**PROJECT 4 REPORT**

*Overview*

I chose to gather stock price data from Yahoo Finance for every trading day for the past two years, including daily volume and open, high, close, low and adjusted close prices. Specifically, Microsoft (MSFT) was used but any similar stock data would function equally. This project still has a lot of room for expansion and could be used to calculate other more pertinent metrics from the price data aside from the mean, median, maximum and minimum values calculated for each field. As of now, the project performs the required actions for the project, but other functions could easily be added for expansion. The calculations are optionally written to a text file that summarizes the results.

*Expected I/O*

A CSV file from Yahoo Finance is used as input for the program and read using the csv library, both with the reader() and DictReader() methods, and stored in appropriate lists for reuse. User choices are also collected as input in the console with the input() function to specify user-made decisions.

Output from this project includes information printed in the IPython console in Spyder as well as the output.txt file written and appended to by the program.

*Use of Concepts*

The csv module was used to read from the CSV files downloaded from Yahoo Finance easily and in a helpful dictionary-based format for easier reference throughout the program. File input and output methods were used in writing, reading, and appending to files for output. The matplot library was used to create a line plot of the adjusted close prices of the stock data, but I found that formatting dates on the x-axis to be tricky. I was not able to complete the other graphs, so I completed option A for the program. The user can specify a key associated with the fieldnames created from the DictReader class method to append a specific variable’s mean, median, max and min calculations to the output file.

*Limitations and Future Expansions*

The main bottleneck on the depth of this program was time. As such, it generates a very shallow analysis of the stock data. The line plot generated with the matplot library does not show clear markings for the date on the x-axis. These were more difficult to format given that dates would have to be formatted. I would have liked this project to be as in-depth as the others that I have completed for this class, but I do know where I will expand when I have the time.

Primarily, I would want to find different metrics to use to analyze the stock data in question. I would use other index data in order to calculate values such as to compare the performance of a given stock with the market. I could also use the data to generate moving average curves with different periods to superimpose over the stock data. Of course, this would all require better working knowledge of the inner-workings of the matplot import.

*Reflection*

For Part 7, I first attempted to complete the graphs, but I found the learning curve for the matplot library to be quite steep. I was attempting to do external research, and most solutions to the problems I was searching for incorporated other modules such as numpy and pandas, yet these were not permitted in the assignment. I underestimated the time that it would take to understand the documentation after judging how long it took me to research other concepts for this class. It is definitely a tool that I will learn to utilize and appreciate but considering the circumstances of this semester I was not able to do that at the time. The last few days of break, my laptop screen stopped working and I was not able to access anything on my laptop for about four days until I could get my computer to display with an external monitor, which set me back quite a bit.

Aside from matplot, the most challenging topics that we studied in this class were, well, classes. When I signed up for this class, I had never heard of the phrase “object-oriented programming”. Halfway through the class I still didn’t really understand what an object even was. Although classes were the most difficult thing to wrap my head around, I also found them to be the most interesting. The idea of using classes in Python came to my attention when I was working on Project 2, and I wished that I had a way to reduce repetitive code from each “location” function in the program. We hadn’t gotten to classes yet, but I had seen very basic examples of them from YouTube tutorials and in the Python mini courses I’ve been taking with Linkedin Learning. Once I had a working knowledge of how to use custom-made classes, other puzzling questions I had suddenly began to make sense, like why some “functions” go at the end of a value (methods) and why others are built into Python as standard functions. Understanding that made me realize the flexibility available to programmers using OOP and has made me interested in learning more complex coding languages like C++ after this class.

The most interesting project was probably the adventure game. It was enjoyable writing the code and script for all of it, and it was really good for practice in writing user-friendly code that followed directions on a basic level. It also helped me come up with inventive ways to loop certain parts of the program and return certain values while overcoming barriers presented by things like function scope. When I’m writing in Python, I tend to write lots of functions, which complicates the flow of data between different parts of the program. Project 2 was where I started to really get a good grasp of functions and writing detailed code in general.

Overall, I have loved the class and it has really made me think about learning to code in the future. I would recommend it to anyone remotely interested in computer science who wants to get a taste of coding and the option to challenge themselves while doing it whenever they want to and are able.

*Program Usage*

Graphical user interface, text

Description automatically generated

A picture containing text

Description automatically generated

*Figure 1: Displaying the assigned row, column, and element from Question 3 to the user in the console. There were about 500 rows so this was rather bulky*

Text

Description automatically generatedText

Description automatically generated

*Figure 2: Displaying initial calculations of mean, median, max and min for three variables, giving the option to append the full data frame to the output.txt file, and giving option to repeatedly append calculations to the file for other variables*

Graphical user interface, chart

Description automatically generated

*Figure 3: Display of matplotlib.pyplot graph of adjusted close price over the course of two years for MSFT stock. X-axis was difficult to format properly but could be done with further work.*

*Copy of .py File*

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Ian Sharff

COSC 010

Project 4 (Final)

"""

import csv

import os

import matplotlib.pyplot as mpp

import datetime

def main():

""" Run various functions required by the project stipulations"""

infilename = 'SharffIan.csv' # Name of CSV file

outfilename = 'output.txt' # Default output file name

print(f"Current output file name is: {outfilename}")

print("Change output file name?") # Option to change output file name

if yesorno():

outfilename = input("Enter new output file name: ") + ".txt"

print(f"Output file updated to: {outfilename}")

array = read\_file(infilename) # List of lists of elements in rows

dict\_array = read\_dict\_file(infilename) # List of dictionaries

# Fieldname keys map to values, defaults to first row

print\_data(array) # Option to print raw data

question3(array) # Option to display the test index searches in Q3

questions4and5(dict\_array, outfilename) # Prints specified variable calcs

# Option to append fill data frame and run the character count functions

print(f"Append full data frame to {outfilename}?")

if yesorno():

line\_lengths = AppendFunction(outfilename, array)

display\_lengths(line\_lengths)

# Option to run the line plot function, work in progress

print("Display graph of two-year adjusted close price?")

if yesorno():

MakeGraphs(dict\_array)

# Option to append calculations to the existing output .txt file

UserChoice(dict\_array, outfilename)

remove\_file(outfilename)

def yesorno():

"""Collect input, if and only if 'yes', returns True"""

if input('-->').lower() == 'yes':

return True

else:

return False

def read\_file(f):

"""Read file to 2D list with csv import"""

with open(f, newline = '') as csvfile:

reader = csv.reader(csvfile)

rows = [r for r in reader]

return rows

def read\_dict\_file(f):

"""Read file to list of dictionaries with csv import DictReader class method"""

with open(f, newline = '') as csvfile:

d\_reader = csv.DictReader(csvfile)

rows = [r for r in d\_reader]

return rows

def print\_data(arr):

"""Print raw data from CSV by row"""

print("Print raw data?")

if yesorno():

for row in arr:

print(','.join(row))

def question3(arr):

"""Display searched values according to question 3"""

print("Display test elements?")

if yesorno():

print("Third row: ")

print(\*arr[2], sep = ',')

print("First column: ")

for row in arr:

print(row[0])

print("Row 0, column 4: ")

print(arr[0][4])

def questions4and5(d\_arr, filename):

"""Print and write calculations for high prices, low prices, and volume to .txt file"""

print("Printing and writing calculations for high prices, low prices, and volumes...")

# List comprehensions converting stings to floats

high\_prices = [float(r['High']) for r in d\_arr]

low\_prices = [float(r['Low']) for r in d\_arr]

volumes = [float(r['Volume']) for r in d\_arr]

# Tuples associating variable names with float lists

fields = [('High', high\_prices), ('Low', low\_prices), ('Volume', volumes)]

# Writes calculations to files using display\_calc functions

with open(filename, 'w') as textfile:

for variable, nums in fields:

calculations = display\_calc(variable, nums)

print(calculations)

textfile.write(calculations)

print(f"File {filename} created.")

def AppendFunction(filename, dataframe):

"""Append raw data frame to output file"""

# First writes lines to the text file

with open(filename, 'a') as appfile:

for item in dataframe:

string\_line = ','.join(item) + '\n'

appfile.write(string\_line)

print(f"Dataframe appended to file '{filename}'")

# Then reads file and adds character lengths and total characters to list

with open(filename, 'r') as readfile:

char\_count = [len(line) for line in readfile]

total\_char = sum(char\_count)

char\_count.append(total\_char)

# Return list to be displayed by function in main()

return char\_count

def UserChoice(dataframe, filename):

"""Collect user input to determine variable calculations to append to outfile"""

if input("Append other calculations?\n-->").lower() != 'yes':

return None

else:

with open(filename, 'a') as f:

variables = list(dataframe[0].keys())

variables.remove('Date')

appending = True

while appending:

print("Enter variable name that you would calculations for")

print("Available variables:")

print(\*variables, sep = ', ')

choice = input('-->').title()

if choice in variables:

workinglist = [float(row[choice]) for row in dataframe]

calculations = display\_calc(choice, workinglist)

print(calculations)

f.write(calculations)

print(f"{choice} calculations appended to {filename}.")

else:

print("Invalid entry")

if input("Keep appending?\n-->").lower() != 'yes':

appending = False

def MakeGraphs(dataframe):

"""Plot adjusted close price against date for two-year MSFT stock data"""

lineplot\_name = "AdjustedClosePrices\_LinePlot.jpg" # Name of line plot

title = "Two-year MSFT Adjusted Close Price" # Title for plot

# Date values converted to datetime.datetime objects to function with mpp

xvals = [datetime.datetime.strptime(row['Date'], '%Y-%m-%d') for row in dataframe]

# Adj Close values converted to float to be plotted on y-axis

yvals = [float(row['Adj Close']) for row in dataframe]

# Plot method

mpp.plot(xvals, yvals)

# Adds title and axis labels

mpp.title(title)

mpp.xlabel("Date")

mpp.ylabel("Adjusted Close Price USD")

# Save to .jpg file in folder

mpp.savefig(lineplot\_name)

# Show plot in IPython

mpp.show()

def calculate\_mean(col):

"""Pass list parameter and return mean value"""

mean = sum(col)/len(col)

return mean

def calculate\_median(col):

"""Pass list parameter and return median value"""

m = len(col) // 2

if len(col) % 2 == 0:

median = (col[m] + col[m + 1]) / 2

else:

median = col[m]

return median

def calculate\_max(col):

"""Pass list parameter and return max value"""

# List is alread sorted by the display\_calc() function, no need to call max()

maximum = col[-1]

return maximum

def calculate\_min(col):

"""Pass list parameter and return min value"""

# List is already sorted

minimum = col[0]

return minimum

def round\_two(num):

"""Pass float and return number rounded to two decimal places"""

return round(num, 2)

def display\_calc(name, li):

"""Pass and sort list parameter, format and return display string to be written to file"""

li.sort()

# Calc functions called with list parameter

values = (calculate\_mean(li), calculate\_median(li), calculate\_max(li), calculate\_min(li))

# Calls rounding function for each calculated value and returns list

rounded\_values = list(map(round\_two, values))

# Formats floats as string, ensuring that two decimals are displayed

display = ("{} calculations:\n"

"Mean: {:.2f}\n"

"Median: {:.2f}\n"

"Maximum: {:.2f}\n"

"Minimum: {:.2f}\n\n")

# format() string method called and name parameter indicates variable

formatted\_display = display.format(name, \*rounded\_values)

return formatted\_display

def display\_lengths(lengths):

"""Display line length data from Question 5"""

print("List of line lengths:\n")

for num in lengths:

print(num, end = ',')

else: print('\n')

print(f"Calculated total characters = {lengths[-1]}")

def remove\_file(filename):

"""Option to delete the file produced by the program"""

print(f"Delete file '{filename}'?")

if yesorno():

os.remove(filename)

print("File deleted")

print("Thank you, signing off...")

if \_\_name\_\_ == '\_\_main\_\_': main()