Nicolas Mavromatis

Nima6629@colorado.edu

CSPB 3287 Final Project Write-Up

Title- Covid-19 Ohio Data Analysis

Technical Goals

The learning goals of this project were to use Pandas, SQL-Lite, SQL-Magic, python (JupyterLab) and Power-BI in order to gain experience manipulating data and databases with commonly used tools. I wanted to both be able to do the required actions (I.E. joins, data editing, etc.) in the higher level pandas tool, outputting to excel sheets to have a physical backup of these steps, as well as perform these actions in the lower level SQL-Lite to demonstrate proficiency in writing SQL code and managing data. I utilized SQL-Magic in order to create an engine and cursor which is the most reliable way to upload to and download from a server.

Finally, I spoke with a colleague that works heavily in the data industry, and he told me of the importance of Power Business Intelligence (Power-B.I.) in analyzing and visualizing data, so I imported two of the processed excel sheets into this in order to visualize the data in a way that would likely be done in real life scenarios.

Data Analysis Questions and Hypotheses

My main goals were to learn to use these relevant toolsets, but in order to make the project interesting, I also wanted to say something relevant about the data and spread of Covid-19. Experts had predicted that Covid-19 would spread most during the cold Winter months, following patterns of Flu virulence, when the cold impedes the immune system's mucus clearance and pathogen response, and people gather indoors (1). Additionally, spread is likely to be high after big Holidays when people are in close contact with family and friends in large groups, especially Halloween, Thanksgiving, and Christmas. I also wanted to say something about the spread of the disease during the summer months, which I expected to be lower due to people being able to safely distance outdoors, and the notion that the virus is poorly adapted to outdoor settings, and is sensitive to UV light and high temperatures (2).

I expected for the positive cases, deaths, and other related factors to be highest during the initial month (May 2020) when the disease was new, and when the WHO stated not to wear masks in a probable attempt to save them for health care professionals. During this time, there was a lot of uncertainty and it appeared as if most people didn't know exactly how to respond to this unprecedented situation, with a high degree of panic. My goal was to confirm if the data supported these predictions based on transmission patterns of other viruses, and the little information available about the spread of Covid-19 itself.

Project Step-by-Step

First, I'll walk through the process of collecting and cleaning the data. Please see the video and accompanying .ipynb pdf. I imported a .CSV file for Ohio Covid data from CovidTracking.com containing 2020 data from 4/2020-12/2020, which had many useless columns with missing data. After reading in the raw .CSV file into a Pandas dataframe, I dropped a number of columns, including deathProbable, hospitalizeCumulative, hospitalizedCurrently, etc. that were redundant to better populated fields. This left the relevant fields of deaths, hospitalized, inICU(Cumulative), and positives. I extracted the month of each date field using a convenient Pandas method, then saved this to OhioDataClean.csv. Next, I grouped by month, calculated the means of each group, and saved this to another dataFrame which I then output into OhioDataMeans.csv. I made a basic plot of the average data by month in Pandas, noticing that the positives field followed a curve similar to the S shaped curve predicted by experts, albeit not yet flattening as this was data from 2020. I created a dataFrame of the percent increase in each of these four columns from the prior month (by average), outputting as OhioPct.csv. I also did a full outer merge on the dataframes by matching field month for averages of each month, and the percent increase of each parameter from the prior month, outputting as Merged.Csv.

Next, using SQL-Magic's engine system, I uploaded the dataFrames o the University Server in order to demonstrate this capability. I then fetched the values from OhioCovidMean to demonstrate data transfer in the other direction. Finally, I imported the excel files containing the average and percent increase by month and learned how to use Power BI in order to create graphs, below. I graphed the average of each parameter by month and the percent increase from each prior month in average number of positives. The positives parameter best fit the expected patterns and helped answer the previous questions the most effectively. Positivity rates are known to most accurately track the spread of the virus, as not all sick people choose to or able to go to the doctor, and 3.4% of reported Covid-19 cases globally have died from it, so the other parameters are interesting but less useful (3). From this, I was able to analyze trends in the data and make educated answers to the questions I posed before.

I also manually recreated the average and percent increase tables in SQL (SQL-LITE) to demonstrate proficiency in creating lower-level code. I used 'Month' as a primary key, and 'ID' as a foreign key, triggering errors if constraints were broken (such as inserting a duplicate month, or a foreign key 'ID' field in one table without a match in the other). I made a specific "ON UPDATE" condition and attempting to delete a foreign key field triggered the necessary error. I then performed a join on the 'ID' field, storing it in a new table. I built an index in this table on 'ID', and then performed a few queries, using Group by in order to better make sense of the data. With that sequence of steps, all the items required in the project were included.

Data Analysis and Conclusions

In Power B.I., I graphed the average of each column as a line chart, and the percent increase in positives from each prior month (by average) as a bar chart. The averages by month of deaths, hospitalized, and inICU(Cumulative) resulted in very similar charts. The hospitalized numbers were much higher than inICU(Cumulative), reaching a max of 33,000 and 3,000 respectively in December, which makes sense in that ICU is only for the most extreme cases. Deaths reached a high of about 8,000 in December, with the largest increase of about 33% between November and December of 2020. This makes sense and agrees with the prediction, in that viruses spread well in Cold months and people were likely in close contact with family indoors for the Thanksgiving and Holiday season. However, the increase from October to November was somewhat lower than expected (only 14%), which might possibly be partially explained by the lower popularity of large Halloween celebrations.

Finally, the positives best demonstrate the virology trends. There was a 186% percent increase from 4/20-5/20, when the virus first hit, likely explained by its novelty and the lack of clear mask messaging from the WHO. The WHO changed their official stance on masks in June of 2020 (4), which I believe resulted in a large reduction in viral spread. Next, the percent increase in positives evened out a lot, but remained high, being 56%, 67%, and 52% from each prior month in June, July, and August respectively, agreeing with the prior predictions. By this point, the WHO had recommended to wear masks. This might also be explained by the virus's relative inability to spread outdoors due to UV sensitivity (2), the tendency of viruses to spread poorly during warm months, as well as people's ability to safely distance safely outdoors.

However, things took a turn for the worst in the colder months, agreeing with the above hypotheses and reasoning, restated here: viruses spread better during cold months, due to reduced immune effectiveness having to do with mucosal surfaces, and people were likely in closer contact. Also, during the Holiday season beginning with Thanksgiving, people were in closer contact with large groups of friends and family, and most likely travelling great distances, excellent conditions for viral spread. The percent increase in positives were lower in September and October from each previous month, being around 30 % each. The weather was just starting to turn cold at this point. However, the increases skyrocketed in November and December, being 71% and 88%. The ultimate number of positives reached the high point of 577,000 in December.

Here are the results for the four highest absolute averages of positives by month, and the three highest ratio of increases (in decimal form, requiring multiplication by 100 to convert to percent) on average from the previous month, presented in tabular form:

[207]:

month	MaxPos
12	577271.1612903225
11	307793.9
10	180360.32258064515
9	139679.93333333332

[209]:

month	MaxPosIncrease
5	1.8560310479770687
12	0.8755120270100301
11	0.7065499528721182

In summary, I learned a great deal about using data analysis tools and was able to form a basic hypothesis based on my prior knowledge of virology in order to answer some questions about Covid-19 in Ohio. My original predictions were pretty much spot-on, and closely followed previous knowledge about the spread of the Influenza virus, and how it follows seasonal changes due to different behaviors and seasonal effectiveness of immune responses. If given more time, it might be interesting to analyze a larger amount of data by state, but this would be a grand project far beyond the scope of this class. I certainly satisfied all my learning goals, and plan to further practice Power B.I. which integrates many other tool sets and is an excellent way to visualize and analyze large amounts of data. Thank you for reading.

Sources:

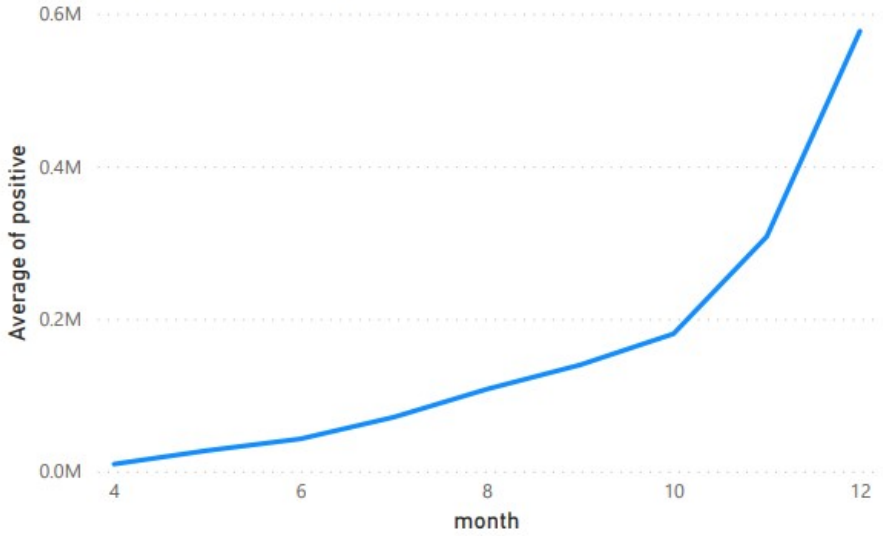
1: Horizon Blue Cross Blue Shield of New Jersey. Why Do Viruses Spread More in Winter? Cold Temps Are Key, Horizon Blue Cross Blue Shield of New Jersey, 22 Feb. 2021, https://www.horizonhealthnews.com/why-do-viruses-spread-more-in-winter-cold-temps are-key/.

2. maggiekb1. "What a Summer of COVID-19 Taught Scientists about Indoor vs. Outdoor Transmission." FiveThirtyEight, FiveThirtyEight, 19 Oct. 2020, https://fivethirtyeight.com/features/what-a-summer-of-covid-19-taught-scientists-about indoor-vs-outdoor-transmission/.

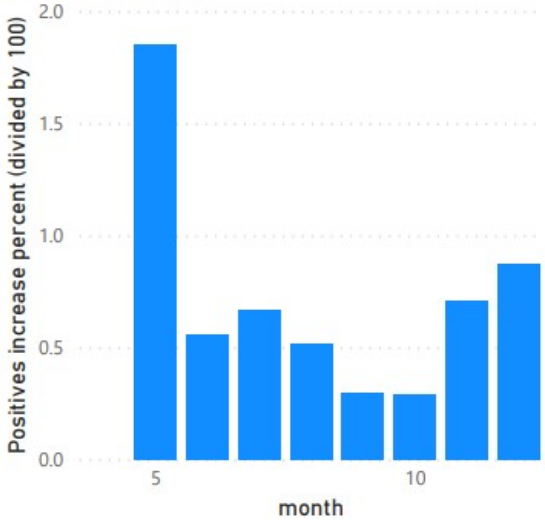
3. "Coronavirus (COVID-19) Mortality Rate." Worldometer, https://www.worldometers.info/coronavirus/coronavirus-death-rate/.

4. Ellis, Ralph. "Who Changes Stance, Says Public Should Wear Masks." WebMD, WebMD, 8 June 2020, https://www.webmd.com/lung/news/20200608/who-changes-stance-says public-should-wear-masks.

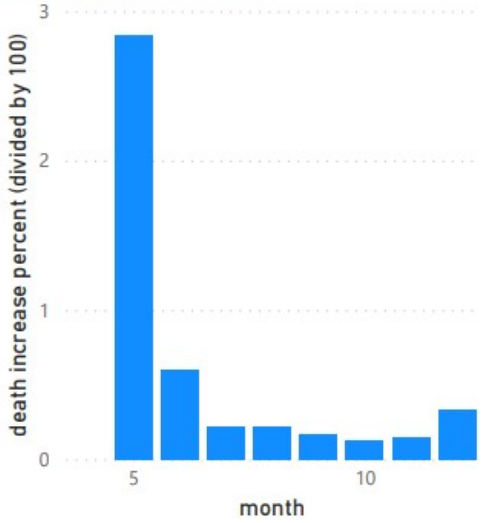
Average of positive cases by month



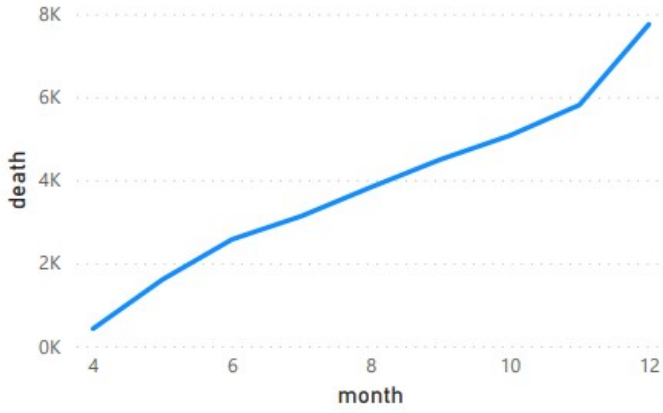
Average of positive percent increase from each prior month



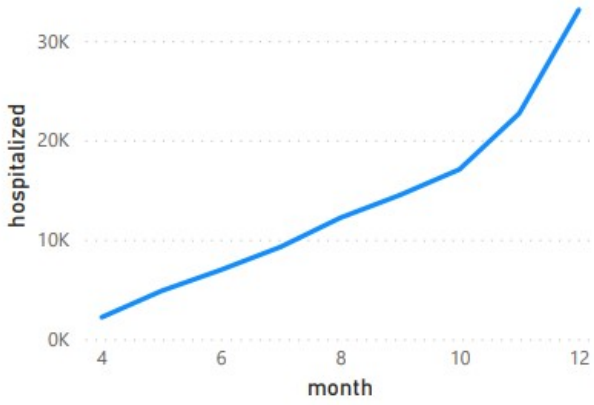
Average of death percent increase from each prior month



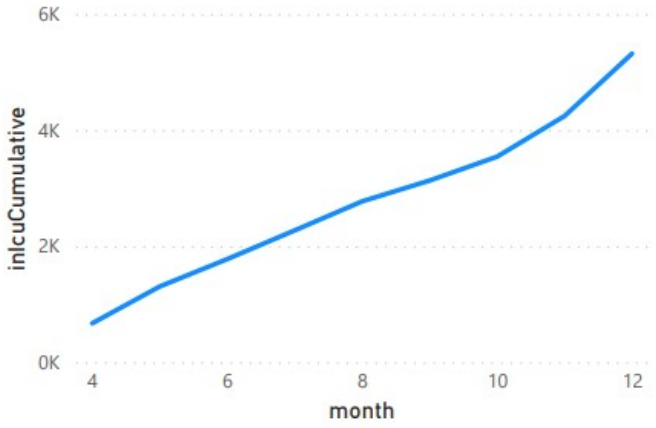
Average of deaths by month



Average of hospitalized by month



Average of inIcuCumulative by month



NicolasMavromatis_3287Project-FINAL

November 3, 2021

[25]:
```
#Nicolas Mavromatis
#nima6629@colorado.edu
#Data from https://covidtracking.com/data/download
```

[26]:
```
import pandas as pd
```

[27]:
```
import os, sys, random, string
```

[28]:
```
import sqlalchemy
from sqlalchemy import *
```

[213]:
```
#Do NOT FORGET to name your schema in MYSQL WS to the same name as dbname*****
#THIS Probably doesn't work, but is here as a server placeholder....
username = "nima6629"

passwd = @@@@@@
host = "applied-sql.cs.colorado.edu"
dbname = @@@@@@@@
```

[214]:
```
db_string = "mysql://{0}:{1}@{2}/{3}".format(
    username, passwd, host, dbname
)
print("Connection string is", db_string)
```

```
Connection string is mysql://nima6629:bowwow!@applied-
sql.cs.colorado.edu/nima6629
```

[215]:
```
try:
    engine = sqlalchemy.create_engine( db_string );
    conn = engine.connect()
except Exception as exp:
    print("Create engine failed:", exp)
```

[32]:
```
#Create database using website,
#then mySQL to upload SQL code....
```

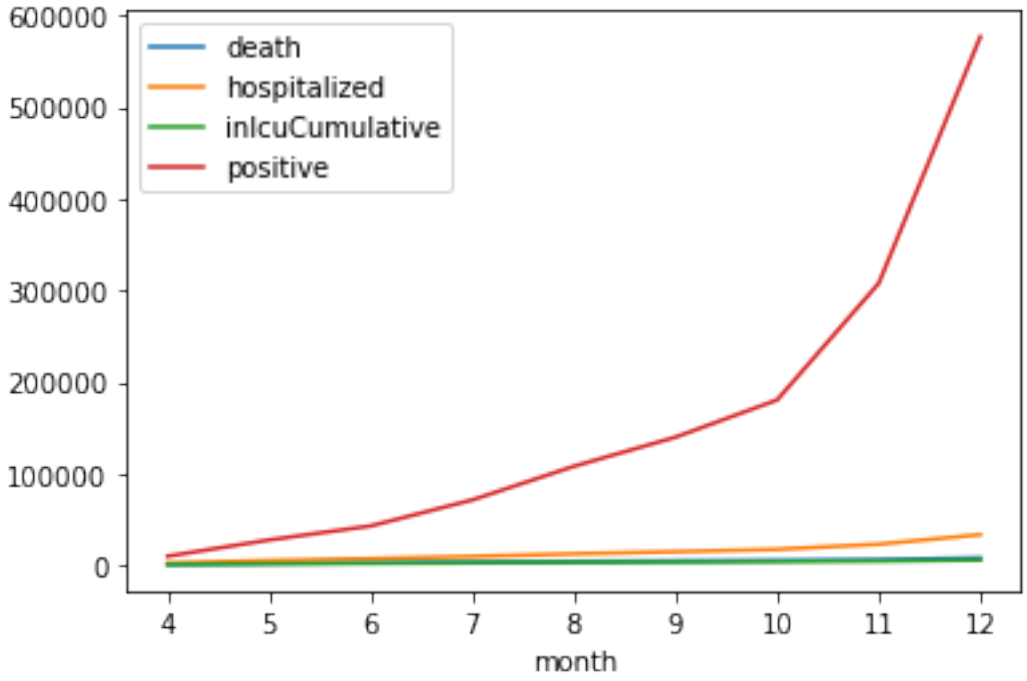
[33]:
```
#Read CSV file into pandas
#Excellent resource:

df=pd.read_csv('OhioCovid2.csv')



#First, show unnecessary Cols.
#State is unnecessary, delete this col.
#...and many more unnecessary columns

#I also cleaned the 2020 data a bit more.

#NOTE: This syntax is crucial. Just doing df.drop(columns='name') DOES NOT WORK
#inplace=true means it is auto applied to the df (you don't need to save a copy␣
 ↪of the new df object)
df.drop('state',  axis='columns', inplace=True)
df.drop('deathConfirmed', axis='columns', inplace=True)
df.drop('deathIncrease', axis='columns', inplace=True)
df.drop('deathProbable', axis='columns', inplace=True)
df.drop('hospitalizedCumulative', axis='columns', inplace=True)
df.drop('hospitalizedCurrently', axis='columns', inplace=True)
df.drop('hospitalizedIncrease', axis='columns', inplace=True)
df.drop('inIcuCurrently', axis='columns', inplace=True)
df.drop('onVentilatorCurrently', axis='columns', inplace=True)
df.drop('positiveCasesViral', axis='columns', inplace=True)
df.drop('positiveIncrease', axis='columns', inplace=True)
df.drop('positiveTestsAntigen', axis='columns', inplace=True)
df.drop('positiveTestsViral', axis='columns', inplace=True)
df.drop('recovered', axis='columns', inplace=True)
df.drop('totalTestResults', axis='columns', inplace=True)
df.drop('totalTestResultsIncrease', axis='columns', inplace=True)
df.drop('totalTestsAntigen', axis='columns', inplace=True)
df.drop('totalTestsViral', axis='columns', inplace=True)
df.drop('totalTestsViralIncrease', axis='columns', inplace=True)


#Print out remaining good columns
for n in df.columns:
    print(n)


#Demonstrate the cleaned data
print(df)
#Now, we need to extract out the month to find the largest increase by month in␣
 ↪relevant fields.
df['month'] = pd.DatetimeIndex(df['date']).month
```

```
#save this raw, cleaned up data into a CSV file to show we did it.
df.to_csv('OhioDataClean.csv', index=False)

#Next, create average groupby(month) column to make sense of things more easily.
monthGroup=df.groupby('month')
#This calculates individual column means separately. This is what we want␣
 ↪anyway, so no biggie.
Means=monthGroup.mean('death')
#Assign DF to this, and output to 3rd CSV
df2=Means
#Use built in percent change method to calculate monthly percent change in␣
 ↪order to say something interesting
#Note: This is stored as a decimal, so 200 percent appears as 2.0
print(df2)
df2.to_csv('OhioDataMeans.csv', index=True)

#Now look at plots
df2.plot()
```

```
date
death
hospitalized
inIcuCumulative
positive
           date  death  hospitalized  inIcuCumulative  positive
0      4/1/2020     65           679              222      2547
1      4/2/2020     81           802              260      2902
2      4/3/2020     91           895              288      3312
3      4/4/2020    102          1006              326      3739
4      4/5/2020    119          1104              346      4043
..          …    …            …                 …        …
270  12/27/2020   8509         36786             5719    670525
271  12/28/2020   8571         37076             5749    675044
272  12/29/2020   8722         37636             5801    682570
273  12/30/2020   8855         38002             5837    690748
274  12/31/2020   8962         38334             5870    700380

[275 rows x 5 columns]
              death  hospitalized  inIcuCumulative       positive
month
4        417.233333   2204.466667       669.566667    9571.866667
5       1603.516129   4849.000000      1303.870968   27337.548387
6       2570.633333   6972.733333      1776.466667   42666.433333
7       3133.161290   9279.193548      2271.290323   71100.516129
8       3829.225806  12202.774194      2772.419355  107855.709677
9       4497.133333  14493.500000      3131.233333  139679.933333
10      5074.741935  17077.032258      3542.806452  180360.322581
```

```
11      5810.466667  22698.766667       4243.833333  307793.900000
12      7755.709677  33123.483871       5319.741935  577271.161290
```

[33]: <AxesSubplot:xlabel='month'>



[34]:
```
#read in percent increase from each prior month
df3=df2.pct_change()
print(df3)
df3.to_csv('OhioPct.csv', index=True)
```

```
           death  hospitalized  inIcuCumulative  positive
month
4            NaN           NaN              NaN       NaN
5       2.843212      1.199625         0.947336  1.856031
6       0.603123      0.437973         0.362456  0.560726
7       0.218829      0.330783         0.278544  0.666427
8       0.222160      0.315068         0.220636  0.516947
9       0.174424      0.187722         0.129423  0.295063
10      0.128439      0.178255         0.131441  0.291240
11      0.144978      0.329199         0.197873  0.706550
12      0.334783      0.459264         0.253523  0.875512
```

[35]:
```
#do a full outer join on 'month' between monthly data and percent increase per␣
 ↪month from prior month
df4=pd.merge(left=df2, right=df3, left_on='month', right_on='month')
```

```
#save as a csv
print(df4)
df4.to_csv('Merged.csv', index=True)
```

```
           death_x  hospitalized_x  inIcuCumulative_x     positive_x  \
month
4       417.233333     2204.466667         669.566667    9571.866667
5      1603.516129     4849.000000        1303.870968   27337.548387
6      2570.633333     6972.733333        1776.466667   42666.433333
7      3133.161290     9279.193548        2271.290323   71100.516129
8      3829.225806    12202.774194        2772.419355  107855.709677
9      4497.133333    14493.500000        3131.233333  139679.933333
10     5074.741935    17077.032258        3542.806452  180360.322581
11     5810.466667    22698.766667        4243.833333  307793.900000
12     7755.709677    33123.483871        5319.741935  577271.161290

         death_y  hospitalized_y  inIcuCumulative_y  positive_y
month
4            NaN             NaN                NaN         NaN
5       2.843212        1.199625           0.947336    1.856031
6       0.603123        0.437973           0.362456    0.560726
7       0.218829        0.330783           0.278544    0.666427
8       0.222160        0.315068           0.220636    0.516947
9       0.174424        0.187722           0.129423    0.295063
10      0.128439        0.178255           0.131441    0.291240
11      0.144978        0.329199           0.197873    0.706550
12      0.334783        0.459264           0.253523    0.875512
```

[36]:
```
#Now, we can upload the dataframe to the server to show we can do it.

df.to_sql('OhioCovid', con=engine, if_exists='replace', index_label=id)

#engine.execute("SELECT * FROM OhioCovid").fetchall()

#Here, we can upload the averages to a different schema (table)

df2.to_sql('OhioCovidMean', con=engine, if_exists='replace', index_label=id)

df3.to_sql('OhioPctIncrease', con=engine, if_exists='replace', index_label=id)

engine.execute("SELECT * FROM OhioCovidMean").fetchall()
```

[36]:
```
[(4, 417.23333333333335, 2204.4666666666667, 669.5666666666667,
9571.866666666667),
 (5, 1603.516129032258, 4849.0, 1303.8709677419354, 27337.548387096773),
 (6, 2570.633333333333, 6972.733333333334, 1776.4666666666667,
```

```
42666.433333333334),
 (7, 3133.1612903225805, 9279.193548387097, 2271.2903225806454,
71100.51612903226),
 (8, 3829.2258064516127, 12202.774193548386, 2772.4193548387098,
107855.70967741935),
 (9, 4497.133333333333, 14493.5, 3131.233333333333, 139679.93333333332),
 (10, 5074.741935483871, 17077.032258064515, 3542.8064516129034,
180360.32258064515),
 (11, 5810.466666666666, 22698.766666666666, 4243.833333333333, 307793.9),
 (12, 7755.709677419355, 33123.48387096774, 5319.741935483871,
577271.1612903225)]
```

[91]:
```
%load_ext sql
%sql sqlite:///dbstring
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

[91]:
```
'Connected: @dbstring'
```

[146]:
```
%%sql
--Now let's do more things in manual MySQL code to show we can
drop table if exists avg;
drop table if exists pct;

--create two new tables, avg and pct, with month being a primary key and ID␣
 ↪being a foreign key NOT NULL constraint.
--ON update FK should return an error
create table avg
(ID int NOT NULL,
 month int,
 death float,
 hosp float,
 icu float,
 pos float,
 PRIMARY KEY(month),
 FOREIGN KEY(ID) REFERENCES pct(ID) ON UPDATE NO ACTION
);

create table pct
(ID int NOT NULL,
 month int,
 death float,
 hosp float,
 icu float,
 pos float,
 PRIMARY KEY(month)
```

```
);
```

```
 * sqlite:///dbstring
Done.
Done.
Done.
Done.
```

[146]:
```
[]
```

[147]:
```
%%sql
--insert the values previously in the DF manually into pct. Must be done first␣
 ↪bc of FK constraint

insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (1, 4, 0, 0, 0, 0);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (2, 5, 2.843211941437065, 1.1996250037802039, 0.9473355420051803, 1.
 ↪8560310479770687);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (3, 6, 0.6031228416048067, 0.4379734653193099, 0.3624558799274289, 0.
 ↪5607263946708456);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (4, 7, 0.21882854691680942, 0.3307827941773809, 0.2785437324542981, 0.
 ↪6664274600493656);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (5, 8, 0.2221604480685282, 0.3150683979072151, 0.2206362732566396, 0.
 ↪5169469301978662);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (6, 9, 0.17442364609483385, 0.1877217237751332, 0.12942269280741558, 0.
 ↪295062947998725);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (7, 10, 0.1284392877278573, 0.1782545456973481, 0.1314412164364107, 0.
 ↪29124003911307583);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (8, 11, 0.1449777625219566, 0.3291985588390116, 0.19787332198214758, 0.
 ↪7065499528721182);
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (9, 12, 0.334782578120981, 0.45926359600893485, 0.2535228218553678, 0.
 ↪8755120270100301);
```

```
 * sqlite:///dbstring
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

```
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

[147]:
```
[]
```

[148]:
```
%%sql

--insert values into avg
--disable foreign keys to create table
PRAGMA foreign_keys = False;

insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (1, 4, 417.23333333333335, 2204.4666666666667, 669.5666666666667, 9571.
 ↪866666666667);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (2, 5, 1603.516129032258, 4849.0, 1303.8709677419354, 27337.
 ↪548387096773);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (3, 6, 2570.633333333333, 6972.733333333334, 1776.4666666666667, 42666.
 ↪433333333334);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (4, 7, 3133.1612903225805, 9279.193548387097, 2271.2903225806454, 71100.
 ↪51612903226);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (5, 8, 3829.2258064516127, 12202.774193548386, 2772.4193548387098,␣
 ↪107855.70967741935);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (6, 9, 4497.133333333333, 14493.5, 3131.233333333333, 139679.
 ↪93333333332);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (7, 10, 5074.741935483871, 17077.032258064515, 3542.8064516129034,␣
 ↪180360.32258064515);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (8, 11, 5810.466666666666, 22698.766666666666, 4243.833333333333, 307793.
 ↪9);
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (9, 12, 7755.709677419355, 33123.48387096774, 5319.741935483871, 577271.
 ↪1612903225);
```

```
 * sqlite:///dbstring
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

```
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

[148]:
```
[]
```

[149]:
```
%%sql
--Now, try to insert a duplicate month into avg table, where month is a PK
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (9, 12, 7755.709677419355, 33123.48387096774, 5319.741935483871, 577271.
 ↪1612903225);
--GET 'unique constraint failed.'
```

```
 * sqlite:///dbstring
```

```
---------------------------------------------------------------------------
IntegrityError                                Traceback (most recent call last)
/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
 ↪_execute_context(self, dialect, constructor, statement, parameters,␣
 ↪execution_options, *args, **kw)
   1770                 if not evt_handled:
-> 1771                     self.dialect.do_execute(
   1772                         cursor, statement, parameters, context

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/default.py in␣
 ↪do_execute(self, cursor, statement, parameters, context)
    716     def do_execute(self, cursor, statement, parameters, context=None):
--> 717         cursor.execute(statement, parameters)
    718

IntegrityError: UNIQUE constraint failed: avg.month

The above exception was the direct cause of the following exception:

IntegrityError                                Traceback (most recent call last)
/tmp/ipykernel_223/894064787.py in <module>
----> 1 get_ipython().run_cell_magic('sql', '', "--Now, try to insert a␣
 ↪duplicate month into avg table, where month is a PK\ninsert INTO avg (ID,␣
 ↪month, death, hosp, icu, pos)\nVALUES (9, 12, 7755.709677419355, 33123.
 ↪48387096774, 5319.741935483871, 577271.1612903225);\n--GET 'unique constraint␣
 ↪failed.'\n")

/opt/conda/lib/python3.9/site-packages/IPython/core/interactiveshell.py in␣
 ↪run_cell_magic(self, magic_name, line, cell)
   2401             with self.builtin_trap:
   2402                 args = (magic_arg_s, cell)
```

```
-> 2403                 result = fn(*args, **kwargs)
   2404             return result
   2405

/opt/conda/lib/python3.9/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

/opt/conda/lib/python3.9/site-packages/IPython/core/magic.py in <lambda>(f, *a,
 ↪**k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/conda/lib/python3.9/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

/opt/conda/lib/python3.9/site-packages/IPython/core/magic.py in <lambda>(f, *a,
 ↪**k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/conda/lib/python3.9/site-packages/sql/magic.py in execute(self, line, cell
 ↪local_ns)
     93
     94         try:
---> 95             result = sql.run.run(conn, parsed['sql'], self, user_ns)
     96
     97             if result is not None and not isinstance(result, str) and
 ↪self.column_local_vars:

/opt/conda/lib/python3.9/site-packages/sql/run.py in run(conn, sql, config,
 ↪user_namespace)
    338             else:
    339                 txt = sqlalchemy.sql.text(statement)
--> 340                 result = conn.session.execute(txt, user_namespace)
```

```
    341             _commit(conn=conn, config=config)
    342             if result and config.feedback:

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪execute(self, statement, *multiparams, **params)
   1261             )
   1262         else:
-> 1263             return meth(self, multiparams, params, _EMPTY_EXECUTION_OPTS)
   1264
   1265     def _execute_function(self, func, multiparams, params,␣
↪execution_options):

/opt/conda/lib/python3.9/site-packages/sqlalchemy/sql/elements.py in␣
↪_execute_on_connection(self, connection, multiparams, params,␣
↪execution_options, _force)
    321     ):
    322         if _force or self.supports_execution:
--> 323             return connection._execute_clauseelement(
    324                 self, multiparams, params, execution_options
    325             )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪_execute_clauseelement(self, elem, multiparams, params, execution_options)
   1450             linting=self.dialect.compiler_linting | compiler.
↪WARN_LINTING,
   1451         )
-> 1452         ret = self._execute_context(
   1453             dialect,
   1454             dialect.execution_ctx_cls._init_compiled,

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪_execute_context(self, dialect, constructor, statement, parameters,␣
↪execution_options, *args, **kw)
   1812
   1813         except BaseException as e:
-> 1814             self._handle_dbapi_exception(
   1815                 e, statement, parameters, cursor, context
   1816             )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪_handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1993                 util.raise_(newraise, with_traceback=exc_info[2],␣
↪from_=e)
   1994             elif should_wrap:
```

```
-> 1995                 util.raise_(

   1996                     sqlalchemy_exception, with_traceback=exc_info[2],␣
 ↪from_=e
   1997                 )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/util/compat.py in␣
 ↪raise_(***failed resolving arguments***)
    205
    206         try:
--> 207             raise exception
    208         finally:
    209             # credit to

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
 ↪_execute_context(self, dialect, constructor, statement, parameters,␣
 ↪execution_options, *args, **kw)
   1769                                 break
   1770                 if not evt_handled:
-> 1771                     self.dialect.do_execute(

   1772                         cursor, statement, parameters, context
   1773                     )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/default.py in␣
 ↪do_execute(self, cursor, statement, parameters, context)
    715
    716     def do_execute(self, cursor, statement, parameters, context=None):
--> 717         cursor.execute(statement, parameters)
    718
    719     def do_execute_no_params(self, cursor, statement, context=None):

IntegrityError: (sqlite3.IntegrityError) UNIQUE constraint failed: avg.month
[SQL: --Now, try to insert a duplicate month into avg table, where month is a P
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (9, 12, 7755.709677419355, 33123.48387096774, 5319.741935483871, 577271.
 ↪1612903225);]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

[150]:
```
%%sql
--Now, try to insert a Null ID into pct
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (NULL, 13, 7755.709677419355, 33123.48387096774, 5319.741935483871,␣
 ↪577271.1612903225);
--GET 'NOT NULL constraint failed: pct.ID'
```

```
* sqlite:///dbstring
```

```
---------------------------------------------------------------------------
IntegrityError                                Traceback (most recent call last)
/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
 ↪_execute_context(self, dialect, constructor, statement, parameters,␣
 ↪execution_options, *args, **kw)
   1770                 if not evt_handled:
-> 1771                     self.dialect.do_execute(
   1772                         cursor, statement, parameters, context

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/default.py in␣
 ↪do_execute(self, cursor, statement, parameters, context)
    716     def do_execute(self, cursor, statement, parameters, context=None):
--> 717         cursor.execute(statement, parameters)
    718

IntegrityError: NOT NULL constraint failed: pct.ID

The above exception was the direct cause of the following exception:

IntegrityError                                Traceback (most recent call last)
/tmp/ipykernel_223/4132084715.py in <module>
----> 1 get_ipython().run_cell_magic('sql', '', "--Now, try to insert a Null ID␣
 ↪into pct\ninsert INTO pct (ID, month, death, hosp, icu, pos)\nVALUES (NULL,␣
 ↪13, 7755.709677419355, 33123.48387096774, 5319.741935483871, 577271.
 ↪1612903225);\n--GET 'NOT NULL constraint failed: pct.ID'\n")

/opt/conda/lib/python3.9/site-packages/IPython/core/interactiveshell.py in␣
 ↪run_cell_magic(self, magic_name, line, cell)
   2401             with self.builtin_trap:
   2402                 args = (magic_arg_s, cell)
-> 2403                 result = fn(*args, **kwargs)
   2404             return result
   2405

/opt/conda/lib/python3.9/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

/opt/conda/lib/python3.9/site-packages/IPython/core/magic.py in <lambda>(f, *a,␣
 ↪**k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
```

```
    189         if callable(arg):

/opt/conda/lib/python3.9/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

/opt/conda/lib/python3.9/site-packages/IPython/core/magic.py in <lambda>(f, *a,
 ↪**k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/conda/lib/python3.9/site-packages/sql/magic.py in execute(self, line, cell
 ↪local_ns)
     93
     94         try:
---> 95             result = sql.run.run(conn, parsed['sql'], self, user_ns)
     96
     97             if result is not None and not isinstance(result, str) and
 ↪self.column_local_vars:

/opt/conda/lib/python3.9/site-packages/sql/run.py in run(conn, sql, config,
 ↪user_namespace)
    338             else:
    339                 txt = sqlalchemy.sql.text(statement)
--> 340                 result = conn.session.execute(txt, user_namespace)
    341             _commit(conn=conn, config=config)
    342             if result and config.feedback:

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in
 ↪execute(self, statement, *multiparams, **params)
   1261             )
   1262         else:
-> 1263             return meth(self, multiparams, params, _EMPTY_EXECUTION_OPT
   1264
   1265     def _execute_function(self, func, multiparams, params,
 ↪execution_options):

/opt/conda/lib/python3.9/site-packages/sqlalchemy/sql/elements.py in
 ↪_execute_on_connection(self, connection, multiparams, params,
 ↪execution_options, _force)
    321     ):
    322         if _force or self.supports_execution:
```

```
--> 323             return connection._execute_clauseelement(

    324                 self, multiparams, params, execution_options
    325             )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪_execute_clauseelement(self, elem, multiparams, params, execution_options)
   1450             linting=self.dialect.compiler_linting | compiler.
↪WARN_LINTING,
   1451         )
-> 1452         ret = self._execute_context(

   1453             dialect,
   1454             dialect.execution_ctx_cls._init_compiled,

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪_execute_context(self, dialect, constructor, statement, parameters,␣
↪execution_options, *args, **kw)
   1812
   1813         except BaseException as e:
-> 1814             self._handle_dbapi_exception(

   1815                 e, statement, parameters, cursor, context
   1816             )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪_handle_dbapi_exception(self, e, statement, parameters, cursor, context)
   1993                 util.raise_(newraise, with_traceback=exc_info[2],␣
↪from_=e)
   1994             elif should_wrap:
-> 1995                 util.raise_(

   1996                     sqlalchemy_exception, with_traceback=exc_info[2],␣
↪from_=e
   1997                 )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/util/compat.py in␣
↪raise_(***failed resolving arguments***)
    205
    206         try:
--> 207             raise exception
    208         finally:
    209             # credit to

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/base.py in␣
↪_execute_context(self, dialect, constructor, statement, parameters,␣
↪execution_options, *args, **kw)
   1769                                 break
   1770                 if not evt_handled:
```

```
-> 1771                     self.dialect.do_execute(
   1772                         cursor, statement, parameters, context
   1773                     )

/opt/conda/lib/python3.9/site-packages/sqlalchemy/engine/default.py in␣
 ↪do_execute(self, cursor, statement, parameters, context)
    715
    716     def do_execute(self, cursor, statement, parameters, context=None):
--> 717         cursor.execute(statement, parameters)
    718
    719     def do_execute_no_params(self, cursor, statement, context=None):

IntegrityError: (sqlite3.IntegrityError) NOT NULL constraint failed: pct.ID
[SQL: --Now, try to insert a Null ID into pct
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (NULL, 13, 7755.709677419355, 33123.48387096774, 5319.741935483871,␣
 ↪577271.1612903225);]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

[151]:
```
%%sql
PRAGMA foreign_keys = True;
--Now, try inserting a value into avg(ID) foreign key, not matched by pct(ID)
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (13, 13, 7755.709677419355, 33123.48387096774, 5319.741935483871, 577271.
 ↪1612903225);
--GET foreign key mismatch - "avg" referencing "pct"
```

```
 * sqlite:///dbstring
Done.
(sqlite3.OperationalError) foreign key mismatch - "avg" referencing "pct"
[SQL: --Now, try inserting a value into avg(ID) foreign key, not matched by
pct(ID)
insert INTO avg (ID, month, death, hosp, icu, pos)
VALUES (13, 13, 7755.709677419355, 33123.48387096774, 5319.741935483871,
577271.1612903225);]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

[152]:
```
%%sql
--But we can insert a new value into the pct(id) field of 13, because it is the␣
 ↪referenced field.
insert INTO pct (ID, month, death, hosp, icu, pos)
VALUES (13, 13, 7755.709677419355, 33123.48387096774, 5319.741935483871, 577271.
 ↪1612903225);
```

```
 * sqlite:///dbstring
Done.
```

[152]: []

[157]:
```
%%sql
PRAGMA foreign_keys=False;
--now we need to update the table and delete that value
DELETE FROM pct WHERE ID==13;
```

```
 * sqlite:///dbstring
Done.
Done.
```

[157]: []

[159]:
```
%%sql
PRAGMA foreign_keys=True;
--now we need to delete a FK row from pct, which activates a trigger
DELETE FROM pct WHERE ID==1;
```

```
 * sqlite:///dbstring
Done.
(sqlite3.OperationalError) foreign key mismatch - "avg" referencing "pct"
[SQL: --now we need to delete a FK row from pct, which activates a trigger
DELETE FROM pct WHERE ID==1;]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

[160]:
```
%%sql
--now we need to delete a FK row from avg, which activates a trigger
PRAGMA foreign_keys=True;
DELETE FROM avg where ID=1
```

```
 * sqlite:///dbstring
Done.
(sqlite3.OperationalError) foreign key mismatch - "avg" referencing "pct"
[SQL: DELETE FROM avg where ID=1]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

[161]:
```
%%sql
select * from pct;
```

```
 * sqlite:///dbstring
Done.
```

[161]:
```
[(1, 4, 0.0, 0.0, 0.0, 0.0),
 (2, 5, 2.843211941437065, 1.1996250037802039, 0.9473355420051803,
1.8560310479770687),
 (3, 6, 0.6031228416048067, 0.4379734653193099, 0.3624558799274289,
0.5607263946708456),
 (4, 7, 0.21882854691680942, 0.3307827941773809, 0.2785437324542981,
```

```
0.6664274600493656),
 (5, 8, 0.2221604480685282, 0.3150683979072151, 0.2206362732566396,
0.5169469301978662),
 (6, 9, 0.17442364609483385, 0.1877217237751332, 0.12942269280741558,
0.295062947998725),
 (7, 10, 0.1284392877278573, 0.1782545456973481, 0.1314412164364107,
0.29124003911307583),
 (8, 11, 0.1449777625219566, 0.3291985588390116, 0.19787332198214758,
0.7065499528721182),
 (9, 12, 0.334782578120981, 0.45926359600893485, 0.2535228218553678,
0.8755120270100301)]
```

[162]:
```
%%sql
select * from avg;
```

```
 * sqlite:///dbstring
Done.
```

[162]:
```
[(1, 4, 417.23333333333335, 2204.4666666666667, 669.5666666666667,
9571.866666666667),
 (2, 5, 1603.516129032258, 4849.0, 1303.8709677419354, 27337.548387096773),
 (3, 6, 2570.633333333333, 6972.733333333334, 1776.4666666666667,
42666.433333333334),
 (4, 7, 3133.1612903225805, 9279.193548387097, 2271.2903225806454,
71100.51612903226),
 (5, 8, 3829.2258064516127, 12202.774193548386, 2772.4193548387098,
107855.70967741935),
 (6, 9, 4497.133333333333, 14493.5, 3131.233333333333, 139679.93333333332),
 (7, 10, 5074.741935483871, 17077.032258064515, 3542.8064516129034,
180360.32258064515),
 (8, 11, 5810.466666666666, 22698.766666666666, 4243.833333333333, 307793.9),
 (9, 12, 7755.709677419355, 33123.48387096774, 5319.741935483871,
577271.1612903225)]
```

[176]:
```
%%sql
--now do a join on the two tables using ID
DROP TABLE IF EXISTS result;

CREATE TABLE result AS
  SELECT
    avg.ID,
    avg.month,
    avg.death AS avgDeath,
    avg.hosp AS avgHosp,
    avg.icu AS avgICU,
    avg.pos AS avgPos,
    pct.death AS pctDeath,
```

```
    pct.hosp AS pctHosp,
    pct.icu AS pctICU,
    pct.pos AS pctPos
    FROM
    avg INNER JOIN pct
    WHERE avg.ID = pct.ID;
```

```
 * sqlite:///dbstring
Done.
Done.
```

[176]:
```
[]
```

[199]:
```
%%sql
select * from result;
```

```
 * sqlite:///dbstring
Done.
```

[199]:
```
[(1, 4, 417.23333333333335, 2204.4666666666667, 669.5666666666667,
9571.866666666667, 0.0, 0.0, 0.0, 0.0),
 (2, 5, 1603.516129032258, 4849.0, 1303.8709677419354, 27337.548387096773,
2.843211941437065, 1.1996250037802039, 0.9473355420051803, 1.8560310479770687),
 (3, 6, 2570.633333333333, 6972.733333333334, 1776.4666666666667,
42666.433333333334, 0.6031228416048067, 0.4379734653193099, 0.3624558799274289,
0.5607263946708456),
 (4, 7, 3133.1612903225805, 9279.193548387097, 2271.2903225806454,
71100.51612903226, 0.21882854691680942, 0.3307827941773809, 0.2785437324542981,
0.6664274600493656),
 (5, 8, 3829.2258064516127, 12202.774193548386, 2772.4193548387098,
107855.70967741935, 0.2221604480685282, 0.3150683979072151, 0.2206362732566396,
0.5169469301978662),
 (6, 9, 4497.133333333333, 14493.5, 3131.233333333333, 139679.93333333332,
0.17442364609483385, 0.1877217237751332, 0.12942269280741558,
0.295062947998725),
 (7, 10, 5074.741935483871, 17077.032258064515, 3542.8064516129034,
180360.32258064515, 0.1284392877278573, 0.1782545456973481, 0.1314412164364107,
0.29124003911307583),
 (8, 11, 5810.466666666666, 22698.766666666666, 4243.833333333333, 307793.9,
0.1449777625219566, 0.3291985588390116, 0.19787332198214758,
0.7065499528721182),
 (9, 12, 7755.709677419355, 33123.48387096774, 5319.741935483871,
577271.1612903225, 0.334782578120981, 0.45926359600893485, 0.2535228218553678,
0.8755120270100301)]
```

[174]:
```
%%sql
--create an index on id to demonstrate
CREATE INDEX i1 ON result(ID);
```

```
 * sqlite:///dbstring
(sqlite3.OperationalError) index i1 already exists
[SQL: --create an index on id to demonstrate
CREATE INDEX i1 ON result(ID);]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

[207]:
```
%%sql
--now find 4 absolute max positive values
--the numbers were highest in the cold months
select month, max(avgPos) AS MaxPos from result
GROUP BY month
ORDER BY MaxPos DESC limit 4;
```

```
 * sqlite:///dbstring
Done.
```

[207]:
```
[(12, 577271.1612903225),
 (11, 307793.9),
 (10, 180360.32258064515),
 (9, 139679.93333333332)]
```

[209]:
```
%%sql
--now find the highest 3 percent increases per month from the previous month
--The highest percent increases were after the first month and in the cold␣
 ↪months predictably
select month, max(pctPos) AS MaxPosIncrease from result
GROUP BY month
ORDER BY MaxPosIncrease DESC limit 3;
```

```
 * sqlite:///dbstring
Done.
```

[209]:
```
[(5, 1.8560310479770687), (12, 0.8755120270100301), (11, 0.7065499528721182)]
```