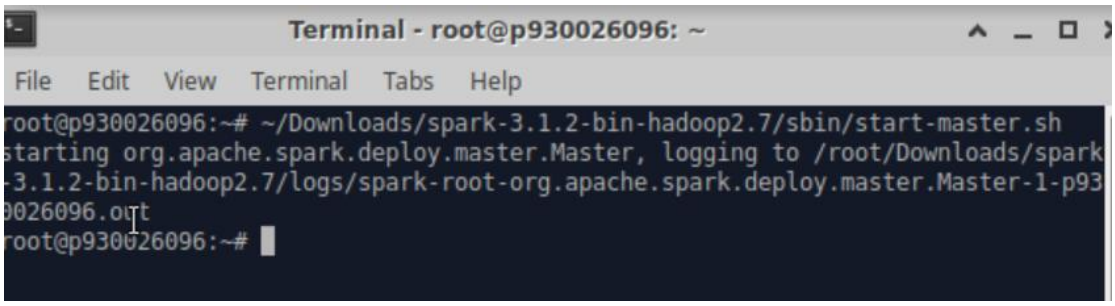
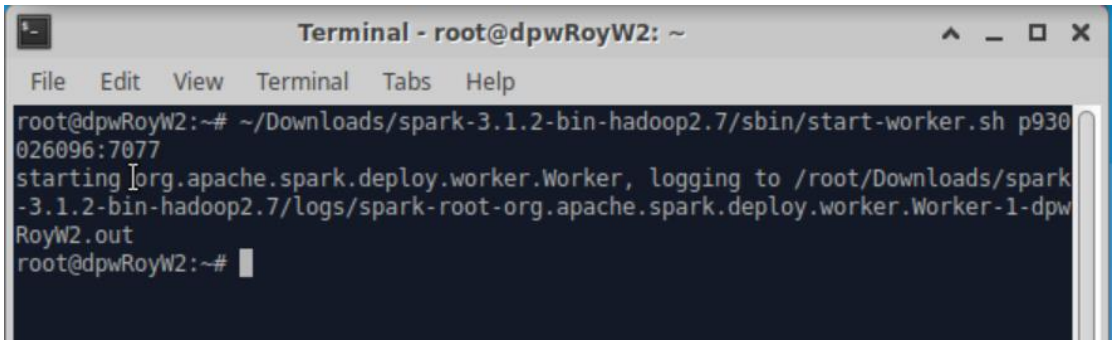
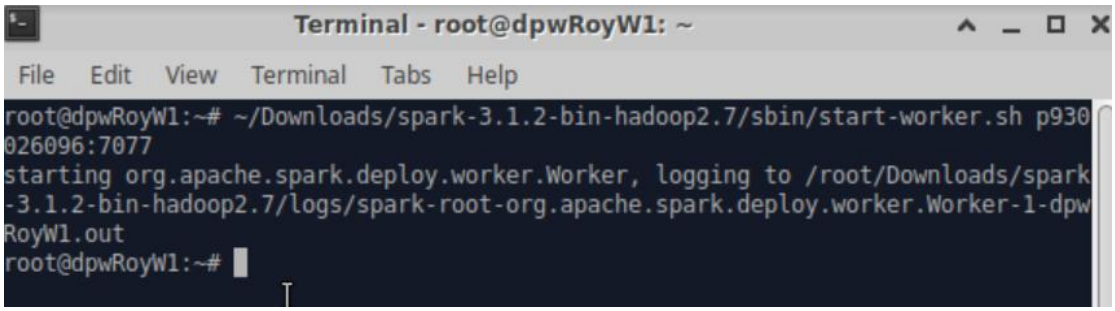
Basic



Start the master



Start the first worker



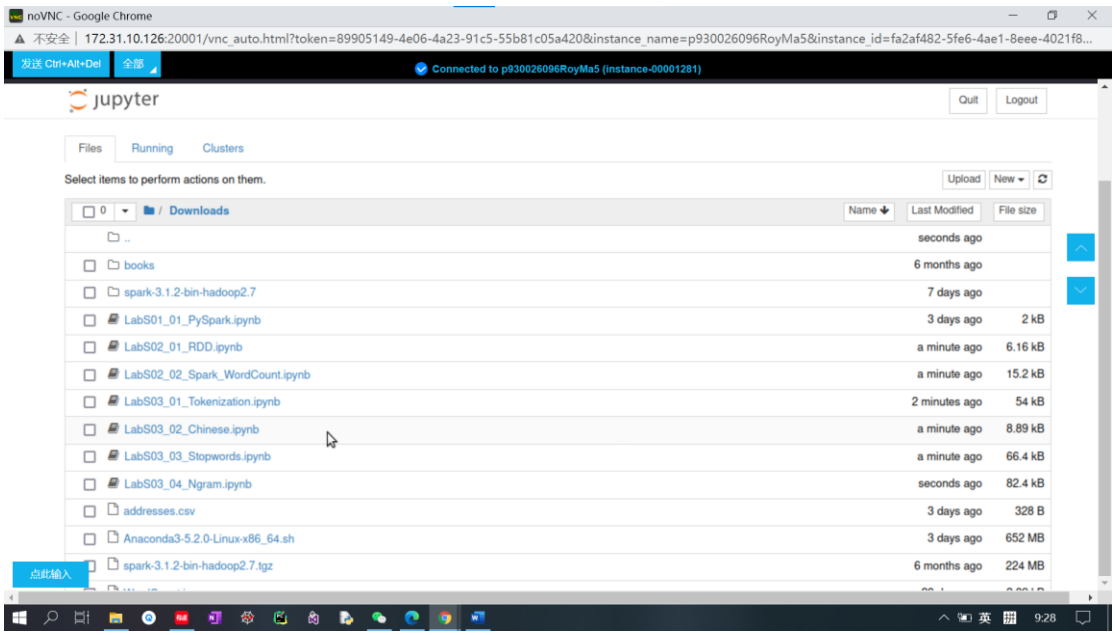
Start the second worker

```
root@p930026096:~# conda install pyspark
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.5.4
  latest version: 4.10.3

Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: /root/anaconda3
```

Install pyspark in anaconda

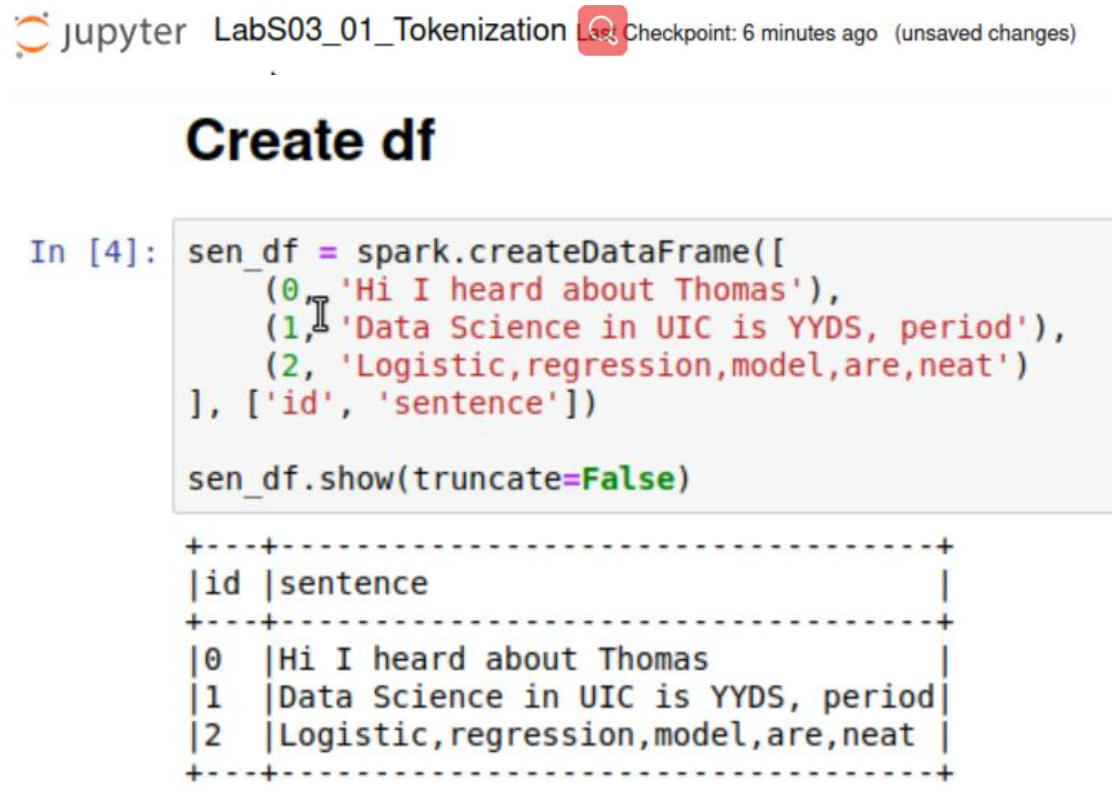
```
root@p930026096:~# pip3 install jieba
Collecting jieba
  Downloading jieba-0.42.1.tar.gz (19.2 MB)
     |[redacted]| 19.2 MB 3.9 MB/s
Building wheels for collected packages: jieba
  Building wheel for jieba (setup.py) ... done
  Created wheel for jieba: filename=jieba-0.42.1-py3-none-any.whl size=19314476
sha256=22526a9331fbab58ae2aa2b9e10a0ce7518b07d13c3879e65b6475a119a4b942
  Stored in directory: /root/.cache/pip/wheels/17/a7/8b/a7e03881534e78558920ac68
aaeca05180c0e2c3d11c4fce3b
Successfully built jieba
Installing collected packages: jieba
Successfully installed jieba-0.42.1
```

Install jieba in anaconda



Open the localhost jupyter notebook

Run the LabS03_1_Tokenization file



Create spark dataframe

Tokenize

```
In [5]: tokenizer = Tokenizer(inputCol = 'sentence', outputCol = 'words')
        regex_tokenizer = RegexTokenizer(inputCol = 'sentence', outputCol = 'words', pattern = '\\W')

        count_tokens = udf(lambda words: len(words), IntegerType())
        tokenized = tokenizer.transform(sen_df)
        tokenized.show(truncate=False)
```

```
+---+-----------------------------------+-----------------------------------------+
|id |sentence                           |words                                    |
+---+-----------------------------------+-----------------------------------------+
|0  |Hi I heard about Thomas            |[hi, i, heard, about, thomas]            |
|1  |Data Science in UIC is YYDS, period|[data, science, in, uic, is, yyds,, period]|
|2  |Logistic,regression,model,are,neat |[logistic,regression,model,are,neat]     |
+---+-----------------------------------+-----------------------------------------+
```

Tokenize the words in sentences

Process the tokens with regex

```
In [7]: regex_tokenized = regex_tokenizer.transform(sen_df)
        regex_tokenized.show(truncate=False)
        regex_tokenized.withColumn('tokens', count_tokens(col('words'))).show(truncate=False)
```

```
+---+-----------------------------------+-----------------------------------------+
|id |sentence                           |words                                    |
+---+-----------------------------------+-----------------------------------------+
|0  |Hi I heard about Thomas            |[hi, i, heard, about, thomas]            |
|1  |Data Science in UIC is YYDS, period|[data, science, in, uic, is, yyds, period]|
|2  |Logistic,regression,model,are,neat |[logistic, regression, model, are, neat]  |
+---+-----------------------------------+-----------------------------------------+

+---+-----------------------------------+-----------------------------------------+------+
|id |sentence                           |words                                    |tokens|
+---+-----------------------------------+-----------------------------------------+------+
|0  |Hi I heard about Thomas            |[hi, i, heard, about, thomas]            |5     |
|1  |Data Science in UIC is YYDS, period|[data, science, in, uic, is, yyds, period]|7     |
|2  |Logistic,regression,model,are,neat |[logistic, regression, model, are, neat]  |5     |
+---+-----------------------------------+-----------------------------------------+------+
```

Process the tokens with regex

Process text file input

```
In [9]: df_text = spark.read.text("1.txt")
        df_text.printSchema()
        df_text.show(truncate=False)
```

```
|0001 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry'
s standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make
a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remai
ning essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ip
sum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem I
psum.
|
|0002 Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin
literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney Colle
ge in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going t
hrough the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sec
tions 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in
45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lore
m Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32
|
|0003 The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.1
0.32 and 1.10.33 from "de Finibus Bonorum et Malorum" by Cicero are also reproduced in their exact original form, a
ccompanied by English versions from the 1914 translation by H. Rackham.
```

Read a text file

```
In [10]: tokenizer = Tokenizer(inputCol = 'value', outputCol = 'words')
         count_tokens = udf(lambda words: len(words), IntegerType())
         tokenized = tokenizer.transform(df_text)
         tokenized.withColumn('tokens', count_tokens(col('words'))).show(truncate=False)
```

```
---------------------------------------------------------------------------------------------------------------
---+------+
|0001 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry'
s standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make
a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remai
ning essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ip
sum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem I
psum.
|[0001, lorem, ipsum, is, simply, dummy, text, of, the, printing, and, typesetting, industry., lorem, ipsum, has, b
een, the, industry's, standard, dummy, text, ever, since, the, 1500s,, when, an, unknown, printer, took, a, galley,
of, type, and, scrambled, it, to, make, a, type, specimen, book., it, has, survived, not, only, five, centuries,, b
ut, also, the, leap, into, electronic, typesetting,, remaining, essentially, unchanged., it, was, popularised, in,
the, 1960s, with, the, release, of, letraset, sheets, containing, lorem, ipsum, passages,, and, more, recently, wit
h, desktop, publishing, software, like, aldus, pagemaker, including, versions, of, lorem, ipsum.]
|92    |
|0002 Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin
literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney Colle
ge in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going t
hrough the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sec
```

Tokenization for the text file

Run the Lab03_02_Chinese file

jupyter LabS03_02_Chinese

Jieba for Chinese text analysis

```
In [1]: import jieba

        string='广东省珠海市唐家湾金同路2000号北京师范大学香港浸会大学联合国际学院'
        string='南京市长江大桥'
        string='无线电法国别研究'

        string='EDG夺冠，爷青回'

        # string='宝，我今天输液了，输什么液 ？ 想你的夜'
        # string='宝，我今天去种地了，种的什么地？对你的死心塌地'
        # string='宝，我今天去吃面了，吃的什么面？突然想见你一面'

        # string='来到杨过曾经生活的地方，小龙女动情地说：我也想过过过儿过过的生活'
        # string='货拉拉拉不拉拉布拉多，取决于货拉拉拉拉布拉多时拉布拉多拉不拉粑粑'
        # string='校长说：校服上除了校徽别别别的，让你们别别别的别别别的你非别别的'
        # string='明明明明明白白白喜欢他，可她就是不说'
        # string='人要是行，干一行行一行，一行行行行行，行行行干哪行都行。要是不行，干一行不行一行，一行不行行行不行，行行不行干哪行都不行'

        #Precise mode
        text_cut=jieba.cut(string)
        print(" ".join(text_cut))

        #Full mode
        text_cut=jieba.cut(string,cut_all=True)
```

Define some Chinese strings as input

```python
# string='宝，我今天输液了，输什么液 ？ 想你的夜'
# string='宝，我今天去种地了，种的什么地？对你的死心塌地'
# string='宝，我今天去吃面了，吃的什么面？突然想见你一面'

# string='来到杨过曾经生活的地方，小龙女动情地说：我也想过过过儿过过的生活'
# string='货拉拉拉不拉拉布拉多，取决于货拉拉拉拉布拉多时拉布拉多拉不拉粑粑'
# string='校长说：校服上除了校徽别别别的，让你们别别别的别别别的你非别别的'
# string='明明明明明白白喜欢他，可她就是不说'
# string='人要是行，干一行行一行，一行行行行行，行行行干哪行都行。要是不行，干一行不行一行，一行不行行行不行，行行不行干哪行都不行'

#Precise mode
text_cut=jieba.cut(string)
print(" ".join(text_cut))

#Full mode
text_cut=jieba.cut(string,cut_all=True)
print(" ".join(text_cut))

#Search mode
text_cut=jieba.cut_for_search(string)
print(" ".join(text_cut))
```

```
Building prefix dict from the default dictionary ...
Dumping model to file cache /tmp/jieba.cache
Loading model cost 0.828 seconds.
Prefix dict has been built successfully.
```

```
EDG 夺冠 ， 爷青回
EDG 夺冠 ， 爷 青 回
EDG 夺冠 ， 爷青回
```

Tokenization of Chinese

You can also provide a text file for Jieba Tokenization

```
In [2]: textFile = 'santi.txt'
        tokenFile = 'santi_token.txt'

        with open(textFile, 'r', encoding = 'utf-8') as sourceFile, open(tokenFile, 'a+', encoding = 'utf-8') as targetFile:
            for line in sourceFile:
                seg = jieba.cut(line.strip(), cut_all = False)
                output = ' '.join(seg)
                targetFile.write(output)
                targetFile.write('\n')
```

Open + santi_token.txt ~/Downloads Save

```
书名 ： 三体 全集 （ 精校 注释 版 ）

祖籍 河南 ， 长于 山西 ， 中国 科普 作家 协会会员 ， 山西省 作家 协会会员 ， 高级 工程师 。
自 1999 年 处女作 《 鲸歌 》 问世 以来 ， 发表 短篇 科幻小说 三十余篇 ， 出版 长篇 科幻小说 七部 ， 并 创下 连续 八年 荣获 中国 科幻 最高奖 “ 银河 奖 ” 的 纪录 。 其 气势恢宏 的 “ 地球 往事 ” 系列 第一部 《 三体 》 开创 了 《 科幻世界 》 月刊 连载 原创 作品 之 先河 ， 一举 成为 2006 年度 最受 关注 、 最 畅销 的 科幻小说 。 2008 年 ， “ 地球 往事 ” 系列 第二部 《 黑暗 森林 》 面世 ， 再次 成为 当年 最 畅销 的 科幻小说 。 2010 年 ， “ 地球 往事 ” 系列 第三部 《 死神 永生 》 出版 ， 一举 拿下 该 年度 中国 科幻 “ 银河 奖 ” 特别奖 、 全球华语 科幻 “ 星云 奖 ” 最佳 长篇 奖 。 刘慈欣 亦 因此 被 评论界 誉为 “ 21 世纪 中国 文坛 最 值得 关注 的 作家 ” 。
刘慈欣 的 作品 恢弘 大气 、 想象 绚丽 ， 既 注意 极端 空灵 与 厚重 现实 的 结合 ， 也 讲求 科学 的 内涵 与 人文 的 美感 ， 具有 浓郁 的 中国 特色 和 鲜明 的 个人风格 ， 为 中国 科幻 确立 了 一个 新 高度 。

写 在 “ 基石 ” 之前
■ 姚 海军



“ 基石 ” 是 个 平实 的 词 ， 不够 “ 炫 ” ， 却 能够 准确 传达 我们 对 构建 中 的 中国 科幻 繁华 巨厦 的 情感 与 信心 ， 因此 ， 我们 用 它 来 作为 这套 原创 丛书 的 名字 。
最近 十年 ， 是 科幻 创作 飞速发展 的 十年 。 王晋康 、 刘慈欣 、 何 宏伟 、 韩松 等 一大批 科幻 作家 发表 了 大量 深受 读者 喜爱 、 极具 开拓 与 探索 价值 的 科幻 佳作 。 科幻 文学 的 龙头 期刊 更是 从 一本 传统 的 《 科幻世界 》 ， 发展壮大 成为 涵盖 各个 读者 层 的 系列 刊物 。 与此同时 ， 科幻 文学 的 市场 环境 也 有 了 改善 ， 省会 级 城市 的 大型 书店 里 终于 有 了 属于 科幻 的 领地 。
仍然 有人 经常 问及 中国 科幻 与 美国 科幻 的 差距 ， 但 现在 的 答案 已 与 十年 前 不同 。 在 很多 作品 上 （ 它们 不再 是 那种 毫无 文学 技巧 与 色彩 、 想象力 拘谨 的 幼稚 故事 ） ， 这种 比较 已经 变成 了 人家 的 牛排 之 于 我们 的 土豆 牛肉 。 差距 是 明显 的 — — 更 准确 地说 ， 应该 是 “ 差别 ” — — 却 已经 无法 再 为 它们 排个 名次 。 口味 问题 有 了 实际意义 ， 这 正是 我们 的 科幻 走向 成熟 的 标志 。
```

Tokenization for Chinese text file

Remove the stopwords for Chinese

Go through the stopwords dictionary word by word, and remove them

In [3]:
```
string='总的来说，人说话的时候废话非常多。与此同时，废话多了有害身体健康，换句话说，不如别说废话！'

text_cut = jieba.cut(string)
print(" ".join(text_cut))

def stopwordslist(filepath):
    stopwords = [line.strip() for line in open(filepath, 'r', encoding='utf-8').readlines()]
    return stopwords

def seg_sentence(sentence):
    sentence_seged = jieba.cut(sentence.strip())
    stopwords = stopwordslist('baidu_stopwords.txt')
    outstr = ''
    for word in sentence_seged:
        if word not in stopwords:
            if word != '\t':
                outstr += word
                outstr += " "
    return outstr

line_seg = seg_sentence(string)
```

```
print(line_seg)
```

```
总的来说 ， 人 说话 的 时候 废话 非常 多 。 与此同时 ， 废话 多 了 有害 身体健康 ， 换句话说 ， 不如 别说 废话 ！
， 说话 废话 。 ， 废话 有害 身体健康 ， ， 废话 ！
```

Remove stop words in Chinese text according to baidu_stopwords.txt

You can also use the built-in function to remove the stopwords

In [4]:
```
text_cut=jieba.cut(string)
print(" ".join(text_cut))

import jieba.analyse

jieba.analyse.set_stop_words('baidu_stopwords.txt')
text_cut = jieba.analyse.extract_tags(string, 20)
print(" ".join(text_cut))
```

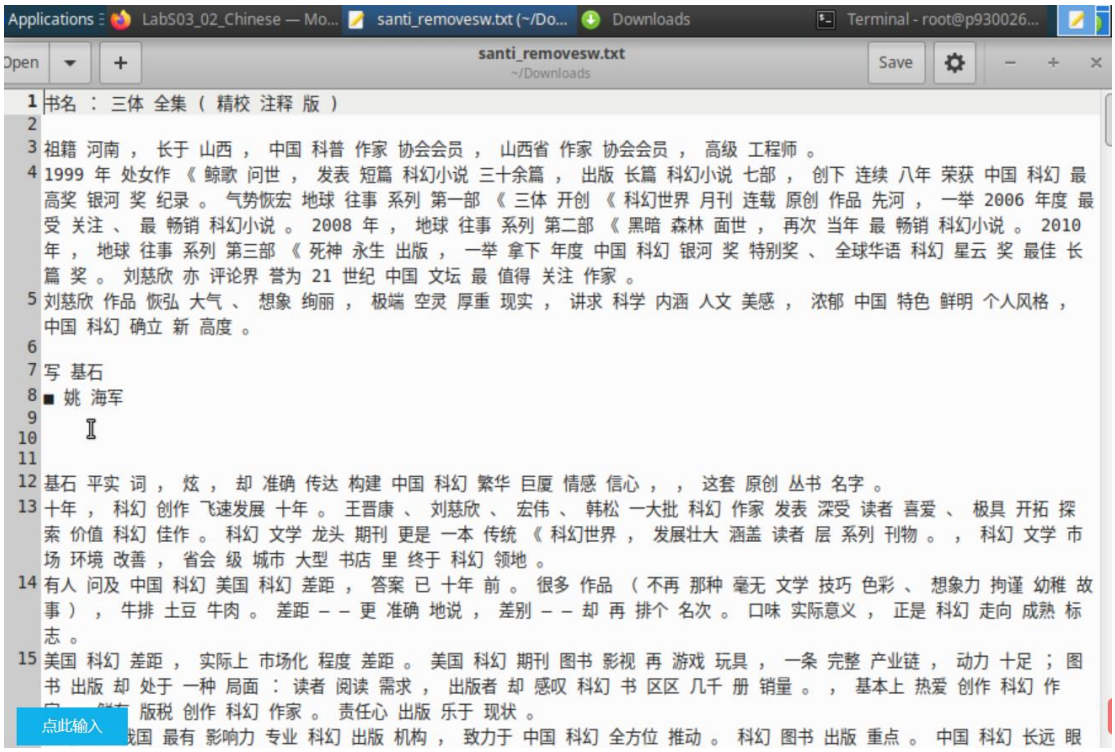
```
总的来说 ， 人 说话 的 时候 废话 非常 多 。 与此同时 ， 废话 多 了 有害 身体健康 ， 换句话说 ， 不如 别说 废话 ！
废话 身体健康 有害 说话
```

Use a built-in function to remove the stop words rather than using baidu_stopwords.txt

You can also provide a text file for stopwords removal, go through the stopwords dictionary word by word, and remove them

In [5]:
```
processedFile = 'santi_removesw.txt'

inputs = open(textFile, 'r', encoding='utf-8')
outputs = open(processedFile, 'w')
for line in inputs:
    line_seg = seg_sentence(line)
    outputs.write(line_seg + '\n')
```

Provide a self-built stop words text file, and remove stop words in the file



The result of a Chinese text file, removing stop words

Provide a text file for stopwords removal, and use the built-in function to remove the stopwords

In [6]:
```
with open(textFile, 'r', encoding = 'utf-8') as file:
    text = file.readlines()
    keywords = jieba.analyse.extract_tags(str(text), topK = 200)
    print(keywords)
```

```
['程心', '罗辑', '三体', '太空', '汪淼', '宇宙', '地球', '智子', '飞船', '叶文洁', '世界', '人类', '太阳', '舰队', '面壁', 'AA
', '文明', '太阳系', '纪元', '光速', 'u3000', '大史', '关一帆', '水滴', '史强', '二维', '北海', '迪亚兹', '感觉', '行星', '恒星
', '时间', '信息', '冬眠', '东西', '天明', '两个', '很快', '维德', '看着', '计划', '威慑', '真的', '黑暗', '发现', '这是', '消
失', '空间', '丁仪', '星环', '一种', '目光', '太空城', '红岸', '恩斯', '战舰', '发出', '地说', '技术', '发射', '只能', '木星',
'世纪', '蓝色', '方向', '穿梭机', '引力波', '研究', '孩子', '公主', '万有引力', '仿佛', '位置', '破壁', '系统', '轨道', '声音',
'城市', '星星', '时代', '状态', '三个', '想象', '生活', '舰长', '泰勒', '肯定', '地面', '速度', '庄颜', '距离', '看上去', '思想
', '森林', '处于', '点点头', '目标', '工作', '生命', '立刻', '感到', '观测', '掩体', '加速', '外面', '毁灭', '眼睛', '显示', '
三维', '一颗', '钢印', '渐渐', '来自', '发生', '回答', '联合国', '苏醒', '危机', '博士', '伊文斯', '像是', '听到', '惠子', '基地
', '天空', '表面', '好像', '窗口', '航行', '沉默', '再次', '运行', '社会', '第一次', '摇摇头', '图像', '地方', '小时', '探测器',
'曹彬', '到达', '移民', '有人', '平面', '天线', '永远', '也许', '生存', '外星', '希望', '自然选择', '延绪', '画师', '星空', '离
开', '科学', '光年', '曲率', '一部分', '减速', '阳光', '变得', '星系', '一条', '一只', '常伟思', '一名', '画面', '默斯', '黑洞',
'大部分', '辐射', '坎特', '杨卫宁', '东方', '形状', '告诉', '计算机', '这次', '内部', '防御', '不到', '字幕', '确实', '军官', '
唯一', '广播', '攻击', '故事', '光芒', '启动', '一点', '物质', '一艘', '核弹', '大脑', '刚才', '银河系', '飘浮', '现实']
```

Remove stop words according to your own stop words file.

Run LabS03_03_Stopwords file

jupyter LabS03_03_Stopwords

Create PySpark environment and import library

In [1]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('MyStopwords').getOrCreate()

from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType
from pyspark.ml.feature import StopWordsRemover
```

Create pyspark environment and import library

Stopwords removal

In [2]:
```
sentenceDataFrame = spark.createDataFrame([
    (0, ['I', 'saw', 'the', 'green', 'horse']),
    (1, ['Mary', 'had', 'a', 'little', 'lamb'])
], ['id', 'tokens'])

sentenceDataFrame.show(truncate=False)

remover = StopWordsRemover(inputCol = 'tokens', outputCol = 'filtered')
remover.transform(sentenceDataFrame).show(truncate=False)
```

```
+---+----------------------------+
|id |tokens                      |
+---+----------------------------+
|0  |[I, saw, the, green, horse] |
|1  |[Mary, had, a, little, lamb]|
+---+----------------------------+

+---+----------------------------+--------------------+
|id |tokens                      |filtered            |
+---+----------------------------+--------------------+
|0  |[I, saw, the, green, horse] |[saw, green, horse] |
|1  |[Mary, had, a, little, lamb]|[Mary, little, lamb]|
+---+----------------------------+--------------------+
```

Remove stop words in English sentences by built-in function

Process text file input

First tokenize, then remove stop words

In [3]:
```
df_text = spark.read.text("1.txt")

df_text.show(truncate=False)

tokenizer = Tokenizer(inputCol = 'value', outputCol = 'words')
tokenized = tokenizer.transform(df_text)

remover = StopWordsRemover(inputCol = 'words', outputCol = 'filtered')
remover.transform(tokenized).show(truncate=False)
```

```
|0001 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry'
s standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make
a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remai
ning essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ip
sum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem I
psum.
|
|0002 Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin
literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney Colle
ge in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going t
hrough the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sec
tions 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in
```

Process text file. First tokenization and then remove stop words

Run LabS03_04_Ngram

jupyter LabS03_04_Ngram

Create PySpark environment and import library

In [1]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('MyNgram').getOrCreate()

from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType
from pyspark.ml.feature import NGram
```

Create pyspark environment and import library

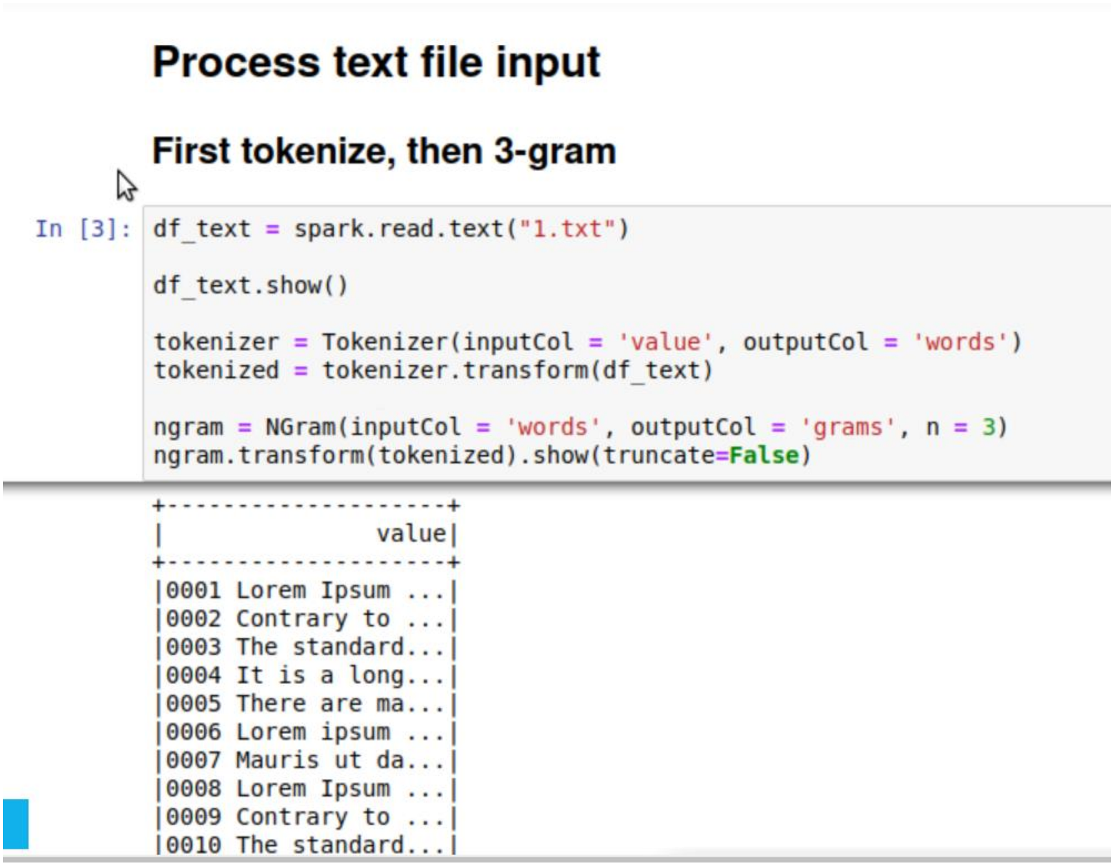
3-gram from words df

In [2]:
```
wordDataFrame = spark.createDataFrame([
    (0, ['Hi', 'I', 'heard', 'about', 'Spark']),
    (1, ['I', 'wish', 'python', 'could', 'use', 'case', 'classes']),
    (2, ['Logistic', 'regression', 'models', 'are', 'neat'])
], ['id', 'words'])

wordDataFrame.show(truncate=False)

ngram = NGram(inputCol = 'words', outputCol = 'grams', n = 3)
ngram.transform(wordDataFrame).show(truncate=False)

ngram.transform(wordDataFrame).select('grams').show(truncate=False)
```

```
+---+-----------------------------------------+
|id |words                                    |
+---+-----------------------------------------+
|0  |[Hi, I, heard, about, Spark]             |
|1  |[I, wish, python, could, use, case, classes]|
|2  |[Logistic, regression, models, are, neat]|
+---+-----------------------------------------+
```

```
--------------------+
|id |words                                    |grams
|
+---+-----------------------------------------+-----------------------------------------------------------------
--------------------+
|0  |[Hi, I, heard, about, Spark]             |[Hi I heard, I heard about, heard about Spark]
|
|1  |[I, wish, python, could, use, case, classes]|[I wish python, wish python could, python could use, could use cas
e, use case classes]|
|2  |[Logistic, regression, models, are, neat]|[Logistic regression models, regression models are, models are nea
t]                |
+---+-----------------------------------------+-----------------------------------------------------------------
--------------------+

+-----------------------------------------------------------------------------------+
|grams                                                                              |
+-----------------------------------------------------------------------------------+
|[Hi I heard, I heard about, heard about Spark]                                     |
|[I wish python, wish python could, python could use, could use case, use case classes]|
|[Logistic regression models, regression models are, models are neat]               |
+-----------------------------------------------------------------------------------+
```

Get 3-gram from words df

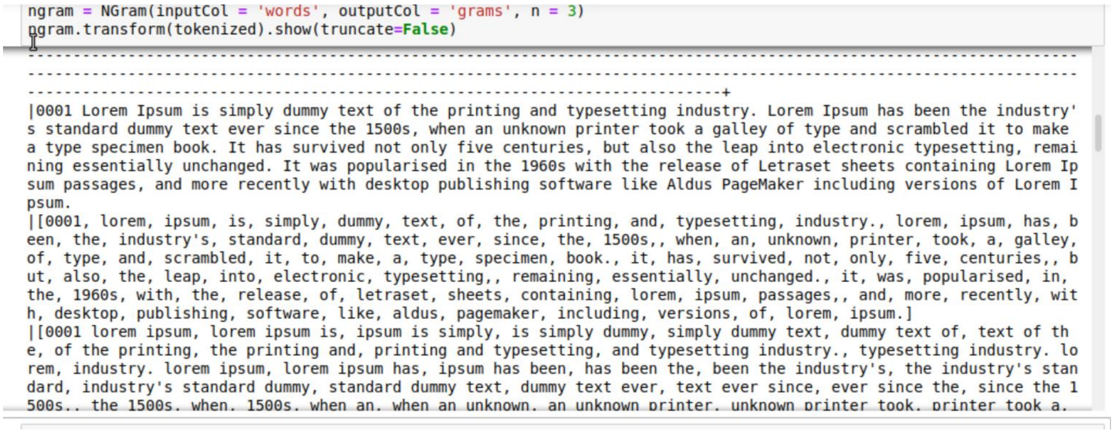


Process text file input

First tokenize, then 3-gram

```
In [3]: df_text = spark.read.text("1.txt")

        df_text.show()

        tokenizer = Tokenizer(inputCol = 'value', outputCol = 'words')
        tokenized = tokenizer.transform(df_text)

        ngram = NGram(inputCol = 'words', outputCol = 'grams', n = 3)
        ngram.transform(tokenized).show(truncate=False)
```

```
+--------------------+
|               value|
+--------------------+
|0001 Lorem Ipsum ...|
|0002 Contrary to ...|
|0003 The standard...|
|0004 It is a long...|
|0005 There are ma...|
|0006 Lorem ipsum ...|
|0007 Mauris ut da...|
|0008 Lorem Ipsum ...|
|0009 Contrary to ...|
|0010 The standard...|
```



```
ngram = NGram(inputCol = 'words', outputCol = 'grams', n = 3)
ngram.transform(tokenized).show(truncate=False)
```

```
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------+
|0001 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry'
s standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make
a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remai
ning essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ip
sum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem I
psum.
|[0001, lorem, ipsum, is, simply, dummy, text, of, the, printing, and, typesetting, industry., lorem, ipsum, has, b
een, the, industry's, standard, dummy, text, ever, since, the, 1500s,, when, an, unknown, printer, took, a, galley,
of, type, and, scrambled, it, to, make, a, type, specimen, book., it, has, survived, not, only, five, centuries,, b
ut, also, the, leap, into, electronic, typesetting,, remaining, essentially, unchanged., it, was, popularised, in,
the, 1960s, with, the, release, of, letraset, sheets, containing, lorem, ipsum, passages,, and, more, recently, wit
h, desktop, publishing, software, like, aldus, pagemaker, including, versions, of, lorem, ipsum.]
|[0001 lorem ipsum, lorem ipsum is, ipsum is simply, is simply dummy, simply dummy text, dummy text of, text of th
e, of the printing, the printing and, printing and typesetting, and typesetting industry., typesetting industry. lo
rem, industry. lorem ipsum, lorem ipsum has, ipsum has been, has been the, been the industry's, the industry's stan
dard, industry's standard dummy, standard dummy text, dummy text ever, text ever since, ever since the, since the 1
500s,, the 1500s, when, 1500s, when an, when an unknown, an unknown printer, unknown printer took, printer took a,
```

Process text file. First tokenization and then get 3-gram

Run LabS02_01_RDD file

jupyter LabS02_01_RDD

Simple RDD parallelize, sum of 1+2+3+4...+100

```
In [1]: from pyspark.sql import SparkSession

        spark = SparkSession.builder.master('local').getOrCreate()
        sc = spark.sparkContext

        rdd = sc.parallelize(range(100 + 1))
        rdd.sum()

Out[1]: 5050
```

Try RDD parallelize. Get the sum from 1 to 100

RDD can also parallelize array of words, and count words

```
In [2]: words = sc.parallelize (
           ["scala",
           "java",
           "hadoop",
           "spark",
           "akka",
           "spark vs hadoop",
           "pyspark",
           "pyspark and spark"]
        )
        counts = words.count()
        print("Number of elements in RDD -> %i" % (counts))

        Number of elements in RDD -> 8
```

RDD can also parallelize array of words, and count words

You can use collect operation to get elements from the RDD

```
In [3]: coll = words.collect()
        print("Elements in RDD -> %s" % (coll))

        Elements in RDD -> ['scala', 'java', 'hadoop', 'spark', 'akka', 'spark vs hadoop', 'pyspark', 'pyspark and spark']
```

Get elements from RDD by .collect() function

The "Map" operation of RDD

```
In [4]: words_map = words.map(lambda x: (x, 1))
        mapping = words_map.collect()
        print("Key value pair -> %s" % (mapping))

        Key value pair -> [('scala', 1), ('java', 1), ('hadoop', 1), ('spark', 1), ('akka', 1), ('spark vs hadoop', 1), ('py
        spark', 1), ('pyspark and spark', 1)]
```

Map elements from RDD by .map() function

The "Reduce" operation of RDD

```
In [5]: from operator import add

        nums = sc.parallelize([1, 2, 3, 4, 5])
        adding = nums.reduce(add)
        print("Adding all the elements -> %i" % (adding))

        Adding all the elements -> 15
```

Aggregate the elements of the dataset by .reduce() function

The "Join" operation of RDD, joining elements containing matching keys

```
In [6]: x = sc.parallelize([("spark", 1), ("hadoop", 4)])
        y = sc.parallelize([("spark", 2), ("hadoop", 5)])
        joined = x.join(y)
        final = joined.collect()
        print("Join RDD -> %s" % (final))

        Join RDD -> [('hadoop', (4, 5)), ('spark', (1, 2))]
```

Join elements containing matching keys by .join function

Create DataFrame for Spark

```
In [7]: data = [('James','','Smith','1991-04-01','M',3000),
          ('Michael','Rose','','2000-05-19','M',4000),
          ('Robert','','Williams','1978-09-05','M',4000),
          ('Maria','Anne','Jones','1967-12-01','F',4000),
          ('Jen','Mary','Brown','1980-02-17','F',-1)
        ]

        columns = ["firstname","middlename","lastname","dob","gender","salary"]
        df = spark.createDataFrame(data = data, schema = columns)
        df.show()

        +---------+----------+--------+----------+------+------+
        |firstname|middlename|lastname|       dob|gender|salary|
        +---------+----------+--------+----------+------+------+
        |    James|          |   Smith|1991-04-01|     M|  3000|
        |  Michael|      Rose|        |2000-05-19|     M|  4000|
        |   Robert|          |Williams|1978-09-05|     M|  4000|
        |    Maria|      Anne|   Jones|1967-12-01|     F|  4000|
        |      Jen|      Mary|   Brown|1980-02-17|     F|    -1|
        +---------+----------+--------+----------+------+------+
```

Create dataframe for spark

You can also submit the Spark job to the Cluster Master

```
In [9]: from pyspark.sql import SparkSession

        spark = SparkSession.builder.master('spark://p930026096:7077').appName('MyAccumulate').getOrCreate()
        sc = spark.sparkContext

        rdd = sc.parallelize(range(10000000000))
        rdd.sum()

Out[9]: 49999999995000000000
```

Use master and two workers to calculate a very large number

Run LabS02_02_Spark_WordCount file

jupyter LabS02_02_Spark_WordCount

Initialize PySpark

```
In [1]: from pyspark import SparkConf, SparkContext

        sc = SparkContext.getOrCreate(SparkConf())
```

Initialize pyspark

Process CSV files as RDD

```
In [2]: lines = sc.textFile('Baby_Names__Beginning_2007.csv')

        print(lines.first())
        print('---')
        print(lines.take(5))

        #length of first 5 elements
        lines.map(lambda s: len(s)).take(5)

        Year,First Name,County,Sex,Count
        ---
        ['Year,First Name,County,Sex,Count', '2013,GAVIN,ST LAWRENCE,M,9', '2013,LEVI,ST LAWRENCE,M,9', '2013,LOGAN,NEW YOR
        K,M,44', '2013,HUDSON,NEW YORK,M,49']

Out[2]: [32, 26, 25, 24, 25]
```

Run .csv file as RDD

```
In [3]: #returns total number of characters
        rdd = lines.map(lambda s: len(s))
        rdd = rdd.map(lambda s: 2*s)
        print(rdd.reduce(lambda a,b: a+b))

        2424036
```

Apply map and reduce functions.

MapReduce on RDD

```
In [4]: rdd = sc.parallelize(['hello','world','hello','thomas'])

        rdd = rdd.map(lambda w: (w,1))
        rdd.collect()

        rdd.reduceByKey(lambda x, y: x + y).collect()

Out[4]: [('hello', 2), ('thomas', 1), ('world', 1)]
```

MapReduce on RDD

Text file can also be processed as RDD in Spark

```
In [5]: #The first line defines a base RDD from an external file.
        #This dataset is not loaded in memory or otherwise acted on: lines is merely a pointer to the file.
        lines = sc.textFile("1.txt")

        #The second line defines lineLengths as the result of a map transformation.
        #Again, lineLengths is not immediately computed, due to laziness.
        lineLengths = lines.map(lambda s: len(s))

        #Finally, we run reduce, which is an action.
        #At this point Spark breaks the computation into tasks to run on separate machines, and each machine runs
        #both its part of the map and a local reduction, returning only its answer to the driver program.
        totalLength = lineLengths.reduce(lambda a, b: a + b)

        print(totalLength)

        7644
```

Process text file as RDD in spark

```
In [13]: #Read text file, you can also provide HDFS URI
         text_file = sc.textFile('1.txt')

         #MapReduce operation from Spark
         counts = text_file.flatMap(lambda line: line.split(' '))\
                            .map(lambda word: (word, 1))\
                            .reduceByKey(lambda a, b: a + b)\
                            .sortBy(lambda x: x[1], False)

         #Option 1: print output
         output = counts.collect()
         for (word, count) in output:
             print("%s: %i" % (word, count))

         #Option 2: save as text file
         #counts.saveAsTextFile("output.txt")
```

```
dictum: 1
nulla: 1
posuere.: 1
cursus: 1
bibendum: 1
pellentesque: 1
Pellentesque: 1
malesuada: 1
nunc: 1
mollis: 1
```

Read a text file and count the word in the document

```
dictum: 1
nulla: 1
posuere.: 1
cursus: 1
bibendum: 1
pellentesque: 1
Pellentesque: 1
malesuada: 1
nunc: 1
mollis: 1
at.: 1
lacus.: 1
Maecenas: 1
ex: 1
condimentum: 1
suscipit.: 1
ut,: 1
lacinia: 1
varius: 1
elementum,: 1
```

Here is part of the word count result

You can also provide HDFS URI for batch processing

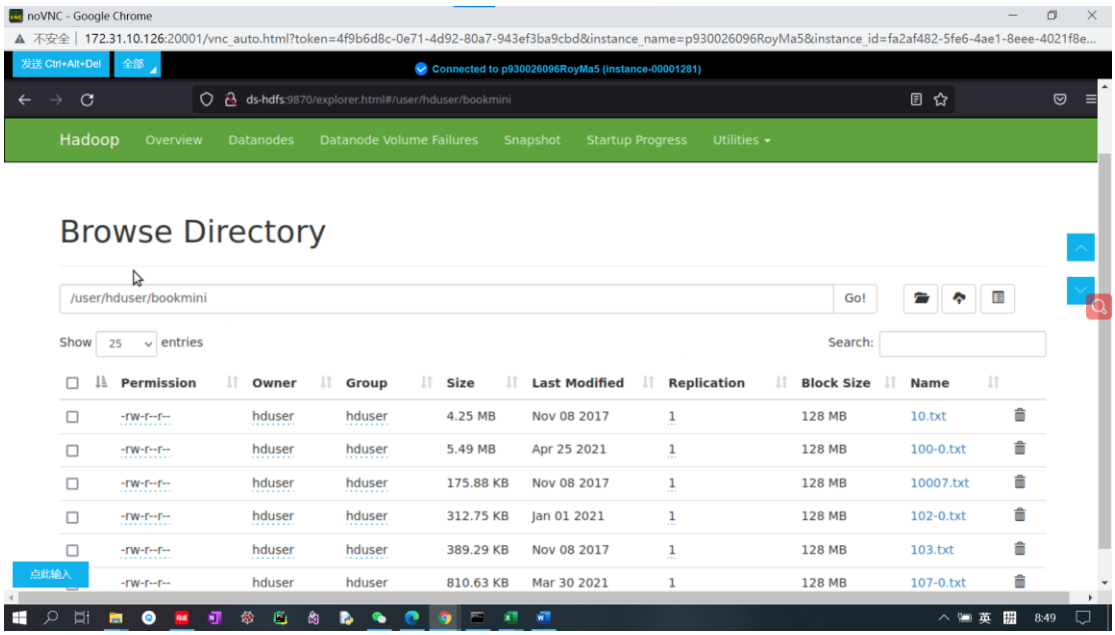
Assuming you have HDFS server @ ds-hdfs:9000

You can also run this command from CLI of your Spark Master node:

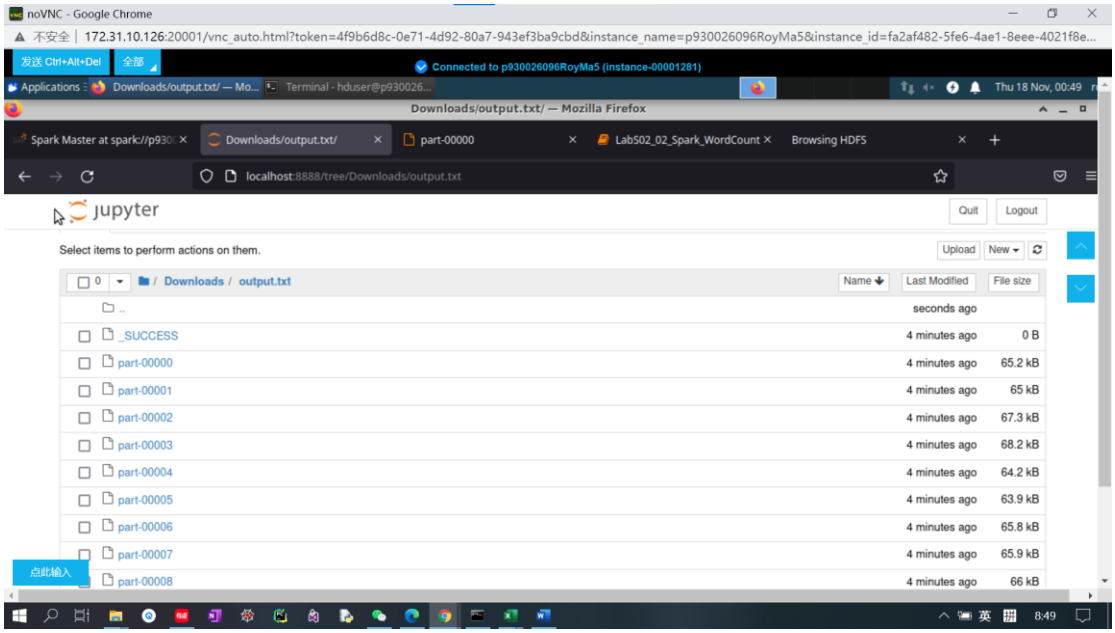
bin/spark-submit –master spark://dpw2tcxu:7077 examples/src/main/python/wordcount.py "hdfs://ds-hdfs:9000/user/hduser/input/*.txt"

```
In [6]: from pyspark.sql import SparkSession

        spark = SparkSession.builder.master('spark://p930026096:7077').appName('MyWordCount').getOrCreate()
        sc = spark.sparkContext

        text_file = sc.textFile('hdfs://ds-hdfs:9000/user/hduser/bookmini/*.txt')
        counts = text_file.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)
        counts.saveAsTextFile('output.txt')
```

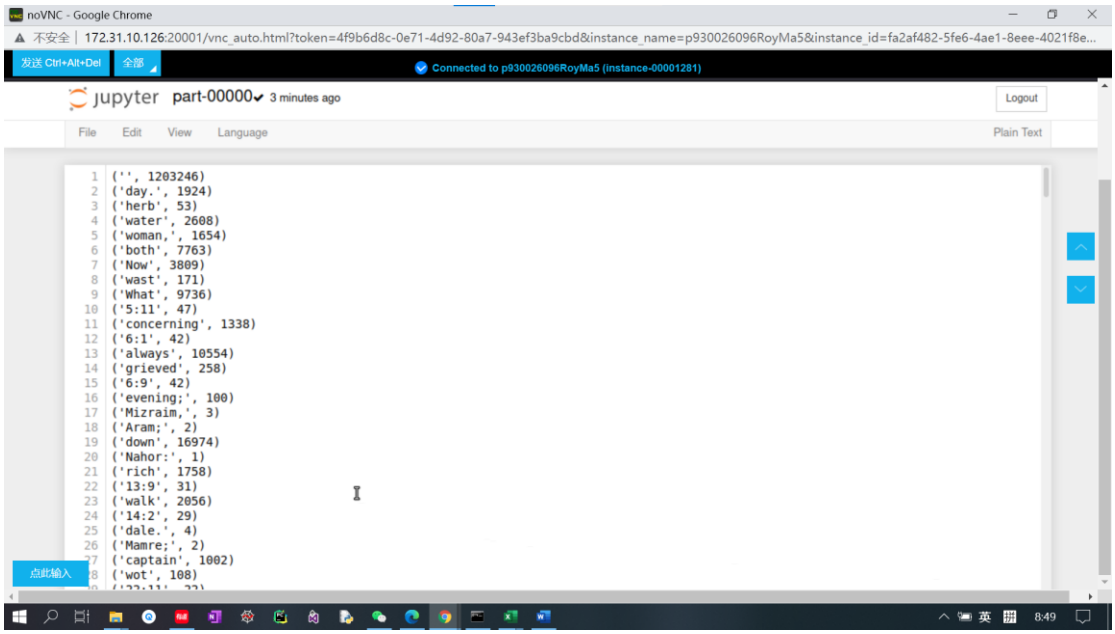
Indicating the HDFS URL. Attach files in HDFS as the input text files to apply word count to every file in bookmini directory



The books in bookmini directory



The output file. Each file contains corresponding word count result



We open the first result file as an example.