Introduction to Data Science
WS24/25 Assignment Part 2

Prof. Dr. Wil van der Aalst, Nina Graves,
Lukas Liss, Christian Rennert, Christopher Schwanen,
Leah Tacke genannt Unterberg

Chair of Process and Data Science
RWTH Aachen University

December 17, 2024

Introduction

In this part of the assignment, you are going to explore some of the more specialized data science approaches covered in the latter half of the lecture. Each question comes with its own dataset(s), as the techniques work on wildly differently structured data: e.g., NLP and time series. You will start with finding patterns in online shopping data via frequent itemset and association rule mining, then get into some sentiment analysis of twitter data via word embeddings and n-gram-based NLP. Next, you will explore the loan application process of a financial institute with process mining techniques, including discovery and identification of performance bottlenecks. Second to last is an explorative time series analysis of the number of recorded UFO sightings over time. Finally, you can have a go at some small-scale big data analysis via (emulated) map reduce as a distributed data processing technique.

Outline

- Frequent Itemsets and Association Rules (23 points)
  - `online_retail.csv`
- Natural Language Processing (21 points)
  - `emotions.csv`
- Process Mining (21 points)
  - `event_log.xes`
- Time Series Analysis (25 points)
  - `ufo-sightings.csv`
  - `ts-yearly.csv`
  - `ts-monthly-10yrs.csv`
  - `ts-train.csv`

    - `ts-test.csv`

- Distributed Data Processing (10 points)
    - `movies.csv`
    - `ratings.csv`
    - `best_movies.py`
    - `movie_id_to_name.py`
    - `haters.py`

Assignment Details

- Total number of points obtainable: 100
- Group size: 2 to 3 students
- Input:
    - this assignment PDF,
    - an **optional** LaTeX template `report_template.tex`,
    - Jupyter notebook templates (one per question),
    - a collection of data sets in named subfolders, and
    - a conda environment file `environment.yml`[1].
- Deadline: 20.01.2025, 23:59:59 (CET)
- Deliverables (to be uploaded as a zip file in Moodle):
    - PDF report (max. 30 pages of content),
    - Jupyter notebooks (one per question),
    - and if you used LLMs: a document or section detailing your use of LLMs in accordance with the Study Guide.

Report. Your written report is the primary basis for grading. In your report, you should concisely but comprehensively present your answers to the questions. Use whole sentences, and include captioned figures or tables if required by the task. Doing so, **clearly indicate which answer belongs to which question.** Please order your questions according to the numbering of this assignment to avoid confusion. The tasks are relatively narrowly guided so that you should not have to make additional assumptions. If you do, state them. Moreover, the report should be **self-contained**, i.e., it should not require references to the notebook. The length should be at **most 30 pages of content**, that is, excluding title page and possible LLM usage statement. Make sure to include the **group number and all group members' names on the title page**. Please respect the following reporting criteria (Severe problems may lead to a point deduction of up to 10 points):

- Proper spelling, punctuation, readability, and comprehensive structure.
- Use of **adequate** visualizations for showing (aggregated) results or illustrating methods.
- Figures have captions, axes have labels, diagrams have headers.

[1]The Jupyter Hub profile has been updated to match this.

- Figure quality (e.g., resolution and relevance).
- All figures, tables, and similar are numbered and referred to in the text.
- All your comments, descriptions, discussions, and interpretations should be explicit and concise. Usually, no more than 2–3 sentences are needed to answer individual parts of a question.

Notebooks. Your Jupyter notebooks will be used for reference and potentially for testing your code. Therefore, they should satisfy the following requirements:

- They have to be executable in the bundled conda environment—that means additional imports are fine but installing additional packages is not!
- Commented and structured code (if not, this will be penalized in the style points).
- Questions separated by markdown headers.
- Top-to-bottom runnable cells to reproduce your results.
- DO NOT CLEAR THE OUTPUT of the notebooks you are submitting!
- Ensure that the code in the notebook runs if placed in the same folder as the provided files, delivering the same outputs as the ones you submit in the notebook and report on in your report.

Do not re-upload the datasets. Besides, notebooks that intentionally access files outside of the bundled subdirectories or are in any way harmful will be graded with zero points.

Optional Resources

- Jupyter: `https://jupyter.org/index.html`
- Jupyter Lab/Notebook installation guide on Moodle

Question:	1	2	3	4	5	Total
Points:	23	21	21	25	10	100

Question 1: Frequent Itemsets and Association Rules

In this question, you will explore a dataset (`online_retail.csv`) containing customer orders of an online retailer in the European economic area. The dataset is provided in the `frequent_itemsets_and_association_rules/` folder. The table below describes the features of the dataset.

Feature	Description
`BillNo`	Identifier of a customer order
`Itemname`	Name of the ordered item
`Quantity`	Number of items that was ordered
`Date`	Timestamp of when the order was placed
`Price`	Price per unit of the ordered item (in €)
`CustomerID`	Identifier of the customer that placed the order (if known)
`Country`	Country from where the customer placed the order

(a) (2 points) Load the dataset. Compute the total sales value (i.e., the total monetary value of products sold) and provide:

- the three items which generated the highest sales value and their sales value, respectively, and
- a plot showing the sales value per country.

Hint: Do not forget to account for the quantities.

(b) (3 points) Use the code snippet provided in the notebook to obtain the transactions based on the feature `BillNo`.

- Provide the number of transactions contained in the dataset.
- Specify the three most common items across all orders and their support.
- Provide a bar chart which shows the support per item (**except** the most frequent one) over all transactions in descending order. You do not need to include the item names in your plot. When using *matplotlib*, you can suppress the item names on the x-axis with the parameter `xticks=([])`.

(c) (1 point) Do the three items that occur most frequent across all transactions differ from the three items that generated the highest sales values? Explain why this is/is not the case.

Frequent Itemsets

(d) (2 points) Mine all frequent itemsets with a minimum support of 1 %. Provide the three most frequent itemsets which include at least three items, and also provide their support.

(e) (3 points) Now mine only maximal frequent itemsets with a minimum support of 1 %.

- Provide the three most frequent maximal itemsets which include at least three items, and also provide their support.
- Compare the three most frequent maximal itemsets including at least three items and their support to the resulting itemsets of part (d). Briefly explain the differences.

(f) (2 points) Consider all frequent itemsets containing at least six items. Per itemset size, state the number of frequent itemsets, closed frequent itemsets, and maximal frequent itemsets.

Association Rules

(g) (3 points) Based on all frequent itemsets (with a minimum support of 1 %), determine all association rules with a minimum confidence of 0.5. Provide a scatter plot of your result where the resulting association rules are represented by points. Use the x-axis for the support, the y-axis for the confidence, and the color for the lift.

(h) (4 points) Based on your discovered association rules, further investigate the one with the lowest and the one with the highest lift.

- Provide both association rules; therefore, state their antecedents and their consequents.
- Interpret both association rules with regard to their support, their confidence, and their lift.

(i) (1 point) The retailer wants to use association rules to add recommendations for other products in the online shop. What thresholds for confidence and lift would you suggest to distinguish good association rules from bad ones?

(j) (2 points) Filter your discovered association rules such that antecedent and consequent form a maximal frequent itemset.

- Provide the same scatter plot as above, but now only including the filtered association rules.
- How did the maximum values of support, confidence, and lift change?

Question 2: Text Mining

In this question, you want to investigate a preprocessed emotion-classified English Twitter dataset[2]. The relevant dataset is in the `text_mining/` folder.

The raw data set `emotions.csv` has the following columns:

- **text:** The text that a user wrote on Twitter about their emotions.
- **label:** The label of the emotion sentiment classification: if a text is classified to show love (`Love`), surprise (`Surprise`), joy (`Joy`), or anger (`Anger`).

You will explore how to generate your own emotion posts using maximum-likelihood estimation (MLE) on n-grams and how to classify your generated texts. For the classification, you will investigate the BoW (bag-of-words) model to preprocess a document for a classifier to learn from. Additionally, you are curious if there is information on emotions that people do or do not connect with Christmas. Therefore, you train a Word2Vec model based on skip-grams to learn contexts of words and to compare them.

First, you preprocess the data.

(a) (1 point) For the preprocessing, you start by balancing your data. Load the dataset `emotions.csv` and plot the distribution among the sentiment classes given in the data using a histogram.

(b) (1 point) As you see, there is an imbalance between the number of reviews in each sentiment label class. Apply undersampling by keeping the first k instances in each sentiment label class, where k is the number of instances in the smallest sentiment label class. You are required to write the undersampling method with this specifications yourself. Afterward, verify your approach by plotting and reporting the newly obtained distribution that should hold the same count for each sentiment label class.

In the following parts, you use only the sampled data. Further, you consider each entry in the dataset, i.e., each tweet, as a document and the entire set of documents as corpus.

(c) (1 point) Next, you create a function for a preprocessing pipeline that should work as follows for an input document:

1. Split the document into tokens,
2. unify the tokens into lowercase,
3. remove all stopwords,
4. lemmatize the tokens, and
5. join all preprocessed tokens into documents separated by whitespaces.

Test your preprocessing function on your corpus and determine how many tokens your corpus contains before and after preprocessing.

(d) (2 points) Split your unpreprocessed corpus into a training corpus and a test corpus such that your training corpus consists of 80 % of the original corpus. Use *scikit-learn*'s `train_test_split` with `random_state=12345`. Use stratification to make sure that you keep the distribution of target labels when sampling. Report the first three lines of each corpus, i.e., the training and test corpus, after your split.

Hint: The third line of training corpus should be:
"i feel calm complete and whole after i meditate".
For the test corpus, the third line should be:
"i had a funny feeling when i accepted them".

[2]Originally from: https://www.kaggle.com/datasets/nelgiriyewithana/emotions.

After all the required preprocessing, you continue by training a classifier based on bag-of-words.

(e) (2 points) In this question, you train the classifier based on the bag-of-words embeddings of documents. First, you learn a bag-of-words (BoW) model for your entire corpus. Make sure that your bag-of-words model uses your preprocessing method. Second, you transform each document from your corpus into a bag-of-words-transformed training embeddings. Third, you train a Stochastic Gradient Descent (SGD) classifier on the embeddings obtained. Use `log_loss` as loss function and `random_state=12345`. Report the accuracy on the training and test corpus using the trained SGD classifier. Round your results to the third decimal.

Hint: Remember to use your preprocessing function in your bag-of-words model.

(f) (4 points) Next, you want to generate text using n-grams with a maximum likelihood estimator and then classify your output. Therefore, create a new preprocessed corpus that only applies tokenization and unifies the tokens in lower case but does not apply lemmatization, does not remove stopwords, and does not join the tokens into new documents. Train a 2-gram and a 5-gram maximum likelihood estimator (MLE). Create the following deliverables for each MLE:

- Generate and report a full document and provide its bag-of-words-based sentiment classification.
- Generate and classify 1000 additional documents, then return the histograms for the generated documents classifications.

(g) (1 point) When investigating the two histograms, you realize that they are imbalanced. Argue in three sentences which distribution seems to be more balanced and why the chosen value of n for each n-gram influences this.

Hint: Remember to make a connection to the distribution of the input data for the n-gram model.

(h) (2 points) Now, you want to find out what emotions are connected to Christmas in the corpus. Therefore, you create embeddings based on Word2Vec with skip-grams. Use the following parameter and inputs for your Word2Vec model:

- Use the text corpus that you created for the n-gram model as input,
- choose the context window size to be 5 (`window=5`),
- choose the minimal word count to be 3 (`min_count=3`),
- choose the vector size of your embeddings to be 25 (`vector_size=25`),
- the number of training epochs of Word2Vec model to be 50 (`epochs=50`),
- the model to use skip-grams (`sg=1`),
- you are free to change the number of `workers` to parallelize the training, and
- leave all other parameters at their default values.

Once you are done, embed the word “christmas” and report the vector.

Note: A Screenshot is sufficient to report this embedding.

(i) (3 points) To further get some intuition of the embeddings, you want to find the most similar and most dissimilar words to the word “christmas” from your corpus. Report the top three words and their cosine similarities. Round to third decimal. Explain what it means if two embeddings have a high cosine similarity for our Word2Vec setting.

Hint: Make sure to use the correct sign when the cosine similarity is negative.

(j) (1 point) You realize that the most similar word of “christmas” is dissimilar to the most dissimilar word of “christmas”. Explain this occurrence using up to three sentences.

(k) (3 points) After getting some intuition on how to use your Word2Vec model, you want to investigate the emotions that are most related to Christmas in your corpus and that are most unrelated in your corpus.

- Use your Word2Vec model for that purpose and report the top three Christmas-related and -unrelated emotions by using the `most_similar` method of the model with “christmas” and “emotion” as positive/negative keywords accordingly to find the top three words with the most similar embeddings.
- Distinguish and give for each call of the `most_similar` method, which keywords are taken in as positive and which are taken in as negative keywords.
- Further, explain in three sentences how the `most_similar` method processes the keywords and what embedding you obtain based on your positive/negative keywords.

Hint: While you may find most similar words that are not emotions, they may still be very emotion-related words.

Question 3: Process Mining

You have received event data (`process_mining/event_log.xes`) of a financing organization which offers loans to customers. The overall process is that customers apply for a loan—this application is then either accepted or rejected. After a customer's loan application is accepted, the process of offering loans to customers begins. Every customer can be presented with multiple offers, but only one offer can be accepted.

The event log you will analyse is the process of (literally!) coming to terms with the customers of accepted loan applications—it begins after the loan application is accepted. In the event log, every entry describes the state change of an offer, i.e., every entry refers to:

1. the id of the loan application (case),
2. the state an offer was set to (activities), and
3. a timestamp.

Additionally, some information for the individual offers is included, such as the offer ID.

Use Python and *PM4Py*[3] to load the data and analyze the process, thereby, answering the following questions.

Event Log Exploration

(a) (1.5 points) You first turn your attention to the activities in the event log.

- Provide a plot showing the absolute frequency of activities. The x-axis should show every activity name and the y-axis should show every activity's absolute frequency. Make sure to label your axes and provide a title for the plot.
- How many events does the log contain?

Hint: No need to combine any activities in the event log, just leave the activities as they are.

(b) (1.5 points) Next, you want to look at the cases in the event log.

- How many cases are in the event log?
- What is the total number of offers made in the event log?
- Provide the mean number of offers per case.

Process Exploration

Next, you want to gain a basic understanding of the process described in the data.

(c) (1 point) You first look at the start and end activities of the process. Create a plot, showing how often every activity is a start and/or end activity of a case. Your plot should have all activity names on the x-axis and the absolute frequency of the activity being a start/end activity on the y-axis. Distinguish between start and end activities by color. Make sure to label your axes and provide a title for the plot.

Process Exploration: Full Model

To understand the process, you want to create a process model.

(d) (1 point) As learned in your IDS course, the Inductive Miner guarantees that the language of the model includes all traces in the event log. Do not actively set any parameters for the Inductive Miner, i.e., leave them at their default values and just pass the event log. Provide a Petri net model obtained from applying the Inductive Miner using the entire event log as input.

[3]`https://processintelligence.solutions/static/api/2.7.11/`

(e) (1.5 points) Consider the discovered model and try to understand the process. Provide one activity sequence that you think describes the process as it should be executed. This activity sequence should be valid according to the discovered model. Briefly explain in 1–2 sentences why you think this sequence describes the process as it is intended (using context information and the knowledge that the offer process only begins once a customer's loan application was accepted).

(f) (2.5 points) Look at the model again and find an activity sequence which is valid according to the model but neither makes sense logically nor occurs in the data. Briefly provide the following information:

- The activity sequence,
- A brief explanation (1–2 sentences) why this activity sequence does not make sense logically (using domain knowledge)
- A brief explanation (3–4 sentences) of how it can happen that a model discovered using the Inductive Miner can allow for an activity sequence that is not observed in the data. Explain this by using your selected non-logical activity sequence and your knowledge of the Inductive Miner. If it helps, you can introduce a small mock example event log to illustrate your explanation.

Hint: To create a sequence that you are sure of not being present in the data, a combination of the process model and your overview of start and end activities might be helpful.

Process Exploration: Variants

Now you have understood the general nature of the process, you want to take a look at the different variants present in the event log and the number of cases they represent.

(g) (1 point) Create a chart showing the percentage of cases that can be covered by the minimum number of variants. The x-axis should show the number of variants (ordered from left to right by their frequency) and the y-axis should depict the relative accumulated number of cases. Make sure to label your axes and provide a title for the plot.

Hint: To determine each variant's value on the y-axis you need to sort the variants by the number of cases they cover (descending) before determining the accumulated sum of the percentage of cases covered. We also recommend adding a count of zero cases for a non-existent 0th variant to make the plot more expressive.

(h) (1.5 points) Answer the following questions:

- What is the minimum number of variants you need to cover 85 % of cases?
- How high is the percentage of cases covered by the two most frequent variants?
- How many variants represent fewer than 10 cases?

Process Model

Your previous process model describes all behavior seen in the data; resulting in a very large and hard to read process model. Now you want to create a simpler model that only shows the most frequent behavior.

(i) (2.5 points) Provide a Petri net model obtained from applying the Inductive Miner using an event log only containing the traces of the **five most frequent variants**. Do not actively set any parameters for the Inductive Miner—leave them at their default values and just pass the filtered event log as input. How is this model's token-based replay fitness (`log_fitness`) on the *full event log*?

(j) (1 point) Create a histogram of the individual trace fitness values as reported by the conformance diagnostics (for token-based replay). Use the bins provided in the Jupyter notebook. Make sure to label your axes and provide a title for the plot.

(k) (2 points) The five most frequent variants cover approximately 75.25 % of cases and the Inductive Miner guarantees that the model fits for all of these cases. However, (1) the percentage of fitting traces is significantly higher than this, and (2) the fitness of the full event log is also a lot higher than just the percentage of fitting traces. Use your knowledge on the valid traces of discovered models and token-based replay to explain these two phenomena in 1–2 sentences each. You may refer to the plot created in the previous question.

Performance Analysis

Finally, you want to consider the mean throughput time (duration) of executing the different frequent variants.

(l) (2.5 points) Provide a bar chart showing the mean throughput time (in days) of the 35 most frequent variants and the number of cases covered by each variant. The x-axis should show the number of cases and the y-axis the mean throughput time. Each of the individual bars should represent a variant, where the width is the number of cases covered by the variant and the height is the variant's mean throughput time in days. Order the bars according to the number of cases covered by the variant (most frequent variant on the left). Make sure to label your axes and provide a title for the plot. What are the mean throughput times (in days) of the two most frequent variants?

(m) (1.5 points) Considering the mean throughput times per variant (only considering the 35 most frequent variants), you suspect the number of offers created per variant might have an impact on the throughput time. Create a suitable boxplot showing the mean throughput time (y-axis) per variant with regard to the number of offers made per variant (x-axis). Make sure to label your axes and provide a title for the plot. Does the plot show an indication on how the number of offers a customer is presented with affects the throughput time?

Question 4: Time Series Analysis

In this question, you get to explore some historical data about UFO sightings. The relevant dataset(s) are in the `time_series/` folder. The features/columns of the raw data `ufo-sightings.csv` are presented in the table below.

Feature	Description
`Date_time`	Date and time of the UFO sighting (YYYY-MM-DD HH:MM:SS)
`date_documented`	Date the sighting was documented
`Year`	Year of the sighting
`Month`	Month of the sighting
`Hour`	Hour of the sighting
`Season`	Season during which the sighting occurred
`Country_Code`	Abbreviation of the country name
`Country`	Full name of the country
`Region`	Region or state within the country
`Locale`	Specific locale or city of the sighting
`latitude`	Latitude coordinate of the sighting location
`longitude`	Longitude coordinate of the sighting location
`UFO_shape`	Reported shape of the UFO
`length_of_encounter_seconds`	Duration of the sighting in seconds
`Encounter_Duration`	Reported duration of the sighting (text format)
`Description`	Description of the sighting provided by the observer

While there are many possible analyzes to be made here, particularly with respect to location and encounter details such as the description, however, this task focuses purely on derived time series. You will explore how UFO sightings have varied over time and attempt to forecast them after some initial analysis and exploration.

(a) First, you need to get an overview over the data.

i. (1 point) Load the dataset `ufo-sightings.csv` and state the date range and number of the UFO sightings.

ii. (1 point) As you may have noticed, the sightings also have recorded positions. Good visualizations can be illuminating (just like UFOs in the sky), so we would like to create an animated geographic plot. Create a `scatter_geo` plot (*plotly*) with the `Year` (ordered) as animation frames in the notebook and include a screenshot/export of one of its frames. Briefly describe one trend you can observe over the temporal evolution of the sightings.

iii. (1 point) To explore further, create a scatter plot that visualizes the sighting occurrences per `Country` over time. Describe the most prominent trend you can observe.

(b) Suppose you want to forecast the number of sightings globally. Some preprocessing is necessary to massage the raw data into a collection of clean time series.

i. (1.5 points) Transform the dataset such that you get time series with columns (*time*, *count*) **for each individual country** that would be usable for a standard forecasting algorithm. Specifically, use a quarter-year aggregation. You may have to make some design decisions, or parameterize your approach. Include tables of the first three and last three rows of the time series for the **USA** and **AUS** (Australia). Additionally, insert a listing of your preprocessing Python function/Jupyter cell.

ii. (1 point) Create and include a time series plot for the USA using your preprocessed dataset. Describe it in 2–3 sentences.

(c) From now on, consider the two transformed datasets `ts-yearly.csv` and `ts-monthly-10yrs.csv`. (Only the top 5 countries (w.r.t. sighting count) are included.)

i. (2 points) Explore both time series of sighting counts and explain one of their major differences in 2–4 sentences. Include visualizations that support your argument.

ii. (1 point) Comment on each series' suitability for forecasting within 2–3 sentences. *Note: Take the forecasting horizon to be relative to the resolution of the time series, i.e., the goal is to forecast a number of periods, not a fixed duration.*

iii. (1 point) Can you explain why the yearly time series drops sharply in the final period (2014)?

(d) Next, you can finally dive into some time series-specific analyses. From now on, consider `ts-monthly-10yrs.csv`.

i. (1 point) Create and include a time series plot of the sighting counts of the country **AUS** (Australia). Include three moving averages: 3-MA, 5-MA and 7-MA.

ii. (3 points) Create and include correlograms (auto-correlation plots) for the counts in the **USA** after differencing $\{0, 1, 2, 3\}$ times. Which lag(s) do you consider significant?

iii. (3 points) To investigate further, create a seasonal plot (`plot_month`) of the once-differenced counts (i.e., count changes) for each country. Include the one for the **USA**, and describe, in 2–4 sentences, what you can infer.

iv. (2 points) Using STL (`statsmodels`), produce a trend-seasonality decomposition (STL/LOESS) for the counts in the **USA** after differencing $\{0, 1, 2, 3\}$ times. Use a period of 12. Are any of these differenced versions (almost-)stationary? Include a figure to support your argument.

(e) Lastly, we want to try some forecasting; even in the face of possibly violated assumptions. To avoid follow-up mistakes, use the prepared files `ts-train.csv` and `ts-test.csv` as training and testing range respectively. They are created from a pivoted version of the monthly time series. The test split covers the last 12 months.

i. (4 points) For each country, fit a univariate **NaiveForecaster** (*mean* and *last* strategy) and **StatsModelsARIMA** models. Evaluate all models on the test split with **RMSE** (root mean squared error), **MAE** (mean absolute error), and **MAPE** (mean absolute percentage error). For the ARIMA models, vary the order (p, d, q) until you find a model that beats both NaiveForecaster baselines in terms of RMSE. Present the results in a table, showing all metrics for the two naive forecasters and the best ARIMA model ($6 \cdot 3 = 12$ models).
Hint: Sensible ranges for the ARIMA parameters are $1 \leq p, q, \leq 12$ *and* $0 \leq d \leq 3$.

ii. (1 point) For each country, create a time series plot that shows both training and test data, as well as the forecasts of the different models (naive mean & last and best ARIMA).
Are there differences between countries? Choose two countries' plots to include in the report.

iii. (1.5 points) Are these good results? Briefly explain which characteristic(s) of the data complicate forecasting the most. Consider especially the differences between the countries.

Question 5: Distributed Data Processing

In this task, you are going to make use of the *MapReduce* (MR) paradigm to parallelize and distribute computations such that they could be scaled to *Big Data*. The data used in this task is from a user-based movie rating platform. There are two CSV files:

- `distributed_data_processing/movies.csv`: movieId, title, list of genres (separated by '|')
- `distributed_data_processing/ratings.csv`: userId, movieId, rating, timestamp, normalized rating (the header row is not present in the file itself to ease working with chunks of the file)

The first one contains information about a selection of movies and is small, whereas the second essentially represents a huge sparse matrix (tensor) of all submitted ratings. It could imaginably grow quite large on a popular platform.

We write the following formal symbols as value domains to use in the mathematical definitions of MR jobs.

- S: domain of strings
- $\mathbb{Q}$: domain for any computations on ratings (e.g., averages)
- $\mathbb{N}_0$: integers (e.g., indices and counts)
- $M \subseteq S$: set of movie ids
- $U \subseteq S$: set of user ids
- $R = \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$: star ratings from 0.5 to 5 as indicated to users on the website
- $R_N \subseteq \mathbb{Q}$: domain of normalized ratings (centered and scaled)
- T: domain of timestamps
- $\{\textsc{None}\}$: a singleton set, used when intending to map all values to a single key

The logical signature of the files, i.e., after splitting the columns and ignoring the key, are:

- `distributed_data_processing/movies.csv`: $(M \times S \times S)$
- `distributed_data_processing/ratings.csv`: $(\mathbb{N}_0 \times U \times M \times R \times T \times R_N)$

To make this task executable, we are using the Python package *mrjob*[4]. It allows us to write the logic in almost-pseudo code and is close to the mathematical formalization of the mapper and reducer as presented in the lecture and exercise. Two technical notes (also covered in the exercise session): The first mapper of a job actually receives a placeholder key None and an entire line of a text file as its value. It thus has to extract the logical structure via CSV parsing (simplified: splitting at commas) to conform to its logical signature. Refer to the provided templates `best_movies.py` and `haters.py`. Mappers further down the chain receive their parameters just like logically defined as key value pairs. "list-ification" via the Kleene operator * is implemented via iterators (generators) which means that mappers yield their resulting key value pairs, and reducers receive a key and iterator-of-values pair.

(a) (6 points) Define a 2-step MapReduce job that takes as input the ratings file and produces a list of the top 8 movies with an average rating greater or equal to 4.0 and at least 10 reviews. That means it produces pairs $(\textsc{None}, (m, avg\_rating, num\_ratings)) \in (\{\textsc{None}\} \times (M \times \mathbb{Q} \times \mathbb{N}_0))$ such that $avg\_rating \geq 4.0$ and $num\_ratings \geq 10$.
To not trivialize the task, *you may **not** perform all computations on a singular reducer* as that defeats the entire point.

[4] `https://mrjob.readthedocs.io/en/stable/`

1. First, write down the mathematical function signatures of your mappers and reducers.
   Hint: The first mapping function has the (logical) domain $K \times V = \mathbb{N}_0 \times (U \times M \times R \times T \times R_N)$.
2. Then, use the provided template `best_movies.py` to specify the logic of the mappers and reducers. The scaffolding in the notebook also allows you to test your idea. In addition to submitting the `.py` file, include the code as a listing in the report.
3. Provide a table of the resulting top movies output.

In practical applications of MapReduce, and distributed algorithms in general, it is highly relevant to reduce the communication overhead, particularly in the shuffling step. One optimization is the introduction of *combiners.* They are essentially pre-reducers that are ran on the output of the inidvidual mapper jobs. As such, they have the same signature as reducers. However, they only receive a subset of values for a particular key.

(b) (2 points) Consider the task of identifying particularly negative reviewers (potential "downvote" bots). We want to consider users who have left at least 50 below 2 star reviews ($\geq$ 50 reviews, $<$ 2 stars). Specifically, the output should be of the form $(\text{None}, u) \in \{\text{None}\} \times U$ where u is such a "movie hater".
Define a mapper and reducer by completing the template in `haters.py`. In addition to submitting the `.py` file, include the code as a listing in the report.
You should structure the job such that it would maximally benefit from the introduction of combiners.

(c) (2 points) Explain, in 2–5 sentences, how using combiners could improve the (theoretical) performance of this task (b). To make your argument, describe a best-case scenario for the impact of combiners for this type of data.