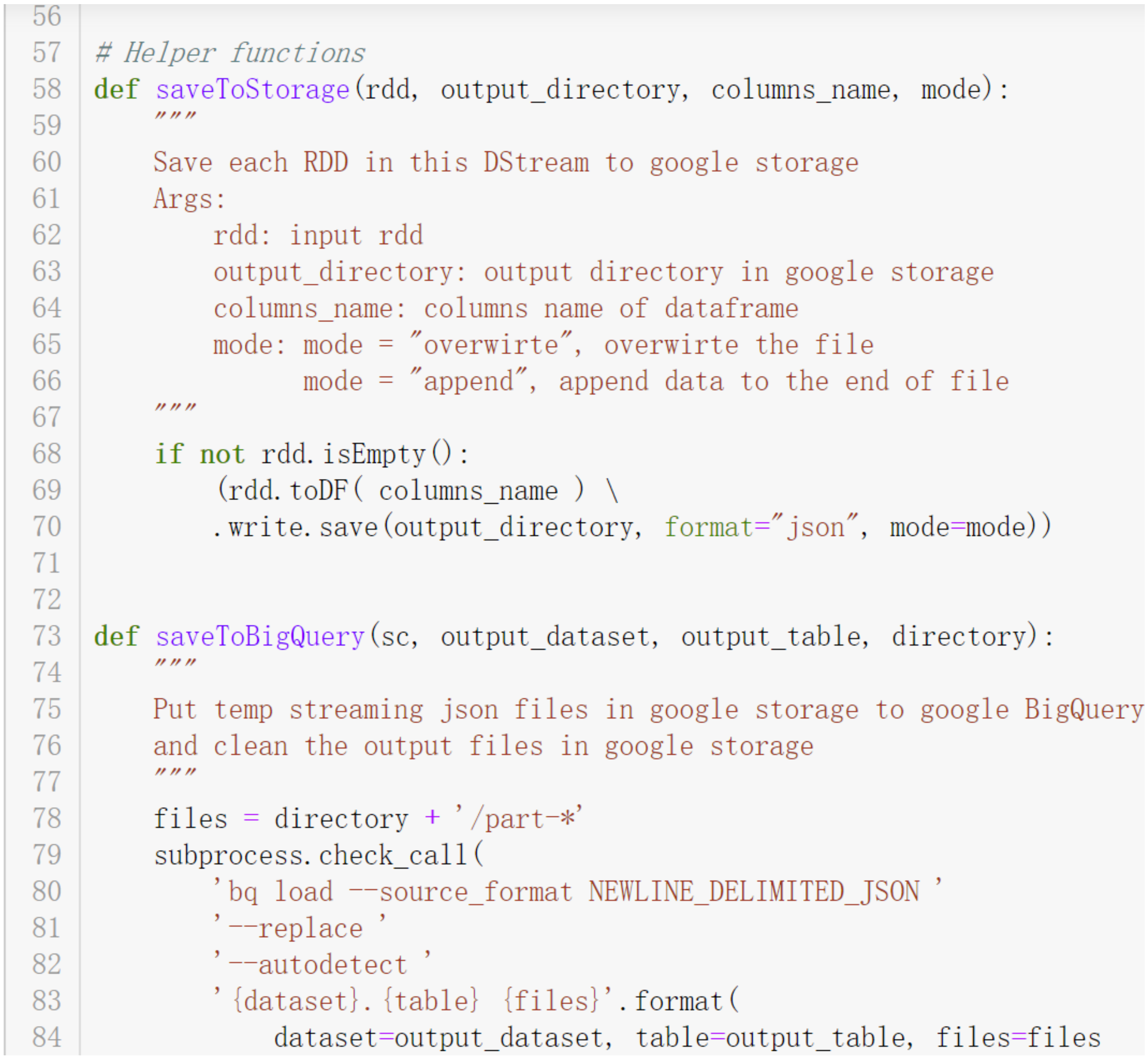
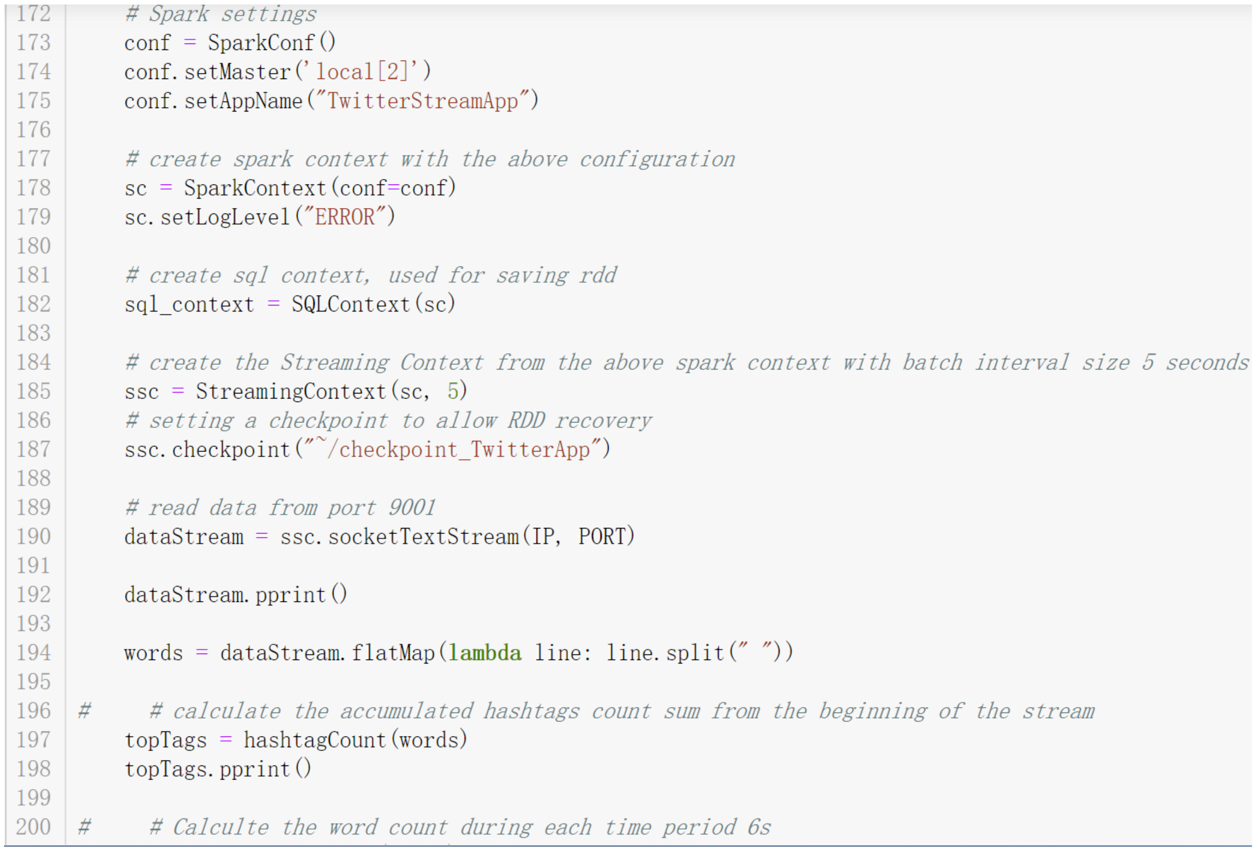
EECS 6893: Big Data Analytics
HW3 PartII
Yutao Zhou UNI: yz4359
Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Columbia EECS E6893 Big Data Analytics
"""
This module is the spark streaming analysis process.


Usage:
    If used with dataproc:
        gcloud dataproc jobs submit pyspark --cluster <Cluster Name> twitterHTTPClient.py

    Create a dataset in BigQurey first using
        bq mk bigdata_sparkStreaming

    Remeber to replace the bucket with your own bucket name


Todo:
    1. hashtagCount: calculate accumulated hashtags count
    2. wordCount: calculate word count every 60 seconds
        the word you should track is listed below.
    3. save the result to google BigQuery

"""

from pyspark import SparkConf,SparkContext
```

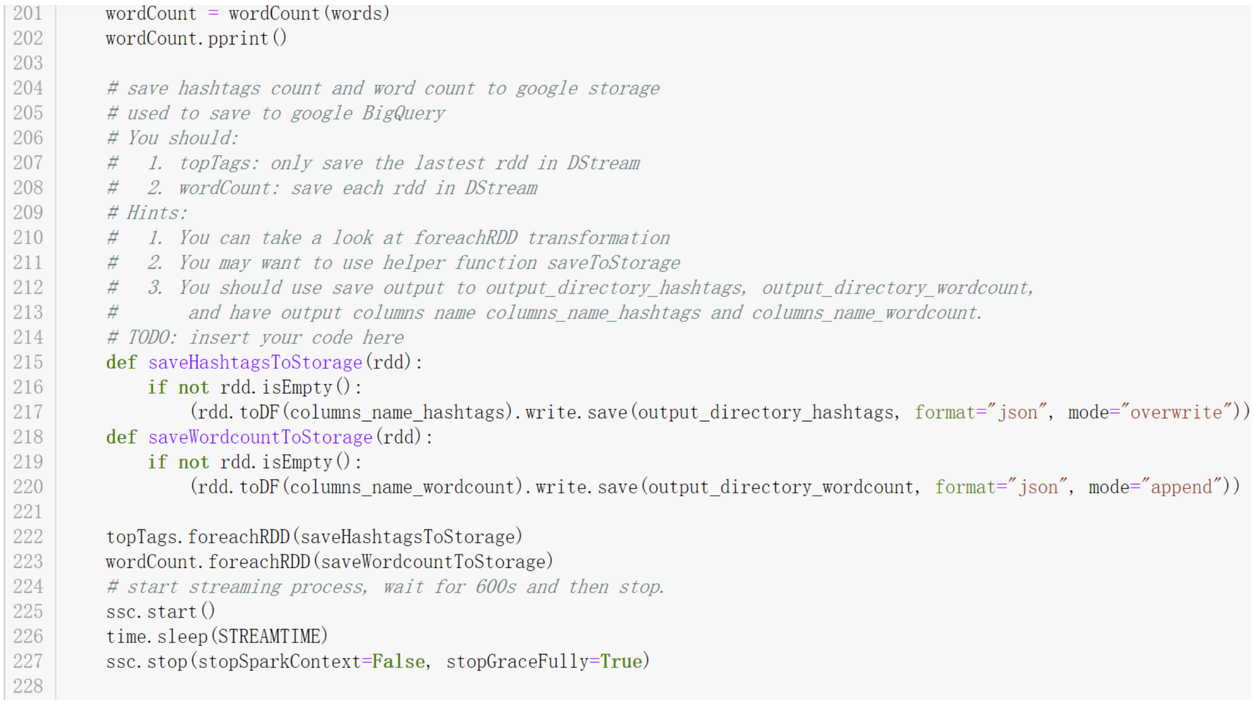
```python
from pyspark.streaming import StreamingContext
from pyspark.sql import Row,SQLContext
import sys
import requests
import time
import subprocess
import re
from google.cloud import bigquery
from datetime import datetime

# global variables

bucket = "hw3twitter"     # TODO : replace with your own bucket name
output_directory_hashtags = 'gs://{}/hadoop/tmp/bigquery/pyspark_output/hashtagscount'.format(bucket)
output_directory_wordcount = 'gs://{}/hadoop/tmp/bigquery/pyspark_output/wordcount'.format(bucket)

# output table and columns name
output_dataset = 'twitterStreaming'                      #the name of your dataset in BigQuery
output_table_hashtags = 'hashtags'
columns_name_hashtags = ['hashtags', 'count']
output_table_wordcount = 'wordcount'
columns_name_wordcount = ['word', 'count', 'time']

# parameter
IP = 'localhost'     # ip port
PORT = 9001       # port
STREAMTIME = 600          # time that the streaming process runs
timeStamp = datetime.fromtimestamp(time.time()).strftime("%Y-%m-%d, %H:%M:%S")
WORD = ['data', 'spark', 'ai', 'movie', 'good']     #the words you should filter and do word count
```



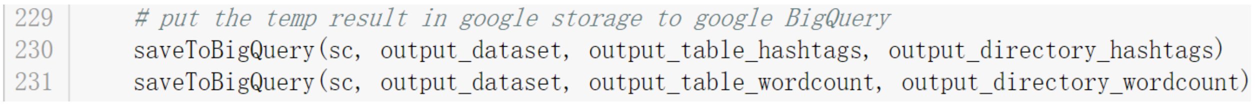
```python

# Helper functions
def saveToStorage(rdd, output_directory, columns_name, mode):
    """
    Save each RDD in this DStream to google storage
    Args:
        rdd: input rdd
        output_directory: output directory in google storage
        columns_name: columns name of dataframe
        mode: mode = "overwirte", overwirte the file
              mode = "append", append data to the end of file
    """
    if not rdd.isEmpty():
        (rdd.toDF( columns_name ) \
        .write.save(output_directory, format="json", mode=mode))


def saveToBigQuery(sc, output_dataset, output_table, directory):
    """
    Put temp streaming json files in google storage to google BigQuery
    and clean the output files in google storage
    """
    files = directory + '/part-*'
    subprocess.check_call(
        'bq load --source_format NEWLINE_DELIMITED_JSON '
        '--replace '
        '--autodetect '
        '{dataset}.{table} {files}'.format(
            dataset=output_dataset, table=output_table, files=files
        ).split())
    output_path = sc._jvm.org.apache.hadoop.fs.Path(directory)
    output_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(
        output_path, True)


def hashtagCount(words):
    """
    Calculate the accumulated hashtags count sum from the beginning of the stream
    and sort it by descending order of the count.
    Ignore case sensitivity when counting the hashtags:
        "#Ab" and "#ab" is considered to be a same hashtag
    You have to:
    1. Filter out the word that is hashtags.
       Hashtag usually start with "#" and followed by a serious of alphanumeric
    2. map (hashtag) to (hashtag, 1)
    3. sum the count of current DStream state and previous state
    4. transform unordered DStream to a ordered Dstream
    Hints:
        you may use regular expression to filter the words
        You can take a look at updateStateByKey and transform transformations
    Args:
        dstream(DStream): stream of real time tweets
    Returns:
        DStream Object with inner structure (hashtag, count)
    """

    # TODO: insert your code here
    def filterHashtag(word):
        m = re.match(r"#[0-9a-z]+", word)
        if m:
            return True
        return False

    def updateFunction(newValues, runningCount):
        if runningCount is None:
            runningCount = 0
        return sum(newValues, runningCount)

    words = words.map(lambda word: word.lower())
    words = words.filter(lambda word: filterHashtag(word))
    words = words.map(lambda word: (re.match(r"#[0-9a-z]+", word).group(), 1) )
    hashTags = words.updateStateByKey(updateFunction)
    hashTags = hashTags.transform(lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False))
    return hashTags

def wordCount(words):
    """
    Calculte the count of 5 sepcial words for every 60 seconds (window no overlap)
    You can choose your own words.
    Your should:
    1. filter the words
    2. count the word during a special window size
    3. add a time related mark to the output of each window, ex: a datetime type
    Hints:
        You can take a look at reduceByKeyAndWindow transformation
        Dstream is a serious of rdd, each RDD in a DStream contains data from a certain interval
        You may want to take a look of transform transformation of DStream when trying to add a time
    Args:
        dstream(DStream): stream of real time tweets
    Returns:
        DStream Object with inner structure (word, (count, time))
    """

    # TODO: insert your code here
    def filterSpecialWords(word):
        if word in WORD:
            return True
        return False
    def wordCountUpdateFunction(runningCount, newValues):
        if runningCount is None:
            runningCount = 0
        return runningCount + newValues
    def wordCountInverseUpdateFunction(runningCount, newValues):
        if runningCount is None:
            runningCount = 0
        return runningCount - newValues

    words = words.map(lambda word: word.lower())
    words = words.filter(lambda word: filterSpecialWords(word))
    words = words.map(lambda word: (word, 1))
    words = words.reduceByKeyAndWindow(wordCountUpdateFunction, wordCountInverseUpdateFunction, 60, 60)
    words = words.transform(lambda timeStamp, rdd: rdd.map(lambda x: (x[0], x[1], str(timeStamp))))
    return words


if __name__ == '__main__':
```



```python
    # Spark settings
    conf = SparkConf()
    conf.setMaster('local[2]')
    conf.setAppName("TwitterStreamApp")

    # create spark context with the above configuration
    sc = SparkContext(conf=conf)
    sc.setLogLevel("ERROR")

    # create sql context, used for saving rdd
    sql_context = SQLContext(sc)

    # create the Streaming Context from the above spark context with batch interval size 5 seconds
    ssc = StreamingContext(sc, 5)
    # setting a checkpoint to allow RDD recovery
    ssc.checkpoint("~/checkpoint_TwitterApp")

    # read data from port 9001
    dataStream = ssc.socketTextStream(IP, PORT)

    dataStream.pprint()

    words = dataStream.flatMap(lambda line: line.split(" "))

#     # calculate the accumulated hashtags count sum from the beginning of the stream
    topTags = hashtagCount(words)
    topTags.pprint()

#     # Calculte the word count during each time period 6s
```

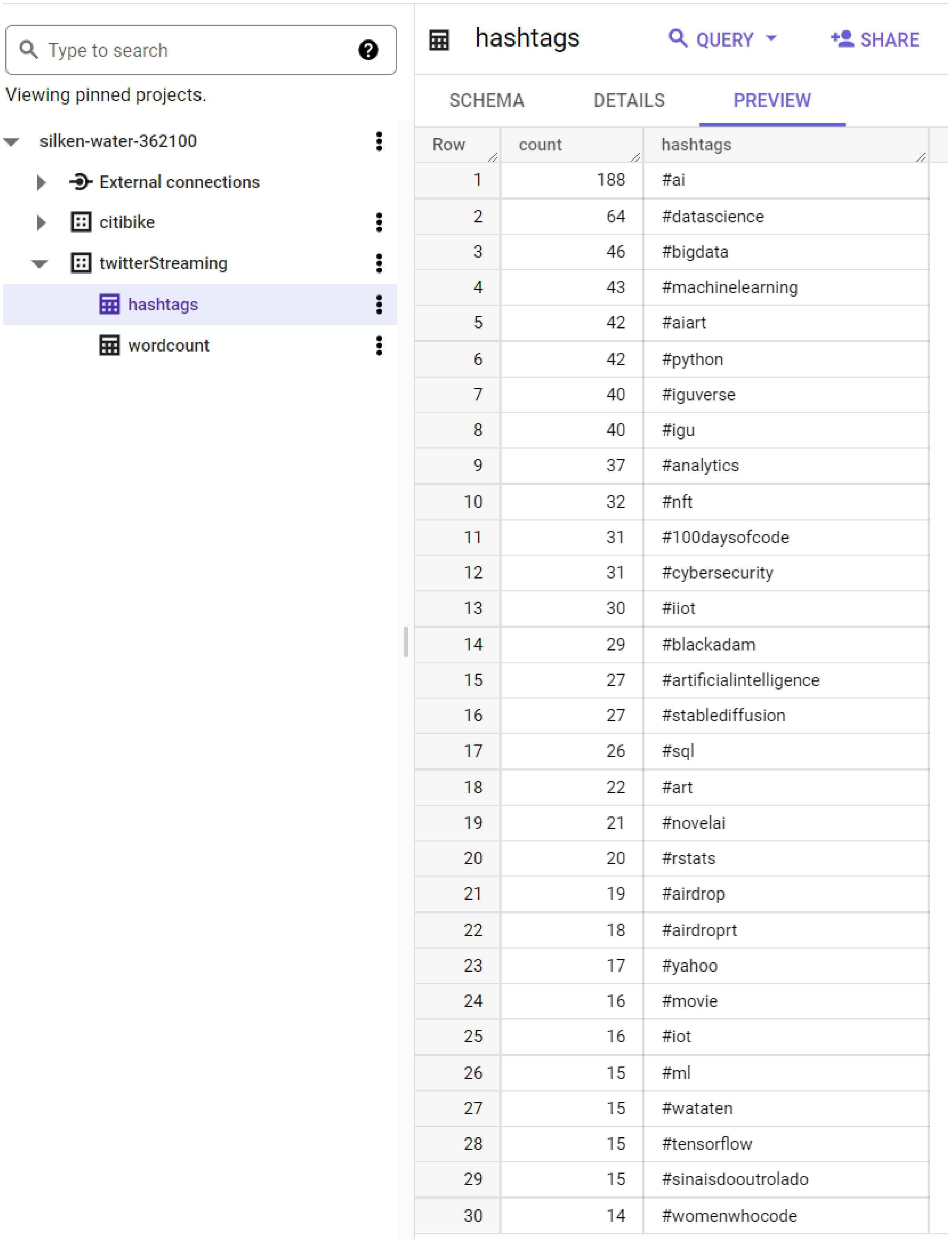


```python
    wordCount = wordCount(words)
    wordCount.pprint()

    # save hashtags count and word count to google storage
    # used to save to google BigQuery
    # You should:
    #   1. topTags: only save the lastest rdd in DStream
    #   2. wordCount: save each rdd in DStream
    # Hints:
    #   1. You can take a look at foreachRDD transformation
    #   2. You may want to use helper function saveToStorage
    #   3. You should use save output to output_directory_hashtags, output_directory_wordcount,
    #       and have output columns name columns_name_hashtags and columns_name_wordcount.
    # TODO: insert your code here
    def saveHashtagsToStorage(rdd):
        if not rdd.isEmpty():
            (rdd.toDF(columns_name_hashtags).write.save(output_directory_hashtags, format="json", mode="overwrite"))
    def saveWordcountToStorage(rdd):
        if not rdd.isEmpty():
            (rdd.toDF(columns_name_wordcount).write.save(output_directory_wordcount, format="json", mode="append"))

    topTags.foreachRDD(saveHashtagsToStorage)
    wordCount.foreachRDD(saveWordcountToStorage)
    # start streaming process, wait for 600s and then stop.
    ssc.start()
    time.sleep(STREAMTIME)
    ssc.stop(stopSparkContext=False, stopGraceFully=True)

```

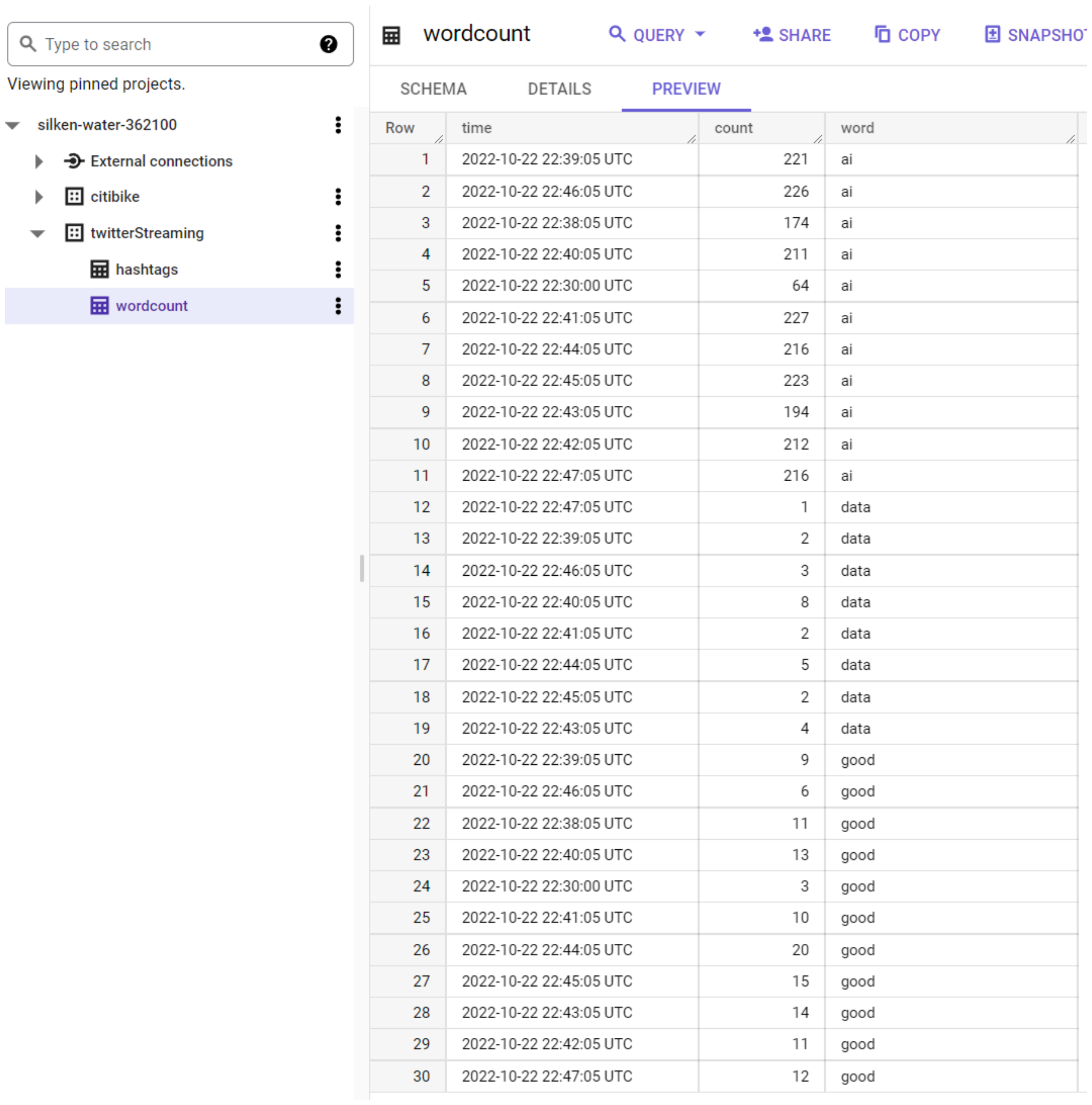


```python
    # put the temp result in google storage to google BigQuery
    saveToBigQuery(sc, output_dataset, output_table_hashtags, output_directory_hashtags)
    saveToBigQuery(sc, output_dataset, output_table_wordcount, output_directory_wordcount)
```

Result



Row	count	hashtags
1	188	#ai
2	64	#datascience
3	46	#bigdata
4	43	#machinelearning
5	42	#aiart
6	42	#python
7	40	#iguverse
8	40	#igu
9	37	#analytics
10	32	#nft
11	31	#100daysofcode
12	31	#cybersecurity
13	30	#iiot
14	29	#blackadam
15	27	#artificialintelligence
16	27	#stablediffusion
17	26	#sql
18	22	#art
19	21	#novelai
20	20	#rstats
21	19	#airdrop
22	18	#airdroprt
23	17	#yahoo
24	16	#movie
25	16	#iot
26	15	#ml
27	15	#watanten
28	15	#tensorflow
29	15	#sinaisdooutrolado
30	14	#womenwhocode



Row	time	count	word
1	2022-10-22 22:39:05 UTC	221	ai
2	2022-10-22 22:46:05 UTC	226	ai
3	2022-10-22 22:38:05 UTC	174	ai
4	2022-10-22 22:40:05 UTC	211	ai
5	2022-10-22 22:30:00 UTC	64	ai
6	2022-10-22 22:41:05 UTC	227	ai
7	2022-10-22 22:44:05 UTC	216	ai
8	2022-10-22 22:45:05 UTC	223	ai
9	2022-10-22 22:43:05 UTC	194	ai
10	2022-10-22 22:42:05 UTC	212	ai
11	2022-10-22 22:47:05 UTC	216	ai
12	2022-10-22 22:47:05 UTC	1	data
13	2022-10-22 22:39:05 UTC	2	data
14	2022-10-22 22:46:05 UTC	3	data
15	2022-10-22 22:40:05 UTC	8	data
16	2022-10-22 22:41:05 UTC	2	data
17	2022-10-22 22:44:05 UTC	5	data
18	2022-10-22 22:45:05 UTC	2	data
19	2022-10-22 22:43:05 UTC	4	data
20	2022-10-22 22:39:05 UTC	9	good
21	2022-10-22 22:46:05 UTC	6	good
22	2022-10-22 22:38:05 UTC	11	good
23	2022-10-22 22:40:05 UTC	13	good
24	2022-10-22 22:30:00 UTC	3	good
25	2022-10-22 22:41:05 UTC	10	good
26	2022-10-22 22:44:05 UTC	20	good
27	2022-10-22 22:45:05 UTC	15	good
28	2022-10-22 22:43:05 UTC	14	good
29	2022-10-22 22:42:05 UTC	11	good
30	2022-10-22 22:47:05 UTC	12	good