

توضیحات مربوط به سوال اول :

ابتدا در سلول های اولیه اسپارک را نصب میکنیم سپس یک session مربوط به آن را مقداردهی میکنیم، سپس مستر را لوکال میگذاریم و از تمامی کورهای قابل دسترس استفاده میکنیم سپس یک نام برای اپ انتخاب میکنیم و آن را میسازیم.

```
[1] !pip install pyspark

Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
    Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425345 sha256=63f82f4bca33409f1350b5fa01de4e2eff5b13c2a12c517c0020b053363e0cb4
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf240771c078cfc8a0b0ac9241e4a4a01940da0f0b17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0

[2] from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[*]").appName("NewsDataAnalysis").getOrCreate()
```

در گام بعد فایل متنی را میخوانیم سپس به کمک تابع count تعداد خبرها ( که در اینجا خط ها هستند) را می‌شماریم، در گام بعد برای شمردن کلمات از تابع flatMap استفاده میکنیم که بر مبنای تابعی که به آن میدهیم عمل میکند که ما در اینجا به کمک lambda یک تابع خطی که هر خط را به کلمات قطعه قطعه میکند استفاده میکنیم و در گام نهایی از خروجی count میگیریم.

برای ۱۰ آیتم اول از تابع take استفاده میکنیم. سپس خروجی هر قسمت را چاپ میکنیم.

```
[3] news_rdd = spark.sparkContext.textFile("news.txt")

total_news = news_rdd.count()

words_rdd = news_rdd.flatMap(lambda line: line.split())
total_words = words_rdd.count()
first_ten_words = words_rdd.take(10)

print("Total news items:", total_news)
print("Total words:", total_words)
print("First ten words:", first_ten_words)

Total news items: 12
Total words: 2787
First ten words: ['JAPAN', 'TO', 'REVISE', 'LONG', '-', 'TERM', 'ENERGY', 'DEMAND', 'DOWNWARDS', 'The']
```

در قسمت بعد برای کوچک کردن حروف از map و از تابع lower استفاده میکنیم در گام بعد هر کلمه را به یک کلید که خودش هست مپ میکنیم، سپس به کمک reduceByKey تمامی این مقادیر با هم جمع میشوند و در واقع به صورت ایتیمال تعداد هر حرف را بدست میاوریم سپس آنها را سورت میکنیم و ۱۰ تای اول را چاپ میکنیم.

```
[4] lower_words_rdd = words_rdd.map(lambda word: word.lower())
word_counts = lower_words_rdd.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)
sorted_word_counts = word_counts.sortBy(lambda word_count: word_count[1], ascending=False)
top_ten_words = sorted_word_counts.take(10)
print("Top ten words:", top_ten_words)

Top ten words: [('the', 123), ('', 102), ('to', 84), ('of', 64), ('said', 55), ('and', 55), ('in', 54), ('a', 45), ('s', 33)]
```

برای قسمت بعد نیاز به یک فانکشن داریم که چک کند آیا کلمه فقط شامل حروف است یا خیر برای این منظور از isalpha استفاده میکنیم که همین مورد را چک میکند، سپس به کمک تابع filter فقط مواردی را نگه میداریم که حروفی هستند و در نهایت ده تا از بیشترین تکرار ها را چاپ میکنیم.

```
[5] def is_alpha(word):
    return word.isalpha()
    filtered_words = sorted_word_counts.filter(lambda word_count: is_alpha(word_count[0]))
    top_ten_filtered_words = filtered_words.take(10)
    print("Top ten filtered words:", top_ten_filtered_words)

Top ten filtered words: [('the', 123), ('to', 84), ('of', 64), ('said', 55), ('and', 55), ('in', 54), ('a', 45), ('s', 33), ('on', 28), ('for', 22)]
```

در گام نهایی برای شمردن کلماتی که شروع یکسانی دارند ابتدا هر کلمه را به حرف اول آن و ۱ مپ میکنیم سپس **reduce** را اعمال میکنیم که اگر حرف ها یکسان بود مقادیر آن جمع شوند سپس آن ها را سورت میکنیم و خروجی ۵ تای پرتکرار را میبینیم.

```
first_letter_counts = filtered_words.map(lambda word_count: (word_count[0][0], 1)).reduceByKey(lambda a, b: a + b)
top_five_letters = first_letter_counts.sortBy(lambda letter_count: letter_count[1], ascending=False).take(5)
print("Top five letters:", top_five_letters)

Top five letters: [('c', 76), ('s', 74), ('p', 68), ('a', 57), ('r', 54)]
```

توضیحات مربوط به سوال ۲:

تا گام لود دیتا که مانند قبل عمل میکنیم، سپس در گام اول به هر داکيومنت که در اینجا لاین هست یک ایندکس میدهیم که در ادامه برای شناسایی استفاده میشود.

سپس **tf** را برای هر کلمه در هر داکيومنت حساب میکنیم (هر دوتایی **word-document**) شمرده میشود.

سپس در گام محاسبه **DF** با ایگنور کردن ایندکس هر داکيومنت، تعداد داکيومنت هایی که یک کلمه خاص را دارند میشماریم.

سپس برای محاسبه **IDF** از فرمول ریاضی آن استفاده میکنیم که **N** در آن تعداد کل داکيومنت هاست و **df** را هم که داریم.

```
[1] !pip install pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").appName("TF-IDF").getOrCreate()

Collecting pyspark
Downloading pyspark-3.5.0.tar.gz (316.9 MB)
Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
Building wheel for pyspark (setup.py) ... done
Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425345 sha256=7a4f1bcf47486a301ab245d02f60c3e656d771e433adcec77b08ef2d0b0967b
Stored in directory: /root/.cache/pip/wheels/41/4e/10/c1cf2607f71c078cfc8a8b0ac9241e5e4a01940da0fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0

[21] sc = spark.sparkContext
rdd = sc.textFile("news.txt")

docs_with_index = rdd.zipWithIndex()
tf_with_doc_index = docs_with_index.flatMap(lambda doc: [(word, doc[1], 1) for word in tokenize(doc[0])])\
    .reduceByKey(lambda a, b: a + b)


df = docs_with_index.flatMap(lambda doc: set(tokenize(doc[0])))\
    .map(lambda words: (word, 1))\
    .reduceByKey(lambda a, b: a + b)

total_documents = docs_with_index.count()
idf = df.map(lambda x: (x[0], math.log10(total_documents / x[1])))

tf_idf = tf_with_doc_index.map(lambda x: (x[0][0], (x[0][1], x[1])))\
    .join(idf)\
    .map(lambda x: (x[0], (x[1][0][0], x[1][0][1] * x[1][1])))\
    .groupByKey()\
    .mapValues(list)
```

حال برای محاسبه **tf-idf** ابتدا هر کدام را **join** میکنیم سپس ضرب که در نهایت نتایج شامل لیستی از ایندکس داکيومنت و **tf-idf** برای هر کلمه هستند میشود.

حال برای پیدا کردن داکيومنت های مشابه به کمک filter سه تای اول که بیشترین امتیاز را بدست آوردند  
برمیگردانیم.



```
[x] sample_data news.txt
```

```
[23] def find_related_documents(word, top_n=3):  
    return (tf_idf.filter(lambda x: x[0] == word)  
            .flatMap(lambda x: [(doc_id, tfidf) for doc_id, tfidf in x[1]])  
            .takeOrdered(top_n, key=lambda x: -x[1]))  
  
    related_documents_japan = find_related_documents("japan")  
    related_documents_gas = find_related_documents("gas")  
    related_documents_market = find_related_documents("market")  
  
    print("Related documents for 'japan':", related_documents_japan)  
    print("Related documents for 'gas':", related_documents_gas)  
    print("Related documents for 'market':", related_documents_market)
```

```
Related documents for 'japan': [(6, 4.294801202476962), (5, 1.4313637641589874), (0, 0.9542425094393249)]  
Related documents for 'gas': [(0, 1.5563825007672873), (4, 0.7781512503836436)]  
Related documents for 'market': [(6, 1.4313637641589874), (7, 0.9542425094393249), (11, 0.9542425094393249)]
```