Massive Data Analytics

Home Work 1

Ilia Hashemi Rad

99102456

Section 1: Reading the data

Explanations:

Running the code section of Reading the data subsection, we result in one sample of the `news_rdd` as below:

```
['{"body": "ایرنا نوشت: با همکاری مشترک گروه زبان و ادبیات فارسی دانشگاه پکن و رایزنی فرهنگی جمهوری اسلامی -\\n\\n.نشست بزرگداشت روز جهانی حافظ شیرازی 20 مهرماه در دانشگاه پکن برگزار می شود
در این نشست، محسن بختیار، سفیر و نعمت الله ایران زاده، رایزن فرهنگی\\n\\n.ایران در چین، نشست بزرگداشت روز جهانی حافظ شیرازی روز پنجشنبه 20 مهرماه ساعت 10:30 تا 12 در دانشگاه پکن برگزار می شود
خانم\\n\\n.جمهوری اسلامی ایران در چین، پروفسور چن مینگ، رییس محترم دانشکده زبان های خارجی دانشگاه پکن، بهادر باقری، استاد مدعو زبان و ادبیات فارسی دانشکده زبان های خارجی دانشگاه پکن، سخنرانی می کنند
وانگ یی دان، استاد رشته زبان و ادبیات فارسی دانشگاه پکن، شی گوانگ، استاد رشته زبان و ادبیات فارسی و مدیر مرکز پژوهش های فرهنگ ایران در دانشگاه پکن و لیو بینگ جون، استاد و مدیر گروه زبان و ادبیات فارسی دانشگاه
پکن از دیگر سخنرانان این نشست هستند.", "image_title_url": "https://ettelaat.com/files/fa/news/1402/7/20/82679_282.jpg", "language": "fa", "source": "موتور جستجوی قطره",
"title": "نشست بزرگداشت روز جهانی | ادای احترام چینی ها به حافظ+عکس", "date_published": 1697070054, "uid": "7de950ad0cf5f22ed1d740feb", "url":
"https://www.ghatreh.com/news/nn14020701825494745088/ادای-احترام-چینی-حافظ-عکس", "crawler_timestamp": 1697070106, "ingestor_timestamp": 1697086977, "summary": "نشست
بزرگداشت روز جهانی حافظ شیرازی 20 مهرماه در دانشگاه پکن برگزار می شود. - ایرنا نوشت: با همکاری مشترک گروه\xa0 ", "hostname": "ghatreh.com", "parser_categories": [], "keywords": ["رییس
محترم دانشکده زبان\u200cهای خارجی دانشگاه پکن", "استاد رشته زبان و ادبیات فارسی", "نشست بزرگداشت روز جهانی حافظ شیرازی", "رایزن فرهنگی جمهوری اسلامی ایران"], "parser_keyword": ["ادبیات
", "فارسی", "دانشگاه", "جمهوری اسلامی ایران", "ادبیات", "زبان", "فارسی", "جمهوری اسلامی"], "author": "حسن قلي پور خطير", "processed_body": "نشست بزرگداشت روز جهانی حافظ شیرازی 20
ایرنا نوشت: با همکاری مشترک گروه زبان و ادبیات فارسی دانشگاه پکن و رایزنی فرهنگی جمهوری اسلامی ایران در چین، نشست بزرگداشت روز جهانی حافظ شیرازی روز -\\n\\n.مهرماه در دانشگاه پکن برگزار می شود
در این نشست، محسن بختیار، سفیر و نعمت الله ایران زاده، رایزن فرهنگی جمهوری اسلامی ایران در چین، پروفسور چن مینگ، رییس محترم\\n\\n.پنجشنبه 20 مهرماه ساعت 10:30 تا 12 در دانشگاه پکن برگزار می شود
خانم وانگ یی دان، استاد رشته زبان و ادبیات فارسی دانشگاه پکن\\n\\n.دانشکده زبان های خارجی دانشگاه پکن، بهادر باقری، استاد مدعو زبان و ادبیات فارسی دانشکده زبان های خارجی دانشگاه پکن، سخنرانی می کنند
شی گوانگ، استاد رشته زبان و ادبیات فارسی و مدیر مرکز پژوهش های فرهنگ ایران در دانشگاه پکن و لیو بینگ جون، استاد و مدیر گروه زبان و ادبیات فارسی دانشگاه پکن از دیگر سخنرانان این نشست هستند.",
"categories": ["politics"], "ner_tags": [{"type": "NORP", "entity": "چینی"}, {"type": "EVE", "entity": "نشست بزرگداشت روز جهانی حافظ شیرازی"}, {"type": "FAC", "entity":
"دانشگاه پکن"}, {"type": "ORG", "entity": "ایرنا"}, {"type": "EVE", "entity": "نشست بزرگداشت روز جهانی حافظ شیرازی"}, {"type": "FAC", "entity": "دانشگاه پکن"}, {"type": "ORG",
"entity": "ایرنا"}, {"type": "ORG", "entity": "گروه زبان و ادبیات فارسی دانشگاه پکن و رایزنی فرهنگی جمهوری اسلامی ایران"}, {"type": "GPE", "entity": "چین"}, {"type": "EVE", "entity":
"نشست بزرگداشت روز جهانی حافظ شیرازی"}, {"type": "FAC", "entity": "دانشگاه پکن"}, {"type": "PER", "entity": "محسن بختیار"}, {"type": "POST", "entity": "سفیر"}, {"type":
"PER", "entity": "نعمت الله ایران زاده"}, {"type": "POST", "entity": "رایزن فرهنگی جمهوری اسلامی ایران"}, {"type": "GPE", "entity": "چین"}, {"type": "PER", "entity": "پروفسور چن
مینگ"}, {"type": "POST", "entity": "رییس محترم دانشکده زبان\u200cهای خارجی دانشگاه پکن"}, {"type": "PER", "entity": "بهادر باقری"}, {"type": "POST", "entity": "استاد مدعو زبان و ادبیات
فارسی دانشکده زبان\u200cهای خارجی دانشگاه پکن"}, {"type": "PER", "entity": "وانگ یی دان"}, {"type": "POST", "entity": "استاد رشته زبان و ادبیات فارسی دانشگاه پکن"}, {"type": "PER",
"entity": "شی گوانگ"}, {"type": "POST", "entity": "استاد رشته زبان و ادبیات فارسی"}, {"type": "POST", "entity": "مدیر مرکز پژوهش\u200cهای فرهنگ ایران"}, {"type": "FAC", "entity":
"دانشگاه پکن"}, {"type": "PER", "entity": "لیو بینگ جون"}, {"type": "POST", "entity": "استاد"}, {"type": "POST", "entity": "مدیر گروه زبان و ادبیات فارسی دانشگاه پکن"}]}']
```

Explanations:

Looking the sample we realize:
The news are in JSON format and have many different fields. The most important one of them are:

- **"body":** The main content of the news article. It contains the text of the article.
- **"source":** The source of the news article, in this case,"Qatre News Search Engine".
- **"title":** The title of the news article.
- **"date__published":** The timestamp representing the date and time when the article was published. It's in Unix timestamp format (seconds since January 1, 1970).
- **"uid":** A unique identifier for the news article.
- **"url":** The URL of the news article.
- **"crawler__timestamp":** The timestamp indicating when the article was crawled or fetched by a web crawler.
- **"ingestor__timestamp":** The timestamp indicating when the article was ingested or processed.
- **"summary":** A summary or snippet of the article.
- **"hostname":** The hostname of the website where the article was published.
- **"parser__categories":** Categories assigned to the article by a parser. In this sample, it seems to be an empty list.
- **"keywords":** Keywords associated with the article.(Some of them are empty as we'll see further.)
- **"parser__keyword":** Keywords assigned by a parser.
- **"author":** The author of the article.
- **"processed__body":** A processed version of the article body.
- **"categories":** Categories to which the article belongs. In this sample, it's labeled as "politics."
- **"ner__tags":** Named Entity Recognition (NER) tags, providing information about named entities in the article, such as people, organizations, locations, etc.

Section 2: Preprocessing

```python
# Parse the JSON lines and extract the body text
parsed_news_rdd = news_rdd.map(lambda line: json.loads(line)['body'])

# Define a function to remove useless characters including '\n', '\u200c', and '\\n'
def remove_useless_characters(text):
    # Remove special characters, digits, and unwanted unicode characters
    text = re.sub(r'\\|\\n|\\u200c|\n', '', text)

    # Remove non-Persian characters and digits
    text = re.sub(r'[^-\s]', ' ', text)

    # Define a regular expression pattern that matches one or more spaces
    pattern = re.compile(r" +")
    # Apply the pattern to the text and replace the matches with a single space
    text = pattern.sub(" ", text)
    return text

# Define a list of Farsi stop words
def load_stop_words(file_paths):
    stop_words_set = set()
    for file_path in file_paths:
        with open(file_path, 'r', encoding='utf-8') as file:
            stop_words_set.update(file.read().splitlines())
    return stop_words_set

# List of paths to your stop words files
stop_words_files = ['verbal.txt', 'persian.txt', 'short.txt', 'chars.txt', 'nonverbal.txt']

# Load stop words from files
stop_words = load_stop_words(stop_words_files)

# Remove useless characters
clean_news_rdd = parsed_news_rdd.map(remove_useless_characters)

# Remove the stop words from clean_news_rdd without splitting the text
clean_news_rdd = clean_news_rdd.map(lambda x: " ".join([w for w in x.split() if w not in stop_words]))

# Process the cleaned news RDD by splitting
processed_news_rdd = clean_news_rdd.flatMap(lambda text: text.split()) \
                                   .filter(lambda word: word not in stop_words)

# Take a random sample from the cleaned news RDD
sample_clean_news = clean_news_rdd.takeSample(False, 1)

# Print the sample news
print('One sample of clean_news_rdd is:')
print(sample_clean_news)

print()

# Take 10 random samples from the processed news RDD
sample_processed_news = processed_news_rdd.takeSample(False, 10)

# Print the sample news
print('Ten samples of processed_news_rdd is:')
print(sample_processed_news)
```

Explanations:

In this code, I considered everything manually as explained further. Because, at first tested the `hazm` library to do so, but that was not working well on this data. Here is a brief explanation from what I did:

1. **Parsing JSON and Extracting Body Text:**
    - I used the `map` transformation to parse each JSON line in the RDD and extract the "body" text.
2. **Removing Useless Characters:**
    - I defined a function, `remove_useless_characters`, to clean the text by removing special characters, unwanted Unicode characters, non-Persian characters, and extra spaces.
    - Then, applied the defined function to each element in the RDD using the `map` transformation.
3. **Loading Stop Words:**
    - At first I define a list of file paths containing Persian stop words downloaded from GitHub.
    - Then, loaded stop words from the files into a set using the `load_stop_words` function.
4. **Removing Stop Words:**
    - I used the `map` transformation to remove stop words from each element in the RDD. Stop words are removed without splitting the text.

5. **Processing Cleaned News RDD by Splitting:**
   - I used, the `flatMap` transformation to split each cleaned news text into words. This RDD is for having a split cleaned text with separated words for further uses.
   - Then, applied a `filter` transformation to remove stop words.(this was not necessary because that was done before.)

   Here is one sample of `clean_news_rdd`:

```
['شماره مجله نامه جمهور ویژهنامهای برنامه هفتم توسعه نقدنامهای باب نابرنامه هفتم توسعه ضدعدالت ضدتوسعه ضد جامعه سه فصل منتشر گزارش خبرگزاری مهر شماره مجله نامه جمهور ویژهنامهای برنامه هفتم توسعه تیتر
نقدنامهنمامهای باب نا برنامه هفتم توسعه ضد عدالت ضد توسعه ضد جامعه سه فصل منتشر بررسی نقد متن فرامتن برنامه هفتم توسعه منظر اجتماعی میپردازد سرمقاله شماره قلم سردبیر مجله مجتبی نامخواه پرسش آغاز زمانهای
سمت جامعه سمت رهبری انقلاب مطالبهای فراگیر پر شکافهای طبقاتی بهنفع سندِ نامقبولی ضد عدالت اجتماعی تدوین مقدمه متن میانه سوتفاهمی میبریم میتوان آشکارترین تفوق رشد اقتصادی عدالت اجتماعی تئوریزه مسئولیت
ناپذیرترین برنامههای توسعه باب سیاستهای اجتماعی حمایتی نوشت سرمقاله ادامه سه مبحث مجزا بررسی برنامه هفتم توسعه میپردازد مبحث سه تضاد اساسی موجود متن لایحه برنامه پرداخته وجوه ضد عدالت ضد توسعه ضد جامعه
موجود بررسی عناوین ضد عدالت تفوق رشد اقتصادی عدالت اجتماعی بندهای ضد عدالت اجتماعی بیتوجهی مسئله نابرابری نابرنامه جاده صاف کاتالاکسی بررسی وجوه ضد توسعه برنامه هفتم ذیل عناوین صورتبندی دولت بهمثابه
نهادی مستقل جامعه عقل پیشرفت فقدان بازاندیشی توسعه بهمثابه تأمین منابع تیترهای فقدان سیاست اجتماعی غفلت نیروی کار تفوق سرمایه بازگشت ایده جراحی شوک درمانی کالایی حقوق اساسی ابعاد ضد جامعه برنامه
میپردازد مبحث سرمقاله سه کلمه تأمل فرامتن برنامه هفتم توسعه پرداخته نقد ناکارآمدی ادبیات تولید الگوی اسلامی ایرانی پیشرفت بحث تاریخ تطور وضعیت کنونی صورتبندی مسئله عقبماندگی مرور نهایت ضرورت برآمدن نیروی
سیاسی معطوف سیاست اجتماعی گوش زد مبحث سرمقاله تاریخچهای مختصر مصور سال تاریخ برنامه نویسی اسناد توسعه ایران انقلاب اسلامی کادرهایی مستقل دنبالهدار متن سرمقاله درج میدهد میتوان جمع تاریخ برنامهریزی
توسعه ایران تاریخ برنامهریزیِ مسئولیتناپذیر ضد عدالت اجتماعی نیروهای جامعه علی اسکندری دبیر تحریریه نامه جمهور بررسی ابعاد ضد اجتماعی برنامه هفتم متنی آشوبنامه پاد جامعه نوشته سه نابرنامه برنامههایی نکردن ناگفتهها
برنامه نیامده مهمتر فجایع فاجعهای برگشت ناپذیر زمینهها نتایج اجتماعی اجرای برنامه بحث نویسنده جمعبندی یادداشت رمز عملیات برنامه هفتم توسعه واگذاری دولت نهادهای عمومی دانسته بیانی هشدار نتیجه اجرای برنامه میداند
دولت سیزدهم آخرین دولت ایران کیوان سلیمانی عضو تحریریه نامه جمهور مطلبی عدالت بهسود سرمایهداری ساختار پیامد برنامه هفتم تنبیه تهیدستان تشویق نوکیسهگان میداند مینویسد لایحه برنامه هفتم توسعه حیث مکتب مسلط
منطق برنامه ضعیف منظر بیپروایی رویکردها سیاستهای مساله زا ضد عدالت بیپروا برنامه ضد مقوله عدالت زندگی جامعه انسانیت عقلانیت دولت کارگران کلیت برنامه ذیل مکتب سرمایهداری رفاقتی منطق بازاری حمایت طبقه مرفه
صورتبندی فصل مجله نطقها گفتوگوی تفصیلی برنامه هفتم گفتوگو حجت الاسلام والمسلمین محسن قنبریان مسئله افول تدریجی عدالت برنامههای توسعه پرداخته بررسی قانون اساسی عدالت اجتماعی سیر قهقرایی عدالت برنامههای
توسعه سیاستهای ابلاغی نیم دهه واکاوی ادامه فصل گفتوگو دکتر حسن سبحانی مسئله توسعهنیافتگی توسعه برنامه اسناد توسعه ایران انقلاب بحث فصل مجله مذاکرات حجت الاسلام المسلمین سیدعلی موسوی مدرس سطح
عالی حوزه علمیه قم رییس پژوهشکده باقرالعلوم السلام نگاهی فقهی حقوقی ابعاد ماده برنامه هفتم توسعه میپردازد بررسی مردمی اقتصاد نهادینه تداول ثروت اغنیا پایان پیشنهادهایی بازسازی برنامه هفتم ختم نویسنده مقاله بررسی
سیر تصویب اصول اقتصادی قانون اساسی بنیادهای فقهی مینویسد انقلابی ضد سرمایهداری جامعه کمرنگ ذیل اصل مستمسکی تُرک تازی لیبرال سرمایهداری ایران گردید روزها خبرگان قانون اساسی فکرش نمیکردند قدرت لیبرال
سرمایهداری تصور ابتدایی غلبه مفاصل دنیا سیاستهای طاغوتیاش میبرد دنبال ایران نقطه مقاومت نظام طاغوتی لیبرال سرمایهداری جهانی زانو درآورد خلاصی هِیمنه طاغوت خلق راهگشای نمونهای الگوی پیشرفته اسلامی مبتنی
ارزشهای اساسی انقلاب محوریت عدالت اساس نویسنده پیشنهاد میدهد توجه محور نبودن عدالت برنامه هفتم مینویسد بیراه تصویب برنامه تأخیر بیافتد ادامه بررسیهای فصل نویسندگان نامه جمهور بخشهای برنامه هفتم بحث بررسی
گذاشتهاند رقیه فاضل دکتری مدیریت آموزشی مطلبی عدالت نمایشی بازتولید تبعیض آموزشی برنامه هفتم توسعه پرداخته میثم ظهوریان دکتری مدیریت برنامههای توسعه کلیت برنامه نویسی نظام سیاستگذاری نقد کشیده دچار
خیالپردازیهای بیثمر خوانده محسن جبارنژاد دکتری علوم سیاسی مطلبی آینده دولت وجوه اساسی بازنمایی دولت لایحه برنامه هفتم توسعه بررسی علی مصدق رویکرد برنامه هفتم توسعه نسبت مسکن شهرسازی نقد روحالله ایزدخواه
عضو کمیسیون تلفیق برنامه هفتم مجلس شورای اسلامی مطلبی ناترازی نابرابری نکته برنامه هفتم مطرح حاشیه گفتمان پیشرفت عدالت افراط دولت مجلس تمرکز مساله ناترازی انتقاد پایان محمد اسکندری کارشناس مدیریت منابع آب
نقدهایی سیاستهای مدیریت منابع آب لایحه برنامه هفتم توسعه مطرح مجله صفحه قیمت تومان منتشر']
```

Explanations:

Here is ten samples of `processed_news_rdd`:

```
['افتاده', 'صحبتهایی', 'مهران', 'بیان', 'اخراج', 'پرده', 'ارزش', 'مدیرعامل', 'تحمیلی', 'رسانهای']
```

Explanations:

As you can see in the outputs, the body part of the news is cleaned completely, all the unwanted characters, extra spaces and stop words are cleaned; and now, we have a text containing important words with a space between.(in the `clean_news_rdd`)

Section 3: Exploration

Title and URL of 5 Longest News

```python
    # Parse the JSON lines to extract the title, url, and body length
titles_urls_length_rdd = news_rdd.map(lambda line: json.loads(line)) \
                              .map(lambda news: (news['title'], news['url'], len(news['body'])))

# Get the top 5 longest news by body length
top_5_longest_news = titles_urls_length_rdd.takeOrdered(5, key=lambda x: -x[2])

# Print the titles and urls of the 5 longest news
for title, url, length in top_5_longest_news:
    print(f"Title: {title}, URL: {url}, Length: {length}")
```

Explanations:

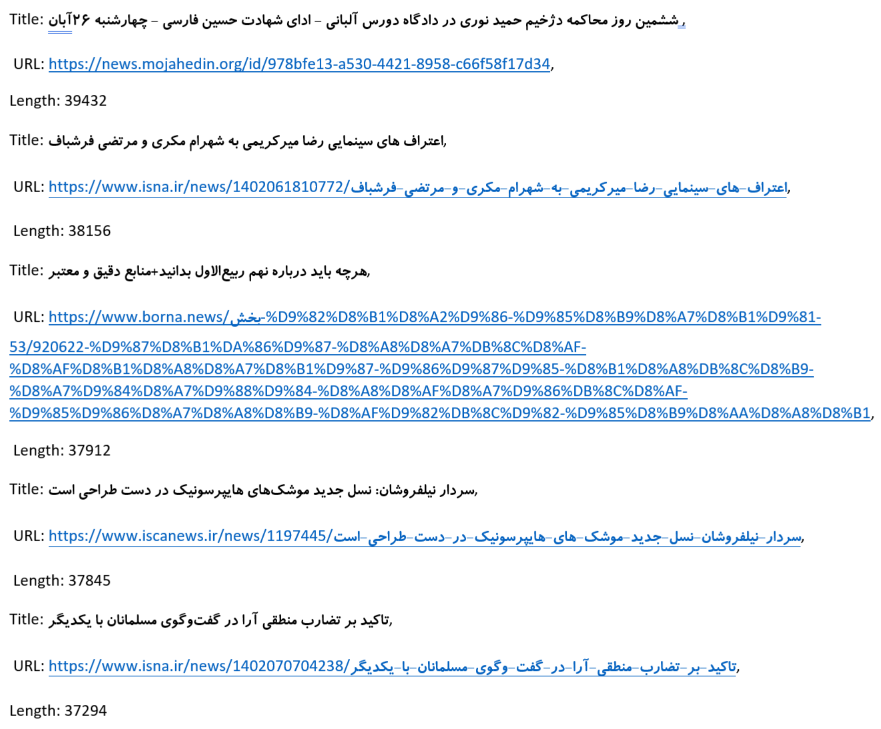
Here is a brief explanation from what I did:

- **Parsing JSON Lines and Extracting Title, URL, and Body Length:**
  - I used the `map` transformation to parse each JSON line in the RDD and extract the title, URL, and length of the body.
  - The lambda function within the `map` transformation converts each JSON line into a tuple containing

the title, URL, and the length of the body.

- **Top 5 Longest News by Body Length:**
  - I used the `takeOrdered` action to retrieve the top 5 longest news articles based on body length.
  - The `key` parameter is set to `-x[2]` to order the articles in descending order of body length, the third field that is parsed.

The output is:



```
Title: ششمین روز محاکمه دژخیم حمید نوری در دادگاه دورس آلبانی – ادای شهادت حسین فارسی – چهارشنبه ۲۶آبان,

 URL: https://news.mojahedin.org/id/978bfe13-a530-4421-8958-c66f58f17d34,

Length: 39432

Title: اعتراف های سینمایی رضا میرکریمی به شهرام مکری و مرتضی فرشباف,

 URL: https://www.isna.ir/news/1402061810772/اعتراف-های-سینمایی-رضا-میرکریمی-به-شهرام-مکری-و-مرتضی-فرشباف,

 Length: 38156

Title: هرچه باید درباره نهم ربیع‌الاول بدانید+منابع دقیق و معتبر,

 URL: https://www.borna.news/بخش-%D9%82%D8%B1%D8%A2%D9%86-%D9%85%D8%B9%D8%A7%D8%B1%D9%81-
53/920622-%D9%87%D8%B1%DA%86%D9%87-%D8%A8%D8%A7%DB%8C%D8%AF-
%D8%AF%D8%B1%D8%A8%D8%A7%D8%B1%D9%87-%D9%86%D9%87%D9%85-%D8%B1%D8%A8%DB%8C%D8%B9-
%D8%A7%D9%84%D8%A7%D9%88%D9%84-%D8%A8%D8%AF%D8%A7%D9%86%DB%8C%D8%AF-
%D9%85%D9%86%D8%A7%D8%A8%D8%B9-%D8%AF%D9%82%DB%8C%D9%82-%D9%85%D8%B9%D8%AA%D8%A8%D8%B1,

 Length: 37912

Title: سردار نیلفروشان: نسل جدید موشک‌های هایپرسونیک در دست طراحی است,

 URL: https://www.iscanews.ir/news/1197445/سردار-نیلفروشان-نسل-جدید-موشک-های-هایپرسونیک-در-دست-طراحي-است,

 Length: 37845

Title: تاکید بر تضارب منطقی آرا در گفت‌وگوی مسلمانان با یکدیگر,

 URL: https://www.isna.ir/news/1402070704238/تاکید-بر-تضارب-منطقی-آرا-در-گفت-وگوی-مسلمانان-با-یکدیگر,

Length: 37294
```

Explanations:

That is as expected and is sorted by the news length.

20 Most frequent Words

```python
import matplotlib.pyplot as plt
from bidi.algorithm import get_display
from arabic_reshaper import reshape

# Determining the RDD which contains the frequency of words
word_counts_rdd = processed_news_rdd.map(lambda word: (word, 1)) \
                                    .reduceByKey(lambda a, b: a + b)

# Get the top 20 most frequent words
top_20_words = word_counts_rdd.takeOrdered(20, key=lambda x: -x[1])

# Plot the distribution of these words
words, counts = zip(*top_20_words)
persian_words = [get_display(reshape(word)) for word in words]

plt.bar(persian_words, counts)
plt.xticks(rotation=90)
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 20 Most Frequent Words')
plt.show()
```

Explanations:

Here is a brief explanation from what I did:

- **Determining RDD for Word Frequencies:**
  - I used the `map` transformation to convert each word in the processed news RDD into a tuple of the

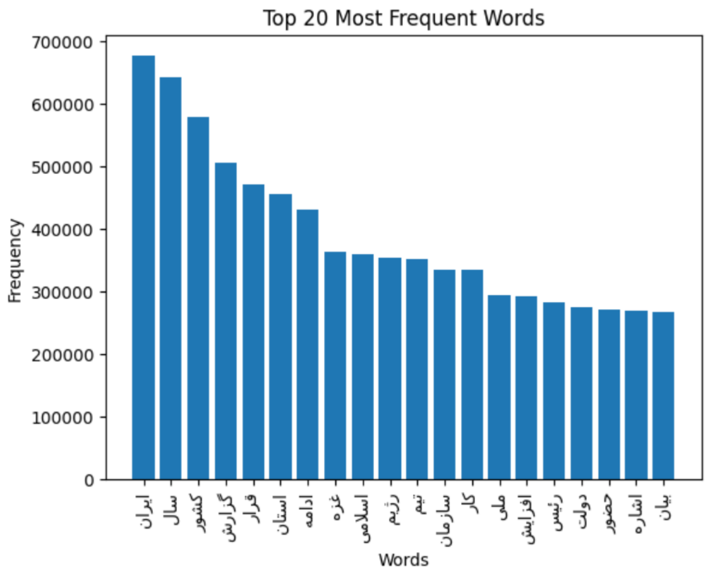
form `(word, 1)`.

  - then, applied the `reduceByKey` transformation to calculate the frequency of each word.

- **Top 20 Most Frequent Words:**
  - I used the `takeOrdered` action to retrieve the top 20 most frequent words based on their frequencies.
  - The `key` parameter is set to `-x[1]` to order the words in descending order of frequency.

- **Plotting the Distribution of Words:**
  - I unpacked the top 20 words and their frequencies into separate lists (`words` and `counts`).
  - Then, used the `get_display` function from `bidi.algorithm` and the `reshape` function from `arabic_reshaper` to handle the correct display of Persian words in the plot.

The output is:



Explanations:

That is as expected and is sorted by frequency. All the results are reasonably according to the Persian news.

The WordCloud of News

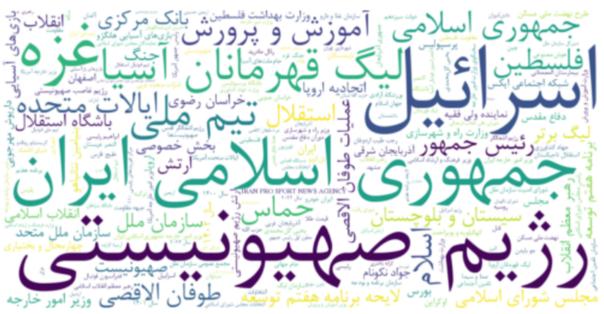
```python
from wordcloud import WordCloud
import math

# Extract keywords and their counts, checking for the existence of the 'keywords' key (keyword fields that are empty are ignored)
keywords_rdd = news_rdd.flatMap(lambda line: json.loads(line).get('keywords', [])) \
                        .map(lambda keyword: (keyword, 1)) \
                        .reduceByKey(lambda a, b: a + b)

# Calculate scores for word clouds
word_cloud_scores_rdd = keywords_rdd.map(lambda x: (x[0], 2 ** (math.log10(x[1]))))

# Normalize scores
max_score = word_cloud_scores_rdd.max(key=lambda x: x[1])[1]
word_cloud_scores_normalized_rdd = word_cloud_scores_rdd.map(lambda x: (x[0], x[1] / max_score))

# Collect keywords and their normalized scores
word_cloud_data = word_cloud_scores_normalized_rdd.collect()

font_path = '/usr/share/fonts/truetype/farsifonts-0.4/nazli.ttf'

# Create a WordCloud object with specified parameters
wordcloud = WordCloud(width=1200, height=600, background_color='white', font_path=font_path)

# Generate the word cloud from the collected data
wordcloud.generate_from_frequencies(dict(word_cloud_data))

# Display the word cloud
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Explanations:

Here is a brief explanation from what I did:

- **Extracting Keywords and Counts:**
  - I used the `flatMap` transformation to extract keywords from each JSON line in the RDD, considering only non-empty keyword fields.
  - I applied the `map` transformation to assign a count of 1 to each keyword.
  - Then, used the `reduceByKey` transformation to calculate the total count for each keyword.
- **Calculating Scores for Word Clouds:**
  - I used the `map` transformation to calculate scores for word clouds based on the formula $2^{\log_{10}(count)}$.
- **Normalizing Scores:**
  - I found the maximum score using the `max` action with the `key` parameter.
  - I used the `map` transformation to normalize scores by dividing each score by the maximum score.
- **Collecting Keywords and Normalized Scores:**
  - I used the `collect` action to retrieve the keywords and their corresponding normalized scores, making `word_cloud_data_rdd` as the RDD to save the word cloud of Keywords in order.
- **Creating WordCloud Object:**
  - I specified the font path for displaying Persian text (`font_path`) which had to be a TrueType font.
  - Then, created a `WordCloud` object with parameters such as width, height, background color, and font path.
- **Generating and Displaying Word Cloud:**
  - Finally, used the `generate_from_frequencies` method to generate the word cloud from the collected data.

The output is:



Explanations:

The output shows the frequency of keywords by their font size in the picture. Means as much the keyword is more frequent, the font size of that, is higher and is represented bigger.
This figure will be used for further inferences.

Total Count Of News Per Day

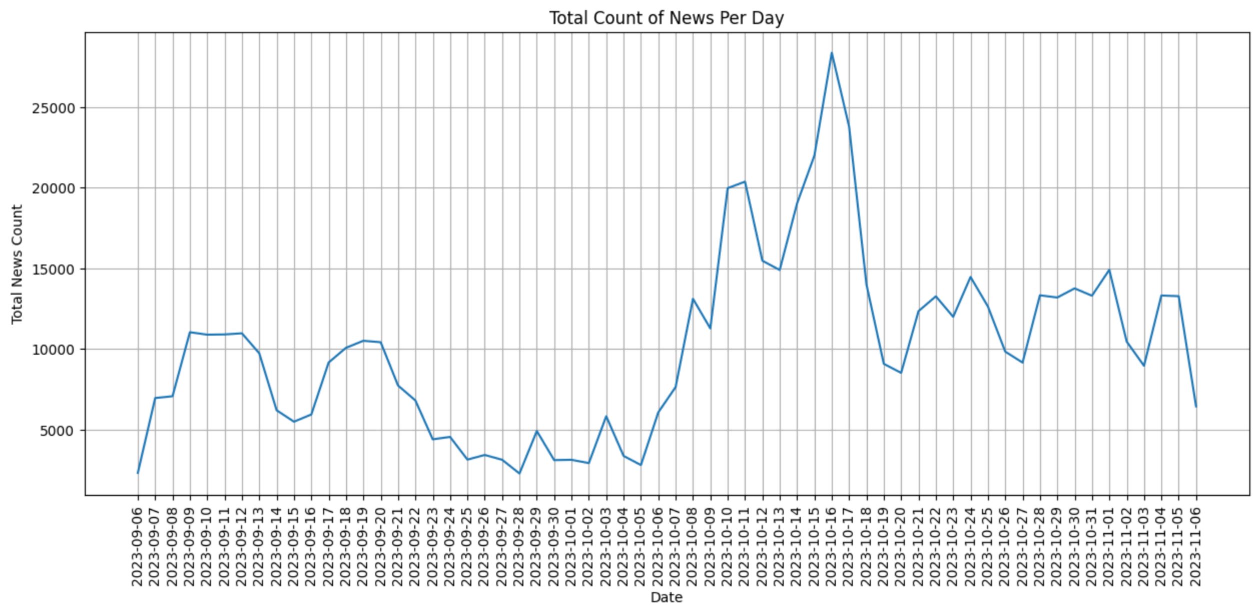
```
import time

# Convert Unix timestamp to human-readable date and count news per day
news_per_day_rdd = news_rdd.map(lambda line: json.loads(line)) \
                              .map(lambda news: (time.strftime('%Y-%m-%d', time.localtime(int(news['date_published']))), 1)) \
                              .reduceByKey(lambda a, b: a + b)

# Collect the data and sort by date
news_per_day = sorted(news_per_day_rdd.collect())

# Plot the timeline
dates, counts = zip(*news_per_day)
plt.figure(figsize=(15,6))
plt.plot(dates, counts)
plt.xticks(rotation=90)
plt.xlabel('Date')
plt.ylabel('Total News Count')
plt.title('Total Count of News Per Day')
plt.grid()
plt.show()
```

Explanations:

Here is a brief explanation from what I did:

- **Converting Unix Timestamp to Human-Readable Date and Counting News Per Day:**
  - I used the `map` transformation to parse each JSON line in the RDD and convert the Unix timestamp to a human-readable date (something as 2023/10/5, to can be used in the reduceByKey part because we want the same days have the same keys.)
  - Then, assigned a count of 1 to each news article.
  - Finally, applied the `reduceByKey` transformation to calculate the total count of news articles for each day.
- **Collecting and Sorting Data by Date:**
  - I used the `collect` action to retrieve the data containing dates and corresponding news counts.
  - Then, sorted the collected data by date.
- **Plotting the Timeline:**
  - At the end, unpacked the sorted data into separate lists (`dates` and `counts`) to use them for visualization of the result further.

The output is:



Explanations:

- The figure shows the total count of news per day from September 6, 2023 to November 6, 2023, using a line graph.

- The total count of news varies from day to day, ranging from around 5,000 to over 20,000.
- There is a clear peak middle October, 2023, where the total count of news reaches more than 20,000. This is the highest value in the entire period and will be explained further.
- After the peak, the total count of news declines gradually until the end of April, and then remains relatively stable in the first week of May.

Based on this analysis, I can infer that the total count of news per day is influenced by the occurrence and magnitude of significant events that capture the public interest. The Notre Dame fire was such an event that generated a lot of news coverage and discussion. Other factors that may affect the total count of news per day include the availability and reliability of news sources, the diversity and quality of news content, and the preferences and habits of news consumers.

Total Count Of News (which have at least one of the 20 word clouds) Per day

```
# Get the top 20 highest score keywords
top_20_keywords = sorted(word_cloud_data, key=lambda x: x[1], reverse=True)[:20]

# Broadcast the set of top 20 word cloud keywords to all workers
top_20_keywords_broadcast = sc.broadcast(set([keyword for keyword, _ in top_20_keywords]))

# Convert Unix timestamp to human-readable date and filter news with top 20 word clouds
news_with_top_word_clouds_per_day_rdd = news_rdd.map(lambda line: json.loads(line)) \
    .filter(lambda news: top_20_keywords_broadcast.value.intersection(news.get('keywords', []))) \
    .map(lambda news: (time.strftime('%Y-%m-%d', time.localtime(int(news['date_published']))), 1)) \
    .reduceByKey(lambda a, b: a + b)

# Collect the data and sort by date
news_with_top_word_clouds_per_day = sorted(news_with_top_word_clouds_per_day_rdd.collect())

# Plot the timeline
dates_with_top_word_clouds, counts_with_top_word_clouds = zip(*news_with_top_word_clouds_per_day)
plt.figure(figsize=(15,6))
plt.plot(dates_with_top_word_clouds, counts_with_top_word_clouds)
plt.xticks(rotation=90)
plt.xlabel('Date')
plt.ylabel('Total News Count with Top Word Clouds')
plt.title('Total Count of News Per Day with Top Word Clouds')
plt.grid()
plt.show()
```

Explanations:

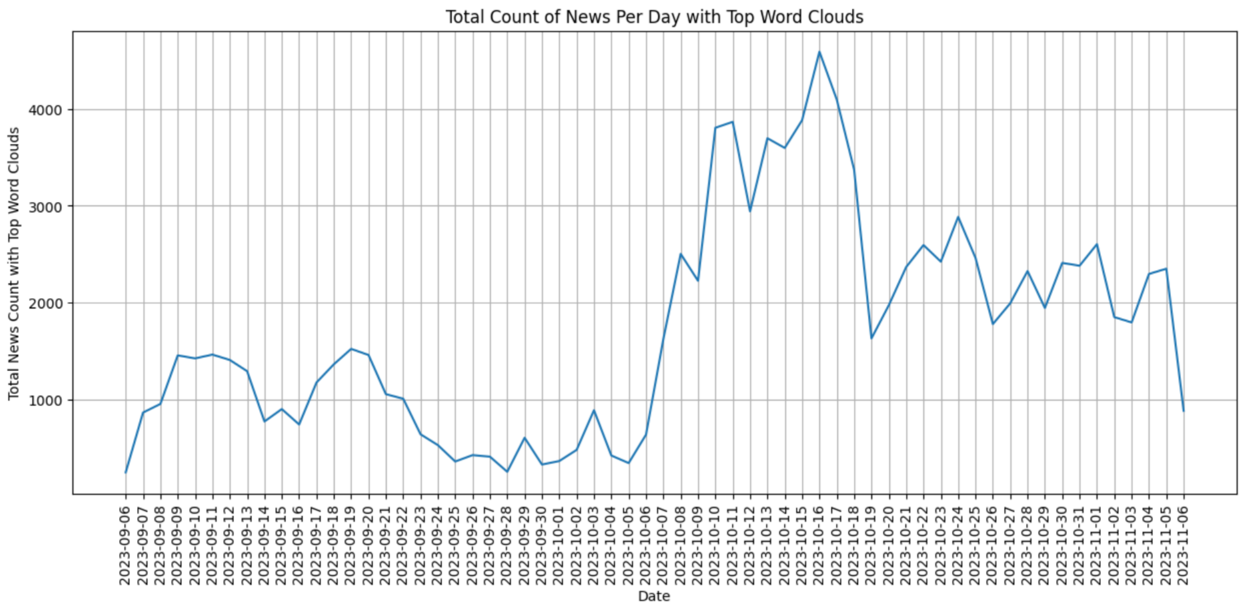
Here is a brief explanation from what I did: (Here, I just used 20 top word clouds as the question had said.)

- **Getting the Top 20 Highest Score Keywords:**
  - I used the `sorted` function to obtain the top 20 highest score keywords from the previously collected word cloud data.
  - Then, sorted the keywords based on their scores in descending order.
- **Broadcasting Top 20 Keywords to All Workers:**
  - I used the `sc.broadcast` function to broadcast the set of top 20 word cloud keywords to all Spark workers.
  - This enables efficient sharing of the top keywords across the distributed computing environment.
- **Filtering News with Top 20 Word Cloud Keywords:**
  - I used the `filter` transformation to select news articles that contain at least one of the top 20 word cloud keywords.
  - Then, applied the `map` transformation to convert the filtered news articles into tuples containing the date and a count of 1.
  - Finally, used the `reduceByKey` transformation to calculate the total count of news articles for each day.
- **Collecting and Sorting Data by Date:**

- I used the `collect` action to retrieve the data containing dates and corresponding news counts with top word clouds.
- Then, sorted the collected data by date.

- **Plotting the Timeline for News with Top Word Clouds:**
  - I unpacked the sorted data into separate lists (`dates_with_top_word_clouds` and `counts_with_top_word_clouds`).
  - Then, created a plot using `matplotlib.pyplot.plot` to visualize the timeline of news counts per day for articles containing top word cloud keywords.

The output is:



Explanations:

The timeline is almost like the previous timeline in general shape.

- The figure shows the total count of news per day that contains at least one word from the top 20 word clouds, from September 6, 2023 to December 6, 2023, using a line graph.
- The total count of news with top word clouds varies from day to day, ranging from around 500 to 4000.
- There is a clear peak around middle October, 2023, where the total count of news with top word clouds reaches almost 4000. This is the highest value in the entire period.
- After the peak, the total count of news with top word clouds declines sharply until the first week of December, and then remains relatively low.

Inference

Explanations:

Comparing the two figures, I can also infer that:

As it can be easily seen from the timelines, the general shape of both timelines and its ups and downs and its peaks are almost the same and they occur on the same dates and have only minor differences. This indicates that the overall shape of the timelines depends on the important news trends that can be seen in the most frequent news keywords or wordcloud. Because the changes in the volume of news are determined by important events, related to which so many news are published that their keywords are also included in the word cloud, and as a result, it creates a very high correlation between the general shape of these two timelines. This is normal because other news that are not trending have the same volume of publication on different days and only add a DC scale to the chart, which is clear from the difference in values on the two timelines.

On the other hand, The total count of news per day is generally higher than the total count of news per day with top word clouds, which means that there are many news articles that do not contain any of the top words.

First sharp growth in the news was in 7 October for the 'Al-Aqsa storm operation' of the Hamas group which caused a lot of news with the start of the war between Palestine and Israel and the successive attacks of Israel (as seen in the word clouds that many of the most frequent keywords are related to it, such as the Zionist regime, Israel, Gaza, Palestine, Hamas, Al-Aqsa storm operation and...)and generated a lot of news coverage and discussion.

Then, the second sudden growth in the news could be related to the killing of Dariush Mehrjooi, the famous director of Iranian cinema, who was killed exactly on October 14, and resulted a sharp increase in the number of news as it can be seen.

But the news related to other top word cloud keywords, probably do not have a special effect on the timeline. Because, for example, the 'seventh news development program' was broadcast around July, which does not fit into the timeline at all, and naturally, any news related to it is evenly distributed throughout the timeline. Or the fact that we see many keywords related to sports news is because they are a large and important part of sports news and naturally have a series of repeated keywords that do not have a special effect on the shape of the timeline.

Section 4: SON, A-priori algorithm

The Most Frequent 3-words Sequences

Explanations:

Here, for the purpose of finding the most frequent itemsets in this Massive data, I used a combination of two known algorithms, named *SON* and *Apriori* algorithms. Let me first explain this algorithms.

SON Algorithm

Explanations:

The SON algorithm is a distributed and scalable method for mining frequent itemsets in large datasets using a two-phase MapReduce approach. The algorithm efficiently breaks down the data into manageable sub-files, allowing the discovery of frequent itemsets. Here's a brief explanation:

Phase 1 (First MapReduce Phase):
Divide the large datafile into smaller sub-files.
Each sub-file is processed by a mapper (similar to the Apriori algorithm) to find frequent itemsets.
The first-phase-reducer aggregates frequent itemsets found in each sub-file.
Unique frequent itemsets are obtained as candidate itemsets.

Phase 2 (Second MapReduce Phase):
Run MapReduce again with the list of candidate itemsets.
The second-phase-mapper checks if each basket contains any candidate itemsets.
Frequencies of itemsets are computed and sent to the second-phase-reducer.

Unique itemsets across the entire dataset are counted to determine frequent itemsets.

The SON algorithm combines parallel processing capabilities with the optimization of the Apriori algorithm, making it effective for mining frequent itemsets in big data scenarios.

Apriori Algorithm

Explanations:

The Apriori algorithm, is a classic method for discovering frequent itemsets in datasets, especially applied in association rule mining. Here's a brief explanation:

1. **Initialization:** Scan the dataset to find 1-itemsets meeting the minimum support threshold.

2. **Iteration:**
Proceed iteratively, increasing itemset size at each step.
Generate candidate itemsets by joining frequent itemsets of the previous size.

3. **Pruning:**
Prune candidate itemsets with an infrequent subset, exploiting the Apriori property.

4. **Support Calculation:**
Scan the dataset to calculate support for remaining candidate itemsets.

5. **Termination:**
Continue until no additional frequent itemsets can be found.

The Apriori algorithm utilizes prior knowledge of frequent itemset properties, that's why we call it Apriori.

Now, I'm going to explain you how I used these two algorithms for the case of question. Because it has a slight difference with the frequent itemsets in general case, which is we want 3-words combinations which come together; Means continuous sequence of 3 words. At first see this diagram to know the combined workflow of the algorithms is:

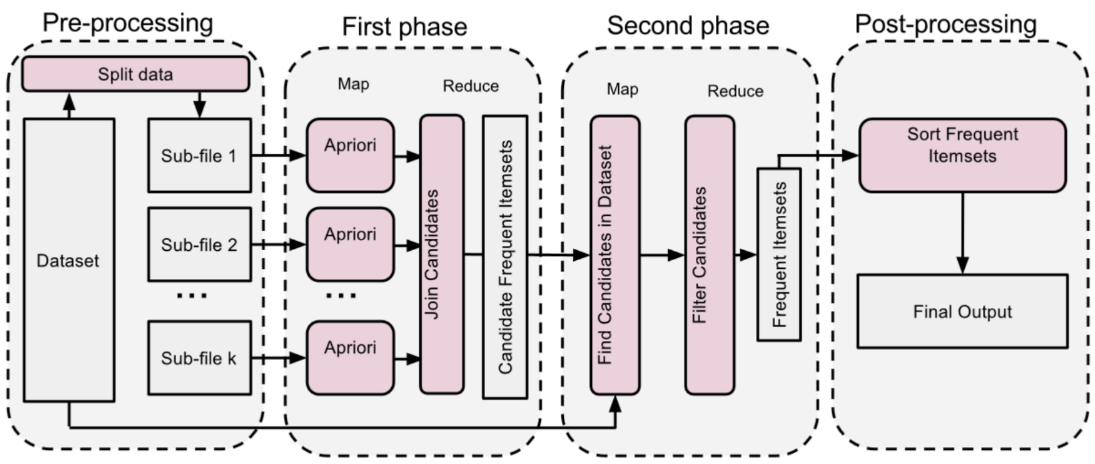


Fig. 1 - High level approach to our implementation of the SON MapReduce Algorithm

```
from nltk import ngrams

# A function to apply Apriori on a partition of the RDD
```

```python
def apriori(iterator, min_support, n):
    # A dictionary to store the counts of each n-gram
    counts = {}
    # A set to store the frequent n-grams
    frequent = set()
    # Iterate over each text in the partition
    for text in iterator:
        # Generate all n-grams from the text
        ngram_list = ngrams(text.split(), n)
        # Increment the count of each n-gram
        for ng in ngram_list:
            counts[ng] = counts.get(ng, 0) + 1
    # Filter out the n-grams that have at least min_support
    for ng, count in counts.items():
        if count >= min_support:
            frequent.add(ng)
    # Return the frequent n-grams as an iterator
    return iter(frequent)

# A function to apply the SON algorithm on the RDD
def son(rdd, min_support, n):
    # Get the number of partitions of the RDD
    num_partitions = rdd.getNumPartitions()
    # Apply Apriori on each partition with a lower support threshold
    candidates = rdd.mapPartitions(lambda x: apriori(x, min_support / num_partitions, n)).distinct()
    # Count the occurrences of each candidate in the entire RDD
    counts = rdd.flatMap(lambda x: ngrams(x.split(), n)).map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
    # Join the candidates with the counts and filter out those that do not meet the global support threshold
    frequent = candidates.map(lambda x: (x, None)).join(counts).filter(lambda x: x[1][1] >= min_support)
    # Return the frequent n-grams as an RDD
    return frequent

# Apply the SON algorithm on clean_news_rdd with min_support = 20000 and n = 3
result = son(clean_news_rdd, 20000, 3)

# Print the result
# Find the top 5 3-words phrases and their counts
top_5_phrases = result.takeOrdered(5, key=lambda x: -x[1][1])

# Print the top 5 3-words phrases and their counts
print("The top 5 3-words phrases and their counts are:")

# Iterate over the top 5 phrases and print each one along with its count
for i, (phrase, count) in enumerate(top_5_phrases, start=1):
    # Extract individual words from the tuple
    word1, word2, word3 = phrase

    # Print the phrase and its count
    print(f"{i}. '{word1} {word2} {word3}' with a repetition of {count[1]} times")
```

Explanations:

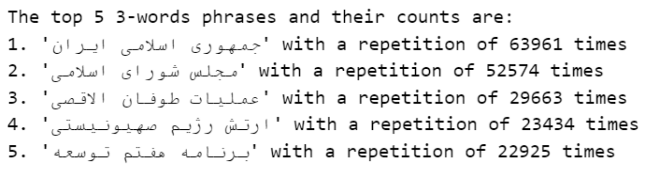
Here, I determined for instance top 5 frequent itemsets. Here's a brief explanation from what I did:

- **Applying Apriori Algorithm on RDD Partition:**
  - I defined an `apriori` function to apply the Apriori algorithm on a partition of the RDD.
  - Initialized a dictionary (`counts`) to store the counts of each n-gram and a set (`frequent`) to store frequent n-grams.(n-grams are a window of 3-words iterating through the text to take all the 3-word continuous sequences.)
  - Iterated over each text in the partition, generated all n-grams, and incremented their counts in the `counts` dictionary.
  - Filtered out n-grams that have at least `min_support`, and returned the frequent n-grams as an iterator.
- **Applying SON Algorithm on the RDD:**
  - I defined a `son` function to apply the SON algorithm on the RDD, using the `apriori` function on each partition.
  - Got the number of partitions of the RDD and apply Apriori on each partition with a lower support threshold.
  - Obtained the distinct set of candidates (`candidates`).
  - Counted the occurrences of each candidate in the entire RDD (`counts`).
  - Joined the candidates with the counts and filtered out those that do not meet the global support threshold.
  - Returned the frequent n-grams as an RDD.
- **Applying SON Algorithm on Cleaned News RDD:**

- Then, applied the SON algorithm on `clean_news_rdd` with a minimum support of 20,000 and n equal to 3.

- **Printing Top 5 3-Words Phrases and Their Counts:**
  - I finally, used `takeOrdered` to retrieve the top 5 3-word phrases based on their counts.
  - Iterated over the top 5 phrases and print each one along with its count.

The output is:



```
The top 5 3-words phrases and their counts are:
1. 'جمهوری اسلامی ایران' with a repetition of 63961 times
2. 'مجلس شورای اسلامی' with a repetition of 52574 times
3. 'عملیات طوفان الاقصی' with a repetition of 29663 times
4. 'ارتش رژیم صهیونیستی' with a repetition of 23434 times
5. 'برنامه هفتم توسعه' with a repetition of 22925 times
```

Explanations:

As you can see, The top 5 most frequent 3-word combinations in the news, had been seen in the word cloud of keywords and in the most frequent words determined earlier. This kind of clarifies the results, but I validate them using a stronger inference in the next part.

Validation Of The Above Algorithm Using TF-IDF

Explanations:

Now, in this part I'm going to use the TF-IDF approach to validate the previous part's results. In general, using the introduced formula for TF-IDF score in the Assignment, we can determine the most kind of important 3-words continuous combinations in the news and compare the results with the previous parts. If they have a reasonable conformity, we can justify them.

Let me first explain the TF-IDF flow and purposes.

TF-IDF

Explanations:

General Explanation:
The Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that reflects the importance of a term in a document relative to a collection of documents. The TF-IDF value is calculated using two main components:

1. **Term Frequency (TF):** This measures the frequency of a term in a document. It is calculated as the ratio of the number of times a term appears in a document to the total number of terms in the document.

$$\mathrm{TF}(t, d) = \frac{f_{t,d}}{\mathrm{len}(d)}$$

Here, $f_{t,d}$ is the frequency of term t in document d, and $\mathrm{len}(d)$ is the length of document d.

2. **Inverse Document Frequency (IDF):** This measures the importance of a term across the entire collection of documents. It is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term.

$$\mathrm{IDF}(t, D) = \log\left(\frac{N}{n_t}\right)$$

Here, N is the total number of documents, and n_t is the number of documents containing term t.
The **TF-IDF** score for a term t in a document d is given by the product of TF and IDF:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

In Context of Our Case:

In the context of finding the most frequent 3-words phrases in Persian texts, TF-IDF can be used to assign importance scores to these phrases. The TF component accounts for the local importance of a phrase within each document, while the IDF component considers the global importance of the phrase across all documents.

The TF-IDF scores can help identify phrases that are not only frequent but also carry significant meaning and uniqueness within the collection of texts. By comparing the top 3-words phrases obtained from the SON algorithm with their TF-IDF scores, one can validate whether the identified phrases are truly informative and distinctive in the given context.

```python
    from math import log

# Step 1: Calculate Term Frequencies (TF)
tf_rdd = clean_news_rdd.flatMap(lambda x: ngrams(x.split(), 3)) \
                        .map(lambda x: (x, 1)) \
                        .reduceByKey(lambda x, y: x + y) \
                        .map(lambda x: (x[0], x[1] / len(x[0])))

# Step 2: Calculate Document Frequencies (DF)
df_rdd = clean_news_rdd.flatMap(lambda x: set(ngrams(x.split(), 3))) \
                        .map(lambda x: (x, 1)) \
                        .reduceByKey(lambda x, y: x + y)

# Step 3: Calculate TF-IDF
N = clean_news_rdd.count()  # Total number of documents
tf_idf_rdd = tf_rdd.join(df_rdd) \
                    .map(lambda x: (x[0], x[1][0] / len(x[0]) * log(1 + N / x[1][1])))

# Print the top 5 3-words phrases and their TF-IDF scores
top_5_tfidf = tf_idf_rdd.takeOrdered(5, key=lambda x: -x[1])

# Print the top 5 most important 3-words phrases based on TF-IDF score
print("The top 5 most important 3-words phrases based on TF-IDF score are:")

# Iterate over the top 5 phrases and print each one along with its TF-IDF score
for i, (phrase, tfidf_score) in enumerate(top_5_tfidf, start=1):
    # Extract individual words from the tuple
    word1, word2, word3 = phrase

    # Print the phrase and its TF-IDF score
    print(f"{i}. '{word1} {word2} {word3}' with the TF-IDF score of {tfidf_score}")
```

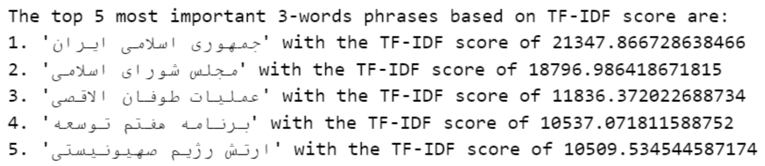
Explanations:

Here's a brief explanation from what I did:

- **Calculating Term Frequencies (TF):**
  - I used the `flatMap` transformation to generate 3-grams from each cleaned news text.
  - Assigned a count of 1 to each 3-gram and used `reduceByKey` to calculate the total count for each 3-gram.
  - Normalized the counts by dividing them by the length of the 3-gram.
- **Calculating Document Frequencies (DF):**
  - I used `flatMap` to create a set of unique 3-grams for each cleaned news text.
  - Assigned a count of 1 to each unique 3-gram and used `reduceByKey` to calculate the total count for each 3-gram.
- **Calculating TF-IDF Scores:**
  - I Joined the TF and DF RDDs based on the 3-grams.
  - Then, Calculated TF-IDF scores for each 3-gram using the formula $\text{TF-IDF} = \frac{\text{TF}}{\text{length of 3-gram}} \times \log\left(1 + \frac{N}{\text{DF}}\right)$, where N is the total number of documents.
- **Printing Top 5 Important 3-Words Phrases Based on TF-IDF:**
  - Finally, used `takeOrdered` to retrieve the top 5 3-word phrases based on TF-IDF scores.

– After all, Iterated over the top 5 phrases and printed each one along with its TF-IDF score.

And the output is:



```
The top 5 most important 3-words phrases based on TF-IDF score are:
1. 'جمهوری اسلامی ایران' with the TF-IDF score of 21347.866728638466
2. 'مجلس شورای اسلامی' with the TF-IDF score of 18796.986418671815
3. 'عملیات طوفان الاقصی' with the TF-IDF score of 11836.372022688734
4. 'برنامه هفتم توسعه' with the TF-IDF score of 10537.071811588752
5. 'ارتش رژیم صهیونیستی' with the TF-IDF score of 10509.534544587174
```

Explanations:

As you can see, The top 5 most important 3-word combinations in the news(based on TF-IDF score), as the most frequent ones, had been seen in the word cloud of keywords and in the most frequent words determined earlier.

The results are surprisingly similar. In fact, it is better to say that the first 3 terms are the same between the result of this part and the previous part, and the last 2 terms are only in a different order, which can be justified by the closeness of the number of repetitions in the news and also the closeness of their TFIDF scores. So it can be concluded that with the introduced criterion, we were able to justify and validate the results of the previous part. This is also logical. Because usually the most frequent expressions are also the most important. Of course, these two cannot be equated, but due to their close relationship and the acceptable correlation they have in the context of news, their results can be used to determine each other's accuracy. If they had similar results to a good extent, both algorithms are correct and work well, otherwise, they may need to be revised depending on the context.