1 Installation

The OS used was Ubuntu 18.04. Hadoop 2.9.2 was already installed in the system.

```
(base) dell@dell-Inspiron-3558:~$ hadoop version
Hadoop 2.9.2
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 826afbeae31ca68
7bc2f8471dc841b66ed2c6704
Compiled by ajisaka on 2018-11-13T12:42Z
Compiled with protoc 2.5.0
From source with checksum 3a9939967262218aa556c684d107985
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-2
.9.2.jar
(base) dell@dell-Inspiron-3558:~$
```

PySpark was installed by following the steps given at : https://towardsdatascience.com/installing-pyspark-with-java-8-on-ubuntu-18-04-6a9dea915b5b

```
(base) dell@dell-Inspiron-3558:~$ cd $SPARK_HOME
(base) dell@dell-Inspiron-3558:~/Downloads/spark-2.4.5-bin-hadoop2.7$ cd bin
(base) dell@dell-Inspiron-3558:~/Downloads/spark-2.4.5-bin-hadoop2.7/bin$ spark-
shell --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5
      /_/

Using Scala version 2.11.12, OpenJDK 64-Bit Server VM, 1.8.0_252
Branch HEAD
Compiled by user centos on 2020-02-02T19:38:06Z
Revision cee4ecbb16917fa85f02c635925e2687400aa56b
Url https://gitbox.apache.org/repos/asf/spark.git
Type --help for more information.
(base) dell@dell-Inspiron-3558:~/Downloads/spark-2.4.5-bin-hadoop2.7/bin$
```

To begin PySpark, open a terminal and run:

```
(base) dell@dell-Inspiron-3558:~$ pyspark
```

On running this line, jupyter notebook will open.

To run the given python notebooks, java, PySpark and jupyter notebook are required. Hadoop isn't necessary for running PySpark, so even without hadoop these notebooks can be executed. All the mandatory exercises have been executed using PySpark.

2 Demo Exercises

Try executing the demo exercises for Hadoop/Spark for:

1. Matrix Multiplication
2. Word Count

2.1 Matrix Multiplication

For both of the above exercises, Map and Reduce techniques will be used to get the output.

We are showing 2 different matrix multiplications. The first is a very simple multiplication to check if it is working correctly. The input is found in "matrix_1.txt". The format of input for each line is: $\{matrixName, i, j, value\}$, where (i, j) represents the position of $value$ in the matrix $matrixName$. The text file is equivalent to the matrices:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

So,

$$A * B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},$$

Refer to the jupyter notebook "1.1_MatrixMultiplication.ipynb" for step by step explanation and outputs of intermediate steps. The output shown for the first matrix multiplication is:

```
In [42]: matrix = reduced_numbers.map(lambda x : (x[0],map_func2(x[1])))
         matrix.collect()
Out[42]: [('1,1', '4'), ('0,0', '1'), ('0,1', '2'), ('1,0', '3')]
```

Although the values aren't sorted by their position, we can see that the output is correct. Now, we apply matrix multiplication on 2 slightly larger matrices. The input is given in the file "matrix_2.txt".

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 3 & 1 \\ 0 & -1 \\ -2 & 3 \end{bmatrix}$$

So,

$$A * B = \begin{bmatrix} 5 & -4 \\ 4 & 5 \end{bmatrix},$$

The output from the program is:

```
In [46]: matrix = reduced_numbers.map(lambda x : (x[0],map_func2(x[1])))
         matrix.collect()

Out[46]: [('1,1', '5'), ('0,0', '5'), ('0,1', '-4'), ('1,0', '4')]
```

We can see that this output is also correct.

2.2 Word Count

The input file is "wc_input.txt". The intermediate steps can be seen in the jupyter notebook, "1.2_WordCount.ipynb". The output is:

```
In [30]: wordCounts = wordCounts.reduceByKey(lambda a,b:a +b)
         wordCounts.collect()

Out[30]: [('PySpark', 2),
          ('is', 3),
          ('python', 1),
          ('binding', 1),
          ('Spark', 1),
          ('Platform', 1),
          ('different', 1),
          ('Java/Scala', 1),
          ('good', 1),
          ('starting', 1),
          ('point', 1),
          ('official', 1),
          ('page', 1),
          ('|', 1),
          ('Apache', 1),
          ('Spark.', 1),
          ('Python', 1),
          ('dynamically', 1),
          ('of', 1),
          ('multiple', 1),
          ('yet', 1),
          ('support', 1),
          ('as', 1),
          ('input', 1),
          ('files,', 1),
          ('though', 1),
          ('these', 1),
          ('added', 1),
          ('in', 1),
```

```
('the', 4),
('for', 1),
('and', 3),
('API', 2),
('not', 2),
('much', 1),
('from', 1),
('versions.', 1),
('A', 1),
('i.e', 1),
('Examples', 1),
('typed,', 1),
('so', 1),
('RDDs', 1),
('can', 1),
('hold', 1),
('objects', 1),
('types.', 1),
('does', 1),
('a', 1),
('few', 1),
('calls,', 1),
('such', 1),
('lookup', 1),
('non-text', 1),
('will', 1),
('be', 1),
('future', 1),
('releases.', 1)]
```

3 FP-Growth

Implement any one Frequent Pattern Mining (FPM) algorithm utilising any larger dataset available over internet like Femi using Hadoop/Pyspark.

Details:

1. The input dataset is "accidents.dat". Source: http://fimi.uantwerpen.be/data/
2. Mapping is done to get the transactions from the dataset.
3. After this FP-Growth is applied using the in-built package available in PySpark. A minimum support of 80% was considered to get frequent itemsets.
4. The intermediate steps and the output can be seen in the jupyter notebook, "2_FPGrowth.ipynb".

4 Classification

Test any one classification (or) clustering algorithm utilising any larger dataset available over internet like Femi using Hadoop/Pyspark.

Details:

1. The input dataset is "heart.csv". Source: https://www.kaggle.com/ronitf/heart-disease-uci
2. Random Forest Classifer with 10 trees was applied on this dataset. This classifer is an in-built package in PySpark.
3. The output of the intermediate steps can be seen in "3_Classification.ipynb".
4. The accuracy of the trained model was 85.0575%.