This course has already ended.

Round 2, Exercises

Points 200 / 200 | My submissions 1 / 10 | Deadline Monday, 14 March 2022, 18:00 | To be submitted alone | Show assignment template

This course has been archived (Monday, 17 April 2023, 12:00).

Week 2, exercise 1: Cleaning and unifying data – COVID statistics

Cleaning and unifying data – COVID statistics:

The exercise is about the following topics:

- Using the Pandas library: reading datasets into DataFrames; manipulating DataFrames
- Using the Numpy library: handling not-a-number values
- Reformatting numbers and dates

These topics are covered in chapters 1 and 2 of the course material.

Background

Often, when we read a dataset from a file, the data is not in a format that we can immediately use. This is even more likely when we attempt to use multiple files from different sources. Before we can unify the data from multiple sources into a single, usable whole, we first have to clean the data and reformat it into a single, matching format.

Initial preparations

Create a new project in Eclipse or PyCharm named "assignment2_1" and create a new module in this project named for example "covid_statistics". This Python file should have a function called "main", through which all other functions (if any) are called.

Description of the exercise

Three different TSV files ("tab-separated values"; i.e. dataset files) contain time series COVID-19 case data for three different countries. Each file corresponds to a single country and has rows corresponding to days 1 to 31 of January 2021. These rows contain both the date and the amount of new COVID-19 cases observed on that date.

However, there are differences between the files, which make them difficult to unify. Your task is to read the files, clean and reformat their data, and unify them into a single Pandas DataFrame. The *date* values should be converted into DateTime objects. The *cases* numbers should be converted into numeric type and split amongst three columns; the names of these columns should be the word "cases" followed by an underscore and the filename (without the file extension). So, for file `usa.tsv`, the column name should be `cases_usa`. This unified dataframe is then used to print some basic information about the datasets, using DataFrame.describe() and DataFrame.head().

Three example files are given for local testing (you can preview them using a plain text editor like Notepad): canada.tsv, south_korea.tsv and brazil.tsv. The file canada.tsv has full data (no invalid *cases* values or missing rows), while south_korea.tsv has invalid values and brazil.tsv has missing rows. Invalid values in these files don't necessarily say "null" – they can be any non-numeric value (e.g. "nothing here" or "???"). Note that, once the exercise has been submitted, the A+ autograder will test your code against other files, as well.

Progression of the program

First, the program asks the user for the names of the three TSV files. If a file cannot be read, the program informs of the error and stops running. If a file has already been submitted, the program informs of this and asks the user again. The program reads the files into Pandas DataFrames, converts the dates and numbers into the same formats, and merges the DataFrames. Finally, the program prints the unified DataFrame's summary statistics using DataFrame.describe() and the first five rows using DataFrame.head().

Structure of the files

Each file has two columns: the *date* column indicates the date of the row and the *cases* column indicates the amount of new COVID-19 cases observed on that day. Each file uses only one date format and one number format (so there won't be a file with both `YYYY-MM-DD` dates and `M/D/YYYY` dates, for instance). The date and number formats do not necessarily correspond with those generally used by the country in question. To make the exercise simpler, you can assume that the date formats always use a separator (dot, hyphen, slash) and they do not use more than one kind of separator.

The following is a list of possible date formats used in the files, using `April 30th, 2021` for the examples:

Format description	strptime Format Code	Example
Year-Month-Day with leading zeros (ISO 8601)	'%Y-%m-%d'	'2021-04-30'
Day.Month.Year with leading zeros	'%d.%m.%Y'	'30.04.2021'
Day.Month.Year without leading zeros and short year format (only last two digits)	'%d.%m.%y'	'30.4.21'
Day-Month-Year (with leading zeros)	'%d-%m-%Y'	'30-04-2021'
Day/Month/Year (with leading zeros)	'%d/%m/%Y'	'30/04/2021'
Month/Day/Year (with leading zeros)	'%m/%d/%Y'	'04/30/2021'

The following is a list of possible number (integer) formats used in the files, using `one million` for the examples:

Format description	Example
No thousands separator	'1000000'
Thousands separator: comma	'1,000,000'
Thousands separator: space	'1 000 000'
Thousands separator: dot / period	'1.000.000'

Error handling

Errors in reading files: If an error occurs while reading the file (the file doesn't exist or the program doesn't have the correct file permissions) the program should print a message saying that an error has occurred while reading the file (see the example execution for the specific phrasing), and close itself.

Repeated filenames: If a user attempts to submit a filename that has already been submitted, the program informs of this (see the example executions for the specific phrasing) and asks for a different file.

Not-a-number (invalid) case counts: If a row in the file contains a *cases* value that is not a number (for example, if the case value reads "not available"), the program should replace the value with a not-a-number value (**numpy.NaN**). The program should continue, but it should not treat these not-a-number values as zeroes. The TSV files used in testing will not include decimal numbers (only integers), so decimals do not need to be accounted for. However, note that even if you convert each of a column's values into integers, if that column also includes a not-a-number value, pandas will treat all values in the column as floats. Similarly, the `pd.to_numeric()` function will convert any columns with NaNs to floats. In the example executions below, canada.tsv is the only file without any not-a-number values, so its column values remain integers.

Missing rows: A file is considered to be missing a row if it does not have a *date* value that can be found in one of the other files. These missing rows should also be treated as NaN values in the unified DataFrame.

Case counts: The program doesn't have to make sure that the *cases* counts listed in the file are reasonable – it should only check if they are numbers according to the formatting examples above. All values that can be interpreted as numbers are valid input.

Dates: The program doesn't have to make sure that the dates listed in the file are reasonable. All values in the *date* columns are valid dates, even if they require reformatting.

Output of the program

Pay close attention that the output of your program is approximately the same as the example execution below (letter case, line breaks, spaces, full stops, commas, exclamation points or question marks are not checked even though they might be highlighted in the test feedback.)

Some hints and helpful functions

Reading TSV files: Tab-separated values (TSV) files are basically the same as comma-separated values (CSV) files, just with a different separator. They can be read using the pandas read_csv function with a tab as the "sep" parameter value. In the case of this exercise, because our files can include numbers with thousands separators (dots, in particular), we should also first read the data as strings (not floats/integers), using `str` as the "dtype" parameter value. If we were to read in the data without a specified data type, the DataFrame could automatically interpret some values as floats even when they're supposed to be integers in the thousands. These strings can then be converted into numeric types, once the thousands separators have been removed (see "Reformatting *cases* values" below). In short, for this exercise, the data should be read in like this: `df = pd.read_csv(filename, sep="\t", dtype=str)`.

Reformatting *cases* values: As seen in the table above, the numbers used in the files' *cases* values can come in four different formats, representing four ways of separating groups of three digits in values above 1000. To reformat an entire column, it is best to first replace each possible thousands separator with an empty string (`""`), and then convert the column into numeric form. Values that cannot be converted into a numeric data type are considered invalid, and they should be coerced into `NaN` values.

Reformatting dates: Dates can be reformatted in many ways. Because regular expressions (regex) are not covered in this course, we use the datetime module for this. The code for reformatting dates is given in the "Regarding date formats" section below.

Merging DataFrames: There are many ways to combine two DataFrames. In this exercise, once we have reformatted the *date* and *cases* values for all three DataFrames, we want to them merge them side-by-side so that they share the "date" column. To merge two DataFrames easily, we can use the pd.merge(), function, like this: `merged_dataframe = pd.merge(dataframe1, dataframe2, how='outer', on='date')`. We can then repeat this step to add our third dataframe: `merged_dataframe = pd.merge(merged_dataframe, dataframe3, how='outer', on='date')`. By using the `how='outer'` parameter, we specify that the merged DataFrame should include rows (date values) that appear in either DataFrame (an outer join).

Renaming columns: An easy way to rename the columns of a DataFrame is to first create a list of the desired column names, and then to assign it to the .columns attribute of the DataFrame:

```
column_names = ['date', 'cases_usa', 'cases_mexico', 'cases_bhutan']
merged_dataframe.columns = column_names
```

Note that you will have to create this list of column names based on the filenames inputted by the user.

Regarding date formats

Due to the wide range of existing date formats, dates can often be ambiguous. For example, it is impossible to know if 01/02/03 means January 2nd, 2003, or February 1st, 2003. Generally speaking, date formats can often be inferred from the country that the data was published in: for example, the United States tends to use Month/Day/Year with leading zeros. In international contexts, ISO 8601 is considered the standard. Often slashes (`/`) are reserved for month-day-year formats while dots (`.`) are reserved for day-month-year combinations, but this is not necessarily universally true. Date formats are also often stated in the documentation of the dataset.

When we cannot tell the date format from the context of the dataset, we have to infer it from the numbers within the dataset. If we do not want to use external libraries or regular expressions, a simple (although crude) way of doing this is to attempt to convert the date strings into DateTime objects using different date formats. This test conversion can be done via try and except statements and the datetime.strptime method. Whichever date format that works for all of the dates in the file is the valid one. For this exercise, the list of possible date formats is given above.

To make the exercise simpler, we give you the function `get_date_format`, which takes a DataFrame column and returns a date format that works on all of the values in the column:

```
import datetime
DATE_FORMATS = ["%Y-%m-%d", "%d.%m.%Y", "%d.%m.%y", "%d-%m-%Y", "%d/%m/%Y", "%m/%d/%Y"]

def get_date_format(dates):
    # Gets a Pandas Series of dates as strings and attempts to convert each string to a DateTime object
    # Conversion is attempted with every format in 'DATE_FORMATS' list
    # Returns a date format that works for all date strings
    for date_format in DATE_FORMATS:
        reject_format = False
        i = 0
        while not reject_format:
            try:
                datetime.datetime.strptime(dates[i], date_format)
                if (i + 1) == len(dates):
                    return date_format
            except ValueError:
                reject_format = True
            i += 1
    raise Exception('An error occurred: no valid date format found.')
```

This function should be called separately for all three DataFrames. Once you have a given DataFrame's date_format, you can use the pandas.to_datetime function to convert the entire column into the format, changing the column's values into DateTime objects. Finally, you can drop the "time" values from these DateTime objects, leaving you with only dates:

```
date_format = get_date_format(df['date']) # Gets an appropriate date format for the DataFrame
df['date'] = pd.to_datetime(df["date"], format=date_format) # Converts column values to DateTime object
df['date'] = df['date'].dt.date  # Removes time value from DateTime object
```

Submitting

When you have written your program, run it with a couple of different inputs and check that the output is correct. Submit the file `covid_statistics.py` to A+.

Examples of the execution of the program:

```
[execution of the program begins]
Enter the name of file #1:
canada.tsv
Enter the name of file #2:
south_korea.tsv
Enter the name of file #3:
brazil.tsv

Printing summary statistics:
       cases_canada  cases_south_korea  cases_brazil
count     31.000000          28.000000     29.000000
mean    6400.741935         553.464286  50038.620690
std     1626.369842         177.974089  19046.553334
min     4013.000000         303.000000  17341.000000
25%     4804.500000         402.250000  28323.000000
50%     6291.000000         518.000000  59119.000000
75%     7393.500000         654.500000  62334.000000
max    10209.000000        1020.000000  87843.000000

Printing first five rows:
         date  cases_canada  cases_south_korea  cases_brazil
0  2021-01-01          4736              824.0       24605.0
1  2021-01-02         10209              651.0           NaN
2  2021-01-03          7135             1020.0       17341.0
3  2021-01-04          7916              715.0       20006.0
4  2021-01-05          7220              839.0       56648.0
[execution of the program ends]
```

```
[execution of the program begins]
Enter the name of file #1:
canada.tsv
Enter the name of file #2:
south_korea.tsv
Enter the name of file #3:
canada.tsv
File canada.tsv is already included in this analysis. Please choose a different file.
Enter the name of file #3:
south_korea.tsv
File south_korea.tsv is already included in this analysis. Please choose a different file.
Enter the name of file #3:
brazil.tsv

Printing summary statistics:
       cases_canada  cases_south_korea  cases_brazil
count     31.000000          28.000000     29.000000
mean    6400.741935         553.464286  50038.620690
std     1626.369842         177.974089  19046.553334
min     4013.000000         303.000000  17341.000000
25%     4804.500000         402.250000  28323.000000
50%     6291.000000         518.000000  59119.000000
75%     7393.500000         654.500000  62334.000000
max    10209.000000        1020.000000  87843.000000

Printing first five rows:
         date  cases_canada  cases_south_korea  cases_brazil
0  2021-01-01          4736              824.0       24605.0
1  2021-01-02         10209              651.0           NaN
2  2021-01-03          7135             1020.0       17341.0
3  2021-01-04          7916              715.0       20006.0
4  2021-01-05          7220              839.0       56648.0
[execution of the program ends]
```

```
[execution of the program begins]
Enter the name of file #1:
canada.tsv
Enter the name of file #2:
atlantis.tsv
Error in reading file atlantis.tsv. Closing program.
[execution of the program ends]
```

Color codes:
Blue: Input by the user
Green: Output by the program
Red: Side remark

covid_statistics.py

Choose File No file chosen

Submit

Feedback (mandatory)

Points 0 / 0 | My submissions 0 | Deadline Monday, 14 March 2022, 18:00 | To be submitted alone

This course has been archived (Monday, 17 April 2023, 12:00).

Feedback

Enter an estimate of the time spent on this section. You can also add free-form feedback of this section (material and exercises).

*
How much time did you spend going through the material and on the video lectures for this section?

- ○ 0 minutes
- ○ 15 minutes
- ○ 30 minutes
- ○ 1 hour
- ○ 2 hours
- ○ 3 hours
- ○ 4 hours
- ○ 5 hours
- ○ 6 hours
- ○ more

*
How much time did you spend completing the exercises of this round?

- ○ 0 minutes
- ○ 15 minutes
- ○ 30 minutes
- ○ 1 hour
- ○ 2 hours
- ○ 3 hours
- ○ 4 hours
- ○ 5 hours
- ○ 6 hours
- ○ 7 hours
- ○ 8 hours
- ○ 9 hours
- ○ 10 hours
- ○ 11 hours
- ○ 12 hours
- ○ 14 hours
- ○ 16 hours
- ○ 20 hours
- ○ more

*
How much time did you spend on something else than mentioned previously in this round (leave out the minutes you entered in the previous questions).

- ○ 0 minutes
- ○ 15 minutes
- ○ 30 minutes
- ○ 1 hour
- ○ 2 hours
- ○ 3 hours
- ○ 4 hours
- ○ more

If you submit this feedback more than once, the times stated on your latest submissions remain. The times of different submissions are therefore not added up.

Enter feedback of this round and the related exercises.

Submit