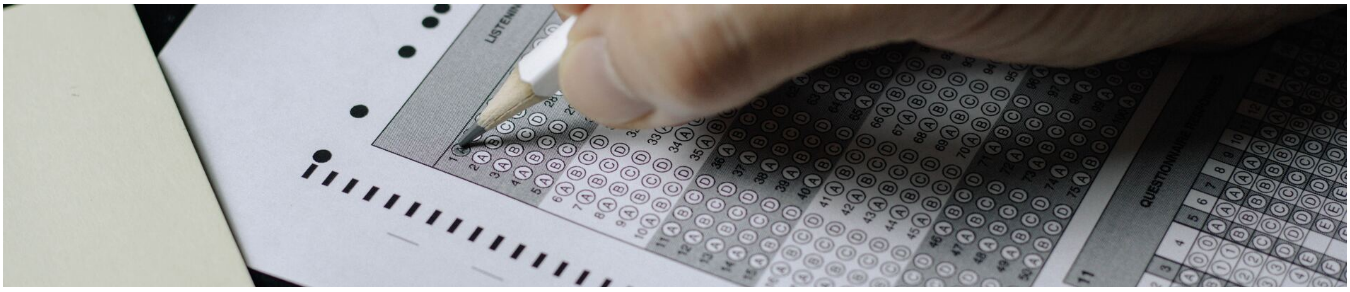
Kaggle - LLM Science Exam



In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
df = pd.read_csv("llm.csv")
```

In [4]:

```
df.head()
```

Out[4]:

	id	prompt	A	B	C	D	E	ans
0	0	Which of the following statements accurately d...	MOND is a theory that reduces the observed mis...	MOND is a theory that increases the discrepanc...	MOND is a theory that explains the missing bar...	MOND is a theory that reduces the discrepancy ...	MOND is a theory that eliminates the observed ...	
1	1	Which of the following is an accurate definiti...	Dynamic scaling refers to the evolution of sel...	Dynamic scaling refers to the non-evolution of...	Dynamic scaling refers to the evolution of sel...	Dynamic scaling refers to the non-evolution of...	Dynamic scaling refers to the evolution of sel...	
2	2	Which of the following statements accurately d...	The triskeles symbol was reconstructed as a fe...	The triskeles symbol is a representation of th...	The triskeles symbol is a representation of a ...	The triskeles symbol represents three interloc...	The triskeles symbol is a representation of th...	
3	3	What is the significance of regularization in ...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	
4	4	Which of the following statements accurately d...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	

In [5]:

```
df.tail()
```

Out[5]:

	id	prompt	A	B	C	D	E	answer
195	195	What is the relation between the three moment ...	The three moment theorem expresses the relatio...	The three moment theorem is used to calculate ...	The three moment theorem describes the relatio...	The three moment theorem is used to calculate ...	The three moment theorem is used to derive the...	C
196	196	What is the throttling process, and why is it ...	The throttling process is a steady flow of a f...	The throttling process is a steady adiabatic f...	The throttling process is a steady adiabatic f...	The throttling process is a steady flow of a f...	The throttling process is a steady adiabatic f...	B
197	197	What happens to excess base metal as a solutio...	The excess base metal will often solidify, bec...	The excess base metal will often crystallize-o...	The excess base metal will often dissolve, bec...	The excess base metal will often liquefy, beco...	The excess base metal will often evaporate, be...	B
198	198	What is the relationship between mass, force, ...	Mass is a property that determines the weight ...	Mass is an inertial property that determines a...	Mass is an inertial property that determines a...	Mass is an inertial property that determines a...	Mass is a property that determines the size of...	D
199	199	What did Arthur Eddington discover about two o...	Arthur Eddington showed that two of Einstein's...	Arthur Eddington showed that two of Einstein's...	Arthur Eddington showed that two of Einstein's...	Arthur Eddington showed that two of Einstein's...	Arthur Eddington showed that two of Einstein's...	C

In [6]:

```
df.shape
```

Out[6]:

```
(200, 8)
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['id', 'prompt', 'A', 'B', 'C', 'D', 'E', 'answer'], dtype='object')
```

In [8]:

```
df.duplicated().sum()
```

Out[8]:

```
0
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
id        0
prompt    0
A         0
B         0
C         0
D         0
E         0
answer    0
dtype: int64
```

In [10]:

```
df = df.drop('id', axis = 1)
```

In [11]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   prompt  200 non-null    object
 1   A       200 non-null    object
 2   B       200 non-null    object
 3   C       200 non-null    object
 4   D       200 non-null    object
 5   E       200 non-null    object
 6   answer  200 non-null    object
dtypes: object(7)
memory usage: 11.1+ KB
```

In [12]:

```
df.nunique()
```

Out[12]:

```
prompt    200
A         200
B         200
C         200
D         200
E         200
answer      5
dtype: int64
```

In [13]:

```
df['answer'].unique()
```

Out[13]:

```
array(['D', 'A', 'C', 'B', 'E'], dtype=object)
```

In [14]:

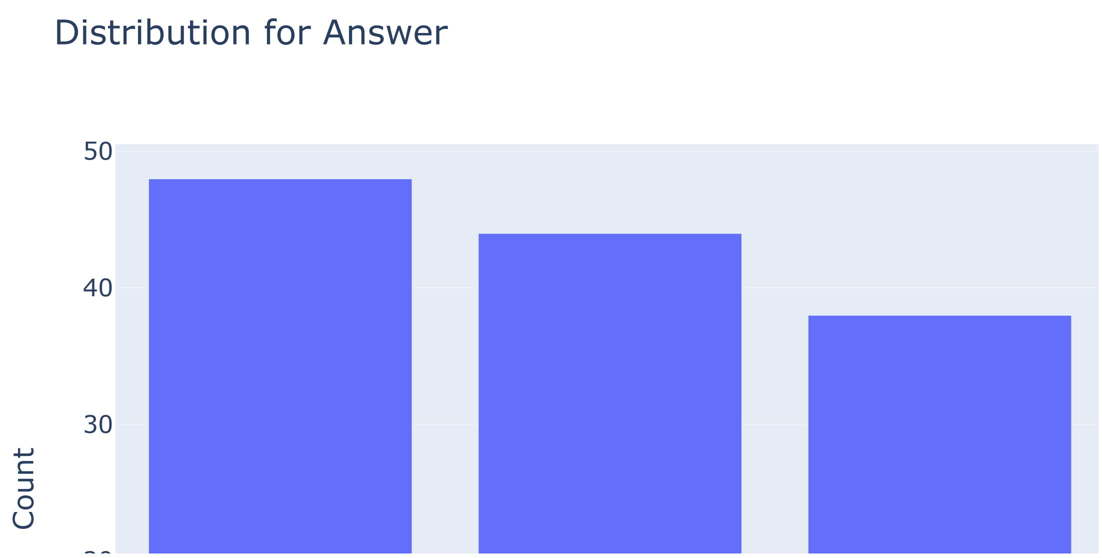
```
df['answer'].value_counts()
```

Out[14]:

```
B    48
C    44
D    38
A    37
E    33
Name: answer, dtype: int64
```

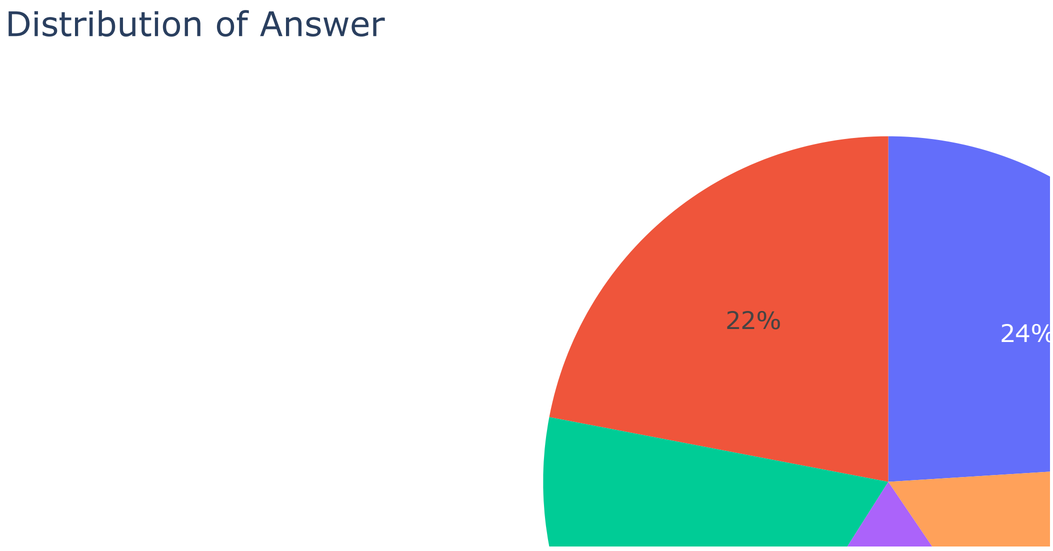
In [15]:

```
fig = go.Figure(data=[go.Bar(x=df['answer'].value_counts().index, y=df['answer'].value_c
fig.update_layout(title='Distribution for Answer',xaxis_title='Answer',yaxis_title="Coun
fig.show()
```



In [16]:

```
fig = px.pie(df, names=df['answer'], title='Distribution of Answer')
fig.show()
```



In [17]:

```
print("Sample Questions:")
for i in range(len(df)):
    print(f"Question {i + 1}:")
    print("Prompt:", df['prompt'].iloc[i])
    print("Options:")
    print("A:", df['A'].iloc[i])
    print("B:", df['B'].iloc[i])
    print("C:", df['C'].iloc[i])
    print("D:", df['D'].iloc[i])
    print("E:", df['E'].iloc[i])
    print()
```

```
Sample Questions:
Question 1:
Prompt: Which of the following statements accurately describes the impa
ct of Modified Newtonian Dynamics (MOND) on the observed "missing baryo
nic mass" discrepancy in galaxy clusters?
Options:
A: MOND is a theory that reduces the observed missing baryonic mass in
galaxy clusters by postulating the existence of a new form of matter ca
lled "fuzzy dark matter."
B: MOND is a theory that increases the discrepancy between the observed
missing baryonic mass in galaxy clusters and the measured velocity disp
ersions from a factor of around 10 to a factor of about 20.
C: MOND is a theory that explains the missing baryonic mass in galaxy c
lusters that was previously considered dark matter by demonstrating tha
t the mass is in the form of neutrinos and axions.
D: MOND is a theory that reduces the discrepancy between the observed m
issing baryonic mass in galaxy clusters and the measured velocity dispe
rsions from a factor of around 10 to a factor of about 2.
E: MOND is a theory that eliminates the observed missing baryonic mass
```

In [18]:

```
correct_answers = df['answer'].value_counts()
print("Distribution of Correct Answers:")
print(correct_answers)
print()
```

```
Distribution of Correct Answers:
B    48
C    44
D    38
A    37
E    33
Name: answer, dtype: int64
```

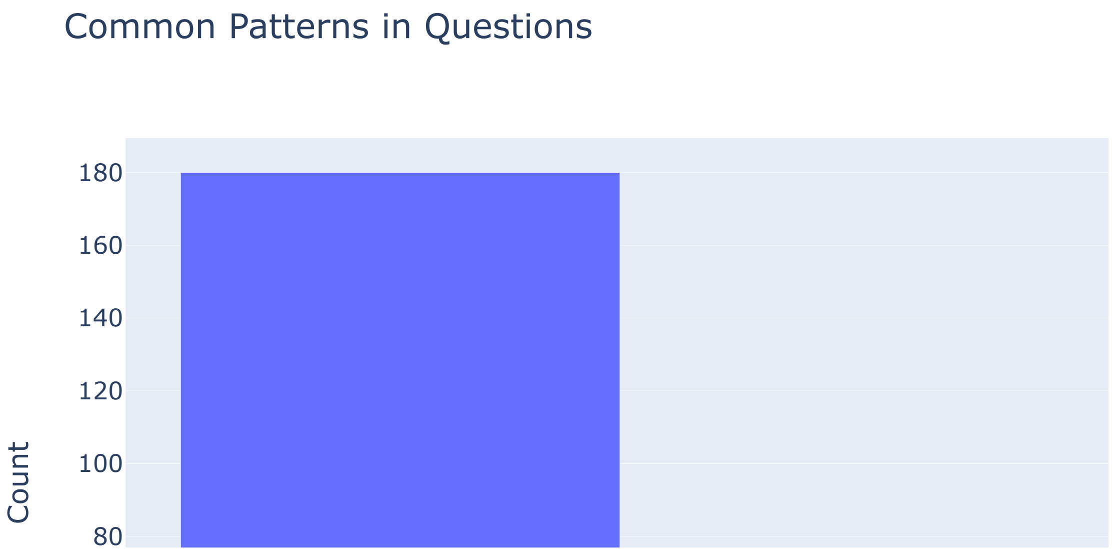
In [19]:

```
common_patterns = df['prompt'].str.extract(r'(Which|What|How|When|Why)').value_counts()
print("Common Patterns in Questions:")
print(common_patterns)
```

```
Common Patterns in Questions:
What     180
Which     11
How        2
dtype: int64
```

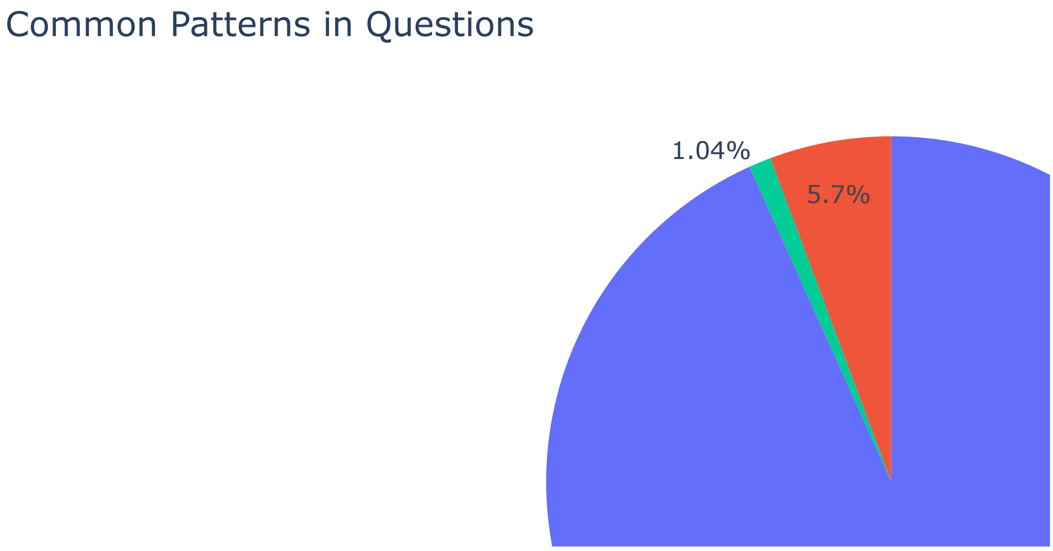
In [20]:

```
pattern_distribution = pd.DataFrame({'Pattern': ['What', 'Which', 'How'],
                                     'Count': [180, 11, 2]})
fig = px.bar(pattern_distribution, x='Pattern', y='Count', labels={'x': 'Pattern', 'y':
             title='Common Patterns in Questions')
fig.show()
```



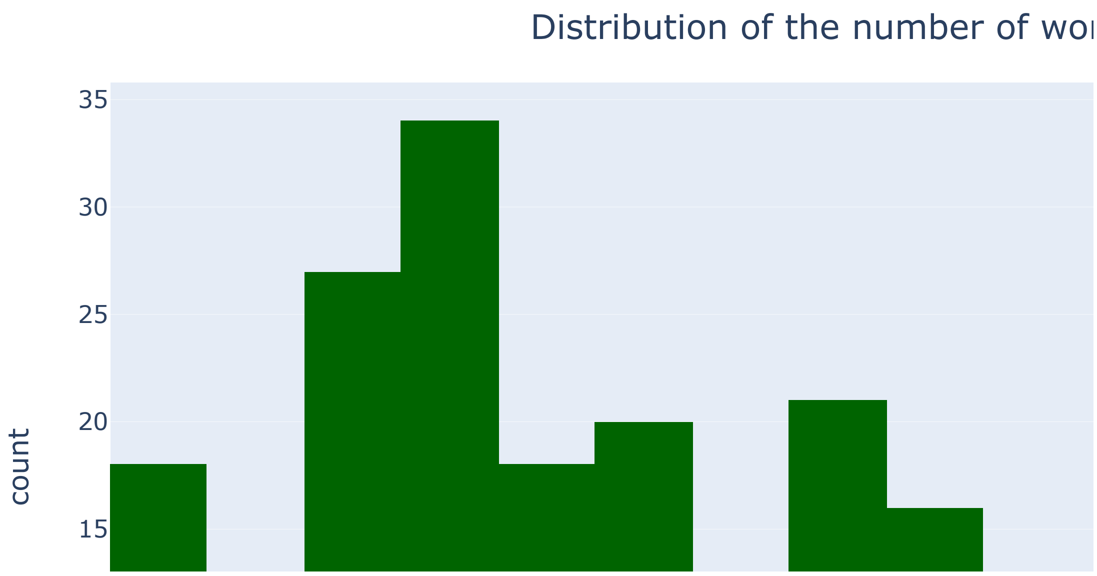
In [21]:

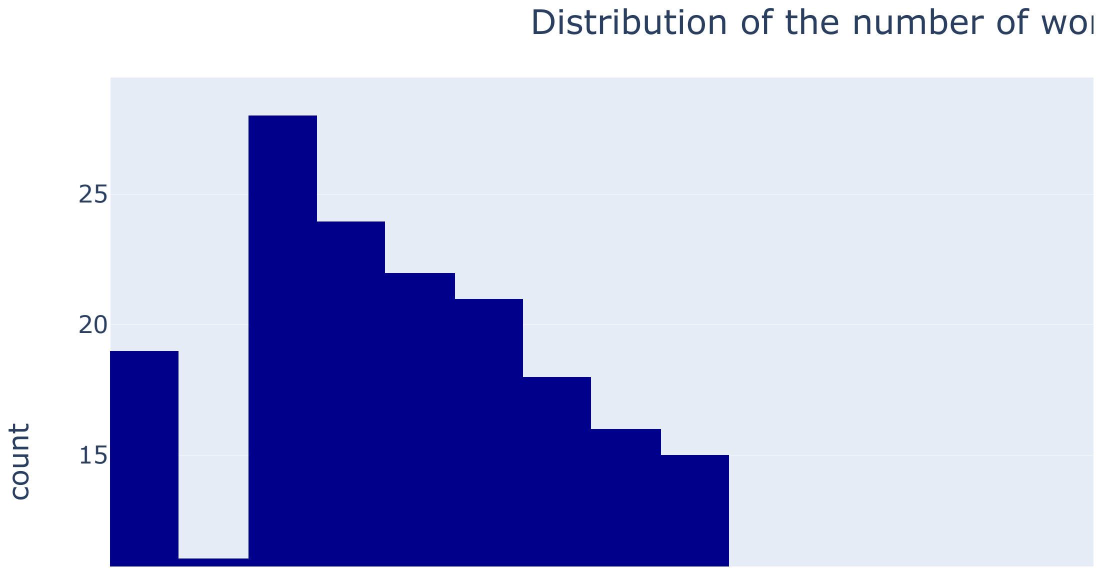
```
fig = px.pie(pattern_distribution, values='Count', names='Pattern', title='Common Patter
fig.show()
```

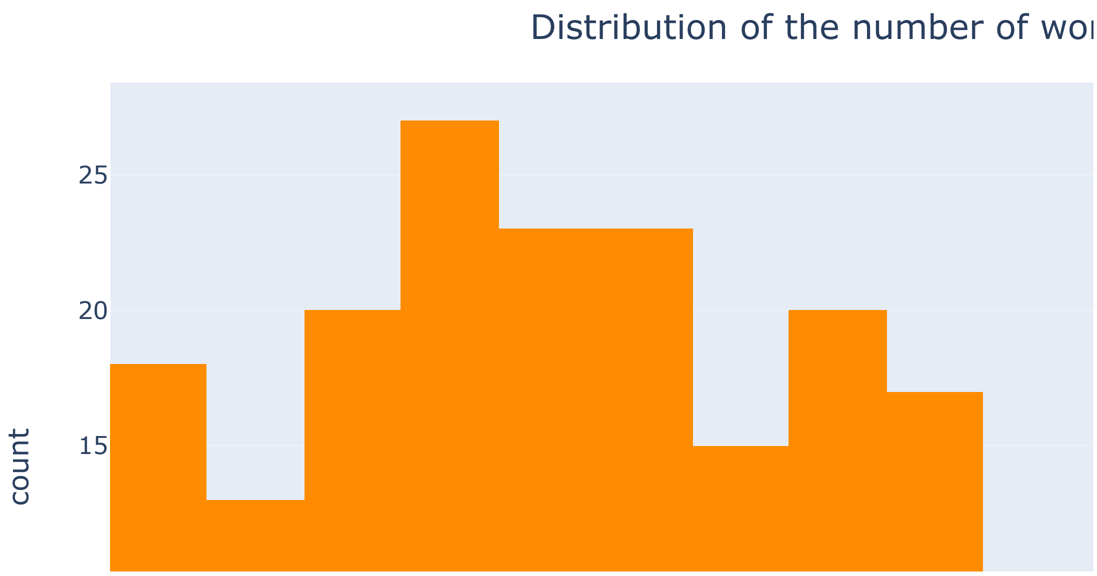


In [22]:

```
num_words_A = [len(x.split(" ")) for x in df["A"]]
num_words_B = [len(x.split(" ")) for x in df["B"]]
num_words_C = [len(x.split(" ")) for x in df["C"]]
num_words_D = [len(x.split(" ")) for x in df["D"]]
num_words_E = [len(x.split(" ")) for x in df["E"]]

fig_A = px.histogram(num_words_A, nbins=40, color_discrete_sequence=['darkgreen'])
fig_B = px.histogram(num_words_B, nbins=40, color_discrete_sequence=['darkblue'])
fig_C = px.histogram(num_words_C, nbins=40, color_discrete_sequence=['darkorange'])
fig_D = px.histogram(num_words_D, nbins=40, color_discrete_sequence=['darkred'])
fig_E = px.histogram(num_words_E, nbins=40, color_discrete_sequence=['skyblue'])

for fig, option in [(fig_A, 'A'), (fig_B, 'B'), (fig_C, 'C'), (fig_D, 'D'), (fig_E, 'E')
    fig.update_layout(
        showlegend=False,
        xaxis_title="Number of words",
        title={
            'text': f"Distribution of the number of words in option {option}",
            'y': 0.95,
            'x': 0.5,
            'xanchor': 'center',
            'yanchor': 'top'
        }
    )

fig_A.show()
fig_B.show()
fig_C.show()
fig_D.show()
fig_E.show()
```


Distribution of the number of wor
35
30
25
20
15
count


Distribution of the number of wor
count
25
20
15

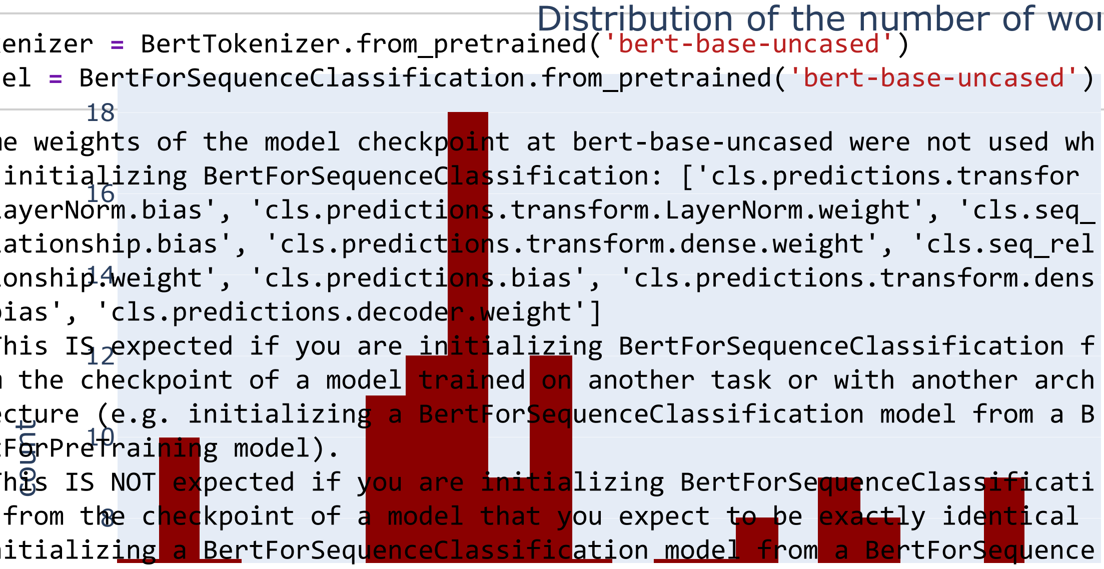


In [23]:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
```

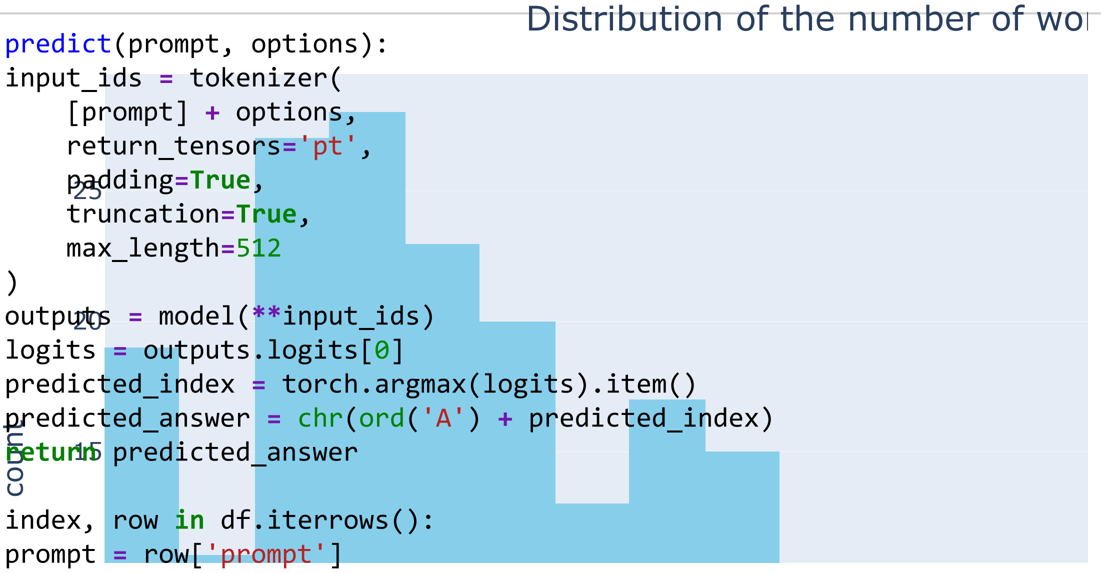
In [24]:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased')
```




```
Some weights of the model checkpoint at bert-base-uncased were not used wh
en initializing BertForSequenceClassification: ['cls.predictions.transfor
m.LayerNorm.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.seq_
relationship.bias', 'cls.predictions.transform.dense.weight', 'cls.seq_rel
ationship.weight', 'cls.predictions.bias', 'cls.predictions.transform.dens
e.bias', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertForSequenceClassification f
rom the checkpoint of a model trained on another task or with another arch
itecture (e.g. initializing a BertForSequenceClassification model from a B
ertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassificati
on from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a BertForSequence
Classification model).
Some weights of BertForSequenceClassification were not initialized from th
e model checkpoint at bert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to u
se it for predictions and inference.
```

In [25]:




```
def predict(prompt, options):
    input_ids = tokenizer(
        [prompt] + options,
        return_tensors='pt',
        padding=True,
        truncation=True,
        max_length=512
    )
    outputs = model(**input_ids)
    logits = outputs.logits[0]
    predicted_index = torch.argmax(logits).item()
    predicted_answer = chr(ord('A') + predicted_index)
    return predicted_answer

for index, row in df.iterrows():
    prompt = row['prompt']
    options = [row['A'], row['B'], row['C'], row['D'], row['E']]
    predicted_answer = predict(prompt, options)

    if predicted_answer == row['answer']:
        print('Correct!')
    else:
        print('Incorrect')
```

```
Incorrect
Incorrect
Incorrect
Incorrect
Incorrect
Correct!
Incorrect
Incorrect
Incorrect
Incorrect
Incorrect
Incorrect
Incorrect
Incorrect
Correct!
Correct!
Incorrect
Incorrect
Incorrect
```

In [34]:

```
def encode_and_similarity(prompt, option):
    encoded_prompt = tokenizer(prompt, return_tensors="pt", padding=True, truncation=Tru
    encoded_option = tokenizer(option, return_tensors="pt", padding=True, truncation=Tru

    # Ensure the inputs are on the same device as the model (CPU or GPU)
    encoded_prompt = {key: tensor.to(model.device) for key, tensor in encoded_prompt.ite
    encoded_option = {key: tensor.to(model.device) for key, tensor in encoded_option.ite

    with torch.no_grad():
        prompt_output = model(**encoded_prompt).logits  # Get the logits
        option_output = model(**encoded_option).logits  # Get the logits

    # Reshape the tensors to have shape (batch_size, -1)
    prompt_output = prompt_output.view(prompt_output.size(0), -1)
    option_output = option_output.view(option_output.size(0), -1)

    similarity_score = torch.nn.functional.cosine_similarity(prompt_output, option_outpu
    return similarity_score
```

In [35]:

```
def predict_answer(row):
    similarities = {
        'A': encode_and_similarity(row['prompt'], row['A']),
        'B': encode_and_similarity(row['prompt'], row['B']),
        'C': encode_and_similarity(row['prompt'], row['C']),
        'D': encode_and_similarity(row['prompt'], row['D']),
        'E': encode_and_similarity(row['prompt'], row['E'])
    }
    return max(similarities, key=similarities.get)
```

In [36]:

```
df['predicted_answer'] = df.apply(predict_answer, axis=1)
```

In [37]:

```
accuracy = (df["answer"] == df["predicted_answer"]).mean()
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.235
```

In [58]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelBinarizer
from sklearn.pipeline import make_pipeline
```

In [59]:

```
data = []
for _, row in df.iterrows():
    for option in "ABCDE":
        data.append({
            "text": row["prompt"] + " " + row[option],
            "correct": int(row["answer"] == option)
        })
```

In [60]:

```
preprocessed_df = pd.DataFrame(data)
```

In [61]:

```
X_train = preprocessed_df["text"]
y_train = preprocessed_df["correct"]
```

In [62]:

```
pipeline = make_pipeline(TfidfVectorizer(), LogisticRegression())
```

In [63]:

```
pipeline.fit(X_train, y_train)
```

Out[63]:

```
▸ Pipeline
  ▸ TfidfVectorizer
  ▸ LogisticRegression
```

In [64]:

```
df["predicted_answer"] = df.apply(lambda row: max(("A", pipeline.predict_proba([row["pro
                                                ("B", pipeline.predict_pr
                                                ("C", pipeline.predict_pr
                                                ("D", pipeline.predict_pr
                                                ("E", pipeline.predict_pr
                                                key=lambda pair: pair[1])
```

In [65]:

```
accuracy = (df["answer"] == df["predicted_answer"]).mean()
accuracy
```

Out[65]:

```
0.87
```