Homework – 3

Fall 2024

Spark Streaming

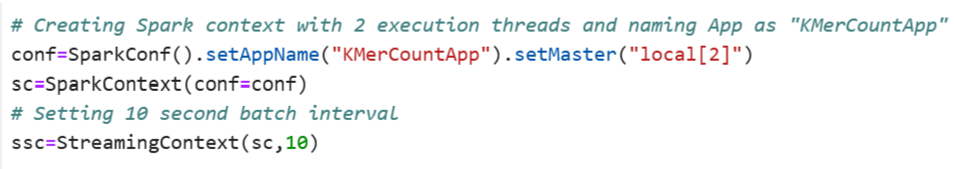
Sahithya Arveti Nagaraju (Person number: 50559752, UBITName: sarvetin)

Implement k-mer count program in Spark Streaming

• Create a local StreamingContext with two execution threads and a batch interval of 10 seconds.

Answer:

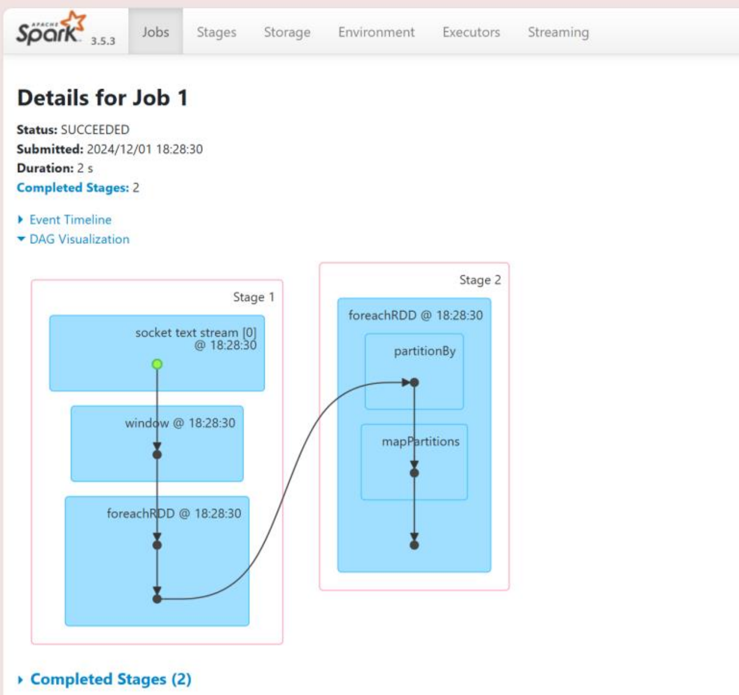
Below is the code for creating StreamingContext with two execution threads and batch interval of 10 seconds:



```
# Creating Spark context with 2 execution threads and naming App as "KMerCountApp"
conf=SparkConf().setAppName("KMerCountApp").setMaster("local[2]")
sc=SparkContext(conf=conf)
# Setting 10 second batch interval
ssc=StreamingContext(sc,10)
```

- SparkConf().setAppName("KMerCountApp") sets the name of the Spark application as 'KMerCountApp'.
- .setMaster("local[2]") sets the Spark context to run locally with 2 execution threads.
- StreamingContext(sc,10) initializes a StreamingContext with a batch interval of 10 seconds, which indicates that the system will handle and process the incoming data in chunks of 10 seconds.

DAG Visualization:



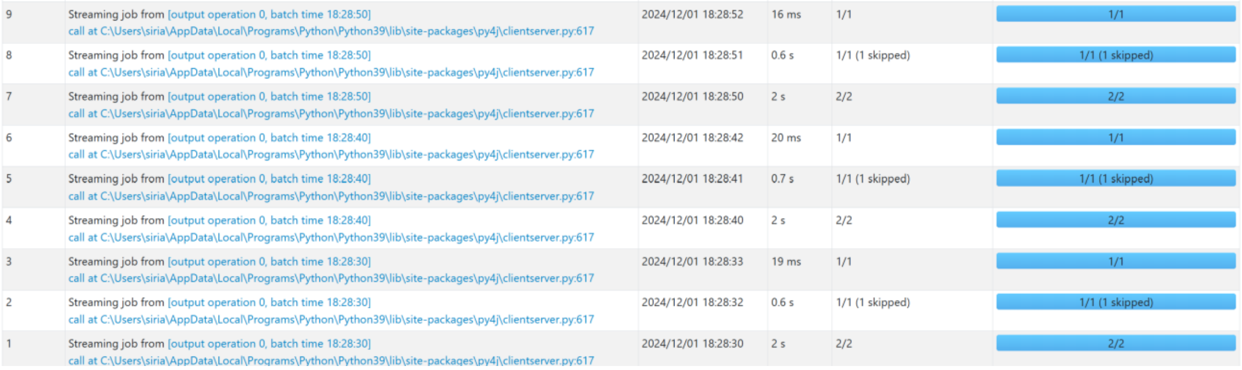
Details for Job 1

Status: SUCCEEDED
Submitted: 2024/12/01 18:28:30
Duration: 2 s
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization

▸ **Completed Stages (2)**

The above DAG shows a Spark Streaming job that continuously reads text data from a socket stream, processes it in batches, and writes the results to a destination. The windowing operation allows for real-time processing of the data stream, and the partitioning and mapping operations enable efficient parallel processing.



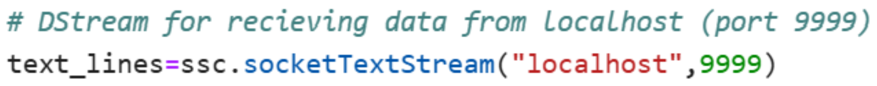
9	Streaming job from [output operation 0, batch time 18:28:50] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:52	16 ms	1/1	1/1
8	Streaming job from [output operation 0, batch time 18:28:50] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:51	0.6 s	1/1 (1 skipped)	1/1 (1 skipped)
7	Streaming job from [output operation 0, batch time 18:28:50] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:50	2 s	2/2	2/2
6	Streaming job from [output operation 0, batch time 18:28:40] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:42	20 ms	1/1	1/1
5	Streaming job from [output operation 0, batch time 18:28:40] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:41	0.7 s	1/1 (1 skipped)	1/1 (1 skipped)
4	Streaming job from [output operation 0, batch time 18:28:40] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:40	2 s	2/2	2/2
3	Streaming job from [output operation 0, batch time 18:28:30] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:33	19 ms	1/1	1/1
2	Streaming job from [output operation 0, batch time 18:28:30] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:32	0.6 s	1/1 (1 skipped)	1/1 (1 skipped)
1	Streaming job from [output operation 0, batch time 18:28:30] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617	2024/12/01 18:28:30	2 s	2/2	2/2

Above Spark UI shows that the each batch is processed every 10 seconds.

- **Create a DStream that represents streaming data from a TCP source (localhost:9999) [hint: use Netcat]**

Answer:

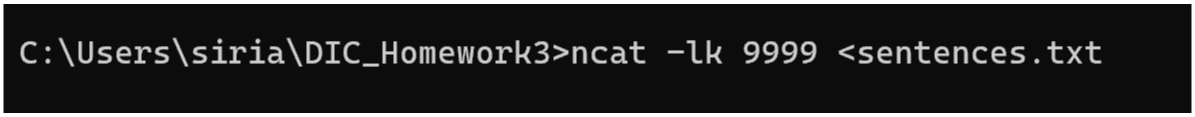
The DStream is created by connecting to a TCP source on localhost on port 9999. This is done by the following line in the code:



```
# DStream for recieving data from localhost (port 9999)
text_lines=ssc.socketTextStream("localhost",9999)
```

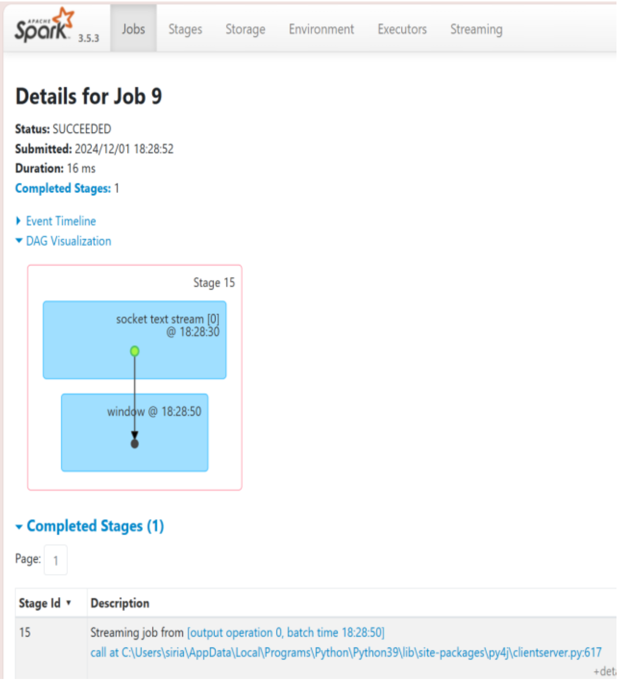
ssc.socketTextStream("localhost", 9999) sets up a DStream that listens for text data from a TCP socket on localhost at port 9999. The data from sentences.txt is streamed into this port using Netcat.

Netcat command for data streaming of sentences.txt into port 9999:



```
C:\Users\siria\DIC_Homework3>ncat -lk 9999 <sentences.txt
```

DAG Visualization:



Details for Job 9

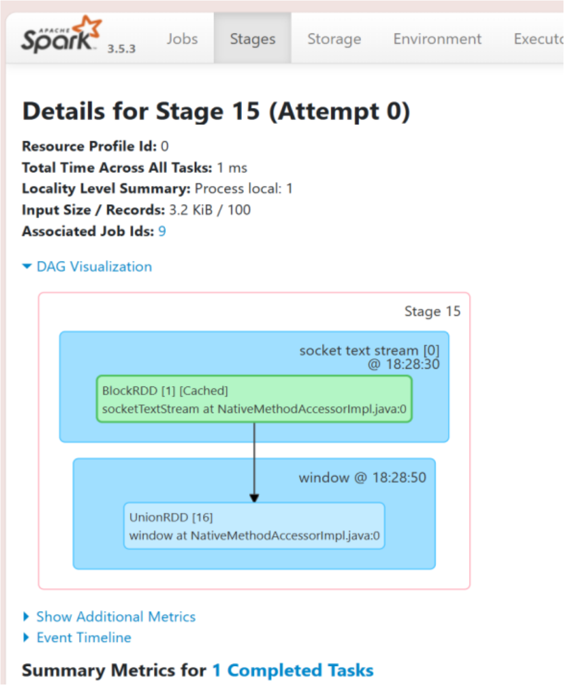
Status: SUCCEEDED
Submitted: 2024/12/01 18:28:52
Duration: 16 ms
Completed Stages: 1

▸ Event Timeline
▾ DAG Visualization

▾ Completed Stages (1)

Page: 1

Stage Id ▾	Description
15	Streaming job from [output operation 0, batch time 18:28:50] call at C:\Users\siria\AppData\Local\Programs\Python\Python39\lib\site-packages\py4j\clientserver.py:617 +det



Spark 3.5.3
Jobs
Stages
Storage
Environment
Executo

Details for Stage 15 (Attempt 0)

Resource Profile Id: 0
Total Time Across All Tasks: 1 ms
Locality Level Summary: Process local: 1
Input Size / Records: 3.2 KiB / 100
Associated Job Ids: 9

▾ DAG Visualization

Stage 15
socket text stream [0] @ 18:28:30
BlockRDD [1] [Cached]
socketTextStream at NativeMethodAccessorImpl.java:0
window @ 18:28:50
UnionRDD [16]
window at NativeMethodAccessorImpl.java:0

▸ Show Additional Metrics
▸ Event Timeline

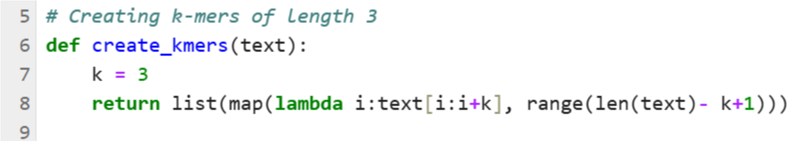
Summary Metrics for 1 Completed Tasks

Above DAG Visualization shows that a socket text stream (Dstream) created to stream data from TCP source (localhost:9999) with a window of 30 seconds with a sliding interval of 10 seconds.

- **Generate k-mers of length 3 from each line of text.**

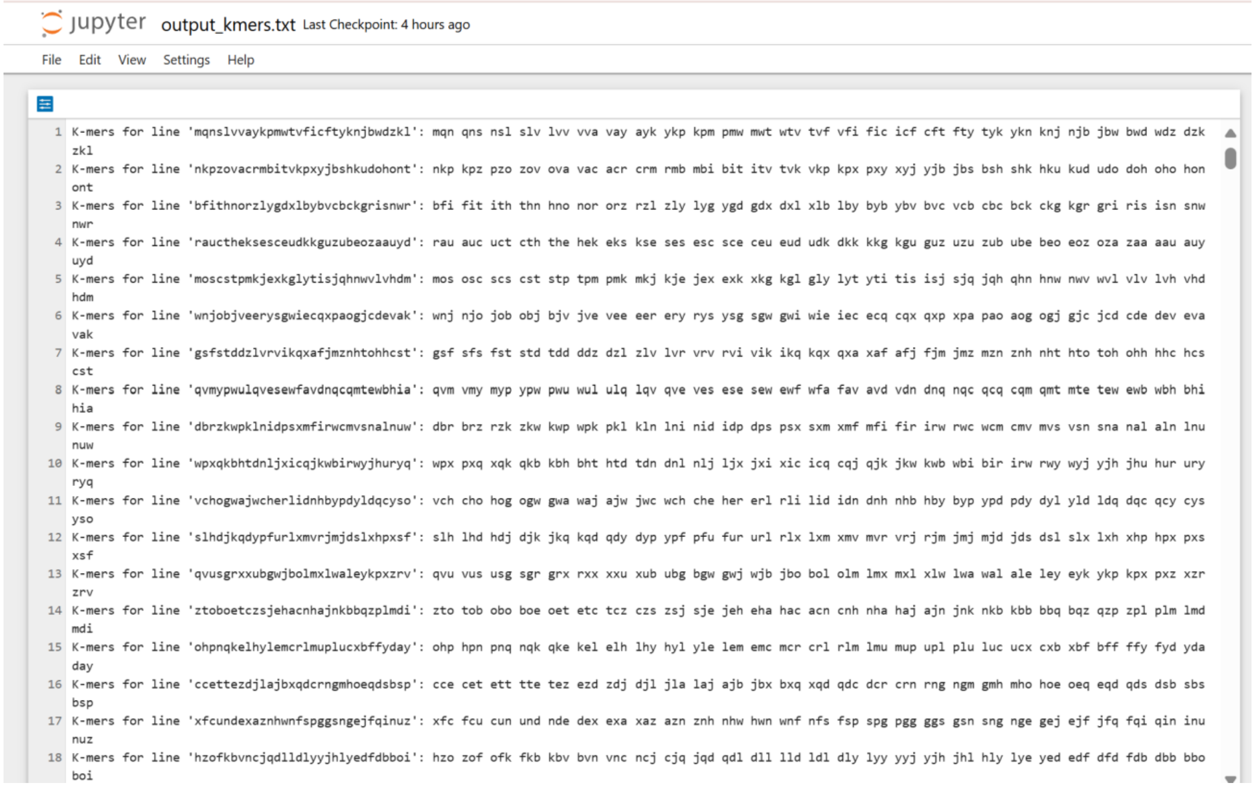
Answer:

The create_kmers function in the code is responsible for generating k-mers of length 3 from each line of text:



```python
# Creating k-mers of length 3
def create_kmers(text):
    k = 3
    return list(map(lambda i:text[i:i+k], range(len(text)- k+1)))

```

The function create_kmers takes a string text as input and generates a list of substrings (k-mers) of length 3. It uses the map function to iterate through all possible starting indices i of length-3 substrings within the text. The range range(len(text) - k + 1) ensures that the function doesn't attempt to create a k-mer starting too close to the end of the string. It generates a list of all the k-mers of length 3 which are possible in the text provided as shown in below screenshot:



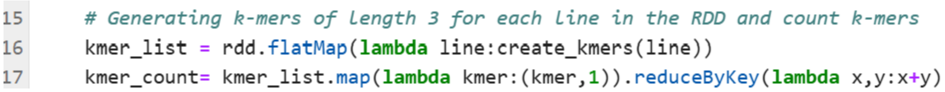
```
K-mers for line 'mqnslvvaykpmwtvficftyknjbwdzkl': mqn qns nsl slv lvv vva vay ayk ykp kpm pmw mwt wtv tvf vfi fic icf cft fty tyk ykn knj njb jbw bwd wdz dzk
zkl
K-mers for line 'nkpzovacrmbitvkpxyjbshkudohont': nkp kpz pzo zov ova vac acr crm rmb mbi bit itv tvk vkp kpx pxy xyj yjb jbs bsh shk hku kud udo doh oho hon
ont
K-mers for line 'bfithnorzlygdxlbybvcbckgrisnwr': bfi fit ith thn hno nor orz rzl zly lyg ygd gdx dxl xlb lby byb ybv bvc vcb cbc bck ckg kgr gri ris isn snw
nwr
K-mers for line 'raucthekseseudkkguzubeozaauyd': rau auc uct cth the hek eks kse ses esc sce ceu eud udk dkk kkg kgu guz uzu zub ube beo eoz oza zaa aau auy
uyd
K-mers for line 'moscstpmkjexkglytisjqhnwvlvhdm': mos osc scs cst stp tpm pmk mkj kje jex exk xkg kgl gly lyt yti tis isj sjq jqh qhn hnw nwv wvl vlv lvh vhd
hdm
K-mers for line 'wnjobjveerysgwiecqxpaogjcdevak': wnj njo job obj bjv jve vee eer ery rys ysg sgw gwi wie iec ecq cqx qxp xpa pao aog ogj gjc jcd cde dev eva
vak
K-mers for line 'gsfstddzlvrvikqxafjmznhtohhcst': gsf sfs fst std tdd ddz dzl zlv lvr vrv rvi vik ikq kqx qxa xaf afj fjm jmz mzn znh nht hto toh ohh hhc hcs
cst
K-mers for line 'qvmypwulqvesewfavdnqcqmtewbhia': qvm vmy myp ypw pwu wul ulq lqv qve ves ese sew ewf wfa fav avd vdn dnq nqc qcq cqm qmt mte tew ewb wbh bhi
hia
K-mers for line 'dbrzkwpklnidpsxmfirwcmvsnalnuw': dbr brz rzk zkw kwp wpk pkl kln lni nid idp dps psx sxm xmf mfi fir irw rwc wcm cmv mvs vsn sna nal aln lnu
nuw
K-mers for line 'wpxqkbhtdnljxicqjkwbirwyjhuryq': wpx pxq xqk qkb kbh bht htd tdn dnl nlj ljx jxi xic icq cqj qjk jkw kwb wbi bir irw rwy wyj yjh jhu hur ury
ryq
K-mers for line 'vchogwajwcherlidnhbypdyldqcyso': vch cho hog ogw gwa waj ajw jwc wch che her erl rli lid idn dnh nhb hby byp ypd pdy dyl yld ldq dqc qcy cys
yso
K-mers for line 'slhdjkqdypfurlxmvrjmjdslxhpxsf': slh lhd hdj djk jkq kqd qdy dyp ypf pfu fur url rlx lxm xmv mvr vrj rjm jmj mjd jds dsl slx lxh xhp hpx pxs
xsf
K-mers for line 'qvusgrxxubgwjbolmxlwaleykpxzrv': qvu vus usg sgr grx rxx xxu xub ubg bgw gwj wjb jbo bol olm lmx mxl xlw lwa wal ale ley eyk ykp kpx pxz xzr
zrv
K-mers for line 'ztoboetczsjehacnhajnkbbqzplmdi': zto tob obo boe oet etc tcz czs zsj sje jeh eha hac acn cnh nha haj ajn jnk nkb kbb bbq bqz qzp zpl plm lmd
mdi
K-mers for line 'ohpnqkelhylemcrlmuplucxbffyday': ohp hpn pnq nqk qke kel elh lhy hyl yle lem emc mcr crl rlm lmu mup upl plu luc ucx cxb xbf bff ffy fyd yda
day
K-mers for line 'ccettezdjlajbxqdcrngmhoeqdsbsp': cce cet ett tte tez ezd zdj djl jla laj ajb jbx bxq xqd qdc dcr crn rng ngm gmh mho hoe oeq eqd qds dsb sbs
bsp
K-mers for line 'xfcundexaznhwnfspggsngejfqinuz': xfc fcu cun und nde dex exa xaz azn znh nhw hwn wnf nfs fsp spg pgg ggs gsn sng nge gej ejf jfq fqi qin inu
nuz
K-mers for line 'hzofkbvncjqdlldlyyjhlyedfdbboi': hzo zof ofk fkb kbv bvn vnc ncj cjq jqd qdl dll lld ldl dly lyy yyj yjh jhl hly lye yed edf dfd fdb dbb bbo
boi
```

Also, saved all the k-mers for each line in a text file named output_kmers.txt for reference.

- **Count the occurrences of each k-mer and print the k-mer counts. Take screenshots of your output.**

Answer:

The k-mer frequencies are counted by using the reduceByKey operation in the process_kmers function:

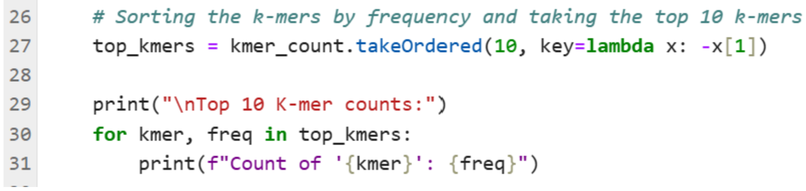


```python
    # Generating k-mers of length 3 for each line in the RDD and count k-mers
    kmer_list = rdd.flatMap(lambda line:create_kmers(line))
    kmer_count= kmer_list.map(lambda kmer:(kmer,1)).reduceByKey(lambda x,y:x+y)
```

- rdd.flatMap(lambda line: create_kmers(line)): Transforms every line of the RDD to a list of k-mers. The flatMap function will flatten all the lists of k-mers into one single RDD.

- reduceByKey(lambda x, y: x + y): reduces the RDD by key - i.e. k-mer - and sums up values, so counts a frequency of every k-mer.

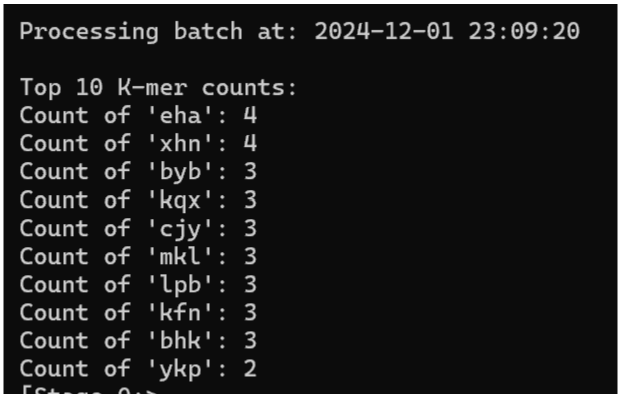
The first top 10 k-mer counts are printed using the following code:



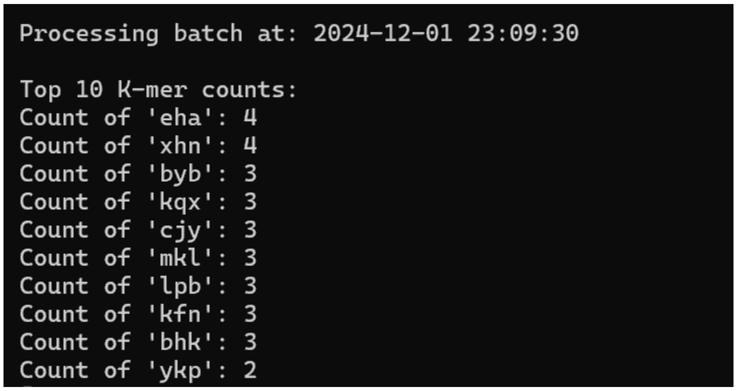
```python
    # Sorting the k-mers by frequency and taking the top 10 k-mers
    top_kmers = kmer_count.takeOrdered(10, key=lambda x: -x[1])

    print("\nTop 10 K-mer counts:")
    for kmer, freq in top_kmers:
        print(f"Count of '{kmer}': {freq}")
```

Explanation:

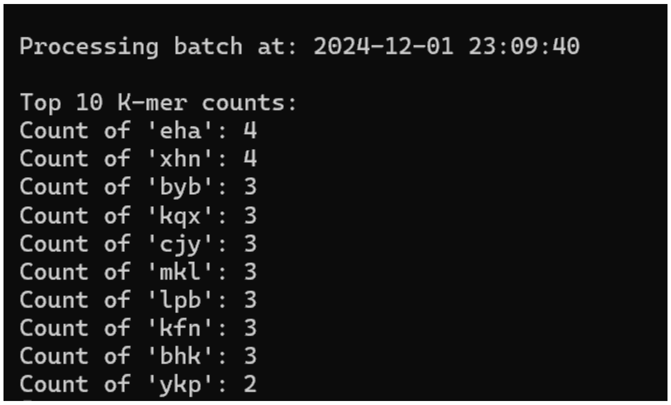
The kmer_count.collect() method collects all the k-mer counts as tuples of the form (kmer, frequency). A for loop iterates over these tuples, printing the first 10 k-mers and their frequencies.



```
Processing batch at: 2024-12-01 23:09:20

Top 10 K-mer counts:
Count of 'eha': 4
Count of 'xhn': 4
Count of 'byb': 3
Count of 'kqx': 3
Count of 'cjy': 3
Count of 'mkl': 3
Count of 'lpb': 3
Count of 'kfn': 3
Count of 'bhk': 3
Count of 'ykp': 2
```



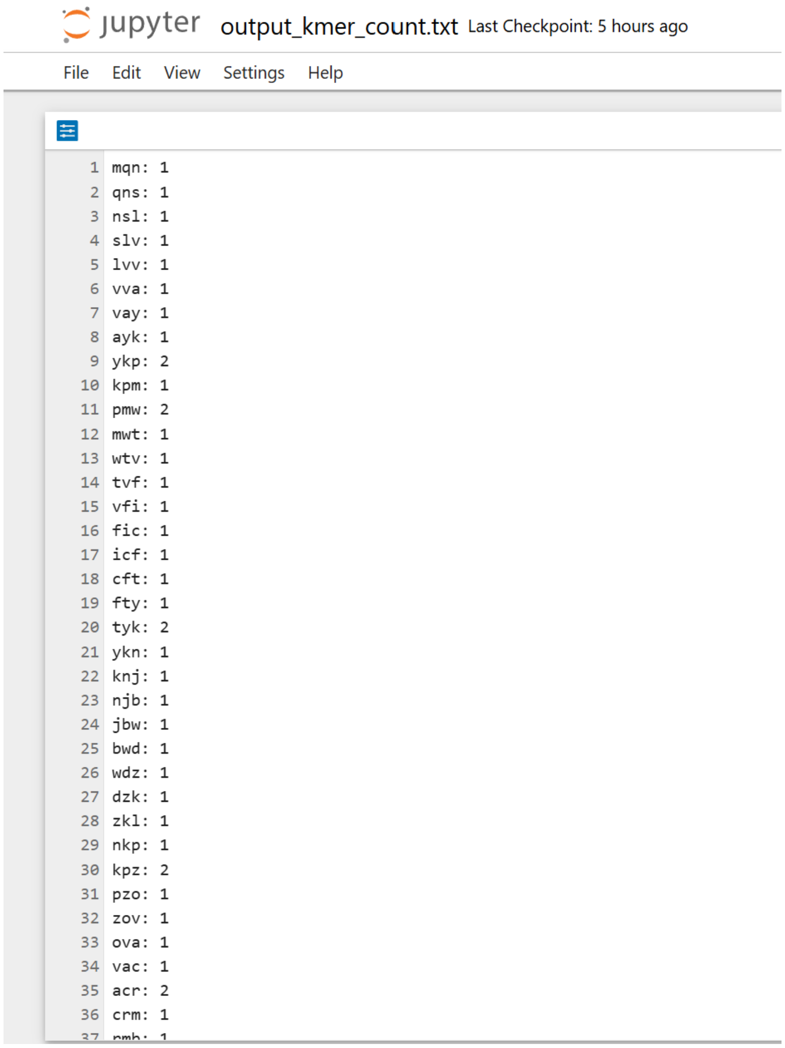
```
Processing batch at: 2024-12-01 23:09:30

Top 10 K-mer counts:
Count of 'eha': 4
Count of 'xhn': 4
Count of 'byb': 3
Count of 'kqx': 3
Count of 'cjy': 3
Count of 'mkl': 3
Count of 'lpb': 3
Count of 'kfn': 3
Count of 'bhk': 3
Count of 'ykp': 2
```



```
Processing batch at: 2024-12-01 23:09:40

Top 10 K-mer counts:
Count of 'eha': 4
Count of 'xhn': 4
Count of 'byb': 3
Count of 'kqx': 3
Count of 'cjy': 3
Count of 'mkl': 3
Count of 'lpb': 3
Count of 'kfn': 3
Count of 'bhk': 3
Count of 'ykp': 2
```

The above is the output of kmer counts, here I'm only displaying only top 10 kmer counts since there are a lot of kmers. Additionally, it can be observed that Dstream is performing kmer counting for every 10 seconds interval.

Also, saved all the k-mers count in a text file named output_kmer_count.txt for reference.

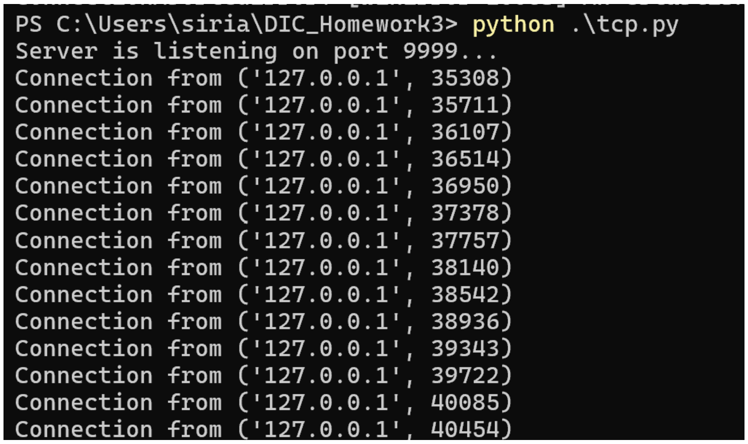


```
mqn: 1
qns: 1
nsl: 1
slv: 1
lvv: 1
vva: 1
vay: 1
ayk: 1
ykp: 2
kpm: 1
pmw: 2
mwt: 1
wtv: 1
tvf: 1
vfi: 1
fic: 1
icf: 1
cft: 1
fty: 1
tyk: 2
ykn: 1
knj: 1
njb: 1
jbw: 1
bwd: 1
wdz: 1
dzk: 1
zkl: 1
nkp: 1
kpz: 2
pzo: 1
zov: 1
ova: 1
vac: 1
acr: 2
crm: 1
rmb: 1
```

Alternative method:

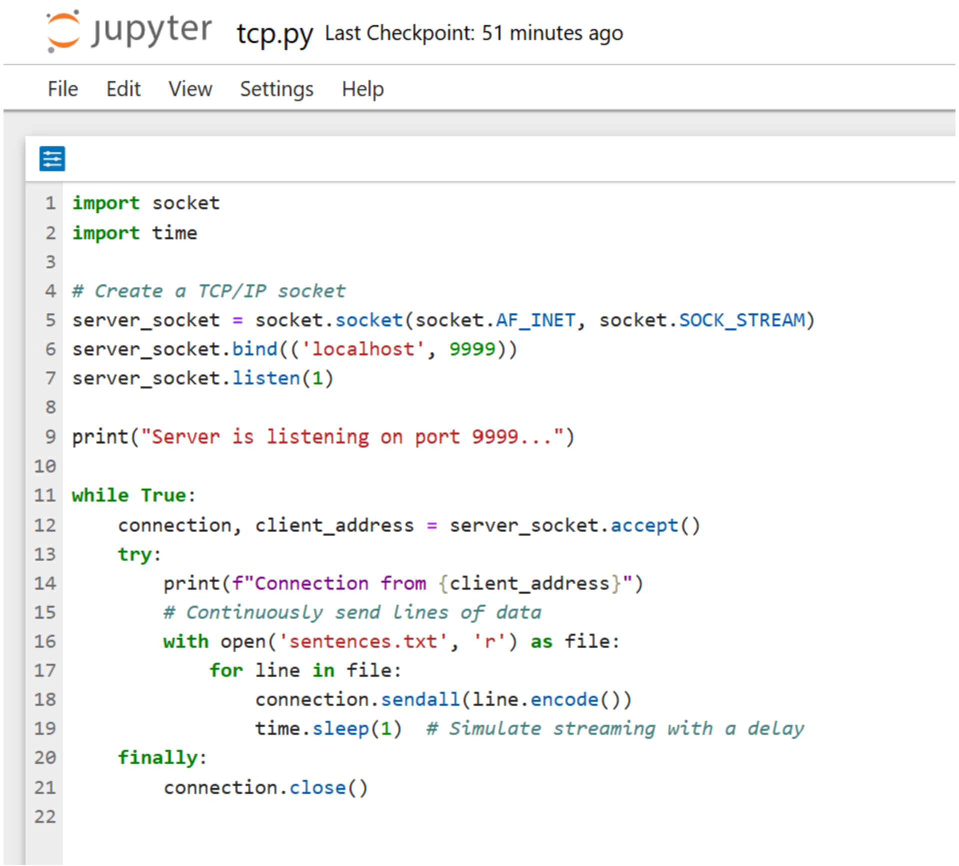
The above method using netcat was getting whole data from sentences.txt at first time interval itself, so it was performing kmer operation at first time interval and was not getting any data streamed after that from netcat.

For alternative method, first run tcp.py file in one terminal like below screenshot and then run code.py in another terminal



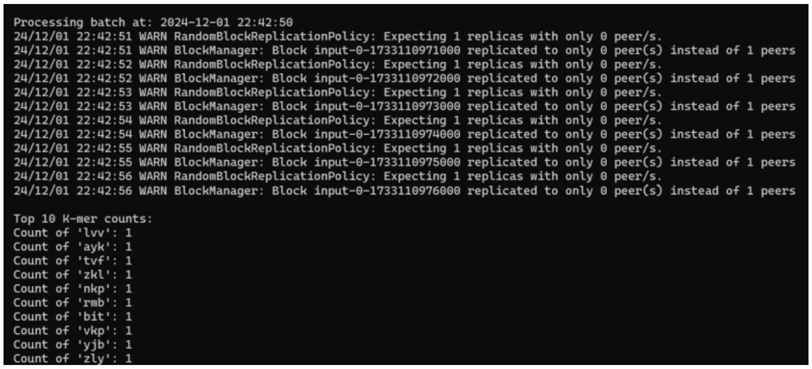
```
PS C:\Users\siria\DIC_Homework3> python .\tcp.py
Server is listening on port 9999...
Connection from ('127.0.0.1', 35308)
Connection from ('127.0.0.1', 35711)
Connection from ('127.0.0.1', 36107)
Connection from ('127.0.0.1', 36514)
Connection from ('127.0.0.1', 36950)
Connection from ('127.0.0.1', 37378)
Connection from ('127.0.0.1', 37757)
Connection from ('127.0.0.1', 38140)
Connection from ('127.0.0.1', 38542)
Connection from ('127.0.0.1', 38936)
Connection from ('127.0.0.1', 39343)
Connection from ('127.0.0.1', 39722)
Connection from ('127.0.0.1', 40085)
Connection from ('127.0.0.1', 40454)
```

So, I have used an alternative method for sending data using TCP socket program as shown below to stream data in localhost 9999,

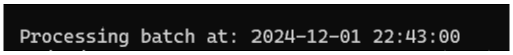


```python
import socket
import time

# Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 9999))
server_socket.listen(1)

print("Server is listening on port 9999...")

while True:
    connection, client_address = server_socket.accept()
    try:
        print(f"Connection from {client_address}")
        # Continuously send lines of data
        with open('sentences.txt', 'r') as file:
            for line in file:
                connection.sendall(line.encode())
                time.sleep(1)  # Simulate streaming with a delay
    finally:
        connection.close()

```

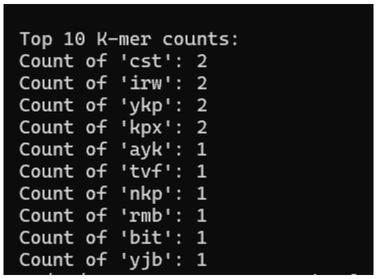
By using above method, data was sending continuously and able to get the kmer counts in batchs. Here are the few screenshots of kmer counts after using this method,



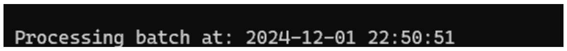
```
Processing batch at: 2024-12-01 22:42:50
24/12/01 22:42:51 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
24/12/01 22:42:51 WARN BlockManager: Block input-0-1733110971000 replicated to only 0 peer(s) instead of 1 peers
24/12/01 22:42:52 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
24/12/01 22:42:52 WARN BlockManager: Block input-0-1733110972000 replicated to only 0 peer(s) instead of 1 peers
24/12/01 22:42:53 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
24/12/01 22:42:53 WARN BlockManager: Block input-0-1733110973000 replicated to only 0 peer(s) instead of 1 peers
24/12/01 22:42:54 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
24/12/01 22:42:54 WARN BlockManager: Block input-0-1733110974000 replicated to only 0 peer(s) instead of 1 peers
24/12/01 22:42:55 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
24/12/01 22:42:55 WARN BlockManager: Block input-0-1733110975000 replicated to only 0 peer(s) instead of 1 peers
24/12/01 22:42:56 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
24/12/01 22:42:56 WARN BlockManager: Block input-0-1733110976000 replicated to only 0 peer(s) instead of 1 peers

Top 10 K-mer counts:
Count of 'lvv': 1
Count of 'ayk': 1
Count of 'tvf': 1
Count of 'zkl': 1
Count of 'nkp': 1
Count of 'rmb': 1
Count of 'bit': 1
Count of 'vkp': 1
Count of 'yjb': 1
Count of 'zly': 1
```



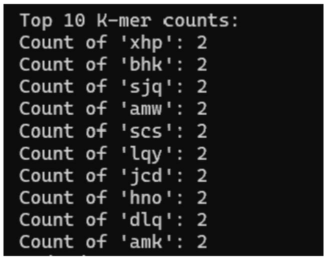
```
Processing batch at: 2024-12-01 22:43:00
```



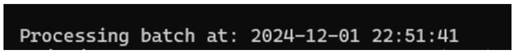
```
Top 10 K-mer counts:
Count of 'cst': 2
Count of 'irw': 2
Count of 'ykp': 2
Count of 'kpx': 2
Count of 'ayk': 1
Count of 'tvf': 1
Count of 'nkp': 1
Count of 'rmb': 1
Count of 'bit': 1
Count of 'yjb': 1
```



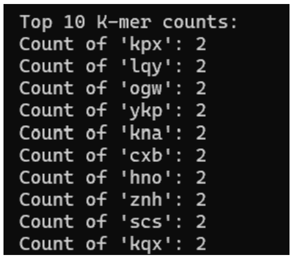
```
Processing batch at: 2024-12-01 22:50:51
```



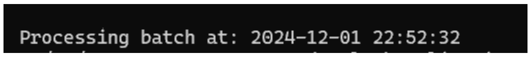
```
Top 10 K-mer counts:
Count of 'xhp': 2
Count of 'bhk': 2
Count of 'sjq': 2
Count of 'amw': 2
Count of 'scs': 2
Count of 'lqy': 2
Count of 'jcd': 2
Count of 'hno': 2
Count of 'dlq': 2
Count of 'amk': 2
```



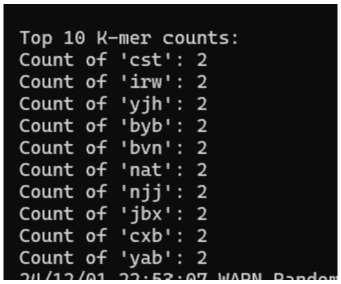
```
Processing batch at: 2024-12-01 22:51:41
```



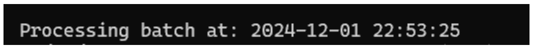
```
Top 10 K-mer counts:
Count of 'kpx': 2
Count of 'lqy': 2
Count of 'ogw': 2
Count of 'ykp': 2
Count of 'kna': 2
Count of 'cxb': 2
Count of 'hno': 2
Count of 'znh': 2
Count of 'scs': 2
Count of 'kqx': 2
```



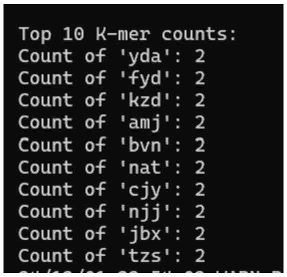
```
Processing batch at: 2024-12-01 22:52:32
```



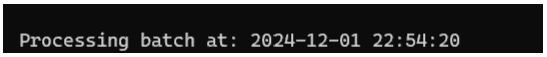
```
Top 10 K-mer counts:
Count of 'cst': 2
Count of 'irw': 2
Count of 'yjh': 2
Count of 'byb': 2
Count of 'bvn': 2
Count of 'nat': 2
Count of 'njj': 2
Count of 'jbx': 2
Count of 'cxb': 2
Count of 'yab': 2
24/12/01 22:53:07 WARN Random
```



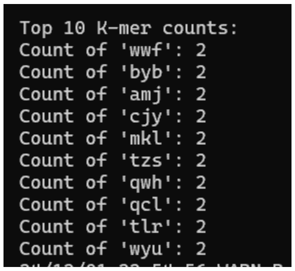
```
Processing batch at: 2024-12-01 22:53:25
```



```
Top 10 K-mer counts:
Count of 'yda': 2
Count of 'fyd': 2
Count of 'kzd': 2
Count of 'amj': 2
Count of 'bvn': 2
Count of 'nat': 2
Count of 'cjy': 2
Count of 'njj': 2
Count of 'jbx': 2
Count of 'tzs': 2
24/12/01 22:54:02 WARN R
```



```
Processing batch at: 2024-12-01 22:54:20
```



```
Top 10 K-mer counts:
Count of 'wwf': 2
Count of 'byb': 2
Count of 'amj': 2
Count of 'cjy': 2
Count of 'mkl': 2
Count of 'tzs': 2
Count of 'qwh': 2
Count of 'qcl': 2
Count of 'tlr': 2
Count of 'wyu': 2
24/12/01 22:54:56 WARN R
```