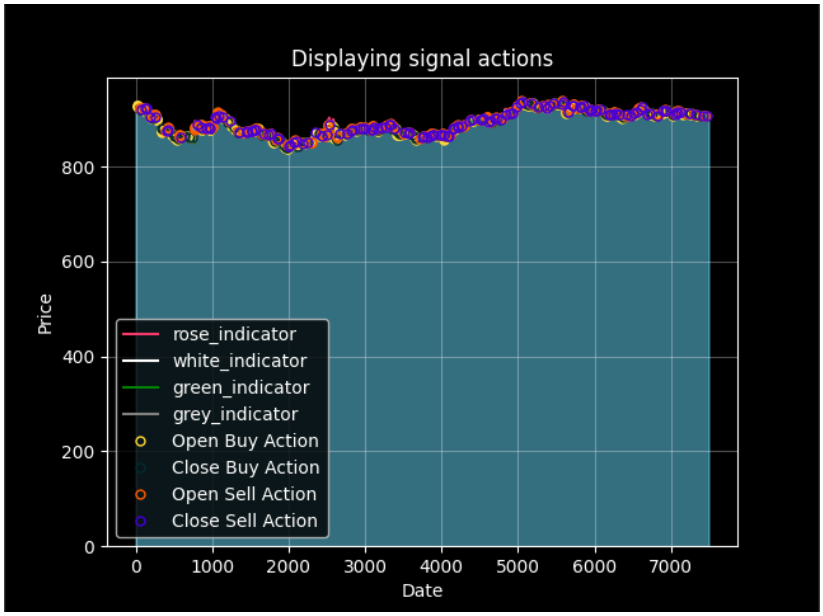
Index	date	1. open	2. high	3. low	4. close	5. volume	grey_indicator	white_indicator	deviation	rose_indicator	green_indicator	Open Sell Action	Close Sell Action	Open Buy Action	Close Buy Action	Sell Signal	Yield Sell	Buy Signal	Yield Buy
0	########	936.67	936.67	931.3	933	5322		933				0	0	0	0				
1	########	933	933	932.98	932.98	2109		932.98				0	0	0	0				
2	########	933.4	934	933.4	933.54	2466		933.54				0	0	0	0				
3	########	933.99	933.99	933.99	933.99	2377		933.99				0	0	0	0				
4	########	933.62	933.62	933.4	933.4	2233		933.4				0	0	0	0				
5	########	933.57	933.72	933.57	933.72	1732		933.72				0	0	0	0				
6	########	935.03	935.03	934.98	934.98	1718		934.98				0	0	0	0				
7	########	934	934	934	934	2346		934				0	0	0	0				
8	########	935	935	934.53	934.53	2048		934.53				0	0	0	0				
9	########	934.5	934.5	934.5	934.5	2027		934.5				0	0	0	0				
10	########	934.5	934.5	934	934	1192		934				0	0	0	0				
11	########	933.59	933.59	933	933	2338		933				0	0	0	0				
12	########	933.54	933.54	933.54	933.54	388		933.54				0	0	0	0				
13	########	933.54	933.54	933	933	1222		933				0	0	0	0				
14	########	933	933	933	933	727		933				0	0	0	0				
15	########	932.13	932.13	932.01	932.01	984		932.01				0	0	0	0				
16	########	932.01	932.01	932.01	932.01	1165		932.01				0	0	0	0				
17	########	931.52	931.52	931.52	931.52	881		931.52				0	0	0	0				
18	########	930.13	930.7	930	930	2736		930				0	0	0	0				
19	########	930.52	930.52	930.52	930.52	204	933.062	930.52	1.297315	935.6566	930.4674	-1	1	-1	-1				
20	########	931	931	931	931	1939	932.962	931	1.376982	935.716	930.208	-1	1	-1	-1				
21	########	931.49	931.49	931.49	931.49	621	932.8875	931.49	1.41572	935.7189	930.0561	-1	1	-1	-1				
22	########	930.5	930.5	930.5	930.5	484	932.7355	930.5	1.502512	935.7405	929.7305	-1	1	-1	-1				
23	########	929.1	929.1	929.1	929.1	1277	932.491	929.1	1.675533	935.8421	929.1399	-1	1	1	-1			Open Buy Action	
24	########	928.3	928.3	928	928	2240	932.221	928	1.93616	936.0933	928.3487	-1	1	1	-1			Open Buy Action	
25	########	927.19	927.5	927.19	927.5	1190	931.91	927.5	2.168337	936.2467	927.5733	-1	1	1	-1			Open Buy Action	
26	########	928.1	928.1	928.1	928.1	1101	931.566	928.1	2.201154	935.9683	927.1637	-1	1	-1	-1				
27	########	928.86	928.86	928.86	928.86	428	931.309	928.86	2.202075	935.7132	926.9048	-1	1	-1	-1				
28	########	929.3	929.35	929.3	929.35	567	931.05	929.35	2.105817	935.2616	926.8384	-1	1	-1	-1				
29	########	929.74	929.74	929.74	929.74	329	930.812	929.74	1.959263	934.7305	926.8935	-1	1	-1	-1				
30	########	929.5	929.5	929.5	929.5	725	930.587	929.5	1.827869	934.2427	926.9313	-1	1	-1	-1				
31	########	929	929	929	929	337	930.387	929	1.767797	933.9226	926.8514	-1	1	-1	-1				
32	########	928.62	928.62	928.62	928.62	340	930.141	928.62	1.64393	933.4289	926.8531	-1	1	-1	-1				
33	########	930	930	929.97	929.97	1720	929.9895	929.97	1.499893	932.9893	926.9897	-1	1	-1	-1				
34	########	929.3	929.3	929.3	929.3	238	929.8045	929.3	1.327278	932.4591	927.1499	-1	1	-1	-1				
35	########	927.41	927.41	926.41	926.41	1491	929.5245	926.41	1.424635	932.3738	926.6752	-1	1	1	-1				
36	########	926	926	926	926	979	929.224	926	1.504388	932.2328	926.2152	-1	1	1	-1				

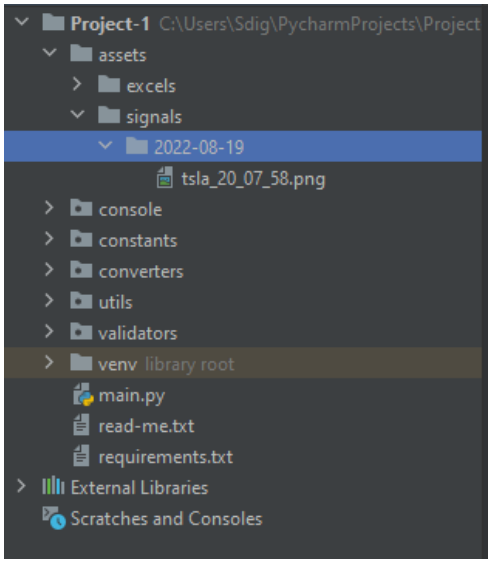
G. Draw signal chart

This final feature is responsible for drawing a signal chart after defining signals(buy-sell). “drawSignalActionsChart '' function is located in “signalActionsChartUtil '' file under the “utils” folder as it indicates in the figure below.

```python
# -*- coding: utf-8 -*-
import re
from typing import List, Optional
import matplotlib.pyplot as plt
from pandas import DataFrame
from constants.AlphaVantage import AlphaVantageDataFields
from constants.Colors import COLORS
from constants.DirectoryName import DirectoryName
from constants.FileExtensions import FileExtensions
from constants.SignalActions import SignalActions
from constants.Stocks import StockIndicators, StockActions
from constants.actionColors import ActionColors
from utils.pathUtil import saveFileInDirectory


# Displaying signal actions chart
def drawSignalActionsChart(df: Optional[DataFrame], metaData: str) -> None:
    try:
        print('-> Drawing signal actions chart in progress. Please wait ...')
        # Get indicator Fields
        fields: List[str] = [el.value for el in StockIndicators if not el.value in (StockIndicators.MOVING_AVERAGE.value, StockIndicators.STANDARD_DEVIATION_INDICATOR.value)]

        # Use dark mode
        plt.style.use('dark_background')

        # Fil close value with color
        df[StockIndicators.WHITE_INDICATOR.value].plot.area(stacked=False, color=COLORS['TRANSPARENT_BLUE'], label='')

        # Check if the given column exists in dataframe else throw exception
        for field in fields:
            colorName = re.sub('_indicator', '', field).upper()
            plt.plot(df[field], label=field, color=COLORS[colorName])

        # Indicators
        sellSignalField = SignalActions.SELL.value
        buySignalField = SignalActions.BUY.value
        openBuyAction = StockActions.OPEN_BUY.value
        closeBuyAction = StockActions.CLOSE_BUY.value
        openSellAction = StockActions.OPEN_SELL.value
        closeSellAction = StockActions.CLOSE_SELL.value
        closePriceField = AlphaVantageDataFields.CLOSE.value

        # Plotting signal actions
        plt.plot(df.loc[df[buySignalField] == openBuyAction].index,
                 df[closePriceField][df[buySignalField] == openBuyAction], 'o',
                 color=ActionColors.OPEN_BUY.value, ms=5, markerfacecolor='none', label=openBuyAction)

        plt.plot(df.loc[df[buySignalField] == closeBuyAction].index,
                 df[closePriceField][df[buySignalField] == closeBuyAction], 'o',
                 color=ActionColors.CLOSE_BUY.value, ms=5, markerfacecolor='none', label=closeBuyAction)

        plt.plot(df.loc[df[sellSignalField] == openSellAction].index,
                 df[closePriceField][df[sellSignalField] == openSellAction], 'o',
                 color=ActionColors.OPEN_SELL.value, ms=5, markerfacecolor='none', label=openSellAction)

        plt.plot(df.loc[df[sellSignalField] == closeSellAction].index,
                 df[closePriceField][df[sellSignalField] == closeSellAction], 'o',
                 color=ActionColors.CLOSE_SELL.value, ms=5, markerfacecolor='none', label=closeSellAction)

        plt.title("Displaying signal actions")
        plt.rcParams['figure.figsize'] = 20, 10
        plt.grid(True, alpha=.3)
        plt.ylabel('Price')
        plt.xlabel('Date')
        fig = plt.gcf()
        plt.legend()
        plt.show()

        print('-> Success ...')

        saveSignalActionsChart(figure=fig, metaData=metaData)
    except Exception as ex:
        print("-> Unable to draw signal actions' plot: {}!".format(ex))
        exit(1)
```

The chart appears. It will be saved by following the same previous protocol mentioned in excel files as it indicates the figures below.





VI. Conclusion

In conclusion, we have demonstrated all the features which are involved in this script.