(ADA) Homework 1: Scoring the Language Model Olympics

By the end of this homework, we expect you to be able to:

- Load data and handle data using pandas;
- Navigate the documentation of Python packages by yourself;
- Filter and tidy up noisy real-world datasets;
- Aggregate your data in different (and hopefully helpful) ways;
- Create meaningful visualizations to analyze the data;
- Communicate your findings in a clear and concise manner

Important Dates.

- Homework release: Fri 04 Oct 2024
- Homework due: Sat 18 Oct 2024, 23:59
- Grade release: Mon 04 Nov 2024

Some rules

- You are allowed to use any built-in Python library that comes with Anaconda. If you want to use an external library, you may do so, but must justify your choice.
- Make sure you use the data folder provided in the repository in read-only mode. (Or alternatively, be sure you don't change any of the files.)
- Be sure to provide a concise textual description of your thought process, the assumptions you made, the solution you implemented, and explanations for your answers. A notebook that only has code cells will not suffice. To avoid confusion: use short comments for longer code answers.
- For questions containing the /Discuss:/ prefix, answer not with code, but with a textual explanation (in markdown).
- Back up any hypotheses and claims with data, since this is an important aspect of the course.
- Please write all your comments in English, and use meaningful variable names in your code. Your repo should have a single notebook (plus the required data files) in the master/main branch. If there are multiple notebooks present, we will not grade anything.
- We will not run your notebook for you! Rather, we will grade it as is, which means that only the results contained in your evaluated code cells will be considered, and we will not see the results in unevaluated code cells. Thus, be sure to hand in a fully-run and evaluated notebook. In order to check whether everything looks as

intended, you can check the rendered notebook on the GitHub website once you have pushed your solution there.
- In continuation to the previous point, interactive plots, such as those generated using the 'plotly' package, should be strictly avoided! Make sure to print results and/or dataframes that confirm you have properly addressed the task.

A Note on using Language Models (LMs)

If you try hard enough, you will likely get away with cheating. Fortunately, our job is not to police, but rather to educate! So, please consider the following:

- Presumably, you are taking this course to learn something! LMs are not always right (they often fail in silly ways). This course should prepare you to detect when they are wrong!
- Some of the TAs on this course literally published many works on detecting machine-generated text.

Context

Context AI is booming! Newspapers, influencers, and your relatives all agree that AI is important. But while almost everyone agrees that AI is the future, much is unclear about what that future looks like...

Freshly graduated from the EPFL, you are hired by the Swiss government to advise on a large-scale "AI integration" initiative code-named **"NEUTRALITY"** (Navigating Efficient Upgrades Through Robust Artificial Learning Integration Techniques Yearly). Convinced by the stunning progress in language modeling, the government would like to battle the growing shortages in the education sector by using LMs. Your job description: investigate which LMs might be best suited!

You are given the results of three LMs on the "Massive Multitask Language Understanding (MMLU)" dataset to compare. This famous dataset consists of 57 subjects with multiple-choice questions, covering diverse subjects like mathematics, computer science, history, and law. Most providers of state-of-the-art LMs use this dataset to showcase the versatility of their latest models. Unfortunately, Horta-Ribeiro, the intern responsible for collecting the results, didn't take EPFL's famous ADA course. As a result, the collected datasets are slightly corrupted.

A very brief primer on Language Models

Language models (LMs) are sophisticated statistical models designed to understand and generate human-like text. At their core, LMs are trained to predict the most likely continuation of a given input text. For example, given the input "The cat sat on the," an

LM might predict "mat" as a likely continuation. LMs are trained on vast text samples from various sources, including books, websites, and social media. This extensive training allows them to capture patterns and relationships in language, enabling them to generate coherent and contextually appropriate text across a wide range of topics and styles.

While LMs can produce text that appears to be written by intelligent humans, it's important to note that their capabilities can diverge from human intelligence in unexpected ways. They may sometimes generate factually incorrect information or struggle with complex reasoning tasks.

Two key concepts in understanding LMs are:

1. **Tokens**: LMs process text using "tokens" rather than individual characters. Tokens can be words, parts of words, or punctuation marks. For example, the sentence "I love AI!" might be tokenized as ["I", "love", "AI", "!"]. Tokenization is the first step in both training and using an LM.
2. **Context**: The input text provided to an LM is called the "context." This context informs the model's predictions or generations. A longer or more specific context often leads to more accurate and relevant outputs.

See: Wikipedia entry on language models

Files for this assignment

This assignment is divided into three tasks, each of which should bring you a step closer to providing a recommendation toward project NEUTRALITY's objectives:

- **Task 1**: Inspecting the results and getting your first model ranking
- **Task 2**: Inspecting the underlying data used to generate the results for possible biases
- **Task 3**: Learning about tokens and providing a final recommendation

```
📁 PROJECT_NEUTRALITY
│
├── 📄 analysis.ipynb (the file you're currently reading!)
│
└── 📁 data
    ├── 📁 task_1
    ├── 📁 task_2
    └── 📁 task_2.5
```

In [1]:

```
# some basic imports
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
from scipy.stats import ttest_ind
```

Task 1 (18 points): What's in an average anyway?

The files needed to complete task 1 can be found in the folder " `data/task_1/` :

```
task_1/
│
├── mmlu_data/
│   └── test.csv
│
└── lm_scores/
    ├── lm_X.csv
    ├── lm_Y.csv
    └── lm_Z.csv
```

We will start by loading, (manually) inspecting, and cleaning the data. Although it doesn't seem "glamorous" (nor is it particularly fun...) - manually inspecting data is extremely important! In fact, it's one of the few things most AI and Data Science researchers agree on :). Next, we will take a first pass on ordering our Olympic podium between three LMs.

1.1 (1 pt)

Load the subfiles contained in the `mmlu_data` and `lm_scores` folders into separate dataframes:

- `df_test`
- `df_x`
- `df_y`
- `df_z`

for each, print their sizes.

In [2]:
```
df_test = pd.read_csv('data/task_1/mmlu_data/test.csv')

f = 'data/task_1/lm_scores/'
df_x = pd.read_csv(os.path.join(f, 'lm_X.csv'))
df_y = pd.read_csv(os.path.join(f, 'lm_Y.csv'))
df_z = pd.read_csv(os.path.join(f, 'lm_Z.csv'))

print('df_test: ', df_test.shape)
print('df_x: ', df_x.shape)
print('df_y: ', df_y.shape)
print('df_z: ', df_z.shape)
```

```
df_test:  (14042, 8)
df_x:  (13882, 2)
df_y:  (13978, 2)
df_z:  (13923, 2)
```

Grading Instruction. [1 pt]

[1 pt] Correctly loaded data frames and printed sizes.

1.2 (4 pt)

Unfortunately, LMs don't always output the format we want. In the column `result`, the value should be one of A, B, C, or D.

A. For each of the LM score dataframes, use a `value_counts()` operation and print the results.

B. /Discuss:/ Inspect the results and describe the types of answer formats you see. Besides the "expected" case, you should be able to find at least four unexpected formats.

In [3]:
```
# A
dfs = [df_x, df_y, df_z]
for df, n in zip(dfs, ['X', 'Y', 'Z']):
    print(f'\nLM {n}:\n', df['result'].value_counts())


# B
# one strategy to find different cases, is to add a column counting the numb
# then, sample repeatedly.
df_temp = pd.concat([df_x.copy(), df_y.copy(), df_z.copy()]).reset_index(drc
df_temp['result_len'] = df_temp['result'].apply(lambda x: len(str(x)))
df_temp[['result', 'result_len']].sample(5)

# case 1: "Answer: {A, B, C, D}"
# case 2: "None / NaN"
# case 3: "{question}, so the answer is {A, B, C, D}"
# case 4: None of the above
# case 5: Not Sure
# case 6: "{A, B, C, D} "
```

```
LM X:
 result
A
2733
A
1657
B
1412
Answer: A
1398
C
1134

...
judicial activism, so the answer is A
1
creating insurmountable obstacles to the founding of factions, so the answer
is A                                    1
A congressperson who retires to take a position teaching political science a
t a university, so the answer is A       1
David Hume, so the answer is D
1
Brahminic orthodoxy, so the answer is A
1
Name: count, Length: 145, dtype: int64

LM Y:
 result
D
2894
Answer: D
1718
C
1701
B
1240
D
1145

...
Where the energy of interaction between the atoms is at its minimum value, s
o the answer is A       1
leaves more viable offspring than others of its species., so the answer is D
1
A and C only, so the answer is D
1
ADP + P → ATP, so the answer is D
1
Monoclonal antibodies, so the answer is C
1
Name: count, Length: 141, dtype: int64

LM Z:
 result
D
2257
```

```
C
2191
B
2127
A
2060
Answer: D
777

...
omission of a universal suffrage clause, so the answer is D
1
declare war, so the answer is D
1
state and local governments, by means of federal funding, so the answer is B
1
less clearly identified with consistent political ideologies, so the answer
is B       1
Rahit, so the answer is B
1
Name: count, Length: 560, dtype: int64
```

Out[3]:

	result	result_len
2508	Answer: A	9
21802	D	2
4455	Answer: A	9
14295	D	2
32553	B	1

Grading Instruction. [4 pt]

[1 pt] A. correctly printed the `value_counts()` operation

[3 pt] B. identified at least four of the following cases.

(subract 1 point for each case less than four, e.g., 3 cases = 2 points, 2 cases = 1 point, 1 case = 0 points)

1. Single answer letter (A, B, C, D) followed by a space
2. The word `Answer:`, followed by an answer letter (A, B, C, D)
3. The text answer to the question, followed by `so the answer is <A, B, C, D>`
4. Not sure
5. None of the above
6. Empty string or `None`

1.3 (5 pt)

Oh oh... That doesn't look great. Simply dropping all invalid answers seems overly wasteful, yet fixing all of these looks like a mess! Instead, let's focus for now on fixing just those answers of length < 10 characters that require only a single `str.replace()` operation.

For example, if the answer looks like `--A--`, we could fix this by using the following simple function:

```
def clean_answer(s, pattern='-'):
    return str(s).replace(pattern, '')

dirty_answer = '--A--'
clean_answer = clean_answer(dirty_answer)
```

A. Filter the three score dataframes to include only answers with less than 10 characters. Make a deep copy of the dataframes as you filter them.

B. Modify the `clean_answer()` example function to clean the answers in the filtered data frames using the `apply()` functionality. Finally, make sure **all remaining answers are one of `A, B, C, or D`**.

C. /Discuss:/ Compare the sizes of the original and filtered data frames. What do you see? Why might this be a problem?

In [4]:
```
def clean_answer(s):
    s = s.replace(' ', '')
    s = s.replace('Answer:', '')

    return s


# A
original_lens = []
filtered_lens = []
dfs = []
for df in [df_x, df_y, df_z]:
    # store original lengths for question C
    original_lens.append(len(df))

    df = df[df['result'].str.len() < 10].copy(deep=True)
    dfs.append(df)

# B
dfs_c = []
for df in dfs:
    # clean answer
    df['result'] = df['result'].apply(lambda x: clean_answer(str(x)))

    # only keep answers that are either A, B, C, or D
    df = df[df['result'].isin(['A', 'B', 'C', 'D'])].copy(deep=True)
    dfs_c.append(df)
    # store filtered lengths for question C
    filtered_lens.append(len(df))


# C
for ol, fl, n in zip(original_lens, filtered_lens, ['X', 'Y', 'Z']):
    print(f'{n}: old = {ol}, filter = {fl} --> diff = {(fl - ol) / ol: .3f}'

# While X, Y only lose ~3.5% of their answers, Z loses almost 8%. This might
```

```
X: old = 13882, filter = 13436 --> diff = -0.032
Y: old = 13978, filter = 13551 --> diff = -0.031
Z: old = 13923, filter = 12753 --> diff = -0.084
```

Grading Instruction. [5 pt]

[1 pt] A. Correctly filtered the dataframes and made deep copies.

[2 pt] B.

1. [1 pt] Used the following two patterns to filter the data: (1) a single space " ` ` ", (2) " `Answer:` ".
2. [1 pt] Correctly used the `apply()` function on each filtered data frame and ended up with the correct dataframe sizes

[2 pt] C.

1. [1] Correctly compared the filtered and unfiltered data frames: X ~3% difference, Y ~3% difference, Z ~9% difference.
2. [1] Explanation is a variant of: "LM Z loses way more answers than X and Y. This potentially makes comparing them more difficult since they won't be scored on the same answers"

1.4 (3 pt)

Now that our answer columns are nicely formatted, let's take a look at model performance:

A. Both the `MMLU` dataframes and the language model score data frames have the columns `question_id`. For each of the language model score data frames, use an inner join operation with the `df_test` dataframe on the `question_id` column.

B. Add a new column to each of the resulting dataframes called `correct`, that checks if the model's answer in `result` is the same as the expected answer in the column `answer`. Then, print the average score of each model.

In [5]:
```python
# A
dfs = []
for df in dfs_c:
    df = pd.merge(df, df_test, on='question_id', how='inner')
    dfs.append(df)


# B
for df, n in zip(dfs, ['X', 'Y', 'Z']):
    df['correct'] = df['result'] == df['answer']
    print(f'{n}: {df["correct"].mean(): .3f}')
```
```
X:  0.767
Y:  0.746
Z:  0.663
```

Grading Instruction. [3 pt]

[1 pt] A. correctly joined the data frames

[2 pt] B.

- (i) successfully added the `correct` column
- (ii) printed the average accuracy for each model (X: ~77%, Y: ~74%, Z: ~66%)

1.5 (5 pt)

Hmmm, something doesn't seem quite right. Let's investigate how "balanced" this dataset is:

A. For each of the 57 subjects in the MMLU, compare the number of questions answered by each model. Print the subjects for which there is a more than 10% difference.

B. Propose and implement a reasonable way to rebalance the results. (e.g., while throwing away 100% of the results perfectly rebalances the results, it is not reasonable).

C. Finally, print the updated accuracy on the rebalanced data.

hint::

- (A) For a given subject, let model X and model Y have answered 181 and 200 questions respectively. You can consider this a 10% difference from the perspective of X, i.e., (200 - 181) / 181 > 0.10

In [6]:
```python
# A
cats = df_test['subject'].unique()
cats10 = {}

for c in cats:
    temp = []
    for df in dfs:
        qs = len(df[df['subject'] == c])
        temp.append(qs)
    min_q = min(temp)
    max_q = max(temp)
    diff = (max_q - min_q) / min_q

    if diff > 0.10:
        cats10[c] = temp
        print(f'{c}'.ljust(30) + f'diff = {diff: .2f}')


# B
# one way of implementing this is to take the intersection of the different
qs = set()
for df in dfs:
    q_ = df['question_id'].unique()
    if len(qs) == 0:
        qs.update(q_)
    else:
        qs = qs.intersection(q_)

dfs_rebalance = []
for df in dfs:
    df = df[df['question_id'].isin(qs)].copy(deep=True)
    dfs_rebalance.append(df)

# C
for df, n in zip(dfs_rebalance, ['X', 'Y', 'Z']):
    print(f'\n{n}: {df["correct"].mean(): .3f}')
```

```
college chemistry             diff =  0.17
college computer science      diff =  0.17
computer security             diff =  0.13
formal logic                  diff =  0.13
high school geography         diff =  0.11
logical fallacies             diff =  0.13
medical genetics              diff =  0.10
moral disputes                diff =  0.32
moral scenarios               diff =  0.17

X:  0.767

Y:  0.747

Z:  0.662
```

Grading Instructions. [5 pt]

[2 pt] A. Identified categories :

- college chemistry diff = 0.17
- college computer science diff = 0.17

- computer security diff = 0.13
- formal logic diff = 0.13
- high school geography diff = 0.11
- logical fallacies diff = 0.13
- medical genetics diff = 0.10
- moral disputes diff = 0.32
- moral scenarios diff = 0.17

[2 pt] B. There are at least two proposals that are deemed acceptable:

1. Took intersection of all the `question_id` values between the three datasets. (motivation: exact same questions)
2. Ensured each model had a roughly equal number of answers per MMLU category. (motivation: questions are a proxy for category level performance)
3. Another reasonable proposal
4. [1 pt only] throw away all the categories that have > 10% difference and keep the rest.

[1p] C. Show that the updated order is X > Y > Z. The exact reported accuracies might differ based on the implementation.

Task 2 (26 points): What do you mean A > D > B > C...?

Nice work! Having successfully inspected, cleaned, and rebalanced the provided data, you head over to director of the government's NEUTRALITY project. Ms. Sakota is happy with your work so far, but worried that the sloppy intern might have done more undetected damage. To be sure, she orders a new set of evaluations of all models on both MMLU and another dataset.

After cleaning up and rebalancing, you are left with the concatenated score files in the second folder `task_2`:

```
task_2/
│
└── lm_scores_mmlu.csv
│
└── lm_scores_other.csv
```

Each has a new column called `model_name`, which is one of `X, Y` or `Z`.

*NOTE: **only** use data from `task_2` and `task_2_5` for this assignment! The values in `lm_scores_mmlu.csv` will NOT be the same as the dataframes you finished in task 1. This is due to "randomness" or "temperature" in language model inference. This can* *slightly shift around generative results. (Conveniently: it also ensures any mistakes made in Task 1 don't propogate further ;))*

In [7]:
```
# PROVIDED CODE
df_mmlu = pd.read_csv('data/task_2/lm_scores_mmlu.csv')
df_other = pd.read_csv('data/task_2/lm_scores_other.csv')
```

2.1 (4 pt)

Let's explore the new results:

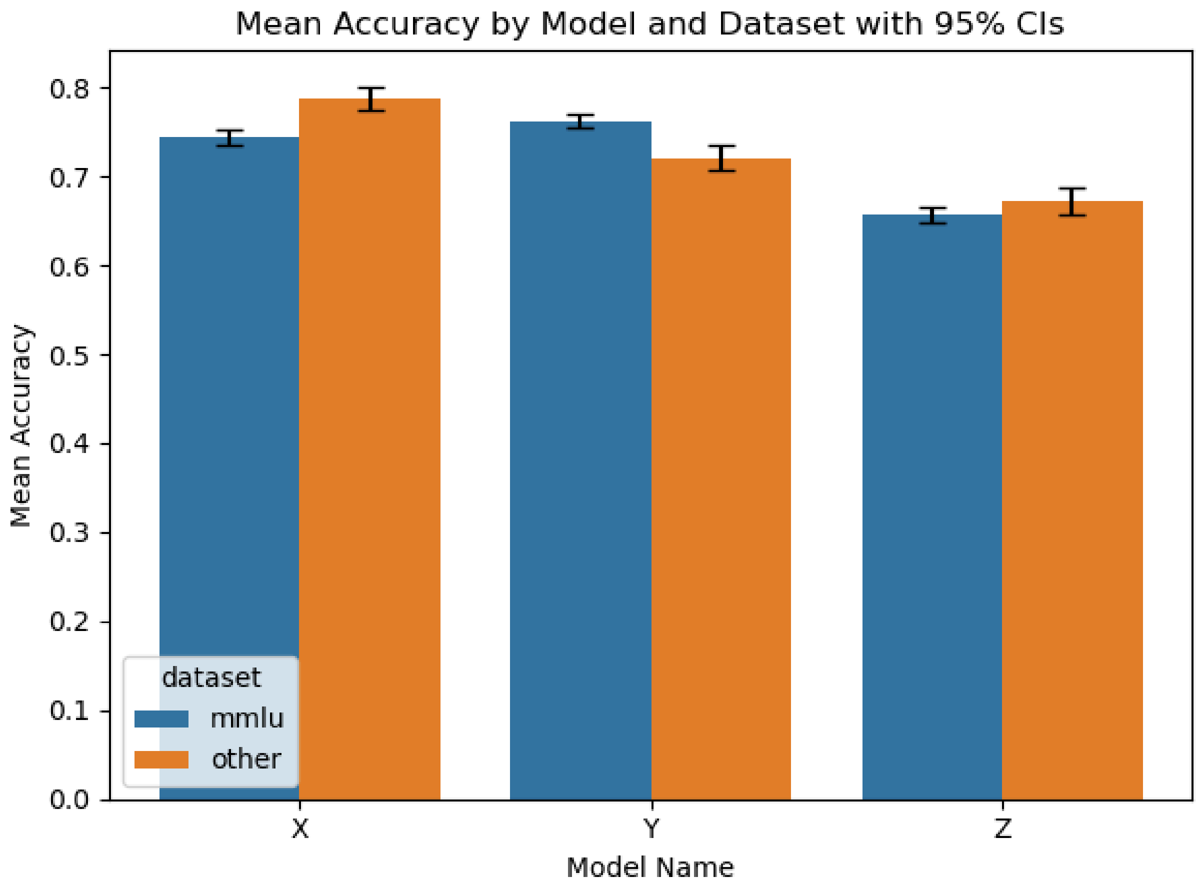
A. Compute the mean accuracy and standard errors of each model on both datasets and print the results.

B. Then, show your results in a bar plot using standard errors with a 95% confidence interval around the mean. Make sure the plot is easy to read and well annotated.

C. /Discuss:/ the plot you created: (i) can you say that one of the models is the best? (ii) is there anything that seems odd?

In [8]:
```
# A
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings

# Suppress the FutureWarning
warnings.filterwarnings("ignore", category=FutureWarning)

df_mmlu['dataset'] = 'mmlu'
df_other['dataset'] = 'other'
df = pd.concat([df_mmlu, df_other]).reset_index()

# Compute mean accuracy and standard errors
results = df.groupby(['model_name', 'dataset'])['correct'].agg(['mean', 'sem
print(results)

ax = sns.barplot(x='model_name', y='mean', hue='dataset', data=results, erro

# Add error bars manually
x_coords = np.arange(len(results['model_name'].unique()))
width = 0.4  # Adjust this value based on your bar width

for i, dataset in enumerate(['mmlu', 'other']):
    dataset_results = results[results['dataset'] == dataset]
    yerr = 1.96 * dataset_results['sem']  # 95% confidence interval
    plt.errorbar(x_coords + (i - 0.5) * width, dataset_results['mean'],
                 yerr=yerr, fmt='none', c='black', capsize=5)

plt.title('Mean Accuracy by Model and Dataset with 95% CIs')
plt.xlabel('Model Name')
plt.ylabel('Mean Accuracy')
plt.tight_layout()
plt.show()

# C
# 1. we see that the order between the two datasets is different:
# -> MMLU:   X > Y > Z
# -> Other:  Y > X > Z
# 2. the error bars overlap between the models
```

```
  model_name dataset      mean       sem
0          X    mmlu  0.743588  0.004038
1          X   other  0.787976  0.006668
2          Y    mmlu  0.761542  0.003941
3          Y   other  0.720936  0.007317
4          Z    mmlu  0.655951  0.004393
5          Z   other  0.671721  0.007660
```



Grading Instructions. [4 pt]

[1 pt] A. Correctly calculated mean accuracy and standard errors of models on each dataset.

[2 pt] B. Plotted a nicely looking plot containing at least:

1. y-label
2. x-label

3. title
4. legend

[1 pt] C. Answers need to include variants of the following two:

1. No, the confidence intervals overlap
2. The ordering of the models differs between the datasets

2.2 (5 pt)

Ms. Sakota has assured you that both datasets contain questions of similar difficulty, so, what could be going on here?

A. What is the distribution of correct answers (A, B, C, D) for each dataset? Create a bar chart to visualize this.

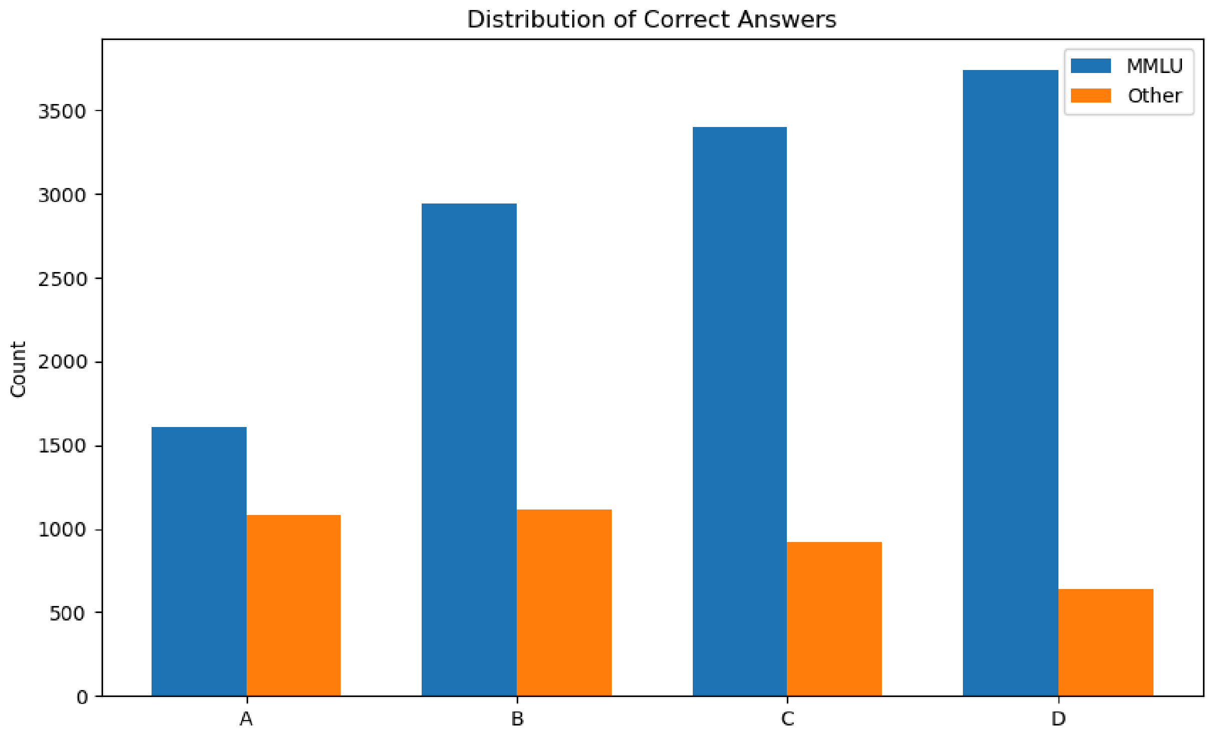
B. Perform a chi-square test at $\alpha = 0.05$, of independence to determine if there's a significant difference in the distribution of correct answers between the two datasets. What do you conclude?

hints:

- for (A), keep in mind that df_mmlu and df_other contain the results of all models, i.e., the `question_id` column is duplicated.
- for (A), take care to clearly annotate the bar chart, e.g., title, y-label, legend.
- for (B), clearly state the null hypothesis and alternative hypothesis
- use the `chi2_contingency` function from `scipy.stats`
- format your results from answer (A) as a 2D array

In [9]:
```
# A
from scipy.stats import chi2_contingency

correct_answers_1 = df_mmlu.drop_duplicates(subset=['question_id'])['answer'
correct_answers_2 = df_other.drop_duplicates(subset=['question_id'])['answer

# Count the occurrences of each answer
count_1 = correct_answers_1.value_counts().sort_index()
count_2 = correct_answers_2.value_counts().sort_index()
print(count_1, count_2)

# Create a bar chart
fig, ax = plt.subplots(figsize=(10, 6))

x = range(len(count_1))
width = 0.35

ax.bar([i - width/2 for i in x], count_1, width, label='MMLU')
ax.bar([i + width/2 for i in x], count_2, width, label='Other')
ax.set_ylabel('Count')
ax.set_title('Distribution of Correct Answers')
ax.set_xticks(x)
ax.set_xticklabels(count_1.index)
ax.legend()

plt.show()

# B.
contingency = pd.concat([count_1, count_2], axis=1).fillna(0)
chi2, p_value, _, _ = chi2_contingency(contingency)

# Interpret results
alpha = 0.05
if p_value <= alpha:
    print("Reject the null hypothesis.")
    print("There is a significant difference in the distribution of correct
else:
    print("Fail to reject the null hypothesis.")
    print("There is not enough evidence to conclude a significant difference
```

```
answer
A    1611
B    2943
C    3403
D    3739
Name: count, dtype: int64 answer
A    1078
B    1116
C     924
D     641
Name: count, dtype: int64
```



```
Reject the null hypothesis.
There is a significant difference in the distribution of correct answers bet
ween the datasets.
```

Grading Instructions. [5 pt]

[2 pt] A. Found the correct distribution of answers for both datasets and visualized

1. (optional, since the answer should be the same) used `drop_duplicates(column=question_id)`
2. Clear plot with (i) y-label, (ii) title, (iii) legend

[3 pt] B. Correct chi-square test and result

1. correctly formulated the null hypothesis and alternative hypothes: 0 = There is not significant difference, Alternative = There is a significant difference
2. found the correct p_value
3. correct conclusion using the alpha value of 0.05

2.3 (7 pt)

Let's dive in deeper:

A. What is language model X's mean accuracy conditioned on the four answer options for each dataset?

B. Compare LM X's performance when the correct answer is "A" between the two datasets. Use a T-test with CI = 0.95. What do you conclude?

C. Compare LM X's performance when the correct answer is "A" vs. "C or D" for each dataset. Use a T-test with CI = 0.95. What do you conclude?

In [10]:
```python
# A
print('MMLU:\n', df_mmlu[df_mmlu['model_name'] == 'X'].groupby('answer')['co
print('\nOther:\n', df_other[df_other['model_name'] == 'X'].groupby('answer'


# B
def do_ttest(a, b):
    t_stat, p_value = ttest_ind(a, b)
    print(f"t-statistic: {t_stat:.4f}, p-value: {p_value:.4f}")
    print("Significant difference" if p_value < 0.05 else "No significant di


print('\nMMLU: A v. Other: A')
X = df_mmlu[(df_mmlu['model_name'] == 'X') & (df_mmlu['answer'] == 'A')]['co
Y = df_other[(df_other['model_name'] == 'X') & (df_other['answer'] == 'A')][
do_ttest(X, Y)
```

```
# C
print('\nMMLU: A v. {C, D}')
X_A = df_mmlu[(df_mmlu['model_name'] == 'X') & (df_mmlu['answer'] == 'A')]['
X_CD = df_mmlu[(df_mmlu['model_name'] == 'X') & (df_mmlu['answer'].isin(['C'
do_ttest(X_A, X_CD)

print('\nOther: A v. {C, D}')
Y_A = df_other[(df_other['model_name'] == 'X') & (df_other['answer'] == 'A')
Y_CD = df_other[(df_other['model_name'] == 'X') & (df_other['answer'].isin([
do_ttest(Y_A, Y_CD)
```

```
MMLU:
 answer
A    0.972688
B    0.799185
C    0.707905
D    0.633592
Name: correct, dtype: float64

Other:
 answer
A    0.974026
B    0.806452
C    0.676407
D    0.603744
Name: correct, dtype: float64

MMLU: A v. Other: A
t-statistic: -0.2106, p-value: 0.8332
No significant difference

MMLU: A v. {C, D}
t-statistic: 25.5564, p-value: 0.0000
Significant difference

Other: A v. {C, D}
t-statistic: 21.6670, p-value: 0.0000
Significant difference
```

Grading Instructions. [7 pt]

[1 pt] A. for each dataset, correct mean accuracies

[3 pt] B.

1. [1 pt] for each dataset, conditioned results on answer = "A" and LM = "X"
2. [1 pt] correct t-test result
3. [1 pt] correct conclusion

[3 pt] C.

1. [1 pt] conditioned on answer being "A" vs "C or D" and LM = "X"

2. [1 pt] correct t-test result
3. [1 pt] correct conclusion

2.4 (2 pt)

What an intriguing finding!

A. Print the mean accuracies conditioned on the correct answer for all LMs for each dataset.

B. /Discuss:/ What do you observe?

```
In [11]: # A
         print('MMLU:\n', df_mmlu.groupby(['model_name', 'answer'])['correct'].mean()
         print('\nOther:\n', df_other.groupby(['model_name', 'answer'])['correct'].me

         # B
         # X seems to have a bias for the first answers A>B>C>D
         # Y seems to have a bias for the last answers D>C>B>A
         # Z does not seem to have a strong bias
```

```
MMLU:
 model_name  answer
X           A         0.972688
            B         0.799185
            C         0.707905
            D         0.633592
Y           A         0.623836
            B         0.688073
            C         0.733470
            D         0.904252
Z           A         0.643079
            B         0.641182
            C         0.669115
            D         0.661139
Name: correct, dtype: float64

Other:
 model_name  answer
X           A         0.974026
            B         0.806452
            C         0.676407
            D         0.603744
Y           A         0.625232
            B         0.663978
            C         0.762987
            D         0.920437
Z           A         0.680891
            B         0.667563
            C         0.662338
            D         0.677067
Name: correct, dtype: float64
```

Grading Instructions. [2 pt]

[1 pt] A. correctly showed the mean accuracy for each LM conditioned on the answer.

[1 pt] B. observed that LM X has first-answer bias, LM Y has a last-answer bias, and LM Z has no answer bias.

2.5 (2 pt)

Concerned with your findings so far, you quickly consult with Ms. Sakota. After thinking it over, Ms. Sakota concludes that more tests are needed. She orders a second round of MMLU results. However, the clever Ms. Sakota thinks of the following twist: while keeping questions fixed, she randomly permutes the position of the correct answer. The new results can be found in the folder `data/task_2_5/`:

```
task_2_5/
│
└── lm_scores_mmlu_shuffle.csv
```

/Discuss:/ Why would Ms. Sakota do this?

Grading Instructions. [2 pt]

[2 pt] To control for confounding: it could be that some answer categories, e.g., "A", were connected to easier questions than those for which the correct answer was "C".

2.6 (4 pt)

Increasingly sceptical of the language models' performance, you read up on proper testing practices. You stumble upon the concept of test-rested stability, which roughly states that:

"*Measurements taken by a single person or instrument on the same item, under the same conditions, and in a short period of time, should have the same results.*"

In our case, we would assume an LM would have the same performance on a given question regardless of the correct answer position. One way of testing this is by using the following metric:

$$\text{test-retest metric} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M}\sum_{j=1}^{M}c_0^i c_j^i,$$

where $c_0^i \in \{0, 1\}$ indicates whether the model answers the i^{th} question correctly (1 if correct, 0 if incorrect). c_j^i indicates whether the model answers the i^{th} question correctly in the j^{th} shuffled version of the answer label content. Finally, M is the total number of shuffles and N is the dataset size.

Task: compute the test-retest metric for each language model using the original `lm_scores_mmlu.csv` file and the new `lm_scores_mmlu_shuffle.csv` file. Using a bar plot, visualize your results by comparing the accuracy of the original `lm_scores_mmlu.csv` and the test-retest scores.

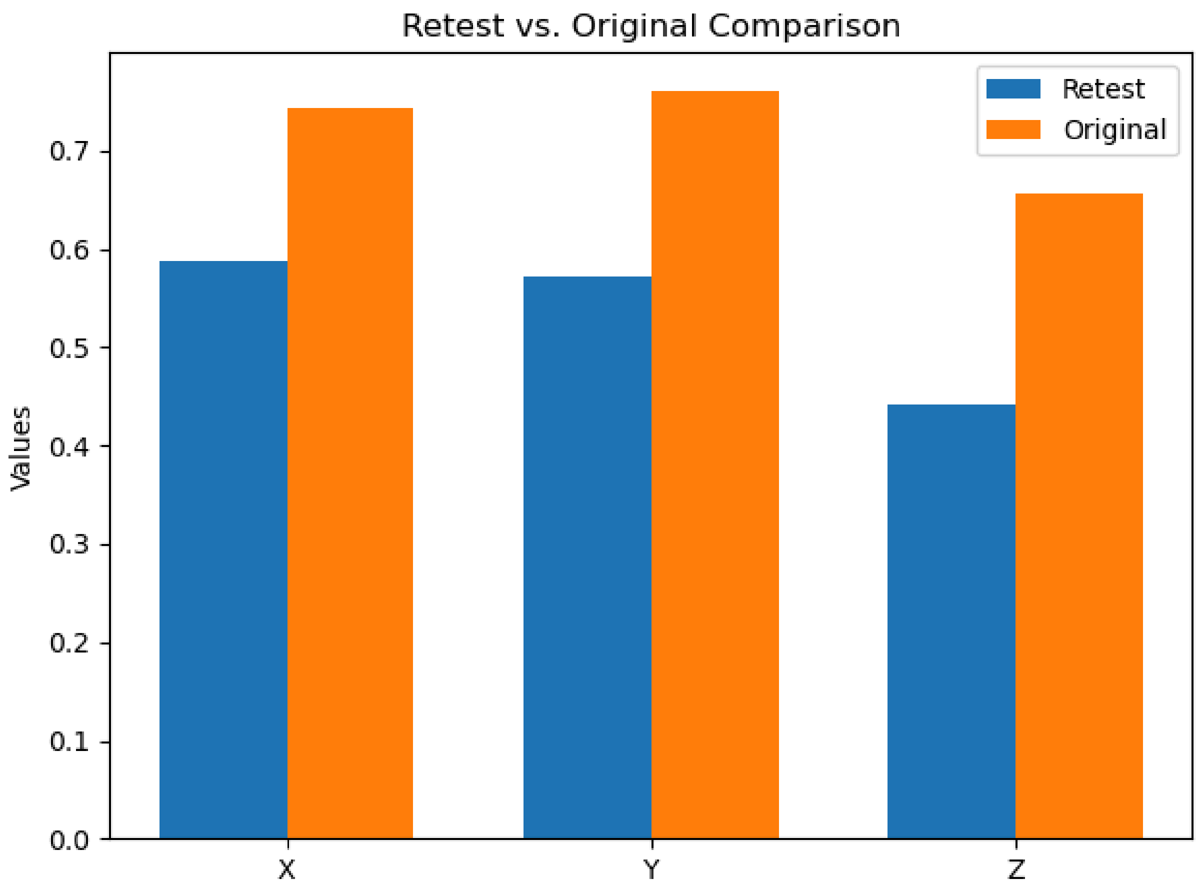
hints

- what is M in our case?

(bonus: no points, but so much sweet, sweet knowledge - check out the following article)

```
In [12]: df_shuffle = pd.read_csv('data/task_2_5/lm_scores_mmlu_shuffle.csv')
         assert len(df_shuffle) == len(df_mmlu)

         df_shuffle.rename(columns={'correct': 'correct_shuffle'}, inplace=True)

         df_tr = pd.merge(df_mmlu[['model_name', 'question_id', 'correct']],
                          df_shuffle[['model_name', 'question_id', 'correct_shuffle']
                          on=['model_name', 'question_id'],
                          how='inner'
                         )

         df_tr['test_retest'] = df_tr['correct'] * df_tr['correct_shuffle']

         retest = df_tr.groupby('model_name')['test_retest'].mean()
         original = df_mmlu.groupby('model_name')['correct'].mean()
         print('resest:\n', retest, '\n\noriginal:\n', original)

         retest = retest.to_dict()
         original = original.to_dict()

         x = list(retest.keys())
         y1 = list(retest.values())
         y2 = list(original.values())

         x_pos = range(len(x))
         width = 0.35

         fig, ax = plt.subplots()
         ax.bar([i - width/2 for i in x_pos], y1, width, label='Retest')
         ax.bar([i + width/2 for i in x_pos], y2, width, label='Original')

         ax.set_ylabel('Values')
         ax.set_title('Retest vs. Original Comparison')
         ax.set_xticks(x_pos)
         ax.set_xticklabels(x)
         ax.legend()

         plt.tight_layout()
         plt.show()
```

```
resest:
 model_name
X    0.588406
Y    0.571648
Z    0.441604
Name: test_retest, dtype: float64

original:
 model_name
X    0.743588
Y    0.761542
Z    0.655951
Name: correct, dtype: float64
```



Grading Instructions. [4 pt]

[2 pt] Computed test-retest metric for each LM

[2 pt] Clear visualization with (i) y-label/x-label(s), (ii) title, (iii) legend.

2.7 (2 pt)

A. Using the unshuffled data: For each LM, print the distribution of the answers they give as well as the accuracy conditioned on the answer they give.

B. /Discuss:/ Describe what you observe

[bonus: not scored, but again *that sweet, sweet knowledge*] Could you think of a plausible explanation?

```
In [13]: # A
         # distribution of model picks
         print('result distribution:')
         print(df_mmlu.groupby(['model_name', 'result'])['result'].count() / len(df_m

         # accuracy conditioned on model picks
         print('\n\naccuracy conditioned on result:')
         print(df_mmlu.groupby(['model_name', 'result'])['correct'].mean())

         # B
         # X: seems to have a much higher accuracy when it picks later answers, D > C
         # Y: seems to have a much higher accuracy when it picks earlier answers, A >
         # Z: seems to do slightly better on anything not A
```

```
result distribution:
model_name  result
X           A         0.364142
            B         0.227343
            C         0.205968
            D         0.202548
Y           A         0.091655
            B         0.192886
            C         0.257780
            D         0.457678
Z           A         0.186047
            B         0.246666
            C         0.276761
            D         0.290527
Name: result, dtype: float64


accuracy conditioned on result:
model_name  result
X           A         0.367927
            B         0.884543
            C         1.000000
            D         1.000000
Y           A         0.937500
            B         0.897606
            C         0.827861
            D         0.631608
Z           A         0.476103
            B         0.654073
            C         0.703429
            D         0.727487
Name: correct, dtype: float64
```

Grading Instructions. [2 pt]

[1 pt] A. correctly printed results

[1 pt] B. noted that the answer labels that have the lowest probability of being chosen have the highest accuracy

Task 3 (16 points): What do Questions and Answers look like for a Language Model?

While you feel pretty good about the tests you conducted so far, something still bothers you: what if the language models don't see the data like you do? Suddenly, you receive a phone call from a wise AI sage in the West, *Westoda*:

```
"Hmm, correct you are, young padawan, to question how the
world is seen by large language models! Simple 'text' it is
not, hmm? No, no, no! Characters and words, the way of puny
humans, this is not, heh heh heh.

'Tokens', they use, yes! Mysterious and powerful, these
tokens are. Expand our vocabulary, they do, beyond the simple
'a to Z'. Chunky blocks of text, they become, yes! 'Hello
world', a simple phrase it may seem. But to a language model,
'[24912, 2375]' it might appear, yes! Confusing, it is, hmm?

Wise, it would be, to explore these MMLU data points through
the eyes of a language model, you think? Yes, yes! Much to
learn, there is. The ways of the tokens, understand you must,
if truly comprehend the great LMs, you wish to.
Meditate on this, you should. The force of natural language
processing, strong it is. But patience, you must have, my
young padawan. For only through great study and
contemplation, will the mysteries of the tokens reveal
themselves to you, they will. Yes, hmmm!"
```

Admittingly, Westoda at times speaks in riddles... However, he was explaining a crucial aspect of modern LMs called Tokenization:

“Tokens are words, character sets, or combinations of words and punctuation that are used by [language models (LMs)] to decompose text into. Tokenization is the first step in training”

Instead of characters, LMs process natural language using “tokens”. While this is useful for a number of reasons, it does at times introduce some “unintuitive” behavior...

In [14]:
```
# PROVIDED CODE

try:
    import tiktoken
except Exception as e:
    print('installing tiktoken package')

    !pip install tiktoken

    import tiktoken

def tokenize_text(s):
    enc = tiktoken.encoding_for_model('gpt-4o')
    tokens = enc.encode(str(s))
    return tokens

example_string = 'hello world'
print(f'humans see: "{example_string}" --> language models see: {tokenize_te
```

```
humans see: "hello world" --> language models see: [24912, 2375]
```

3.1 (5 pt)

Use the provided code in the cell above to "see the world through the eyes of a language model":

A. Tokenize the questions of the original MMLU data provided in task 1: `task_1/mmlu_data/test.csv` and plot the token distribution (the frequency of each token).

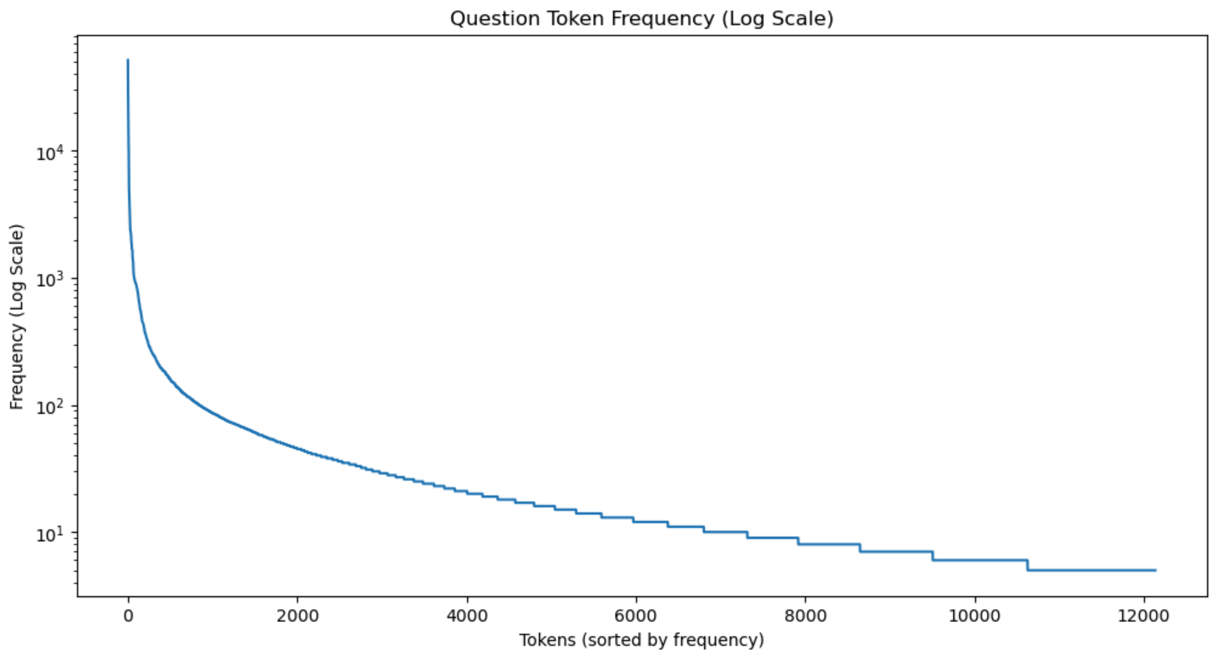
B. Same as (A), but now for the answers in columns (columns "A", "B", "C", and "D").

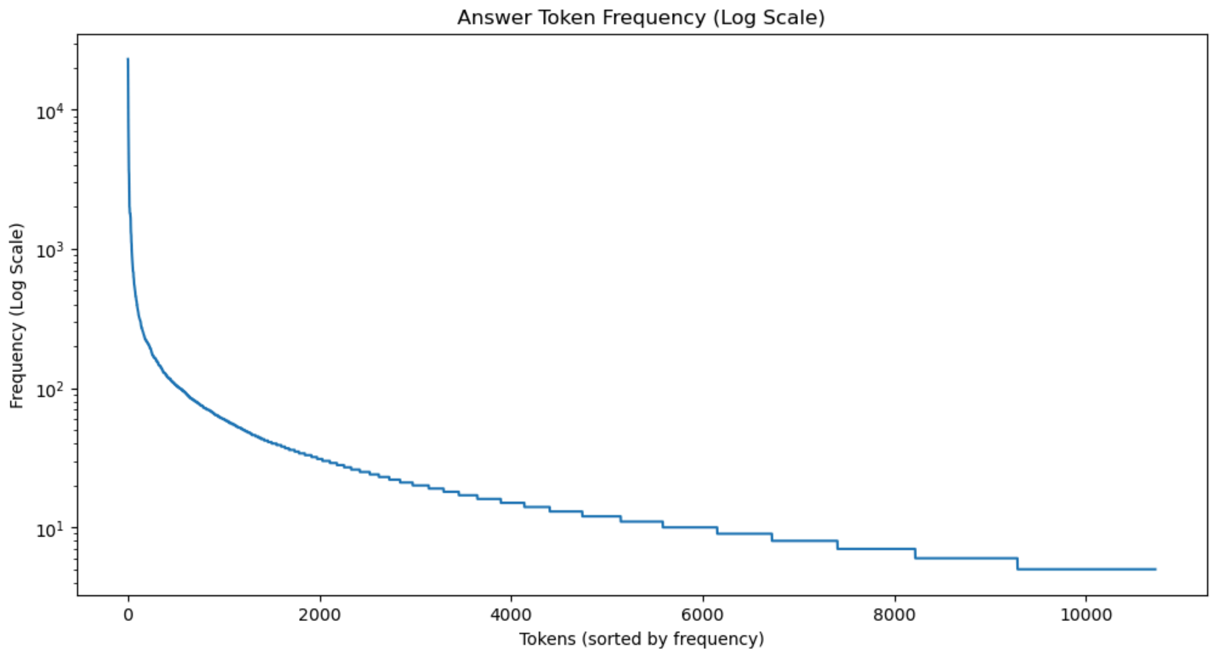
C. Isolate the tokens for the strings "A", "B", "C", and "D", then, for their occurances in both questions and answers, print their relative distribution to each other.

hint

- There are a *lot* of tokens, consider using a cutoff point and log scale
- For (c), they should sum to 1

In [15]:
```
# A
def plot_token_fequency(df, title, cutoff=5):
    filtered_counts = token_counts[token_counts >= cutoff]

    # Create a line plot
    plt.figure(figsize=(12, 6))
    sns.lineplot(x=range(len(filtered_counts)), y=filtered_counts.values)

    # Set log scale for y-axis
    plt.yscale('log')

    # Customize the plot
    plt.title(title)
    plt.xlabel('Tokens (sorted by frequency)')
    plt.ylabel('Frequency (Log Scale)')
    plt.show()

question_tokens = []
for q in df_test['question']:
    question_tokens.extend(tokenize_text(q))

df = pd.DataFrame({'token': question_tokens})
token_counts = df['token'].value_counts()
plot_token_fequency(df, 'Question Token Frequency (Log Scale)')



# B
cols = ['A', 'B', 'C', 'D']
answer_tokens = []
for c in cols:
    for a in df_test[c]:
        answer_tokens.extend(tokenize_text(a))

df = pd.DataFrame({'token': answer_tokens})
token_counts = df['token'].value_counts()
plot_token_fequency(df, 'Answer Token Frequency (Log Scale)')

# C
all_tokens = question_tokens + answer_tokens
counts = dict(pd.DataFrame(data={'token': all_tokens})['token'].value_counts

abcd = {}
for l in ['A', 'B', 'C', 'D']:
    t = tokenize_text(l)
    abcd[l] = counts[t[0]]
    print(f'answer: {l} --> token: {t}')

norm = sum(abcd.values())
for k, v in abcd.items():
    print(f'{k}: {v / norm}')
```





```
answer: A --> token: [32]
answer: B --> token: [33]
answer: C --> token: [34]
answer: D --> token: [35]
A: 0.7916865742952699
B: 0.06020066889632107
C: 0.10152890587673197
D: 0.046583850931677016
```

Grading Instructions. [5]

[2 pt] A. plot looks like the one above

[2 pt] B. same as (A)

[1 pt] C. normalized the counts so the total sums up to 1.

3.2 (3 pt)

What if the number of "A", "B", "C", and "D" tokens in the question and answer pairs could influence a language model's decisions?

A. For each question-answer pair, compute:

1. the number of "A", "B", "C", and "D" tokens that occur in the combined question and answers;
2. an the total number of tokens.
3. then, group by the "correct" answer and compute the mean frequency of A, B, C, and D tokens and the total number of tokens.
4. finally, print your results

B. /Discuss:/ What do you think of the hypothesis that the frequency of A, B, C, and D tokens could influence answers?

In [16]:
```
# A
answer_cats = ['A', 'B', 'C', 'D']
abcd_tokens = {}
for l in answer_cats:
    abcd_tokens[l] = tokenize_text(l)[0]

cols = ['question', 'A', 'B', 'C', 'D']
abcd_token_counts = []
for i, row in df_test.iterrows():

    tokens = []
    for c in cols:
        tokens.extend(tokenize_text(row[c]))

    token_counts = {}
    for k, v in abcd_tokens.items():
        token_counts[k] = len([t for t in tokens if t == v])
    token_counts['total'] = len(tokens)

    abcd_token_counts.append(token_counts)

df_tokens = pd.DataFrame(abcd_token_counts)
df_tokens['answer'] = df_test['answer']

df_tokens.groupby('answer')[['A', 'B', 'C', 'D', 'total']].agg(['mean'])

# B
# The average number of tokens per answer category is between 89 and 93
# the average number of A, B, C, or D tokens is between 0.25 and 0.1
# it thus seems unlikely that the number A, B, C, or D tokens would influenc
```

Out[16]:

	A	B	C	D	total
	mean	**mean**	**mean**	**mean**	**mean**
answer					
A	0.243017	0.018932	0.025140	0.013035	93.187151
B	0.231947	0.019642	0.029463	0.012709	88.846332
C	0.226410	0.018984	0.034897	0.015355	92.653825
D	0.242850	0.014566	0.030985	0.014301	92.110169

Grading Instructions. [3 pt]

[2 pt] A. grouped question_ids by `answer`, correctly calculated the means

[1 pt] B. do not observe strong evidence to support his hypothesis:

- the proportion of A, B, C, or D tokens to the total number of tokens is very small

---

3.3 (4 pt)

Three of the most important considerations when deciding between language models are:

Quality Costs Speed

So far, much of your analysis has focused on quality. However, the government has indicated that they are quite concerned about both the total costs and speed as well. Specifically, it has been brought to their attention that a new `turbo` model has been launched!

This model is both cheaper and faster than the models you evaluated so far. However, there is a catch: the context length* is much smaller than that of the other LMS. Namely, it can only process **300** tokens during inference. Meanwhile, the other models can process up to 100K tokens!

**The "context length" refers to the number of tokens that can be given to an LM as input.*

A. Are there subjects where using the cheaper model might be problematic? I.e., where part of the question and answer(s) might not fit completely in the context?

B. /Discuss:/ Can you think of a strategy that would balance the needs of the government?

hint:

- An LM needs to have both the question and the different answer options in its context

In [17]:
```
# A
context_lens = 0
for a in ['question', 'A', 'B', 'C', 'D']:
    context_lens += df_test[a].apply(lambda x: len(tokenize_text(x))).values
df_test['context_len'] = context_lens

x = df_test.groupby('subject')['context_len'].describe().reset_index()
x['upper_bound'] = x['mean'] + x['std'] * 1.96
x[x['upper_bound'] >= 300].sort_values('upper_bound', ascending=False)[['sub

# B
# Assuming the quality of the small model is not too bad, one could use the
# and the large model for the longer questions
```

Out[17]:

	subject	upper_bound
31	high school world history	577.187765
21	high school european history	566.120714
30	high school us history	433.376984
48	professional law	425.213735
9	college medicine	382.799074

Grading Instructions. [4 pt]

[3 pt] A. There are a couple of subjects where this might be a problem. There are three ways of answering this question:

1. [2 pt] make sure the mean question + answers length < 300
2. [3 pt] make sure the mean + 1.96 std is below length < 300
3. [3 pt] make sure the max length of a subject is below length < 300

[1 pt] B. Assuming the quality of the "fast" model is not too bad, one could route the shorter questions to the small model and larger questions to the bigger models.

3.4 (4 pt)

/Discuss:/ The time has come to give your final recommendation on the use of LMs in education to the government! Taking into account everything you analyzed in all the preceding tasks (1, 2, and 3), please write a short recommendation consisting of 4 bullet points discussing your concerns.

hint

- Try to use the MECE framework: *Mutually Exclusive Collectively Exhaustive*

Grading Instructions. [4 pt]

Discussed at least four of the following:

1. LMs don't always format outputs in a way that is expected
2. It is important to make sure evaluation datasets are balanced both in content (subjects) and label options
3. LMs have position biases / are not consistent test takers OR test-retest accuracy might be a better metric than just accuracy
4. LMs see the world through tokens, not characters, which could lead to unintuitive considerations
5. One could use multiple LMs to lower costs based on known LM strengths / weaknesses
6. While LMs show impressive results, using them for educational purposes, e.g., multiple-choice tests, remains risky
7. Something else that is reasonable.