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 (https://platform.openai.com/account/api-keys)

In [55]:
```
#!pip install python-dotenv
#!pip install openai
```

In [1]:
```
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 [2]:
```
# 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 [3]:
```
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 [4]:
```
get_completion("What is 1+1?")
```

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

In [5]:
```
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 [6]:
```
style = """American English \
in a calm and respectful tone
"""
```

In [7]:
```
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 wit
h smoothie! And to make matters worse,the warranty don't cover the cost of cleanin
g up me kitchen. I need yer help right now, matey!
```
```

In [8]:
```
response = get_completion(prompt)
```

In [9]:
```
response
```

```
'I am quite upset that my blender lid came off and caused my smoothie to splatter
all over my kitchen walls. Additionally, the warranty does not cover the cost of c
leaning up the mess. Would you be able to assist me at this time, my friend?'
```

Chat API : LangChain

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

In [10]:
```
#!pip install --upgrade langchain
```

Model

In [11]:
```
from langchain.chat_models import ChatOpenAI
# integration of langchain and OPENAI model, internally it uses openAI model only
```

In [12]:
```
# To control the randomness and creativity of the generated
# text by an LLM, use temperature = 0.0, default value temperature=0.7
chat = ChatOpenAI(temperature=0.0, model=llm_model)
chat
```

```
ChatOpenAI(verbose=False, callbacks=None, callback_manager=None, client=<class 'op
enai.api_resources.chat_completion.ChatCompletion'>, model_name='gpt-3.5-turbo-030
1', temperature=0.0, model_kwargs={}, openai_api_key=None, openai_api_base=None, o
penai_organization=None, request_timeout=None, max_retries=6, streaming=False, n=
1, max_tokens=None)
```

Prompt template

In [13]:
```
template_string = """Translate the text \
that is delimited by triple backticks \
into a style that is {style}. \
text: ```{text}```
"""
```

In [14]:
```
from langchain.prompts import ChatPromptTemplate

prompt_template = ChatPromptTemplate.from_template(template_string)
```

In [57]:
```
#input_variables: contains list of variables to replace in our prompt
prompt_template.messages[0].prompt
```

```
PromptTemplate(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 not or
unknown.\n\ndelivery_days: How many days did it take for the product to arrive? If
this information is not found, output -1.\n\nprice_value: Extract any sentences ab
out the value or price,and output them as a comma separated Python list.\n\nFormat
the output as JSON with the following keys:\ngift\ndelivery_days\nprice_value\n\nt
ext: {text}\n', template_format='f-string', validate_template=True)
```

In [16]:
```
prompt_template.messages[0].prompt.input_variables
```

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

In [17]:
```
customer_style = """American English \
in a calm and respectful tone
"""
```

In [18]:
```
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 [19]:
```
customer_messages = prompt_template.format_messages(
                    style=customer_style,
                    text=customer_email)
```

In [20]:
```
print(type(customer_messages))
print(type(customer_messages[0]))
```

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

In [21]:
```
print(customer_messages[0])
```

```
content="Translate the text that is delimited by triple backticks into a style tha
t is American English in a calm and respectful tone\n. text: ```\nArrr, I be fumin
g 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 kitche
n. I need yer help right now, matey!\n```\n" additional_kwargs={} example=False
```

In [22]:
```
# Call the LLM to translate to the style of the customer message
customer_response = chat(customer_messages)
```

In [23]:
```
print(customer_response.content)
```

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

In [24]:
```
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 [25]:
```
service_style_pirate = """\
a polite tone \
that speaks in English Pirate\
"""
```

In [26]:
```
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 po
lite tone that speaks in English Pirate. text: ```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 [27]:
```
service_response = chat(service_messages)
print(service_response.content)
```

```
Ahoy there, matey! I must kindly inform ye that the warranty be not coverin' the e
xpenses o' cleaning yer galley, as 'tis yer own fault fer misusin' yer blender by
forgettin' to put the lid on afore startin' it. Aye, tough luck! Farewell and may
the winds be in yer favor!
Ahoy there, matey! I must kindly inform ye that the warranty be not coverin' the e
xpenses o' cleaning yer galley, as 'tis yer own fault fer misusin' yer blender by
forgettin' to put the lid on afore startin' it. Aye, tough 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 [28]:
```
{
  "gift": False,
  "delivery_days": 5,
  "price_value": "pretty affordable!"
}
```

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

In [29]:
```
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 [30]:
```
from langchain.prompts import ChatPromptTemplate

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

```
input_variables=['text'] output_parser=None partial_variables={} messages=[HumanMe
ssagePromptTemplate(prompt=PromptTemplate(input_variables=['text'], output_parser=
None, partial_variables={}, template='For the following text, extract the followin
g information:\n\ngift: Was the item purchased as a gift for someone else? Answer
True if yes, False if not or unknown.\n\ndelivery_days: How many days did it take
for the product to arrive? If this information is not found, output -1.\n\nprice_v
alue: Extract any sentences about the value or price,and output them as a comma se
parated Python list.\n\nFormat the output as JSON with the following keys:\ngift\n
delivery_days\nprice_value\n\ntext: {text}\n', template_format='f-string', validat
e_template=True), additional_kwargs={})]
```

In [31]:
```
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 for the extra features."]
}
```

In [32]:
```
type(response.content)
```

```
str
```

In [33]:
```
# 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[33], 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 [42]:
```
from langchain.output_parsers import ResponseSchema
from langchain.output_parsers import StructuredOutputParser
```

In [43]:
```
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 [44]:
```
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)
```

In [45]:
```
format_instructions = output_parser.get_format_instructions()
```

In [46]:
```
print(format_instructions)
```

```
The output should be a markdown code snippet formatted in the following schema, in
cluding 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 thi
s                                        information is not found,
output -1.
        "price_value": string  // Extract any                                      s
entences about the value or                                        price, and output
them as a                                    comma separated Python list.
}
```
```

In [47]:
```
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 [48]:
```
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 informa
tion is not found, output -1.

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

text: This leaf blower is pretty amazing.  It has four settings:candle blower, gen
tle breeze, windy city, and tornado. It arrived in two days, just in time for my w
ife'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 blo
wers 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, in
cluding 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 thi
s                                      information is not found,
output -1.
        "price_value": string  // Extract any                                    s
entences about the value or                                    price, and output
them as a                                   comma separated Python list.
}
```
```

In [49]:
```
response = chat(messages)
```

In [50]:
```
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 it for the extra features."]
}
```
```

In [51]:
```
output_dict = output_parser.parse(response.content)
```

In [52]:
```
output_dict
```

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

In [53]:
```
type(output_dict)
```

```
dict
```

In [54]:
```
output_dict.get('delivery_days')
```

```
'2'
```

Reminder: Download your notebook to you local computer to save your work.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: