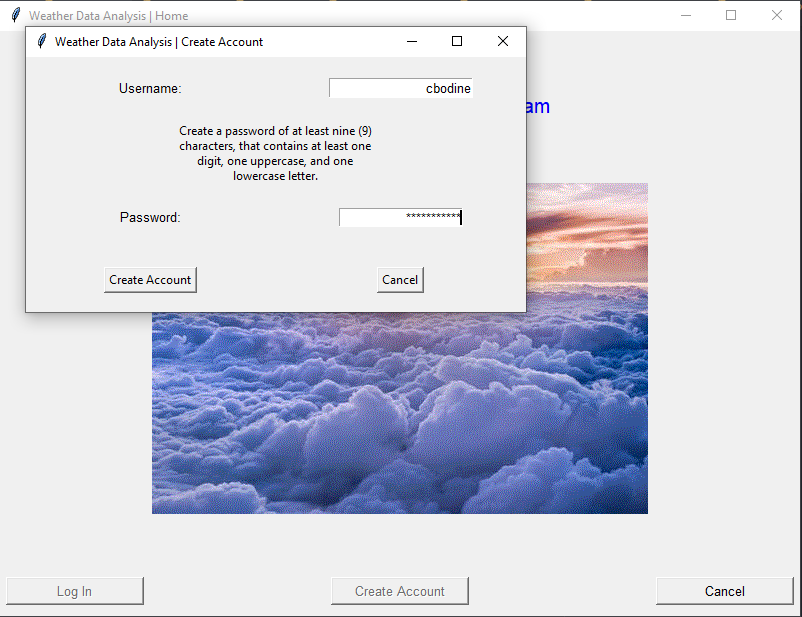
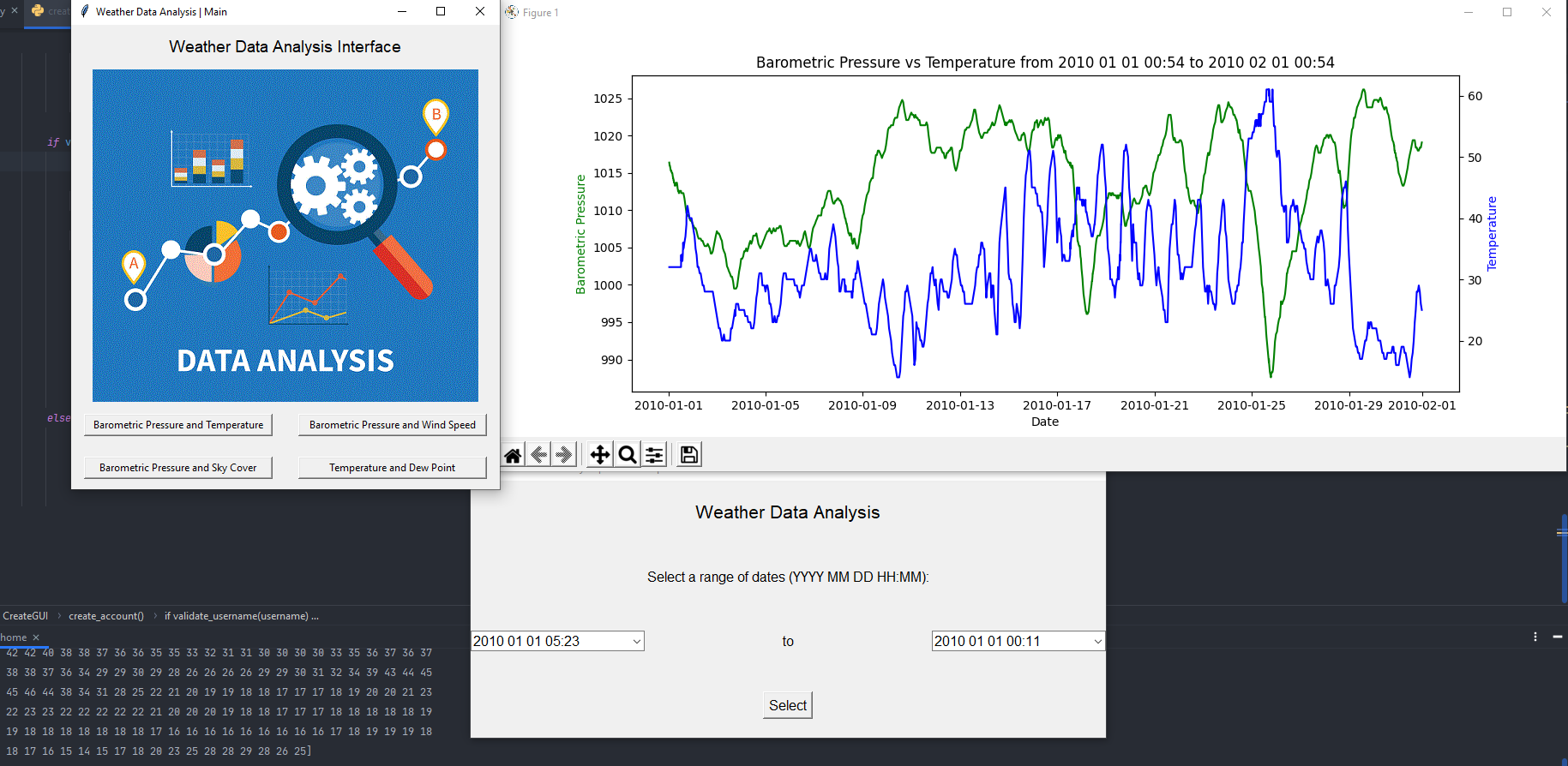
**Weather Data Analysis Project**

This project is a program written in the Python programming language that allows a user to access a data analysis system. The data analysis system extracts data from a data library file and allows the user to select from a range of data sets and times to be plotted using a graphical user interface (GUI), created using the tkinter and matplotlib libraries. It also allows users to create and permanently store accounts, using SHA256 encryption for passwords, provided by the hashlib library.

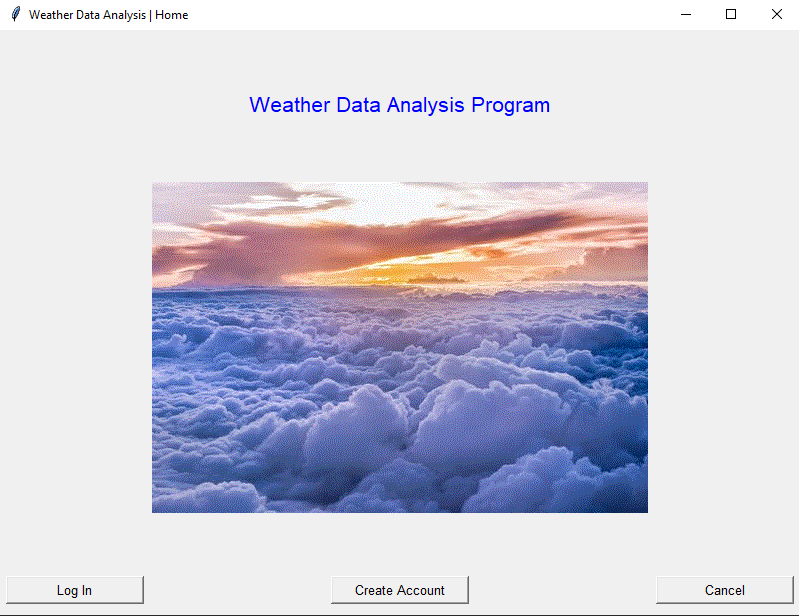




|  |  |
| --- | --- |
| **Cole Bodine** | **CSE-222-101 Fall 2020** |

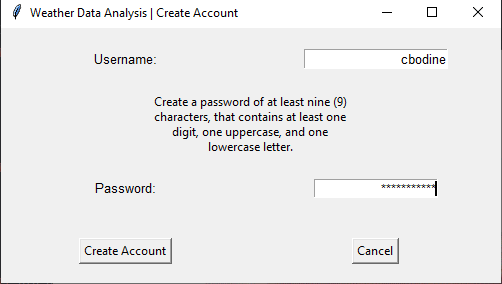
**Project Milestone #1 – Main GUI, Account Creation, and Password Handling**

When the program is initially started, the user will be taken to the screen shown in Figure 1. The screen contains user login, account creation, and exit buttons.



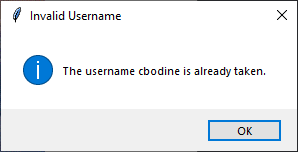
*Figure 1: Home GUI*

If the user does not have an account, they can create one by clicking the “Create Account” button shown in Figure 1. Upon clicking the button, they will be taken to a new window, where they can enter their username and password, as shown in Figure 2.

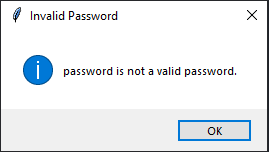


*Figure 2: Account Creation Window*

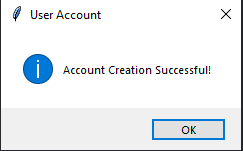
If the user enters a username that has already been taken, they will receive an error telling them that the username has been taken (Figure 3). Likewise, if they enter an invalid password, they will receive a prompt informing them that the password is not valid (Figure 4). Finally, if the user enters a valid username and password, their account will be created and stored in a file (Figure 5).



*Figure 3: Invalid Username*

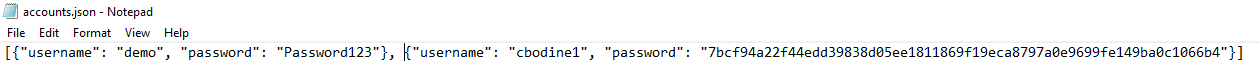


*Figure 4: Invalid Password*



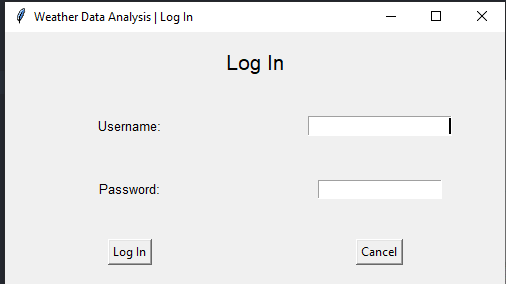
*Figure 5: Successful Account Creation*

When an account is successfully created, it is appended to a list of dictionaries in a file called “accounts.json” (Figure 6) that resides in the same location as the Python program. Before being stored, the password is passed through a hashing function provided by the hashlib library, which encrypts the password.



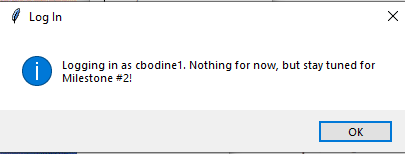
*Figure 6: Username and Password as stored in the file.*

After the user’s account is created, the account creation button is disabled, and the login button is re-enabled. If they could not already, the user may now log in by cligking the “Log In” button from Figure 1. This will take them to the screen shown below, in Figure 7.



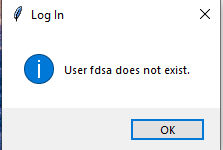
*Figure 7: Login Window*

Like with the account creation window, activating the login window deactivates the “Log In” and “Create Account” buttons from the main window. Here, the user will be prompted to enter a valid username and password. The program determines which username and password is valid by opening the “accounts.json” file and parsing through the user\_accounts list. If the program finds a dictionary that contains a username and password matching what the user entered, that user will be logged in to the system (Figure 8). This doesn’t do anything besides display a message right now.

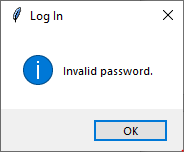


*Figure 8: Successful Login*

If they provide an invalid username (Figure 9) or password (Figure 10), they will receive a prompt informing them.



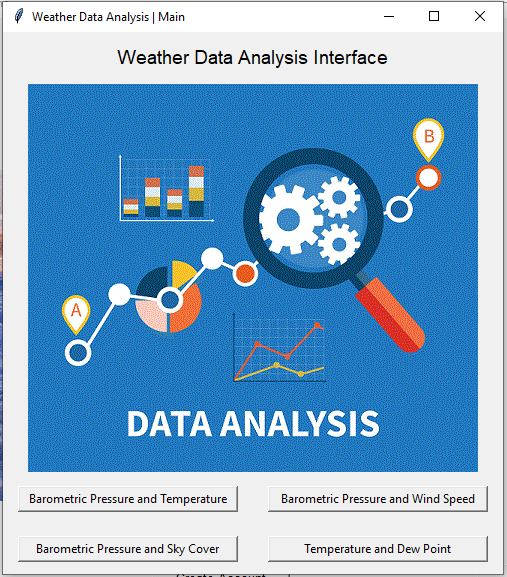
*Figure 9: Invalid User*



*Figure 10: Invalid Password*

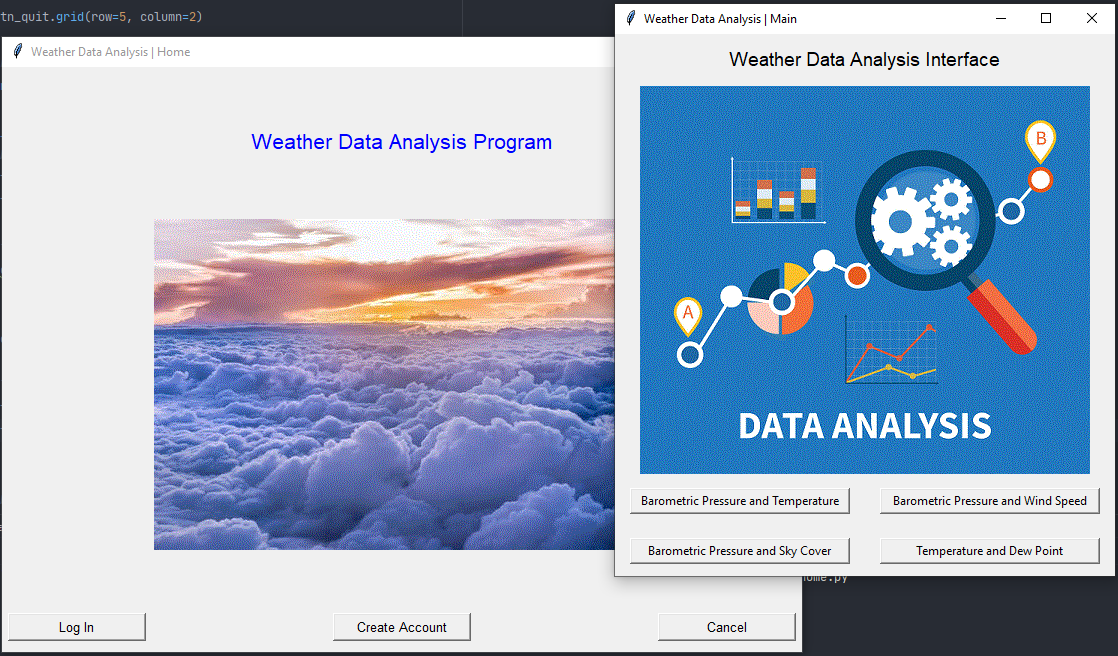
**Project Milestone #2**

The program now has complete login functionality and a main GUI that will allow the user to select which data they would like to view. The message box from Figure 8 has been removed and logging in will now take the user to the main GUI, as shown in Figure 11.



*Figure 11: Main GUI.*

Most of the other work done to the project is simple maintenance and changing the code around, as I accidentally went too far in Milestone #1 by completing the login process that was meant to be saved for Milestone #2. For example, I renamed the original main.py to home.py and changed any code with the word main to have the word home, to make it more easily differentiable from the new main.py. I opted for creating an entirely new window rather than morphing the original because I want the user to still have access to the home window in case they want to log out of the program and let someone else use it without having to close the entire program. Now, all they will have to do is close the main window and they can start the program anew by logging in as someone else (Figure 12).



*Figure 12: Allow the user to log in as someone else after closing main interface.*

**Project Milestone #3**

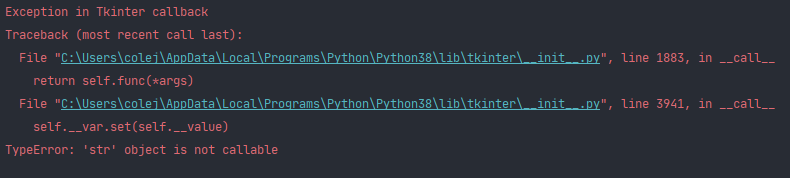
Milestone #3 has proven to be more difficult than I expected. The basic idea of what I was trying to do was for the user to be able to click on one of the options on the main GUI, which would then take them to a window that would provide them with two dropdown menus, allowing them to pick a range of dates like the diagram shown in Figure 13.

Graphical user interface, text, application

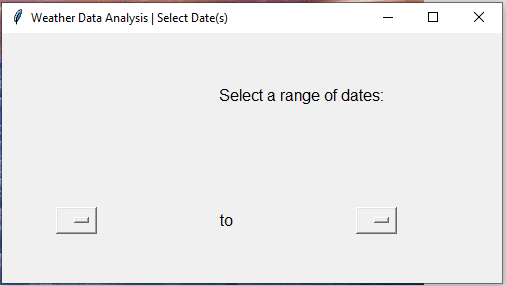
Description automatically generated

*Figure 13: Mockup design of Window*

Unfortunately, I ran into some issues that did not allow me to complete the Milestone completely. The main problem I ran into with the code was that my variable for the list of dates was not being passed correctly to the tkinter OptionMenu (the dropdown menu). This left me with the error code seen in Figure 14 and the GUI shown in Figure 15.



*Figure 14: Error code*

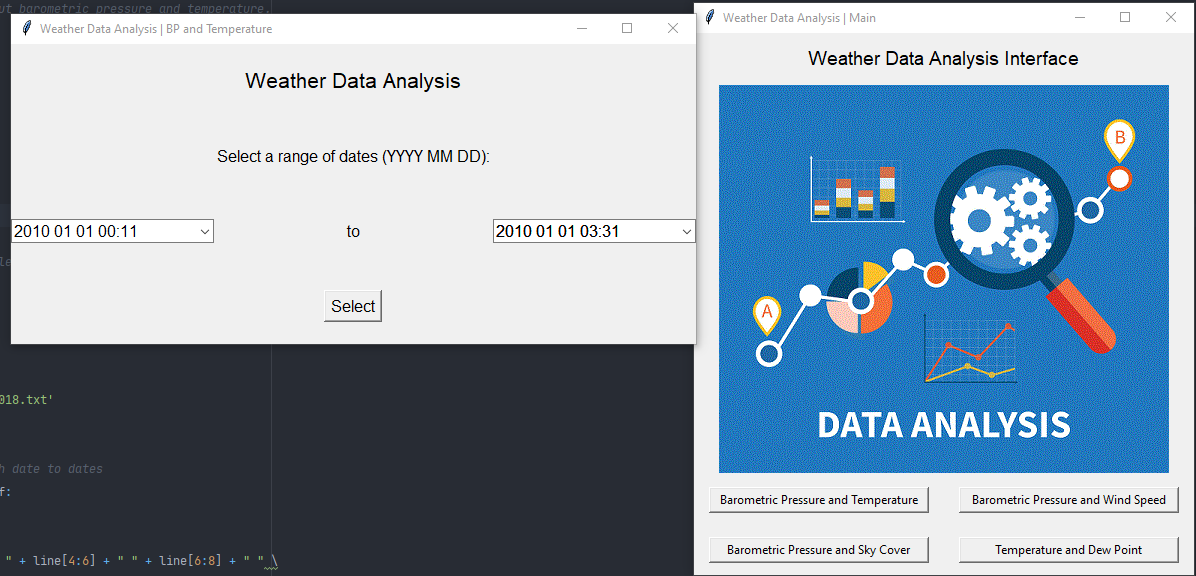


*Figure 15: The broken GUI*

The callback error comes from the tkinter library’s \_\_init()\_\_ function. For whatever reason, my list of dates looks like it is being passed as a string, rather than a list. My idea to fix this problem between now and Milestone #4 is to break this problem down into a smaller size by isolating it in its own environment (to give me less things to work with). Once I figure it out in a smaller setting, I will adapt that solution to the larger setting that is this project.

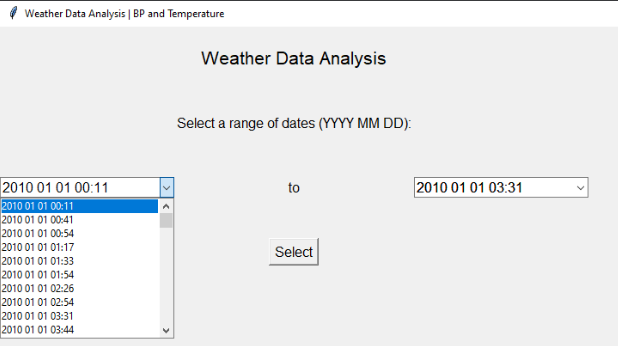
**Project Milestone #4**

For Project Milestone #4, the program works almost as intended, with only a couple of areas left to really work on. We will start with where we left off in Milestone #3.



*Figure 16: Date Select GUI*

As you can see in Figure 16, while not exactly pleasing to the eye as it could be, the GUI that allows you to select from the range of dates has been completed. I was able to fix the problem I had by putting the specific area of code that draws the GUI into a smaller environment and working from there. The first problem I discovered was that I was reading the dates into a tuple, rather than a list, which made it impossible for my dropdown menu to understand the data that was coming in. Once that was fixed, I noticed another problem: OptionLists are remarkably ineffective for large sets of choices, because they lack a scroll bar. I researched for a better option and found ttk’s Combobox, which is essentially just a dropdown menu with a scroll bar that is also a little nicer to look at. A demonstration can be seen in Figure 17:

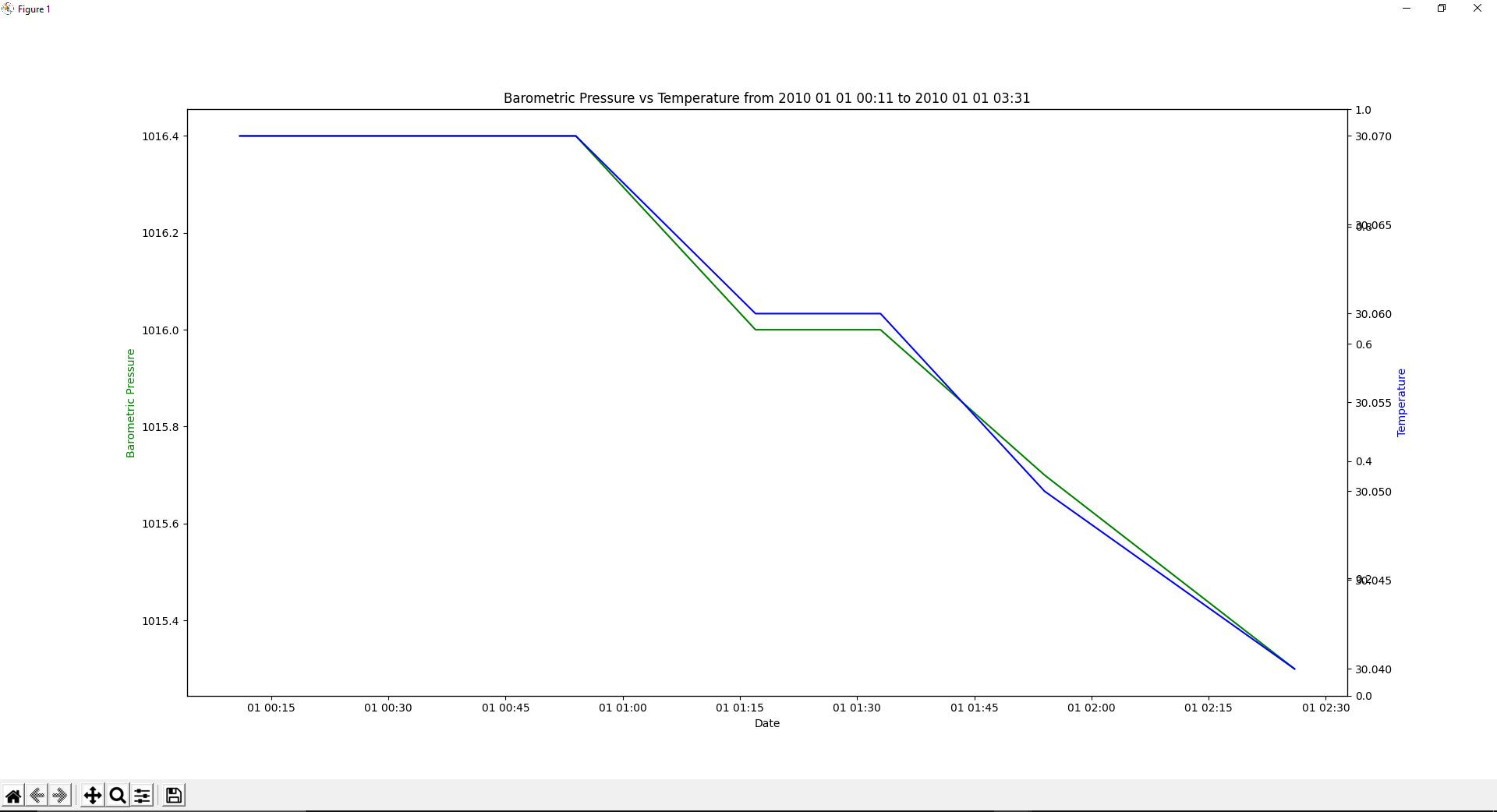


*Figure 17: Combobox menu*

As one could guess, this makes it a lot easier to select a date, especially when looking for one in the latter end of the list.

The other problem I had early on was separating the columns. Originally, I tried to grab the columns by width. For example, if a column were 10 spaces wide, I would grab 10 spaces within that line. I soon found out, however, that the columns would often be off by 1 or 2, and sometimes even 3, so this didn’t end up working. What I did to solve this problem was I looked back to when we counted words in a string. I figured if I treated this like grabbing the nth word in a line, I could translate that to the graph. Sure enough, it worked successfully, and I was able to grab whatever piece of data I wanted to plot on the graph! The final challenge was columns that had asterisks in them. For this, I took the easy approach of simply omitting any rows that had a column with asterisks in them. For example, even if temperature had a number and barometric pressure had an asterisk, I would omit the entire row. I might change this later, but it is working very well now and since it is such a rare occurrence anyway I don’t see where it could be a problem.

When the user hits the select button, they will be provided with the graph. The graph is made with matplotlib and pyplot. Figure 18 shows the graph made with the range of dates specified by the user in Figure 16:



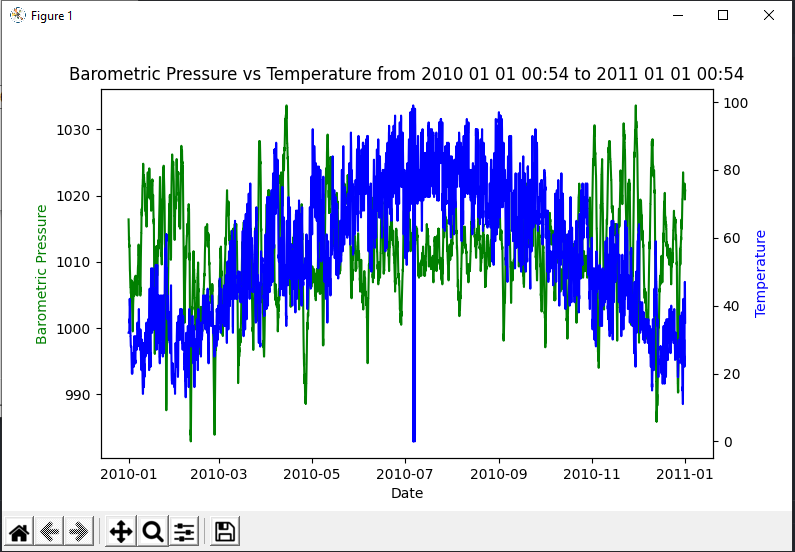
*Figure 18: The first Graph*

The graph is labeled with the appropriate axes: In this case, there are 2 y-axes, one for barometric pressure and one for temperature, color coded to match the line that represents them, and the x axis is for the date, from earliest to latest. The graph is also titled with the numbers that are being measured and the date/time range that is being used.

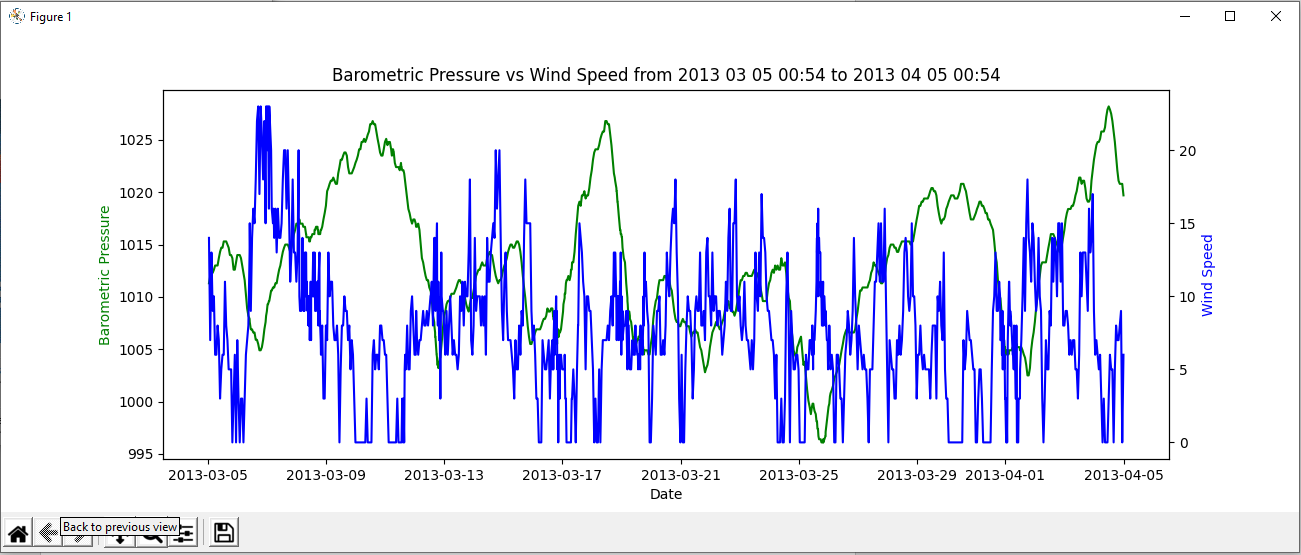
For Milestone #5, I will be working on the temperature and dew point part of the assignment, since they are listed as separate, but in the data dictionary they are considered the same thing. I will also be working on making my code more lightweight and less redundant. You can see in the current state of code that I have an entire file for each of the four choices on the main GUI. This, to me, seems ridiculous: I should be able to use conditional statements to see which button the user picks, and morph the GUI to fit that. Unfortunately, I have not yet figured out how to do that, so that will be my next step after getting temperature and dew point.

**Milestone #5**

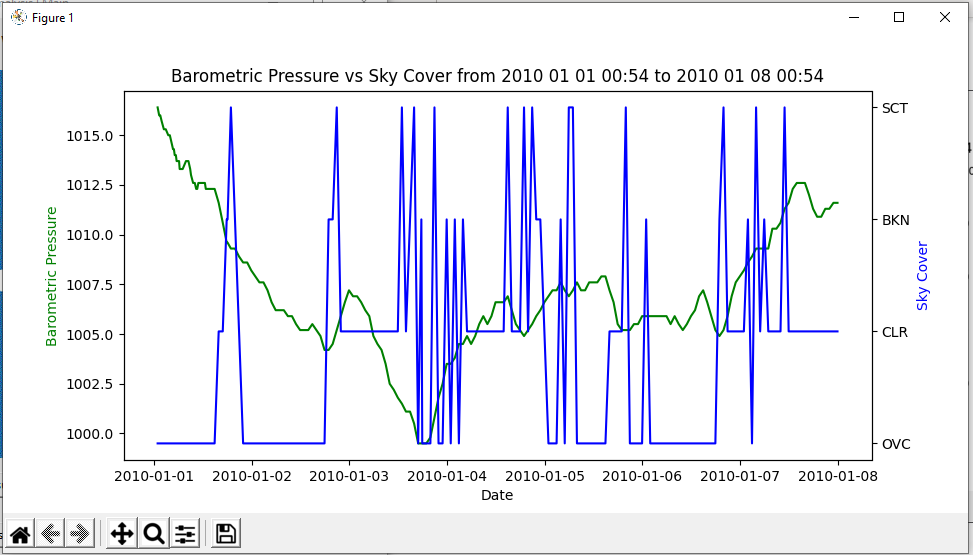
For Milestone #5, I took the new data set that was provided on Blackboard and adjusted my code accordingly. Now, all 4 plots work as intended. I also went back to my old algorithm from Milestone #3, gathering the data via the index range of the line string. There is not a whole lot to say here, so I will show it in screen captures:



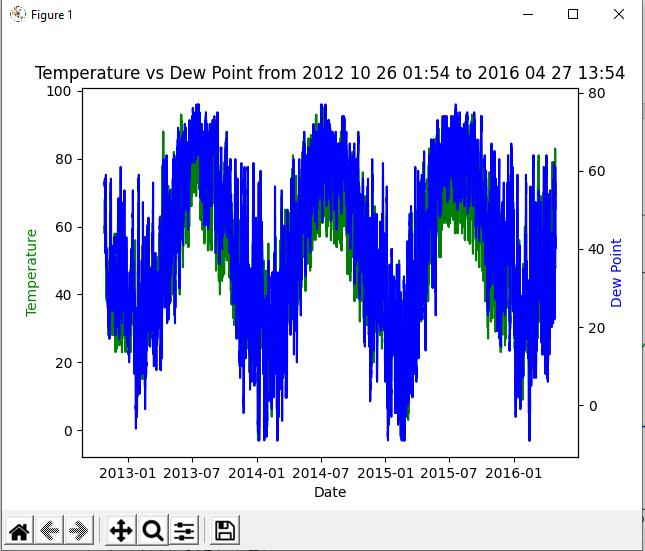
*Figure 19: A demonstration of Barometric Pressure vs Temperature over the course of a year.*



*Figure 20: A demonstration of Barometric Pressure vs Wind Speed over the course of a month.*



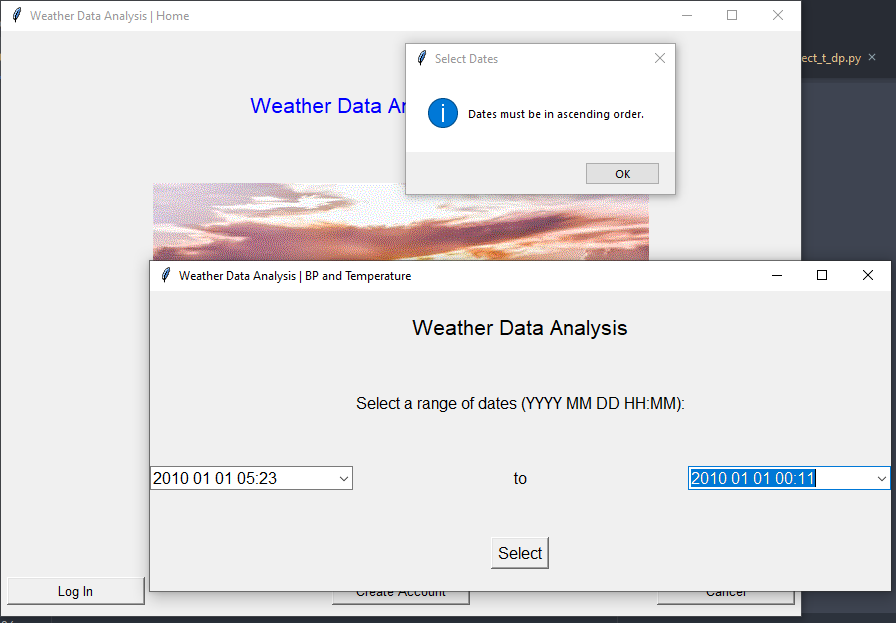
*Figure 21: A demonstration of Barometric Pressure vs Sky Cover over the course of a week.*



*Figure 22: A demonstration of Temperature vs Dew Point over a randomly chosen span of time.*

**Milestone #6**

For Milestone #6, I did some small touchups. I added a cover page to the design document and made sure to center all the images to give it a more cohesive, aesthetically pleasing design. For the actual program, I put a conditional statement check before the plotting of the graph to make sure that the dates are in ascending order. If the user tries to pick, for example, January 1st, 2010 at 5 AM to January 1st, 2010 at 12 AM, they will be provided with an error message like that shown in Figure 23:



*Figure 23: What happens when a user incorrectly selects a date range*.

Overall, I think this project was a very insightful experience, albeit sometimes frustrating, and incredibly fun as well. I learned what it is like to work in a real SCRUM environment, and I was surprised how easy it was to pick up the Python programming language and its extensive libraries. Because of being forced to work remotely, I was able to learn how to be a more independent developer and how to solve problems myself through looking up documentation.

Some challenges that I faced included working with a new programing language, troubleshooting new error messages, and learning new libraries, all of which were documented in the making of this design document. A problem that stuck with me was when the columns in the text file were misaligned. When I finally found a solution to my problem and fixed it, we were given a new file with the columns lined up properly, so I had to go back to an older solution that I tried to use the first time! Another challenge I faced for about a week was learning how to properly use OptionMenu for dropdown menus. I eventually found out that I was storing my values in a tuple, which was not compatible with OptionMenus, but in the process of figuring this out I learned about ttk’s ComboBoxes, so it was a sort of silver lining that I made the initial mistake, since the scrollbar that comes with ComboBoxes makes the dropdown menus much easier to navigate. In general, I was given more than enough time to troubleshoot these problems, plus some to spare so I could experiment with my own solutions, such as using a JSON file instead of a regular text file for storing passwords, which I think worked out well in the end, and I did not run into too many problems.

The design of this program could use more work, as an engineer’s job is never truly done. If I were given more time, I would likely make the program more modular. I mentioned earlier that I wanted to make it more compact by merging the four decision files into one, and unfortunately never had the time to do so because of other classes and the semester being so short. In the end, the program still works as intended, so I think I can call this one a success.

**Current (and final) state of code:**

# home.py

# This file stores the home window for the weather data analysis project.

from tkinter import \*

import tkinter as tk

import create

import login

import main

# home window

class HomeGUI:

"""A class holding the home window."""

def \_\_init\_\_(self):

"""Initialize the home window."""

# Create the home window.

self.win\_home = tk.Tk()

self.win\_home.title("Weather Data Analysis | Home")

self.win\_home.minsize(width=450, height=300)

# Configure columns

self.win\_home.columnconfigure(0, minsize=150)

self.win\_home.columnconfigure(1, minsize=150)

self.win\_home.columnconfigure(2, minsize=150)

# Configure Rows

self.win\_home.rowconfigure(0, minsize=50)

self.win\_home.rowconfigure(1, minsize=50)

self.win\_home.rowconfigure(2, minsize=50)

self.win\_home.rowconfigure(3, minsize=50)

self.win\_home.rowconfigure(4, minsize=50)

self.win\_home.rowconfigure(5, minsize=50)

# Create label widget.

self.lbl\_header = tk.Label(text="Weather Data Analysis Program",

font=("Helvetica", 16),

fg="blue")

self.lbl\_header.grid(row=1, column=0, columnspan=3)

# Create the picture widget.

photo = PhotoImage(file="weather.gif")

self.labelGIF = tk.Label(image=photo)

self.labelGIF.image = photo

self.labelGIF.grid(row=3, column=1)

# Create the button widgets.

self.btn\_login = tk.Button(text="Log In",

font=("Helvetica", 10),

width=16,

command=self.log\_in)

self.btn\_create\_acct = tk.Button(text="Create Account",

font=("Helvetica", 10),

width=16,

command=self.create\_account)

self.btn\_quit = tk.Button(text="Cancel",

font=("Helvetica", 10),

width=16,

command=self.win\_home.destroy)

self.btn\_login.grid(row=5, column=0)

self.btn\_create\_acct.grid(row=5, column=1)

self.btn\_quit.grid(row=5, column=2)

# Enter home tkinter loop

tk.mainloop()

def create\_account(self):

# Disable the buttons

self.btn\_create\_acct.config(state=DISABLED)

self.btn\_login.config(state=DISABLED)

# Create an account creation GUI

self.acct\_GUI = create.CreateGUI()

# Wait for the window to be destroyed.

self.acct\_GUI.win\_create.wait\_window()

# Enable login button again.

self.btn\_login.config(state=NORMAL)

self.btn\_create\_acct.config(state=NORMAL)

def log\_in(self):

# Disable the buttons

self.btn\_create\_acct.config(state=DISABLED)

self.btn\_login.config(state=DISABLED)

# Create an account creation GUI

self.login\_GUI = login.LoginGUI()

# Wait for the windows to be destroyed.

self.login\_GUI.win\_login.wait\_window()

# Enable login and create buttons again.

self.btn\_login.config(state=NORMAL)

self.btn\_create\_acct.config(state=NORMAL)

dataProgram = HomeGUI()

# create.py

# This file holds the account creation window and logic.

from tkinter import \*

from tkinter import messagebox

import tkinter as tk

import json

import os

import hashlib

class CreateGUI:

"""A class containing the create account GUI"""

def \_\_init\_\_(self):

"""Initialize the AccountGUI class."""

# Create window

self.win\_create = tk.Tk()

self.win\_create.title("Weather Data Analysis | Create Account")

self.win\_create.minsize(width=500, height=250)

# Configure columns

self.win\_create.columnconfigure(0, minsize=250)

self.win\_create.columnconfigure(1, minsize=250)

# Configure Rows

self.win\_create.rowconfigure(0, minsize=62.5)

self.win\_create.rowconfigure(1, minsize=62.5)

self.win\_create.rowconfigure(2, minsize=62.5)

self.win\_create.rowconfigure(3, minsize=62.5)

# Create username widgets.

self.win\_create.lbl\_username = tk.Label(self.win\_create,

text="Username:",

font=("Helvetica", 10))

self.win\_create.lbl\_username.grid(row=0, column=0)

self.win\_create.entry\_username = tk.Entry(self.win\_create,

justify="right",

font=("Helvetica", 10))

self.win\_create.entry\_username.grid(row=0, column=1)

self.win\_create.entry\_username.focus\_force()

# Create the password widgets.

self.win\_create.lbl\_password\_guide = tk.Label(self.win\_create,

text="Create a password "

"of at least nine "

"(9) characters, "

"that contains at "

"least one digit, "

"one uppercase, "

"and one lowercase "

"letter.",

wraplength=200)

self.win\_create.lbl\_password\_guide.grid(row=1, column=0, columnspan=2)

self.win\_create.lbl\_password = tk.Label(self.win\_create,

text="Password:",

font=("Helvetica", 10))

self.win\_create.lbl\_password.grid(row=2, column=0)

self.win\_create.entry\_password = tk.Entry(self.win\_create,

width=20,

justify="right",

show="\*")

self.win\_create.entry\_password.grid(row=2, column=1)

self.win\_create.btn\_create = tk.Button(self.win\_create,

text="Create Account",

command=self.create\_account)

self.win\_create.btn\_create.grid(row=3, column=0)

self.win\_create.btn\_cancel = tk.Button(self.win\_create,

text="Cancel",

command=self.win\_create.destroy)

self.win\_create.btn\_cancel.grid(row=3, column=1)

# Lift to top

self.win\_create.lift()

def create\_account(self):

"""Create a user account."""

password = self.win\_create.entry\_password.get()

username = self.win\_create.entry\_username.get()

# If a file does not exist for user accounts, create one with

# placeholder data.

if not os.path.isfile("accounts.json"):

acct\_file = open("accounts.json", "w")

json.dump([{"username": "demo", "password": "Password123"}],

acct\_file)

acct\_file.close()

try:

acct\_file = open("accounts.json", "r")

user\_accounts = json.load(acct\_file)

except FileNotFoundError:

print(f"File {acct\_file} does not exist.")

def validate\_username(username):

"""Check to see if the username is taken."""

if not any(user['username'] == username.lower() for

user in user\_accounts):

return True

else:

tk.messagebox.showinfo("Invalid Username",

f"The username {username} is already "

f"taken.")

def validate\_password(password):

"""Validate user's password."""

long\_enough = False

has\_lower = False

has\_upper = False

has\_digit = False

if len(password) >= 9:

long\_enough = True

for ch in password:

if ch.islower():

has\_lower = True

if ch.isupper():

has\_upper = True

if ch.isdigit():

has\_digit = True

if long\_enough and has\_lower and has\_upper and has\_digit:

return True

else:

tk.messagebox.showinfo("Invalid Password", f"{password} is "

f"not a valid "

f"password.")

if validate\_username(username) and validate\_password(password):

hashed\_password = hashlib.sha256(str.encode(password)).hexdigest()

user\_accounts.append({'username': username.lower(),

'password': hashed\_password})

tk.messagebox.showinfo("User Account", "Account Creation "

"Successful!")

acct\_file.close()

acct\_file = open("accounts.json", 'w')

json.dump(user\_accounts, acct\_file)

acct\_file.close()

self.win\_create.entry\_username.delete(0, END)

self.win\_create.entry\_password.delete(0, END)

self.win\_create.destroy()

else:

print("Couldn't create account. Please try again.")

self.win\_create.entry\_username.delete(0, END)

self.win\_create.entry\_password.delete(0, END)

# login.py

# This login window and logic.

from tkinter import \*

import tkinter as tk

from tkinter import messagebox

import json

import hashlib

import main

class LoginGUI:

"""A class holding the login GUI."""

def \_\_init\_\_(self):

"""Initialize the login GUI."""

# Create window

self.win\_login = tk.Tk()

self.win\_login.title("Weather Data Analysis | Log In")

self.win\_login.minsize(width=500, height=250)

# Configure columns

self.win\_login.columnconfigure(0, minsize=250)

self.win\_login.columnconfigure(1, minsize=250)

# Configure Rows

self.win\_login.rowconfigure(0, minsize=62.5)

self.win\_login.rowconfigure(1, minsize=62.5)

self.win\_login.rowconfigure(2, minsize=62.5)

self.win\_login.rowconfigure(3, minsize=62.5)

# Create Login Label

self.win\_login.lbl\_title = tk.Label(self.win\_login,

text="Log In",

font=("Arial", 16))

self.win\_login.lbl\_title.grid(row=0, column=0, columnspan=2)

# Create username widgets.

self.win\_login.lbl\_username = tk.Label(self.win\_login,

text="Username:",

font=("Helvetica", 10))

self.win\_login.lbl\_username.grid(row=1, column=0)

self.win\_login.entry\_username = tk.Entry(self.win\_login,

justify="right",

font=("Helvetica", 10))

self.win\_login.entry\_username.grid(row=1, column=1)

self.win\_login.entry\_username.focus\_force()

# Create the password widgets.

self.win\_login.lbl\_password = tk.Label(self.win\_login,

text="Password:",

font=("Helvetica", 10))

self.win\_login.lbl\_password.grid(row=2, column=0)

self.win\_login.entry\_password = tk.Entry(self.win\_login,

width=20,

justify="right",

show="\*")

self.win\_login.entry\_password.grid(row=2, column=1)

self.win\_login.btn\_create = tk.Button(self.win\_login,

text="Log In",

command=self.log\_in)

self.win\_login.btn\_create.grid(row=3, column=0)

self.win\_login.btn\_cancel = tk.Button(self.win\_login,

text="Cancel",

command=self.win\_login.destroy)

self.win\_login.btn\_cancel.grid(row=3, column=1)

# Lift to top

self.win\_login.lift()

def log\_in(self):

"""Log the user in."""

username = self.win\_login.entry\_username.get()

password = self.win\_login.entry\_password.get()

hashed\_password = hashlib.sha256(str.encode(password)).hexdigest()

# Try to open the file

try:

acct\_file = open("accounts.json", "r")

user\_accounts = json.load(acct\_file)

except IOError:

tk.messagebox.showinfo("FILE ERROR", f"File {acct\_file} does not "

f"exist.")

if any(user['username'] == username.lower() and

user['password'] == hashed\_password for user in user\_accounts):

self.main\_GUI = main.MainGUI()

self.win\_login.destroy()

if not any(user['username'] == username.lower() for

user in user\_accounts):

tk.messagebox.showinfo("Log In", f"User {username} does not exist.")

self.win\_login.entry\_username.delete(0, END)

self.win\_login.entry\_password.delete(0, END)

elif any(user['username'] == username.lower() and

not user['password'] == hashed\_password for

user in user\_accounts):

tk.messagebox.showinfo("Log In", "Invalid password.")

self.win\_login.entry\_username.delete(0, END)

self.win\_login.entry\_password.delete(0, END)

# main.py

# This file holds the Main GUI that accesses the weather station data.

from tkinter import \*

import tkinter as tk

import select\_bp\_t

import select\_bp\_ws

import select\_bp\_sc

import select\_t\_dp

class MainGUI:

"""A class holding the login GUI."""

def \_\_init\_\_(self):

"""Initialize the login GUI."""

# Create window

self.win\_main = tk.Toplevel()

self.win\_main.title("Weather Data Analysis | Main")

self.win\_main.minsize(width=500, height=250)

# Configure columns

self.win\_main.columnconfigure(0, minsize=250)

self.win\_main.columnconfigure(1, minsize=250)

# Configure rows

self.win\_main.rowconfigure(0, minsize=50)

self.win\_main.rowconfigure(1, minsize=50)

self.win\_main.rowconfigure(2, minsize=50)

self.win\_main.rowconfigure(3, minsize=50)

# Create Main Label

self.win\_main.lbl\_main = tk.Label(self.win\_main,

text="Weather Data Analysis "

"Interface",

font=("Helvetica", 14))

self.win\_main.lbl\_main.grid(row=0, column=0, columnspan=2)

# Create picture widget

photo = PhotoImage(file="data\_analysis.gif")

self.labelGIF = tk.Label(self.win\_main, image=photo)

self.labelGIF.image = photo # Retain a reference

self.labelGIF.grid(row=1, column=0, columnspan=2)

# Create Buttons

self.win\_main.btn\_1 = tk.Button(self.win\_main,

text="Barometric Pressure and "

"Temperature",

width=30,

command=self.open\_select\_bp\_t\_gui)

self.win\_main.btn\_1.grid(row=2, column=0)

self.win\_main.btn\_2 = tk.Button(self.win\_main,

text="Barometric Pressure and Wind "

"Speed",

width=30,

command=self.open\_select\_bp\_ws\_gui)

self.win\_main.btn\_2.grid(row=2, column=1)

self.win\_main.btn\_3 = tk.Button(self.win\_main,

text="Barometric Pressure and Sky "

"Cover",

width=30,

command=self.open\_select\_bp\_sc\_gui)

self.win\_main.btn\_3.grid(row=3, column=0)

self.win\_main.btn\_4 = tk.Button(self.win\_main,

text="Temperature and Dew Point",

width=30,

command=self.open\_select\_t\_dp\_gui)

self.win\_main.btn\_4.grid(row=3, column=1)

def open\_select\_bp\_t\_gui(self):

"""This function opens up the selection gui."""

self.select\_bp\_t\_GUI = select\_bp\_t.SelectBPTGUI()

def open\_select\_bp\_ws\_gui(self):

"""This function opens up the selection gui."""

self.select\_bp\_ws\_GUI = select\_bp\_ws.SelectBPWSGUI()

def open\_select\_bp\_sc\_gui(self):

"""This function opens up the selection gui."""

self.select\_bp\_sc\_GUI = select\_bp\_sc.SelectBPSCGUI()

def open\_select\_t\_dp\_gui(self):

"""This function opens up the selection gui."""

self.select\_t\_dp\_GUI = select\_t\_dp.SelectTDPGUI()

# select\_bp\_sc.py

# This file allows the user to select from the range of dates what they would

# like to view and prints a graph containing barometric pressure and sky cover

# over that span.

import tkinter as tk

from tkinter import \*

from tkinter import ttk

from tkinter import messagebox

import matplotlib.pyplot as plt

import numpy as np

import datetime

class SelectBPSCGUI:

"""This class holds the window for selecting a date range."""

def \_\_init\_\_(self):

"""Initialize the GUI"""

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.dates = []

# Read in the file and append each date to dates

with open(self.filename, 'r') as f:

for line in f:

self.date = line[13:17] + " " + line[17:19] + " " + line[19:21]\

+ " " + line[21:23] + ":" + line[23:25]

self.dates.append(self.date)

f.close()

# Create the window

self.win\_select = tk.Tk()

self.win\_select.title("Weather Data Analysis | BP and Sky Cover")

self.win\_select.minsize(width=500, height=250)

# Configure columns

self.win\_select.columnconfigure(0, minsize=150)

self.win\_select.columnconfigure(1, minsize=150)

self.win\_select.columnconfigure(2, minsize=150)

# Configure rows

self.win\_select.rowconfigure(0, minsize=75)

self.win\_select.rowconfigure(1, minsize=75)

self.win\_select.rowconfigure(2, minsize=75)

self.win\_select.rowconfigure(3, minsize=75)

# Create main label

self.lbl\_main = tk.Label(self.win\_select,

text="Weather Data Analysis",

font=("Arial", 16))

self.lbl\_main.grid(row=0, column=1)

# Create description label

self.lbl\_desc = tk.Label(self.win\_select,

text="Select a range of dates (YYYY MM DD "

"HH:MM):",

font=("Arial", 12))

self.lbl\_desc.grid(row=1, column=1)

# Create first OptionMenu.

self.opt1 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt1.config(font=('Helvetica', 12))

self.opt1.grid(row=2, column=0)

# Create middle label.

self.lbl\_to = tk.Label(self.win\_select,

text="to",

font=("Arial", 12))

self.lbl\_to.grid(row=2, column=1)

# Create second OptionMenu.

self.opt2 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt2.config(font=('Helvetica', 12))

self.opt2.grid(row=2, column=2)

# Create select button

self.btn\_select = tk.Button(self.win\_select,

text="Select",

font=("Arial", 12),

command=self.get\_values)

self.btn\_select.grid(row=3, column=1)

def get\_values(self):

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.file = open(self.filename, 'r')

self.lines = self.file.readlines()

val1 = self.opt1.get()

val2 = self.opt2.get()

ind1 = self.opt1.current()

ind2 = self.opt2.current()

bps = []

scs = []

dates = []

if val1 < val2:

# Gather range for data

self.lines = self.lines[ind1:ind2]

# Store bp, temp and date

for line in self.lines:

bp = line[106:112]

sc = line[42:45]

date = datetime.datetime(int(line[13:17]), int(line[17:19]),

int(line[19:21]), int(line[21:23]),

int(line[23:25]))

# Ignore invalid variables and append valid ones to correct lists.

if '\*' not in bp and '\*' not in sc:

dates.append(date)

bps.append(getdouble(bp))

scs.append(sc)

# Create plot variables

x = np.array(dates)

print(x)

y1 = np.array(bps)

print(y1)

y2 = np.array(scs)

print(y2)

# Create graph, plot first axis.

fig, ax1 = plt.subplots()

ax1.plot(x, y1, 'g-', label="Barometric Pressure")

ax1.set\_xlabel("Date")

ax1.set\_ylabel("Barometric Pressure", color='g')

# Create second axis.

ax2 = ax1.twinx()

ax2.plot(x, y2, 'b-', label="Sky Cover")

ax2.set\_ylabel("Sky Cover", color='b')

# Show graph.

plt.title(f"Barometric Pressure vs Sky Cover from {val1} to {val2}")

plt.show()

else:

tk.messagebox.showinfo("Select Dates", "Dates must be in "

"ascending order.")

# select\_bp\_t.py

# This file allows the user to select from the range of dates what they would

# like to view and prints a graph containing barometric pressure and temperature

# over that span.

import tkinter as tk

from tkinter import \*

from tkinter import ttk

from tkinter import messagebox

import matplotlib.pyplot as plt

import numpy as np

import datetime

class SelectBPTGUI:

"""This class holds the window for selecting a date range."""

def \_\_init\_\_(self):

"""Initialize the GUI"""

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.dates = []

# Read in the file and append each date to dates

with open(self.filename, 'r') as f:

for line in f:

self.date = line[13:17] + " " + line[17:19] + " " + line[19:21]\

+ " " + line[21:23] + ":" + line[23:25]

self.dates.append(self.date)

f.close()

# Create the window

self.win\_select = tk.Tk()

self.win\_select.title("Weather Data Analysis | BP and Temperature")

self.win\_select.minsize(width=500, height=250)

# Configure columns

self.win\_select.columnconfigure(0, minsize=150)

self.win\_select.columnconfigure(1, minsize=150)

self.win\_select.columnconfigure(2, minsize=150)

# Configure rows

self.win\_select.rowconfigure(0, minsize=75)

self.win\_select.rowconfigure(1, minsize=75)

self.win\_select.rowconfigure(2, minsize=75)

self.win\_select.rowconfigure(3, minsize=75)

# Create main label

self.lbl\_main = tk.Label(self.win\_select,

text="Weather Data Analysis",

font=("Arial", 16))

self.lbl\_main.grid(row=0, column=1)

# Create description label

self.lbl\_desc = tk.Label(self.win\_select,

text="Select a range of dates (YYYY MM DD "

"HH:MM):",

font=("Arial", 12))

self.lbl\_desc.grid(row=1, column=1)

# Create first OptionMenu.

self.opt1 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt1.config(font=('Helvetica', 12))

self.opt1.grid(row=2, column=0)

# Create middle label.

self.lbl\_to = tk.Label(self.win\_select,

text="to",

font=("Arial", 12))

self.lbl\_to.grid(row=2, column=1)

# Create second OptionMenu.

self.opt2 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt2.config(font=('Helvetica', 12))

self.opt2.grid(row=2, column=2)

# Create select button

self.btn\_select = tk.Button(self.win\_select,

text="Select",

font=("Arial", 12),

command=self.get\_values)

self.btn\_select.grid(row=3, column=1)

def get\_values(self):

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.file = open(self.filename, 'r')

self.lines = self.file.readlines()

val1 = self.opt1.get()

val2 = self.opt2.get()

ind1 = self.opt1.current()

ind2 = self.opt2.current()

bps = []

ts = []

dates = []

if val1 < val2:

# Gather range for data

self.lines = self.lines[ind1:ind2]

# Store bp, temp and date

for line in self.lines:

bp = line[106:112]

t = line[85:87]

date = datetime.datetime(int(line[13:17]), int(line[17:19]),

int(line[19:21]), int(line[21:23]),

int(line[23:25]))

# Ignore invalid variables and append valid ones to correct lists.

if '\*' not in bp and '\*' not in t:

dates.append(date)

bps.append(getdouble(bp))

ts.append(getint(t))

# Create plot variables

x = np.array(dates)

print(x)

y1 = np.array(bps)

print(y1)

y2 = np.array(ts)

print(y2)

# Create graph, plot first axis.

fig, ax1 = plt.subplots()

ax1.plot(x, y1, 'g-', label="Barometric Pressure")

ax1.set\_xlabel("Date")

ax1.set\_ylabel("Barometric Pressure", color='g')

# Create second axis.

ax2 = ax1.twinx()

ax2.plot(x, y2, 'b-', label="Temperature")

ax2.set\_ylabel("Temperature", color='b')

# Show graph.

plt.title(f"Barometric Pressure vs Temperature from {val1} to {val2}")

plt.show()

else:

tk.messagebox.showinfo("Select Dates", "Dates must be in "

"ascending order.")

# select\_bp\_ws.py

# This file allows the user to select from the range of dates what they would

# like to view and prints a graph containing barometric pressure and wind speed

# over that span.

import tkinter as tk

from tkinter import \*

from tkinter import ttk

from tkinter import messagebox

import matplotlib.pyplot as plt

import numpy as np

import datetime

class SelectBPWSGUI:

"""This class holds the window for selecting a date range."""

def \_\_init\_\_(self):

"""Initialize the GUI"""

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.dates = []

# Read in the file and append each date to dates

with open(self.filename, 'r') as f:

for line in f:

self.date = line[13:17] + " " + line[17:19] + " " + line[19:21]\

+ " " + line[21:23] + ":" + line[23:25]

self.dates.append(self.date)

f.close()

# Create the window

self.win\_select = tk.Tk()

self.win\_select.title("Weather Data Analysis | BP and Wind Speed")

self.win\_select.minsize(width=500, height=250)

# Configure columns

self.win\_select.columnconfigure(0, minsize=150)

self.win\_select.columnconfigure(1, minsize=150)

self.win\_select.columnconfigure(2, minsize=150)

# Configure rows

self.win\_select.rowconfigure(0, minsize=75)

self.win\_select.rowconfigure(1, minsize=75)

self.win\_select.rowconfigure(2, minsize=75)

self.win\_select.rowconfigure(3, minsize=75)

# Create main label

self.lbl\_main = tk.Label(self.win\_select,

text="Weather Data Analysis",

font=("Arial", 16))

self.lbl\_main.grid(row=0, column=1)

# Create description label

self.lbl\_desc = tk.Label(self.win\_select,

text="Select a range of dates (YYYY MM DD "

"HH:MM):",

font=("Arial", 12))

self.lbl\_desc.grid(row=1, column=1)

# Create first OptionMenu.

self.opt1 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt1.config(font=('Helvetica', 12))

self.opt1.grid(row=2, column=0)

# Create middle label.

self.lbl\_to = tk.Label(self.win\_select,

text="to",

font=("Arial", 12))

self.lbl\_to.grid(row=2, column=1)

# Create second OptionMenu.

self.opt2 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt2.config(font=('Helvetica', 12))

self.opt2.grid(row=2, column=2)

# Create select button

self.btn\_select = tk.Button(self.win\_select,

text="Select",

font=("Arial", 12),

command=self.get\_values)

self.btn\_select.grid(row=3, column=1)

def get\_values(self):

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.file = open(self.filename, 'r')

self.lines = self.file.readlines()

val1 = self.opt1.get()

val2 = self.opt2.get()

ind1 = self.opt1.current()

ind2 = self.opt2.current()

bps = []

wss = []

dates = []

if val1 < val2:

# Gather range for data

self.lines = self.lines[ind1:ind2]

# Store bp, temp and date

for line in self.lines:

bp = line[106:112]

ws = line[30:33].strip()

date = datetime.datetime(int(line[13:17]), int(line[17:19]),

int(line[19:21]), int(line[21:23]),

int(line[23:25]))

# Ignore invalid variables and append valid ones to correct lists.

if '\*' not in bp and '\*' not in ws:

dates.append(date)

bps.append(getdouble(bp))

wss.append(getint(ws))

# Create plot variables

x = np.array(dates)

print(x)

y1 = np.array(bps)

print(y1)

y2 = np.array(wss)

print(y2)

# Create graph, plot first axis.

fig, ax1 = plt.subplots()

ax1.plot(x, y1, 'g-', label="Barometric Pressure")

ax1.set\_xlabel("Date")

ax1.set\_ylabel("Barometric Pressure", color='g')

# Create second axis.

ax2 = ax1.twinx()

ax2.plot(x, y2, 'b-', label="Wind Speed")

ax2.set\_ylabel("Wind Speed", color='b')

# Show graph.

plt.title(f"Barometric Pressure vs Wind Speed from {val1} to {val2}")

plt.show()

else:

tk.messagebox.showinfo("Select Dates", "Dates must be in "

"ascending order.")

# select\_t\_dp.py

# This file allows the user to select from the range of dates what they would

# like to view and prints a graph containing temperature and dew point over that

# span.

import tkinter as tk

from tkinter import \*

from tkinter import ttk

from tkinter import messagebox

import matplotlib.pyplot as plt

import numpy as np

import datetime

class SelectTDPGUI:

"""This class holds the window for selecting a date range."""

def \_\_init\_\_(self):

"""Initialize the GUI"""

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.dates = []

# Read in the file and append each date to dates

with open(self.filename, 'r') as f:

for line in f:

self.date = line[13:17] + " " + line[17:19] + " " + line[19:21]\

+ " " + line[21:23] + ":" + line[23:25]

self.dates.append(self.date)

f.close()

# Create the window

self.win\_select = tk.Tk()

self.win\_select.title("Weather Data Analysis | Temperature and Dew "

"Point")

self.win\_select.minsize(width=500, height=250)

# Configure columns

self.win\_select.columnconfigure(0, minsize=150)

self.win\_select.columnconfigure(1, minsize=150)

self.win\_select.columnconfigure(2, minsize=150)

# Configure rows

self.win\_select.rowconfigure(0, minsize=75)

self.win\_select.rowconfigure(1, minsize=75)

self.win\_select.rowconfigure(2, minsize=75)

self.win\_select.rowconfigure(3, minsize=75)

# Create main label

self.lbl\_main = tk.Label(self.win\_select,

text="Weather Data Analysis",

font=("Arial", 16))

self.lbl\_main.grid(row=0, column=1)

# Create description label

self.lbl\_desc = tk.Label(self.win\_select,

text="Select a range of dates (YYYY MM DD "

"HH:MM):",

font=("Arial", 12))

self.lbl\_desc.grid(row=1, column=1)

# Create first OptionMenu.

self.opt1 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt1.config(font=('Helvetica', 12))

self.opt1.grid(row=2, column=0)

# Create middle label.

self.lbl\_to = tk.Label(self.win\_select,

text="to",

font=("Arial", 12))

self.lbl\_to.grid(row=2, column=1)

# Create second OptionMenu.

self.opt2 = ttk.Combobox(self.win\_select, values=self.dates)

self.opt2.config(font=('Helvetica', 12))

self.opt2.grid(row=2, column=2)

# Create select button

self.btn\_select = tk.Button(self.win\_select,

text="Select",

font=("Arial", 12),

command=self.get\_values)

self.btn\_select.grid(row=3, column=1)

def get\_values(self):

# Create variables

self.filename = 'Data\_10\_Years\_ALL\_NEW.txt'

self.file = open(self.filename, 'r')

self.lines = self.file.readlines()

val1 = self.opt1.get()

val2 = self.opt2.get()

ind1 = self.opt1.current()

ind2 = self.opt2.current()

ts = []

dps = []

dates = []

if val1 < val2:

# Gather range for data

self.lines = self.lines[ind1:ind2]

# Store bp, temp and date

for line in self.lines:

t = line[85:87]

dp = line[90:92]

date = datetime.datetime(int(line[13:17]), int(line[17:19]),

int(line[19:21]), int(line[21:23]),

int(line[23:25]))

# Ignore invalid variables and append valid ones to correct lists.

if '\*' not in t and '\*' not in dp:

dates.append(date)

ts.append(getint(t))

dps.append(getint(dp))

# Create graph variables

x = np.array(dates)

print(x)

y1 = np.array(ts)

print(y1)

y2 = np.array(dps)

print(y2)

# Create graph, plot first axis.

fig, ax1 = plt.subplots()

ax1.plot(x, y1, 'g-', label="Temperature")

ax1.set\_xlabel("Date")

ax1.set\_ylabel("Temperature", color='g')

# Create second axis.

ax2 = ax1.twinx()

ax2.plot(x, y2, 'b-', label="Dew Point")

ax2.set\_ylabel("Dew Point", color='b')

# Show graph.

plt.title(f"Temperature vs Dew Point from {val1} to {val2}")

plt.show()

else:

tk.messagebox.showinfo("Select Dates", "Dates must be in "

"ascending order.")