In [10]:
```
#!pip install python-dotenv
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: python-dotenv in c:\users\btina\appdata\roaming\pytho
n\python310\site-packages (1.0.0)
```

In [ ]:
```
#!pip uninstall openai
```

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

In [10]:
```
# python -m pip install openai
#!pip install openai
!pip show openai
```

In [11]:
```
#Get open api key -- from https://platform.openai.com/account/api-keys
# name = LangChain-API-Key
# key =  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

# path where .evn file is located with a new jey OPENAI_API_KEY = C:\Users\btina\ju

import os
import openai
```

In [12]:
```
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
openai.api_key = os.environ['OPENAI_API_KEY']
```

In [13]:
```
# Note: LLM's do not always produce the same results.
# When executing the code in your notebook, you may get slightly different answers
```

In [14]:
```
# 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"
```

In [15]:
```
# Chat API : OpenAI
# Let's start with a direct API calls to OpenAI.
```

In [16]:
```
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 [17]:
```
get_completion("What is 1+1?")
```

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

In [20]:
```
prompt = f"""Translate the text \
that is delimited by triple backticks
into a style that is {style}.
text: ```{customer_email}```
"""
print(prompt)
```

In [21]:
```
response = get_completion(prompt)
```

In [23]:
```
print(response)
```

In [ ]:
```
# Chat API : LangChain
# Let's try how we can do the same using LangChain.
```

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

In [27]:
```
!pip show langchain
```

```
Name: langchain
Version: 0.0.322
Summary: Building applications with LLMs through composability
Home-page: https://github.com/langchain-ai/langchain
Author:
Author-email:
License: MIT
Location: c:\users\btina\appdata\roaming\python\python310\site-packages
Requires: aiohttp, anyio, async-timeout, dataclasses-json, jsonpatch, langsmith, num
py, pydantic, PyYAML, requests, SQLAlchemy, tenacity
Required-by:
```

In [28]:
```
from langchain.chat_models import ChatOpenAI
```

In [29]:
```
# 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
```

Out[29]:
```
ChatOpenAI(client=<class 'openai.api_resources.chat_completion.ChatCompletion'>, model_name='gpt-3.5-turbo-0301', temperature=0.0, openai_api_key='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', openai_api_base='', openai_organization='', openai_proxy='')
```

In [31]:
```
template_string = """Translate the text \
that is delimited by triple backticks \
into a style that is {style}. \
text: ```{text}```
"""

print(template_string)
```

```
Translate the text that is delimited by triple backticks into a style that is {style}. text: ```{text}```
```

In [32]:
```
from langchain.prompts import ChatPromptTemplate

prompt_template = ChatPromptTemplate.from_template(template_string)

print(prompt_template)
```

```
input_variables=['style', 'text'] messages=[HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['style', 'text'], template='Translate the text that is delimited by triple backticks into a style that is {style}. text: ```{text}```\n'))]
```

In [33]:
```
prompt_template.messages[0].prompt
```

Out[33]:
```
PromptTemplate(input_variables=['style', 'text'], template='Translate the text that is delimited by triple backticks into a style that is {style}. text: ```{text}```\n')
```

In [34]:
```
prompt_template.messages[0].prompt.input_variables
```

Out[34]:
```
['style', 'text']
```

In [35]:
```
customer_style = """American English \
in a calm and respectful tone
"""

print(customer_style)
```

```
American English in a calm and respectful tone
```

In [36]:
```
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!
"""

print(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 u
p me kitchen. I need yer help right now, matey!
```

In [37]:
```
customer_messages = prompt_template.format_messages(
                    style=customer_style,
                    text=customer_email)

print(customer_messages)
```

```
[HumanMessage(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 smoothi
e! And to make matters worse, the warranty don't cover the cost of cleaning up me ki
tchen. I need yer help right now, matey!\n```\n")]
```

In [38]:
```
print(type(customer_messages))
print(type(customer_messages[0]))
```

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

In [39]:
```
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 th
at me blender lid flew off and splattered me kitchen walls with smoothie! And to mak
e matters worse, the warranty don't cover the cost of cleaning up me kitchen. I need
yer help right now, matey!\n```\n"
```

In [40]:
```
# Call the LLM to translate to the style of the customer message
customer_response = chat(customer_messages)
```

In [41]:
```
print(customer_response.content)
```

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

In [42]:
```
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!
"""

print(service_reply)
```

```
Hey there customer, the warranty does not cover cleaning expenses for your kitchen b
ecause 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 [43]:
```
service_style_pirate = """\
a polite tone \
that speaks in English Pirate\
"""

print(service_style_pirate)
```

```
a polite tone that speaks in English Pirate
```

In [44]:
```
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 poli
te tone that speaks in English Pirate. text: ```Hey there customer, the warranty doe
s not cover cleaning expenses for your kitchen because it's your fault that you misu
sed your blender by forgetting to put the lid on before starting the blender. Tough
luck! See ya!
```
```

In [45]:
```
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 cov
erin' the expenses o' cleaning yer galley, as 'tis yer own fault fer misusin' yer bl
ender by forgettin' to put the lid on afore startin' it. Aye, tough luck! Farewell a
nd may the winds be in yer favor!
```

In [46]:
```
service_style_spanish = """\
a polite tone \
that speaks in Spanish\
"""

print(service_style_spanish)
```

```
a polite tone that speaks in Spanish
```
```

In [47]:
```
service_messages = prompt_template.format_messages(
    style=service_style_spanish,
    text=service_reply)

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

```
Translate the text that is delimited by triple backticks into a style that is a poli
te tone that speaks in Spanish. text: ```Hey there customer, the warranty does not c
over cleaning expenses for your kitchen because it's your fault that you misused you
r blender by forgetting to put the lid on before starting the blender. Tough luck! S
ee ya!
```
```

In [48]:
```
service_response2 = chat(service_messages)
print(service_response2.content)
```

```
Hola estimado cliente, lamentablemente la garantía no cubre los gastos de limpieza d
e su cocina debido a que fue su culpa haber utilizado incorrectamente la licuadora a
l olvidar poner la tapa antes de encenderla. Lamentamos las molestias. ¡Hasta pront
o!
```

In [49]:
```
#Output Parsers
#Let's start with defining how we would like the LLM output to look like:
```

In [50]:
```
{
  "gift": False,
  "delivery_days": 5,
  "price_value": "pretty affordable!"
}
```

Out[50]:
```
{'gift': False, 'delivery_days': 5, 'price_value': 'pretty affordable!'}
```
```

In [51]:
```
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 [52]:
```
from langchain.prompts import ChatPromptTemplate

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

```
input_variables=['text'] messages=[HumanMessagePromptTemplate(prompt=PromptTemplate
(input_variables=['text'], template='For the following text, extract the following i
nformation:\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: Extr
act any sentences about the value or price,and output them as a comma separated Pyth
on list.\n\nFormat the output as JSON with the following keys:\ngift\ndelivery_days\
nprice_value\n\ntext: {text}\n'))]
```

In [53]:
```
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 th
ere, but I think it's worth it for the extra features."]
}
```

In [54]:
```
type(response.content)
```

Out[54]:
```
str
```

In [55]:
```
# 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[55], 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'
```

In [56]:
```
#Parse the LLM output string into a Python dictionary
```

In [57]:
```
from langchain.output_parsers import ResponseSchema
from langchain.output_parsers import StructuredOutputParser
```

In [58]:
```
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 [59]:
```
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)
```

In [60]:
```
format_instructions = output_parser.get_format_instructions()
```

In [61]:
```
print(format_instructions)
```

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

```json
{
        "gift": string  // Was the item purchased                              as a g
ift 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                                      sen
tences about the value or                                        price, and output them
as a                                        comma separated Python list.
}
```
```

In [62]:
```
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 [63]:
```
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 i
f not or unknown.

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

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

text: This leaf blower is pretty amazing.  It has four settings:candle blower, gentl
e 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.


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

```json
{
        "gift": string  // Was the item purchased                              as a g
ift 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                                    sen
tences about the value or                                  price, and output them
as a                                    comma separated Python list.
}
```
```

In [64]:
```
response = chat(messages)
```

In [65]:
```
print(response.content)
```

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

In [66]:
```
output_dict = output_parser.parse(response.content)
```

In [67]:
```
print(output_dict)
```

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

In [68]:
```
type(output_dict)
```

Out[68]:
```
dict
```

In [69]:
```
output_dict.get('delivery_days')
```

Out[69]:
```
'2'
```

In [ ]:
```
# THE END!
```