NLP (NATURAL LANGUAGE PROCESSING) :

PAGE 28

NOTE :

In [ ]:

In [1]:
```
# How to work on NLP?
# The essential guide to how NLP work
# Steps to build an NLP Pipeline
# Step 1: Sentence Segmentation. ...
# Step 2: Word Tokenization. ...
# Step 3: Predicting Parts of Speech for Each Token. ...
# Step 4: Text Lemmatization. ...
# Step 5: Identifying Stop Words. ...
# Step 6: Dependency Parsing. ...
# Step 7: Finding Noun Phrases. ...
# Step 8: Named Entity Recognition (NER)
```

1. INSTALLING SPACY LIBRARY : IN ANACONDA POWER SHELL PROMPT :

PAGE 36

In [2]:
```
# We have multiple things like,

# 1) 'conda install spacy'
# 2) 'conda install -c conda-forge spacy-model-en_core_web_sm'
# 3) 'pip install spacy'
# 4) 'python -m spacy download en_core_web_sm'
```

A) Upgraded 'Numpy' to latest Version :

At First - " import spacy " didn't worked in my System. After that find out the 'Error' in installing the 'spacy' in Anacoda powershell prompt, that the version 'Numpy' not supported

So, with this syntax - Means, Upgrading 'Numpy' to the latest Version.

pip install numpy --upgrade

i solved the issue, by upgrading the "Numpy" in my System. Installed 'spacy' Successfully.

2. SPACY :

PAGE 38

In [6]:
```
# Now the First Library - i'm calling the 'spacy'
```

In [4]:
```
import numpy as np
```

```
C:\Users\my pc\anaconda3\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .
libs:
C:\Users\my pc\anaconda3\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd
64.dll
C:\Users\my pc\anaconda3\lib\site-packages\numpy\.libs\libopenblas64__v0.3.23-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [5]:
```
import spacy
```

```
C:\Users\my pc\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0
is required for this version of SciPy (detected version 1.25.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [6]:
```
import spacy
```

NOW THE ENGLISH LIBRARY I AM CALLING :

PAGE 38

In [7]:
```
import en_core_web_sm
```

In [8]:
```
# Now in this particular ENGLISH LIBRARY, i'm keeping one particular NLP(Means, One lefthand Variable called 'nlp')
# and We need load this particular English Library.
# Means, Here the 'Segregation' of an a Text Data we are 100% Handling with an a particular Text Data.
# That's the main reason.
```

In [9]:
```
nlp = en_core_web_sm.load()
```

In [10]:
```
# Now in this Text Data - doc(lefthand variable) = nlp of ('u' of some particular Data "Tesla is looking at buying
# U.S startup for $6 million") This is a particular Data we are Calling.
```

In [11]:
```
doc = nlp (u"Tesla is looking at buying U.S startup for $6 million")
```

In [12]:
```
# Now Whenever we call 'doc' it looks like this :

doc
```

Out[12]:
```
Tesla is looking at buying U.S startup for $6 million
```

In [13]:
```
# Declaration of an a 'String'
```

Default Delimiter for 'String' is..... 'Space' :

page 39

In [14]:
```
# Default Delimiter for a 'String' is 'Space' :
# First i want to 'Split the String' :

# Here Based on Delimiter We are Splitting the Data :
```

In [15]:
```
# Once We Split the thing Which we call it as an a Parts of Speech : Means, "Assigning the parts into Token"
# Means Whatever we have did in below 15th Sum is Called as 'Tokening',
# Means 'Splitting the Data Based on Delimiter'
```

In [16]:
```
# This is called as an a 'TOKENING'

for token in doc:
    print(token.text)
```

```
Tesla
is
looking
at
buying
U.S
startup
for
$
6
million
```

In [50]:
```
# Here for each and every word 'pos' - means parts of speech -> "Noun of ASCII Keywords" is going to be Represented :
# ASCII -> "AMERICAN STANDARD CODE FOR INFORMATION INTER CHANGE" :
# Like Tesla 96, is 87 .... Numerically is going to be Assigned. One particular Number of each Word :
```

In [18]:
```
for token in doc:
    print(token.text, token.pos)
```

```
Tesla 96
is 87
looking 100
at 85
buying 100
U.S 96
startup 92
for 85
$ 99
6 93
million 93
```

In [51]:
```
# Now i want to read in a way like - PRONOUN (OR) NOUN like English Verbs, i want to see, that how it's looks like,
# By adding 'Underscore_' beside the word 'pos_'
```

In [19]:
```
for token in doc:
    print(token.text, token.pos_)
```

```
Tesla PROPN
is AUX
looking VERB
at ADP
buying VERB
U.S PROPN
startup NOUN
for ADP
$ SYM
6 NUM
million NUM
```

In [52]:
```
# Now We can take the 'Dependency' - Which is an a 'Subject', 'Dependency', 'Root', 'Objective', 'Compound' - by taki
# the 'TOKEN' -> Means, OBJECT NAME - 'token.dependency_underscore'
```

In [20]:
```
for token in doc:
    print(token.text, token.pos_, token.dep_)
```

```
Tesla PROPN nsubj
is AUX aux
looking VERB ROOT
at ADP prep
buying VERB pcomp
U.S PROPN compound
startup NOUN dobj
for ADP prep
$ SYM quantmod
6 NUM compound
million NUM pobj
```

Using 'pipeline' :

page 42

In [ ]:
```
# Here, By using an a 'pipeline' of NLP - We are doing 'Token to Vector'.
# Each and Every Word - Whenever we are Converting to an a 'NLP' is Called as an a 'VECTOR OBJECT'
```

In [1]:
```
# 'ner' - means Which We give NTT Recognizer - means, "IDENTIFYING THE OBJECT"
# Up to here, We have done - 'lammatization' and 'ner' We are able to see here.

# By Using an a English Library, We Can Work with this particular stuffing (output of 21st sum)
```

In [21]:
```
nlp.pipeline
```

Out[21]:
```
[('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec at 0x1a400a791c0>),
 ('tagger', <spacy.pipeline.tagger.Tagger at 0x1a400a793a0>),
 ('parser', <spacy.pipeline.dep_parser.DependencyParser at 0x1a400ade660>),
 ('attribute_ruler',
  <spacy.pipeline.attributeruler.AttributeRuler at 0x1a400de9f80>),
 ('lemmatizer', <spacy.lang.en.lemmatizer.EnglishLemmatizer at 0x1a400df8840>),
 ('ner', <spacy.pipeline.ner.EntityRecognizer at 0x1a400ade740>)]
```

In [2]:
```
# Here We Can see - 'tok2vec',tagger,parser....e.t.c, Only names, if we want to get it and dependence we have seen her
```

In [22]:
```
nlp.pipe_names
```

Out[22]:
```
['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```

In [ ]:
```
# Now 'doc' means, the entire data is printing :
```

In [23]:
```
doc
```

Out[23]:
```
Tesla is looking at buying U.S startup for $6 million
```

In [ ]:
```
# 2nd position in 'doc' :
# Tokenization getting the Words :
```

In [24]:
```
doc[2]
```

Out[24]:
```
looking
```

In [25]:
```
doc[2].pos_
```

Out[25]:
```
'VERB'
```

In [26]:
```
# Now something like, i'm taking 'quote' - left hand variable :
# page 43
```

In [27]:
```
# [2:5] - LEAVE FIRST TWO : TAKE FIRST FIVE

quote = doc[2:5]
quote
```

Out[27]:
```
looking at buying
```

In [3]:
```
# Now What's the 'TYPE' :
# page 44
```

In [29]:
```
# This is called 'Tokenization on the String' - Tokenization on the particular String Data :

type(quote)
```

Out[29]:
```
spacy.tokens.span.Span
```

In [4]:
```
# Now i'm taking some lefthand variable 'doc2'
# page 44
```

In [31]:
```
# Gap should not given between - 'u' and " " :

doc2 = nlp(u"This is First Sentence. This is Second Sentence. This is Third Sentence.")
```

In [32]:
```
doc2
```

Out[32]:
```
This is First Sentence. This is Second Sentence. This is Third Sentence.
```

In [33]:
```
# Now if we observe :
# page 44
```

In [34]:
```
for sentence in doc2.sents:
    print(sentence)
```

```
This is First Sentence.
This is Second Sentence.
This is Third Sentence.
```

In [35]:
```
# Tokenization process - The Tokenization is totally depends upon 'SPACE' Working Environment only:
# 1) Splitting Original Data based on 'White Space' and next
# 2) Prefix - It Removes the Data.
# 3) Exception - It is Splitting again.
# 4) Suffix - at the 'end' it is doing.
# 5) In the Suffix - also there is Exception.
# 6) And at the last - Whenever We apply the TOKENIZATION on any String Particular Data - It looks like 'Done' :

# page 46
```

In [36]:
```
mystring = '"We \ "re moving to L.A!""'
```

In [37]:
```
mystring
```

Out[37]:
```
'"We \\ "re moving to L.A!""'
```

In [38]:
```
# Now we need to convert to an a 'NLP' first :
# Without Converting to an a 'NLP', We Can't work it out :
```

In [39]:
```
doc = nlp(mystring)
```

In [40]:
```
for token in doc:
    print(token.text)
```

```
"
We
\
"
re
moving
to
L.A
!
"
"
```

Noun Things : Identify only Nouns in Statement :

page 47

In [41]:
```
doc4 = nlp (u"Autonomous cars shift insurance liability towards manufactures")
```

In [42]:
```
# Here we are Printing the Nouns :

for chunks in doc4.noun_chunks:
    print(chunks)
```

```
Autonomous cars
insurance liability
manufactures
```

Displacy( ) -- Represents data in 'Graph' :

page 47

In [43]:
```
doc
```

Out[43]:
```
"We \ "re moving to L.A!""
```

In [ ]:
```

```

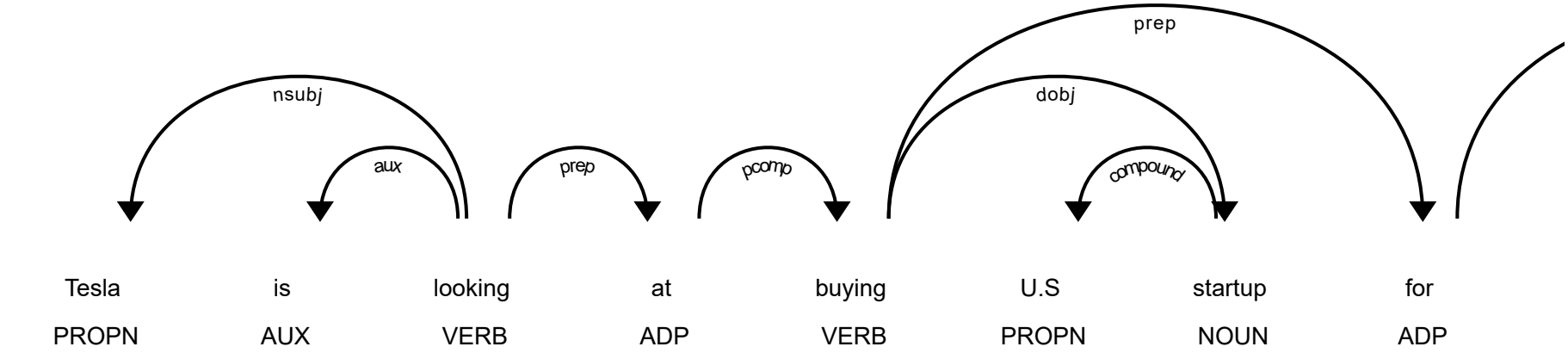
In [44]:
```
doc5 = nlp (u"Tesla is looking at buying U.S startup for $6 million")
```

First Calling 'Displacy' and 'Representing Graph' :

page 48

In [45]:
```
# This is What dependency - 1) Long dependency, 2) Short dependency :

from spacy import displacy
displacy.render(doc5,style = 'dep', jupyter = True, options = {'distance':110})
```



In [ ]:
```

```

In [46]:
```
# Now let's see the Next dependency :
```

In [47]:
```
# Here, 'ent' means entity :

displacy.render(doc5, style = 'ent', jupyter = True)
```

Tesla **ORG** is looking at buying U.S **GPE** startup for $6 million **MONEY**

Now 'Displacy' applying on Server : 127.0.01:5000

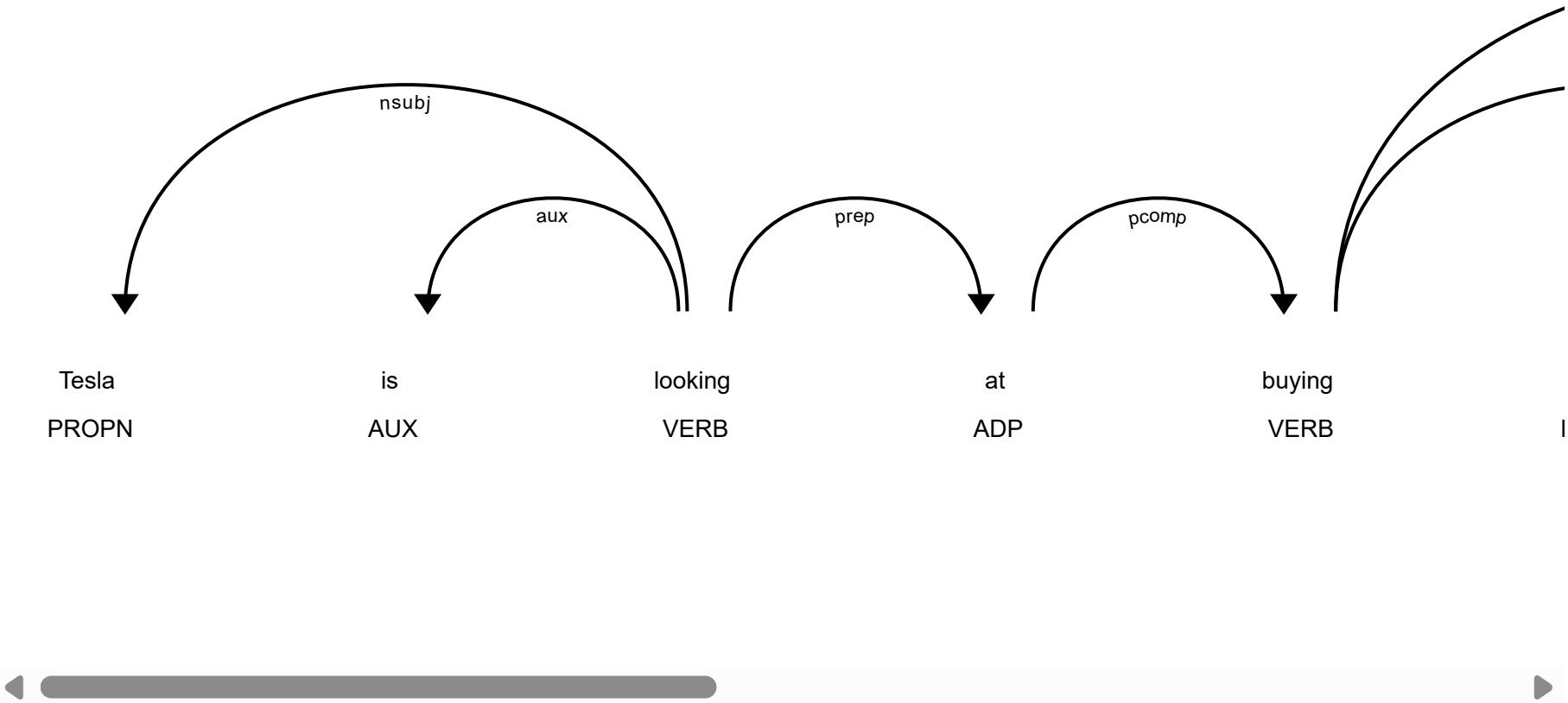
page 48

In [48]:
```
# With an a Serving Port of 5000, also we can see - Diplacy applying on Server
```

In [49]:
```
# displacy.serve on doc5,style....

displacy.serve(doc5, style = 'dep')
```

```
C:\Users\my pc\anaconda3\lib\site-packages\spacy\displacy\__init__.py:108: UserWarning: [W011] It looks like you're
calling displacy.serve from within a Jupyter notebook or a similar environment. This likely means you're already run
ning a local web server, so there's no need to make displaCy start another one. Instead, you should be able to repla
ce displacy.serve with displacy.render to show the visualization.
  warnings.warn(Warnings.W011)
```



```
Using the 'dep' visualizer
Serving on http://0.0.0.0:5000 (http://0.0.0.0:5000) ...

Shutting down server on port 5000.
```

In [ ]: