

# ICP 2

<https://github.com/JAWolfe04/CS5542/wiki/ICP-2>

## Introduction

---

Write a spark program to group the words in a given text file based on their starting letters.

Use the Text File provided with ICP ([icp2.txt](#))

### Example Input

United States Incident  
Separated Unified  
Investments Board



### Example Output

U, United, Unified  
S, States, Separated  
I, Incident, Investments  
B, Board

ICP Requirements:

1. Spark Integration with Colab (or IDE that you are using) (50 points)
2. Creating a well commented Spark program and outputting the correct results and writing it to output file. (40 points)
3. Code quality, Pdf Report quality, video explanation (10 points)

## Spark

---

[Code](#)  
[output.txt](#)

The first part of the code involved downloading java, spark and pyspark:

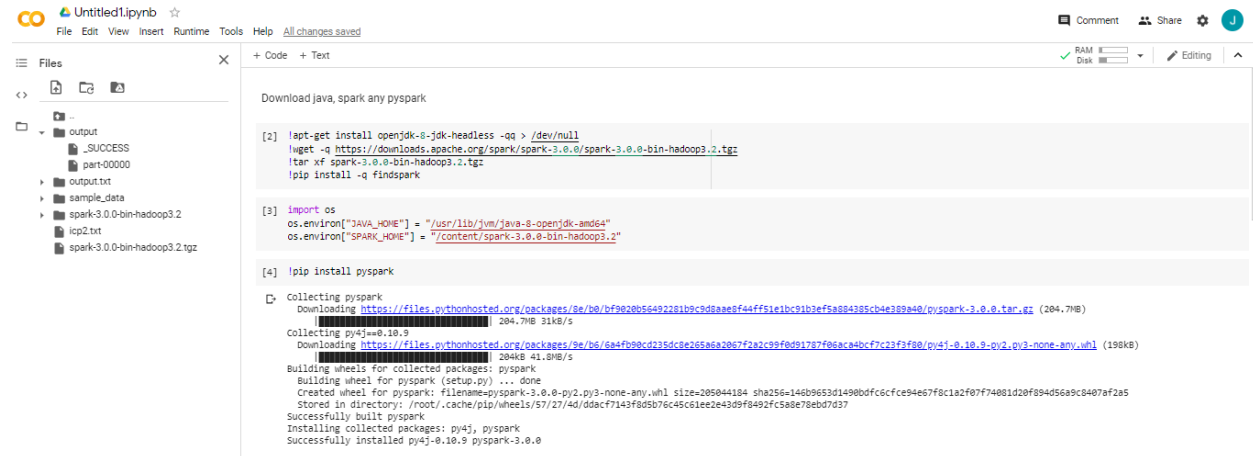
```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
!tar xf spark-3.0.0-bin-hadoop3.2.tgz
!pip install -q findspark
```

```
import os
```

```
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"
```

```
!pip install pyspark
```

The following is a screenshot of the code executed in Google Colab:



The screenshot shows the Google Colab interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'output', '\_SUCCESS', 'part-00000', 'output.txt', 'sample\_data', 'spark-3.0.0-bin-hadoop3.2', 'icp2.txt', and 'spark-3.0.0-bin-hadoop3.2.tgz'. The code editor shows the following code:

```
[2] !apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
!tar -xzf spark-3.0.0-bin-hadoop3.2.tgz
!pip install -q findspark

[3] !import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"

[4] !pip install pyspark
```

The output of the code shows the successful installation of Spark and PySpark. It includes details about downloading the Spark binaries and the PySpark package, and the successful installation of the PySpark package.

The next part involved preparing the Spark session, creating a spark context to create a RDD and importing the data into an RDD:

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
spark =
SparkSession.builder.master("local[*]").appName("Word_Count_Application").getOrCreate()
sc = spark.sparkContext

from google.colab import files
files.upload()

word_data = sc.textFile("icp2.txt")
```



The screenshot shows the Google Colab interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'output', '\_SUCCESS', 'part-00000', 'output.txt', 'sample\_data', 'spark-3.0.0-bin-hadoop3.2', 'icp2.txt', and 'spark-3.0.0-bin-hadoop3.2.tgz'. The code editor shows the following code:

```
[5] from pyspark.sql import SparkSession
from pyspark import SparkContext
spark = SparkSession.builder.master("local[*]").appName("Word_Count_Application").getOrCreate()
sc = spark.sparkContext

Download the text file and save it as a RDD

[7] from google.colab import files
files.upload()

[8] word_data = sc.textFile("icp2.txt")
```

The output of the code shows the successful preparation of a Spark session and the upload of a file. It includes details about the Spark session configuration and the file upload process.

The next step performs all of the processing of the data using map-reduce. First map the data to remove anything but whitespaces and alphanumeric characters to remove all punctuation. The re library is imported to perform this regex processing. The next 2

flatmaps divide the data into words separated by either spaces or tabs. The next 2 maps convert each word to lowercase and creates a pairing of the word as a key and the value. The next reduceByKey reduces all of same words together. The following map creates a pairing with the first letter of the word as the key and the word itself as the value. Then words starting with the same first letters are grouped together and then the groups are sorted by the first letter key.

```
import re
words = word_data.map(lambda s: re.sub(r'^\w\s', '', s)) \
.flatMap(lambda line: line.split(' ')) \
.flatMap(lambda line: line.split('\t')) \
.map(lambda s: s.lower()) \
.map(lambda word: (word, word)) \
.reduceByKey(lambda word1, word2: word1) \
.map(lambda key : (key[0][0], key[0])) \
.reduceByKey(lambda key, word: key + ", " + word) \
.sortByKey()
```

The following shows the map-reduce processing:

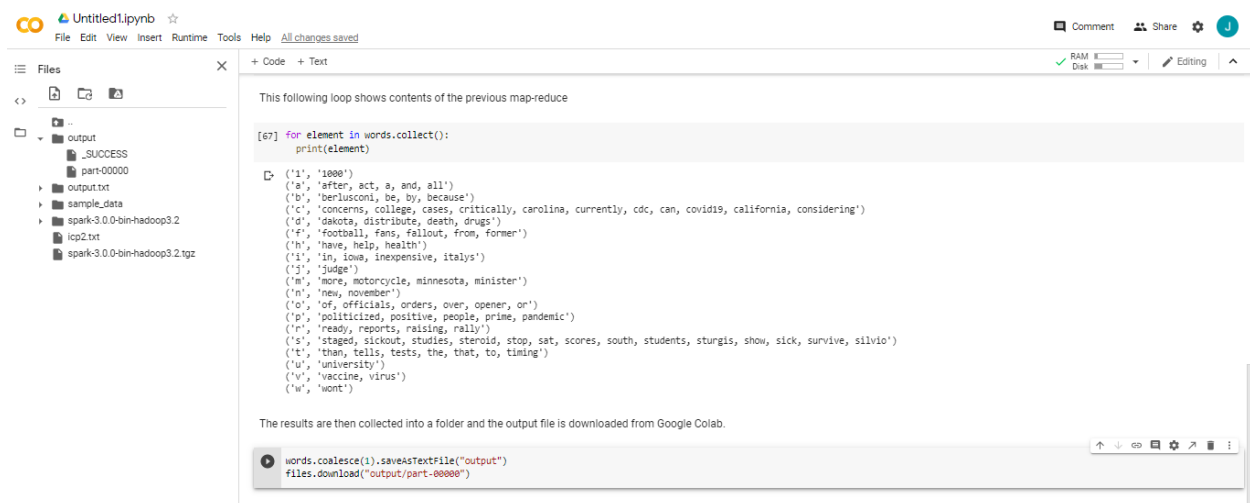
First map the data to remove anything but whitespaces and alphanumeric characters to remove all punctuation. The next 2 flatmaps divide the data into words separated by either spaces or tabs. The next 2 maps convert each word to lowercase and creates a pairing of the word as a key and the value. The next reduceByKey reduces all of same words together. The following map creates a pairing with the first letter of the word as the key and the word itself as the value. Then words starting with the same first letters are grouped together and then the groups are sorted by the first letter key.

```
[66] import re
words = word_data.map(lambda s: re.sub(r'^\w\s', '', s)) \
.flatMap(lambda line: line.split(' ')) \
.flatMap(lambda line: line.split('\t')) \
.map(lambda s: s.lower()) \
.map(lambda word: (word, word)) \
.reduceByKey(lambda word1, word2: word1) \
.map(lambda key : (key[0][0], key[0])) \
.reduceByKey(lambda key, word: key + ", " + word) \
.sortByKey()
```

Finally, the results of the map-reduce are shown and output to a file:

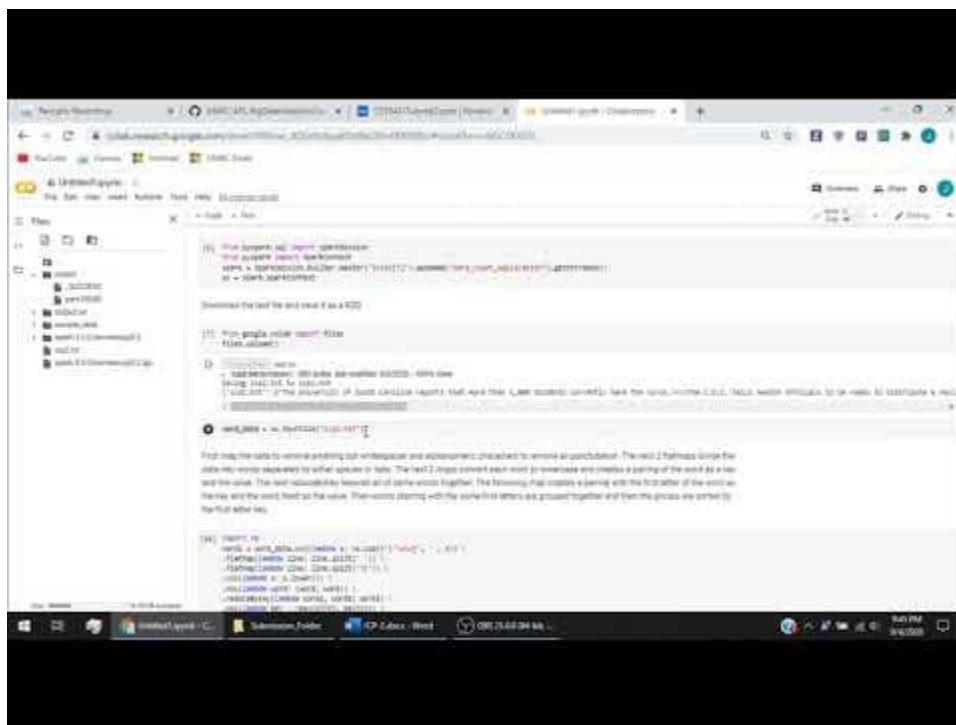
```
for element in words.collect():
    print(element)

words.coalesce(1).saveAsTextFile("output")
files.download("output/part-00000")
```



## Youtube Video

<https://www.youtube.com/watch?v=g-bki8LW1YA>



## References

No references were used in this ICP