B173513
2020-08-16

Part A

Data management is very important in data science. A good data management plan provides a formalized methodology to avoid all-too-common pitfalls surrounding data.

Having a data management plan limits the culture of ‘making it up as we go’, which often involves reinventing data formatting standard, leaving few breadcrumbs or ‘metadata’ behind and introduces a risk of completely losing the data if not stored properly.

If a given set of data is deemed important to a given individual or organisation, it would make sense to have rules in place that mean the data is kept safe. When considering how to store the data, the most pertinent concern is protection from hardware failures. This can mean uploading to remote servers (in the cloud or within organisation), simply backing up to another storage medium or upgrading the current storage medium with added redundancy (RAID / hdfs). There are numerous aspects to consider in making this decision and are not limited to:

- Persistency
    - Does the given cloud subscription run out? Does the storage medium start to die after a number of years? How long would you actually **want** the data to be ‘alive’?
        - This last point can tie into an organisations retention policy.
- Accessibility / Security
    - Which stakeholders should have access? Should it be openly available (research paper attached)? Will it be though a link, or user access? What level of authentication?
        - Quite a lot of infrastructure needed to just deal with access control (a security consideration)
    - Is encryption needed?
    - What happens when a given individual who had access leaves?
        - How do you ensure enough people will retain access.
- Frequency of access / Reliability
    - Perhaps sufficient to have magnetic tape backups to cover organisation legal requirement.
    - If many accesses are required, is the nessecary bandwidth / software infrastructure in place to handle this?
    - Is this a critical real-time data service (estimated bus times for example?)
- Cost
    - For any features that the above might necessitate – there will be a cost incurred. Is the ongoing cost reasonable?

You may also consider the storage efficiency. If storing lots of databases, potentially worth looking to see if there’s data redundancy that could be minimise with the use of a relational database.

A further consideration is on understandability. The data we produce should also have metadata created. If a given individual is not able to explain the data to ‘consumers’ of the data, then they should be able to sufficiently discern the meaning and intent of the data set through metadata sources. This also enhances a given organisation’s operational scalability.

Having a strict metadata format / scheme also helps in the ability to process the data programmatically. In the context of the web, providing an index or sitemap allows google to better discover pages. Similarly, a data-set mapping or index (preferably through a URL/URI/DOI scheme) enhances discoverability and potentially enables a much greater ability to 'link' disparate datasets together.

Capturing metadata can enable an organisation to be more dynamic and robust against change. A given data set might have it's importance 'upgraded' and perhaps there would be a desire to expose previous company private data through an API. If the standard of quality of the data is already excellent internal to the organisation, then this sort of shift becomes much easier.

Data management plans can also help to 'on-board' new individuals to an organisation/project as there is a 'tome' of reference that they can simply adhere to without inefficient discussions or painful lessons.

Part B

B.1

I first set up my project into data, results and scripts folders. I set out a plan to have a set of utility scripts to perform data gathering and some cleaning, then a main Jupyter notebook for each part of the assignment.

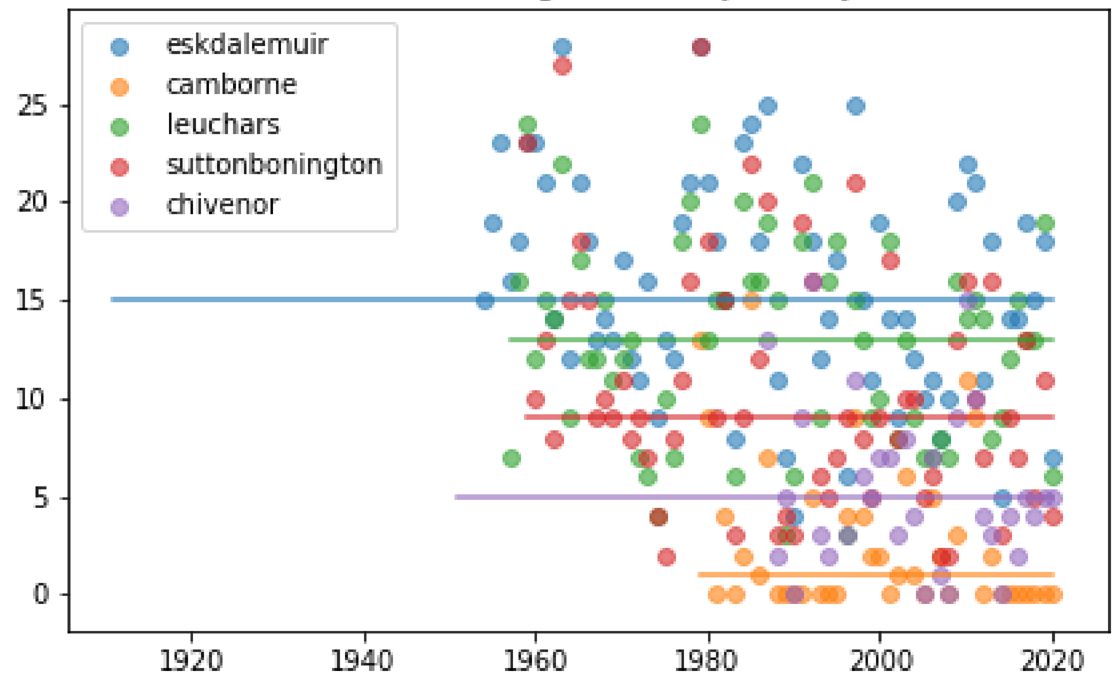
- A python script is used to simply download the weather data for all stations and store each station in it's own file. It takes the stations.txt file as an input and makes a http request to each stations data at the provided link location. It saves the content of those files 'raw' into data/weather/<station_name>.txt
    - https://www.metoffice.gov.uk/pub/data/weather/uk/climate/stationdata/'+station+'data.txt'

- Then a further python script is used to 'parse' the data, create a Pandas DataFrame which contains every station's data and save that to a number of file formats. The parsing of the data can be split up into two functions. Regular expressions are heavily used throughout.
    - getDataFrame. This finds the start of the table of data after the preamble and converts each column into a pandas DataFrame column. It then casts the year and month columns into integers and sets that as a MultiIndex for the dataframe. The remaining columns are cast to floats.
    - getDataExtras. This uses regular expressions to find the lines of text that have height above sea level (asml), grid references and Latitude, Longitude values. Where latitude values can't be found, it makes a website query and parses the reponse to convert from grid reference to longitude.
        - `http://www.nearby.org.uk/coord.cgi?p='+gridRef+'&f=conv'`

The cluster_weather Jupyter notebook is used to perform the clustering in this exercise. It takes the generated stationData.csv file and coverts it into the dataframe. It was clear that each station had a different range of yearly data, I took a small sample to investigate if I could accurately take the median of a given month-column over all years. In the figure below, the range of data for four stations is shown, with a horizontal line demonstrating the calculated mean for that station's month.

In this case, the air frost days are shown, but the importance wasn't the column.

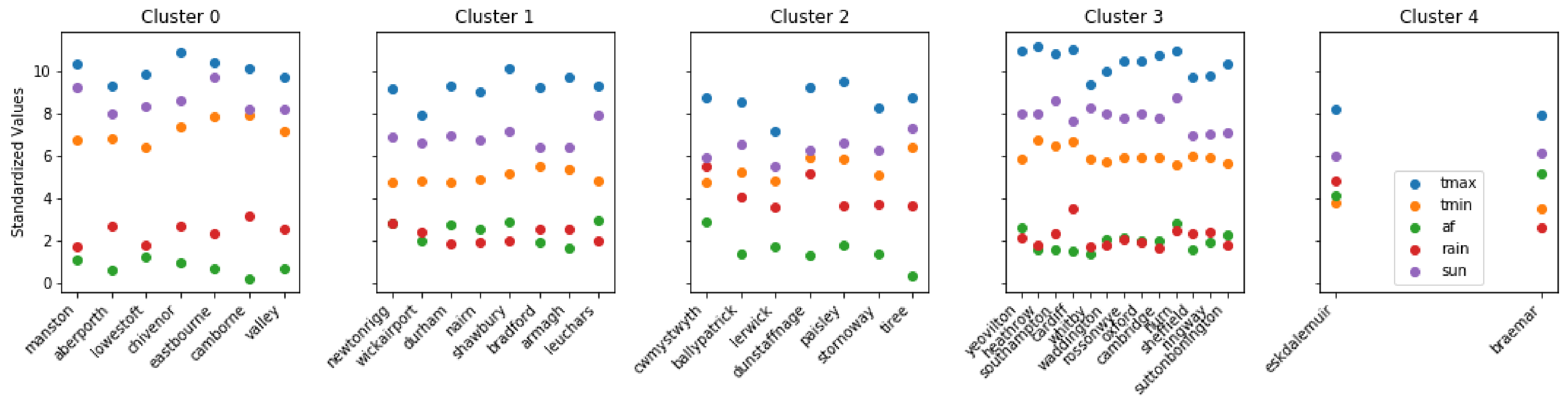
Demonstrating mean for janurary



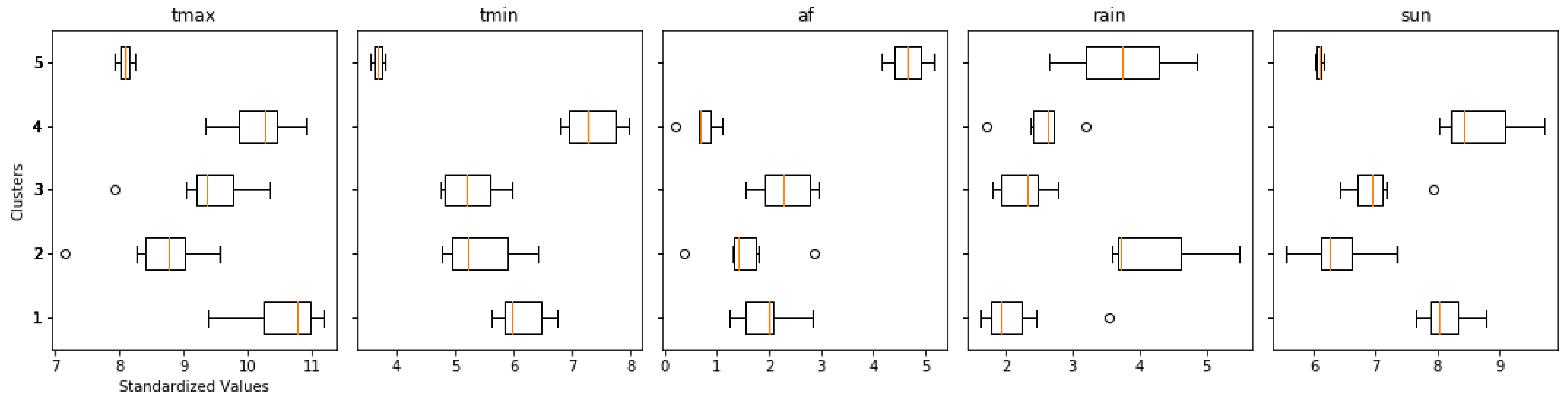
Initially, I had wanted to feed the month as a 'feature' in order to indicate that certain places would perhaps experience weather at different extremes throughout the year, and perhaps that could weigh-in on the clustering.

I abandoned this approach and decided to take the mean of a given column value over the months of a given station, which had already found the common mid-point over the years. I figured this gives a much more simple overall view of a given stations weather, but perhaps loses the time-variant dynamics.

I chose to use a Kmeans clustering, using all columns as features. To get rid of numerical bias, I first standardised the data by dividing a given column by it's standard deviation. This also helped to visualise all the data on a single y-axis. I chose the number of clusters based on the number of resultant members in each group. I got the following result:



Viewing the subplots horizontally makes it easy to compare distinct clusters. We can see that the result is quite good. Cluster 3 has generally better weather than cluster 1 for example. We can evaluate the performance with a bit more rigour in the following BoxPlot.



We can see that a given cluster does indeed look very different from another in terms of their statistical characteristics.

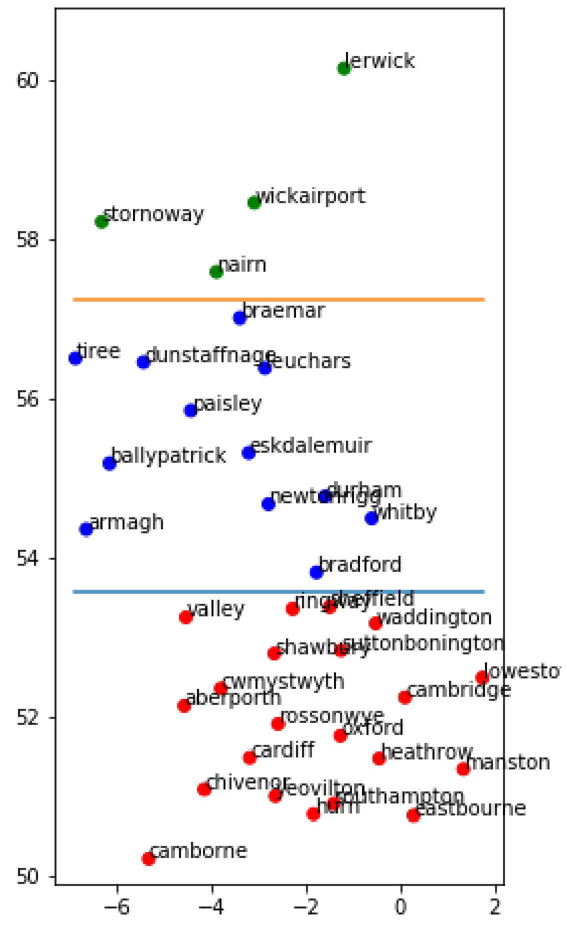
Part B.2 – Classification

As before, we import the csv into a pandas dataframe.

We use the min Latitude of 49.9 and the max Latitude of 60.9 and divide the range into thirds. A new column is appended onto the dataframe with a 1, 2 or 3 depending on a given station's Latitude or Longitude.

I had originally intended to capture site changes over the years, but the purposes of all the exercises, there wasn't much benefit (a site change of a given station doesn't change which third of UK it falls).

I split the dataframe into 60% training and 40% test.



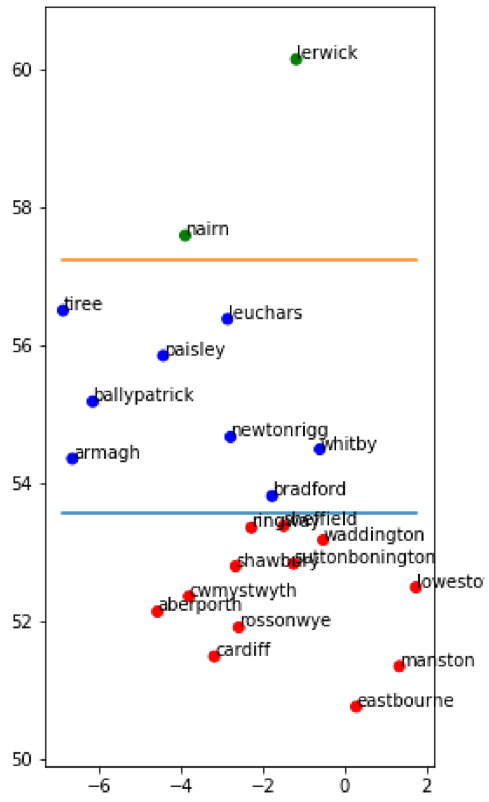
I also plotted the training set to create a general image of the UK. Coloured points indicate the third a station falls under, and there are extra horizontal lines showing this boundary.

Straight away, there's a clear issue in terms of the number of northern samples in the training set.

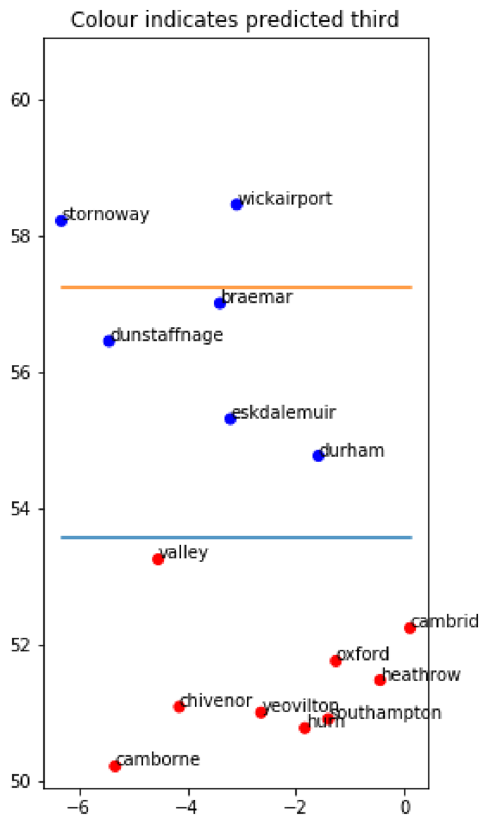
I used a KNeighboursClassifier with 5 input features; tmax,tmin,airFrost (af), rain and SunDays (sun).

As before, I took the median from all years and the mean of all months to provide a single value for each column at a station.

An accuracy score from Sklearn.metrics revealed that the trained classifier achieved ~86.67%, which is quite reasonable, given the small dataset.



With the 60% split training set shown on the left, we predict the below stations.



Unsurprisingly, we do not accurately detect northern regions. This is most likely due to the sample size.

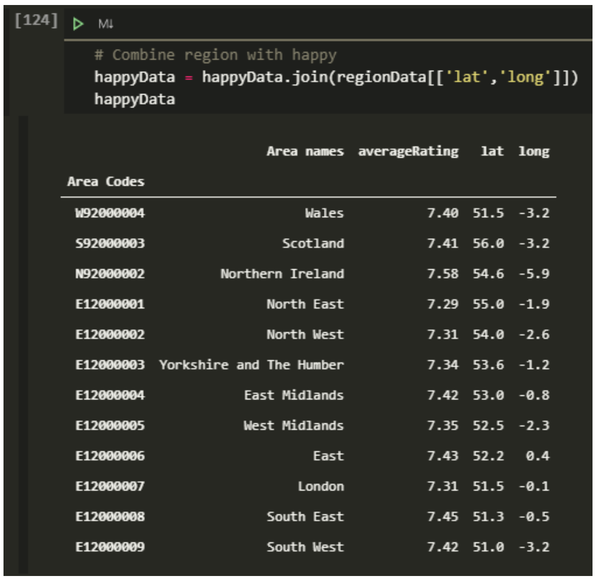
I also ran an experiment with adding the month data back into the sample set, thinking this would increase the amount of data available and produce a more accurate result. However, the opposite was the case and we generally got lower accuracy with adding the month as a feature.

I was interesting to note that the accuracy of the model highly depended on the training test split. The sklearn.model_selection.train_test_split function performs a shuffle, essentially randomising the stations chosen to be in the test/train set. I saw that the accuracy score varied by a significant amount with each iteration.

B173513
2020-08-16

Part B.3 – Linear Regression

I downloaded the happiness data at :https://www.ons.gov.uk/peoplepopulationandcommunity/wellbeing/datasets/measuringnationalwellbeinghappiness



```
[124]
# Combine region with happy
happyData = happyData.join(regionData[['lat','long']])
happyData
```

```
                        Area names  averageRating   lat  long
Area Codes
W92000004                    Wales           7.40  51.5  -3.2
S92000003                 Scotland           7.41  56.0  -3.2
N92000002         Northern Ireland           7.58  54.6  -5.9
E12000001               North East           7.29  55.0  -1.9
E12000002               North West           7.31  54.0  -2.6
E12000003 Yorkshire and The Humber           7.34  53.6  -1.2
E12000004            East Midlands           7.42  53.0  -0.8
E12000005            West Midlands           7.35  52.5  -2.3
E12000006                     East           7.43  52.2   0.4
E12000007                   London           7.31  51.5  -0.1
E12000008               South East           7.45  51.3  -0.5
E12000009               South West           7.42  51.0  -3.2
```

In this instance, I manually took the column and indexes of interest and created a csv. The contents are printed as in the figure to the right.

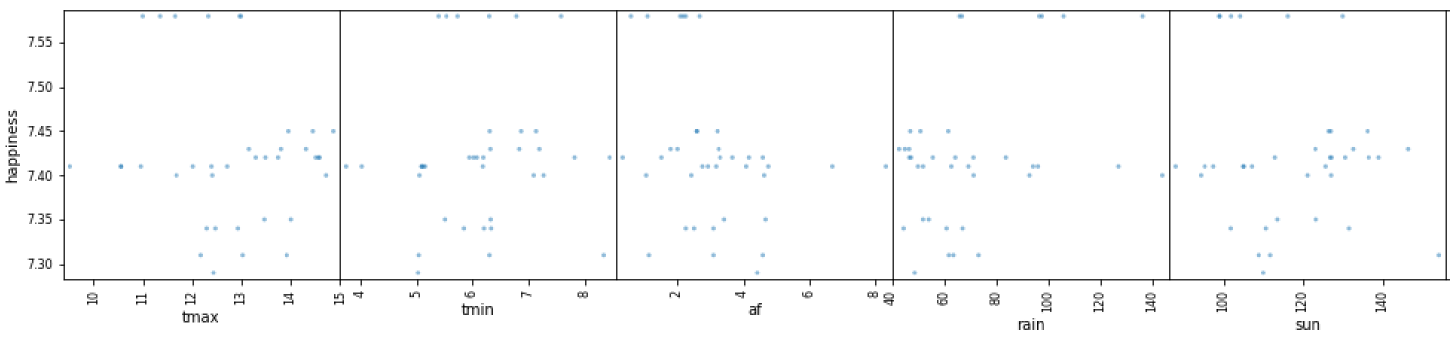
I used the provided regions.txt file to get a latitude and longitude centre point for each region.

Note, there was an error in the provided regions.txt file where Northern Ireland had the wrong longitude (missing negative sign). I manually fixed this.

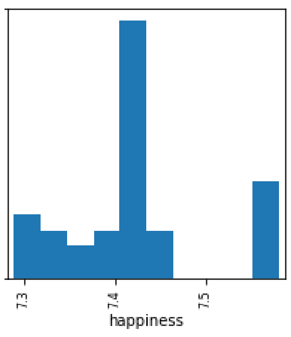
As previously, I summarised the station data into a single figure for each column at each station.

I then used the 'cdist' function within scipy.spatial.distance, which calculated the Euclidean distance between a given station to each region centre point. I then took the region with the minimum distance as the region for that station. I manually verified this was correct.

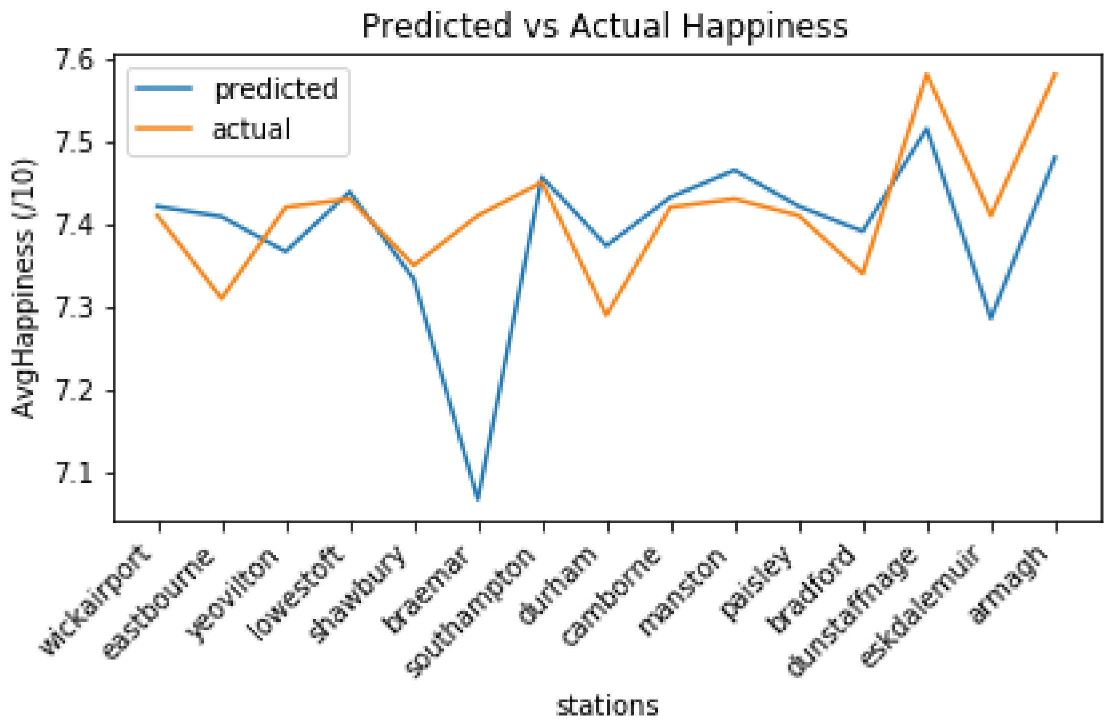
With a dataframe containing stations and happiness, I thought it wise to manually check if there were any correlations that I could see.



The data certainly looked uncorrelated to my eyes. There was also not much of a distribution to use.



Splitting the data into 60% training 40% testing again, I used a LinearRegression with the five features mentioned previously.



The linear regression performs excellently. As can be seen in the figure above, with the exception of braemar being more happy than they should be, the rest of the stations are very closely predicted. In this case, I’m more impressed because I was not able to see any correlation myself.