Overview

This report was made by collecting data from http://m.bom.gov.au/vic/melbourne/ and https://www.timeanddate.com/weather/australia/melbourne/ext. After collection of data tables were established by Web Scrapping Tool in Python in respective CSV files. For regular Interval I took reading after the interval of every 20 minutes and for Discrete Interval, I took the reading after every 1 Celsius Temperature Change. The Respective changes can be seen under their specific headings. For Discrete, the program took 15.5 hours to collect the data whereas for Regular Interval, the program took the time of around three hours. After that we chose that which sampling is optimal. Once, it was done, than Data Integration (or Data Fusion) from two different data source takes. In end, Predictions takes place of the "Fused Data".

Regular Interval

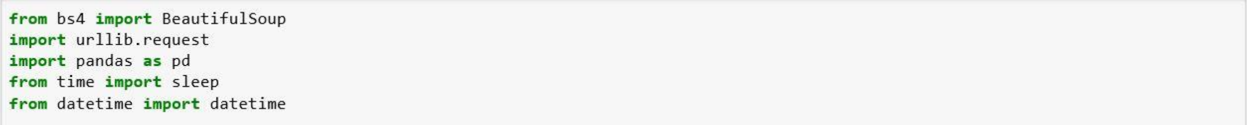
The *Figure 1.1* shows the values obtained from the running the Regular Interval Code (shown on *Figure 1.2.1,1.2.2, 1.2.3,1.2.4,1.2.5,1.2.6)* on the website http://m.bom.gov.au/vic/melbourne/. These Records were taken from the interval of every 20 minutes and once the values were obtained, they were put into the CSV named as Regular-1 as shown on the Figure 1.3. Approximately, it took around 3 hours to collect 10 samples of data.

Note: The Current Time is in PM.

Current Time	**Temperature**	**Wind Speed**	**Humidity**
4:02	16.6	9	67
4:22	16.1	7	73
4:42	16.1	11	69
5:02	16.1	13	74
5:22	16	7	77
5:42	16	7	75

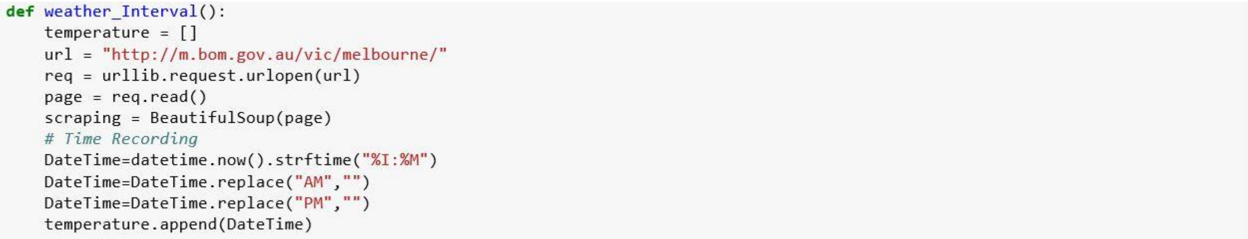
6:02	16	11	73
6:22	15.9	11	73
6:42	15.4	15	74
7:02	15	13	72

Figure 1.1



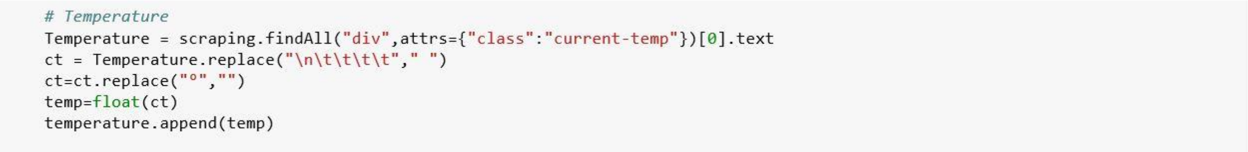
```
from bs4 import BeautifulSoup
import urllib.request
import pandas as pd
from time import sleep
from datetime import datetime
```

Figure 1.2.1 – The Libraries required to run the program



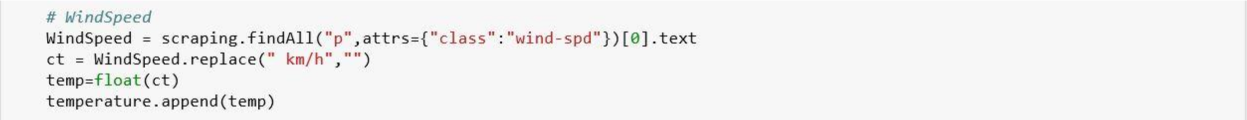
```
def weather_Interval():
    temperature = []
    url = "http://m.bom.gov.au/vic/melbourne/"
    req = urllib.request.urlopen(url)
    page = req.read()
    scraping = BeautifulSoup(page)
    # Time Recording
    DateTime=datetime.now().strftime("%I:%M")
    DateTime=DateTime.replace("AM","")
    DateTime=DateTime.replace("PM","")
    temperature.append(DateTime)
```

Figure 1.2.2 – Time Recording inside the Weather Interval Function



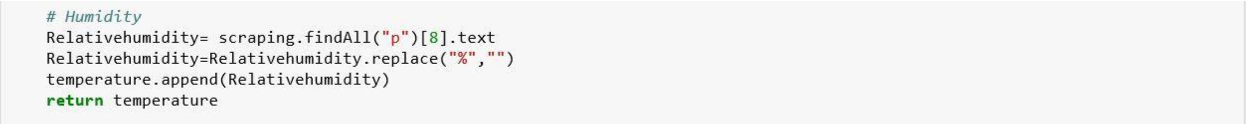
```
    # Temperature
    Temperature = scraping.findAll("div",attrs={"class":"current-temp"})[0].text
    ct = Temperature.replace("\n\t\t\t\t"," ")
    ct=ct.replace("°","")
    temp=float(ct)
    temperature.append(temp)
```

Figure 1.2.3 – Code for Recording the Current Temperature from the Website



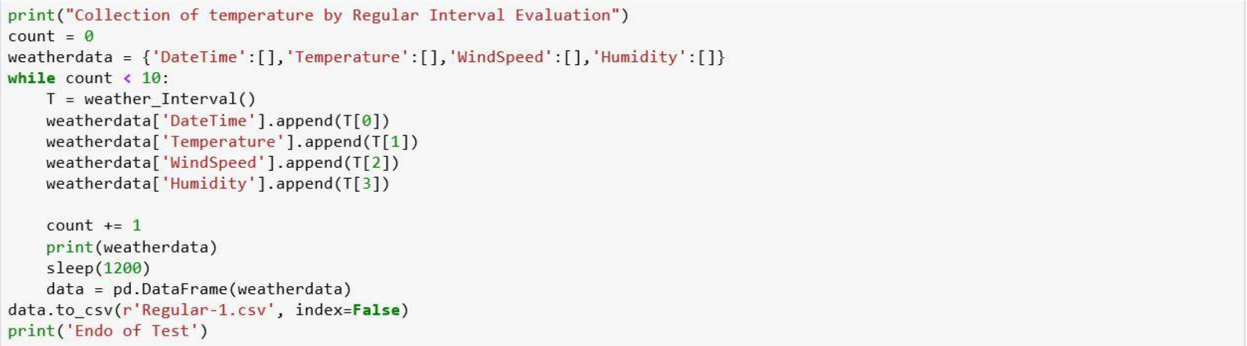
```
    # WindSpeed
    WindSpeed = scraping.findAll("p",attrs={"class":"wind-spd"})[0].text
    ct = WindSpeed.replace(" km/h","")
    temp=float(ct)
    temperature.append(temp)
```

Figure 1.2.4-Code for Recording the Wind Speed



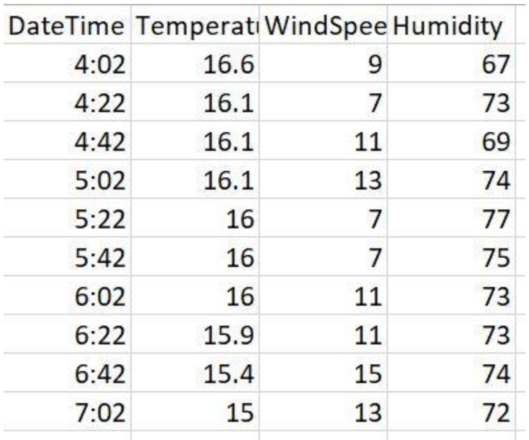
```
    # Humidity
    Relativehumidity= scraping.findAll("p")[8].text
    Relativehumidity=Relativehumidity.replace("%","")
    temperature.append(Relativehumidity)
    return temperature
```

Figure 1.2.5 – Code for Recording the Humidity



```
print("Collection of temperature by Regular Interval Evaluation")
count = 0
weatherdata = {'DateTime':[],'Temperature':[],'WindSpeed':[],'Humidity':[]}
while count < 10:
    T = weather_Interval()
    weatherdata['DateTime'].append(T[0])
    weatherdata['Temperature'].append(T[1])
    weatherdata['WindSpeed'].append(T[2])
    weatherdata['Humidity'].append(T[3])

    count += 1
    print(weatherdata)
    sleep(1200)
    data = pd.DataFrame(weatherdata)
data.to_csv(r'Regular-1.csv', index=False)
print('Endo of Test')
```

Figure 1.2.6 – All values were Recorded for interval of every 20 Minutes for 10 times



DateTime	Temperat	WindSpee	Humidity
4:02	16.6	9	67
4:22	16.1	7	73
4:42	16.1	11	69
5:02	16.1	13	74
5:22	16	7	77
5:42	16	7	75
6:02	16	11	73
6:22	15.9	11	73
6:42	15.4	15	74
7:02	15	13	72

Figure 1.3

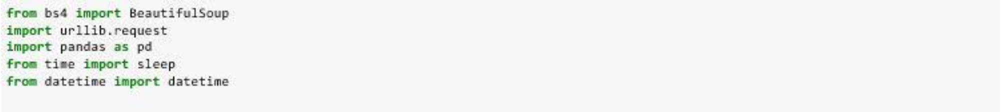
Discrete Interval

The *Figure 2.1* shows the values obtained from the running the Discrete Interval Code (show on *Figure 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.6)* on the website http://m.bom.gov.au/vic/melbourne/. These Records were taken if there was a change in temperature of 1 °C and once the values were obtained, they were put into the CSV named as Discreete.csv as shown on the *Figure 2.3*. Approximately it took around 12.5 hours to collect 10 samples of data.

Note: The following time is mentioned in AM.

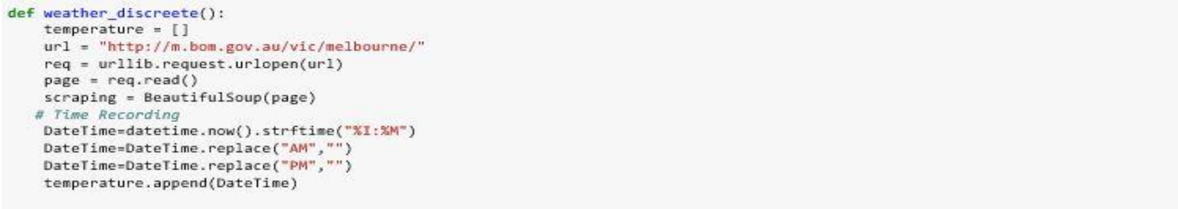
Current Time	**Temperature**	**Wind Speed**	**Humidity**
1:00	8.1	13	30
2:20	9.1	5	20
3:30	10.1	7	10
4:35	11.1	11	10
5:20	13.1	7	5
7:50	14.1	6	5
9:10	15.1	4	10
10:30	16.1	2	20
11:40	17.1	5	30
12:50	18.1	8	30

Figure 2.1



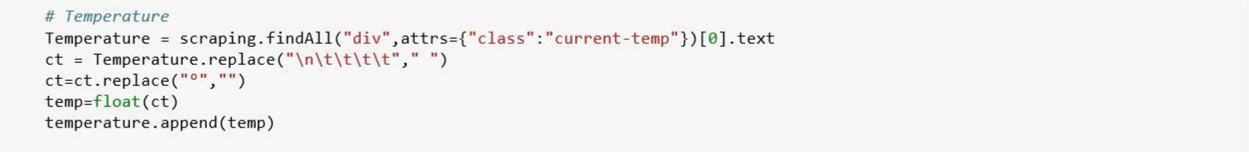
```
from bs4 import BeautifulSoup
import urllib.request
import pandas as pd
from time import sleep
from datetime import datetime
```

Figure 2.2.1 – The Libraries required to run the program



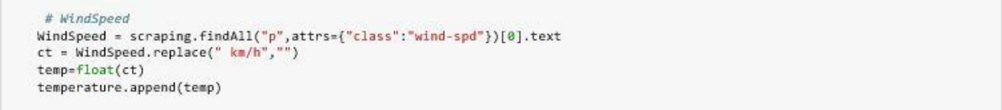
```
def weather_discreete():
    temperature = []
    url = "http://m.bom.gov.au/vic/melbourne/"
    req = urllib.request.urlopen(url)
    page = req.read()
    scraping = BeautifulSoup(page)
   # Time Recording
    DateTime=datetime.now().strftime("%I:%M")
    DateTime=DateTime.replace("AM","")
    DateTime=DateTime.replace("PM","")
    temperature.append(DateTime)
```

Figure 2.2.2 – Time Recording inside the Weather Discrete Function



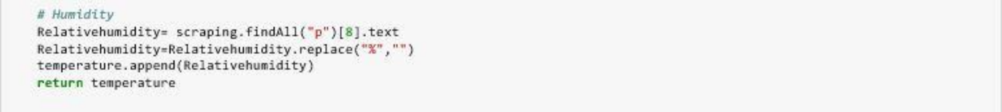
```
    # Temperature
    Temperature = scraping.findAll("div",attrs={"class":"current-temp"})[0].text
    ct = Temperature.replace("\n\t\t\t\t"," ")
    ct=ct.replace("°","")
    temp=float(ct)
    temperature.append(temp)
```

Figure 2.2.3 – Code for Recording the Current Temperature from the Website



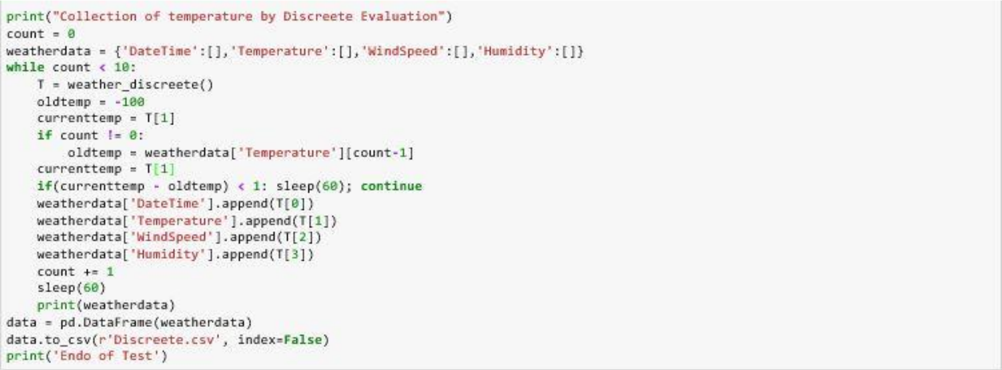
```
     # WindSpeed
    WindSpeed = scraping.findAll("p",attrs={"class":"wind-spd"})[0].text
    ct = WindSpeed.replace(" km/h","")
    temp=float(ct)
    temperature.append(temp)
```

Figure 2.2.4-Code for Recording the Wind Speed



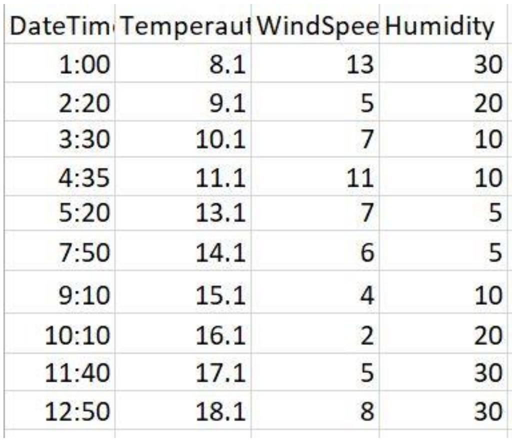
```
    # Humidity
    Relativehumidity= scraping.findAll("p")[8].text
    Relativehumidity=Relativehumidity.replace("%","")
    temperature.append(Relativehumidity)
    return temperature
```

Figure 2.2.5 – Code for Recording the Humidity



```
print("Collection of temperature by Discreete Evaluation")
count = 0
weatherdata = {'DateTime':[],'Temperature':[],'WindSpeed':[],'Humidity':[]}
while count < 10:
    T = weather_discreete()
    oldtemp = -100
    currenttemp = T[1]
    if count != 0:
        oldtemp = weatherdata['Temperature'][count-1]
    currenttemp = T[1]
    if(currenttemp - oldtemp) < 1: sleep(60); continue
    weatherdata['DateTime'].append(T[0])
    weatherdata['Temperature'].append(T[1])
    weatherdata['WindSpeed'].append(T[2])
    weatherdata['Humidity'].append(T[3])
    count += 1
    sleep(60)
    print(weatherdata)
data = pd.DataFrame(weatherdata)
data.to_csv(r'Discreete.csv', index=False)
print('Endo of Test')
```

Figure 2.2.6 – All values were Recorded for if there was a change of 1 Degree Celsius for 10 times



DateTim	Temperaut	WindSpee	Humidity
1:00	8.1	13	30
2:20	9.1	5	20
3:30	10.1	7	10
4:35	11.1	11	10
5:20	13.1	7	5
7:50	14.1	6	5
9:10	15.1	4	10
10:10	16.1	2	20
11:40	17.1	5	30
12:50	18.1	8	30

Figure 2.3

Conclusion

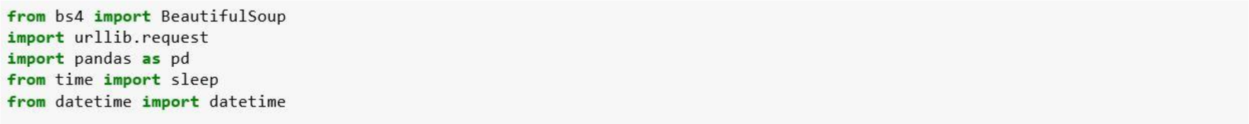
After taking the data through Regular Interval Sampling and Discrete Interval Sample, I believe that the Regular Interval Sampling is optimal than Discrete Interval for http://m.bom.gov.au/vic/melbourne/. It is because of the following analysis.

- It took me around 12.5 hours just to collect the Discrete Sample Data. That's makes the average 1-2 hours just to collect the 1 degree difference in data. Ultimately, wasting valuable energy and processing power.
- Whereas Regular Interval can provide number of samples which could be used for better analysis.

- Larger sample set in short time, would provide precise mean and also allows researchers to pinpoint outliers more easily ultimately providing better predictions with high accuracy.

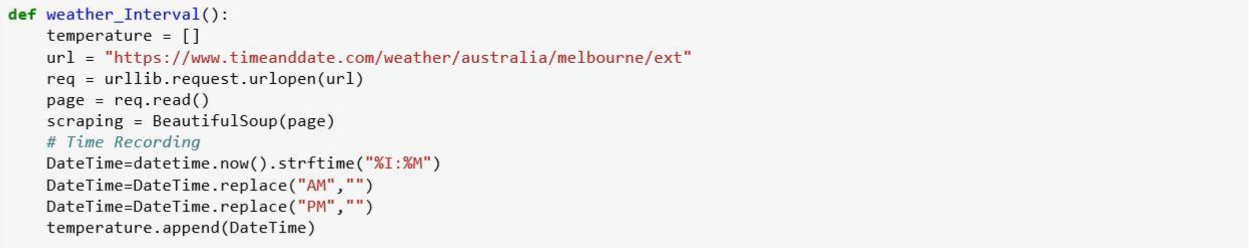
Schema Alignment

For Schema Alignment, I took the second website as https://www.timeanddate.com/weather/australia/melbourne/ext. As I considered Regular Interval to be the optimal one hence the following code shown on *Figure 3.1* (along with 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5) was used to collect the data which was saved in CSV file as shown on Figure 3.2. Once the data was taken, a schema was established between the http://m.bom.gov.au/vic/melbourne/ and https://www.timeanddate.com/weather/australia/melbourne/ext. Both of the website consisted of Temperature and Humidity so these elements were taken as "prominent member for establishing a Schema". I was already having the dataset for http://m.bom.gov.au/vic/melbourne/ as shown on the Regular Interval Part in the *Figure 1.3*.Through the use of the code as shown on *Figure 3.3*, the schema was aligned to provide the following results as shown on the *Figure3.4*.



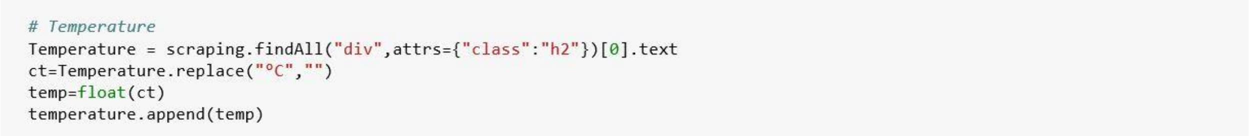
```
from bs4 import BeautifulSoup
import urllib.request
import pandas as pd
from time import sleep
from datetime import datetime
```

Figure 3.1.1 – The Libraries required to run the program



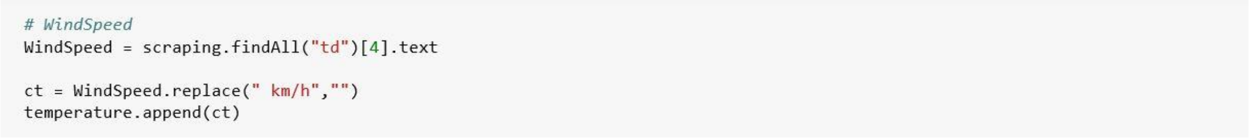
```
def weather_Interval():
    temperature = []
    url = "https://www.timeanddate.com/weather/australia/melbourne/ext"
    req = urllib.request.urlopen(url)
    page = req.read()
    scraping = BeautifulSoup(page)
    # Time Recording
    DateTime=datetime.now().strftime("%I:%M")
    DateTime=DateTime.replace("AM","")
    DateTime=DateTime.replace("PM","")
    temperature.append(DateTime)
```

Figure 3.1.2*– Time Recording inside the Weather Interval Function*



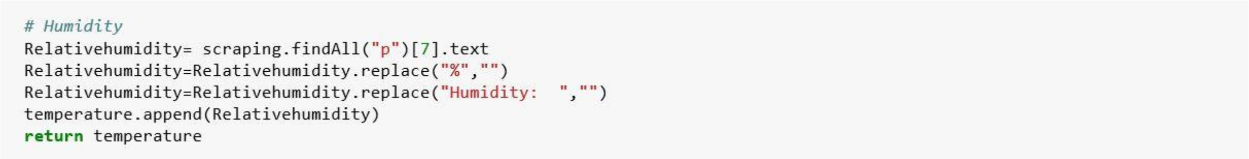
```
    # Temperature
    Temperature = scraping.findAll("div",attrs={"class":"h2"})[0].text
    ct=Temperature.replace("°C","")
    temp=float(ct)
    temperature.append(temp)
```

Figure 3.1.3-Code for Recording the Current Temperature from the Website



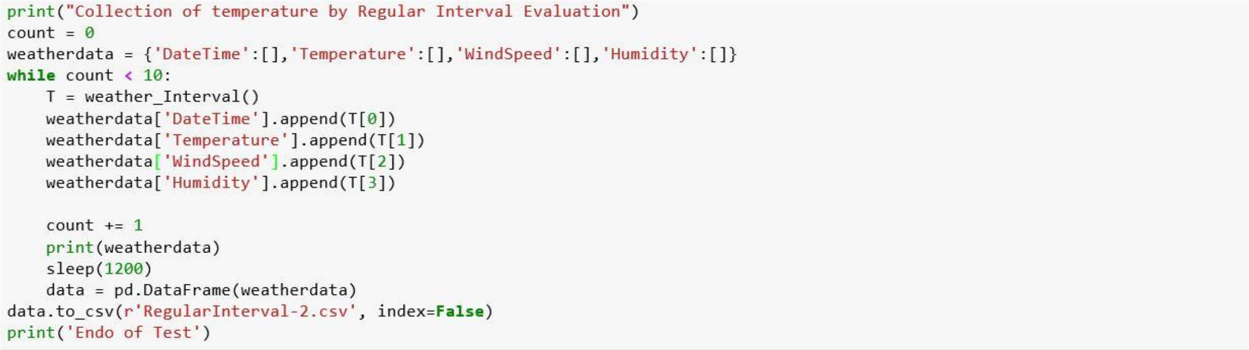
```
# WindSpeed
WindSpeed = scraping.findAll("td")[4].text

ct = WindSpeed.replace(" km/h","")
temperature.append(ct)
```

Figure 3.1.4-Code for Recording the Wind Speed



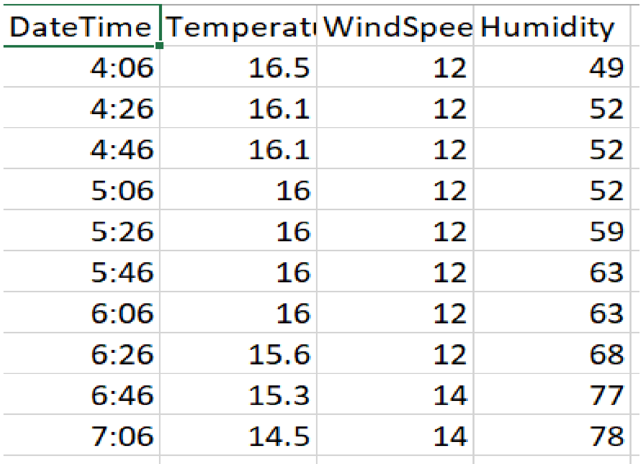
```
# Humidity
Relativehumidity= scraping.findAll("p")[7].text
Relativehumidity=Relativehumidity.replace("%","")
Relativehumidity=Relativehumidity.replace("Humidity:  ","")
temperature.append(Relativehumidity)
return temperature
```

Figure 3.1.5– Code for Recording the Humidity



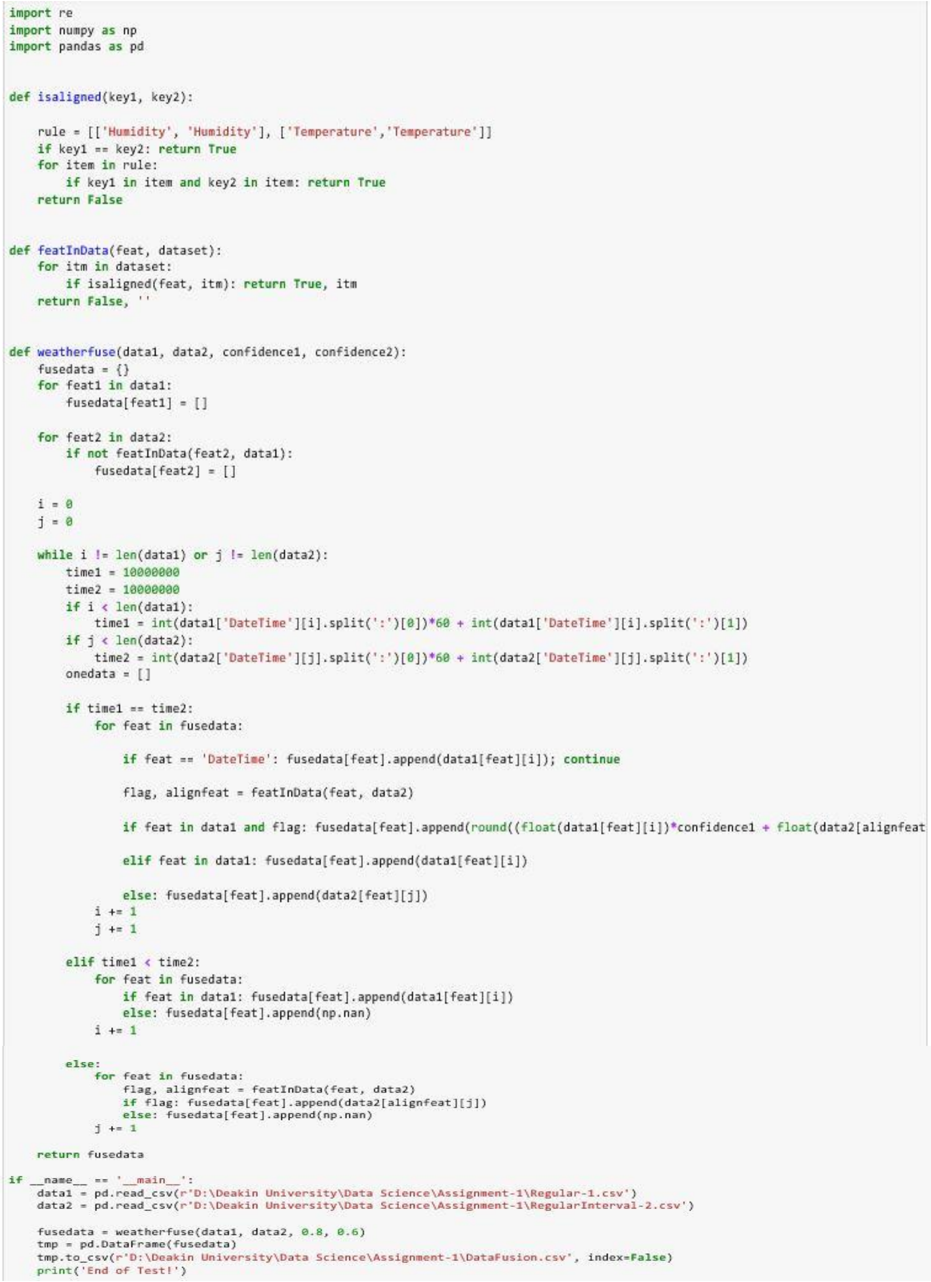
```
print("Collection of temperature by Regular Interval Evaluation")
count = 0
weatherdata = {'DateTime':[],'Temperature':[],'WindSpeed':[],'Humidity':[]}
while count < 10:
    T = weather_Interval()
    weatherdata['DateTime'].append(T[0])
    weatherdata['Temperature'].append(T[1])
    weatherdata['WindSpeed'].append(T[2])
    weatherdata['Humidity'].append(T[3])

    count += 1
    print(weatherdata)
    sleep(1200)
    data = pd.DataFrame(weatherdata)
data.to_csv(r'RegularInterval-2.csv', index=False)
print('Endo of Test')
```

Figure 3.1.6 – All values were Recorded for interval of every 20 Minutes for 10 times



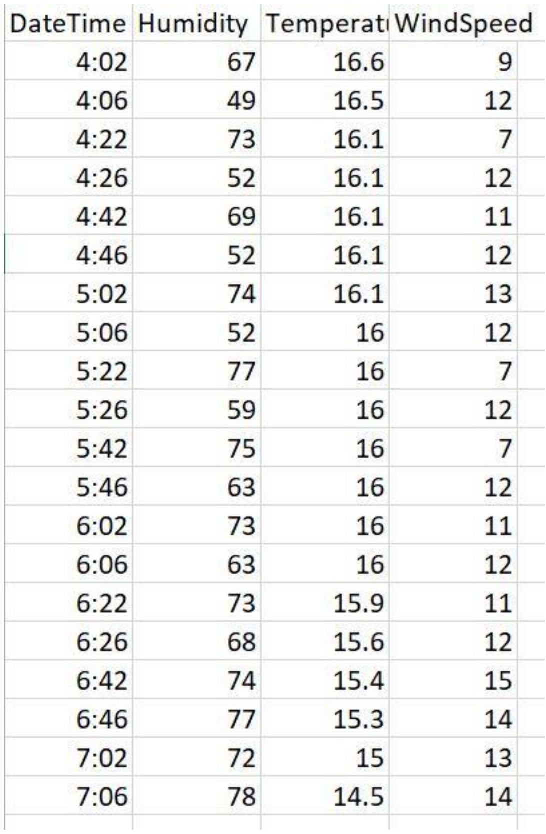
DateTime	Temperat	WindSpee	Humidity
4:06	16.5	12	49
4:26	16.1	12	52
4:46	16.1	12	52
5:06	16	12	52
5:26	16	12	59
5:46	16	12	63
6:06	16	12	63
6:26	15.6	12	68
6:46	15.3	14	77
7:06	14.5	14	78

Figure 3.2



```python
import re
import numpy as np
import pandas as pd


def isaligned(key1, key2):

    rule = [['Humidity', 'Humidity'], ['Temperature','Temperature']]
    if key1 == key2: return True
    for item in rule:
        if key1 in item and key2 in item: return True
    return False


def featInData(feat, dataset):
    for itm in dataset:
        if isaligned(feat, itm): return True, itm
    return False, ''


def weatherfuse(data1, data2, confidence1, confidence2):
    fusedata = {}
    for feat1 in data1:
        fusedata[feat1] = []

    for feat2 in data2:
        if not featInData(feat2, data1):
            fusedata[feat2] = []

    i = 0
    j = 0

    while i != len(data1) or j != len(data2):
        time1 = 10000000
        time2 = 10000000
        if i < len(data1):
            time1 = int(data1['DateTime'][i].split(':')[0])*60 + int(data1['DateTime'][i].split(':')[1])
        if j < len(data2):
            time2 = int(data2['DateTime'][j].split(':')[0])*60 + int(data2['DateTime'][j].split(':')[1])
        onedata = []

        if time1 == time2:
            for feat in fusedata:

                if feat == 'DateTime': fusedata[feat].append(data1[feat][i]); continue

                flag, alignfeat = featInData(feat, data2)

                if feat in data1 and flag: fusedata[feat].append(round((float(data1[feat][i])*confidence1 + float(data2[alignfeat

                elif feat in data1: fusedata[feat].append(data1[feat][i])

                else: fusedata[feat].append(data2[feat][j])
            i += 1
            j += 1

        elif time1 < time2:
            for feat in fusedata:
                if feat in data1: fusedata[feat].append(data1[feat][i])
                else: fusedata[feat].append(np.nan)
            i += 1

        else:
            for feat in fusedata:
                flag, alignfeat = featInData(feat, data2)
                if flag: fusedata[feat].append(data2[alignfeat][j])
                else: fusedata[feat].append(np.nan)
            j += 1

    return fusedata

if __name__ == '__main__':
    data1 = pd.read_csv(r'D:\Deakin University\Data Science\Assignment-1\Regular-1.csv')
    data2 = pd.read_csv(r'D:\Deakin University\Data Science\Assignment-1\RegularInterval-2.csv')

    fusedata = weatherfuse(data1, data2, 0.8, 0.6)
    tmp = pd.DataFrame(fusedata)
    tmp.to_csv(r'D:\Deakin University\Data Science\Assignment-1\DataFusion.csv', index=False)
    print('End of Test!')
```

Figure 3.3

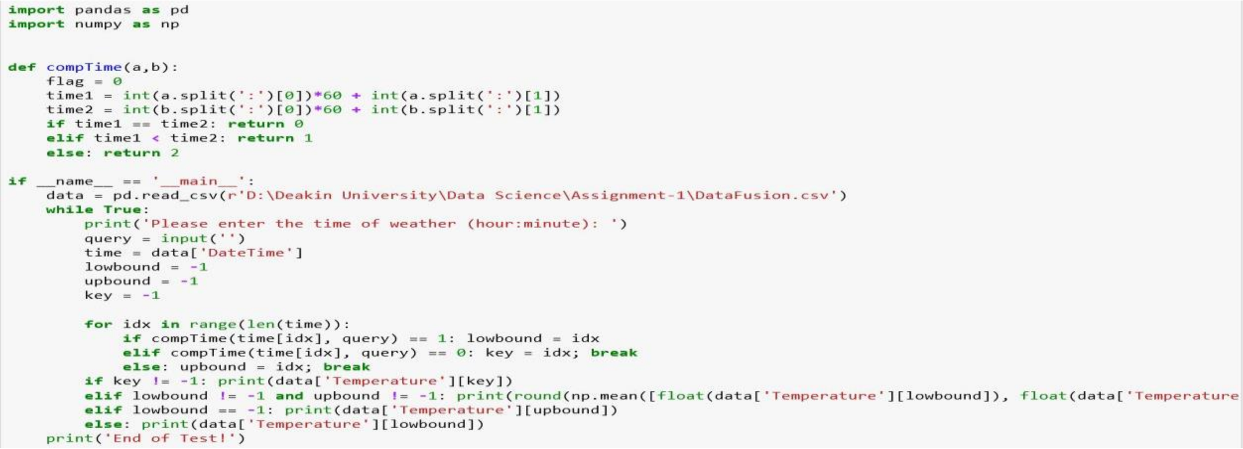


DateTime	Humidity	Temperat	WindSpeed
4:02	67	16.6	9
4:06	49	16.5	12
4:22	73	16.1	7
4:26	52	16.1	12
4:42	69	16.1	11
4:46	52	16.1	12
5:02	74	16.1	13
5:06	52	16	12
5:22	77	16	7
5:26	59	16	12
5:42	75	16	7
5:46	63	16	12
6:02	73	16	11
6:06	63	16	12
6:22	73	15.9	11
6:26	68	15.6	12
6:42	74	15.4	15
6:46	77	15.3	14
7:02	72	15	13
7:06	78	14.5	14

Figure3.4

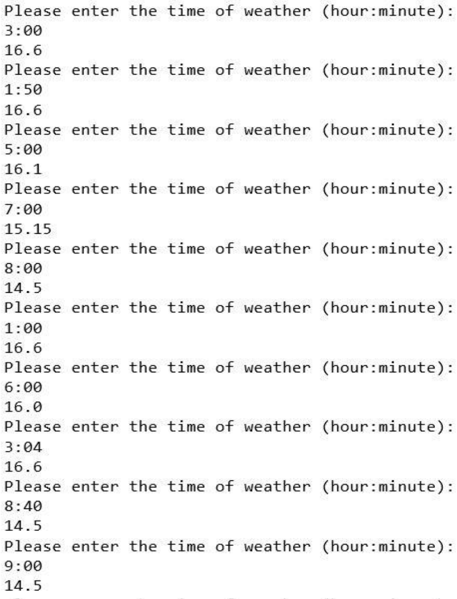
Weather Prediction

The values which were gained through the fusion would be used for the prediction, the following code on *Figure 4.1* could be used to achieve the result. 10 prediction were done and their results are below on *Figure 4.2*. The time is in PM, hence when the time is added along with hours: minute format, the prediction would be done.



```
import pandas as pd
import numpy as np


def compTime(a,b):
    flag = 0
    time1 = int(a.split(':')[0])*60 + int(a.split(':')[1])
    time2 = int(b.split(':')[0])*60 + int(b.split(':')[1])
    if time1 == time2: return 0
    elif time1 < time2: return 1
    else: return 2

if __name__ == '__main__':
    data = pd.read_csv(r'D:\Deakin University\Data Science\Assignment-1\DataFusion.csv')
    while True:
        print('Please enter the time of weather (hour:minute): ')
        query = input('')
        time = data['DateTime']
        lowbound = -1
        upbound = -1
        key = -1

        for idx in range(len(time)):
            if compTime(time[idx], query) == 1: lowbound = idx
            elif compTime(time[idx], query) == 0: key = idx; break
            else: upbound = idx; break
        if key != -1: print(data['Temperature'][key])
        elif lowbound != -1 and upbound != -1: print(round(np.mean([float(data['Temperature'][lowbound]), float(data['Temperature
        elif lowbound == -1: print(data['Temperature'][upbound])
        else: print(data['Temperature'][lowbound])
    print('End of Test!')
```

Figure 4.1-Weather Prediction Code



```
Please enter the time of weather (hour:minute):
3:00
16.6
Please enter the time of weather (hour:minute):
1:50
16.6
Please enter the time of weather (hour:minute):
5:00
16.1
Please enter the time of weather (hour:minute):
7:00
15.15
Please enter the time of weather (hour:minute):
8:00
14.5
Please enter the time of weather (hour:minute):
1:00
16.6
Please enter the time of weather (hour:minute):
6:00
16.0
Please enter the time of weather (hour:minute):
3:04
16.6
Please enter the time of weather (hour:minute):
8:40
14.5
Please enter the time of weather (hour:minute):
9:00
14.5
```

Figure 4.2- Prediction Results