StockX Code Assessment

Chuyu Chen

January 25, 2022

Get any 7 consecutive days of weather data for your city (or any US city of your choice), as well as weather from Detroit, Michigan for the same period. The script should have the ability to run for a date. Have it load to S3/Google Cloud Storage (GCS) location. Share that location and the github code repo.

Requirements:

1. Script should work when we git clone.
2. Script pulls weather data and date/time from public rest-api's for Detroit and the additional city you chose. Load your results to S3/GCS.
3. S3/GCS is accessible to us, you decide how we get access.
4. Data in S3/GCS is formatted in a way that's easily loadable to a database table.
5. Commands to load data from S3/GCS to postgres or mysql or redshift table or Bigquery and DDL for table.
6. Ideally the goal is for us to easily query : select * from pubilc.weather_daily_table order by weather_date, location;
7. Optional not required bonus: Publish the data to Tableau public or similar location.

References:

- https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-csv
- https://googleapis.dev/python/storage/latest/index.html
- API: https://www.visualcrossing.com/weather-data

Pull data from public api

In [1]:
```
# import required modules
import requests, json
import pandas as pd
```

In [2]:
```
#Downloading weather data using Python as a CSV using the Visual Crossing Weather API
#See https://www.visualcrossing.com/resources/blog/how-to-load-historical-weather-data-using-python-without-scraping/ for more information.
import csv
import codecs
import urllib.request
import urllib.error
import sys
import datetime
from datetime import timedelta

# core of weather query URL
BaseURL = 'https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/'

# load ApiKey
ApiKey='VPCTK6KLC4VYY5WLKZRY6SK9T'

# UnitGroup sets the units of the output - us or metric
UnitGroup='us'

# Prompt user input location city and state for the weather data
# create a list including input cities and Detroit,MI
# Location = ['Chicago,IL', 'Detroit,MI','Indianapolis,IN','Columbus,OH','Milwaukee,WI']

Location = input("Enter desired city names, follow by semicolon (eg. Detroit,MI;): ")

l = [item.replace(" ",'') for item in Location.split(";")] + ['Detroit,MI']
if "" in l:
    l.remove("")
Location = l



# get 7 consecutive days of weather data in corresponding locations
StartDate = input("Enter the first date of 7 consecutive days (in YYYY-MM-DD): ")
EndDate=str(datetime.datetime.strptime(StartDate, "%Y-%m-%d") + timedelta(days=7)).split()[0]
print("The corresponding end date is: " + str(EndDate))

#JSON or CSV
#JSON format supports daily, hourly, current conditions, weather alerts and events in a single JSON package
#CSV format requires an 'include' parameter below to indicate which table section is required
ContentType="csv"

#include sections
#values include days,hours,current,alerts
Include="days"
```

```
Enter desired city names, follow by semicolon (eg. Detroit,MI;): Chicago,IL; Minneapolis,MN;
Enter the first date of 7 consecutive days (in YYYY-MM-DD): 2021-01-01
The corresponding end date is: 2021-01-08
```

In [3]:
```
data = []
print(Location)
for i in Location:
    print('')
    print(f' - Requesting weather for {i}: ')

    #basic query including location
    ApiQuery=BaseURL + i

    #append the start and end date if present
    if (len(StartDate)):
        ApiQuery+="/"+StartDate
        if (len(EndDate)):
            ApiQuery+="/"+EndDate

    #Url is completed. Now add query parameters (could be passed as GET or POST)
    ApiQuery+="?"

    #append each parameter as necessary
    if (len(UnitGroup)):
        ApiQuery+="&unitGroup="+UnitGroup

    if (len(ContentType)):
        ApiQuery+="&contentType="+ContentType

    if (len(Include)):
        ApiQuery+="&include="+Include

    ApiQuery+="&key="+ApiKey



    print(' - Running query URL: ', ApiQuery)
    print()

    try:
        CSVBytes = urllib.request.urlopen(ApiQuery)
    except urllib.error.HTTPError  as e:
        ErrorInfo= e.read().decode()
        print('Error code: ', e.code, ErrorInfo)
        sys.exit()
    except  urllib.error.URLError as e:
        ErrorInfo= e.read().decode()
        print('Error code: ', e.code,ErrorInfo)
        sys.exit()


    # Parse the results as CSV
    CSVText = csv.reader(codecs.iterdecode(CSVBytes, 'utf-8'))

    RowIndex = 0

    # The first row contain the headers and the additional rows each contain the weather metrics for a single day
    # To simply our code, we use the knowledge that column 0 contains the location and column 1 contains the date.  The data starts at column 4
    for Row in CSVText:
        if RowIndex == 0:
            FirstRow = Row
            columns = Row
        else:
            data.append(Row)
            #print('Weather in ', Row[0], ' on ', Row[1])

            ColIndex = 0
            for Col in Row:
                if ColIndex >= 4:
                    continue
                    #print('   ', FirstRow[ColIndex], ' = ', Row[ColIndex])
                ColIndex += 1
        RowIndex += 1

    # If there are no CSV rows then something fundamental went wrong
    if RowIndex == 0:
        print('Sorry, but it appears that there was an error connecting to the weather server.')
        print('Please check your network connection and try again..')

    # If there is only one CSV  row then we likely got an error from the server
    if RowIndex == 1:
        print('Sorry, but it appears that there was an error retrieving the weather data.')
        print('Error: ', FirstRow)

    #print()
```

```
['Chicago,IL', 'Minneapolis,MN', 'Detroit,MI']

 - Requesting weather for Chicago,IL:
 - Running query URL:  https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/Chicago,IL/2021-01-01/2021-01-08?&unitGroup=us&contentType
=csv&include=days&key=VPCTK6KLC4VYY5WLKZRY6SK9T


 - Requesting weather for Minneapolis,MN:
 - Running query URL:  https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/Minneapolis,MN/2021-01-01/2021-01-08?&unitGroup=us&content
Type=csv&include=days&key=VPCTK6KLC4VYY5WLKZRY6SK9T


 - Requesting weather for Detroit,MI:
 - Running query URL:  https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/Detroit,MI/2021-01-01/2021-01-08?&unitGroup=us&contentType
=csv&include=days&key=VPCTK6KLC4VYY5WLKZRY6SK9T
```

In [4]:
```
df = pd.DataFrame(data, columns = columns)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['name', 'datetime', 'tempmax', 'tempmin', 'temp', 'feelslikemax',
       'feelslikemin', 'feelslike', 'dew', 'humidity', 'precip', 'precipprob',
       'precipcover', 'preciptype', 'snow', 'snowdepth', 'windgust',
       'windspeed', 'winddir', 'sealevelpressure', 'cloudcover', 'visibility',
       'solarradiation', 'solarenergy', 'uvindex', 'severerisk', 'sunrise',
       'sunset', 'moonphase', 'conditions', 'description', 'icon', 'stations'],
      dtype='object')
```

In [6]:
```
df.shape
```

Out[6]:
```
(24, 33)
```

In [7]:
```
# generate a python dataframe with weather data

df = pd.DataFrame(data, columns = columns)
df[["City", "State", "Country"]] = df["name"].str.split(pat=",", expand=True)

df1 = df
cols = list(df.columns)
cols = cols[-3:] + cols[:-3]
df = df[cols]

# only include 13 columns for our purposes
df = df.iloc[:, : 11]
df['conditions'] = df1['conditions']
df['description'] = df1['description']
```

In [8]:
```
#df.to_csv("test.csv")
```

Load data to Google Cloud Storage

In [9]:
```
from gcloud import storage
from oauth2client.service_account import ServiceAccountCredentials
import os
```

In [10]:
```
credentials_dict = {
  "type": "service_account",
  "project_id": "nimble-net-337716",
  "private_key_id": "5ea5e115ddb0e068eb21996d2334ee942b9c7a68",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQC5VLAazQ9Szpnf\n/d1bag3Heg3rRt45039yW8n5UYOac7qcJ/jcTeLMPVcpZsQVc7u
  "client_email": "weather-data@nimble-net-337716.iam.gserviceaccount.com",
  "client_id": "112657965371709403967",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/weather-data%40nimble-net-337716.iam.gserviceaccount.com"
}



credentials = ServiceAccountCredentials.from_json_keyfile_dict(
    credentials_dict
)

client = storage.Client(credentials=credentials, project='nimble-net-337716')
bucket = client.get_bucket('weather-data-cc')
blob = bucket.blob(df)
```

In [11]:
```
bucket.blob('weather.csv').upload_from_string(df.to_csv(index=False), 'text/csv')
```

Load data from GCS to BigQuery

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   City          24 non-null     object
 1   State         24 non-null     object
 2   Country       24 non-null     object
 3   name          24 non-null     object
 4   datetime      24 non-null     object
 5   tempmax       24 non-null     object
 6   tempmin       24 non-null     object
 7   temp          24 non-null     object
 8   feelslikemax  24 non-null     object
 9   feelslikemin  24 non-null     object
 10  feelslike     24 non-null     object
 11  conditions    24 non-null     object
 12  description   24 non-null     object
dtypes: object(13)
memory usage: 2.6+ KB
```

In [13]:
```
import os

# get credential key file
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="nimble-net-337716-5ea5e115ddb0.json"
```

In [14]:
```
from google.cloud import bigquery

# Construct a BigQuery client object.
client = bigquery.Client()

# Set table_id to the ID of the table to create.
table_id = "nimble-net-337716.public.weather_daily_table"

job_config = bigquery.LoadJobConfig(
    schema=[
        bigquery.SchemaField("City", "STRING", mode="REQUIRED"),
        bigquery.SchemaField("State", "STRING", mode="REQUIRED"),
        bigquery.SchemaField("Country", "STRING"),
        bigquery.SchemaField("name", "STRING"),
        bigquery.SchemaField("datetime", "DATETIME", mode="REQUIRED"),
        bigquery.SchemaField("tempmax", "FLOAT"),
        bigquery.SchemaField("tempmin", "FLOAT"),
        bigquery.SchemaField("temp", "FLOAT"),
        bigquery.SchemaField("feelslikemax", "FLOAT"),
        bigquery.SchemaField("feelslikemin", "FLOAT"),
        bigquery.SchemaField("feelslike", "FLOAT"),
        bigquery.SchemaField("conditions", "STRING"),
        bigquery.SchemaField("description", "STRING")
    ],
    skip_leading_rows=1,
    # The source format defaults to CSV, so the line below is optional.
    source_format=bigquery.SourceFormat.CSV,
)
uri = "gs://weather-data-cc/test.csv"

load_job = client.load_table_from_uri(
    uri, table_id, job_config=job_config
)  # Make an API request.

load_job.result()  # Waits for the job to complete.

destination_table = client.get_table(table_id)  # Make an API request.
print("Loaded {} rows.".format(destination_table.num_rows))
```

```
Loaded 24 rows.
```

In [15]:
```
from google.cloud import bigquery

bqclient = bigquery.Client()

# Download query results.
query_string = """
SELECT *
FROM `nimble-net-337716.public.weather_daily_table`
ORDER BY datetime,City DESC
"""

dataframe = (
    bqclient.query(query_string)
    .result()
    .to_dataframe(
        create_bqstorage_client=True,
    )
)
print(dataframe.head())
```

```
      City State         Country                        name   datetime  \
0  Detroit    MI   United States  Detroit, MI, United States 2022-01-01
1  Chicago    IL   United States  Chicago, IL, United States 2022-01-01
2   Boston    MA   United States   Boston, MA, United States 2022-01-01
3  Detroit    MI   United States  Detroit, MI, United States 2022-01-02
4  Chicago    IL   United States  Chicago, IL, United States 2022-01-02

   tempmax  tempmin  temp  feelslikemax  feelslikemin  feelslike  \
0     42.6     27.5  36.5          41.9          15.6       29.2
1     40.9     29.6  34.8          33.4          16.7       24.3
2     51.2     44.2  46.9          51.2          40.9       45.7
3     26.9     17.6  24.3          16.4           8.9       13.3
4     29.1     15.9  23.5          20.7           6.8       13.0

                conditions                                      description
0  Snow, Partially cloudy        Partly cloudy throughout the day with snow.
1          Snow, Overcast         Cloudy skies throughout the day with snow.
2  Rain, Partially cloudy  Partly cloudy throughout the day with afternoo...
3  Snow, Partially cloudy  Partly cloudy throughout the day with early mo...
4  Snow, Partially cloudy  Partly cloudy throughout the day with early mo...
```

In [ ]:

In [ ]: