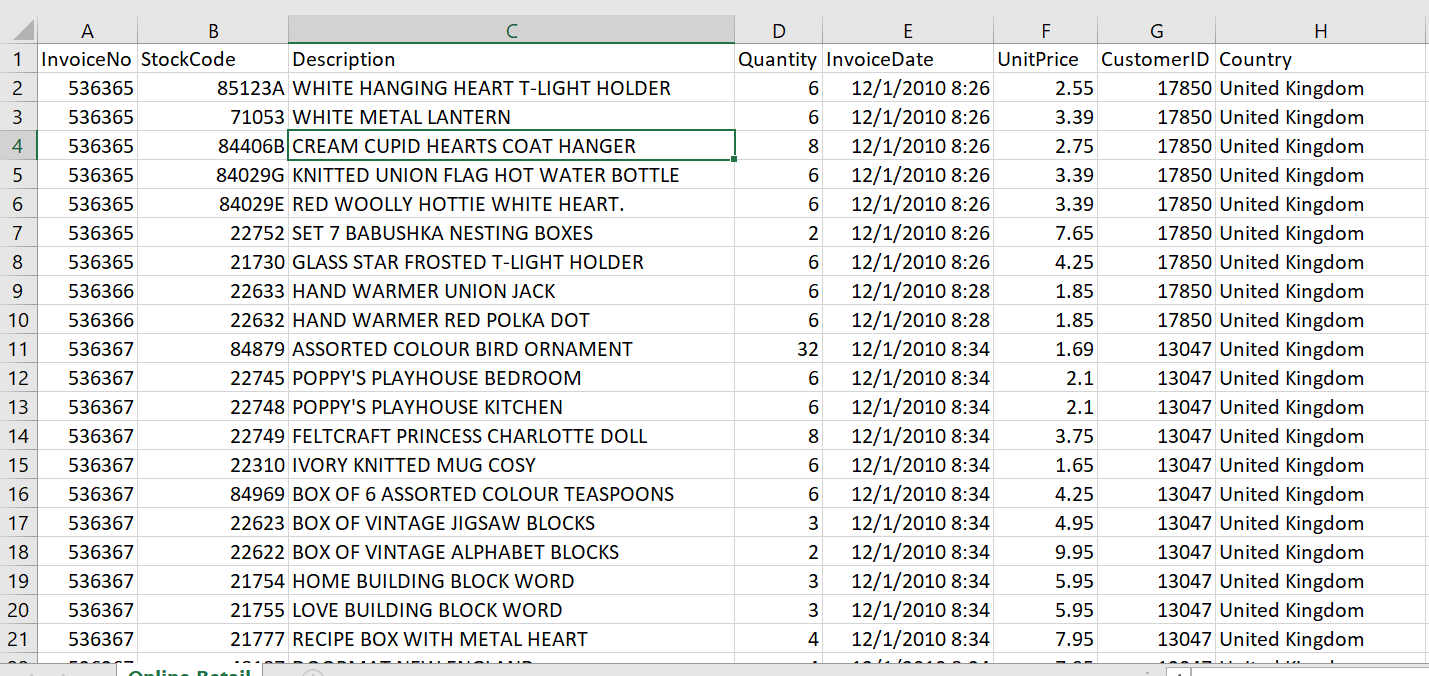
**Final Project Part 2**

**Dataset Description:**

This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The dataset by the name “Online Retail” is taken from UCI Machine Learning Respository Website- [https://archive.ics.uci.edu/ml/datasets/online+retail#](https://archive.ics.uci.edu/ml/datasets/online+retail)



**Problem Statement**: Given an item (StockCode), find the top 5 items that are frequently bought with by customers.

**Summary of Implementation:**

1. The first Mapreduce phrase is to output each Customer ID with the list of Items they bought e.g (Key: CustomerId, Value: [Item1, Item 2, Item3, Item 4, Item 4,…,ItemN] )
2. The second mapper uses the input from the first Mapreduce phrase and implement Stripe design pattern to group together pairs into an associative array (Hash Table).

e.g Key: Item1, Value: { (Item 2,1), (Item 3,1), (Item 4,2) }

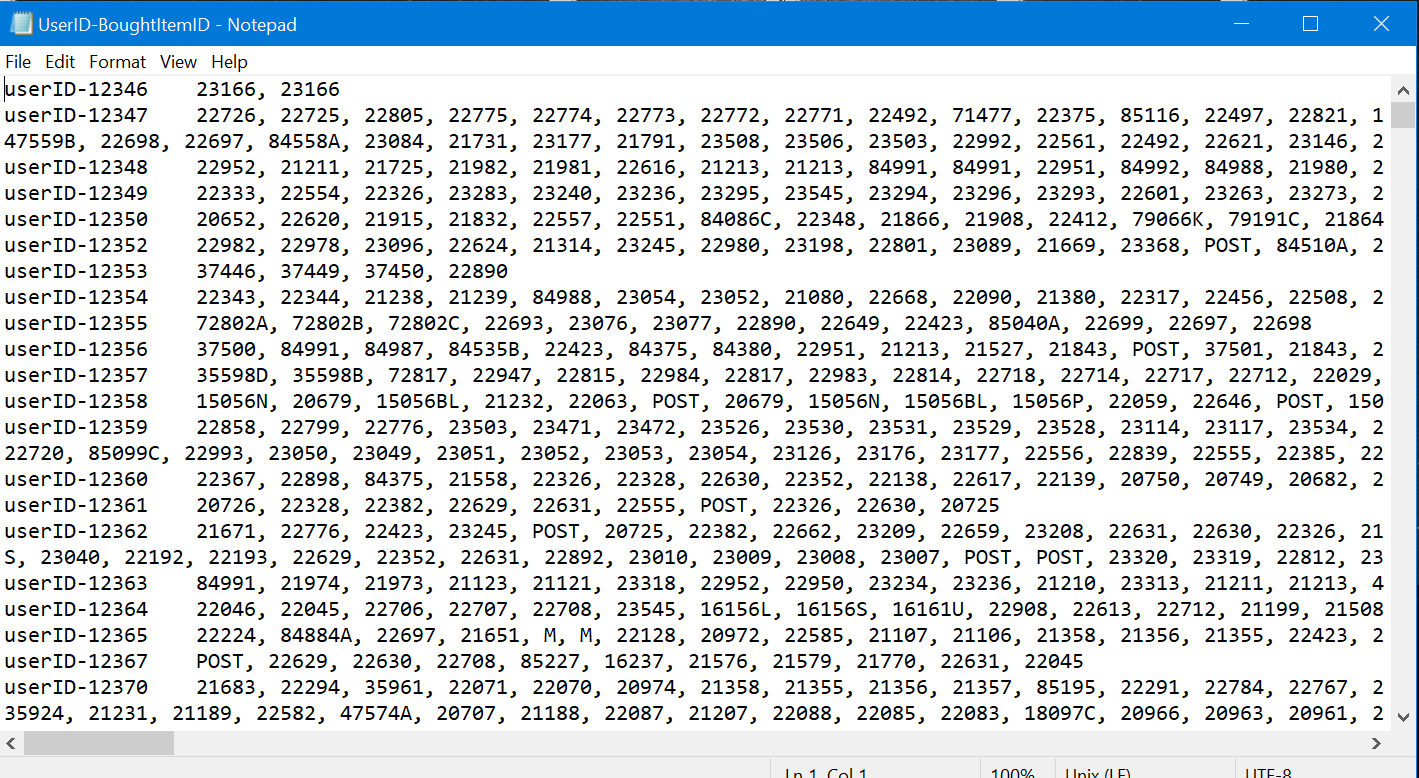
1. In the second reducer, the input contains an Item key with a list of hash tables. Firstly, create a new HashMap object. Then, fill in the object’s key with the keys present in each hash table in the list and the corresponding value equals the sum of all the hashtable in the list that share the same key. Finally, determine the top 5 common items associated with the key Item using the Treemap data structure.

**Code Screenshots:**

public class Mapper extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text,Text,Text> {  
 @Override  
 protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 String line = value.toString();  
 String[] record = line.split(",");  
 if(key.get() != 0 && record.length == 8 && record[6].matches("(\\d){5}") ) {  
  
 String stockCode = record[1];  
 String customerID = record[6];  
 context.write(new Text(customerID),new Text(stockCode));  
 }  
 }  
}

public class Reducer extends org.apache.hadoop.mapreduce.Reducer<Text, Text,Text, Text> {  
 @Override  
 protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {  
 StringBuffer sb = new StringBuffer();  
  
 for(Text t : values){  
 sb.append(t.toString()+", ");  
 }  
 sb.deleteCharAt(sb.length()-2);  
 String items = sb.toString();  
 String userid="userID-"+ key.toString() ;  
 context.write(new Text(userid), new Text(items));

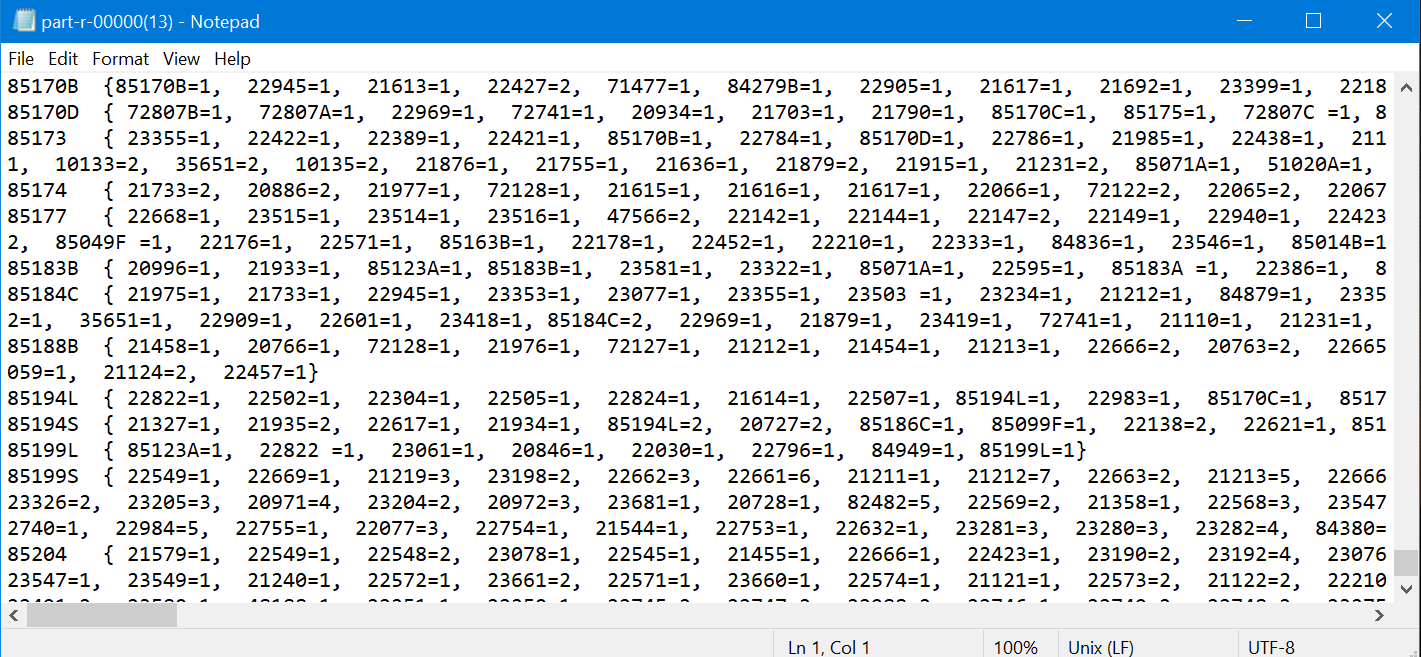
//(Key: CustomerId, Value: [Item1, Item 2, Item3, Item 4, Item 4,…,ItemN] )  
 }  
}



public class mapper2 extends Mapper<LongWritable, Text,Text,MapWritable> {  
 @Override  
 protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 String line = value.toString();  
 String[] record = line.split("\t");  
 String subRecord = record[1];  
 String[] items = subRecord.split(","); // {item1,item3,item5,item3,…}  
  
 for(String item: items){ // item1

Text itemKey = new Text(item);  
 MapWritable map = new MapWritable(); //map<writable,writable> ={}  
  
 for(String j: items){ //j = item1  
 Text current = new Text(j);   
  
 if(map.containsKey(current)){

int k=Integer.*parseInt*(map.get(current).toString());  
 map.put(current,new IntWritable(k+1));  
 } else{  
 map.put(current,new IntWritable(1));   
  
 }  
  
 }  
 context.write(itemKey,map);  
 }  
 }  
}



public class reducer2 extends Reducer<Text, MapWritable,Text,Text> {  
 //{item1,{item1:1,item2:1,item3:5}, {item3:2,item4:1,item2:4}, {item2:2,item3:1}}  
 MapWritable mw= new MapWritable();  
 TreeMap<Integer, Text> treeMap = new TreeMap();  
 Text output = new Text();

@Override  
 protected void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedException {  
  
 for(MapWritable cur: values){

for(Map.Entry<Writable,Writable> entry : cur.entrySet()){

if(mw.containsKey(entry.getKey())){  
 IntWritable entryValue= (IntWritable) entry.getValue();  
 IntWritable mwValue = (IntWritable) mw.get(entry.getKey());  
  
 int sumValue = Integer.*parseInt*(entryValue.toString()) + Integer.*parseInt*(mwValue.toString());

IntWritable valueResult = new IntWritable(sumValue);  
  
 mw.put(entry.getKey(),valueResult);  
 }  
 else {  
 mw.put(entry.getKey(), entry.getValue());  
 }  
 }  
 }

// adding each entry from the MapWritable object to the Treemap object  
 for(Map.Entry<Writable,Writable> curr : mw.entrySet()){  
 IntWritable i = (IntWritable)curr.getValue();  
 int value = Integer.*parseInt*(i.toString());  
 treeMap.put(value,new Text(curr.getKey().toString()));  
 if(treeMap.size()>5){  
 treeMap.remove(treeMap.firstKey());  
 }  
 }  
  
 StringBuffer sb = new StringBuffer("Customers Who Bought This Item Also Bought: ");  
 for(Text t: treeMap.values()){  
 sb.append(t.toString() +",");  
 }  
 output.set(sb.toString());  
  
 context.write(key,output);  
  
 }

public class Driver {  
 public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {  
 Configuration conf =new Configuration();  
 Job job = Job.*getInstance*(conf,"job1");  
 job.setJarByClass(Driver.class);  
  
 FileInputFormat.*addInputPath*(job,new Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job,new Path(args[1]));  
  
 job.setMapOutputKeyClass(Text.class);  
 job.setMapOutputValueClass(Text.class);  
  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(Text.class);  
  
 job.setMapperClass(Mapper.class);  
 job.setReducerClass(Reducer.class);  
  
 job.waitForCompletion(true);  
  
 Configuration conf2 = new Configuration();  
 Job job2 = Job.*getInstance*(conf2,"job2");  
 job2.setJarByClass(Driver.class);  
  
 job2.setMapperClass(mapper2.class);  
 job2.setReducerClass(reducerTemp.class);  
  
 job2.setOutputKeyClass(Text.class);  
 job2.setOutputValueClass(MapWritable.class);  
  
 job2.setMapOutputKeyClass(Text.class);  
 job2.setMapOutputValueClass(MapWritable.class);  
  
 FileInputFormat.*addInputPath*(job2,new Path(args[1] + "/part-r-00000"));  
 FileOutputFormat.*setOutputPath*(job2,new Path(args[2]));  
 System.*exit*(job2.waitForCompletion(true)?0:1);  
 }

