A Python Project

Analyzing and Predicting Stock Trend via Python

Zhichong Li

Overall

This project is to build a native App by using Python. The App can be used to analyze and predict future trends for one stock by using history data.

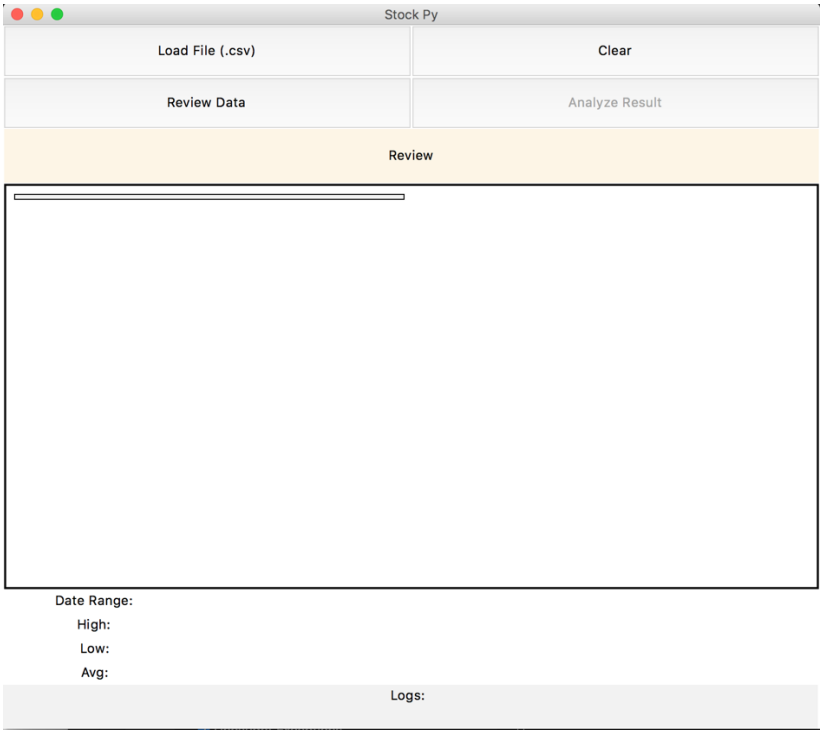
- The data to be loaded or imported is required to be ".csv" format. And it's recommended to be downloaded from Yahoo Finance. Data date range should be at least 1 year for better result and it cannot be less than 120 days.

- Load File should import raw data and show history data in canvas figure for 120 days, simple analyze for history data should show date range, high, low and average price for whole raw data.

- Analyze result is to predict the trends for the upcoming 30 days by using history data. Trends will show in canvas figure.

UI Components

There are four main UI components for this App: function buttons, canvas frame, raw data board, status bar.



Landing Screen

There are four buttons:

- Load Files – click on this button will bring up a file dialog box so users can select the ".csv" file to import.
- Clear – clear button will work after data gets loaded, if users don't want to see current stock data, clear function will clear history data from the canvas and the data review board.
- Analyze Result – once history data has been loaded, users can click on analyze result to do the prediction for selected stock for 30 days in the future. And result will be plotted on canvas. If there is no ".csv" file be imported, this button should be disabled.
- Review Data – users can switch back to history data review from analyze view.

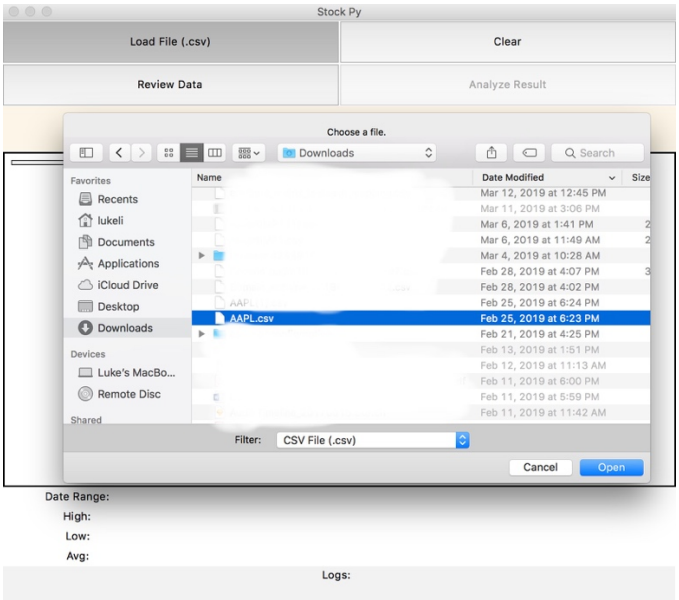
Functions

Prepare

- Users can select one stock from Yahoo Finance, and download Historical Data CSV file. It is recommended to download data for at least 1 year, and longer can be better, but should be at least data for 120 days.

Load Data

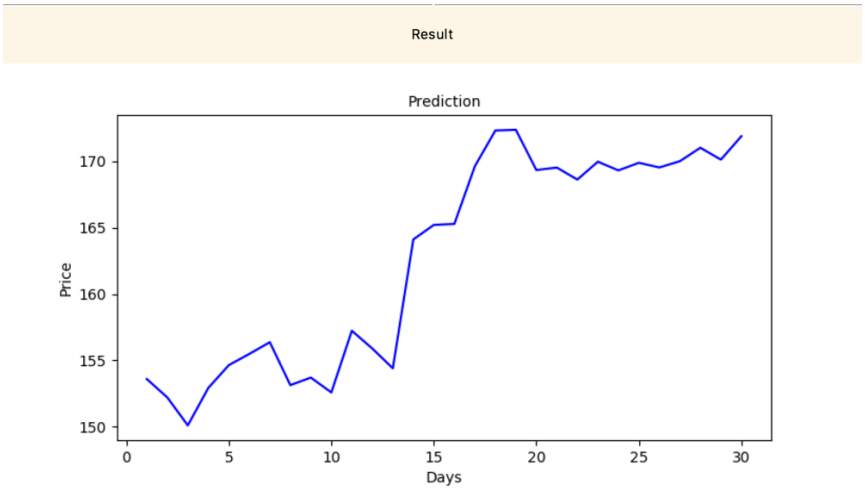
- Users can click on load data button and select the downloaded file:



- Before any data be loaded, Analyze Result button should be disabled.
- After data be loaded, canvas should show figure for the most recent 120 days, and data review board should show date range for raw data and high, low, average price during this period. Logs should show file directory and name.

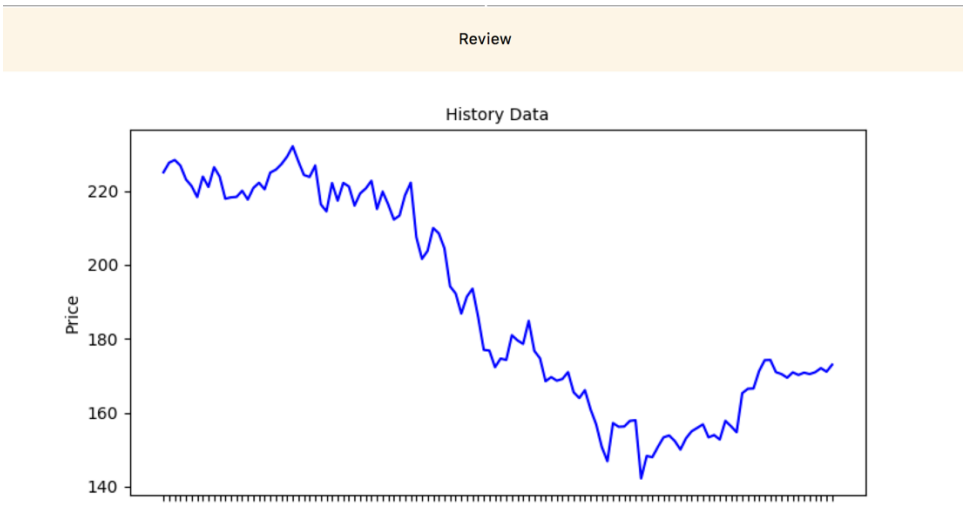
Analyze Result

- Once history data has been loaded, Analyze Result button will be enabled.
- Users can click on Analyze Result button, and get the prediction data for upcoming 30 days. The prediction data is computed based on historic data of the past 30 days, it uses linear regression algorithms to do the computations.
- The prediction data will be plotted on the canvas.



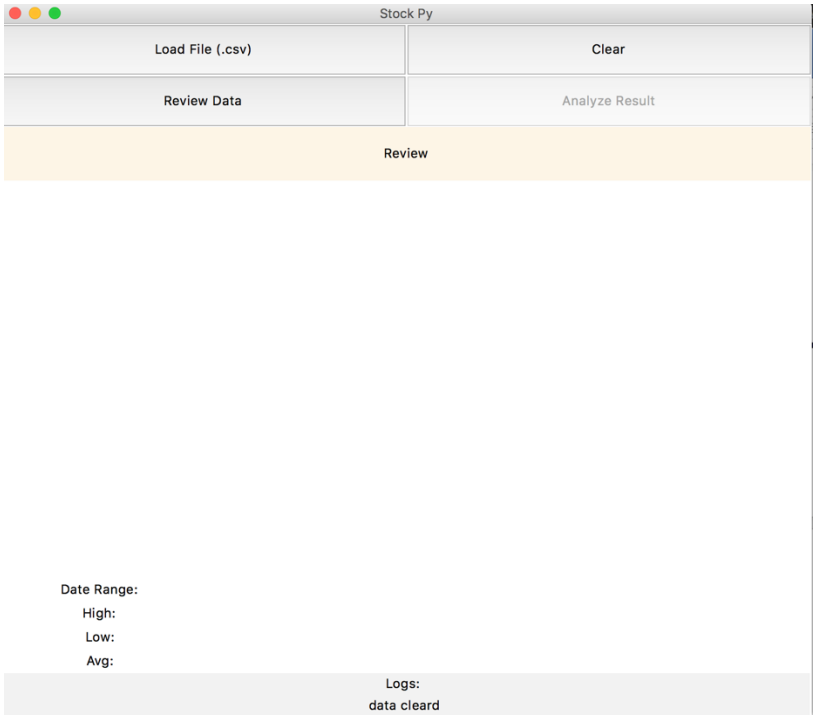
Review Data

- When users are seeing predicted data, they can go back to see loaded historic data by click on Review Data button, the canvas will show historic data again.



Clear

- When users are seeing either historical data or predicted data, they can clean the loaded stock data by click on Clear button. Then the canvas, review data board will be cleaned and log will show data cleared. Then users can either quit or load data for other stocks. Since there is no data loaded, the Analyze Result button will be disabled.



Code

Modules

- There are two files for this App, “stock.py” and “analyze.py”.
- “stock.py” is the main file, it has the UI code and functions for button events and canvas plots. App should be started from this file.
- “analyze.py” is a separated module created for data computations. It is imported by main file.
- All other modules being used by this app:
  Tkinter, csv, numpy, pandas, matplotlib, sklearn

Implementations

- UI of this App is built by tkniter, buttons, frames, canvas, labels are used for different components. “filedialog” from tkinter is used to get import file directory.
- Multi modules method is used, ad data computation functions are reusable when either load or analyze or review data. These functions are written in the module called analyze.py. and the main file imports functions and use them with button events.
- OOP method is used in the code. Class “dataSwitch” is used to switch canvas figure when users are switching between Review and Result. This class contains three methods, and since they are all used for data view switch, so they can be grouped in the same object class.
- “csv” and “pandas” modules are used to import csv data to format data into list or columns, and “numpy” is used for math and array computations. “matplotlib” is used to plot stock figures. “sklearn” is used for data training and linear regression processing to predict stock trends.

Run

- Install python libraries:

  Pip3 install numpy sklearn pandas matlpotlib

Source Code
Stock.py

```
from tkinter import *
from tkinter import filedialog
import csv
import numpy as np
import pandas as pd
import matplotlib as mpl
mpl.use('TkAgg')
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import analyze

root = Tk()


def load():
    dir = filedialog.askopenfilename(initialdir="/",
                                     filetypes=(("CSV File", "*.csv"),
                                                ("All Files", "*.*")),
                                     title="Choose a file."
                                     )
    global stockData, dateRangeData, adjCloseData
    if dir:
        with open(dir) as f:
            stockData = [{k: v for k, v in row.items()}
                         for row in csv.DictReader(f, skipinitialspace=True)]
        adjCloseData = pd.read_csv(dir, usecols=['Adj Close'])
        dateRangeData = analyze.getDateRangeData(stockData)

        date_label_content.config(text='{}   to  {} / {} days'.format(
            dateRangeData["startDate"], dateRangeData["endDate"], len(stockData)))
        high_label_content.config(text=dateRangeData["high"])
        low_label_content.config(text=dateRangeData["low"])
        avg_label_content.config(text=dateRangeData["avg"])
        console_label_body.config(text='loaded from:' + dir)
        result_btn['state'] = 'normal'

        # plot 120 days history data
        x = np.array([o['Date'] for o in stockData[-120:]])
        y = np.array([o['Close'] for o in stockData[-120:]]).astype(np.float)
        plotCanvas(x, y, "History Data", "Price", "Date")


def clear():
    global stockData, adjCloseData
    stockData = []
```

```
    adjCloseData = []
    date_label_content.config(text='')
    high_label_content.config(text='')
    low_label_content.config(text='')
    avg_label_content.config(text='')
    console_label_body.config(text='data cleard')
    result_btn['state'] = 'disabled'
    fig = Figure(figsize=(8, 4))
    canvas = FigureCanvasTkAgg(fig, master=console_canvas)
    canvas.get_tk_widget().grid(row=0, column=0)
    canvas.draw()


class dataSwitch():

    def switchFrame(self, frame):
        review_label.config(text=frame)

    def getResult(self):
        global adjCloseData
        self.switchFrame('Result')
        resuls = analyze.analyze(adjCloseData)
        x = np.arange(1, 31)
        plotCanvas(x, resuls, 'Prediction', 'Price', 'Days')

    def reviewData(self):
        self.switchFrame('Review')
        global stockData
        if stockData:
            x = np.array([o['Date'] for o in stockData[-120:]])
            y = np.array([o['Close']
                          for o in stockData[-120:]]).astype(np.float)
            plotCanvas(x, y, "History Data", "Price", "Date")


def plotCanvas(x, y, title, ylabel, xlabel):
    fig = Figure(figsize=(8, 4))
    a = fig.add_subplot(111)
    a.plot(x, y, color='blue')
    a.set_title(title, fontsize=10)
    a.set_ylabel(ylabel, fontsize=10)
    a.set_xlabel(xlabel, fontsize=10)

    canvas = FigureCanvasTkAgg(fig, master=console_canvas)
    canvas.get_tk_widget().grid(row=0, column=0)
    canvas.draw()


def log_motion_event(event):
```

```
    global log_drag, x1, x2, y1, y2, mouse_x, mouse_y
    x = event.x
    y = event.y
    if not log_drag:
        mouse_x = x
        mouse_y = y
        log_drag = True
        return
    x1 += (x - mouse_x)
    x2 += (x - mouse_x)
    y1 += (y - mouse_y)
    y2 += (y - mouse_y)
    console_canvas.coords(rect, x1, y1, x2, y2)
    mouse_x = x
    mouse_y = y


def log_release_event(event):
    global log_drag
    log_drag = False


root.title('Stock Py')

bar_frame = Frame(root)
review_frame = Frame(root)
result_frame = Frame(root)
console_frame = Frame(root)

# bar panel
load_btn = Button(bar_frame, text='Load File (.csv)',
                  height=3, width=45, command=load)
load_btn.grid(row=0, column=0)

clear_btn = Button(bar_frame, text='Clear', height=3, width=45, command=clear)
clear_btn.grid(row=0, column=1)

review_btn = Button(bar_frame, text='Review Data', height=3,
                    width=45, command=lambda: dataSwitch().reviewData())
review_btn.grid(row=1, column=0)

result_btn = Button(bar_frame, text='Analyze Result', height=3,
                    width=45, command=lambda: dataSwitch().getResult(),
state='disabled')
result_btn.grid(row=1, column=1)

# review panel
figure_frame = Frame(review_frame)
figure_frame.grid(row=1, column=0, sticky='NW')
```

```
grid_frame = Frame(review_frame)
grid_frame.grid(row=3, column=0, sticky='NW')
date_label = Label(grid_frame, text='Date Range: ', width='20')
date_label_content = Label(grid_frame, width='50')
date_label.grid(row=0, column=0)
date_label_content.grid(row=0, column=1)
high_label = Label(grid_frame, text='High: ', width='20')
high_label_content = Label(grid_frame, width='20')
high_label.grid(row=1, column=0)
high_label_content.grid(row=1, column=1)
low_label = Label(grid_frame, text='Low: ', width='20')
low_label.grid(row=2, column=0)
low_label_content = Label(grid_frame, width='20')
low_label_content.grid(row=2, column=1)
avg_label = Label(grid_frame, text='Avg: ', width='20')
avg_label.grid(row=3, column=0)
avg_label_content = Label(grid_frame, width='20')
avg_label_content.grid(row=3, column=1)

console_label_header = Label(console_frame, height=1,
                             text='Logs: ', width=90, bg='gray95')
console_label_header.grid(row=0, column=0, sticky='NW')
console_label_body = Label(console_frame, height=1,
                           width=90, bg='gray95')
console_label_body.grid(row=1, column=0, sticky='NW')

console_canvas = Canvas(review_frame, width=810,
                        height=400)
console_canvas.bind('<B1-Motion>', log_motion_event)
console_canvas.bind('<ButtonRelease-1>', log_release_event)
console_canvas.focus_set()
console_canvas.grid(row=2, column=0)
mouse_x = 0
mouse_y = 0
log_drag = False

x1 = y1 = 10
y2 = 15
x2 = 400
rect = console_canvas.create_rectangle(x1, y1, x2, y2, fill='gray95')


review_label = Label(bar_frame, height=3, width=90,
                     text='Review', bg='old lace')
review_label.grid(row=2, column=0, columnspan='40')
bar_frame.grid(row=0, column=0)
review_frame.grid(row=1)
console_frame.grid(row=2, column=0, sticky='NW')
```

```
# initialize stock data
stockData = []
adjCloseData = []
dateRangeData = {"startDate": "", "endDate": "",
                 "high": "", "low": "", "avg": ""}

mainloop()
```

analyze.py

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing, model_selection, svm


def analyze(data, days=30):
    dataFrame = data
    # set predicting {days} into future
    forecastLength = int(days)
    # data column shifted {days} up
    dataFrame['Prediction'] = dataFrame[['Adj Close']].shift(-forecastLength)
    X = np.array(dataFrame.drop(['Prediction'], 1))
    X = preprocessing.scale(X)
    # set X_forecast
    X_forecast = X[-forecastLength:]
    # remove last {days} from X
    X = X[:-forecastLength]
    y = np.array(dataFrame['Prediction'])
    y = y[:-forecastLength]

    X_train, X_test, y_train, y_test = model_selection.train_test_split(
        X, y, test_size=0.2)

    # Training
    linear = LinearRegression()
    linear.fit(X_train, y_train)
    # Testing
    accurate = linear.score(X_test, y_test)
    print("accurate: ", accurate)
    #  Result
    prediction = linear.predict(X_forecast)
```

```
    return prediction


def getDateRangeData(stockData):
    dateRangeData = {}
    dateRangeData["startDate"] = stockData[0]["Date"]
    dateRangeData["endDate"] = stockData[-1]["Date"]
    dateRangeData["high"] = float(stockData[0]["Adj Close"])
    dateRangeData["low"] = float(stockData[0]["Adj Close"])
    total = 0
    for i in stockData:
        if float(i["Close"]) > dateRangeData["high"]:
            dateRangeData["high"] = float(i["Adj Close"])
        if float(i["Close"]) < dateRangeData["low"]:
            dateRangeData["low"] = float(i["Adj Close"])
        total += float(i["Adj Close"])

    dateRangeData["avg"] = total / len(stockData)
    return dateRangeData
```