Developer Documentation

Data and Introduction

This project collects monthly oceanic surface temperature and surface salinity data. This data typically comes in as a NetCDF file type, and has to be converted to a TIFF file format. This data is collected from Copernicus at the following source:
https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-oras5?tab=overview

Additionally, there are two program files that we use to accomplish these tasks. The primary program file that everything runs from is called get_tiff.py. The primary program file is also responsible for making the API calls. The secondary program file is called convert_folder.py, and its purpose is to convert the NetCDF files to TIFF files.

API

The source provides an API that we use to pull data. Python has a dedicated library that we use to interact with the API. A call to this API is shown in Figure 1.

```
c.retrieve(
    'reanalysis-oras5',
    {
        'format': 'zip',
        'year': year,
        'month': month,
        'variable': [
            'sea_surface_salinity', 'sea_surface_temperature',
        ],
        'vertical_resolution': 'single_level',
        'product_type': [
            'consolidated', 'operational',
        ],
    },
    temp_data_zip_filepath)
```

Figure 1: Example API Request

Here ‘reanalysis-oras5’ is the dataset from Copernicus that we want to grab data from. We then specify the format that we want our data to come in as, in this case we want it in a Zip file format. Then we can specify the month and year that we want our data for, the data variables that we want, and a few other parameters. Finally,

temp_data_zip_filepath specifies the name of the Zip file that we want to send the response of this API call to.

In our project we take user input as a single month, or a range of months to collect data from. We then iterate through those months and pull data using the above API call. We then extract each response from the API call and store that data in a folder labeled “raw_data”, which we create in the same directory that the primary program file is in.

Conversion Technique

Once all of the raw NetCDF files are downloaded we use the secondary program file (convert_folder.py) to convert all of the files in the newly created “raw_data” folder into TIFF files. We iterate through each file in the “raw_data” folder and convert each file one at a time. The final TIFF file is then saved in a folder called “tiff_data”, which is also created in the same directory that the primary program file is located in. The final TIFF files take on the naming format: MM_YYYY_sosaline.tiff for salinity data and MM_YYYY_sosstsst.tiff for surface temperature data. The final data is also organized by surface temp./salinity and year within the “tiff_data” folder.

Now, we will go over the conversion technique. The conversion requires several intermediate steps before we can produce a final TIFF file. Firstly, we must convert the NetCDF files into CSV files. This has to be done because of the nature of the latitude and longitude data with this particular dataset. They are stored in 2D arrays that are not properly recognized by GDAL. GDAL is a powerful conversion library for working with geospatial information. Thus, we use a library called netcdf2csv to convert the NetCDF files to CSV. This process is shown in Figure 2.

```
convert_file(file_path, os.getcwd(), os.getcwd(), clean_choice=0)   #convert .nc to csv file
```

Figure 2: Conversion Process

After this, we open the CSV files up in a Pandas dataframe and change some of the column names to make the next step in the conversion more easy. We also replace all NaN values with -9999.0. This is an accepted convention within the community, and not replacing the NaN values can produce bad TIFF files in the next step of the conversion. We then save this as a new CSV file. This process is shown in Figure 3.

```
df = pd.read_csv(uncleaned_csv_filepath)    #open csv file
df = df[['nav_lat', 'nav_lon', data_type]]
df.columns = ['y', 'x', 'z']    #change variable names
df = df[['x', 'y', 'z']]
df = df.replace(np.NaN, -9999.0)    #replace nan values with -9999
df.to_csv(cleaned_csv_filename, index=False)
```

Figure 3: Data Cleaning

The next step in the conversion process is going from CSV to TIFF. First, we create another intermediate file, which is a VRT file from the cleaned CSV. The VRT file acts as a format driver for GDAL, which is what we will ultimately use to convert to TIFF. This process is shown in Figure 4.

```
with open(vrt_fn, 'w') as fn_vrt:   #build vrt file
    fn_vrt.write('<OGRVRTDataSource>\n')
    fn_vrt.write('\t<OGRVRTLayer name="%s">\n' % lyr_name)
    fn_vrt.write('\t\t<SrcDataSource>%s</SrcDataSource>\n' % cleaned_csv_filename)
    fn_vrt.write('\t\t<GeometryType>wkbPoint</GeometryType>\n')
    fn_vrt.write('\t\t<GeometryField encoding="PointFromColumns" x="x" y="y" z="z"/>\n')
    fn_vrt.write('\t</OGRVRTLayer>\n')
    fn_vrt.write('</OGRVRTDataSource>\n')
```

Figure 4: VRT Step

Finally, we can convert the VRT file to the final TIFF format using GDAL. We also clean up the working directory by removing all of the intermediate files that we created during the conversion process that are no longer necessary. This process is shown in Figure 5.

```
r = gdal.Rasterize(out_tif, vrt_fn, outputSRS = "EPSG:4326", xRes = res, yRes = -res, attribute = "z", noData = -9999.0) #rasterize vrt
r = None

g = gdal.Grid(out_tif, vrt_fn, outputSRS = "EPSG:4326", algorithm = 'nearest:radius=0.0:nodata=-9999.0') #use gdal to convert from vrt
g = None

os.remove(uncleaned_csv_filepath)   #remove uncessary files created during conversion process
os.remove(os.getcwd() + "/" + cleaned_csv_filename)
os.remove(os.getcwd() + "/" + vrt_fn)
```

Figure 5: Using GDAL

Conclusion

This document encompasses the high level details about most of the processes in the get_tiff.py and convert_folder.py program files. These files are also well commented, so you may also consult the code itself for additional information.