
**Budapest University of Technology and Economics**
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

# Critical Systems Integration Laboratory

# Data Analysis Techniques for Benchmark Evaluation

Ágnes Salánki, Attila Klenik

October 23, 2019

# Contents

# Chapter 1

# Data Analysis Techniques for Performance Evaluation

**Authors:** Ágnes Salánki, Attila Klenik

## 1.1 How to read this document

This short syllabus summarizes the basics of a typical data analysis workflow, provides a very high-level overview of some useful Python packages (e.g., Pandas, SKLearn and Bokeh), Jupyter Notebook for creating self-contained data analysis reports and the basics of linear regression.

## 1.2 Requirements

**The solution that students submit after the session has to meet the following criteria:**

- The solution is documented in the Jupyter Notebook.
- Each plot has the following minimal documentation (where not trivial):
- Axis names, scales and metrics
- How the data was acquired? (if the data transformation is not trivial)
- Each plot answers a dedicated question and is interpreted in 1-2 sentences.
- Each exercise is at least started, if not completed.

## 1.3 Practice

**Before the laboratory**, you have to perform a performance test on your implementation and upload the results in a well-formed .csv file here. At the beginning of the lab a collector script will create one large data set from your files (`MERGED.csv`), this will be your input data set to analyze. **Without your uploaded benchmark file, you will not be allowed to start the lab.**

## 1.4 "Uploading" the file:

- Fork the BenchmarkPool repository.
- Clone the forked repository.
- Place the `<team_name>.csv` file into the `validator/benchmark_files/2019` directory.
- Add, commit and push the changes to your fork.
- Open a pull request on GitHub, and wait for your changes to be merged.

**The deadline to open the pull request is 2019.10.27. 12:00 UTC+2 (Sunday noon)!**

## 1.5 Measurements

The input files (parts from two Jane Austen novel with varying sizes) are here.

You should run a comparison on **word granularity** with a **shingle size of 2** for each `Sense` (Sense and Sensibility) and `Pride` (Pride and Prejudice) text pair, meaning 36 (6x6) comparisons at the end (`Pride1.txt` and `Sense1.txt`, `Pride1.txt` and `Sense2.txt`, etc.). Please run each comparison independently from each other to assure the computation times to be as precise as possible. **Use the Akka-based solution for comparison!**

The format of the output is theoretically the same as what you used for logging, besides presenting the name of the team and measurement ID as extra columns. So, it has to contain 6 columns, in this order: `type`, `name`, `input_id`, `time`, `team`, `meas_id`.

- `type` is either `QUEUE`, `START` or `END`
- `name` is the unique name of the processing node in your workflow, like `Tokenize1`, `ComputeScalar3`, etc.
- `input_id` is the name of the document (`Sense1`, `Pride6`, etc.) or document pair (`Sense1_Pride6`, `Sense6_Pride1`, etc.) the node is working on. **For the document pairs, please follow the syntax above.**
- `time` is the output of your `System.nanoTime()` call;
- `team` is the name of your team.
- `meas_id` is the document pair names on which you perform the current measurement, e.g., `Pride1_Sense1`, `Pride1_Sense2`, etc. **Enumerate the 36 combinations in this format (`PrideX_SenseY`), so it's consistent across teams!**

**Every row in your CSV must be unique if you properly set the above attributes!**

Based on our experiences, it is possible that your solution is not fast enough to produce the results for each pair of documents. (Note that Sense6 and Pride6 contain words in order of magnitude of 10.000.) In this case please leave the time column empty (note the last row in the output examples) but keep the configuration information in each other column.

*It is possible that your logging format implementation differs from the above format. In this case, don't change the implementation, but perform some basic log transformation (like find & replace) on your initial log file.*

If you have any questions or comments about the input format feel free to reach me.

## 1.6 Validating your CSV

This is the script which will validate your CSV log file and merge each submission together into a single large CSV. If you feel uncertain about your submission, you should test it with the script above until it does not return with an error. Example CSV files can be found here.

**If this script fails on your submission then you will not be allowed to start the laboratory.**

Requirements to run the script:

- Pyhton 3.X
- The following Python packages (installed through your favorite package manager):
- glob2
- pandas
- numpy

Once you place your CSV in the `validator/benchmark_files/2019` directory, run the following command from the `validator` directory:

```
python ./validateAndMerge.py ./benchmark_files/2019
```

## 1.7 Project structure

Create an `eda` root directory in your team repository, which will contain every artifact needed to reproduce your results (and serve as the working directory for the notebook docker container):

- The `MERGED.csv` data file

- The Jupyter notebook file
- (The docker-compose file for starting the Jupyter Notebook server)

---

## 1.8 The technology stack

### 1.8.1 Programming

The primary coding language of this laboratory is Python. In order to spend the laboratory time with meaningful analysis tasks, obtaining a basic understanding of Python syntax is highly recommended before the laboratory session. If you have never used the language, please consider gaining some experience. Many good *getting started* tutorials are available, an excellent one is this by w3schools.

The gathered data will be stored in Pandas data frames, a great library for table-like data manipulation. A short (10 minutes) introduction of Pandas is available here.

### 1.8.2 Visualization

As you will see, visualization is a key component in analytics. Any type of visualization is allowed in the lab, however, the recommended library to use is `bokeh`. A quick-start guide for `bokeh` can be found here. It introduces the high-level concepts of `bokeh` and provides some examples for basic visualization. `bokeh` makes it easy to create highly customized multi-figure plots with a few lines of code. During the seminar, you'll use the `output_notebook()` mode of `bokeh`, since you're working in the Jupyter Notebook environment.

### 1.8.3 Reporting

For documenting the tasks, students have to create a Jupyter Notebook, which allows mixing (Python) code and (Markdown) documentation (similarly to R Markdown). An introduction to Jupyter Notebook can be found here.

## 1.9 Starting the Jupyter Notebook server

The easiest way to get started is to use the appropriate Docker image to start a Jupyter Notebook server, and simply connect to it through your favorite browser. The advantage of using such prebuilt environments is that they usually contain a LOT of preinstalled libraries/packages, so we don't have to manage these in our own machine

Execute the following command from your `eda`directory:

```
docker run -i -p 8888:8888 -v "./:/home/jovyan/notebooks" jupyter/scipy-notebook
```

Note, that after running this command, the current working directory (`eda`) will be attached to the running container as the `/home/jovyan/notebooks` directory, so the notebooks saved in that directory will be automatically available on the host. After executing the command something similar should appear on the output:

```
Executing the command: jupyter notebook
[I 11:50:17.179 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtim
[W 11:50:17.540 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encry
[I 11:50:17.590 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterla
[I 11:50:17.590 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 11:50:17.602 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 11:50:17.603 NotebookApp] The Jupyter Notebook is running at:
[I 11:50:17.603 NotebookApp] http://(2b34f9d720ea or 127.0.0.1):8888/?token=00989fb56061981545b2caee52a3ea13fa
[I 11:50:17.603 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confir
[C 11:50:17.603 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(2b34f9d720ea or 127.0.0.1):8888/?token=00989fb56061981545b2caee52a3ea13faf05da78146d5ca
```

Accordingly, the server can be reached at `http://127.0.0.1:8888/?token=00989fb56061981545b2caee52a3ea13faf05da78146d5ca`

Another way to launch the server is to create a `docker-compose.yaml` file in the `eda` directory with the following content:

```
version: '3.0'

services:
  jupyter:
    image: jupyter/scipy-notebook
    ports:
      - 8888:8888
    volumes:
      - ./:/home/jovyan/notebooks
```

Then simply execute the following command from the `eda` directory:

```
docker-compose up
```

---

## 1.10 Data Analysis Basics

The goal of data analysis projects is extracting data-based information about an observed system or phenomenon. In an ideal case, at the end of the project, we have a deeper insight into how our system works (what is happening?) and have some hypotheses about the cause-effect relationships (why is it happening this way?).

In our case, the observed system is the implemented workflow, our goal is to analyze its performance, find the possible bottlenecks and document suggestions for code improvement (both data structures and algorithms).

The analysis usually consists of two steps: exploratory and confirmatory phases.

### 1.10.1 Exploratory data analysis (EDA)

This phase consists of activities related to the exploration of the system. It answers the most basic questions related to the *marginal* and *joint* distributions of variables, e.g., the marginal distribution of each individual variable and basic relationships between them. Since one- and two-dimensional visualization techniques are excellent (i.e., intuitive, fast, reliable) tools for extracting this information, the exploratory analysis in practice means the plotting of, and inspection into, many graphical plots.

The most frequently used one- and two-dimensional visualization techniques are:

- Histograms and boxplots for one-dimensional numerical variables;
- Barcharts for one-dimensional categorical variables;
- Scatterplots for visualization of two numerical variables;
- Mosaic plots for visualization of two categorical variables;
- Colored/grouped one-dimensional plots (either histograms and boxplots) for visualization of the relationship between a categorical and a numerical variable.

### 1.10.2 Confirmatory data analysis (CDA)

At the end of the EDA phase, we already have some ideas, so-called hypotheses about the basic phenomenon in the system. E.g., *the distribution family of the processing time is a two-modal Gaussian.* However, to prove (or publish) these, ad-hoc ideas are not enough, we need *statistically significant* results. This is the main task of the CDA phase. Primary tools here are the statistical tests (z-test, chi-square test, etc.).

This laboratory focuses on the exploratory phase, thus, concepts of statistical testing are not covered here.

---

## 1.11 Data Manipulation

Data representation can be *long* or *wide*.

- Long data frames have as few as possible columns, minimally three: the ID of the object, a feature name (what we have measured, observed, etc.) and the value itself.
- Wide data frames have one ID column and the features appear in separate columns.

**Running example:** we have run three experiments on two different documents, each of them producing two processing times (e.g., one for tokenization and one for shingle collection). The results can be stored in both long and wide way.

General Data:

```
  Document.ID Experiment Tokenize.Times ShingleCollecting.Times
1           A          1            0.5                     0.6
2           A          2            0.6                     0.5
3           A          3            0.7                     0.8
4           B          1            1.3                     1.6
5           B          2            1.2                     1.4
6           B          3            1.6                     1.2
```

Long format:

```
   Document.ID Experiment                variable value
1            A          1          Tokenize.Times   0.5
2            A          2          Tokenize.Times   0.6
3            A          3          Tokenize.Times   0.7
4            B          1          Tokenize.Times   1.3
5            B          2          Tokenize.Times   1.2
6            B          3          Tokenize.Times   1.6
7            A          1 ShingleCollecting.Times   0.6
8            A          2 ShingleCollecting.Times   0.5
9            A          3 ShingleCollecting.Times   0.8
10           B          1 ShingleCollecting.Times   1.6
11           B          2 ShingleCollecting.Times   1.4
12           B          3 ShingleCollecting.Times   1.2
```

Wide format:

```
  Document.ID 1_Tokenize.Times 1_ShingleCollecting.Times 2_Tokenize.Times 2_ShingleCollecting.Times 3_Tokenize
1           A              0.5                       0.6              0.6                       0.5
2           B              1.3                       1.6              1.2                       1.4
```

The long format allows the exploration of processing time values together, the wide format is ideal for computing an aggregate value over processing times (e.g., the median of times). Somewhere in between, the processing.times.experiment is good for calculating e.g. the sum of the times or for the analysis of only one attribute.

As conclusion: there is no “ideal” data format, it should be adopted to the characteristics of the task and the variables to be emphasized.

Fortunately, Pandas data frames provide a so-called pivot operation to transform long data format to wide format.

---

## 1.12 Regression

Regression is one of the most frequently used machine learning methods; the goal is to find a function that estimates a single numeric *target variable* we want to predict, based on given *input variables*, so that the estimate is as good as possible. Assuming that a function

family (linear, exponential, etc.) has already been chosen, the main task of regression is to choose the parameters of the function (that identify a single member of the family) so that the function fits the data (estimates/predicts the target variable) as well as possible (minimizing some measure of error).

**Note:** choosing a function family is not in the scope of regression, it has to be performed in another way. In this example it is going to be a human task and detection will be made by visualization.

Then, what does regression do? It computes:

- the best parametrization minimizing estimation error,
- the residuals for given data points and
- the goodness of fitting – a numeric value indicating how well the function works.

### 1.12.1 Details

The example above is simple enough to give an idea of what we can expect while using the `fit` function of SKLearn. Below, there is a brief summary about the basic definitions of linear regression.

Assuming linear relationship between two numeric variables `X` and `Y`, we are searching for the best `(a, b)` parametrization of the line `Y = aX + b`. `X` and `Y` are variables with `(x_i, y_i)` data points in pairs. After we computed coefficients `a` and `b`, we can calculate a target `f(x_i)` value for each `x_i` data point.

- **Residual** – the difference between estimated `f(x)` and actual `y` value for a single `x_i` data point.
- **LSE** – Least Square Error is computed as the sum of residual squares: `LSE = sum(y - f(x))^2)`
- **R-square values** – it is a good indicator of how strong the established relationship is: it determines *how the variance of the target variable can be explained by the variance of input variables.* In the case of linear regression, it is simply the (Pearson's) correlation coefficient between Y and the vector containing the predicted values.

**Acceptance rules of thumb:** the linear model is considered good, if

- the R-square value is high enough and
- the distribution of residuals is Gaussian.

A nice overview of using either the SKLearn or Statsmodels library for linear regression problems can be found here

## 1.13 Exercises

1. Write a script which will transform your .csv into a data frame containing only processing times of nodes in `ms` and not the output of the `System.nanoTime()` call. (1p)
2. Analyze *your own* processing times first! How does the processing time of the individual task types scale according to the input sizes (note, that the size of the input texts increases exponentially)? (2p)
3. How determined are the processing times among different rounds of running? (Note that you will run the Tokenize process e.g. on Sense1 6 times) (2p)
4. How do the executed steps of the workflow contribute to the total execution time? How does this change when increasing the input size? Can you guess the performance bottleneck of your solution based on these data? (2p)
5. Read in the large .csv file containing the results of other teams and compare your times with those of other teams. (2p)
6. Document your results in a Jupyter Notebook file and upload it to your repository. (1p)

The team with the best visualization gets 1 bonus point.

## 1.14 Remarks

1. Bokeh visuals work with `ColumnDataSources` (CDS) and views derived from them. You can bind a CDS to a pandas data frame (DF). It is your choice whether you do the necessary data filtering on the DF first, then bind to the CDS, or bind the original DF to the CDS and derive additional view from the CDS. *Note: sharing a CDS among multiple visuals is an easy way to use cross-selection/filtering on the plots.*

2. General remark for every exercise: use the “pythonic” and pandas ways to perform your data transformations. You probably don’t need a for-loop to iterate over the DF, pandas has built-in functions for most scenarios. This will make your code more efficient and more readable!
3. For exercise 1: Since you will work with the wide format DF throughout the seminar, this step is crucial. If you suspect that your solution might be wrong, ask for help!
4. For exercise 2:
5. Since your processing nodes have unique IDs, you need to derive the type of the node from its name (which should be easy if you followed the suggested logging format).
6. The input text sizes are not included in your log by default. But you can process the files easily (defining some measure of size), assemble a DF from it, which can be merged with your original data.
7. For exercise 4: There could be overlaps between the processing nodes, so be careful about defining the “total” execution time.
8. For exercise 5: Comparing the performance of different solutions needs a target metric that will serve as the basis of comparison. It is up to you to select this metric. Different metrics could result in different winners :)

## 1.15 References

- Python
- Pandas
- Bokeh
- Jupyter Notebook
- Pandas pivot
- Linear regression in Python