Инструменты обработки и визуализации данных

Тема №4 – Обработка текстовых данных

Основные показатели и индексы читабельности

Рассмотрим вычисление базовых показателей, таких как количество слов, количество символов, среднюю длину слова и количество специальных символов (например, хэштегов). Также рассмотрим вычисление показателей читабельности и определение уровня знаний, необходимый для понимания текста.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (8, 6)
```

Проектирование признаков для NLP

Прямое кодирование

Рассмотрим фрейм данных `df1` , содержащий категориальные признаки, и, следовательно, непригодный для применения алгоритмов машинного обучения.

Наша задача — преобразовать `df1` в формат, подходящий для машинного обучения.

In [2]:
```
df1 = pd.read_csv('./FE_df1.csv')
```

In [3]:
```
# Print the features of df1
print(df1.columns)

# Perform one-hot encoding
df1 = pd.get_dummies(df1, columns=['feature 5'])

# Print the new features of df1
print(df1.columns)

# Print first five rows of df1
print(df1.head())
```

```
Index(['feature 1', 'feature 2', 'feature 3', 'feature 4', 'featur
e 5',
       'label'],
      dtype='object')
Index(['feature 1', 'feature 2', 'feature 3', 'feature 4', 'labe
l',
       'feature 5_female', 'feature 5_male'],
      dtype='object')
   feature 1  feature 2  feature 3  feature 4  label  feature 5_fe
male  \
0    29.0000          0          0   211.3375      1
True
1     0.9167          1          2   151.5500      1             F
alse
2     2.0000          1          2   151.5500      0
True
3    30.0000          1          2   151.5500      0             F
alse
4    25.0000          1          2   151.5500      0
True

   feature 5_male
0           False
1            True
2           False
3            True
4           False
```

Базовое извлечение признаков

Количество символов в русскоязычных твитах

Рассмотрим фрейм данных `tweets` , содержащий несколько твитов, связанных с Российским агентством интернет-исследований.

Наша задача — создать новый признак `'char_count'` в tweets, который будет вычислять количество символов в каждом твите. Также вычислите среднюю длину каждого твита. Твиты доступны в разделе `content` раздела `tweets` .

Имейте в виду, что это реальные данные из Twitter, и поэтому всегда существует риск, что они могут содержать ненормативную лексику или другой оскорбительный контент (в этом упражнении и любых последующих, где также используются реальные данные Twitter).

In [69]:
```
tweets = pd.read_csv('./russian_tweets.csv')
tweets.head()
```

Out[69]:

	Unnamed: 0	content
0	127447	LIVE STREAM VIDEO=> Donald Trump Rallies in Co...
1	123642	Muslim Attacks NYPD Cops with Meat Cleaver. Me...
2	226970	.@vfpatlas well that's a swella word there (di...
3	138339	RT wehking_pamela: Bobby_Axelrod2k MMFlint don...
4	166788	RT odom_1: HE'S DONE! THIS SHOCKING NEW EVIDEN...

In [5]:
```
# Create a feature char_count
tweets['char_count'] = tweets['content'].apply(len)

# Print the average character count
print(tweets['char_count'].mean())
```

```
103.462
```

Обратите внимание, что среднее количество символов в этих твитах составляет около 104, что значительно превышает общую среднюю длину твита, составляющую около 40 символов. В зависимости от того, над чем вы работаете, это может быть полезно для изучения. К вашему сведению, есть исследования, которые показывают, что статьи с фейковыми новостями, как правило, имеют более длинные заголовки! Поэтому даже такие базовые функции, как подсчёт символов, могут оказаться очень полезными в некоторых приложениях.

Количество слов в выступлениях на TED

`ted` — это датафрейм, содержащий расшифровки 500 выступлений на конференциях TED (https://www.ted.com (https://www.ted.com)). Ваша задача — вычислить новый признак `word_count`, который содержит приблизительное количество слов в каждом выступлении. Следовательно, вам также необходимо вычислить среднее количество слов в выступлениях. Расшифровки доступны как признак `transcript в ted`.

Для выполнения этой задачи вам необходимо определить функцию `count_words`, которая принимает строку в качестве аргумента и возвращает количество слов в строке. Затем вам нужно применить эту функцию к признаку `transcript в ted`, чтобы создать новый признак `word_count` и вычислить его среднее значение.

In [6]:
```
ted = pd.read_csv('./ted.csv')
ted.head()
```

Out[6]:

	transcript	url
0	We're going to talk — my — a new lecture, just...	https://www.ted.com/talks/al_seckel_says_our_b...
1	This is a representation of your brain, and yo...	https://www.ted.com/talks/aaron_o_connell_maki...
2	It's a great honor today to share with you The...	https://www.ted.com/talks/carter_emmart_demos_...
3	My passions are music, technology and making t...	https://www.ted.com/talks/jared_ficklin_new_wa...
4	It used to be that if you wanted to get a comp...	https://www.ted.com/talks/jeremy_howard_the_wo...

In [7]:
```
# Function that returns number of words in a string
def count_words(string):
    # Split the string into words
    words = string.split()

    # Return the number of words
    return len(words)

# Create a new feature word_count
ted['word_count'] = ted['transcript'].apply(count_words)

# Print the average word count of the talks
print(ted['word_count'].mean())
```

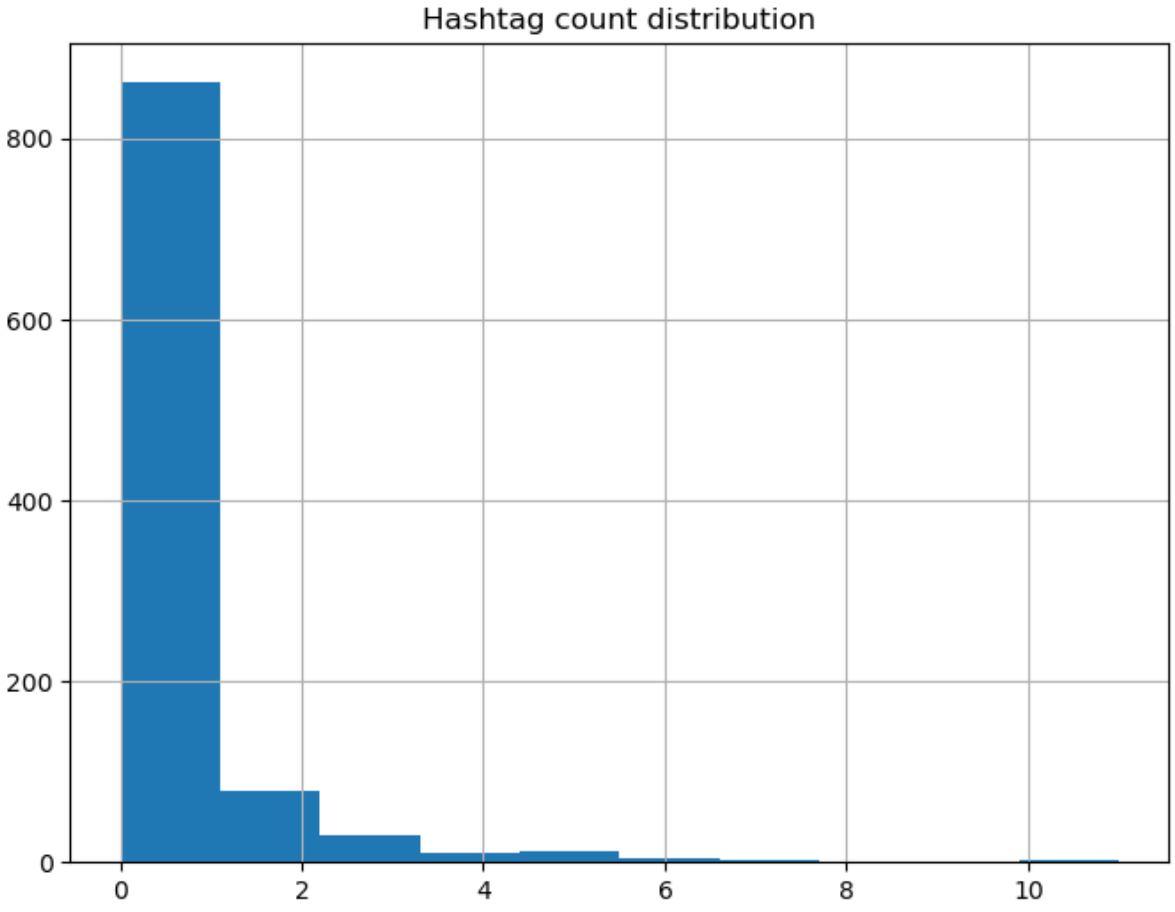
```
1987.1
```

Теперь вы знаете, как подсчитать количество слов в заданном тексте. Обратите внимание, что средняя длина выступления составляет около 2000 слов. Вы можете использовать функцию `word_count` для вычисления корреляции этого показателя с другими переменными, такими как количество просмотров, количество комментариев и т. д., и получить крайне интересную информацию о TED.

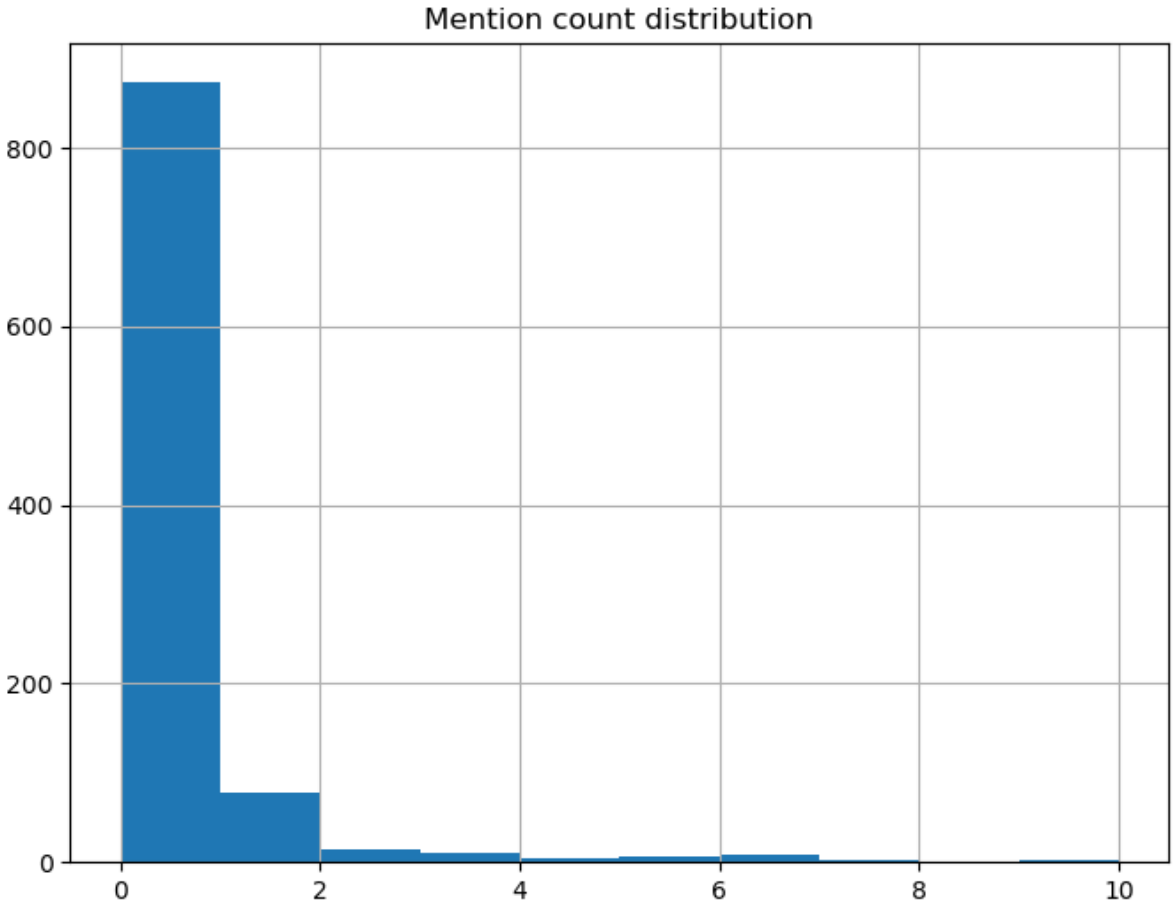
Хэштеги и упоминания в русскоязычных твитах

Давайте вернёмся к фрейму данных твитов, содержащему русскоязычные твиты. В этом упражнении вы вычислите количество хэштегов и упоминаний в каждом твите, определив две функции `count_hashtags()` и `count_mentions()` соответственно и применив их к содержанию твитов.

In [8]:
```
# Function that returns number of hashtags in a string
def count_hashtags(string):
    # Split the string into words
    words = string.split()

    # Create a list of words that are hashtags
    hashtags = [word for word in words if word.startswith('#')]

    # Return number of hashtags
    return(len(hashtags))

# Create a feature hashtag_countand display distribution
tweets['hashtag_count'] = tweets['content'].apply(count_hashtags)
tweets['hashtag_count'].hist();
plt.title('Hashtag count distribution');
```



In [9]:
```
# Function that returns number of mentions in a string
def count_mentions(string):
    # Split the string into words
    words = string.split()

    # Create a list of words that are mentions
    mentions = [word for word in words if word.startswith('@')]

    # Return number of mentions
    return(len(mentions))

# Create a feature mention_count and display distribution
tweets['mention_count'] = tweets['content'].apply(count_mentions)
tweets['mention_count'].hist();
plt.title('Mention count distribution');
```



Теперь вы хорошо разбираетесь в вычислении различных типов сводных признаков. В следующем уроке мы изучим более сложные признаки, которые позволяют получить более подробную информацию, выходящую за рамки простого подсчёта количества слов и символов.

Тесты на читабельность

- Тест на читабельность
  - Определение читабельности английского текста
  - Шкала, охватывающая уровень от начальной школы до уровня выпускника колледжа
  - Математическая формула, использующая количество слов, слогов и предложений
  - Используется для обнаружения фейковых новостей и спама
- Примеры
  - Легкость чтения по Флешу
  - Индекс туманности Ганнинга
  - Простая мера абракадабры (Simple Measure of Gobbledygook, SMOG)
  - Шкала Дейла-Чалла
- Легкость чтения по Флешу
  - Один из старейших и наиболее широко используемых тестов
  - Зависит от двух факторов:
    - Чем больше средняя длина предложения, тем сложнее читать текст
    - Чем больше среднее количество слогов в слове, тем сложнее читать текст
  - Чем выше балл, тем выше читаемость
- Индекс туманности Ганнинга
  - Разработан в 1954 году
  - Также зависит от средней длины предложения
  - Чем больше процент сложных слов, тем сложнее читать текст
  - Чем выше индекс, тем хуже читаемость

Читабельность «Мифа о Сизифе»

В этом упражнении нам предстоит вычислить индекс лёгкости чтения по шкале Флеша для знаменитого эссе Альбера Камю «Миф о Сизифе». Затем мы интерпретируем значение этого индекса, и попытаемся определить уровень читабельности эссе.

In [10]:
```
with open('./sisyphus_essay.txt', 'r') as f:
    sisyphus_essay = f.read()
sisyphus_essay[:100]
```

Out[10]:
```
'\nThe gods had condemned Sisyphus to ceaselessly rolling a rock t
o the top of a mountain, whence the '
```

In [11]:
```
# pip install textatistic
from textatistic import Textatistic

# Compute the readability scores
readability_scores = Textatistic(sisyphus_essay).scores

# Print the flesch reading ease score
flesch = readability_scores['flesch_score']
print('The Flesch Reading Ease is %.2f' % (flesch))
```

```
The Flesch Reading Ease is 81.67
```

In [12]:
```
readability_scores
```

Out[12]:
```
{'flesch_score': 81.67466335836913,
 'fleschkincaid_score': 5.485083154506439,
 'gunningfog_score': 7.913698140200286,
 'smog_score': 8.110721755262034,
 'dalechall_score': 7.487433762517882}
```

Мы рассмотрели, как рассчитать индекс лёгкости чтения по Флешу для заданного текста. Обратите внимание, что для этого эссе оценка составляет приблизительно 81,67. Это означает, что эссе соответствует уровню читабельности американского ученика 6-го класса.

Читабельность различных публикаций

В этом упражнении нам даны отрывки статей из четырёх публикаций. Наша задача — оценить читабельность этих отрывков с помощью индекса туманности Ганнинга и, следовательно, определить относительную сложность чтения этих публикаций.

Отрывки доступны в виде следующих строк:

- `forbes` — отрывок из статьи журнала Forbes о китайской системе социального кредита.
- `harvard_law` — отрывок из рецензии на книгу, опубликованной в Harvard Law Review.
- `r_digest` — отрывок из статьи Reader's Digest о турбулентности в полёте.
- `time_kids` — отрывок из статьи о вреде потребления соли, опубликованной в журнале TIME for Kids.

In [13]:
```
# conda install glob2
import glob
texts = []
text_list = glob.glob('./*.txt')

text_list
```

Out[13]:
```
['./mother.txt',
 './hey.txt',
 './lotf.txt',
 './tc.txt',
 './gettysburg.txt',
 './hopes.txt',
 './forbes.txt',
 './sisyphus_essay.txt',
 './harvard_law.txt',
 './time_kids.txt',
 './blog.txt',
 './r_digest.txt']
```

In [14]:
```
file_list = ['time_kids', 'forbes', 'r_digest', 'harvard_law']
for text in text_list:
    for f in file_list:
        if f in text:
            with open(text, 'r') as f:
                texts.append(f.read())

time_kids, forbes, r_digest, harvard_law = texts
```

In [15]:
```
# List of excerpts
excerpts = [forbes, harvard_law, r_digest, time_kids]

# Loop through excerpts and compute gunning fog index
gunning_fog_scores = []
for excerpt in excerpts:
    readability_scores = Textatistic(excerpt).scores
    gunning_fog = readability_scores['gunningfog_score']
    gunning_fog_scores.append(gunning_fog)

# Print the gunning fog indices
print(gunning_fog_scores)
```

```
[20.735401069518716, 11.085587583148559, 5.926785009861934, 14.436
002482929858]
```

Обратите внимание, что отрывок из Harvard Law Review имеет самый высокий индекс туманности Ганнинга, что говорит о том, что его могут понять только читатели, окончившие колледж. С другой стороны, статья из Time for Kids, предназначенная для детей, имеет гораздо более низкий индекс неясности и может быть понята учениками пятого класса.

Предварительная обработка текста, POS-тегирование и NER

> Вначале рассмотрим токенизацию и лемматизацию. Затем рассмотрим очистку текста, частеречную маркировку и распознавание именованных сущностей с помощью библиотеки spaCy. Освоив эти концепции, мы сможем адаптировать речь в Геттисберге для машинного перевода, проанализировать использование существительных в фейковых новостях и идентифицировать людей, упомянутых в статье TechCrunch.

Токенизация и лемматизация

- Методы предварительной обработки текста
  - Преобразование слов в строчные буквы
  - Удаление начальных и конечных пробелов
  - Удаление знаков препинания
  - Удаление стоп-слов
  - Раскрытие сокращений
- Токенизация
  - процесс разбиения строки на составляющие её токены
- Лемматизация
  - процесс преобразования слова в его базовую форму с строчными буквами или лемму

Токенизация Геттисбергской речи

В этом упражнении будем токенизировать одну из самых известных речей – Геттисбергскую речь, произнесённую президентом США Авраамом Линкольном во время Гражданской войны в США.

In [16]:
```
with open('./gettysburg.txt', 'r') as f:
    gettysburg = f.read()
```

In [17]:
```
# conda install spacy
# python -m spacy download en_core_web_sm
# python -m spacy validate
import spacy

# Load the en_core_web_sm model
nlp = spacy.load('en_core_web_sm')

# create a Doc object
doc = nlp(gettysburg)

# Generate the tokens
tokens = [token.text for token in doc]
print(tokens)
```

```
['Four', 'score', 'and', 'seven', 'years', 'ago', 'our', 'father
s', 'brought', 'forth', 'on', 'this', 'continent', ',', 'a', 'ne
w', 'nation', ',', 'conceived', 'in', 'Liberty', ',', 'and', 'dedi
cated', 'to', 'the', 'proposition', 'that', 'all', 'men', 'are', '
created', 'equal', '.', 'Now', 'we', "'re", 'engaged', 'in', 'a',
'great', 'civil', 'war', ',', 'testing', 'whether', 'that', 'natio
n', ',', 'or', 'any', 'nation', 'so', 'conceived', 'and', 'so', 'd
edicated', ',', 'can', 'long', 'endure', '.', 'We', "'re", 'met',
'on', 'a', 'great', 'battlefield', 'of', 'that', 'war', '.', 'We',
"'ve", 'come', 'to', 'dedicate', 'a', 'portion', 'of', 'that', 'fi
eld', ',', 'as', 'a', 'final', 'resting', 'place', 'for', 'those',
'who', 'here', 'gave', 'their', 'lives', 'that', 'that', 'nation',
'might', 'live', '.', 'It', "'s", 'altogether', 'fitting', 'and',
'proper', 'that', 'we', 'should', 'do', 'this', '.', 'But', ',', '
in', 'a', 'larger', 'sense', ',', 'we', 'ca', "n't", 'dedicate', '
-', 'we', 'can', 'not', 'consecrate', '-', 'we', 'can', 'not', 'ha
llow', '-', 'this', 'ground', '.', 'The', 'brave', 'men', ',', 'li
ving', 'and', 'dead', ',', 'who', 'struggled', 'here', ',', 'hav
e', 'consecrated', 'it', ',', 'far', 'above', 'our', 'poor', 'powe
r', 'to', 'add', 'or', 'detract', '.', 'The', 'world', 'will', 'li
ttle', 'note', ',', 'nor', 'long', 'remember', 'what', 'we', 'sa
y', 'here', ',', 'but', 'it', 'can', 'never', 'forget', 'what', 't
hey', 'did', 'here', '.', 'It', 'is', 'for', 'us', 'the', 'livin
g', ',', 'rather', ',', 'to', 'be', 'dedicated', 'here', 'to', 'th
e', 'unfinished', 'work', 'which', 'they', 'who', 'fought', 'her
e', 'have', 'thus', 'far', 'so', 'nobly', 'advanced', '.', 'It',
"'s", 'rather', 'for', 'us', 'to', 'be', 'here', 'dedicated', 't
o', 'the', 'great', 'task', 'remaining', 'before', 'us', '-', 'tha
t', 'from', 'these', 'honored', 'dead', 'we', 'take', 'increased',
'devotion', 'to', 'that', 'cause', 'for', 'which', 'they', 'gave',
'the', 'last', 'full', 'measure', 'of', 'devotion', '-', 'that', '
we', 'here', 'highly', 'resolve', 'that', 'these', 'dead', 'shal
l', 'not', 'have', 'died', 'in', 'vain', '-', 'that', 'this', 'nat
ion', ',', 'under', 'God', ',', 'shall', 'have', 'a', 'new', 'birt
h', 'of', 'freedom', '-', 'and', 'that', 'government', 'of', 'th
e', 'people', ',', 'by', 'the', 'people', ',', 'for', 'the', 'peop
le', ',', 'shall', 'not', 'perish', 'from', 'the', 'earth', '.']
```

Лемматизация речи в Геттисберге

В этом упражнении мы выполним лемматизацию той же речи `gettysburg` , что и раньше.

Однако на этот раз мы также рассмотрим речь до и после лемматизации и попытаемся оценить, какие изменения вносятся, чтобы сделать фрагмент более удобным для машинного восприятия.

In [18]:
```
# Print the gettysburg address
print(gettysburg)
```

```
Four score and seven years ago our fathers brought forth on this c
ontinent, a new nation, conceived in Liberty, and dedicated to the
proposition that all men are created equal. Now we're engaged in a
great civil war, testing whether that nation, or any nation so con
ceived and so dedicated, can long endure. We're met on a great bat
tlefield of that war. We've come to dedicate a portion of that fie
ld, as a final resting place for those who here gave their lives t
hat that nation might live. It's altogether fitting and proper tha
t we should do this. But, in a larger sense, we can't dedicate - w
e can not consecrate - we can not hallow - this ground. The brave
men, living and dead, who struggled here, have consecrated it, far
above our poor power to add or detract. The world will little not
e, nor long remember what we say here, but it can never forget wha
t they did here. It is for us the living, rather, to be dedicated
here to the unfinished work which they who fought here have thus f
ar so nobly advanced. It's rather for us to be here dedicated to t
he great task remaining before us - that from these honored dead w
e take increased devotion to that cause for which they gave the la
st full measure of devotion - that we here highly resolve that the
se dead shall not have died in vain - that this nation, under God,
shall have a new birth of freedom - and that government of the peo
ple, by the people, for the people, shall not perish from the eart
h.
```

In [19]:
```
# Generate lemmas
lemmas = [token.lemma_ for token in doc]

# Convert lemmas into a string
print(' '.join(lemmas))
```

```
four score and seven year ago our father bring forth on this conti
nent , a new nation , conceive in Liberty , and dedicate to the pr
oposition that all man be create equal . now we be engage in a gre
at civil war , test whether that nation , or any nation so conceiv
ed and so dedicated , can long endure . we be meet on a great batt
lefield of that war . we have come to dedicate a portion of that f
ield , as a final resting place for those who here give their life
that that nation might live . it be altogether fitting and proper
that we should do this . but , in a large sense , we can not dedic
ate - we can not consecrate - we can not hallow - this ground . th
e brave man , living and dead , who struggle here , have consecrat
e it , far above our poor power to add or detract . the world will
little note , nor long remember what we say here , but it can neve
r forget what they do here . it be for we the living , rather , to
be dedicate here to the unfinished work which they who fight here
have thus far so nobly advanced . it be rather for we to be here d
edicate to the great task remain before we - that from these honor
dead we take increase devotion to that cause for which they give t
he last full measure of devotion - that we here highly resolve tha
t these dead shall not have die in vain - that this nation , under
God , shall have a new birth of freedom - and that government of t
he people , by the people , for the people , shall not perish from
the earth .
```

Обратите внимание на лемматизированную версию речи. Она не очень удобна для восприятия человеком, но гораздо удобнее для обработки машиной.

Очистка текста

- Методы очистки текста
  - Ненужные пробелы и управляющие последовательности
  - Знаки препинания
  - Специальные символы (цифры, эмодзи и т. д.)
  - Стоп-слова
- Стоп-слова
  - Слова, которые встречаются очень часто
  - Например, артикли, глаголы типа «быть», местоимения и т. д.

Очистка записи в блоге

В этом упражнении нам предоставлен отрывок записи в блоге. Наша задача — очистить этот текст, придав ему более удобный для машинного ввода формат. Это включает в себя преобразование в строчные буквы, лемматизацию и удаление стоп-слов, знаков препинания и неалфавитных символов.

In [20]:
```
with open('./blog.txt', 'r') as file:
    blog = file.read()

stopwords = spacy.lang.en.stop_words.STOP_WORDS
blog = blog.lower()
```

In [21]:
```
# Generate doc Object: doc
doc = nlp(blog)

# Generate lemmatized tokens
lemmas = [token.lemma_ for token in doc]

# Remove stopwords and non-alphabetic tokens
a_lemmas = [lemma for lemma in lemmas
            if lemma.isalpha() and lemma not in stopwords]

# Print string after text cleaning
print(' '.join(a_lemmas))
```

```
century politic witness alarming rise populism europe warning sign
come uk brexit referendum vote swinge way leave follow stupendous
victory billionaire donald trump president united states november
europe steady rise populist far right party capitalize europe immi
gration crisis raise nationalist anti europe sentiment instance in
clude alternative germany afd win seat enter bundestag upset germa
ny political order time second world war success star movement ita
ly surge popularity neo nazism neo fascism country hungary czech r
epublic poland austria
```

Взгляните на очищенный текст: он набран строчными буквами и лишён цифр, знаков препинания и часто используемых стоп-слов. Также обратите внимание, что в исходном тексте присутствовало слово «U.S.». Поскольку между ними были точки, процесс очистки текста полностью удалил его. Это может быть неправильным решением. В более сложных случаях всегда рекомендуется использовать собственные функции вместо `isalpha()` .

Очистка выступлений на TED в датафрейме

В этом упражнении вернёмся к выступлениям на конференциях TED. Нам предоставлен датафрейм `ted` , состоящий из 5 выступлений TED. Наша задача — очистить эти выступления, используя ранее обсуждавшиеся методы, написав функцию `preprocess` и применив её к свойству `transcript` датафрейма.

In [22]:
```
ted = pd.read_csv('./ted.csv')
ted['transcript'] = ted['transcript'].str.lower()
ted.head()
```

Out[22]:

	transcript	url
0	we're going to talk — my — a new lecture, just...	https://www.ted.com/talks/al_seckel_says_our_b...
1	this is a representation of your brain, and yo...	https://www.ted.com/talks/aaron_o_connell_maki...
2	it's a great honor today to share with you the...	https://www.ted.com/talks/carter_emmart_demos_...
3	my passions are music, technology and making t...	https://www.ted.com/talks/jared_ficklin_new_wa...
4	it used to be that if you wanted to get a comp...	https://www.ted.com/talks/jeremy_howard_the_wo...

In [23]:
```
# Function to preprocess text
def preprocess(text):
    # Create Doc object
    doc = nlp(text, disable=['ner', 'parser'])

    # Generate lemmas
    lemmas = [token.lemma_ for token in doc]

    # Remove stopwords and non-alphabetic characters
    a_lemmas = [lemma for lemma in lemmas
                if lemma.isalpha() and lemma not in stopwords]

    return ' '.join(a_lemmas)

# Apply preprocess to ted['transcript']
ted['transcript'] = ted['transcript'].apply(preprocess)
print(ted['transcript'])
```

```
0      talk new lecture ted I illusion create ted I t...
1      representation brain brain break left half log...
2      great honor today share digital universe creat...
3      passion music technology thing combination thi...
4      use want computer new program programming requ...
                             ...
495    today I unpack example iconic design perfect s...
496    brother belong demographic pat percent accord ...
497    john hockenberry great tom I want start questi...
498    right moment kill car internet little mobile d...
499    real problem math education right basically ha...
Name: transcript, Length: 500, dtype: object
```

Мы выполнили предварительную обработку всех стенограмм выступлений TED, содержащихся в `ted`, и теперь они готовы к таким операциям, как векторизация. Теперь вы хорошо понимаете, как работает предварительная обработка текста и почему она важна. Перейдём к созданию признаков на уровне слов для наших текстов.

Разметка частей речи

- Часть речи (Part-of-Speech, POS)
  - помогает определить различия, идентифицируя одно слово (например, `bear`) как существительное, а другое (`bear`) как глагол
  - Разрешение смысловой неоднозначности
    - "The bear is a majestic animal"
    - "Please bear with me" (Пожалуйста, потерпите меня.)
  - Анализ тональности
  - Ответы на вопросы
  - Обнаружение фейковых новостей и спама
- Разметка POS
  - Назначение каждому слову соответствующей части речи
- Аннотация POS в spaCy
  - `PROPN` — имя собственное
  - `DET` — определение

POS-теги в романе «Повелитель мух»

В этом упражнении мы выполним частичную маркировку известного отрывка из романа «Повелитель мух» (Lord of the Flies) Уильяма Голдинга.

In [24]:
```
with open('./lotf.txt', 'r') as file:
    lotf = file.read()
```

In [25]:
```
# Create Doc object
doc = nlp(lotf)

# Generate tokens and pos tags
pos = [(token.text, token.pos_) for token in doc]
print(pos)
```

```
[('He', 'PRON'), ('found', 'VERB'), ('himself', 'PRON'), ('understanding', 'VERB'), ('the', 'DET'), ('wearisomeness', 'NOUN'), ('of', 'ADP'), ('this', 'DET'), ('life', 'NOUN'), (',', 'PUNCT'), ('where', 'SCONJ'), ('every', 'DET'), ('path', 'NOUN'), ('was', 'AUX'), ('an', 'DET'), ('improvisation', 'NOUN'), ('and', 'CCONJ'), ('a', 'DET'), ('considerable', 'ADJ'), ('part', 'NOUN'), ('of', 'ADP'), ('one', 'NUM'), ('’s', 'VERB'), ('waking', 'VERB'), ('life', 'NOUN'), ('was', 'AUX'), ('spent', 'VERB'), ('watching', 'VERB'), ('one', 'NUM'), ('’s', 'PART'), ('feet', 'NOUN'), ('.', 'PUNCT')]
```

Изучите различные POS-теги, прикреплённые к каждому токену, и оцените, понятны ли они вам интуитивно. Вы заметите, что они действительно правильно промаркированы в соответствии со стандартными правилами английской грамматики.

Подсчёт существительных в тексте

В этом упражнении мы напишем две функции, `nouns()` и `proper_nouns()`, которые будут подсчитывать количество других существительных и имён собственных в тексте соответственно.

Эти функции принимают текст и генерируют список, содержащий POS-теги для каждого слова. Затем они возвращают количество имён собственных/других существительных в тексте. Мы используем эти функции в следующем упражнении для получения интересных сведений о фейковых новостях.

In [26]:
```
# Returns number of proper nouns
def proper_nouns(text, model=nlp):
    # Create doc object
    doc = model(text)

    # Generate list of POS tags
    pos = [token.pos_ for token in doc]

    # Return number of proper nouns
    return pos.count('PROPN')

print(proper_nouns(
    'Abdul, Bill and Cathy went to the market to buy apples.', nlp
))
```

```
3
```

In [27]:
```
# Returns number of other nouns
def nouns(text, model=nlp):
    # create doc object
    doc = model(text)

    # Generate list of POS tags
    pos = [token.pos_ for token in doc]

    # Return number of other nouns
    return pos.count('NOUN')

print(nouns(
    'Abdul, Bill and Cathy went to the market to buy apples.', nlp
))
```

```
2
```

Использование существительных в фейковых новостях

В этом упражнении нам предоставлена таблица данных `headlines`, содержащая заголовки новостей, которые могут быть как фейковыми, так и настоящими. Наша задача — сгенерировать два новых признака `num_propn` и `num_noun`, которые представляют количество имён собственных и других существительных, содержащихся в признаке `title` из `headlines`.

Далее мы вычислим среднее количество имён собственных и других существительных, используемых в фейковых и настоящих новостных заголовках, и сравним результаты. Если разница значительна, то велика вероятность, что использование признаков `num_propn` и `num_noun` в детекторах фейковых новостей повысит их эффективность.

In [28]:
```
headlines = pd.read_csv('./fakenews.csv')
headlines.head()
```

Out[28]:

	Unnamed: 0	title	label
0	0	You Can Smell Hillary's Fear	FAKE
1	1	Watch The Exact Moment Paul Ryan Committed Pol...	FAKE
2	2	Kerry to go to Paris in gesture of sympathy	REAL
3	3	Bernie supporters on Twitter erupt in anger ag...	FAKE
4	4	The Battle of New York: Why This Primary Matters	REAL

In [29]:
```
headlines['num_propn'] = headlines['title'].apply(proper_nouns)
headlines['num_noun'] = headlines['title'].apply(nouns)

# Compute mean of proper nouns
real_propn = headlines[headlines['label'] == 'REAL']['num_propn'].me
fake_propn = headlines[headlines['label'] == 'FAKE']['num_propn'].me

# Compute mean of other nouns
real_noun = headlines[headlines['label'] == 'REAL']['num_noun'].mean
fake_noun = headlines[headlines['label'] == 'FAKE']['num_noun'].mean

# Print results
print("Mean no. of proper nouns in real and fake headlines are\n",
      "%.2f and %.2f respectively" % (real_propn, fake_propn))
print("Mean no. of other nouns in real and fake headlines are\n",
      "%.2f and %.2f respectively" % (real_noun, fake_noun))
```

```
Mean no. of proper nouns in real and fake headlines are
 2.42 and 4.81 respectively
Mean no. of other nouns in real and fake headlines are
 2.37 and 1.65 respectively
```

Мы рассмотрели, как создавать признаки, используя информацию из POS-тегов. Обратите внимание, что среднее количество имён собственных в фейковых новостях значительно выше, чем в настоящих. В случае с другими существительными, похоже, всё наоборот. Этот факт можно эффективно использовать при разработке детекторов фейковых новостей.

Распознавание именованных сущностей

- Распознавание именованных сущностей (Named Entity Recognition, NER)
  - Идентификация и классификация именованных сущностей по предопределенным категориям
  - Категории включают в себя человека, организацию, страну и т. д.

Именованные сущности в предложении

В этом упражнении мы определим и классифицируем обозначения различных именованных сущностей в тексте, используя одну из статистических моделей spaCy. Мы также проверим достоверность этих обозначений.

In [30]:
```
# Create a Doc instance
text = """
Sundar Pichai is the CEO of Google.
Its headquarter is in Mountain View.
"""

doc = nlp(text)

# Print all named entities and their labels
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
Sundar Pichai PERSON
Google ORG
Mountain View GPE
```

Определение людей, упомянутых в новостной статье

В этом упражнении нам предоставлен отрывок из новостной статьи, опубликованной в TechCrunch. Наша задача — написать функцию `find_people`, которая находит имена людей, упомянутых в определённом фрагменте текста. Затем мы будем использовать `find_people` для определения интересующих нас людей в статье.

In [31]:
```
with open('./tc.txt', 'r') as file:
    tc = file.read()
```

In [32]:
```
def find_persons(text):
    # Create Doc object
    doc = nlp(text)

    # Indentify the persons
    persons = [ent.text for ent in doc.ents if ent.label_ == 'PERSO

    # Return persons
    return persons

print(find_persons(tc))
```

```
['Sheryl Sandberg', 'Mark Zuckerberg']
```

Наша функция правильно идентифицировала обоих упомянутых людей. Теперь вы видите, как NER можно использовать в различных приложениях. Издатели могут использовать подобный метод для классификации новостных статей по упоминаемым в них людям. Вопросно-ответная система также может использовать нечто подобное для ответа на вопросы, например: «Кто упомянут в этом отрывке?».

Модели N-грамм

> Рассмотрим теперь моделирование N-грамм и используем его для анализа тональности обзоров фильмов.

Построение модели «мешка слов»

- Модель «мешка слов» (bag of words)
  - Извлечение токенов слов
  - Вычисление частот встречаемости токенов слов
  - Построение вектора слов на основе этих частот и словарного запаса корпуса текстов

Модель BoW для слоганов фильмов

В этом упражнении нам предоставлен корпус из более чем 7000 слоганов фильмов. Наша задача — сгенерировать набор слов `bow_matrix` для этих слоганов. В этом упражнении мы пропустим этап предварительной обработки текста и сгенерируем матрицу `bow_matrix` напрямую.

In [33]:
```
movies = pd.read_csv('./movie_overviews.csv').dropna()
movies['tagline'] = movies['tagline'].str.lower()
movies.head()
```

Out[33]:

	id	title	overview	tagline
1	8844	Jumanji	When siblings Judy and Peter discover an encha...	roll the dice and unleash the excitement!
2	15602	Grumpier Old Men	A family wedding reignites the ancient feud be...	still yelling. still fighting. still ready for...
3	31357	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...	friends are the people who let you be yourself...
4	11862	Father of the Bride Part II	Just when George Banks has recovered from his ...	just when his world is back to normal... he's ...
5	949	Heat	Obsessive master thief, Neil McCauley leads a ...	a los angeles crime saga

In [34]:
```
corpus = movies['tagline']
```

In [35]:
```
from sklearn.feature_extraction.text import CountVectorizer

# Create CountVectorizer object
vectorizer = CountVectorizer()

# Generate matrix of word vectors
bow_matrix = vectorizer.fit_transform(corpus)

# Print the shape of bow_matrix
print(bow_matrix.shape)
```

```
(7033, 6614)
```

Мы рассмотрели, как создать представление «мешок слов» для заданного корпуса документов. Обратите внимание, что созданные векторы слов имеют более 6600 измерений. Однако большинство компонент этих векторов имеют нулевое значение, поскольку большинство слов не встречаются в одном слогане.

Анализ размерности и предварительная обработка

В этом упражнении нам предоставлен `lem_corpus`, содержащий предварительно обработанные версии слоганов фильмов из предыдущего упражнения. Другими словами, слоганы были переведены в строчные буквы и лемматизированы, а стоп-слова удалены.

Наша задача — сгенерировать представление `bow_lem_matrix` для этих лемматизированных слоганов и сравнить его форму с формой `bow_matrix`, полученной в предыдущем упражнении.

In [36]:
```
nlp = spacy.load('en_core_web_sm')
stopwords = spacy.lang.en.stop_words.STOP_WORDS
```

In [37]:
```
lem_corpus = corpus.apply(
    lambda row: ' '.join([t.lemma_ for t in nlp(row)
                          if t.lemma_ not in stopwords
                          and t.lemma_.isalpha()])
)
```

In [38]:
```
lem_corpus
```

Out[38]:
```
1                              roll dice unleash excitement
2                                     yell fight ready love
3                              friend people let let forget
4                                world normal surprise life
5                                    los angeles crime saga
                                ...
9091                           kingsglaive final fantasy xv
9093                         happen vegas stay vegas happen
9095    decorate officer devoted family man defend hon...
9097                               god incarnate city doom
9098                                       band know story
Name: tagline, Length: 7033, dtype: object
```

In [39]:
```
# Create CountVectorizer object
vectorizer = CountVectorizer()

# Generate of word vectors
bow_lem_matrix = vectorizer.fit_transform(lem_corpus)

# Print the shape of how_lem_matrix
print(bow_lem_matrix.shape)
```

```
(7033, 4946)
```

Сопоставление индексов объектов с их названиями

Объекты класса `CountVectorizer` не обязательно индексируют словарь в алфавитном порядке. В этом упражнении мы научимся сопоставлять каждый индекс объекта с соответствующим ему названием объекта из словаря.

In [40]:
```
sentences = ['The lion is the king of the jungle',
             'Lions have lifespans of a decade',
             'The lion is an endangered species']
```

In [41]:
```
# Create CountVectorizer object
vectorizer = CountVectorizer()

# Generate matrix of word vectors
bow_matrix = vectorizer.fit_transform(sentences)

# Convert bow_matrix into a DataFrame
bow_df = pd.DataFrame(bow_matrix.toarray())
bow_df
```

Out[41]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	1	1	1	0	1	0	1	0	3
1	0	1	0	1	0	0	0	1	0	1	1	0	0
2	1	0	1	0	1	0	0	0	1	0	0	1	1

In [42]:
```
# Map the column names to vocabulary
bow_df.columns = vectorizer.get_feature_names_out()

# Print bow_df
bow_df
```

Out[42]:

	an	decade	endangered	have	is	jungle	king	lifespans	lion	lions	of	species	the
0	0	0	0	0	1	1	1	0	1	0	1	0	3
1	0	1	0	1	0	0	0	1	0	1	1	0	0
2	1	0	1	0	1	0	0	0	1	0	0	1	1

Обратите внимание, что названия столбцов относятся к токену, частота которого регистрируется. Следовательно, поскольку имя первого столбца — `'an'`, первый признак представляет собой количество употреблений слова `'an'` в конкретном предложении. `get_feature_names_out()` по сути даёт нам список, представляющий сопоставление индексов признаков с именем признака в словаре.

Построение моделей n-грамм

- Недостатки BoW
  - Пример
    - `The movie was good and not boring` -> положительный
    - `The movie was not good and boring` -> отрицательный
  - Одинаковое представление BoW!
  - Теряется контекст слов.
  - Тональность зависит от позиции `not`
- n-граммы
  - Непрерывная последовательность из n элементов (или слов) в данном документе.
  - Биграммы / Триграммы
- Недостатки n-грамм
  - Увеличение размерности, возникает проклятие размерности
  - n-граммы более высокого порядка встречаются редко

Модели n-грамм для слоганов фильмов

В этом упражнении нам предоставлен корпус из более чем 9000 слоганов фильмов. Наша задача — сгенерировать модели n-грамм для этих данных с $n = 1$, $n = 2$ и $n = 3$ и определить количество признаков для каждой модели.

Затем мы сравним количество признаков, сгенерированных для каждой модели.

In [43]:
```
# Generate n-grams upto n=1
vectorizer_ng1 = CountVectorizer(ngram_range=(1, 1))
ng1 = vectorizer_ng1.fit_transform(corpus)

# Generate n-grams upto n=2
vectorizer_ng2 = CountVectorizer(ngram_range=(1, 2))
ng2 = vectorizer_ng2.fit_transform(corpus)

# Generate n-grams upto n=3
vectorizer_ng3 = CountVectorizer(ngram_range=(1, 3))
ng3 = vectorizer_ng3.fit_transform(corpus)

# Print the number of features for each model
print("ng1, ng2 and ng3 have %i, %i and %i features respectively" %
      (ng1.shape[1], ng2.shape[1], ng3.shape[1]))
```

```
ng1, ng2 and ng3 have 6614, 37100 and 76881 features respectively
```

Мы рассмотрели, как генерировать модели n-грамм, содержащие n-граммы более высокого порядка. Обратите внимание, что у `ng2` более 37 000 признаков, тогда как у `ng3` — более 76 000. Это значительно больше, чем 6000 измерений, полученных для `ng1`. С увеличением диапазона n-грамм растёт и количество признаков, что приводит к увеличению вычислительных затрат и проблеме, известной как «проклятие размерности».

TF-IDF и оценки сходства

Построение векторов документов tf-idf

- Моделирование n-грамм
  - Вес измерения зависит от частоты слова, соответствующего измерению
- Приложения
  - Автоматическое определение стоп-слов
  - Поиск
  - Рекомендательные системы
  - Повышение эффективности предиктивного моделирования в некоторых случаях
- Term frequency-inverse document frequency (tf-idf)
  - Пропорциональная зависимость от частоты термина
  - Обратная зависимость от количества документов, в которых термин встречается
  - Математическая формула

$$w_{i,j} = \text{tf}_{i,j} \cdot \log(\frac{N}{\text{df}_i})$$

    - $w_{i,j} \rightarrow$ вес термина i в документе j
    - $\text{tf}_{i,j} \rightarrow$ частота термина i в документе j
    - $N \rightarrow$ количество документов в корпусе
    - $\text{df}_i \rightarrow$ количество документов, содержащих термин i

Векторы tf-idf для выступлений на конференциях TED

В этом упражнении нам предоставлен корпус `ted`, содержащий стенограммы 500 выступлений на TED. Наша задача — сгенерировать векторы tf-idf для этих выступлений.

Далее мы будем использовать эти векторы для генерации рекомендаций похожих выступлений на основе стенограммы.

In [44]:
```
df = pd.read_csv('./ted.csv')
df.head()
```

Out[44]:

	transcript	url
0	We're going to talk — my — a new lecture, just...	https://www.ted.com/talks/al_seckel_says_our_b...
1	This is a representation of your brain, and yo...	https://www.ted.com/talks/aaron_o_connell_maki...
2	It's a great honor today to share with you The...	https://www.ted.com/talks/carter_emmart_demos_...
3	My passions are music, technology and making t...	https://www.ted.com/talks/jared_ficklin_new_wa...
4	It used to be that if you wanted to get a comp...	https://www.ted.com/talks/jeremy_howard_the_wo...

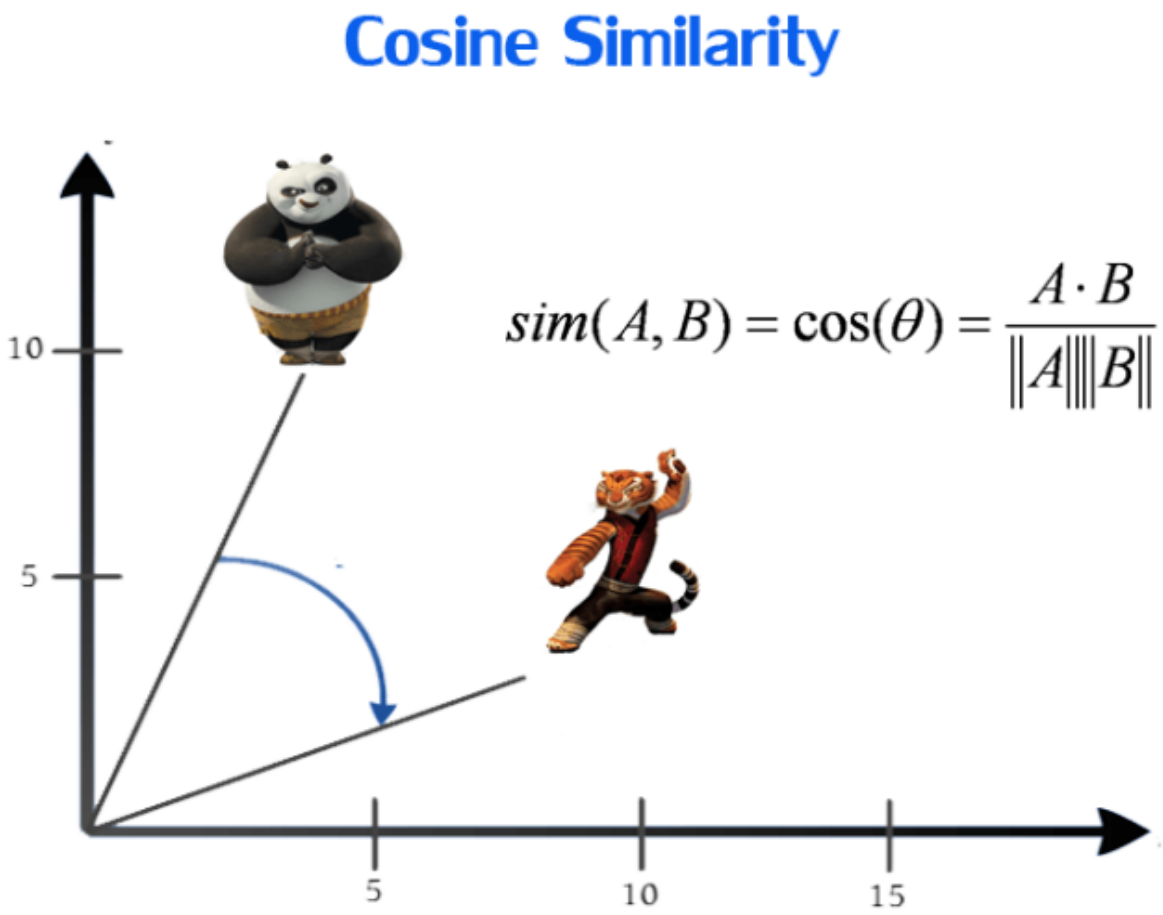
In [45]:
```
ted = df['transcript']
```

In [46]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Create TfidfVectorizer object
vectorizer = TfidfVectorizer()

# Generate matrix of word vectors
tfidf_matrix = vectorizer.fit_transform(ted)

# Print the shape of tfidf_matrix
print(tfidf_matrix.shape)
```

```
(500, 29158)
```

Мы рассмотрели, как генерировать векторы tf-idf для заданного корпуса текстов. Вы можете использовать эти векторы для предиктивного моделирования, как мы это делали с помощью `CountVectorizer` . Далее мы рассмотрим ещё одно чрезвычайно полезное применение векторизованных документов: генерацию рекомендаций.

Косинусное сходство



- Скалярное произведение
  - Рассмотрим два вектора:

$$V = (v_1, v_2, \dots, v_n), W = (w_1, w_2, \dots, w_n)$$

  - Тогда скалярное произведение V и W равно:

$$V \cdot W = (v_1 \times w_1) + (v_2 \times w_2) + \cdots + (v_n \times w_n)$$

- Величина вектора
  - Для любого вектора:

$$V = (v_1, v_2, \dots, v_n)$$

  - Величина определяется как:

$$\|V\| = \sqrt{(v_1)^2 + (v_2)^2 + \cdots + (v_n)^2}$$

- Свойства косинусного сходства:
  - Значение от -1 до 1
  - В естественно-языковом анализе значение от 0 (нет сходства) и 1 (одинаковый)
  - Устойчив к длине документа

Вычисление скалярного произведения

В этом упражнении мы научимся вычислять скалярное произведение двух векторов, $A = (1, 3)$ и $B = (-2, 2)$, используя библиотеку `numpy`. А именно, мы воспользуемся функцией `np.dot()` для вычисления скалярного произведения двух массивов numpy.

In [47]:
```
# Initialize numpy vectors
A = np.array([1, 3])
B = np.array([-2, 2])

# Compute dot product
dot_prod = np.dot(A, B)

# Print dot product
print(dot_prod)
```

```
4
```

Матрица косинусного сходства корпуса

В этом упражнении нам дан корпус, представляющий собой список из пяти предложений. Нам необходимо вычислить матрицу косинусного сходства, содержащую парную оценку косинусного сходства для каждой пары предложений (векторизованную с помощью tf-idf).

Помните, что значение, соответствующее i-й строке и j-му столбцу матрицы сходства, обозначает оценку сходства для i-го и j-го векторов.

In [48]:
```
= ['The sun is the largest celestial body in the solar system',
   'The solar system consists of the sun and eight revolving planet
   'Ra was the Egyptian Sun God',
   'The Pyramids were the pinnacle of Egyptian architecture',
   'The quick brown fox jumps over the lazy dog']
```

In [49]:
```
from sklearn.metrics.pairwise import cosine_similarity

# Initialize an instance of tf-idf Vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Generate the tf-idf vectors for the corpus
tfidf_matrix = tfidf_vectorizer.fit_transform(corpus)

# compute and print the cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
print(cosine_sim)
```

```
[[1.         0.36413198 0.18314713 0.18435251 0.16336438]
 [0.36413198 1.         0.15054075 0.21704584 0.11203887]
 [0.18314713 0.15054075 1.         0.21318602 0.07763512]
 [0.18435251 0.21704584 0.21318602 1.         0.12960089]
 [0.16336438 0.11203887 0.07763512 0.12960089 1.        ]]
```

Вычисление матрицы косинусного сходства лежит в основе многих практических систем, таких как рекомендательные системы. Из нашей матрицы сходства видно, что первое и второе предложения наиболее похожи. Кроме того, пятое предложение имеет в среднем самые низкие парные косинусные оценки. Это интуитивно понятно, поскольку оно содержит сущности, которых нет в других предложениях.

Создание рекомендательного алгоритма на основе сюжетной линии

- Шаги
  1. Предварительная обработка текста
  2. Генерация векторов tf-idf
  3. Генерация матрицы косинусного сходства
- Рекомендательная функция
  1. Принять название фильма, матрицу косинусного сходства и ряд индексов в качестве аргументов
  2. Извлечь парные оценки косинусного сходства для фильма
  3. Сортировка оценок по убыванию
  4. Вывести названия, соответствующие наивысшим оценкам
  5. Игнорировать наивысшую оценку сходства (из 1)

Сравнение linear_kernel и cosine_similarity

В этом упражнении нам дана матрица `tfidf_matrix` , содержащая векторы tf-idf тысячи документов. Наша задача — сгенерировать матрицу косинусного сходства для этих векторов сначала с помощью `cosine_similarity` , а затем с помощью `linear_kernel` .

Затем мы сравним время вычисления обеих функций.

In [50]:
```
import time

# Record start time
start = time.time()

# Compute cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Print cosine similarity matrix
print(cosine_sim)

# Print time taken
print("Time taken: %s seconds" % (time.time() - start))
```

```
[[1.         0.36413198 0.18314713 0.18435251 0.16336438]
 [0.36413198 1.         0.15054075 0.21704584 0.11203887]
 [0.18314713 0.15054075 1.         0.21318602 0.07763512]
 [0.18435251 0.21704584 0.21318602 1.         0.12960089]
 [0.16336438 0.11203887 0.07763512 0.12960089 1.        ]]
Time taken: 0.0004107952117919922 seconds
```

In [51]:
```
from sklearn.metrics.pairwise import linear_kernel

# Record start time
start = time.time()

# Compute cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Print cosine similarity matrix
print(cosine_sim)

# Print time taken
print("Time taken: %s seconds" % (time.time() - start))
```

```
[[1.         0.36413198 0.18314713 0.18435251 0.16336438]
 [0.36413198 1.         0.15054075 0.21704584 0.11203887]
 [0.18314713 0.15054075 1.         0.21318602 0.07763512]
 [0.18435251 0.21704584 0.21318602 1.         0.12960089]
 [0.16336438 0.11203887 0.07763512 0.12960089 1.        ]]
Time taken: 0.0003390312194824219 seconds
```

Обратите внимание, что `linear_kernel` и `cosine_similarity` дали одинаковый результат. Однако `linear_kernel` выполнился быстрее. При работе с очень большими объёмами данных и представлении векторов в формате tf-idf рекомендуется использовать `linear_kernel` по умолчанию для повышения производительности. (ПРИМЕЧАНИЕ: Если вы видите, что `linear_kernel` занимает больше времени, это связано с тем, что набор данных, с которым мы работаем, чрезвычайно мал, а модуль времени Python не способен точно учесть такую незначительную разницу во времени.)

Рекомендательная функция

В этом упражнении мы создадим рекомендательную функцию `get_recommendations()`, которая принимает в качестве аргументов название фильма, матрицу косинусного сходства, а также название фильма и сопоставление индексов и выводит список из 10 фильмов, наиболее похожих на исходный (исключая сам фильм).

Нам предоставлен набор метаданных, состоящий из названий фильмов и их обзоров. Заголовок этого набора данных выведен на консоль.

In [52]:
```
metadata = pd.read_csv('./movie_metadata.csv').dropna()
metadata.head()
```

Out[52]:

	Unnamed: 0	id	title	overview	tagline
0	0	49026	The Dark Knight Rises	Following the death of District Attorney Harve...	The Legend Ends
1	1	414	Batman Forever	The Dark Knight of Gotham City confronts a das...	Courage now, truth always...
2	2	268	Batman	The Dark Knight of Gotham City begins his war ...	Have you ever danced with the devil in the pal...
3	3	364	Batman Returns	Having defeated the Joker, Batman now faces th...	The Bat, the Cat, the Penguin.
4	4	415	Batman & Robin	Along with crime-fighting partner Robin and ne...	Strength. Courage. Honor. And loyalty.

In [53]:
```
# Generate mapping between titles and index
indices = pd.Series(
    metadata.index, index=metadata['title']
).drop_duplicates()

def get_recommendations(title, cosine_sim, indices):
    # Get the index of the movie that matches the title
    idx = indices[title]
    # Get the pairwsie similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=Tru
    # Get the scores for 10 most similar movies
    sim_scores = sim_scores[1:11]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]
    # Return the top 10 most similar movies
    return metadata['title'].iloc[movie_indices]
```

Рекомендательный движок фильмов

В этом упражнении мы создадим рекомендательный движок, который будет предлагать фильмы на основе сходства сюжетных линий. Нам дана функция `get_recommendations()`, которая принимает в качестве аргументов название фильма, матрицу сходства и ряд индексов и выводит список наиболее похожих фильмов.

Нам также дан объект `movie_plots`, содержащий сюжетные линии нескольких фильмов. Наша задача — сгенерировать матрицу косинусного сходства для векторов tf-idf этих сюжетов.

Мы проверим эффективность движка, сгенерировав рекомендации для фильма — «Тёмный рыцарь: Возрождение легенды».

In [54]:
```
movie_plots = metadata['overview']
movie_plots
```

Out[54]:
```
0       Following the death of District Attorney Harve...
1       The Dark Knight of Gotham City confronts a das...
2       The Dark Knight of Gotham City begins his war ...
3       Having defeated the Joker, Batman now faces th...
4       Along with crime-fighting partner Robin and ne...
                              ...
1002    When a Sumatran rat-monkey bites Lionel Cosgro...
1003    Robert Gould Shaw leads the US Civil War's fir...
1005    The life of a divorced television writer datin...
1006    Set in 1929, a political boss and his advisor ...
1007    At an elite, old-fashioned boarding school in ...
Name: overview, Length: 820, dtype: object
```

In [55]:
```
# Initialize the TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')
```

In [56]:
```
# Construct the TF-IDF matrix
tfidf_matrix = tfidf.fit_transform(movie_plots)
tfidf_matrix
```

Out[56]:
```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 21875 stored elements and shape (820, 8195)>
```

In [57]:
```
# Generate the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim
```

Out[57]:
```
array([[1.        , 0.27337431, 0.22950585, ..., 0.        , 0.
,
        0.00784944],
       [0.27337431, 1.        , 0.12453176, ..., 0.        , 0.
,
        0.00766224],
       [0.22950585, 0.12453176, 1.        , ..., 0.        , 0.
,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.
,
        0.        ],
       [0.00784944, 0.00766224, 0.        , ..., 0.        , 0.
,
        1.        ]], shape=(820, 820))
```

In [58]:
```
# Generate recommendations
print(get_recommendations("The Dark Knight Rises", cosine_sim, indi
```

```
1                              Batman Forever
2                                      Batman
8                  Batman: Under the Red Hood
3                              Batman Returns
9                            Batman: Year One
10    Batman: The Dark Knight Returns, Part 1
11    Batman: The Dark Knight Returns, Part 2
5                Batman: Mask of the Phantasm
7                               Batman Begins
4                              Batman & Robin
Name: title, dtype: object
```

Обратите внимание, как рекомендательная система правильно определяет «Тёмный рыцарь: Возрождение легенды» как фильм о Бэтмене и в результате рекомендует другие фильмы о нём. Эта система, конечно, очень примитивна, и есть множество способов её улучшить. Один из способов — учитывать актёрский состав, съёмочную группу и жанр, а также сюжет при формировании рекомендаций.

Рекомендательная система выступлений на TED

В этом упражнении мы построим рекомендательную систему, которая будет предлагать выступления на TED на основе их стенограмм. Нам дана функция `get_recommendations()`, которая принимает в качестве аргументов название выступления, матрицу сходства и объект `indexes` и выводит список наиболее похожих выступлений.

Нам также дан объект `transcripts`, содержащий стенограммы около 500 выступлений на TED. Наша задача — сгенерировать матрицу косинусного сходства для векторов tf-idf стенограмм выступлений.

Затем мы сгенерируем рекомендации на основе выступления бразильского предпринимателя Бела Песке под названием «5 способов убить свои мечты».

In [59]:
```
ted = pd.read_csv('./ted_clean.csv', index_col=0)
ted.head()
```

Out[59]:

Unnamed: 0	Unnamed: 0.1	title	url	transcri
0	1407	10 top time-saving tech tips	https://www.ted.com/talks/david_pogue_10_top_t...	I've notice somethir interestir about socie
1	1524	Who am I? Think again	https://www.ted.com/talks/hetain_patel_who_am_...	Hetain Pat (Chinese)Yu Rau: Hi, I' He
2	2393	"Awoo"	https://www.ted.com/talks/sofi_tukker_awoo\n	(Music)Soph Hawley-Wel OK, you dor have
3	2313	What I learned from 2,000 obituaries	https://www.ted.com/talks/lux_narayan_what_i_l...	Joseph Kell used to jo around tl Stanford
4	1633	Why giving away our wealth has been the most s...	https://www.ted.com/talks/bill_and_melinda_gat...	Chr Anderso So, this is a interview wi

In [60]:
```
def get_recommendations(title, cosine_sim, indices):
    # Get the index of the movie that matches the title
    idx = indices[title]
    # Get the pairwsie similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=Tru
    # Get the scores for 10 most similar movies
    sim_scores = sim_scores[1:11]
    # Get the movie indices
    talk_indices = [i[0] for i in sim_scores]
    # Return the top 10 most similar movies
    return ted['title'].iloc[talk_indices]
```

In [61]:
```
# Generate mapping between titles and index
indices = pd.Series(ted.index, index=ted['title']).drop_duplicates(
transcripts = ted['transcript']
```

In [62]:
```
# Initialize the TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')

# Construct the TF-IDF matrix
tfidf_matrix = tfidf.fit_transform(transcripts)

# Generate the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Generate recommendations
print(get_recommendations(
    '5 ways to kill your dreams', cosine_sim, indices
))
```

```
Unnamed: 0
453             Success is a continuous journey
157                       Why we do what we do
494                  How to find work you love
149          My journey into movies that matter
447                       One Laptop per Child
230             How to get your ideas to spread
497          Plug into your hard-wired happiness
495    Why you will fail to have a great career
179             Be suspicious of simple stories
53                         To upgrade is human
Name: title, dtype: object
```

Мы успешно создали систему рекомендаций для выступлений TED. Эта система работает на удивление хорошо, несмотря на то, что обучена лишь на небольшой выборке выступлений на TED.

За пределами n-грамм: векторные представления слов

- Векторные представления слов (word embeddings)
  - Отображение слов в n-мерном векторном пространстве
  - Создаются с использованием глубокого обучения и огромных объёмов данных
  - Учитываются степени схожести двух слов
  - Используются для обнаружения синонимов и антонимов
  - Учитываются сложные взаимосвязи
  - Зависят от модели spacy, не зависят от используемого набора данных

> Примечание: перед использованием встраивания слов через `spacy` необходимо загрузить модель `en_core_web_lg` (`python -m spacy download en_core_web_lg`)

In [63]:
```
nlp = spacy.load('en_core_web_lg')
```

Генерация векторов слов

Сгенерируем парные оценки сходства всех слов в предложении.

In [64]:
```
sent = 'I like apples and orange'

# Create the doc object
doc = nlp(sent)

# Compute pairwise similarity scores
for token1 in doc:
    for token2 in doc:
        print(token1.text, token2.text, token1.similarity(token2))
```

```
I I 1.0
I like 0.5554912686347961
I apples 0.20442721247673035
I and 0.3160786032676697
I orange 0.3033278286457062
like I 0.5554912686347961
like like 1.0
like apples 0.32987141609191895
like and 0.5267485976219177
like orange 0.3551868498325348
apples I 0.20442721247673035
apples like 0.32987141609191895
apples apples 1.0
apples and 0.24097736179828644
apples orange 0.5123849511146545
and I 0.3160786032676697
and like 0.5267485976219177
and apples 0.24097736179828644
and and 1.0
and orange 0.2545080780982971
orange I 0.3033278286457062
orange like 0.3551868498325348
orange apples 0.5123849511146545
orange and 0.2545080780982971
orange orange 1.0
```

Обратите внимание, что слова «яблоки» и «апельсины» имеют самый высокий показатель парного сходства. Это ожидаемо, поскольку оба слова обозначают фрукты и связаны друг с другом сильнее, чем любая другая пара слов.

Вычисление сходства песен Pink Floyd

В этом последнем упражнении нам даны тексты трёх песен британской группы Pink Floyd: «High Hopes», «Hey You» и «Mother». Тексты этих песен представлены в виде файлов «hopes», «hey» и «mother» соответственно.

Наша задача — вычислить парное сходство между «mother» и «hopes», а также «mother» и «hey».

In [65]:
```
with open('./mother.txt', 'r') as f:
    mother = f.read()

with open('./hopes.txt', 'r') as f:
    hopes = f.read()

with open('./hey.txt', 'r') as f:
    hey = f.read()
```

In [66]:
```
# Create Doc objects
mother_doc = nlp(mother)
hopes_doc = nlp(hopes)
hey_doc = nlp(hey)

# Print similarity between mother and hopes
print(mother_doc.similarity(hopes_doc))

# Print similarity between mother and hey
print(mother_doc.similarity(hey_doc))
```

```
0.8653563261032104
0.9595261812210083
```

Обратите внимание, что песни «Mother» и «Hey You» имеют коэффициент схожести 0,96, тогда как песни «Mother» и «High Hopes» — всего 0,86. Вероятно, это связано с тем, что «Mother» и «Hey You» — песни с одного альбома «The Wall» и были написаны Роджером Уотерсом. С другой стороны, песня «High Hopes» вошла в альбом «Division Bell» со словами Дэвида Гилмора и его жены Пенни Сэмсон.

Задание на лабораторную работу

В материалах к ЛР прилагаются файлы с текстами 10 коротких рассказов (на английском языке), пронумерованных от 0 до 9. В соответствии тремя последними различными цифрами номера Вашего студенческого билета выберите три различных файла для обработки и выполните следующие работы:

1. Для каждого из текстов подсчитайте и выведите кол-во символов, кол-во слов и кол-во предложений в текстах. Определите и выведите на экран три самых длинных слова в текстах. Для каждого из текстов подсчитайте и выведите минимальное, максимальное и среднее кол-во слов в предложениях текста.
2. Оцените и выведите, какой из текстов более читабелен по критерию индекса лёгкости чтения по шкале Флеша и индексу туманности Ганнинга.
3. Для каждого из текстов выполните токенизацию и определите пять наиболее часто встречающихся токена и частоты их повторения.
4. Для каждого из текстов выполните лемматизацию текстов и визуализируйте при помощи гистограммы распределение длин лемм в текстах.
5. Для каждого из текстов удалите стоп-слова, знаки препинания и неалфавитные символы и определите, какое кол-во токенов было.
6. Для каждого из текстов выполните разметку частей речи, определите и выведите кол-во токенов каждого типа. Визуализируйте кол-во токенов каждого типа в текстах в виде столбчатых диаграмм.
7. Для каждого из текстов найдите именованные сущности, выведите и подсчитайте их кол-во для каждого типа.
8. Для имеющихся текстов постройте векторы tf-idf и выведите форму (shape) построенной матрицы.
9. Вычислите попарное сходство текстов при помощи косинусного расстояния и определите два наиболее похожих текста из трех.

In [ ]: