Tidy-up Japan Typhoon Data

Tidy-up Japan Typhoon Data

Tidy up Japan Typhoon data from 1951 to 2023.

import modules

```
import os
from typing import List

import polars as pl
pl.Config.set_tbl_rows(7)   # limit num of lines for table preview
print('polars version', pl.__version__)
```

```
polars version 0.20.9
```

create df schema as a dict {"col_name", pl.DataType}

```
df_schema = {
    'h_a_indicator'               : pl.String,
    'h_b_int_num_id'              : pl.String,
    'h_c_num_data'                : pl.String,
    'h_d_tropical_cyclone_num_id' : pl.String,
    'h_e_int_num_id'              : pl.String,
    'h_f_flag_last_data_line'     : pl.String,
    'h_g_diff_hour'               : pl.String,
    'h_h_storm_name'              : pl.String,
```

```
    'h_i_date_last_rev'           : pl.String,
    'd_a_date_time'               : pl.String,
    'd_b_indicator'               : pl.String,
    'd_c_grade'                   : pl.String,
    'd_d_latitude'                : pl.String,
    'd_e_longitude'               : pl.String,
    'd_f_central_pressure_dPa'    : pl.String,
    'd_g_max_wind_speed_kt'       : pl.String,
    'd_h_dir_longest_r_50kt_wind' : pl.String,
    'd_i_longest_r_50kt_wind_nm'  : pl.String,
    'd_j_shortest_r_50kt_wind_nm' : pl.String,
    'd_k_dir_longest_r_30kt_wind' : pl.String,
    'd_l_longest_r_30kt_wind_nm'  : pl.String,
    'd_m_shortest_r_30kt_wind_nm' : pl.String,
    'd_p_landfall_or_passage'     : pl.String,
}

# print('type of df_schema:', type(df_schema))
print(df_schema)
```

```
{'h_a_indicator': String, 'h_b_int_num_id': String, 'h_c_num_data': String, 'h_d_tropical_cy
```

create an empty dataframe

```
list_col_names = list(df_schema.keys())

dict_empty_data = {
    'h_a_indicator'               : [],
    'h_b_int_num_id'              : [],
    'h_c_num_data'                : [],
    'h_d_tropical_cyclone_num_id' : [],
    'h_e_int_num_id'              : [],
    'h_f_flag_last_data_line'     : [],
    'h_g_diff_hour'               : [],
    'h_h_storm_name'              : [],
    'h_i_date_last_rev'           : [],
    'd_a_date_time'               : [],
    'd_b_indicator'               : [],
    'd_c_grade'                   : [],
```

```
    'd_d_latitude'                : [],
    'd_e_longitude'               : [],
    'd_f_central_pressure_dPa'    : [],
    'd_g_max_wind_speed_kt'       : [],
    'd_h_dir_longest_r_50kt_wind' : [],
    'd_i_longest_r_50kt_wind_nm'  : [],
    'd_j_shortest_r_50kt_wind_nm' : [],
    'd_k_dir_longest_r_30kt_wind' : [],
    'd_l_longest_r_30kt_wind_nm'  : [],
    'd_m_shortest_r_30kt_wind_nm' : [],
    'd_p_landfall_or_passage'     : [],
}

df= pl.DataFrame(dict_empty_data, df_schema)
df
```

shape: (0, 23)

str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str

read the data file, extract header lines and data lines

```
%%time

# row data file path
# data_file_path = './data/RSMC_Tokyo_Typhoon_1951-2023.txt'
# data_file_path = './data/RSMC_Tokyo_Typhoon_2013-2023.txt'
# data_file_path = './data/RSMC_Tokyo_Typhoon_2019-2023.txt'
# data_file_path = './data/RSMC_Tokyo_Typhoon_2021-2023.txt'
data_file_path = './data/RSMC_Tokyo_Typhoon_2023.txt'

# extracted header and data file paths
e_header_f_path = './data/header.txt'
e_data_f_path = './data/data.txt'

# remove header and data files, if they exists
if os.path.exists(e_header_f_path):
```

```
    os.remove(e_header_f_path)

if os.path.exists(e_data_f_path):
    os.remove(e_data_f_path)

# open header and data file to write
f_header = open(e_header_f_path, 'w')
f_data = open(e_data_f_path, 'w')

# extract header lines and data lines into different files
with open(data_file_path) as f:
    # read all the lines
    lines = f.readlines()

    # read each line
    for line in lines:
        # check if a line is a header/data by the length of its 1st split
        if len(line.split()[0]) == 5:
            # add header to f_header
            f_header.write(line)

        elif len(line.split()[0]) == 8:
            # add data line to f_data
            f_data.write(line)

f_header.close()
f_data.close()
```

```
FileNotFoundError: [Errno 2] No such file or directory: './data/RSMC_Tokyo_Typhoon_2023.txt'
```

func to vstack each data record to df

```
def vstack_df(
    _col_names: List[str],
    _record_items: List[str],
    _df: pl.DataFrame,
) -> pl.DataFrame:

    # create a dict from list of col names and a list of data items
```

```
    dict_record = dict(
        zip(
            _col_names,
            _record_items,
        )
    )

    df_to_stack = pl.DataFrame(dict_record)

    _df = _df.vstack(df_to_stack)

    return _df
```

read header and data files to create dataframe

```
with open(e_header_f_path, 'r') as f_h:
    h_lines = f_h.readlines()

    header_cnt = 0

    f_data_line_num_start = 0
    f_data_line_num_end = -1

    for h_line in h_lines:
        # create an empty list to store header and data for each record
        headedr_items = []

        # get info from header col by col
        h_a = h_line[0:5]
        h_b = h_line[6:10]
        h_c = h_line[12:15]
        h_d = h_line[16:20]
        h_e = h_line[21:25]
        h_f = h_line[26]
        h_g = h_line[28]
        h_h = h_line[30:50].strip()
        h_i = h_line[64:72]

        # calc obsolute start and end line num in f_data to read
```

```
        # ...new start_line_num = previous end_line_num +1
        f_data_line_num_start = f_data_line_num_end + 1
        # ...new end_line_num = new start_line_num + num_data_lines of the current data chunk
        f_data_line_num_end = f_data_line_num_start + int(h_c) - 1

        headedr_items.extend([
            h_a,
            h_b,
            h_c,
            h_d,
            h_e,
            h_f,
            h_g,
            h_h,
            h_i
        ])

        print('header:', header_cnt, '\t', headedr_items)

        # read data file by the start and end line numbers
        # f_data = open('./data/data.txt', 'r')
        f_data = './data/data.txt'

        with open(f_data, 'r') as f_data:
            for idx, d_line in enumerate(f_data):
                record_items = []

                if f_data_line_num_start <= idx <= f_data_line_num_end:
                    data_items = []

                    d_a = d_line[0:8].strip()
                    d_b = d_line[9:12].strip()
                    d_c = d_line[13:14].strip()
                    d_d = d_line[15:18].strip()
                    d_e = d_line[19:23].strip()
                    d_f = d_line[24:28].strip()
                    d_g = d_line[33:36].strip()
                    d_h = d_line[41].strip()
                    d_i = d_line[42:46].strip()
                    d_j = d_line[47:51].strip()
                    d_k = d_line[52].strip()
                    d_l = d_line[53:57].strip()
```

```
                    d_m = d_line[58:62].strip()
                    d_p = d_line[71].strip()

                    data_items.extend([
                        d_a,
                        d_b,
                        d_c,
                        d_d,
                        d_e,
                        d_f,
                        d_g,
                        d_h,
                        d_i,
                        d_j,
                        d_k,
                        d_l,
                        d_m,
                        d_p
                    ])

                    # join the lists of header and data items
                    record_items = headedr_items + data_items

                    # stack the df created from the current record to df
                    # option 1:
                    # dict_record = dict(zip(list_col_names, record_items))
                    # df_to_stack = pl.DataFrame(dict_record)
                    # df = df.vstack(df_to_stack)
                    # option 2:
                    df = vstack_df(list_col_names, record_items, df)

                    # # cherry-print a data line for verification
                    # if idx == 337:
                    #     print(record_items)

        header_cnt += 1
df
```

shape: (0, 23)

h_a	h_b	h_c	h_d	h_e	h_f	h_g	d_a	d_b	d_c	d_d	d_e	d_f	d_g	d_h	d_i	d_j	d_k	d_l	d_m	d_n	d_o	d_p
str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str

replace empty string with 'null'

```
df = df.select(
    # pl.when(pl.col(pl.Utf8).str.lengths()==0) # lengths() deprecated
    #   .then(None)
    #   .otherwise(pl.col(pl.Utf8))             # pl.Utf8 replaced by pl.String
    #   .keep_name()                            # .keep_name() deprecated

    pl.when(pl.col(pl.Utf8).str.len_bytes()==0)
      .then(None)
      .otherwise(pl.col(pl.String))
      .name.keep()
)
df
```

shape: (0, 23)

h_a	h_b	h_c	h_d	h_e	h_f	h_g	d_a	d_b	d_c	d_d	d_e	d_f	d_g	d_h	d_i	d_j	d_k	d_l	d_m	d_n	d_o	d_p
str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str	str

cast data type for columns

```
df = df.with_columns(
    pl.col('h_a_indicator').cast(pl.Int32),
    pl.col('h_b_int_num_id').cast(pl.Int16),
    pl.col('h_c_num_data').cast(pl.Int16),
    pl.col('h_d_tropical_cyclone_num_id').cast(pl.Int8),
    pl.col('h_e_int_num_id').cast(pl.Int16),
    pl.col('h_f_flag_last_data_line').cast(pl.Int8),
    pl.col('h_g_diff_hour').cast(pl.Int8),
    pl.col('d_b_indicator').cast(pl.Int8),
    pl.col('d_c_grade').cast(pl.Int8),
```

```
    pl.col('d_d_latitude').cast(pl.Float64),
    pl.col('d_e_longitude').cast(pl.Float64),
    pl.col('d_f_central_pressure_dPa').cast(pl.Int16),
    pl.col('d_g_max_wind_speed_kt').cast(pl.Int16),
    pl.col('d_h_dir_longest_r_50kt_wind').cast(pl.Int16),
    pl.col('d_i_longest_r_50kt_wind_nm').cast(pl.Int16),
    pl.col('d_j_shortest_r_50kt_wind_nm').cast(pl.Int16),
    pl.col('d_k_dir_longest_r_30kt_wind').cast(pl.Int16),
    pl.col('d_l_longest_r_30kt_wind_nm').cast(pl.Int16),
    pl.col('d_m_shortest_r_30kt_wind_nm').cast(pl.Int16),
)
df
```

shape: (0, 23)

h_…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
i32	i16	i16	i8	i16	i8	i8	str	str	str	i8	i8	f64	f64	i16	i16	i16	i16	i16	i16	i16	i16	str

add 4 `0` @ end of each item in `d_a_date_time` column

```
df =  df.with_columns(
    pl.col('d_a_date_time').str.pad_end(12, "0")
)
df
```

shape: (0, 23)

h_…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
i32	i16	i16	i8	i16	i8	i8	str	str	str	i8	i8	f64	f64	i16	i16	i16	i16	i16	i16	i16	i16	str

add datetime column

```
df = df.with_columns(
    # pl.col('h_i_date_last_rev')
    #   .str.strptime(pl.Date, format="%Y%m%d")
```

```
    #   .alias('date_last_rev'),
    pl.col('d_a_date_time')
      .str.strptime(pl.Datetime, format="%y%m%d%H%M%S")
      .alias('date_time'),
)
df
```

shape: (0, 24)

h_a	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
i32	i16	i16	i8	i16	i8	i8	str	str	str	i8	i8	f64	f64	i16	i16	i16	i16	i16	i16	i16	i16	str	datetime[s]

adjust lat and lon value

```
df = df.with_columns(
    pl.col("d_d_latitude").mul(0.1),
    pl.col("d_e_longitude").mul(0.1)
)
df
```

shape: (0, 24)

h_a	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
i32	i16	i16	i8	i16	i8	i8	str	str	str	i8	i8	f64	f64	i16	i16	i16	i16	i16	i16	i16	i16	str	datetime[s]

check result

```
print('final: ')
df
```

```
final:
```

shape: (0, 24)

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
i32	i16	i16	i8	i16	i8	i8	str	str	str	i8	i8	f64	f64	i16	i16	i16	i16	i16	i16	i16	str	datetime[s]

```
# df.describe()
```

write df as parquet

```
df_parquet_f_path = "./data/df.parquet"

# remove parquet file, if it exists
if os.path.exists(df_parquet_f_path):
    os.remove(df_parquet_f_path)

df.write_parquet(df_parquet_f_path)
print('df written as parquet file')
```

```
df written as parquet file
```

create map

```
import plotly.express as px

df = pl.read_parquet(df_parquet_f_path)

fig = px.line_mapbox(
    df,
    # lat=df["d_d_latitude"],
    # lon=df["d_e_longitude"],
    # color=df["h_b_int_num_id"],
    lat="d_d_latitude",
    lon="d_e_longitude",
    color="h_b_int_num_id",
    zoom=3,
    height=1000,
    # animation_frame="h_b_int_num_id"
)
```

```
fig.update_layout(
    # mapbox_style="open-street-map",
    mapbox_style="carto-darkmatter",
    # mapbox_zoom=4,
    mapbox_center_lat=36,
    margin={"r":0,"t":0,"l":0,"b":0},
)

fig.show()
```

```
    <script type="text/javascript">
    window.PlotlyConfig = {MathJaxConfig: 'local'};
    if (window.MathJax && window.MathJax.Hub && window.MathJax.Hub.Config) {window.MathJax.Hu
    if (typeof require !== 'undefined') {
    require.undef("plotly");
    requirejs.config({
        paths: {
            'plotly': ['https://cdn.plot.ly/plotly-2.29.1.min']
        }
    });
    require(['plotly'], function(Plotly) {
        window._Plotly = Plotly;
    });
    }
    </script>
```

```
<div id="b8ea8a48-516b-41eb-ac0a-20f665b889ef" class="plotly-graph-di
```

var gd = document.getElementById(‘b8ea8a48-516b-41eb-ac0a-20f665b889ef’); var x = new MutationObserver(function (mutations, observer) {{ var display = window.getComputedStyle(gd).display; if (!display || display === ‘none’) {{ console.log([gd, ‘removed!’]); Plotly.purge(gd); observer.disconnect(); }} }});

// Listen for the removal of the full notebook cells var notebookContainer = gd.closest(‘#notebook-container’); if (notebookContainer) {{ x.observe(notebookContainer, {childList: true}); }}

// Listen for the clearing of the current output cell var outputEl = gd.closest(‘.output’); if (outputEl) {{ x.observe(outputEl, {childList: true}); }}

```
})                };                });            </script>        </div
```