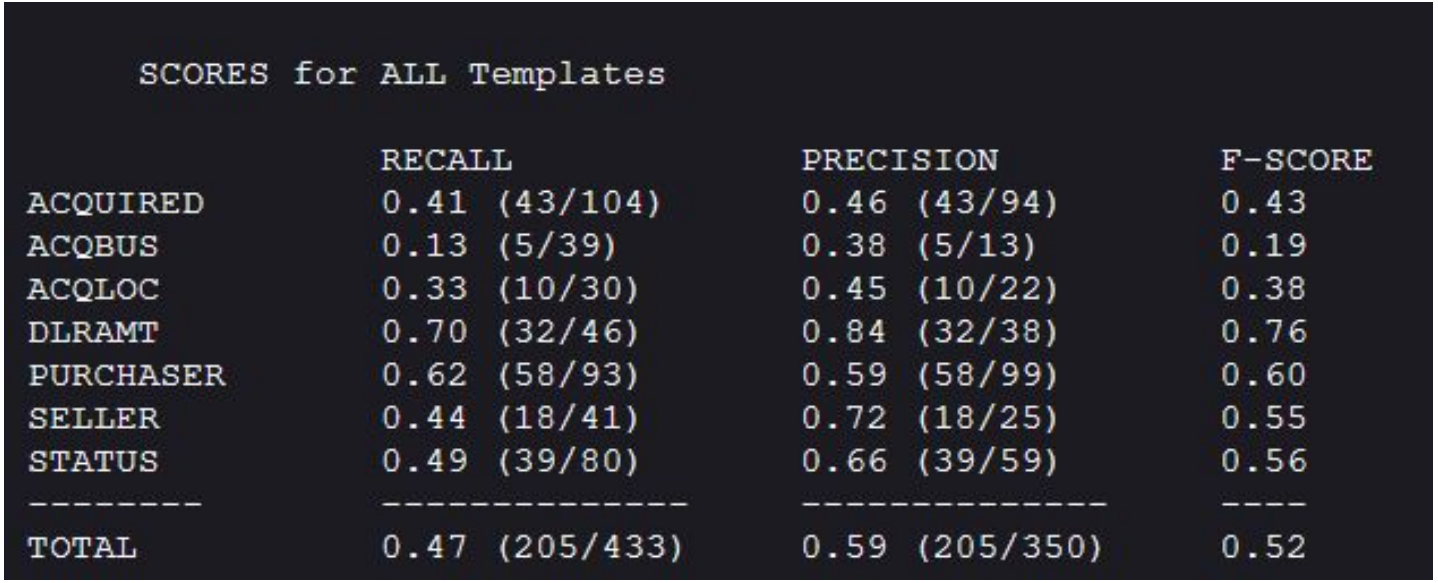
NLP Final Project: Team Parsertongue

By: Marissa Angell and Brandon Walters

Overview

- 6th place
- Uses SpaCy, manual pattern recognition, and regex to implement extraction



```
     SCORES for ALL Templates

                RECALL              PRECISION           F-SCORE
ACQUIRED        0.41 (43/104)       0.46 (43/94)        0.43
ACQBUS          0.13 (5/39)         0.38 (5/13)         0.19
ACQLOC          0.33 (10/30)        0.45 (10/22)        0.38
DLRAMT          0.70 (32/46)        0.84 (32/38)        0.76
PURCHASER       0.62 (58/93)        0.59 (58/99)        0.60
SELLER          0.44 (18/41)        0.72 (18/25)        0.55
STATUS          0.49 (39/80)        0.66 (39/59)        0.56
--------        --------------      --------------      ----
TOTAL           0.47 (205/433)      0.59 (205/350)      0.52
```

System Architecture & Methods

External Resources:
spaCy (https://spacy.io/)

spaCy

DependencyParser

Entity Recognizer

Morphologizer

POS Tagger

Pruning

Manually-Defined Rules

Subject/Object Analysis

Regular Expressions

Keyword Identification

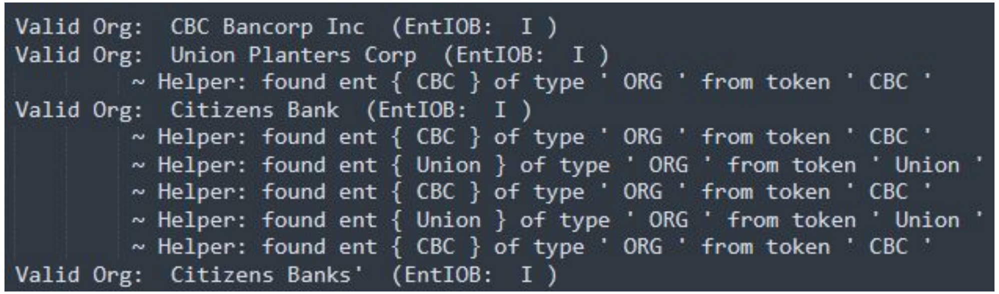
Noun Phrase Extraction

Pruning

Output

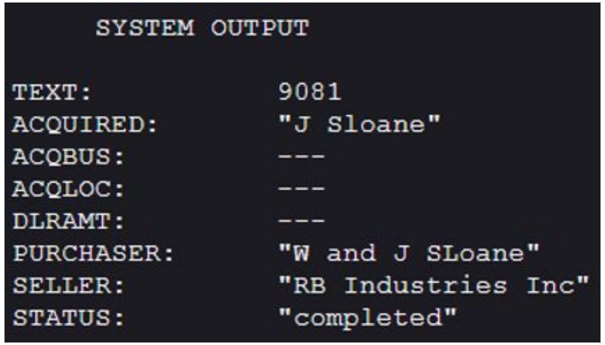
Organizations (Acquired, Purchaser, Seller)

- Used SpaCy’s Named Entity Recognition to find all words tagged “ORG”.
- Pruning and an attempt at Coreference Resolution
    - Strip whitespace, punctuation, leading words like “the”
    - If any new ORG is a substring of a *previous* one, assume it is a coreference.



```
Valid Org:  CBC Bancorp Inc  (EntIOB:  I )
Valid Org:  Union Planters Corp  (EntIOB:  I )
         ~ Helper: found ent { CBC } of type ' ORG ' from token ' CBC '
Valid Org:  Citizens Bank  (EntIOB:  I )
         ~ Helper: found ent { CBC } of type ' ORG ' from token ' CBC '
         ~ Helper: found ent { Union } of type ' ORG ' from token ' Union '
         ~ Helper: found ent { CBC } of type ' ORG ' from token ' CBC '
         ~ Helper: found ent { Union } of type ' ORG ' from token ' Union '
         ~ Helper: found ent { CBC } of type ' ORG ' from token ' CBC '
Valid Org:  Citizens Banks'  (EntIOB:  I )
```

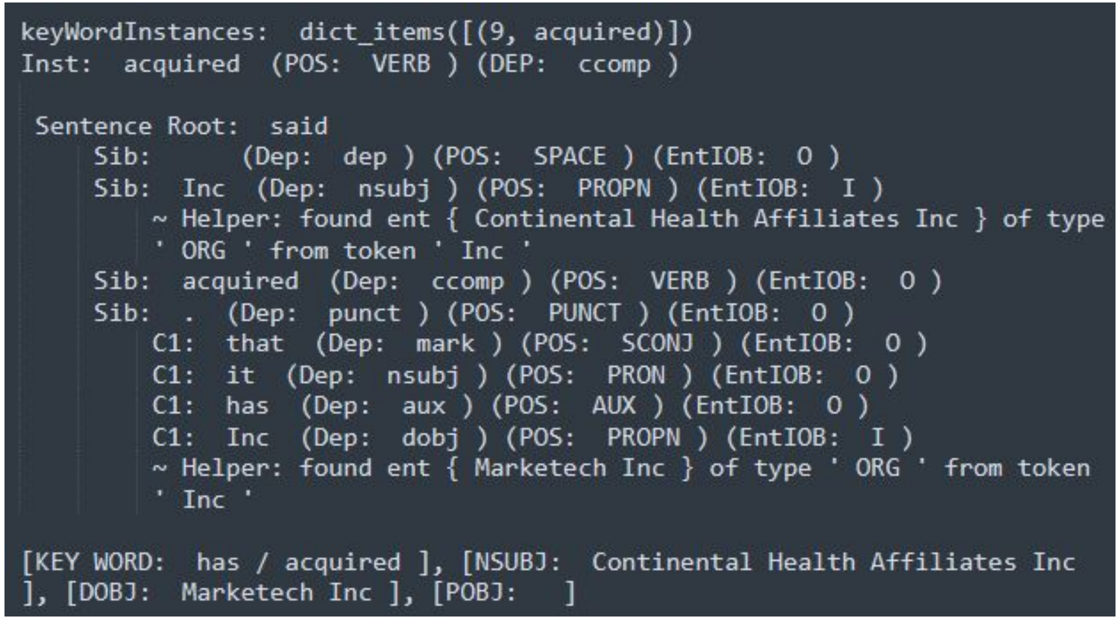
Coreferences were still troublesome...



```
      SYSTEM OUTPUT

TEXT:             9081
ACQUIRED:         "J Sloane"
ACQBUS:           ---
ACQLOC:           ---
DLRAMT:           ---
PURCHASER:        "W and J SLoane"
SELLER:           "RB Industries Inc"
STATUS:           "completed"
```

Organizations, cont.

Then, use morphological analysis with keywords to try and match orgs to roles

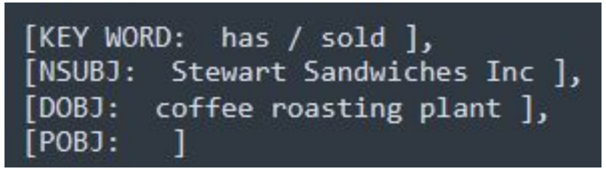


```
keyWordInstances:  dict_items([(9, acquired)])
Inst:  acquired  (POS:  VERB ) (DEP:  ccomp )

 Sentence Root:  said
     Sib:       (Dep:  dep ) (POS:  SPACE ) (EntIOB:  O )
     Sib:  Inc  (Dep:  nsubj ) (POS:  PROPN ) (EntIOB:  I )
         ~ Helper: found ent { Continental Health Affiliates Inc } of type
         ' ORG ' from token ' Inc '
     Sib:  acquired  (Dep:  ccomp ) (POS:  VERB ) (EntIOB:  O )
     Sib:  .  (Dep:  punct ) (POS:  PUNCT ) (EntIOB:  O )
         C1:  that  (Dep:  mark ) (POS:  SCONJ ) (EntIOB:  O )
         C1:  it  (Dep:  nsubj ) (POS:  PRON ) (EntIOB:  O )
         C1:  has  (Dep:  aux ) (POS:  AUX ) (EntIOB:  O )
         C1:  Inc  (Dep:  dobj ) (POS:  PROPN ) (EntIOB:  I )
         ~ Helper: found ent { Marketech Inc } of type ' ORG ' from token
         ' Inc '

[KEY WORD:  has / acquired ], [NSUBJ:  Continental Health Affiliates Inc
], [DOBJ:  Marketech Inc ], [POBJ:   ]
```

Trying to match sentence patterns:

- "__ has acquired __"
- "__ has been acquired by __"
- "__ sold __ to __"
- "__ purchased __ from __"

Limited success with non-ORG acquired entities:



```
[KEY WORD:  has / sold ],
[NSUBJ:  Stewart Sandwiches Inc ],
[DOBJ:  coffee roasting plant ],
[POBJ:   ]
```

Organizations, cont.

If no patterns matched… use baseless assumptions!

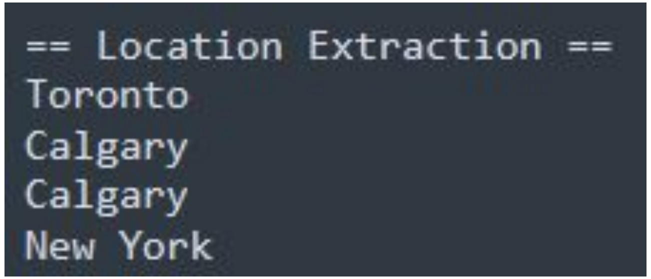
- First valid organization is ‘Acquired’
- If there is another valid organization, use it for ‘Purchaser’
- (Don’t assume seller)

Surprisingly successful!

- This single set of assumptions improved midpoint F-score by over 0.10

Acquired Location

- Used SpaCy's Named Entity Recognition to find all words tagged "GPE" (**G**eo-**P**olitical **E**ntity).
    - Example of SpaCy's GPE list for a document:

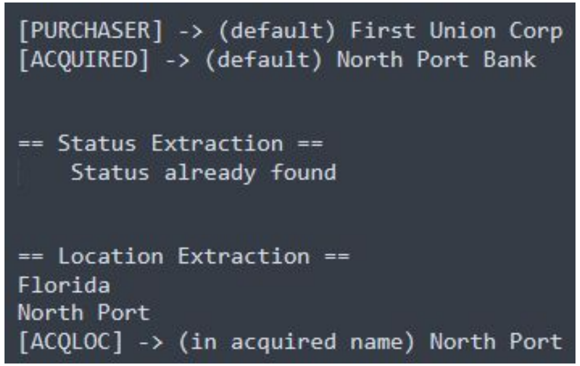


```
== Location Extraction ==
Toronto
Calgary
Calgary
New York
```

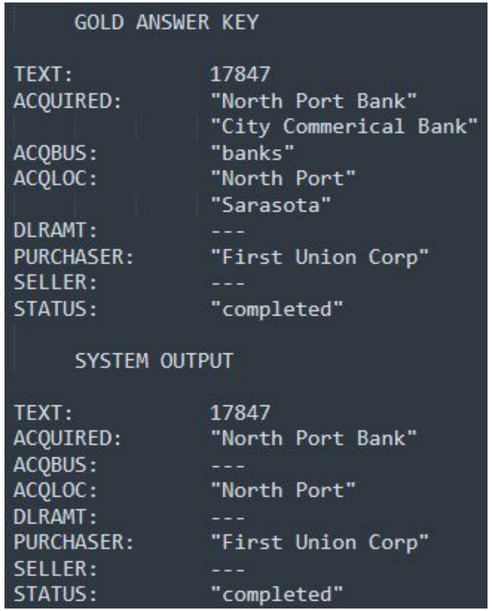
- Then...

Acquired Location, cont.

1. If any entity was a substring of Acquired, use it for AcqLoc.



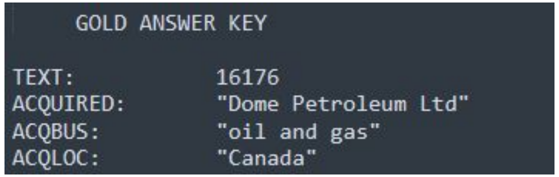
```
[PURCHASER] -> (default) First Union Corp
[ACQUIRED] -> (default) North Port Bank


== Status Extraction ==
   Status already found


== Location Extraction ==
Florida
North Port
[ACQLOC] -> (in acquired name) North Port
```



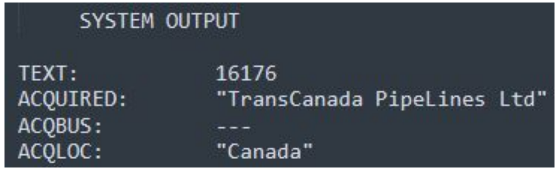
```
    GOLD ANSWER KEY

TEXT:           17847
ACQUIRED:       "North Port Bank"
                "City Commerical Bank"
ACQBUS:         "banks"
ACQLOC:         "North Port"
                "Sarasota"
DLRAMT:         ---
PURCHASER:      "First Union Corp"
SELLER:         ---
STATUS:         "completed"

    SYSTEM OUTPUT

TEXT:           17847
ACQUIRED:       "North Port Bank"
ACQBUS:         ---
ACQLOC:         "North Port"
DLRAMT:         ---
PURCHASER:      "First Union Corp"
SELLER:         ---
STATUS:         "completed"
```

Caveat: often relied on having correctly identified Acquired...

...but sometimes paid off anyway!



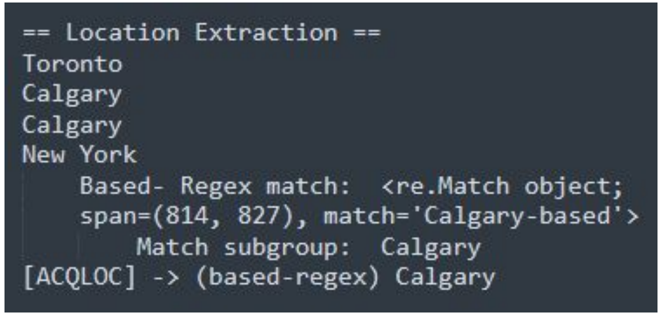
```
    GOLD ANSWER KEY

TEXT:           16176
ACQUIRED:       "Dome Petroleum Ltd"
ACQBUS:         "oil and gas"
ACQLOC:         "Canada"
```



```
    SYSTEM OUTPUT

TEXT:           16176
ACQUIRED:       "TransCanada PipeLines Ltd"
ACQBUS:         ---
ACQLOC:         "Canada"
```

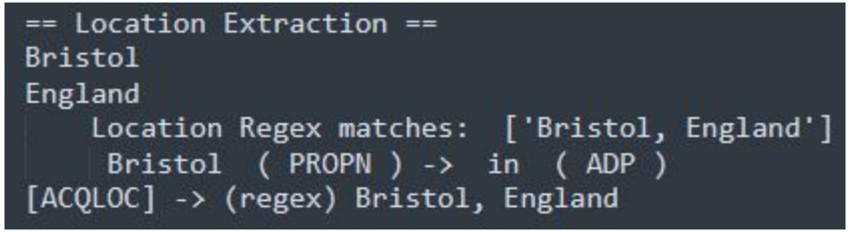
Acquired Location, cont.

2. Else, if a phrase like "Xxxx-based" appeared in the document, extract the first part and use it for AcqLoc.



```
== Location Extraction ==
Toronto
Calgary
Calgary
New York
    Based- Regex match:  <re.Match object;
    span=(814, 827), match='Calgary-based'>
        Match subgroup:  Calgary
[ACQLOC] -> (based-regex) Calgary
```

3. Else, extract strings that match the pattern "Xxxx, Xxxx", and use the first with all words tagged as GPE for AcqLoc.



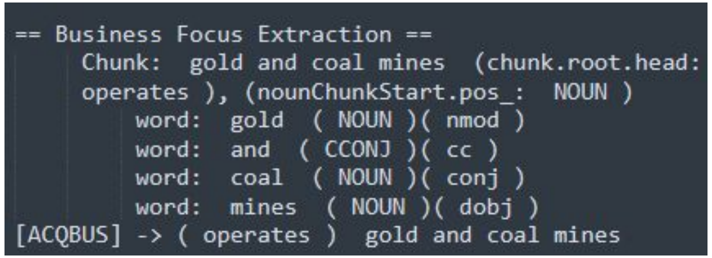
```
== Location Extraction ==
Bristol
England
    Location Regex matches:  ['Bristol, England']
     Bristol  ( PROPN ) ->  in  ( ADP )
[ACQLOC] -> (regex) Bristol, England
```

Here, SpaCy tagged 'Bristol' and 'England' as separate GPEs, but it was accounted for with this rule.

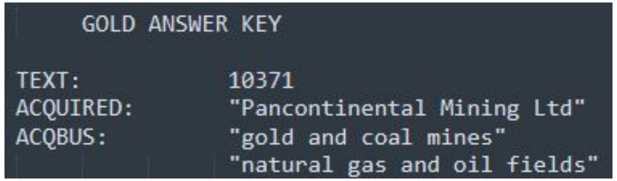
Acquired Business Focus

- Used SpaCy's "noun-chunk" feature to find nouns with a specific base root verb ("sells", "manufactures", "distributes", etc.)
- If none found, try a keyword regex check
- This was our worst category:
  - So many edge cases!
  - Many different ways to present business focuses
  - SpaCy noun chunking/POS tagger issues

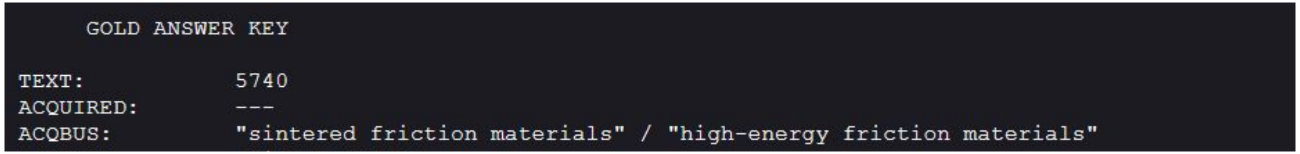
Partial success...



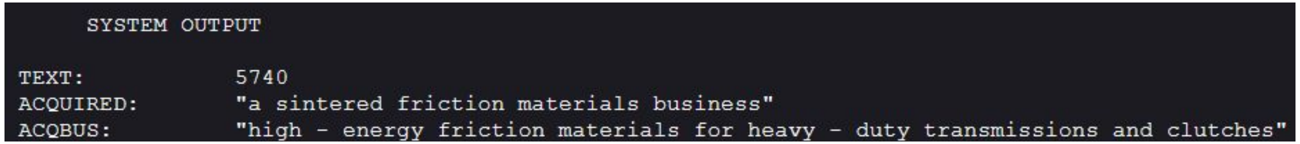
```
== Business Focus Extraction ==
     Chunk:  gold and coal mines  (chunk.root.head:
     operates ), (nounChunkStart.pos_:  NOUN )
         word:  gold  ( NOUN )( nmod )
         word:  and  ( CCONJ )( cc )
         word:  coal  ( NOUN )( conj )
         word:  mines  ( NOUN )( dobj )
[ACQBUS] -> ( operates )  gold and coal mines
```



```
     GOLD ANSWER KEY

TEXT:           10371
ACQUIRED:       "Pancontinental Mining Ltd"
ACQBUS:         "gold and coal mines"
                "natural gas and oil fields"
```



```
     GOLD ANSWER KEY

TEXT:           5740
ACQUIRED:       ---
ACQBUS:         "sintered friction materials" / "high-energy friction materials"
```

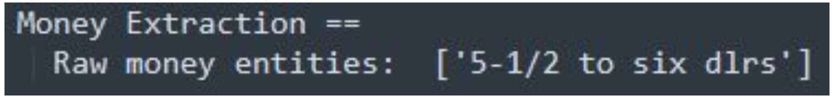


```
     SYSTEM OUTPUT

TEXT:           5740
ACQUIRED:       "a sintered friction materials business"
ACQBUS:         "high - energy friction materials for heavy - duty transmissions and clutches"
```

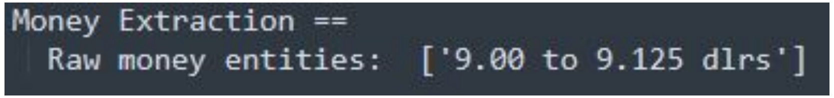
Disappointing miss :(

Dollar Amount

- Gathered all phrases tagged with "MONEY" by SpaCy's NER system
  - Prune phrases that are not likely to be correct...

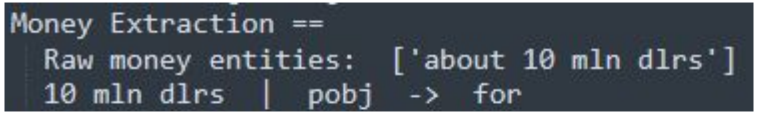


```
Money Extraction ==
  Raw money entities:  ['5-1/2 to six dlrs']
```

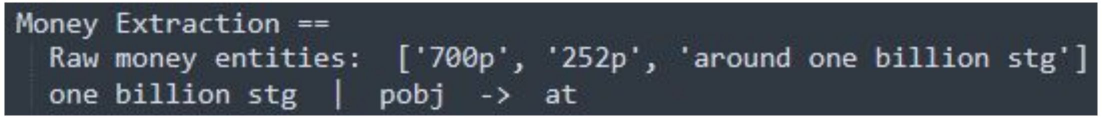


```
Money Extraction ==
  Raw money entities:  ['9.00 to 9.125 dlrs']
```

  - Prune additional words, whitespace, etc.



```
Money Extraction ==
  Raw money entities:  ['about 10 mln dlrs']
  10 mln dlrs  |  pobj  ->  for
```



```
Money Extraction ==
  Raw money entities:  ['700p', '252p', 'around one billion stg']
  one billion stg  |  pobj  ->  at
```

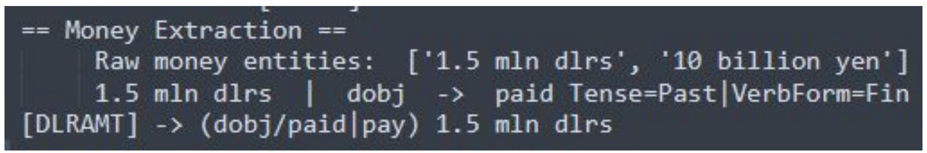
- If any entities survived pruning...

Dollar Amount, cont.

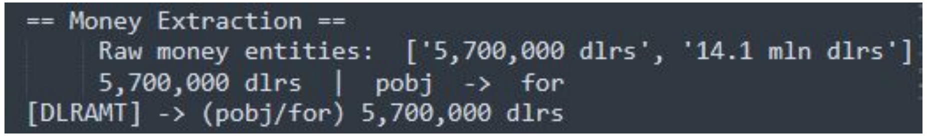
Assume the first ‘correctly-formatted’ dollar amount is the purchase price.

- Use dependency parse patterns to decide if its likely/unlikely to be correct.

If the amount fits a defined dependency parse pattern, use it for DLRAMT.

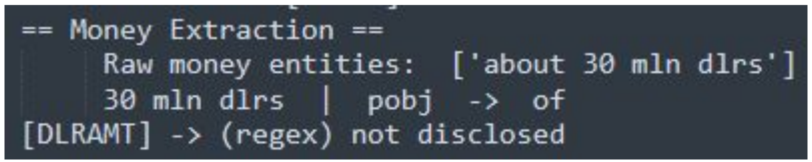


```
== Money Extraction ==
     Raw money entities:  ['1.5 mln dlrs', '10 billion yen']
     1.5 mln dlrs  |  dobj  ->  paid Tense=Past|VerbForm=Fin
[DLRAMT] -> (dobj/paid|pay) 1.5 mln dlrs
```

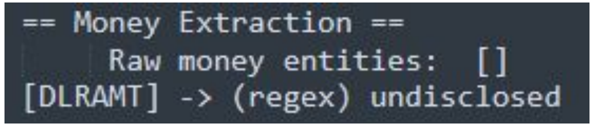


```
== Money Extraction ==
     Raw money entities:  ['5,700,000 dlrs', '14.1 mln dlrs']
     5,700,000 dlrs  |  pobj  ->  for
[DLRAMT] -> (pobj/for) 5,700,000 dlrs
```

If it doesn’t fit any patterns, search the doc for strings like ‘undisclosed’ to use for DLRAMT.



```
== Money Extraction ==
     Raw money entities:  ['about 30 mln dlrs']
     30 mln dlrs  |  pobj  ->  of
[DLRAMT] -> (regex) not disclosed
```



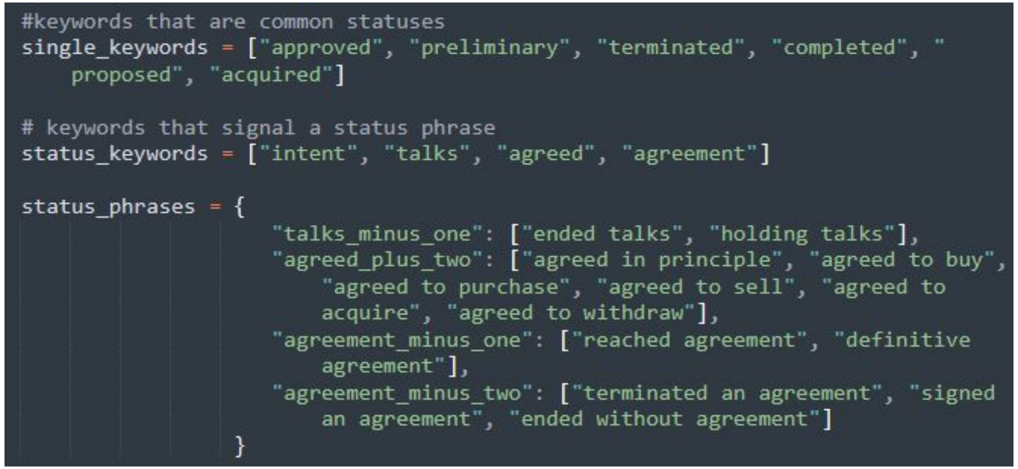
```
== Money Extraction ==
     Raw money entities:  []
[DLRAMT] -> (regex) undisclosed
```

Deal Status

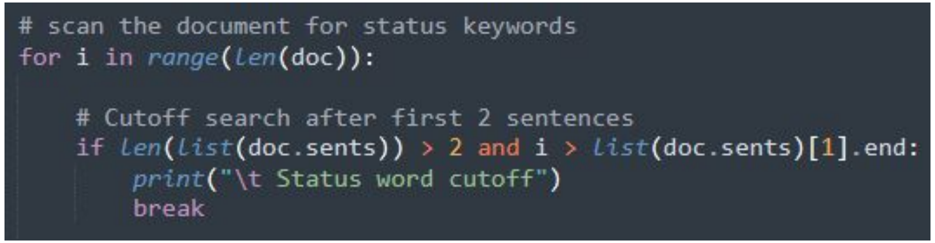
- Try to pull specific phrases from the document ("reached agreement", "terminated", etc.)

- Caught more than 50% of all status phrases
    - Initial Doclist: 0.52 Recall, 0.66 Precision, F-Score of 0.58

- Very Helpful: if no keywords in the first 2 sentences, abandon the search!



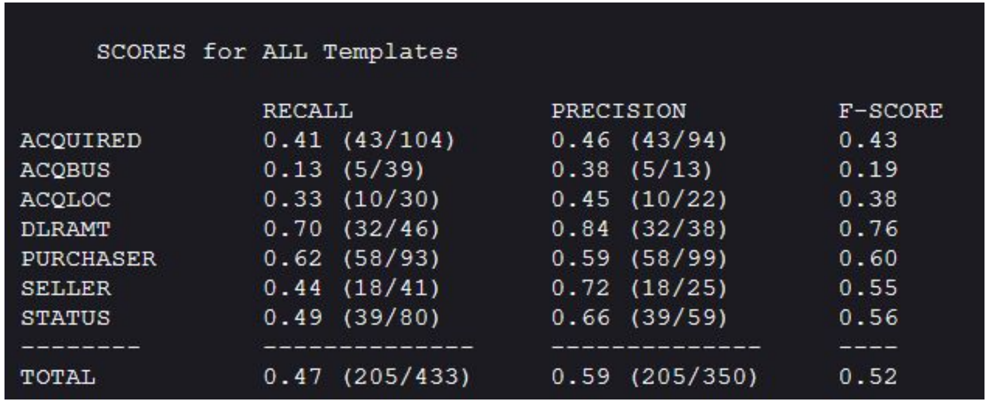
```
#keywords that are common statuses
single_keywords = ["approved", "preliminary", "terminated", "completed", "
    proposed", "acquired"]

# keywords that signal a status phrase
status_keywords = ["intent", "talks", "agreed", "agreement"]

status_phrases = {
                    "talks_minus_one": ["ended talks", "holding talks"],
                    "agreed_plus_two": ["agreed in principle", "agreed to buy",
                        "agreed to purchase", "agreed to sell", "agreed to
                        acquire", "agreed to withdraw"],
                    "agreement_minus_one": ["reached agreement", "definitive
                        agreement"],
                    "agreement_minus_two": ["terminated an agreement", "signed
                        an agreement", "ended without agreement"]
                }
```



```
# scan the document for status keywords
for i in range(len(doc)):

    # Cutoff search after first 2 sentences
    if len(list(doc.sents)) > 2 and i > list(doc.sents)[1].end:
        print("\t Status word cutoff")
        break
```

Performance Summary

Test Set 2

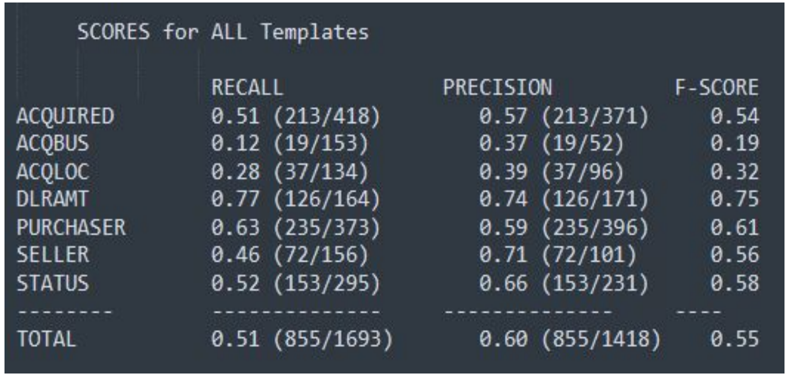


```
     SCORES for ALL Templates

                RECALL             PRECISION          F-SCORE
ACQUIRED        0.41 (43/104)      0.46 (43/94)       0.43
ACQBUS          0.13 (5/39)        0.38 (5/13)        0.19
ACQLOC          0.33 (10/30)       0.45 (10/22)       0.38
DLRAMT          0.70 (32/46)       0.84 (32/38)       0.76
PURCHASER       0.62 (58/93)       0.59 (58/99)       0.60
SELLER          0.44 (18/41)       0.72 (18/25)       0.55
STATUS          0.49 (39/80)       0.66 (39/59)       0.56
--------        --------------     --------------     ----
TOTAL           0.47 (205/433)     0.59 (205/350)     0.52
```

Notes:

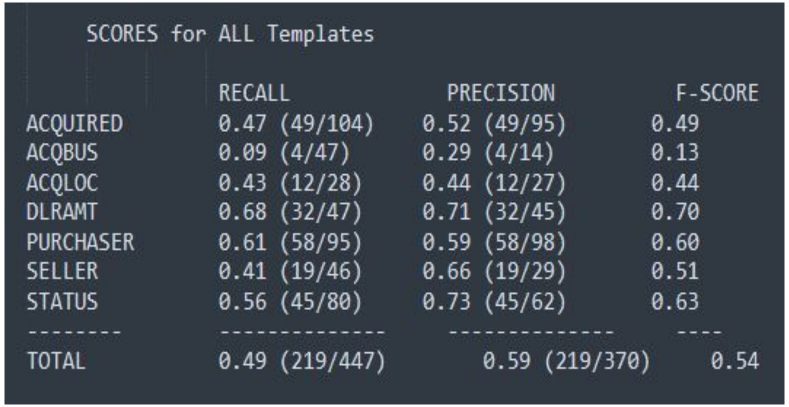
- **AcqBus** and **DlrAmt** did **better** than expected
- **Acquired** and **Status** did **worse** than expected

Initial Documents



```
     SCORES for ALL Templates

                RECALL             PRECISION          F-SCORE
ACQUIRED        0.51 (213/418)       0.57 (213/371)     0.54
ACQBUS          0.12 (19/153)        0.37 (19/52)       0.19
ACQLOC          0.28 (37/134)        0.39 (37/96)       0.32
DLRAMT          0.77 (126/164)       0.74 (126/171)     0.75
PURCHASER       0.63 (235/373)       0.59 (235/396)     0.61
SELLER          0.46 (72/156)        0.71 (72/101)      0.56
STATUS          0.52 (153/295)       0.66 (153/231)     0.58
--------        --------------     --------------     ----
TOTAL           0.51 (855/1693)      0.60 (855/1418)    0.55
```

Test Set 1



```
     SCORES for ALL Templates

                RECALL             PRECISION          F-SCORE
ACQUIRED        0.47 (49/104)      0.52 (49/95)       0.49
ACQBUS          0.09 (4/47)        0.29 (4/14)        0.13
ACQLOC          0.43 (12/28)       0.44 (12/27)       0.44
DLRAMT          0.68 (32/47)       0.71 (32/45)       0.70
PURCHASER       0.61 (58/95)       0.59 (58/98)       0.60
SELLER          0.41 (19/46)       0.66 (19/29)       0.51
STATUS          0.56 (45/80)       0.73 (45/62)       0.63
--------        --------------     --------------     ----
TOTAL           0.49 (219/447)       0.59 (219/370)     0.54
```

Successes and Regrets

Successes:

- Identified simple patterns that tended to be correct more often than not
- Got consistent results, no drastic over-fitting or under-fitting
- Good precision scores across the board, had accurate guesses

Regrets:

- Some low recalls, not guessing as often as we’d like, slight over-fitting for some fields
- AcqBus
- Not having more time to work!

Lessons Learned

- Language is predictable
  - Keywords and simple assumptions can perform extremely well
- Getting ‘almost-correct’ is much easier than expected
  - Matching Gold exactly was the hardest challenge
- Manually-defined rules are rewarding
  - ...and take SO MUCH TIME.

Thanks!