Frequent Itemsets with PySpark in Colab

To run spark in Colab, we need to first install all the dependencies in Colab environment i.e. Apache Spark 2.3.2 with hadoop 2.7, Java 8 and Findspark to locate the spark in the system.

Follow the steps to install the dependencies:

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

```
!wget -qN https://archive.apache.org/dist/spark/spark-3.2.1/spark-3.2.1-bin-hadoop3.2.tgz
!tar xf spark-3.2.1-bin-hadoop3.2.tgz
```

```
!pip install -q findspark
```

Set the location of Java and Spark by running the following code:

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "spark-3.2.1-bin-hadoop3.2"
```

Install PySpark and run a local spark session to test the installation:

```
!pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     |████████████████████████████████| 281.4 MB 37 kB/s
Collecting py4j==0.10.9.3
  Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 52.9 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.2.1-py2.py3-none-any.whl size=281853642
  Stored in directory: /root/.cache/pip/wheels/9f/f5/07/7cd8017084dce4e93e84e92efd1e1d5
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.3 pyspark-3.2.1
```

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
```

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

Let's create a spark DataFrame to confirm that we can run PySpark, and preload that DataFrame with test baskets.

Transaction ID	Stock Items
100	milk, coke, beer
200	milk, pepsi, juice
300	milk, beer
400	coke, juice
500	milk, pepsi, beer
600	milk, coke, beer, juice
700	coke, beer, juice
800	beer, coke

Each DataFrame row is `<Transaction ID, [Stock Items]>`

```
m,c,b,p,j = 12,3,2,15,9
basket_df = spark.createDataFrame([
    (100, [m,c,b]),
    (200, [m,p,j]),
    (300, [m,b]),
    (400, [c,j]),
    (500, [m,p,b]),
    (600, [m,c,b,j]),
    (700, [c,b,j]),
    (800, [b,c])
], ["id", "items"])
basket_df.show()
stockIDs = {b: 'Beer', c: 'Coke', m: 'Milk', j: 'Juice', p: 'Pepsi'}
```

```
+---+-------------+
| id|        items|
+---+-------------+
|100|   [12, 3, 2]|
|200|  [12, 15, 9]|
|300|      [12, 2]|
|400|       [3, 9]|
|500|  [12, 15, 2]|
|600|[12, 3, 2, 9]|
|700|    [3, 2, 9]|
|800|       [2, 3]|
+---+-------------+
```

PySpark Code

FP-Growth Algorithm

Ready to run FP-Growth? References:

- PySpark introduction for FP-Growth.
- PySpark Dataframes

First, run FP-Growth example from the documentation

```
from pyspark.ml.fpm import FPGrowth

fpGrowth = FPGrowth(itemsCol="items", minSupport=0.5, minConfidence=0.6)
model = fpGrowth.fit(basket_df)

# Skipping display of frequent itemsets.
# model.freqItemsets.show()

# Display generated association rules.
model.associationRules.show()
```

```
spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:127: FutureWarning: Deprecated i
  FutureWarning
+----------+----------+------------------+------------------+-------+
|antecedent|consequent|        confidence|              lift|support|
+----------+----------+------------------+------------------+-------+
|       [3]|       [2]|               0.8|1.0666666666666667|    0.5|
|      [12]|       [2]|               0.8|1.0666666666666667|    0.5|
|       [2]|       [3]|0.6666666666666666|1.0666666666666667|    0.5|
|       [2]|      [12]|0.6666666666666666|1.0666666666666667|    0.5|
+----------+----------+------------------+------------------+-------+
```

Q1. Interpreting association rules [15]

The above table, has columns antecedent, consequent, confidence, lift and support.

1. Explain the first row, `[3] [2] 0.8 1.0666666666666667 0.5` in plain English.
2. The first and the third rows have the antecedent and consequent switched, but different confidence values. (Same with second and fourth rows). How do you explain those results?
3. What does support = 0.5 for all the rows mean?

1. the first row says that confidence of [3] that has consequent on [2] is 0.8 which means that 80% items that contains [3] contains [2]. Lifts is the conditional probability in math, in other word [3] helps [2] to lift the probability of having [2] since lift is greater than 1(probability that

given 3 that has 2 over the probality of 2 is 1.067). Support says half of the items have both [2] and [3].

2. When [3] is in the items there are 0.8 chance that [2] will be in the items as well, but when [2] is in the items there are only 0.67 chance that [3] is also in the chart. It says that [2] is more 'necessary' to [3] than [3] to [2]. Namely, [3] lifts the probability of [2]
3. half of items contains both [2] and [3] and half of items contains [2] and [12]

Association Rules with changed minSupport and minConfidence values

Modify the support threshold to be 0.375 and minimum confidence to be 0.75 to make the parameters consistent with the settings in the textbook.

```
from pyspark.ml.fpm import FPGrowth

fpGrowth = FPGrowth(itemsCol="items", minSupport=0.375, minConfidence=0.75)
model = fpGrowth.fit(basket_df)

# Display frequent itemsets.
model.freqItemsets.show()

# Display generated association rules.
model.associationRules.show()
```

```
spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:127: FutureWarning: Deprecated i
  FutureWarning
+-------+----+
|  items|freq|
+-------+----+
|    [3]|   5|
| [3, 2]|   4|
|    [2]|   6|
|   [12]|   5|
|[12, 2]|   4|
|    [9]|   4|
| [9, 3]|   3|
+-------+----+

+----------+----------+----------+------------------+-------+
|antecedent|consequent|confidence|              lift|support|
+----------+----------+----------+------------------+-------+
|       [3]|       [2]|       0.8|1.0666666666666667|    0.5|
|      [12]|       [2]|       0.8|1.0666666666666667|    0.5|
|       [9]|       [3]|      0.75|               1.2|  0.375|
+----------+----------+----------+------------------+-------+
```

Q2. Interpreting the new association rules [10]

The third row of the result shows a low support (0.375) and a high lift (1.2). What does this line tell us?

There are not many items that contains 9, more items having both 3 and 9 than 9 without 3 which means that possibly having 9 is to have 3

Q3. Association Rules for an Online Retail Dataset [5]

The main part of this exercise involves processing a sampled dataset from a UK-based online retailer. We'll be working with a 8050 record subset.

- Read in the data from the dataset `online_retail_III.csv`. For your convenience, I have already thrown away bad records using `dropna()`.
- There are a couple of wrinkles to keep in mind in case you are curious, though you may not really need them.
  - An invoice represents a shopping cart and it can contain multiple items.
  - Some invoice numbers start with a "C." Invoice number C123456 is to be interpreted as a return of items in invoice 123456. The `inum` column represents the Invoice number as well as the credit (return). In other words, Invoice numbers `123456` and `C123456` would have `inum` == 123456.

```
import pandas as pd
df_orig = pd.read_csv('https://storage.googleapis.com/119-quiz7-files/online_retail_II.csv')
df_orig.dropna(inplace=True)
df_orig.drop(df_orig[df_orig['StockCode'] == 'POST'].index, inplace = True)
df_orig.drop(df_orig[df_orig['StockCode'] == 'M'].index, inplace = True)
df = df_orig
df.drop(['Description', 'Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country'], axis =
df2 = df.groupby('Invoice')['StockCode'].unique().apply(list).reset_index(name="StockCode")
# df3 = df2.groupby('Invoice')['StockCode'].apply(list).reset_index(name="StockCode")
df2
```

	Invoice	StockCode
0	489434	[85048, 79323P, 79323W, 22041, 21232, 22064, 2...
1	489435	[22350, 22349, 22195, 22353]
2	489436	[48173C, 21755, 21754, 84879, 22119, 22142, 22...
3	489437	[22143, 22145, 22130, 21364, 21360, 21351, 213...
4	489438	[21329, 21252, 21100, 21033, 20711, 21410, 214...
...	...	...
44160	C581470	[23084]

Data Scrubbing

Remove the rows we should filter away. They aren't necessarily visible in the summary view but we know they exist.

- StockCode `POST`,
- StockCode `M`.

Q4. Connecting Online Retail Data to FP-Growth [30]

Adapt the DataFrame to look like `df_basket` above.

- `df_orig` is a Pandas DataFrame whereas `df_basket`-equivalent will have to be Spark DataFrames.
- `Invoice` and `StockCode` are strings but FP-Growth needs inputs to be integers. You'd need to map strings to integers before feeding them to FP-Growth and convert the resulting antecedents and consequents back.

```
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
# df.astype({'Invoice': 'int32'}).dtypes
mySchema = StructType([StructField("Invoice", StringType(), True),StructField("StockCode", St
sparkDF = spark.createDataFrame(df2)
sparkDF.printSchema()
sparkDF.show()
```

```
root
 |-- Invoice: string (nullable = true)
 |-- StockCode: array (nullable = true)
 |    |-- element: string (containsNull = true)

+-------+--------------------+
|Invoice|           StockCode|
+-------+--------------------+
```

```
| 489434|[85048, 79323P, 7...|
| 489435|[22350, 22349, 22...|
| 489436|[48173C, 21755, 2...|
| 489437|[22143, 22145, 22...|
| 489438|[21329, 21252, 21...|
| 489439|[22065, 22138, 22...|
| 489440|      [22350, 22349]|
| 489441|[22321, 22138, 84...|
| 489442|[21955, 22111, 22...|
| 489443|[20754, 21035, 22...|
| 489445|[35916C, 35916B, ...|
| 489446|[21733, 85123A, 2...|
| 489448|[20827, 20825, 20...|
| 489450|[22087, 85206A, 2...|
| 489460|[79323P, 21977, 8...|
| 489461|[21668, 21669, 21...|
| 489462|[90200D, 90200E, ...|
| 489465|[21707, 21710, 21...|
| 489488|[84031B, 84032A, ...|
| 489505|[21472, 85099B, 4...|
+-------+--------------------+
only showing top 20 rows
```

Establish the mapping between Invoice IDs, StockCodes and unique integers.

Q5. Fine-tuning FP-Growth runs [20]

- Set `minConfidence` = 0.75.
- Set `minSupport` such that the total number of association rules is between 10 and 20. (If `minSupport` is small, the number of association rules will increase. As it increases, the number of association rules will decrease.).

```
fpGrowth = FPGrowth(itemsCol="StockCode", minSupport=0.01, minConfidence=0.75)
# unique = df2.map(lambda x: list(set(x))).cache()
model = fpGrowth.fit(sparkDF)

# Display frequent itemsets.
model.freqItemsets.show()

# Display generated association rules.
model.associationRules.show()
```

```
spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:127: FutureWarning: Deprecated i
  FutureWarning
+--------------------+----+
|               items|freq|
+--------------------+----+
|             [20718]| 874|
```

```
|             [22427]| 721|
|             [22147]|1039|
|             [21592]| 467|
|             [22296]| 742|
|             [21429]| 783|
|             [21669]| 652|
|             [22418]| 482|
|            [47590B]| 820|
|             [22467]| 992|
|            [72760B]| 449|
|             [22961]| 916|
|             [21509]| 500|
|             [21484]| 680|
|             [21507]| 472|
|             [22697]|1004|
|      [22697, 22423]| 539|
|      [22697, 22699]| 774|
|[22697, 22699, 22...| 449|
|             [82580]| 935|
+--------------------+----+
only showing top 20 rows

+----------------+----------+------------------+------------------+--------------------+
|      antecedent|consequent|        confidence|              lift|             support|
+----------------+----------+------------------+------------------+--------------------+
|         [22698]|   [22697]|0.8154613466334164|  35.8713649144072|0.014808105966262877|
|         [22698]|   [22699]|0.7793017456359103|30.431354196295295|0.014151477414242046|
|  [22698, 22699]|   [22697]|            0.8832| 38.85112350597609|0.012498584852258576|
|         [21124]|   [21122]|0.8023255813953488| 47.69139879182447|0.010936261745726254|
|[85099F, 22386]|   [85099B]|0.7538940809968847| 9.959836101473948|0.010958904109589041|
|         [22748]|   [22745]|0.7641357027463651| 60.37218839319001|0.010709838107098382|
|  [22698, 22697]|   [22699]|0.8440366972477065|32.959222576432325|0.012498584852258576|
|  [22697, 22423]|   [22699]|0.8330241187384044|32.529186741009404|0.010166421374391487|
|         [21086]|   [21094]|            0.7872| 49.88047058823529| 0.01114004302049134|
|         [82581]|   [82580]|0.7544097693351425| 35.63476733977173|0.012589154307709726|
|         [22745]|   [22748]|0.8461538461538461| 60.37218839319001|0.010709838107098382|
|         [22697]|   [22699]|0.7709163346613546|30.103907975524958| 0.01752518962979735|
+----------------+----------+------------------+------------------+--------------------+
```

Q6. Final Association Rules [20]

Present the resulting Association Rules in terms of the original StockCodes and Descriptions, in descending order of lift.

Q3. NLTK [60 pts]

a. Convert all words to lowercase,

b. Use NLTK to break the poem into sentences & sentences into tokens. Here is the code for doing that, after you set the variable paragraph to hold the text of the poem.

c. Tag all remaining words in the poem as parts of speech using the Penn POS Tags. This SO answer shows how to obtain the POS tag values. Create and print a dictionary with the Penn POS Tags as keys and a list of words as the values.

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.data import load
from collections import defaultdict

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('tagsets')
tagdict = load('help/tagsets/upenn_tagset.pickle')

poem = "My first week in Cambridge a car full of white boys tried to run me off the road, and
def tag(poem, tagdict):

  poem = poem.lower()

  # nltk.download()


  sent_text = nltk.sent_tokenize(poem) # this gives us a list of sentences
  # now loop over each sentence and tokenize it separately
  all_tagged = [nltk.pos_tag(nltk.word_tokenize(sent)) for sent in sent_text]


  data ={k : set() for k in tagdict.keys()}

  for itemset in all_tagged:
      for item in itemset:
          data[item[1]].add(item[0])

  return data

print(tag(poem, tagdict))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package tagsets to /root/nltk_data...
[nltk_data]   Unzipping help/tagsets.zip.
{'LS': set(), 'TO': {'to'}, 'VBN': {'gyrated', 'stuffed', 'drummed'}, "''": set(), 'WP'
```

4. The main point of this exercise is to set up a PySpark DataFrame as a structure for analyzing large numbers of such poems. This structure is designed such that hundreds of Spark workers can be deployed to do similar analysis for different poems in parallel.

   Each column row will represent a poem. The rows columns will be as follows:

   The text of the poem,

   Two-letter prefixes of each tag,

   For example NN, VB, RB, JJ etc.and the words belonging to that tag in the poem.

```
import pandas as pd

def pdFrame(poems):

  poemDF = pd.DataFrame()
  for key in tagdict.keys():
    poemDF[key] = None

  for title in poems.keys():
    poem = poems[title]
    tags = tag(poem, tagdict)
    temp = []
    for key in tagdict.keys():
      if len(tags.get(key)) != 0:
        temp.append(tags[key])
      else:
        temp.append("")
    poemDF.loc[title] = temp



  return poemDF

poems = {}
poems['first poem'] = poem

pdFrame(poems)
```

	LS	TO	VBN	''	WP	UH	VBG	JJ	VBZ	--	...	MD	
first poem		{to}	{gyrated, stuffed,		{what}		{singing, eating, asking, string,	{i, tiny, trolley, first, unreadable,	{leaves}		...	{'d, would}	{... disa