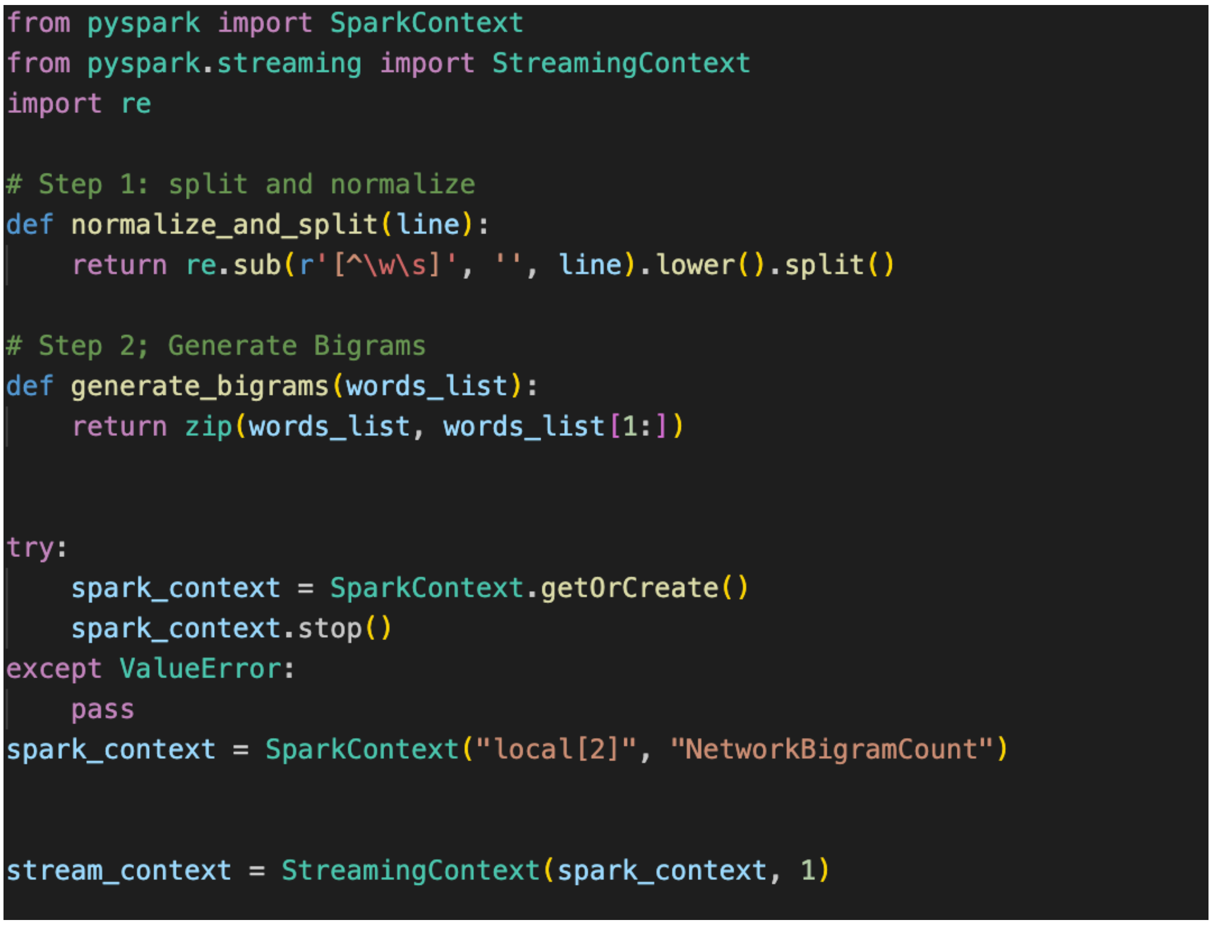
Assignment 4- Spark Streaming and Bias Data

Lolla Aditya Srivatsav- 50559685
Sai Krishna Goud Valdas- 50560726

Part -1 Implement word co-occurrence program.

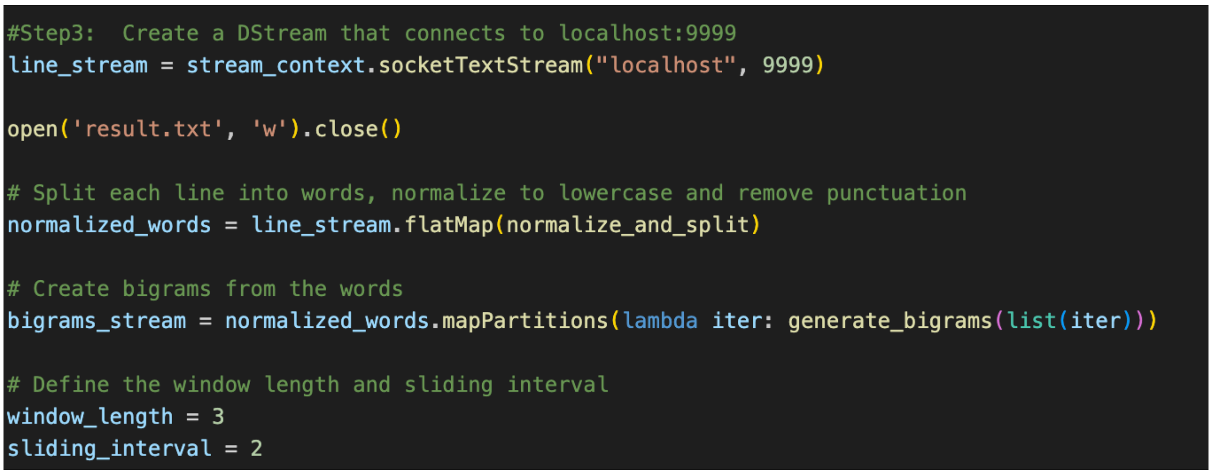
Step 1: Create a local StreamingContext with two execution threads and batch interval of 1 second.



```python
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
import re

# Step 1: split and normalize
def normalize_and_split(line):
    return re.sub(r'[^\w\s]', '', line).lower().split()

# Step 2; Generate Bigrams
def generate_bigrams(words_list):
    return zip(words_list, words_list[1:])


try:
    spark_context = SparkContext.getOrCreate()
    spark_context.stop()
except ValueError:
    pass
spark_context = SparkContext("local[2]", "NetworkBigramCount")


stream_context = StreamingContext(spark_context, 1)
```

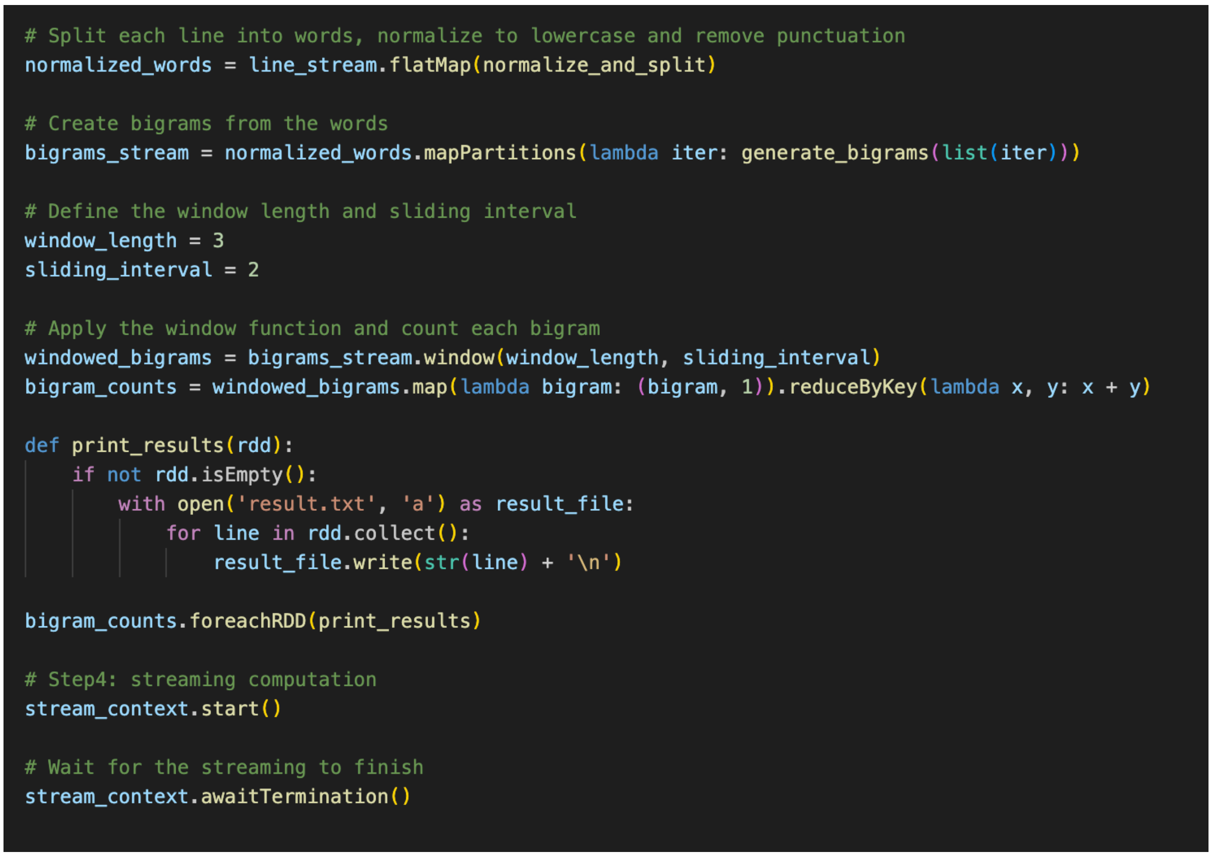
Step 2: Create a DStream that represents streaming data from a TCP source (localhost:9999). [hint: use Netcat]

Step 3: Split each line into words, normalise it to lowercase, and remove punctuation.



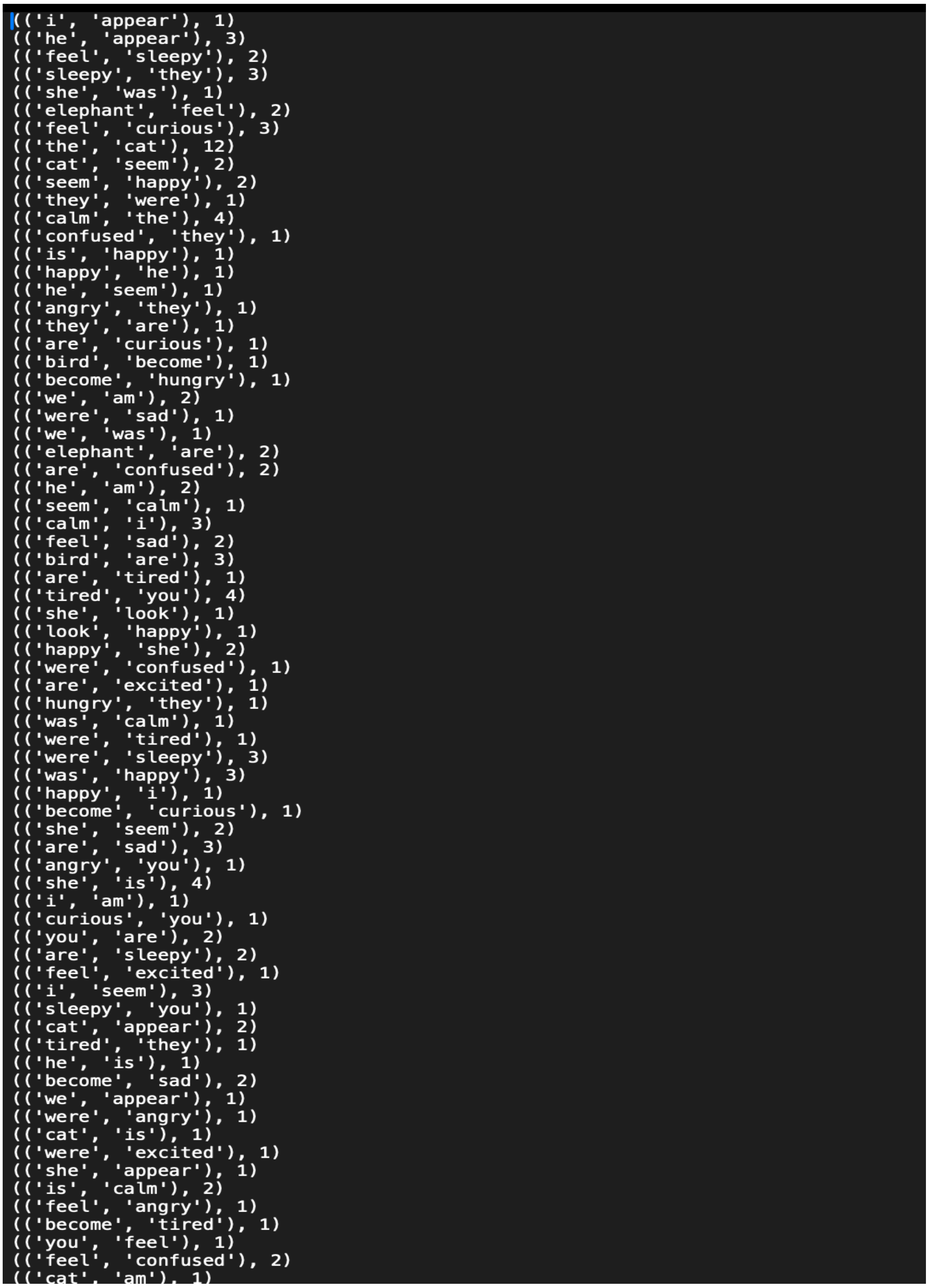
```
#Step3:  Create a DStream that connects to localhost:9999
line_stream = stream_context.socketTextStream("localhost", 9999)

open('result.txt', 'w').close()

# Split each line into words, normalize to lowercase and remove punctuation
normalized_words = line_stream.flatMap(normalize_and_split)

# Create bigrams from the words
bigrams_stream = normalized_words.mapPartitions(lambda iter: generate_bigrams(list(iter)))

# Define the window length and sliding interval
window_length = 3
sliding_interval = 2
```

Step 4 & Step 5: Create bigrams from a list of words; If you want to know how to create bigrams. Apply sliding window transformation to generate bigrams within a window. (hint: window length=3 and sliding interval=2). Count the occurrences of each bigram and print word co-occurrence counts.

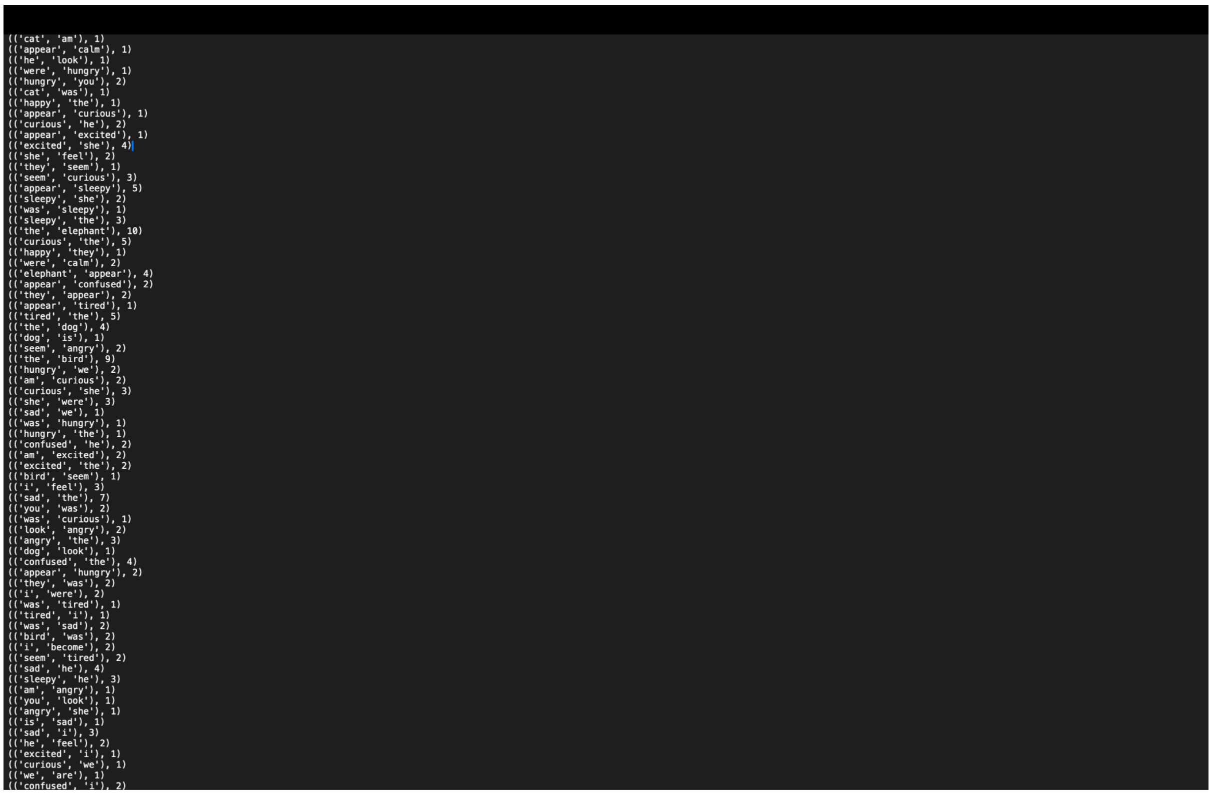


```
# Split each line into words, normalize to lowercase and remove punctuation
normalized_words = line_stream.flatMap(normalize_and_split)

# Create bigrams from the words
bigrams_stream = normalized_words.mapPartitions(lambda iter: generate_bigrams(list(iter)))

# Define the window length and sliding interval
window_length = 3
sliding_interval = 2

# Apply the window function and count each bigram
windowed_bigrams = bigrams_stream.window(window_length, sliding_interval)
bigram_counts = windowed_bigrams.map(lambda bigram: (bigram, 1)).reduceByKey(lambda x, y: x + y)

def print_results(rdd):
    if not rdd.isEmpty():
        with open('result.txt', 'a') as result_file:
            for line in rdd.collect():
                result_file.write(str(line) + '\n')

bigram_counts.foreachRDD(print_results)

# Step4: streaming computation
stream_context.start()

# Wait for the streaming to finish
stream_context.awaitTermination()
```

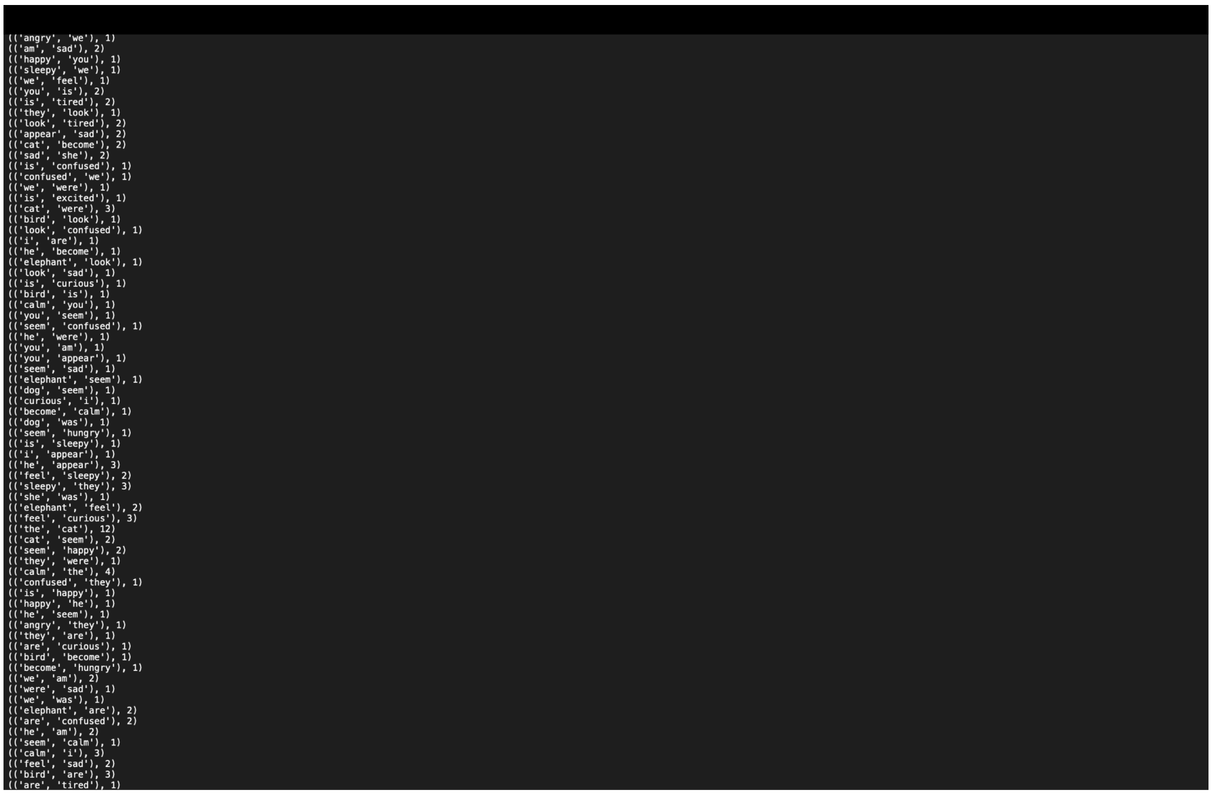
Output :



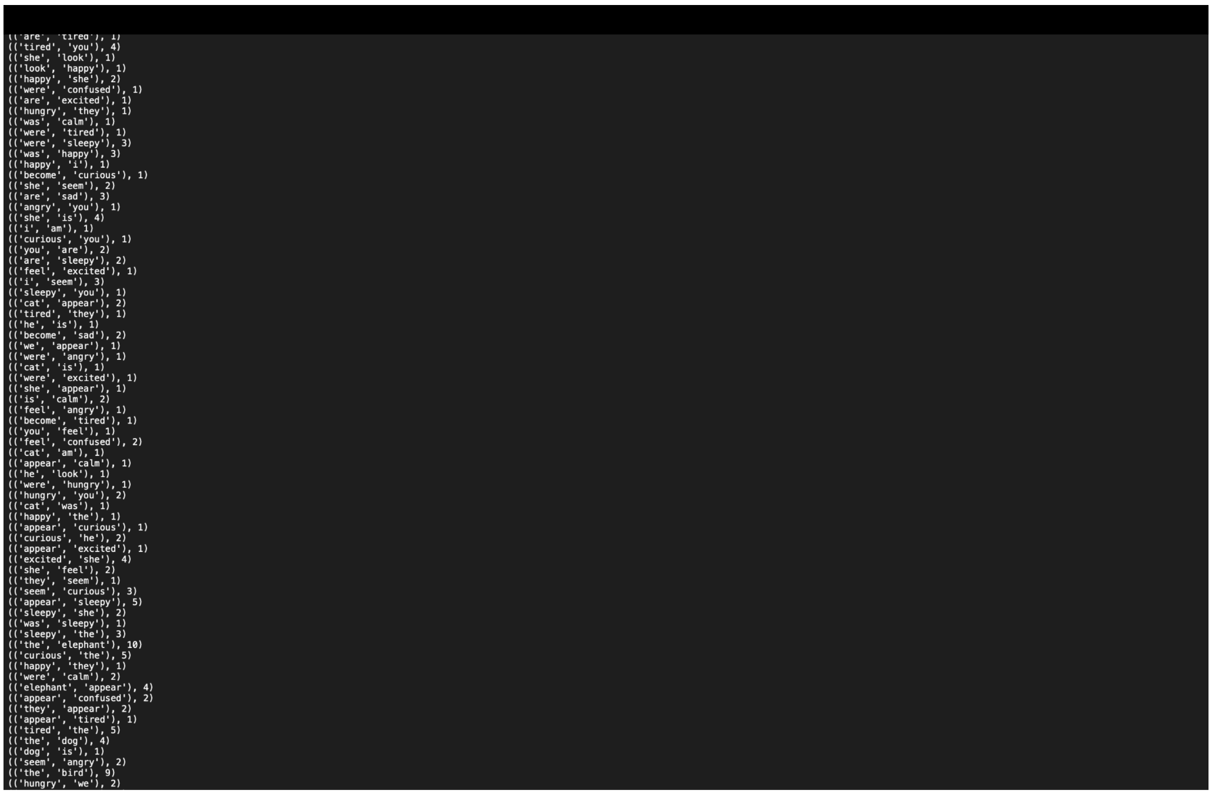
```
(('i', 'appear'), 1)
(('he', 'appear'), 3)
(('feel', 'sleepy'), 2)
(('sleepy', 'they'), 3)
(('she', 'was'), 1)
(('elephant', 'feel'), 2)
(('feel', 'curious'), 3)
(('the', 'cat'), 12)
(('cat', 'seem'), 2)
(('seem', 'happy'), 2)
(('they', 'were'), 1)
(('calm', 'the'), 4)
(('confused', 'they'), 1)
(('is', 'happy'), 1)
(('happy', 'he'), 1)
(('he', 'seem'), 1)
(('angry', 'they'), 1)
(('they', 'are'), 1)
(('are', 'curious'), 1)
(('bird', 'become'), 1)
(('become', 'hungry'), 1)
(('we', 'am'), 2)
(('were', 'sad'), 1)
(('we', 'was'), 1)
(('elephant', 'are'), 2)
(('are', 'confused'), 2)
(('he', 'am'), 2)
(('seem', 'calm'), 1)
(('calm', 'i'), 3)
(('feel', 'sad'), 2)
(('bird', 'are'), 3)
(('are', 'tired'), 1)
(('tired', 'you'), 4)
(('she', 'look'), 1)
(('look', 'happy'), 1)
(('happy', 'she'), 2)
(('were', 'confused'), 1)
(('are', 'excited'), 1)
(('hungry', 'they'), 1)
(('was', 'calm'), 1)
(('were', 'tired'), 1)
(('were', 'sleepy'), 3)
(('was', 'happy'), 3)
(('happy', 'i'), 1)
(('become', 'curious'), 1)
(('she', 'seem'), 2)
(('are', 'sad'), 3)
(('angry', 'you'), 1)
(('she', 'is'), 4)
(('i', 'am'), 1)
(('curious', 'you'), 1)
(('you', 'are'), 2)
(('are', 'sleepy'), 2)
(('feel', 'excited'), 1)
(('i', 'seem'), 3)
(('sleepy', 'you'), 1)
(('cat', 'appear'), 2)
(('tired', 'they'), 1)
(('he', 'is'), 1)
(('become', 'sad'), 2)
(('we', 'appear'), 1)
(('were', 'angry'), 1)
(('cat', 'is'), 1)
(('were', 'excited'), 1)
(('she', 'appear'), 1)
(('is', 'calm'), 2)
(('feel', 'angry'), 1)
(('become', 'tired'), 1)
(('you', 'feel'), 1)
(('feel', 'confused'), 2)
(('cat', 'am'), 1)
```



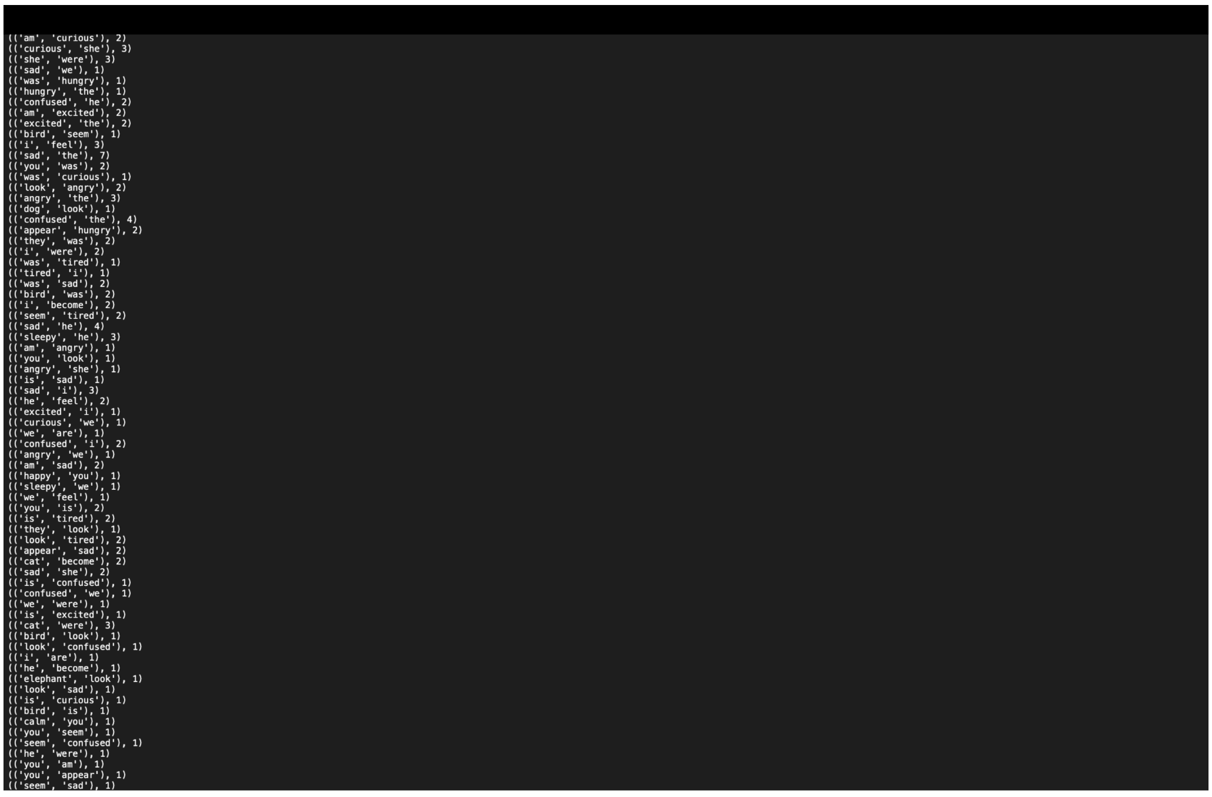
```
(('cat', 'am'), 1)
(('appear', 'calm'), 1)
(('he', 'look'), 1)
(('were', 'hungry'), 1)
(('hungry', 'you'), 2)
(('cat', 'was'), 1)
(('happy', 'the'), 1)
(('appear', 'curious'), 1)
(('curious', 'he'), 2)
(('appear', 'excited'), 1)
(('excited', 'she'), 4)
(('she', 'feel'), 2)
(('they', 'seem'), 1)
(('seem', 'curious'), 3)
(('appear', 'sleepy'), 5)
(('sleepy', 'she'), 2)
(('was', 'sleepy'), 1)
(('sleepy', 'the'), 3)
(('the', 'elephant'), 10)
(('curious', 'the'), 5)
(('happy', 'they'), 1)
(('were', 'calm'), 2)
(('elephant', 'appear'), 4)
(('appear', 'confused'), 2)
(('they', 'appear'), 2)
(('appear', 'tired'), 1)
(('tired', 'the'), 5)
(('the', 'dog'), 4)
(('dog', 'is'), 1)
(('seem', 'angry'), 2)
(('the', 'bird'), 9)
(('hungry', 'we'), 2)
(('am', 'curious'), 2)
(('curious', 'she'), 3)
(('she', 'were'), 3)
(('sad', 'we'), 1)
(('was', 'hungry'), 1)
(('hungry', 'the'), 1)
(('confused', 'he'), 2)
(('am', 'excited'), 2)
(('excited', 'the'), 2)
(('bird', 'seem'), 1)
(('i', 'feel'), 3)
(('sad', 'the'), 7)
(('you', 'was'), 2)
(('was', 'curious'), 1)
(('look', 'angry'), 2)
(('angry', 'the'), 3)
(('dog', 'look'), 1)
(('confused', 'the'), 4)
(('appear', 'hungry'), 2)
(('they', 'was'), 2)
(('i', 'were'), 2)
(('was', 'tired'), 1)
(('tired', 'i'), 1)
(('was', 'sad'), 2)
(('bird', 'was'), 2)
(('i', 'become'), 2)
(('seem', 'tired'), 2)
(('sad', 'he'), 4)
(('sleepy', 'he'), 3)
(('am', 'angry'), 1)
(('you', 'look'), 1)
(('angry', 'she'), 1)
(('is', 'sad'), 1)
(('sad', 'i'), 3)
(('he', 'feel'), 2)
(('excited', 'i'), 1)
(('curious', 'we'), 1)
(('we', 'are'), 1)
(('confused', 'i'), 2)
```



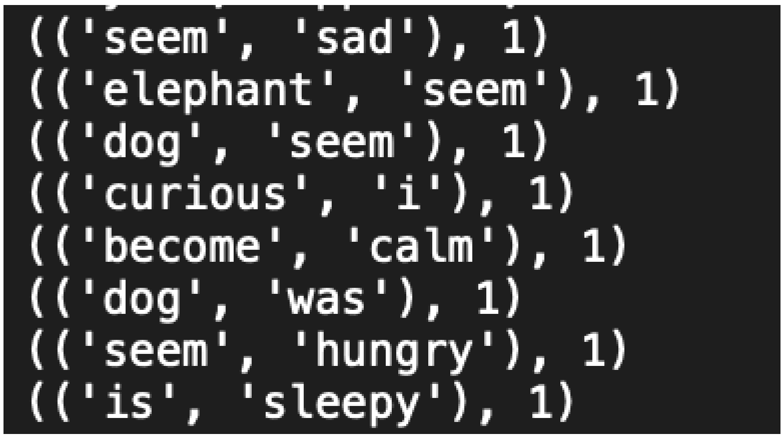
```
(('angry', 'we'), 1)
(('am', 'sad'), 2)
(('happy', 'you'), 1)
(('sleepy', 'we'), 1)
(('we', 'feel'), 1)
(('you', 'is'), 2)
(('is', 'tired'), 2)
(('they', 'look'), 1)
(('look', 'tired'), 2)
(('appear', 'sad'), 2)
(('cat', 'become'), 2)
(('sad', 'she'), 2)
(('is', 'confused'), 1)
(('confused', 'we'), 1)
(('we', 'were'), 1)
(('is', 'excited'), 1)
(('cat', 'were'), 3)
(('bird', 'look'), 1)
(('look', 'confused'), 1)
(('i', 'are'), 1)
(('he', 'become'), 1)
(('elephant', 'look'), 1)
(('look', 'sad'), 1)
(('is', 'curious'), 1)
(('bird', 'is'), 1)
(('calm', 'you'), 1)
(('you', 'seem'), 1)
(('seem', 'confused'), 1)
(('he', 'were'), 1)
(('you', 'am'), 1)
(('you', 'appear'), 1)
(('seem', 'sad'), 1)
(('elephant', 'seem'), 1)
(('dog', 'seem'), 1)
(('curious', 'i'), 1)
(('become', 'calm'), 1)
(('dog', 'was'), 1)
(('seem', 'hungry'), 1)
(('is', 'sleepy'), 1)
(('i', 'appear'), 1)
(('he', 'appear'), 3)
(('feel', 'sleepy'), 2)
(('sleepy', 'they'), 3)
(('she', 'was'), 1)
(('elephant', 'feel'), 2)
(('feel', 'curious'), 3)
(('the', 'cat'), 12)
(('cat', 'seem'), 2)
(('seem', 'happy'), 2)
(('they', 'were'), 1)
(('calm', 'the'), 4)
(('confused', 'they'), 1)
(('is', 'happy'), 1)
(('happy', 'he'), 1)
(('he', 'seem'), 1)
(('angry', 'they'), 1)
(('they', 'are'), 1)
(('are', 'curious'), 1)
(('bird', 'become'), 1)
(('become', 'hungry'), 1)
(('we', 'am'), 2)
(('were', 'sad'), 1)
(('we', 'was'), 1)
(('elephant', 'are'), 2)
(('are', 'confused'), 2)
(('he', 'am'), 2)
(('seem', 'calm'), 1)
(('calm', 'i'), 3)
(('feel', 'sad'), 2)
(('bird', 'are'), 3)
(('are', 'tired'), 1)
```



```
(('are', 'tired'), 1)
(('tired', 'you'), 4)
(('she', 'look'), 1)
(('look', 'happy'), 1)
(('happy', 'she'), 2)
(('were', 'confused'), 1)
(('are', 'excited'), 1)
(('hungry', 'they'), 1)
(('was', 'calm'), 1)
(('were', 'tired'), 1)
(('were', 'sleepy'), 3)
(('was', 'happy'), 3)
(('happy', 'i'), 1)
(('become', 'curious'), 1)
(('she', 'seem'), 2)
(('are', 'sad'), 3)
(('angry', 'you'), 1)
(('she', 'is'), 4)
(('i', 'am'), 1)
(('curious', 'you'), 1)
(('you', 'are'), 2)
(('are', 'sleepy'), 2)
(('feel', 'excited'), 1)
(('i', 'seem'), 3)
(('sleepy', 'you'), 1)
(('cat', 'appear'), 2)
(('tired', 'they'), 1)
(('he', 'is'), 1)
(('become', 'sad'), 2)
(('we', 'appear'), 1)
(('were', 'angry'), 1)
(('cat', 'is'), 1)
(('were', 'excited'), 1)
(('she', 'appear'), 1)
(('is', 'calm'), 2)
(('feel', 'angry'), 1)
(('become', 'tired'), 1)
(('you', 'feel'), 1)
(('feel', 'confused'), 2)
(('cat', 'am'), 1)
(('appear', 'calm'), 1)
(('he', 'look'), 1)
(('were', 'hungry'), 1)
(('hungry', 'you'), 2)
(('cat', 'was'), 1)
(('happy', 'the'), 1)
(('appear', 'curious'), 1)
(('curious', 'he'), 2)
(('appear', 'excited'), 1)
(('excited', 'she'), 4)
(('she', 'feel'), 2)
(('they', 'seem'), 1)
(('seem', 'curious'), 3)
(('appear', 'sleepy'), 5)
(('sleepy', 'she'), 2)
(('was', 'sleepy'), 1)
(('sleepy', 'the'), 3)
(('the', 'elephant'), 10)
(('curious', 'the'), 5)
(('happy', 'they'), 1)
(('were', 'calm'), 2)
(('elephant', 'appear'), 4)
(('appear', 'confused'), 2)
(('they', 'appear'), 2)
(('appear', 'tired'), 1)
(('tired', 'the'), 5)
(('the', 'dog'), 4)
(('dog', 'is'), 1)
(('seem', 'angry'), 2)
(('the', 'bird'), 9)
(('hungry', 'we'), 2)
```



```
(('am', 'curious'), 2)
(('curious', 'she'), 3)
(('she', 'were'), 3)
(('sad', 'we'), 1)
(('was', 'hungry'), 1)
(('hungry', 'the'), 1)
(('confused', 'he'), 2)
(('am', 'excited'), 2)
(('excited', 'the'), 2)
(('bird', 'seem'), 1)
(('i', 'feel'), 3)
(('sad', 'the'), 7)
(('you', 'was'), 2)
(('was', 'curious'), 1)
(('look', 'angry'), 2)
(('angry', 'the'), 3)
(('dog', 'look'), 1)
(('confused', 'the'), 4)
(('appear', 'hungry'), 2)
(('they', 'was'), 2)
(('i', 'were'), 2)
(('was', 'tired'), 1)
(('tired', 'i'), 1)
(('was', 'sad'), 2)
(('bird', 'was'), 2)
(('i', 'become'), 2)
(('seem', 'tired'), 2)
(('sad', 'he'), 4)
(('sleepy', 'he'), 3)
(('am', 'angry'), 1)
(('you', 'look'), 1)
(('angry', 'she'), 1)
(('is', 'sad'), 1)
(('sad', 'i'), 3)
(('he', 'feel'), 2)
(('excited', 'i'), 1)
(('curious', 'we'), 1)
(('we', 'are'), 1)
(('confused', 'i'), 2)
(('angry', 'we'), 1)
(('am', 'sad'), 2)
(('happy', 'you'), 1)
(('sleepy', 'we'), 1)
(('we', 'feel'), 1)
(('you', 'is'), 2)
(('is', 'tired'), 2)
(('they', 'look'), 1)
(('look', 'tired'), 2)
(('appear', 'sad'), 2)
(('cat', 'become'), 2)
(('sad', 'she'), 2)
(('is', 'confused'), 1)
(('confused', 'we'), 1)
(('we', 'were'), 1)
(('is', 'excited'), 1)
(('cat', 'were'), 3)
(('bird', 'look'), 1)
(('look', 'confused'), 1)
(('i', 'are'), 1)
(('he', 'become'), 1)
(('elephant', 'look'), 1)
(('look', 'sad'), 1)
(('is', 'curious'), 1)
(('bird', 'is'), 1)
(('calm', 'you'), 1)
(('you', 'seem'), 1)
(('seem', 'confused'), 1)
(('he', 'were'), 1)
(('you', 'am'), 1)
(('you', 'appear'), 1)
(('seem', 'sad'), 1)
```



```
(('seem', 'sad'), 1)
(('elephant', 'seem'), 1)
(('dog', 'seem'), 1)
(('curious', 'i'), 1)
(('become', 'calm'), 1)
(('dog', 'was'), 1)
(('seem', 'hungry'), 1)
(('is', 'sleepy'), 1)
```

Part-2 Review on Discretized Streams: Fault-Tolerant Streaming Computation at Scale”

The paper presents “Discretized Streams : Fault-Tolerant Streaming Computation at Scale” where it presents a large-scale data stream processing through D-streams system also called Discretized Streams. It identifies challenges like fault tolerance, straggler handling , traditional stream processing and consistency. In this we can also learn about the deterministic batch processing model.

In this paper a lot of innovations and key concepts were discussed. Now we will discuss them.
D- Streams Model,The main idea of D-Streams is to transform deterministic computations over the discrete time intervals by streaming computations. In this the operators continuously process the records that are incoming where this approach contrasts with the time processing model. The large incoming data in an interval and process these batches as RDD, which is fault-tolerant data structure. In this method the inputs always have the same output, where the recovery and parallel processing are reliable for facilitating.

Fault Tolerance and Straggler Mitigation mostly relies on costly upstream strategies which are used for backup. D-Streams, RDDs, employ parallel recovery mechanisms where the recovery workload in multiple nodes loses its state by distributing workload. Because of this it reduces the time used to recover for backup methods, where the lost nodes to be reconstructed. Here, D-Streams handle stragglers through execution with speculative, which is a technique executed on multiple nodes for the timely completion.

Unified Batch and Streaming Processing is the most useful benefit of D-Streams is that it can unify the batch and streaming process in the same framework. Using RDD data structures and their process models, it can integrate with sparks batch capabilities. It can unify complex applications where it works in real-time and data of historical analysis have to be implemented in a more efficient way.

Talking about the scalability and performance, D-Streams can implement streaming demonstrations of spark to an impressive performance and scalability. In that 60 million records per sec in a 100-node cluster, where the latencies are as low as half second. It can be done by not sacrificing fault tolerance and consistency. The paper also discusses the use cases, like video distribution system and online ML algorithm, for the flexibility and efficiency of D-Streams.

We can also discuss that D-Streams offer will have many advantages. The minimum latency Can be acceptable for the real-world applications that do not require millisecond-level response. Paper also suggests tips for future research like optimization , multi-tenancy and the result computation. The following discussed methods have potential to diversify the data streaming for scenarios.

The “Discretized Streams: Fault-Tolerant Streaming Computation at Scale” is very advanced in the field of distributed stream processing. We can adapt to the processing approach of the batch. D-Streams face many limitations for streaming systems, where it provides fault tolerance , straggler mitigation, and also the unified framework of streaming analytics. The innovations in this paper will teach about scalable and reliable real-time data processing where it helps to address the big data applications. The D-Streams will always contribute to fault-tolerance and scalable contributions will make it more valuable for research purposes and used for systems with distributed and the final data analytics.

Part-3 Biases in DATA

Bias in data tells about the errors which are systematic and the deviations of collected data, interpretation, analysis and inaccurate presentation which are unfair for the results. It leads to incorrect conclusions which impact the decision making process.

Types of Bias :
1) Historical Bias : This bias occurs when the data relates to historical inequalities, stereotypes.
2) Representation Bias : This bias occurs when the data does not fairly represent the population as a whole, where it results in undersampling of groups.
3) Measurement Bias : measurement bias occurs when it favours some of the relatable outcomes or information is not well represented when the data is collected.
4) Aggregation Bias : It occurs when the data is collected from various groups then combines them without taking into account their differences.
5) Evaluation Bias : This bias occurs when evaluation of data is done by criteria which leads to inaccurate evaluation and conclusions.
6) Deployment Bias : This bias works well when a training and controlled environment in the real-world performs poorly due to its variations in conditions.

Scenario 1 : Student attendance tracking using wearable technology to monitor

Bias type : Representation Bias

Manifesting : Representation Bias may apply for this scenario because students may not use wearable devices every time and may not wear a smartwatch. Students can also choose not to use the app or utilise the app. These causes for representation bias to occur.

Fix : regardless of their situations we have to make sure every student has access to wearable technology. We can also provide alternate methods where instead of wearable accessories, smartphones can be used for tracking apps, which is more reliable for students.

Scenario 2 : Improving roaster building decisions with data-intensive computing

Bias type : Historical Bias

Manifesting : Historical Bias focuses more on the game performance of the players from the last five years and their changes in performance over the time which contributed to winning the games. This may read the skill level of players inaccurately which might not be the best way to select the players.

Fix : To create the best team possible, we should be more concerned about the individual player's contribution to the game and their stats which helped the team to win. Stats of player skills will contribute better for selecting the team. We can use contextual information and their performance metrics to address the bias.