

# Data Preprocessor

April 2, 2024

#

Data Preprocessor

## 0.1 Required Imports

```
[6]: import pandas as pd
import os
import glob
import warnings
```

## 0.2 Class Code

This is the first quarter of the entire project. It is meant to preprocess the data that will be used for everything else. While it didn't necessarily need to be well documented and modular, it is for the sake of readability and future iterations.

```
[7]: class Pypil_Dataframe_Processor:
    """
    Purpose:
    This class represents an automated approach to trimming and storing all of
    ↪the
    .csv files as newly named ones in a single folder that were given to me for
    ↪the task

    It will strip the columns to only "wold_index" and "diameter" which are
    ↪presumed
    to be the timestamp and the diameter of the pupil features. If these don't
    ↪exist
    it will then drop those dataframes so we can view them later.

    ...

    Attributes:
    directory : str
        This is meant to be our working directory (default: C.W.D.)

    ...
```

```

Methods:
list_directories(directory)
    This is meant to make a list of all possible directories from this_
↳location.

process_csv_files(columns_to_keep, file_suffix, output_folder)
    This will take all .csv files from the list given to it and read them,
    remove all of their NaN/Nulls, trim it down to only the listed allowed
    columns, then save it as a new dataframe wherever the class was ran.
    E.g. If you change it's directory before running, it will save all of
    the .csv files where the .py/.ipynb file is and not where the
    directory is.

"""
def __init__(self):
    try:
        import glob
        import os
        import warnings
        import pandas as pd
    except ImportError:
        raise ImportError("You are missing a required library from this_
↳group: glob, os, pandas.")

    # This will immediately run list_directories()
    self.directory = self.list_directories(directory=os.getcwd())

    # Initialize list to store skipped files if we want to view them
    self.skipped_files = []

def list_directories(self, directory=os.getcwd(), max_dir=50):
    """
    Purpose:
    This will take in a directory, then it will use os.walk to find all
    possible paths that can lead from here via folders and stores them in
    a list.

    ...

    Parameters:
    directory: str
        This is the directory in which you want it to check from. It is
        defaulted to checking the current working directory and is best
        suited to working from there.
        default: os.getcwd()
    max_dir : int

```

```

        This is the number of directories you can go into before the code_
    ↪will
        break you out with an error.
        default: 50
    ...

    Output:
    This will output all potential folder locations in the form of strings
    within a single list. NOTE: This can get problematically large if you
    do this in a high enough folder location.
    """

    # We have to initialize the list first to store the directories
    all_dir = []

    # We have to set this to 0 before we start, if this reaches 50 we break
    num_dir_trav = 0

    # os.walk will take us through every folder system in the directory
    for root, dirs, files in os.walk(directory):
        num_dir_trav += 1

        # As noted before, if we traverse too many directories, we'll break_
    ↪it
        if num_dir_trav > max_dir:
            warnings.warn(f"Exceeded maximum number of directories_
    ↪traversed ({max_dir}). "
                        f"Consider setting a lower limit.", UserWarning)
            break

        # This is where we'll append everything
        all_dir.append(root)

        # This is where we add the roots to everything
        for dir_name in dirs:
            all_dir.append(os.path.join(root, dir_name))
    return all_dir

    def process_csv_files(self, columns_to_keep=["world_timestamp",_
    ↪"world_index", "diameter"],
                        file_suffix='_trimmed', output_folder='trimmed_files',
                        additional_preprocessing=False):
        """
        Purpose:
        This method is meant to do very light pre-processing to the dataframes
        that are fed into it. It will take in only a list of directories and

```

after that it will read through each one for the .csv files. Then it\_

↳will remove NaNs/Null values, trim it to the parameters of columns\_to\_keep and then save it to the current working directory with an altered name. ...

Parameters:

columns\_to\_keep : list

This will be the list of columns within the dataframe that it will save for the new version. As this is built for a certain project in mind, it already has its default set to the columns that are\_

↳wanted,

but this can easily be altered.

default: ["world\_index", "diameter"]

file\_suffix : str

The suffix to be added to the new file name. Added as a potential option for you to change if the names get a little too long.

default: '\_trimmed'

output\_folder : str

The name of the folder where the new CSV files will be stored.\_

↳Another

option for you to change if you wanted to store items differently.

default: 'trimmed\_files'

...

Output:

There is no return statement for this method as it will create a large number of new .csv files for the user.

```

"""
# This is on the off chance you input a single dataframe into here
if isinstance(self.directory, list):
    directories = self.directory
else:
    directories = [self.directory]

# Sets our C.W.D. so we have it for later
save_dir = os.getcwd()

# This is where we initialize the list of all skipped files
# It's also a set because I'm struggling to make it not repeat
self.skipped_files = set()

# Create output folder if it doesn't exist
output_folder_path = os.path.join(save_dir, output_folder)
os.makedirs(output_folder_path, exist_ok=True)

```

```

# I keep getting multiple of the same errors so I will make it a set
dec_err_file = set()

# Now we'll take all of those directories and iterate through them here
for directory in directories:
    # Get a list of all CSV files in the directory we are checking
    csv_files = glob.glob(os.path.join(directory, "*.csv"))

    # Now we'll do all of the preprocessing steps here
    for csv_file in csv_files:
        # This will try the utf encoding of the document, if it cannot
        do it then
            # it will skip it
            try:
                df = pd.read_csv(csv_file)

                # This is to prevent us from going through dataframes that
                don't
                # have the columns we want, if they don't have it we store
                it elsewhere
                if set(columns_to_keep).issubset(df.columns):
                    # Drops NaN's and trims the columns in the DF
                    df.dropna(inplace=True)
                    df = df[columns_to_keep]

                    # Additional preprocessing step: groupby "world_index"
                    and average "diameter"
                    if additional_preprocessing:
                        df = df.groupby('world_index')['diameter'].mean().
                        reset_index()

                        # Extract the prefix from the directory name for ease
                        of reading
                        fold_name = os.path.basename(directory)
                        prefix = fold_name.split("_")[0].split("-")[0]

                        # Save trimmed DataFrame in the output folder we made
                        earlier
                        base = os.path.basename(csv_file)

                        # Makes the new name and sets it as specified earlier
                        new_file_name = f"{prefix}_{os.path.
                        splittext(base)[0]}{file_suffix}.csv"

                        # Sets the path to the folder to save it to

```

```

        new_file_path = os.path.join(output_folder_path,
↪new_file_name)

        # Saves each one to that new folder
        df.to_csv(new_file_path, index=False)
#         print(f"Processed DataFrame saved as {new_file_path}")
    else:
        # Add skipped file to the list for us to view later
        self.skipped_files.add(os.path.basename(csv_file))
        file_name = os.path.basename(csv_file)
#         print(f"\nSkipped DataFrame due to missing columns:
↪{file_name}")
    except UnicodeDecodeError:
        if csv_file not in dec_err_file:
            dec_err_file.add(csv_file)
            file_name = os.path.basename(csv_file)
            print(f"\nError reading file: {file_name}. \nPlease try
↪a different encoding.")

```

```
[14]: # os.chdir("whatever your directory is")
```

```
[12]: processor = Pypil_Dataframe_Processor()

# This is used to make the files without any additional preprocessing
processor.process_csv_files()
```

Error reading file: recording01.csv.  
Please try a different encoding.

Error reading file: recording02.csv.  
Please try a different encoding.

Error reading file: recording03.csv.  
Please try a different encoding.

Error reading file: recording04.csv.  
Please try a different encoding.

```
[13]: # This is used to groupby world_index for each file so that there
# is only one world_index per number
processor.process_csv_files(output_folder="grouped_trimmed_folder",
                             additional_preprocessing=True)
```

Error reading file: recording01.csv.  
Please try a different encoding.

Error reading file: recording02.csv.  
Please try a different encoding.

Error reading file: recording03.csv.  
Please try a different encoding.

Error reading file: recording04.csv.  
Please try a different encoding.