



گزارش پروژه پایانی – فاز اول

استخراج رابطه در زبان فارسی با بررسی وابستگی جهانی Seraji و PerDT

مبانی پردازش زبان و گفتار

استاد: دکتر بهروز مینایی بیدگلی

دانیال بازمانده – محمدحسین کریمیان

پاسخ سوال اول تئوری:

همانطور که در صورت سوال عنوان شد، فهرست ابزارها و کتابخانه‌ها برای این کاربرد طولانی است. از این رو، به طور اختصار به سه مورد از آنها اشاره می‌کنیم:

- کتابخانه‌ی NLTK: این کتابخانه که نام آن مخفف عبارت Natural Language Toolkit می‌باشد، تقریباً محبوب‌ترین کتابخانه‌ی پایتون برای کار در موزه‌ی متن‌کاوی می‌باشد. اگرچه دستیابی به تسلط کامل بر این کتابخانه فرایندی زمانبر است، اما این کتابخانه مجموعه کامل و جامعی از ابزارها را برای عملیات مربوط به متن در اختیار کاربران قرار می‌دهد و یک نقطه شروع خوب برای مبتدیان است. از دیگر ویژگی‌های خوب این کتابخانه می‌توان به متن‌باز (Open-source) بودن آن اشاره کرد. از جمله امکاناتی که این ابزار در اختیار کاربران قرار می‌دهد، می‌توان به طبقه‌بندی متن، برپسب گذاری بفرشی از گفتار، استخراج موجودیت، نشانه گذاری، تجزیه و استدلال معنایی اشاره کرد.
- ابزار MonkeyLearn: این ابزار یک پلتفرم کاربر پسند و مجهز است که کمک می‌کند بینش‌ها و ارزیابی‌های دقیق و ارزشمندی را از داده‌های متنی به دست آوریم. به عنوان شروع اگر یکی از مدل‌های از پیش آموزش‌دیده را انتخاب کنیم، می‌توانیم توانایی و نحوه کار با این ابزار در وظایفی از جمله تحلیل احساسات، طبقه‌بندی موضوع یا استخراج کلمات کلیدی را متوجه شویم. ویژگی منمصر به فرد این نرم‌افزار این است که پس از آموزش مدل‌ها می‌توان بدون نیاز به مهارت‌های کدنویسی آن را به برنامه‌هایی از جمله Google Sheets, Zendesk, Excel متصل کرد. همچنین در صورت برقراری ارتباط با این ابزار، API های مربوطه در تمامی زبان‌های برنامه‌نویسی موجود می‌باشد.
- کتابخانه‌ی SpaCy: این کتابخانه یکی از جدیدترین کتابخانه‌های موجود در زبان پایتون برای این کاربرد است. از ویژگی‌های خوب آن می‌توان به سرعت پردازشی بالا، یادگیری آسان، مستندسازی خوب و پشتیبانی از حجم زیاد داده‌ها اشاره کرد. همچنین این کتابخانه مجموعه‌ای از مدل‌های پردازش متن از پیش آموزش‌دیده شده را داراست که باعث سهولت کار ما خواهد شد. یکی از برتری‌های این ابزار بر خلاف NLTK این است که در هر کار و مسئله‌ای بهترین ابزار (و نه تمام گزینه‌های موجود) را ارائه می‌دهد و پیشنهاد می‌کند. برای آماده‌سازی یک متن برای یادگیری عمیق نیز بسیار گزینه خوبی خواهد بود. از دیگر ویژگی‌های آن می‌توان به متن‌باز بودن، مطابقت کامل برای مقایسه پروفایل مشتری و برقرارداری از امکان کلمه‌برداری است.

پاسخ سوال دوم تئوری:

هریک از عبارات منظم خواسته شده به فرمت زیر خواهند بود:

- آ: شماره تلفن با حداکثر ۱۶ رقم

`(09 | 00989 | \+989) [0-9] {9-14}`

- ب: تاریخها با فرمت dd-mm-yyyy

`(0[1-9] | [1-2] [0-9] | 30) - (0[1-9] | 1[0-2]) - (000[1-9] | 00[10-99] | 0[100-999] | [1000-2022])`

- ج: آدرسها (URL) با پسوندهای ir/org

`(http(s)?:\//(\www\.)?([a-zA-Z0-9@:%._\+~#=]+\.$)+(org|ir)`

- د: پلاک ماشینها به فرمت 54M235IR44

`[10-99] [A-Z] [100-999] IR [10-99]`

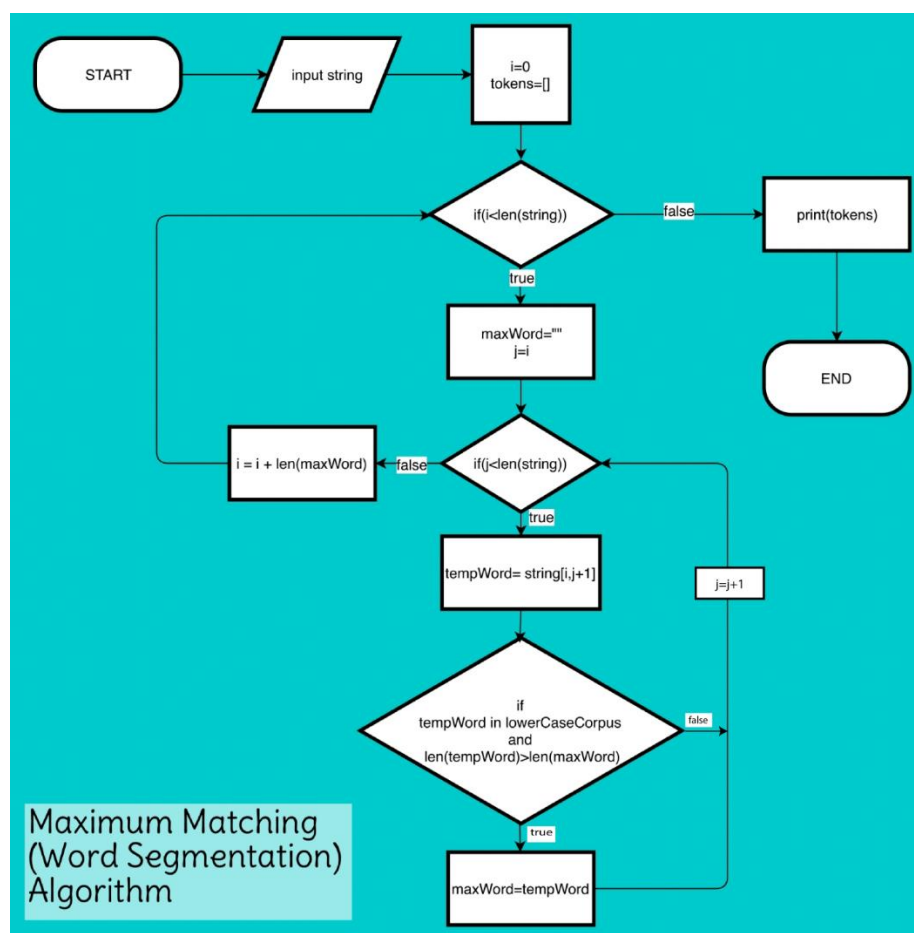
پاسخ سوال سوم تئوری:

این الگوریتم یکی از مهم‌ترین روش‌های *word segmentation* است که کاربرد آن در این است که اگر به عنوان ورودی یک عبارت که تمامی حروف آن کنار همدیگر و بدون فاصله آمده باشند را به این الگوریتم بدهیم، عبارت را به کلمات بامعنی تشکیل‌دهنده آن می‌شکند و در انتها لیستی از این کلمات را به عنوان خروجی برمی‌گرداند. به طور کلی مراحل اصلی اجرای این الگوریتم سه مرحله به صورت زیر است:

- مرحله اول: ابتدا از اولین کاراکتر عبارت داده شده شروع می‌کنیم.
- مرحله دوم: به دنبال بزرگترین عبارتی که از با شروع از این کاراکتر (کاراکتر اول) تشکیل کلمه‌ای معنادار می‌دهد، می‌گردیم.
- مرحله سوم: در صورتی که کلمه‌ی موردنظر پیدا شود، محدوده (*boundary*) جدید را ست می‌کنیم و اشاره‌گر را جلوتر می‌بریم و از آن کاراکتر به بعد عملیات جستجو را انجام می‌دهیم و در صورت عدم پیدا شدن کلمه معنادار، خود کاراکتر را به عنوان یک کلمه در نظر می‌گیریم.

به عنوان نمونه، فرض کنید می‌فواهیم این الگوریتم را روی کلمه‌ی «*thecatinthehat*» اجرا کنیم. الگوریتم به این صورت عمل می‌کند که ابتدا اشاره‌گر را روی *t* می‌گذارد و سعی در پیدا کردن طولانی‌ترین کلمه معنادار دارد. پس از اینکه کلمه‌ی *the* را پیدا کرد، اشاره‌گر جلو می‌رود و از کاراکتر *c* به جستجو می‌پردازد که نتیجه *cat* می‌باشد. در انتها پس از اینکه اشاره‌گر به انتهای رشته رسید، لیست ['*the*', '*cat*', '*in*', '*the*', '*hat*'] به عنوان خروجی بازگردانده می‌شود.

فلوچارت الگوریتم فوق به شکل زیر است:



پاسخ سوال اول عملی:

برای حل این سوال از کتابخانه‌ی پرکاربرد مربوط به کار با regExp ها در زبان پایتون (re) استفاده کرده‌ایم. تمامی ورودی‌ها را خط به خط در یک فایل متنی (txt) ذخیره کرده‌ایم و سپس در یک حلقه با بررسی عبارت منظم نوشته‌شده بر روی تمامی عبارات، match بودن و یا نبودن آن را بررسی کرده‌ایم که همانطور که مشاهده می‌شود، تمامی عبارات پوشش داده شده‌اند و مقادیر True برگردانده شده‌است.

```
1 import re
2
3 input_file = open('Q1-text.txt', 'r')
4 regex = r"((Dr\.|Doctor)\s)?([a-zA-Z]+\.)|[a-zA-Z]+(\s|,)+?"
5
6 for line in input_file.readlines():
7     if re.search(regex, line):
8         print(True)
9     else:
10        print(False)
11
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
D:\University Courses\Term 8\Natural Language Processing\HW> d: && cd "d:\University Courses\Term 8\Natural Language Processing\HW" && python36\python.exe c:\Users\Daniel\.vscode\extensions\ms-python.python-2021.8.1159\bin\python.exe --
D:\University Courses\Term 8\Natural Language Processing\HW> python Q1.py "
```

عبارت منظمی که در این سوال برای بررسی عبارات روی آن در نظر گرفته ایم، به صورت زیر است:

$$((\text{Dr}\backslash. | \text{Doctor}) \backslash s)? ((([\text{a-z A-z}] \backslash.)+) | [\text{a-z A-Z}] +)(\backslash s | ,)?$$

پاسخ سوال دوم عملی:

همانطور که در تصویر زیر مشخص است، ابتدا اقدام به نصب کتابخانه NLTK با استفاده از دستور `pip install nltk` کردیم.

```
(venv) D:\Pycharm Projects\Natural Language Processing\HW1-Practical>pip install nltk
Collecting nltk
  Downloading nltk-3.7-py3-none-any.whl (1.5 MB)
    ----- 1.5/1.5 MB 250.4 kB/s eta 0:00:00
Collecting tqdm
  Downloading tqdm-4.63.1-py2.py3-none-any.whl (76 kB)
    ----- 76.6/76.6 KB 265.4 kB/s eta 0:00:00
Collecting regex>=2021.8.3
  Downloading regex-2022.3.15-cp37-cp37m-win_amd64.whl (273 KB)
    ----- 273.8/273.8 KB 205.6 kB/s eta 0:00:00
Collecting click
  Downloading click-8.1.2-py3-none-any.whl (96 kB)
    ----- 96.6/96.6 KB 240.5 kB/s eta 0:00:00
Collecting joblib
```

مال با استفاده از متدهای `sent_tokenize` و `word_tokenize` از این کتابخانه، متنی را از دافل فایل متنی خوانده و به ترتیب، جملات این متن و کلمات آن را به عنوان فروبی دریافت می‌کنیم.

```

1 from nltk.tokenize import sent_tokenize, word_tokenize
2
3 input_file = open('Q2-text.txt', 'r')
4 text_str = input_file.read()
5
6 print("Sentences Tokenization:")
7 print(sent_tokenize(text_str))
8 print("\n*100")
9 print("Word Tokenization:")
10 print(word_tokenize(text_str))
11

```

Run: Q2

"D:\Pycharm Projects\HW1-Practical\venv\Scripts\python.exe" "D:\Pycharm Projects\Natural Language Processing\HW1-Practical\Q2.py"

Sentences Tokenization:

['Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data.', 'It is seen as a part of artificial intelligence.', 'Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.', 'Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.', 'A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning.', 'The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning.', 'Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning.', 'Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.', 'In its application across business problems, machine learning is also referred to as predictive analytics.', 'Learning algorithms work on the basis that strategies, algorithms, and inferences that worked well in the past are likely to continue working well in the future.', 'These inferences can be obvious, such as "since the sun rose every morning for the last 10,000 days, it will probably rise tomorrow morning as well".', 'They can be nuanced, such as "XX% of families have geographically separate species with color variants, so there is a Y% chance that undiscovered black swans exist".', 'Machine learning programs can perform tasks without being explicitly programmed to do so.', 'It involves computers learning from data provided so that they carry out certain tasks.', 'For simple tasks assigned to computers, it is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; on the computer's part, no learning is needed.', 'For more advanced tasks, it can be challenging for a human to manually create the needed algorithms.', 'In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step.', 'The discipline of machine learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available.', 'In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid.', 'This can then be used as training data for the computer to improve the algorithm(s) it uses to determine correct answers.', 'For example, to train a system for the task of digital character recognition, the MNIST dataset of handwritten digits has often been used.']

Word Tokenization:

['Machine', 'learning', '(', 'ML', ')', 'is', 'the', 'study', 'of', 'computer', 'algorithms', 'that', 'can', 'improve', 'automatically', 'through', 'experience', 'and', 'by', 'the', 'use', 'of', 'data', 'It', 'is', 'seen', 'as', 'a', 'part', 'of', 'artificial', 'intelligence', 'Machine', 'learning', 'algorithms', 'build', 'a', 'model', 'based', 'on', 'sample', 'data', 'known', 'as', 'training', 'data', 'in', 'order', 'to', 'make', 'predictions', 'or', 'decisions', 'without', 'being', 'explicitly', 'programmed', 'to', 'do', 'so', 'Machine', 'learning', 'algorithms', 'are', 'used', 'in', 'a', 'wide', 'variety', 'of', 'applications', 'such', 'as', 'in', 'medicine', 'email', 'filtering', 'speech', 'recognition', 'and', 'computer', 'vision', 'where', 'it', 'is', 'difficult', 'or', 'unfeasible', 'to', 'develop', 'conventional', 'algorithms', 'to', 'perform', 'the', 'needed', 'tasks', 'A', 'subset', 'of', 'machine', 'learning', 'is', 'closely', 'related', 'to', 'computational', 'statistics', 'which', 'focuses', 'on', 'making', 'predictions', 'using', 'computers', 'but', 'not', 'all', 'machine', 'learning', 'is', 'statistical', 'learning', 'The', 'study', 'of', 'mathematical', 'optimization', 'delivers', 'methods', 'theory', 'and', 'application', 'domains', 'to', 'the', 'field', 'of', 'machine', 'learning', 'Data', 'mining', 'is', 'a', 'related', 'field', 'of', 'study', 'focusing', 'on', 'exploratory', 'data', 'analysis', 'through', 'unsupervised', 'learning', 'Some', 'implementations', 'of', 'machine', 'learning', 'use', 'data', 'and', 'neural', 'networks', 'in', 'a', 'way', 'that', 'mimics', 'the', 'working', 'of', 'a', 'biological', 'brain', 'In', 'its', 'application', 'across', 'business', 'problems', 'machine', 'learning', 'is', 'also', 'referred', 'to', 'as', 'predictive', 'analytics', 'Learning', 'algorithms', 'work', 'on', 'the', 'basis', 'that', 'strategies', 'algorithms', 'and', 'inferences', 'that', 'worked', 'well', 'in', 'the', 'past', 'are', 'likely', 'to', 'continue', 'working', 'well', 'in', 'the', 'future', 'These', 'inferences', 'can', 'be', 'obvious', 'such', 'as', 'since', 'the', 'sun', 'rose', 'every', 'morning', 'for', 'the', 'last', '10,000', 'days', 'it', 'will', 'probably', 'rise', 'tomorrow', 'morning', 'as', 'well', 'They', 'can', 'be', 'nuanced', 'such', 'as', 'XX%', 'of', 'families', 'have', 'geographically', 'separate', 'species', 'with', 'color', 'variants', 'so', 'there', 'is', 'a', 'Y%', 'chance', 'that', 'undiscovered', 'black', 'swans', 'exist', 'Machine', 'learning', 'programs', 'can', 'perform', 'tasks', 'without', 'being', 'explicitly', 'programmed', 'to', 'do', 'so', 'It', 'involves', 'computers', 'learning', 'from', 'data', 'provided', 'so', 'that', 'they', 'carry', 'out', 'certain', 'tasks', 'For', 'simple', 'tasks', 'assigned', 'to', 'computers', 'it', 'is', 'possible', 'to', 'program', 'algorithms', 'telling', 'the', 'machine', 'how', 'to', 'execute', 'all', 'steps', 'required', 'to', 'solve', 'the', 'problem', 'at', 'hand', 'on', 'the', 'computer', 's', 'part', 'no', 'learning', 'is', 'needed', 'For', 'more', 'advanced', 'tasks', 'it', 'can', 'be', 'challenging', 'for', 'a', 'human', 'to', 'manually', 'create', 'the', 'needed', 'algorithms', 'In', 'practice', 'it', 'can', 'turn', 'out', 'to', 'be', 'more', 'effective', 'to', 'help', 'the', 'machine', 'develop', 'its', 'own', 'algorithm', 'rather', 'than', 'having', 'human', 'programmers', 'specify', 'every', 'needed', 'step', 'The', 'discipline', 'of', 'machine', 'learning', 'employs', 'various', 'approaches', 'to', 'teach', 'computers', 'to', 'accomplish', 'tasks', 'where', 'no', 'fully', 'satisfactory', 'algorithm', 'is', 'available', 'In', 'cases', 'where', 'vast', 'numbers', 'of', 'potential', 'answers', 'exist', 'one', 'approach', 'is', 'to', 'label', 'some', 'of', 'the', 'correct', 'answers', 'as', 'valid', 'This', 'can', 'then', 'be', 'used', 'as', 'training', 'data', 'for', 'the', 'computer', 'to', 'improve', 'the', 'algorithm', 's', 'it', 'uses', 'to', 'determine', 'correct', 'answers', 'For', 'example', 'to', 'train', 'a', 'system', 'for', 'the', 'task', 'of', 'digital', 'character', 'recognition', 'the', 'MNIST', 'dataset', 'of', 'handwritten', 'digits', 'has', 'often', 'been', 'used', '']

Process finished with exit code 0

```

(venv) D:\Pycharm Projects\Natural Language Processing\HW1-Practical>python Q2.py
Sentences Tokenization:
['Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data.', 'It is seen as a part of artificial intelligence.', 'Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.', 'Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.', 'A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning.', 'The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning.', 'Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning.', 'Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.', 'In its application across business problems, machine learning is also referred to as predictive analytics.', 'Learning algorithms work on the basis that strategies, algorithms, and inferences that worked well in the past are likely to continue working well in the future.', 'These inferences can be obvious, such as "since the sun rose every morning for the last 10,000 days, it will probably rise tomorrow morning as well".', 'They can be nuanced, such as "XX% of families have geographically separate species with color variants, so there is a Y% chance that undiscovered black swans exist".', 'Machine learning programs can perform tasks without being explicitly programmed to do so.', 'It involves computers learning from data provided so that they carry out certain tasks.', 'For simple tasks assigned to computers, it is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; on the computer's part, no learning is needed.', 'For more advanced tasks, it can be challenging for a human to manually create the needed algorithms.', 'In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step.', 'The discipline of machine learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available.', 'In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid.', 'This can then be used as training data for the computer to improve the algorithm(s) it uses to determine correct answers.', 'For example, to train a system for the task of digital character recognition, the MNIST dataset of handwritten digits has often been used.']
Word Tokenization:
['Machine', 'learning', '(', 'ML', ')', 'is', 'the', 'study', 'of', 'computer', 'algorithms', 'that', 'can', 'improve', 'automatically', 'through', 'experience', 'and', 'by', 'the', 'use', 'of', 'data', 'It', 'is', 'seen', 'as', 'a', 'part', 'of', 'artificial', 'intelligence', 'Machine', 'learning', 'algorithms', 'build', 'a', 'model', 'based', 'on', 'sample', 'data', 'known', 'as', 'training', 'data', 'in', 'order', 'to', 'make', 'predictions', 'or', 'decisions', 'without', 'being', 'explicitly', 'programmed', 'to', 'do', 'so', 'Machine', 'learning', 'algorithms', 'are', 'used', 'in', 'a', 'wide', 'variety', 'of', 'applications', 'such', 'as', 'in', 'medicine', 'email', 'filtering', 'speech', 'recognition', 'and', 'computer', 'vision', 'where', 'it', 'is', 'difficult', 'or', 'unfeasible', 'to', 'develop', 'conventional', 'algorithms', 'to', 'perform', 'the', 'needed', 'tasks', 'A', 'subset', 'of', 'machine', 'learning', 'is', 'closely', 'related', 'to', 'computational', 'statistics', 'which', 'focuses', 'on', 'making', 'predictions', 'using', 'computers', 'but', 'not', 'all', 'machine', 'learning', 'is', 'statistical', 'learning', 'The', 'study', 'of', 'mathematical', 'optimization', 'delivers', 'methods', 'theory', 'and', 'application', 'domains', 'to', 'the', 'field', 'of', 'machine', 'learning', 'Data', 'mining', 'is', 'a', 'related', 'field', 'of', 'study', 'focusing', 'on', 'exploratory', 'data', 'analysis', 'through', 'unsupervised', 'learning', 'Some', 'implementations', 'of', 'machine', 'learning', 'use', 'data', 'and', 'neural', 'networks', 'in', 'a', 'way', 'that', 'mimics', 'the', 'working', 'of', 'a', 'biological', 'brain', 'In', 'its', 'application', 'across', 'business', 'problems', 'machine', 'learning', 'is', 'also', 'referred', 'to', 'as', 'predictive', 'analytics', 'Learning', 'algorithms', 'work', 'on', 'the', 'basis', 'that', 'strategies', 'algorithms', 'and', 'inferences', 'that', 'worked', 'well', 'in', 'the', 'past', 'are', 'likely', 'to', 'continue', 'working', 'well', 'in', 'the', 'future', 'These', 'inferences', 'can', 'be', 'obvious', 'such', 'as', 'since', 'the', 'sun', 'rose', 'every', 'morning', 'for', 'the', 'last', '10,000', 'days', 'it', 'will', 'probably', 'rise', 'tomorrow', 'morning', 'as', 'well', 'They', 'can', 'be', 'nuanced', 'such', 'as', 'XX%', 'of', 'families', 'have', 'geographically', 'separate', 'species', 'with', 'color', 'variants', 'so', 'there', 'is', 'a', 'Y%', 'chance', 'that', 'undiscovered', 'black', 'swans', 'exist', 'Machine', 'learning', 'programs', 'can', 'perform', 'tasks', 'without', 'being', 'explicitly', 'programmed', 'to', 'do', 'so', 'It', 'involves', 'computers', 'learning', 'from', 'data', 'provided', 'so', 'that', 'they', 'carry', 'out', 'certain', 'tasks', 'For', 'simple', 'tasks', 'assigned', 'to', 'computers', 'it', 'is', 'possible', 'to', 'program', 'algorithms', 'telling', 'the', 'machine', 'how', 'to', 'execute', 'all', 'steps', 'required', 'to', 'solve', 'the', 'problem', 'at', 'hand', 'on', 'the', 'computer', 's', 'part', 'no', 'learning', 'is', 'needed', 'For', 'more', 'advanced', 'tasks', 'it', 'can', 'be', 'challenging', 'for', 'a', 'human', 'to', 'manually', 'create', 'the', 'needed', 'algorithms', 'In', 'practice', 'it', 'can', 'turn', 'out', 'to', 'be', 'more', 'effective', 'to', 'help', 'the', 'machine', 'develop', 'its', 'own', 'algorithm', 'rather', 'than', 'having', 'human', 'programmers', 'specify', 'every', 'needed', 'step', 'The', 'discipline', 'of', 'machine', 'learning', 'employs', 'various', 'approaches', 'to', 'teach', 'computers', 'to', 'accomplish', 'tasks', 'where', 'no', 'fully', 'satisfactory', 'algorithm', 'is', 'available', 'In', 'cases', 'where', 'vast', 'numbers', 'of', 'potential', 'answers', 'exist', 'one', 'approach', 'is', 'to', 'label', 'some', 'of', 'the', 'correct', 'answers', 'as', 'valid', 'This', 'can', 'then', 'be', 'used', 'as', 'training', 'data', 'for', 'the', 'computer', 'to', 'improve', 'the', 'algorithm', 's', 'it', 'uses', 'to', 'determine', 'correct', 'answers', 'For', 'example', 'to', 'train', 'a', 'system', 'for', 'the', 'task', 'of', 'digital', 'character', 'recognition', 'the', 'MNIST', 'dataset', 'of', 'handwritten', 'digits', 'has', 'often', 'been', 'used', '']

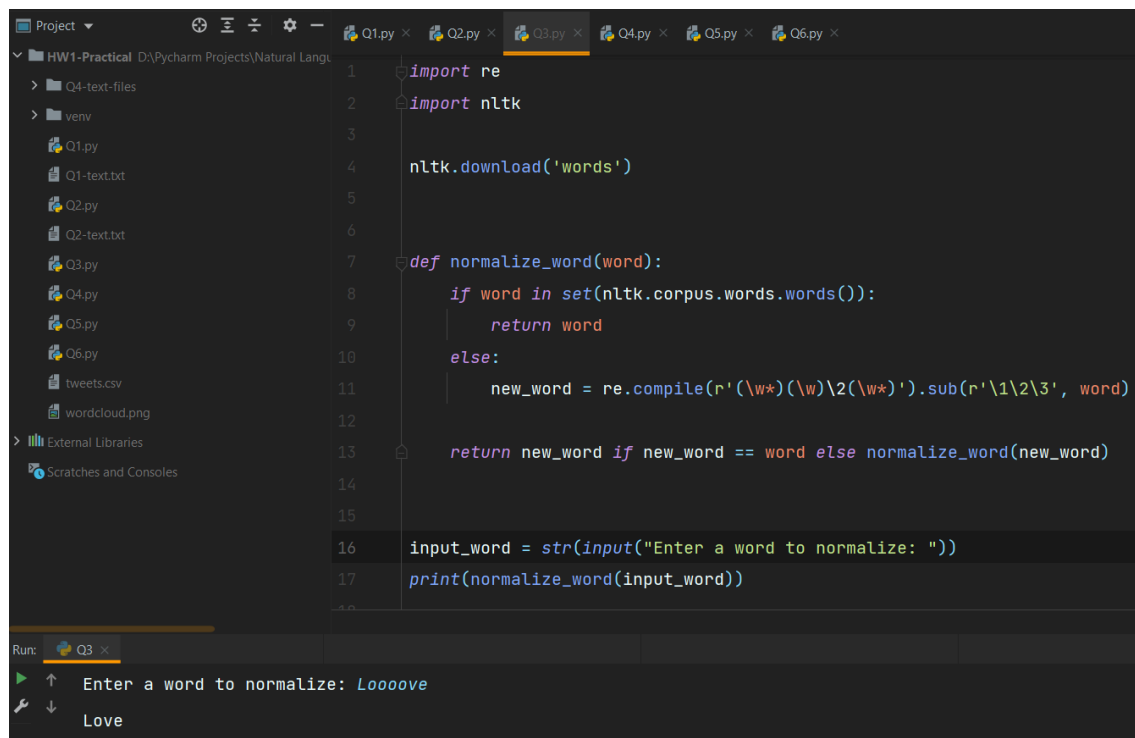
```

پاسخ سوال سوم عملی:

برای مل این سوال، با توجه به توضیحات تکمیلی داده‌شده در گروه تلگرام قصد حذف کلمات تکراری را داریم و این کار را باید تا جایی ادامه دهیم که به یک کلمه معنادار برسیم که از لحاظ املائی مشکلی نداشته باشد. برای انجام این کار، اقدام به تعریف تابع `normalize_word` کرده‌ایم. عملکرد این تابع به این صورت است که تابعی از نوع بازگشتی است. در هر بار اجرای تابع ابتدا بررسی می‌کنیم کلمه موردنظر در لیست واژگان زبان انگلیسی وجود دارد یا خیر. در صورت عدم وجود چنین کلمه‌ای، در هر بار اجرا اقدام به حذف یک حرف تکراری می‌کنیم. حالت پایه (`base case`) برای این تابع حالتی است که کلمه کوتاه‌شده با کلمه ورودی برابر باشد. در صورت برابر نبودن، به صورت بازگشتی تابع را برای کلمه کوتاه‌شده

(*new_word*) صدا می‌زنیم. در نهایت، به حالتی می‌رسیم که دیگر کلمه امکان کوتاه‌شدن ندارد و در دیکشنری واژگان زبان انگلیسی دارای معنا می‌باشد و در این حالت، از تابع خروج می‌کنیم.

در تصویر زیر، اجرای برنامه و نتیجه‌ی آن مشاهده می‌شود:



```
1 import re
2 import nltk
3
4 nltk.download('words')
5
6
7 def normalize_word(word):
8     if word in set(nltk.corpus.words.words()):
9         return word
10    else:
11        new_word = re.compile(r'(\w*)(\w)\2(\w*)').sub(r'\1\2\3', word)
12
13    return new_word if new_word == word else normalize_word(new_word)
14
15
16 input_word = str(input("Enter a word to normalize: "))
17 print(normalize_word(input_word))
```

Run: Q3 x

Enter a word to normalize: Love

پاسخ سوال چهارم عملی:

ابتدا اقدام به باز کردن فایل‌های متنی داده‌شده کرده‌ایم و *handle* مربوط به هر فایل را در متغیر مربوط به هر فایل نمونه قرار داده‌ایم.

```
# open the input files we need to work with their sentences
albert_einstein_text = open('Q4-text-files/AlbertEinstein.txt').read()
shahnameh_text = open('Q4-text-files/Shahnameh.txt').read()
short_sample_english_text = open('Q4-text-files/ShortSampleEnglish.txt').read()
short_sample_persian_text = open('Q4-text-files/ShortSamplePersian.txt').read()
```

به عنوان اولین پارت، تعداد توکن‌ها و *type* های آنها را در هریک از متن‌های مربوطه با استفاده از *TreebackWordTokenizer* پیدا می‌کنیم. این کار را با استفاده از متود *tokenize* انجام می‌دهیم. در تصویر زیر، نمونه انجام این کار آورده شده است:

```
# Part I
print("*****Part I - TreebankWordTokenizer*****")
tree_bank_word_tokenizer = TreebankWordTokenizer()
albert_einstein_text_tokens = tree_bank_word_tokenizer.tokenize(albert_einstein_text)
shahnameh_text_tokens = tree_bank_word_tokenizer.tokenize(shahnameh_text)
short_sample_english_text_tokens = tree_bank_word_tokenizer.tokenize(short_sample_english_text)
short_sample_persian_text_tokens = tree_bank_word_tokenizer.tokenize(short_sample_persian_text)

print("Number of tokens in AlbertEinstein.txt:", len(albert_einstein_text_tokens))
print("Type of tokens in AlbertEinstein.txt:", len(set(albert_einstein_text_tokens)))
print("=" * 50)
print("Number of tokens in Shahnameh.txt:", len(shahnameh_text_tokens))
print("Type of tokens in Shahnameh.txt:", len(set(shahnameh_text_tokens)))
print("=" * 50)
print("Number of tokens in ShortSampleEnglish.txt:", len(short_sample_english_text_tokens))
print("Type of tokens in ShortSampleEnglish.txt:", len(set(short_sample_english_text_tokens)))
print("=" * 50)
print("Number of tokens in ShortSamplePersian.txt:", len(short_sample_persian_text_tokens))
print("Type of tokens in ShortSamplePersian.txt:", len(set(short_sample_persian_text_tokens)))
```

فروچی اجرای کدهای تصویر بالا به صورت زیر خواهد بود:

```
*****Part I - TreebankWordTokenizer*****
Number of tokens in AlbertEinstein.txt: 250
Type of tokens in AlbertEinstein.txt: 131
=====
Number of tokens in Shahnameh.txt: 10099
Type of tokens in Shahnameh.txt: 36
=====
Number of tokens in ShortSampleEnglish.txt: 85
Type of tokens in ShortSampleEnglish.txt: 26
=====
Number of tokens in ShortSamplePersian.txt: 72
Type of tokens in ShortSamplePersian.txt: 19
```

در پارت دوم، با استفاده از *RegexTokenizer* کلمات مربوط به متن کوتاه انگلیسی و فارسی و همچنین اعداد مربوط به متن نمونه انگلیسی را استخراج می‌کنیم. برای به دست آوردن کلمات از عبارت منظم `"w+"` و برای به دست آوردن اعداد از عبارت `"d+"` استفاده می‌کنیم. در تصویر زیر این کار را انجام داده‌ام.


```
# Part II
print("*****Part II - RegexpTokenizer*****")
regexp_word_tokenizer = RegexpTokenizer(r"\w+")
print("Words in ShortSampleEnglish.txt:", regexp_word_tokenizer.tokenize(short_sample_english_text))
print("=" * 50)
print("Words in ShortSamplePersian.txt:", regexp_word_tokenizer.tokenize(short_sample_persian_text))
print("=" * 50)
regexp_digit_tokenizer = RegexpTokenizer(r"\d+")
print("Digits in AlbertEinstein.txt:", regexp_digit_tokenizer.tokenize(albert_einstein_text))
print("=" * 50)
```

فروبی اجرای قطعه کد بالا در تصویر زیر مشاهده می‌شود. همانطور که در تصویر مشخص است، کلمات و اعداد به طور صمیم از متون مربوطه استخراج شده‌اند.

```
*****Part II - RegexpTokenizer*****
Words in ShortSampleEnglish.txt: ['Hello', 'Hope', 'you', 're', 'doing', 'well', 'It', 'is', 'a', 'sample', 'short', 'text', 'This', 'is', 'our',
=====
Words in ShortSamplePersian.txt: ['است', 'ما', 'ا', 'تعمین', 'این', 'است', 'نقشه', 'کوتاه', 'متن', 'یک', 'این', 'باشید', 'خوب', 'امیدوارم', 'م',
=====
Digits in AlbertEinstein.txt: ['14', '1879', '1896', '1901', '1905', '1908', '1909', '1911', '1914', '1914', '1933', '1940', '1945']
=====
```

در پارت سوم، جداسازی کلمات را با استفاده از *WhiteSpaceTokenizer* انجام می‌دهیم. نمونه عملکرد این روش به این صورت است که کاراکتر جداکننده‌ی توکن‌ها را *whitespace*، *n* و *t* در نظر می‌گیرد و توکن‌ها را براساس اینها جدا می‌کند. بنابراین ممکن است مثلاً انتهای جمله‌ای که نقطه آمده باشد، نقطه هم جزو توکن در نظر گرفته شود. اگر بخواهیم مشابه همین کار را با *RegexpTokenizer* انجام دهیم، کافی است برای عبارت منظم از *"S+"* استفاده کنیم. همانطور که می‌دانیم، این عبارت منظم به معنای در نظر گرفتن عبارات و جدا کردن آنها با استفاده از کاراکترهای بجز فاصله (*whiteSpace*) می‌باشد و به محض اینکه به فاصله برسد، توکن جدید در نظر گرفته می‌شود.

```
# Part III
print("*****Part III - WhitespaceTokenizer*****")
white_space_tokenizer = WhitespaceTokenizer()
print("Words in ShortSampleEnglish.txt: (with WhitespaceTokenizer)",
      white_space_tokenizer.tokenize(short_sample_english_text))
print("=" * 50)
regexp_word_tokenizer = RegexpTokenizer(r"S+")
print("Words in ShortSampleEnglish.txt (with RegexpTokenizer):",
      regexp_word_tokenizer.tokenize(short_sample_english_text))
print("=" * 50)
```

نتیجه اجرای قطعه کد بالا به صورت زیر است که همانطور که می‌بینیم، نتیجه‌ی استفاده از هر دو کلاس یک فروجی را به ما خواهد داد.

```
*****Part III - WhitespaceTokenizer*****
Words in ShortSampleEnglish.txt: (with WhitespaceTokenizer) ['Hello!', 'Hope', 'you're', 'doing', 'well.', 'It', 'is', 'a', 'sam
=====
Words in ShortSampleEnglish.txt (with RegexpTokenizer): ['Hello!', 'Hope', 'you're', 'doing', 'well.', 'It', 'is', 'a', 'sample'
=====
```

به عنوان آخرین پارت، در این قسمت با استفاده از *WordPunctTokenizer* اقدام به جداکردن توکن‌ها می‌کنیم. نمونه عملکرد این کلاس به این صورت است که کاراکترها و توکن‌ها را براساس *Alphabetic* و *non-Alphabetic* جدا می‌کند. به صورتی که تا زمانی که به کاراکترهای غیر الفبایی نرسیده باشیم، همچنان توکن ادامه می‌یابد و به محض رسیدن به کاراکتری که جزو الفبای زبان نباشد، توکن جدا می‌شود.

در تصاویر زیر، نمونه‌ی انجام این کار و نتیجه‌ی اجرای آن را می‌بینیم.

```
# Part IV
print("*****Part IV - WordPunctTokenizer*****")
word_punct_tokenizer = WordPunctTokenizer()
print("Words in ShortSampleEnglish.txt:",
      word_punct_tokenizer.tokenize(short_sample_english_text))
print("=" * 50)
```

```
*****Part IV - WordPunctTokenizer*****
Words in ShortSampleEnglish.txt: ['Hello', '!', 'Hope', 'you', '', 're', 'doing', 'well', '.', 'It', 'is', 'a', 'sample', 'short',
=====
```

پاسخ سوال پنجم عملی:

نمونه‌ی استفاده از *PorterStemmer* و *LanceterStemmer* به این صورت است که در ابتدا، پس از *import* کردن هر دو کتابخانه، باید یک آبجکت از هرکدام از آنها بسازیم و مال روی هر کلمه‌ای که قصد داریم *stem* شده‌ی آن را به دست آوریم، متود *stem* را از آبجکت مربوط به هرکدام از روشهایی که قصد استفاده از آنها را داریم، صدا می‌زنیم. فروجی این متود، فواستهی ما خواهد بود.

به عنوان پارت اول، قصد داریم توکن‌هایی با اندیس‌های مشخص‌شده در صورت سوال که از روش *TreebankWordTokenizer* در سوال قبل به دست آمده‌اند را یکبار با *PorterStemmer* و بار دیگر با *LancasterStemmer* نمایش دهیم و مقایسه کنیم. این کار را در تصویر زیر انجام داده‌ایم.

```
from nltk import TreebankWordTokenizer, WordNetLemmatizer
from nltk.stem import PorterStemmer, LancasterStemmer

nltk.download('wordnet')

|
albert_einstein_text = open('Q4-text-files/AlbertEinstein.txt').read()

tree_bank_word_tokenizer = TreebankWordTokenizer()
porter = PorterStemmer()
lancaster = LancasterStemmer()
word_net_lemmatizer = WordNetLemmatizer()

# Part I
print("*****Part I - PorterStemmer vs LancasterStemmer*****")
albert_einstein_text_tokens = tree_bank_word_tokenizer.tokenize(albert_einstein_text)
for i in [2, 10, 18, 19, 21, 22, 42]:
    print(porter.stem(albert_einstein_text_tokens[i]), '-', lancaster.stem(albert_einstein_text_tokens[i]))
print("=" * 50)
```

خروجی برنامه به صورت زیر خواهد بود:

```
*****Part I - PorterStemmer vs LancasterStemmer*****
wa - was
germani - germany
week - week
later - lat
famili - famy
move - mov
itali - ita
=====
```

در پارت دوم، با استفاده از *WordNetLemmatizer* کلمات فواسته‌شده را به حالت اولیه نگارشی آنها درمی‌آوریم.

```
# Part II
print("*****Part II - WordNetLemmatizer*****")
for word in ['Waves', 'fishing', 'rocks', 'was', 'corpora', 'better', 'ate', 'broken']:
    print(word_net_lemmatizer.lemmatize(word))
```

*****Part II - WordNetLemmatizer*****

Waves
fishing
rock
wa
corpus
better
ate
broken

درمورد پاسخ سوال آخر که آیا *lemmatize* به صورت پیش فرض همواره پاسخ درست را برمی گرداند، پاسخ خیر است. همانطور که در تصویر بالا هم مشاهده می شود، به عنوان نمونه *wa* کلمه‌ی بی معنی است که نتوانسته است بازسازی شود. برای گرفتن نتیجه‌ی صحیح برای تمامی کلمات بهتر است بجای فراخوانی تابع *lemmatize* به صورت پیش فرض، برای هر کلمه پوزیشن و نقش گرامری آن کلمه را هم بهتر است تحت پارامتری با نام *pos* به تابع پاس دهیم تا براساس نقش آن کلمه تصمیم گیری درست تری انجام بگیرد و نتیجه به چیزی که صحیح است، نزدیکتر باشد. مثلاً در صورتی که فعل باشد، باید از *pos='v'* و برای اسامی نیز از *pos='n'* استفاده می شود.

پاسخ سوال ششم عملی:

برای پیش پردازش هریک از توئیتهای طبق مراملی که صورت سوال خواسته است، پیش می روییم تا متن و کلمات را به فرمت تمیز و قابل استفاده درآوریم. در ابتدا، فایل *csv* مربوطه را باز می کنیم و داده های مربوط به هر توئیتهای را در یک سطر در *csv_reader* می خوانیم. حال برای برقرار کردن هریک از مرامل پیش پردازش خواسته شده به ترتیب زیر عمل می کنیم:

- مرحله اول: حذف فضاهای خالی اضافه در میان کلمات یک توئیتهای با استفاده از تابع *strip()*
- مرحله دوم: تبدیل کلیه مروف کلمات به مروف کوچک با استفاده از تابع *lower()*
- مرحله سوم: حذف کلیه *Handle* ها از داخل متن با استفاده از عبارات منظم و یافتن عباراتی با فرم *"@|w+"* و جایگزین کردن آنها با کاراکتر خالی (دستور *sub*)
- مرحله چهارم: حذف کلیه علائم نگارشی، اعداد و کاراکترهای خاص با بررسی شرط *isalpha()*
- مرحله پنجم: عملیات *tokenize* بر روی هر توئیتهای با استفاده از *word_tokenize*. روش جدا کردن توکن ها در این متود به این صورت است که تلاش می کند یک سیلابس از هر کلمه را در بیاورد. همانطور که می دانیم، هر کلمه از یک یا دو سیلابس تشکیل شده است. در انتها لیستی شامل کلمات جدا شده را برمی گرداند.

- **مرمله ششم:** حذف کلمات *Stop Words* از متن. کلمات توقف یا *Stop Words* کلماتی بدون استفاده هستند که در مین پردازش فایده‌ای ندارند. کلمات توقف، در واقع کلماتی هستند که به طور متداول استفاده می‌شوند و موتورهای جستجو، به گونه‌ای برنامه‌نویسی شده‌اند که این کلمات را هم در هنگام ایندکس کردن صفحات وب و هم در هنگام بازیابی آن‌ها در نتیجه یک کوئری جستجو، نادیده می‌گیرند. از جمله این کلمات متداول می‌توان به «از، به، و، را» و چنین مواردی در زبان فارسی و «*the, a, an, in*» در زبان انگلیسی اشاره کرد. در این مرملة، اقدام به حذف این نوع کلمات با بررسی وجود آنها در کالشن *stopwords* از پکیج *nlTK.corpus* می‌کنیم. در صورتی که کلمه‌ای جزو این کالشن باشد، آن را از مجموعه کلمات قابل بررسی حذف می‌کنیم.
- **مرملة هفتم:** در این مرملة کلماتی با طول کمتر از ۳ را حذف می‌کنیم. دلیل حذف این نوع کلمات این است که این کلمات معمولاً معنای خاصی ندارند و یا جزو کلمات توقف هستند و اگر جزو کلمات توقف نباشند، عمدتاً صرف اضافه هستند و یا کلماتی هستند که کاربرد زیادی ندارند.
- **مرملة هشتم:** در این مرملة قصد داریم با استفاده از *PorterStemmer* بر روی توکن‌های هر توییت، عملیات *Stemming* را انجام دهیم و هر توییت را مجدداً بازسازی کنیم. در مورد مقایسه‌ی دو روش *Stemming* می‌توان گفت که تفاوت اصلی *Lancaster* و *Porter* در این است که *Lancaster* کمی *Aggressive* تر و تهاجمی‌تر نسبت به *Porter* برافورد می‌کند. پورتر بدون شک پرمصرف‌ترین *Stemmer* است و یکی از محدود الگوریتم‌هایی است که از جاوا پشتیبانی می‌کند. اگرچه که از نظر پیچیدگی محاسباتی واقعاً پیچیده است و جزو قدیمی‌ترین الگوریتم‌هاست. در مقابل، *Lancaster* بسیار تهاجمی است و گاهی اوقات ممکن است به خطا رود. در *Lancaster* بسیاری از کلمات کوتاه ممکن است کاملاً برای خواننده مبهم شوند. این الگوریتم سرعت اجرای بالاتری دارد و مجموعه کلمات را به شدت کاهش می‌دهد.

توضیحات داده‌شده به صورت عملی در تصویر زیر مشهود است:

```
all_tokens = []
num_of_tweets = 0
with open('tweets.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        tweet = row[0]
        tokens = [word for word in word_tokenize(re.sub(r"@w+", '', tweet.strip()).lower())
                    if word.isalpha()
                    and word not in stopwords.words("english")
                    and len(word) >= 3]

        stemmed_tokens = [porter.stem(token) for token in tokens]
        print(' '.join(stemmed_tokens))

        all_tokens = all_tokens + tokens
        num_of_tweets += 1
```

برای به دست آوردن پرتکرارترین کلمات در این مجموعه داده (که در لیست ذخیره شده اند)، از کلاس *Counter* استفاده می‌کنیم که با استفاده از متود *most_common* می‌توانیم آنها را نمایش دهیم. برای نمایش این کلمات نیز در بین داده‌ها می‌توانیم از *WordCloud* استفاده کنیم تا میزان فراوانی یک کلمه را در مقیاس و مقایسه با سایر کلمات ببینیم.

```
show_wordcloud(' '.join(all_tokens), 'Tweets Wordcloud')
print("=" * 100)
print("Top 10 the most frequent words in the tweets are:")
print(Counter(all_tokens).most_common(10))
```

```
def show_wordcloud(data, title=None):
    wordcloud = WordCloud(
        background_color='black',
        width=800,
        height=400,
        max_words=200,
        # max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(figsize=(20, 10))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
```

برای به دست آوردن ترندها باید به دنبال هشتکها در توییتها بگردیم که این کار را فیلری رامت با استفاده از عبارات منظم انجام داده‌ایم. به این صورت که تمام عباراتی که شامل هشتک بودند را داخل یک لیست ذخیره کردیم تا در انتها با استفاده از *Counter* بتوانیم ۱۰ ترند اول را شناسایی کنیم.

```
hashtags = re.findall(r"#\w+", tweet)
if len(hashtags) > 0:
    trends = trends + hashtags
```

```
top_10_trends = Counter(trends).most_common(10)
print("Top 10 trends in the tweets are:")
print(top_10_trends)
print("=" * 100)
```

در انتها در تابع *draw_trends_chart* به ترسیم نمودار مربوط به فراوانی ترندها پرداخته‌ایم.

```
def draw_trends_chart(trends, values):
    fig, ax = plt.subplots(figsize=(16, 9))
    ax.barh(trends, values)
    for s in ['top', 'bottom', 'left', 'right']:
        ax.spines[s].set_visible(False)

    ax.xaxis.set_ticks_position('none')
    ax.yaxis.set_ticks_position('none')
    ax.xaxis.set_tick_params(pad=5)
    ax.yaxis.set_tick_params(pad=10)

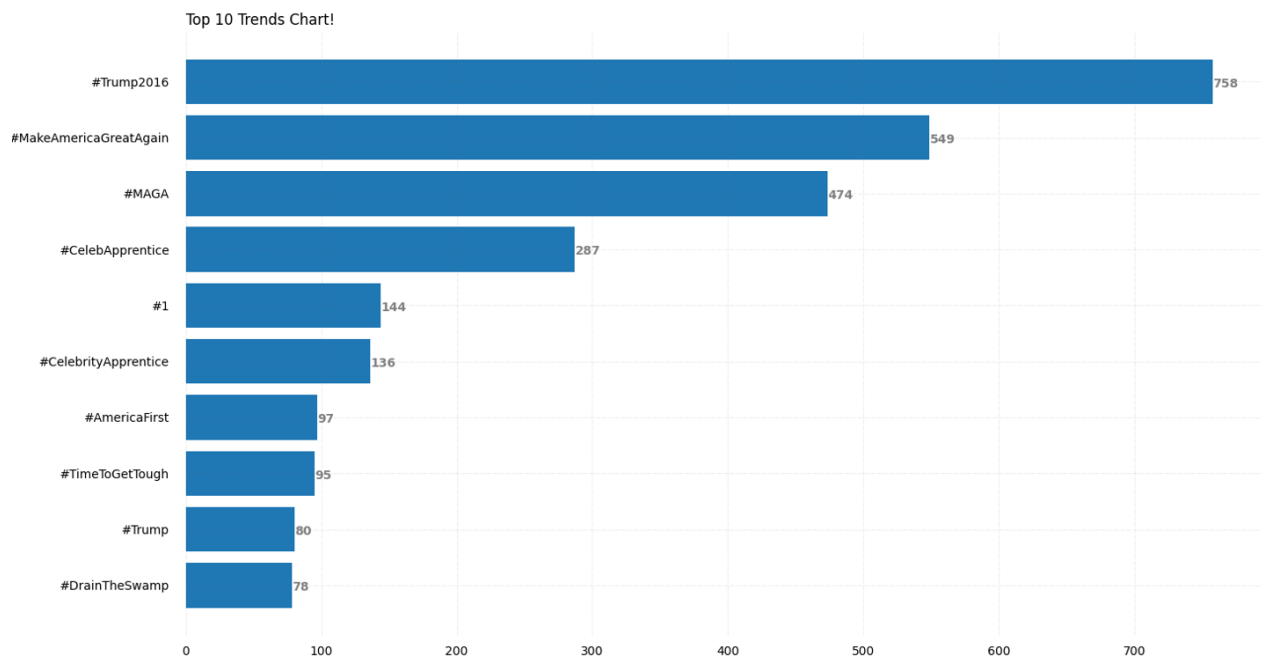
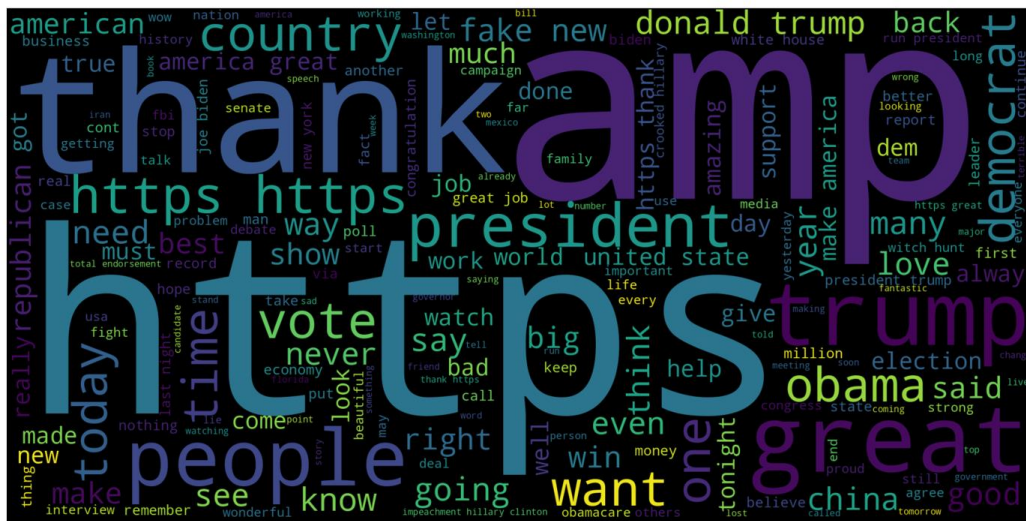
    ax.grid(b=True, color='grey',
            linestyle='--.', linewidth=0.5,
            alpha=0.2)

    ax.invert_yaxis()
    for i in ax.patches:
        plt.text(i.get_width() + 0.2, i.get_y() + 0.5,
                 str(round((i.get_width()), 2)),
                 fontsize=10, fontweight='bold',
                 color='grey')
    ax.set_title('Top 10 Trends Chart!',
                 loc='left', )
```

```
x_trends = [trend[0] for trend in top_10_trends]
y_values = [trend[1] for trend in top_10_trends]
draw_trends_chart(x_trends, y_values)
```

نمودار ترسیم‌شده برای ترندها و *Wordcloud* به دست آمده به صورت زیر خواهند بود:

Tweets Wordcloud



در انتهای اجرای برنامه، ۱۰ کلمه پرکاربرد و ۱۰ ترند مربوطه در داخل ترمینال نیز چاپ شده‌اند.

```
Top 10 the most frequent words in the tweets are:
[('https', 12112), ('great', 7622), ('trump', 6277), ('amp', 5682), ('president', 4583), ('http', 4513), ('thank', 3590),
=====
Top 10 trends in the tweets are:
[('#Trump2016', 758), ('#MakeAmericaGreatAgain', 549), ('#MAGA', 474), ('#CelebApprentice', 287), ('#1', 144), ('#Celebrity', 134), ('#Trump', 124), ('#MAGA2016', 121), ('#Trump2017', 119), ('#MAGA2017', 118)]
```


سه نمونه از توییت‌ها پیش و پس از فرایند پیش‌پردازش:

```
The Media is just as corrupt as the Election itself!
***
media corrupt elect
```

```
Ammar is a puppet for Nancy Pelosi and the Radical Left. He spells higher taxes, weak Military and Vet support, and the obliteration of the American Dream.
***
ammar puppet nanci pelosi radical left spell higher tax weak military vet support obliteration amend vote darrel issa http
```

```
RT @WhiteHouse: JUST ANNOUNCED: We have finalized a partnership with @CVSHealth and @Walgreens to deliver the Coronavirus vaccine, when appropriate.
***
announc final partnership deliver coronavirus vaccin
```

در انتها نیز با دستوراتی که در تصویر زیر آمده است، توییت‌های اصلی و توییت‌های پس از پیش‌پردازش را همگی در یک فایل CSV جدید می‌ریزیم.

```
with open('new_tweets.csv', 'w') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(csv_fields)
    csv_writer.writerows(csv_rows)
```

ساختار فایل CSV جدید به صورت زیر می‌باشد. (شامل دو ستون برای توییت اصلی و پس از پیش‌پردازش)

Original Tweet	After Preprocessing
◆◆◆Republicans and Democrats have both created our economic problems.,democrat creat econom problem	
I was thrilled to be back in the Great city of Charlotte, North Carolina with thousands of hardworking American Patriots who I	
RT @CBS_Herridge: READ: Letter to surveillance court obtained by CBS News questions where there will be further discipl	
The Unsolicited Mail In Ballot Scam is a major threat to our Democracy, & the Democrats know it. Almost all recent elec	
RT @MZHemingway: Very friendly telling of events here about Comey's apparent leaking to compliant media. If you read the	
RT @WhiteHouse: President @realDonaldTrump announced historic steps to protect the Constitutional right to pray in publi	
Getting a little exercise this morning! https://t.co/fyAAcbhbgk.get littl exercis morn http	