Assignment 02
Programming for scientists
CSM0120
Tob31

i. Executive summary

Similar, to the previous Assignment I wrote a single Python file split up into various functions that will be called in main. When these called functions are uncommented from main the program will run that code for that specific task, this was done do that the code for each task is clearly presented for the reader so that it can be tested and executed effectively. The UKMap class, image and plotting capability is imported from a separate python file called UKMap.py, the functions within that class are used to plot the points of the map obtained from the project directory Gb4dot_merged_mapcolors.png.

Task 1: For this task two functions were created, importTownData_csv and plotSpecificTowns, importTownData_csv imports the latlon.csv file and parses the data into a dictionary towndict and then returns it. plotSpecificTowns takes in the towndict from the previous function and the map image from the UKMap.py file and plots all the cities/towns from the towndict on the map using the latitude and longitude if it ends with (bury or ampton) or starts with either (A, B, C, L, M). However, Aberystwyth, London, Cardiff, Birmingham are plotted in red with a size of 5.

Task 2: For this Task three functions were created, getWeatherResponse, displayWeatherForcast and WeatherForcast. getWeatherResponse uses the requests.get function to retrieve the weather response data. The DisplayWeatherForcast function prints the data from the weather response dictionary onto the screen. The WeatherForcast function puts the weather forecast data into a dictionary of the current cities.

Task 3: For this Task one function was created, saveWeather, saveWeather takes the weather forecast dictionary and saves the information to a csv file in exactly the same output format as the Display weatherforcast function.

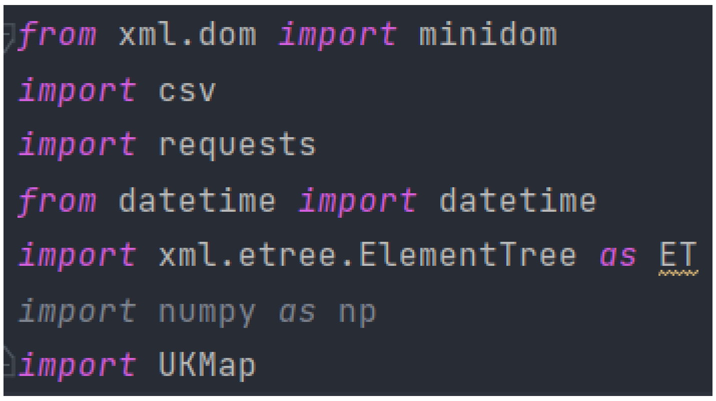
Task 4: For this Task two functions were created, citiesWeathercategories and displaySuggestions, citiesWeathercategories takes in the WeatherForcast weatherData dictionary and sums up the forecast for each city and hour and day and appends that specific city to a dictionary of categories where the weather categories are the keys and the city's the values. This dictionary is then returned and parsed into the displaySuggestions function which displays the Weather categories and all the city's experiencing that Weather type with a weather suggestion about what do to on the screen.

Task 5: For this Task one function was created, presentXML, PresentXML present the weather categories dictionary data into an xml file with all the correct indentations, the logic of this function is extremely similar to the displaySugeestions function expect instead of printing the info to the screen it adds the info to an xml element tree and then saves it to an xml file.

Task 6: For this Task two functions were created, ImportUserData_csv and plotWeatherDataOnMap, ImportUserData_csv imports the user information from a csv file users.csv and parses the information into a dictionary with cities as the key and their user counts as the values. plotWeatherDataOnMap takes in the usercount dictionary and weatherCategories data and plots the the current weather category at each significant city and makes the plotted marker proportional to the number of registered users at each city.

ii. Technical overview

imports



```python
from xml.dom import minidom
import csv
import requests
from datetime import datetime
import xml.etree.ElementTree as ET
import numpy as np
import UKMap
```

These are the inputs that I am using for this project.

From xml.dom import minidom, is imported so that I can organize the xml file output to, “look pretty”, have the correct indentations when the file is viewed. Xml.dom is an xml manipulation module.

Import csv, is imported so that csv files can be imported properly. CSV is a csv file reading and writing module.

Import requests, I imported so that the weather api data request can be requested and responded to using HTTP.

From datetime import datetime, is a module used for manipulating dates and times, it is used to get the current date and time.

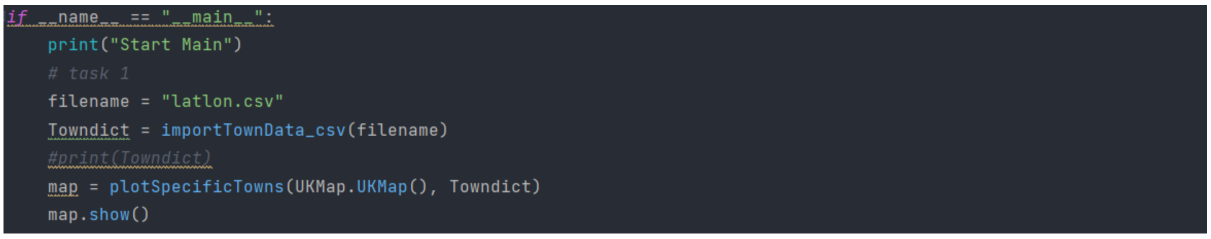
Import xml.etree.ElementTree as Et, Is a module used to help build/create xml trees used to create xml files

Import numpy as np numpy is a mathematical module, was it is used in this project to help add a weighting to the usernumbers that effect the size of the plotted markers on the uk map, it isn’t currently being used, however with more users it would be a beneficial module to make use of.

Import UKMap, this imports the UKMap class from the UKMap python file in the project directory, to be used to create and plot the UK map.

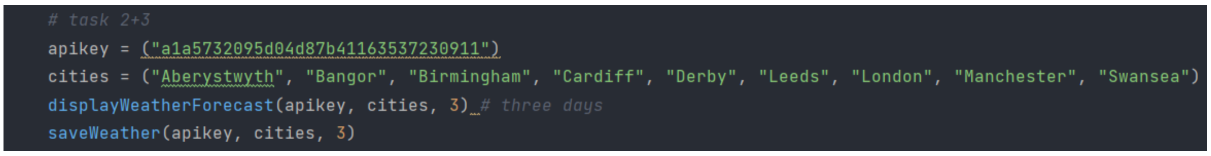
The main

Each commented section below correlates to the functions for each question. There are multiple variables that are duplicated here allow this seems like bad programming practice, it helps with running the sections of code when previous ones will me commented out stopping them from running.



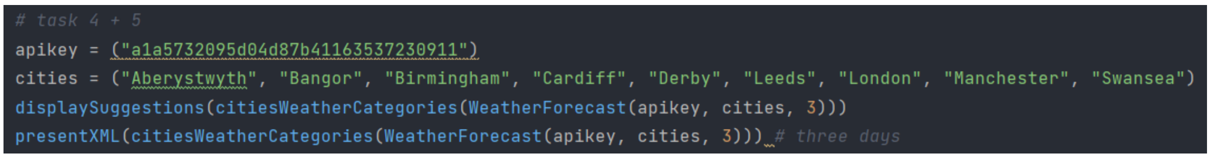
```python
if __name__ == "__main__":
    print("Start Main")
    # task 1
    filename = "latlon.csv"
    Towndict = importTownData_csv(filename)
    #print(Towndict)
    map = plotSpecificTowns(UKMap.UKMap(), Towndict)
    map.show()
```

In task 1 a dictionary with the town/city data is initialized using the importTownData_csv function, this dictionary is then parsed into the plotSpecificTowns function which also parses in the UKMap class from the UKMap python file in the project directory. The map is then shown to the screen.



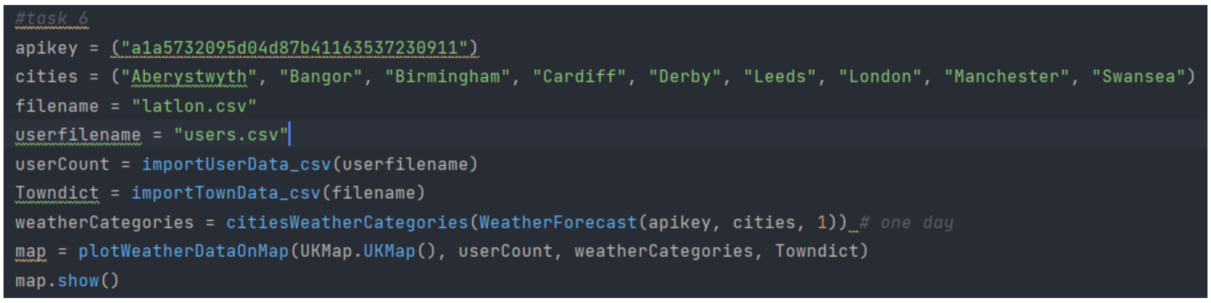
```python
    # task 2+3
    apikey = ("a1a5732095d04d87b41163537230911")
    cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea")
    displayWeatherForecast(apikey, cities, 3) # three days
    saveWeather(apikey, cities, 3)
```

In task 2 and 3 a string is initialized apikey containing the apikey needed to access the api weather data. a list of strings containing the significant cities is also initialized before being parsed into both the displayWeatherforcast function and the SaveWeather function along with an integer which represents the number of days the forecast will cover.



```python
# task 4 + 5
apikey = ("a1a5732095d04d87b41163537230911")
cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea")
displaySuggestions(citiesWeatherCategories(WeatherForecast(apikey, cities, 3)))
presentXML(citiesWeatherCategories(WeatherForecast(apikey, cities, 3))) # three days
```

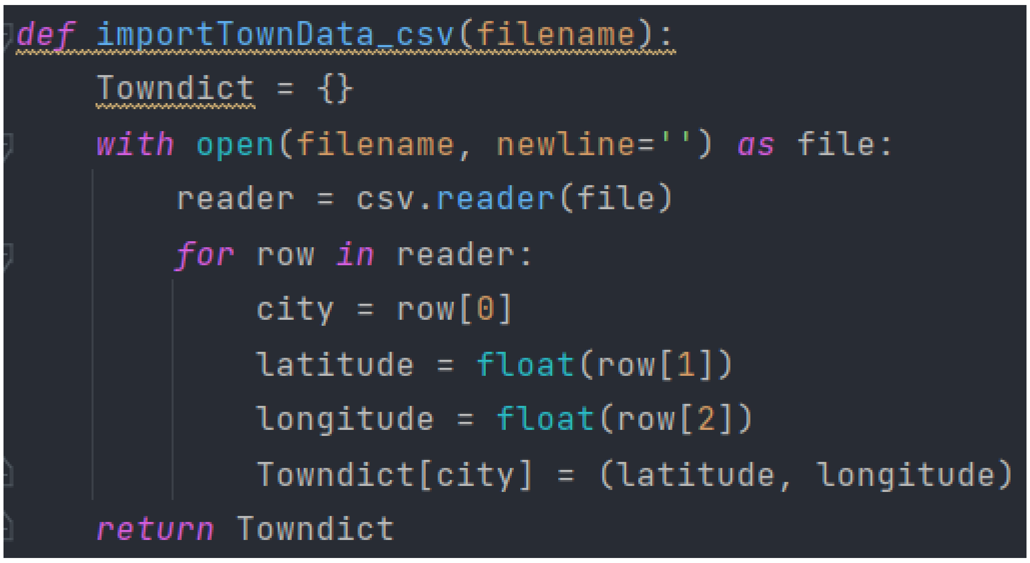
In task 4 and 5 the apikey, cities and the integer 3 are initialized similarly and parsed into the correct functions WeatherForcast to get the weather response data then citiesWeatherCategories to get the weather categories dictionary and then displaySuggestions which will display the information to the screen. presentXML, the apikey, cities and the integer 3 are similarly parsed into the correct functions WeatherForcast to get the weather response data then citiesWeatherCategories to get the weather categories dictionary, the data is then presented in an XML format/ output to an XML file, Notice the integer being parsed, it is to represent the number of days the forecast will cover.



```python
#task 6
apikey = ("a1a5732095d04d87b41163537230911")
cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea")
filename = "latlon.csv"
userfilename = "users.csv"
userCount = importUserData_csv(userfilename)
Towndict = importTownData_csv(filename)
weatherCategories = citiesWeatherCategories(WeatherForecast(apikey, cities, 1)) # one day
map = plotWeatherDataOnMap(UKMap.UKMap(), userCount, weatherCategories, Towndict)
map.show()
```

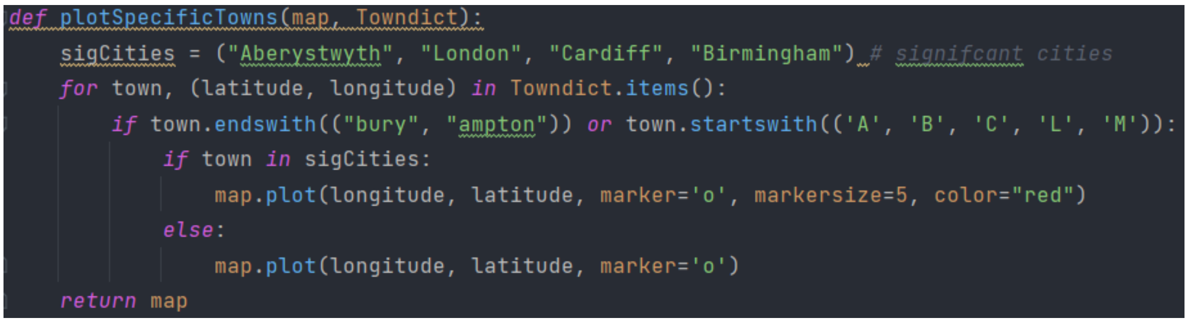
In task 6 the varaiables are initialized again along with the files that it will import latlon.csv and users.csv, notice how all the variables are parsed into the specific functions like “latlon.csv” to filename and “users.csv” to userfilename. Notice the forecast day for the weatherCategories function also takes in an integer of only one as to only get the current weather for the day. Userfilename is parsed into the importuserdata_csv function and similarly the filename is parsed into the importTownData_csv function so that both output the relevant data dictionaries. the Weather categories dictionary is also inialized in the same fashion. These dictionaries are then parsed into the PlotWeatherDataOnMap function along with the UKMap class from the UKMap python file so that the data can be plotted to the map. The plotted map is then displayed to the screen using map.show().

Task 1



```python
def importTownData_csv(filename):
    Towndict = {}
    with open(filename, newline='') as file:
        reader = csv.reader(file)
        for row in reader:
            city = row[0]
            latitude = float(row[1])
            longitude = float(row[2])
            Towndict[city] = (latitude, longitude)
    return Towndict
```

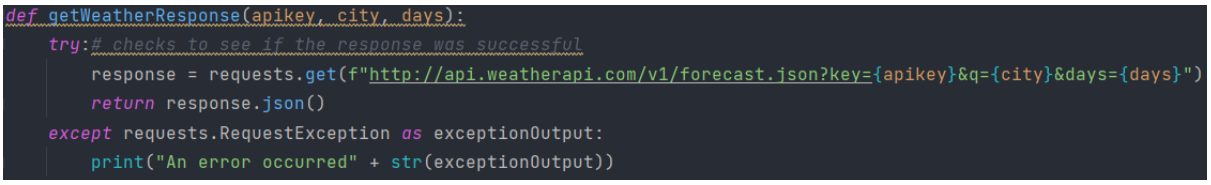
Using the csv reader import, this function opens the csv file for the towndata “latlon.csv” and for each row or line in the file the important information is extracted (i.e. the latitude, longitude and city found at index 0 for city, 1 for latitude and 2 for longitude) and added to a dictionary which is returned to be used later.



```python
def plotSpecificTowns(map, Towndict):
    sigCities = ("Aberystwyth", "London", "Cardiff", "Birmingham") # signifcant cities
    for town, (latitude, longitude) in Towndict.items():
        if town.endswith(("bury", "ampton")) or town.startswith(('A', 'B', 'C', 'L', 'M')):
            if town in sigCities:
                map.plot(longitude, latitude, marker='o', markersize=5, color="red")
            else:
                map.plot(longitude, latitude, marker='o')
    return map
```

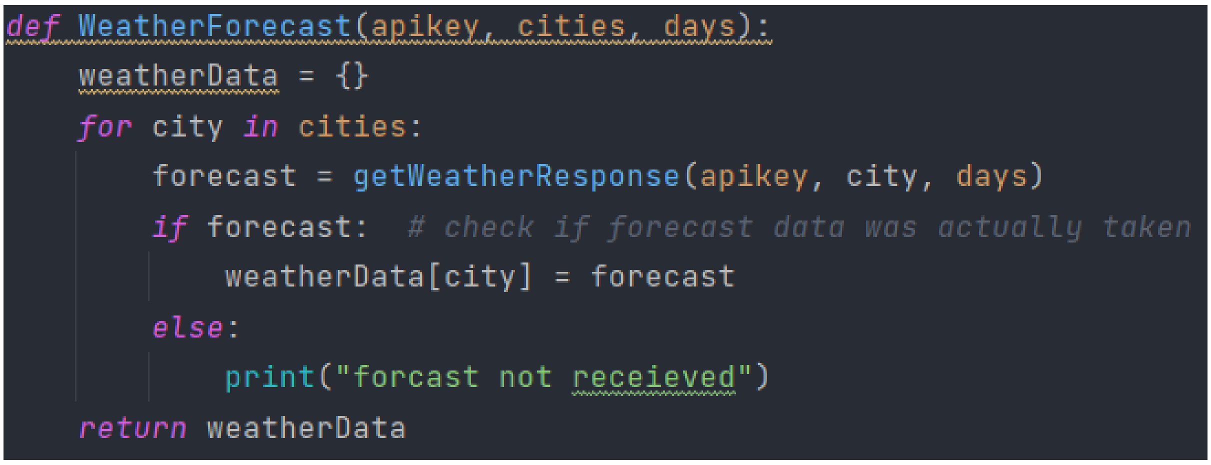
For this task the only significant cities/towns were Aberystwyth, London, Cardiff and Birmingham. So, a list of them as initialized at the beginning of this function. This function uses a for loop to loop through the towns/cities in the town dictionary and if it ends with wither bury or ampton, or Starts with A, B, C, L or M the Town/city is plotted on the map. However, inside that if statement is another that plots the town/city on the map in red at size 5 if it is in the list of significant cities.

Task 2



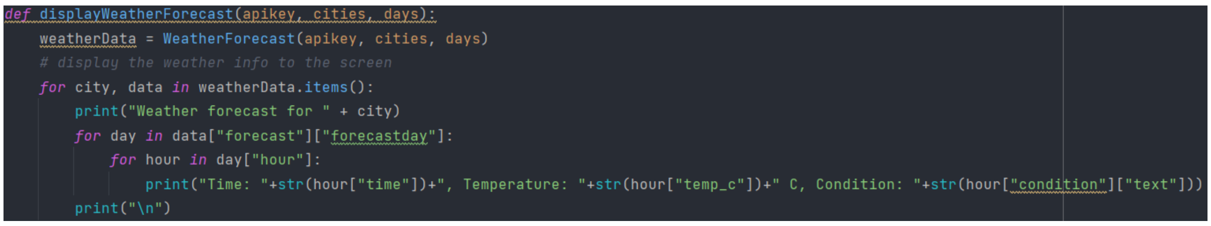
```python
def getWeatherResponse(apikey, city, days):
    try:# checks to see if the response was successful
        response = requests.get(f"http://api.weatherapi.com/v1/forecast.json?key={apikey}&q={city}&days={days}")
        return response.json()
    except requests.RequestException as exceptionOutput:
        print("An error occurred" + str(exceptionOutput))
```

This function getWeatherResponse, gets the weather response data from the weatherapi.com site using the api key, the cities and the number of days (3) using the request.get function. A try and expect statement is used to make this more defensing and provide the user with more information should it fail.



```python
def WeatherForecast(apikey, cities, days):
    weatherData = {}
    for city in cities:
        forecast = getWeatherResponse(apikey, city, days)
        if forecast:  # check if forecast data was actually taken
            weatherData[city] = forecast
        else:
            print("forcast not receieved")
    return weatherData
```

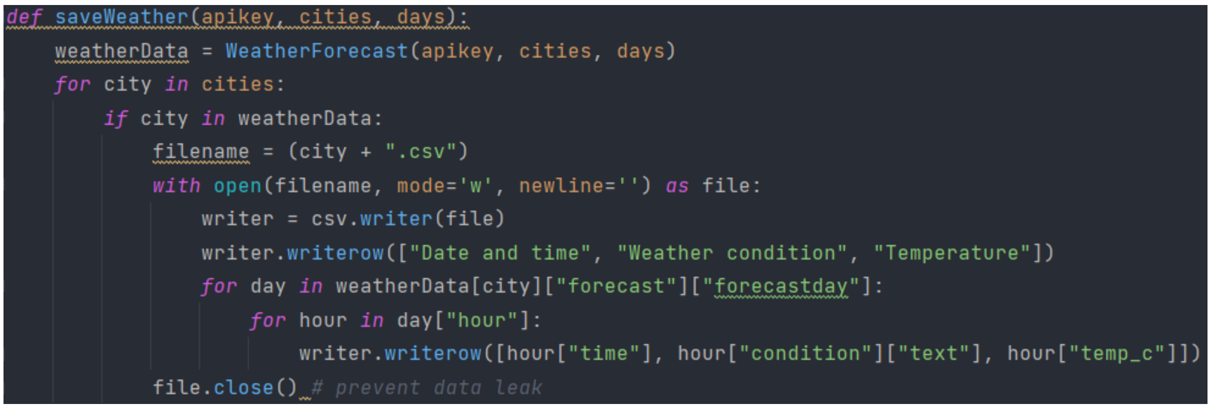
The WeatherForecast function is used to add the weather forcast data to a dictionary where the forcast data is the value and its city the key. The for loop loops through the elements/cities in the cities list and An If statement is used to check if the data was actually received, if not an appropriate response will be displayed on the screen. The weather data dictionary is then returned.



```python
def displayWeatherForecast(apikey, cities, days):
    weatherData = WeatherForecast(apikey, cities, days)
    # display the weather info to the screen
    for city, data in weatherData.items():
        print("Weather forecast for " + city)
        for day in data["forecast"]["forecastday"]:
            for hour in day["hour"]:
                print("Time: "+str(hour["time"])+", Temperature: "+str(hour["temp_c"])+" C, Condition: "+str(hour["condition"]["text"]))
        print("\n")
```

The displayWeatherForcast function displays the weather data dictionary retrieved from the api. It does this by using a for loop to loop through the cities in dictionary then uses a second for loop to loop through the days in the dictionary for each city so that each element about the time, temperature and condition can be indexed and printed to the screen with the print functions. A new line is needed to be printed after doing so, so that the data will be displayed on a new line.

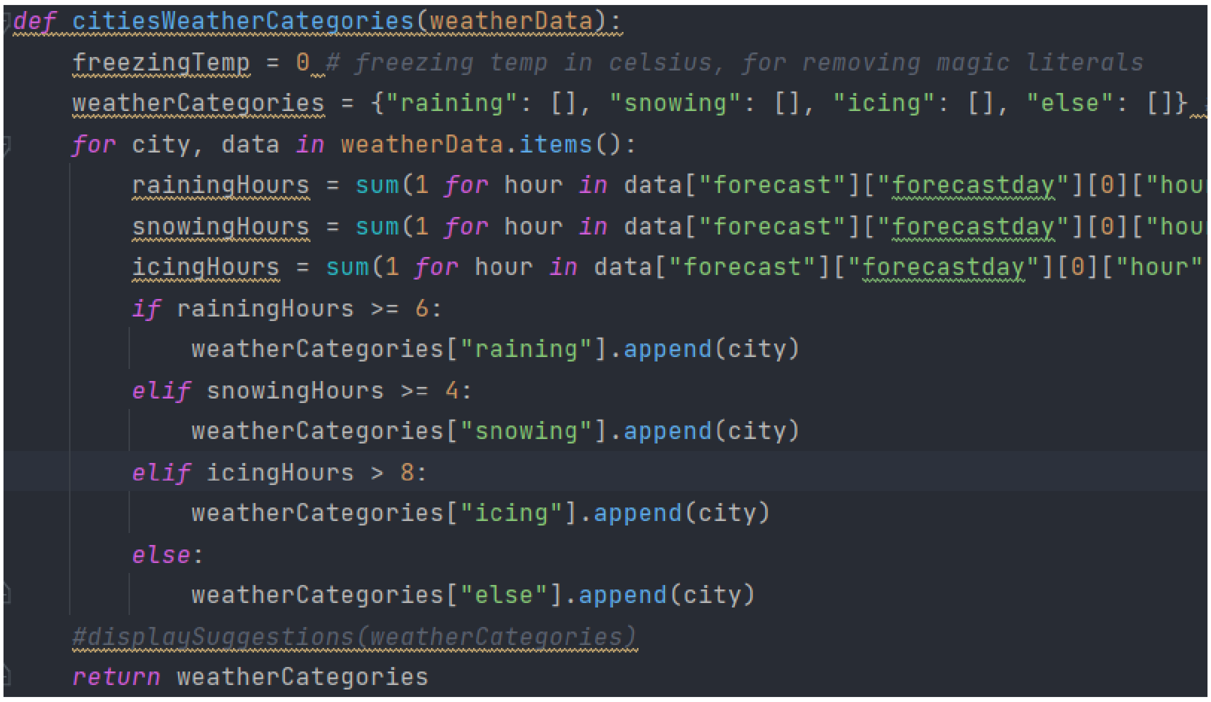
Task 3



```python
def saveWeather(apikey, cities, days):
    weatherData = WeatherForecast(apikey, cities, days)
    for city in cities:
        if city in weatherData:
            filename = (city + ".csv")
            with open(filename, mode='w', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(["Date and time", "Weather condition", "Temperature"])
                for day in weatherData[city]["forecast"]["forecastday"]:
                    for hour in day["hour"]:
                        writer.writerow([hour["time"], hour["condition"]["text"], hour["temp_c"]])
            file.close() # prevent data leak
```

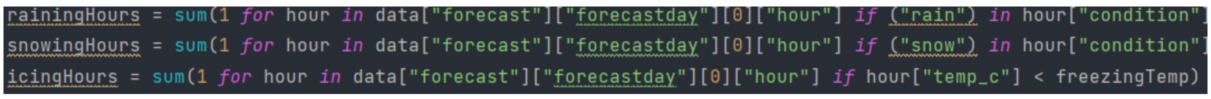
The saveWeather function here is logically almost exactly the same as the display weather function, however the difference is when in save weather it would print the data to the screen, it instead writes the data to a csv file. A for loop is used to loop through each city in the cities list, if the city is in WeatherData dictionary a csv file is created for that city, the weather data for that city is then written to the file using for loops to write the forecast for each hour of each day to the individual city csv file. A file.close() statement is used at the end to prevent a data leak.

Task 4



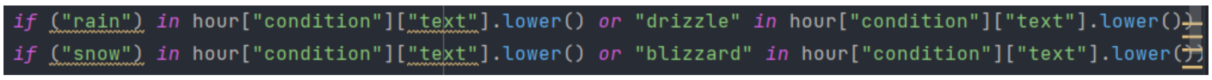
```python
def citiesWeatherCategories(weatherData):
    freezingTemp = 0 # freezing temp in celsius, for removing magic literals
    weatherCategories = {"raining": [], "snowing": [], "icing": [], "else": []}
    for city, data in weatherData.items():
        rainingHours = sum(1 for hour in data["forecast"]["forecastday"][0]["hou
        snowingHours = sum(1 for hour in data["forecast"]["forecastday"][0]["hou
        icingHours = sum(1 for hour in data["forecast"]["forecastday"][0]["hour"]
        if rainingHours >= 6:
            weatherCategories["raining"].append(city)
        elif snowingHours >= 4:
            weatherCategories["snowing"].append(city)
        elif icingHours > 8:
            weatherCategories["icing"].append(city)
        else:
            weatherCategories["else"].append(city)
    #displaySuggestions(weatherCategories)
    return weatherCategories
```

These two images bellow represent the long line just after the for loop so that I can be seen.



```python
rainingHours = sum(1 for hour in data["forecast"]["forecastday"][0]["hour"] if ("rain") in hour["condition"]
snowingHours = sum(1 for hour in data["forecast"]["forecastday"][0]["hour"] if ("snow") in hour["condition"]
icingHours = sum(1 for hour in data["forecast"]["forecastday"][0]["hour"] if hour["temp_c"] < freezingTemp)
```

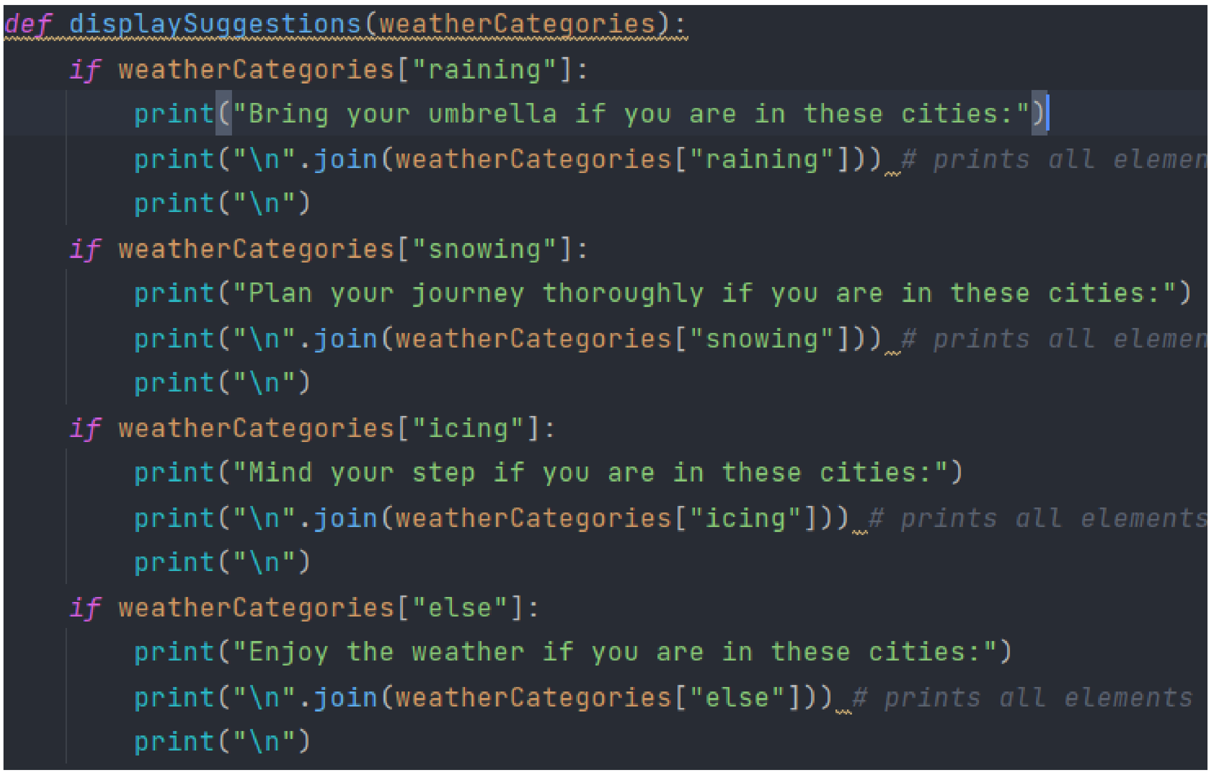
+/



```python
if ("rain") in hour["condition"]["text"].lower() or "drizzle" in hour["condition"]["text"].lower())
if ("snow") in hour["condition"]["text"].lower() or "blizzard" in hour["condition"]["text"].lower())
```

This function is for creating the Weather categories dictionary. First it initializes a freezing temperature integer value to 0 for Celsius. Then the Weather categories dictionary is initialized with the 4 categories as the keys. A for loop is used to then loop through the cities in the weatherData dictionary summing up the total hours for each weather category for each city. For example, blizzard and snow conditions are equivalent to when counting the hours of snowing which is the same for rain and drizzle, when the temp is below freezing temperature i.e. 0 degrees Celsius then those hours are freezing hours.

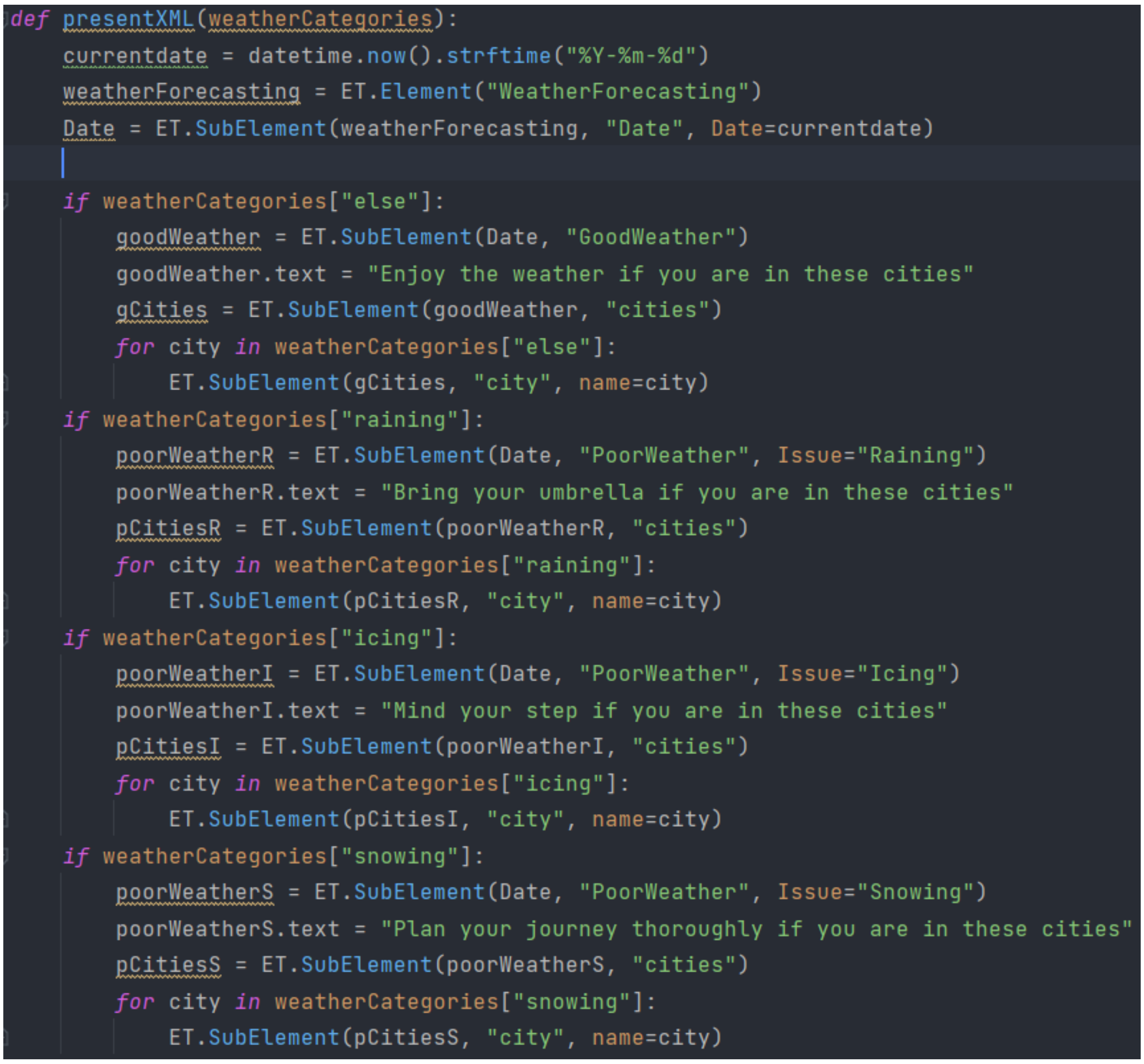
Then an if statement with elif's is used to append the current city in the for loop to the weather categories dictionary if weather categories hours exceeds the specified amount, for example if it has been raining in the city for 6 or more hours then the city will be appended to the raining category in the weather categories dictionary, or 4 or more hours for snowing and over 8 hours for icing if the temp has been below 0.



```
def displaySuggestions(weatherCategories):
    if weatherCategories["raining"]:
        print("Bring your umbrella if you are in these cities:")
        print("\n".join(weatherCategories["raining"])) # prints all elemen
        print("\n")
    if weatherCategories["snowing"]:
        print("Plan your journey thoroughly if you are in these cities:")
        print("\n".join(weatherCategories["snowing"])) # prints all elemen
        print("\n")
    if weatherCategories["icing"]:
        print("Mind your step if you are in these cities:")
        print("\n".join(weatherCategories["icing"])) # prints all elements
        print("\n")
    if weatherCategories["else"]:
        print("Enjoy the weather if you are in these cities:")
        print("\n".join(weatherCategories["else"])) # prints all elements
        print("\n")
```

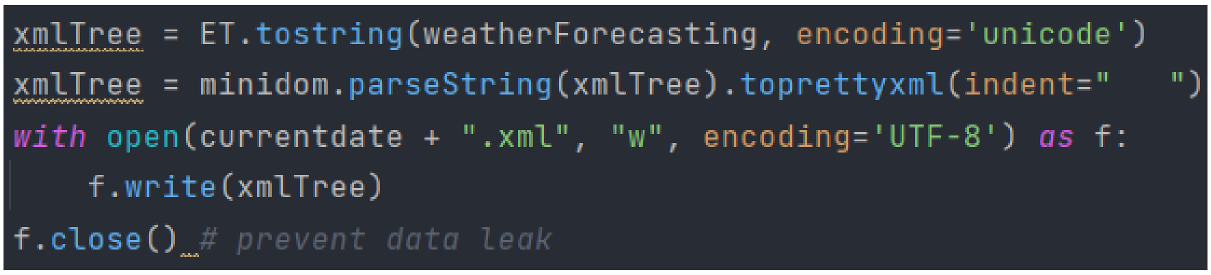
This function displaySuggestions is used to display the weather suggestions to the screen. It takes in the weather categories dictionary obtained after parsing in the output from the previous function. Four if statements are used here, so that each has equal opportunity to run, one for each weather category, if the weather category in the weatherCategories dictionary equals “raining” then it will print the weather suggestion “Bring your umbrella if you are in these cities:” and then it will print all of the elements within that dictionary were the category is raining. on a newline detonated by a “\n”. A newline “\n” is then used again so that when the next weather category is output to the screen, there is a newline separating it.

Task 5



```python
def presentXML(weatherCategories):
    currentdate = datetime.now().strftime("%Y-%m-%d")
    weatherForecasting = ET.Element("WeatherForecasting")
    Date = ET.SubElement(weatherForecasting, "Date", Date=currentdate)

    if weatherCategories["else"]:
        goodWeather = ET.SubElement(Date, "GoodWeather")
        goodWeather.text = "Enjoy the weather if you are in these cities"
        gCities = ET.SubElement(goodWeather, "cities")
        for city in weatherCategories["else"]:
            ET.SubElement(gCities, "city", name=city)
    if weatherCategories["raining"]:
        poorWeatherR = ET.SubElement(Date, "PoorWeather", Issue="Raining")
        poorWeatherR.text = "Bring your umbrella if you are in these cities"
        pCitiesR = ET.SubElement(poorWeatherR, "cities")
        for city in weatherCategories["raining"]:
            ET.SubElement(pCitiesR, "city", name=city)
    if weatherCategories["icing"]:
        poorWeatherI = ET.SubElement(Date, "PoorWeather", Issue="Icing")
        poorWeatherI.text = "Mind your step if you are in these cities"
        pCitiesI = ET.SubElement(poorWeatherI, "cities")
        for city in weatherCategories["icing"]:
            ET.SubElement(pCitiesI, "city", name=city)
    if weatherCategories["snowing"]:
        poorWeatherS = ET.SubElement(Date, "PoorWeather", Issue="Snowing")
        poorWeatherS.text = "Plan your journey thoroughly if you are in these cities"
        pCitiesS = ET.SubElement(poorWeatherS, "cities")
        for city in weatherCategories["snowing"]:
            ET.SubElement(pCitiesS, "city", name=city)
```

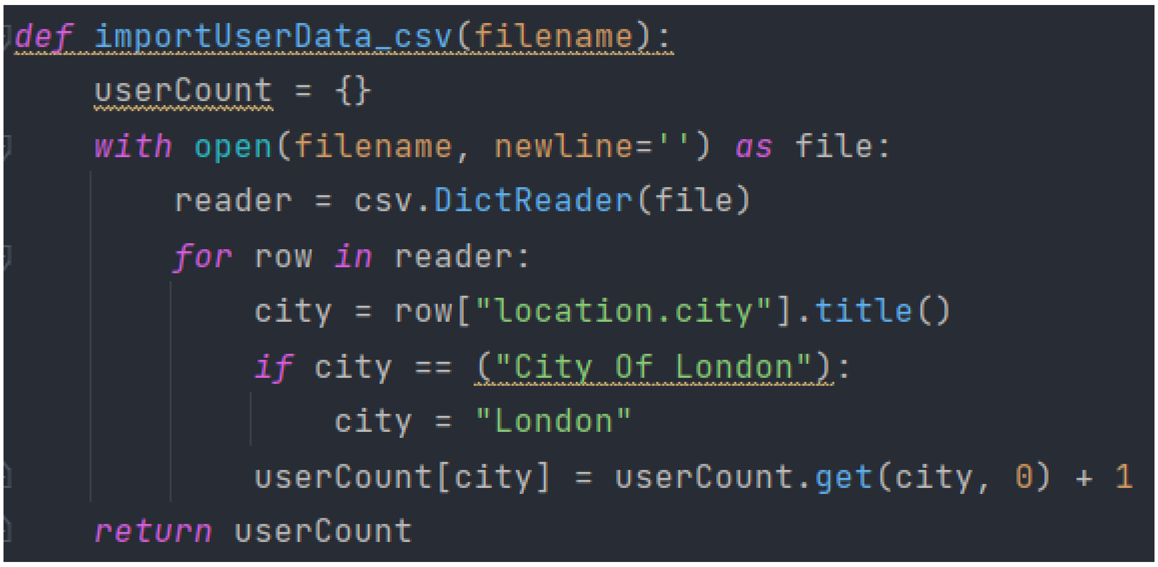


```python
xmlTree = ET.tostring(weatherForecasting, encoding='unicode')
xmlTree = minidom.parseString(xmlTree).toprettyxml(indent="    ")
with open(currentdate + ".xml", "w", encoding='UTF-8') as f:
    f.write(xmlTree)
f.close() # prevent data leak
```

This function PresentXML is used to output the data to an XML file / present the data as an xml formatting style, this function is logically very similar to the previous function display suggestions, it starts out by getting the current date, this will be used to name the file. The Element for the xml element tree is the named Weather forcasting and the sub element under that will be named date with the current date. Then the if statements begin, starting with the weather category “else” if there is a city with the weather category “else” then a subelement is created named good weather, with the text “enjoy the weather if you live in these cities” another sub element is then created to contain the cities, a for loop is then used to loop through the weather categories dictionary and create a sub element for the city with the name being the name of the city. For the other Weather categories they are under the poor weather sub element, so an if statement is then used again similar to the weather category else, the weather category next will be “raining” the sub element is then created with the name poor weather and the text is created for that sub element to be “bring your umbrella if you are in these cities, a for loop is used to loop through the weather categories dictionary and add a sub element for the city in cities with the name of that city. The same process is then done for the last two weather categories snowing and icing, where the text for those categories is “Mind your step if you are in these cities” for icing and “plan your journey thoroughly if you are in these cities” for snowing.

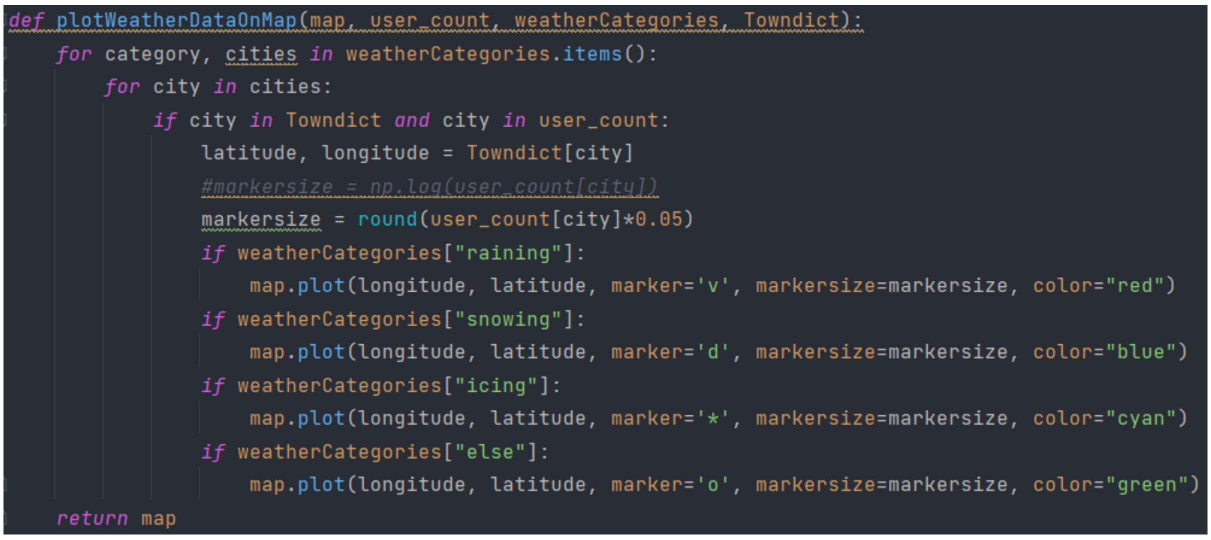
Finally an ET.tostring is used to form the xml tree, the next line minidom.parseString(xmltree).toprettyxml is used to make the xml file look “pretty” It will have the correct indentations for every element and sub elements. The Xml file is then created with the name being the current date with the extension “.xml”, a file close() is used to close the file if it is not closed and used to prevent a potential data leak.

Task 6



```python
def importUserData_csv(filename):
    userCount = {}
    with open(filename, newline='') as file:
        reader = csv.DictReader(file)
        for row in reader:
            city = row["location.city"].title()
            if city == ("City Of London"):
                city = "London"
            userCount[city] = userCount.get(city, 0) + 1
    return userCount
```

This function is used to import the user data from the “users.csv” file and output a dictionary with the desired data, firstly an empty dictionary userCount is initialized then the file is opened as a file, the csv reader module that was imported is then used to read the data into a variable reader. A for loop is then used to loop through the csv reader data and add the city names from the file to a variable city. An if statement is used to remove the occurrences of City of London and Replace them with just London. The userCount dictionary is then updated with the city name as the key and the value being the incremented count of those cities in the dictionary. The userCount dictionary is then returned.



```python
def plotWeatherDataOnMap(map, user_count, weatherCategories, Towndict):
    for category, cities in weatherCategories.items():
        for city in cities:
            if city in Towndict and city in user_count:
                latitude, longitude = Towndict[city]
                #markersize = np.log(user_count[city])
                markersize = round(user_count[city]*0.05)
                if weatherCategories["raining"]:
                    map.plot(longitude, latitude, marker='v', markersize=markersize, color="red")
                if weatherCategories["snowing"]:
                    map.plot(longitude, latitude, marker='d', markersize=markersize, color="blue")
                if weatherCategories["icing"]:
                    map.plot(longitude, latitude, marker='*', markersize=markersize, color="cyan")
                if weatherCategories["else"]:
                    map.plot(longitude, latitude, marker='o', markersize=markersize, color="green")
    return map
```

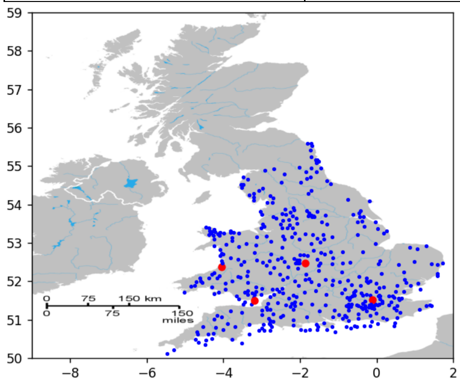
This function is then used to plot the user data on the map of the UK. Two for loops are used, one to loop through the weather categories dictionary then the second to loop through the each city in cities from the weather categories, if the city is in both the town dictionary towndict and user count dictionary user_count, then the local variables latitude and longitude are initialized from indexing the town dictionary with the city. The marker size is then created by multiplying the user count number by 0.05 and rounding it. If statements are then used to check which category of weather is affecting the city so that when the marker is plotted on to the map it has the correct colour and marker type. Raining is a red triangle, snowing is a blue diamond, icing is a cyan star, and else is a green circle. The map is then returned and displayed.

iii. Software testing

For the testing of this software, incrementally unit testes were used to tested the functionality and output of each function in the program by printing the returned data from the called function in main, this method was beneficial as problems could be found and solved easily at the source, For example the error with London being named “the city of London” in the users.csv file was found using this method.

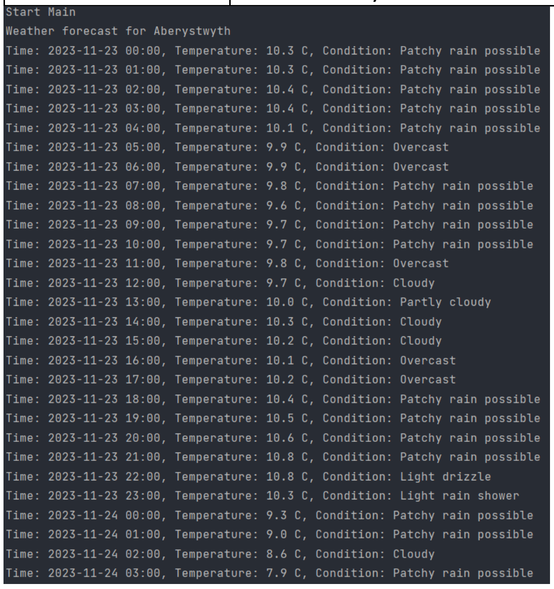
Task 1 Testing:

	input	Expected output	Actual output pass/fail
Expected input	"latlon.csv" file	The correct points plotted on the UK map	🟩



Task 2 Testing:

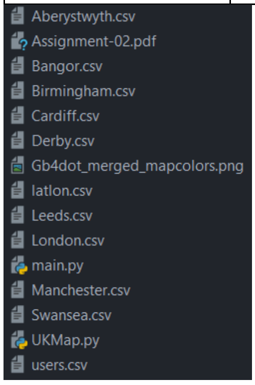
	input	Expected output	Actual output pass/fail
Expected input	apikey = ("a1a5732095d04d87b41163537230911") cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea")	For each city a report for every hour for 3 days with the temp and condition	🟩



```
Start Main
Weather forecast for Aberystwyth
Time: 2023-11-23 00:00, Temperature: 10.3 C, Condition: Patchy rain possible
Time: 2023-11-23 01:00, Temperature: 10.3 C, Condition: Patchy rain possible
Time: 2023-11-23 02:00, Temperature: 10.4 C, Condition: Patchy rain possible
Time: 2023-11-23 03:00, Temperature: 10.4 C, Condition: Patchy rain possible
Time: 2023-11-23 04:00, Temperature: 10.1 C, Condition: Patchy rain possible
Time: 2023-11-23 05:00, Temperature: 9.9 C, Condition: Overcast
Time: 2023-11-23 06:00, Temperature: 9.9 C, Condition: Overcast
Time: 2023-11-23 07:00, Temperature: 9.8 C, Condition: Patchy rain possible
Time: 2023-11-23 08:00, Temperature: 9.6 C, Condition: Patchy rain possible
Time: 2023-11-23 09:00, Temperature: 9.7 C, Condition: Patchy rain possible
Time: 2023-11-23 10:00, Temperature: 9.7 C, Condition: Patchy rain possible
Time: 2023-11-23 11:00, Temperature: 9.8 C, Condition: Overcast
Time: 2023-11-23 12:00, Temperature: 9.7 C, Condition: Cloudy
Time: 2023-11-23 13:00, Temperature: 10.0 C, Condition: Partly cloudy
Time: 2023-11-23 14:00, Temperature: 10.3 C, Condition: Cloudy
Time: 2023-11-23 15:00, Temperature: 10.2 C, Condition: Cloudy
Time: 2023-11-23 16:00, Temperature: 10.1 C, Condition: Overcast
Time: 2023-11-23 17:00, Temperature: 10.2 C, Condition: Overcast
Time: 2023-11-23 18:00, Temperature: 10.4 C, Condition: Patchy rain possible
Time: 2023-11-23 19:00, Temperature: 10.5 C, Condition: Patchy rain possible
Time: 2023-11-23 20:00, Temperature: 10.6 C, Condition: Patchy rain possible
Time: 2023-11-23 21:00, Temperature: 10.8 C, Condition: Patchy rain possible
Time: 2023-11-23 22:00, Temperature: 10.8 C, Condition: Light drizzle
Time: 2023-11-23 23:00, Temperature: 10.3 C, Condition: Light rain shower
Time: 2023-11-24 00:00, Temperature: 9.3 C, Condition: Patchy rain possible
Time: 2023-11-24 01:00, Temperature: 9.0 C, Condition: Patchy rain possible
Time: 2023-11-24 02:00, Temperature: 8.6 C, Condition: Cloudy
Time: 2023-11-24 03:00, Temperature: 7.9 C, Condition: Patchy rain possible
```

Task 3 Testing:

	input	Expected output	Actual output pass/fail
Expected input	apikey = ("a1a5732095d04d87b41163537230911") cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea")	A csv output file for each city with the a weather report for each hour for 3 days	🟩

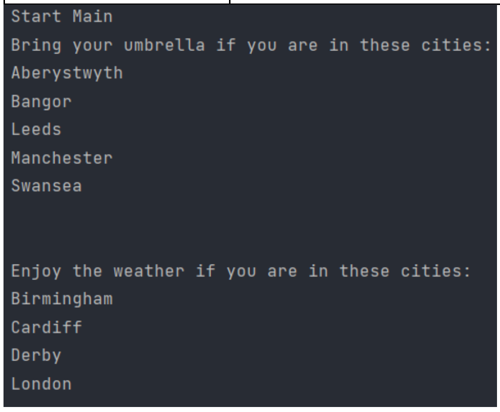


```
Aberystwyth.csv
Assignment-02.pdf
Bangor.csv
Birmingham.csv
Cardiff.csv
Derby.csv
Gb4dot_merged_mapcolors.png
latlon.csv
Leeds.csv
London.csv
main.py
Manchester.csv
Swansea.csv
UKMap.py
users.csv
```

Task 4 Testing:

	input	Expected output	Actual output pass/fail
Expected input	apikey = ("a1a5732095d04d87b41163537230911")	The Weather suggestions for	🟩

	cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea")	the weather categories effecting the cities and nothing else	🟩

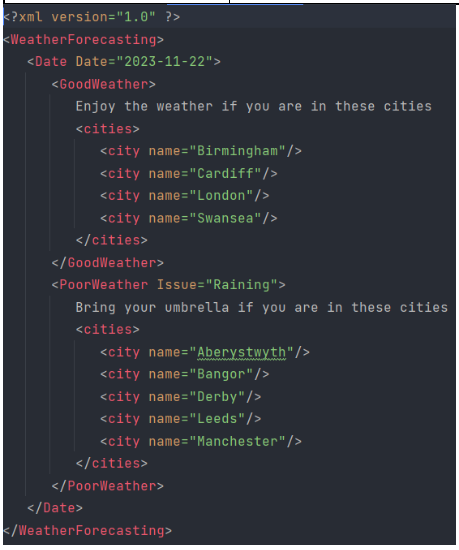


```
Start Main
Bring your umbrella if you are in these cities:
Aberystwyth
Bangor
Leeds
Manchester
Swansea


Enjoy the weather if you are in these cities:
Birmingham
Cardiff
Derby
London
```

Task 5 Testing:

	input	Expected output	Actual output pass/fail
Expected input	apikey = ("a1a5732095d04d87b41163537230911") cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea")	The Weather suggestions for the weather categories effecting the cities and nothing else in an xml format with the correct indentations	🟩

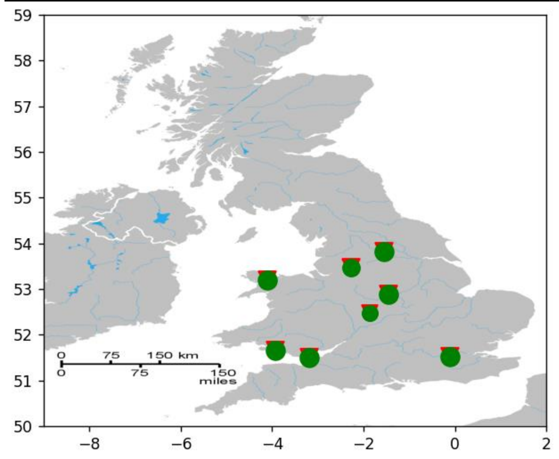


```
<?xml version="1.0" ?>
<WeatherForecasting>
   <Date Date="2023-11-22">
      <GoodWeather>
         Enjoy the weather if you are in these cities
         <cities>
            <city name="Birmingham"/>
            <city name="Cardiff"/>
            <city name="London"/>
            <city name="Swansea"/>
         </cities>
      </GoodWeather>
      <PoorWeather Issue="Raining">
         Bring your umbrella if you are in these cities
         <cities>
            <city name="Aberystwyth"/>
            <city name="Bangor"/>
            <city name="Derby"/>
            <city name="Leeds"/>
            <city name="Manchester"/>
         </cities>
      </PoorWeather>
   </Date>
</WeatherForecasting>
```

Task 6 Testing:

	input	Expected output	Actual output pass/fail

Expected input	apikey = ("a1a5732095d04d87b41163537230911") cities = ("Aberystwyth", "Bangor", "Birmingham", "Cardiff", "Derby", "Leeds", "London", "Manchester", "Swansea") "latlon.csv" file "users.csv" file	The significant cities ploted with the current weather effecting them over the course of this day, with their sizing matching the number of users at each city.	🟩



iv. Reflections and future work

Overall, I have concluded that I have completed the tasks for the assignment in a clean and elegant fashion, all functions output the desired information given the api input, the implementation is defensive as is used try and except functions when needed to return error information back to the user. Future considerations for this work are, in the final task if there are multiple weather categories for that day i.e., raining and icing, both shapes will be plotted on the graph at the same location unless I implement elif instead of if statements, I would like it to be the most dominate weather category without repeating large chunks of code again in the program. Also, looking back at task 4 + 5 in main, I could have pre-initialized the inputs for the two functions there and then just reused the same variable additionally I should have created a number of days integer variable so that I could avoid using magic literals in my program. This assignment took me around 26 hours to complete in addition to 8 hours for the report.