Project Report Gavin Horan

# GitHub URL

https://github.com/gwavin/UCDPA\_gavinhoran.git

username: **E10623416**

# Abstract

I set out to focus on COVID data. After searching several resources, I settled on the European Centre for Disease Control (ECDC) COVID Data; there are a range of datasets, these cover all aspects of COVID pandemic data as captured by the ECDC and hosted at <https://www.ecdc.europa.eu/en/covid-19/data>. I used a variety of means to download and inspect the data for insight.

# Introduction

The SARS-COV-2 pandemic is the most significant health emergency in modern history. The pandemic may follow the same lifecycle as those of the past, but now we can record vastly more data than previously possible. It is difficult to make sense of this information torrent information.

Hospital facilities may be affected by COVID; for example, we may have to decide as to whether we close or limit access to the pharmacy, depending on how dire the situation becomes.

Weekly analysis of the datasets collated by the ECDC may give us some insight which would assist our decision-making.

I decided to focus mainly on Ireland, and to an extent on those countries I considered to have health systems and populations closely matched to our own, the Netherlands, Germany, Spain and France.

# Dataset

# The ECDC is an EU agency which engages in disease surveillance, epidemic intelligence, and response, among a wide range of other roles.

# As a result of this, they have an excellent selection of datasets from which I drew to build reports which I hoped would be informative. These were located at <https://www.ecdc.europa.eu/en/covid-19/data> and it was here that I really found the most interesting data to manipulate.

# I did not start with the ECDC; Kaggle is an invaluable resource for all manner of datasets, and this is where I first looked. From here I took the fetal\_health.csv. I also downloaded NHANESdata233052221221.csv which is the data from the CDC National Health and Nutrition Examination Survey.

# Implementation Process

# In order to harness the power of API’s, I sampled some datasets from Kaggle.com.

# The Kaggle command line interface (CLI) tool is the easiest way to interact with Kaggle’s public API.

# “*pip install Kaggle*” from the terminal, ran in PyCharm. This used the python package manager, PIP, to install Kaggle.

# Instructions on the use of Kaggle are saved in Kaggle.txt. This is the Kaggle help page.

# *kaggle datasets list -s health > KaggleHealthList3.csv* downloaded the list of Health data files which were available for examination and saved these at KaggleHealthList2.csv.

# *kaggle datasets download andrewmvd/fetal-health-classification* saved the files placed on Kaggle by the user andrewmvd. I hoped that these might have information which would be of interest, so I spent some time working on those.

# In my first program, fetalHealthchecker.py I imported the fetal\_health.csv CSV file into a Pandas DataFrame, to see if it contained data which would be of interest. I printed the .info() and the .head() and unfortunately, didn’t really find much of value to look at. I am sure Kaggle is full of great information, but I went looking further afield.

# I examined the NHANES dataset, in a similar fashion. This was at <https://healthdata.gov/w/wwc6-um3r/default?cur=kq4-OidPS0Z>

# They provide a similar tool, in the nhanes package, which allows the downloading and inspection of their data. However, again, I had no interest in this. The very cursory file, usingNHANES.py has the initial investigation included. NHANESdata233052221221.csv was the output from this; I used a similar CLI tool to take in the dataframe, and used a function which I wrote, and placed in functionFile.py, to use pd.read\_csv() to send it to a named file.

# At this point, I cast the net wider and found the ECDC website.

# I downloaded the .csv using the mouse, which was not good enough.

# I used a web scraping method in *myExampleOfScraping.py* to demonstrate that I can use the methods of webscraping that we learned in the course.

# The comments at the top of this program indicate the steps I took in this program. I used Beautiful Soup to handle the html and parse it into a dataframe.

The *BeautifulSoup.soup.findall*() function pulls all html links out of this; These are then inserted into the *links* list. I used a list because it was not important to me that there be a key available to access the items, as in a dictionary. I wanted to run through all the values in *links* and inspect them with a regex.

items in dictionaries are accessed via keys and not via their position; this was not necessary.

Lists can have a variety of types and there was a problem with this; I couldn't run a regex where the system may have encountered ints and floats.

I converted all entries in *links* to strings, using the built-in *.str()* function, and then ran a regex to find those ending in .csv. I knew there would be at least one. There may be more in future; I can edit and run this program again to find or "scrape" as many .csv

files as a given webpage holds.

Next, I converted the list to a dataframe, because the pandas package makes it simpler to manipulate dataframes.

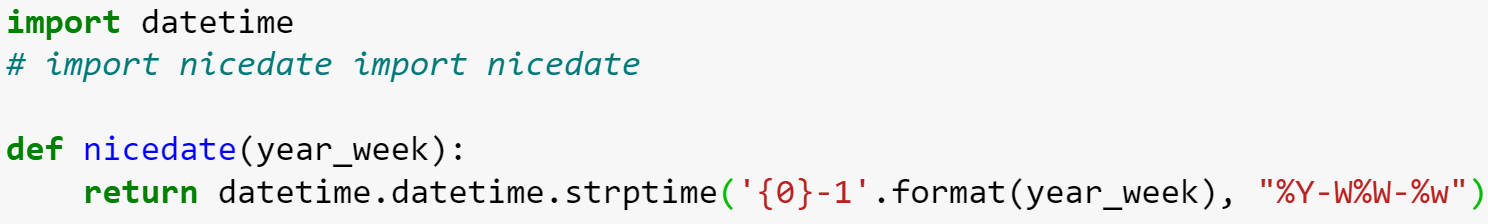
I then used a function I had already written to push this out to a .csv file. This includes the date and time appended at the end using a function which I prepared already, *csv\_time\_namer.* This creates a file which is in the format: *“<filename>115922122021.csv”* It takes the time and the date at any given moment and appends them the end of a given filename, along with “.csv” so that this file is easily interpreted.

This was just to demonstrate that I could perform webscraping if I had to, and to make use of a list. Which I haven’t done anywhere else to my knowledge. But I have here.

As mentioned, for other work I downloaded the .csv files directly from the URL. An example of this is: *lmPlotChecker.py*

I created a re-usable custom function with the ECDC dataset; they have an annoying convention of using 2020-W30 etc to refer to the 30th week of the year.

I rewrote a code snippet to deal with this.



This takes in the nasty format, saved in the column, year\_week, and interprets it by slicing out the year, and the week number, and then incorporating those into a datetime which includes day 1 by default. This meant that the labels at the bottom of my graph could look much nicer.

The re-useable custom functions which I wrote were all placed in functionFile.py and imported when needed using *from* *UCDPA\_gavinhoran\_Pycharm.functionFile import my\_dater.*

This was used in the program with a slicing process: I created a new column which I called TestingDates and then took the column [‘year\_week’] and applied my function to it, and these were then loaded into the TestingDates column.

*df\_country['TestingDates'] = df\_country['year\_week'].apply(my\_dater)*

In order to become familiar with the various utilities that one might take advantage of I also created and used a Jupyter notebook for the project. I didn’t need this at all, I had several implementations of PyCharm to work with.

LmPlotChecker.py also included the first plot I made. This made use of the Seaborn and Matplotlib packages to generate a range of graphs.

The first is the number of tests done plotted against the number of new cases detected.

In a program called: FromJan2021.py I used the following line.

*df\_sixmonths = df\_sixmonths[(df\_sixmonths['TestingDates'].dt.year == 2021)*

The dataframe df\_sixmonths is created (actually, it was created as a copy of df\_country, for some reason it was throwing an error; this line was in danger of causing a conflict at another point in the program and needed the line: *df\_sixmonths = df\_country.copy()* to be added before it would function smoothly.

This line included the use of a Boolean; I have extracted the year from the TestingDates column, which contains a datetime object. The dt.year function produces the year, this is compared with 2021. In honesty, this didn’t do much for the quality of the resulting graph. What I was trying to do was to check if there was a correlation between the number of tests done and the number of cases of COVID found.

When applied against the full dataset, as I will show below, the correlation was reasonably good. As time went by, and our testing facilities expanded, we presumably had enough testing to cover all people who needed tests. Consequently, the linkage between tests done and number of cases fell. And the linkage weakened, so the regression plot failed to be as accurate.

Although I would argue it helped to confirm the earlier insight.

FromJan2021 also includes two graphs on one plot; this was the testing rate on one line over time and the rate of new\_case detection on the other.

MergingCSVcopyExport.py is where I include an example of a merged trio of csv data input. In this program I used matplotlib and seaborn, and experimented with all the styles available.

I use the pd.read\_csv() function to read in two dataframes and then merged them together.

Quite a few programs include a line of the sort below: df\_sort\_by\_ICU.to\_csv("twentyWorstDays.csv")

This sends the contents of the dataframe in question to a .csv file. In functionFile.py I also stored a function I defined which is:

*def csv\_time\_namer(filename):*

*t = '{0:%H%M%S%d%m%y}'.format(datetime.datetime.now())*

*b = '.csv'*

*return (filename + t + b)*

This snippet takes the input argument of filename, and takes t, the formatted time and date, and then appends this along with .csv to the end of the file. This is reused manytimes across the course of the project.

I used a loop here,

for col in df\_new.columns:

print(col)

This was simply to run down through the columns so I could get a look at what was in there for future analysis. But it is an example of a loop. It checks that the condition is satisfied: is there a column left in the dataframe? If yes, it prints the head of the column, if no, it exits the loop.

I created dropper.py because there were no duplicates that I could see in the ECDC data, and I stayed within that set of datasets in my analysis, so I had to synthesise the duplication.

I vertically merged a dataset with itself using concat(), which obviously doubled the number of rows. I then removed the duplicates again, using drop\_duplicates. This proved the capacity to perform the act rather than having a practical application.

Once dataframes provided by the ECDC are merged carefully, across all columns that they have in common, duplicates are not easy to come by; they have demonstrated understanding of the data they are dealing with and have kept their datasets very clean.

In deathNumPy.py I looked at the deaths dataset, using .loc[] to remove rows where the value for deaths reported was zero.

I then used NumPy to generate an average number of deaths reported across the period in the report, which is from 1.3.2021 this year. It is 19.65 people per day in Ireland, which I was surprised by. I didn’t think it was that high, even saturated as we have been by the coverage.

I then used group\_by to generate a group object for the deaths dataset. I then sorted the group by the average daily value; Italy has by far the highest average deaths as reported in this datasheet, which rings true, given the number of deaths which were occurring last year.

Finally, in deathNumPy.py I used indexing to generate a list of countries who had a higher average death than Ireland.

# Results

Please find below some of the charts generated and some insights I believe I gleaned, from the ECDC datasets.

# Presumed Insights

# Insight 1

This graph was generated by regressionCheck.py

This is a regression analysis of testing and new cases in Ireland.

Chart

Description automatically generated

This shows that the impact of the independent variable is not constant. Early in the pandemic, the number of tests done linked strongly to the number of new cases diagnosed. This implies that the increase in testing facilities early in the pandemic meant that we were catching a proportion of cases limited by the number of tests we were capable of doing.

I then added a second polynomial, to see if the data fitted this graph; generated by the same program:

Chart, scatter chart

Description automatically generated

This is a second order regression plot. When the number of tests and the number of cases were both reasonably low, they were relatively tightly bound. Now that testing is more freely available this is not longer the case.

# Insight 2

The following graph was made using lmPlotCheck.py; this uses the same data as previously; except from the Netherlands,

Chart, scatter chart

Description automatically generatedand they do not have the same linkage as us at all; it makes sense I would have thought that they would have a more efficient system, and as a result, they would have started to pick up an independent number of cases early on, before Ireland. (Insight 2)

# Insight 3.

Chart, line chart

Description automatically generated

The above graph generated by labelsBest19122021 was from data for Ireland only, this was what gave me an idea as to the link between testing and new cases, and how they had become detached.

And insight here is that we had three waves which could be measured with test results. The area under the curve in this current wave, in December 2021, seems vastly higher, it appears that we are indeed at a tipping point, and the numbers are going up in a way they have not before, and our testing facilities are adequate to show this.

By simply changing the country under examination, we can see the German figures; I would assume the testing line would be much flatter for Germany.

Chart, line chart

Description automatically generated

This graph was generated using the same program, labelsBest19122021, except swapping out Germany for Ireland, and sure enough, the line is flat, as projected.

Insight 4

Rates of testing:

The graph below is the rate of testing in Ireland - green, and the rate in Germany – red, and Spain – blue, derived from CovidTestsEurope.py; German demand and delivery of tests peaked around January of this year, and then remained constant. Ours had a peak but has continued to go up since. This implies that we were slower to reach capacity for testing than Germany and Spain.

Chart, histogram

Description automatically generated

Insight 5

The graph below is from dateDefGraphs.py.

Chart, histogram

Description automatically generated

This demonstrates that as time goes by the Netherlands and France were significantly more responsive to demand in the requirement for tests than Ireland. Our peaks were quite limited relative to theirs. It was this difference that gave me the idea to check if the fact we were building capacity might have led to a link between testing and cases detected. If we had adequate tests to respond when needed in January and April 2021, we may have been able to test more and thus detect all the cases.

Insight 6

This graph is from Testing5Nations19122021.py

Chart, line chart, histogram

Description automatically generated

The Netherlands have always been somewhat ahead of us for positivity peaks. It looks like they are heading for another one, slightly ahead of us. Hopefully, this pattern is broken over Christmas, and we continue a downward trajectory.

# Insight 7

The below graph was produced with ICUIreland.py.

Chart, line chart

Description automatically generated

This gives me some ground for optimism here. This makes it quite clear that while cases now are at a historically high level, ICU attendance is lower, and is trending down. With any luck, we will find ourselves out the other side of all this before we know it.

# References

<https://www.ecdc.europa.eu/en> was used for most of the data used in this analysis.

Datacamp.com provided the skeleton for most of the work.

Python for Data Analysis by Wes McKinney provided some reference material.