Cleaning Minerals Pull Data for Minerals Field Inspections

Cat Schooley
GIS Analyst
November 5, 2021

Using Pull Data

One of the capabilities of Survey123 is a feature called 'Pull Data'. I should note this is only available in Survey123 Connect, not the online survey creator. Pull data allows the creators of the survey to connect a seperate CSV file to the backend of the survey that the survey can then pull information from. The pull data allows a sort of auto-fill feature to be used. A user can input one piece of information, say an email address, and the survey will grab all the other information related to that survey (name, address, phone number, etc.) and fill it in. This has to be coded into the XLS file of course. It may seem miniscle, but this approach leads to less errors caused by typos, provides up-to-date information (if the CSV is updated regularly). Also a minute saved at the beginning of each survey adds up.

I'll break down the code and why it's written the way it is.

Pulling Data into the Python Script

Google sheets has become more and more widely used as the standard for sharing of information formally stored in programs like Microsoft Excel. It's on the cloud, easy to use, easy to collaborate, and easy to track changes. Where this can be problematic is pulling it into scripts that aren't Google Scripts. There is a Google API that makes this easier, but requires a subscription for some functions. There is a grace credit amount ($200) for the Google Sheets API, then you are billed for any credit use after that. My organization at this time is not a fan of this and therefore has a block to using the Google APIs. For more information on the Google Sheets API check this link (https://apipheny.io/google-sheets-api/) .

Due to this rescriction, I have to use a work around. When you use the export capabilities of Google Sheets, there is a hidden URL (I use hidden loosely). At the time of this writing it looks something like this for exporting CSVs from the web.

```
docs.google.com/spreadsheets/d/{SPREADSHEETID}/export?format=csv&id{SPREADSHEETID}&
`**gid=2113599541**` < Notice this section here
```

The gid refers to the sheet id and this changes if a Google Sheet is completely overwritten. This is often done by people using the Google Drive Desktop. They go in and just overwrite the whole thing instead of making changes to the sheet itself. If this gid changes, your code will no longer work. I'm sure this isn't a common issue but I ran into so I found a different route. I'll go over both routes below.

As always, pull in the module you need for the script. For the first part of this script we use the pandas module. Pandas is often pulled in as pd. I'm not sure why, but it's what we do.

In [ ]:
```
### import modules ###

import pandas as pd
```

Pull Data sheet directly from Google Sheet using the url

First, I'll show you how to pull directly from the web. As stated above, this doesn't work well for my use case but can be used for other workflows. To do this you have to write the url the same as the url when exporting the google sheet to a CSV. Google has been known to change this formatting though so if it doesn't work check out Stack Overflow (https://stackoverflow.com/) or other online resources for an updated version.

When you visit the google sheet you'll see a url that looks like this:

```
"https://docs.google.com/spreadsheets/d/{SPREADSHEETID}/edit#gid=2113599541"
```

You will have to insert this between the spreadsheet id and the page id:

```
"/export?format=csv&id{SPREADSHEETID}&"
```

In order to get this final url to use in the code:

```
"https://docs.google.com/spreadsheets/d/{SPREADSHEETID}/export?format=csv&id{SPREAD
SHEETID}&gid=2113599541"
```

Now that we have the url we need we can go ahead and code it in.

```
url = "https://docs.google.com/spreadsheets/d/{SPREADSHEETIDENTIFIER}/export?format
=csv&id{SPREADSHEETIDENTIFIER}&gid=2113599541"
```

I'm selecting for the columns that I want and not filtering out any NA values

```
df = pd.read_csv(url, dtype = object, usecols= [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], n
a_filter = False)
```

Pull Data from local Google Drive Desktop location

Now I'm going to show you how to do it using Google Drive Desktop (https://www.google.com/drive/download/) which circumvents the issue of a changing page id.

In [ ]:
```
# This is my T:/ Drive, where the excel workbook is saved

file = '{MYGOOGLEDOC}.xlsm'

# Here I call the file, the sheet from the workbook, the dtype for the datafra
me and the columns

df = pd.read_excel(file, sheet_name="For Pull Data", dtype = str, usecols= [3,
4, 5, 6, 7, 8, 9, 10, 11, 12], na_filter = False, index_col= None)

# This is a bit redundant but makes sure that the data in the dataframe is rea
d as a 'string'
df = df.astype(str)
```

Remove Whitespaces and commas

When data is pulled into pandas you should do a cleaning of the data. Python is sensitive to white spaces and tables can be sensitive to some special characters such as commas in the case of CSVs. Keep that in mind whenever performing tasks like this.

`str.strip` is used in the function below. This is why it was so important to have the data in the dataframe read as a string. Now you'll see if you were to use `df.dtypes` you would see that each field is of dtype object. This is because there's some columns in the database I use that have both numbers and letters as well as some columns with large empty spaces. There is also a total row that appears at the bottom. All of these make it difficult for pandas to know if these of of type `str`, `int`, or something else so it chooses the catch-all `object`. Since we have the dataframe reading as a string, we can use the `.str` function and `.replace` function to clean up our data.

In [ ]:
```
def whitespaceRemover(dataframe):

    # iterating over the columns
    for i in dataframe.columns:

        # checking datatype of each columns
        if dataframe[i].dtype == 'object': #helps pinpoint if something is wro
ng with the data type

            # applying strip function on column
            dataframe[i] = dataframe[i].map(str.strip)

            dataframe[i] = dataframe[i].str.replace(",","")

        else:

            # if condn. is False then it will do nothing.
            pass

            print(f"{i} was skipped because field value was not a string")

### End Function ###

print("Commas and whitespaces being removed...")

whitespaceRemover(df) # plug your dataframe into function
```

Overwriting Pull Data

After it's clean we can write it to a CSV file overwriting the previous pull data table in your survey. Notice the location of I am writing the new CSV to. The Media folder holds items used in various ArcGIS Survey123 workflows. The Media folder can contain offline basemaps that survey authors want downloaded with a specific survey in the ArcGIS Survey123 Field App. Additional workflows that use the Media folder include image questions that use the draw and annotate appearance, or consuming images directly in your survey.

Please note that all care should be taken to ensure the files you are updating with this script have the exact same name as the files in the originally published media folder. Also, the files should have the same format, with the same field names as the original files (in the case of CSV files). Only rows of data should be updated or additional rows added. If you want to change the format of the files (rename or add columns) you should update the files via Connect in the media folder, update the survey XLS Form, and then re-publish the survey to ensure the changes do not break anything.

It is recommended to test the script on a backup copy of your survey and ensure the survey can be downloaded and updated in the field app, checking that the external choice lists and/or media items are working as expected. This should be confirmed before running the script and updating the media folder and files on your real survey that is currently in use by other ArcGIS Survey123 Field App users.

In [ ]:
```
print("Overwriting old Minerals Pulldata with cleaned Minerals Pulldata...")

df.to_csv(r"C:\Users\cschooley\ArcGIS\My Survey Designs\PullUpdate\MineralsPulldata.csv", index = False)
```

Replacing Online Survey123 Pull Data with Local Update

To start we will import the required modules and define our variables.
Please format folder directories as C:/Users/username… include a trailing slash after the final folder name as the updated file directory is concatenated to the updated file name

The variables are defined as follows:

- **portalURL** - The URL for your WebGIS Portal (ex. www.arcgis.com)
- **username** - Your WebGIS Portal username (ex. gisadmin)
- **password** - You WebGIS Portal password (ex. gisadmin1)
- **itemID** - The Item ID for the ArcGIS Survey123 Form Item in you WebGIS Portal (ex. 89bc8c7844e548e09baa3aad4695e78b)
- **download_folder** - The folder directory you would like the Survey123 Form Item to be downloaded with the trailing slash (ex. C:/temp/)
- **updated_file** - The updated file name containing the extension (ex. pullData.csv)
- **source_loc** - Folder directory where the updated file is located (ex. C:/users/username/arcgis/my survey designs/)

In [ ]:
```
### Modules ###

import arcgis
from arcgis.gis import GIS
import zipfile
import shutil
import arcpy
import os

token = arcpy.GetSigninToken()
if token is not None:
    print(token['token'])

portalURL = r'https://arcgis.com'
username = 'username'
password = 'password'
itemID = 'itemID'
download_folder = r'C:/temp/'
updated_file = r'pullData.csv'
source_loc = r'C:\Users\cschooley\ArcGIS\My Survey Designs\PullUpdate/'# this
 is where mine is saved. PullUpdate is a custom folder I added


### Connect to GIS ###

gis = GIS(portalURL, username, password, verify_cert=False)
```

Download the survey

We will start by grabbing the properties of the Survey123 Form Item. These properties are used later when we update the Form Item with a zip file containing the new media content.

In [ ]:
```
survey_manager = arcgis.apps.survey123.SurveyManager(gis)
surveyId = survey_manager.get(itemID)
surveyProp = surveyId.properties

# Find the Form item in the gis and download as a zip file to the *download_fo
lder* directory.

itm = arcgis.gis.Item(gis,itemID)
print(itm)
savedZip = itm.download(save_path=download_folder)
```

Extract the zip file to an *_extracted* folder in the download location. This *_extracted* folder is where the updated media files will be copied and rezipped later on.

In [ ]:
```
def extractZIP(filename,folder):
    zfile = zipfile.ZipFile(filename)
    zfile.extractall(folder)

extractZIP(savedZip, download_folder + "_extracted/")
```

Copy the updated file to the media folder replacing the old file.

In [ ]:
```
source_file = source_loc + updated_file
dest_file = download_folder + "_extracted/esriinfo/media/" + updated_file
shutil.copyfile(source_file, dest_file)
print (updated_file + " updated to: " + download_folder + "_extracted/esriinf
o/media/")
```

Delete the old zip file that was previously downloaded. This will prevent any namespace issues and ensure the process of zipping and uploading the updated survey goes smoothly.

In [ ]:
```
os.remove(savedZip)
print ("Old zip file deleted from: " + download_folder)
```

Upload the updated survey

We will now zip the updated survey and place it in the download folder. The code below grabs the survey title from the survey properties and passes it into the function which zips the updated survey contents to the download folder.

In [ ]:
```
zipFileName = surveyProp['title']
os.chdir(download_folder)
updateZip = shutil.make_archive(zipFileName, 'zip', download_folder + '_extrac
ted/')
print (updateZip)
```

Upload the new zip file and update the Form Item with the new Media folder content.

Then, clean up intermediate data. This process will delete the updated zip file as well as the extracted folder containing the unzipped survey content.

In [ ]:
```
itm.update({},updateZip)

os.remove(updateZip)
print (zipFileName + " deleted from: " + download_folder)

shutil.rmtree(download_folder + "_extracted/")
print ("extracted folder deleted from: " + download_folder)
print (zipFileName + " successfully updated with " + source_file + " and uploa
ded to your ArcGIS portal!")
```

Send Email Notifications

This can also be done with a Google API that is safer and requires stricter authentication. Due to my organization's reluctance I used another work around. Be aware that using Yagmail requires you change some settings in your Gmail account. Here's an article (https://mailtrap.io/blog/yagmail-tutorial/) to help you use Yagmail. Use this link (https://developers.google.com/gmail/api/guides/sending#python) to find more information about the Gmail API.

In [ ]:
```
### Modules ###

import yagmail
from datetime import date
from datetime import datetime

print("Sending Emails...")
now = datetime.now()
date = now.strftime("%m/%d/%Y")
time = now.strftime("%I:%M:%S %p")
reciever = ['email1', 'email2', 'email3']
body = f'Hello,\n\n The {zipFileName} survey successfully updated with the lat
est Pulldata available as of {date} and uploaded to the {username}'s' account!
\n\nCompleted on {date} at {time}.\n\nThank you!'
yag = yagmail.SMTP("email", 'password')
for recipient in reciever:
    yag.send(
        to=recipient,
        subject='subject',
        contents = body
    )

print('Emails sent successfully.')
```

Full Code

In [ ]:

```
#!/usr/bin/env python
# coding: utf-8

# # Cleaning Minerals Pull Data for Minerals Field Inspections

# Cat Schooley
# GIS Analyst
# November 5, 2021
#
# ### Using Pull Data

### import modules ###

import pandas as pd



### Remember the url format ###

# "https://docs.google.com/spreadsheets/d/{SPREADSHEETID}/export?format=csv&id
{SPREADSHEETID}&gid=2113599541"


###### Uncomment this section to use URL instead of Google Drive Desktop #####
#

# url = "https://docs.google.com/spreadsheets/d/{SPREADSHEETIDENTIFIER}/expor
t?format=csv&id{SPREADSHEETIDENTIFIER}&gid=2113599541"


# df = pd.read_csv(url, dtype = object, usecols= [3, 4, 5, 6, 7, 8, 9, 10, 11,
12], na_filter = False)

##############################################################################
##



###### Comment out this section to use URL instead of Google Drive Desktop ###
###

file = '{MYGOOGLEDOC}.xlsm'


df = pd.read_excel(file, sheet_name="For Pull Data", dtype = str, usecols= [3,
4, 5, 6, 7, 8, 9, 10, 11, 12], na_filter = False, index_col= None)

##############################################################################

# Ensure dataframe is read as a 'string'
df = df.astype(str)



##### Remove Whitespaces and commas #####
```

```
def whitespaceRemover(dataframe):

    # iterating over the columns
    for i in dataframe.columns:

        # checking datatype of each columns
        if dataframe[i].dtype == 'object': #helps pinpoint if something is wro
ng with the data type

            # applying strip function on column
            dataframe[i] = dataframe[i].map(str.strip)

            dataframe[i] = dataframe[i].str.replace(",","")

        else:

            # if condn. is False then it will do nothing.
            pass

            print(f"{i} was skipped because field value was not a string")

### End Function ###

print("Commas and whitespaces being removed...")

whitespaceRemover(df) # plug your dataframe into function


##### Overwriting Pull Data #####


print("Overwriting old Minerals Pulldata with cleaned Minerals Pulldata...")

df.to_csv(r"C:\Users\cschooley\ArcGIS\My Survey Designs\PullUpdate\MineralsPul
ldata.csv", index = False)


# #### Replacing Online Survey123 Pull Data with Local Update
#
# To start we will import the required modules and define our variables.<br>
# **Please format folder directories as C:/Users/username… include a trailing
 slash after the final folder name as the updated file directory is concatenat
ed to the updated file name**
#
#
# The variables are defined as follows:
#
# * **portalURL** - The URL for your WebGIS Portal (ex. www.arcgis.com)
# * **username** - Your WebGIS Portal username (ex. gisadmin)
# * **password** - You WebGIS Portal password (ex. gisadmin1)
# * **itemID** - The Item ID for the ArcGIS Survey123 Form Item in you WebGIS
 Portal (ex. 89bc8c7844e548e09baa3aad4695e78b)
# * **download_folder** - The folder directory you would like the Survey123 Fo
rm Item to be downloaded with the trailing slash (ex. C:/temp/)
```

```
# * **updated_file** - The updated file name containing the extension (ex. pul
lData.csv)
# * **source_loc** - Folder directory where the updated file is located (ex.
 C:/users/username/arcgis/my survey designs/)



### Modules ###

import arcgis
from arcgis.gis import GIS
import zipfile
import shutil
import arcpy
import os

token = arcpy.GetSigninToken()
if token is not None:
    print(token['token'])

portalURL = r'https://arcgis.com'
username = 'username'
password = 'password'
itemID = 'itemID'
download_folder = r'C:/temp/'
updated_file = r'pullData.csv'
source_loc = r'C:\Users\cschooley\ArcGIS\My Survey Designs\PullUpdate/'# this
 is where mine is saved. PullUpdate is a custom folder I added


### Connect to GIS ###

gis = GIS(portalURL, username, password, verify_cert=False)


# #### Download the survey
#
# We will start by grabbing the properties of the Survey123 Form Item.
# These properties are used later when we update the Form Item with a zip file
# containing the new media content.

survey_manager = arcgis.apps.survey123.SurveyManager(gis)
surveyId = survey_manager.get(itemID)
surveyProp = surveyId.properties

# Find the Form item in the gis and download as a zip file to the
# *download_folder* directory.

itm = arcgis.gis.Item(gis,itemID)
print(itm)
savedZip = itm.download(save_path=download_folder)


# Extract the zip file to an *`_extracted`* folder in the download location.
# This *`_extracted`* folder is where the updated media files will be copied
# and rezipped later on.
```

```
def extractZIP(filename,folder):
    zfile = zipfile.ZipFile(filename)
    zfile.extractall(folder)

extractZIP(savedZip, download_folder + "_extracted/")


# Copy the updated file to the media folder replacing the old file.


source_file = source_loc + updated_file
dest_file = download_folder + "_extracted/esriinfo/media/" + updated_file
shutil.copyfile(source_file, dest_file)
print (updated_file + " updated to: " + download_folder + "_extracted/esriinf
o/media/")


# Delete the old zip file that was previously downloaded. This will prevent an
y namespace issues and ensure the process of zipping and uploading the updated
survey goes smoothly.


os.remove(savedZip)
print ("Old zip file deleted from: " + download_folder)


##### Upload the updated survey #####


zipFileName = surveyProp['title']
os.chdir(download_folder)
updateZip = shutil.make_archive(zipFileName, 'zip', download_folder + '_extrac
ted/')
print (updateZip)


# Upload the new zip file and update the Form Item with the new Media folder c
ontent.

# Then, clean up intermediate data. This process will delete the updated zip f
ile as well as the extracted folder containing the unzipped survey content.


itm.update({},updateZip)

os.remove(updateZip)
print (zipFileName + " deleted from: " + download_folder)

shutil.rmtree(download_folder + "_extracted/")
print ("extracted folder deleted from: " + download_folder)
print (zipFileName + " successfully updated with " + source_file + " and uploa
ded to your ArcGIS portal!")


###### Send Email Notifications ######
```

```
### Modules ###

import yagmail
from datetime import date
from datetime import datetime

print("Sending Emails...")
now = datetime.now()
date = now.strftime("%m/%d/%Y")
time = now.strftime("%I:%M:%S %p")
reciever = ['email1', 'email2', 'email3']
body = f'Hello,\n\n The {zipFileName} survey successfully updated with the lat
est Pulldata available as of {date} and uploaded to the {username}'s' account!
\n\nCompleted on {date} at {time}.\n\nThank you!'
yag = yagmail.SMTP("email", 'password')
for recipient in reciever:
    yag.send(
        to=recipient,
        subject='subject',
        contents = body
    )

print('Emails sent successfully.')
```