

**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، تقریبا محبوب‌ترین کتابخانه‌ی پایتون برای کار در حوزه‌ی 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 دیکر 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 طبقه بندی متن، برچسب گذاری بخشی 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 ازجمله تحلیل احساسات، طبقه‌بندی موضوع یا استخراج کلمات کلیدی را متوجه شویم. ویژگی منحصر 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 درصورت 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 آموزش‌دیده‌شده را داراست که باعث سهولت کار ما خواهد شد. یکی از برتری‌های این ابزار برخلاف* 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 برخورداری از امکان کلمه‌برداری است.*

**پاسخ سوال دوم تئوری:**

*هریک از عبارات منظم خواسته‌شده به فرمت زیر خواهند بود:*

* آ: شماره تلفن با حداکثر 16 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]

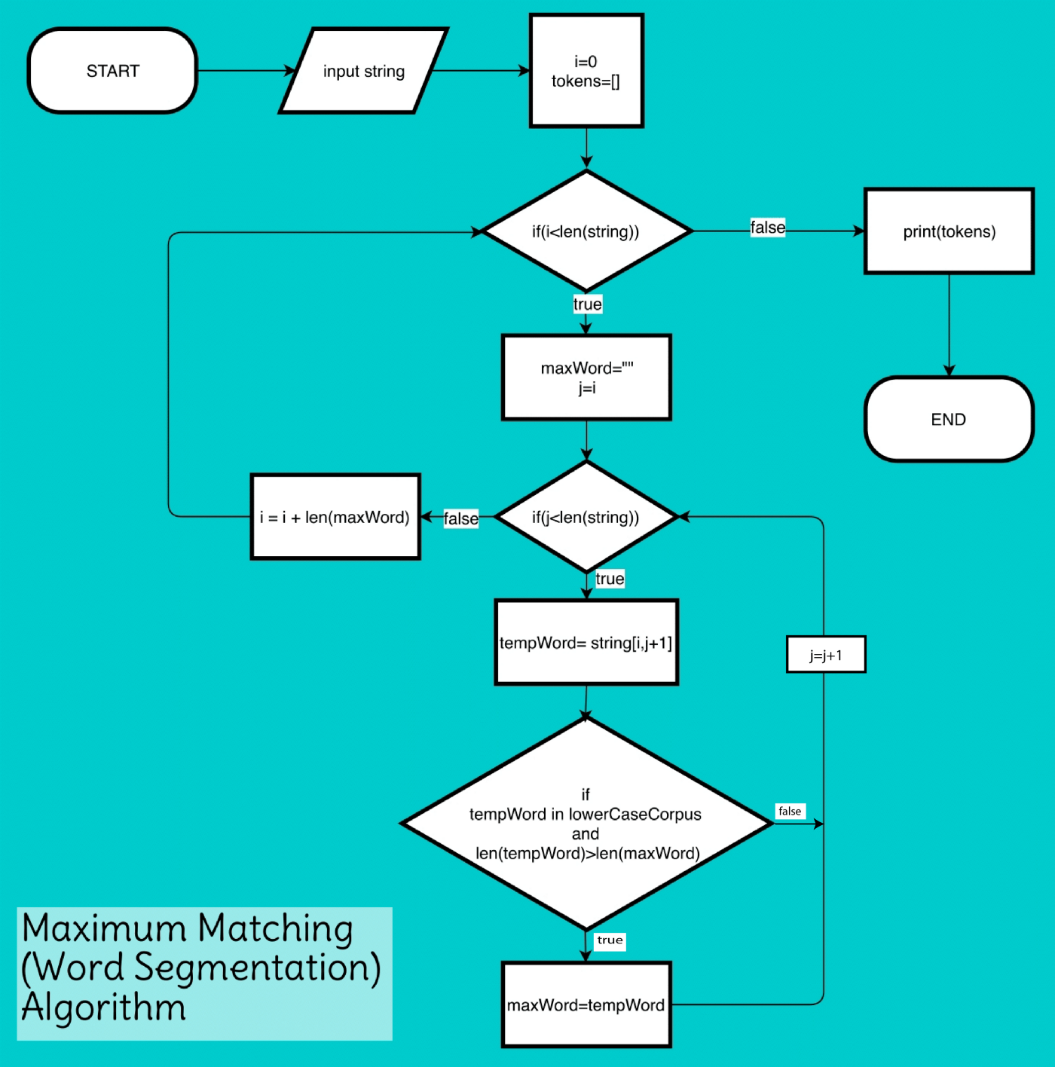
**پاسخ سوال سوم تئوری:**

*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 تشکیل‌دهنده‌ی 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 داده‌شده شروع می‌کنیم.*
* *مرحله دوم: به دنبال بزرگترین عبارتی که از با شروع از این کاراکتر (کاراکتر اول) تشکیل کلمه‌ای معنادار می‌دهد، می‌گردیم.*
* *مرحله سوم: درصورتی 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 درصورت عدم پیدا شدن کلمه معنادار، خود کاراکتر را به عنوان یک کلمه درنظر می‌گیریم.*

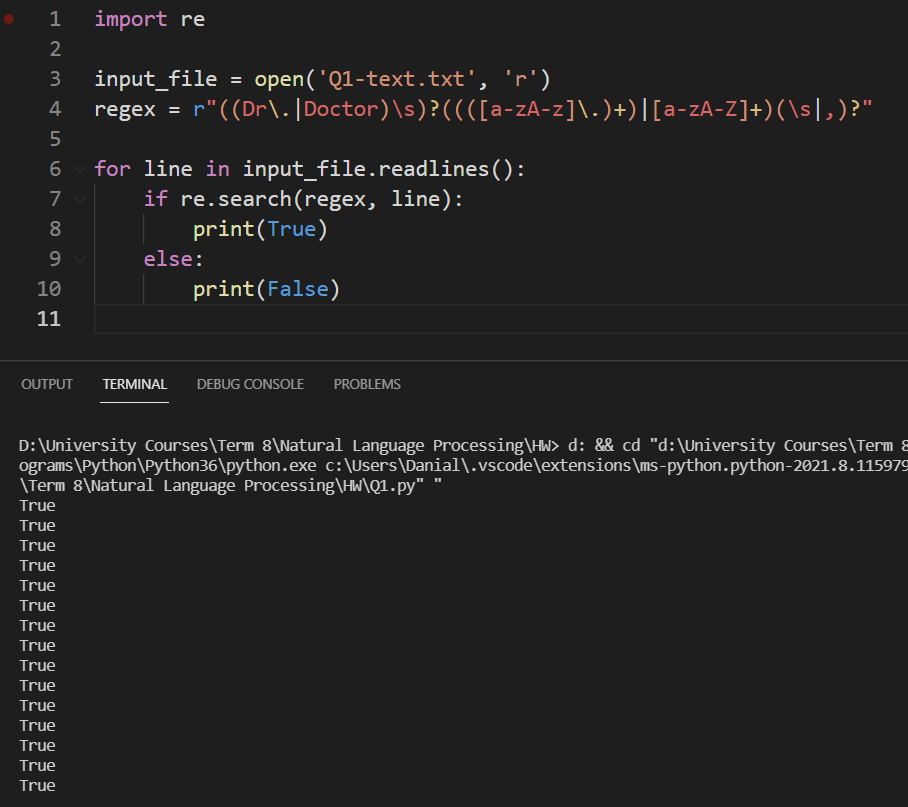
*به عنوان نمونه، فرض کنید می‌خواهیم این الگوریتم را روی کلمه‌ی «thecatinthehat» اجرا کنیم. الگوریتم به این صورت عمل می‌کند که ابتدا اشاره‌گر را روی t می‌گذارد و سعی در پیدا کردن طولانی‌ترین کلمه معنادار دارد. پس از اینکه کلمه‌ی the را پیدا کرد، اشاره‌گر جلو می‌رود و از کاراکتر c به جستجو می‌پردازد که 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 داده‌شده‌اند و مقادیر* True *برگردانده‌شده‌است.*

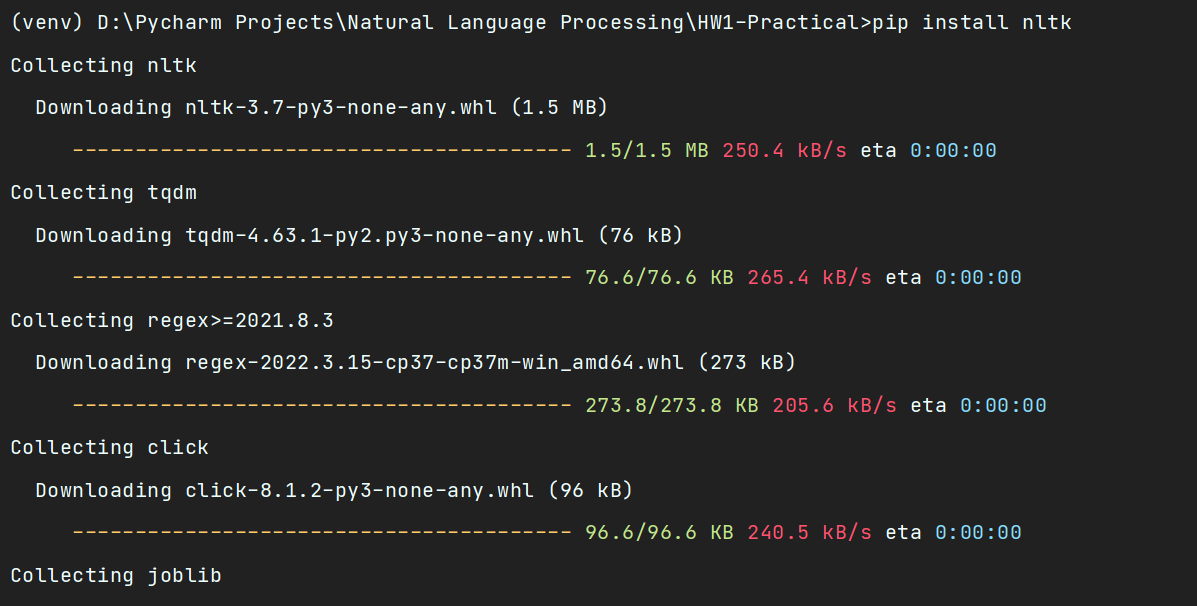
**

*عبارت منظمی که در این سوال برای بررسی عبارات روی آن درنظر گرفته‌ایم، به صورت زیر است:*

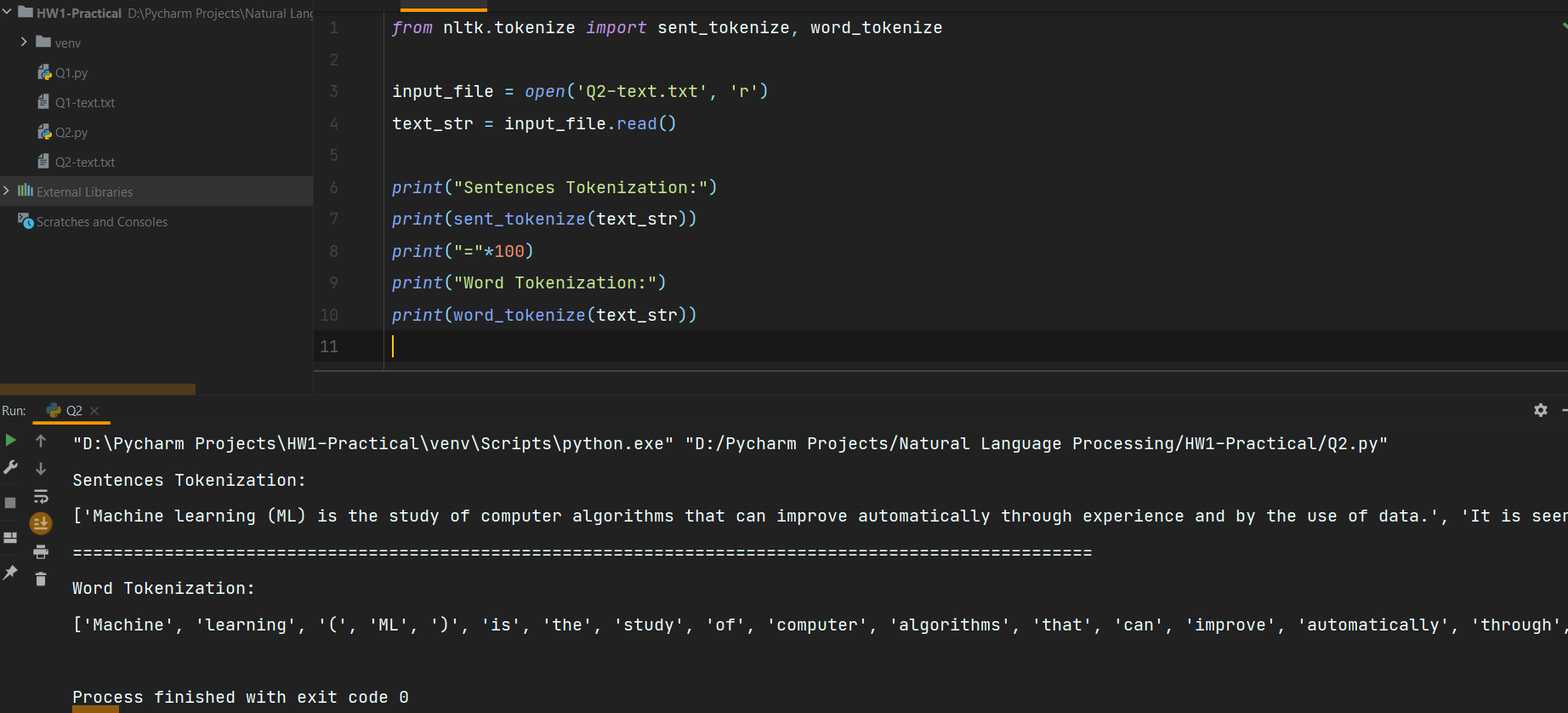
((Dr\. | Doctor) \s)? ((([a-z A-z] \.)+) | [a-z A-Z]+)(\s | ,)?

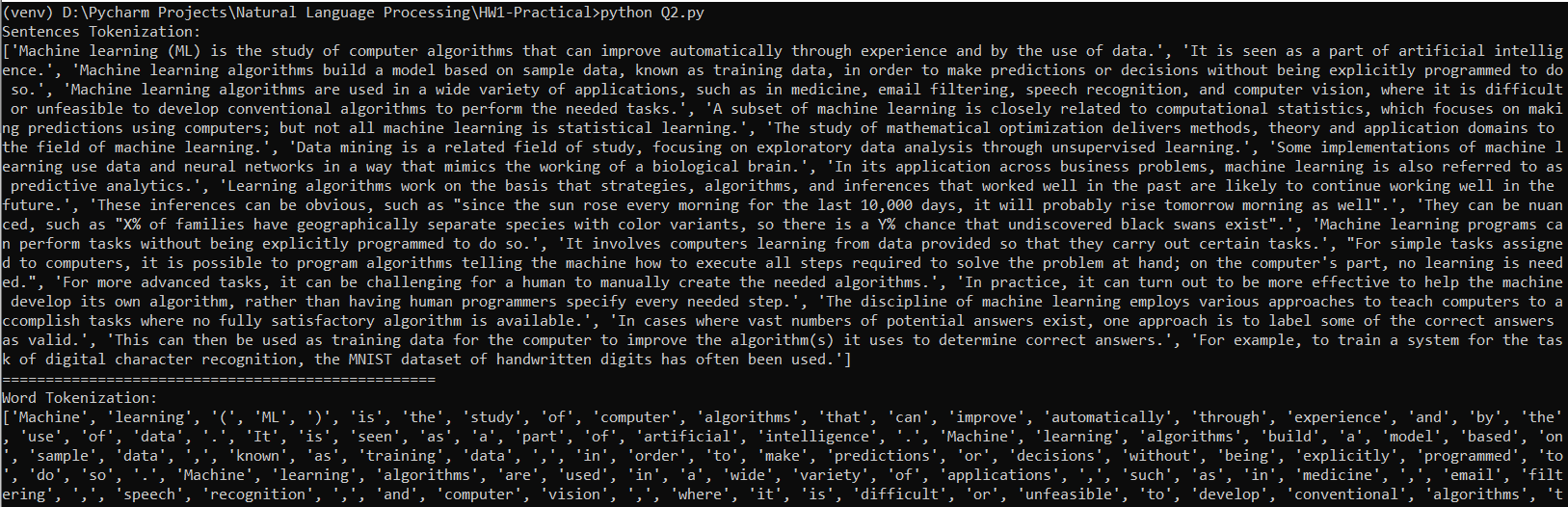
**پاسخ سوال دوم عملی:**

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



*حال با استفاده از متودهای* sent\_tokenize *و* word\_tokenize *از این کتابخانه، متنی را از داخل فایل متنی خوانده و به ترتیب، جملات این متن و کلمات آن را به عنوان خروجی دریافت می‌کنیم.*



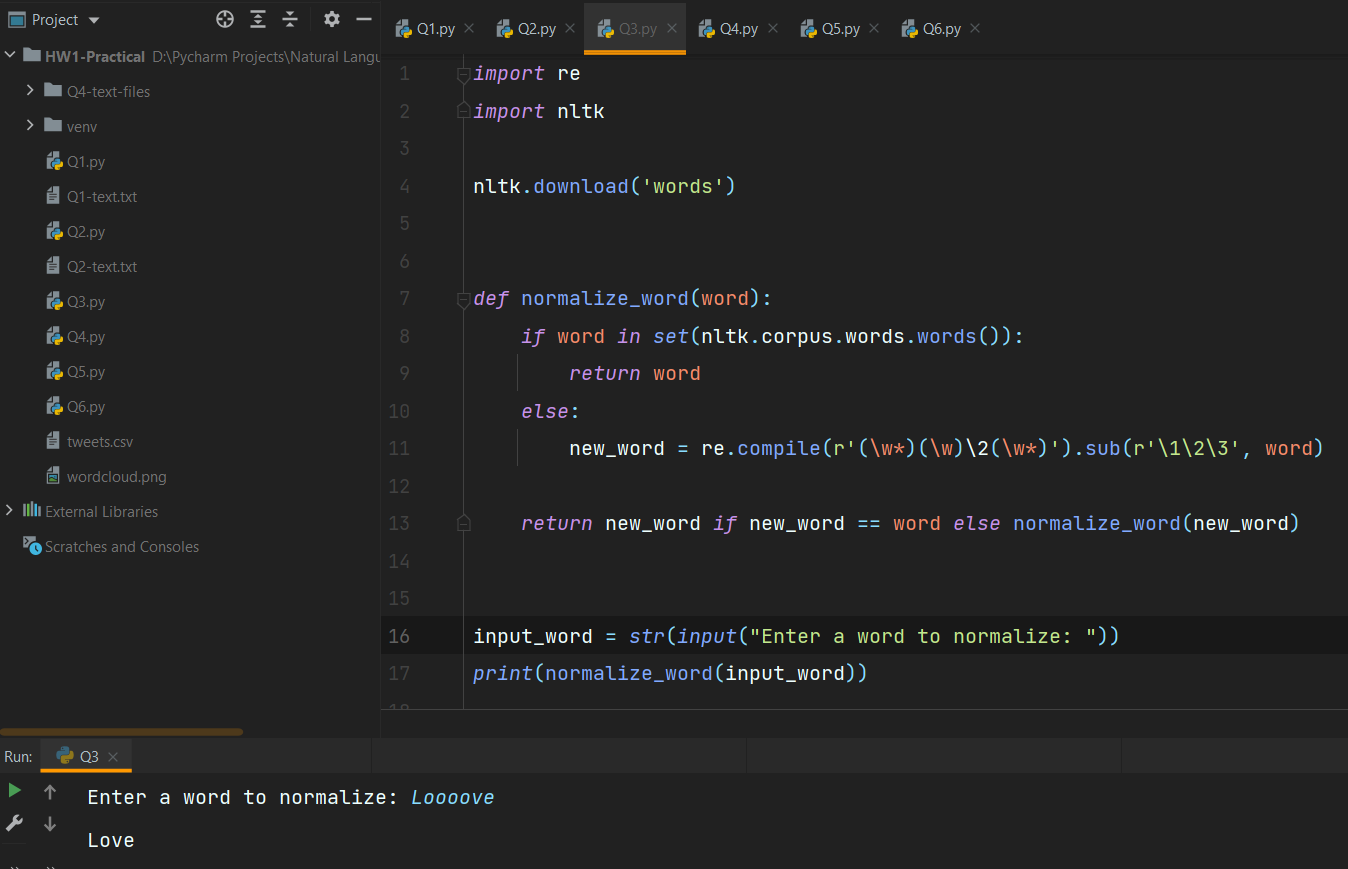


**پاسخ سوال سوم عملی:**

*برای حل 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 املایی 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.*

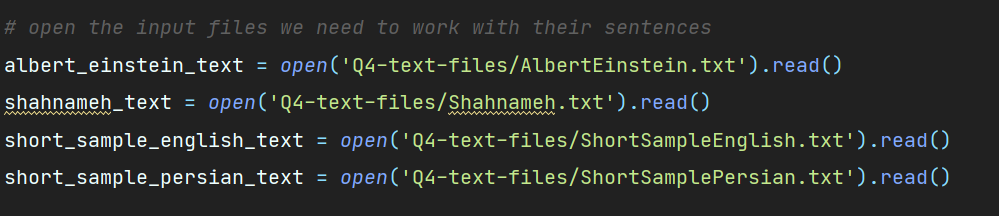
*در هر بار اجرای تابع ابتدا بررسی می‌کنیم کلمه موردنظر در لیست واژگان زبان انگلیسی وجود دارد یا خیر. درصورت عدم وجود چنین کلمه‌ای، در هر بار اجرا اقدام به حذف یک حرف تکراری می‌کنیم. حالت پایه (base case) برای این تابع حالتی است که کلمه کوتاه‌شده با کلمه ورودی برابر باشد. درصورت برابر نبودن، به صورت بازگشتی 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:*

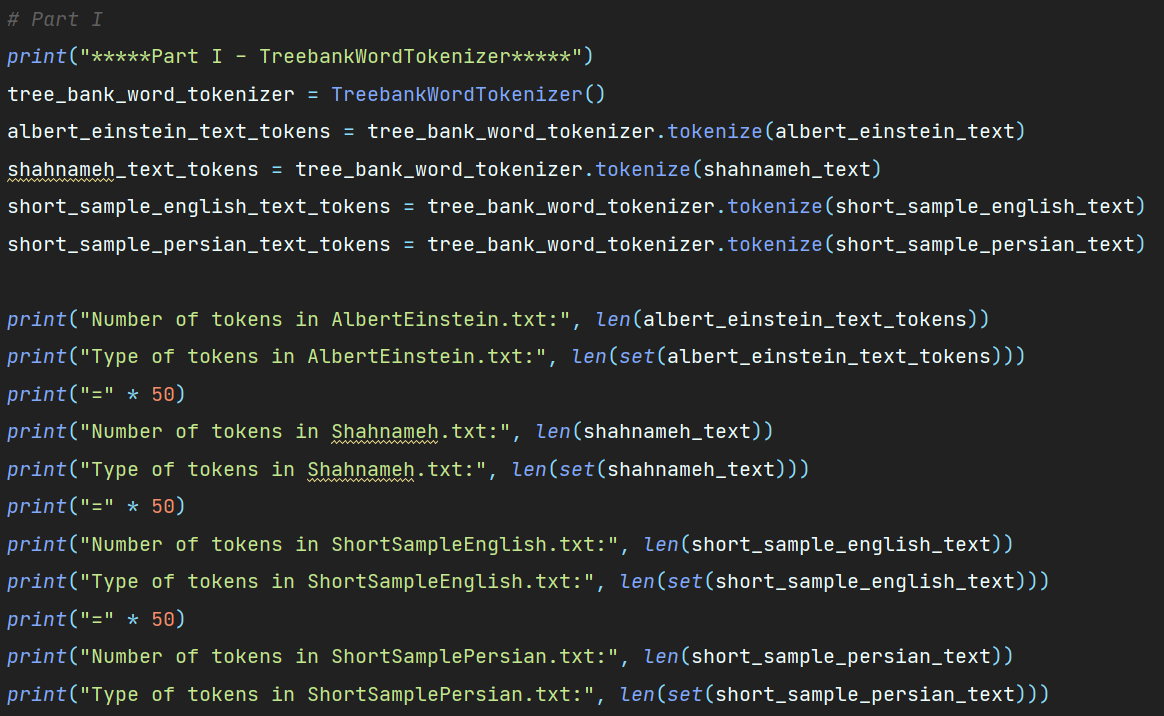


**پاسخ سوال چهارم عملی:**

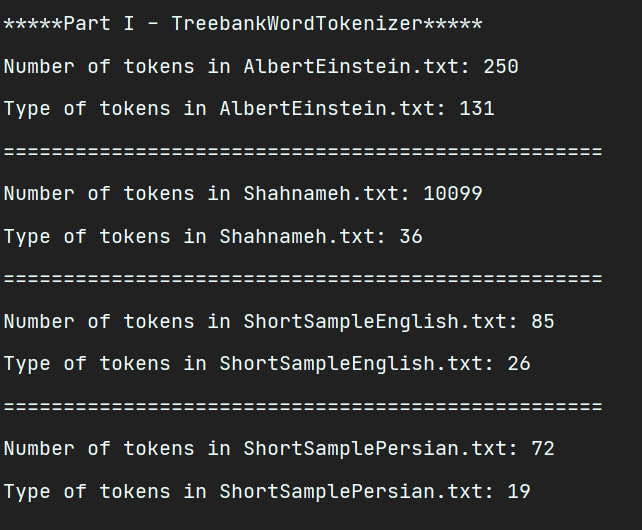
*ابتدا اقدام به باز کردن فایل‌های متنی داده‌شده کرده‌ایم و handle مربوط به هر فایل را در متغیر مربوط به هر فایل نمونه قرار داده‌ایم.*



*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، نحوه انجام این کار آورده شده است:*



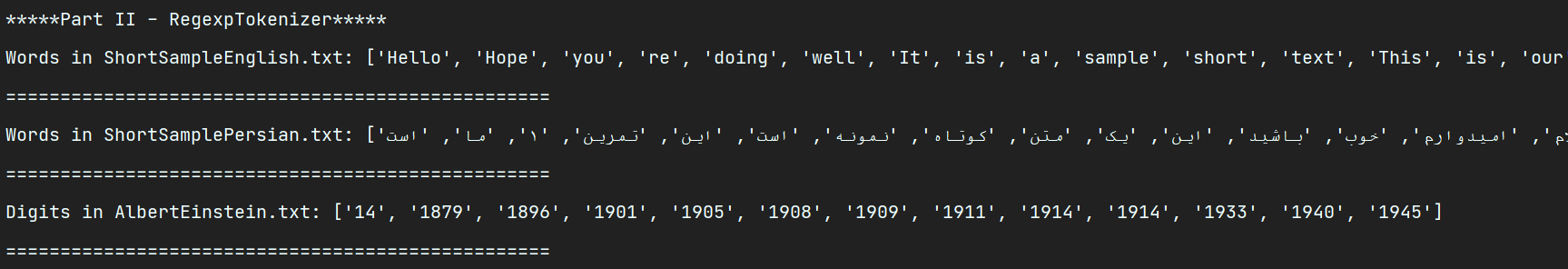
*خروجی اجرای کدهای تصویر بالا به صورت زیر خواهد بود:*



*در پارت دوم، با استفاده از RegexpTokenizer 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.*



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

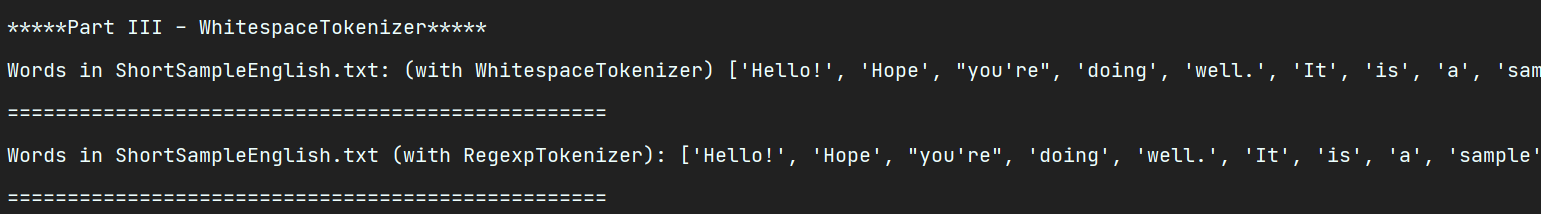


*در پارت سوم، جداسازی کلمات را با استفاده از WhiteSpaceTokenizer انجام می‌دهیم. نحوه عملکرد این روش به این صورت است که کاراکتر جداکننده‌ی توکن‌ها را whitespace، \n و \t درنظر می‌گیرد و توکن‌ها را براساس اینها جدا می‌کند. بنابراین ممکن است مثلا انتهای جمله‌ای که نقطه آمده باشد، نقطه هم جزو توکن درنظر 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 درنظرگرفتن عبارات و جداکردن آنها با استفاده از کاراکترهای بجز فاصله (whiteSpace) می‌باشد و به محض اینکه به فاصله برسد، توکن جدید درنظر گرفته می‌شود.*

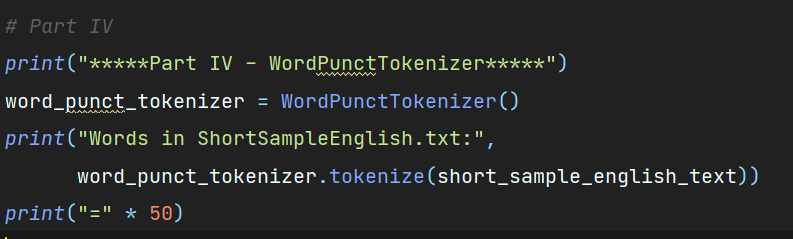


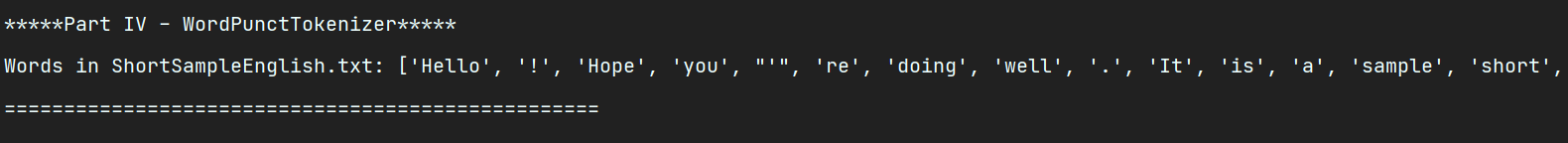
*نتیجه اجرای قطعه کد بالا به صورت زیر است که همانطور که می‌بینیم، نتیجه‌ی استفاده از هر دو کلاس یک خروجی را به ما خواهد داد.*



*به عنوان آخرین پارت، در این قسمت با استفاده از WordPunctTokenizer اقدام به جداکردن توکن‌ها می‌کنیم. نحوه 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، نحوه‌ی انجام این کار و نتیجه‌ی اجرای آن را می‌بینیم.*

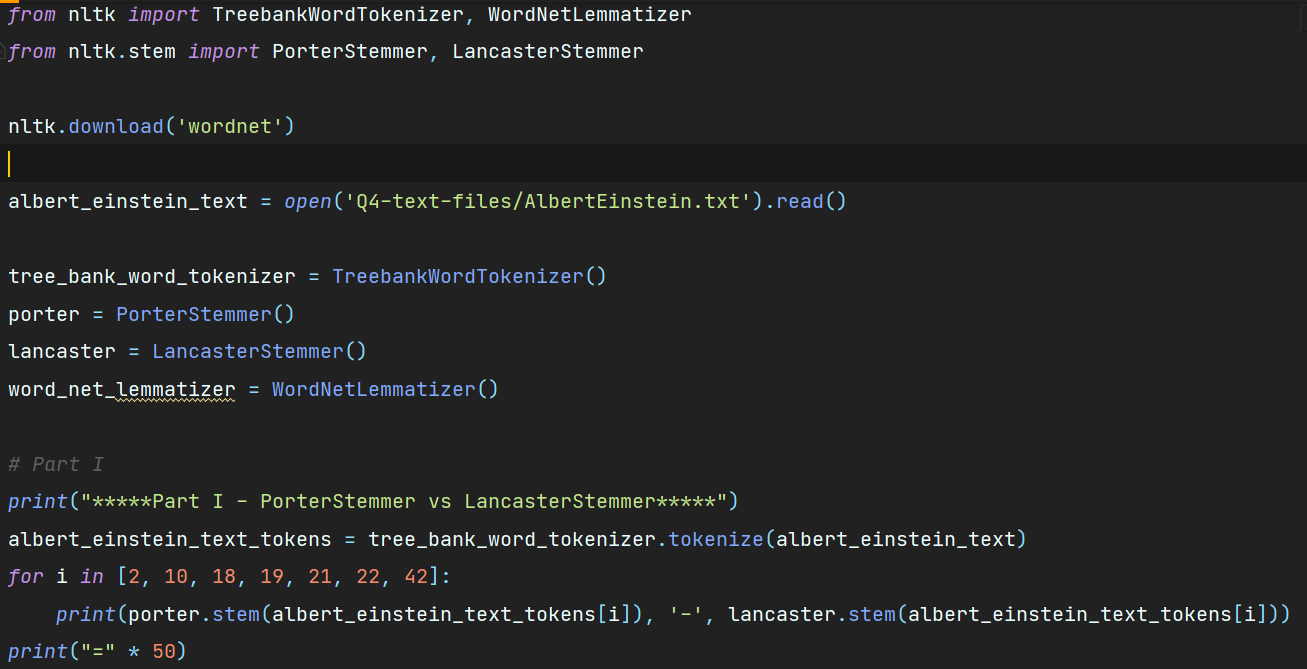




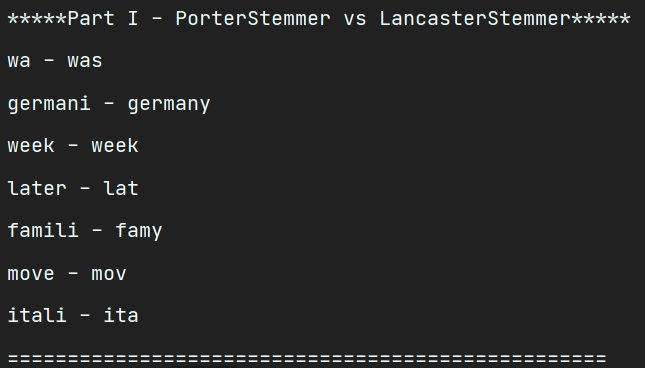
**پاسخ سوال پنجم عملی:**

*نحوه‌ی 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 حال 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. خروجی این متود، خواسته‌ی ما خواهد بود.*

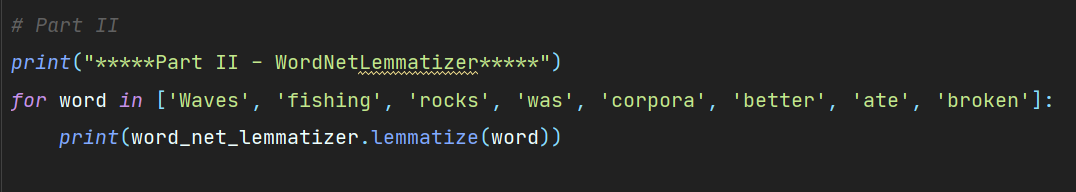
*به عنوان پارت اول، قصد داریم توکن‌هایی با اندیس‌های مشخص‌شده در صورت سوال که از روش TreebankWordTokenizer در سوال قبل به دست آمده‌اند را یکبار با PorterStemmer و بار دیگر با LancasterStemmer نمایش دهیم و مقایسه کنیم. این کار را در تصویر زیر انجام داده‌ایم.*

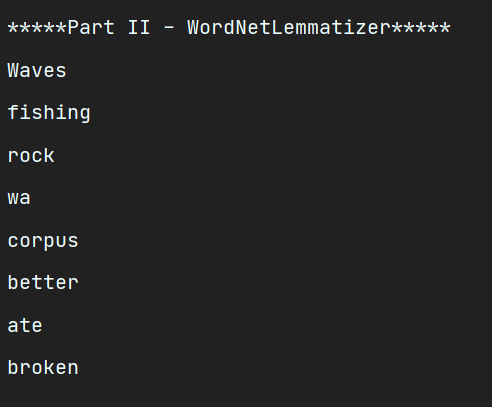


*خروجی برنامه به صورت زیر خواهد بود:*



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





*درمورد پاسخ سوال آخر که آیا lemmatize به صورت پیش‌فرض همواره پاسخ درست را برمی‌گرداند، پاسخ خیر است. همانطور که در تصویر بالا هم مشاهده می‌شود، به عنوان نمونه wa کلمه‌ی بی‌معنی است که نتوانسته است بازسازی شود. برای گرفتن نتیجه‌ی صحیح برای تمامی کلمات بهتر است بجای فراخوانی تابع lemmatize به صورت پیش‌فرض، 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 تصمیم‌گیری درست‌تری انجام بگیرد و نتیجه به چیزی که صحیح است، نزدیکتر باشد. مثلا درصورتی که فعل باشد، باید از pos=’v’ و برای اسامی نیز از pos=’n’ استفاده می‌شود.*

**پاسخ سوال ششم عملی:**

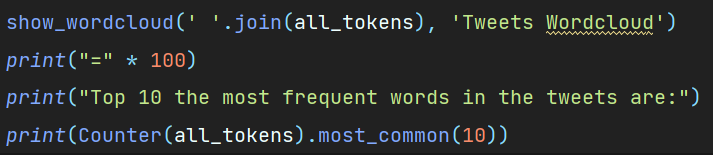
*برای پیش‌پردازش هریک از توییت‌ها طبق مراحلی که صورت سوال خواسته است، پیش می‌رویم تا متن و کلمات را به فرمت تمیز و قابل استفاده درآوریم. در ابتدا، فایل csv مربوطه را باز می‌کنیم و داده‌های مربوط به هر توییت را در یک سطر در csv\_reader می‌خوانیم. حال برای برقرار کردن هریک از مراحل پیش‌پردازش خواسته‌شده به ترتیب زیر عمل می‌کنیم:*

* *مرحله اول: حذف فضاهای خالی اضافه در میان کلمات یک توییت با استفاده از تابع strip()*
* *مرحله دوم: تبدیل کلیه حروف کلمات به حروف 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 چهار: حذف کلیه علائم نگارشی، اعداد و کاراکترهای خاص با بررسی شرط isalpha()*
* *مرحله پنجم: عملیات tokenize بر روی هر توییت با استفاده از word\_tokenize . روش جداکردن توکن‌ها در این متود به این صورت است که تلاش می‌کند یک سیلابس از هر کلمه را دربیاورد. همانطور که می‌دانیم، هر کلمه از یک یا دو سیلابس تشکیل شده است. در انتها لیستی شامل کلمات جداشده را برمی‌گرداند.*
* *مرحله ششم: حذف کلمات StopWords از متن. کلمات توقف یا Stop Words کلماتی بدون استفاده هستند که در حین 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 مرحله، اقدام به حذف این نوع کلمات با بررسی وجود آنها در کالشکن stopwords از پکیج nltk.corpus می‌کنیم. درصورتی که کلمه‌ای جزو این کالکشن باشد، آن را از مجموعه کلمات قابل بررسی حذف می‌کنیم.*
* *مرحله هفتم: در این مرحله کلماتی با طول کمتر از 3 را حذف می‌کنیم. دلیل حذف این نوع کلمات این است که این کلمات معمولا معنای خاصی ندارند و یا جزو کلمات توقف هستند و اگر جزو کلمات توقف نباشند، عمدتا حرف اضافه هستند و یا کلماتی هستند که کاربرد زیادی ندارند.*
* *مرحله هشتم: در این مرحله قصد داریم با استفاده از PorterStemmer بر روی توکن‌های هر توییت، عملیات Stemming را انجام دهیم و هر توییت را مجددا بازسازی کنیم. درمورد 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 برخورد می‌کند. پورتر بدون شک پرمصرف‌ترین Stemmer است و یکی از معدود الگوریتم‌*هایی است که از جاوا پشتیبانی می‌کند. اگرچه که از نظر پیچیدگی محاسباتی واقعا 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 کاملا برای خواننده مبهم شوند. این الگوریتم سرعت اجرای بالاتری دارد و مجموعه کلمات را به شدت کاهش می‌دهد.

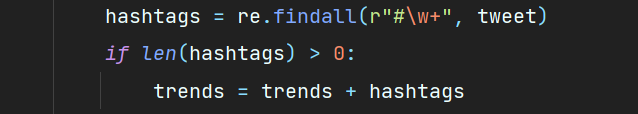
*توضیحات داده‌شده به صورت عملی در تصویر زیر مشهود است:*

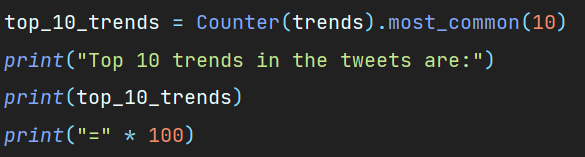


*برای به دست آوردن پرتکرارترین کلمات در این مجموعه داده (که در لیست ذخیره شده اند)، از کلاس Counter استفاده می‌کنیم که با استفاده از متود most\_common می‌توانیم آنها را نمایش دهیم. برای نمایش این کلمات نیز در بین داده‌ها می‌توانیم از WordCloud استفاده کنیم تا میزان فراوانی یک کلمه را در مقیاس و مقایسه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

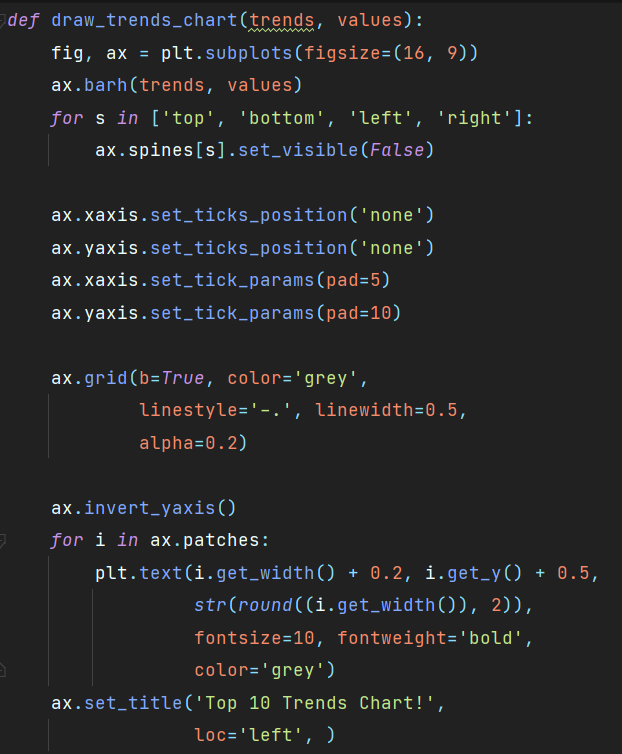


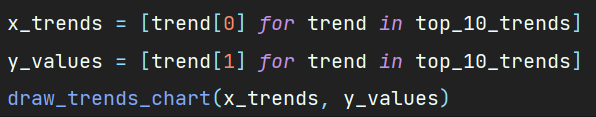
*برای به دست آوردن ترندها باید به دنبال هشتگ‌ها در توییت‌ها بگردیم که این کار را خیلی راحت با استفاده از عبارات منظم انجام داده‌ایم. به این صورت که تمام عباراتی که شامل هشتگ بودند را داخل یک لیست ذخیره کردیم تا در انتها با استفاده از Counter بتوانیم 10 ترند اول را شناسایی کنیم.*



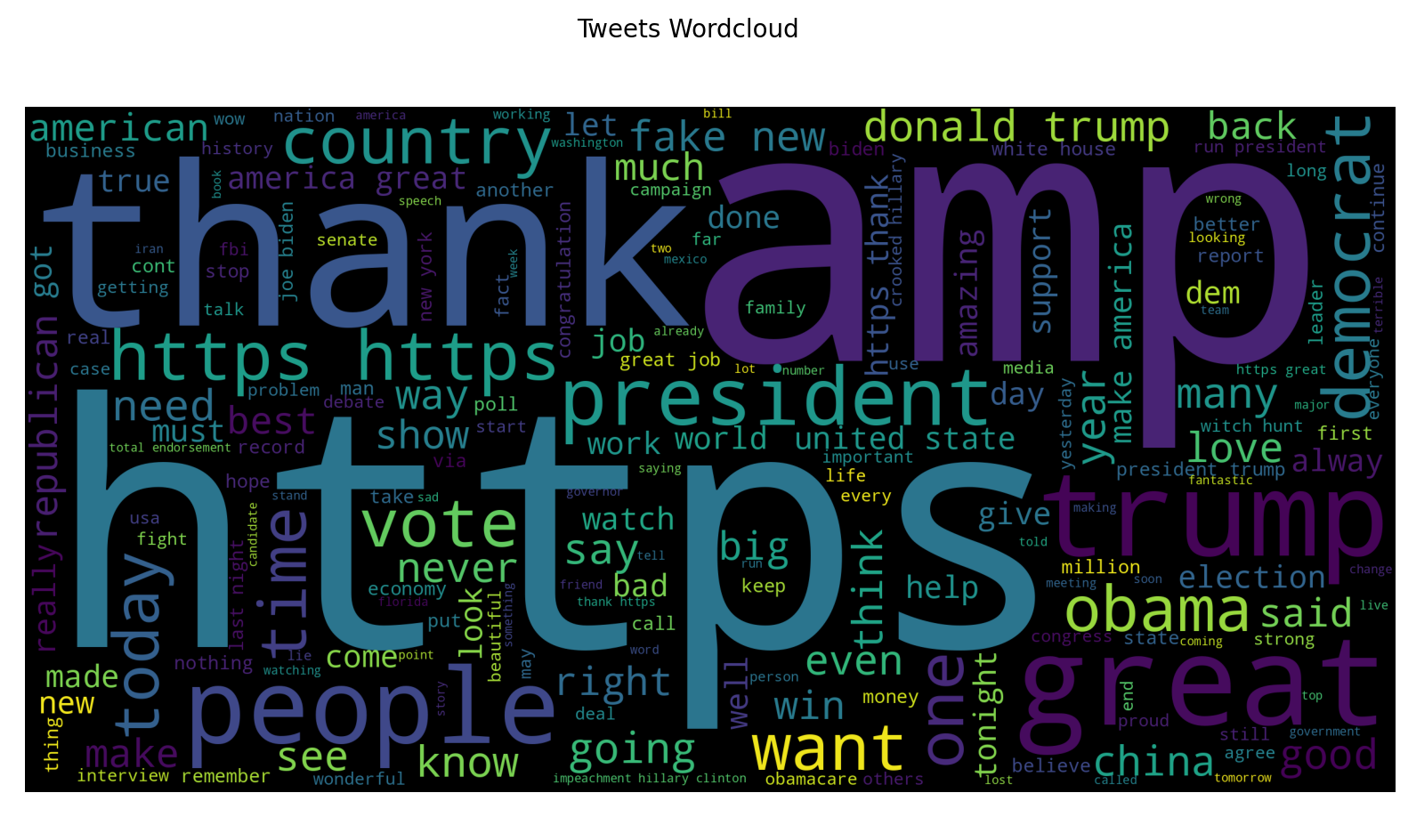


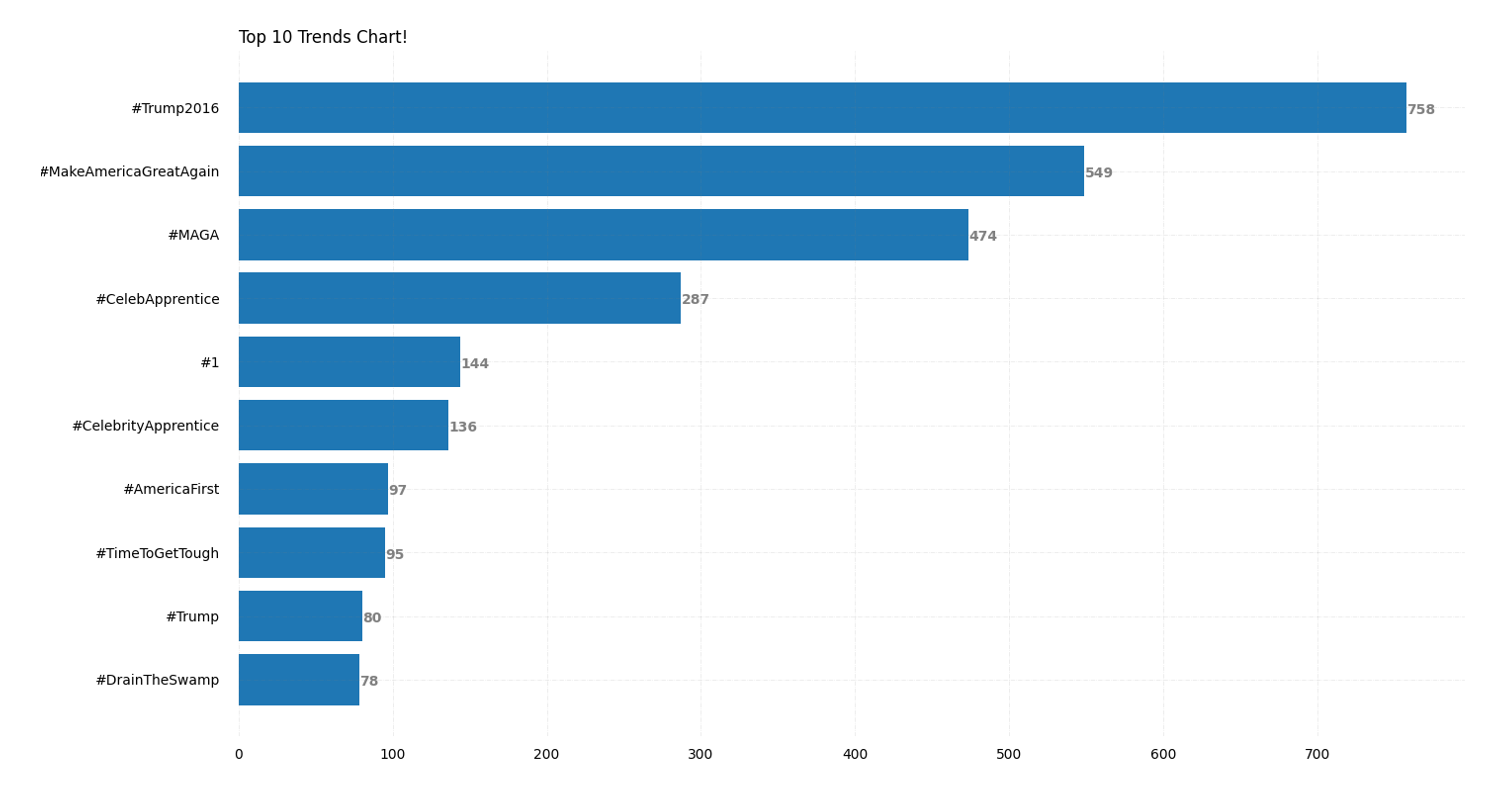
*در انتها در تابع draw\_trends\_chart به ترسیم نمودار مربوط به فراوانی ترندها پرداخته‌ایم.*



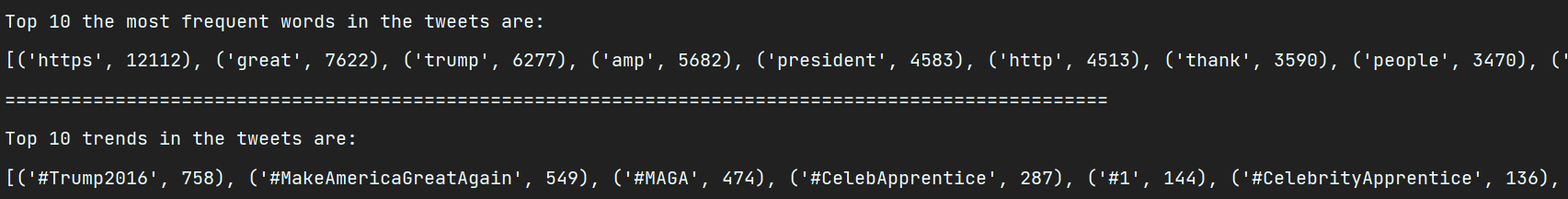


*نمودار ترسیم‌شده برای ترندها و Wordcloudبه‌دست‌آمده به صورت زیر خواهند بود:*

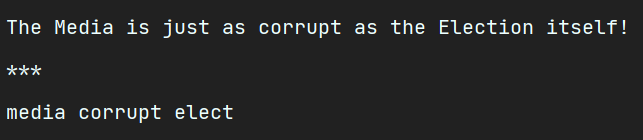


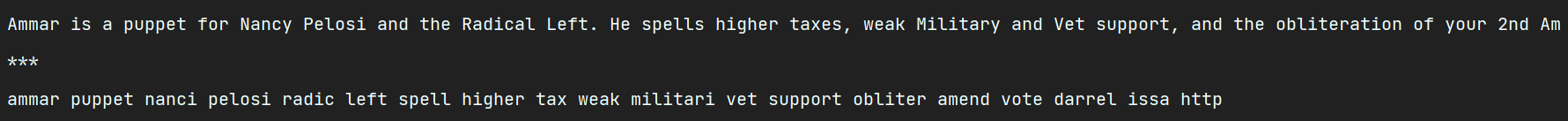


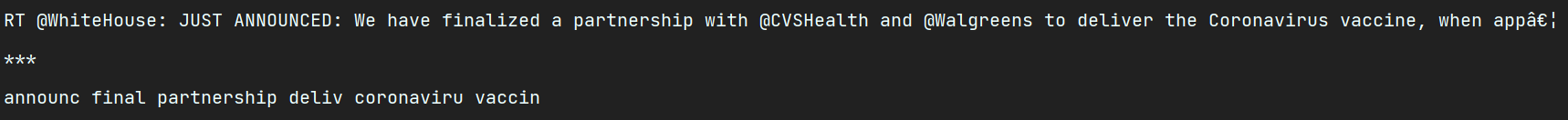
*در انتهای اجرای برنامه، 10 کلمه پرکاربرد و 10 ترند مربوطه در داخل ترمینال نیز چاپ شده‌اند.*



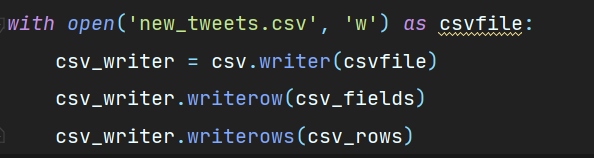
*سه نمونه از توییت‌ها پیش و پس از فرایند پیش‌پردازش:*







*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.*



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

