# Met Office EMC Report

Frances Britton, Orson Hart, Olivia Knight, Tim Moore, Eve Rimmer

## Abstract

We have been given a large amount of data from weather stations across the globe. Our task set to us by the MET Office is to process and analyse this data in order to identify and amend erroneous data readings. We accomplished this by researching and devising methods for completing the task, programming these methods in languages such as Python, R, and Excel, and eventually bringing together all the constituent components of the project in a way so that they all link seamlessly. This approach made effective use of our team size and time-frame that we were given to complete the project, and allowed us to create a prototype system for separating weather station files and analysing air temperature data, that could be expanded upon to work quickly and easily for even larger amounts of data.

## Introduction

The task given was to find a way to identify errors in a data set, so that the anomalous data could be corrected or discarded.

The data set is weather and climate data from thousands of stations from all over the world, spanning decades. Many separate data stations were merged together into the same file, often allocated the same ID, this made reading and analysing data difficult. As the data is a mix of manually inputted and automatically inputted values, there are sometimes errors in the data. For example: if no data could be read, a -999 value will be inputted. If it's manually inputted, then human error can be at fault - for example, if a negative sign is missed or a decimal point is misplaced.

We initially tried looking at the temperature, as we thought that this would be a simple variable to analyse. However, we realised that the data was periodic, and so had to expand our knowledge of identifying anomalies. We then decided to focus on the locations of the stations. To do this, we had to separate the weather station files, to refine all the raw data down into the core components that we needed; the station ID, its recorded longitude and latitude, and the temperature data. We devised a way to see when a station had moved a certain distance using the Haversine Formula. We deemed that the move was significant enough if the city nearest to the weather station had changed. This then gave us a list of stations whose locations had drastically changed. We wanted to graphically represent this, so we created a virtual map with all of the stations that we flagged up on it.

We chose this task because we share an interest in data science, are interested in working with the Met Office, and we wanted to solve real life problems. This also gave us an opportunity to develop both our programming skills, and the skills that come with working in a team on a large-scale project.

# Methodology

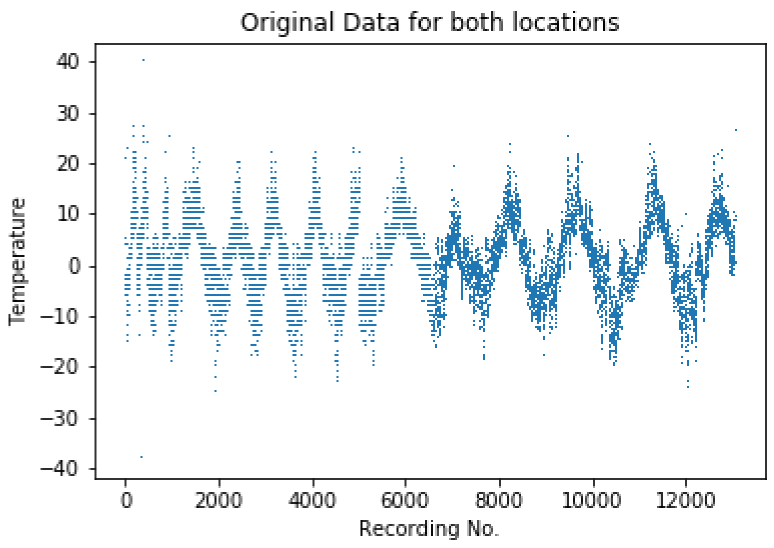
## Temperature analysis

We chose temperature as a variable to analyse because the temperature of one reading should be related to adjacent readings, resulting in a pattern that could be easily analysed. Temperature also follows a consistent pattern over the span of a year, as opposed to a more ‘random’ data point, such as *Wind Speed.*

Initially, we tried to identify anomalous results by taking results which were two standard deviations away from the overall mean. This identified that around 5% of data points were anomalies, which is the expected result of a normal distribution, as 95% of values should be within 2 standard deviations. However, this did not help us as it would not be the case that 5% of the data points were erroneous (either incorrectly entered, or missing entirely).

We then attempted to analyse temperature data using a Linear Regression script - written in Python - which plots a straight line through the data and the distance of each data point from the line of best fit. However, this attempt was also unfruitful, as the data could not be accurately represented by a linear relationship.

When we plotted the temperature data against time [see below] we noticed that the data is evidently periodic, and so our present knowledge of how to detect outliers in relation to linear data relationships could not be used in the case of Air Temperature data. We would have to learn new methods of identifying data points which did not fit the expected pattern.



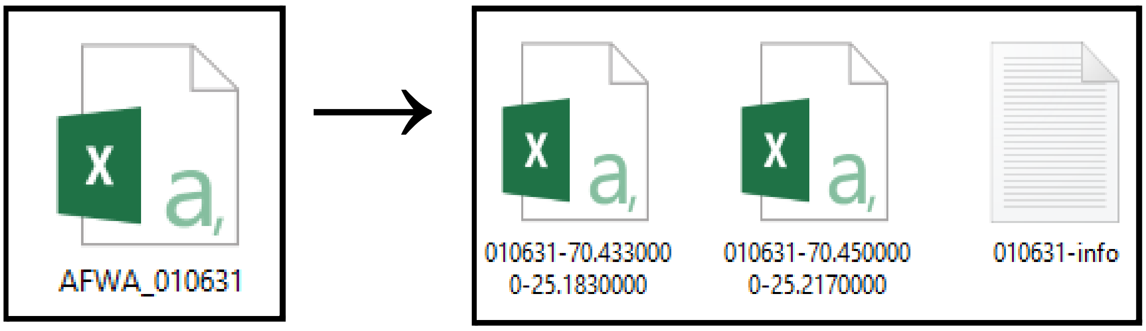
## Separation of Weather Station Files

In order to correctly analyse the temperature data from a weather station data file, it is paramount that all of the data came from the same location. Therefore, before data could be analysed, we decided to separate out a file into sub-files, where each sub-file has only one, unique, latitude longitude coordinate.

We decided to do this through a python script, and so had to learn how to handle CSV files (Comma Separated Variable files - the format in which the weather station data was stored) within this language. The script we produced would run through every csv file within a specified directory, and separate it into its constituent parts, as well as outputting an information file about the initial file.

The code for this script is below, and an example of the outputs is also included.

```
""" --- REQUIREMENTS ---
In same directory:
    CLE.py
    City Locations.txt
Required subdirectories:
    /csv
    /subfiles
    The input met office csv files go in /csv
    The output id-lat-long.csv files will end up in /subfiles
"""

# Used for runtime analysis.
import time
start_time = time.time()

import os
import CLE
import csv

# Constants used to refer to specific column positions.
ID = 0
LAT = 4
LONG = 5

# Function to declare if a string is a number.
def is_num(s):
    try:
        float(s)
        return True
    except:
        return False

# For each file in the directory
for filename in os.listdir("csv"):
    file_start_time = time.time()
    # Read all data from file into a list (might be slow with huge
files?).
    data = []
    f =  open("csv/" + filename,"r")
    for line in f.readlines():
        #CSV line into list
        currentLine = line.split(",")
        # Add list to data list
        data.append(currentLine)

    f.close()
    # Extract acceptable Latitude and Longitude data from the file.
    locs = []
    station_id = "-999"
    for row in range(len(data)):
        if is_num(data[row][0]):
            if is_num(data[row][LAT]) and is_num(data[row][LONG]):
                locs.append([data[row][LAT],data[row][LONG]])
                if station_id == "-999":
                    station_id = data[row][ID]

            # Create new subfile with id-lat-long
            curID = str(data[row][ID])
            curLat = str(data[row][LAT])
            curLong = str(data[row][LONG])
            subfilename = curID + "-" + curLat + "-" + curLong
            with open("subfiles/"+subfilename+".csv", mode='a', newline
="") as subfile:
                subfile_writer = csv.writer(subfile, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)
                subfile_writer.writerow(data[row])

    # Get the number of unique locations.
    # Get the number of changes between locations.
    uniquelocs = []
    changes = 0
    for i in range(len(locs)):
        if locs[i] not in uniquelocs:
            [uniquelocs.append(locs[i]), 1]
        if i > 1 and locs[i] != locs[i-1]:
            changes += 1

    # Output info on station.
    infofile = open("subfiles/" + curID + "-info.txt","w")

    infofile.write("ID:"+station_id+"\n")
    infofile.write("Number of Readings:"+str(len(locs))+"\n")
    infofile.write("Number of Locations:"+str(len(uniquelocs))+"\n")
    infofile.write("Number of Changes:"+str(changes)+"\n")
    infofile.write("List of Locations:"+"\n")
    for loc in uniquelocs:
        curLat = str(loc[0])
        curLong = str(loc[1])
        curCount = str(locs.count(loc))
        curCity = str(CLE.closestcity(float(loc[0]),float(loc[1])))
        infofile.write("LAT:"+ curLat + " LONG:" + curLong + " COUNT:" +
curCount + " NEAREST CITY:" + curCity+"\n")
    infofile.write("")
    infofile.close()


    fileTime = time.time() - file_start_time
    print(curID + ": " + str(fileTime))

totalTime = time.time() - start_time
print("Total time: " + str(totalTime))
```



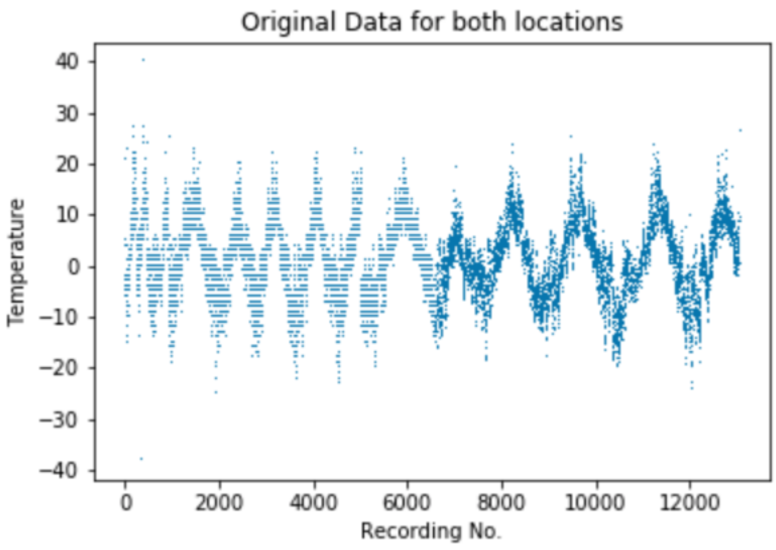
The format of the sub-files are “id-lat-long.txt”, and the format of the info-file is “id-info.txt”.
The sub-files contain all information contained within the original file, but only contain lines with the same latitude-longitude coordinates.

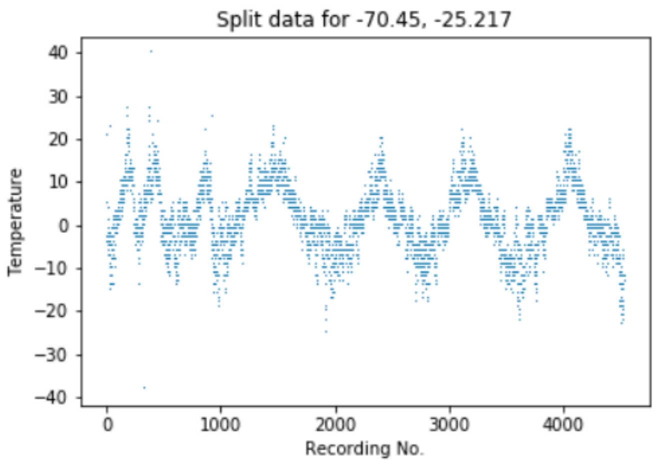
An example of the information contained within an info file is displayed below:

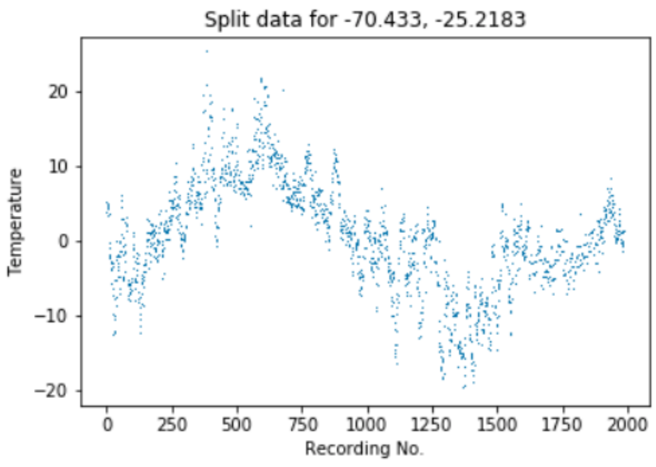
```
ID:010631
Number of Readings:13041
Number of Locations:2
Number of Changes:1
List of Locations:
LAT:70.4500000 LONG:25.2170000 COUNT:9069 NEAREST CITY:['Russia',
'Murmansk']
LAT:70.4330000 LONG:25.1830000 COUNT:3972 NEAREST CITY:['Russia',
'Murmansk']
```

Once a weather station data file has been separated into its sub-files, the sub-files can be analysed for patterns and to spot anomalies from within the data.

This is clearly shown when you compare the plotted data from the initial file to the plotted data for each of the sub-files.







Here it can be seen that in the initial file, there is a slight change in the pattern of the data when it is separated from one location to another, although this change would be more prevalent for a data set which has larger, and more frequent changes.

Once the data has been separated out, it should have a more consistent pattern which can be more easily analysed to identify and correct erroneous results.

## Weather Station Location Analysis

Within a single weather station data file, we found there were often multiple latitude and longitude pairs of coordinates, which we established could be due to three different reasons:

1. A different weather station had been assigned the same weather station ID, and so the two data files had been merged.
2. The station itself had actually moved, and so had a new pair of latitude and longitude coordinates
3. The station hadn't moved, but it's coordinates changed due to an error in the input, such as rounding to a different number of significant figures, or the omission of a minus sign.

We had to create solutions to each of the three problems, and in order to do this, we needed an essential piece of information - how far a station had moved.

As latitude and longitude are representations of points on the surface of a sphere, it was not possible to figure out the distance moved by only comparing the coordinates, as we needed to know the distance between two points as if the surface of the sphere was laid flat. Our research lead us to the haversine formula, a formula dating to 1805 often used within navigation. The formula takes two pairs of latitude longitude coordinates and the radius of the sphere on which they are plotted, and gives the shortest distance between them on the surface of the sphere, often referred to as the great-circle distance. The formula is as follows:

$$2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

$\varphi_1$, $\varphi_2$ - the latitude of point 1 and latitude of point 2
$\lambda_1$, $\lambda_2$ - the longitude of point 1 and longitude of point 2
r - the radius of the sphere.

This function is limited in part by the assumption that the Earth is a sphere, when in actuality it is an oblate spheroid. However, a sphere was a close enough model to the shape of the Earth to not cause any significant error in the output.

By programming this formula into a python function, we were able to pass it two values that we sourced from the spreadsheets and determine the distance of a move.

```python
from math import radians, cos, sin, asin, sqrt

def haversine(lon1, lat1, lon2, lat2):
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371
    return c * r
```

This was the first step towards determining what type of error had caused the different sets of latitude and longitude coordinates to appear within the same file. Our first approach was to pick some value, say 50km, and determine that if the distance of the move was below this value, then it was that the weather station itself had moved; if it were above this value, then it was due to multiple weather stations being assigned to the same id, and therefore being merged into the same file.

However, we decided to use a more advanced approach, as this method was limited by our lack of knowledge of the range of the size for a weather station move, and so we could not accurately pick a value to determine whether a move was legitimate or not. We also realised that a large move could sometimes be legitimate if it was in an area with little weather station coverage. For example, a move of 100km in a sparsely populated area such as Alaska could still be from the same weather station if it had been moved to get a more accurate sense for the weather in that area. Conversely, a move of 100km in England would not make sense, as there would be data from other weather stations nearby that could be used instead.

Therefore, we decided in order to separate a ‘minor’ move (a legitimate move of a weather station) from a ‘major’ move (an infeasible move that would be indicative of duplicate IDs), we would use the measure of the nearest major city. If the nearest major city had changed, then we could conclude that the weather station itself had moved too far for it to be a legitimate move.

We found a list of major cities online, with their respective latitude and longitude coordinates, and wrote a python script which, using the haversine formula, would find the nearest major city to a coordinate pair. Below is an example of the format of the list of city locations.

**City Locations.txt:**

```
"1";"Afghanistan";"Kabul";"34.5166667";"69.1833344";"1808.0"
"2";"Afghanistan";"Kandahar";"31.6100000";"65.6999969";"1015.0"
```

```
"3";"Afghanistan";"Mazar-e Sharif";"36.7069444";"67.1122208";"369.0"
"4";"Afghanistan";"Herat";"34.3400000";"62.1899986";"927.0"
"5";"Afghanistan";"Jalalabad";"34.4200000";"70.4499969";"573.0"
```

The python code for the function which calculates the nearest city is also displayed below:

```python
f = open("City Locations.txt", "r")
rawrows = [f.readline() for row in f]
f.close()
rows = [i.replace('"','').replace('\n','').split(";") for i in rawrows]
del rows[5283] # Due to an issue with the formatting of the text file.

def closestcity(lat, long):
    distances = []
    for city in rows:
        distances.append(haversine(long, lat, float(city[4]),
float(city[3])))
    smallest = [100000,100000]
    pos = -1
    for dis in distances:
        pos += 1
        if dis < smallest[0]:
            smallest[0] = dis
            smallest[1] = pos
    return(rows[smallest[1]][1:3])
```

This function would then be used to output the closest city to a latitude longitude pair when a weather station data file was separated out; if they were the same they could be recompiled together, and if they were different, kept seperate.

```
ID:025141
Number of Readings:92965
Number of Locations:3
Number of Changes:398
List of Locations:
LAT:57.7170000 LONG:11.7830000 COUNT:63984 NEAREST CITY:['Sweden',
'Moelndal']
LAT:57.7000000 LONG:11.7830000 COUNT:28804 NEAREST CITY:['Sweden',
'Moelndal']
LAT:58.2000000 LONG:12.3000000 COUNT:177 NEAREST CITY:['Sweden',
'Trollhattan']
```

This is an example of the information outputted when the data file for station ID 025141 was processed, and separated into three files for the three different latitude longitude coordinates within the original file.
As the nearest city to the first two locations is the same, (Moelndal, Sweden), then we decided they could be recompiled together for the data to be analysed, as it could be that this was a legitimately moved weather station. However, as the third location was closest to a different major city (Trollhattan, Sweden), we concluded it is more likely that it belongs to a different weather station altogether.

```
ID:684060
Number of Readings:44370
Number of Locations:3
Number of Changes:3
List of Locations:
LAT:28.5670000 LONG:16.5330000 COUNT:12622 NEAREST CITY:['Libya',
'Waddan']
LAT:-28.5670000 LONG:16.5330000 COUNT:31747 NEAREST CITY:['South
Africa', 'Atlantis']
LAT:-28.5700000 LONG:16.5300000 COUNT:1 NEAREST CITY:['South Africa',
'Atlantis']
```

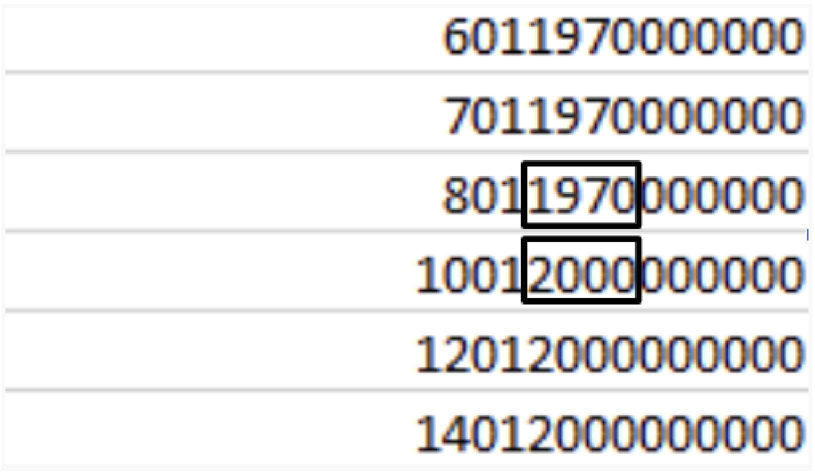
Another example of processed information above shows a case where we believe the error is a result of incorrect data entry, as the first two sets of coordinates are identical apart from the fact that one has a negative sign in front of the latitude coordinate, and the last set of coordinates is identical to the second except it is rounded to 2 decimal places instead of 3.

In this example, we could choose to combine the data aligned with the two latitude longitude coordinate pairs, as it is likely they come from the same station. We would, however, be unable to deduce the true location of the station. One method of figuring out the real location of the weather station consists of comparing the data from other weather stations nearby to the possible locations. If the temperature data from nearby stations to one location was very similar to the data from this file, then it could be concluded with a high probability of accuracy that this is the true location that the data had been collected from.

Due to time constraints, we were not able to develop any solutions to identify errors of this type automatically, but if given more time this would be a line of work we would want to pursue. For example, we could check if the absolute values of coordinates are the same, the values are the same when rounded to a certain number of decimal places, or if one coordinate is exactly one order of magnitude larger than the other (i.e. a misplaced decimal point).

## Date Errors

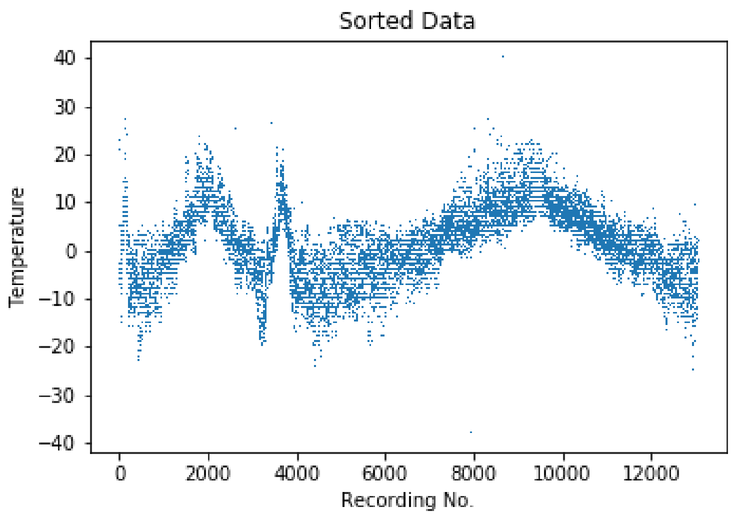
Another thing we noticed about some of the data files was errors in the actual recording of the date. We noticed that in some of the files, the date would periodically change between different years, like shown below:



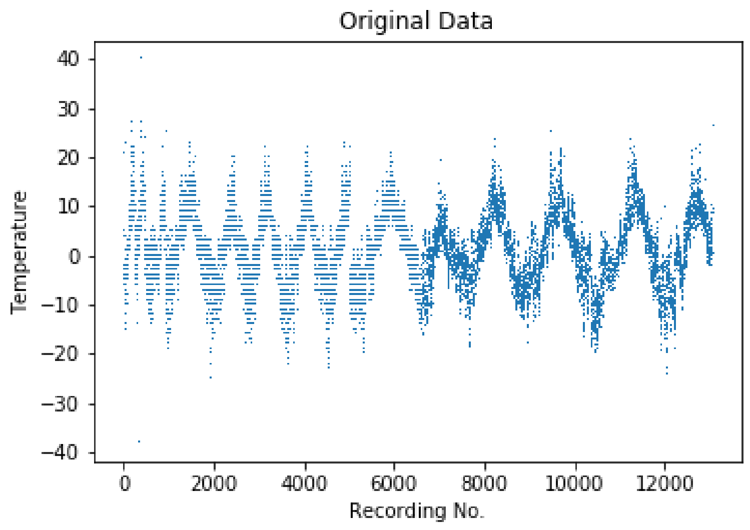
| |
|---|
| 6011970000000 |
| 7011970000000 |
| 801$\boxed{1970}$000000 |
| 1001$\boxed{2000}$000000 |
| 12012000000000 |
| 14012000000000 |

In this file, the date would jump from 1970 to 2000 every 10th day of the month, and then back to 1970 in the next month. We realised later in the project that this is due to Unix Time which is used by computers to measure the date. Any date which was not entered would have automatically have been set to 1970 due to this. However it is still interesting that this only happened every 10th day of the month.

At the start of the project though, didn't quite know what caused this time shift, and that maybe two files had been merged somehow. We decided to sort the data by date and look at something predictable like the temperature to determine whether the dates where actually chronological or if there was some sort of error. We used python to sort the data, which took us a little while due to the format of the date recordings, but when it was sorted, the air temperature looked like this:



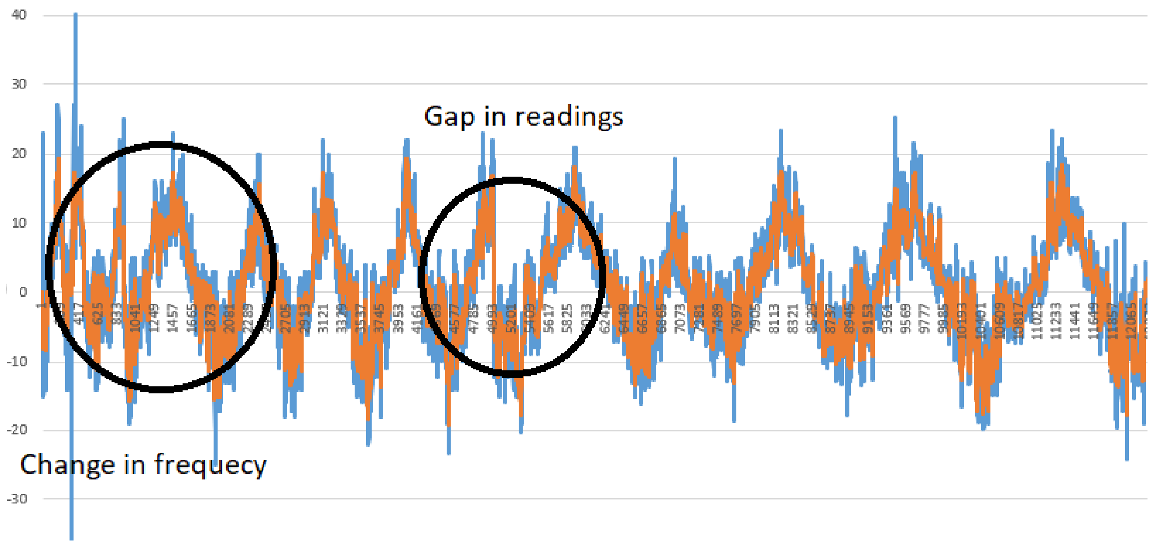
Compared to the normal, ‘unsorted’ data, this was much more chaotic (considering this data represents about 10 years) and showed us that the data was most likely chronological.



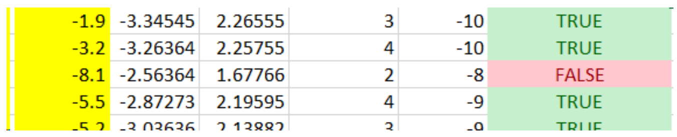
## Periodic Data Analysis

The first method we used to try and identify major anomalies in the air temperature recordings (e.g. accidentally recording a digit incorrectly which made it way too large, or small, or placeholder values like -999) was to use standard deviations and a moving average and look for patterns that we could possibly use to our advantage.

After looking at a graph of the temperature data (blue) plotted against the moving mean (orange), we noticed the datas period nature was much like that of a sine curve. Therefore we attempted to map a sine function to the data in R. However after several attempts and a fair bit of research we found this was not going to be a viable option within our time frame with our level of expertise. Part of this reason was due to the inconsistent nature of the gaps between the data readings. [highlighted in graph below]



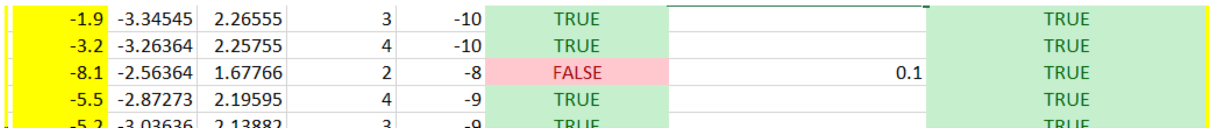
The next technique we came up with was to use the mean and standard deviation to create bounds in which we hoped anomalous data would fall outside of in order to help highlight possible errors. For this we used a moving average to ensure it changes at the same rate that the air temperature does throughout the seasons. Then we compared temperature recordings to those a certain distance either side of it (e.g. ± 5 recordings). A data point was to be considered possibly anomalous if it was within a set number of standard deviations from the mean. This method worked somewhat, however, even after trying to optimise the number of standard deviations used, we noticed that there were some data points identified as anomalies that didn't seem to be out of place. Example (temperature is -8 which doesn't seem anomalous):



| | | | | | |
|---|---|---|---|---|---|
| -1.9 | -3.34545 | 2.26555 | 3 | -10 | TRUE |
| -3.2 | -3.26364 | 2.25755 | 4 | -10 | TRUE |
| -8.1 | -2.56364 | 1.67766 | 2 | -8 | FALSE |
| -5.5 | -2.87273 | 2.19595 | 4 | -9 | TRUE |
| -5.2 | -3.03636 | 2.13882 | 3 | -9 | TRUE |

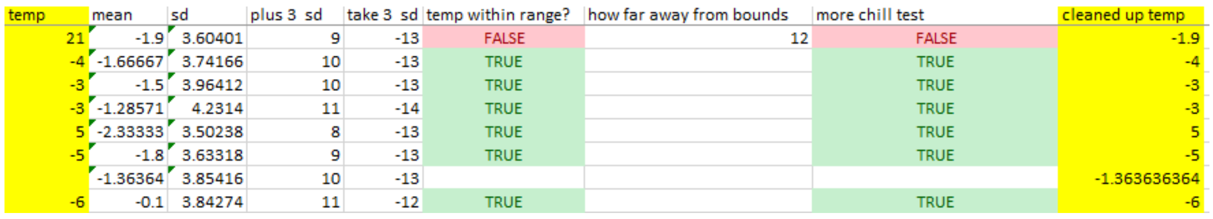
This seemed to happen most often when the data points either side were fairly close to each other as is brought down the standard deviation. Frequently this was due to all the readings in a section being taken at a similar time e.g. around noon except one which was taken during the night. This method only really worked with extreme anomalies and using a relatively high amount of standard deviations.

We created a second, less strict test which reads 'true' if the temperature is within a certain distance from the bounds of the local mean. At first this did not change much as the anomalies would drag the local mean up or down. This was fixed by excluding the temperature reading from being included in the calculation for the mean that would be used to create its own set of bounds. Now, as shown in the image, temperature that are only just out of the bounds won't be included as possibly anomalous (the exact value away can be changed).



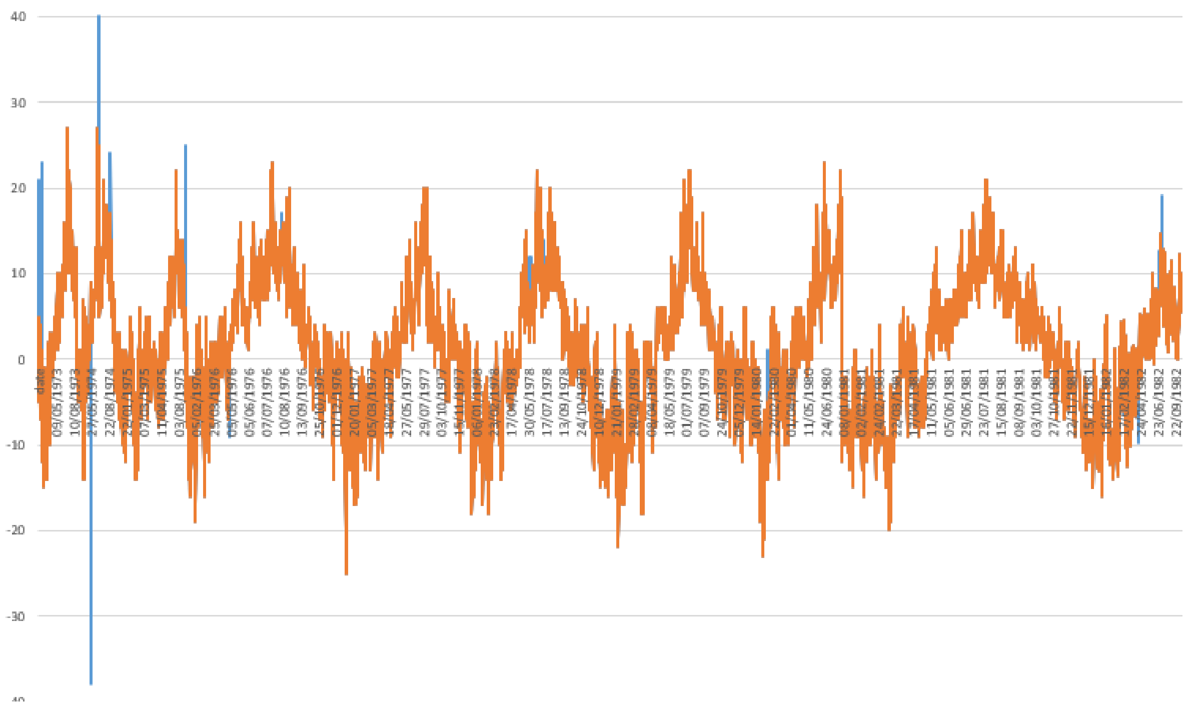
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -1.9 | -3.34545 | 2.26555 | 3 | -10 | TRUE | | TRUE |
| -3.2 | -3.26364 | 2.25755 | 4 | -10 | TRUE | | TRUE |
| -8.1 | -2.56364 | 1.67766 | 2 | -8 | FALSE | 0.1 | TRUE |
| -5.5 | -2.87273 | 2.19595 | 4 | -9 | TRUE | | TRUE |
| -5.2 | -3.03636 | 2.13882 | 3 | -9 | TRUE | | TRUE |

Finally we added a column in which blank readings and those considered to be errors are replaced with a value we believe may be more accurate. Currently it is the moving mean, however, with more time we would have liked to improve this by taking into account data from nearby weather stations and what error we thought was made, for example a misplaced decimal point.



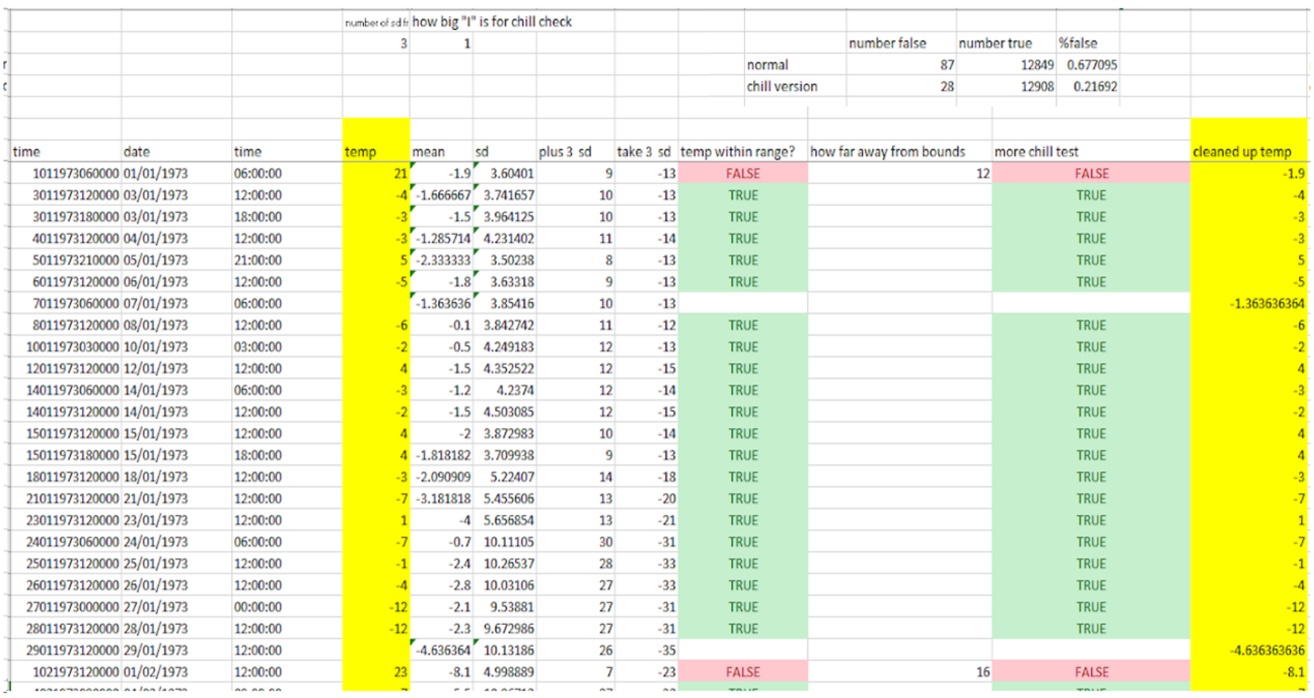
| temp | mean | sd | plus 3 sd | take 3 sd | temp within range? | how far away from bounds | more chill test | cleaned up temp |
|---|---|---|---|---|---|---|---|---|
| 21 | -1.9 | 3.60401 | 9 | -13 | FALSE | 12 | FALSE | -1.9 |
| -4 | -1.66667 | 3.74166 | 10 | -13 | TRUE | | TRUE | -4 |
| -3 | -1.5 | 3.96412 | 10 | -13 | TRUE | | TRUE | -3 |
| -3 | -1.28571 | 4.2314 | 11 | -14 | TRUE | | TRUE | -3 |
| 5 | -2.33333 | 3.50238 | 8 | -13 | TRUE | | TRUE | 5 |
| -5 | -1.8 | 3.63318 | 9 | -13 | TRUE | | TRUE | -5 |
| | -1.36364 | 3.85416 | 10 | -13 | | | | -1.363636364 |
| -6 | -0.1 | 3.84274 | 11 | -12 | TRUE | | TRUE | -6 |

From looking at the graph created using this data [shown below] you can see the possible anomalous temperature readings. (Blue is original temperature and orange is the prediction of what would have been more accurate).



This method of course may incorrectly identify many recordings as errors, where there have simply been unusual temperatures on a day, however, it does not seem to do this too frequently. Moreover with more time we would have wanted to be able to take the possible anomalies and compare them against the data from nearby weather stations. This would mean we could identify if it was just a particular hot/cold day in that location or if it was most likely an error. We had started this by finding out which weather stations are near each other but not yet developed a method of making a comparison.

A view of the whole excel file:



| number of sd fr | how big "I" is for chill check |
|---|---|
| 3 | 1 |

| | number false | number true | %false |
|---|---|---|---|
| normal | 87 | 12849 | 0.677095 |
| chill version | 28 | 12908 | 0.21692 |

| time | date | time | temp | mean | sd | plus 3 sd | take 3 sd | temp within range? | how far away from bounds | more chill test | cleaned up temp |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1011973060000 | 01/01/1973 | 06:00:00 | 21 | -1.9 | 3.60401 | 9 | -13 | FALSE | 12 | FALSE | -1.9 |
| 3011973120000 | 03/01/1973 | 12:00:00 | -4 | -1.666667 | 3.741657 | 10 | -13 | TRUE | | TRUE | -4 |
| 3011973180000 | 03/01/1973 | 18:00:00 | -3 | -1.5 | 3.964125 | 10 | -13 | TRUE | | TRUE | -3 |
| 4011973120000 | 04/01/1973 | 12:00:00 | -3 | -1.285714 | 4.231402 | 11 | -14 | TRUE | | TRUE | -3 |
| 5011973210000 | 05/01/1973 | 21:00:00 | 5 | -2.333333 | 3.50238 | 8 | -13 | TRUE | | TRUE | 5 |
| 6011973120000 | 06/01/1973 | 12:00:00 | -5 | -1.8 | 3.63318 | 9 | -13 | TRUE | | TRUE | -5 |
| 7011973060000 | 07/01/1973 | 06:00:00 | | -1.363636 | 3.85416 | 10 | -13 | | | | -1.363636364 |
| 8011973120000 | 08/01/1973 | 12:00:00 | -6 | -0.1 | 3.842742 | 11 | -12 | TRUE | | TRUE | -6 |
| 10011973030000 | 10/01/1973 | 03:00:00 | -2 | -0.5 | 4.249183 | 12 | -13 | TRUE | | TRUE | -2 |
| 12011973120000 | 12/01/1973 | 12:00:00 | 4 | -1.5 | 4.352522 | 12 | -15 | TRUE | | TRUE | 4 |
| 14011973060000 | 14/01/1973 | 06:00:00 | -3 | -1.2 | 4.2374 | 12 | -14 | TRUE | | TRUE | -3 |
| 14011973120000 | 14/01/1973 | 12:00:00 | -2 | -1.5 | 4.503085 | 12 | -15 | TRUE | | TRUE | -2 |
| 15011973120000 | 15/01/1973 | 12:00:00 | 4 | -2 | 3.872983 | 10 | -14 | TRUE | | TRUE | 4 |
| 15011973180000 | 15/01/1973 | 18:00:00 | 4 | -1.818182 | 3.709938 | 9 | -13 | TRUE | | TRUE | 4 |
| 18011973120000 | 18/01/1973 | 12:00:00 | -3 | -2.090909 | 5.22407 | 14 | -18 | TRUE | | TRUE | -3 |
| 21011973120000 | 21/01/1973 | 12:00:00 | -7 | -3.181818 | 5.455606 | 13 | -20 | TRUE | | TRUE | -7 |
| 23011973120000 | 23/01/1973 | 12:00:00 | 1 | -4 | 5.656854 | 13 | -21 | TRUE | | TRUE | 1 |
| 24011973060000 | 24/01/1973 | 06:00:00 | -7 | -0.7 | 10.11105 | 30 | -31 | TRUE | | TRUE | -7 |
| 25011973120000 | 25/01/1973 | 12:00:00 | -1 | -2.4 | 10.26537 | 28 | -33 | TRUE | | TRUE | -1 |
| 26011973120000 | 26/01/1973 | 12:00:00 | -4 | -2.8 | 10.03106 | 27 | -33 | TRUE | | TRUE | -4 |
| 27011973000000 | 27/01/1973 | 00:00:00 | -12 | -2.1 | 9.53881 | 27 | -31 | TRUE | | TRUE | -12 |
| 28011973120000 | 28/01/1973 | 12:00:00 | -12 | -2.3 | 9.672986 | 27 | -31 | TRUE | | TRUE | -12 |
| 29011973120000 | 29/01/1973 | 12:00:00 | | -4.636364 | 10.13186 | 26 | -35 | | | | -4.636363636 |
| 1021973120000 | 01/02/1973 | 12:00:00 | 23 | -8.1 | 4.998889 | 7 | -23 | FALSE | 16 | FALSE | -8.1 |

<u>Formulas used on excel:</u>

Rearranges date and time of reading to make easier to read

```
=IF(LEN(A19)=13,"0"&MID(A19,1,1)&"/"&MID(A19,2,2)&"/"&MID(A19,4,4),MID(A19,1,2)&"/"&MID(A19,3,2)&"/"&MID(A19,5,4))
```

```
=IF(LEN(A19)=13,MID(A19,8,2)&":"&MID(A19,10,2)&":"&MID(A19,12,2),MID(A19,9,2)&":"&MID(A19,11,2)&":"&MID(A19,13,2))
```

Finds average

```
=AVERAGE(D14:D18,D20:D25)
```

Finds standard deviation

```
=STDEV.S(D14:D18,D20:D25)
```

Creates bounds

```
=ROUND(E20+($D$2*F20),0)
```

```
=ROUND(E20-($D$2*F20),0)
```

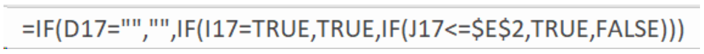
States whether within bounds

```
=IF(D18="","",((D18<=G18)&(D18>=H18))="truetrue")
```

If point is not within bounds states distance to closest one

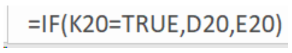
```
=IF(I20=FALSE,MIN(ABS(D20-G20),ABS(D20-H20)),"")
```

States whether within new bounds



```
=IF(D17="","",IF(I17=TRUE,TRUE,IF(J17<=$E$2,TRUE,FALSE)))
```

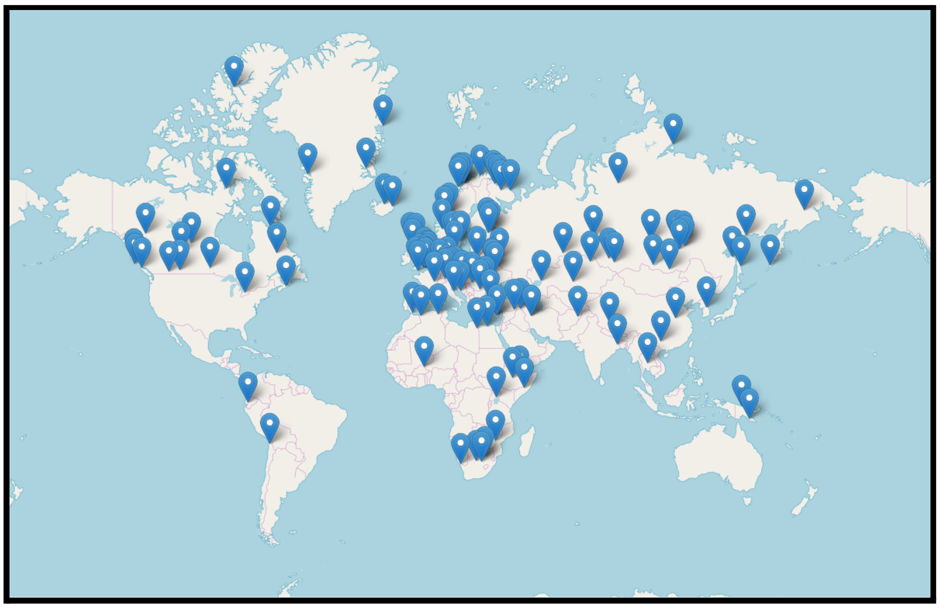
Replaces blank readings and possible errors with average temperature at that point

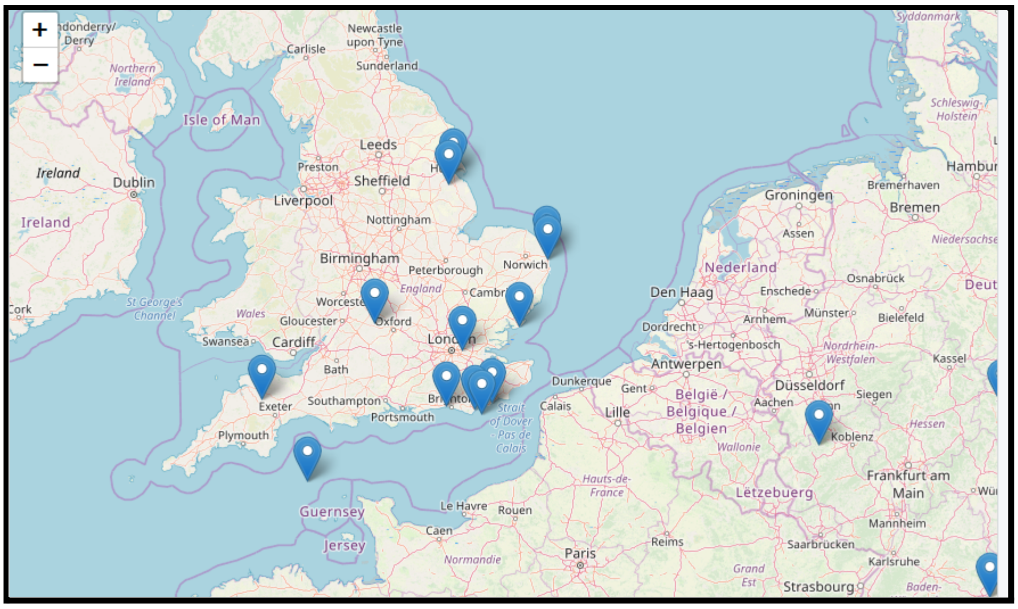


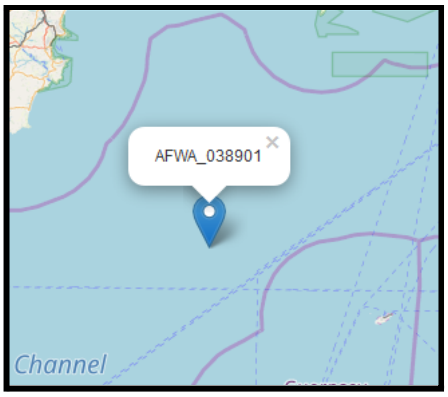
```
=IF(K20=TRUE,D20,E20)
```

## Visual Representation of Data

We also wanted to create a way to represent the weather stations visually, through some sort of interactive map, which would also allow us to identify incorrect latitude and longitude positions due to the fact that they are far out at sea. Originally we decided to use the R programming language and use some of its functions to plot the locations of the weather stations on a map. We used R because it can handle large data sets like the weather data we were given, and also comes with tools which were useful in plotting the locations of the weather stations.







These images show the final result of the data plot, and this helped us to analyse obviously erroneous latitude and longitude readings, such as the coordinates of station AWFA_038901 which has been placed in the middle of the ocean.

## Results

The results of our combined efforts were a method of separating out weather station data into groupings which we believe have a high likelihood of being taken from the same weather station location. Once the data is in this form, its temperature data can then be analysed through an excel spreadsheet which highlights anomalies based on a comparison to a rolling average of the data points either side of potential erroneous result. If the data point in question is too far from the average, and is therefore determined to be an anomaly, the spreadsheet is able to amend the data point by replacing it with the average of the data points either side of it.

Whilst the whole process is not automated, the majority is - with the only human requirement to copy data from the separate spreadsheets into the temperature analysis spreadsheet, and then manually export the results.

## Conclusion

In conclusion, our project was a success, as we were able to successfully and automatically, separate merged or incorrect weather station data from a large amount of files. Once the data was in a form such that it is separated by the location that the weather data had been read from, we could analyse and correct erroneous data readings, and also highlight and amend possible anomalous data points so that the overall pattern of the weather could be analysed further.

If we had more time, we would have wanted to streamline the entire process more, so that every piece of the project was seamlessly linked together, and you would only have to give an input of the weather station data in order to receive the amended and separated temperature data in a format which is easily processable.
We would also aim to increase the efficiency of our algorithms, as we did not focus on the speed that our code runs, and this could dramatically improve the time that the process takes, since file manipulation and separation takes a very long time using our current methods, especially if running on a laptop or computer which does not have powerful hardware.

We also wish to pursue other data points that could be analysed and amended, whether this be fields that are similar in nature to air temperature, such as dew point temperature, or develop new methods to analyse a wider range of data points that do not relate to temperature.

Nevertheless, the project was enjoyable to work on, taught us a lot of valuable skills, and gave us the opportunity to complete a genuine, large-scale task which had been set by an industry leader such as the MET office.

## Sources Used

- Met Office spidas documents https://www.metoffice.gov.uk/hadobs/spidas/
  (used to retrieve the initial weather station data)
- Stack Overflow https://stackoverflow.com/
  (used to debug python scripts and learn about handling csv files in python)
- Python documentation  https://docs.python.org/3/
  (used to debug python scripts)
- City Location Data https://github.com/tidwall/cities/blob/master/cities.go
  (used to find locations of major cities)
- Haversine Formula https://www.movable-type.co.uk/scripts/latlong.html
  (used to find distance between two points)
- Haversine Formula Background Research:
  https://books.google.com/books?id=0BCCz8Sx5wkC&pg=PR7
  (backs up claims such as the time of development and uses of haversine formula)
- Stuart: stuartallen@exetermathematicsschool.ac.uk
  (used as a contact point between the MET Office and our group, answered questions about statistical analysis, and helped with the development of the temperature analysis spreadsheet. We cannot thank him enough for the enormous help he has given with the project.)

.