

Unit 2.

Text Preprocessing

- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

Pre-processing

I Pre-processing

- ▶ **Tokenization:** break down the raw text into words or sentences.
- ▶ **Cleaning:**
 - Remove punctuation marks, excess spaces, special characters, etc. (*)
 - Also remove excessively short or infrequent words, etc.
- ▶ **Normalization:**
 - Conversion to lowercase.
 - Remove the stop words.
 - Stemming and Lemmatization.
 - Expansion of abbreviations. (*)
- ▶ (*) Regular expressions can be useful.

Tokenization

I Often the first step in the data pre-processing

► a) Tokenization into sentences:

- Usually, the sentence boundary is marked by a period, exclamation mark, question mark, etc.
- But sometimes, a period does not mark the end of a sentence: abbreviations, for example.

Ex

“He got his *M.D.* from the University of Wisconsin.”

- A good sentence tokenizer should recognize these exceptions.

► b) Tokenization into words:

- Usually splitting by white space or punctuation mark works.
- In some cases, there is ambiguity: apostrophe, for example.

Ex

“*Don’t* lose hope. *Everything’s* possible.”

⇒ How do we tokenize “*Don’t*” and “*Everything’s*”? ⇒ It depends on the tokenizer.

Tokenization

```
In [ ]: # NLTK offers tools necessary for tokenization of English corpus.
```

```
In [1]: from nltk.tokenize import word_tokenize
print(word_tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a past
[ 'Do', 'n't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.', 'Jone', "'s", 'Orphanage', 'is', 'as',
'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

```
In [ ]: # word_tokenize separated Don't into Do and n't, but Jone's into Jone and 's
```

```
In [2]: from nltk.tokenize import WordPunctTokenizer
print(WordPunctTokenizer().tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery
['Don', "'", 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr', '.', 'Jone', "'", 's', 'Orphanage',
'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

```
In [ ]: # Keras, also as a tokenization tool, supports text_to_word_sequence
```

```
In [3]: from tensorflow.keras.preprocessing.text import text_to_word_sequence
print(text_to_word_sequence("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes fo
['don't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'mr', "jone's", 'orphanage', 'is', 'as', 'cheery', 'a
s', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```

Unit 2.

Text Preprocessing

- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

Stop Words

Stop words

- ▶ Stop words are commonly used words that do not contain semantic information:

Ex Definite and indefinite articles: “the,” “a,” “an”

Ex Prepositions: “on,” “with,” “into,” “upon,” etc.

- ▶ **Stop words are removed** during the “**data normalization**” process.



Keywords always after removing stopwords

- ▶ Keywords are selected among the available words after removing the stop words.
 - Often, the most frequent words can be selected as keywords.
 - Sometimes, we should also take into account the purpose and context.



Stop words and keywords

- ▶ In the NLTK library, there are 179 English stop words:

i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, him, his, himself, she, she's, her, hers, herself, it, it's, its, itself, they, them, their, theirs, themselves, what, which, who, whom, this, that, that'll, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with, about, against, between, into, through, during, before, after, above, below, to, from, up, down, in, out, on, off, over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most, other, some, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren, aren't, couldn, couldn't, didn, didn't, doesn, doesn't, hadn, hadn't, hasn, hasn't, haven, haven't, isn, isn't, ma, mightn, mightn't, mustn, mustn't, needn, needn't, shan, shan't, shouldn, shouldn't, wasn, wasn't, weren, weren't, won, won't, wouldn, wouldn't

Stop words and keywords

```
In [9]: from nltk.corpus import stopwords
stopwords.words('english')[:10]

Out[9]: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]

In [11]: # NLTK defines words such as 'i', 'me', 'my' as stop words

In [10]: from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example)

result = []
for w in word_tokens:
    if w not in stop_words:
        result.append(w)

print(word_tokens)
print(result)

['Family', 'is', 'not', 'an', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
['Family', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
```


Unit 2.

Text Preprocessing

- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

Lemmatization and Stemming

Lemmatization



Stemming



Lemmatization and Stemming

I Lemmatization

- ▶ “The word “**better**” has “**good**” as its lemma. This **link is missed by stemming**, as it **requires a dictionary look-up**.
- ▶ The word “**walk**” is the base form for the word “**walking**”, and hence this is matched in both **stemming and lemmatization**.
- ▶ The word “**meeting**” can be either the base form of a **noun or** a form of a **verb** (“to meet”) depending on the context; e.g., “in our last meeting” or “We are meeting again tomorrow”. Unlike stemming, lemmatization attempts to select the correct lemma depending on the context.”[\(1\)](#)



Lemmatization and Stemming

Lemmatization

```
In [12]: from nltk.stem import WordNetLemmatizer
n=WordNetLemmatizer()
words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
print([n.lemmatize(w) for w in words])

['policy', 'doing', 'organization', 'have', 'going', 'love', 'life', 'fly', 'dy', 'watched', 'ha', 'starting']

In [13]: n.lemmatize('dies', 'v')
Out[13]: 'die'

In [14]: n.lemmatize('watched', 'v')
Out[14]: 'watch'

In [15]: n.lemmatize('has', 'v')
Out[15]: 'have'
```

Stemming

The most common algorithm in english language is Porter's Algorithm. The algorithm is too complex to explain. This algorithm works words through 5 phases and in each one there is a set of rules. These rules typically work by removing suffixes.

(F)	Rule		Example
	SSSES	→ SS	caresses → caress
	IES	→ I	ponies → poni
	SS	→ SS	caress → caress
	S	→	cats → cat



Stemming

```
In [16]: from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
s = PorterStemmer()
text="This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights
words=word_tokenize(text)
print(words)

['This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', "'s", 'chest', ',', 'but', 'an', 'accurate',
'copy', ',', 'complete', 'in', 'all', 'things', '--', 'names', 'and', 'heights', 'and', 'soundings', '--', 'with', 'the',
'single', 'exception', 'of', 'the', 'red', 'crosses', 'and', 'the', 'written', 'notes', '.']
```

```
In [17]: print([s.stem(w) for w in words])

['thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone', "'s", 'chest', ',', 'but', 'an', 'accur', 'copi',
',', 'complet', 'in', 'all', 'thing', '--', 'name', 'and', 'height', 'and', 'sound', '--', 'with', 'the', 'singl', 'excep
t', 'of', 'the', 'red', 'cross', 'and', 'the', 'written', 'note', '.']
```

Stemming

```
In [18]: #Stemmization of porter algorithm has rules as below.  
         #ALIZE → AL  
         #ANCE → Delete  
         #ICAL → IC
```

```
In [19]: words=['formalize', 'allowance', 'electrical']  
         print([s.stem(w) for w in words])  
  
         ['formal', 'allow', 'electric']
```

```
In [20]: from nltk.stem import PorterStemmer  
         s=PorterStemmer()  
         words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']  
         print([s.stem(w) for w in words])  
  
         ['polici', 'do', 'organ', 'have', 'go', 'love', 'live', 'fli', 'die', 'watch', 'ha', 'start']
```

```
In [21]: from nltk.stem import LancasterStemmer  
         l=LancasterStemmer()  
         words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']  
         print([l.stem(w) for w in words])  
  
         ['policy', 'doing', 'org', 'hav', 'going', 'lov', 'liv', 'fly', 'die', 'watch', 'has', 'start']
```

Unit 2.

Text Preprocessing

- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

POS Tagging

Part of Speech (POS) tag set in English (1/2)

Tag Set	Description	Example
CC	Coordinating conjunction	
CD	Cardinal digit	
DT	Determiner	
EX	Existential 'there'	there is
FW	Foreign word	
IN	Preposition/subordinating conjunction	
JJ	Adjective	Big
JJR	Adjective, comparative	Bigger
JJS	Adjective, superlative	Biggest
LS	List marker	1)
MD	Modal	could, will
NN	Noun, singular	Desk
NNS	Noun, plural	Desks
NNP	Proper noun, singular	Harrison
NNPS	Proper noun, plural	Americans
PDT	Predeterminer	'all' the kids
POS	Possessive ending	parent's

Part of Speech (POS) tag set in English (2/2)

Tag Set	Description	Example
PRP	Personal pronoun	I, he, she
PRP\$	Possessive pronoun	my, his, hers
RB	Adverb	very, silently
RBR	Adverb, comparative	better
RBS	Adverb, superlative	best
RP	Particle	give up
TO	To	go 'to' the store
UH	Interjection	errrrrrrm
VB	Verb, base form	take
VBD	Verb, past tense	took
VBG	Verb, gerund/present participle	taking
VBN	Verb, past participle	taken
VBP	Verb, sing. Present, non-3d	take
VBZ	Verb, 3rd person sing. Present	takes
WDT	Wh-determiner	which
WP	Wh-pronoun	who, what
WP\$	Possessive wh-pronoun	whose
WRB	Wh-abverb	where, when

Part of Speech (POS) tag set in English (Example)

```
In [22]: # Test sentence.
my_sentence = "The Colosseum was built by the emperor Vespassian"

In [24]: import nltk

In [25]: # Simple pre-processing.
my_words = nltk.word_tokenize(my_sentence)
for i in range(len(my_words)):
    my_words[i] = my_words[i].lower()
my_words

Out[25]: ['the', 'colosseum', 'was', 'built', 'by', 'the', 'emperor', 'vespassian']

In [26]: # POS tagging.
# OUTPUT: A list of tuples.
my_words_tagged = nltk.pos_tag(my_words)
my_words_tagged

Out[26]: [('the', 'DT'),
          ('colosseum', 'NN'),
          ('was', 'VBD'),
          ('built', 'VBN'),
          ('by', 'IN'),
          ('the', 'DT'),
          ('emperor', 'NN'),
          ('vespassian', 'NN')]
```

NLTK Library

I NLTK (Natural Language Toolkit)

- ▶ One of the most used Python libraries for the NLP:

a) Tokenization: `sent_tokenize`, `word_tokenize`, etc.

b) Stop words: 179 in total. The list of stop words needs to be downloaded.

```
nltk.download('stopwords')
```

c) Stemming: `PorterStemmer`, `LancasterStemmer`, `SnowballStemmer`, etc.

d) Lemmatization: `WordNetLemmatizer`.

e) POS tagging: Uses the “Penn Treebank tag set.”

f) Lexical resource: `WordNet`, `SentiWordNet`, etc. Useful for sentiment analysis.

Unit 2.

Text Preprocessing

- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

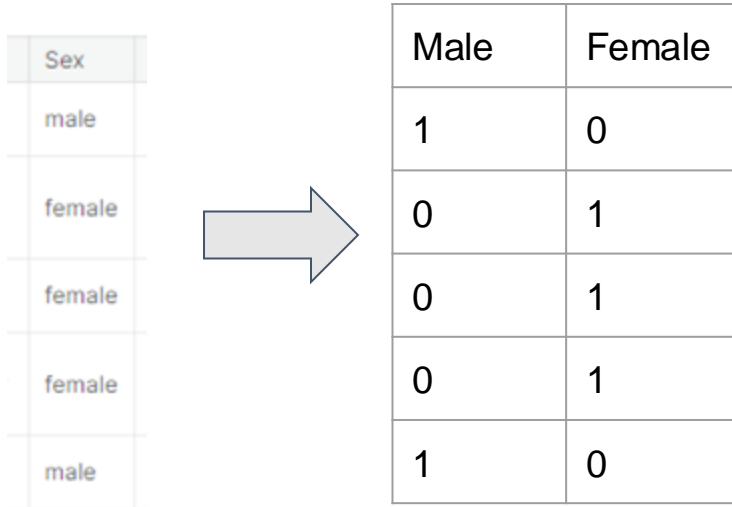
One-Hot Encoding

■ One-hot encoding using `category_encoders`

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

One-Hot Encoding

One-hot encoding using `category_encoders`



Sex	
male	
female	
female	
female	
male	

Male	Female
1	0
0	1
0	1
0	1
1	0

One-Hot Encoding

One-hot encoding using category_encoders

🏠 / One Hot

[View page source](#)

One Hot

```
class category_encoders.one_hot.OneHotEncoder(verbose=0, cols=None, drop_invariant=False,  
return_df=True, handle_missing='value', handle_unknown='value', use_cat_names=False) [source]
```

Onehot (or dummy) coding for categorical features, produces one feature per category, each binary.

Unit 2.

Text Preprocessing

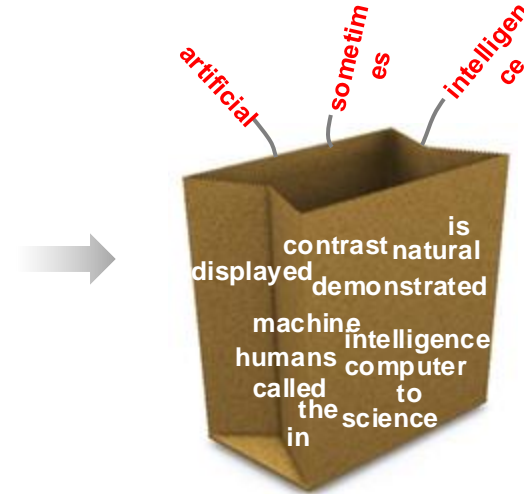
- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

Representation Model

Bag-of-Words (BOW) model

- ▶ A document is represented by a collection of its words.
- ▶ Word ordering and grammar are ignored.
- ▶ Only the word frequencies matter.

Ex “In computer science, artificial intelligence, sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans.”



I Bag-of-Words (BOW) model

Ex “It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness.”

⇒ after removing the stop words such as “it,” “the,” and “of,” the BOW can be expressed as an array.

age	best	foolishness	times	was	wisdom	worst
2	1	1	2	1	1	1

Unit 2.

Text Preprocessing

- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

Visualization

Word Cloud

- Visualization of the keywords where the size is related to the frequency.



Word Cloud

- ▶ We can create a word cloud following the steps below:
 - 1) Tokenize by words.
 - 2) Apply cleaning and normalization.
 - 3) Make a frequency table of the words.
 - 4) Sort by frequency and keep only the top keywords. The number of keywords is adjustable.
 - 5) Customize the arguments of the WordCloud() function.

Unit 2.

Text Preprocessing

- | 2.1. Tokenization
- | 2.2. Stop Words
- | 2.3. Lemmatization and Stemming
- | 2.4. POS Tagging
- | 2.5. One-Hot Encoding
- | 2.6. Bag of words
- | 2.7. Wordclouds
- | 2.8. tf-idf

Document-Term Matrix (DTM) and Term-Document Matrix (TDM)

- Documents expressed as BOW are the rows of DTM.
- Documents expressed as BOW are the columns of TDM.

	Feature #1	Feature #2	Feature #3	Feature #4	...
Document #1	1	0	1	0	0
Document #2	0	0	2	0	0
Document #3	0	1	0	0	1
Document #4	0	0	0	1	0
...	0	0	1	0	0

DTM

	Document #1	Document #2	Document #3	Document #4	...
Feature #1	1	0	0	0	0
Feature #2	0	0	1	0	0
Feature #3	1	2	0	0	1
Feature #4	0	0	0	1	0
...	0	0	1	0	0

TDM

I Document-Term Matrix (DTM) and Term-Document Matrix (TDM)

Ex Let's suppose the following pre-processed documents:

Document #1: "learning intelligence machine learning statistics"

Document #2: "machine classification learning performance"

Document #3: "machine classification, machine learning, machine performance"

DTM =

	learning	intelligence	machine	statistics	classification	performance
Doc. #1	2	1	1	1	0	0
Doc. #2	1	0	1	0	1	1
Doc. #3	1	0	3	0	1	1

I Document-Term Matrix (DTM) and Term-Document Matrix (TDM)

Ex Let's suppose the following pre-processed documents:

Document #1: “learning intelligence machine learning statistics”

Document #2: “machine classification learning performance”

Document #3: “machine classification, machine learning, machine performance”

TDM =

	Doc. #1	Doc. #2	Doc. #3
learning	2	1	1
intelligence	1	0	0
machine	1	1	3
statistics	1	0	0
classification	0	1	1
performance	0	1	1

I Term Frequency (TF)

- Indicates the relative importance of each word (term) within a document.
- A frequently occurring word within a short document would have a large TF value.

$$TF(\text{word}, \text{document}) = \frac{\text{Frequency of the word within the document}}{\text{The document length}}$$

- TF has to be calculated per word and per document.

I Term Frequency (TF)

Ex Let's suppose the following pre-processed documents:

Document #1: "learning intelligence machine learning statistics"

⇒ length = 5

Document #2: "machine classification learning performance"

⇒ length = 4

Document #3: "machine classification, machine learning, machine performance"

⇒ length = 6

TF =

	Doc. #1	Doc. #2	Doc. #3
learning	$2/5 = 0.4$	$1/4 = 0.25$	$1/6 = 0.17$
intelligence	$1/5 = 0.2$	0	0
machine	$1/5 = 0.2$	$1/4 = 0.25$	$3/6 = 0.5$
statistics	$1/5 = 0.2$	0	0
classification	0	$1/4 = 0.25$	$1/6 = 0.17$
performance	0	$1/4 = 0.25$	$1/6 = 0.17$

Document Frequency (DF) and Inverse Document Frequency (IDF)

- ▶ DF: the number of documents where a particular word appears
- ▶ IDF: a measure of rarity and information carried by a particular word

$$IDF(\textit{word}) = \textit{Log} \left(\frac{\textit{Total number of documents}}{\textit{Number of documents that include the word}} \right)$$

- ▶ IDF is a property of the corpus and has to be calculate per **word** only.

I Document Frequency (DF) and Inverse Document Frequency (IDF)

Ex Let's suppose the following pre-processed documents:

Document #1: "learning intelligence machine learning statistics"

Document #2: "machine classification learning performance"

Document #3: "machine classification, machine learning, machine performance"

IDF =

	DF	IDF
learning	3	$\text{Log}(3/3) = 0$
intelligence	1	$\text{Log}(3/1) = 0.48$
machine	3	$\text{Log}(3/3) = 0$
statistics	1	$\text{Log}(3/1) = 0.48$
classification	2	$\text{Log}(3/2) = 0.18$
performance	2	$\text{Log}(3/2) = 0.18$

TF IDF representation

Ex Let's suppose the following pre-processed documents:

Document #1: "learning intelligence machine learning statistics"

Document #2: "machine classification learning performance"

Document #3: "machine classification, machine learning, machine performance"

TF IDF =

	Doc. #1	Doc. #2	Doc. #3
learning	0.4	0.25	0.17
intelligence	0.2	0	0
machine	0.2	0.25	0.5
statistics	0.2	0	0
classification	0	0.25	0.17
performance	0	0.25	0.17

×

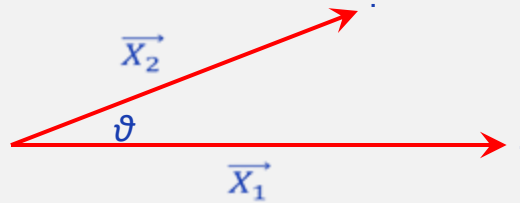
	IDF
learning	0
intelligence	0.48
machine	0
statistics	0.48
classification	0.18
performance	0.18

=

	Doc. #1	Doc. #2	Doc. #3
learning	0	0	0
intelligence	0.095	0	0
machine	0	0	0
statistics	0.095	0	0
classification	0	0.044	0.03
performance	0	0.044	0.03

■ Cosine similarity

- Documents are vectors. The similarity between two documents can be quantified.



Cosine similarity is
$$\cos(\theta) = \frac{\vec{X_1} \cdot \vec{X_2}}{|\vec{X_1}| |\vec{X_2}|}$$