Overview

Objective of this notebook is 4-folds -

1. **Assemble / Generate** meeting transcripts data. (obvious needs of preprocessing and cleaning with a little EDA)
2. Using some example outputs - **form heuristics** (based on NLP features of the interested input)
3. **Validate the heuristics** -- how well does the heuristic extract necessary items? (accordingly repeat this step and the previous till desired results comeup)
4. Formalize the operations to a single function for **service intergration**

Evee-Output

Calling out my assumptions of what can be a good response from the NLP service.
From the transcription, I am considering following information to be of interest for a meeting MoM.

Content of interest in input

1. **Directly Address Evee**

   - `Hey Evee, {can you | please  | can you please} add  (task) X to person (Y)`
   - *Just a sample. Same sentence can be said in various ways*
     - `Evee, {can you | please | ... } put (task) X to person (Y)'s list`
     - `Remember to assign (task) X to (person) Y, Evee`
   - Above is easier to implement as we can narrow down to those part of transcripts where Evee is adressed

1. **Actionable or Commitable Item phrases**

   - Dialogues with possible action items
     `I have been doing this but will update you with other alternatives`
   - **Note:-** Action items can be part of a previous reference

```
 05:10 --> 05:12 : I was looking into (task) X and I did action A.
 05:13 --> 05:16 : True (person name) Y, but I feel we should also try other (ac
tion) B. Can (you| person Y) do it?
 05:16 --> 05:17 : Sure
```

   - Once the actions are identified, map with the person's involved
     - Involved person may be already mentioned in the tasks
     - Check for phrases of `I|You` etc.

Note: I will assume there can be false positives of tasks generated although it wasn't decided to go ahead with it..
Overall, more tasks generated is better than less tasks

Also, assumption is if the ownership of a task changes we can add both previous and the new user and let it be modifiable through actionable cards

Service Output

```
## list of todos
results = [{actionEntity  , intent , peopleEntity, MoMString}, ....]
```

where
actionEntity = token(s) describing the tasks

intent = verb - Design/Develop etc.

peopleEntity = names list of persons involved

MoMString = generated from above three to be directly used in adaptive card

Data assemble

In [ ]:

```
## Taking very simple static strings as text for now -
## In hack, we need a vtt file parser to extract text from therein
## For example,

"""
0:0:0.0 --> 0:0:5.290
<v 1ae37f18-c4fb-454a-8514-ad5b764b92f1@72f988bf-86f1-41af-91ab-2d7cd011db47>Yea
h, I think please don't read the transcript because this transcripts for my Engli
sh especially is very bad so.</v>

0:0:9.250 --> 0:0:11.520
<v 1ae37f18-c4fb-454a-8514-ad5b764b92f1@72f988bf-86f1-41af-91ab-2d7cd011db47>Yea
h, I think so. Can I start now?</v>

0:0:14.970 --> 0:0:15.450
<v 1725246c-8250-4d5b-b165-64eb54294550@72f988bf-86f1-41af-91ab-2d7cd011db47>Yea
h.</v>

"""

## My temporary starting samples
texts = [
    "Hey Evee, note that Patrick will do the calendar search stuff",
    "Hey Evee, note that the calendar search stuff will be done by Patrick",
    "Well, Evee, please put the NLP model formation as part of Amartya's work",
# direct-address item
    "Yes, I was trying to make an extensible design, but I will look for other p
ossibilities and come back to you..", # intrinsic action item
    "Can you please do it ??", ## example of action item having previous referenc
es
    "Hey Evee, can you add NLP modelling to Amartya's list"
]
```

Overall NLP workflow

This will consist of -

- Applying heuristics based on **(1) POS tags (2) Dependency tree as shown below**
- Thereafter, we need to check the impact %age of the heuristics and change them accordingly
    - Idea is to have a high recall i.e. more tasks are preferred and false positives are not very harmful

We also need to look for usual relationships sentences within a doc and a tokens within a sentence

In [ ]:
```
import spacy
from spacy import displacy
```

In [ ]:
```
# load english language model
nlp = spacy.load('en_core_web_sm',disable=['textcat'])
```

In [ ]:
```
!pip install visualise-spacy-tree
```

In [ ]:
```
import visualise_spacy_tree
from IPython.display import Image, display
```

Testing out the libs with sample sentences --

In [ ]:
```
for text in texts:
    print(text)
    doc = nlp(text)

#     for token in doc:

#         print(token.text, token.dep_, token.head.text, token.head.pos_,
#               [child for child in token.children])

#     displacy.render(doc, style='dep',jupyter=True)
    png = visualise_spacy_tree.create_png(doc)
    display(Image(png))
    print (80*"=")
```

In [ ]:
```
## A bit of noun-chunking as well

for text in texts:

    print (text)
    doc= nlp(text)

    for i, chunk in enumerate(doc.noun_chunks):
        print(i, chunk.text, chunk.root.text, chunk.root.dep_,
                chunk.root.head.text)

    print(80 * "=")
```

Directly mentioning evee

In [ ]:

```
## Single sentence statments, all mentioning evee
## A task is explicityly mentioned in the statment
## Owner may or may not be explicitly mentioned
## Also it is assumed Evee is mentioned at the beginning
## Note! Currently , no transcript will have these kind of sentences
## IN reality transcripts might not add "," after Evee or evee can be misplaced b
y some other token
## we can also employ some audio transcription to transcribe and give quick updat
e in the meeting

evee_texts = [

    ## Starters
    "Evee, add calendar data source implementation as a next step", # task witho
ut a owner
    "Hey Evee, note NLP model formation as Amartya's task item", # task + owner

    ## Conjunctions
    "Hey Evee, note down that Jessica will organize a meeting tomorrow", # task
+ owner ('that' as a conjunction)
    "Hey Evee, note that Calendar Search CR is to be done by Patrick", # task +
owner ('that' as a conjunction')

    ## Interjections
    "Well Evee, please give implementation of databse to John", # task + owner
('please' as interjection)
    "Hi Evee, please assign to Jane the task of review and approval of the desig
n", # another task + owner  (ditto but owner comes first)

    ## Questions
    "Well Evee, Can you assign the implementation of databse to John?", # task +
owner ('question' starting with can)
    "Hey Evee, Can you put design review to Jake's Todo?", # task + owner

    ## Implicit
    "Hey Evee, Jack will update the designs with new redlines", # task+owner wit
h indirect intent (implicitly evee takes a note)
    "Evee, We will organize Ravi's birthday-party tomorrow",
    "Evee, We will organize a welcome party for David ",
    "Hey Evee, designs will be updated by Jack" # task+owner with indirect inten
t (passive voice)
]
```

In [ ]:

```
def gen_sent_deps(sent):

    png = visualise_spacy_tree.create_png(nlp(sent))
    display(Image(png))
```

In [ ]:

```
for text in evee_texts:
    print (text)
    gen_sent_deps(text)
```

In [ ]:

```
spacy.explain("ADP")
```

In [51]:

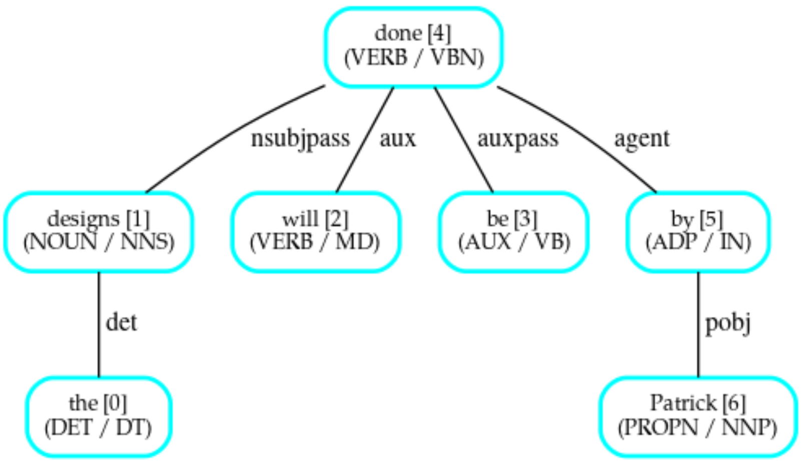
```
text = "the designs will be done by Patrick"
doctest = nlp(text)

rt = next(tok for tok in doctest if tok.dep_ =="ROOT")
nsubjpass = next(r for r in rt.lefts if r.dep_ == 'nsubjpass')
phrase = " ".join([t.text for t in nsubjpass.subtree])
phrase = rt.lemma_ + " " + phrase

print (phrase)
png = visualise_spacy_tree.create_png(doctest)
display(Image(png))

## nsubjpass will give the subject matter
```

```
do the designs
```



In [ ]:
```
text = "Evee note that Calendar Search CR is to be done by Patrick"

doctest = nlp(text)

png = visualise_spacy_tree.create_png(doctest)
display(Image(png))
```

In [ ]:
```
from spacy.matcher import Matcher
matcher = Matcher(nlp.vocab)



doctest = nlp("Hey Evee, Can you put design review to Jake's Todo?")

print(doctest[0:3])

pattern = [{'POS': 'VERB'}, {'POS': 'PROPN'}]
pattern2 = [{'POS': 'VERB'}, {'POS': 'PRON'}]
matcher = Matcher(nlp.vocab)

matcher.add("question", None, pattern, pattern2)
matches = matcher(doctest)
isQuestion = len(matches) > 0

print (isQuestion)
print ([(ent.label_, ent.text) for ent in doctest.ents]) ## By trying out differ
ent input types,
                                                        ##we can see that NER is
not that well defined here
```

In [ ]:
```
import logging

logging.basicConfig(filename='log.txt',level=logging.INFO, format='%(asctime)s
%(message)s')
```

In [50]:

```
## Simple rules from above

## DOBJ says a lot!
## If A root verb has a dobj subtree it probably can be the task (NOT ALWAYS)
## even if I add a interjection like 'please', dobj subtrees can identify and ext
ract the action phrases easily


## DOBJ exceptions -- CONJUNCTIONS
## dobj subtree can be absent if ccomp subtrees are present (here again, it's a p
assive voice)
## ccomp subtrees are present for keywords like 'that', 'if', 'whether' etc. (the
se are SCONJ)
## We can determine these conjunctive items and find their head verbs.. then appl
y the dobj/ nsubj rules

## Further Exceptions -- IMPLICIT INTENT
## If Root verb has nsubj subtree then intent is implicit and the root verb  (+do
bj subtree) likely to be part of action
## if root verb has nsubjpass then the intent might be implicit as well but now w
e need to find the action somewhere within nsubjpass subtree
## Note: nsubjpass is absent when we have explicit intent given in the text, even
when we have it in a passive voice

## OWNER DETECTION
## Owners can be in a dative part of the tree
## When owner's missing, just no other PROPN nouns come up
## May be, tracking PROPNs is a good idea of finding them


from spacy.matcher import Matcher

def get_owners(dialog):

    doc = nlp(dialog)
    pers = [ent.text for ent in doc.ents\
            if ent.label_ == "PERSON" or ent.label_ == "ORG"\
            or ent.label_ == "PRODUCT" or ent.label_ == "GPE"]
    pers = [p for p in pers if "Evee" not in p]

    logging.debug ("Owners = "+ str(pers))
    return pers


def get_actions(dialog):

    logging.debug ("Dialog -" + dialog)
    doc = nlp(dialog)
    rootTok = next(tok for tok in doc if tok.dep_=="ROOT")

    isPassive = any(cTok.dep_ == "nsubjpass" for cTok in rootTok.children)
    hasMultiClause = any(cTok.dep_ == 'ccomp' for cTok in rootTok.children)
    hasDobj = any (cTok.dep_ == 'dobj' for cTok in rootTok.children)
    isSubjPresent = any(cTok.dep_ == 'nsubj' for cTok in rootTok.children)
    phrase = ''

    if hasMultiClause:
        logging.debug ("Multi-claused statement :"+dialog)
        logging.debug (rootTok.text)

        conjTok = [(tok,i) for i, tok in enumerate(doc) if tok.pos_ == "SCONJ"]
        conjTok, i = conjTok[0]
        newdoc = doc[i+1:]
        new_dialog = " ".join(tok.text for tok in newdoc)
        phrase = get_actions(new_dialog)


    elif (isPassive):

        ## Passive voice, Subject is the object here

        nsubjpass = next(r for r in rootTok.lefts if r.dep_ == 'nsubjpass')
        phrase = " ".join([t.text for t in nsubjpass.subtree])
        phrase = rootTok.lemma_ + " " + phrase

    elif (hasDobj):

        dobjTok = next(cTok for cTok in rootTok.children if cTok.dep_ == "dob
j")
        phrase = " ".join(t.text for t in dobjTok.subtree)

        ## If rootverb has subject, prepend it only if it is not referring to Eve
e
        if (isSubjPresent):

            ## Good way to determine is to find whether there is a question to Ev
ee
            ## The nsubj will then be referring to evee only

            ## Is there a question in the sentence ?

            p1 = [{'POS': 'VERB', 'DEP': {'NOT_IN': ['ROOT']}}, {'POS': 'PROPN
'}]
            p2 = [{'POS': 'VERB', 'DEP': {'NOT_IN': ['ROOT']}}, {'POS': 'PRON'}]

            matcher = Matcher(nlp.vocab)
            matcher.add("question", None, p1, p2)
            matches = matcher(doc)
            logging.debug ("Matches -{}".format(matches))

            isQuestion = len(matches) > 0

            if (not isQuestion):
                phrase = rootTok.text + ' ' + phrase
    else:

        ## It might only have a subj
        assert(isSubjPresent)

        nsubj = next(r for r in rootTok.lefts if r.dep_ == 'nsubj')
        phrase = " ".join([t.text for t in nsubj.subtree])


    logging.debug ("Root -"+rootTok.text)
    logging.debug ("Action phrase- "+phrase)


    return phrase

def get_meeting_todos(text):

    print ("Text -"+text)
    action = get_actions(text)
    owners = get_owners(text)

    print ("Action- " + action)
    print (80*"-")
    print ("Owner - " + str(owners))
    print (80*"=")


for text in evee_texts:
    get_meeting_todos(text)
```

```
Text -Evee, add calendar data source implementation as a next step
Action- calendar data source implementation
-----------------------------------------------------------------------------
----
Owner - []
=============================================================================
====
Text -Hey Evee, note NLP model formation as Amartya's task item
Action- NLP model formation
-----------------------------------------------------------------------------
----
Owner - ['NLP', 'Amartya']
=============================================================================
====
Text -Hey Evee, note down that Jessica will organize a meeting tomorrow
Action- organize a meeting tomorrow
-----------------------------------------------------------------------------
----
Owner - ['Jessica']
=============================================================================
====
Text -Hey Evee, note that Calendar Search CR is to be done by Patrick
Action- Calendar Search CR
-----------------------------------------------------------------------------
----
Owner - ['Calendar Search CR', 'Patrick']
=============================================================================
====
Text -Well Evee, please give implementation of databse to John
Action- implementation of databse
-----------------------------------------------------------------------------
----
Owner - ['John']
=============================================================================
====
Text -Hi Evee, please assign to Jane the task of review and approval of the
design
Action- the task of review and approval of the design
-----------------------------------------------------------------------------
----
Owner - ['Jane']
=============================================================================
====
Text -Well Evee, Can you assign the implementation of databse to John?
Action- the implementation of databse
-----------------------------------------------------------------------------
----
Owner - ['John']
```

```
================================================================================
====
Text -Hey Evee, Can you put design review to Jake's Todo?
Action- design review
--------------------------------------------------------------------------------
----
Owner - ['Jake']
================================================================================
====
Text -Hey Evee, Jack will update the designs with new redlines
Action- update the designs with new redlines
--------------------------------------------------------------------------------
----
Owner - ['Jack']
================================================================================
====
Text -Evee, We will organize Ravi's birthday-party tomorrow
Action- organize Ravi 's birthday - party
--------------------------------------------------------------------------------
----
Owner - ['Ravi']
================================================================================
====
Text -Evee, We will organize a welcome party for David
Action- organize a welcome party
--------------------------------------------------------------------------------
----
Owner - ['David']
================================================================================
====
Text -Hey Evee, designs will be updated by Jack
Action- update designs
--------------------------------------------------------------------------------
----
Owner - ['Jack']
================================================================================
====
```