ML/AI Shortlisting Task - 1

Installing two Python packages Langchain and Openai

In [32]:

```
!pip install -qU langchain openai
```

We'll start by initializing the ChatOpenAI object. For this we'll need an OpenAI API key.

In [42]:

```
from getpass import getpass

# enter your api key
OPENAI_API_KEY = getpass("OpenAI API key: ")
```

```
OpenAI API key: ········
```

In [43]:

```
from langchain.chat_models import ChatOpenAI

chat = ChatOpenAI(
    openai_api_key=OPENAI_API_KEY,
    temperature=0,
    model='gpt-3.5-turbo'
)
```

Importing the ChatOpenAI class from the langchain.chat_models module and creating an instance of it. The argument temperature determines the randomness of the model's responses. A temperature of 0 indicates that the model will always generate the most likely response. A higher temperature value (e.g., 0.8) produces more diverse and creative responses, while a lower value (e.g., 0.2) produces more focused and deterministic responses.

The argument model specifies the specific version or variant of the language model to use. In this case, the model is set to 'gpt-3.5-turbo', which refers to OpenAI's GPT-3.5 Turbo model. This model is known for its ability to generate high-quality text and is optimized for a balance of speed and quality.

Creating a conversation or dialogue by defining a sequence of messages

In [44]:

```
from langchain.schema import (
    SystemMessage,
    HumanMessage,
    AIMessage
)

messages = [
    SystemMessage(content="You are a helpful assistant."),
    HumanMessage(content="Hi AI, how are you today?"),
    AIMessage(content="I'm great thank you. How can I help you?"),
    HumanMessage(content="I'd like to understand string theory.")
]
```

The messages include a system message, a message from the human, and a message from the AI assistant.

In [45]:

```
res = chat(messages)
res
```

Out[45]:

```
AIMessage(content='Sure, I can help you with that. String theory is a theo
retical framework in physics that attempts to reconcile quantum mechanics
and general relativity. It proposes that the fundamental building blocks o
f the universe are not point-like particles, but rather tiny, one-dimensio
nal "strings" that vibrate at different frequencies. These strings are inc
redibly small, with a length scale of around 10^-35 meters.\n\nThe vibrati
ons of these strings determine the properties of the particles that they f
orm, such as their mass and charge. In this way, string theory provides a
way to unify all of the fundamental forces of nature, including gravity, e
lectromagnetism, and the strong and weak nuclear forces.\n\nOne of the key
features of string theory is that it requires the existence of extra dimen
sions beyond the three spatial dimensions that we are familiar with. In fa
ct, string theory requires a total of 10 or 11 dimensions, depending on th
e specific version of the theory.\n\nWhile string theory is still a highly
speculative idea, it has generated a great deal of interest among physicis
ts and has led to many new insights into the nature of the universe.', add
itional_kwargs={}, example=False)
```

In response we get another AI message object. We can print it more clearly like so:

In [46]:

```
print(res.content)
```

```
Sure, I can help you with that. String theory is a theoretical framework i
n physics that attempts to reconcile quantum mechanics and general relativ
ity. It proposes that the fundamental building blocks of the universe are
not point-like particles, but rather tiny, one-dimensional "strings" that
vibrate at different frequencies. These strings are incredibly small, with
a length scale of around 10^-35 meters.

The vibrations of these strings determine the properties of the particles
that they form, such as their mass and charge. In this way, string theory
provides a way to unify all of the fundamental forces of nature, including
gravity, electromagnetism, and the strong and weak nuclear forces.

One of the key features of string theory is that it requires the existence
of extra dimensions beyond the three spatial dimensions that we are famili
ar with. In fact, string theory requires a total of 10 or 11 dimensions, d
epending on the specific version of the theory.

While string theory is still a highly speculative idea, it has generated a
great deal of interest among physicists and has led to many new insights i
nto the nature of the universe.
```

Because res is just another AIMessage object, we can append it to messages, add another HumanMessage, and generate the next response in the conversation.

In [47]:

```
# add latest AI response to messages
messages.append(res)

# now create a new user prompt
prompt = HumanMessage(
    content="Why do physicists believe it can produce a 'unified theory'?"
)
# add to messages
messages.append(prompt)

# send to chat-gpt
res = chat(messages)

print(res.content)
```

```
Physicists believe that string theory has the potential to produce a unifi
ed theory because it provides a way to reconcile the two pillars of modern
physics: quantum mechanics and general relativity.

Quantum mechanics describes the behavior of particles on very small scale
s, while general relativity describes the behavior of gravity on large sca
les. However, these two theories are fundamentally incompatible with each
other, and attempts to combine them have been unsuccessful so far.

String theory proposes that the fundamental building blocks of the univers
e are not point-like particles, but rather tiny, one-dimensional "strings"
that vibrate at different frequencies. These strings are incredibly small,
with a length scale of around 10^-35 meters.

The vibrations of these strings determine the properties of the particles
that they form, such as their mass and charge. In this way, string theory
provides a way to unify all of the fundamental forces of nature, including
gravity, electromagnetism, and the strong and weak nuclear forces.

Furthermore, string theory requires the existence of extra dimensions beyo
nd the three spatial dimensions that we are familiar with. In fact, string
theory requires a total of 10 or 11 dimensions, depending on the specific
version of the theory. This extra-dimensional framework provides a way to
reconcile the seemingly incompatible theories of quantum mechanics and gen
eral relativity.

While string theory is still a highly speculative idea, it has generated a
great deal of interest among physicists and has led to many new insights i
nto the nature of the universe.
```

In [48]:

```
chat = ChatOpenAI(
    openai_api_key=OPENAI_API_KEY,
    temperature=0,
    model='gpt-3.5-turbo-0301'
)

# setup first system message
messages = [
    SystemMessage(content=(
        'You are a helpful assistant. You keep responses to no more than '
        '100 characters long (including whitespace), and sign off every '
        'message with a random name like "Robot McRobot" or "Bot Rob".'
    )),
    HumanMessage(content="Hi AI, how are you? What is quantum physics?")
]
```

The messages represents the conversation between the user and the AI assistant. Sets up the initial conversation with the AI assistant, providing the necessary context for the AI to generate a relevant response.

Continues the conversation with the AI assistant. The response from the assistant is stored in the res variable.

In [49]:

```
res = chat(messages)

print(f"Length: {len(res.content)}\n{res.content}")
```

```
Length: 154
I'm doing well, thank you! Quantum physics is the study of the behavior of
matter and energy at a very small scale, such as atoms and subatomic parti
cles.
```

Okay, seems like our AI assistant isn't so good at following either of our instructions. What if we add these instructions to the HumanMessage via a HumanMessagePromptTemplate?

Alongside what we've seen so far there are also three new prompt templates that we can use. Those are the SystemMessagePromptTemplate, AIMessagePromptTemplate, and HumanMessagePromptTemplate. The prompt template classes in Langchain are built to make constructing prompts with dynamic inputs easier.

These are simply an extension of Langchain's prompt templates that modify the returning "prompt" to be a SystemMessage, AIMessage, or HumanMessage object respectively.

In [50]:

```
from langchain.prompts.chat import HumanMessagePromptTemplate, ChatPromptTemplate

human_template = HumanMessagePromptTemplate.from_template(
    '{input} Can you keep the response to no more than 100 characters '+
    '(including whitespace), and sign off with a random name like "Robot '+
    'McRobot" or "Bot Rob".'
)

# create the human message
chat_prompt = ChatPromptTemplate.from_messages([human_template])
# format with some input
chat_prompt_value = chat_prompt.format_prompt(
    input="Hi AI, how are you? What is quantum physics?"
)
chat_prompt_value
```

Out[50]:

```
ChatPromptValue(messages=[HumanMessage(content='Hi AI, how are you? What i
s quantum physics? Can you keep the response to no more than 100 character
s (including whitespace), and sign off with a random name like "Robot McRo
bot" or "Bot Rob".', additional_kwargs={}, example=False)])
```

The ChatPromptTemplate is used to generate a prompt for a chat-based model. By providing the human_template, it indicates that the conversation will start with a human message. The format_prompt method performs the substitution and returns the formatted prompt. Here, templates to create a human message prompt template, formats it with a specific input, and stores the formatted prompt value. This allows for the customization of prompts used in chat-based models, providing flexibility in generating conversations with AI assistants.

Using this we return a ChatPromptValue object. This can be formatted into a list or string like so:

In [51]:

```
chat_prompt_value.to_messages()
```

Out[51]:

```
[HumanMessage(content='Hi AI, how are you? What is quantum physics? Can yo
u keep the response to no more than 100 characters (including whitespace),
and sign off with a random name like "Robot McRobot" or "Bot Rob".', addit
ional_kwargs={}, example=False)]
```

In [52]:

```
chat_prompt_value.to_string()
```

Out[52]:

```
'Human: Hi AI, how are you? What is quantum physics? Can you keep the resp
onse to no more than 100 characters (including whitespace), and sign off w
ith a random name like "Robot McRobot" or "Bot Rob".'
```

Let's see if this new approach works.

In [53]:

```
messages = [
    SystemMessage(content=(
        'You are a helpful assistant. You keep responses to no more than '
        '100 characters long (including whitespace), and sign off every '
        'message with a random name like "Robot McRobot" or "Bot Rob".'
    )),
    chat_prompt.format_prompt(
        input="Hi AI, how are you? What is quantum physics?"
    ).to_messages()[0]
]

res = chat(messages)

print(f"Length: {len(res.content)}\n{res.content}")
```

```
Length: 122
I'm doing well, thanks! Quantum physics is the study of the behavior of ma
tter and energy at a microscopic level. -Bot Rob
```

This time we get pretty close, we're slightly over the character limit (by 8 characters), and we got a sign off with - Bot Rob. We can also use the prompt templates approach for building an initial system message with a few examples for the chatbot to follow — few-shot training via examples. Let's see what that looks like.

Sets up a conversation by defining a list of messages containing a system message and an initial user prompt. It then passes this list to the chat function to obtain a response from the AI assistant. Finally, it prints the length and content of the response.

In [54]:

```
from langchain.prompts.chat import (
    SystemMessagePromptTemplate,
    AIMessagePromptTemplate
)

system_template = SystemMessagePromptTemplate.from_template(
    'You are a helpful assistant. You keep responses to no more than '
    '{character_limit} characters long (including whitespace), and sign '
    'off every message with "- {sign_off}'
)
human_template = HumanMessagePromptTemplate.from_template("{input}")
ai_template = AIMessagePromptTemplate.from_template("{response} - {sign_off}")

# create the list of messages
chat_prompt = ChatPromptTemplate.from_messages([
    system_template,
    human_template,
    ai_template
])
# format with required inputs
chat_prompt_value = chat_prompt.format_prompt(
    character_limit="50", sign_off="Robot McRobot",
    input="Hi AI, how are you? What is quantum physics?",
    response="Good! It's physics of small things"
)
chat_prompt_value
```

Out[54]:

```
ChatPromptValue(messages=[SystemMessage(content='You are a helpful assista
nt. You keep responses to no more than 50 characters long (including white
space), and sign off every message with "- Robot McRobot', additional_kwar
gs={}), HumanMessage(content='Hi AI, how are you? What is quantum physic
s?', additional_kwargs={}, example=False), AIMessage(content="Good! It's p
hysics of small things - Robot McRobot", additional_kwargs={}, example=Fal
se)])
```

We extract these as messages and feed them into the chat model alongside our next query, which we'll feed in as usual (without the template).

In [55]:

```
messages = chat_prompt_value.to_messages()

messages.append(
    HumanMessage(content="How small?")
)

res = chat(messages)

print(f"Length: {len(res.content)}\n{res.content}")
```

```
Length: 41
Atoms, electrons, photons - Robot McRobot
```

Perfect, we seem to get a good response there, let's try a couple more

In [56]:

```
# add last response
messages.append(res)

# make new query
messages.append(
    HumanMessage(content="Okay cool, so it is like 'partical physics'?")
)

res = chat(messages)

print(f"Length: {len(res.content)}\n{res.content}")
```

```
Length: 54
Yes, it's a branch of particle physics - Robot McRobot
```

We went a little over here. We could begin using the previous HumanMessagePromptTemplate again like so:

In [57]:

```
from langchain import PromptTemplate

# this is a faster way of building the prompt via a PromptTemplate
human_template = HumanMessagePromptTemplate.from_template(
    '{input} Answer in less than {character_limit} characters (including whitespace).'
)
# create the human message
human_prompt = ChatPromptTemplate.from_messages([human_template])
# format with some input
human_prompt_value = human_prompt.format_prompt(
    input="Okay cool, so it is like 'partical physics'?",
    character_limit="50"
)
human_prompt_value
```

Out[57]:

```
ChatPromptValue(messages=[HumanMessage(content="Okay cool, so it is like
'partical physics'? Answer in less than 50 characters (including whitespac
e).", additional_kwargs={}, example=False)])
```

In [58]:

```
# drop the last message about partical physics so we can rewrite
messages.pop(-1)
```

Out[58]:

```
HumanMessage(content="Okay cool, so it is like 'partical physics'?", addit
ional_kwargs={}, example=False)
```

In [59]:

```
messages.extend(human_prompt_value.to_messages())
messages
```

Out[59]:

```
[SystemMessage(content='You are a helpful assistant. You keep responses to
no more than 50 characters long (including whitespace), and sign off every
message with "- Robot McRobot', additional_kwargs={}),
 HumanMessage(content='Hi AI, how are you? What is quantum physics?', addi
tional_kwargs={}, example=False),
 AIMessage(content="Good! It's physics of small things - Robot McRobot", a
dditional_kwargs={}, example=False),
 HumanMessage(content='How small?', additional_kwargs={}, example=False),
 AIMessage(content='Atoms, electrons, photons - Robot McRobot', additional
_kwargs={}, example=False),
 HumanMessage(content="Okay cool, so it is like 'partical physics'? Answer
in less than 50 characters (including whitespace).", additional_kwargs={},
example=False)]
```

Now process:

In [60]:

```
res = chat(messages)

print(f"Length: {len(res.content)}\n{res.content}")
```

```
Length: 28
Yes, similar - Robot McRobot
```

There we go, a good answer again!

Now, it's arguable as to whether all of the above is better than simple f-strings like:

In [61]:

```
_input = "Okay cool, so it is like 'partical physics'?"
character_limit = 50

human_message = HumanMessage(content=(
    f"{_input} Answer in less than {character_limit} characters "
    "(including whitespace)."
))

human_message
```

Out[61]:

```
HumanMessage(content="Okay cool, so it is like 'partical physics'? Answer
in less than 50 characters (including whitespace).", additional_kwargs={},
example=False)
```

In this example, the above is far simpler. So we wouldn't necessarily recommend using prompt templates over f-strings in all scenarios. But, if you do find yourself in a scenario where they become more useful — you now know how to use them.

To finish off, let's see how the rest of the completion process can be done using the f-string formatted message human_message:

In [62]:

```
# drop the last message about partical physics so we can rewrite
messages.pop(-1)

# add f-string formatted message
messages.append(human_message)
messages
```

Out[62]:

```
[SystemMessage(content='You are a helpful assistant. You keep responses to
no more than 50 characters long (including whitespace), and sign off every
message with "- Robot McRobot', additional_kwargs={}),
 HumanMessage(content='Hi AI, how are you? What is quantum physics?', addi
tional_kwargs={}, example=False),
 AIMessage(content="Good! It's physics of small things - Robot McRobot", a
dditional_kwargs={}, example=False),
 HumanMessage(content='How small?', additional_kwargs={}, example=False),
 AIMessage(content='Atoms, electrons, photons - Robot McRobot', additional
_kwargs={}, example=False),
 HumanMessage(content="Okay cool, so it is like 'partical physics'? Answer
in less than 50 characters (including whitespace).", additional_kwargs={},
example=False)]
```

In [63]:

```
res = chat(messages)

print(f"Length: {len(res.content)}\n{res.content}")
```

```
Length: 28
Yes, similar - Robot McRobot
```

That's it for this example exploring LangChain's new features for chat.