C. Marisol Ramirez

Vaccine Python Script Documentation

Geog 375

May 12, 2021

Summary

The efficacy of the Covid-19 vaccine is a controversial topic. The United States population is divided into those who are pro-vaccine and those who are anti-vaccine. Within the pro-vaccine population, there are sub-groups labeled by vaccine brands. Individuals often believe one brand is more effective and less of a risk than the others. Sometimes they simply go with what is available at the time. Nevertheless, the comparison between the vaccine brands is a frequent and relevant topic. Since the beginning of the 2020 pandemic, there have been multiple brands of vaccines developed with hope of defeating the virus. Only three of those brands have been approved by the F.D.A. in the U.S. for emergency use. These brands are Pfizer, Moderna, and Johnson & Johnson. For this project, a python script was developed to assist in sorting through the total number of vaccines administered for each brand.

Purpose

The overall goal of the Python script was to gain a better understanding of how the vaccine brands compare to one another. While there are various ways of viewing and analyzing the data on vaccines administered, this script narrowed in on a specific date of administration. The original data included the total number of vaccines administered for each brand by county and date, the combined total number of vaccines administered by county and date, and the cumulative total of all three brands since the first day vaccines were administered. The desired outcome would be to map the number of vaccines administered by brand on a given day. The data could be normalized by dividing each brand total by the combined total to see percentages. The convenience of the script allows the user to filter the data based on any chosen date and to produce multiple maps accordingly. Being able to focus on a more defined search could provide a clarified comparison of vaccine brands and rates of usage.

Methods Used

Before manipulating the vaccine data, a California county shapefile was downloaded from the California open data portal. The shapefile was then made into a feature layer using the “make feature layer” function. The shapefile was then narrowed down to only use data from California counties. The original file had counties from the entire United States. California counties were selected using a query along with the “select layer by attribute” function . To ensure accuracy, the “get count” function was then input and a print statement was attached to ensure all 58 California counties were selected. The selected features were copied to their own shapefile with their attributes written out to a database table (figure A).

Figure A

```
# Copy the selected polygon features to a new shapefile
if arcpy.Exists(ca_counties):
    arcpy.Delete_management(ca_counties)

arcpy.CopyFeatures_management(counties_flayer, ca_counties)
print ("Copied selected features from " + counties_flayer  + " to " + ca_counties)

# Write out selected attributes to a table file (DBF)
if arcpy.Exists(ca_county_attributes):
    arcpy.Delete_management(ca_county_attributes)

arcpy.CopyRows_management(counties_flayer, ca_county_attributes)
print ("Copied selected attributes from " + counties_flayer + " to " + ca_county_attributes)
```

After prepping the county shapefile, the vaccines administered data was addressed. Since the vaccine data was downloaded as an excel file, the spreadsheet was first converted into a database table. This was done using the “excel to table conversion” function. The table was then used to make a table view. The table view was created using the “make table view” function. The table view would make the vaccine data “joinable” data.

Since the California counties were selected and exported to a new shapefile earlier, a new feature layer had to be created from the new shapefile. This new feature layer of only the selected counties was made using the same function used prior, the “make feature layer” function. This allowed the shapefile to be joined to the vaccine table. Both the feature class and the table had a county field within the data. Using the “Name” field from the county shapefile and the “County” field from the vaccine data, the data was joined together. This was done using the “Add join” function (figure B).

Figure B

```
# Join Ca Counties fl to vaccines table view
arcpy.AddJoin_management(ca_counties_flayer, "NAME", vaccines_tv, "county", "KEEP_COMMON")
```

For practical use later, the “list fields” function was then inserted into the script. The list was printed in the Python Shell (figure C) which allowed easier selection in the next steps.

Figure C

```
ca_counties.FID
ca_counties.Shape
ca_counties.STATEFP
ca_counties.COUNTYFP
ca_counties.COUNTYNS
ca_counties.AFFGEOID
ca_counties.GEOID
ca_counties.NAME
ca_counties.LSAD
ca_counties.ALAND
ca_counties.AWATER
vaccines_table.OBJECTID
vaccines_table.county
vaccines_table.administered_date
vaccines_table.total_doses
vaccines_table.cumulative_total_doses
vaccines_table.pfizer_doses
vaccines_table.cumulative_pfizer_doses
vaccines_table.moderna_doses
vaccines_table.cumulative_moderna_doses
vaccines_table.jj_doses
vaccines_table.cumulative_jj_doses
vaccines_table.partially_vaccinated
vaccines_table.total_partially_vaccinated
vaccines_table.fully_vaccinated
vaccines_table.cumulative_fully_vaccinated
vaccines_table.at_least_one_dose
vaccines_table.cumulative_at_least_one_dose
vaccines_table.california_flag
```

A new empty table was then created and saved to a file geodatabase. This table was created to insert only the desired fields from the joined table. This would allow better data management, in theory, depending on the project. The new table was created using the "create table" function. Fields were then added to the table using the "add fields" function and a field list was defined using a variable "field_list." Figure D shows the specific fields added to the new table.

Figure D

```
print ('Will create a table named:', newcounty_table, 'in ', output_path)

arcpy.CreateTable_management(output_path, newcounty_table)


#add fields to the table
arcpy.AddField_management(vcounty_table, 'FID', 'SHORT')
arcpy.AddField_management(vcounty_table, 'County', 'TEXT', '', '', 50)
arcpy.AddField_management(vcounty_table, 'Total_Doses', 'LONG')
arcpy.AddField_management(vcounty_table, 'Pfizer_Doses', 'SHORT')
arcpy.AddField_management(vcounty_table, 'Moderna_Doses', 'SHORT')
arcpy.AddField_management(vcounty_table, 'JJ_Doses', 'SHORT')


print ('Created table and added fields')

field_list = ['FID', 'County', 'Total_Doses', 'Pfizer_Doses', 'Moderna_Doses', 'JJ_Doses']
```

The script was then automated to search through the joined table and insert specific data into the new table. In this case a specific date was chosen to narrow done results. This was done by setting up an "insert cursor," followed by a "search cursor" (figures E & F). A query was used to select only vaccines administered on the selected date.

Figure E

```
# Create cursor object
with arcpy.da.InsertCursor(vcounty_table, field_list) as irows:

    #prep search cursor
    #for field in fields:
       # print (field.name)
    query = """administered_date" = timestamp '2021-03-01 00:00:00'"""
    field_list_joined = ['ca_counties.FID', \
                         'vaccines_table.county', \
                         'vaccines_table.total_doses', \
                         'vaccines_table.pfizer_doses', \
                         'vaccines_table.moderna_doses', \
                         'vaccines_table.jj_doses']
```

Figure F

```
#create search cursor
with arcpy.da.SearchCursor(ca_counties_flayer, field_list_joined, query) as srows:

    for srow in srows:

        ID= srow[0] #ca_counties.FID
        CName = srow[1] # vaccines_table.county
        Total = srow[2] # vaccines_table.total_doses
        Pfizer= srow[3] # vaccines_table.pfizer_doses
        Moderna = srow[4] # vaccines_table.moderna_doses
        JJ = srow[5] # vaccines_table.jj_doses

        print (ca_counties + ' FID is: ' + str(ID))
        print (vaccines_table + ' County is: ' + CName)
        print (vaccines_table + ' Total Doses is: ' + str(Total))
        print (vaccines_table + ' Pfizer Doses is: ' + str(Pfizer))
        print (vaccines_table + ' Moderna Doses is: ' + str(Moderna))
        print (vaccines_table + ' Johnson + Johnson Doses is: ' + str(JJ))

        #insert into new table
        irows.insertRow((ID, CName, Total, Pfizer, Moderna, JJ))
```

The last steps taken were for safe keeping and data clean up purposes. The joined table created from the county shapefile and vaccine feature class earlier, was exported into its own feature class. This was done using the “copy features” function. The join was then removed from the original shapefile and table using the “remove join” function. Throughout the script, multiple if/then statements were included to ensure that previous tables, feature classes, feature layers, table views, and joins were deleted prior to re-running the python script.

Issues Encountered

The main issue encountered was the outcome of the script. The new table was created but little analysis could be completed from it. Although the insert cursor worked well in the end, the output table was not geocoded, meaning no maps could be made from it directly. One resolution would be to geocode the table in ArcPro. Another option would be to add spatial coordinates to the table which could potentially be done within the script itself. This would create a new feature class as opposed to just an info table.

If the table were to be geocoded and maps were to be produced from the data, still only little analysis could be done with the given data. Since the table that was created was queried to select a specific administration date, findings could only be made for that one day. One way of resolving this issue would be focus more on the cumulative total of vaccines administered by the latest date available rather than using data pertaining to one day at a time. This would give a better understanding of the vaccine brands in comparison to one another as a whole. There are too many variables that offset the value of one day’s totals.

Overall, creating this script was a intensive lesson in how to write Python code in general. Some of the initial problems encountered involved syntax. One of my first If/Then statements did not work when running the script due to a lack of a closing parenthesis. This occurred again later in the script when running the insert cursor. The computer could not find one of my defined variables. After reformatting syntax, renaming variables, checking output paths, etc. it came down to a missing comma within the “add fields” function. After hours of problem solving, the script ran smoothly.