COL761
Assignment1

Vigneshwar – 2020CS10344
Rutvik – 2020CS10420
Sainath – 2020CS50421

contributions: All of us have contributed equally(33% each)

Algorithm:
We read the file and store it as vector<vector<string>>.

```
std::ifstream inputFile(path_to_dataset);
if (!inputFile.is_open())
{
    std::cerr << "Failed to open the file." << std::endl;
}

std::vector<std::vector<std::string>> transactions;
std::string line;
while (std::getline(inputFile, line))
{
    std::vector<std::string> tokens;
    std::istringstream tokenStream(line);
    std::string token;

    while (tokenStream >> token)
    {
        tokens.push_back(token);
    }

    sort(tokens.begin(), tokens.end(), numericComparator);
    max_item = max(max_item, stoll(tokens[tokens.size() - 1]));
    transactions.push_back(tokens);
}
inputFile.close();
```

We are using FP-Tree algorithm to find frequent items. We are setting the threshold based on mean and standard deviation of frequency of single itemsets.

```
uint64_t threshold = 2500;
unordered_map<Item, Transaction> pre_decoder = {};
unordered_map<Item, Transaction> decoder = Compress(transactions, threshold, pre_decoder);
```

Now we sort the frequent items based on length of the item. We take a transaction and loop through the frequent itemsets and replace them in the transaction with their labels.

```
vector<string> new_transaction = {};
while (i < transaction.size() && j < itemset.first.size())
{
    if (transaction[i] == itemset.first[j])
    {
        i += 1;
        j += 1;
    }
    else
    {
        try
        {
            if (stoi(transaction[i]) > stoi(itemset.first[j]))
            {
                break;
            }
        }
        catch (const std::invalid_argument &e)
        {
            break;
        }
        new_transaction.push_back(transaction[i]);
        i += 1;
    }
}
```

We do this recursively by changing the support threshold dynamically and compressing it at each stage untill threshold drops to a defined minimum value which is max(100,mean-SD/2) in our case.

```
unordered_map<Item, Transaction> Compress(vector<Transaction> &transactions, uint64_t threshold, unc
{
    const FPTree fptree{transactions, threshold};
    const std::set<Pattern> patterns = fptree_growth(fptree);

    vector<pair<Transaction, int64_t>> freq_items = {};
    for (auto i : patterns)
    {
        if (i.first.size() > 1)
        {
            vector<string> trans = {};
            for (auto j : i.first)
            {
                trans.push_back(j);
            }
            sort(trans.begin(), trans.end(), numericComparator);
            freq_items.push_back({trans, i.second});
        }
    }

    sort(freq_items.begin(), freq_items.end(), compare);

    std::string freqItemsFileName = "freq_items_1500.txt";
    std::ofstream freqItemsFile(freqItemsFileName, std::ios::app);
    if (!freqItemsFile.is_open())
    {
        std::cout << "Failed to open the output file." << std::endl;
    }

    for (const auto &i : freq_items)
    {
        const std::vector<std::string> &strings = i.first;
        int intValue = i.second;

        for (const std::string &str : strings)
        {
            freqItemsFile << str << " ";
        }

        freqItemsFile << "--- " << intValue << "\n";
    }
    freqItemsFile << "------------------------------------------------------ "
                  << "\n";
    freqItemsFile.close();

    unordered_map<Item, Transaction> decoder;
    for (auto &transaction : transactions)
    {
        for (auto &itemset : freq_items)
```

In the compressed file we have kept mapping at the start and compressed transaction after this with $$$ as a seperator.

For decompression we just took the mappings and replaced them to get the original file.

References: For FP-tree implementation we have taken this code.