Entity Extraction with Claude

> *This notebook should work well with the* ***`Python 3`*** *kernel in SageMaker Studio*

Context

Entity extraction is an NLP technique that allows us to automatically extract specific data from naturally written text, such as news, emails, books, etc. That data can then later be saved to a database, used for lookup or any other type of processing.

Classic entity extraction programs usually limit you to pre-defined classes, such as name, address, price, etc. or require you to provide many examples of types of entities you are interested in. By using a LLM for entity extraction in most cases you are only required to specify what you need to extract in natural language. This gives you flexibility and accuracy in your queries while saving time by removing necessity of data labeling.

In addition, LLM entity extraction can be used to help you assemble a dataset to later create a customised solution for your use case, such as Amazon Comprehend custom entity recognition.

Setup

In [1]:
```
%pip install -U langchain-aws=='0.1.17'
```

```
Collecting langchain-aws==0.1.17
  Downloading langchain_aws-0.1.17-py3-none-any.whl (82 kB)
     |████████████████████████████████| 82 kB 1.8 MB/s eta 0:00:011
Collecting boto3<1.35.0,>=1.34.131
  Downloading boto3-1.34.162-py3-none-any.whl (139 kB)
     |████████████████████████████████| 139 kB 33.9 MB/s eta 0:00:01
Collecting numpy<2,>=1
  Downloading numpy-1.26.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(18.2 MB)
     |████████████████████████████████| 18.2 MB 84.3 MB/s eta 0:00:01
Collecting langchain-core<0.3,>=0.2.33
  Downloading langchain_core-0.2.43-py3-none-any.whl (397 kB)
     |████████████████████████████████| 397 kB 101.3 MB/s eta 0:00:01
Collecting botocore<1.35.0,>=1.34.162
  Downloading botocore-1.34.162-py3-none-any.whl (12.5 MB)
     |████████████████████████████████| 12.5 MB 77.6 MB/s eta 0:00:01
Requirement already satisfied: s3transfer<0.11.0,>=0.10.0 in /home/studio-lab-user/.
conda/envs/default/lib/python3.9/site-packages (from boto3<1.35.0,>=1.34.131->langch
ain-aws==0.1.17) (0.10.4)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /home/studio-lab-user/.cond
a/envs/default/lib/python3.9/site-packages (from boto3<1.35.0,>=1.34.131->langchain-
aws==0.1.17) (1.0.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/studio-lab-user/.cond
a/envs/default/lib/python3.9/site-packages (from botocore<1.35.0,>=1.34.162->boto3<
1.35.0,>=1.34.131->langchain-aws==0.1.17) (1.26.20)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /home/studio-lab-use
r/.conda/envs/default/lib/python3.9/site-packages (from botocore<1.35.0,>=1.34.162->
boto3<1.35.0,>=1.34.131->langchain-aws==0.1.17) (2.9.0)
Collecting pydantic<3,>=1
  Downloading pydantic-2.10.1-py3-none-any.whl (455 kB)
     |████████████████████████████████| 455 kB 74.2 MB/s eta 0:00:01
Collecting jsonpatch<2.0,>=1.33
  Downloading jsonpatch-1.33-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: PyYAML>=5.3 in /home/studio-lab-user/.conda/envs/defa
ult/lib/python3.9/site-packages (from langchain-core<0.3,>=0.2.33->langchain-aws==0.
1.17) (6.0.1)
Collecting tenacity!=8.4.0,<9.0.0,>=8.1.0
  Downloading tenacity-8.5.0-py3-none-any.whl (28 kB)
Requirement already satisfied: packaging<25,>=23.2 in /home/studio-lab-user/.conda/e
nvs/default/lib/python3.9/site-packages (from langchain-core<0.3,>=0.2.33->langchain
-aws==0.1.17) (24.0)
Collecting langsmith<0.2.0,>=0.1.112
  Downloading langsmith-0.1.145-py3-none-any.whl (310 kB)
     |████████████████████████████████| 310 kB 33.1 MB/s eta 0:00:01
Requirement already satisfied: typing-extensions>=4.7 in /home/studio-lab-user/.cond
a/envs/default/lib/python3.9/site-packages (from langchain-core<0.3,>=0.2.33->langch
ain-aws==0.1.17) (4.11.0)
Requirement already satisfied: jsonpointer>=1.9 in /home/studio-lab-user/.conda/env
s/default/lib/python3.9/site-packages (from jsonpatch<2.0,>=1.33->langchain-core<0.
3,>=0.2.33->langchain-aws==0.1.17) (2.4)
Collecting orjson<4.0.0,>=3.9.14
  Downloading orjson-3.10.11-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l (142 kB)
     |████████████████████████████████| 142 kB 72.8 MB/s eta 0:00:01
Requirement already satisfied: httpx<1,>=0.23.0 in /home/studio-lab-user/.conda/env
s/default/lib/python3.9/site-packages (from langsmith<0.2.0,>=0.1.112->langchain-cor
```

```
e<0.3,>=0.2.33->langchain-aws==0.1.17) (0.27.0)
Collecting requests-toolbelt<2.0.0,>=1.0.0
  Downloading requests_toolbelt-1.0.0-py2.py3-none-any.whl (54 kB)
     |████████████████████████████████| 54 kB 6.1 MB/s  eta 0:00:01
Requirement already satisfied: requests<3,>=2 in /home/studio-lab-user/.conda/envs/d
efault/lib/python3.9/site-packages (from langsmith<0.2.0,>=0.1.112->langchain-core<
0.3,>=0.2.33->langchain-aws==0.1.17) (2.32.3)
Requirement already satisfied: sniffio in /home/studio-lab-user/.conda/envs/default/
lib/python3.9/site-packages (from httpx<1,>=0.23.0->langsmith<0.2.0,>=0.1.112->langc
hain-core<0.3,>=0.2.33->langchain-aws==0.1.17) (1.3.1)
Requirement already satisfied: anyio in /home/studio-lab-user/.conda/envs/default/li
b/python3.9/site-packages (from httpx<1,>=0.23.0->langsmith<0.2.0,>=0.1.112->langcha
in-core<0.3,>=0.2.33->langchain-aws==0.1.17) (4.3.0)
Requirement already satisfied: httpcore==1.* in /home/studio-lab-user/.conda/envs/de
fault/lib/python3.9/site-packages (from httpx<1,>=0.23.0->langsmith<0.2.0,>=0.1.112-
>langchain-core<0.3,>=0.2.33->langchain-aws==0.1.17) (1.0.5)
Requirement already satisfied: idna in /home/studio-lab-user/.conda/envs/default/li
b/python3.9/site-packages (from httpx<1,>=0.23.0->langsmith<0.2.0,>=0.1.112->langcha
in-core<0.3,>=0.2.33->langchain-aws==0.1.17) (3.7)
Requirement already satisfied: certifi in /home/studio-lab-user/.conda/envs/default/
lib/python3.9/site-packages (from httpx<1,>=0.23.0->langsmith<0.2.0,>=0.1.112->langc
hain-core<0.3,>=0.2.33->langchain-aws==0.1.17) (2024.8.30)
Requirement already satisfied: h11<0.15,>=0.13 in /home/studio-lab-user/.conda/envs/
default/lib/python3.9/site-packages (from httpcore==1.*->httpx<1,>=0.23.0->langsmith
<0.2.0,>=0.1.112->langchain-core<0.3,>=0.2.33->langchain-aws==0.1.17) (0.14.0)
Collecting pydantic-core==2.27.1
  Downloading pydantic_core-2.27.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (2.1 MB)
     |████████████████████████████████| 2.1 MB 64.8 MB/s eta 0:00:01
Collecting typing-extensions>=4.7
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Collecting annotated-types>=0.6.0
  Downloading annotated_types-0.7.0-py3-none-any.whl (13 kB)
Requirement already satisfied: six>=1.5 in /home/studio-lab-user/.conda/envs/defaul
t/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.35.0,>=
1.34.162->boto3<1.35.0,>=1.34.131->langchain-aws==0.1.17) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/studio-lab-user/.co
nda/envs/default/lib/python3.9/site-packages (from requests<3,>=2->langsmith<0.2.0,>
=0.1.112->langchain-core<0.3,>=0.2.33->langchain-aws==0.1.17) (3.3.2)
Requirement already satisfied: exceptiongroup>=1.0.2 in /home/studio-lab-user/.cond
a/envs/default/lib/python3.9/site-packages (from anyio->httpx<1,>=0.23.0->langsmith<
0.2.0,>=0.1.112->langchain-core<0.3,>=0.2.33->langchain-aws==0.1.17) (1.2.0)
Installing collected packages: typing-extensions, pydantic-core, annotated-types, re
quests-toolbelt, pydantic, orjson, botocore, tenacity, langsmith, jsonpatch, numpy,
langchain-core, boto3, langchain-aws
  Attempting uninstall: typing-extensions
    Found existing installation: typing-extensions 4.11.0
    Uninstalling typing-extensions-4.11.0:
      Successfully uninstalled typing-extensions-4.11.0
  Attempting uninstall: botocore
    Found existing installation: botocore 1.35.68
    Uninstalling botocore-1.35.68:
      Successfully uninstalled botocore-1.35.68
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
```

```
    Successfully uninstalled numpy-2.0.2
  Attempting uninstall: boto3
    Found existing installation: boto3 1.35.68
    Uninstalling boto3-1.35.68:
      Successfully uninstalled boto3-1.35.68
ERROR: pip's dependency resolver does not currently take into account all the packag
es that are installed. This behaviour is the source of the following dependency conf
licts.
awscli 1.36.9 requires botocore==1.35.68, but you have botocore 1.34.162 which is in
compatible.
Successfully installed annotated-types-0.7.0 boto3-1.34.162 botocore-1.34.162 jsonpa
tch-1.33 langchain-aws-0.1.17 langchain-core-0.2.43 langsmith-0.1.145 numpy-1.26.4 o
rjson-3.10.11 pydantic-2.10.1 pydantic-core-2.27.1 requests-toolbelt-1.0.0 tenacity-
8.5.0 typing-extensions-4.12.2
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import json
import os
import sys

import boto3
import botocore

boto3_bedrock = boto3.client('bedrock-runtime')
```

Configure langchain

We begin with instantiating the LLM. Here we are using Anthropic Claude v3 for text generation.

Note: It is possible to choose other models available with Bedrock. For example, you can replace the `model_id` as follows to change the model to Titan Text Premier. Make sure your account has access to the model you want to try out before trying this!

```
llm = ChatBedrock(model_id="amazon.titan-text-premier-v1:0")
```

Check documentation for Available text generation model Ids under Amazon Bedrock.

In [3]:
```
from langchain_aws import ChatBedrock

llm = ChatBedrock(
    model_id="anthropic.claude-3-sonnet-20240229-v1:0",
    model_kwargs={
        "max_tokens": 200,
        "temperature": 0, # Using 0 to get reproducible results
        "stop_sequences": ["\n\nHuman:"]
    }
)
```

Entity Extraction

Now that we have our LLM initialised, we can start extracting entities.

For this exercise we will pretend to be an online bookstore that receives questions and orders by email. Our task would be to extract relevant information from the email to process the order.

Let's begin by taking a look at the sample email:

In [5]:
```
from pathlib import Path

emails_dir = Path(".") / "emails"
with open(emails_dir / "00_treasure_island.txt") as f:
    book_question_email = f.read()

print(book_question_email)
```

```
Dear Sir or Madam,

I would like to order Treasure Island, do you have it in stock?

Also, is it possible to pay by cheque?

Yours sincerely,
John Smith
```

Basic approach

For basic cases we can directly ask the model to return the result. Let's try extracting the name of the book.

In [6]:
```
query = f"""
Given the email inside triple-backticks, please read it and analyse the contents.
If a name of a book is mentioned, return it, otherwise return nothing.

Email: ```
{book_question_email}
```

"""

messages = [
    (
        "system",
        "You are a helpful assistant that processes orders from a bookstore.",
    ),
    ("human", query),
]
```

In [7]:
```
result = llm.invoke(messages)
print(result.content)
```

```
The name of the book mentioned in the email is "Treasure Island".
```
```

Model specific prompts

While basic approach works, to achieve best results we recommend to customise your prompts for the particular model you will be using. In this example we are using `anthropic.claude-3`, prompt guide for which can be found here.

Here is the a more optimised prompt for Claude v3.

In [8]:
```
prompt = """

Given the email provided, please read it and analyse the contents.
If a name of a book is mentioned, return it.
If no name is mentioned, return empty string.
The email will be given between <email></email> XML tags.

<email>
{email}
</email>

Return the name of the book between <book></book> XML tags.

"""
```

In [9]:
```
query = prompt.format(email=book_question_email)
messages = [
    (
        "system",
        "You are a helpful assistant that processes orders from a bookstore.",
    ),
    ("human", query),
]
result = llm.invoke(messages).content
print(result)
```

```
<book>Treasure Island</book>
```

To extract results easier, we can use a helper function:

In [15]:
```
from bs4 import BeautifulSoup
from typing import Union, List

def extract_by_tag(response: str, tag: str, extract_all=False) -> Union[str, List[s
    soup = BeautifulSoup(response, 'html.parser')  # Especifica el parser para evit
    if extract_all:
        results = soup.find_all(tag)
        return [result.get_text() for result in results]
    else:
        result = soup.find(tag)
        return result.get_text() if result else None
```

In [16]:
```
extract_by_tag(result, "book")
```

Out[16]: 'Treasure Island'

We can check that our model doesn't return arbitrary results when no appropriate information is given (also know as 'hallucination'), by running our prompt on other emails.

In [17]:
```
with open(emails_dir / "01_return.txt") as f:
    return_email = f.read()

print(return_email)
```

```
I didn't like the last book I ordered and would like to return it.
```

In [18]:
```
query = prompt.format(email=return_email)
messages = [
    (
        "system",
        "You are a helpful assistant that processes orders from a bookstore.",
    ),
    ("human", query),
]
result = llm.invoke(query).content
print(result)
```

```
<book></book>
```

Using tags also allows us to extract multiple pieces of information at the same time and makes extraction much easier. In the following prompt we will extract not just the book name, but any questions, requests and customer name.

In [19]:
```
prompt = """

Human: Given email provided , please read it and analyse the contents.

Please extract the following information from the email:
- Any questions the customer is asking, return it inside <questions></questions> XM
- The customer full name, return it inside <name></name> XML tags.
- Any book names the customer mentions, return it inside <books></books> XML tags.

If a particular bit of information is not present, return an empty string.
Make sure that each question can be understoon by itself, incorporate context if re
Each returned question should be concise, remove extra information if possible.
The email will be given between <email></email> XML tags.

<email>
{email}
</email>

Return each question inside <question></question> XML tags.
Return the name of each book inside <book></book> XML tags.

Assistant:"""
```

In [20]:
```
query = prompt.format(email=book_question_email)
messages = [
```

```
    (
        "system",
        "You are a helpful assistant that processes orders from a bookstore.",
    ),
    ("human", query),
]
result = llm.invoke(query).content
print(result)
```

```
<questions>
<question>Do you have Treasure Island in stock?</question>
<question>Is it possible to pay by cheque?</question>
</questions>
<name>John Smith</name>
<books>
<book>Treasure Island</book>
</books>
```

In [21]:
```
extract_by_tag(result, "question", extract_all=True)
```

Out[21]:
```
['Do you have Treasure Island in stock?', 'Is it possible to pay by cheque?']
```

In [22]:
```
extract_by_tag(result, "name")
```

Out[22]:
```
'John Smith'
```

In [23]:
```
extract_by_tag(result, "book", extract_all=True)
```

Out[23]:
```
['Treasure Island']
```

Conclusion

Entity extraction is a powerful technique using which you can extract arbitrary data using plain text descriptions.

This is particularly useful when you need to extract specific data which doesn't have clear structure. In such cases regex and other traditional extraction techniques can be very difficult to implement.

Take aways

- Adapt this notebook to experiment with different models available through Amazon Bedrock such as Amazon Titan and AI21 Labs Jurassic models.
- Change the prompts to your specific usecase and evaluate the output of different models.
- Apply different prompt engineering principles to get better outputs. Refer to the prompt guide for your chosen model for recommendations, e.g. here is the prompt guide for Claude.

In [ ]: