Multi-dimensional DataFrames for analysis and manipulation of weather data

Afshin Zamani Zakaria

December 3, 2025

1 Overview, aims and scope

This project is designed and executed to capture and master the intricate piplines in pandas and create a the big picture of structured data creation manipulation, cleaning, indexing and so on.

The following bullet points are covered in this project:

- Creation of multi-dimensional indexes for dataframes both for rows and columns.
- Adding `timeseries` column to the dataframe.
- Assigning systematic random data to the dataframes using predefined functions.
- Adding columns from predefined dictionaries.
- Merging and concatenation of dataframes.
- Introducing random imperfections to the data using Gaussian distribution.
- Add some irregularities to a string column.
- Use RegEx to clean the introduced irregularities.
- Group data and create descriptive tables based on the existing parameters on the dataframes (using pivot tables and grouping).
- Create plots and bar graphs from the dataframe.
- Create benchmark latex tables and include them in this document (compare pivot vs grouping, masking vs query and dataframe creation timings).
- Create an automated workflow for containerizing the python project that runs and updates the program as well as updating the current latex document.

2 Index creation and assigning data

- Create lists of parameters as below for row index:
  1. `Location = ['North','South','East','West']`
  2. `Sensor = ['S1','S2','S3','S4']`
  3. `'time'` as a pandas time series that includes dates "2012-01-01" to "2018-06-01" with a frequency of hour.
  4. Split the time into am and pm.
  5. Create two multiindexes one using (Location, Sensor, `time_am`) and the other with combination of (Location, Sensor, `time_pm`) for rows.
- Create lists as below for column indexes:
  1. `data_type = ['Temperature','Humidity']`
  2. `trial = ['First','Second','Third','Fourth']`
  3. Create two multiindexes for columns from product of ('Temperature', trial) and ('Humidity', trial).

- Create four null arrays with the following lengths to create four dataframes:
    1. Temp_am = am_index length * trail length
    2. Temp_pm = pm_index length * trial length
    3. Humidity_am = am_index length * trail length
    4. Humidity_pm = pm_index length * trail length
- Create four dataframes (`df1, df2, df3, df4`) with the created data in the previous step and corresponding row and column indexes.
- Create four dataframes with resetted indexes (These will be used later) (`df1_resetted, df2_resetted, df3_resetted, df4_resetted`).

3 Build functions library for structured random data creation for dataframes

According to the datasheet of sensors manufacturer, each sensor (`'S1','S2','S3','S4'`) are behaving differently with some variation during 12 hours of usage. Here is the list:

1. `S1`: Exponential (`a.exp(x)+b`)
2. `S2`: Linear (ax+b)
3. `S3`: Logarithm (a.ln(x+1)+b) # added 1 inside the natural log, so the array of `np.arange(12)`, which starts from 0, would not cause negative sign inside `ln`.
4. `S4`: Trigonometric (a.$|\cos(\frac{2\pi x}{11})|$+b) # receives an `np.arange(12)` and makes a full cycle of 2π.

- All sensors are creating temperature variation of 5 degrees Celsius in a day with up to 2.5 degrees actual temperature variation. Also they create a fluctuation (error) of 10 percent in humidity with up to 5 percent of actual humidity variation in 12 hours (Suppose there is no tangible change in the temperature and humidity in a single day).
- Temperature values are in the range of 10 to 60 degrees and Humidity have the values in the range of 0 to 100.
- Each function should take a optional seed and create all of the outputs using that given seed. So, the function will be in the form of `func(array, 'Temperature', seed=None)`. Seed can be any number in the dataframe so the created temperature and humidity will be unique. However, all trials should be created using the same seed with minimal random variation introduced.

4 Create dataframes using the generated random functions

- create a dictionary with the keys of "df1", "df2", "df3", "df4" and values as lists like [df1_resetted, 'Temperature', df1_t], where `df1_t` is the creation time of the dataframe to be measured.
- Create a `group_id` column for each dataframe in the dictionary above by `groupby` method. Each group should have 12 members ID starting from 1 to (`day*Sensor*Location`).
- record the start time.
- Use a loop iterated in `trial` and create data for each column using the `group_id` column created earlier and function library defined for each sensor type. In this step, use groupby method to fill all the columns of the dataframes with number and print dataframes to verify they are obeying the functions in the library.
- Drop the `group_id` column added earlier.
- record the data creation time by deducting the start time from the current time and assign it to the `df1_t`, ... defined in the dictionary earlier.
- Restore the hierarchical indexes from the resetted dataframes.
- Merge and concatenate the dataframes (`df1, df2, df3, df4`) into a single dataframe `df. df1` and `df3` should be added horizontally using merge (They are both for the same time with temperature and humidity data, respectively and sharing the same row index). Likely, `df_2` and `df_4` should be added into `df_24`. Later df_13 and `df_24` should be concatenated vertically.

5 Add "average", "flag" and "Operator" columns to the created dataframes

- Add a column named "average" to each of `df1, df2, df3, df4`, averaging all of the data in trials.
- Add a column named "Condition" to each of the above dataframes based on their average values if their range are in the following bins:
  `Temp_bins = [10, 22.5, 35, 47.5, 60]`
  `Humidity_bins = [0, 25, 50, 75, 100]`
  and flag them according to the following list:
  `labels = ['Low','Below average','Above average','High']`
- Add `Operator` column randomly to `df1, df2` from the list below:
  `operators = ['George','Michael','David','Sara','Alison','John']`

6 Add string irregularities to the `"Operator"` column and clean these irregularities using RegEx

- Create irregularity function based on the following criteria:
  1. Randomly make the whole string uppercase or lowercase.
  2. Add one or three of the following characters anywhere in the string: space, "_" or "-"
  3. Add a random number from the list below to end of the string:
     `['12','65','23','156','189']`
  4. Add a special character from the special characters list below:
     `['@','<>','()','# ',' ']`
- Apply irregularity function to the `Operator` column in `df2` and `df4`.
- Merge and concatenate dataframes (`df1, df2, df3, df4`)

7 Create descriptive tables by "`groupby`" and "`pivot_table`" methods

Create the following tables using `groupby` and `pivot_table` methods:

1. The average and standard deviation of temperature and humidity by year for all sensors.
2. The average and standard deviation of temperature and humidity by year for all location.
3. The average and standard deviation of temperature and humidity by year for each operator.

Create a bar plot of the pivot tables with error bars and save the plot to "`output`" directory.

8 Masking and query benchmarking

Create the following masks and queries and store their creation time in a dictionary:

- Mask temperatures higher than 35 degrees for columns 1-4 of merged dataframe (`df`).
- Mask humidity higher than 50 percent for columns 7-11 of merged dataframe (`df`).
- Use query to filter temperatures higher than 35 degrees for columns 1-4 of the merged dataframe (`df`).
- Use query to filter the humidity higher than 50 percent in columns 7-11 of the merged dataframe (`df`).

9 Creating output LaTeXtables

Create the following LaTeXtables and save them inside output folder in the current directory:

1. LaTeXtable of comparison for the timing of dataframe generation using `"groupby"` and `"for loops"`.
2. LaTeXtable of comparison for the timing of pivoting and grouping methods.
3. LaTeXtable of comparison for the timing of query and masking.

10 Create docker image and container from the created files and push them to GitHub

- Create `Dockerfile` for the current project to containerize and push to GitHub.
- Initiate , add and commit all files into a git repository.

11 Results: plots and benchmark tables

Here is only the results of graphs and tables from the repo without further discussion of the results and only to prove the automated compiling workflow.

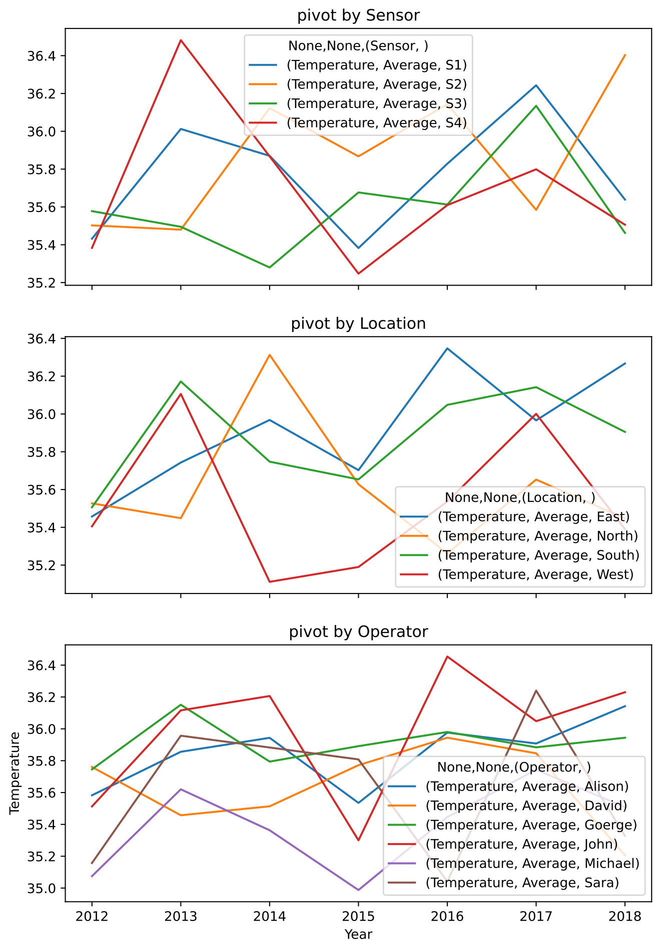

Figure 1: Average temperature by year line plot.

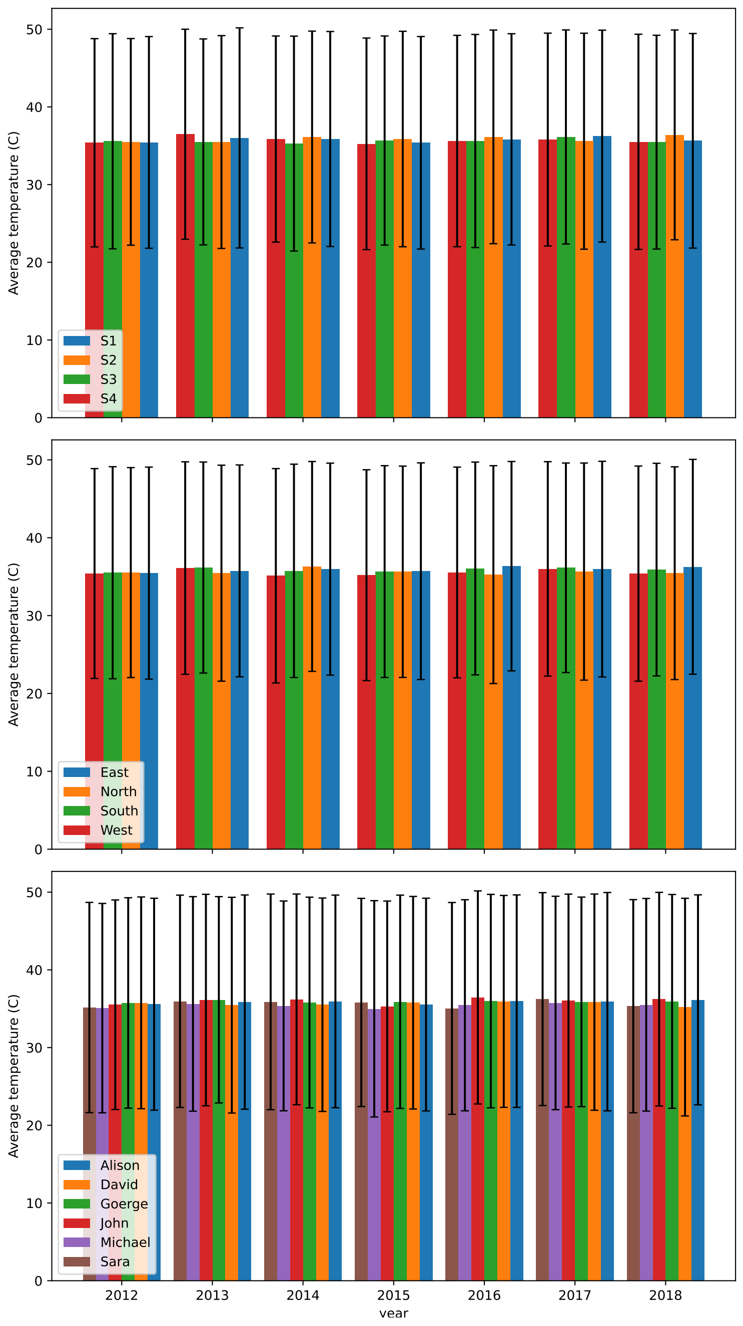

Figure 2: Average temperature by year bar plot.

Table 1: Comparison of timing for normal masking versus query method for dataframes.

Category	Measurement	First	Second	Third	Fourth
Masking	Temp	0.062180	0.057300	0.054410	0.052624
	Humid	0.052664	0.053766	0.052406	0.052703
Query	Temp	0.101588	0.101652	0.100417	0.100653
	Humid	0.101039	0.102682	0.100372	0.101337

Table 2: Benchmark of the timing for `groupby` versus `pivot` methods.

	First	Second	Third
pivot	0.240192	0.232900	0.330445
groupby	0.106818	0.103377	0.185415

Table 3: Comparison of the times to create dataframes using cython level `groupby` and normal python loops (Dates in columns headers are the end dates of the dataframes with starting date fixed to `'2012-01-01'` and the unit for table is seconds).

	2012-04-01	2012-07-01	2013-07-01	2015-01-01	2019-01-01
Looping method	11.317440	26.517643	121.238940	364.711107	1532.432478
Groupby method	6.637370	13.230573	39.649977	83.548375	192.231985

12 Conclusions on the benchmark tables

Here are the main conclusions of the benchmark tables presented earlier. Try to change the `end_dates` in the main script (`multidim_sensor_analysis.tex`) to customize the results as the latex will update automatically.

- Masking of dataframes is almost two times faster than `query` method with more keystrokes trade off.
- There is an identical way to create pivot tables by `pivot_table` and `groupby` method. While `groupby` method is faster compared to `pivot_table`, it needs more lines of code.
- Creation times for small dataframes is almost identical using loops and groupby methods. However, groupby method timing scales almost linearly with the size of the dataframe, while looping method scales exponentially. So, for larger dataframes, using cython level `groupby` methods pays off.