

✓ Build an LLM Analytics Pipeline in Python

Before you get started: **File/Save As**. You will need a copy of this notebook in your own google drive.

In this notebook, you will learn to build a pipeline to interact with an LLM in ipython. You may recall that ipython is interactive python. Tools like jupyter and Google Colab are ipython environments. Our LLM pipeline will use a few different packages.

- [Ollama](#): A google product that runs in the background to impliment LLM models
- [Langchain](#): A toolset that helps build LLM pipelines. It takes care of all the steps to prepare our inputs for the LLM model, and ready our outputs for the next step.
- [xterm](#): Programs such as ollama are services that run on a computer, virtual machine, or cloud environment. We'll use the xterm terminal to launch ollama in a virtual machine behind the scenes. Then we can connect to ollama from our python environment.
- [NLTK](#): The Natural Language Toolkit includes many methods to perform analysis on unstructured text.

Graded Assignment: After running the code you will be equipped with the fundamentals of creating an LLM Analytics pipeline. Complete the section at the end of this notebook called *Graded Assignment*.

Pro Tip: Google colab offers a limited amount of computer resources at the free tier. While you will have enough compute to run this assignment, it is best to shut down the session when not actively running code. Conserve compute resources. LLM text generation will run in several seconds if you [enable a GPU runtime in colab](#), but will run in several minutes if you enable a CPU runtime.

```
import os
import nltk
```

✓ Install and launch Xterm

xterm is a package that enables you to open a terminal within a Colab notebook cell. This can be particularly useful for tasks that are more easily done in a terminal environment, such as:

- **Running shell commands:** You can execute any command that you'd normally run in a terminal, like installing packages or running system utilities.
- **Managing files:** You can use commands like ls, cd, and mkdir to navigate and manipulate files in the Colab environment.
- **Running background processes:** You can start processes in the background using commands like nohup or screen, which is essential for tasks that run concurrently with your python tasks.

```
!pip install colab-xterm #https://pypi.org/project/colab-xterm/
%load_ext colabxterm
```

```
Collecting colab-xterm
  Downloading colab_xterm-0.2.0-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: ptyprocess~=0.7.0 in /usr/local/lib/python3.11/dist-packages (from colab-xterm) (0.7.0)
Requirement already satisfied: tornado>5.1 in /usr/local/lib/python3.11/dist-packages (from colab-xterm) (6.4.2)
Downloading colab_xterm-0.2.0-py3-none-any.whl (115 kB)
----- 115.6/115.6 kB 3.5 MB/s eta 0:00:00
Installing collected packages: colab-xterm
Successfully installed colab-xterm-0.2.0
```

✓ Use Xterm terminal to complete tasks outside of python

Now we will leverage a command terminal to launch the ollama service in the background. This service is required for us to use LLM's in python.

1. Download [ollama](#) application to the virtual machine `curl https://ollama.ai/install.sh | sh`
2. Initiate the [ollama](#) service on the virtual machine `ollama serve &`
3. Download the [Mistral](#) llm model file to the virtual machine `ollama pull mistral`

```
%xterm # this command launches xterm
# # curl https://ollama.ai/install.sh | sh #note: its easiest to run this command within this cell. Not necessary to past
# # ollama serve & #copy/paste/return in xterm
# # ollama pull mistral #copy/paste/return xterm
!curl https://ollama.ai/install.sh | sh #This command is uncommented so it will run with the cell. Not necessary to also pa
```

```

Launching Xterm...
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 13281    0 13281    0     0  54263      0 --:--:-- --:--:-- --:--:-- 54430
>>> Installing ollama to /usr/local
>>> Downloading Linux amd64 bundle
##### 100.0%
>>> Creating ollama user...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
WARNING: systemd is not running
WARNING: Unable to detect NVIDIA/AMD GPU. Install lspci or lshw to automatically detect and install GPU dependencies.
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.

```

Verify that the mistral model is available for use. We expect to see details about the 7.2B Mistral model below.

```
!ollama show mistral
```

```
Error: could not connect to ollama app, is it running?
```

✓ Setup Langchain

Great! Your LLM model is setup to use. Now we will install and import the langchain package to help setup an LLM pipeline. Langchain helps us interact with the LLM in a more practical, manageable way.

```
%pip install -U langchain-ollama
```

```

Collecting langchain-ollama
  Downloading langchain_ollama-0.3.0-py3-none-any.whl.metadata (1.5 kB)
Collecting ollama<1,>=0.4.4 (from langchain-ollama)

```

```

Downloading ollama-0.4.7-py3-none-any.whl.metadata (4.7 kB)
Requirement already satisfied: langchain-core<1.0.0,>=0.3.47 in /usr/local/lib/python3.11/dist-packages (from langchain-olla
Requirement already satisfied: langsmith<0.4,>=0.1.125 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0
Requirement already satisfied: tenacity!=8.4.0,<10.0.0,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from langchain-co
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,>=
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,>=0.3.47->l
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,>=0
Requirement already satisfied: typing-extensions>=4.7 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,
Requirement already satisfied: pydantic<3.0.0,>=2.5.2 in /usr/local/lib/python3.11/dist-packages (from langchain-core<1.0.0,
Requirement already satisfied: httpx<0.29,>=0.27 in /usr/local/lib/python3.11/dist-packages (from ollama<1,>=0.4.4->langchai
Requirement already satisfied: anyio in /usr/local/lib/python3.11/dist-packages (from httpx<0.29,>=0.27->ollama<1,>=0.4.4->l
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx<0.29,>=0.27->ollama<1,>=0.4.4-
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx<0.29,>=0.27->ollama<1,>=
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from httpx<0.29,>=0.27->ollama<1,>=0.4.4->la
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx<0.29,>=
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=1.33->langc
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.12
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.125->lang
Requirement already satisfied: requests-toolbelt<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.
Requirement already satisfied: zstandard<0.24.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.5.
Requirement already satisfied: pydantic-core==2.33.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.5.2
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2->lan
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2->langsmith
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio->httpx<0.29,>=0.27->ollam
Downloading langchain_ollama-0.3.0-py3-none-any.whl (20 kB)
Downloading ollama-0.4.7-py3-none-any.whl (13 kB)
Installing collected packages: ollama, langchain-ollama
Successfully installed langchain-ollama-0.3.0 ollama-0.4.7

```

▼ Generate Text

Langchain expects a [template](#) to structure inputs and outputs of the LLM. By convention, we will refer to our model inputs as questions, and outputs as answers. The LLM will use the template information as it generates a response. The phrase "Let's think step by step" is designed to encourage the language model to provide a reasoned, logical response. This prompting approach is called [Chain of Thought](#). You can change the text in the template, and the model behavior may change.

```
#Here we read in the langchain library. Langchain is the toolset we use to build an LLM data pipeline Use the gemini AI feature
from langchain_core.prompts import ChatPromptTemplate
from langchain_ollama.llms import OllamaLLM
```

```
template = """Question: {question}
```

```
Answer: Let's think step by step."""
```

```
prompt = ChatPromptTemplate.from_template(template)
```

```
model = OllamaLLM(model="mistral") #mistral is one of the LLM models available in langchain
```

```
chain = prompt | model
```

With langchain setup to integrate our inputs the LLM, and the outputs back to python environment, we are ready to prompt the LLM model. Then we will print the output of the LLM. Note: text generation runs in several seconds if GPU is enabled on colab. It may take several minutes if CPU only is enabled.

```
gen_string = chain.invoke({"question": "What years did Boston College Mens Hockey win the National Championship"})
print (gen_string)
```

```
#Example generatd string using Langchain template
```



1. First, I will search for the list of Boston College Men's Hockey National Championships.
2. The search results show that Boston College has won the NCAA Division I Men's Ice Hockey Championship four times:
 - 1949 (Frozen Four)
 - 2001
 - 2008
 - 2012
3. So, the years when Boston College Mens Hockey won the National Championship were 1949, 2001, 2008, and 2012.

▼ Import the Natural Language Toolkit to characterize the response.

Great! So far we use the langchain framework to interact with the mistral model running on the ollama service. Now that we have some text saved to the variable `gen_string`, we can use NLTK (Natural language toolkit) to do some analysis.

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

Some tools in NLTK don't install with the baseline package. NLTK has an integrated `download` function to pull in anything else you may need, and save it to your system. In this case, the package files will be downloaded to your google drive.

```
! python -m nltk.downloader -d /usr/local/share/nltk_data all
```

```
<frozen runpy>:128: RuntimeWarning: 'nltk.downloader' found in sys.modules after import of package 'nltk', but prior to exec
[nltk_data] Downloading collection 'all'
[nltk_data] |
[nltk_data] | Downloading package abc to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/abc.zip.
[nltk_data] | Downloading package alpino to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/alpino.zip.
[nltk_data] | Downloading package averaged_perceptron_tagger to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] | Downloading package averaged_perceptron_tagger_eng to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping
[nltk_data] | taggers/averaged_perceptron_tagger_eng.zip.
[nltk_data] | Downloading package averaged_perceptron_tagger_ru to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping
[nltk_data] | taggers/averaged_perceptron_tagger_ru.zip.
[nltk_data] | Downloading package averaged_perceptron_tagger_rus to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping
[nltk_data] | taggers/averaged_perceptron_tagger_rus.zip.
[nltk_data] | Downloading package basque_grammars to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping grammars/basque_grammars.zip.
[nltk_data] | Downloading package bcp47 to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Downloading package biocreative_ppi to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/biocreative_ppi.zip.
[nltk_data] | Downloading package blip_wsj_no_aux to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping models/blip_wsj_no_aux.zip.
[nltk_data] | Downloading package book_grammars to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping grammars/book_grammars.zip.
[nltk_data] | Downloading package brown to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/brown.zip.
[nltk_data] | Downloading package brown_tei to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/brown_tei.zip.
[nltk_data] | Downloading package cess_cat to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/cess_cat.zip.
[nltk_data] | Downloading package cess_esp to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/cess_esp.zip.
[nltk_data] | Downloading package chat80 to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/chat80.zip.
[nltk_data] | Downloading package city_database to
[nltk_data] | /usr/local/share/nltk_data...
[nltk_data] | Unzipping corpora/city_database.zip.
[nltk_data] | Downloading package cmudict to
[nltk_data] | /usr/local/share/nltk_data...
```

✓ Sentiment Analysis

NLTK can provide sentiment analysis of text data. Let's investigate the *sentiment* of the generated text.

```
# Initialize the sentiment analyzer
analyzer = SentimentIntensityAnalyzer()
```

```
# Perform sentiment analysis
scores = analyzer.polarity_scores(gen_string)
```

```
# Print the sentiment scores
print(scores)
```

```
{'neg': 0.0, 'neu': 0.764, 'pos': 0.236, 'compound': 0.9468}
```

If you've made it this far, you're doing great. We have used NLTK to perform sentiment analysis of the generated text. Now visualize this with ggplot. You may use gen AI or your own creativity to improve on this basic plot.

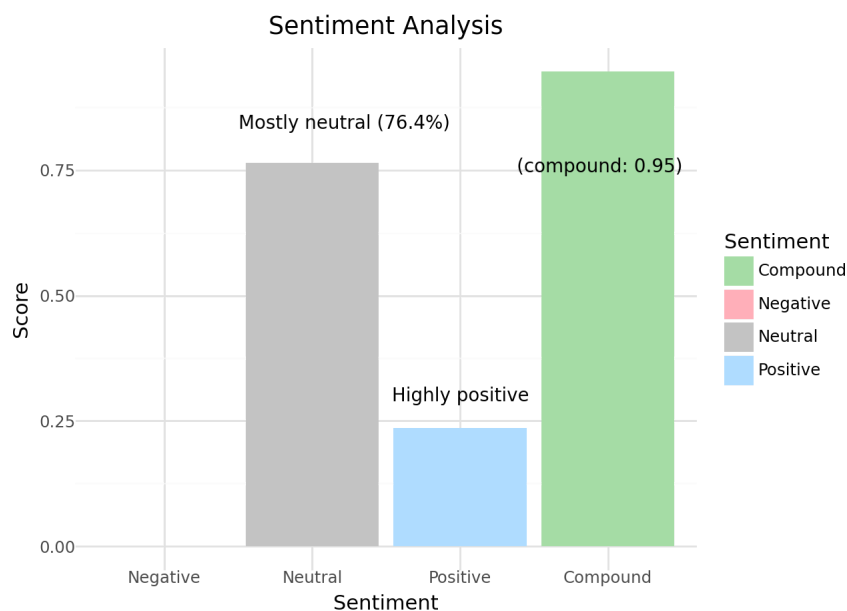
```
from plotnine import ggplot, aes, geom_bar, scale_fill_manual, labs, theme_minimal, scale_x_discrete, annotate, theme
import pandas as pd
```

```
# Create DataFrame with ordered sentiment categories
df = pd.DataFrame({
    'Sentiment': ['Negative', 'Neutral', 'Positive', 'Compound'],
    'Score': [scores['neg'], scores['neu'], scores['pos'], scores['compound']]
})
```

```
# Define color palette
color_scheme = {
    'Negative': '#FFB3BA', # Soft red (muted)
    'Neutral': '#C4C4C4', # Light gray
    'Positive': '#B3E0FF', # Light blue
    'Compound': '#A7DCA7' # Soft green
}
```

```
# Create the ggplot-style bar chart
plot = (
    ggplot(df, aes(x='Sentiment', y='Score', fill='Sentiment'))
    + geom_bar(stat='identity')
    + scale_fill_manual(values=color_scheme)
    + scale_x_discrete(limits=['Negative', 'Neutral', 'Positive', 'Compound']) # Ensures order
    + labs(
        title='Sentiment Analysis',
        x='Sentiment',
        y='Score',
        caption="The sentiment analysis of this text indicates a positive and strong tone."
    )
    + annotate(
        "text", x=1.5, y=0.8, label="Mostly neutral (76.4%) \n \n (compound: 0.95) ",
        size=10, ha='left'
    )
    + annotate(
        "text", x=2.5, y=0.3, label=" Highly positive ",
        size=10, ha='left'
    )
    + theme_minimal()
)
```

```
plot
```



Frequency Distribution

Now let's analyze the frequency distribution of words in the generated text. The first thing we'll do is tokenize the string with NLTK's built-in tokenizer.

```
tokens = nltk.word_tokenize(gen_string)
print(tokens)
```

```
['I', '.', 'First', ',', 'I', 'will', 'search', 'for', 'the', 'list', 'of', 'Boston', 'College', 'Men', "'s", 'Hockey', 'Nat
```

```
from nltk import FreqDist
```

```
# Calculate frequency distribution
fdist = FreqDist(tokens)
```

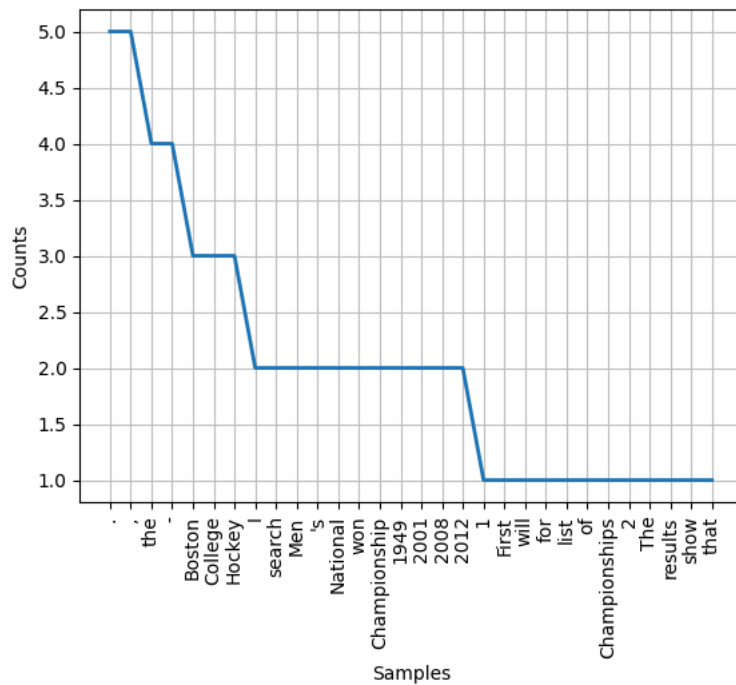
```
# Print the most common words
print(fdist.most_common(10)) # Prints the 10 most common words
```

```
# Plot the frequency distribution
fdist.plot(30, cumulative=False) # Plots the 30 most common words
```

```

[[('.', 5), (',', 5), ('the', 4), ('-', 4), ('Boston', 3), ('College', 3), ('Hockey', 3), ('I', 2), ('search', 2), ('Men', 2)]
<Axes: xlabel='Samples', ylabel='Counts'>

```



Parts of Speech Analysis

Next we will analyze the [parts of speech](#) in the generated text. To get started, we must perform tagging.

```

# Perform POS tagging
tagged_tokens = nltk.pos_tag(tokens)
print("The output of tagging is a: ", type(tagged_tokens))

```

```

The output of tagging is a: <class 'list'>


```

Let's plot the parts-of-speech distribution for this text.

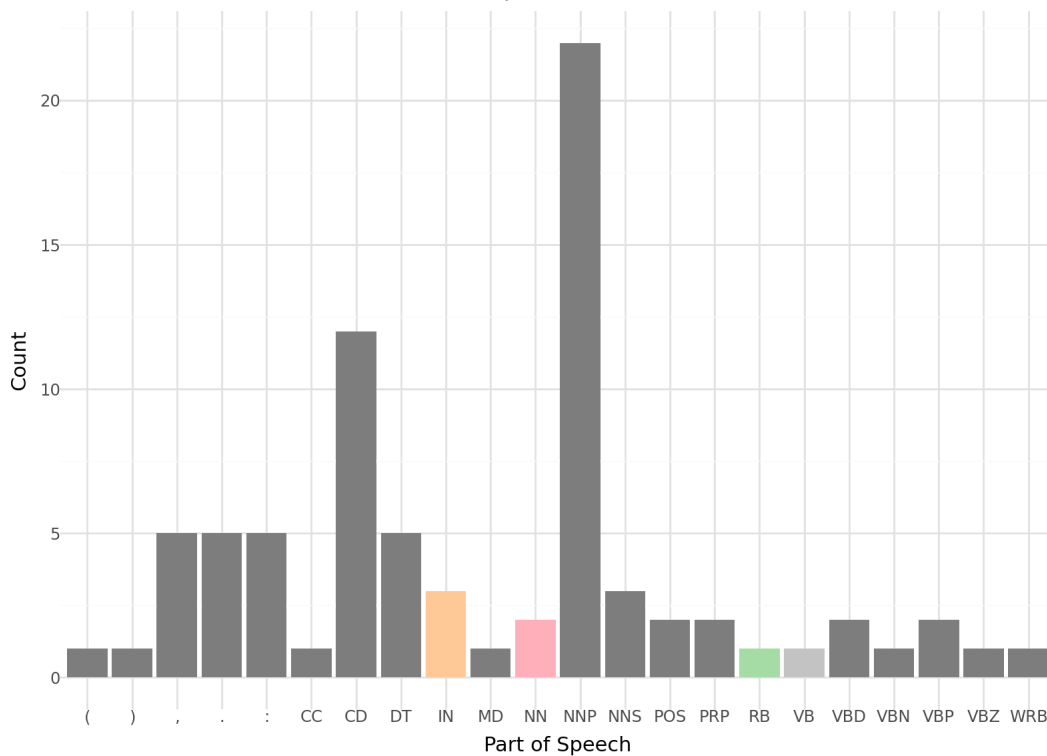
```

plot = (
    ggplot(pos_counts, aes(x='POS', y='Count', fill='POS'))
    + geom_bar(stat='identity')
    + scale_fill_manual(values=color_scheme)
    + labs(
        title='Parts of Speech Distribution',
        x='Part of Speech',
        y='Count'
    )
    + theme_minimal()
    + theme(
        legend_position='none', # Remove the legend
        axis_text_x=theme.text(angle=45, hjust=1), # Rotate x-axis labels
        figure_size=(8, 6) # Increase the plot width
    )
)
plot

```

 /usr/local/lib/python3.11/dist-packages/plotnine/scales/scale_manual.py:39: PlotnineWarning: The palette of scale_fill_manua

Parts of Speech Distribution



✓ Bleu Score: Quantify the similarity of generated text to reference data

Let's use the wikipedia API to pull in truth data about BC Hockey. We will use this truth data as a baseline to assess the correctness generated text.

```
!pip install wikipedia-api
```

```
Collecting wikipedia-api
  Downloading wikipedia_api-0.8.1.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from wikipedia-api) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->wikipedia-api) (3.10)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->wikipedia-api) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->wikipedia-api) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->wikipedia-api) (2024.7.4)
Building wheels for collected packages: wikipedia-api
  Building wheel for wikipedia-api (setup.py) ... done
  Created wheel for wikipedia-api: filename=Wikipedia_API-0.8.1-py3-none-any.whl size=15383 sha256=223bd318ce823556239c3d00c
  Stored in directory: /root/.cache/pip/wheels/0b/0f/39/e8214ec038ccd5aeb8c82b957289f2f3ab2251febae5c2860
Successfully built wikipedia-api
Installing collected packages: wikipedia-api
Successfully installed wikipedia-api-0.8.1
```

```
import wikipediaapi
# Initialize the Wikipedia API
wiki_wiki = wikipediaapi.Wikipedia('english') #use 'english' 'simple english' or any other supported language

# Specify the Wikipedia page to search for
page_to_search = "Boston College Eagles men's ice hockey"

# Obtain the page and save to variable
page = wiki_wiki.page(page_to_search)

#Preview the content
if page.exists():
    # Print the text of the page
    print(page.text[0:500]) #just 500 character
```



```

#This is the truth data
runner_up_truth = page.section_by_title('National Championships') #section name 'Championships'
print(runner_up_truth)

```

↪ The Boston College Eagles are an NCAA Division I college ice hockey program that represents Boston College in Chestnut Hill,

✓ Save the truth (reference) data

We will focus our analysis to the *Section* of the page called *National Championships* and report the dates BC was runner up.

```

#This is the truth data
runner_up_truth = page.section_by_title('National Championships') #section name 'Championships'
print(runner_up_truth)

```

```

↪ Section: National Championships (2):
Runners-up in 1965, 1978, 1998, 2000, 2006, 2007 and 2024.
Subsections (0):

```

```

#Prepare the truth data (reference) as a list
reference = runner_up_truth.text.split()
reference

```

```

↪ ['Runners-up',
   'in',
   '1965,',
   '1978,',
   '1998,',
   '2000,',
   '2006,',
   '2007',
   'and',
   '2024.']

```

✓ Generate test data

We use the langchain template from above, but update the question.

```

runner_up_gen = chain.invoke({"question": "Concisely answer: What years was Boston College Mens Hockey the runner up for the Na
print(runner_up_gen)

```

```

↪ 1949 and 2010

```

```

#Prepare generated text as a list
generated = runner_up_gen.split()
generated

```

```

↪ ['1949', 'and', '2010']

```

Use the metric *Bleu score* to analyze the similarity of the generated data to the reference data. This is a quantitative approach to measuring the correctness of generated text.

```

from nltk.translate.bleu_score import sentence_bleu

```

```

score = sentence_bleu([reference], generated)
score

```

```

↪ /usr/local/lib/python3.11/dist-packages/nltk/translate/bleu_score.py:577: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
/usr/local/lib/python3.11/dist-packages/nltk/translate/bleu_score.py:577: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.

```