Fine-tuning GPT-3 for data visualization code generation

JORIS BOSMAN, University of Twente, The Netherlands

Data and data visualization play crucial roles in the field of technology. However, data visualization is not always a simple and quick process, and it often involves a learning curve. To address this challenge and enhance the efficiency of creating graphical displays, one approach is to utilize a GPT (Generative Pre-trained Transformer). GPTs are trained to understand natural language prompts and generate responses, including code snippets. By providing specific instructions in natural language, the GPT can generate code tailored to a data visualization library, which can then interpret and create the desired visual representation of the data.

It's important to note that while GPTs are powerful tools, they are not without flaws and may make mistakes. However, by fine-tuning a GPT, we can improve its performance and reduce the occurrence of errors. Therefore, this project aims to investigate the effectiveness of fine-tuning GPT-3 specifically for data visualization tasks. By evaluating its capabilities and limitations, we can gain insights into the potential of leveraging GPTs for more efficient and accurate data visualization processes.

Additional Key Words and Phrases: GPT-3, Fine-tuning, Data visualization, Code generation, LLM

1 INTRODUCTION

In the ever-growing world of technology, one factor is always important: data. Data is used everywhere, from social media applications suggesting the best possible videos to the users [2] to medical data analysis in order to advance medical research [5]. In order for people to analyze data in a meaningful way, data visualization is a key part of understanding the data in the first place.

Making graphical displays, like graphs and charts, can be done in a variety of ways. Microsoft Excel, or any other spreadsheet-type application, is the first that comes to mind, which is a program used for data management and visualization. The spreadsheet creates a table-like structure that is good for data input. This input can be selected and converted into graphical displays. This is a great way of making data visualizations, but Excel has a bit of a learning curve attached to it and making the graphs with the correct data and axis can be a challenge. To achieve mastery in Excel, or an application like it, will require time and effort. More data visualization tools exist, like Tableau, Dundas BI, and Google Charts, to name a few, but they are similar to Excel, as they also require time and effort to learn them.

A different manner of creating graphs from data is to use a Generative Pre-trained Transformer(GPT), for example ChatGPT [9]. A GPT is an autoregressive large language model meant for generative artificial intelligence. A large language model is a language model, which is a probability distribution over a sequence of words, consisting of a large neural network usually containing billions or more weights. This network is then pre-trained using semi-supervised learning or self-supervised learning on a large quantity of text. The model is trained to predict the continuation of a natural language prompt. For example, the prompt `'It is healthy to '` would result in the response `'exercise'`. Despite a GPT's ability to be very powerful, code generation based on large language models, like ChatGPT, is not perfect, resulting in errors [3][6].

This project aims to find a way to make data visualization easier and faster, while still being reliable. A GPT could be an excellent alternative for data visualization, as a user can simply tell it what they want in natural language. Because a GPT is error-prone and fine-tuning trains the model further, the following hypothesis is constructed. Fine-tuning a GPT, like GPT-3, will improve the correctness of code generated for data visualization. In order to test the hypothesis, the subsequent research questions are proposed.

- To what extent can fine-tuning GPT-3 code generation from natural language interpretation improve accurate code output for data visualization and reduce time spent on data visualization?
- How to create a valid dataset that enables fine-tuning for GPT-3?

In order to address the first research question, it is necessary to answer the second research question first. This is because the process of fine-tuning GPT-3 requires the availability of data. The data should consist of prompts and corresponding responses. For the purpose of data visualization, the Python library Plotly is utilized, as it offers visually appealing and customizable graphs that are responsive. Before creating the dataset, specific criteria for selecting the data are discussed. Subsequently, a portion of the dataset is utilized to test GPT-3 without any fine-tuning. Another portion of the dataset is employed to fine-tune GPT-3. Once the fine-tuning process is completed, a different segment of the dataset is used to evaluate the performance of the fine-tuned GPT-3 model, and the results are then analyzed.

2 RELATED WORK

2.1 Data visualization from natural language

Chat2VIS is a program that is able to produce data visualization based on a natural language input [10]. The code generation is done using three different large language models: ChatGPT, GPT-3, and CODEX, although CODEX has now been deprecated and can therefore not be used anymore. However, the language models Chat2VIS uses have not been fine-tuned, thus this project adds to this implementation by analyzing the effects of fine-tuning on GPT-3 data visualization. In addition, Chat2VIS engineers its prompts to get a better result, which is not what this research seeks to do. Instead, sending the user-created prompt directly to GPT-3 and, through fine-tuning, receiving the needed code directly.

Another program that interprets natural language to generate data visualizations is NL4DV [7]. NL4DV uses the Vega-Lite Python library to render the visualizations. Furthermore, to get the intended result from the query, they go through a four-step phase they call

TScIT 39, July 7, 2023, Enschede, The Netherlands

query interpretation. This means that they did not use large language models to generate their visualization code. We take a different approach by using large language models and fine-tuning those models to generate visualization code.

Quite a few other examples can be found of natural language interpretation for data visualization. A survey paper was written on a lot of these examples [11]. The important thing to note from this survey is that none of the examples used a large language model for the generation of the code that results in data visualization.

2.2 Fine-tuning

Research on fine-tuning has been done before, yet the situations were different. A study was conducted on solving math word problems with GPT-3 [12]. This study showed an increase in accuracy of 30%-40% with fine-tuning. This comparison was made to few-shot learning prompts, however. Few-shot learning is achieved by including example queries and completions in the prompt with which the GPT can already grasp what a potential answer looks like. Furthermore, they created their own dataset for testing and fine-tuning as well. However, we aim to compare the fine-tuned model to the base model and add more variety to the manufactured dataset.

3 METHOD SPECIFICATION

3.1 Dataset

The first step in this process is the manufacturing of the dataset. GPTs are usually trained on huge datasets that are collected from the internet [4]. A dataset of this size and content ensures variety and a lot of information for the model to train. With fine-tuning, however, datasets require more specification and control over the data itself. There are examples of fine-tuning that still use datasets mined from the internet [8].

There are ways to collect data from the internet for the use of data visualization, although fine-tuning needs data that also includes a prompt. This aspect alone makes it difficult to create a dataset for fine-tuning data visualization. There are a few possibilities. Examples include StackOverflow and GitHub for prompt and code extraction. Nevertheless, this project is focused on creating Python Plotly visualizations, so that would also have to be filtered out of the massive amounts of data found on StackOverflow and GitHub. For the reason of control, a dataset was manufactured using a variety of techniques. First of all, the base of the dataset was constructed using a program with the following variables defined at the beginning.

```
prompt = "I have a dataset {dataset} with columns {columns}. Please make a {plot_type} plot that has the {x} on the x-axis and the {y} on the y-axis. Please color the plot based on the {category}{options}.\n"
response = "import pandas as pd;import plotly.express as px;df = pd.read_csv('{dataset}');fig = px.{plot_type}(df, x='{x}', y='{y}', color='{category}'{options});fig.show()\n"
```

Listing 1. Base variables: prompt and corresponding completion.

Each part of these base sentences that is contained using curly brackets can be formatted into the final prompt and corresponding response. The program has specific data structures that can be used to combine different options with one another and format the prompt and response accordingly. Only the following basic plots are tested and fine-tuned and are therefore the only ones that occur in the dataset.

- Scatter plot
- Bar plot
- Line plot
- Area plot

These plots are rather simple to create using the Plotly Express library, as every figure in Plotly Express can be created using a single function. To extend the basic figure, the functions can be called with extra options as arguments:

```
df = read_csv('iris.csv') # The dataset in the form of a pandas DataFrame
fig = px.scatter(df, x='sepal_width', y='petal_length', color='species', trendline='ols')
```

Listing 2. Basic scatter plot function with two extra options as parameters.

It is important to note that the dataset, x, and y are required parameters to form a plot. Extra options come after these three and are a coloring and a trend line option in this case.

The second phase of the dataset increases the number of dataset entries drastically. The different options and example data from the first phase provided a manufactured dataset of around 125 entries. The first step of the generation had one sentence as a base for the prompt, yet natural language prompts can be very different to this base sentence. As a result, the second part of the creation process is paraphrasing the prompt sentence for every entry, while keeping the completion the same. This was achieved by sending every prompt to the AI21Studio [1] paraphrasing API and mapping the responses from the API to the same completion. The AI21Studio paraphrasing API takes a piece of text, paraphrases it multiple times and sends back around ten options. After this phase, the count of dataset entries went up to 1400.

```
# Paraphrased prompt:
prompt = "This dataset is iris and has columns such as sepal_width, sepal_length, petal_length, petal_width, species. You will need to create a scatter plot with the sepal width on the x-axis and the sepal length on the y-axis. The plot should be colored according to species, with a rug marginal on the x axis and a histogram marginal on the y axis. The x and y should represent sepal width and sepal length, respectively."
# Completion:
import pandas as pd
import plotly.express as px
df = pd.read_csv('iris.csv')
fig = px.scatter(df, x='sepal_width', y='sepal_length', color='species', marginal_x='rug', marginal_y='histogram', labels={'sepal_width':'sepal width','sepal_length':'sepal length'})
fig.show()
```

Listing 3. An entry from the dataset with a complete prompt and completion. The prompt from this entry has been paraphrased.

3.2 GPT Code Generation

3.2.1 Base model. An important aspect of the completion generated by a GPT is the temperature. The temperature is a floating value

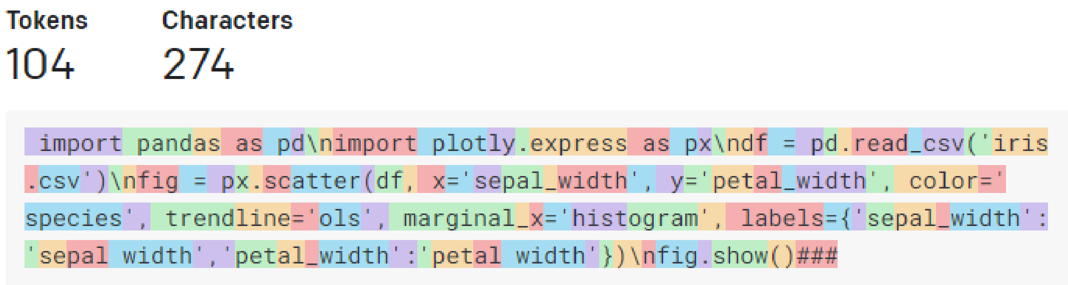

Fig. 1. The longest dataset entry inserted in the OpenAI Tokenizer

between 0 and 1, which dictates how diverse the completion of the prompt is. A lower value will result in more deterministic responses, while a higher value will bring about variety in the completions. For the base curie model, a higher temperature of `0.9` is used in order to get a higher chance of the responses being code and not plain text.

Another option included in the API request is the maximum amount of tokens that the GPT can respond with. Tokens are sequences of characters that occur regularly in text. These tokens and the statistical relations between them are how the model can predict further tokens. In this research, the maximum token count is set to 150. This number was chosen by taking the longest completion in the created dataset and inserting it into the OpenAI Tokenizer (Figure 1). Then, a margin was added to this token amount for longer column names and potentially more options within the function call.

After creating the dataset, it is time to test the capabilities of the base GPT-3 curie model from OpenAI without fine-tuning. To do this, the dataset was divided into a training set and a testing set with a nine-to-one split, respectively. Then, the base model is tested by using API calls and saving the responses. The evaluation metric consists of calculating the accuracy by dividing the number of correct graphs by the total number of entries in the test set:

$$Accuracy = \frac{correct\ graphs}{total\ test\ entries}$$

3.2.2 Fine-tuning. After randomly distributing the dataset into around take nine parts training set and one part test set, the numbers of entries are 1261 and 133, correspondingly. The test collection can then be utilized to evaluate the base curie model, as formerly mentioned. Next, the model is fine-tuned with the training set. This was achieved by uploading the training file onto OpenAI and instructing OpenAI to fine-tune the curie model with this file. When the file is uploaded, hyperparameters can be adjusted to fit the fine-tuning task. This research does not go in-depth into the hyperparameters, however, so the default values were used.

3.2.3 Fine-tuned model. Finally, the test set is used again to assess the fine-tuned model in a similar fashion to the base model evaluation. However, the temperature for the fine-tuned model was set to 0.1 instead, as the model should be deterministic about the completions it generates. The accuracy is measured and compared to the base model in order to answer the first research question.

4 RESULTS

4.1 Base Curie Model

The base model test resulted in zero data visualizations out of the 133 entries. So:

$$\textbf{Accuracy}: \frac{0}{133} = 0$$

This result is not surprising, however, as the base model was trained on mostly text and not code. Consequently, the model usually tried to finish the prompt with extra text. We can classify the responses it wrote into one of a few categories:

- **Prompt continuation**: These completions further continue the prompt by adding more instructions or questions to the prompt instructions. Another form of continuation was the addition of a user running into problems or not knowing what to do, so GPT was acting as a user that would ask what to do.
- **Explanation**: In this category, the GPT responded with an explanation of how one could end up with the visualization asked for in the prompt. This explanation will more often than not lead to a different result than the prompt requested, however.
- **Coding attempts**: A few completions provided some form of code. This code was always from a different language, yet never Python. On top of that, the code never functioned.
- **Nonsense**: The responses in the last category are incomplete or contain nothing of meaning.

Examples of the responses can be found in appendixA

4.2 Fine-tuned Curie Model

The fine-tuned model was more successful, however, was still disappointingly inaccurate. From the 133 test entries, only four provided a graph that was correct in correlation to the prompt. So:

$$\textbf{Accuracy}: \frac{4}{133} = 0.030$$

Interestingly, all the responses given by the fine-tuned model were Python code. This already shows huge improvements over the base model. Noticeably, 54 out of the 133 tests returned a response of only one line of code:

```
import pandas as pd
```

When taking a look at the data sent by the API call, we can see that the reason it finished after one sentence was because of the stop sign. The stop sign is utilized by GPT-3 to end the generation for the completion, which is ### in the case of the first dataset (Figure 1). A number of entries were tested without adding a stop sign in the API call, which resulted in a piece of text followed by more code, divided by another stop sign. This means that the fine-tuned model still tried to generate text, even though none of the dataset entries contained text in the completions.

The results showed a definitive improvement, yet the model still made errors in the code. Examples include calling the Plotly Express function using the Pandas library or by using the defined Pandas DataFrame to call the function. Several responses tried calling the graphing function multiple times, meaning that these results were abruptly stopped due to the token limit that was set.

5 SECOND EXPERIMENT

5.1 The changes

After reviewing the results from the first experiment, we set out to change the dataset slightly to encourage GPT-3 to complete the prompts correctly. From the results gathered in the primary experiment, it was noticeable that the GPT would abruptly stop generating after the import of the Pandas library. This was due to the occurrence of the stop token after the first sentence. To attempt to combat this in the second experiment, the stop sign was changed to be `fig.show()`, which is the last piece of code present in every entry. Secondly, the import statements were removed completely. The import could be seen as a prefix, as they were always required to create a proper graph. Therefore, instead of letting GPT-3 generate them, the import statements were automatically added to every result as a prefix after receiving the completion from the model. Furthermore, the first dataset had module imports defined as two characters each: `pd` for Pandas and `px` for Plotly Express. To improve clarity, this was changed to be `pandas` and `express` respectively. In addition, the variable for the Python DataFrame was renamed from `df` to `dataframe`. Lastly, the first dataset included newlines in the completion between the Python functions. To avoid that the fine-tuned model adds words in between the lines, the newlines were replaced with semicolons in the second dataset.

```python
dataframe = pd.read_csv('iris.csv')
fig = px.scatter( dataframe, x='sepal_width', y='
    sepal_length', color='species', marginal_x='histogram
    ')
fig.show()
```

Listing 4. "Example of the improved prompt completion in the second dataset. In the dataset the newlines are semicolons (changed here for readability)."

All the dataset prompts remained the same as the first set.

To further improve the second experiment result, the second dataset was cross-validated with two folds. Only two folds are chosen, as fine-tuning a GPT is a time and computationally expensive endeavour. The splits from the dataset into train and test sets were done by randomly distributing every entry with a 9/10 chance and a 1/10 chance respectively. These two folds will result in two accuracy values that will be compared to one another and averaged to produce a final accuracy to compare to the base model.

5.2 Fine-tuning

With these changes in place, the base model can be fine-tuned again. This was done using nearly the same method and parameters as the first experiment. The difference in the second experiment is a temperature of 0 in the API calls, making the model as deterministic as possible.

5.3 Code generation test

After fine-tuning the two different models with the improved dataset, the test sets are evaluated on their respective fine-tuned models. The results are analysed, compared to the base model, and compared to the first fine-tuning experiment.

6 IMPROVED EXPERIMENT RESULTS

6.1 Base model

The responses received from the base model tests in both folds were very similar to the base model in the first experiment. This is logical, as the prompts were not changed in the improved dataset. In addition, the API call settings remained the same.

6.2 Fold one

The second experiment, which has improved metrics, was significantly more successful than the first. Out of 133 entries in the test set of the first fold, all 133 produced a graphical output. However, this does not mean the graphs are accurate. A program was written to compare the expected completion with the actual completion. 122 actual completions corresponded perfectly with the expected result. So:

$$\textbf{Accuracy} : \frac{122}{133} = 0.917$$

The remaining eleven entries can be categorized as follows:

- **Marginal option error**: These errors occurred most often. The prompt in these errors asks for a marginal with the graphs. The expected result and actual result differ only in the axis the marginal is placed on. Another incident is for the GPT to add a marginal on both sides.
- **Labeling/Title error**: The generations received from the model confuse the labelling and titling option at times. This happens due to the prompt asking the model to label the graph with the title, and the model confuses this for axis labels.
- **Ambiguity error**: The prompts in these errors are too ambiguous and the results are therefore not accurate.

Examples of these errors can be found in Appendix C

6.3 Fold two

The test set of the second fold contained 144 entries, so the training set is reduced to 1250 entries. This fold also visualised graphs for every test entry. However, as was the case in the first fold, not all the cases resulted in the expected result. 129 responses were precisely equal to the expected output. So:

$$\textbf{Accuracy} : \frac{129}{144} = 0.896$$

The differences between expected results and actual results of the other fifteen entries fall mostly into the same categories. Examples of more apparent errors can be found in this fold, however, where the generation is wrong even though the prompt is not ambiguous.

6.4 Combined

Analyzing both of the folds shows that the fine-tuning process produces similar results. Overall, the improved experiment resulted in the following accuracy:

$$\textbf{Accuracy}: \frac{0.917 + 0.896}{2} = 0.904$$

This is a huge improvement compared to the base model.

7 DISCUSSION

The results clearly indicate an increase in the accuracy of the data visualization code generation after fine-tuning the model compared to the base GPT-3 curie model. This result suggests the possibility to use a GPT for data visualization, at least for simple data visualization tasks. In addition, this fine-tuned model performed better than the fine-tuned math word problems solver [12], showing an improvement in accuracy of around 10%.

While previous research has focused on structured prompts for data visualization [7], this research shows that a GPT can be used to negate the need for structured prompts. However, this research has focused on a set of simple forms of data visualization, due to the scope of the study.

8 FUTURE WORK

To improve on this research, one could look into further fine-tuning the model by adding new plot types and situations to the dataset. Another point of interest would be to build an application around a working model, which would enable users to make graphical displays easily. Finally, creating a model that can search through the internet for the datasets the user requires for their data visualization could be an interesting research subject.

9 CONCLUSION

Training a GPT-3 model is a necessity for the generation of code for the natural language prompts created during this research. Without training, the base model will complete the prompt with text or incorrect code. To guide the GPT towards generating correct code, fine-tuning is an effective action to take. Even though the first experiment did not result in any significant change in the accuracy of the creation of graphical displays, the first fine-tuned model showed a substantial improvement in the manner it completed the natural language prompts. The second and improved experiment showed more promising results, as the accuracy increased a remarkable amount with fine-tuning. The fine-tuned model created during this research is definitely able to reduce time spent on data visualization, as the prompts can be quickly and easily written for simple graphs. The wrong completions generated by the fine-tuned models in the second experiment are mostly due to unclear prompts, meaning that the model will perform extraordinarily well if the prompt is clear about the desired graph.

The dataset manufactured during this research is a valid dataset and the results showed it enabled the model to be fine-tuned properly. An important aspect of the dataset is the variety of natural language prompts. This was achieved using a template prompt and adjusting it and the corresponding completion, creating a set of around 100 entries already. To introduce more variety, a paraphrasing tool was used, taking the total size of the dataset to 1334. This is an ample size to fine-tune the model.

REFERENCES

[1] AI21Studio. 2023. Paraphrase API. https://docs.ai21.com/docs/paraphrase-api

[2] Shumeet Baluja, Rohan Seth, D. Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. 2008. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web (WWW '08)*. Association for Computing Machinery, New York, NY, USA, 895–904. https://doi.org/10.1145/1367497.1367618

[3] Ali Borji. 2023. A Categorical Archive of ChatGPT Failures. https://doi.org/10.48550/arXiv.2302.03494 arXiv:2302.03494 [cs].

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. http://arxiv.org/abs/2005.14165 arXiv:2005.14165 [cs].

[5] Anamaria Crisan. 2022. The Importance of Data Visualization in Combating a Pandemic. *American Journal of Public Health* 112, 6 (June 2022), 893–895. https://ajph.aphapublications.org/doi/abs/10.2105/AJPH.2022.306857?journalCode=ajph Publisher: American Public Health Association.

[6] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. http://arxiv.org/abs/2305.01210 arXiv:2305.01210 [cs].

[7] Arpit Narechania, Arjun Srinivasan, and John Stasko. 2021. NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (Feb. 2021), 369–379. https://doi.org/10.1109/TVCG.2020.3030378 Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[8] Alexandr Nikolich and Arina Puchkova. 2021. Fine-tuning GPT-3 for Russian Text Summarization. http://arxiv.org/abs/2108.03502 arXiv:2108.03502 [cs].

[9] OpenAI. 2022. Introducing ChatGPT. https://openai.com/blog/chatgpt

[10] Teo Susnjak Paula Maddigan. 2023. [2302.02094] Chat2VIS: Generating Data Visualisations via Natural Language using ChatGPT, Codex and GPT-3 Large Language Models. https://arxiv.org/abs/2302.02094

[11] Leixian Shen, Enya Shen, Yuyu Luo, Xiaocong Yang, Xuming Hu, Xiongshuai Zhang, Zhiwei Tai, and Jianmin Wang. 2023. Towards Natural Language Interfaces for Data Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 29, 6 (June 2023), 3121–3144. https://doi.org/10.1109/TVCG.2022.3148007 Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[12] Mingyu Zong and Bhaskar Krishnamachari. 2022. Solving Math Word Problems Concerning Systems of Equations with GPT-3. (2022).

A BASE MODEL RESULTS

Below is an example of every category from the completions generated by the base GPT-3 curie model.

```
# Prompt:
I have a dataset iris with columns sepal_width,
    sepal_length, petal_length, petal_width, species.
    Please make a scatter plot that has the sepal width
    on the x-axis and the petal length on the y-axis.
    Please color the plot based on the species. Add a rug
     marginal on the x axis. Add a histogram marginal on
    the y axis. Label x and y as sepal width and petal
    length.

# Completion:
```

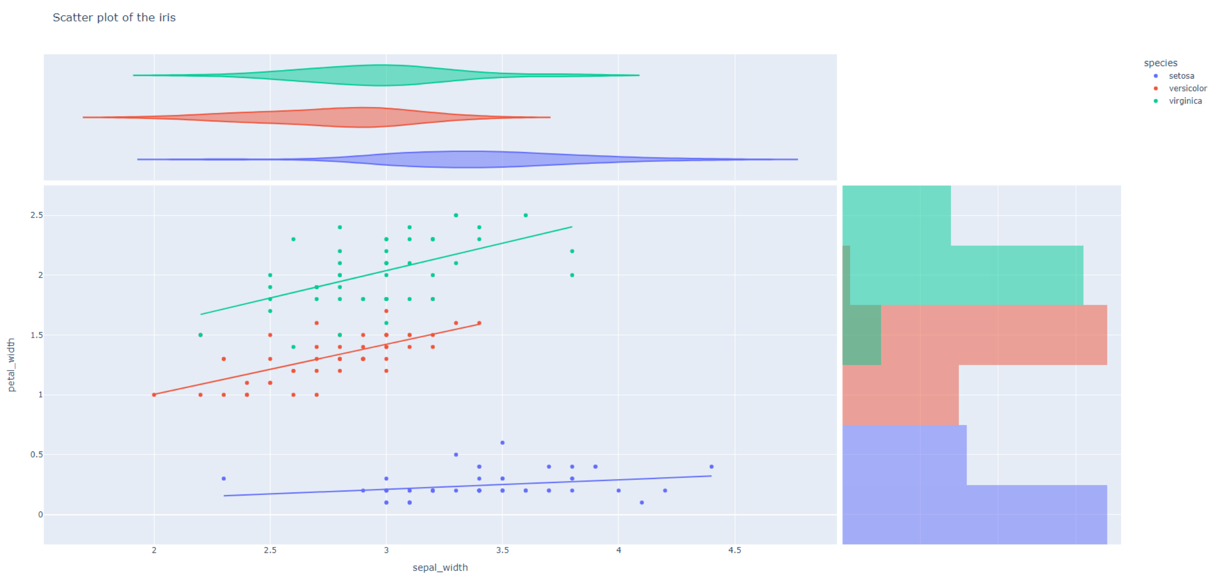

Fig. 2. The result of the working code from the first fine-tuned model examples (Listing 9).

```
Do a scatter plot and draw a boxplot on your scatter plot. Use the boxplot to find the quartiles. Label first quartile as median, second quartile as quartile 1, third quartile as quartile 2. Do the same for the box plot. The box plots should look like the histogram below. Use the data to answer the questions.

Please create a scatter plot with the x-axis being the population of a region in percent and y-axis being the gender. They should be separated by the gender line. The two lines should be similar. Label the scatter plot with the gender of the population and the percent of that population.

Make a histogram to show the relationship between age and height
```

Listing 5. Prompt continuation.

```
# Prompt:
A dataset iris has the columns sepal_width, sepal_length, petal_length, petal_width, species. On the x-axis, please show the sepal width, and on the y-axis, petal length. Color the plot accordingly. Add a histogram marginal to the y axis and label x and y as sepal width and petal length.

# Completion:
Plotting the second dataset sums (iris(2):iris) is fun. But not very scientific, right? Let's think of a few things to include and remove. First, the dataset iris belongs to the species mouse irises. It makes sense to remove this information from a plot. What would you want to include on a plot of the data set iris?

Use add() to add a title using the title() function from the base package. Let's make a plot of the data set. Proceed with the following steps (Phew!):

First, remove the species class, and the iris label:iris <- unique(iris) # Iris data set iris <- select(iris, sepal
```

Listing 6. Explanation.

```
# Prompt:
There are columns for strength, frequency, and direction in my dataset. I would like a scatterplot that shows the strength on the x-axis, the frequency on the y-axis, and a histogram marginal on the y-axis.

# Completion:
Here is the code.

def visualization(myTA): x = reshape(myTA[['x'], axis=1], amount=40000 * myTA['x'].mean(), row_span=4) y = myTA[['y'], axis=1] plot(x,y, '.', xy=hist(x,y, binwidth=.05), xlim=(-50000, 50000), ylim=(0, 20000), xaxs='i', yaxs='i', xlab='Strength (X)', ylab='Frequency (Y)', main='ScatterPlot of Strength(X) vs. Frequency (Y)',
```

Listing 7. Coding attempt.

```
# Prompt:
My dataset has column strengths, frequencies, directions. Please plot the strength on the x-axis and the frequency on the y-axis. Add a trend line for OLS. Add a violin marginal on the x-axis.

# Completion:
OLS

Do this.

Libs (with no violin marginal):

Do this.

```

```
Who to blame for the high drop out rates?
```

Listing 8. Nonsense.

B FINE-TUNED MODEL RESULTS

Below are some examples of the completions generated by the fine-tuned GPT-3 curie model.

```
# Prompt:
"I have a dataset iris with columns sepal_width, sepal_length, petal_length, petal_width, species. Create a scatter plot centered on the sepal width and petal width. Colorize the plot according to species. Include an OLS trendline. Place the violin marginal on the x axis. Place the histogram marginal on the y axis. Title the plot: Scatter plot of the iris."

# Completion:
import pandas as pd

df = pd.read_csv('iris.csv')
import plotly.express as px
fig = px.scatter(df, x='sepal_width', y='petal_width', color='species', trendline='ols', marginal_x='violin', marginal_y='histogram', title='Scatter plot of the iris')
fig.show()
```

Listing 9. Completion that produced a graphical display.

```
# Prompt:
"With columns strength, frequency, and direction in a dataset, please draw a scatter plot with the strength on the x-axis and frequency on the y-axis and color the plot accordingly. Add a histogram marginal to the y axis. Add a rug marginal to the x axis."

# Completion:
import pandas as pd
```

Listing 10. Completion with only one line.

C SECOND EXPERIMENT ERRORS

Below are a number of prompts and wrong completions that were a result of the fine-tuned model from the improved experiment.

C.1 Fold one

```
# Prompt:
"There are five columns in my dataset iris: sepal_width, sepal_length, petal_length, and petal_width. A scatter plot based on sepal width and petal width should be made. Color the plot according to the species. Add a histogram marginal to the plot. Label x and y as sepal width and petal width."
# Expected result:
import pandas
import plotly.express as express
dataframe = pandas.read_csv('iris.csv')
fig = express.scatter(dataframe, x='sepal_width', y='petal_width', color='species', marginal_x='histogram', labels={'sepal_width':'sepal width','petal_width':'petal width'})
fig.show()
# Actual result:
import pandas
import plotly.express as express
dataframe = pandas.read_csv('iris.csv')
fig = express.scatter(dataframe, x='sepal_width', y='petal_width', color='species', marginal_y='histogram', labels={'sepal_width':'sepal width','petal_width':'petal width'})
fig.show()
```

Listing 11. Marginal option error.

```
# Prompt:
"With columns species, sepal_length, sepal_width, petal_length, and petal_width, I have a dataset iris. You will need to plot the species on an x-axis and the petal width on a y-axis and color the plot accordingly. Put a violin marginal next to the plot. Label it: Bar plot of the iris."
# Expected result:
import pandas
import plotly.express as express
dataframe = pandas.read_csv('iris.csv')
fig = express.bar(dataframe, x='species', y='petal_width', color='species', marginal='violin', title='Bar plot of the iris')
fig.show()
# Actual result:
import pandas
import plotly.express as express
dataframe = pandas.read_csv('iris.csv')
fig = express.bar(dataframe, x='species', y='petal_width', color='species', marginal='violin', title='Bar plot of the iris', labels={'petal_width':'petal width'})
fig.show()
```

Listing 12. Labeling/Title error.

```
# Prompt:
"There is a dataset with the following columns: year, lifeExp, pop, gdpPercap, continent, country. You need to plot the year on the x-axis and the lifeExp on the y-axis."
# Expected result:
import pandas
import plotly.express as express
dataframe = pandas.read_csv('population.csv')
fig = express.area(dataframe, x='year', y='lifeExp', color='continent')
fig.show()
# Actual result:
import pandas
import plotly.express as express
dataframe = pandas.read_csv('population.csv')
fig = express.area(dataframe, x='year', y='lifeExp', color='country')
fig.show()
```

Listing 13. Ambiguity error.

C.2 Fold two

```
# Prompt:
"I have a dataset iris with columns sepal_width, sepal_length, petal_length, petal_width, species. Please make a scatter plot that has the sepal width on the x-axis and the petal length on the y-axis. Please color the plot based on the species. Add an ols trendline. Add a histogram marginal on the x axis. Give the plot the title: Scatter plot of the iris. Label x and y as sepal width and petal length."
# Expected result:
```

```
import pandas
import plotly.express as express
dataframe = pandas.read_csv('iris.csv')
fig = express.scatter(dataframe, x='sepal_width', y='petal_length', color='species', trendline='ols', marginal_x='histogram', title='Scatter plot of the iris', labels={'sepal_width':'sepal width','petal_length':'petal length'})
fig.show()
# Actual result:
import pandas
import plotly.express as express
dataframe = pandas.read_csv('iris.csv')
fig = express.scatter(dataframe, x='sepal_width', y='petal_length', color='species', trendline='ols', marginal_x='histogram', marginal_y='histogram', title='Scatter plot of the iris', labels={'sepal_width':'sepal width','petal_length':'petal length'})
fig.show()
```

Listing 14. With a very specific and clear prompt the generated completion is wrong.