Script	Explanation
import requests import pandas as pd from datetime import datetime import os import sys import json import gspread from google.oauth2.service_account import Credentials	Import packages needed to run the script

filename = 'data.json'
if os.path.exists(filename):
  with open(filename) as f:
  content = f.read()
  if not content.strip():
  raise ValueError("JSON input is empty.")
  data_json = json.loads(content)
else:
  raise FileNotFoundError(f"{filename} does not exist.")
	Sets the file name.

SHEET_NAME = 'Entergy'	Sets the name of the Google Sheet.

creds_json_string = os.environ.get('GOOGLE_CREDS_JSON')
if not creds_json_string:
  print("Error: The GOOGLE_CREDS_JSON environment variable is not set.")
  sys.exit(1)
	Finds the google service account credentials in JSON format.

try:
  creds_dict = json.loads(creds_json_string)
  gc = gspread.service_account_from_dict(creds_dict)
	Accesses Google Sheets using gspread library.

except json.JSONDecodeError:
  print("Error: Could not decode the GOOGLE_CREDS_JSON. Ensure it's a valid, single-line JSON in your GitHub Secret.")
  sys.exit(1)
except Exception as e:
  print(f"An error occurred during gspread authorization: {e}")
  sys.exit(1)
 	
def current_entergy(location, area):
  url = f"https://entergy.datacapable.com/datacapable/v1/entergy/Entergy{location}/{area}"
  print(url)
  now = datetime.now()
  r = requests.get(url)
  data = r.json()
  entergy = pd.DataFrame(data)
  entergy["utility"] = "Entergy"
  entergy["time pulled"] = now
  # Optionally save to CSV (as your original code does), or skip this step
  # title = f"{now.year}-{now.month}-{now.day} {now.hour}.{now.minute}"
  # path = f"data/{location.lower()}/{area}/entergy/{title}.csv"
  # entergy.to_csv(path)
  return entergy
 	
data = current_entergy("Louisiana", "county")
data["state"] = "Louisiana"
 	
data["percent without power"] = (100 * data["customersAffected"] / data["customersServed"]).round(2)
	Creates a URL to access the Google Sheet.
	
	Gets the current timestamp.
	
	Creates a DataFrame.
	
	Creates two new columns I needed.

data["time pulled"] = pd.to_datetime(data["time pulled"]) data["time pulled"] = data["time pulled"].dt.strftime("%Y-%m-%d %H:%M")	This formats the time to only include the hour and minute, not seconds.

# County renaming logic (unchanged) 	
replace_dict = {
"E. BATON ROUGE": "EAST BATON ROUGE",
"W. BATON ROUGE": "WEST BATON ROUGE",
"E. CARROLL": "EAST CARROLL",
"W. CARROLL": "WEST CARROLL",
"E. FELICIANA": "EAST FELICIANA",
"W. FELICIANA": "WEST FELICIANA",
"LA SALLE": "LASALLE",
}
	The county names needed to be reformatted for Tableau to recognize them.

data = data[data['county'].str.lower() != 'other']	This deleted an empty row called “other”.

if 'county' in data.columns:
data['county'] = data['county'].replace(replace_dict)
else:
print("Warning: 'county' column not found in DataFrame.")
data['county'] = ""
	Checks for the column that the county names are in and prints an error message if it can’t be found.

sheet_data = [data.columns.tolist()] + data.astype(str).values.tolist()	This makes sure I’m only pulling the data each time, not pulling the header with every run.
data_rows = data.astype(str).values.tolist()	
sh = gc.open(SHEET_NAME) worksheet = sh.get_worksheet(0)	This opens the spreadsheet.

if worksheet.row_count == 0 or not worksheet.get_all_values(): worksheet.append_row(data.columns.tolist()) worksheet.append_rows(data.astype(str).values.tolist())	This checks to see if the spreadsheet is empty. This gets the column names from the DataFrame. This makes sure that the data is added to the end instead of erasing the existing data.