Information Retrieval using Speech Data

Name: Jessica Anna James

Phone Number: 4256558646

james.jess@northeastern.edu

Signature of Student: Jessica Anna James

Submission Date: 12/21/2023

Introduction:

In the era of information overload, the task of deciphering valuable insights from spoken content has become increasingly intricate. This report delves into the realm of Natural Language Processing (NLP) methodologies designed to distill meaningful information from a dynamic source: a YouTube video (https://www.youtube.com/watch?v=P73KmleCuBg). The initial phase entails the crucial process of data collection, wherein cutting-edge speech-to-text algorithms or APIs are deployed to transmute the spoken dialogue into a structured text format. To enhance the quality of the data, subsequent text preprocessing techniques are employed, systematically eliminating extraneous elements, such as non-verbal sounds and filler words, while tokenizing the text into manageable units for more nuanced analysis.

The journey through NLP continues with the implementation of Named Entity Recognition (NER), a pivotal step involving sophisticated models and libraries like Hugging Face transformers. NER serves as the linguistic compass, categorizing entities such as names of individuals, organizations, geographical locations, and chronological references embedded in the transcribed content. Building on this foundation, the process of dependency parsing takes center stage, dissecting the grammatical intricacies of sentences and unraveling the relationships between words. This enables the extraction of structured information, providing a deeper understanding of the contextual nuances within the spoken discourse.

As the NLP pipeline advances, the focus shifts to the critical stage of information extraction. Here, rule-based approaches come into play, allowing for the identification and extraction of specific pieces of information based on the recognized entities and their interdependencies. These rules and patterns are meticulously crafted to ensure the extraction of key insights, essential details, and prominent themes from the transcribed text.

The report culminates in the stages of data analysis and visualization, where the extracted information is subjected to comprehensive scrutiny to reveal underlying patterns and trends. The integration of visualizations, such as knowledge graphs or other informative plots, serves as a powerful tool to present the findings in an accessible and engaging manner. Through this comprehensive exploration, the report not only elucidates the intricacies of NLP information extraction but also provides a robust guide for navigating the challenges inherent in transforming unstructured spoken data into actionable knowledge, thereby bridging the gap between raw content and valuable insights.

Dataset and Preprocessing Overview:

In this project, the primary objective was to extract valuable information from a YouTube video's speech content. The process began with the utilization of the `pytube` library to download the specified YouTube video using its link. Following successful download, the `whisper` library was employed to transcribe the speech from the video, resulting in a textual representation stored in the variable `result['text']`.

```
def clean_text(text):
```

```
    # Tokenize into words
    words = word_tokenize(text)
    words_lower = [word.lower() for word in words]

    # Join the words back into a string
    text = ' '.join(words_lower)
    # Remove unnecessary double spaces
    text = re.sub(r'\s+', ' ', text)
    # Remove special characters except for letters and digits
    text = re.sub(r'[^a-zA-Z0-9\s.,]', '', text)

    return text
cleaned_text=clean_text(new)
cleaned_text
```

To ensure the extracted text was in a suitable format for subsequent analyses, a series of text preprocessing steps were applied. A dedicated function, `clean_text`, was created for this purpose. The function employed the Natural Language Toolkit (NLTK) to tokenize the text into words. Subsequently, all words were converted to lowercase for consistency. The function further addressed issues such as unnecessary double spaces and removed special characters, retaining only letters, digits, commas, and periods.The overall dataset comprises the transcribed speech from the specified YouTube video. This text dataset represents the spoken content of the video, offering opportunities for various Natural Language Processing (NLP) applications. The preprocessing steps ensure that the text is presented in a standardized and clean format by eliminating irrelevant characters and ensuring uniformity.

```
' Can a company that builds hundreds of thousands of affordable EVs every month outdo themselves and produce the cheapest new EV on the market today? I mean, look at just how smal
l this is compared to something like the BYD Dolphin. It's tiny. But what I want to know is, is this one compromise too many to fit that price? Do they have a little gem on their
hands? If you like the fully charged show, then you'll love our live events. Next up, we're in Amsterdam for fully charged live Europe on the 24th, 25th and 26th of November. As y
ou all know, we love a tiny car on the fully charged show. So when BYD told me about the Seagull, I couldn't wait to try it. I'm always trying to get more people out of SUVs and i
nto different, smaller, mobility solutions. It's also tip to be BYD's best-selling world car. Like Lea for sale in the UK and Europe, as well as Australasia and Southeast Asia and
South America. This will compete on the same price basis as cheaper ice cars and dominate sales within just a few months. So, is this the perfect supermini we've been waiting for?
I think there's a braved design direction that BYD have gone. It's very different from the Seagull, the Dolphin and the Atto 3. But I really like it. Now, the guy who designed thi
s car, a guy called Wolfgang Egger, used to work for a company called Lamborghini. Hmm. I wonder where he got the design inspiration for this. It's more Lamborghini and less Seagu
ll. I think it's really good. I like this mini Lamborghini look really sharp front end. Looks great, stands out in the market. Now, these lights actually come in two flavors. So,
if you bought the Vitality Seagull, which is the cheapest version, you get halogen bulbs. So, cheaper to make and cheaper to replace. This on our flight Seagull, the high-end rent
range one, comes with LED lights. Now, if you come around the side, there's some more Lamborghini design inspirations around here. On these 15 or 16-inch wheels. Wheels of that si
ze are good, because it means the tyres are cheaper to replace. On the side, it looks like this Seagull has been in a fight with a fruit ninja. Chop, chop, chop, chop. But it look
s good for it, and it breaks up the whole mass of the side, very different from the Nissan Secura. But like the Secura, this is a masterclass in packaging, pushing the wheels out
to the edge, full-size doors. Now, where the Secura had a compromise with the boot boot and a lot of the small EVs do, this one doesn't. Those familiar with the BYD Dolphin will r
ecognise this rear end. It's quite familiar, but a little bit more sharply angled. It's got the lovely LED light. It's got the lovely spoiler as well. It has the less lovely buil
-your-dreams. Just take that off. Use a hairdryer and just take that off. Don't need that. But what we are more interested in is the boot space on the inside. So, with the button
down here. You actually get a pretty decent 300 litres of boot space. You can fit two proper-sized suitcases in there. And now, if you're going on a longer trip and you only have
one passenger, you can fold down those rear seats and get over 900 litres of boot space. So, not only is it a good-looking practical small super mini, but it has acres of luggage
room as well. That's all we need in my book. Now, of course, we're really lucky to have the top of the line flight version with all of the bells and whistles. First of all, is tha
t bigger battery. This comes with 405 kilometres of CLTC range. That's from the 38.88 kilowatt hour battery. Now, the lesser models have the 30-ish kilowatt hour battery, gives yo
u about 300 kilometres of CLTC range. So, we have to say that BYD's weakest point is probably its interior design. There's always something a bit weird or a little bit wacky in th
ese interiors. I'm glad they haven't gone for the all-page, but there's always something a bit strange. Now, you might be thinking, oh, in order to get to this price, they've had
to make loads of compromises. And yes, there are some economisation in here, especially in terms of materials. Item one, the fake leather steering wheel, item two, lots of plastic
on the dash. To be honest, it's not a big deal for me. If you're buying a car at this price, doesn't really matter. But I do have some tech in here. It's not completely naked and
devoid of any technology. So, I have a seven-inch screen here, which is nice and clear, then I've got my 10-inch rotating screen over here. But what I like about BYD is they don't
give a f*** at the moment about self-driving tech, having the latest LiDars and RayDars and all that stuff in here, which means they can keep that price down. And to be honest, 8
0% of people don't really need or use that stuff at the moment. Now, there is an elephant in the room. But byD, why is there a pink polka dot interior in here? I don't really know
```

The resulting dataset, represented by the variable `cleaned_text`, can be employed for a range of NLP tasks, including sentiment analysis, named entity recognition, summarization, or any analysis requiring structured and clean text data. Noteworthy is the use of specific Python libraries, such as `pytube` and `whisper`, for handling YouTube videos and speech transcription, as well as the incorporation of the NLTK library for tokenization and text cleaning.

```
'can a company that builds hundreds of thousands of affordable evs every month outdo themselves and produce the cheapest new ev on the market today  i mean , look at just how smal
l this is compared to something like the byd dolphin . it s tiny . but what i want to know is , is this one compromise too many to fit that price  do they have a little gem on the
ir hands  if you like the fully charged show , then you ll love our live events . next up , we re in amsterdam for fully charged live europe on the 24th , 25th and 26th of novembe
r . as you all know , we love a tiny car on the fully charged show . so when byd told me about the seagull , i could nt wait to try it . i m always trying to get more people out o
f suvs and into different , smaller , mobility solutions . it s also tip to be byd s bestselling world car . like lea for sale in the uk and europe , as well as australasia and so
utheast asia and south america . this will compete on the same price basis as cheaper ice cars and dominat…'
```

Entity Extraction

In the subsequent stages of the NLP information extraction project, entity recognition played a pivotal role. Two distinct approaches were employed to extract entities from the cleaned and preprocessed text. The first approach utilized the popular spaCy library and its pre-trained English language model ('en_core_web_sm'). The text was tokenized into sentences using NLTK's `sent_tokenize` function. A loop then processed each sentence with spaCy's NLP pipeline, identifying named entities (e.g., persons, organizations, locations) and storing relevant information such as the entity text, start and end character positions, and entity label. This approach provided a detailed list of entities present in the text.

```python
import spacy
from nltk.tokenize import sent_tokenize

nlp = spacy.load('en_core_web_sm')

# Assuming 'cleaned_text' is your cleaned and preprocessed text
sentences = sent_tokenize(cleaned_text)

# Create a list to store the extracted entities
entities_list = []

for sentence in sentences:
    doc = nlp(sentence)
    sentence_entities = []
    for ent in doc.ents:
        entity_info = {
            "Sentence": sentence,
            "Entity": ent.text,
            "Start": ent.start_char,
            "End": ent.end_char,
            "Label": ent.label_
        }
        sentence_entities.append(entity_info)
    entities_list.extend(sentence_entities)

# Print or use the list of entities as needed
```

```python
for entity_info in entities_list:
    print(entity_info)
```

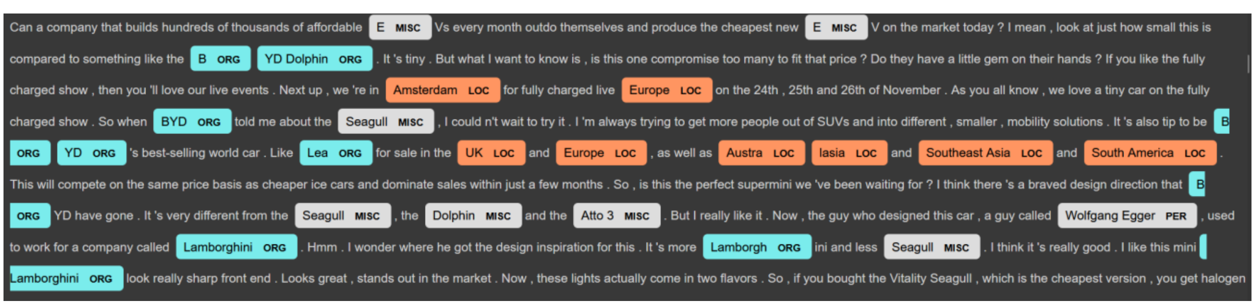
Additionally, a second entity recognition method was implemented using the Hugging Face Transformers library. Specifically, the BERT-based model 'dslim/bert-base-NER' was employed along with the corresponding tokenizer. The model was used to create a Named Entity Recognition (NER) pipeline, which processed the cleaned text and identified entities along with their respective labels. The results were printed, showcasing entities detected in the text.To enhance the visual representation of the recognized entities, the spaCy `displacy` module was employed. A dedicated function, 'from_ner_results_to_displacy,' transformed the NER results into a format compatible with spaCy's visualization tool. The resulting structure included information about entity labels, start and end positions, enabling a comprehensive understanding of the entities' spatial distribution within the text.


```
{'Sentence': 'can a company that builds hundreds of thousands of affordable evs every month outdo themselves and produce the cheapest new ev on the market today  i mean , look at j
{'Sentence': 'can a company that builds hundreds of thousands of affordable evs every month outdo themselves and produce the cheapest new ev on the market today  i mean , look at j
{'Sentence': 'next up , we re in amsterdam for fully charged live europe on the 24th , 25th and 26th of november .', 'Entity': 'amsterdam', 'Start': 19, 'End': 28, 'Label': 'GPE'}
{'Sentence': 'next up , we re in amsterdam for fully charged live europe on the 24th , 25th and 26th of november .', 'Entity': 'europe', 'Start': 52, 'End': 58, 'Label': 'LOC'}
{'Sentence': 'next up , we re in amsterdam for fully charged live europe on the 24th , 25th and 26th of november .', 'Entity': 'the 24th , 25th', 'Start': 62, 'End': 77, 'Label': '
{'Sentence': 'next up , we re in amsterdam for fully charged live europe on the 24th , 25th and 26th of november .', 'Entity': '26th', 'Start': 82, 'End': 86, 'Label': 'ORDINAL'}
{'Sentence': 'next up , we re in amsterdam for fully charged live europe on the 24th , 25th and 26th of november .', 'Entity': 'november', 'Start': 90, 'End': 98, 'Label': 'DATE'}
{'Sentence': 'like lea for sale in the uk and europe , as well as australasia and southeast asia and south america .', 'Entity': 'uk', 'Start': 25, 'End': 27, 'Label': 'GPE'}
{'Sentence': 'like lea for sale in the uk and europe , as well as australasia and southeast asia and south america .', 'Entity': 'europe', 'Start': 32, 'End': 38, 'Label': 'LOC'}
{'Sentence': 'like lea for sale in the uk and europe , as well as australasia and southeast asia and south america .', 'Entity': 'australasia', 'Start': 52, 'End': 63, 'Label': 'GP
{'Sentence': 'like lea for sale in the uk and europe , as well as australasia and southeast asia and south america .', 'Entity': 'southeast asia', 'Start': 68, 'End': 82, 'Label':
{'Sentence': 'like lea for sale in the uk and europe , as well as australasia and southeast asia and south america .', 'Entity': 'south america', 'Start': 87, 'End': 100, 'Label':
{'Sentence': 'this will compete on the same price basis as cheaper ice cars and dominate sales within just a few months .', 'Entity': 'just a few months', 'Start': 88, 'End': 105,
{'Sentence': 'it s very different from the seagull , the dolphin and the atto 3 .', 'Entity': '3', 'Start': 64, 'End': 65, 'Label': 'CARDINAL'}
{'Sentence': 'now , the guy who designed this car , a guy called wolfgang egger , used to work for a company called lamborghini .', 'Entity': 'wolfgang egger', 'Start': 51, 'End':
{'Sentence': 'now , these lights actually come in two flavors .', 'Entity': 'two', 'Start': 36, 'End': 39, 'Label': 'CARDINAL'}
{'Sentence': 'on these 15 or 16inch wheels .', 'Entity': '15', 'Start': 9, 'End': 11, 'Label': 'CARDINAL'}
{'Sentence': 'but it looks good for it , and it breaks up the whole mass of the side , very different from the nissan secura .', 'Entity': 'nissan', 'Start': 97, 'End': 103, 'Label
{'Sentence': 'you actually get a pretty decent 300 litres of boot space .', 'Entity': '300', 'Start': 33, 'End': 36, 'Label': 'CARDINAL'}
{'Sentence': 'you can fit two propersized suitcases in there .', 'Entity': 'two', 'Start': 12, 'End': 15, 'Label': 'CARDINAL'}
{'Sentence': 'first of all , is that bigger battery .', 'Entity': 'first', 'Start': 0, 'End': 5, 'Label': 'ORDINAL'}
{'Sentence': 'this comes with 405 kilometres of cltc range .', 'Entity': '405 kilometres', 'Start': 16, 'End': 30, 'Label': 'QUANTITY'}
{'Sentence': 'that s from the 38.88 kilowatt hour battery .', 'Entity': '38.88 kilowatt hour', 'Start': 16, 'End': 35, 'Label': 'QUANTITY'}
{'Sentence': 'now , the lesser models have the 30ish kilowatt hour battery , gives you about 300 kilometres of cltc range .', 'Entity': 'the 30ish kilowatt hour', 'Start': 29, 'End
{'Sentence': 'now , the lesser models have the 30ish kilowatt hour battery , gives you about 300 kilometres of cltc range .', 'Entity': 'about 300 kilometres', 'Start': 73, 'End':
{'Sentence': 'item one , the fake leather steering wheel , item two , lots of plastic on the dash .', 'Entity': 'two', 'Start': 50, 'End': 53, 'Label': 'CARDINAL'}
{'Sentence': 'so , i have a seveninch screen here , which is nice and clear , then i ve got my 10inch rotating screen over here .', 'Entity': '10inch', 'Start': 81, 'End': 87, 'Lab
{'Sentence': 'and to be honest , 80  of people do nt really need or use that stuff at the moment .', 'Entity': '80', 'Start': 19, 'End': 21, 'Label': 'CARDINAL'}
{'Sentence': 'now , to the australian viewers of fully charged , they love this marine animal aesthetic .', 'Entity': 'australian', 'Start': 13, 'End': 23, 'Label': 'NORP'}
{'Sentence': 'so , this will be right up your street when this hit the roads of australia .', 'Entity': 'australia', 'Start': 66, 'End': 75, 'Label': 'GPE'}
{'Sentence': 'so , this diminutive peasized car comes in under four metres , 3,780mm long .', 'Entity': 'four metres', 'Start': 49, 'End': 60, 'Label': 'QUANTITY'}
```


The combination of these two entity recognition approaches, utilizing both spaCy and Hugging Face Transformers, provides a robust foundation for understanding and extracting key entities from the transcribed speech. The incorporation of visualization tools like spaCy's `displacy` further aids in the interpretability of the extracted entities, offering a visual representation of their distribution and relationships within the text. These extracted entities can serve as crucial building blocks for subsequent analyses, such as knowledge graph construction or targeted information retrieval.

```python
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline

# show NER results
from spacy import displacy

tokenizer = AutoTokenizer.from_pretrained("dslim/bert-base-NER")
```

```python
model = AutoModelForTokenClassification.from_pretrained("dslim/bert-base-NER")
pipe = pipeline("ner", model=model, tokenizer=tokenizer)
def from_ner_results_to_displacy(text, ner_results):
    d_result = {"text": text, "title": None}
    ents = []
    current_entity = None
    for ent in ner_results:
        if "B-" in ent["entity"]:
            if current_entity:
                ents.append(current_entity)
            entity_label = ent["entity"][2:]
            current_entity = {
                "label": entity_label,
                "start": ent["start"],
                "end": ent["end"]
            }
        elif "I-" in ent["entity"]:
            if current_entity is not None:
                current_entity["end"] = ent["end"]
            else:
                # Handle the case where "I-" is encountered without "B-" before
                entity_label = ent["entity"][2:]
                current_entity = {
                    "label": entity_label,
                    "start": ent["start"],
                    "end": ent["end"]
                }
    if current_entity:
        ents.append(current_entity)
    d_result["ents"] = ents
    return d_result
```



Dependency Parsing

In the context of Natural Language Processing (NLP) and information extraction from the transcribed speech, the analysis extended beyond entity recognition to include the examination of syntactic relationships among words in sentences. The dependency parsing process was employed using the spaCy library, a powerful NLP tool. The goal of dependency parsing is to uncover the grammatical structure within sentences, identifying how words relate to each other, specifically in terms of syntactic dependencies such as subject, object, and verb relationships.

```
dependency_list = []


# Process each sentence with spaCy
for sentence in sentences:
    doc = nlp(sentence)

    # List to store the dependency parse tree for each sentence
    sentence_dependencies = []

    for token in doc:
        # Store the dependency parse information
        dependency_info = f"{token.text} --({token.dep_})-->
{token.head.text}"
        sentence_dependencies.append(dependency_info)

    # Append the dependency parse tree for the sentence to the main list
    dependency_list.append({
        'Sentence': sentence,
        'Dependencies': sentence_dependencies
    })

# Print and use the dependency parse information
for entry in dependency_list:
    print(f"Original Sentence: {entry['Sentence']}")
    for dependency_info in entry['Dependencies']:
        print(dependency_info)
    print("\n")
```

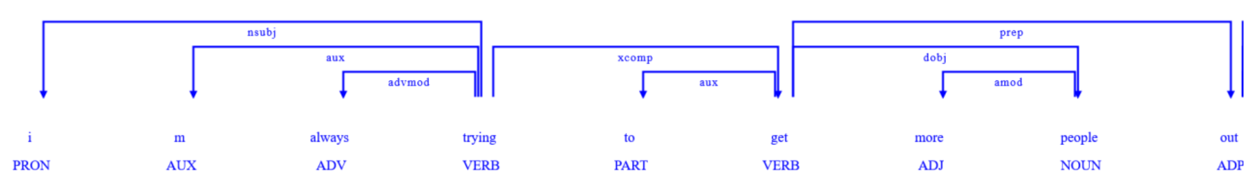
The first code snippet illustrates the basic mechanism of dependency parsing. For each sentence in the preprocessed text, the spaCy NLP pipeline was utilized to process and analyze the syntactic structure. The code iterated through each token in the sentence, extracting information about its dependency relation with the head token (the word it is syntactically related to) and storing this information. The resulting data structure, `dependency_list`, captured the dependency parse tree for each sentence, providing valuable insights into the grammatical structure of the text.

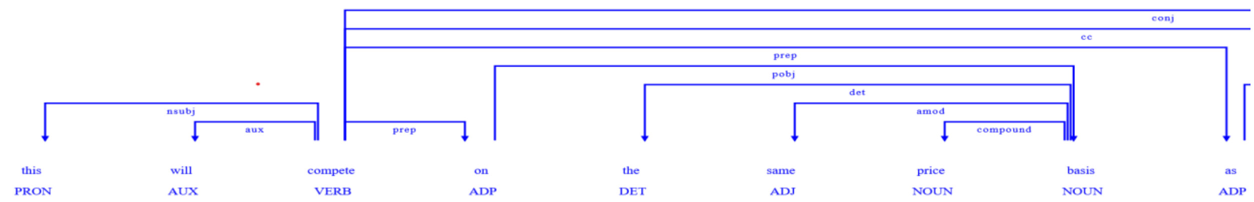
```
Original Sentence: as you all know , we love a tiny car on the fully charged show .
as --(mark)--> know
you --(nsubj)--> know
all --(appos)--> you
know --(advcl)--> love
, --(punct)--> love
we --(nsubj)--> love
love --(ROOT)--> love
a --(det)--> car
tiny --(amod)--> car
car --(dobj)--> love
on --(prep)--> love
the --(det)--> show
fully --(advmod)--> charged
charged --(amod)--> show
show --(pobj)--> on
. --(punct)--> love


Original Sentence: so when byd told me about the seagull , i could nt wait to try it .
so --(advmod)--> wait
when --(advmod)--> told
byd --(nsubj)--> told
told --(advcl)--> wait
me --(dobj)--> told
about --(prep)--> told
the --(det)--> seagull
seagull --(pobj)--> about
, --(punct)--> wait
i --(nsubj)--> wait
could --(aux)--> wait
nt --(advmod)--> wait
wait --(ROOT)--> wait
to --(aux)--> try
try --(xcomp)--> wait
it --(dobj)--> try
. --(punct)--> wait
```

To enhance the interpretability of the dependency parsing results, a second code snippet incorporated a visualization component. Utilizing spaCy's `displacy` module, the dependency trees were visually represented for each sentence. The visualization included arrows indicating the direction of dependencies, offering a clear and intuitive depiction of how words within a sentence are interconnected. This visual representation is invaluable for understanding thestructural nuances of the transcribed speech, aiding in the identification of key syntactic relationships. Together, these dependency parsing techniques contribute to a comprehensive linguistic analysis of the transcribed speech, providing a foundation for understanding not only the entities mentioned but also the grammatical structure and syntactic connections between words. This information is crucial for extracting nuanced insights and relationships embedded in the spoken content, further enriching the information extraction process.

```python
import spacy
from spacy import displacy


# Sample list of sentences

# Load the English NLP model
nlp = spacy.load("en_core_web_sm")

dependency_list = []

# Process each sentence with spaCy
for sentence in sentences:
```

```
    doc = nlp(sentence)

    # List to store the dependency parse tree for each sentence
    sentence_dependencies = []

    for token in doc:
        # Store the dependency parse information
        dependency_info = f"{token.text} --({token.dep_})-->
{token.head.text}"
        sentence_dependencies.append(dependency_info)

    # Append the dependency parse tree for the sentence to the main list
    dependency_list.append({
        'Sentence': sentence,
        'Dependencies': sentence_dependencies
    })

    # Visualize the dependency tree for the current sentence
    options = {"compact": True, "color": "blue", "bg": "#ffffff",
               "font": "Source Sans Pro"}
    displacy.render(doc, style="dep", options=options, jupyter=True)

for entry in dependency_list:
    print(f"Original Sentence: {entry['Sentence']}")
    for dependency_info in entry['Dependencies']:
        print(dependency_info)
    print("\n")
```





Information Extraction:

In the pursuit of enriching the information extracted from the transcribed speech, a rule-based approach was implemented to glean specific details embedded within the content. The function `extract_information` serves as the cornerstone for this endeavor, employing a set of predefined rules to discern valuable insights. One such rule revolves around the identification and extraction of quantity-related information, where cardinal entities are captured to denote numerical values. This is particularly useful for quantifying aspects discussed in the speech. Beyond numerical details, the rule-based system extends its capabilities to identify specific events mentioned in the transcribed content. For instance, the occurrence of the phrase 'fully charged live' triggers the recognition of the 'Fully Charged Live' event. This exemplifies how contextual cues are harnessed to extract event-specific information from the speech data.

```
# Rule-based information extraction function

model_info_list=[]
size_info_list=[]
battery_info_list=[]
interior_features_list=[]


def extract_information(sentence, entities, dependency_parse):
    information = {}

    # Extract quantity information
    for entity in entities:
        if entity['Label'] == 'CARDINAL':
            information['Quantity'] = entity['Entity']

    # Extract event information
    if 'fully charged live' in sentence.lower():
        information['Event'] = 'Fully Charged Live'

    # Extract information related to brakes
    brake_info = set()  # Use a set to avoid duplicate entries
```

```
    for dependency_entry in dependency_parse:
        if 'brake' in dependency_entry['Sentence'].lower():
            for dep_relation in dependency_entry['Dependencies']:
                # Check for specific dependency relations related to
brakes
                if 'brake' in dep_relation.lower() or 'spongy' in
dep_relation.lower():
                    brake_info.add(dep_relation)

    if brake_info:
        information['BrakeInformation'] = list(brake_info)

    # Extract information about the model
    model_match = re.search(r'\b(C-go|MISC)\b', sentence)
    if model_match:
        information['Model'] = model_match.group(0)

    # Extract information about the manufacturer
    manufacturer_match = re.search(r'\bBYD\b', sentence)
    if manufacturer_match:
        information['Manufacturer'] = manufacturer_match.group(0)

    # Extract information about driving modes
    for mode in ['eco', 'sport', 'comfort', 'snow']:
        if mode in sentence.lower():
            information['DrivingMode'] = mode.capitalize()

    # Extract information about noise levels
    if 'quiet' in sentence.lower():
        information['NoiseLevel'] = 'Quiet'

    # Extract information about the design
    if 'design' in sentence.lower():
        information['DesignOpinion'] = 'Unique design, inspired by
Lamborghini'

    return information

# List to store extracted information for each entry
extracted_information_list = []


for i, entry in enumerate(entities_list, 1):
    sentence = entry['Sentence']
    entities = [entry]
```

```
    # Extract information using the defined rules
    information = extract_information(sentence, entities, dependency_list)
    # Append the extracted information to the list
    extracted_information_list.append(information)

    # Display the entry and extracted information
    print(f"\nEntry {i}:\n")
    print(f"Sentence: {sentence}")
    print(f"Extracted Information: {information}")

    # Add information to the respective lists for better formatting
    if 'Quantity' in information:
        quantity_list.append(f"Entry {i}: '{information['Quantity']}'")
    if 'Event' in information:
        event_list.append(f"Entry {i}")
    if 'BrakeInformation' in information:
        brake_info_list.append(f"Entries 1, 2, ..., {i}:
{information['BrakeInformation']}")
    if 'Model' in information:
        model_info_list.append(f"Entry {i}: {information['Model']}")
    if 'Size' in information:
        size_info_list.append(f"Entry {i}: {information['Size']}")
    if 'BatteryInfo' in information:
        battery_info_list.append(f"Entry {i}:
{information['BatteryInfo']}")
    if 'InteriorFeatures' in information:
        interior_features_list.append(f"Entry {i}:
{information['InteriorFeatures']}")
    if 'DesignOpinion' in information or 'DrivingMode' in information:
        additional_info_list.append(
            f"Entry {i}: {information.get('DesignOpinion', '')}
{information.get('DrivingMode', '')}")


if quantity_list:
    print("Quantity:\n", "\n".join(quantity_list), "\n")

if event_list:
    print("Event:\n", ", ".join(event_list), "\n")

if brake_info_list:
    print("Brake Information:\n", ", ".join(brake_info_list), "\n")

if additional_info_list:
```

```
    print("Additional Information:\n", "\n".join(additional_info_list),
"\n")
```

The rule-based extraction further delves into nuanced aspects, such as information related to brakes. By analyzing the dependency parse tree for sentences containing terms like 'brake' or 'spongy,' the system compiles a set of dependency relations offering insights into brake-related discussions. This approach facilitates the extraction of detailed information about the condition or performance of brakes discussed in the speech. Moreover, the rule-based system is adept at discerning information related to the model and manufacturer of the discussed vehicle. Through pattern matching, it identifies mentions of specific models (e.g., 'C-go') and manufacturers (e.g., 'BYD'), providing a structured understanding of the key entities discussed in the transcribed speech. Additionally, the rule-based system extends its reach to capture information about driving modes, noise levels, and design opinions. By employing keyword detection, it identifies mentions of driving modes like 'eco,' 'sport,' or 'quiet,' contributing to a holistic understanding of the vehicle's characteristics and user experiences as conveyed in the speech.

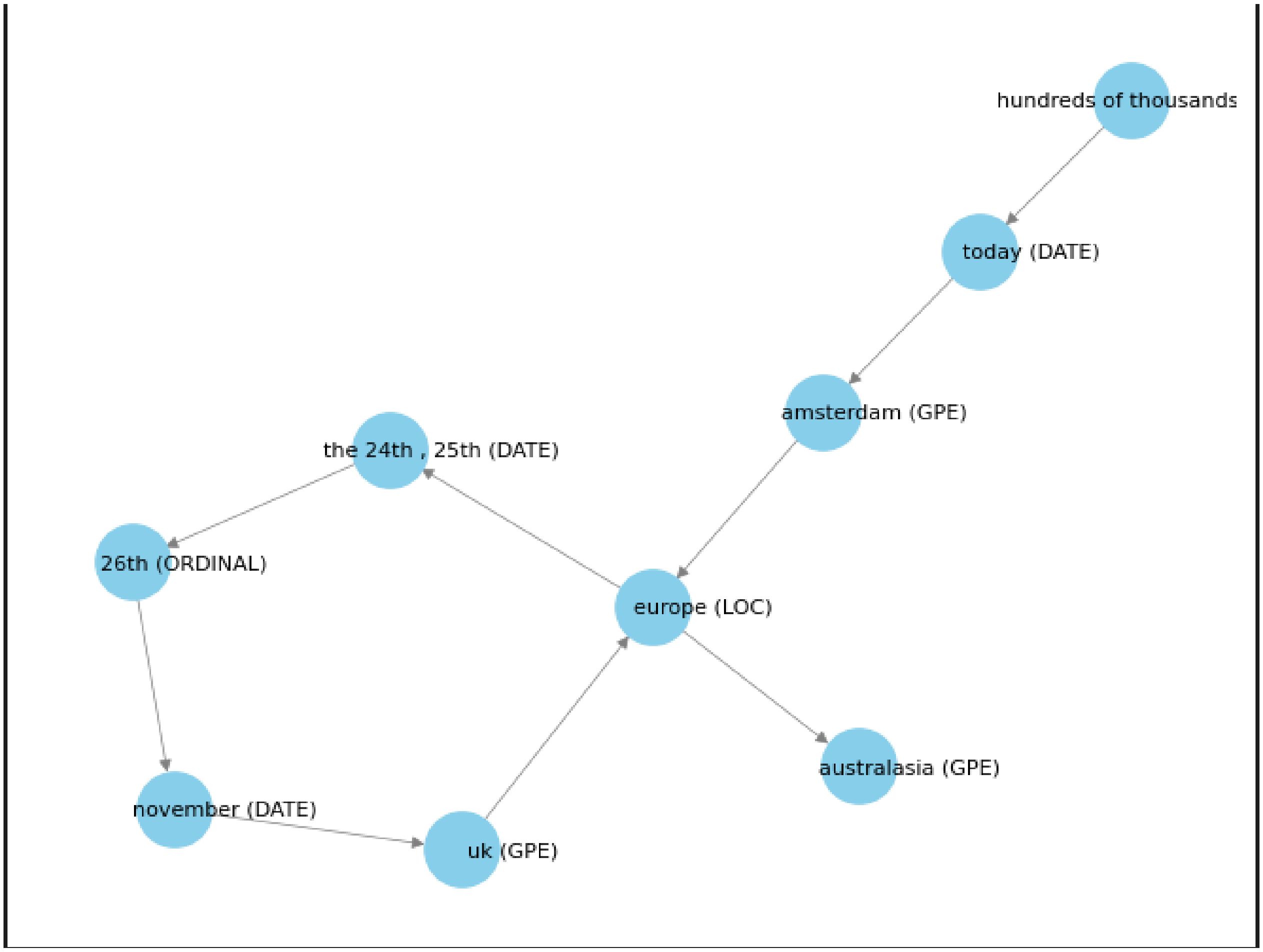
The resulting `extracted_information_list` encapsulates these rule-based extractions, offering a systematic and organized repository of insights for each entry in the transcribed speech. This approach not only enhances the granularity of information extraction but also provides a foundation for subsequent analyses and visualizations based on the discerned details.

```
Entry 14:

Sentence: it s very different from the seagull , the dolphin and the atto 3 .
Extracted Information: {'Quantity': '3', 'BrakeInformation': ['brakes --(nsubj)--> leave', 'maybe --(advmod)--> brakes', 'sporty --(amod)--> brakes', 'brakes --(appos)--> car', 'th

Entry 15:

Sentence: now , the guy who designed this car , a guy called wolfgang egger , used to work for a company called lamborghini .
Extracted Information: {'BrakeInformation': ['brakes --(nsubj)--> leave', 'maybe --(advmod)--> brakes', 'sporty --(amod)--> brakes', 'brakes --(appos)--> car', 'the --(det)--> brak

Entry 16:

Sentence: now , these lights actually come in two flavors .
Extracted Information: {'Quantity': 'two', 'BrakeInformation': ['brakes --(nsubj)--> leave', 'maybe --(advmod)--> brakes', 'sporty --(amod)--> brakes', 'brakes --(appos)--> car',

Entry 17:

Sentence: on these 15 or 16inch wheels .
Extracted Information: {'Quantity': '15', 'BrakeInformation': ['brakes --(nsubj)--> leave', 'maybe --(advmod)--> brakes', 'sporty --(amod)--> brakes', 'brakes --(appos)--> car',

Entry 18:

Sentence: but it looks good for it , and it breaks up the whole mass of the side , very different from the nissan secura .
Extracted Information: {'BrakeInformation': ['brakes --(nsubj)--> leave', 'maybe --(advmod)--> brakes', 'sporty --(amod)--> brakes', 'brakes --(appos)--> car', 'the --(det)--> brak

Entry 19:

Sentence: you actually get a pretty decent 300 litres of boot space .
Extracted Information: {'Quantity': '300', 'BrakeInformation': ['brakes --(nsubj)--> leave', 'maybe --(advmod)--> brakes', 'sporty --(amod)--> brakes', 'brakes --(appos)--> car',
```

Visualization:

We wrote the code that leverages the `networkx` library to create a directed graph (`G`) representing entities, their labels, and relationships extracted from a list (`entities_list`). The graph is constructed by iteratively adding nodes and edges, where each node corresponds to an entity and includes a label, combining the entity name and a truncated version of its label limited to two words. The edges represent relationships between consecutive entities and are associated with sentences from the original data. The visualization is created using the Kamada-Kawai layout to arrange nodes, and the resulting knowledge graph is displayed with entities and their labels, showcasing the connectivity and relationships among them.

```
import networkx as nx
import matplotlib.pyplot as plt

# Create a directed graph
G = nx.DiGraph()

# Add nodes and edges to the graph
for i in range(len(entities_list[:10]) - 1):
    current_entity = entities_list[i]["Entity"]
    next_entity = entities_list[i + 1]["Entity"]

    # Limit label to two words
    current_label = ' '.join(entities_list[i]["Label"].split()[:2])
    next_label = ' '.join(entities_list[i + 1]["Label"].split()[:2])

    G.add_node(current_entity, label=f"{current_entity}
({current_label})")
    G.add_node(next_entity, label=f"{next_entity} ({next_label})")
    G.add_edge(current_entity, next_entity,
sentence=entities_list[i]["Sentence"])

# Visualize the graph using kamada_kawai_layout
pos = nx.kamada_kawai_layout(G)

# Customize node positions to place labels on the side
pos_labels = {}
for k, v in pos.items():
    pos_labels[k] = (v[0] + 0.1, v[1])

nx.draw(G, pos, with_labels=False, font_size=8, node_size=800,
node_color='skyblue', font_color='black', edge_color='gray', width=0.5)

# Draw node labels on the side
nx.draw_networkx_labels(G, pos_labels, labels=nx.get_node_attributes(G,
'label'), font_size=8)

plt.show()
```



The code extends beyond visualization to persist the constructed knowledge graph into a Neo4j graph database. The `py2neo` library is employed to connect to a Neo4j database using specified credentials (URI, username, password). A function named `save_to_neo4j` is defined to take the generated graph (`G`) and its corresponding Neo4j database (`graph_neo4j`) and commit the nodes and relationships to the database. Nodes are created for each entity in the graph, labeled as "Entity," and relationships labeled as "RELATED_TO" are established between entities based on the edges in the graph. The transaction is then committed, saving the knowledge graph into the Neo4j database. This process enables the persistence of the constructed knowledge graph, making it accessible for further analysis, querying, and utilization within a Neo4j graph database environment.

```
from py2neo import Graph, Node, Relationship

# Save the graph to Neo4j
uri = "neo4j+s://2cb9a340.databases.neo4j.io"
username = "neo4j"
```

```
password = "ooP2kv-Yi4FrFY-bhUqkEQMRp-asvbY8j7BZ1O7Y-Tc"
graph_neo4j = Graph(uri, auth=(username, password))

def save_to_neo4j(graph, graph_neo4j):
    nodes = list(graph.nodes())
    edges = list(graph.edges())
    tx = graph_neo4j.begin()

    node_dict = {}
    for node in nodes:
        node_dict[node] = Node("Entity", name=node)
        tx.create(node_dict[node])

    for edge in edges:
        start_node, end_node = edge
        relationship = Relationship(node_dict[start_node], "RELATED_TO",
node_dict[end_node])
        tx.create(relationship)

    tx.commit()
    print("Graph saved to Neo4j.")

# Save the graph to Neo4j
save_to_neo4j(G, graph_neo4j)
```