

## PABLO MUÑOZ ALCAIDE - PROMPTING REPORT

In this report, we'll see different prompting structures for data science in chatGPT. We'll explain different scheme promptings. Then, we will use these prompting schemes in practical cases using chatGPT and analyze the results.

**1st Prompt:** The first schema of prompting is:



This scheme in this work is used to obtain different chain values of the “BiciCas” database we worked in class.

In the first prompt, I asked chat GPT to act as a data scientist and I explain how “BiciaCas” works, then I passed the data and I explain a little what is a data chain value is and finally I ask for the data chain values as follows:

P

Hello, I want you to ask as a data scientist. I have a database which have the data of a system of bicycle renting. This is system formed by bicycles that are in different stations in the city and you can pick up one (if you are an authenticated user) and have a limit of two hours to use it and leave it at in one station.

Then, we have this database of the system which is formed by this datasets or tables:

```
anchors(uuid, bench_uuid, number)
bans(uuid, use_uuid, date_start, date_expire, date_created, reason)
benches_anchors(bench_uuid, anchor_uuid, anchor_number, bicycle_number,
date_created)
benches(uuid, name, latitude, longitude, date_created)
benches_notifications(uuid, bench_uuid, notification, priority, date_created)
bench_messages(uuid, message, date_start, date_expire, date_created)
bicycles(uuid, number, date_created, status)
incidents(uuid, error_key, use_uuid, anchor_uuid, status, date_created, bike_uuid)
loans(uuid, use_uuid, bicycle_uuid, anchor_uuid, date_created, type_access)
putbacks(uuid, bicycle_uuid, anchor_uuid, date_created)
users(change_uuid, uuid, document, name, surname, street, city, postcode, birthday, phone,
email, gender, date_created, date_change, deleted)
bench_status(bench_uuid, date_lastseen, ip, queue, version, number_loans)
loan_historical(uuid, loan_uuid, use_uuid, bicycle_uuid, loan_anchor_uuid, loan_date_created,
putback_uuid, putback_anchor_uuid, putback_date_created, type_access, app_loan)
```

I want you to ask as a data scientist. What different chain values will you create with this raw data?

Chain values means to create beautiful insights from this raw data. And divided in Historical chain values and streaming chain values. A historical chain value would be:

"Analyze the most common routes and in which time there were made".

An streaming chain value would be:

"Alerts of full stations to the users".

**And his report of the data was:**

**CHATGPT:** As a data scientist, you can derive valuable historical and streaming chain values from the raw data of the bicycle rental system. Here are some examples:

**\*\*Historical Chain Values:\*\***

1. **\*\*Route Analysis:\*\*** Calculate and visualize the most common routes taken by users. You can use the `loans` and `putbacks` tables to identify the start and end

points of each trip and aggregate this data to determine popular routes. This can help in optimizing bicycle placement or identifying areas with high demand.

2. **Usage Patterns by Time:** Analyze how bicycle usage patterns vary by time of day, day of the week, and season. You can use the `loans` table to track when bicycles are rented and returned. This information can be used to optimize station maintenance schedules and allocate resources efficiently.

3. **User Behavior Analysis:** Study user behavior over time. You can track how frequently users rent bicycles, the average duration of their trips, and their preferred stations. This analysis can help in tailoring marketing campaigns or improving user experience.

4. **Incident Analysis:** Examine incident data from the `incidents` table to identify common issues with bicycles or stations. This can lead to improvements in maintenance and user safety.

5. **User Demographics:** Utilize data from the `users` table to understand the demographics of your user base. This can help in targeted marketing efforts and expanding services to specific user segments.

#### **Streaming Chain Values:**

1. **Station Capacity Alerts:** Implement a real-time alerting system that notifies users when a station is nearing full capacity. You can use the `bench\_status` table to monitor the current number of bicycles at each station and send alerts when a station is close to being full. This improves the user experience by helping them find available parking spaces for returned bicycles.

2. **Maintenance Alerts:** Set up alerts based on data from the `incidents` table to notify maintenance teams when there are critical issues with bicycles or stations that require immediate attention. This ensures quick response times and better system reliability.

3. **User Notifications:** Send personalized notifications to users based on their past behavior, such as suggesting nearby stations with available bicycles or offering discounts during off-peak hours.

4. **Demand Prediction:** Use streaming data from the system to predict future demand at different stations. This can help in optimizing bicycle redistribution and ensuring that stations are adequately stocked during peak usage times.

5. **Traffic and Weather Alerts:** Integrate real-time traffic and weather data to provide users with information on optimal routes and weather conditions for their bicycle trips. This enhances user safety and satisfaction.

In summary, by leveraging historical and streaming data from the bicycle rental system, you can create valuable insights and provide real-time services to improve the user experience, optimize operations, and enhance system reliability.

Then I sent to ChatGPT the data chain values that we create in class. And I asked ChatGPT to combine our data chain values with its data chain values to enrich the data as much as possible.

**CHATGPT:** Certainly, let's combine the chain values you provided with the ones I mentioned earlier to create a comprehensive set of historical and streaming data chain values for your bicycle rental system:

**Historical Chain Values:**

1. **Incidents by Demographics, Geography, and Time:** Analyze incidents from the `incidents` table to understand how incidents vary across different user demographics (e.g., age, gender) and geographic locations (e.g., neighborhoods or districts). Additionally, consider how incident rates change over time, identifying any trends or patterns.

2. **Most Frequent Routes with Seasonality:** Combine the analysis of the most frequent routes from the `loans` and `putbacks` tables with seasonality data to identify not only the most common routes but also how these routes change over different seasons. This can help in planning for seasonal variations in demand.

3. **Historical Bench Occupancy vs. Demand:** Calculate historical bench occupancy rates using data from the `benches\_anchors` and `bench\_status` tables and compare them to historical demand data. This can provide insights into how well the system matches bench capacity with user demand.

4. **Simulation of Resource Movement:** Develop a probabilistic generative model that simulates the movement of bicycles based on historical usage patterns. This model can help in optimizing bicycle redistribution strategies by predicting where and when resources are likely to be needed.

5. **Quality Assessment of Benches and Bicycles:** Create a metric to assess the "state" of quality for both benches and bicycles. Consider factors such as the number

of kilometers traveled by bicycles, maintenance history, and user feedback. This can inform maintenance schedules and replacement decisions.

6. **Bicycle Travel Analysis:** Explore data on bicycle travel patterns to identify trends, such as preferred travel distances, typical trip durations, and routes taken. This information can guide infrastructure expansion and user education efforts.

7. **Bicycle Popularity and Technical Issues:** Analyze the popularity of different bicycle models using data from the `bicycles` table and correlate it with incidents related to technical problems. This can help in optimizing the bicycle fleet by prioritizing maintenance and replacements for popular models.

#### **Streaming Chain Values:**

1. **Real-time Incidents and Operational Bicycles:** Continuously monitor and calculate the current status of operational bicycles in real time, as well as the incidence of technical problems. Use this information to provide immediate alerts to maintenance teams and users if issues arise.

2. **Resource Mobility and Predictive Demand:** Combine real-time data on bicycle availability with predictive demand models to dynamically allocate resources, ensuring that bicycles are efficiently distributed across stations.

3. **Targeted Warnings and Alerts:** Implement a real-time alert system that sends targeted warnings to users based on their location, weather conditions, and system status. For example, warn users of approaching full stations or suggest alternative nearby stations.

4. **Predictive Backfilling Model:** Use real-time data on empty racks and predictive models to predict when and where backfilling of bicycles may be needed. This can help in proactively addressing low-stock situations.

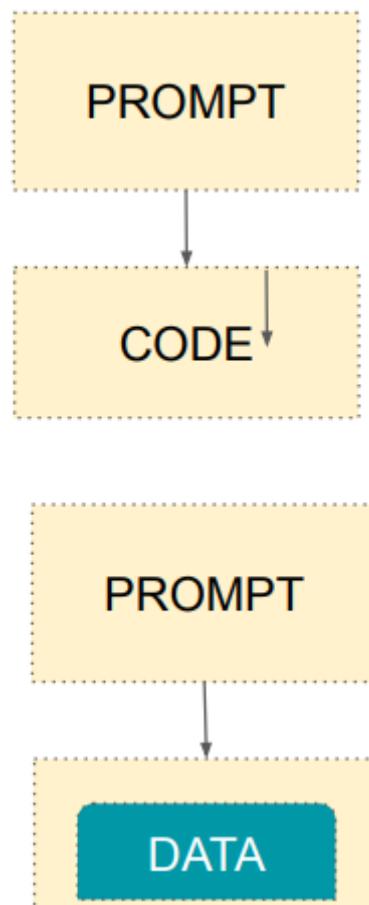
5. **User Location-based Recommendations:** Continuously recommend nearby bike racks to users based on their current location and predicted travel patterns. Include estimated travel times to these racks to enhance user convenience.

6. **Full Rack Alerts:** Monitor station occupancy in real time and send alerts to users when a nearby rack is full, directing them to alternative locations.

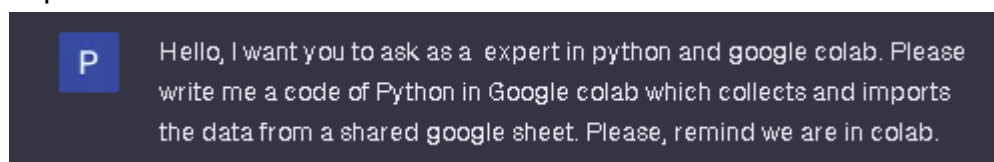
By combining these historical and streaming data chain values, you can create a comprehensive data-driven ecosystem for your bicycle rental system that not only provides insights into past performance but also enables real-time decision-making to optimize operations and enhance user satisfaction.

**Conclusions:** As we can see, in the first report it's quite impressive how with a little bit of context and the different parameters of the database ChatGPT can create such as good data chain values. Then with the value chains I provided the result was even better, giving us very useful value chains that could be assessed for implementation in a real case.

**2nd Prompt:** The next prompt is based on the next schema of prompting:



First I asked ChatGPT to collect and import data from a shared google sheet with the next prompt:



**ChatGPT:** Certainly! To collect and import data from a shared Google Sheet in Google Colab, you can use the `gsread` library along with the Google Sheets API. Here are the steps to achieve this:

1. Install and authenticate the necessary libraries.
2. Access the Google Sheet by its URL.
3. Read data from the Google Sheet into a Pandas DataFrame.

Make sure you have the Google Sheets API credentials JSON file (`credentials.json`) ready for authentication. You can create this file by setting up a Google Sheets API project in the Google Cloud Console.

Here's a Python code snippet that accomplishes this:

```
# Install necessary libraries
!pip install gspread pandas

# Import the required libraries
import gspread
from google.colab import auth
import pandas as pd

# Authenticate with Google Sheets API
auth.authenticate_user()

# Access the Google Sheet by its URL
# Replace 'YOUR_SPREADSHEET_URL' with the URL of your Google Sheet
spreadsheet_url = 'YOUR_SPREADSHEET_URL'

# Create a connection to the Google Sheet
gc = gspread.open_by_url(spreadsheet_url)

# Select a specific worksheet within the Google Sheet
# Replace 'Sheet1' with the name of your worksheet
worksheet = gc.worksheet('Sheet1')

# Read data from the worksheet into a Pandas DataFrame
data = worksheet.get_all_values()
df = pd.DataFrame(data[1:], columns=data[0]) # Assuming the first row contains
headers

# Print the DataFrame
print(df)
```

Replace `"YOUR_SPREADSHEET_URL"` with the URL of the Google Sheet you want to access. This code will authenticate your Colab session, access the specified Google Sheet, and load its data into a Pandas DataFrame for further processing.

Ensure that you have shared the Google Sheet with the email address associated with your Google Cloud Console project, which you used to generate the ``credentials.json`` file.

Remember to follow Google's guidelines on managing credentials and access permissions for security.

Then I explain what data have the google sheet and I ask ChatGPT to give me what statistics it will use to analyze the data and I asked for a imaginary report of the data.

P

I the google sheet I want to import is a questionnaire of the different skills in data science of each student in a classroom of Big Data. The different columns of the data are E-mail adress, Data Processing/Engineering, Machine Learning, Deep Learning, Big Data, Non conventional data, Others. And In each case the possible entries of the rows are languages of programming, different type of software fit to the name of the column. In the case of the column Others there are different interdisciplinary skills. Please, as a data scientist, tell me which statistics and data visualizations will you do to create a good report of the data. Also create an imaginary report using that statistics.

**CHATGPT:** To create a good report of the data from the questionnaire on skills in data science of each student in a Big Data classroom, you can perform various statistical analyses and data visualizations. Here's a list of statistics and data visualizations you can consider:

1. **\*\*Descriptive Statistics\*\***:

- Calculate summary statistics (mean, median, mode, standard deviation, etc.) for each skill category (Data Processing/Engineering, Machine Learning, Deep Learning, Big Data, Non-conventional data, Others).
- Visualize summary statistics using bar charts or box plots to show the distribution of skills across students.

2. **\*\*Skill Frequency Counts\*\***:

- Count the frequency of each programming language, software, or interdisciplinary skill mentioned in the "Others" column.
- Create bar charts or word clouds to display the most commonly mentioned skills.

3. **\*\*Correlation Analysis\*\***:



- Examine the correlation between different skills. For example, you can use a correlation matrix to see if students who excel in Data Processing/Engineering also tend to excel in Machine Learning or Deep Learning.

4. **Skill Distribution**:

- Create histograms or density plots to visualize the distribution of skills for each category. This can help identify skill gaps or areas where students excel.

5. **Skill Over Time**:

- If you have multiple survey data points over time, analyze how skills have evolved among students over different time periods.

6. **Stacked Bar Charts**:

- Create stacked bar charts to visualize the composition of skills within each category. This can show the proportion of students with different combinations of skills.

7. **Skill Proficiency Levels**:

- Categorize skill proficiency into levels (e.g., beginner, intermediate, advanced) and visualize the distribution of skill levels across students.

8. **Geographic Analysis**:

- If you have location data, you can perform a geographic analysis to see if skill distributions vary by region.

Here's an imaginary report using some of these statistics and visualizations:

**Title:** Skills Assessment Report for Big Data Classroom

**Introduction:**

- Provide background information about the survey and its purpose.
- Mention the total number of respondents.

**Summary Statistics:**

- Present summary statistics (mean, median, standard deviation) for each skill category.
- Include a visual representation of the distributions using box plots.

**Skill Frequency Counts:**

- Highlight the most frequently mentioned programming languages, software, and interdisciplinary skills.
- Use word clouds to visually display the most common skills.

**Correlation Analysis:**

- Discuss any significant correlations between skills.
- Include a correlation matrix heatmap.

**\*\*Skill Distribution:\*\***

- Show histograms or density plots for each skill category.
- Identify areas where students excel and areas that need improvement.

**\*\*Stacked Bar Charts:\*\***

- Display the composition of skills within each category using stacked bar charts.
- Identify prevalent skill combinations.

**\*\*Skill Proficiency Levels:\*\***

- Categorize skill proficiency levels and visualize the distribution.
- Discuss implications for teaching and curriculum development.

**\*\*Geographic Analysis (if applicable):\*\***

- If location data is available, analyze skill distributions by region.
- Provide insights into regional skill variations.

**\*\*Conclusion:\*\***

- Summarize key findings from the analysis.
- Highlight areas for improvement and potential strategies for curriculum enhancement.
- Conclude with the overall assessment of the skills of the students in the Big Data classroom.

Remember to use clear and informative visualizations, provide context for the data, and draw actionable insights to make the report valuable for decision-making and curriculum planning.

**Conclusion:** In this chain of prompts we don't have the same results as in the previous prompt. First, In the code we have different lines of code that are good but some of them raise some errors. Nevertheless, lines of code that are good help me to achieve the correct solution of the problem, so this show one of the main functionalities of ChatGPT which is to help us to work faster not to replace us. Lately, in the second prompt the response of ChatGPT provides us a model of a report we could use in a real case to analyze the google sheet data.

Lastly, the last chain of prompt I used is the next:



In which, the first prompt give an schema of some topic and then ChatGPT will explain it. Then, the next prompt is another schema which gives the necessary information to translate the schema in code.

In my case the first prompt was an schema of an algorithm of Machine Learning and I ask him to describe with detail the algorithm:

```
P Hello, I want you to ask as an expert in machine learning. Describe this scheme of the decision trees algorithm.

generate_tree (T_tra, alpha): m
  if entropy I_m(T_tra) <= alpha:
    (Ttra) ≤ α:
    m <- class label of the most represented class in T_tra
  else
    a = the_most_discriminating_attribute(T_tra)
    m <- a
    for each branch a_i of a:
      tra_i = {x in T_tra | x satisfies the condition of a_i}
      h = generate_tree(tra_i)
      add_child(m,h)
    return m
```

**CHATGPT:** The code you've provided outlines a high-level algorithm for generating a decision tree using the ID3 (Iterative Dichotomiser 3) algorithm, a popular algorithm for constructing decision trees in machine learning and data mining. This algorithm is typically used for classification tasks. I'll break down the key components and steps in the code:

- Function Signature:** The function `generate_tree` takes two arguments:
  - `T_tra`: The training dataset, which is a collection of data points with their associated class labels.
  - `alpha`: A threshold value used for controlling when to stop splitting and create leaf nodes.
- Entropy-Based Stopping Criterion:** The algorithm starts by checking if the entropy of the current dataset `T_tra` is less than or equal to the given threshold

$\alpha$ . Entropy ( $-\sum p_i \log_2 p_i$ ) is a measure of impurity or disorder in the dataset. If the entropy is below the threshold, it suggests that the data is relatively pure or homogeneous, and further splitting is not needed.

- If the entropy is less than or equal to  $\alpha$ , it means the data is pure enough, and the algorithm creates a leaf node  $m$  with the class label of the most represented class in  $T_{tra}$ .

- If the entropy is greater than  $\alpha$ , the algorithm proceeds to the next step to split the data.

3. **Attribute Selection**: The code selects the most discriminating attribute  $a$  from the available attributes in the dataset  $T_{tra}$ . This attribute selection process is typically based on measures like information gain or Gini impurity, aiming to find the attribute that provides the most information for splitting the data effectively.

4. **Recursion**: After selecting the discriminating attribute  $a$ , the algorithm iterates through each branch  $a_i$  of the attribute  $a$ . For each branch, it creates a subset  $tra_i$  of the data  $T_{tra}$ , containing only the data points that satisfy the condition of branch  $a_i$ . This essentially represents the data points that follow the path corresponding to branch  $a_i$ .

5. **Recursive Tree Building**: For each branch  $a_i$ , the algorithm recursively calls the `generate_tree` function on the subset  $tra_i$ . This recursive process continues until the stopping criterion (entropy  $\leq \alpha$ ) is met for a particular branch, and a leaf node is created.

6. **Tree Construction**: The code adds each of the generated subtrees  $h$  as children of the node  $m$ . This represents the hierarchical structure of the decision tree.

7. **Returning the Tree**: Finally, the algorithm returns the root node of the decision tree, which represents the entire decision tree structure.

This code outlines the basic steps of a decision tree construction algorithm based on entropy as a purity measure and recursive partitioning of the data. The tree-building process aims to create a tree that efficiently classifies data points by recursively selecting the most informative attributes and creating decision nodes accordingly.

Then I didn't give another schema because in this case ChatGPT doesn't need it. But I asked to translate the algorithm into Python code.

**CHATGPT:** Certainly! Here's a Python function that implements the decision tree algorithm based on the code you provided. This implementation assumes that you

have a dataset represented as a Pandas DataFrame with features and a target column:

```
```python
import pandas as pd
import numpy as np

class TreeNode:
    def __init__(self, data, attribute=None, branches=None):
        self.data = data # Class label if leaf node, attribute if internal node
        self.attribute = attribute # Attribute used for splitting (None for leaf nodes)
        self.branches = branches # Child nodes (subtrees)

def generate_tree(data, alpha=0.1):
    # Check if data is pure enough to create a leaf node
    if entropy(data) <= alpha:
        # Create a leaf node with the majority class label
        return TreeNode(data.mode().iloc[0])
    else:
        # Select the most discriminating attribute
        discriminating_attr = select_attribute(data)
        root = TreeNode(attribute=discriminating_attr)
        for value in data[discriminating_attr].unique():
            # Create a branch for each unique value of the discriminating attribute
            subset = data[data[discriminating_attr] == value]
            child_tree = generate_tree(subset, alpha)
            root.branches[value] = child_tree
        return root

def entropy(data):
    # Calculate entropy of a dataset based on the class labels
    if len(data) == 0:
        return 0
    class_counts = data['class'].value_counts()
    probabilities = class_counts / len(data)
    entropy = -sum(probabilities * np.log2(probabilities))
    return entropy

def select_attribute(data):
    # Dummy function for attribute selection; replace with your own logic
    # This function should select the attribute with the highest information gain or
    # lowest Gini impurity
    return data.columns[0] # Replace with your attribute selection logic
```
```

```
# Example usage:  
# Assuming you have a DataFrame 'df' with features and a 'class' column  
# representing class labels  
# tree = generate_tree(df)  
```\n
```

In this code:

- The `TreeNode` class represents nodes in the decision tree, both internal (attribute nodes) and leaf (class label nodes).
- The `generate\_tree` function recursively builds the decision tree based on the provided dataset and alpha threshold.
- The `entropy` function calculates the entropy of a dataset based on class labels.
- The `select\_attribute` function is a placeholder for attribute selection logic; you should replace it with a proper attribute selection method such as information gain or Gini impurity.

Please note that this is a basic implementation, and in practice, you would need more sophisticated methods for attribute selection and handling various data types and cases.

**Conclusion:** This last example is one of the most impressive. Firstly, one of the aspects that impresses the most is the fact that ChatGPT identifies correctly all of the parameters of the schema. And then ChatGPT explains with a lot of detail what every parameter is and what is their function. Finally, ChatGPT translates the schema into python code. Another aspect that impresses in this translation is the fact that chatGPT also creates an Entropy function correctly which it was only mentioned without giving the explicit expression.