LangChain: Models, Prompts and Output Parsers

Outline

- Direct API calls to OpenAI
- API calls through LangChain:
  - Prompts
  - Models
  - Output parsers

Get your OpenAI API Key

In [48]:
```
#!pip install python-dotenv
#!pip install openai
```

In [49]:
```
import os
import openai

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
openai.api_key = os.environ['OPENAI_API_KEY']
```

Note: LLM's do not always produce the same results. When executing the code in your notebook, you may get slightly different answers that those in the video.

In [50]:
```
# account for deprecation of LLM model
import datetime
# Get the current date
current_date = datetime.datetime.now().date()

# Define the date after which the model should be set to "gpt-3.5-turbo"
target_date = datetime.date(2024, 6, 12)

# Set the model variable based on the current date
if current_date > target_date:
    llm_model = "gpt-3.5-turbo"
else:
    llm_model = "gpt-3.5-turbo-0301"
```

Chat API : OpenAI

Let's start with a direct API calls to OpenAI.

In [51]:
```
def get_completion(prompt, model=llm_model):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0,
    )
    return response.choices[0].message["content"]
```

In [52]:
```
get_completion("What is 1+1?")
```

```
'As an AI language model, I can tell you that the answer to 1+1 is 2.'
```

In [53]:
```
customer_email = """
Arrr, I be fuming that me blender lid \
flew off and splattered me kitchen walls \
with smoothie! And to make matters worse,\
the warranty don't cover the cost of \
cleaning up me kitchen. I need yer help \
right now, matey!
"""
```

In [54]:
```
style = """American English \
in a calm and respectful tone
"""
```

In [55]:
```
prompt = f"""Translate the text \
that is delimited by triple backticks
into a style that is {style}.
text: ```{customer_email}```
"""

print(prompt)
```

```
Translate the text that is delimited by triple backticks
into a style that is American English in a calm and respectful tone
.
text: ```
Arrr, I be fuming that me blender lid flew off and splattered me kitchen walls with smoothie! And to make matters wor
se,the warranty don't cover the cost of cleaning up me kitchen. I need yer help right now, matey!
```
```

In [56]:
```
response = get_completion(prompt)
```

In [57]:
```
response
```

```
"I'm really frustrated that my blender lid flew off and made a mess of my kitchen walls with smoothie. To add to my f
rustration, the warranty doesn't cover the cost of cleaning up my kitchen. Can you please help me out, friend?"
```

Chat API : LangChain

Let's try how we can do the same using LangChain.

In [58]:
```
#!pip install --upgrade langchain
```

Model

In [59]:
```
from langchain.chat_models import ChatOpenAI
```

In [60]:
```
# To control the randomness and creativity of the generated
# text by an LLM, use temperature = 0.0
chat = ChatOpenAI(temperature=0.0, model=llm_model)
chat
```

```
ChatOpenAI(verbose=False, callbacks=None, callback_manager=None, client=<class 'openai.api_resources.chat_completion.
ChatCompletion'>, model_name='gpt-3.5-turbo-0301', temperature=0.0, model_kwargs={}, openai_api_key=None, openai_api_
base=None, openai_organization=None, request_timeout=None, max_retries=6, streaming=False, n=1, max_tokens=None)
```
```

Prompt template

In [61]:
```
template_string = """Translate the text \
that is delimited by triple backticks \
into a style that is {style}. \
text: ```{text}```
"""
```

In [62]:
```
from langchain.prompts import ChatPromptTemplate

prompt_template = ChatPromptTemplate.from_template(template_string)
```

In [63]:
```
prompt_template.messages[0].prompt
```

```
PromptTemplate(input_variables=['style', 'text'], output_parser=None, partial_variables={}, template='Translate the t
ext that is delimited by triple backticks into a style that is {style}. text: ```{text}```\n', template_format='f-str
ing', validate_template=True)
```

In [64]:
```
prompt_template.messages[0].prompt.input_variables
```

```
['style', 'text']
```

In [65]:
```
customer_style = """American English \
in a calm and respectful tone
"""
```

In [66]:
```
customer_email = """
Arrr, I be fuming that me blender lid \
flew off and splattered me kitchen walls \
with smoothie! And to make matters worse, \
the warranty don't cover the cost of \
cleaning up me kitchen. I need yer help \
right now, matey!
"""
```

In [67]:
```
customer_messages = prompt_template.format_messages(
                    style=customer_style,
                    text=customer_email)
```

In [68]:
```
print(type(customer_messages))
print(type(customer_messages[0]))
```

```
<class 'list'>
<class 'langchain.schema.HumanMessage'>
```

In [69]:
```
print(customer_messages[0])
```

```
content="Translate the text that is delimited by triple backticks into a style that is American English in a calm and
respectful tone\n. text: ```\nArrr, I be fuming that me blender lid flew off and splattered me kitchen walls with smo
othie! And to make matters worse, the warranty don't cover the cost of cleaning up me kitchen. I need yer help right
now, matey!\n```\n" additional_kwargs={} example=False
```

In [70]:
```
# Call the LLM to translate to the style of the customer message
customer_response = chat(customer_messages)
```

```
Retrying langchain.chat_models.openai.ChatOpenAI.completion_with_retry.<locals>._completion_with_retry in 1.0 seconds
as it raised APIError: HTTP code 504 from API (<html>
<head><title>504 Gateway Time-out</title></head>
<body>
<center><h1>504 Gateway Time-out</h1></center>
</body>
</html>
).
```

In [71]:
```
print(customer_response.content)
```

```
I'm really frustrated that my blender lid flew off and made a mess of my kitchen walls with smoothie. To add to my fr
ustration, the warranty doesn't cover the cost of cleaning up my kitchen. Can you please help me out, friend?
```

In [72]:
```
service_reply = """Hey there customer, \
the warranty does not cover \
cleaning expenses for your kitchen \
because it's your fault that \
you misused your blender \
by forgetting to put the lid on before \
starting the blender. \
Tough luck! See ya!
"""
```

In [73]:
```
service_style_pirate = """\
a polite tone \
that speaks in English Pirate\
"""
```

In [74]:
```
service_messages = prompt_template.format_messages(
    style=service_style_pirate,
    text=service_reply)

print(service_messages[0].content)
```

```
Translate the text that is delimited by triple backticks into a style that is a polite tone that speaks in English Pi
rate. text: ```Hey there customer, the warranty does not cover cleaning expenses for your kitchen because it's your f
ault that you misused your blender by forgetting to put the lid on before starting the blender. Tough luck! See ya!
```
```

In [75]:
```
service_response = chat(service_messages)
print(service_response.content)
```

```
Ahoy there, me hearty customer! I be sorry to inform ye that the warranty be not coverin' the expenses o' cleaning ye
r galley, as 'tis yer own fault fer misusin' yer blender by forgettin' to put the lid on afore startin' it. Aye, toug
h luck! Farewell and may the winds be in yer favor!
```

Output Parsers

Let's start with defining how we would like the LLM output to look like:

In [76]:
```
{
  "gift": False,
  "delivery_days": 5,
  "price_value": "pretty affordable!"
}
```

```
{'gift': False, 'delivery_days': 5, 'price_value': 'pretty affordable!'}
```

In [77]:
```
customer_review = """\
This leaf blower is pretty amazing.  It has four settings:\
candle blower, gentle breeze, windy city, and tornado. \
It arrived in two days, just in time for my wife's \
anniversary present. \
I think my wife liked it so much she was speechless. \
So far I've been the only one using it, and I've been \
using it every other morning to clear the leaves on our lawn. \
It's slightly more expensive than the other leaf blowers \
out there, but I think it's worth it for the extra features.
"""

review_template = """\
For the following text, extract the following information:

gift: Was the item purchased as a gift for someone else? \
Answer True if yes, False if not or unknown.

delivery_days: How many days did it take for the product \
to arrive? If this information is not found, output -1.

price_value: Extract any sentences about the value or price,\
and output them as a comma separated Python list.

Format the output as JSON with the following keys:
gift
delivery_days
price_value

text: {text}
"""
```
```

In [78]:
```
from langchain.prompts import ChatPromptTemplate

prompt_template = ChatPromptTemplate.from_template(review_template)
print(prompt_template)
```

```
input_variables=['text'] output_parser=None partial_variables={} messages=[HumanMessagePromptTemplate(prompt=PromptTe
mplate(input_variables=['text'], output_parser=None, partial_variables={}, template='For the following text, extract
the following information:\n\ngift: Was the item purchased as a gift for someone else? Answer True if yes, False if n
ot or unknown.\n\ndelivery_days: How many days did it take for the product to arrive? If this information is not foun
d, output -1.\n\nprice_value: Extract any sentences about the value or price,and output them as a comma separated Pyt
hon list.\n\nFormat the output as JSON with the following keys:\ngift\ndelivery_days\nprice_value\n\ntext: {text}\n',
template_format='f-string', validate_template=True), additional_kwargs={})]
```

In [79]:
```
messages = prompt_template.format_messages(text=customer_review)
chat = ChatOpenAI(temperature=0.0, model=llm_model)
response = chat(messages)
print(response.content)
```

```
{
    "gift": true,
    "delivery_days": 2,
    "price_value": ["It's slightly more expensive than the other leaf blowers out there, but I think it's worth it fo
r the extra features."]
}
```

In [80]:
```
type(response.content)
```

```
str
```

In [81]:
```
# You will get an error by running this line of code
# because'gift' is not a dictionary
# 'gift' is a string
response.content.get('gift')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[81], line 4
      1 # You will get an error by running this line of code
      2 # because'gift' is not a dictionary
      3 # 'gift' is a string
----> 4 response.content.get('gift')

AttributeError: 'str' object has no attribute 'get'
```

Parse the LLM output string into a Python dictionary

In [82]:
```
from langchain.output_parsers import ResponseSchema
from langchain.output_parsers import StructuredOutputParser
```

In [83]:
```
gift_schema = ResponseSchema(name="gift",
                             description="Was the item purchased\
                             as a gift for someone else? \
                             Answer True if yes,\
                             False if not or unknown.")
delivery_days_schema = ResponseSchema(name="delivery_days",
                                      description="How many days\
                                      did it take for the product\
                                      to arrive? If this \
                                      information is not found,\
                                      output -1.")
price_value_schema = ResponseSchema(name="price_value",
                                    description="Extract any\
                                    sentences about the value or \
                                    price, and output them as a \
                                    comma separated Python list.")

response_schemas = [gift_schema,
                    delivery_days_schema,
                    price_value_schema]
```

In [84]:
```
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)
```

In [85]:
```
format_instructions = output_parser.get_format_instructions()
```

In [86]:
```
print(format_instructions)
```

```
The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "\
`\`\`json" and "\`\`\`":

```json
{
        "gift": string  // Was the item purchased                              as a gift for someone else?
Answer True if yes,                                 False if not or unknown.
        "delivery_days": string  // How many days                                          did it take for the product
to arrive? If this                                        information is not found,
output -1.
        "price_value": string  // Extract any                                    sentences about the value or
price, and output them as a                                    comma separated Python list.
}
```
```

In [87]:
```
review_template_2 = """\
For the following text, extract the following information:

gift: Was the item purchased as a gift for someone else? \
Answer True if yes, False if not or unknown.

delivery_days: How many days did it take for the product\
to arrive? If this information is not found, output -1.

price_value: Extract any sentences about the value or price,\
and output them as a comma separated Python list.

text: {text}

{format_instructions}
"""

prompt = ChatPromptTemplate.from_template(template=review_template_2)

messages = prompt.format_messages(text=customer_review,
                                  format_instructions=format_instructions)
```

In [88]:
```
print(messages[0].content)
```

```
For the following text, extract the following information:

gift: Was the item purchased as a gift for someone else? Answer True if yes, False if not or unknown.

delivery_days: How many days did it take for the productto arrive? If this information is not found, output -1.

price_value: Extract any sentences about the value or price,and output them as a comma separated Python list.

text: This leaf blower is pretty amazing.  It has four settings:candle blower, gentle breeze, windy city, and tornad
o. It arrived in two days, just in time for my wife's anniversary present. I think my wife liked it so much she was s
peechless. So far I've been the only one using it, and I've been using it every other morning to clear the leaves on
our lawn. It's slightly more expensive than the other leaf blowers out there, but I think it's worth it for the extra
features.


The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "\
`\`\`json" and "\`\`\`":

```json
{
        "gift": string  // Was the item purchased                              as a gift for someone else?
Answer True if yes,                             False if not or unknown.
        "delivery_days": string  // How many days                                      did it take for the product
to arrive? If this                                      information is not found,
output -1.
        "price_value": string  // Extract any                                      sentences about the value or
price, and output them as a                                     comma separated Python list.
}
```
```

In [89]:
```
response = chat(messages)
```

In [90]:
```
print(response.content)
```

```
```json
{
        "gift": true,
        "delivery_days": "2",
        "price_value": ["It's slightly more expensive than the other leaf blowers out there, but I think it's worth i
t for the extra features."]
}
```
```

In [91]:
```
output_dict = output_parser.parse(response.content)
```

In [92]:
```
output_dict
```

```
{'gift': True,
 'delivery_days': '2',
 'price_value': ["It's slightly more expensive than the other leaf blowers out there, but I think it's worth it for t
he extra features."]}
```

In [93]:
```
type(output_dict)
```

```
dict
```

In [94]:
```
output_dict.get('delivery_days')
```

```
'2'
```