Week 10 Coursework 1 - GP Data

Barry Rowlingson

December 12, 2021

1 Outline

For this assignment you will be investigating a data set of consulations at NHS General Practices (GPs). Each week, many GPs submit these statistics on consultations to a central NHS agency which then compiles and publishes tables and reports.

For this project, work in an interactive Jupyter Notebook, with one or zero external Python files if you want to put any code in a separate module. Make sure your Notebook file runs cleanly from the start before submitting it and any Python file.

2 Reading the Data

Each row of the data is a week of counts of consultations in each of four categories (influenza-like symptoms, vomiting, diarrhoea, and other gastro-intestinal) for all reporting practices in a local authority. There is also the total number of people registered at the reporting practices that week.

The data is in an SQLite3 database file called `gpinhours.sqlite`. The `sqlite3` package is supplied with Python and enables you to create a database connection for reading tables from the file.

The table we are interested in is called `inhours`. Read this table into a Pandas DataFrame. Its "info" method should look like this:

```
inhours.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47084 entries, 0 to 47083
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   CODE        47084 non-null  object
 1   NAME        47084 non-null  object
 2   TOTAL_POP   46862 non-null  object
 3   Flu_OBS     46669 non-null  float64
 4   Vom_OBS     46672 non-null  float64
 5   Diarr_OBS   46672 non-null  float64
 6   Gastro_OBS  46757 non-null  float64
 7   lastdate    47084 non-null  float64
 8   week        47084 non-null  float64
dtypes: float64(6), object(3)
memory usage: 3.2+ MB
```

and a selection of columns should look like this:

```
inhours[["CODE","NAME","TOTAL_POP","Flu_OBS","lastdate"]]
```

```
          CODE                 NAME TOTAL_POP  Flu_OBS  lastdate
0      E09000002  Barking and Dagenham     63009      1.0   17741.0
1      E09000003                Barnet    225813      1.0   17741.0
2      E09000004                Bexley     46339      1.0   17741.0
3      E09000005                 Brent    178399      1.0   17741.0
4      E09000006               Bromley    168381      0.0   17741.0
```

```
...          ...                 ...        ...      ...       ...
47079  E06000041           Wokingham    156573     15.0   18287.0
47080  E10000014           Hampshire   1315029    135.0   18287.0
47081  E06000046       Isle of Wight     87081      1.0   18287.0
47082  E06000044          Portsmouth    183688      1.0   18287.0
47083  E06000045         Southampton    224219     12.0   18287.0

[47084 rows x 5 columns]
```

3 Data Cleaning

3.1 Fix the Population Count

The reported population should be numeric, but the column description is not one of the numeric types. This means there must be non-numerically formatted text in that column. There is a Pandas function to attempt conversion of text to numeric. Note that this fails if any of the text items cannot be interpreted as numbers:

```
test1 = ["1.0","2.2",3]
print(f"converting test1 = {pd.to_numeric(test1)}")
test2 = ["1.0", 2, "3.14", "", "None", "A"]
# in a try/except to avoid a long error traceback in this document
try:
  print(f"converting test2 = {pd.to_numeric(test2)}")
  print("Success")
except ValueError:
  print("ValueError converting text as numbers")
```

```
converting test1 = [1.  2.2 3. ]
ValueError converting text as numbers
```

Check the help for `pd.to_numeric` and write a function that converts to numeric but replaces invalid values with Python's "not a number" code.

```
def as_number_or_nan(x):
  """ write this """

  return something
```

Run some tests to make sure you get the expected results. Expand this test set to more possible variants:

```
tests = [1, 2, -1, "-1", "A", "O", "0"]
print(tests)
print(as_number_or_nan(tests))
```

```
[1, 2, -1, '-1', 'A', 'O', '0']
[ 1.  2. -1. -1. nan nan  0.]
```

Next create a new column, `POP`, with the converted numeric value (or "not a number"), from `TOTAL_POP`. Don't overwrite `TOTAL_POP` since our next task is to investigate which rows have these bad population values.

Produce a table of the local authority names (from the `NAME` column) for the rows where the population couldn't be converted to a number. You may need the `np.isnan` function and the `.value_counts()` method of a Pandas Series for this.

```
Bury                     115
Isle of Wight             53
Wokingham                 50
Lambeth                    1
```

```
Knowsley                        1
Herefordshire, County of        1
Barking and Dagenham            1
Rochdale                        1
Greenwich                       1
Sutton                          1
Bexley                          1
Merton                          1
Cheshire West and Chester       1
Name: NAME, dtype: int64
```

3.2 Fix the Date

The `lastdate` column is the number of days since the first of January in 1970. Use `pd.to_datetime` to convert this to a column of dates. This should give a column with these properties:

```python
inhours.date.describe(datetime_is_numeric=True)
```

```
count                            47084
mean     2017-01-17 20:39:29.620252928
min                2014-01-05 00:00:00
25%                2015-07-17 06:00:00
50%                2017-01-18 12:00:00
75%                2018-07-23 18:00:00
max                2020-01-26 00:00:00
Name: date, dtype: object
```

3.3 Trim the Years

The `.dt` attribute of the `date` column gives access to Python's methods for dates. Add a new column holding the year for each row. Look at the count of how many times each year appears. You should notice that there is not much of 2020 included so subset the data to remove rows for 2020.

3.4 Drop Zero Populations

A number of records are present with population totals of zero. These should be excluded as missing data. Remove all rows with zero for the `POP` column.

3.5 Final Clean Data

A few columns of the final cleaned dataset should look like this:

```python
inhours[["CODE","POP","Flu_OBS","Vom_OBS","date","year"]]
```

```
            CODE        POP  Flu_OBS  Vom_OBS       date  year
0      E09000002    63009.0      1.0     13.0 2018-07-29  2018
1      E09000003   225813.0      1.0     40.0 2018-07-29  2018
2      E09000004    46339.0      1.0     11.0 2018-07-29  2018
3      E09000005   178399.0      1.0     43.0 2018-07-29  2018
4      E09000006   168381.0      0.0     19.0 2018-07-29  2018
...          ...        ...      ...      ...        ...   ...
46483  E06000041   150642.0     14.0     12.0 2019-12-28  2019
46484  E10000014  1284063.0    117.0    120.0 2019-12-28  2019
46485  E06000046    67289.0      1.0      1.0 2019-12-28  2019
46486  E06000044   183875.0      1.0      1.0 2019-12-28  2019
46487  E06000045   222761.0     11.0     11.0 2019-12-28  2019

[45924 rows x 6 columns]
```

4 North-South Divide

England is often divided into north and south via an informal division by a line from "the Bristol Channel to the Wash" (look these up on a map if you're not familiar with the country!). There is always much political and social discussion about "The North-South Divide". In this section we'll see if a divide is visible between these regions in the GP data.

The database contains a table called `localauth` which has the local authority names and a column indicating if the area is north or south.

4.1 Read the Table

Read this table into Python from the database to get:

```
localauth
```

```
                            NAME     NS
0          Barking and Dagenham  South
1                        Barnet  South
2                      Barnsley  North
3    Bath and North East Somerset  South
4                       Bedford  South
..                           ...    ...
144                      Wirral  North
145                   Wokingham  South
146               Wolverhampton  North
147              Worcestershire  North
148                        York  North

[149 rows x 2 columns]
```

4.2 Merge with the GP Data

These names all match with the names in the GP data, so we can code each row in the GP data as being "North" or "South". Use the `pd.merge` function to join the local authority code to the GP data to create a new data frame, `inhoursNS`, with a north-south column. This table shows some of the columns:

```
inhoursNS[["date","NAME","NS","Flu_OBS","POP","year"]]
```

```
            date                  NAME     NS  Flu_OBS       POP  year
0     2018-07-29  Barking and Dagenham  South      1.0   63009.0  2018
1     2019-06-23  Barking and Dagenham  South      1.0  142085.0  2019
2     2018-10-07  Barking and Dagenham  South     10.0  121126.0  2018
3     2017-09-03  Barking and Dagenham  South      1.0  103819.0  2017
4     2014-01-05  Barking and Dagenham  South      5.0   91464.0  2014
...          ...                   ...    ...      ...       ...   ...
45919 2019-12-01           Southampton  South      1.0  112897.0  2019
45920 2019-12-08           Southampton  South      8.0  139623.0  2019
45921 2019-12-15           Southampton  South     13.0  145758.0  2019
45922 2019-12-22           Southampton  South     20.0  156676.0  2019
45923 2019-12-28           Southampton  South     11.0  222761.0  2019

[45924 rows x 6 columns]
```

4.3 Group and Aggregate

To investigate a possible north-south divide, create a new data frame called `aggNS` by grouping the data by the north-south column and by year. Compute total population and total observed consultations for each of the four classifications. Use the `.groupby` method to make groups and the `.agg` method to specify aggregations. This should produce a data frame like this:

```
aggNS
```

NS	year	Total	Flu_OBS	Vom_OBS	Diarr_OBS	Gastro_OBS
North	2014	719655399.0	31870.0	144469.0	279606.0	524322.0
	2015	812875983.0	44352.0	155056.0	302202.0	555413.0
	2016	852949550.0	48895.0	159742.0	307934.0	569276.0
	2017	737539944.0	30220.0	122787.0	248197.0	462994.0
	2018	607900399.0	46228.0	90951.0	194246.0	365863.0
	2019	614454668.0	31826.0	87114.0	186457.0	357594.0
South	2014	759999787.0	47093.0	141909.0	283547.0	507028.0
	2015	838254817.0	61180.0	147032.0	300360.0	537786.0
	2016	908035557.0	66822.0	156401.0	308993.0	558159.0
	2017	769928672.0	40572.0	117494.0	245726.0	442456.0
	2018	641245125.0	45791.0	84614.0	191328.0	343863.0
	2019	647648492.0	34353.0	79300.0	180489.0	324279.0

4.4 Consulation Rates

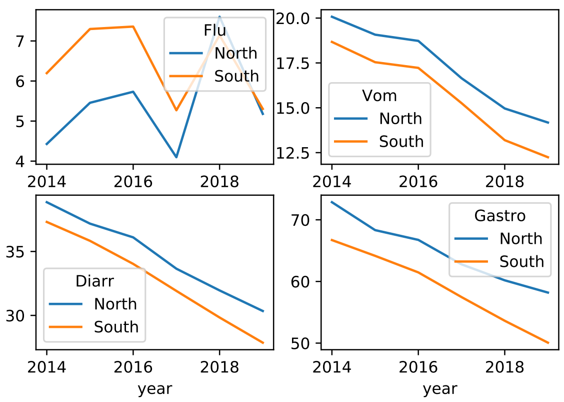
Add four new columns of consultations per 100,000 population for the four classes of consultation. That should give this:

```
aggNS[["Flu_rate","Vom_rate", "Diarr_rate","Gastro_rate"]]
```

NS	year	Flu_rate	Vom_rate	Diarr_rate	Gastro_rate
North	2014	4.428508	20.074747	38.852762	72.857370
	2015	5.456183	19.074988	37.176889	68.326905
	2016	5.732461	18.728189	36.102252	66.742048
	2017	4.097405	16.648183	33.652008	62.775447
	2018	7.604535	14.961497	31.953590	60.184695
	2019	5.179552	14.177449	30.345119	58.196970
South	2014	6.196449	18.672242	37.308826	66.714229
	2015	7.298497	17.540251	35.831587	64.155432
	2016	7.358963	17.224105	34.028734	61.468848
	2017	5.269579	15.260375	31.915424	57.467142
	2018	7.140951	13.195266	29.836952	53.624267
	2019	5.304266	12.244296	27.868358	50.070216

4.5 Plot the Rates

Make four graphs, one for each of the consultation classes. Each graph should have two lines for the annual rates for north and south. The resulting basic plots should look like this, but you may want to improve the graphics.



5 Influenza Data

Influenza has a well-known seasonal pattern with increasing case rates during the English winter. Real-time disease surveillance is used to detect the start of the influenza season as well as its severity. For this exercise we'll look at the national picture by aggregating the data over all local authorities for each date. We are not looking at the North-South information in this section.

5.1 Grouping and Aggregating

Group the data by date and compute the total reported population and the total influenza consultations for each date.

5.2 Compute Rates

Calculate the influenza consult rate per 100,000 population and add as a new column called `Flu_rate` to the data. The index of this data frame should be the date as it was the grouping variable for an aggregation. Add the year of the index to the data frame as a new column called `year`.

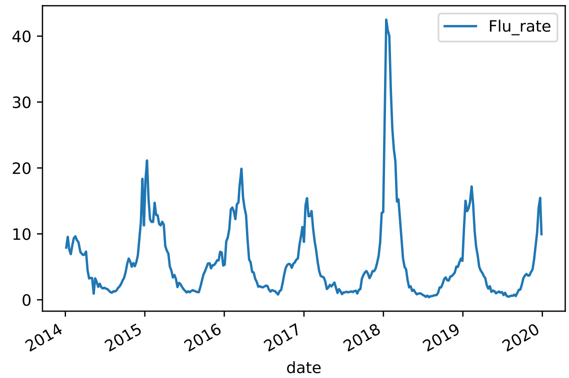
```
flu
```

```
                 Total  Flu_OBS   Flu_rate  year
date
2014-01-05  28607373.0   2264.0   7.914044  2014
2014-01-12  29197811.0   2789.0   9.552086  2014
2014-01-19  29350395.0   2236.0   7.618296  2014
2014-01-26  28766178.0   1990.0   6.917846  2014
2014-02-02  29268260.0   2419.0   8.264926  2014
...                ...      ...        ...   ...
2019-12-01  22565354.0   1866.0   8.269314  2019
2019-12-08  23279755.0   2378.0  10.214884  2019
2019-12-15  26375563.0   3687.0  13.978849  2019
2019-12-22  27625895.0   4271.0  15.460133  2019
2019-12-28  39813063.0   3963.0   9.954019  2019

[312 rows x 4 columns]
```

5.3 Plot National Weekly Rates

Use the `.plot` method to make a time-series plot of the influenza rate. Here is a basic plot design which you might want to improve:



6 Season Start Detection

In each year, after the peak is reached early in the year, the rates decrease until they start rising again, but there is noise in the data so that triggering a "Season Start Alert" on any increase in the raw data would result in early false alarms.

6.1 Finding the Minimum

Write a function called `when_min` that returns the index of the minimum value of a Series. Make sure it only returns one value if there are multiple minima. Test that it works with these examples.

```
S1 = pd.Series([5,4,3,2,1,2,3,4]) # one minimum
print(when_min(S1))
S2 = pd.Series([5,4,3,2,1,1,2,3]) # multiple minima
print(when_min(S2))
```

```
4
5
```

Group the flu data by year and use the `.agg()` method to return the date of the minimum value:

```
       Flu_rate
year
2014 2014-05-11
2015 2015-07-12
2016 2016-09-04
2017 2017-06-25
2018 2018-07-29
2019 2019-07-28
```

6.2 Simple Season Start Detection

There can be various ways to define when the flu season has really started, and as long as we are consistent we can make comparisons across years. For this exercise, define the season start as the point in time *after the yearly minimum* where the rate is more than twice what it was at the minimum.

Write a function, `season_start`, that when given a Series returns the index of the Series for the definition above. It should first find the index of the minimum and the value at the minimum. It should then subset the Series for elements with index value after that element, and where the value is more than twice the minimum. Then it should return the first of those index values.

```
# min is 1, first > 2 is sixth element (the first "3")
S3 = pd.Series([3,2,1,1,2,2, 3 ,3,3])
print(season_start(S3))
```

```
6
```

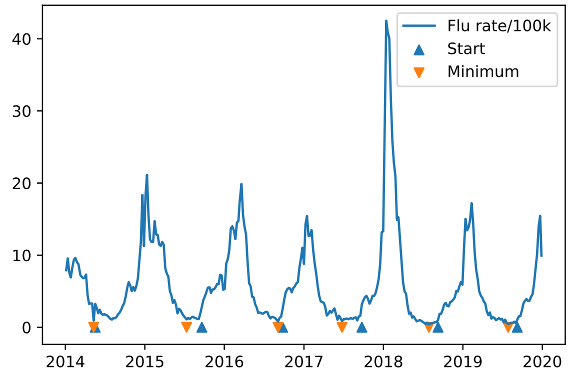
6.3 Yearly Season Starts

Now group the flu data by year and use the aggregate method to return the start of the season:

```
       Flu_rate
year
2014 2014-05-18
2015 2015-09-20
2016 2016-09-25
2017 2017-09-24
2018 2018-09-09
2019 2019-09-08
```

6.4 Plot Data with Minima and Season Starts

Create a plot showing the flu rate with the yearly minima and season starts like this:

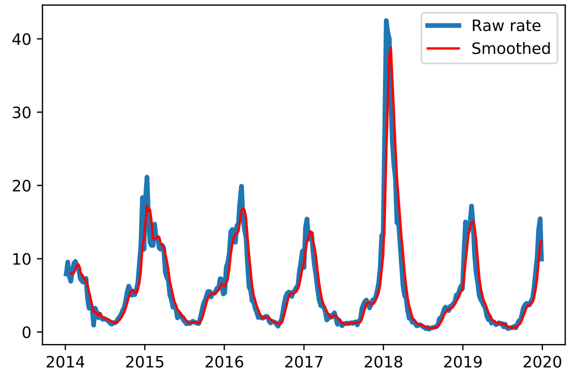


To do this, initially plot the flu rate, then add two scatter plots for the minimum and season start. Set the Y coordinate value for these two scatter plots to zero by adding a new column to their data frames. Use the `marker` option to choose appropriate marker styles, and use `label` to give each part an entry in the legend. Feel free to improve the plot.

6.5 Smoothing

The season start detection for 2014 is clearly too early because of an anomalously low value. To get round this we can smooth the rate by applying a rolling mean over a window of a few days.

Use the `rolling()` method to apply a four-day rolling `mean()` to the raw rates. Assign this to a new column. You can then plot a smoothed flu rate over the raw rate. Use a thicker line width for the first line so that it is still visible under the second.



6.6 Season Start for Smoothed Data

Now apply the minumum and season-start finding methods used previously to the smoothed data.