Python in the VE

VE Scripts – working with National Grid carbon intensity web-based data; making freeform profiles and APP files

What? We access the UK National Grid website and download a live dataset of carbon intensity data ; we process the data and import the data into the VE as freeform profiles. We also save the data into an APP file so we can chart it in VistaPro

Why? In this example we use the REQUESTS module, load the data into a Pandas dataframe , tidy the data then process the data into a format that we can import into the VE as a freeform profile . We also take the data, add meta-data and create an APP file

```
"""
=========================
UK NGESO CEF import to FFP
=========================

Module description
------------------
Imports UK National Grid electricity carbon intensity data from the NG live dataset to
VE free-from profiles and to an APP file (for charting in VistaPro).

Only whole year data should be imported; years 2018 onwards are available.
This script only produces non-leap year output so for leap year 2020 Feb 29 is omitted.
FFP data is converted to kgCO2/kWh to be consistent with VE units.

See https://data.nationalgrideso.com/carbon-intensity1/national-carbon-intensity-forecast#

"""

import io
import os
import appfile
import iesve
import requests
import pandas as pd

def download_ng_cef():
    """ Downloads data from NG ESO website and returns a pandas dataframe
        Imported units are gCO2/kWh
        Returned data includes forecast, actual & index columns
        Data may have gaps
        Failure to download quits the script

    Args:
        None

    Returns:
        df (pandas df) : CEF data
    """

    # Source url
    url = 'https://data.nationalgrideso.com/backend/dataset/f406810a-1a36-48d2-'\
        'b542-1dfb1348096e/resource/0e5fde43-2de7-4fb4-833d-c7bca3b658b0/download'\
        '/gb_carbon_intensity.csv'

    # Request a response object from NG ESO
    req = requests.get(url, verify=True)

    # Check response object is valid
    if req.status_code == 200:
        # Get response object content
        url_content = req.content
        # Create dataframe directly from content
        df = pd.read_csv(io.StringIO(url_content.decode('utf-8')))
        # Optionally save content to csv file
        #csv_file = open('downloaded.csv', 'wb')
        #csv_file.write(url_content)
        #csv_file.close()
        return df
    else:
        print('Error: Could not connect to the NG API (HTTP {})'.format(req))
        quit()

def tidy_cef(df):
    """ Sets datetime index, removes unwanted columns, converts column type to float,
        fills data gaps and removes Feb 29 2020 leap year rows

    Args:
        df (Pandas df) : NG CEF data

    Returns:
        df (pandas df) : CEF data tidied for actual data set and year
    """

    # Convert datetime column from imported string to datetime object and set as index
    # We will set dayfirst in datetime to reflect the UK data source format
    df['datetime'] = pd.to_datetime(df['datetime'], dayfirst=True)
    df = df.set_index(['datetime'])

    # Delete columns that are not required
    df = df.drop(['forecast', 'index'], axis=1)

    # Cast actual column to type float
    df['actual'] = df['actual'].astype(float)

    # Remove rows for leap year Feb 29th using a datetime index
    datetime_range = pd.date_range(start ='29/02/2020  00:00', end ='29/02/2020  23:30', freq ='30T')
    df.drop(datetime_range, inplace=True)

    # Convert imported units from gCO2/kWh to kgCO2/kWh as used in the VE
    df['actual'] = df['actual'].div(1000)

    # Rename actual column to what will be used in app file
    df.rename(columns={'actual': 'Grid_kgCO2/kWh'}, inplace=True)

    # Print off some stats
    print('Total number data points: ', len(df))
    print('Number of data points that are null: ', df['Grid_kgCO2/kWh'].isnull().sum(axis = 0))

    # Fill any data gaps using backward & forward filling to cover all situations
    df.fillna(method='bfill', inplace=True)
    df.fillna(method='ffill', inplace=True)

    return df

def create_ffp_data(df):
    """ Creates a new dataframe in FFP column format (month, day, hour, min, value)
        Works on non-leap year data

    Args:
        df (Pandas df) : NG cef tidied data for one year

    Returns:
        df (pandas df) : NG cef data in FFP format
    """

    # Determine step mins
    step = int(60/(round(len(df)/8760)))

    # Create new data
    count = 0
    data = []
    days_in_month = [31,28,31,30,31,30,31,31,30,31,30,31]
    for month in range (1, 13):
        days = days_in_month [month-1]
        for day in range (1, days+1):
            for hour in range (0,24):
                for min in range (0, 60, step):
                    value = df.iat[count,0]
                    row_data = [month, day, hour, min, value]
                    data.append (row_data)
                    count += 1
    # Create the extra final row at the end of the year
    row_data = [12, 31, 24, 0, df.iat[-1,0]]
    data.append (row_data)

    # Create dataframe
    df1=pd.DataFrame(data,columns=['month', 'day', 'hour', 'min', 'value'])

    return df1

def create_abs_ffp(df, year):
    """ Creates an absolute FFP (non-leap year) using output from create_ffp_data()
        FFP is named with a year suffix

    Args:
        df (Pandas df) : columns in FFP format
        year (str) : year
    """

    # Create name for FFP
    name = 'NGESO_Grid_emission_factors_' + year

    # Get the current VE project...
    project = iesve.VEProject.get_current_project()

    # Convert the dataframe to a list of lists whilst maintaining data type
    # integer for first 4 columns, float for last column
    free_profile_data = list(map(list, df.itertuples(index=False)))

    # Create ffd in project
    free_profile = project.create_profile(type='freeform', reference=name, modulating=False)
    free_profile.set_data(free_profile_data)
    free_profile.save_data()
    print('Free-form profile for ', year, ' added to project > ApPro')

def create_app(data):
    """ Creates an APP file for the CEF data for one or more years

    Args:
        data (dict) : year key : list of floats
    """

    # Process all defined years in the data dict to a df (one column per year)
    df = pd.DataFrame(data)

    # Create a default datetime series (non-leap year) with a 30 min frequency
    # Add to the dataframe and set as the index
    df['Timestamp'] = pd.date_range(start='1/1/2018', periods=17520, freq='30T')
    df = df.set_index(['Timestamp'])

    # Create a name for the APP file
    name = 'NGESO_Grid_Emission_Factors' + '.app'

    # Get the current VE project...
    project = iesve.VEProject.get_current_project()

    # get the df column names
    var_list = df.columns.values.tolist()

    # Create a dict for each column to assign the VistaPro category & unit
    # As units kgCO2/kWh is not in the IES units list use the Number unit
    variables = {}
    category = 'NGESO_Grid_Emission_Factors'
    for var in var_list:
        variables[var] = {
            'category': [category],'unit': 'Number', 'metadata': {}}

    # APP year is limited to 2009-2120 so use a default value (any will do)
    # NGESO data is a fixed 30 min timestep so 48 entries per 24 hours
    variables['Date/Time Stamp'] = {'year': 2018,'rpd': str(48)}

    # Create and write .app file
    file_path = os.path.join(project.path, "vista", name)
    app = appfile.APPFile.from_dataframe(df, variables, list(df.columns))

    try:
        with open(file_path, "w") as f:
            appfile.dump(app, f)
            print('APP file added to project > VistaPro')
    except PermissionError as e:
        print('Error', 'The current filename cannot be written to, as it is open. \
        If it is currently open in VistaPro, try closing VistaPro or choosing another\
        filename.')

# Main loop
if __name__ == '__main__':

    # Download NGESO data
    df = download_ng_cef()

    # Tidy the data
    df1 = tidy_cef(df)
    #df1.to_csv('NG_tidied_data.csv', encoding='utf-8', index=True)

    # Define a list of the years required
    years = ['2018', '2019', '2020', '2021']

    # Setup an empty dict for the APP file data (a list per year key)
    data = {}

    # Process the tidied data to free form profiles
    for year in years:
        # Take a df slice for the specified whole year
        df2 = df1.loc[year]
        # Process the data into the FFP format we need
        df3 = create_ffp_data(df2)
        # Create an absolute ffd in the current model
        create_abs_ffp(df3, year)

        # Finally add the data to the app file dict (for the next step)
        data[year] = df2['Grid_kgCO2/kWh'].values.tolist()

    # Create the APP file
    create_app(data)
```

The National Grid provides datasets that document Carbon Intensity, Demand & Wind availability; historic and live data is available … see https://data.nationalgrideso.com/search

We will use the carbon intensity dataset as per this URL; it is updated every 30 minutes

Load the modules we will need; *appfile* is a IESVE module provided in *VE Scripts >iesutils*

Function to download the raw data

Note the units of the source data and the data columns provided

The NG website supprts cURL, which stands for client URL. This is a tool that we can use to transfer data to and from a server. At the most fundamental, cURL lets you talk to a server by specifying the location (in the form of a URL) and the data you want to send

We assign a url string to a variable

We use the requests module GET method to request a response object from the source

We check that the response object is valid by checking it's status_code attribute

If the object is valid we GET the object's content. It is in the form a of a CSV file which we import to a Pandas datafame using the Pandas *read_csv* method. We have used the *decode* method to resolve any non UTF 8 character issues and StringIO to output this output into an object that *read_csv* can use. You may want to save the CSV & take a look at the raw data format

Print a message if the response object is not valid

Function to tidy the raw dataset

Convert the imported datetime string column to a datetime object type and set the index to this column

Remove unwanted columns

Cast data column to float type

Remove leap year extra day data rows

Convert to the units used in the VE

Rename the data column

There can be gaps in the data; fill the gaps as we do not want any NaN entries

Function to transform the tidied data in to columns that are needed for Freeform profile creation

Work out the frequency of the data

Create an empty list and define the number of days in each non-leap year month

Loop through the months, days, hours & minutes by time step … create a list for each ordinate and append to this to the main list to make a list of lists

The Freeform profiles format includes an additional terminating line; create this … we use df.iat[-1,0] to copy the last value in the dataframe

Create a new dataframe df1 using the list of lists

Function to create a Freeform profile in the current model

Create a unique string to name the profile

Get the current project & assign to a variable

We need to convert the dataframe columns into a list of lists whilst maintaining data type; however the Pandas *tolist()* method converts integers to floats so we will use the *map()* function with *df.itertuples* as the iterables input (*an object to iterate over namedtuples for each row in the DataFrame*) with *list* as the function to apply. We then apply list to make this a list of lists

We create the profile, set the data and save the profile

Function to create an APP file

Create a dataframe from the input DICT containing one or more years of data

Create a datetime series with a 30 min timestep, add it to the dataframe and set it as the index

Create a string for the APP file name

Get the current project & assign to a variable

Get a list of column names in the dataframe; these will become variable names in VistaPro

The APP format requires a DICT for each variable to define where it will appear in the VistaPro category tree, it's data type and any metadata (none in this case)

We set the year and timestep data and add it to the variables DICT

We create a file path to the project Vista folder then call the APPFile method to create the APP file

We write the APP file to the file path; we wrap this in a try statement to catch if the file already exists and is open e.g. VistaPro is open

This where the script starts executing

Call the function to download the data

Call the function to tidy the data Optionally save the CSV to see the tidied data format

Make a list of the years we want to process

Create an empty DICT; we will use this to hold the data for the APP file creation

Loop through the years list

Create a dataframe by slice for the year

Call the function to the data in the required format

Call the function to create the freeform profile

Add the data for the year to the APP file DICT as a list of floats

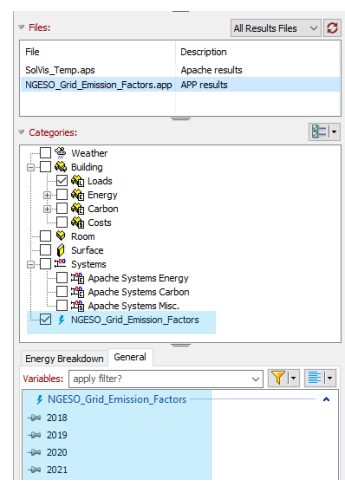
Call the function to create the APP file

Sample output:

VE Scripts output pane:

```
Total number data points:  76013
Number of data points that are null:  2080
Free-form profile for  2018  added to project > ApPro
Free-form profile for  2019  added to project > ApPro
Free-form profile for  2020  added to project > ApPro
Free-form profile for  2021  added to project > ApPro
APP file added to project > VistaPro
  >>> Runtime: 9.93 seconds
```

VistaPro picker panes:



ApPro > Freeform profiles view: