Question 1:

1. Codes:

```
import requests
stopwords_list =
requests.get("https://gist.githubusercontent.com/rg089/35e00abf8941d72
d419224cfd5b5925d/raw/12d899b70156fd0041fa9778d657330b024b959c/stopwor
ds.txt").content
stopwords = set(stopwords_list.decode().splitlines())
stopwords = list(stopwords)

def preprocess(doc):
      list_words = doc.split(" ")
      # remove stop words and lowercase
      list_words = [word.lower() for word in list_words if word not in
stopwords]
      # re-join with spaces
      processed = ' '.join(list_words)
      return processed

def shingles(doc, n):
      shingling = set()
      for i in range(len(doc) - n + 1):
            shingling.add(doc[i:i+n])
      return shingling
```

2. Codes:

```
def jaccard(sh_set_1, sh_set_2):
      set_union = sh_set_1.union(sh_set_2)
      set_intersection = sh_set_1.intersection(sh_set_2)
      return len(set_intersection)/float(len(set_union))
```

3. Codes:

```
doc1 = 'Life is suffering'
doc2 = 'Suffering builds character'
doc3 = 'Character is the essence of life'

doc1 = preprocess(doc1)
doc2 = preprocess(doc2)
doc3 = preprocess(doc3)

sh_set1 = shingles(doc1, 2)
sh_set2 = shingles(doc2, 2)
sh_set3 = shingles(doc3, 2)

print("Jaccard between doc1 and doc2: {}".format(jaccard(sh_set1,
sh_set2)))
print("Jaccard between doc2 and doc3: {}".format(jaccard(sh_set2,
sh_set3)))
print("Jaccard between doc3 and doc1: {}".format(jaccard(sh_set3,
sh_set1)))
```

<u>Output:</u>

Jaccard between doc1 and doc2: 0.2857142857142857
Jaccard between doc2 and doc3: 0.25
Jaccard between doc3 and doc1: 0.17857142857142858

__Question 2:__

1. Interpret association rules:
    a. We are examining the pattern when the purchase of item 3 (coke) leads to the purchase of item 2 (beer). The support is 0.5, meaning that in the whole database, the item 3 and 2 appear together in half of the transactions. The confidence is 0.8, meaning that out of all the transactions that contain item 3, 80% of them also contain item 2. The lift value greater than 1 indicates that item 3 and item 2 appear more often together than expected, which means that the occurrence of item 3 has a positive effect on the occurrence of item 2.
    b. The confidence is different because they mean quite opposite things. The confidence of the first rule tells us how frequent both items 3 and 2 appear together, out of all the times **item 3 appears**, whereas the third rule tells us how frequent both items appear together out of all the times **item 2 appears** (so the denominator is different). The same thing applies for the second and fourth rule).
    c. The supports of 0.5 indicate that each of these pairs of items is observed in half of the transactions.
2. Interpret association rules:
   The third row of the result shows a low support (0.375), meaning that item 3 (coke) and 9 (juice) are not purchased together for a lot of times. A high lift (1.2) indicates that the occurrence of item 9 has a positive effect on the occurrence of item 3. Together (also combined with the confidence), we see that item 9 is not frequently bought, but when it is, it's likely to lead to item 3 being purchased alongside.

3, 4, 5, 6: In the attached notebook.

__Question 3:__

1. Codes (`boston_year` represents the poem text)
```
list_words = boston_year.split(" ")
list_words = [word.lower() for word in list_words]
reconstruct = " ".join(list_words)
```

2. Example of output
```
[[('my', 'PRP$'), ('first', 'JJ'), ('week', 'NN'), ('in', 'IN'), ('cambridge', 'NN'), ('a', 'DT'), ('car', 'NN'), ('full',
 ('off', 'IN'), ('the', 'DT'), ('road', 'NN'), (',', ','), ('and', 'CC'), ('spit', 'NN'), ('through', 'IN'), ('the', 'DT'
i', 'NN'), ('was', 'VBD'), ('always', 'RB'), ('asking', 'VBG'), ('directions', 'NNS'), ('and', 'CC'), ('always', 'RB'), (
town', 'NN'), ('to', 'TO'), ('buy', 'VB'), ('figs', 'NNS'), ('and', 'CC'), ('string', 'VBG'), ('cheese', 'NN'), (',', ','
'), ('barrels', 'NNS'), (',', ','), ('tubes', 'NNS'), ('of', 'IN'), ('paste', 'NN'), ('with', 'IN'), ('unreadable', 'JJ')
, ('leaves', 'VBZ'), ('and', 'CC'), ('watched', 'VBD'), ('my', 'PRP$'), ('lips', 'NNS'), ('swell', 'VBP'), ('in', 'IN'),
```

3. The codes:
```
tagdict = load('help/tagsets/upenn_tagset.pickle')

from collections import defaultdict
pos_dict = defaultdict(list)
```

```
for sent in all_tagged:
      for word in sent:
            tag = word[1][:2]
            if word[0] not in pos_dict[tag]:
                  pos_dict[tag].append(word[0])
```

Output for the first few tags:

```
PR
['my', 'me', 'they', 'them', 'you', 'he']

JJ
['first', 'full', 'white', 'open', 'armenian', 'dark', 'unreadable', 'arabic', 'i', 'other', 'colored', 'smile', 'countless', 'chinese', 'portuguese', 'red', 'entire', 'brazilian', '
tiny', 'festooned', 'certain', 'fabergé', 'harriet', 'grey', 'trolley']

NN
['week', 'cambridge', 'car', 'boys', 'road', 'spit', 'window', 'directions', 'i', 'driving', 'market', 'watertown', 'figs', 'cheese', 'apricots', 'spices', 'olives', 'barrels', 'tube
s', 'paste', 'labels', 'grape', 'lips', 'mirror', 'floors', 'apartment', 'clean', 'people', 'bookshops', 'museums', 'cafeterias', 'shyly', 'spoke', 'come', 'home', 'mother', 'restaur
ants', 'almond', 'cookies', 'tea', 'spoons', 'sugar', 'popcorn', 'coffee', 'dinner', 'migraine', 'grocery', 'store', 'man', 'breakfast', 'orange', 'juice', 'chocolate', 'bars', 'colo
r', 'sprang', 'relief', 'wagner', 'walküre', 'tribes', 'head', 'samba', 'glitter', 'filigreed', 'egg', 'one', 'door', 'salesmen', 'mormons', 'meter', 'readers', 'exterminators', 'tub
man', 'notes', 'town']

IN
['in', 'of', 'off', 'through', 'from', 'with', 'before', 'above', 'into', 'that', 'inside', 'at']

DT
['a', 'the', 'an', 'no']
```

4. ```
   data = []
   data.append((boston_year, dict(pos_dict)))

   from pyspark import SparkContext,SparkConf
   from pyspark.sql import SQLContext
   sc = SparkContext()
   spark = SQLContext(sc)

   df = spark.createDataFrame(data, ["Poem", "word_dict"])

   from pyspark.sql.functions import explode,map_keys,col
   keysDF = df.select(explode(map_keys(df.word_dict))).distinct()
   keysList = keysDF.rdd.map(lambda x:x[0]).collect()
   keyCols = list(map(lambda x:
   col("word_dict").getItem(x).alias(str(x)), keysList))
   df.select(df.Poem, *keyCols).toPandas()
   print(pandas_df[['Poem', 'NN', 'VB', 'JJ']])
   ```

   The output (Poem is the column that contains the text of the poem):

```
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
                                          Poem                                            NN                                             VB                                             JJ
0  My first week in Cambridge a car full of white...  [week, cambridge, car, boys, road, spit, windo...  [tried, run, ask, was, asking, buy, string, at...  [first, full, white, open, armenian, dark, unr...
```

<u>Question 4</u>: In the attached notebook.
```

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:

In [ ]:

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

In [ ]:

```
!wget -qN https://archive.apache.org/dist/spark/spark-3.2.1/spark-3.2.1-bin-hado
op3.2.tgz
!tar xf spark-3.2.1-bin-hadoop3.2.tgz
```

In [ ]:

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

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

In [ ]:

```
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:

In [ ]:

```
!pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     |████████████████████████████████| 281.4 MB 33 kB/s
Collecting py4j==0.10.9.3
  Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 54.6 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-an
y.whl size=281853642 sha256=34c458f9f1528f93e791011ea107b2f3154e6b65
23f44a69de9a030ba03ed6fd
  Stored in directory: /root/.cache/pip/wheels/9f/f5/07/7cd8017084dc
e4e93e84e92efd1e1d5334db05f2e83bcef74f
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.3 pyspark-3.2.1
```

In [ ]:

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

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]>`

In [ ]:

```
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 (https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html#fp-growth).
- PySpark Dataframes (https://sparkbyexamples.com/pyspark/convert-pandas-to-pyspark-dataframe/)

First, run FP-Growth example from the documentation

In [ ]:

```
from pyspark.ml.fpm import FPGrowth

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

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

```
spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:127: FutureW
arning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate()
instead.
  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] (In the writeup)

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?

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.

In [ ]:

```
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: FutureW
arning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate()
instead.
  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] (In the writeup)

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

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.

In [ ]:

```
import pandas as pd
df_orig = pd.read_csv('https://storage.googleapis.com/119-quiz7-files/online_ret
ail_II.csv')
df_orig.dropna(inplace=True)
df_orig
```

Out[ ]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Countr
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	Unitec Kingdon
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unitec Kingdon
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	Unitec Kingdon
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	Unitec Kingdon
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	Unitec Kingdon
...	...	...	...	...	...	...	...	..
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France
1067370	581587	POST	POSTAGE	1	2011-12-09 12:50:00	18.00	12680.0	France

824364 rows × 8 columns

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`.

In [ ]:

```
# filter out rows with POST and M
df_orig = df_orig[(df_orig.StockCode != 'POST') & (df_orig.StockCode != 'M')]
```

In [ ]:

```
# mkae inum column (remove character 'C' in Invoice)
df_orig.loc[:,'inum'] = [x if x[0] != 'C' else x[1:] for x in df_orig.Invoice]
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1667:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py
  self.obj[key] = value
```

In [ ]:

```
# make into type int
df_orig['inum'] = df_orig['inum'].astype('int')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py
```

In [ ]:

```
# get all unique stockcode to make a mapping of integers to code
stockcode = pd.unique(df_orig.StockCode)
stockcode
```

Out[ ]:

```
array(['85048', '79323P', '79323W', ..., '23562', '23561', '23843'],
      dtype=object)
```

In [ ]:

```
# make the unique integer to stock codes map
code_map = {}
for index, code in enumerate(stockcode):
  code_map[code] = index
```

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.

In [ ]:

```
from collections import defaultdict
df_dict = defaultdict(set)
for _, row in df_orig.iterrows():
  inv = row['inum']
  code_num = code_map[row['StockCode']]
  df_dict[inv].add(code_num)
```

In [ ]:

```
df_list = []
for inv in df_dict:
  df_list.append((inv, list(df_dict[inv])))
```

In [ ]:

```
df_basket = spark.createDataFrame(df_list, ["invoice", "stock"])
df_basket.show()
```

```
+-------+--------------------+
|invoice|               stock|
+-------+--------------------+
| 489434|[0, 1, 2, 3, 4, 5...|
| 489435|      [8, 9, 10, 11]|
| 489436|[12, 13, 14, 15, ...|
| 489437|[30, 31, 32, 33, ...|
| 489438|[64, 65, 66, 67, ...|
| 489439|[33, 5, 70, 71, 7...|
| 489440|              [8, 9]|
| 489441|    [87, 30, 86, 71]|
| 489442|[18, 30, 48, 49, ...|
| 489443|[3, 107, 108, 109...|
| 489445|[128, 33, 71, 86,...|
| 489446|[129, 130, 131, 1...|
| 489448|[152, 149, 150, 151]|
| 489449|[6, 136, 46, 153,...|
| 489450|[6, 136, 46, 153,...|
| 489459|[160, 161, 162, 1...|
| 489460|[1, 2, 71, 72, 17...|
| 489461|[131, 132, 4, 24,...|
| 489462|[160, 161, 162, 1...|
| 489465|[129, 131, 137, 1...|
+-------+--------------------+
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.).

In [ ]:

```
from pyspark.ml.fpm import FPGrowth

fpGrowth = FPGrowth(itemsCol="stock", minSupport=0.01, minConfidence=0.75)
model = fpGrowth.fit(df_basket)
```

In [ ]:

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

```
spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:127: FutureW
arning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate()
instead.
  FutureWarning


+------------+----------+------------------+------------------+-----
---------------+
|  antecedent|consequent|        confidence|              lift|
support|
+------------+----------+------------------+------------------+-----
---------------+
| [855, 3012]|     [140]|0.7538940809968847| 9.959836101473948|0.010
958904109589041|
|[3728, 3729]|    [3727]|0.8440366972477065|32.959222576432325|0.012
498584852258576|
|      [3839]|    [3843]|0.7641357027463651| 60.37218839319001|0.010
709838107098382|
|[3729, 3310]|    [3727]|0.8330241187384044|32.529186741009404|0.010
166421374391487|
|[3728, 3727]|    [3729]|            0.8832| 38.85112350597609|0.012
498584852258576|
|      [3728]|    [3729]|0.8154613466334164|  35.8713649144072|0.014
808105966262877|
|      [3728]|    [3727]|0.7793017456359103|30.431354196295295|0.014
151477414242046|
|      [3843]|    [3839]|0.8461538461538461| 60.37218839319001|0.010
709838107098382|
|       [376]|     [387]|            0.7872| 49.88047058823529| 0.01
114004302049134|
|       [441]|     [440]|0.8023255813953488| 47.69139879182447|0.010
936261745726254|
|      [1013]|     [110]|0.7544097693351425| 35.63476733977173|0.012
589154307709726|
|      [3729]|    [3727]|0.7709163346613546|30.103907975524958| 0.01
752518962979735|
+------------+----------+------------------+------------------+-----
---------------+
```

Q6. Final Association Rules [20]

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

In [ ]:

```
final_rules = model.associationRules.select('antecedent', 'consequent','lift').t
oPandas()
```

```
spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:127: FutureW
arning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate()
instead.
  FutureWarning
```

In [ ]:

```
final_rules.columns = ['Invoice', 'StockCodes', 'Lift']
```

In [ ]:

```
inv_map = {v: k for k, v in code_map.items()}
```

In [ ]:

```
new_list = []
for index, c in enumerate(final_rules.StockCodes):
  map_back = inv_map[c[0]]
  new_list.append(map_back)
new_list
```

Out[ ]:

```
['85099B',
 '22699',
 '22745',
 '22699',
 '22697',
 '22697',
 '22699',
 '22748',
 '21094',
 '21122',
 '82580',
 '22699']
```

In [ ]:

```
final_rules['StockCodes'] = new_list
final_rules
```

Out[ ]:

	Invoice	StockCodes	Lift
0	[855, 3012]	85099B	9.959836
1	[3728, 3729]	22699	32.959223
2	[3839]	22745	60.372188
3	[3729, 3310]	22699	32.529187
4	[3728, 3727]	22697	38.851124
5	[3728]	22697	35.871365
6	[3728]	22699	30.431354
7	[3843]	22748	60.372188
8	[376]	21094	49.880471
9	[441]	21122	47.691399
10	[1013]	82580	35.634767
11	[3729]	22699	30.103908

In [ ]:

```
final_rules.sort_values('Lift', ascending=False, inplace=True)
```

In [ ]:

```
description_df = df_orig[['StockCode', 'Description']].drop_duplicates()
```

In [ ]:

```
with_desc = final_rules.merge(description_df, how = 'left', left_on = 'StockCode
s', right_on = 'StockCode')
with_desc.drop("StockCode", axis=1, inplace=True)
```

In [ ]:

```
with_desc.drop_duplicates(subset=["StockCodes", "Lift"], keep='first', inplace=T
rue)
with_desc
```

Out[ ]:

	Invoice	StockCodes	Lift	Description
0	[3839]	22745	60.372188	POPPY'S PLAYHOUSE BEDROOM
1	[3843]	22748	60.372188	POPPY'S PLAYHOUSE KITCHEN
2	[376]	21094	49.880471	SET/6 RED SPOTTY PAPER PLATES
3	[441]	21122	47.691399	SET/10 PINK SPOTTY PARTY CANDLES
5	[3728, 3727]	22697	38.851124	GREEN REGENCY TEACUP AND SAUCER
7	[3728]	22697	35.871365	GREEN REGENCY TEACUP AND SAUCER
9	[1013]	82580	35.634767	BATHROOM METAL SIGN
10	[3728, 3729]	22699	32.959223	ROSES REGENCY TEACUP AND SAUCER
12	[3729, 3310]	22699	32.529187	ROSES REGENCY TEACUP AND SAUCER
14	[3728]	22699	30.431354	ROSES REGENCY TEACUP AND SAUCER
16	[3729]	22699	30.103908	ROSES REGENCY TEACUP AND SAUCER
18	[855, 3012]	85099B	9.959836	JUMBO BAG RED WHITE SPOTTY

cs-119 Quiz7 q4 (LDA)

The first 8 cells of this notebook install PySpark. The quiz questions follow.

In [ ]:

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

In [ ]:

```
!wget -qN https://archive.apache.org/dist/spark/spark-3.2.1/spark-3.2.1-bin-hado
op3.2.tgz
!tar xf spark-3.2.1-bin-hadoop3.2.tgz
```

In [ ]:

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

In [ ]:

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

In [ ]:

```
!pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     |████████████████████████████████| 281.4 MB 22 kB/s
Collecting py4j==0.10.9.3
  Downloading py4j-0.10.9.3-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 56.9 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-an
y.whl size=281853642 sha256=4578e47a83a069405982103aba2414d39ab944d2
37ff3c7c4c1f3e50f5772131
  Stored in directory: /root/.cache/pip/wheels/9f/f5/07/7cd8017084dc
e4e93e84e92efd1e1d5334db05f2e83bcef74f
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.3 pyspark-3.2.1
```

In [ ]:

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[1]") \
                    .appName('Reviews LDA') \
                    .getOrCreate()

sc = spark.sparkContext

from pyspark.sql.types import *
```

In [ ]:

```
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[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 wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[ ]:

```
True
```

In [ ]:

```
import pandas as pd
import pyspark
from pyspark.sql import SQLContext
```

Q1: Read the reviews data from the source and scrub it

The data source is here (https://storage.googleapis.com/jsingh-bigdata-public/online-retail/reviews.csv).

Download it to a Pandas DataFrameClean the text in the Title and Review Text columns according to these rules:

1. Remove all punctuations,
2. Turn all words into lowercase,
3. Reject all 3-character or smaller words.

In [ ]:

```
df = pd.read_csv('https://storage.googleapis.com/119-quiz7-files/reviews.csv')
```

In [ ]:

```
df = df[['Title', 'Review Text']]
df.dropna(inplace=True)
```

In [ ]:

```
df['Title'] = df['Title'].str.lower()
df['Review Text'] = df['Review Text'].str.lower()
df
```

Out[ ]:

	Title	Review Text
2	some major design flaws	i had such high hopes for this dress and reall...
3	my favorite buy!	i love, love, love this jumpsuit. it's fun, fl...
4	flattering shirt	this shirt is very flattering to all due to th...
5	not for the very petite	i love tracy reese dresses, but this one is no...
6	cagrcoal shimmer fun	i aded this in my basket at hte last mintue to...
...	...	...
23481	great dress for many occasions	i was very happy to snag this dress at such a ...
23482	wish it was made of cotton	it reminds me of maternity clothes. soft, stre...
23483	cute, but see through	this fit well, but the top was very see throug...
23484	very cute dress, perfect for summer parties an...	i bought this dress for a wedding i have this ...
23485	please make more like this one!	this dress in a lovely platinum is feminine an...

19675 rows × 2 columns

In [ ]:

```
import re
df['Title'] = df['Title'].map(lambda x: re.sub(r'\b\w{1,3}\b', '', x))
df['Title'] = df['Title'].map(lambda x: re.sub(r'[^\w\s]', '', x))
df['Review Text'] = df['Review Text'].map(lambda x: re.sub(r'\b\w{1,3}\b', '', x
))
df['Review Text'] = df['Review Text'].map(lambda x: re.sub(r'[^\w\s]', '', x))
df
```

Out[ ]:

	Title	Review Text
2	some major design flaws	such high hopes this dress really wanted ...
3	favorite	love love love this jumpsuit flirty fabulo...
4	flattering shirt	this shirt very flattering adjustable fr...
5	very petite	love tracy reese dresses this very peti...
6	cagrcoal shimmer	aded this basket last mintue what woul...
...	...	...
23481	great dress many occasions	very happy snag this dress such great pri...
23482	wish made cotton	reminds maternity clothes soft stretchy shi...
23483	cute through	this well very through this never would ...
23484	very cute dress perfect summer parties	bought this dress wedding have this summer...
23485	please make more like this	this dress lovely platinum feminine fits p...

19675 rows × 2 columns

Each review is organized as a row in the PySpark DataFrame, and the objective is to do the same processing on each review in parallel!

Q2: Convert the Pandas DataFrame into a PySpark DataFrame

In [ ]:

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

In [ ]:

```
pysparkDF = spark.createDataFrame(df)
pysparkDF.show()
```

```
+--------------------+--------------------+
|               Title|         Review Text|
+--------------------+--------------------+
|some major design...|  such high hopes...|
|           favorite | love love love t...|
|    flattering shirt|this shirt  very ...|
|         very petite| love tracy reese...|
|   cagrcoal shimmer | aded this   bask...|
|shimmer surprisin...| ordered this  ca...|
|          flattering| love this dress ...|
|        such   dress|      ordered   p...|
|dress looks like ...|dress runs small ...|
|             perfect|more  more  find ...|
|               runs |bought  black    ...|
|pretty party dres...|this   nice choic...|
|       nice     body| took these    pa...|
| need    least av...|material  color  ...|
|looks great with ...|took  chance  thi...|
|    super cute  cozy| flattering super...|
|stylish  comfortable| love  look  feel...|
|    cute crisp shirt| this product   p...|
|                torn| upset because   ...|
|    what  looks like|first   this   pu...|
+--------------------+--------------------+
only showing top 20 rows
```

Q3: Tokenize Review Text

NLTK provides its own stop words. Using these has the advantage that we can use NLTK-provided stop words for a variety of supported languages.

This is an excellent place to further process the text. A tokenizer for the Review Text is provided for you here, you may use it as-is or modify it as you see fit.

```
def tokenize(pyspark_DataFrame):
    import re
    from nltk.corpus import stopwords
    reviews = pyspark_DataFrame.rdd.map(lambda x : x['Review Text'])   \
        .filter(lambda x: x is not None)
    StopWords = stopwords.words("english")
    tokens = reviews                                                   \
        .map( lambda doc: doc.strip().lower())                         \
        .map( lambda doc: re.split(" ", doc))                          \
        .map( lambda word: [x for x in word if x.isalpha()])           \
        .map( lambda word: [x for x in word if x not in StopWords])    \
        .zipWithIndex()
    return tokens
```

In [ ]:

```
def tokenize(pyspark_DataFrame):
    from nltk.corpus import stopwords
    reviews = pyspark_DataFrame.rdd.map(lambda x : x['Review Text'])    \
        .filter(lambda x: x is not None)
    StopWords = stopwords.words("english")
    tokens = reviews                                                     \
        .map( lambda doc: doc.strip().lower())                           \
        .map( lambda doc: re.split(" ", doc))                            \
        .map( lambda word: [x for x in word if x.isalpha()])             \
        .map( lambda word: [x for x in word if x not in StopWords])     \
        .zipWithIndex()
    return tokens
result_tokens = tokenize(pysparkDF)
tokenized_df = spark.createDataFrame(result_tokens)
tokenized_df.show()
```

```
+--------------------+---+
|                  _1| _2|
+--------------------+---+
|[high, hopes, dre...|  0|
|[love, love, love...|  1|
|[shirt, flatterin...|  2|
|[love, tracy, ree...|  3|
|[aded, basket, la...|  4|
|[ordered, carbon,...|  5|
|[love, dress, usu...|  6|
|[ordered, petite,...|  7|
|[dress, runs, sma...|  8|
|[find, reliant, r...|  9|
|[bought, black, l...| 10|
|[nice, choice, ho...| 11|
|[took, package, w...| 12|
|[material, color,...| 13|
|[took, chance, bl...| 14|
|[flattering, supe...| 15|
|[love, look, feel...| 16|
|[product, petite,...| 17|
|[upset, price, dr...| 18|
|[first, pullover,...| 19|
+--------------------+---+
only showing top 20 rows
```

Q4: TF.IDF Calculation

Feed the resulting tokens into a TF.IDF calculation. TF calculation is provided by `CountVectorizer` , and IDF calculation by `IDF` , both are available in the `pyspark.ml.feature` library.

```
def tfidf(sc, tokens):
    from pyspark.sql import SQLContext
    from pyspark.ml.feature import CountVectorizer , IDF
    sqlContext = SQLContext(sc)
    df_txts = sqlContext.createDataFrame(tokens, ["list_of_words",'inde
x'])
    #
    # TF
    #
    cv = CountVectorizer(inputCol="list_of_words", outputCol="raw_feature
s", \
        vocabSize=5000, minDF=10.0)
    cvmodel = cv.fit(df_txts)
    result_cv = cvmodel.transform(df_txts)
    #
    # IDF
    #
    idf = IDF(inputCol="raw_features", outputCol="features")
    idfModel = idf.fit(result_cv)
    tfidf_result = idfModel.transform(result_cv)
    #
    return tfidf_result
```

In [ ]:

```
def tfidf(sc, tokens):
    from pyspark.sql import SQLContext
    from pyspark.ml.feature import CountVectorizer , IDF
    sqlContext = SQLContext(sc)
    df_txts = sqlContext.createDataFrame(tokens, ["list_of_words",'index'])
    #
    # TF
    #
    cv = CountVectorizer(inputCol="list_of_words", outputCol="raw_features", \
        vocabSize=5000, minDF=10.0)
    cvmodel = cv.fit(df_txts)
    result_cv = cvmodel.transform(df_txts)
    #
    # IDF
    #
    idf = IDF(inputCol="raw_features", outputCol="features")
    idfModel = idf.fit(result_cv)
    tfidf_result = idfModel.transform(result_cv)
    #
    return tfidf_result, cvmodel

result_tfidf, cvmodel = tfidf(sc, result_tokens)
result_tfidf.show()
```

```
/usr/local/lib/python3.7/dist-packages/pyspark/sql/context.py:79: Fu
tureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCrea
te() instead.
  FutureWarning

+--------------------+-----+--------------------+-------------------
-+
|       list_of_words|index|        raw_features|            feature
s|
+--------------------+-----+--------------------+-------------------
-+
|[high, hopes, dre...|    0|(2810,[0,1,8,11,1...|(2810,[0,1,8,11,
1...|
|[love, love, love...|    1|(2810,[2,4,5,93,1...|(2810,[2,4,5,93,
1...|
|[shirt, flatterin...|    2|(2810,[2,4,14,15,...|(2810,[2,4,14,1
5,...|
|[love, tracy, ree...|    3|(2810,[0,2,4,6,8,...|(2810,[0,2,4,6,
8,...|
|[aded, basket, la...|    4|(2810,[1,3,6,9,10...|(2810,[1,3,6,9,1
0...|
|[ordered, carbon,...|    5|(2810,[1,9,11,12,...|(2810,[1,9,11,1
2,...|
|[love, dress, usu...|    6|(2810,[0,1,2,12,1...|(2810,[0,1,2,12,
1...|
|[ordered, petite,...|    7|(2810,[0,2,4,7,12...|(2810,[0,2,4,7,1
2...|
|[dress, runs, sma...|    8|(2810,[0,7,8,12,1...|(2810,[0,7,8,12,
1...|
|[find, reliant, r...|    9|(2810,[0,6,7,40,4...|(2810,[0,6,7,40,
4...|
|[bought, black, l...|   10|(2810,[0,6,21,42,...|(2810,[0,6,21,4
2,...|
|[nice, choice, ho...|   11|(2810,[0,1,3,7,8,...|(2810,[0,1,3,7,
8,...|
|[took, package, w...|   12|(2810,[6,7,8,9,11...|(2810,[6,7,8,9,1
1...|
|[material, color,...|   13|(2810,[1,9,10,15,...|(2810,[1,9,10,1
5,...|
|[took, chance, bl...|   14|(2810,[4,5,6,14,1...|(2810,[4,5,6,14,
1...|
|[flattering, supe...|   15|(2810,[5,8,10,15,...|(2810,[5,8,10,1
5,...|
|[love, look, feel...|   16|(2810,[0,2,8,10,1...|(2810,[0,2,8,10,
1...|
|[product, petite,...|   17|(2810,[6,13,18,19...|(2810,[6,13,18,1
9...|
|[upset, price, dr...|   18|(2810,[0,1,6,7,9,...|(2810,[0,1,6,7,
9,...|
|[first, pullover,...|   19|(2810,[3,23,25,64...|(2810,[3,23,25,6
4...|
+--------------------+-----+--------------------+-------------------
-+
only showing top 20 rows
```

Q5: LDA Training

The TF.IDF calculations form the input into LDA. An `lda_train()` function (shown below) takes columns `index` and `features` from the `tfidf` DataFrame and calculates the LDA Model. This calculation is referred to as *Training* the model.

```
def lda_train(result_tfidf):
    from pyspark.ml.linalg import Vectors, SparseVector
    from pyspark.ml.clustering import LDA
    #
    lda = LDA(k=10, seed=1, optimizer="em")
    lda.setMaxIter(100)
    #
    model = lda.fit(result_tfidf[['index', 'features']])
    return model
```

With the reviews dataset, LDA training takes about 15-20 minutes in a Colab environment.

In [ ]:

```
def lda_train(result_tfidf):
    from pyspark.ml.linalg import Vectors, SparseVector
    from pyspark.ml.clustering import LDA
    #
    lda = LDA(k=10, seed=1, optimizer="em")
    lda.setMaxIter(100)
    #
    model = lda.fit(result_tfidf[['index', 'features']])
    return model
```

This will take about 15 minutes

In [ ]:

```
model = lda_train(result_tfidf)
```

Q6: Reporting on the LDA Model

Examine the model generated during training. It will show the 10 topics it generated.

What is a topic? Just a list of words that "hang together." Does the collection of words describe a topic to you? What might it be?

In [ ]:

```
from pyspark.sql.functions import udf
from pyspark.sql.types import *

vocab = cvmodel.vocabulary
vocab_broadcast = sc.broadcast(vocab)

# getting the topics
ldatopics = model.describeTopics()

def map_termID_to_Word(termIndices):
    words = []
    for termID in termIndices:
        words.append(vocab_broadcast.value[termID])

    return words

udf_map_termID_to_Word = udf(map_termID_to_Word , ArrayType(StringType()))

ldatopics_mapped = ldatopics.withColumn("topic_desc", udf_map_termID_to_Word(lda
topics.termIndices))
```

```
/usr/local/lib/python3.7/dist-packages/pyspark/sql/context.py:127: F
utureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCre
ate() instead.
  FutureWarning
```

In [ ]:

```
ldatopics_mapped.show(truncate=False)
```

```
+-----+-------------------------------------------+-----------------
--------------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------------------------------------+-
--------------------------------------------------------------------
--------+
|topic|termIndices                                |termWeights
|topic_desc
|
+-----+-------------------------------------------+-----------------
--------------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------------------------------------+-
--------------------------------------------------------------------
--------+
|0    |[27, 52, 61, 18, 74, 3, 11, 65, 34, 101]   |[0.03042054616311
599, 0.024453990429804267, 0.02349893059982536, 0.01843815676258837,
0.017198123956741643, 0.017000280905539334, 0.016893526126113223, 0.
016253204733195335, 0.016112173611003065, 0.014471928161295004]  |[s
hirt, short, sleeves, back, front, like, really, design, long, wante
d]      |
|1    |[35, 51, 9, 55, 16, 124, 5, 2, 155, 158]   |[0.03150864976349
9555, 0.02829238909442366, 0.023035718606536283, 0.02188818262400200
5, 0.0193743215893472, 0.01717297032831965, 0.01689959079753611, 0.0
14836427715101453, 0.0143160639489339, 0.014295519160750555]       |
[jeans, pants, color, black, soft, pair, great, love, leggings, gree
n]      |
|2    |[76, 41, 82, 98, 88, 109, 152, 40, 178, 7] |[0.02023208971832
5134, 0.019553891023826885, 0.018999543063626733, 0.0177263616879779
24, 0.015999179925325898, 0.015292968087719648, 0.01337024515018442,
0.012850985873878012, 0.012588307885714795, 0.011653352489537845]|[l
ooked, quality, price, thought, going, thin, return, even, pockets,
fabric]|
|3    |[32, 87, 92, 0, 95, 94, 105, 119, 125, 112]|[0.03135922568449
286, 0.01938721877619622, 0.01756248113705193, 0.017276609694653663,
0.01705183647017351, 0.016811864310078656, 0.0166935676198645, 0.015
500481097959054, 0.01489748271353299, 0.014890243096076736]      |[s
weater, jacket, worn, dress, fall, many, piece, compliments, wore, c
asual] |
|4    |[8, 1, 25, 38, 62, 47, 104, 111, 130, 132] |[0.04154948263628
003, 0.0348197783218852, 0.03188758125113466, 0.02868397715309568,
0.02213728104475774, 0.01948460463410236, 0.017688269074351125, 0.01
667256347764485, 0.015655015153166543, 0.014758362343166917]       |
[small, size, large, medium, runs, usually, arms, shoulders, print,
usual]   |
|5    |[43, 46, 78, 60, 93, 117, 136, 45, 84, 166]|[0.02802033772318
4612, 0.025423140602040185, 0.020310981882225305, 0.0178786756293051
4, 0.016895699053582645, 0.01657095123881948, 0.015033716395905724,
0.014342795845725455, 0.01400045264366639, 0.01307852196438552]   |
[retailer, store, sale, online, time, went, dresses, tried, first, h
appy]    |
|6    |[0, 36, 71, 91, 100, 113, 85, 56, 86, 134] |[0.03532479675144
0686, 0.028757006850649157, 0.01966001936449338, 0.01766375888774601
4, 0.017129358171881364, 0.016392808015768208, 0.016109985529091438,
0.015227770211346895, 0.015064640180922649, 0.014789467506719573]|[d
ress, waist, tight, around, chest, bust, body, right, bottom, hips]
|
|7    |[20, 4, 5, 24, 66, 58, 72, 70, 22, 57]     |[0.02869634570945
0014, 0.024835344618907237, 0.02386861890495785, 0.02372245371456328
5, 0.022651229207385465, 0.021899260146196073, 0.02155878117921527,
0.01936510579210543, 0.018890438141805903, 0.017794585264617212]  |
```

```
[comfortable, wear, great, fits, white, super, summer, enough, cute,
true]   |
|8    |[48, 23, 42, 73, 79, 120, 126, 139, 3, 167]|[0.029317893008145773, 0.02571765335856018, 0.02295357842628883, 0.020245315975324626, 0.019756376876765677, 0.017692182725159513, 0.016312493780100454,
0.01512071189631979, 0.014367059350229468, 0.013150712414909307]  |
[skirt, looks, pretty, model, person, blouse, side, picture, like, p
hoto]    |
|9    |[33, 12, 90, 80, 129, 83, 21, 54, 45, 9]   |[0.03393670054518
546, 0.021023655645263442, 0.02029476352903465, 0.01908603493822198,
0.01669554897070685, 0.015599564253719438, 0.015365130930761717, 0.0
1492982694700259, 0.013617849720250248, 0.012404651345061142]    |[p
etite, ordered, regular, loved, wish, blue, bought, still, tried, co
lor]   |
+-----+-------------------------------------------+-----------------
--------------------------------------------------------------------
--------------------------------------------------------------------
-----------------------------------------------------------------+-
--------------------------------------------------------------------
--------+
```

The overarching topic is about clothing, but the individual topic differs from each other. With the model having 10 topics, some of the topics are not visibly distinct from each other, there are a lot of overlapping words/ideas between them. However, there are some topics that really stood out, such as topic 5, which appears to talk about the means of shopping, or topic 4, which talks about sizes. Topics 0, 1, 3, 8 seem to talk about different items of clothing but they are not entirely distinct to me.