**Task 1.1:**

**Pseudocode:**  
Import the pymongo module for MongoDB interaction

Import the os module for file operations

Define the default\_directory variable with the path to the default directory

Change the current working directory to the default\_directory using os.chdir()

Connect to the MongoDB server running on localhost and port 27017 using pymongo.MongoClient()

Access the 'Assignment1' database within the MongoDB connection and assign it to the db variable

Access the 'Song' collection within the 'Assignment1' database and assign it to the collection variable

Open a new text file named 'task1\_1\_output.txt' in write mode with UTF-8 encoding using the open() function and assign it to the output\_file variable

Iterate over each document in the 'Song' collection using a loop and collection.find() method:

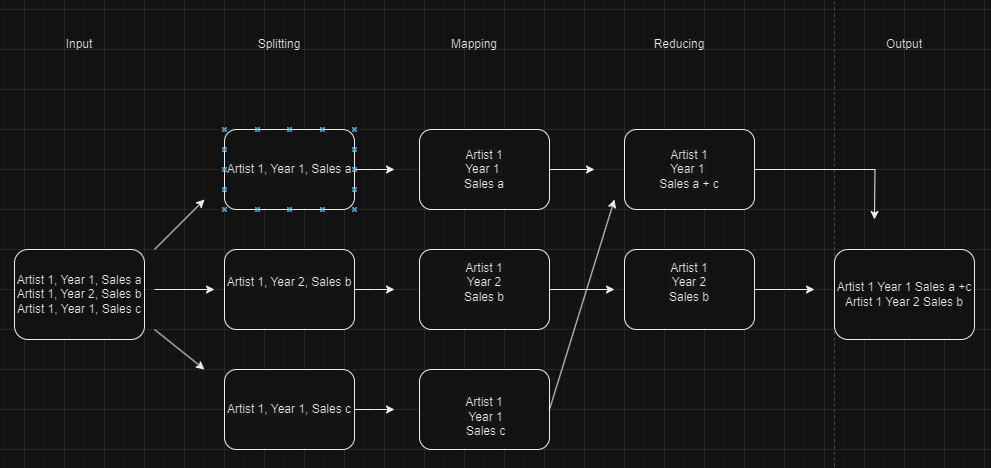
- Retrieve the 'Artist', 'Year', and 'Sales' fields from the current document using the get() method

- Construct a triplet string containing the artist, year, and sales separated by semicolons and terminated by a newline character

- Write the triplet string to the output\_file using the write() method

Close the output\_file using the close() method to ensure all data is written and the file is properly closed

**Task 1.2:**



*Figure 1: Flow Chart for Task 1.2*

**Pseudocode:**

class MRSongSales(MRJob):

method mapper(\_, line):

artist, year, sales = line.split('; ')

emit (artist, year), float(sales)

method reducer(key, values):

total\_sales = sum(values)

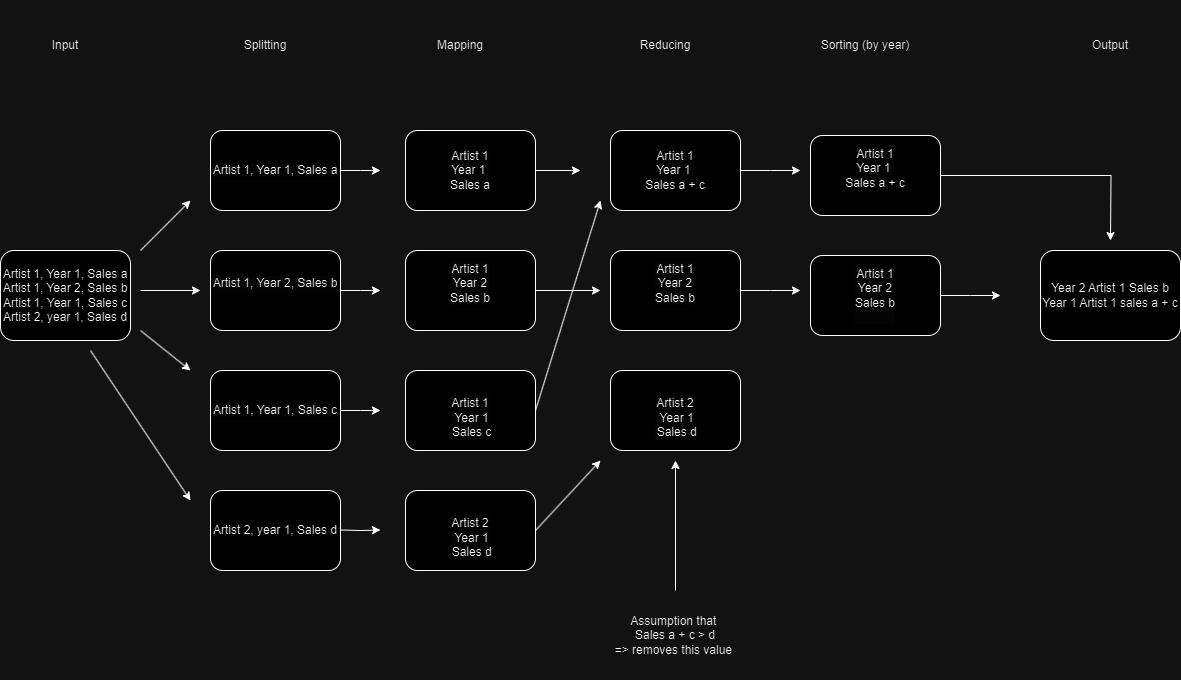
formatted\_sales = format(total\_sales, ",.3f")

emit key, formatted\_sales

if \_\_name\_\_ == '\_\_main\_\_':

MRSongSales.run()

**Task 2.1:**



*Figure 2: Flowchart for Task 2.1*

**Pseudocode:**

Import the MRJob and MRStep classes from the mrjob library

Import the re module for regular expression operations

Define a regular expression pattern to match the input lines containing artist, year, and sales information

Define a class named TopArtistByYear that inherits from MRJob

Define a method named mapper that takes self and line as input

Match the input line with the defined pattern using re.match()

If a match is found:

Extract the artist, year, and sales information from the matched groups

Convert sales to a floating-point number

Emit key-value pairs where the key is the year and the value is a tuple containing the artist and sales

Define a method named reducer that takes self, year, and artists\_sales as input

Initialize variables top\_artist and max\_sales to None and 0 respectively

For each artist, sales tuple in artists\_sales:

If sales is greater than max\_sales:

Update max\_sales to sales

Update top\_artist to artist

Emit a single key-value pair where the key is None and the value is a tuple containing the year, top\_artist, and max\_sales

Define a method named sort that takes self, \_, and year\_artist\_sales as input

Sort the year\_artist\_sales by year in descending order

For each year, artist, sales tuple in the sorted results:

Emit key-value pairs where the key is the year and the value is a tuple containing the artist and sales

Define a method named steps that returns a list of MRStep objects

The list contains two MRStep objects:

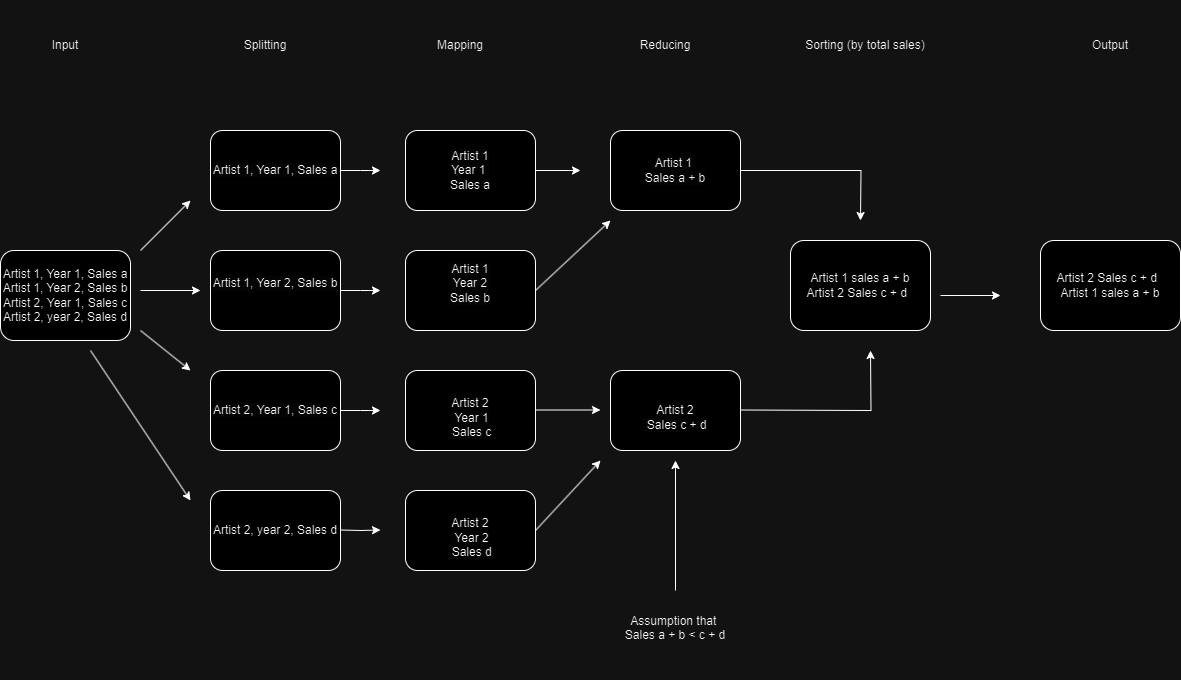
The first MRStep object with mapper as the mapper function and reducer as the reducer function

The second MRStep object with None as the reducer function and sort as the sort function

If the script is executed directly (not imported as a module):

Call the run() method of the TopArtistByYear class to execute the MapReduce job

**Task 2.2:**



*Figure 3: Flowchart for Task 2.2*

**Pseudocode:**

Import the MRJob and MRStep classes from the mrjob library

Import the re module for regular expression operations

Define a regular expression pattern to match the input lines containing artist, year, and sales information

Define a class named AlltimeArtist that inherits from MRJob

Define a method named mapper that takes self and line as input

Match the input line with the defined pattern using re.match()

If a match is found:

Extract the artist and sales information from the matched groups

Convert sales to a floating-point number

Emit key-value pairs where the key is the artist and the value is the sales

Define a method named reducer that takes self, artist, and sales as input

Calculate the total sales for each artist by summing up the sales

Yield a single key-value pair where the key is None and the value is a tuple containing the total sales and artist

Define a method named top\_5\_artists that takes self, \_, and sales\_artists as input

Initialize an empty list named top\_5

For each total\_sales, artist tuple in sales\_artists:

Append the tuple to the top\_5 list

Sort the top\_5 list in descending order based on total sales and keep only the top 5 elements

For each total\_sales, artist tuple in the top\_5 list:

Yield key-value pairs where the key is the artist and the value is the formatted total sales with three decimal places

Define a method named steps that returns a list of MRStep objects

The list contains two MRStep objects:

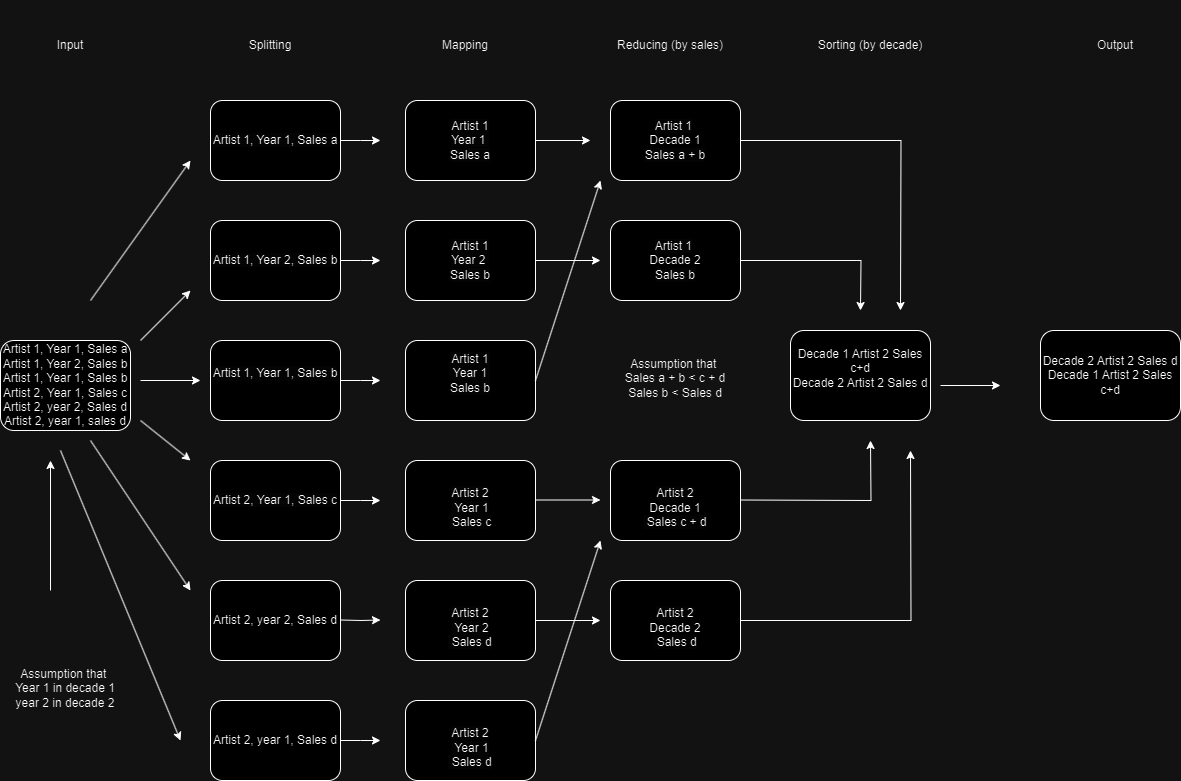
The first MRStep object with mapper as the mapper function and reducer as the reducer function

The second MRStep object with None as the reducer function and top\_5\_artists as the step function

If the script is executed directly (not imported as a module):

Call the run() method of the AlltimeArtist class to execute the MapReduce job

**Task 2.3**



*Figure 4: Flowchart for Task 2.3*

**Pseudocode:**

Import the MRJob and MRStep classes from the mrjob library

Import the re module for regular expression operations

Define a regular expression pattern to match the input lines containing artist, year, and sales information

Define a class named TopSellingArtistByDecade that inherits from MRJob

Define a method named mapper\_decade that takes self and line as input

Match the input line with the defined pattern using re.match()

If a match is found:

Extract the artist and sales information from the matched groups

Calculate the decade for the year

Generate a decade label in the format 'YYYY - YYYY' for the decade

Yield key-value pairs where the key is None and the value is a tuple containing the decade label, artist, and sales

Define a method named reducer\_sort\_decade that takes self, \_, and values as input

Sort the values by decade label in descending order

For each decade label, artist, and sales tuple in the sorted values:

Yield key-value pairs where the key is None and the value is a tuple containing the decade label, artist, and sales

Define a method named reducer\_sort\_sales that takes self, \_, and values as input

Initialize an empty dictionary named artist\_sales

Iterate over the values

For each decade, artist, and sales tuple:

If the decade is not in artist\_sales, add it as a key with a dictionary value containing the artist and sales

If the artist is not in the dictionary value for the decade, add it with its sales value

If the artist is already in the dictionary value for the decade, increment its sales value

Iterate over each decade in artist\_sales

Sort the artists by their total sales for the decade in descending order

Yield the top 3 selling artists for each decade along with their total sales

Define a method named steps that returns a list of MRStep objects

The list contains two MRStep objects:

The first MRStep object with mapper\_decade as the mapper function and reducer\_sort\_decade as the reducer function

The second MRStep object with None as the reducer function and reducer\_sort\_sales as the reducer function

If the script is executed directly (not imported as a module):

Open a file named "debug.txt" in write mode for debugging purposes

Call the run() method of the TopSellingArtistByDecade class to execute the MapReduce job

Close the debug file after running the job