Put input file into hdfs before streaming:

```
hdfs dfs -put /home/hdoop/dse062/week2/q1/input1.txt /062/
```

Streaming:

```
hadoop jar '/home/hdoop/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar' \
-files /home/hdoop/dse062/w2/q1/mapper.py#mapper.py,/home/hdoop/dse062/w2/q1/reducer.py#reducer.py \
-mapper 'python3 mapper.py' \
-reducer 'python3 reducer.py' \
-input /062/input1.txt \
-output /062/outq1
```

Average of all numbers;

Mapper

```
#!/usr/bin/env python

import sys

# input comes from standard input
for line in sys.stdin:
    # split the line into words
    words = line.strip().split()
    # process each word
    for word in words:
        try:
            # try to convert the word to a number
            number = float(word)
            # output the number as the value and '1' as the key
            print('average', number)
        except ValueError:
            # ignore words that are not numbers
            Pass
```

Reducer

```
#!/usr/bin/env python

import sys

# initialize variables to store the sum and count of numbers
total = 0
count = 0

# input comes from standard input
for line in sys.stdin:
    # split the line into key and value
    key, value = line.strip().split()
    # convert the value to a number
    number = float(value)
    # add the number to the total sum
    total += number
    # increment the count
    count += 1

# calculate the average
if count > 0:
    average = total / count
    # output the average
    print('average', average)
```

25 commands hdfs

```
# Create a directory in HDFS
hdfs dfs -mkdir /user/hadoop/example

# Create an empty file in HDFS
hdfs dfs -touchz /user/hadoop/example/empty_file.txt

# Copy a file from the local file system to HDFS
hdfs dfs -copyFromLocal local_file.txt /user/hadoop/example/

# Print file contents
hdfs dfs -cat /user/hadoop/example/local_file.txt

# Copy files from HDFS to the local file system
```

```
hdfs dfs -copyToLocal /user/hadoop/example/local_file.txt local_directory/

# Move a file from the local file system to HDFS
hdfs dfs -moveFromLocal local_file.txt /user/hadoop/example/

# Copy files within HDFS
hdfs dfs -cp /user/hadoop/example/local_file.txt /user/hadoop/example/copied_file.txt

# Move files within HDFS
hdfs dfs -mv /user/hadoop/example/copied_file.txt /user/hadoop/example/moved_file.txt

# Size of each file in a directory
hdfs dfs -du -h /user/hadoop/example/

# Total size of a directory or file
hdfs dfs -du -s -h /user/hadoop/example/

# Last modified time of a directory or path
hdfs dfs -stat /user/hadoop/example/

# Change the replication factor of a file/directory in HDFS
hdfs dfs -setrep -w 4 /user/hadoop/example/local_file.txt

# List the contents of a directory in HDFS
hdfs dfs -ls /user/hadoop/example/

# Remove a file from HDFS
hdfs dfs -rm /user/hadoop/example/local_file.txt

# Change File Permissions
hdfs dfs -chmod 755 /user/hadoop/example/local_file.txt

# Changing File Ownership
hdfs dfs -chown hadoop:hadoop /user/hadoop/example/local_file.txt

# Checksum Calculation
hdfs dfs -checksum /user/hadoop/example/local_file.txt

# File Concatenation
hdfs dfs -getmerge /user/hadoop/example/merged_file.txt /user/hadoop/example/local_file.txt

# File Compression/Decompression
hdfs dfs -gzip /user/hadoop/example/local_file.txt
hdfs dfs -gunzip /user/hadoop/example/local_file.txt.gz

# File Block Location Information
hdfs fsck /user/hadoop/example/local_file.txt -files -blocks -locations

# File Encryption/Decryption
hdfs dfs -encrypt /user/hadoop/example/local_file.txt
hdfs dfs -decrypt /user/hadoop/example/local_file.txt
```

avg of integers in alphanumeric input:

mapper

```
import sys
import re

# Input: lines of text from stdin
for line in sys.stdin:
    # Find all numbers in the line
    numbers = re.findall(r'\d+', line)
    # Emit each number
    for number in numbers:
        print(f"{number}\t1")
```

reducer

```
import sys

# Initialize variables
total = 0
count = 0
```

```
# Input: number-count pairs from stdin
for line in sys.stdin:
    # Parse the input
    number, cnt = line.strip().split('\t')
    cnt = int(cnt)
    # Update total and count
    total += int(number) * cnt
    count += cnt

# Calculate the average
if count > 0:
    average = total / count
    print(f"Average\t{average}")
```

PIG

Loading and Filtering Data:

```
-- Load data from 'input_data.txt'
data = LOAD 'input_data.txt' USING PigStorage(',') AS (id:int, name:chararray, age:int);

-- Filter out rows where age is greater than 18
filtered_data = FILTER data BY age > 18;

-- Store the filtered data
STORE filtered_data INTO 'output_data.txt' USING PigStorage(',');
```

Grouping and Aggregating Data:

```
-- Load data from 'input_data.txt'
data = LOAD 'input_data.txt' USING PigStorage(',') AS (id:int, name:chararray, age:int, gender:chararray);

-- Group data by gender
grouped_data = GROUP data BY gender;

-- Calculate average age for each gender
avg_age = FOREACH grouped_data GENERATE group AS gender, AVG(data.age) AS average_age;

-- Store the average age
STORE avg_age INTO 'output_avg_age.txt' USING PigStorage(',');
```

Joining Data:

```
-- Load data from 'users.txt' and 'transactions.txt'
users = LOAD 'users.txt' USING PigStorage(',') AS (user_id:int, name:chararray);
transactions = LOAD 'transactions.txt' USING PigStorage(',') AS (user_id:int, amount:double);

-- Join the two datasets on user_id
joined_data = JOIN users BY user_id, transactions BY user_id;

-- Store the joined data
STORE joined_data INTO 'output_joined_data.txt' USING PigStorage(',');
```

MOVIE RATING:

```
-- Load the ratings data
ratings = LOAD 'ratings.data' USING PigStorage('\t') AS (userID:int, movieID:int, rating:int, timestamp:int);

-- Load the movies data
movies = LOAD 'movies.item' USING PigStorage('|') AS (movieID:int, movieTitle:chararray, releaseDate:chararray,
videoReleaseDate:chararray, imdbURL:chararray, unknown:int, Action:int, Adventure:int, Animation:int, Childrens:int, Comedy:int,
Crime:int, Documentary:int, Drama:int, Fantasy:int, FilmNoir:int, Horror:int, Musical:int, Mystery:int, Romance:int, SciFi:int, Thriller:int,
War:int, Western:int);

-- Join the ratings and movies data on movieID
joined_data = JOIN ratings BY movieID, movies BY movieID;

-- Group by movie title and calculate total number of ratings for each movie
ratings_count = FOREACH (GROUP joined_data BY movies::movieTitle) GENERATE group AS movieTitle, COUNT(joined_data) AS
totalRatings;

-- Find the movie with the highest number of ratings
ordered_data = ORDER ratings_count BY totalRatings DESC;
top_movie = LIMIT ordered_data 1;

-- Store the result
STORE top_movie INTO 'output_directory' USING PigStorage(',');
```

WORD COUNT:

```
-- Load the data using PigStorage
```

```
words = LOAD 'input_file.txt' USING PigStorage(',') AS (word:chararray);

-- Use STREAM operator to invoke Python script for word count
word_counts = STREAM words THROUGH 'python_word_count.py' AS (word:chararray, count:int);

-- Group by word and count occurrences
word_group = GROUP word_counts BY word;
word_count = FOREACH word_group GENERATE group AS word, COUNT(word_counts) AS count;

-- Store the word counts
STORE word_count INTO 'output_directory' USING PigStorage(',');
```

PYTHON script:

```
#!/usr/bin/env python

import sys

current_word = None
current_count = 0

for line in sys.stdin:
    word, count = line.strip().split(',', 1)

    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_count = count
        current_word = word

if current_word == word:
    print(f"{current_word}\t{current_count}")
```