MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Mikoláš Fromm

Design of LLM prompts for iterative data exploration

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Tomáš Petříček, Ph.D.
Studijní program: Informatika

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V .... dne ....

Podpis autora

Poděkování.

Název práce: Design of LLM prompts for iterative data exploration

Autor: Mikoláš Fromm

Department: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Tomáš Petříček, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Abstrakt.

Klíčová slova: klíčové slovo, složitější fráze

Title: Design of LLM prompts for iterative data exploration

Author: Mikoláš Fromm

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Tomáš Petříček, Ph.D., Department of Distributed and Dependable Systems

Abstract: Abstract.

Keywords: key, words

Obsah

Introduction

Large language models (LLMs) are increasingly becoming part of a wide range of user applications to streamline user work or enhance the application environment. The currently most widespread family of conversational models, OpenAI's ChatGPT, demonstrates that their use significantly extends beyond the realm of meaningless conversations, and their generative capabilities are often better in limited domains compared to humans. [1], [2], [3]. However, it is not always straightforward and clear how to communicate with such a model, especially how to achieve a specific format of response or how to deliver the necessary context for understanding the user's assignment.

LLM prompting is the answer to the mentioned problem, which can influence how the LLM response should look or what procedure the LLM should use in generating responses. Therefore, an LLM *prompt* can be understood as a set of rules defining the response format, instructions for interpreting incoming messages, or as a description of the context in which the assistant is situated.

Although current LLMs are capable of generating relatively complex exploratory scripts, whether it be SQL queries, a short program in Python or R, which are very popular for working with data sets, it is very often difficult for users, especially in more complicated cases, to decide whether the script is valid considering original user query. Therefore, an *iterative data exploration* method could be proposed, emphasizing the way the data set is traversed and how information from the data set is obtained.

Following this method, the user builds the final script or query sequentially through repeated selections of all possible atomic operations that can be performed at the given point. Not only is the user acquainted with all available steps through this approach, but it is also assumed that the user is better able to decide whether an atomic operation they want to perform on the data set is appropriate.

Having such iterative exploration tool, especially in conjunction with LLM prompting, can be very useful for exploring data sets that may be either factually complicated in terms of size or semantics of the data or that require knowledge of a query language (such as SQL for relational databases) through which data can be explored.

This work aims to combine the aforementioned into a single solution, which will serve as a proposal for a tool for easier work with data sets. The thesis will be divided into two parts, implementation and experimental. In the implementation part, we will build a web application in C# programming language operating over a virtual representation of tabular data, which, using the LLM assistant ChatGPT, will suggest the best next step to obtain a fitting result of the user's initial query. The goal of this part is to build a functional prototype of this nature, serving as a representative of the entire class of such tools. In its creation, emphasis will be placed on a general solution that will potentially be applicable to a broader range of systems requiring structured user input of a sequential nature. In the experimental part, we will focus on the challenges of LLM prompting and, based on the experiences gained from implementation, we will further test selected approaches on how to conceive LLM prompting and how to maximize the assistant's success in finding the correct next step, where the prompting will be tested on different cases and combined with proposed procedures.

1 Background

In this chapter, we introduce sources that served as the main motivation for creating an interface with a custom assistant for iterative data exploration. We will highlight the key elements we consider important for the utilization in our own implementation of such an assistant and demonstrate in which parts of the project it is appropriate to use an LLM assistant to increase the success in finding the final query.

1.1 The Gamma

The Gamma is a data exploration environment specifically designed to make the power of programmatic data analysis accessible to non-experts, particularly targeting users such as data journalists. It achieves this through the iterative prompting approach, where users construct simple, transparent, scripts by selecting from a range of code completions rather than writing their own code from scratch. This design significantly lowers the barrier to entry for data exploration, allowing users to effortlessly navigate and analyze data from various sources, without requiring prior programming knowledge. By transforming the complex process of data exploration into a series of simple, guided selections, *The Gamma* not only simplifies data analysis but also encourages a deeper engagement with data. Its emphasis on simplicity and reproducibility aligns with the broader goal of informed decision-making in data journalism and beyond. As such, The Gamma represents a critical advancement in making sophisticated data analysis tools more accessible, fostering a more data-literate society where users can easily leverage open data to uncover insights, tell compelling stories, and support factual claims.

Although this approach technically facilitates data exploration and potentially aids in decision-making about which option to choose, it still holds that it cannot help the user if, at a given step, it is impossible to confidently determine which choice is correct. This may not only be due to a lack of knowledge of the defined operations but also the nature of the source data. Since LLMs are capable of efficiently analyzing large volumes of text and making micro-decisions over that text [4], they seem like a suitable candidate for integration into such a system.

1.1.1 Downsides

The method of displaying all available methods from a given point requires an internal definition of all possible operations generally applicable in the given context, regardless of the current position. However, for some query languages, such as SQL, it may be difficult to define all existing operations in the given language and, above all, transform them into a reasonable sequential form containing atomic elements. It is therefore quite understandable that such a system contains only a limited set of operations for which the data exploration is supported.

However, even if the system will contain only a limited set of defined operations, there are scenarios in which it will be difficult to generate all next options, since in some cases it is desirable for the menu to be created dynamically, according to the current data, which can be not only potentially unlimitedly large, but also time-consuming to obtain.

If we disregard the problem with the time-consuming nature of processing all the next options, its total size is a problem that remains present even in the case of using LLM, as their contextual window, in which communication between the user and the assistant is done, is limited.

1.1.2 Inspirations

In our own project, we primarily want to adopt the *dot* approach in constructing the final user query, which offers all subsequent available options at each step and, upon applying a given step, displays the current state of the queried data source. Especially the first mentioned feature is key for engaging an LLM assistant, as it clearly defines the available selection of all options that can be provided to the LLM assistant as input, based on which the assistant can decide which of the selected options most closely matches the query the user is searching for.

1.2 A prompt Pattern Catalogue

For successful integration of LLM prompted hints into a data exploration tool with the aim of helping the user, it is crucial to carefully select how to construct the prompt and ensure the highest success rate in processing the assistant's response. In the work *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT* [5], which contains a catalog of generically structured basic approaches to prompting with conversational bots, several approaches were mentioned that seem to be applicable in this work as well. Namely, these approaches include:

1. **Persona Pattern**: The assistant will be introduced in its role as an assistant, helping to select the best possible choice from the given selection with respect to the user's intent and previous choices. From this prompt pattern, we expect that the assistant will be better placed in the context of the application and will not generate machine-unprocessable and undesirable responses, such as suggestions of random source codes solving the user's query. The following example ilustrates, how an assistant could be introduced to a situation of helping user navigate through eurostat database.

   > *You are an assistant that is helping a user to navigate through the set of tables from eurostat database. You will be given a query and all available subsections. You should help the user to navigate to the correct subsection.*

2. **The Template Pattern**: Since the user will not interact with the LLM directly, but only indirectly through the proposed best response from the given option set, it is important to ensure that the LLM produces formatted output according to predetermined rules, based on which the generated output can be processed and evaluated. The example below ilustrates how to give LLM a specification of a template that will be used to communicate together.

> *You will be given a list of all available options, each associated with unique index. Answer only with the number of the subsection you think is the best fitting. If you don't know the answer, return -1."*

The other introduced patterns seem to be no less interesting for the purpose of obtaining guidance in selecting the right option from the list of all possible operations over the data set, yet they mostly require a human recipient, who is naturally capable of understanding even more complex and indirect structures of guidance. An example of such an approach could be the *Flipped Interaction Pattern* defined as follows:

I would like you to ask me questions to achieve X.
You should ask questions until this condition is met or to achieve this goal (alternatively, forever).
(Optional) ask me the questions one at a time, two at a time, etc.

Tabulka 1.1 Definition of *Flipped Interaction Pattern* [5, page 6]

Which could look in the context of data exploration as follows:

context	*User wants to find the average age of customers in a certain database table...*
User:	Ask me questions so that you can find what is the average age of all customers in our database system.
Assistant:	What is the name of the database table?
User:	...
Assistant:	What are the column names in the table?
User:	...
Assistant:	*Other related questions...*
User:	...

Assistant:	The query you are looking for might look like:
	`SELECT AVG(age) AS average_age`
	`FROM customers;`

Tabulka 1.2 An example use of *Flipped Interaction Pattern* for data exploration.

It is clear from this example that this approach is almost impossible to automate in the background of a data exploration tool, as it requires direct user responses and at the same time produces the final answer in any format with any additional comments.

1.3 Iterative Prompting Assistant

Consider the following scenario for this subsection: We are a user who wants to find the correct table in the Eurostat database [6]. This database has systematically structured content, where values are stored in tables that are thematically classified into structures reaching a maximum depth of more than 10 subcategories.

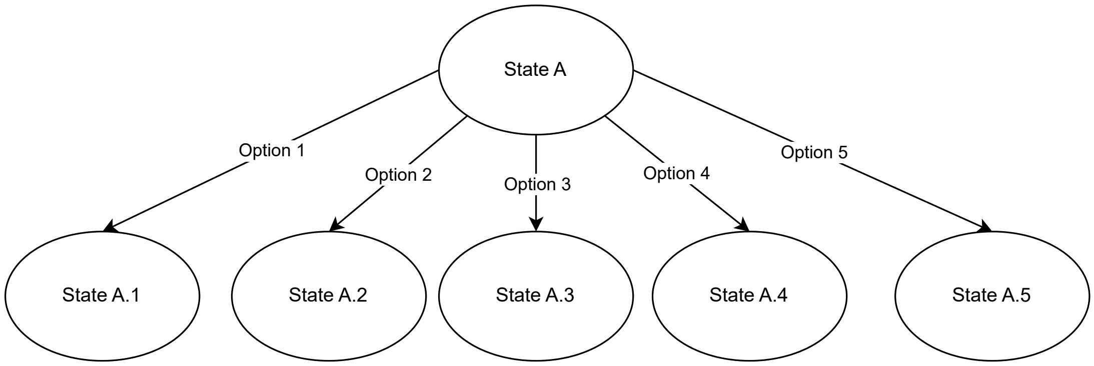
```
Eurostat
    - Detailed datasets
        ...
    - Selected datasets
            ...
        - Economy and finance
            - Government statistics
                ...
            ...
        ...
    ...
```

As a user seeing this database for the first time and having no experience with how the database is structured, any help with finding our table would be appreciated. Internally, we set our goal, which could be formed like this:

> *I am looking for a table with an overview of the year-on-year inflation rate development for the Federal Republic of Germany.*

Now, our only option to navigate through such a database and potentially find the desired table is to trust our instinct and repeatedly try to enter various categories as long as we dont find the correct table and we are heading in the right direction. If the subcategory names are familiar to us and we understand their meaning, we have a relatively high chance of successfully finding what we are looking for. However, if we know almost or absolutely nothing about the topic ourselves, then help from an LLM assistant can be a valuable contribution. Let's therefore define, how to represent such situation.

Any such environment that can be iteratively navigated can alternatively be understood as a decision tree, where a vertex represents the current state and an edge represents the operation of transitioning to the next state. Typically, there will be as many edges as there are operations the user can perform at the vertex (state). If we had to represent more complex operations than just the operation *open subcategory* with one argument *name of the subcategory*, we would break down such an operation into atomic units so that the above-mentioned formalism can again be applied.



Obrázek 1.1 Tree representation of the iterative approach.

In Figure 1.1, we represent a situation in which we are in state A, where 5 operations (1 - 5) can be applied, which will take us to states A.1 - A.5

If we were to model such an environment for communication with an LLM assistant, whose task would be to select the correct option, we must provide the assistant with the following input information:

1. User query,
2. already visited vertices,
3. edges leading from the current vertex.

The description of the current situation is thus expanded by the original user query and the already visited vertices, representing the *path* through the graph to the current position. These *metadata* are attached to the prompt due to saving contextual resources. Not only can it be assumed that it would be difficult for the LLM to search in a long history of prompts which operations have already been performed and to orient itself correctly, but such a history could, in more complex examples, exceed the maximum volume of context insertable into the LLM to get a response.

The prompt describing the tree illustration from Figure 1.1 could thus look as follows:

User query:	*The query given by the user at the beginning.*
Path done:	A
Next moves:	
	A.1
	A.2
	A.3
	A.4
	A.5

Tabulka 1.3 Prompt representation of the Image 1.1.

In the following chapter, we will use this procedure when implementing a sample solution of the iterative prompting assistant.

- tahle nemusí obsahovat žádný code-blocky
- stačí pouze popsat myšlenky

2 Iterative Prompting Assistant - C# Implementation

In this chapter, we will introduce an implementation solution for the iterative prompt assistant. As a demonstration scenario, we will simulate basic spreadsheet operations on tabular data. We chose this scenario because it illustrates well the use of an LLM assistant for obtaining next-step hints while also revealing the pitfalls associated with mapping the LLM assistant's textual response to calls of specific procedures in the program code.

In the program, all available operations will be statically defined, including their description, list of arguments, and their functions applicable to the data set. At each step of creating the final script by the user, we will prompt the LLM in parallel with a list of currently available options, along with the already performed history of operations, for advice on which of the following options is suitable to perform. This scenario is relatively straightforward as long as it involves choosing one of the listed options. However, some operations also require a user argument that cannot be selected from a pre-generated list, typically one word (e.g., filtering a data set by given value), and ensuring an acceptable response from the LLM for such operations is more complex.

We will show the approach chosen for communication with the LLM and introduce the pitfalls this approach brings.

I chose C# as the programming language for my web application due to my higher level of proficiency and familiarity with the language. This familiarity allowed for a smoother and more efficient development process, enabling me to focus on implementing features rather than learning a new language.

ASP.NET was selected as the development framework because it offers seamless integration with C#. Its compatibility with C# was crucial for the project, as it allowed for direct calls to C# functions from the web, aligning perfectly with the application's requirements.

As an extension for this back-end solution, I chose pure JavaScript for creating the user web interface, whose main purpose is to only process user input and forward it through the back-end API to the LLM assistant.

2.1 Description

The user will interact with the application via a web interface, which will be written in HTML and controlled using JavaScript. In this environment, the user will have the option to upload their file containing tabular data in CSV format (along with specifying the delimiter of the file), enter their desired intention with the data, iteratively create the final sequence of operations, and in real-time view the already (partially) transformed data. Operations will be entered using a *dot* approach, meaning that each argument will be separated by a dot and each dot will trigger the LLM suggester, which will advise the user on the next step. In the background of this web application, with each dot, the current input will be divided into individual operations, the already entered operations will be executed, and with the last operation, the LLM will be sent a prompt with the currently

- nedávej tam kód, ke kterému nemám co říct

– „teď vám jdu ukázat, jak se to tu řeší"

available and numbered upcoming options, followed by a request for selecting a suitable option in regard to the user's intent, which will also be included in the prompt. Upon receiving a response, an attempt will be made to process it (assigning it to the correct operation from the list) and if successful, that operation will be added to the collection of already entered operations, so that the web application can serve a preview of the current transformed data at any time. At every step, the user will be shown an offer of all available operations, along with the option highlighted by the LLM assistant. This way, the user will be able to conduct assisted iterative data exploration.

2.2 Architecture

When designing the architecture of the program, great emphasis was placed on the functional separation of the individual units of the program and on the possible implementation substitutability of the units participating in the creation of the final transformations.

The provider of the entire service is the WhisperService2.2 class, which internally holds a collection of all the specified operations on the data set and its own reference to IQueryAgent 2.2, which manages prompting with any given conversational bot, internally holds a current copy of the transformed data in order to be able to create an available offer for the creation of future prompts, and a reference to ICommunicationAgent 2.2, which manages specific format of a communication with the selected LLM agent.

Controller

The application controller, which is tasked with receiving queries posed from the web interface, performing basic input data control, and calling the appropriate methods of the WhisperService class 2.2. The controller contains the following endpoints providing:

- *api/whisper/upload-csv*: Uploading tabular data.
- *api/whisper/upload-user-input*: Uploading user query.
- *api/whisper/process*: Processing current operations.
- *api/whisper/get-current*: Getting the current view of the data.

Together with forwarding requests to the service class, it also handles simple validity checks of the input files and of the user query.

WhisperService

The intermediary for queries captured from the application controller 2.2. Divides the received user input into single operations and passes them to the query agent. It internally stores already processed operations to be able to provide a view into the current state of the data. Also, upon request, it creates a view of the current data set after applying all captured operations.

– víc provést programem

IQueryAgent

The main part of the entire program, ensuring the creation of prompts with respect to the current situation and user input. It sequentially goes through the individual operation names from the input, with which, until there are some left, creates a ”history”of already performed operations for the LLM assistant, to subsequently create, according to the current state, a new prompt asking for the best next move.

ICommunicationAgent

The final link for communication with the LLM agent, covering work with the public API models of OpenAI. Two types of messages are used for communication with the LLM agent:

1. System message: This is the part of the prompt that is present in every user message in the conversation. It is therefore primarily suitable for familiarization with the context and understanding of the role. In our case, it is used for the *Persona pattern* mentioned in section 1.2.

2. User message: This is already the standard part of the prompt, carrying the main information of the entire message. It does not adhere to any text formatting, since it is possible to communicate in any text format with LLMs. In our case, the *Template pattern* from section 1.2 was used for this type of messages.

Interfaces and Datatypes

Since one of the goals of the work is to create a tool that can simulate work with any data set, a custom structure, compatible with any defined transformation, was created to represent the data set.

Primarily, it is defined with the following types:

Program 1 Definitions of classes representing dataset.

```
public enum FieldDataType
{
    Bool,
    String,
    Number,
    Date
};

public class Header
{
    public FieldDataType Type { get; set; }
    public string Name { get; set; }
    public int Index { get; set; } = 0;
}

public class Cell : IComparable<Cell>
{
    public string Content { get; set; }
    public int Index { get; set; }

    public int CompareTo(Cell? other) {...}
    public int CompareToTypeDependent(FieldDataType type,
        Cell? other) {...}
}

public class EmptyField
{
    public Header Header { get; set; }
}

public class Field : EmptyField
{
    public List<Cell> Data { get; set; } = new();
}

public class DataSet
{
    public List<Field> Fields { get; set; } = new();
}

public class QueryViewModel
{
    public int BotSuggestionIndex {...}
    public string BotSuggestion {...}
    public IEnumerable<ITransformation> Transformations {...}
    public IEnumerable<string> NextMoves {...};
    public void AddTransformation(ITransformation transf) {...}
    public string AddBotSuggestion(string suggestion,
        bool userInputExpected = false) {...}
}
```

Program 2 Definitions of main interfaces.

```
public interface IQueryAgent
{
    public void AddUserQuery(string userQuery = null);
    public void StartNewQueryAttempt();
    public QueryViewModel PerformQuerying(IList<string> queryItems,
        IList<Field> fields);
}

public interface ICommunicationAgent
{
    public bool Verbose { get; }
    public CommunicationAgentMode Mode { get; }
    public void FlushCurrentChat();
    public void AddUserQuery(string userQuery = null);
    public string InsertSystemMessage(string message);
    public string InsertUserMessage(string message);
    public string ErrorMessage(string message);
    public string GetResponse(string querySoFar = null,
        string nextMove = null, int nextMoveIndex = -1,
        bool isUserInputExpected = false);
    public void ShowConversationHistory();
    public void Indent();
    public IEnumerable<string> CreateNextQuestion(
        string question,
        IEnumerable<string> possibleChoices = null);
}

public interface ITransformation
{
    public TransformationType Type { get; }
    public bool HasArguments { get; }
    public bool HasFollowingHumanArguments { get; }
    public int TotalStepsNeeded { get; }
    public List<Field> PerformTransformation(
        List<Field> input_list);
    public List<EmptyField> Preprocess(
        List<EmptyField> list);
    public string GetTransformationName();
    public string GetNextMovesInstructions();
    public IEnumerable<string> GetNextMoves(
        IEnumerable<EmptyField> fields);
    public string GetArgumentsInstructions();
    public IEnumerable<string> GetArguments();
    public string GetArgumentAt(int index);
    public string GetFollowingHumanArgumentsInstructions();
}
```

As a result, *IQueryAgent* works with the *ITransformation* interface and the implementation is identical (and only one) for any transformation that is available in the program. For practical reasons, there is also a class *EmptyField* defined, which serves as a type representative of a field and carries only the name and type of values contained in the given field. This is required since the type-information can affect the selection of available subsequent operations in the user selection, and it is therefore necessary to maintain (and possibly change) the type of fields even during the iterative construction of the user query.

2.3 Communication with LLM

In this section, we would like to describe the format of conversation that is conducted between the program and the LLM agent for the purpose of obtaining guidance for the next step. During the design of the LLM prompts was discovered that the way we communicate with the agent radically affects the validity and correctness of the answers and we therefore devote a whole chapter 3 to this topic in our work. Here, we will only describe the approach that was the motivation for chapter 3 and is part of the program implementation.

The form of the prompt used in the application is described in the following example (the dataset contains journeys and consumption of traction energy by railway carriers in Germany):

```
"User initial input is: Group data by carrier 'CDC'
and show count of distinct values in each column.

The query built so far: GroupBy.Carrier

---> Choose one of the following Aggregations
you want to apply to the grouped dataset:
> [0] Sum
> [1] Avg
> [2] Concat
> [3] CountDistinct
> [4] CountAll

Answer the appropriate number!"
```

1. With each subsequent step of an iterative built of the final query, the LLM agent is informed about the original user request,
2. the previously selected operations are listed to the agent too,
3. then, an enumerated collection of further options is presented, and
4. finally, the agent is reminded to respond only with a number from the collection.

During the first usage, we discovered that the absence of the original user query in the option offer (item 1) may cause the agent to consider the query as final, although especially in queries combining multiple operations at once, a query was created for only part of the query. Therefore, the user query is added to each prompt.

For similar reasons, with each prompt, the agent is reminded of the already created query (item 2) even though the agent has these data available in the chat history, which forms the context of the prompt and which is iteratively created with the entered operations. However, it can be assumed that the agent places greater emphasis on the most recently added messages and especially with longer queries may neglect already performed operations.

When designing the communication with an agent, we assumed that with each prompt, the agent would only be presented with a list of **names** of individual operations and the agent would respond only with the name of the selected operation. However, when applying this approach, the agent's response often contained not only the name of the operation but also a lot of confusing text:

```
"From the list above, I would choose 'CountDistinct'"
```

Instead of the response looking like this:

```
"CountDistinct"
```

Even though the model parameters were set to the maximum extent possible so that the response would be as straightforward as possible and contain a minimum of ancillary text. Therefore, all options are uniquely numbered (item 3) and it is expected from the agent to respond, assuming the correct answer is *[3] CountDistinct*, with:

```
"3"
```

However, since this approach was still susceptible to responses of a similar character;

```
"Let's start with GroupBy carrier 'CDC'
and show count of distinct values.
So the next transformation would be [3] GroupBy."
```

an imperative sentence was added to the end of the prompt (item 4) specifying what type of response we currently expect from the agent.

This design of prompts proved to be functional in the application's use and the agent responds in a valid format in most cases. It thus appears that the more restrictions about the input the agent receives, the less prone it is to deviate from the given format.

2.4 Showcase

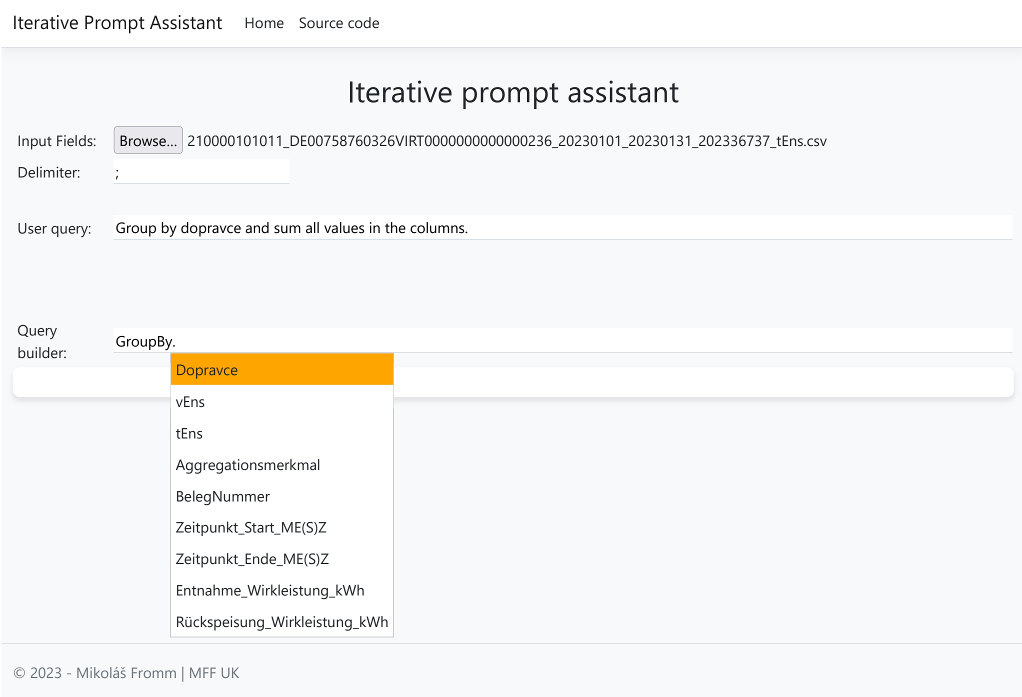
In this section, we demonstrate the final form of the web application with several screenshots.

After uploading a dataset in CSV format and specifying the delimiter used in the file, the user can enter their request. The request is formulated using a simple sentence, which is ended with a period. The presence of the period simultaneously triggers the first prompt to the LLM agent, followed by displaying available options. The preferred option by the LLM agent is always placed at the first position and is orange-highlighted. The rest of the options is in the original order.

If the construction of the query is at the beginning or if the construction of one of the transformations has just been completed, the user is offered a list of all available transformations. On the other hand, if a specific transformation has already been specified but is not yet fully constructed, all possible following arguments are displayed in the offer.

Some transformations have fixed additional arguments (such as *OrderBy. <header-name>. <ascending/descending>*) for which the agent is always generated a complete numbered list of available options, but some (case of *FilterBy. <header-name>. <equality>. <any-value>*) transformations allow entering any argument. In such cases, the LLM agent is prompted using a different imperative sentence at the end of the prompt (item 4) with a helping instruction to enter the textual argument that is most likely considered correct.

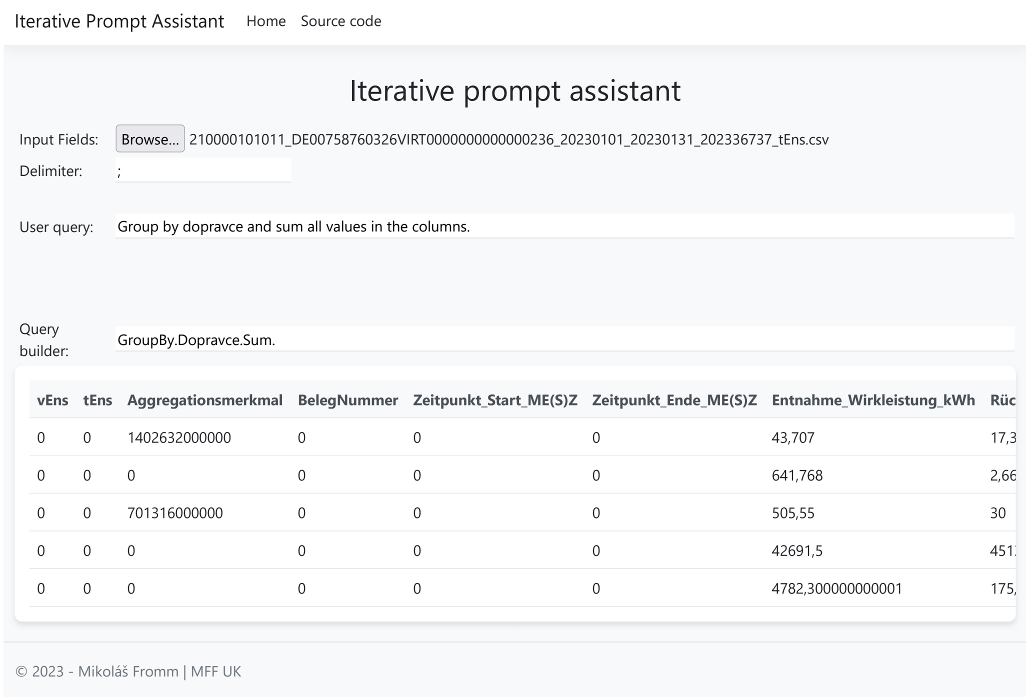
In the example 2.1, we can see that in the case of arguments requiring the selection of a specific column name, the offer is dynamically generated according to the current structure of the dataset.



Obrázek 2.1 Dynamically created dropdown menu with highlighted choice from LLM agent.

At the moment the user successfully creates at least one complete transformation, the current form of the dataset is displayed in the preview. The user may continue to build the final query or decide to finish building the query. If they decide to continue, each subsequent transformation will then be applied to the current data from the preview, thus the application implicitly supports the layering of a larger number of operations in sequence.

In the example 2.2 below, we can see the final form of one completed transformation and a preview of the current state of the dataset.



Obrázek 2.2 Preview panel with transformed dataset.

Finally, an example from testing the application shows that even if we describe in detail to the agent the format we expect from him when processing responses, the validity of his responses is not guaranteed.

Input prompt:

```
User initial input is: Show only rows with dates later than 10/01/2023
    and then filter by carrier = CDC.
The query built so far: FilterBy.StartDate.GreaterThan.
    10/01/2023.FilterBy.Carrier.Equals
Write down the right side of the relation.
```

Agent's response:

```
I'm sorry, I made a mistake earlier.
The last transformation you chose was FilterBy on the field "Carrier".
Therefore, you should choose a value for the "Carrier" field,
not the "GreaterThan" relation.

Here are the possible values for the "Carrier" field:

> [0] CEZ
> [1] EON
> [2] PRE

Please choose one of these values.
```

In this case, the agent was prompted to fill in the right side of the *Equals* relation, thus any response was understood as an argument, yet it is obvious that such a response is incorrect. The agent not only adds apology messages that his procedure was wrong but also defines new options in the format we defined, completely unrelated to the dataset.

In the following chapter 3, we will therefore verify whether it is possible to at least partially prevent problems with incorrect responses and how frequent is the error of these conversational bots in a similar application environment.

3 Constructing better prompts

In this section, we will focus on the issue of constructing an appropriate prompt. Creating the Iterative Prompt Assistant from chapter 2 has proven that choosing the right strategy for designing the final form of prompt communication is complicated. Therefore, in this section, we will experimentally verify several hypotheses related to the creation of LLM prompts, which can help in designing other systems that depend on prompting with LLM.

3.1 Motivation

The main motivation for experimenting with the form of LLM prompting is to maximize the success of the final prompt form. However, it is very difficult to define what „success“ of a prompt means in a general case, so we will focus on the application environment from chapter 2. Similar to this application, we will iteratively build the final query using LLM prompting, but unlike the application described above, LLM will interact only with a testing framework that will assess the correctness of the next proposed step.

The goal of the LLM agent will be to guide a pseudo-user through a selected data structure based on a request given in a natural language and help to choose the next appropriate step to achieve the goal, which is to find a correct path to the requested object in the underlying data structure. This form of the assignment was particularly designed to suit our testing needs to determine all correct solutions in advance and decide whether the LLM agent's proposal is correct or not afterwards.

Secondarily, we will be interested in whether the success is influenced by the language in which the query or data structure is described and whether the proposed improvements are invariant to the change of the LLM.

3.2 Framework description

The framework represents a methodology for conducting automated tests on LLM prompt effectiveness. It is specifically designed to evaluate how different prompting strategies influence the performance of LLMs, OpenAI's GPTs exactly, in navigating structured data sources like databases or file systems. The system incorporates a setup that enables the simulation of real-world data querying tasks across multiple data domains and languages, testing the adaptability and efficiency of LLM prompts in guiding users to their desired information.

3.2.1 Key Components and Methodology

The framework is structured into several core components, each responsible for a distinct aspect of the testing environment:

Data Source Simulation

Simulates three types of structured data sources: A copy of a content structure of the Eurostat database [6] (obtained on 01/02/2024), a snapshot of a file system containing personal user data (such as photos, videos, documents etc.) and a snapshot of a linux server file system.

Each data source is represented as a navigable tree structure with nodes and leaves, where nodes represent directories or categories, and leaves represent files or final data entries.

The Eurostat [6] offers its content table in three languages: English, French and German. The personal file system has its directory and file names in Czech and the linux server file system is in English.

Prompt Engineering

Implements a prompt engineering system capable of generating prompting strategies to interact with the LLM. These strategies vary from simple step-by-step navigation prompts to more specific look-ahead prompts that provide an overview of potential navigation paths. Alternative prompting method such as keyword generation and matching, where the LLM is tasked with identifying relevant subsections based on generated keywords, is also implemented.

Test Definition and Execution

A testing mechanism to systematically execute predefined navigation tasks within each simulated data source, using a list of correct navigation paths as benchmarks. The performance of each prompting strategy is evaluated based on metrics such as the number of steps taken, the presence of incorrect moves, and the overall success in reaching the correct destination. Extended testing configurations allow for penalization based on error locations.

Evaluation and Vizualization

Lastly the framework includes tools for statistical analysis and visualization of test results, allowing to easily interpret the obtained results and generate plots ilustrating the test results.

3.3 Hypotheses

To experiment with increasing the „success“ of LLM prompt responses, we have set the following hypotheses, which we will verify in this chapter:

3.3.1 Language Invariance

The success of selecting the correct answer is not dependent on the specific language of the query or dataset.

Since the testing framework will primarily contain English data sets and user queries in English, we first want to verify whether we would achieve the same results when testing in another language. For these purposes, we will use the Eurostat database [6], which offers its „list of content“ in three languages (other languages are only machine-translated); English, French, and German. Tests will be conducted over all combinations of data sets in English and German and user queries in English and German. Thus, a total of 4 sets of tests will be conducted, which will subsequently be compared.

3.3.2 Expanding Context Increases Success

The success of selecting the correct answer can be increased by enlarging the contextual information in the LLM prompt.

When designing the form of LLM prompts, it has been shown to be effective in small test examples to offer the LLM agent not only all currently available next steps but also to include more information about the given step (for example, all subsequent steps related to the given step). We will therefore want to verify whether the „look-ahead-X“ method, which attaches all further subsequent available steps up to a distance of $(X - 1)$ from the currently named step, increases the success of the LLM agent.

3.3.3 Reason for Higher Success

The increase in success for hypothesis 3.3.2 is not just a result of a more accurate answer when deciding at the end of the search path.

Consider now, for example, the „look-ahead-3“ method and a test example that contains only 3 steps to find the right target. Then the initial prompt that the LLM agent receives as the first one will already contain the textual naming of the goal. It can therefore be assumed that in each such case, where the distance of the „look-ahead-X“ is the same or shorter than the number of remaining steps to reach the goal, the measured success will be artificially higher. Therefore, we introduce a penalty score, which will place more emphasis on the agent's first steps rather than on later steps.

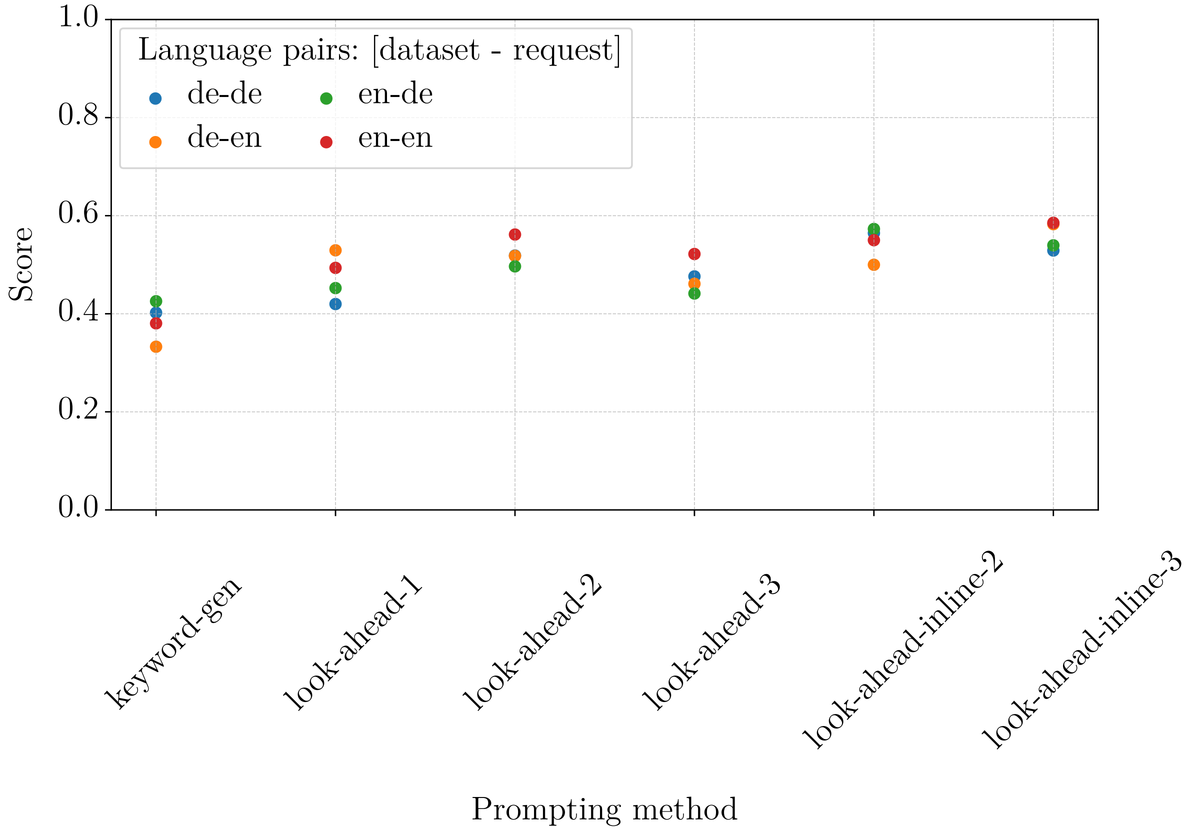
Finally, we will be interested in whether there is a significant difference between the score of the penalized and non-penalized agent, that is, alternatively whether the potential increase in success with „look-ahead“ is caused only by an artificially more accurate answer in the last steps of the decision-making process, or whether this approach generally helps to find the right step when searching for a target.

3.3.4 Model Invariance

Methods increasing success are invariant to the LLM used.
For the trends measured when testing hypothesis 3.3.2, we would like to verify that it is a general trend that can also be applied to another LLM and is not just a specific feature of one particular tested model. Therefore, we will perform all series of tests on two LLMs simultaneously (GPT-3.5 and GPT-4) and verify that the trend in success relative to the size of the context is the same for both models.

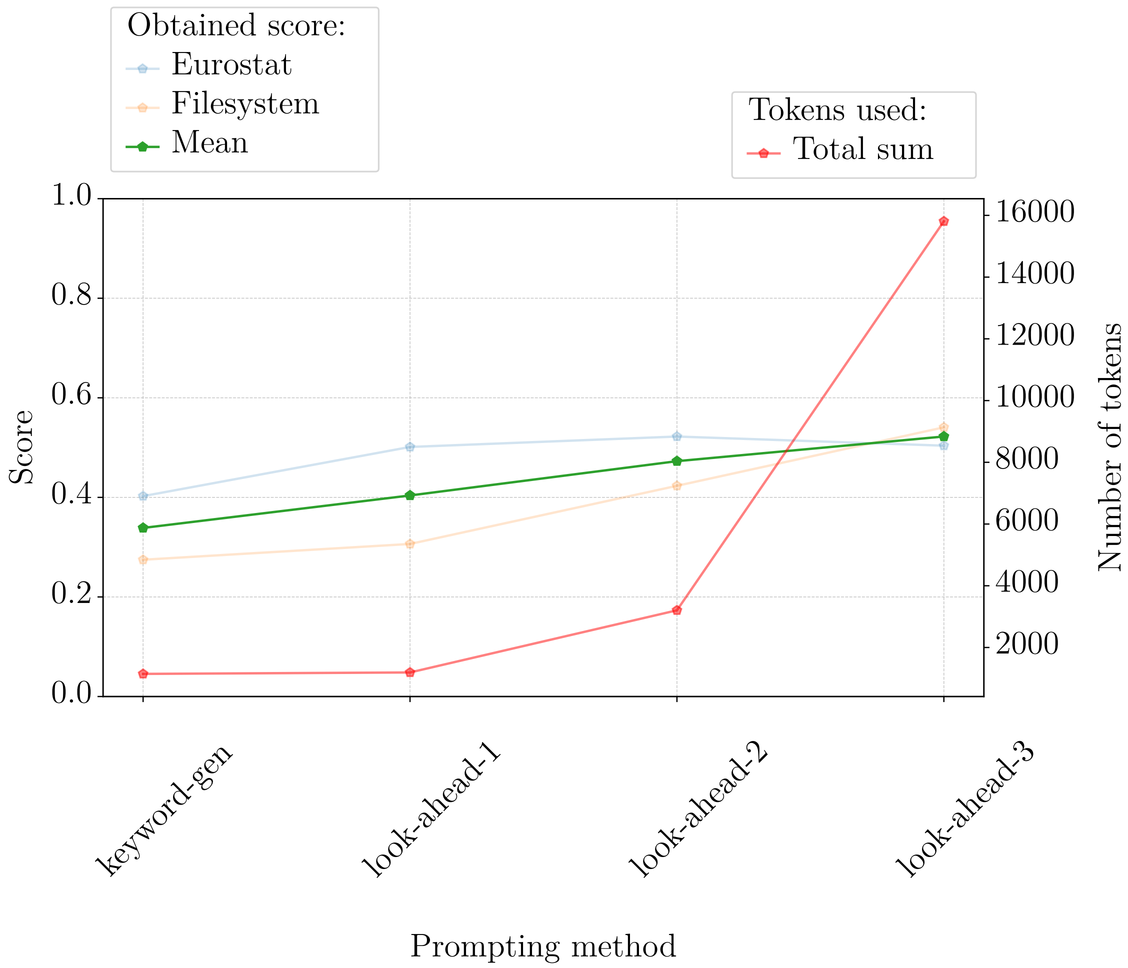
3.4 Metodology

3.5 Results

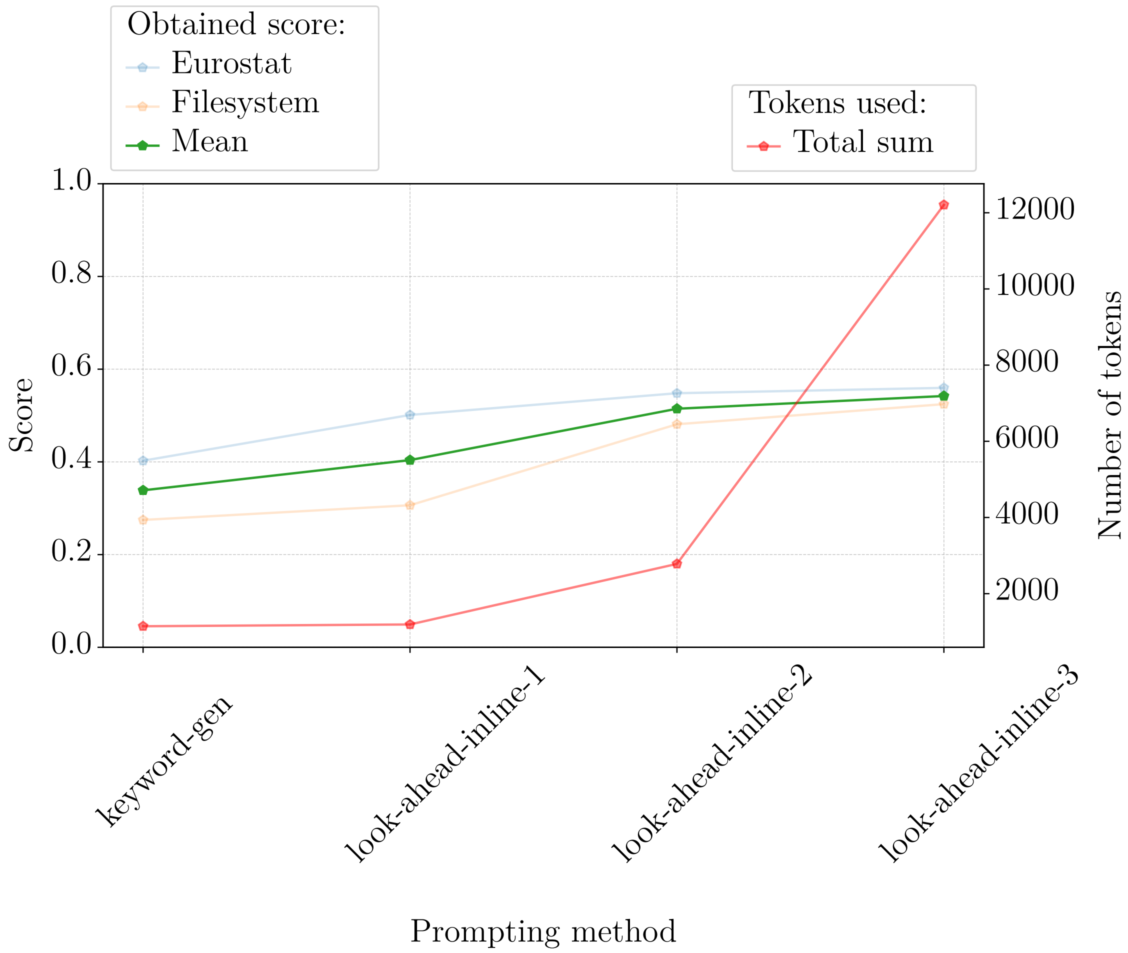


Obrázek 3.1 Mean score of all language pairs on different prompting methods.

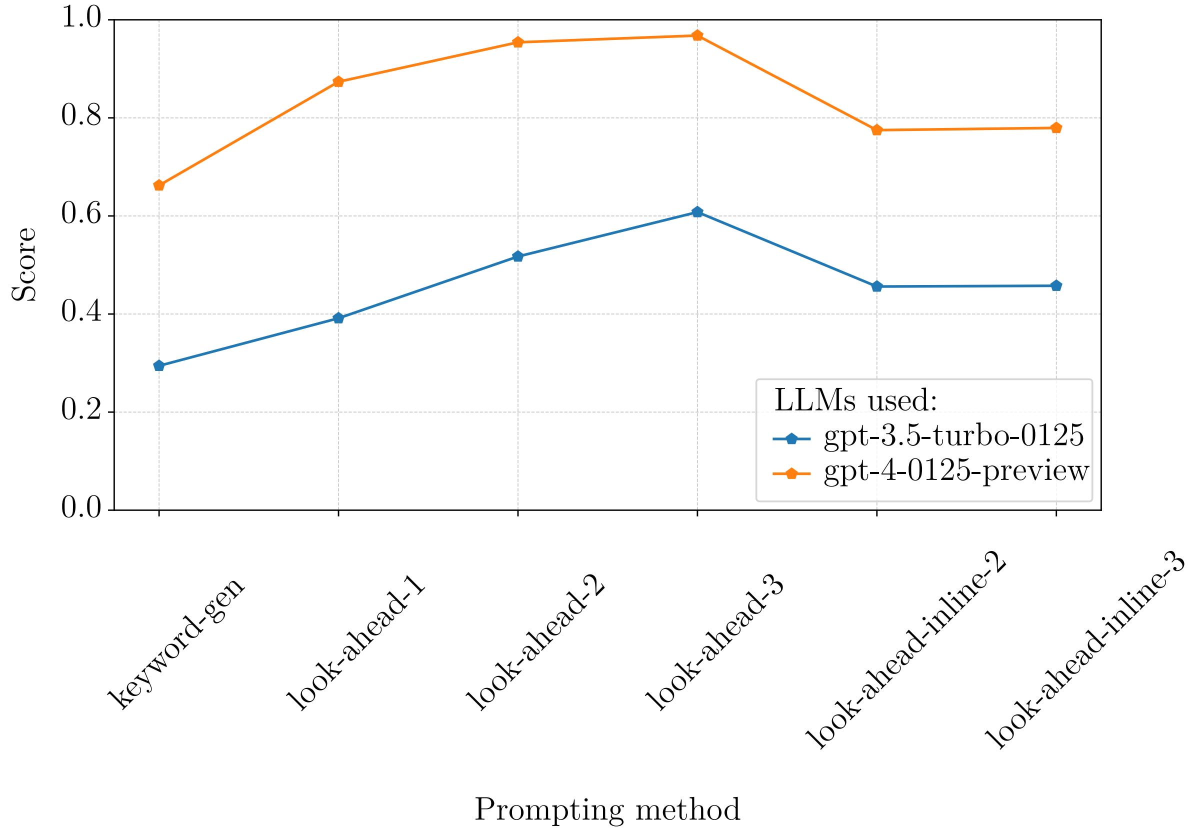
oddělení dvou y-os



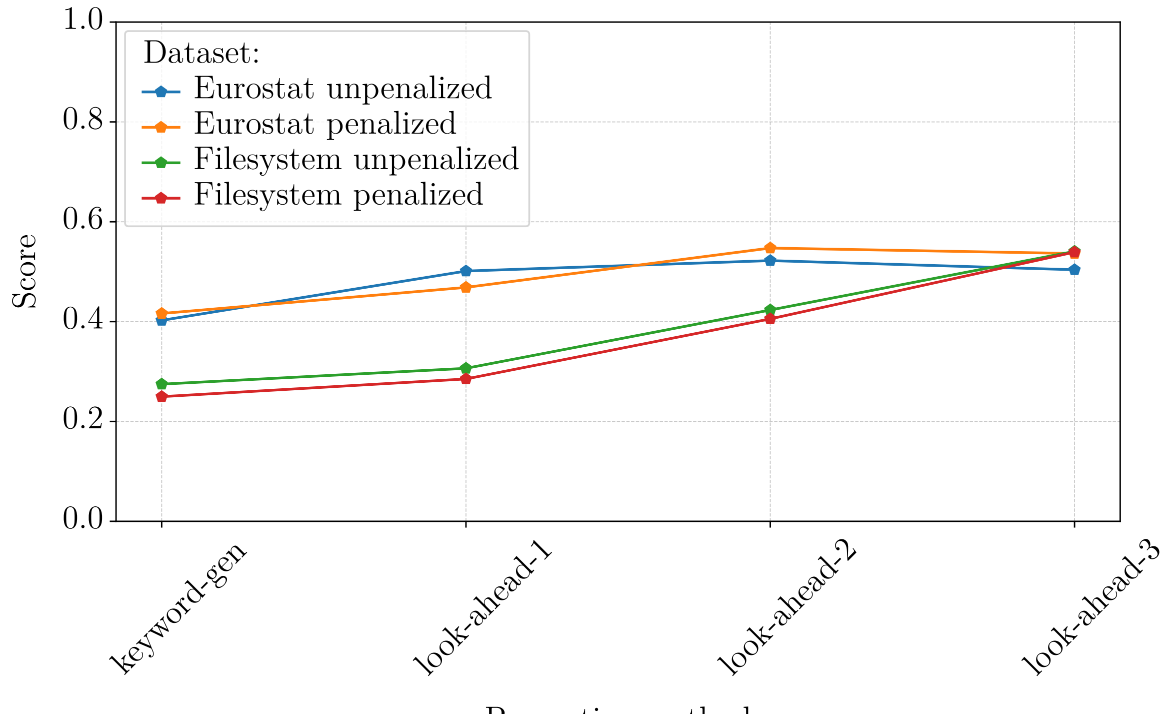
Obrázek 3.2 The impact of context size on the final score. Prompt content being separated by tabs, remaining human-friendly.



Obrázek 3.3 The impact of context size on the final score. Prompt content being compressed into one line, human-unfriendly.



Obrázek 3.4 Comparison of the two LLMs, GPT-4 being an evolution of the GPT-3 while remaining the same score trends.



Obrázek 3.5 Overview of score changes when penalizing earlirer mistakes heavier.

4 Conlcusion

5 Nápověda k sazbě

5.1 Úprava práce

Vlastní text práce je uspořádaný hierarchicky do kapitol a podkapitol, každá kapitola začíná na nové straně. Text je zarovnán do bloku. Nový odstavec se obvykle odděluje malou vertikální mezerou a odsazením prvního řádku. Grafická úprava má být v celém textu jednotná.

Práce se tiskne na bílý papír formátu A4. Okraje musí ponechat dost místa na vazbu: doporučen je horní, dolní a pravý okraj 25 mm, levý okraj 40 mm. Číslují se všechny strany kromě obálky a informačních stran na začátku práce; první číslovaná strana bývá obvykle ta s obsahem.

Písmo se doporučuje dvanáctibodové (12 pt) se standardní vzdáleností mezi řádky (pokud píšete ve Wordu nebo podobném programu, odpovídá tomu řádkování 1,5; v TEXu není potřeba nic přepínat).

Primárně je doporučován jednostranný tisk (příliš tenkou práci lze obtížně svázat). Delší práce je lepší tisknout oboustranně a přizpůsobit tomu velikosti okrajů: 40 mm má vždy *vnitřní* okraj. Rub titulního listu zůstává nepotištěný.

Zkratky použité v textu musí být vysvětleny vždy u prvního výskytu zkratky (v závorce nebo v poznámce pod čarou, jde-li o složitější vysvětlení pojmu či zkratky). Pokud je zkratek více, připojuje se seznam použitých zkratek, včetně jejich vysvětlení a/nebo odkazů na definici.

Delší převzatý text jiného autora je nutné vymezit uvozovkami nebo jinak vyznačit a řádně citovat.

5.2 Jednoduché příklady

K různým účelům se hodí různé typy písma. Pro běžný text používáme vzpřímené patkové písmo. Chceme-li nějaký pojem zvýraznit (třeba v okamžiku definice), používáme obvykle *kurzívu* nebo **tučné písmo.** Text matematických vět se obvykle tiskne pro zdůraznění *skloněným (slanted)* písmem; není-li k dispozici, může být zastoupeno *kurzívou.* Text, který je chápan doslova (například ukázky programů) píšeme `psacím strojem`. Důležité je být ve volbě písma konzistentní napříč celou prací.

Čísla v českém textu obvykle sázíme v matematickém režimu s desetinnou čárkou: $\pi \doteq 3{,}141\,592\,653\,589$. V matematických textech je často lepší používat desetinnou tečku (pro lepší odlišení od čárky v roli oddělovače). Nestřídejte však obojí. Numerické výsledky se uvádějí s přiměřeným počtem desetinných míst.

Mezi číslo a jednotku patří úzká mezera: šířka stránky A4 činí 210 mm, což si pamatuje pouze 5 % autorů. Pokud ale údaj slouží jako přívlastek, mezeru vynecháváme: 25mm okraj, 95% interval spolehlivosti.

Rozlišujeme různé druhy pomlček: červeno-černý (krátká pomlčka), strana 16–22 (střední), $45 - 44$ (matematické minus), a toto je — jak se asi dalo čekat — vložená věta ohraničená dlouhými pomlčkami.

V českém textu se používají „české“ uvozovky, nikoliv “anglické”.

Na některých místech je potřeba zabránit lámání řádku (v~TEXu značíme vlnovkou): u~předložek (neslabičnych, nebo obecně jednopísmenných), ahojda vrchol v, před k~kroky, a~proto, … obecně kdekoliv, kde by při rozlomení čtenář „škobrtnul“.

5.3 Matematické vzorce a výrazy

Proměnné sázíme kurzívou (to TEX v matematickém módu dělá sám, ale nezapomínejte na to v okolním textu a také si matematický mód zapněte). Názvy funkcí sázíme vzpřímeně. Tedy například: $\mathrm{var}(X) = \mathsf{E}\, X^2 - \left(\mathsf{E}\, X\right)^2$.

Zlomky uvnitř odstavce (třeba $\frac{5}{7}$ nebo $\frac{x+y}{2}$) mohou být příliš stísněné, takže je lepší sázet jednoduché zlomky s lomítkem: $5/7$, $(x+y)/2$.

Nechť

$$\mathbb{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{pmatrix}.$$

Povšimněme si tečky za maticí. Byť je matematický text vysázen ve specifickém prostředí, stále je gramaticky součástí věty a tudíž je zapotřebí neopomenout patřičná interpunkční znaménka. Výrazy, na které chceme později odkazovat, je vhodné očíslovat:

$$\mathbb{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{pmatrix}. \tag{5.1}$$

Výraz (5.1) definuje matici $\mathbb{X}$. Pro lepší čitelnost a přehlednost textu je vhodné číslovat pouze ty výrazy, na které se autor někde v další části textu odkazuje. To jest, nečíslujte automaticky všechny výrazy vysázené některým z matematických prostředí.

Zarovnání vzorců do několika sloupečků:

$$\begin{aligned} S(t) &= \mathsf{P}(T > t), & t &> 0 & &\text{(zprava spojitá)},\\ F(t) &= \mathsf{P}(T \leq t), & t &> 0 & &\text{(zprava spojitá)}. \end{aligned}$$

Dva vzorce se spojovníkem:

$$\left.\begin{aligned} S(t) &= \mathsf{P}(T > t)\\[1ex] F(t) &= \mathsf{P}(T \leq t) \end{aligned}\right\} \quad t > 0 \qquad \text{(zprava spojité)}. \tag{5.2}$$

Dva centrované nečíslované vzorce:

$$\boldsymbol{Y} = \mathbb{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

$$\mathbb{X} = \begin{pmatrix} 1 & \boldsymbol{x}_1^\top \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_n^\top \end{pmatrix}.$$

Dva centrované číslované vzorce:

$$\boldsymbol{Y} = \mathbb{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{5.3}$$

$$\mathbb{X} = \begin{pmatrix} 1 & \boldsymbol{x}_1^\top \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_n^\top \end{pmatrix}. \tag{5.4}$$

Definice rozdělená na dva případy:

$$P_{r-j} = \begin{cases} 0, & \text{je-li } r-j \text{ liché,} \\ r!\,(-1)^{(r-j)/2}, & \text{je-li } r-j \text{ sudé.} \end{cases}$$

Všimněte si použití interpunkce v této konstrukci. Čárky a tečky se dávají na místa, kam podle jazykových pravidel patří.

$$\begin{aligned} x &= y_1 - y_2 + y_3 - y_5 + y_8 - \dots = && \text{z (5.3)} \\ &= y' \circ y^* = && \text{podle (5.4)} \\ &= y(0) y' && \text{z Axiomu 1.} \end{aligned} \tag{5.5}$$

Dva zarovnané vzorce nečíslované:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^{n} f_i(y_i;\, \boldsymbol{\theta}), \\ \ell(\boldsymbol{\theta}) &= \log\Bigl\{L(\boldsymbol{\theta})\Bigr\} = \sum_{i=1}^{n} \log\Bigl\{f_i(y_i;\, \boldsymbol{\theta})\Bigr\}. \end{aligned}$$

Dva zarovnané vzorce, první číslovaný:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^{n} f_i(y_i;\, \boldsymbol{\theta}), \qquad (5.6) \\ \ell(\boldsymbol{\theta}) &= \log\Bigl\{L(\boldsymbol{\theta})\Bigr\} = \sum_{i=1}^{n} \log\Bigl\{f_i(y_i;\, \boldsymbol{\theta})\Bigr\}. \end{aligned}$$

Vzorec na dva řádky, první řádek zarovnaný vlevo, druhý vpravo, nečíslovaný:

$$\begin{multline*} \ell(\mu, \sigma^2) = \log\Bigl\{L(\mu, \sigma^2)\Bigr\} = \sum_{i=1}^{n} \log\Bigl\{f_i(y_i;\, \mu, \sigma^2)\Bigr\} = \\ = -\,\frac{n}{2}\,\log(2\pi\sigma^2) \;-\; \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu)^2. \end{multline*}$$

Vzorec na dva řádky, zarovnaný na =, číslovaný uprostřed:

$$\begin{split} \ell(\mu, \sigma^2) &= \log\Bigl\{L(\mu, \sigma^2)\Bigr\} = \sum_{i=1}^{n} \log\Bigl\{f(y_i;\, \mu, \sigma^2)\Bigr\} = \\ &= -\,\frac{n}{2}\,\log(2\pi\sigma^2) \;-\; \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu)^2. \end{split} \tag{5.7}$$

5.4 Definice, věty, důkazy, ...

Konstrukce typu definice, věta, důkaz, příklad, ...je vhodné odlišit od okolního textu a případně též číslovat s možností použití křížových odkazů. Pro každý typ těchto konstrukcí je vhodné mít v souboru s makry (`makra.tex`) nadefinované jedno prostředí, které zajistí jak vizuální odlišení od okolního textu, tak automatické číslování s možností křížově odkazovat.

Definice 1. *Nechť náhodné veličiny* $X_1, \dots, X_n$ *jsou definovány na témž pravděpodobnostním prostoru* $(\Omega, \mathcal{A}, P)$. *Pak vektor* $\boldsymbol{X} = (X_1, \dots, X_n)^\top$ *nazveme* náhodným vektorem*.*

Definice 2 (náhodný vektor)**.** *Nechť náhodné veličiny* $X_1, \dots, X_n$ *jsou definovány na témž pravděpodobnostním prostoru* $(\Omega, \mathcal{A}, P)$. *Pak vektor* $\boldsymbol{X} = (X_1, \dots, X_n)^\top$ *nazveme* náhodným vektorem*.*

Definice 1 ukazuje použití prostředí pro sazbu definice bez titulku, definice 2 ukazuje použití prostředí pro sazbu definice s titulkem.

Věta 1. *Náhodný vektor* $\boldsymbol{X}$ *je měřitelné zobrazení prostoru* $(\Omega, \mathcal{A}, P)$ *do* $(\mathbb{R}_n, \mathcal{B}_n)$.

Lemma 2 (Anděl [7], str. 29)**.** *Náhodný vektor* $\boldsymbol{X}$ *je měřitelné zobrazení prostoru* $(\Omega, \mathcal{A}, P)$ *do* $(\mathbb{R}_n, \mathcal{B}_n)$.

Důkaz. Jednotlivé kroky důkazu jsou podrobně popsány v práci Anděl [7, str. 29]. □

Věta 1 ukazuje použití prostředí pro sazbu matematické věty bez titulku, lemma 2 ukazuje použití prostředí pro sazbu matematické věty s titulkem. Lemmata byla zavedena v hlavním souboru tak, že sdílejí číslování s větami.

6 Odkazy na literaturu

Při zpracování bibliografie (přehledu použitých zdrojů) se řídíme normou ISO 690 a zvyklostmi oboru. V LaTeXu nám pomohou balíčky biblatex, biblatex-iso690. Zdroje definujeme v souboru `literatura.bib` a pak se na ně v textu práce odkazujeme pomocí makra `\cite`. Tím vznikne odkaz v textu a odpovídající položka v seznamu literatury.

V matematickém textu obvykle odkazy sázíme ve tvaru „Jméno autora/autorů [číslo odkazu]“, případně „Jméno autora/autorů (rok vydání)“. V českém/slovenském textu je potřeba se navíc vypořádat s nutností skloňovat jméno autora, respektive přechylovat jméno autorky. K doplňování jmen se hodí příkazy `\citet`, `\citep` z balíčku natbib, ale je třeba mít na paměti, že produkují referenci se jménem autora/autorů v prvním pádě a jména autorek jsou nepřechýlena.

Jména časopisů lze uvádět zkráceně, ale pouze v kodifikované podobě.

6.1 Několik ukázek

Aktuální verzi této šablony najdete v gitovém repozitáři [8]. Také se může hodit prohlédnout si další návody udržované Martinem Marešem [9].

Mezi nejvíce citované statistické články patří práce Kaplana a Meiera a Coxe [10, 11]. Student [12] napsal článek o t-testu.

Prof. Anděl je autorem učebnice matematické statistiky [13]. Teorii odhadu se věnuje práce Lehmann a Casella [14]. V případě odkazů na specifickou informaci (definice, důkaz, …) uvedenou v knize bývá užitečné uvést specificky číslo kapitoly, číslo věty atp. obsahující požadovanou informaci, např. viz Anděl [7, Věta 4.22].

Mnoho článků je výsledkem spolupráce celé řady osob. Při odkazování v textu na článek se třemi autory obvykle při prvním výskytu uvedeme plný seznam: Dempster, Laird a Rubin [15] představili koncept EM algoritmu. Respektive: Koncept EM algoritmu byl představen v práci Dempstera, Lairdové a Rubina [15]. Při každém dalším výskytu již používáme zkrácenou verzi: Dempster et al. [15] nabízejí též několik příkladů použití EM algoritmu. Respektive: Několik příkladů použití EM algoritmu lze nalézt též v práci Dempstera a kol. [15].

U článku s více než třemi autory odkazujeme vždy zkrácenou formou: První výsledky projektu ACCEPT jsou uvedeny v práci Genbergové a kol. [16]. V textu *nenapíšeme:* První výsledky projektu ACCEPT jsou uvedeny v práci Genberg, Kulich, Kawichai, Modiba, Chingono, Kilonzo, Richter, Pettifor, Sweat a Celentano [16].

7 Tabulky, obrázky, programy

Používání tabulek a grafů v odborném textu má některá společná pravidla a některá specifická. Tabulky a grafy neuvádíme přímo do textu, ale umístíme je buď na samostatné stránky nebo na vyhrazené místo v horní nebo dolní části běžných stránek. LaTeX se o umístění plovoucích grafů a tabulek postará automaticky.

Každý graf a tabulku očíslujeme a umístíme pod ně legendu. Legenda má popisovat obsah grafu či tabulky tak podrobně, aby jim čtenář rozuměl bez důkladného studování textu práce.

Na každou tabulku a graf musí být v textu odkaz pomocí jejich čísla. Na příslušném místě textu pak shrneme ty nejdůležitější závěry, které lze z tabulky či grafu učinit. Text by měl být čitelný a srozumitelný i bez prohlížení tabulek a grafů a tabulky a grafy by měly být srozumitelné i bez podrobné četby textu.

Na tabulky a grafy odkazujeme pokud možno nepřímo v průběhu běžného toku textu; místo *„Tabulka 7.1 ukazuje, že muži jsou v průměru o* 9,9 kg *těžší než ženy“* raději napíšeme *„Muži jsou o* 9,9 kg *těžší než ženy (viz Tabulka 7.1)“*.

7.1 Tabulky

U **tabulek** se doporučuje dodržovat následující pravidla:

- Vyhýbat se svislým linkám. Silnějšími vodorovnými linkami oddělit tabulku od okolního textu včetně legendy, slabšími vodorovnými linkami oddělovat záhlaví sloupců od těla tabulky a jednotlivé části tabulky mezi sebou. V LaTeXu tuto podobu tabulek implementuje balík `booktabs`. Chceme-li výrazněji oddělit některé sloupce od jiných, vložíme mezi ně větší mezeru.

- Neměnit typ, formát a význam obsahu políček v tomtéž sloupci (není dobré do téhož sloupce zapisovat tu průměr, onde procenta).

- Neopakovat tentýž obsah políček mnohokrát za sebou. Máme-li sloupec *Rozptyl*, který v prvních deseti řádcích obsahuje hodnotu 0,5 a v druhých deseti řádcích hodnotu 1,5, pak tento sloupec raději zrušíme a vyřešíme to jinak. Například můžeme tabulku rozdělit na dvě nebo do ní vložit popisné řádky, které informují o nějaké proměnné hodnotě opakující se v následujícím oddíle tabulky (např. *„Rozptyl* = 0,5“ a níže *„Rozptyl* = 1,5“).

Efekt	**Odhad**	**Směrod. chyba**^{a}	**P-hodnota**
Abs. člen	−10,01	1,01	—
Pohlaví (muž)	9,89	5,98	0,098
Výška (cm)	0,78	0,12	< 0,001

Pozn: ^{a} Směrodatná chyba odhadu metodou Monte Carlo.

Tabulka 7.1 Maximálně věrohodné odhady v modelu M.

- Čísla v tabulce zarovnávat na desetinnou čárku.
- V tabulce je někdy potřebné používat zkratky, které se jinde nevyskytují. Tyto zkratky můžeme vysvětlit v legendě nebo v poznámkách pod tabulkou. Poznámky pod tabulkou můžeme využít i k podrobnějšímu vysvětlení významu některých sloupců nebo hodnot.

7.2 Obrázky

Dodejme ještě několik rad týkajících se obrázků a grafů.

- Graf by měl být vytvořen ve velikosti, v níž bude použit v práci. Zmenšení příliš velkého grafu vede ke špatné čitelnosti popisků.
- Osy grafu musí být řádně popsány ve stejném jazyce, v jakém je psána práce (absenci diakritiky lze tolerovat). Kreslíme-li graf hmotnosti proti výšce,enecháme na nich popisky `ht` a `wt`, ale osy popíšeme *Výška [cm]* a *Hmotnost [kg]*. Kreslíme-li graf funkce $h(x)$, popíšeme osy x a $h(x)$. Každá osa musí mít jasně určenou škálu.
- Chceme-li na dvourozměrném grafu vyznačit velké množství bodů, dáme pozor, aby se neslily do jednolité černé tmy. Je-li bodů mnoho, zmenšíme velikost symbolu, kterým je vykreslujeme, anebo vybereme jen malou část bodů, kterou do grafu zaneseme. Grafy, které obsahují tisíce bodů, dělají problémy hlavně v elektronických dokumentech, protože výrazně zvětšují velikost souborů.
- Budeme-li práci tisknout černobíle, vyhneme se používání barev. Čáry rozlišujeme typem (plná, tečkovaná, čerchovaná,…), plochy dostatečně rozdílnými intensitami šedé nebo šrafováním. Význam jednotlivých typů čar a ploch vysvětlíme buď v textové legendě ke grafu anebo v grafické legendě, která je přímo součástí obrázku.
- Vyhýbejte se bitmapovým obrázkům o nízkém rozlišení a zejména JPEGům (zuby a kompresní artefakty nevypadají na papíře pěkně). Lepší je vytvářet obrázky vektorově a vložit do textu jako PDF.

7.3 Programy

Algoritmy, výpisy programů a popis interakce s programy je vhodné odlišit od ostatního textu. Pro programy se hodí prostředí `lstlisting` z LaTeXového balíčku `listings`, které umí i syntakticky zvýrazňovat běžné programovací jazyky. Většinou ho chceme obalit prostředím `listing`, čímž z něj uděláme plovoucí objekt s popiskem (viz Program 3).

Pro algoritmy zapsané v pseudokódu můžeme použít prostředí `algorithmic` z balíčku `algpseudocode`. Plovoucí objekt z něj uděláme obalením prostředím `algorithm`. Příklad najdete v Algoritmu 1.

Program 3 Můj první program

```c
#include <stdio.h>

int main(void)
{
  printf("Hello, world!\n");
  return 0;
}
```

Algoritmus 1 Primitivní pravděpodobnostní test prvočíselnosti. Pokud odpoví NE, číslo x určitě není prvočíslem. Pokud odpoví ANO, nejspíš se mýlí.

function ISPRIME(x)
    $r \leftarrow$ rovnoměrně náhodné celé číslo mezi 2 a $x - 1$
    $z \leftarrow x \bmod r$
    if $z > 0$ **then**
        Vrátíme NE ▷ Našli jsme dělitele
    else
        Vrátíme ANO ▷ Možná se mýlíme
    end if
end function

Další možností je použití LaTeXového balíčku `fancyvrb` (fancy verbatim), pomocí něhož je v souboru `makra.tex` nadefinováno prostředí `code`. Pomocí něho lze vytvořit např. následující ukázky.

V základním nastavení dostaneme:

```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Můžeme si říci o menší písmo:

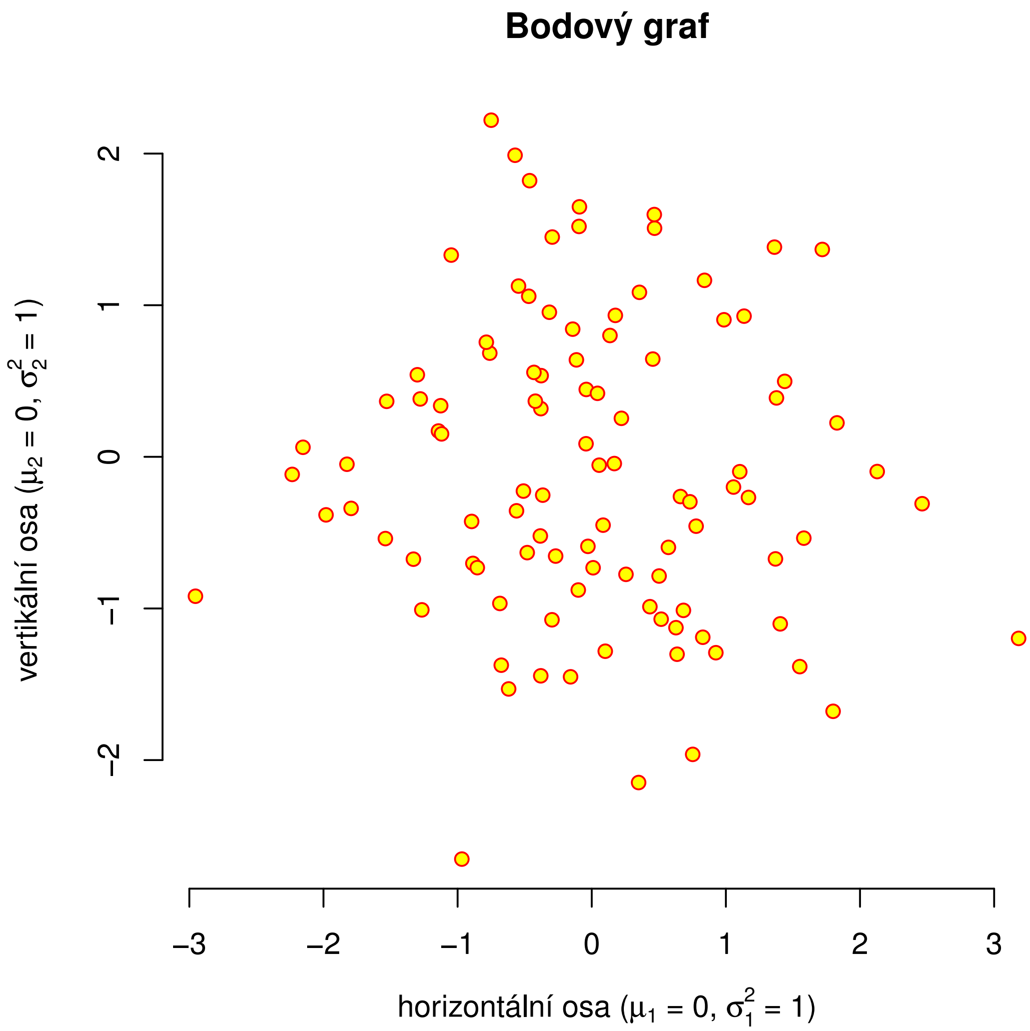
```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Nebo vypnout rámeček:

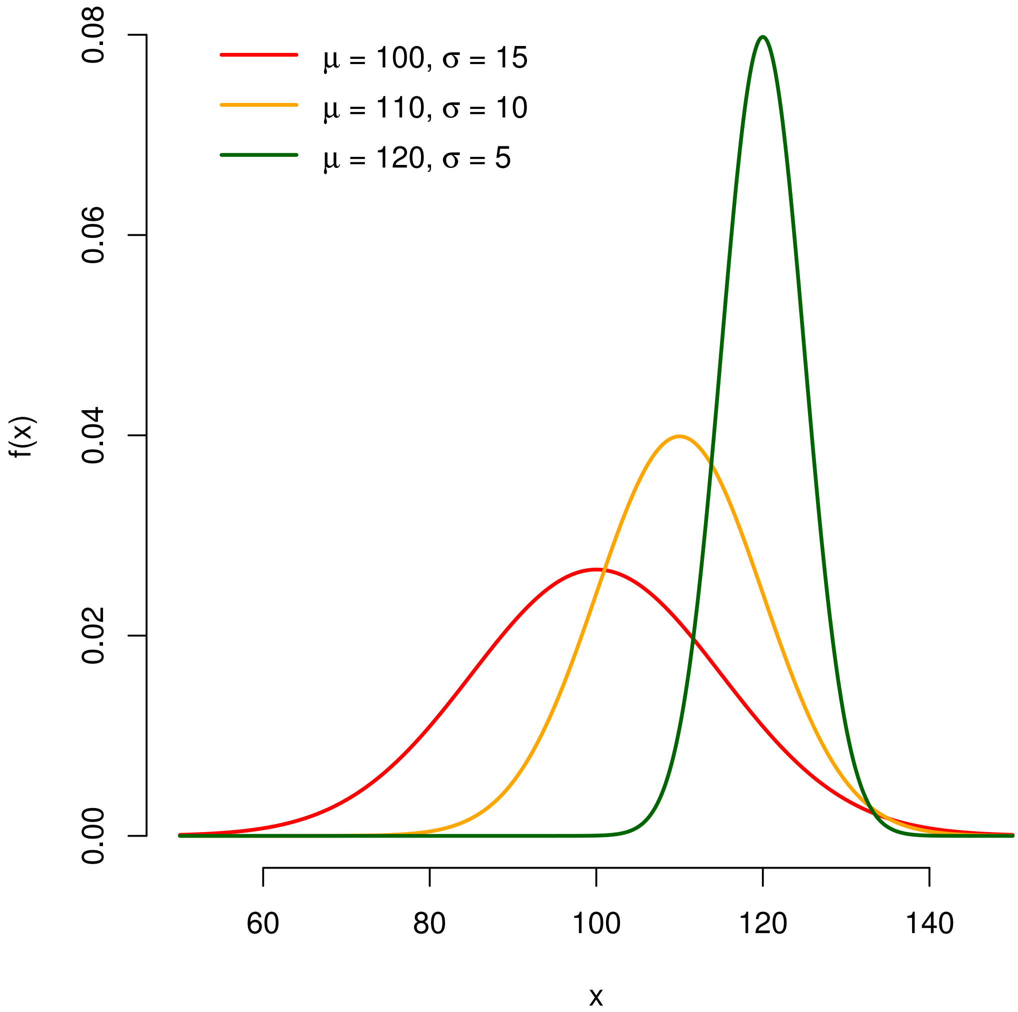
```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Případně si říci o užší rámeček:

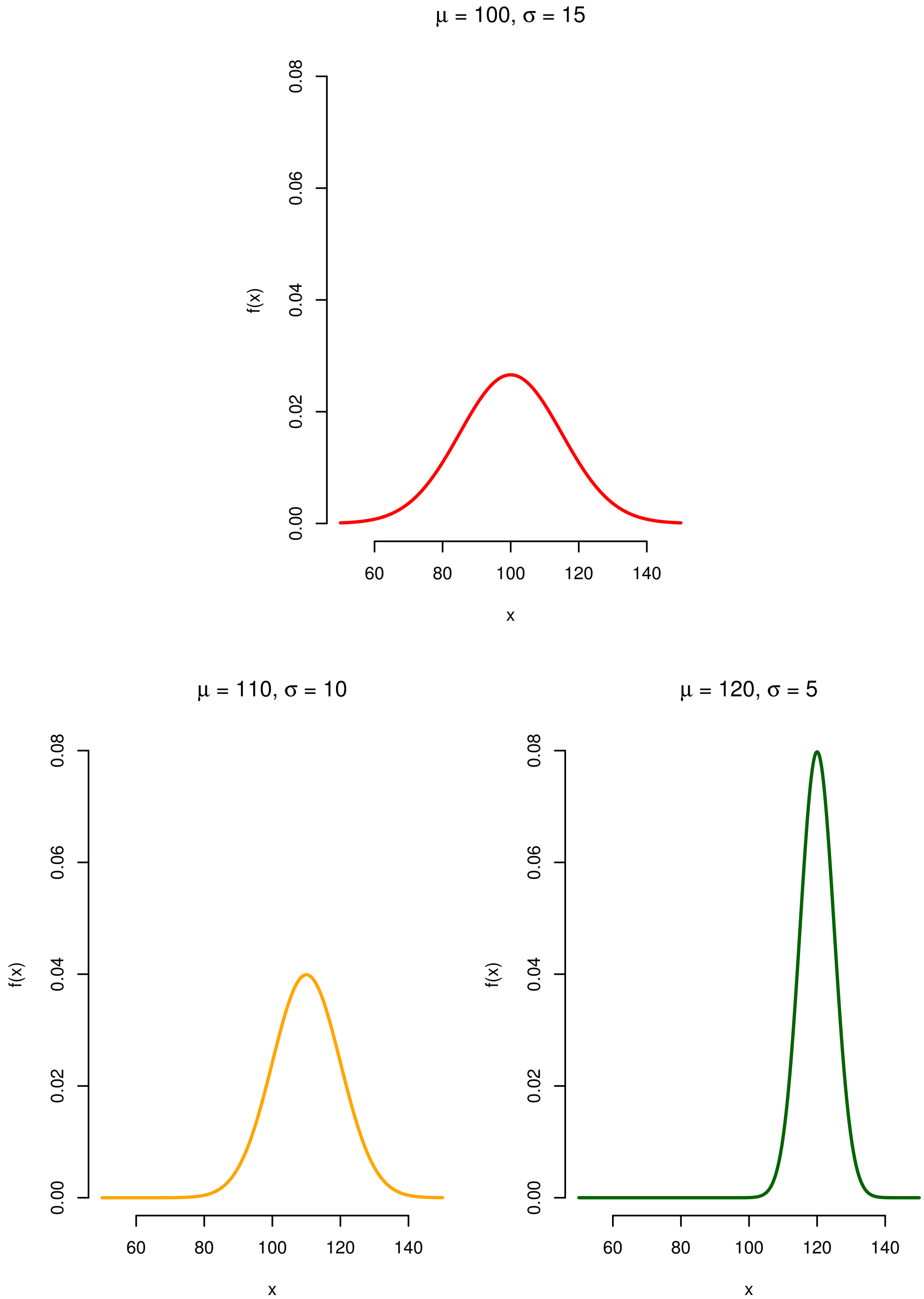
```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```



Obrázek 7.1 Náhodný výběr z rozdělení $\mathcal{N}_2(\mathbf{0}, I)$.



Obrázek 7.2 Hustoty několika normálních rozdělení.



Obrázek 7.3 Hustoty několika normálních rozdělení.

8 Formát PDF/A

Opatření rektora č. 13/2017 určuje, že elektronická podoba závěrečných prací musí být odevzdávána ve formátu PDF/A úrovně 1a nebo 2u. To jsou profily formátu PDF určující, jaké vlastnosti PDF je povoleno používat, aby byly dokumenty vhodné k dlouhodobé archivaci a dalšímu automatickému zpracování. Dále se budeme zabývat úrovní 2u, kterou sázíme TEXem.

Mezi nejdůležitější požadavky PDF/A-2u patří:

- Všechny fonty musí být zabudovány uvnitř dokumentu. Nejsou přípustné odkazy na externí fonty (ani na „systémové", jako je Helvetica nebo Times).
- Fonty musí obsahovat tabulku ToUnicode, která definuje převod z kódování znaků použitého uvnitř fontu to Unicode. Díky tomu je možné z dokumentu spolehlivě extrahovat text.
- Dokument musí obsahovat metadata ve formátu XMP a je-li barevný, pak také formální specifikaci barevného prostoru.

Tato šablona používá balíček `pdfx,` který umí LATEX nastavit tak, aby požadavky PDF/A splňoval. Metadata v XMP se generují automaticky podle informací v souboru `prace.xmpdata` (na vygenerovaný soubor se můžete podívat v `pdfa.xmpi`).

Validitu PDF/A můžete zkontrolovat pomocí nástroje VeraPDF, který je k dispozici na `http://verapdf.org/`.

Pokud soubor nebude validní, mezi obvyklé příčiny patří používání méně obvyklých fontů (které se vkládají pouze v bitmapové podobě a/nebo bez unicodových tabulek) a vkládání obrázků v PDF, které samy o sobě standard PDF/A nesplňují.

Další postřehy o práci s PDF/A najdete na `http://mj.ucw.cz/vyuka/bc/pdfaq.html`.

Závěr

Literatura

1. Chang, Edward Y. Examining gpt-4: Capabilities, implications and future directions. In: *The 10th International Conference on Computational Science and Computational Intelligence*. 2023.
2. Achiam, Josh; Adler, Steven; Agarwal, Sandhini; Ahmad, Lama; Akkaya, Ilge; Aleman, Florencia Leoni; Almeida, Diogo; Altenschmidt, Janko; Altman, Sam; Anadkat, Shyamal et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*. 2023.
3. Hendrycks, Dan; Burns, Collin; Basart, Steven; Zou, Andy; Mazeika, Mantas; Song, Dawn; Steinhardt, Jacob. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*. 2020.
4. Bang, Yejin; Cahyawijaya, Samuel; Lee, Nayeon; Dai, Wenliang; Su, Dan; Wilie, Bryan; Lovenia, Holy; Ji, Ziwei; Yu, Tiezheng; Chung, Willy; Do, Quyet V.; Xu, Yan; Fung, Pascale. *A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity*. 2023. Dostupné z arXiv: `2302.04023 [cs.CL]`.
5. White, Jules; Fu, Quchen; Hays, Sam; Sandborn, Michael; Olea, Carlos; Gilbert, Henry; Elnashar, Ashraf; Spencer-Smith, Jesse; Schmidt, Douglas C. *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. 2023. Dostupné z arXiv: `2302.11382 [cs.SE]`.
6. Union, European. *Database - Eurostat*. Dostupné také z: `https://ec.europa.eu/eurostat/data/database`.
7. Anděl, J. *Základy matematické statistiky*. Praha: Matfyzpress, 2007. Druhé opravené vydání. ISBN 80-7378-001-1.
8. Mareš, Martin (ed.). *Šablona závěrečné práce na MFF UK v LaTeXu* [online]. [cit. 2024-03-02]. Dostupné z: `https://gitlab.mff.cuni.cz/teaching/thesis-templates/thesis-cs`.
9. Mareš, Martin. *Jak psát bakalářskou (či jinou) práci* [online]. [cit. 2024-03-02]. Dostupné z: `https://mj.ucw.cz/vyuka/bc/`.
10. Kaplan, E. L.; Meier, P. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*. 1958, roč. 53, č. 282, s. 457–481.
11. Cox, D. R. Regression models and life-tables (with Discussion). *Journal of the Royal Statistical Society, Series B*. 1972, roč. 34, č. 2, s. 187–220.
12. Student. On the probable error of the mean. *Biometrika*. 1908, roč. 6, s. 1–25.
13. Anděl, J. *Statistické metody*. Praha: Matfyzpress, 1998. Druhé přepracované vydání. ISBN 80-85863-27-8.
14. Lehmann, E. L.; Casella, G. *Theory of Point Estimation*. New York: Springer-Verlag, 1998. Second Edition. ISBN 0-387-98502-6.
15. Dempster, A. P.; Laird, N. M.; Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*. 1977, roč. 39, č. 1, s. 1–38.

16. Genberg, B. L.; Kulich, M.; Kawichai, S.; Modiba, P.; Chingono, A.; Kilonzo, G. P.; Richter, L.; Pettifor, A.; Sweat, M.; Celentano, D. D. HIV risk behaviors in Sub-Saharan Africa and Northern Thailand: Baseline behavioral data from project Accept. *Journal of Acquired Immune Deficiency Syndrome*. 2008, roč. 49, s. 309–319.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A Přílohy

A.1 První příloha