Northeastern University

INFO 6105
Data Sci Eng Mth & Tools
Lecture 2 WordCloud Labs

11 September 2022

Part 1

WHAT IS DATA SCIENCE?

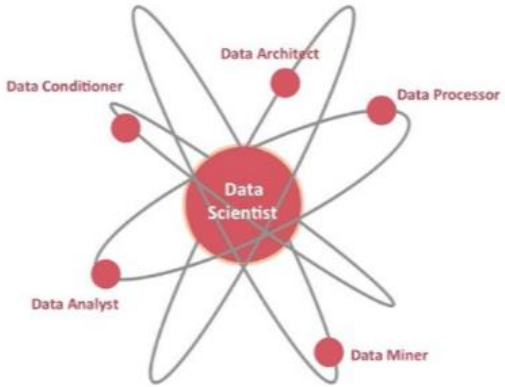
What is Data Science

- **Using automated methods to analyze sometimes massive amounts of data for the purpose of extracting knowledge learned from derived results**
- **Extracting resulting insights used to understand and improve *everything***

3

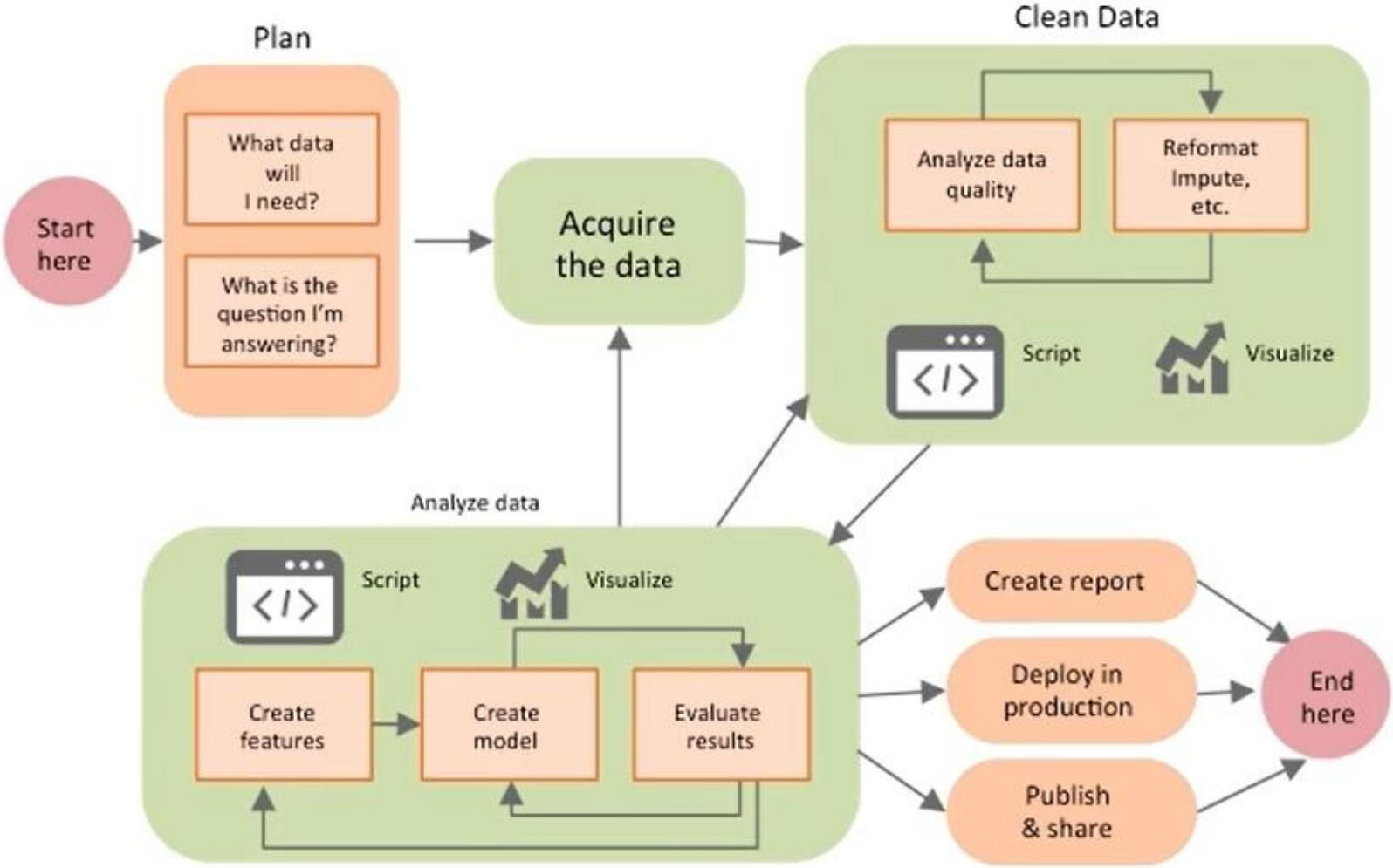
Data Science skillset

- ***Statistician***
  - **Individual who models and summarizes datasets**
- ***Mathematician***
  - **Linear Algebra the lingua franca of Deep Learning**
- ***Computer scientist***
  - **Writes algorithms to summarize, visualize, and store data**
- ***Domain expertise***
  - **Subject matter expert**



4

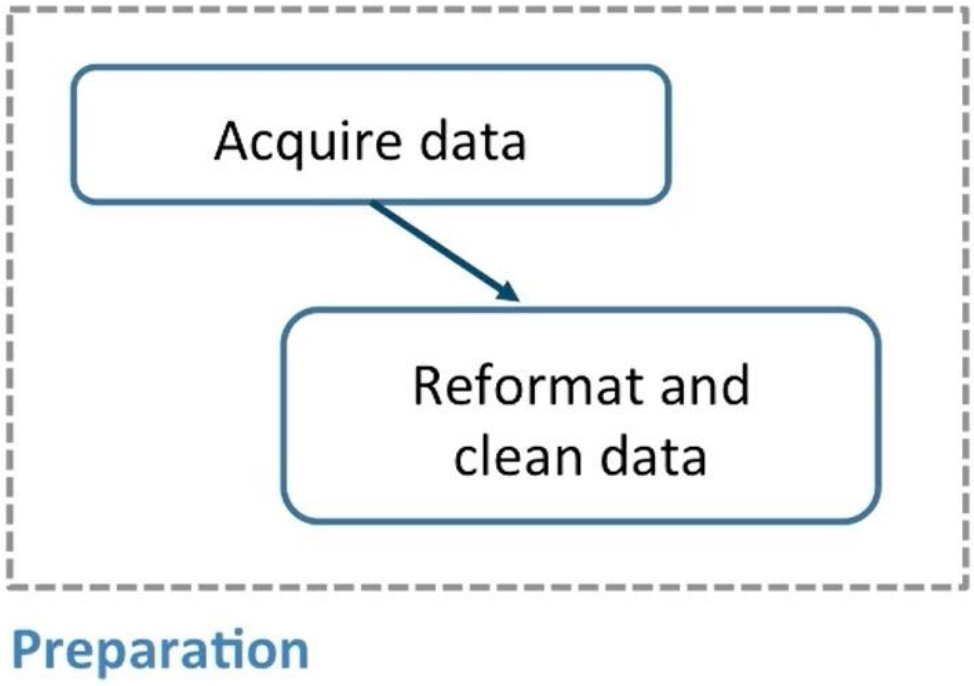
Data Science Process Flow



5

Stage 1 – Planning/Preparation

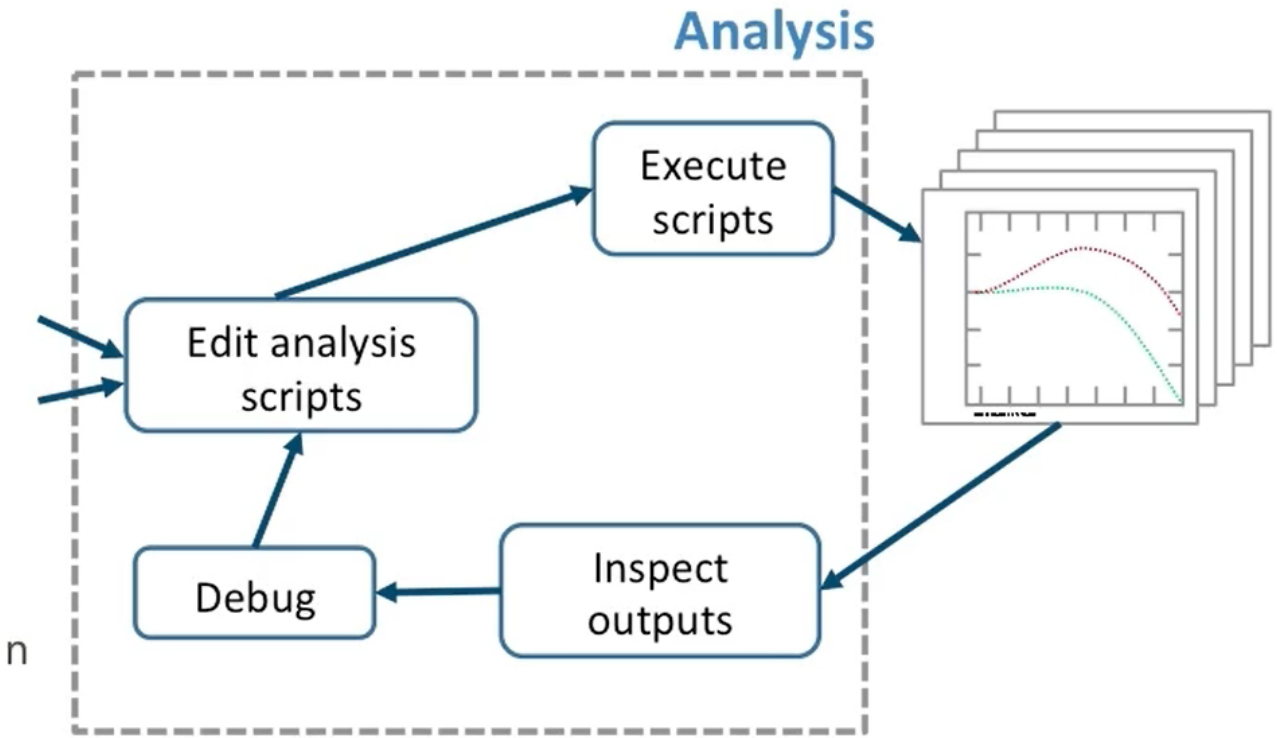
- **Obtain data**
- **Format and clean data**



6

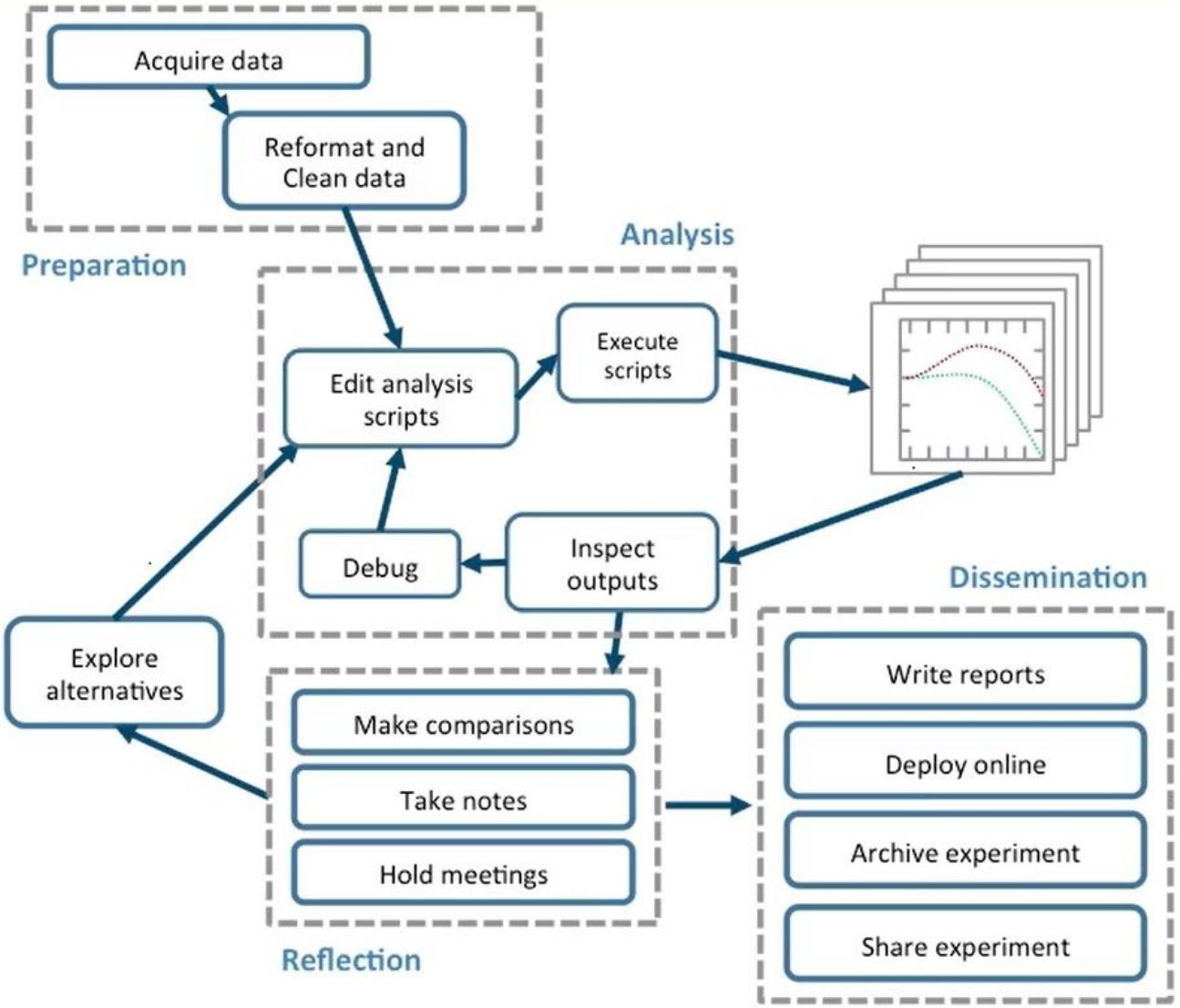
Stage 2 - Analysis

- Create scripts for analysis
- Run scripts
- Debug
- Inspect output
- Debug
- Repeat

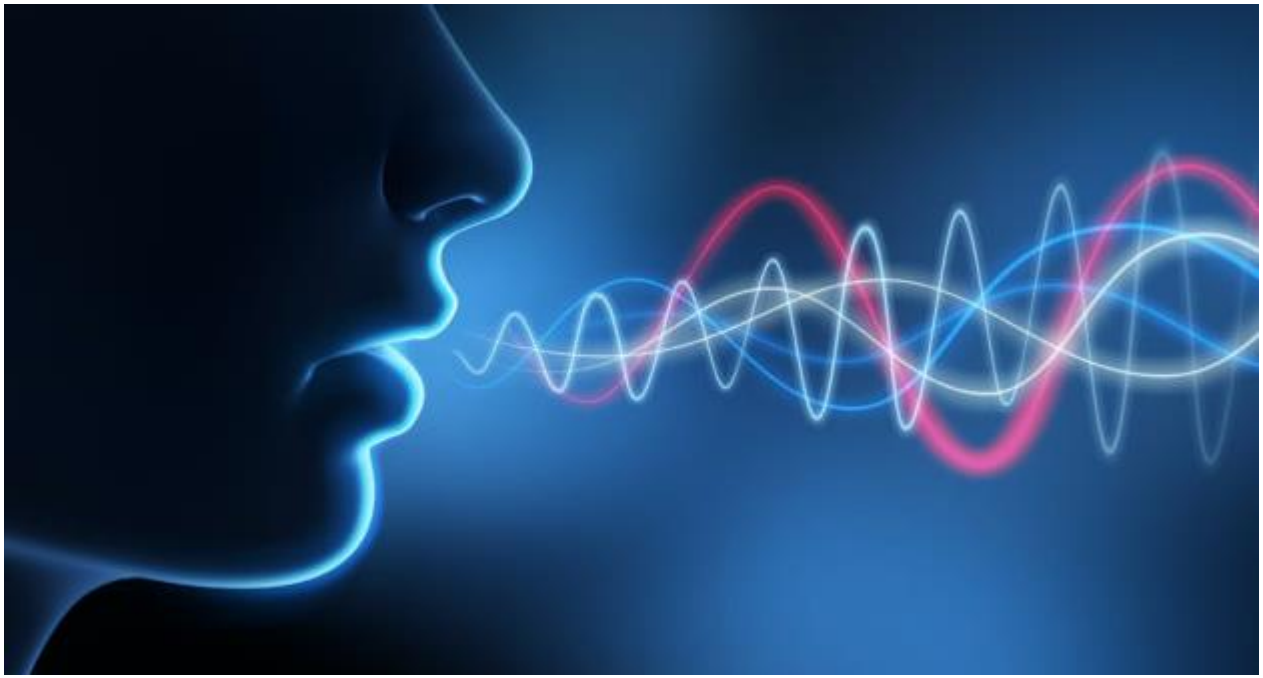


7

Stage 3 - Publish



8



Part 2

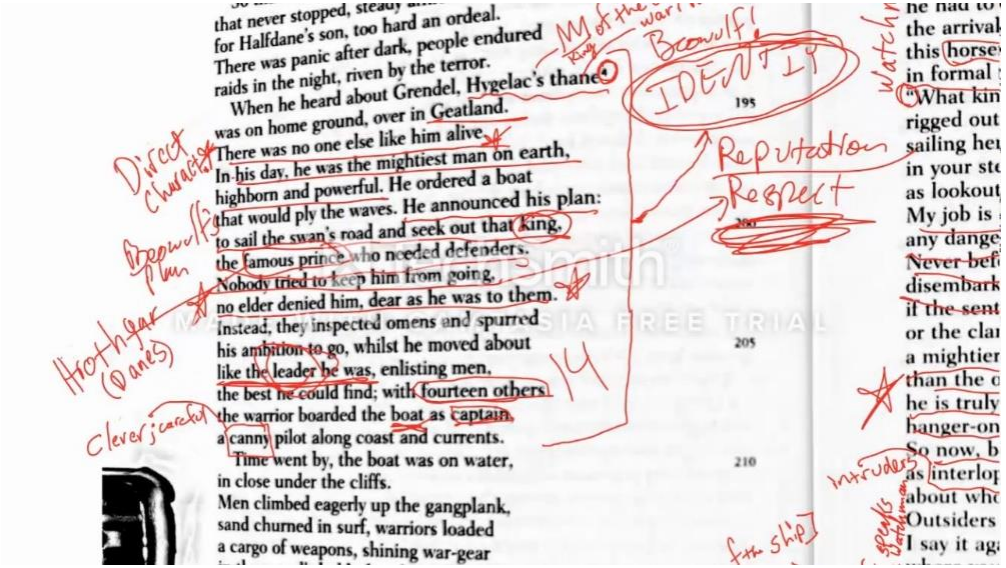
EXAMPLE: NATURAL LANGUAGE PROCESSING (NLP)

What is NLP?

- *Natural Language Processing* (NLP) is a field of computer science and engineering that has developed from the study of language and computational linguistics within the field of Artificial Intelligence
- The goals of NLP are to design and build applications that facilitate human interaction with machines through the use of natural language
  - Document classification, machine translation, and speech recognition are major areas of NLP
- 2018 was a milestone year for NLP and machines, because machines achieved human parity in translation (all languages) *and* speech to text
  - In other words, machine perform *as well as* best human experts in the field
- MGEN has an advanced ML/NLP class that you can take after you take this class

Annotations

- It is not enough to simply provide a computer with a large amount of data and expect it to learn to speak—the data has to be prepared in such a way that the computer can more easily find patterns and inferences
- This can be accomplished by adding (or machine-*learning*) relevant *metadata* to a dataset. Any metadata tag used to mark up elements of the dataset is called an ***annotation*** over the input



11

What kinds of Annotations?

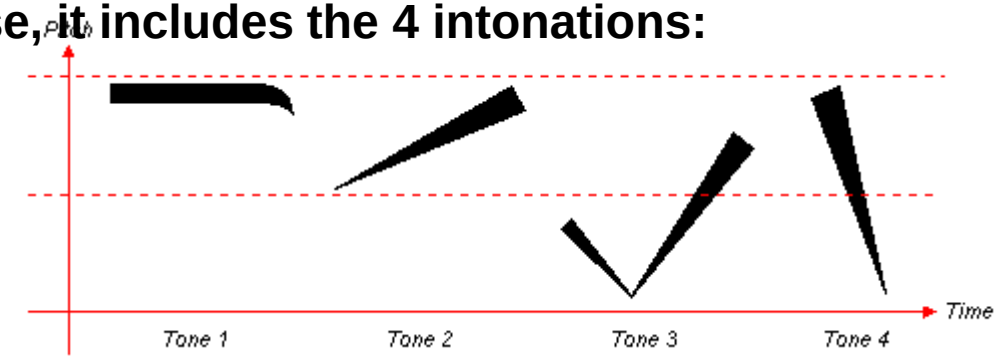
- ***Grammar*** is the name typically given to the mechanisms responsible for creating well-formed structures in language
- Most linguists view grammar as itself consisting of distinct modules or systems, either by cognitive design or for descriptive convenience These areas usually include:
- **Syntax**
  - **The study of how words are combined to form sentences. This includes examining parts of speech and how they combine to make larger constructions ("*I love you*", "*Wo ai ni*", etc..)**
- **Semantics**
  - **The study of meaning in language. Semantics examines the relations between words and what they are being used to represent.**
- **Morphology**
  - **The study of units of meaning in a language. A morpheme is the smallest unit of language that has meaning or function, a definition that includes words, prefixes, affixes, and other word structures that impart meaning. In Chinese, it's an *ideogram***

(fire)

12

Grammar areas (continued)

- **Phonology**
  - The study of the sound patterns of a particular language
  - Aspects of study include determining which patterns are significant and have meaning (i.e., the phonemes); how syllables are structured and combined; and what features are needed to describe the discrete units (segments) in the language, and how they are interpreted
  - In Chinese, it includes the 4 intonations:



- **Phonetics**
  - The study of the sounds of human speech, and how they are made and perceived. A phoneme is the term for an individual sound, and is essentially the *smallest* unit of human speech.

Grammar areas (continued)

- **Lexicon**
  - **The study of the words and phrases used in a language, that is, a language's vocabulary.**
- **Discourse analysis**
  - **The study of exchanges of information, usually in the form of conversations, and particularly the flow of information across sentence boundaries.**
- **Pragmatics**
  - **The study of how the context of text affects the meaning of an expression, and what information is necessary to infer a hidden or presupposed meaning.**
- **Text structure analysis**
  - **The study of how narratives and other textual styles are constructed to make larger textual compositions**

Reference: *adapted from* **Natural Language Annotation for Machine Learning**

https://www.safaribooksonline.com/library/view/natural-language-annotation/9781449332693/

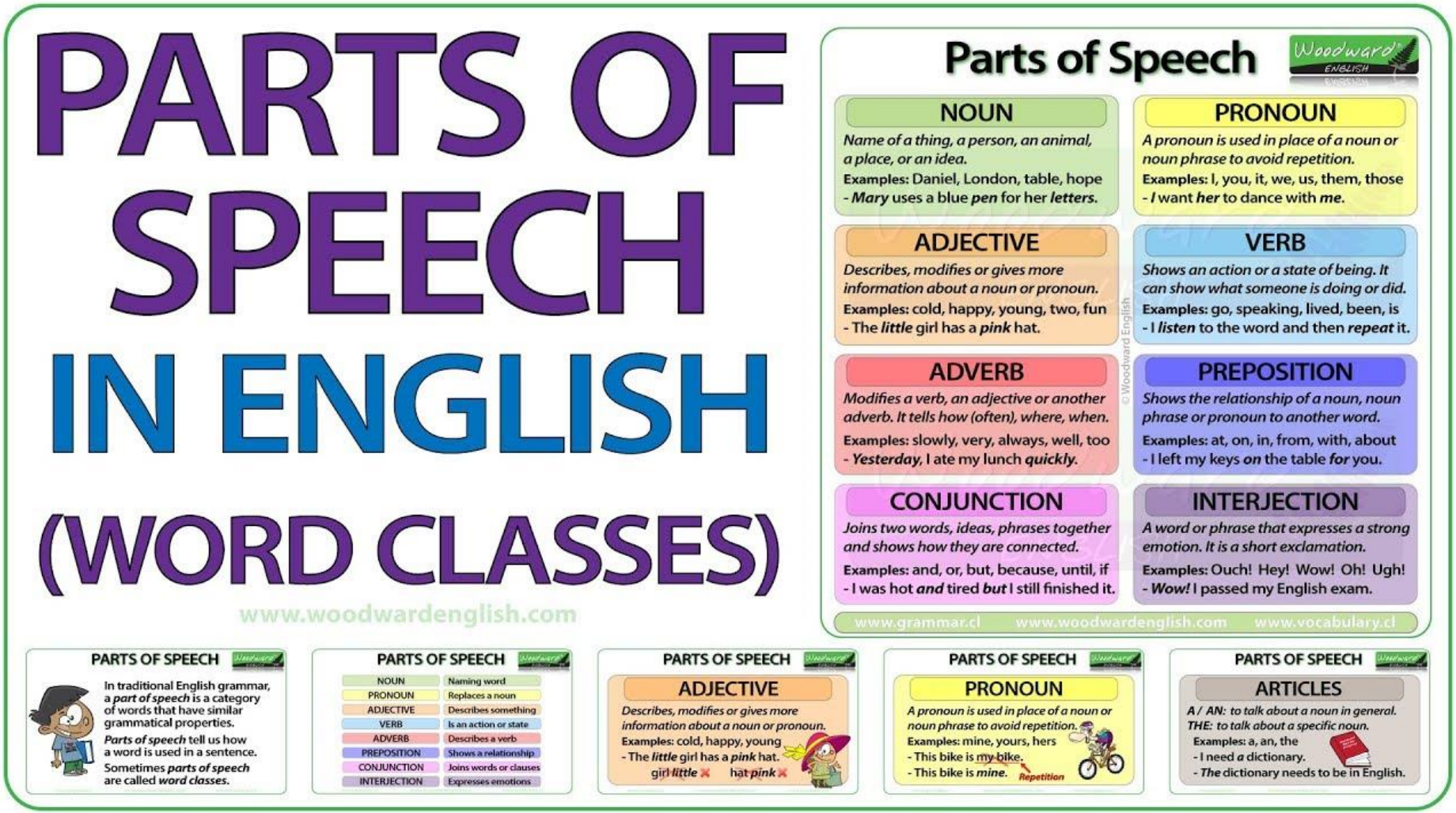
14

Parts of Speech

- In traditional grammar, a **part of speech** (abbreviated as **PoS** or **POS**) is a category of words that have similar *grammatical properties*
- Words that are assigned to the *same* POS generally display *similar* behavior in terms of syntax—they play similar roles within the grammatical structure of sentences—and sometimes in terms of morphology, in that they undergo inflection for similar properties
- Commonly listed *English* parts of speech are *noun*, *verb*, *adjective*, *adverb*, *pronoun*, *preposition*, *conjunction*, and *interjection*

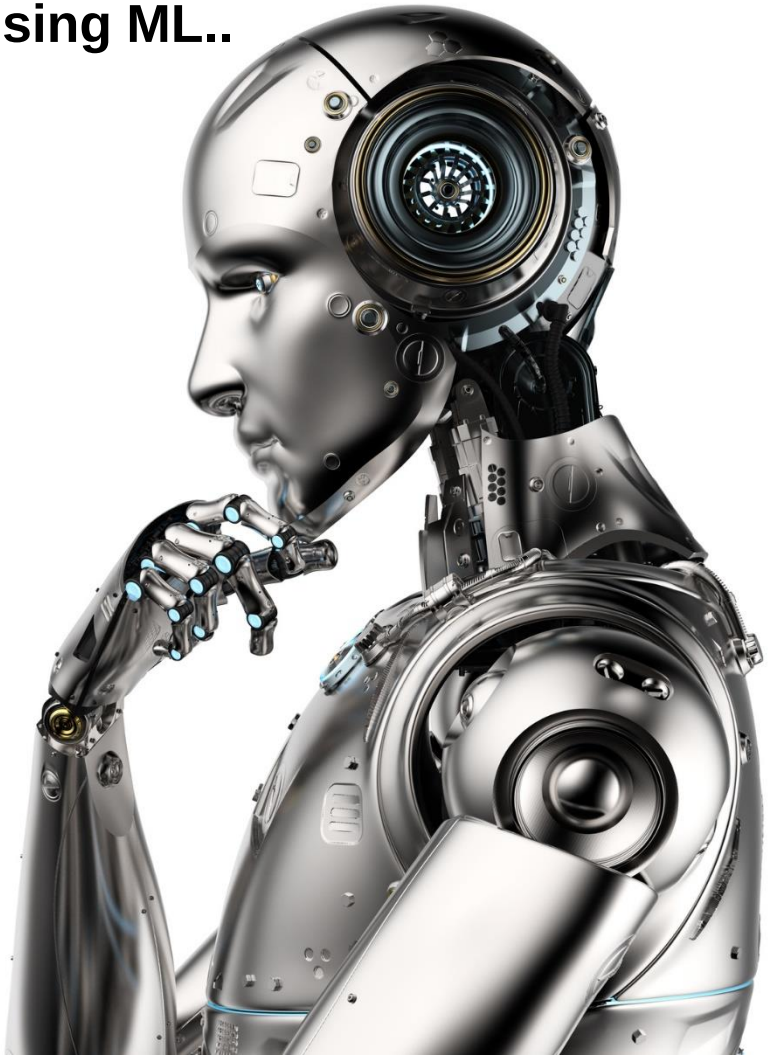
15

Parts of Speech



ML

- **POS can also be *learned* using ML..**

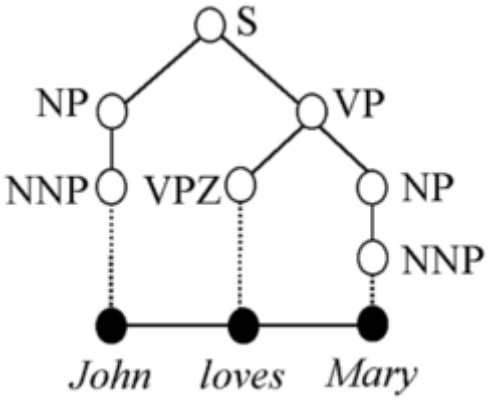


The Penn Treebank

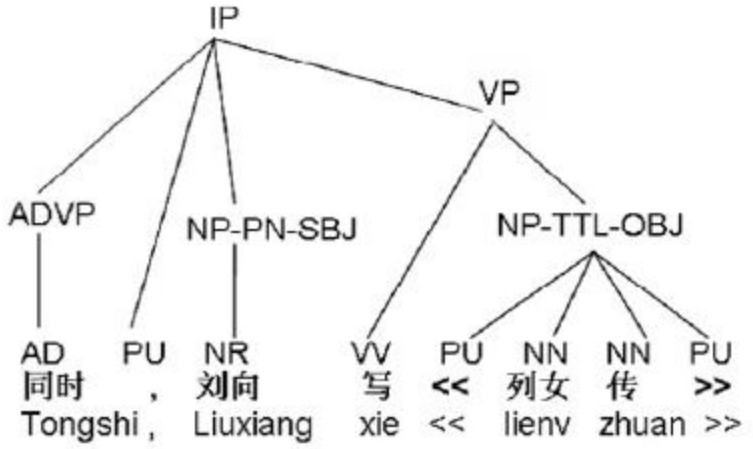
- **In linguistics, a treebank is a parsed text corpus that annotates syntactic or semantic sentence structure**
  - ***Syntax*** **(**from ancient greek: σύνταξις "coordination“) **is the set of rules, principles, and processes that govern the structure of sentences in a given language, usually including word order**
  - ***Semantics*** **(**from ancient greek: σημαντικός, "significant") **is the linguistic and philosophical study of meaning, in language. It is concerned with the relationship between signifiers—like words, phrases, signs, and symbols—and what they stand for, their denotation**
- **The most famous treebank *for the English language* is the Penn Treebank, which in its eight years of operation (1989-1996), produced approximately 7 million words of part-of-speech tagged text, 3 million words of skeletally parsed text, and over 2 million words of text parsed for structure**
  - **Material annotated includes such wide-ranging genres as IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations**
- **https://catalog.ldc.upenn.edu/ldc99t42**

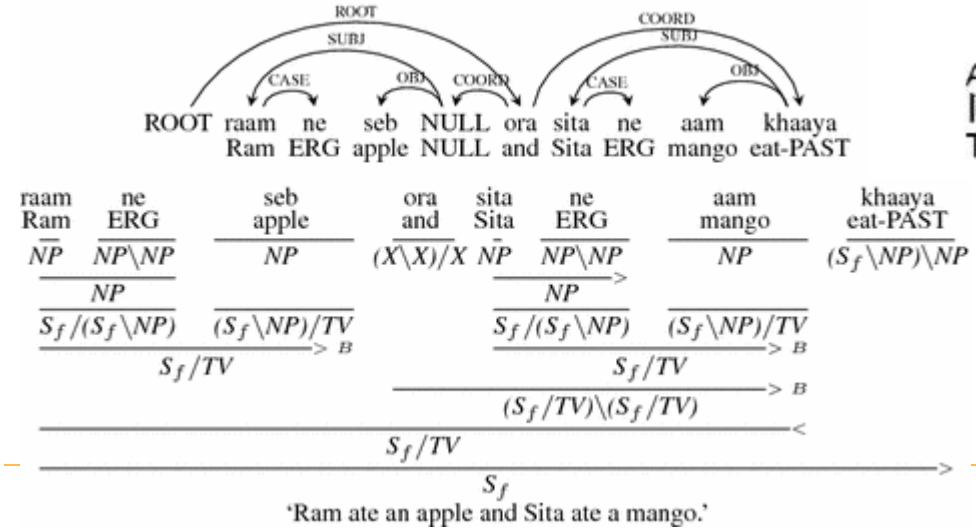
Languages have their own treebanks

- **There are treebanks for each language of the world**
  - **What is *yours?***



bracketed sentence:
(IP (ADVP (AD 同时)) (PU ，) (NP-PN-SBJ (NR 刘向)) (VP (VV 写)
(NP-TTL-OBJ (PU 《) (NN 列女) (NN 传) (PU 》))))
English: At the same time, Liuxiang wrote <<The Biography of Strong-minded Women>>





Universal dependencies integrated Treebank

- Universal Dependencies (UD) is a framework for cross-linguistically consistent grammatical annotation and an open community effort with over 200 contributors producing more than 100 treebanks *in over 60 languages*
  - **http://universaldependencies.org/**
- **Udpipe** provides language-agnostic ‘tokenization’ and ‘parts of speech tagging’, of raw text in many languages, including Chinese and Hindi
- We speak a lot of Chinese and Hindi in class, so this is perfect

小娃撐小艇

偷采白蓮回

不解藏蹤跡

浮萍一道開

मैं तन्हा हूँ मुझे तन्हा ही रहने दो

देखकर मेरे बहते आंसू

तुम अपने लहू न बहने दो

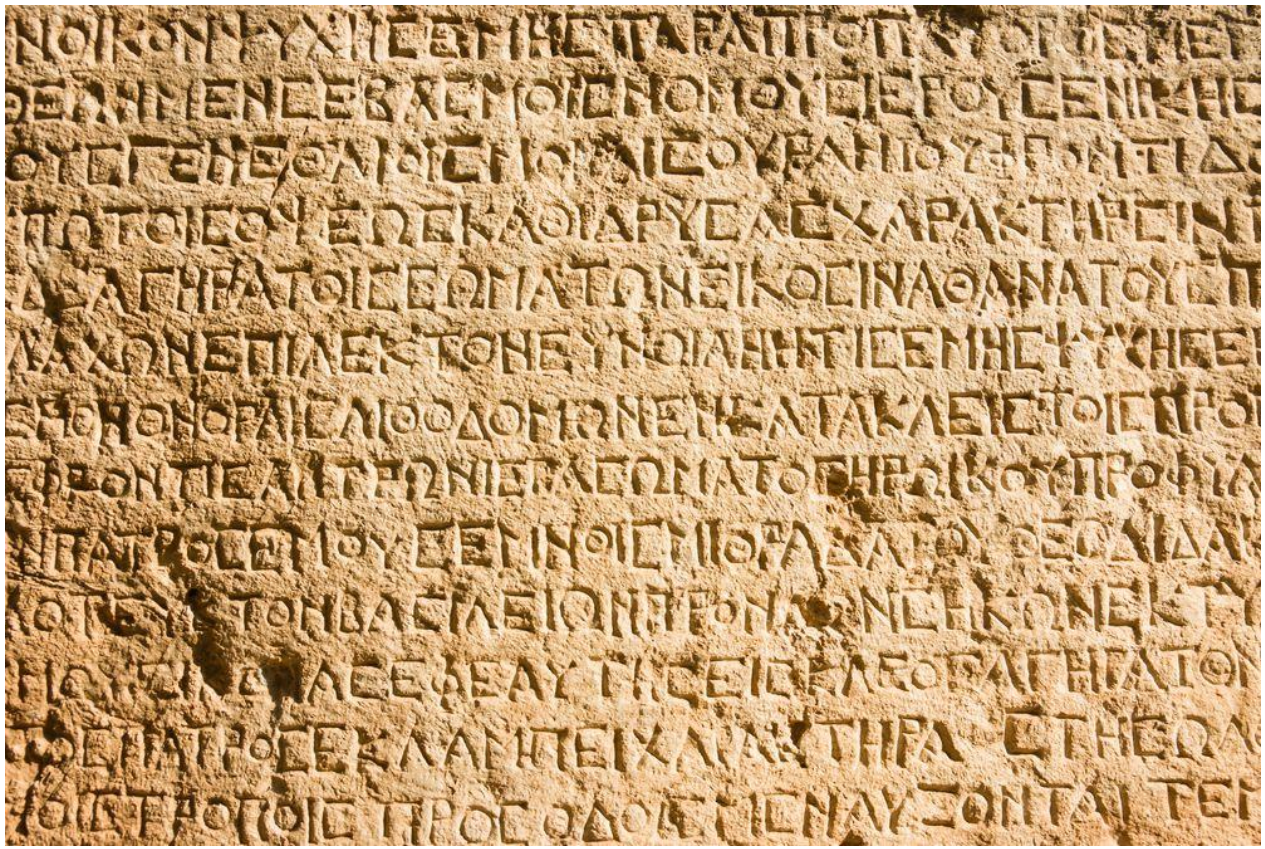
मैं आपका दीवाना हूँ

मुझे बस अपना पागल रहने दो

20

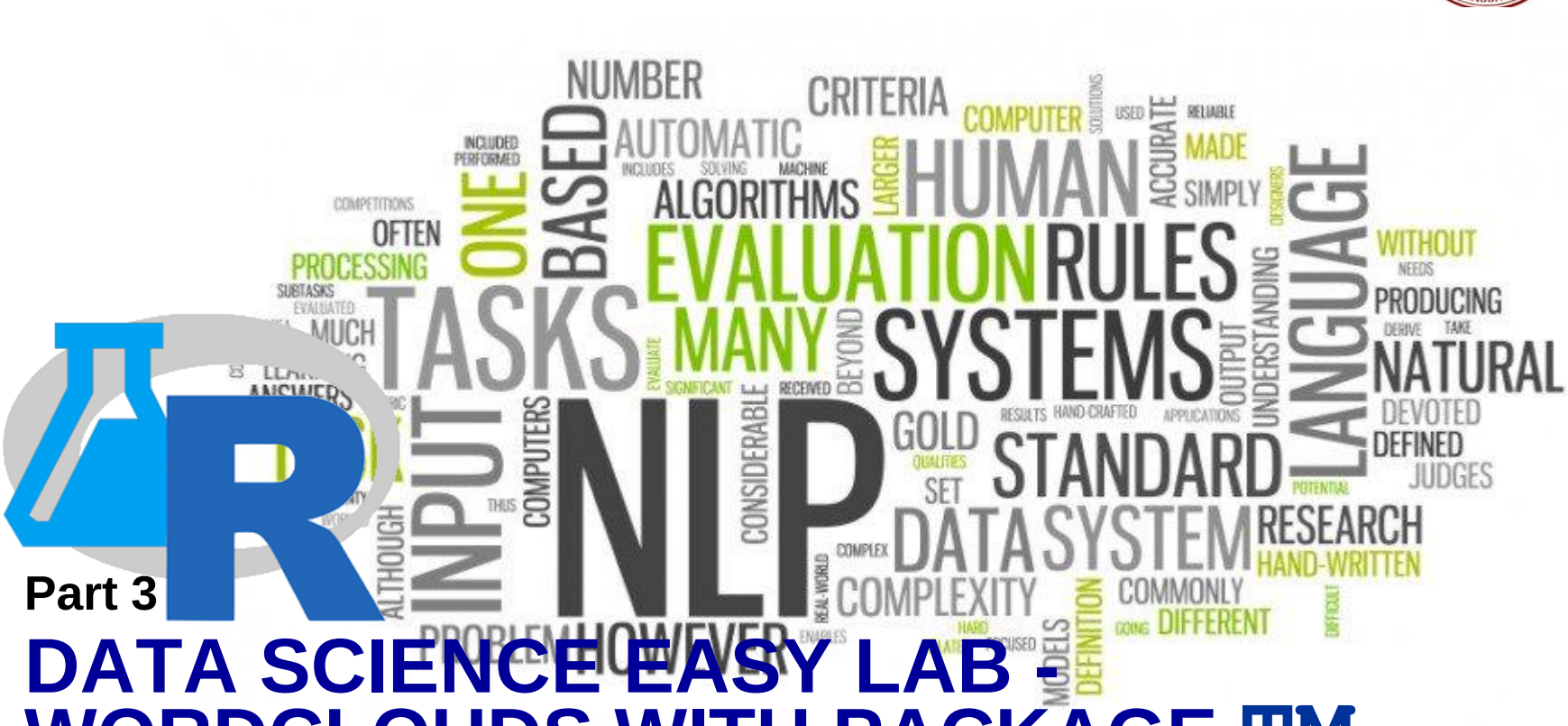
Greek, too

- One of the ancient languages in the world



Universal Dependencies

- Project that seeks to develop cross-linguistically consistent treebank annotation for many languages, with the goal of facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective
- The annotation scheme is based on (universal) Stanford dependencies (de Marneffe et al., 2006, 2008, 2014), Google universal part-of-speech (POS) tags (Petrov et al., 2012), and the Interset interlingua for morphosyntactic tagsets (Zeman, 2008)
- Pre-trained Universal Dependencies 2.0 models on all UD treebanks are made available for more than 50 languages:
  - afrikaans, ancient_greek-proiel, ancient_greek, arabic, basque, belarusian, bulgarian, catalan, chinese, coptic, croatian, czech-cac, czech-cltt, czech, danish, dutch-lassysmall, dutch, english-lines, english-partut, english, estonian, finnish-ftb, finnish, french-partut, french-sequoia, french, galician-treegal, galician, german, gothic, greek, hebrew, hindi, hungarian, indonesian, irish, italian, japanese, kazakh, korean, latin-ittb, latin-proiel, latin, latvian, lithuanian, norwegian-bokmaal, norwegian-nynorsk, old_church_slavonic, persian, polish, portuguese-br, portuguese, romanian, russian-syntagrus, russian, sanskrit, serbian, slovak, slovenian-sst, slovenian, spanish-ancora, spanish, swedish-lines, swedish, tamil, turkish, ukrainian, urdu, uyghur, vietnamese
- https://universaldependencies.org/introduction.html
- https://ufal.mff.cuni.cz/udpipe/users-manual



DATA SCIENCE EASY LAB - WORDCLOUDS WITH PACKAGE TM

Document-Term and Term-Document Matrices

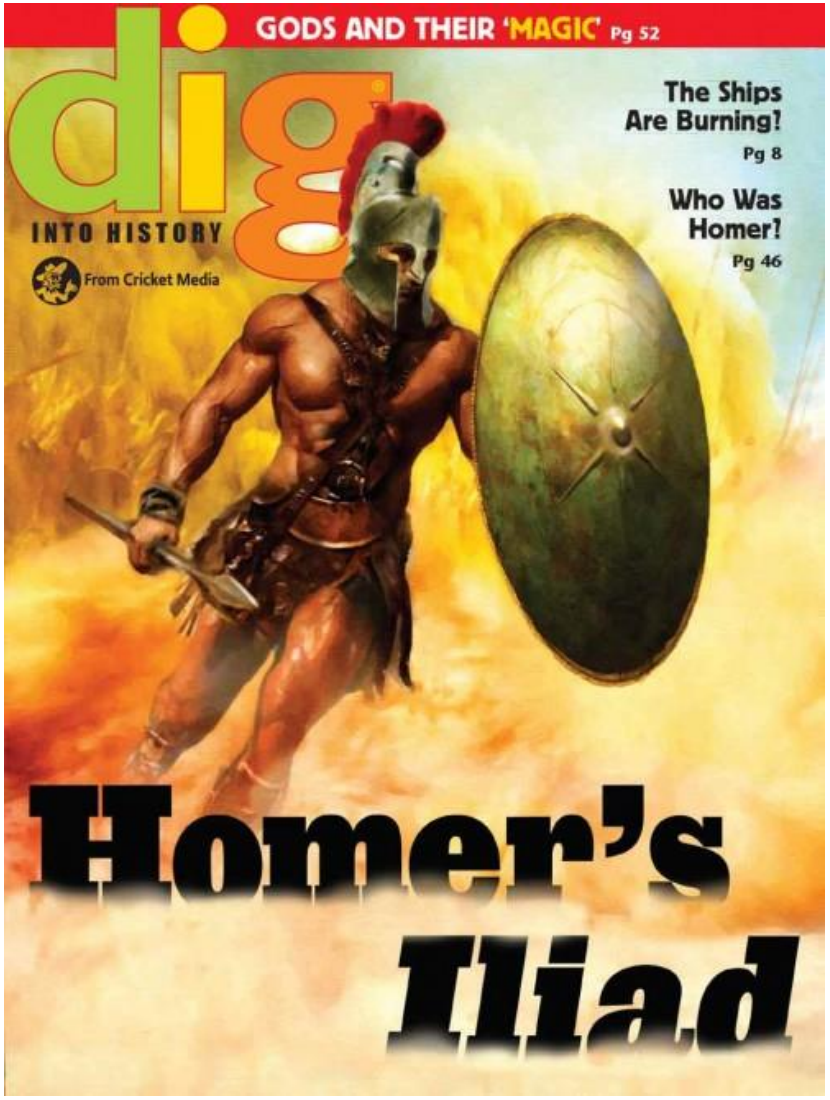
- A **document-term matrix** or **term-document matrix** is a mathematical matrix that describes the frequency of terms that occur in a collection of documents
- In a **document-term matrix**, rows correspond to documents in the collection and columns correspond to terms
- In a **term-document matrix**, rows correspond to terms, and columns to documents
- There are various schemes for determining the value that each cell in the matrix should take
  - In our lab and for now, we are just going to count occurrences

25

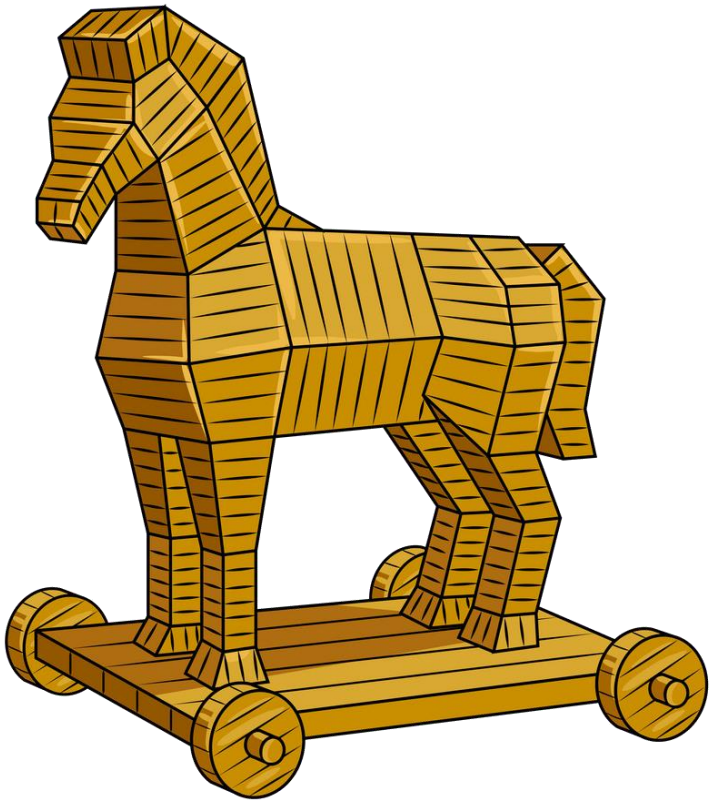
tf-idf

- **Stands for *Term Frequency – Inverse Document Frequency***
  - A numerical statistic that is intended to reflect how important a word is to a document in a collection of documents (also called a *corpus*)
- The tf-idf value:
  - Increases *proportionally* to the number of times a word appears in the document
  - *Offset* by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general
- So, if "**Troy**" appears very often in the document "Iliad", then "**Troy**" is a good descriptor for the document
- ..But so is the word "**the**"..
- However, the word "**the**" appears in ***all*** documents a lot
- And so its tf-idf should be low, while that of "**Troy**" high..
- Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query

Homer's iliad



- A War in antiquity
  - Athens vs Troy
  - And lessons in humanity



27

Odyssey

- **Ulysses' return trip to Ithaca**

28

R packages for text mining

- **Our main text mining package:**
  - `tm` **(text mining)**
  - **https://cran.r-project.org/web/packages/tm/index.html**
- **Other packages we are going to use:**
  - `SnowballCC, RColorBrewer, ggplot2, wordcloud, biclust, cluster, igraph, fpc`
- **Package references on CRAN:**
  - **https://cran.r-project.org/web/views/NaturalLanguageProcessing.html**

29

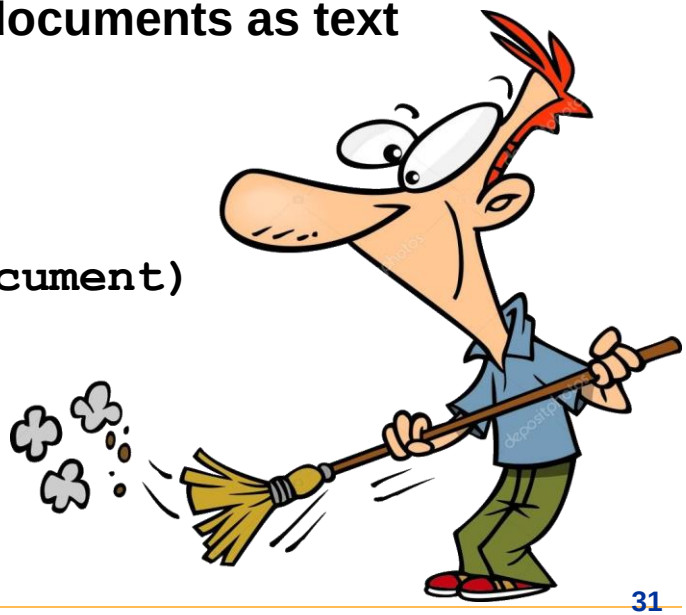
Loading texts

- Start by saving your text files in a folder titled `texts` This will be the "corpus" (body) of texts you will be mining
  - You are going to work with *Homer*'s *Iliad* and the *Odyssey*, some of the oldest literature about Wars
  - You are going to also to work in your own language, so get started in English for now, but plan ahead..
- Download corpus from blackboard
- Mac:
  - `cname <- file.path("~", "Desktop", "texts")`
  - `cname`
- Windows:
  - `cname <- file.path("C:", "texts")`
  - `cname`
- Read your documents in the R terminal with `inspect(docs)`
- if you prefer to look at only one of the documents you loaded, then you can specify `inspect(docs[1])`

30

Preprocessing

- **Remove numbers, capitalization, common words, punctuation, and prepare your texts for analysis**
  - **Remove punctuation, numbers, stopwords, common word endings, strip whitespace, convert to lower case, and combine words that should stay together**
- **Be sure to use the following script once you have completed preprocessing**
  - **This tells R to treat your preprocessed documents as text documents**
  - `library(tm)`
  - `docs <- Corpus(DirSource(cname))`
  - `docs <- tm_map(docs, PlainTextDocument)`

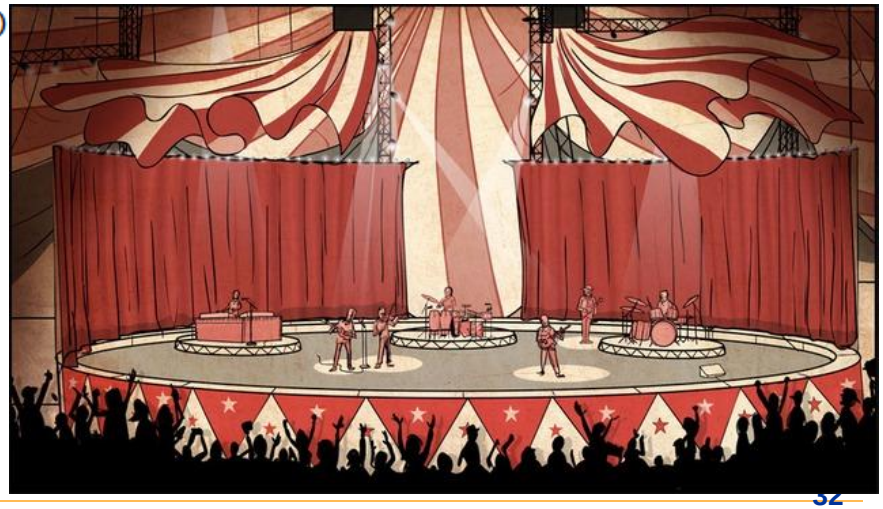


31

Data staging

- **Create a document term matrix and a term document matrix**

```
> dtm <- DocumentTermMatrix(docs)
> dtm
<<DocumentTermMatrix (documents: 2, terms: 11050)>>
Non-/sparse entries: 14535/7565
Sparsity           : 34%
Maximal term length: 35
Weighting          : term frequency (tf)
> tdm <- TermDocumentMatrix(docs)
> tdm
<<TermDocumentMatrix (terms: 11050, documents: 2)>>
Non-/sparse entries: 14535/7565
Sparsity           : 34%
Maximal term length: 35
Weighting          : term frequency (tf)
```



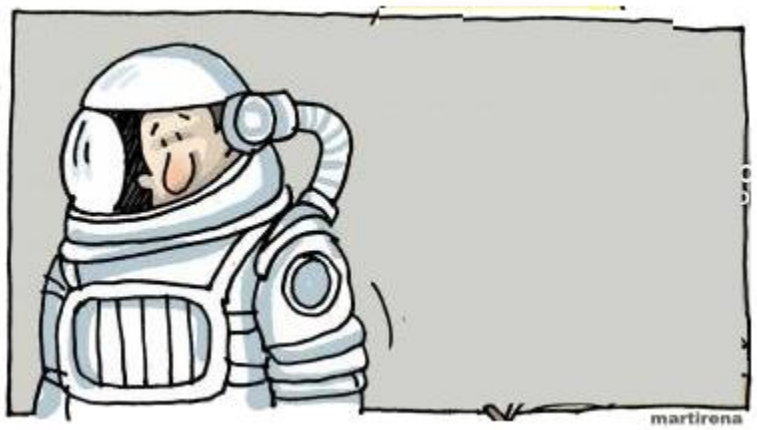
32

Data Exploration

- Organize terms by their frequency
  - `freq <- colSums(as.matrix(dtm))`
  - `length(freq)`
  - `ord <- order(freq)`
- Remove sparse terms
  - `dtms <- removeSparseTerms(dtm, 0.1)` **# This makes a matrix that is 10% empty space, maximum**
  - `inspect(dtms)`

```
                Terms
Docs             god now one said shall son thi thus ulyss will
  content        626 573 348  211   511 420 943  621   123  188
  meta           267 303 573  483   217 336   3  132   646  745
```

- *Least* frequently occurring words
  - `freq[head(ord)]`
- *Most* frequently occurring words
  - `freq[head(ord)]`



Frequency of frequencies

- Output is two rows of numbers, one per document
  - The top number is the frequency with which words appear
  - The bottom number reflects how many words appear that frequently
  - How many terms appear only once?
- Lowest word frequences:
  - `head(table(freq), 20)`
  - considering only the 20 *lowest* word frequencies, we can see that ??? terms appear only once
- Highest word frequencies:
  - `tail(table(freq), 20)`
  - Considering only the 20 greatest frequencies, there's usually a big disparity in how frequently some terms appear

34

Coarse-grained term frequencies

- For a less fine-grained look at term frequency, view a table of sparse terms:
  - `freq <- colSums(as.matrix(dtms))`
  - `freq`

```
> freq <- colSums(as.matrix(dtms))
> freq
     abandon         abat        abhor         abid          abl         abod       abound
           4            2            3            5           42           50            7
      abridg       abroad       absenc       absent      absolut       absurd         abus
           2            8           31           12            3           10            5
      accept       access     accident     accommod     accompani    accomplish       accord
          32           27            3            2           12           14           40
     account        accur      accustom          ach       achill      acquaint      acquiesc
```

- Alternatively:
  - `freq <- sort(colSums(as.matrix(dtm)), decreasing=TRUE)`
  - `head(freq, 40)`

```
head(freq, 40)
 thi    will     one     god     now   ulyss     son    thus   shall    said   great    hand     arm    jove
 946     933     921     893     876     769     756     753     728     694     686     653     625     617
 man  heaven     day     may     men     let     see    come  hector     can    ship    like     war  achill
 595     567     534     513     499     496     489     469     461     453     450     447     440     436
hous    troy     yet   chief   first  father   greek     eye    upon  trojan     oer    make
 429     425     420     417     414     408     395     391     391     368     364     363
```


35

Coarse-grained term frequencies

- **All terms that appear frequently (250 or more times):**
- **`findFreqTerms(dtm, lowfreq=250)`**

```
> findFreqTerms(dtm,lowfreq=250)
 [1] "achill"    "arm"       "back"      "came"      "can"       "care"      "chief"
 [8] "come"      "day"       "dead"      "death"     "even"      "everi"     "eye"
[15] "fall"      "fate"      "father"    "field"     "fight"     "fire"      "first"
[22] "fli"       "forc"      "friend"    "give"      "god"       "good"      "great"
[29] "greek"     "ground"    "hand"      "head"      "heart"     "heaven"    "hector"
[36] "hero"      "high"      "home"      "hous"      "jove"      "king"      "know"
[43] "let"       "lie"       "like"      "long"      "made"      "make"      "man"
[50] "may"       "men"       "much"      "must"      "now"       "oer"       "old"
[57] "one"       "place"     "plain"     "power"     "rage"      "round"     "said"
[64] "say"       "sea"       "see"       "shall"     "ship"      "son"       "stand"
[71] "still"     "suitor"    "take"      "telemachus" "tell"      "thi"       "thou"
[78] "though"    "thus"      "till"      "time"      "trojan"    "troy"      "two"
[85] "ulyss"     "upon"      "vain"      "war"       "way"       "went"      "whose"
[92] "will"      "word"      "work"      "yet"
```

Alternatively:

- **`wf <- data.frame(word=names(freq), freq=freq)`**
- **`head(wf)`**

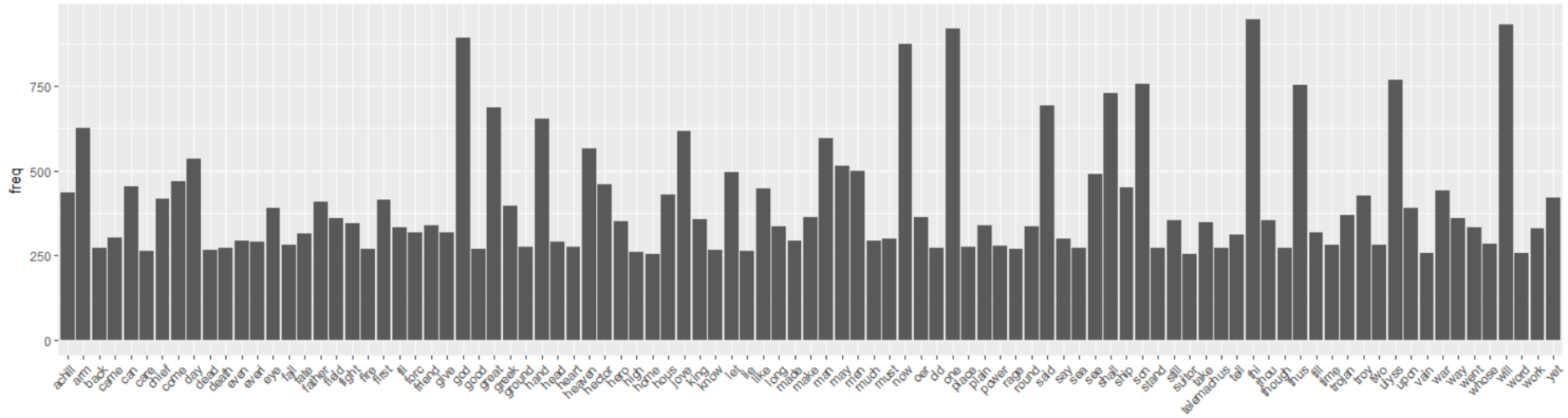
```
> head(wf)
       word freq
thi     thi  946
will   will  933
one     one  921
god     god  893
now     now  876
ulyss ulyss  769
```


36

Plotting with ggplot2

- `library(ggplot2)`
- ```
  p <- ggplot(subset(wf, freq>50), aes(word, freq))
  p <- p + geom_bar(stat="identity")
  p <- p + theme(axis.text.x=element_text(angle=45, hjust
  =1))
  p
  ```
  ```

Fixing potential ggplot2 error

- `library(ggplot2)`
  - **Error in loadNamespace(i, c(lib.loc, .libPaths()), versionCheck = vI[[i]]) : namespace 'scales' 0.4.0 is being loaded, but >= 0.4.1 is required**
    Error: package or namespace load failed for 'ggplot2'
- **Then look at the complete error message. It lists a couple of packages which are missing or have errors. Uninstall and reinstall them. So for me:**
  - ```
    remove.packages("ggplot2")
    install.packages('ggplot2', dependencies = TRUE)
    remove.packages("scales")
    install.packages('scales', dependencies = TRUE)
    library(ggplot2)
    ```
    ```

Words with frequencies > 250 in iliad + odyssey



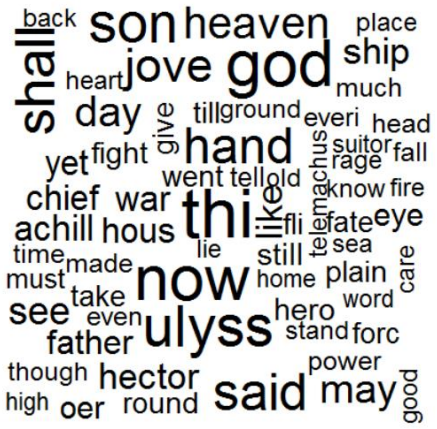
39

Relationships between terms

- If you have a term in mind that you have found to be particularly meaningful to your analysis, then you may find it helpful to identify the words that most highly correlate with that term
  - If words *always* appear together, then correlation=1
  - `library(tm)`
  - `findAssocs(dtm, "achilles", corlimit=0.98) # specifying a correlation limit of 0.98`
- Can also find associations for word collections:
  - `findAssocs(dtm, c("achilles", "hector"), corlimit=0.98)`

Word Clouds

- **Humans are generally strong at *visual analytics***
  - **We are going to construct word clouds on our document corpus**
- **Load the package that makes word clouds in R**
  - `library(wordcloud)`
- **Plot words that occur *at least 250 times***
  - `set.seed(142)  #The "set.seed() function just makes the configuration of the layout of the clouds consistent each time you plot them. You can omit that part if you are not concerned with preserving a particular layout`
  - `wordcloud(names(freq), freq, min.freq=250)`



41

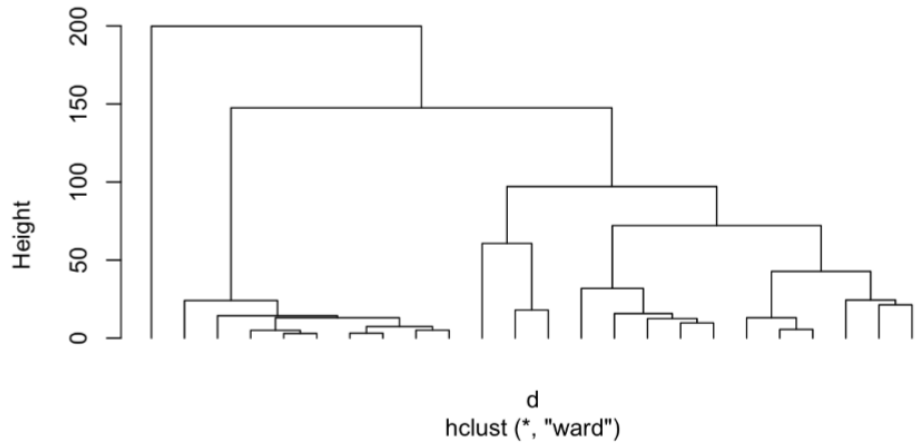
..With Color

- **set.seed(142)**
  dark2 <- brewer.pal(6, "Dark2")
  wordcloud(names(freq), freq, max.words=250, rot.per=0.2, c
  olors=dark2)

42

Clustering by term similarity

- First calculate distance between words & then cluster them according to similarity:
  - `library(cluster)`
  - `dtmss <- removeSparseTerms(dtm, 0.15) # This makes a matrix that is only 15% empty space, maximum`
  - `inspect(dtmss)`
  - `d <- dist(t(dtmss),method="euclidian")`
  - `fit <- hclust(d=d, method="ward.D2")`
  - `plot(fit, hang=-1)`



43

Ask R to help identify the clusters

- `plot.new()`
- `plot(fit, hang=-1)`
- `groups <- cutree(fit, k=5)   # "k=" defines the number of clusters we are using`
- `rect.hclust(fit, k=5, border="red") # draw dendogram with red borders around the 5 clusters`

44

K-Means clustering

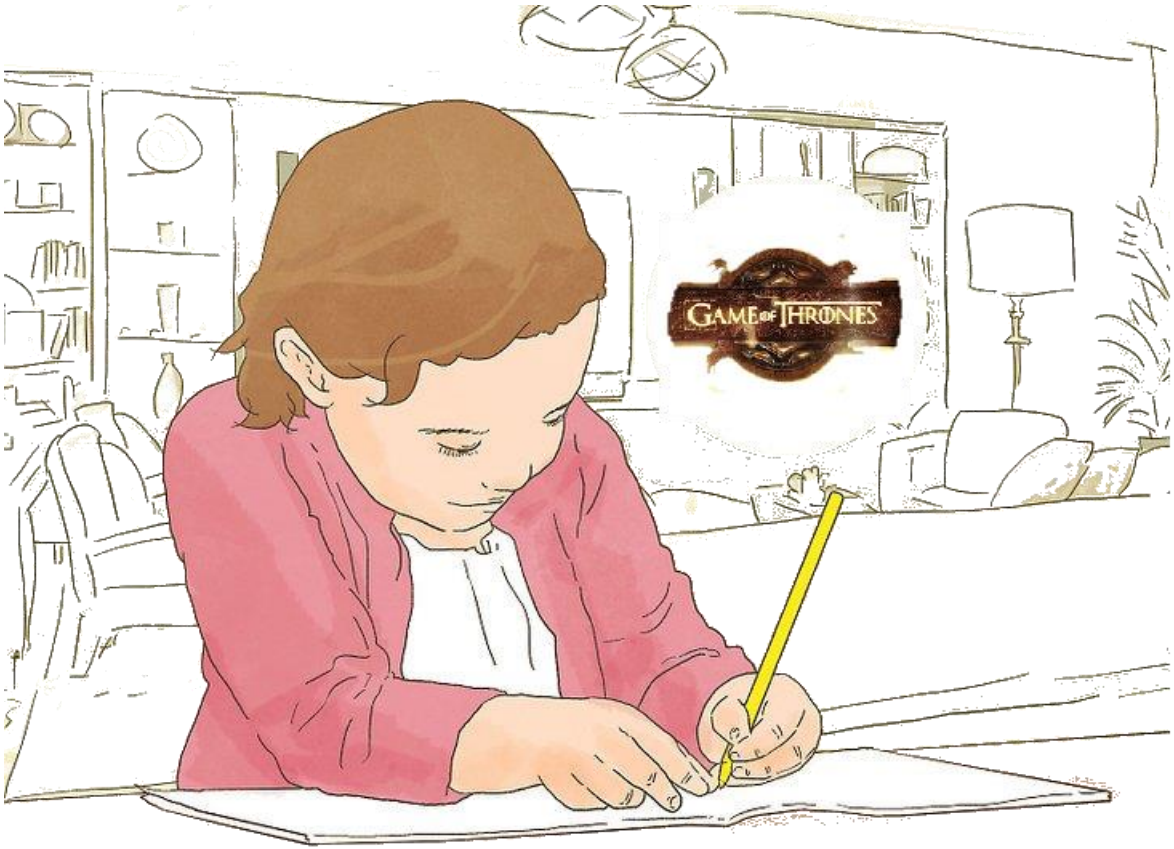
- **Cluster words into a specified number of groups (in this case 2), such that the sum of squared distances between individual words and one of the group centers is minimized**
  - `library(fpc)`
  - `d <- dist(t(dtmss), method="euclidian")`
  - `kfit <- kmeans(d, 2)`
  - `clusplot(as.matrix(d), kfit$cluster, color=T, shade=T, labels=2, lines=0)`

45

Errata

- **In your wordcloud R file, the last line below *may* error out, if that is the case make sure it is commented out (as below)**

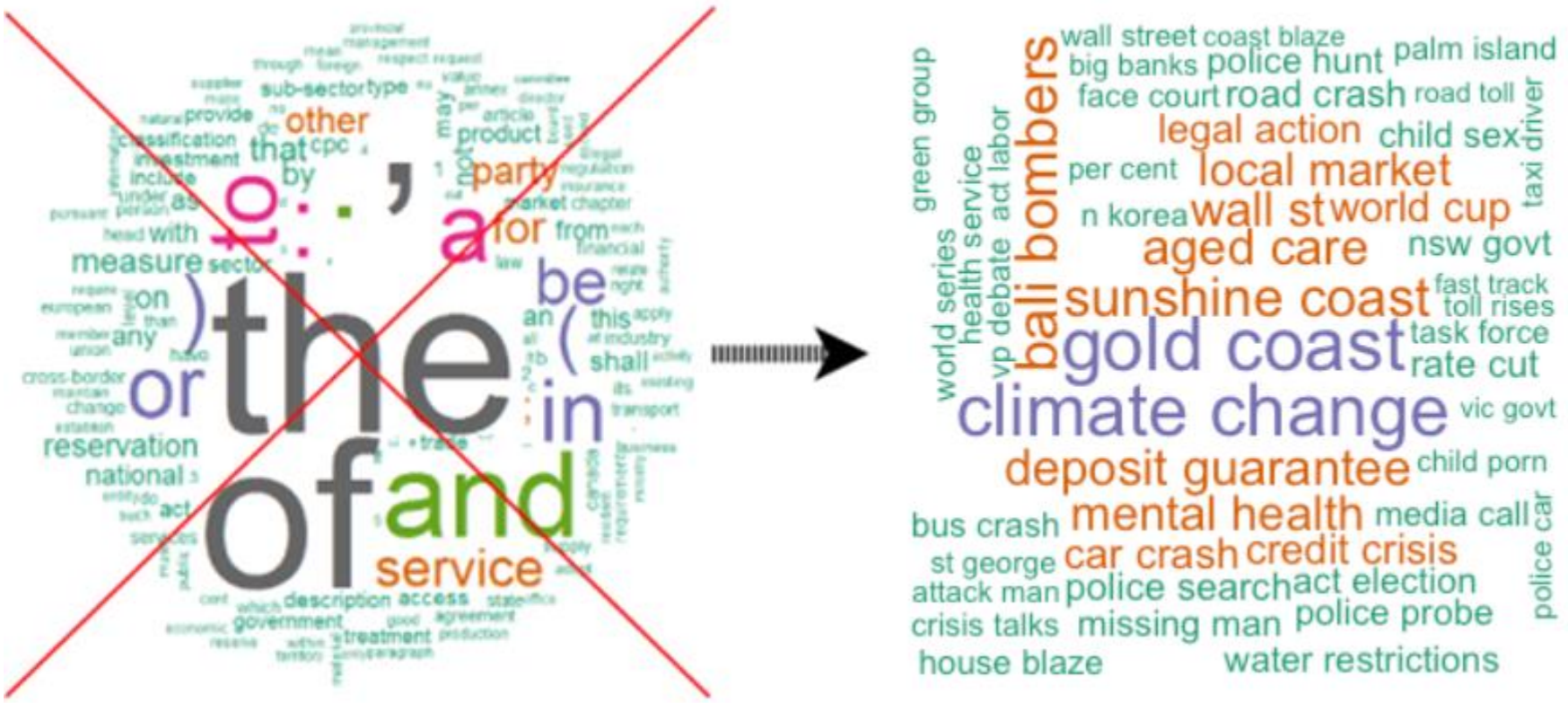
```
## Don't do this :-)
## To Finish
## Be sure to use the following script once you have completed preprocessing.
## This tells R to treat your preprocessed documents as text documents.
##docs <- tm_map(docs, PlainTextDocument)
```



Part 4

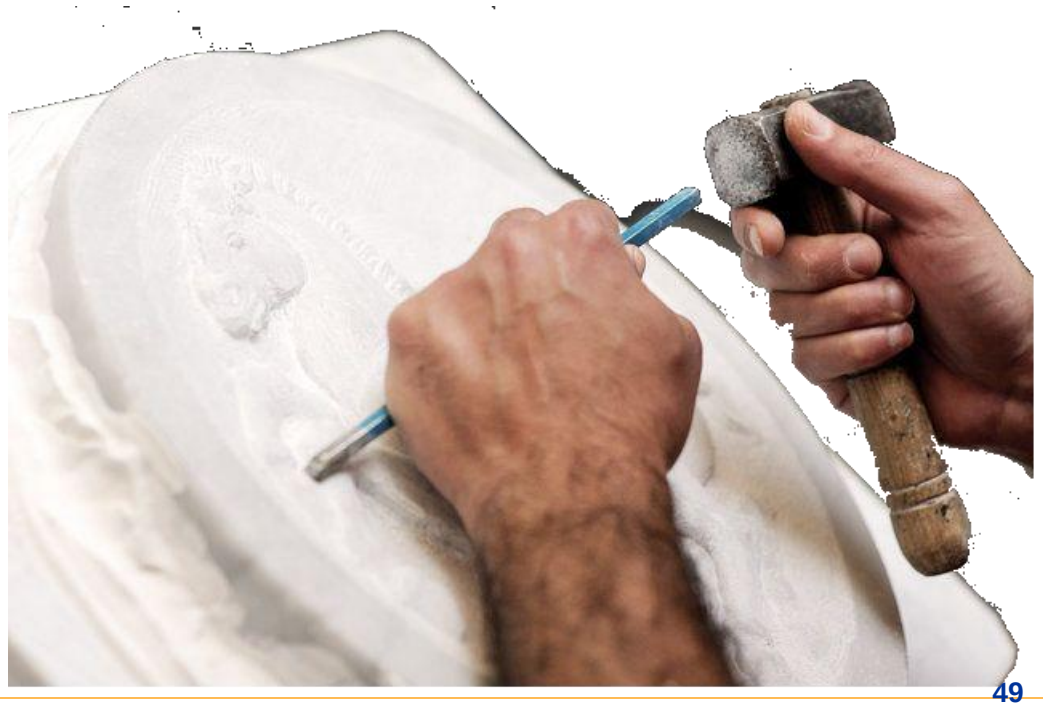
DATA SCIENCE ADVANCED LAB – WORDCLOUDS WITH PACKAGE UDPIPE

Better word clouds with better data science



Versions

- **This lab may require the *latest* versions and research..**
  - **You may have to download the latest R runtime and RStudio by hand..**
  - ***Not sure*, but keep that in mind as a possibility**



49

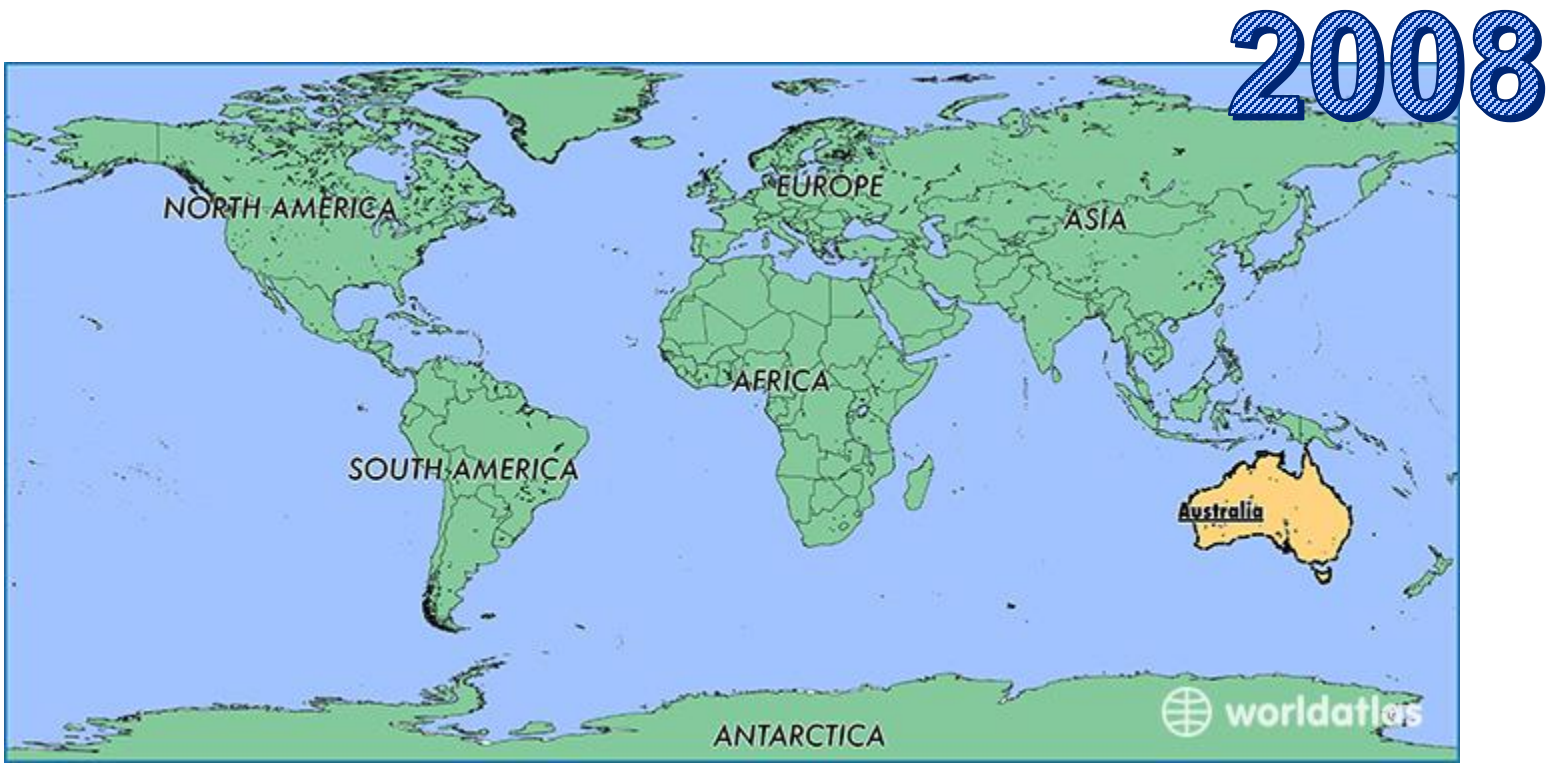
Goal

- **You are going to do NLP..**
  - **in *English, with English dataset***
  - ***Then, you can try with <u>your own</u> dataset in <u>your</u> language***

50

Ok, so what's the lab?

- **A friend told you he dreamt about the years 2008 and Australia, and he knows you are a Data scientist, so he asks you to tell him what happened in 2008 that was noteworthy, in Australia..**



51

What is the first thing a data scientist does?

- Go look for data!
- You locate some news headlines published over a period of 15 years..
  - From Australian news source ABC (Australian Broadcasting Corp.)
  - At: http://www.abc.net.au/
- The Dataset is also available on Kaggle:
  - **https://www.kaggle.com/therohk/million-headlines**
- Better use the version I put up on canvas, instead (more data)

52

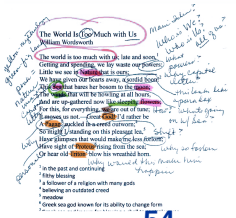
Package `udpipe`

- Then you go looking for data science libraries
  - You're still learning Python in class, so…
  - You go looking for R libraries on CRAN..
- You find.. `udpipe` provides *pretrained* language models for spoken languages (not programming languages) and you can download the required model using `udpipe_download_model()`

53

udpipe

- **Udpipe** provides language-agnostic ‘tokenization’ and ‘parts of speech tagging’, of raw text in many languages, including Chinese and Hindi.
- `library(udpipe)`
- `model <- udpipe_download_model(language = "english")`
- `# When you download the language, you will see the associated filename download from GitHub, pass that filename in the next command below..`
- `udmodel_english <- udpipe_load_model(file = 'english-ewt-ud-2.4-190531.udpipe')`
- `#Now annotate your corpus or sentence (or haiku)`
- `s <- udpipe_annotate(udmodel_english, "An old silent pond... A frog jumps into the pond, splash! Silence again.")`
- `x <- data.frame(s)`
- `colnames(x)`



54

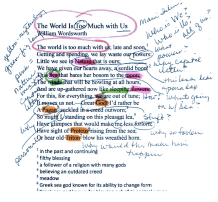
Annotating (continued)

```
> colnames(x)
 [1] "doc_id"        "paragraph_id"  "sentence_id"
 [4] "sentence"      "token_id"      "token"
 [7] "lemma"         "upos"          "xpos"
[10] "feats"         "head_token_id" "dep_rel"
[13] "deps"          "misc"
```

```
> x$token
 [1] "An"      "old"     "silent"  "pond"    "..."     "A"
 [7] "frog"    "jumps"   "into"    "the"     "pond"    ","
[13] "splash"  "!"       "Silence" "again"   "."
```

- **And your Universal Parts of Speech (UPOS):**

```
> x$upos
 [1] "DET"   "ADJ"   "ADJ"   "NOUN"  "PUNCT" "DET"   "NOUN"
 [8] "VERB"  "ADP"   "DET"   "NOUN"  "PUNCT" "NOUN"  "PUNCT"
[15] "ADV"   "ADV"   "PUNCT"
```

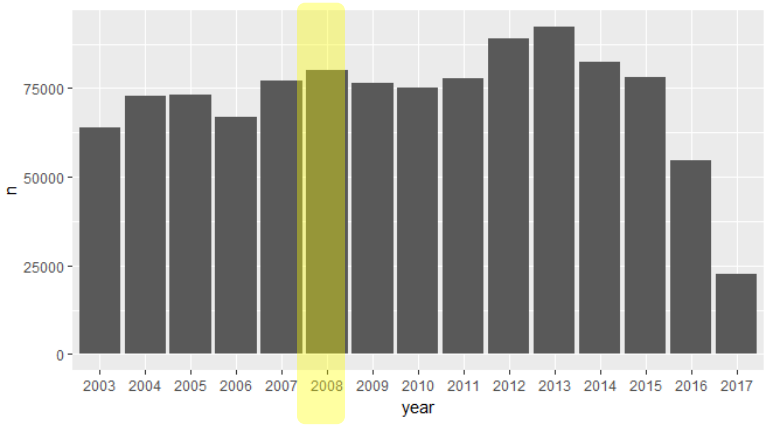


Getting part of speech

- `verbs <- subset(x, upos %in% c("VERB"))`
- `stats$token`

Cleaning dataset

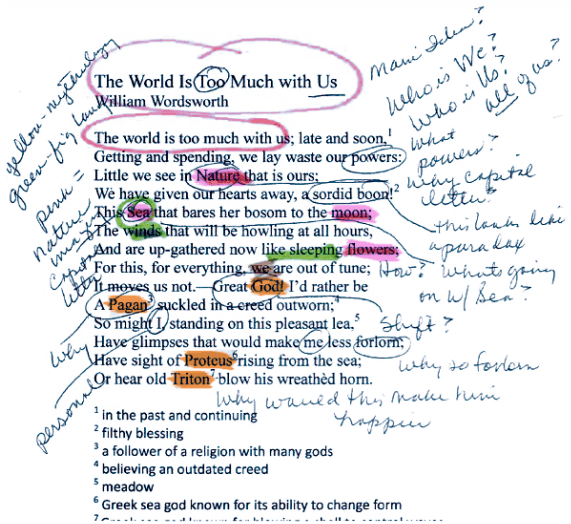
- **Read in the data**
  - `news <- read.csv('abcnews-date-text.csv', header = T, stringsAsFactors = F)`
- **Do a quick exploration:**
  - `news %>% group_by(publish_date) %>% count() %>% arrange(desc(n))`
  - What is `%>%` ? Go to page 37 for an explanation
- **To subset data only for 2008:**
  - `news_more_2008 <- news_more %>% filter(year == 2008 & month == 10)`



57

Staging with pretrained language models

```
library(udpipe)
model <- udpipe_download_model(language = "english")
udmodel_english <- udpipe_load_model(file = 'english-ewt-ud-2.4-190531.udpipe')
s <- udpipe_annotate(udmodel_english, news_more_2008$headline_text)
x <- data.frame(s)
```



58

Package `lattice`

- Powerful and elegant high-level data visualization system inspired by Trellis graphics, with an emphasis on multivariate data
  - Sufficient for typical graphics needs, and flexible enough to handle most nonstandard requirements
  - Nice barcharts ☺
  - **`barchart(key ~ freq, data = head(stats, 20), col = "magenta", main = "Keywords-simple noun phrases", xlab = "Frequency")`**
  - **https://cran.r-project.org/web/packages/lattice/index.html**

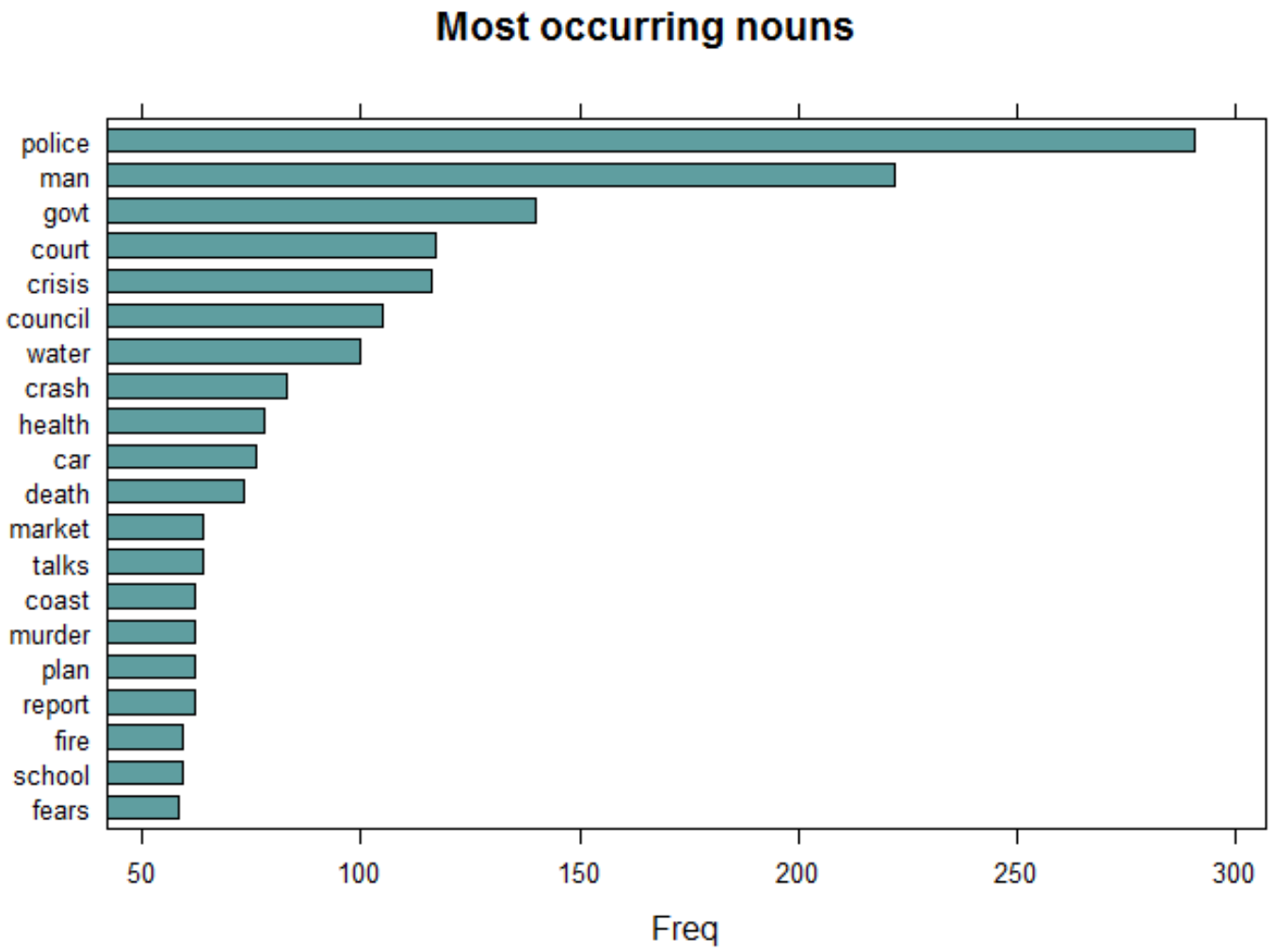
Factors in R

- Variables that take on a *limited* number of different values
  - Often referred to as *categorical* variables
- Very important in statistical modeling
  - Categorical variables enter into statistical models differently than continuous variables..
- Factors are also a very efficient way to store character values
  - Because each unique character value is stored only once, and the data itself is stored as a vector of integers

```
data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)
fdata = factor(data)
fdata
[1] 1 2 2 3 1 2 3 3 1 2 3 3 1
Levels: 1 2 3
```


60

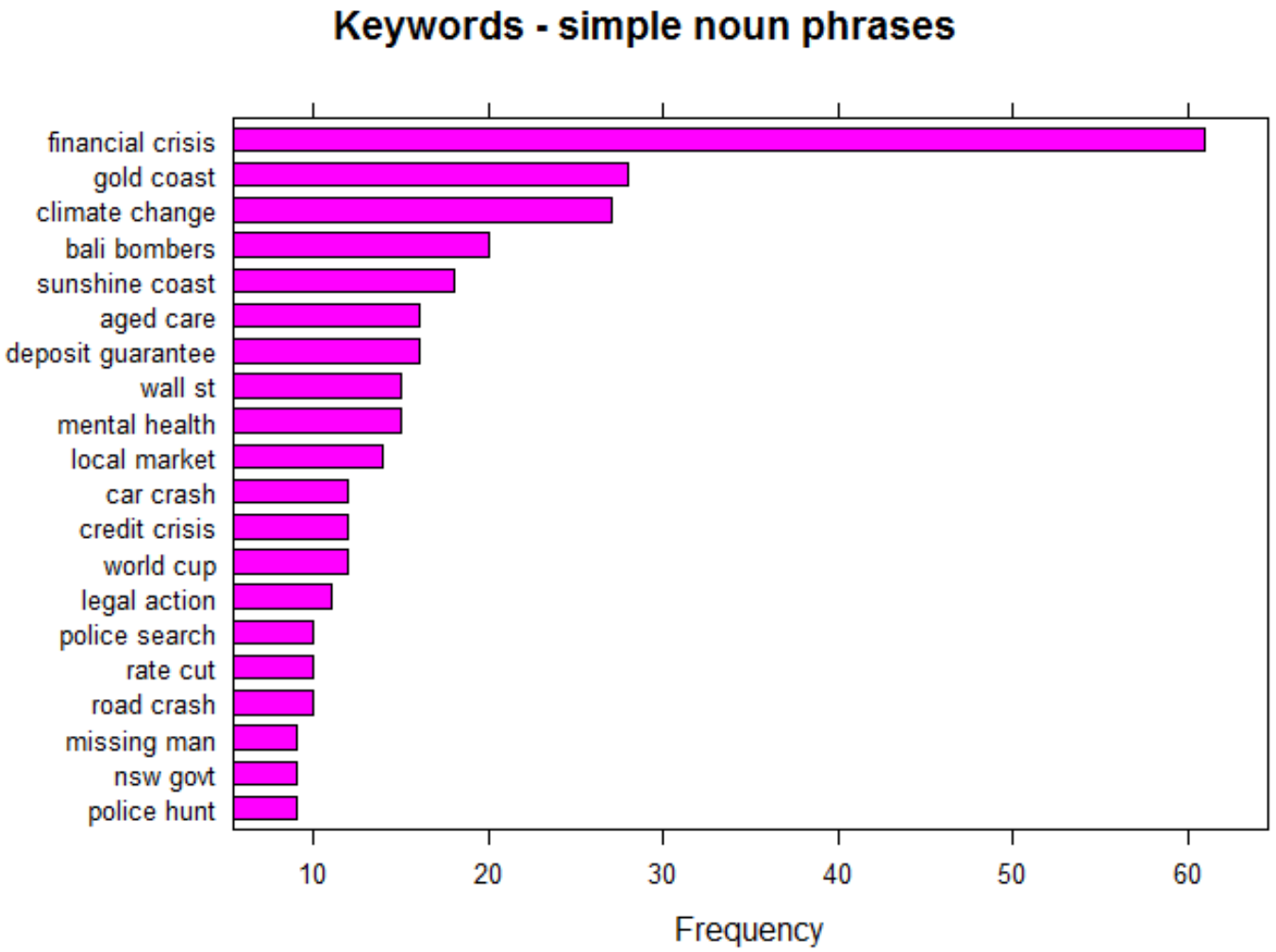
Most occurring nouns for 2008



RAKE/RKEA

- One you get good at this, you realize scientists wrote a package that already does what you're doing..
- Rapid Keyword Extraction Algorithm (RKEA) or Rapid Algorithm for Keyword Extraction (RAKE)?

Simple noun phrases



And you give your friend his word cloud..

```
library(wordcloud)
wordcloud(words = stats$key, freq = stats$freq,
min.freq = 3, max.words = 100, random.order =
FALSE, colors = brewer.pal(6, "Dark2"))
```

Important note: Versioning

- You may have to modify the name of your model to take versioning into account
- What works on my version of RStudio may not work on yours
- The latest version for hindi annotations as of this writing is:
  - `udmodel_hindi <- udpipe_load_model(file='hindi-hdtb-ud-2.4-190531.udpipe')`
- Check for language versions at:
  - https://cran.r-project.org/web/packages/udpipe/vignettes/udpipe-annotation.html

अ a	आ aa	इ i	ई ii	उ u	ऊ uu	ए e	ऐ ai	ओ o	औ au	ऋ R

क k	ख K	ग g	घ G	ङ q	च c	छ C	ज j	झ J	ञ z	ट tw	ठ Tw

ड dw	ढ Dw	ण nw	त t	थ T	द d	ध D	न n	प p	फ P	ब b	भ B	म m

य y	र r	ल l	व v	श x	ष sw	स s	ह h

Hints

LANGUAGE ENCODINGS

tm

- R text mining library
- `install.packages("tm")`
- `library(tm)`

RStudio locale

- `Sys.setlocale(category="LC_ALL", locale="chinese")`
- `Sys.setlocale(category="LC_ALL", locale="hindi")`

Load hindi text & remove punctuation

- Assume `hindi.txt` contains Unicode for poem in hindi
  - Contained in RStudio Session folder
- `h <- Corpus(VectorSource(readLines("hindi.txt", n=1, encoding="UTF-8")))`

```
> inspect(h)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] साजन!होलीआईहै!,सुखसेहँसना,जीभरगाना,मस्तीसेमनकोबहलाना,पर्वहोगयाआज-,साजन!होलीआ
ईहै!,हँसानेहमकोआईहै!
```

```
> h <- tm_map(h, removePunctuation)
> inspect(h)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] साजनहोलीआई हैसुखसेहँसनाजीभरगानामस्तीसेमनकोबहलानापर्वहोगयाआजसाजनहोलीआई हैहँसानेहम
कोआई है
```

Works?

- Hmm..
- Maybe try different encoding?
  - How about `"UCS-2LE"`?
- R encoding is somewhat of a pain
  - http://kevinushey.github.io/blog/2018/02/21/string-encoding-and-r/

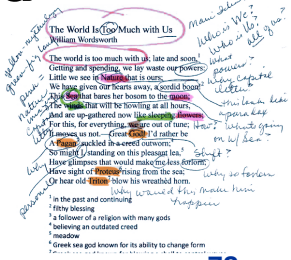
Removing stopwords from hindi frame

```
> inspect(h)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] साजनहोलीआई हैसुखसेहँसनाजीभरगानामस्तीसेमनकोबहलानापर्वहोगयाआजसाजनहोलीआई हैहँसानेहम
कोआई है

> h <- tm_map(h, removeWords, c("साजनहोलीआई", "email"))
> inspect(h)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1]  हैसुखसेहँसनाजीभरगानामस्तीसेमनकोबहलानापर्वहोगयाआजसाजनहोलीआई हैहँसानेहमकोआई है
>
```

udpipe

- **Udpipe** provides language-agnostic ‘tokenization’ and ‘parts of speech tagging’, of raw text in many languages, including Chinese and Hindi.

```
library(udpipe)
model <- udpipe_download_model(language = "english")
# When you download the language, you will see the associated filename download from GitHub, pass that filename in the next command below..
udmodel_english <- udpipe_load_model(file = 'english-ewt-ud-2.4-190531.udpipe')
#Now annotate your corpus or sentence (or haiku)
s <- udpipe_annotate(udmodel_english, "An old silent pond... A frog jumps into the pond, splash! Silence again.")
x <- data.frame(s)
colnames(x)
```



72

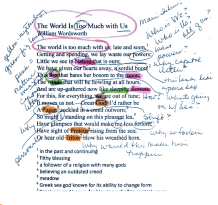
Annotating (continued)

```
> colnames(x)
 [1] "doc_id"       "paragraph_id"  "sentence_id"
 [4] "sentence"     "token_id"      "token"
 [7] "lemma"        "upos"          "xpos"
[10] "feats"        "head_token_id" "dep_rel"
[13] "deps"         "misc"
```

```
> x$token
 [1] "An"     "old"    "silent"  "pond"    "..."     "A"
 [7] "frog"   "jumps"  "into"    "the"     "pond"    ","
[13] "splash" "!"      "Silence" "again"   "."
```

- **And your Universal Parts of Speech (UPOS):**

```
> x$upos
 [1] "DET"   "ADJ"   "ADJ"   "NOUN"  "PUNCT" "DET"   "NOUN"
 [8] "VERB"  "ADP"   "DET"   "NOUN"  "PUNCT" "NOUN"  "PUNCT"
[15] "ADV"   "ADV"   "PUNCT"
```



Getting parts of speech (PoS): Verbs

- `verbs <- subset(x, upos %in% c("VERB"))`
- `stats$token`

And now..

- **You can do a much better text analysis since you know about tokens *and their roles (grammar)* in the text..**

Unicode package

- `install.packages("utf8")`
- `library(utf8)`

Example: Greek

- `library(udpipe)`
- `udmodel <- udpipe_download_model(language = "greek")`
- `udmodel_greek <- udpipe_load_model(file = 'greek-gdt-ud-2.4-190531.udpipe')`
- `s <- udpipe_annotate(udmodel_greek,` "Πενθώ τόν ήλιο καί πενθώ τα χρόνια που έρχονται. Χωρίς εμάς καί τραγουδώ τ' άλλα πού πέρασαν. Εάν είναι αλήθεια. Μιλημένα τα σώματα καί οί βάρκες πού έκρουζαν γλυκά.. Οί κιθάρες πού αναβόσβησαν κάτω από τα νερά")
- `x <- data.frame(s)`
- `colnames(x)`
- `utf8_print(unlist(x$token))`
- `x$upos`
- `verbs <- subset(x, upos %in% c("VERB"))`
- `utf8_print(unlist(verbs$token))`

Example: Hindi


```
library(udpipe)
udmodel <- udpipe_download_model(language = "hindi")
udmodel_hindi <- udpipe_load_model(file = 'hindi-hdtb-ud-2.4-190531.udpipe')
s <- udpipe_annotate(udmodel_hindi, "जंगल में मोर नाचा किस ने देखा?")
x <- data.frame(s)
colnames(x)
utf8_print(unlist(x$token))
x$upos
verbs <- subset(x, upos %in% c("VERB"))
utf8_print(unlist(verbs$token))
```

Example: Chinese

- `udmodel <- udpipe_download_model(language = "chinese")`
- `udmodel_zhongwen <- udpipe_load_model(file = 'chinese-gsd-ud-2.4-190531.udpipe')`
- `s <- udpipe_annotate(udmodel_zhongwen, "授人以鱼不如授人以渔")`
- `x <- data.frame(s)`
- `colnames(x)`
- `utf8_print(unlist(x$token))`
- `x$upos`
- `verbs <- subset(x, upos %in% c("VERB"))`
- `utf8_print(unlist(verbs$token))`

Important note: Versioning

- You may have to modify the name of your model to take versioning into account
- What works on my version of RStudio may not work on yours
- The latest version for hindi annotations as of this writing is:
  - `udmodel_hindi <- udpipe_load_model(file='hindi-hdtb-ud-2.4-190531.udpipe')`
- Check for language versions at:
  - https://cran.r-project.org/web/packages/udpipe/vignettes/udpipe-annotation.html

Example: Dutch with new model

```
library(udpipe)
dl <- udpipe_download_model(language = "dutch")
## Either give a file in the current working directory
udmodel_dutch <- udpipe_load_model(file = "dutch-alpino-ud-2.4-190531.udpipe")
## Or give the full path to the file
udmodel_dutch <- udpipe_load_model(file = dl$file_model)
txt <- c("Ik ben de weg kwijt, kunt u me zeggen waar de Lange Wapper ligt? Jazeker meneer",
         "Het gaat vooruit, het gaat verbazend goed vooruit")
x <- udpipe_annotate(udmodel_dutch, x = txt)
x <- as.data.frame(x)
str(x)
table(x$upos)
```

Only part of the annotation

```
## Tokenization + finds sentences, does not execute POS tagging, nor lemmatization or dependency parsing
x <- udpipe_annotate(udmodel_dutch, x = txt, tagger = "none", parser = "none")
x <- as.data.frame(x)
table(x$upos)
table(x$dep_rel)

## Tokenization + finds sentences, does POS tagging and lemmatization but does not execute dependency parsing
x <- udpipe_annotate(udmodel_dutch, x = txt, tagger = "default", parser = "none")
x <- as.data.frame(x)
table(x$upos)
table(x$dep_rel)

## Tokenization + finds sentences and executes dependency parsing but does not do POS tagging nor lemmatization
x <- udpipe_annotate(udmodel_dutch, x = txt, tagger = "none", parser = "default")
x <- as.data.frame(x)
table(x$upos)
table(x$dep_rel)
```

My text data is already tokenized

```
## Either put every token on a new line and use tokenizer: vertical
input <- list(doc1 = c("Ik", "ben", "de", "weg", "kwijt", ",", "kunt", "u", "me", "zeggen",
                       "waar", "de", "Lange Wapper", "ligt", "?", "Jazeker", "meneer"),
              doc2 = c("Het", "gaat", "vooruit", ",", "het", "gaat", "verbazend", "goed", "vooruit"))
txt <- sapply(input, FUN=function(x) paste(x, collapse = "\n"))
x <- udpipe_annotate(udmodel_dutch, x = txt, tokenizer = "vertical")
x <- as.data.frame(x)

## Or put every token of each document in 1 string separated by a space and use tokenizer: horizontal
##   Mark that if a token contains a space, you need to replace the space
##   with the 'NO-BREAK SPACE' (U+00A0) character to make sure it is still considered as one token
txt <- sapply(input, FUN=function(x){
  x <- gsub(" ", intToUtf8(160), x) ## replace space with no-break-space
  paste(x, collapse = " ")
})
x <- udpipe_annotate(udmodel_dutch, x = as.character(txt), tokenizer = "horizontal")
x <- as.data.frame(x)
```

Hindi


```
> model <- udpipe_download_model(language = "hindi")
Downloading udpipe model from https://raw.githubusercontent.com/jwijf
fels/udpipe.models.ud.2.0/master/inst/udpipe-ud-2.0-170801/hindi-ud-2
.0-170801.udpipe to D:/user/docs/NU/_Info6101/Lecture 2/labs/udpipe/m
odels/hindi-ud-2.0-170801.udpipe
trying URL 'https://raw.githubusercontent.com/jwijffels/udpipe.models
.ud.2.0/master/inst/udpipe-ud-2.0-170801/hindi-ud-2.0-170801.udpipe'
Content type 'application/octet-stream' length 26137581 bytes (24.9 M
B)
downloaded 24.9 MB

> model <- udpipe_load_model(file = "hindi-ud-2.0-170801.udpipe")
> x <- udpipe_annotate(model, " मैं तन्हा हूँ मुझे तन्हा ही रहने दो, देखकर मेरे बहते
 आंसू, तुम अपने लहू न बहने दो, मैं आपका दीवाना हूँ, मुझे बस अपना पागल रहने दो    "
)#hindi poem
> x <- data.frame(x)
>
```

Hindi uPOS

```
> x$token
 [1] "मैं"      "तन्हा"    "हूँ"      "मुझे"     "तन्हा"    "ही"       "रहने"
 [8] "दो"       ","        "देखकर"   "मेरे"     "बहते"     "आंसू"     ","
[15] "तुम"      "अपने"     "लहू"      "न"        "बहने"     "दो"       ","
[22] "मैं"      "आपका"    "दीवाना"  "हूँ"      ","        "मुझे"     "बस"
[29] "अपना"    "पागल"    "रहने"     "दो"

> x$upos
 [1] "PRON"  "VERB"  "AUX"   "PRON"  "NOUN"  "PART"  "VERB"
 [8] "NUM"   "PUNCT" "VERB"  "PRON"  "VERB"  "NOUN"  "PUNCT"
[15] "NOUN"  "PRON"  "ADV"   "PART"  "VERB"  "NUM"   "PUNCT"
[22] "PRON"  "PRON"  "ADJ"   "NOUN"  "PUNCT" "PRON"  "PART"
[29] "PRON"  "ADJ"   "VERB"  "NUM"
```

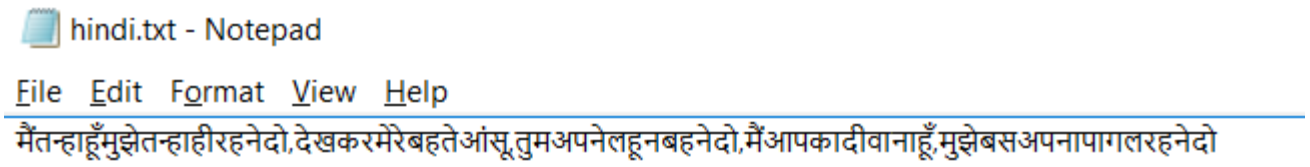
Printing Unicode to console

- `install.packages("utf8")`
- `library(utf8)`
- `utf8_print(unlist(x$token))`
- `#concatenating:`
  `paste( unlist(x$token), collapse='')`

```
> unlist(x$token)
 [1] "मैं"      "तन्हा"   "हूँ"      "मुझे"     "तन्हा"   "ही"      "रहने"
 [8] "दो"       ","       "देखकर"   "मेरे"     "बहते"    "आंसू"      ","
[15] "तुम"     "अपने"    "लहू"      "न"       "बहने"    "दो"      ","
[22] "मैं"     "आपका"    "दीवाना" "हूँ"       ","       "मुझे"    "बस"
[29] "अपना"   "पागल"    "रहने"    "दो"
> utf8_print(unlist(x$token))
 [1] "मैं"      "तन्हा"   "हूँ"      "मुझे"     "तन्हा"   "ही"      "रहने"
 [8] "दो"       ","       "देखकर"   "मेरे"     "बहते"    "आंसू"      ","
[15] "तुम"     "अपने"    "लहू"      "न"       "बहने"    "दो"      ","
[22] "मैं"     "आपका"    "दीवाना" "हूँ"       ","       "मुझे"    "बस"
[29] "अपना"   "पागल"    "रहने"    "दो"
> paste( unlist(x$token), collapse='')
[1] "मैंतन्हाहूँमुझेतन्हाहीरहनेदो,देखकरमेरेबहतेआंसू,तुमअपनेलहूनबहनेदो,मैंआपकादीवानाहूँ,मुझेबसअप
नापागलरहनेदो"
```

Printing Hindi Unicode to file

```
writeLines(text = paste( unlist(x$token),
collapse=''), con = "hindi.txt", useBytes = T)
```



Reading Hindi Unicode from file

- `hindi <- readLines(con <- file("hindi-poem.txt", encoding = "UCS-2LE"))`
  - Other option: `hindi <- readLines(con <- file("hindi-poem.txt", encoding = "UTF-16")) )`
- `close(con)`
- `unique(Encoding(hindi))`
- `x <- udpipe_annotate(model, hindi)`
- `x <- data.frame(x)`

```
> A <- readLines(con <- file("hindi-poem.txt", encoding = "UCS-2LE"))
> close(con)
> unique(Encoding(A))
[1] "UTF-8"
> A
[1] "मैं तन्हा हूँ मुझे तन्हा ही रहने दो, देखकर मेरे बहते आंसू, तुम अपने लहू न बहने दो, मैं
 आपका दीवाना हूँ, मुझे बस अपना पागल रहने दो"
> x <- udpipe_annotate(model, A)
> x <- data.frame(x)
> x$token
 [1] "मैं"      "तन्हा"   "हूँ"      "मुझे"     "तन्हा"   "ही"      "रहने"
 [8] "दो"       ","       "देखकर"   "मेरे"     "बहते"    "आंसू"      ","
[15] "तुम"     "अपने"    "लहू"      "न"        "बहने"    "दो"        ","
[22] "मैं"     "आपका"   "दीवाना"  "हूँ"        ","       "मुझे"    "बस"
[29] "अपना"   "पागल"    "रहने"    "दो"
```

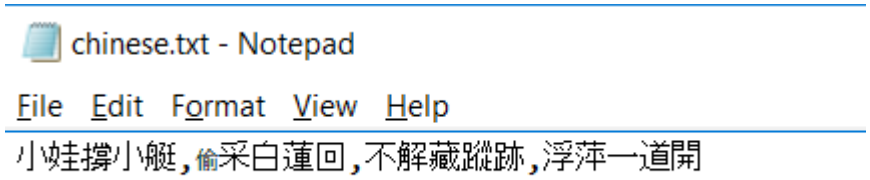
References

- https://www.rdocumentation.org/packages/base/versions/3.5.0/topics/readLines
- https://www.twilio.com/docs/glossary/what-is-ucs-2-character-encoding

Chinese

```
> model <- udpipe_load_model(file = "chinese-ud-2.0-170801.udpipe")
> x <- udpipe_annotate(model, " 小娃撐小艇 , 偷采白蓮回 , 不解藏蹤跡 , 浮萍
一道開    ")#mandarin poem
> x <- data.frame(x)
> x$token
 [1] "小"   "娃撐" "小艇" ","    "偷采" "白"   "蓮"   "回"   ","
[10] "不"   "解"   "藏蹤" "跡"   ","    "浮萍" "一"   "道"   "開"
> x$upos
 [1] "PART"  "NOUN"  "NOUN"  "PUNCT" "VERB"  "PROPN" "PROPN"
 [8] "VERB"  "PUNCT" "ADV"   "VERB"  "VERB"  "NOUN"  "PUNCT"
[15] "PROPN" "NUM"   "NOUN"  "VERB"
>
```

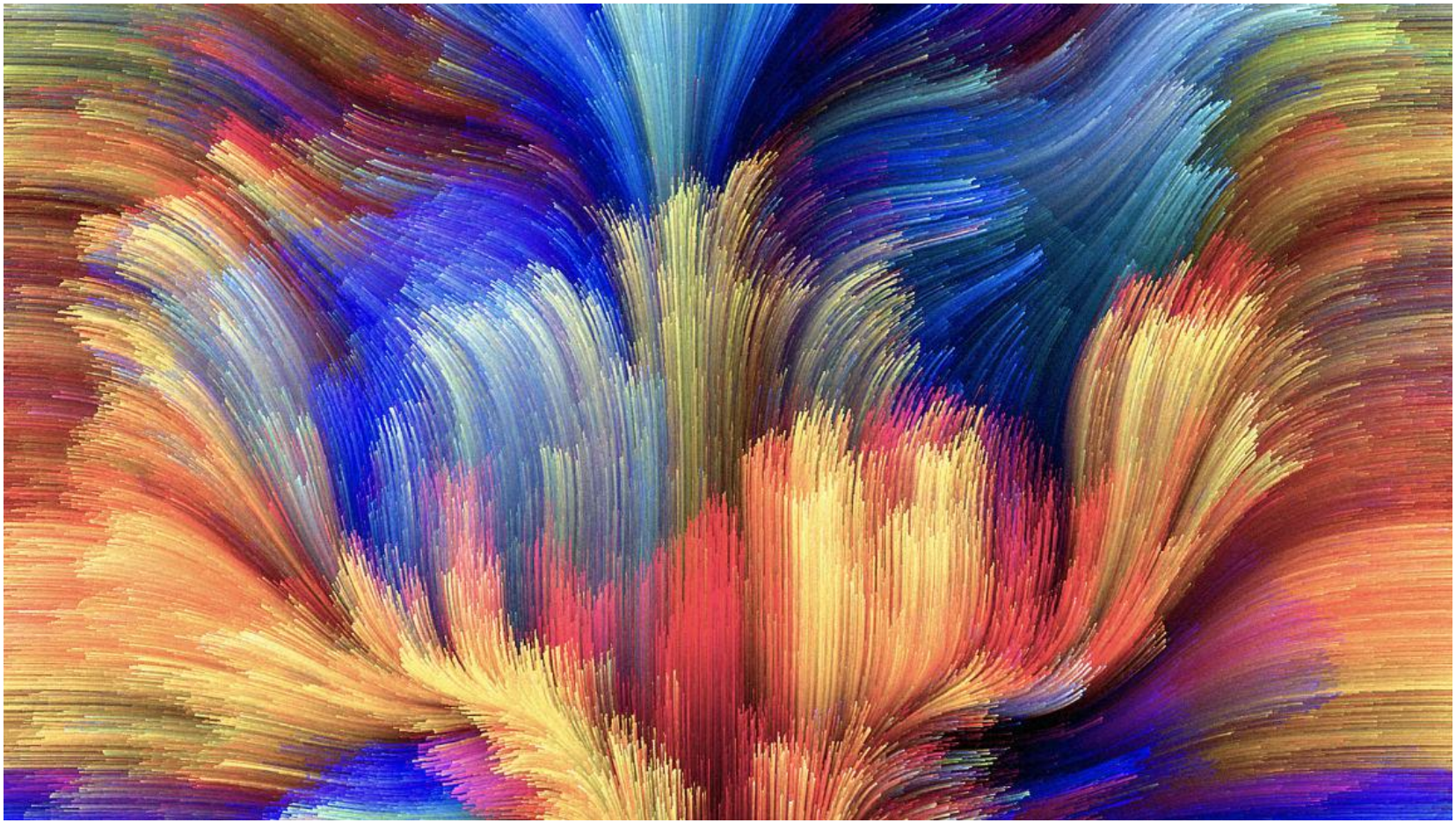
- **writeLines(text = paste(unlist(x$token), collapse=''), con = "Chinese.txt", useBytes = T)**



Challenge with Chinese

- Written mandarin consists of ideograms (very large vocabulary)
- Problem: Many ideograms (hanzi) mean different things depending on context
- Would conversion to *pinyin* yield a potentially more meaningful analysis?
- How to do wordclouds in 中文?

The power of Data



https://www.fastcompany.com/3040671/the-power-of-data-to-create-powerful-change


CLASS
DISMISSED
iStockphoto