Summer 2017
MIS 6V99 – Special Topics – Programming for Data Science
Programming Assignment #3
Market Basket Analytics – given a purchasing history of products together, recommend additional product to customers making purchases
Kevin R. Crook

Scenario

A new startup company has been selling their products online with several million sales transactions. They have hired you as a data scientist to design a prototype of a market basket analytics system. The system will look at the products the customer has placed in their online shopping cart and recommend another product.

The company has provided us with a training set of 1 million sales of 2 or more products.

For simplicity for this first prototype, they have:

- Limited their training set to a maximum of 4 products per sales transaction
- Limited individual sales to no more than 1 of each product
- Temporal reasoning should not be considered (time and date of the sale should not be considered)

The company only has 10 products. The products are named P01, P02, …, P10. Some products may be new without any sales yet.

The company has provided us with a test set of 100 online shopping carts, and has asked us to recommend 1 additional product for each of the 100 online shopping carts.

When considering the training set, previous purchases of 4 products should be considered most influential, followed by 3 and then 2.

Download the training set of 1 million sales transactions

Your Python program should download the training set from the following link:
http://kevincrook.com/utd/market_basket_training.txt

You Python program should load and analyze this training set in order to recommend products for the test set.

The training set has no header record.

The training set has 1 million records. Each record should be considered an historical sales transaction. Each record starts with a line number (starting with 0000001 and ending with 1000000), followed by a comma separated products list. There will be 2 or 3 or 4 products per record.

Download the test set of 100 online shopping carts

Your Python program should download the test set from the following link:
http://kevincrook.com/utd/market_basket_test.txt

You Python program should load this test set, apply the analytics from the training set, and recommend 1 product for each shopping cart.

The test set has no header record.

The test set has 100 records. Each record should be considered an online shopping cart. Each record starts with a line number (starting with 001 and ending with 100), followed by comma separated products in the shopping cart. There will be 1 or 2 or 3 products per shopping cart.

Create the recommendations file

You Python program will create a file of recommendations in the local directory called
market_basket_recommendations.txt

Do not create a header record for this file.

The file must be a proper text file using utf-8 encoding, with each line (including the last line) properly terminated by a machine independent end of line character.

Each line will be 1 recommendation. The line should start with the line number from the test file. Line numbers should all be 3 digits, with leading zeroes if necessary (starting with 001 and ending with 100). Follow the line number with a comma. Follow the comma with the recommended product. Follow the recommended product with an end of line. No spaces anywhere in the file!

Your Python code must be algorithmic in nature

Your Python code must be algorithmic in nature.

Hardcoding output statements that are not algorithmic in nature is considered cheating and is explicitly listed as an act of academic dishonesty in UTD official regulations, with possible referrals for academic dishonesty.

Individual Assignment

This assignment in an individual assignment. You may consult with other student about general approaches to solving the problem and for asking for help to resolve stack traces, but all coding must be your own work. An electronic comparison for similarities in submissions will be made. Any similarities greater than 70% will be investigated by the instructor, with possible referrals for academic dishonesty.

Auto Grader

All directory, file, and other names must be spelled exactly as given, case sensitive. All outputs must be in the correct format, also case sensitive. The reason for this is that the auto grader will look for directories, files, etc. based on an exact spelling, case sensitive, and if it is not found it will not be run nor graded.

If the final source does not run to completion without a stack trace, no credit for the assignment will be given. The source code will be run in an Anaconda Python 3 sandbox using the release available on the first day of class. Unless explicitly specified otherwise, all files should be created in the local directory without any device names nor path names. If a subdirectory is specified, it must be created using relative pathnames and using the machine independent functions of Python to join directory names.

Only source code properly checked into GitHub will be run for grading. Only output files from the auto grader run of the program will be considered for grading. Student submitted output files will not be considered.

GitHub Repository ("repo")

You must create a GitHub repository called **mis_6v99_2017_summer** as a private repository and grant access to the instructor account **kevin-crook-ucb**. You must create a directory in the repository called **assignment_03**, with 1 and only 1 read me file (either **README.txt** or **README.md**, but not both) with at least 1 line of meaningful comment, and place the **market_basket_analytics.py** file in that directory.

Only source code properly checked into GitHub will be considered for grading. The time of check into GitHub of source code will be the determining factor where time limits are considered.

Python Program

You will write a single file Python program, **market_basket_analytics.py**, to accomplish them. The program must download and read all files correctly. The program must run without stack trace. The program must create the specified output files. Only output files created from the instructor's run of your source code can be considered for grading.

Documentation Strings and Ratio of Source Code to Comments

In your Python code all functions, classes, and methods should have a documentation string with at least 1 line of meaningful documentation. The ratio of non-empty source code lines to comments should be no more than 5 to 1. Documentation strings do not count as comments.

Grading Rubrics

Basic Criteria	Points
GitHub private repository was created correctly, instructor's account was given permissions, directory & files created correctly with exact names given, program runs without stack trace, and recommended products match instructor's solution	0 to 100 points (1 per correct product recommendation)

Programming Productivity Points To be eligible for programming productivity points, your submission of source code in GitHub must meet all of the following criteria: • Final submission must be on time. Late submissions are not eligible. • Final submission must meet all of the basic criteria. • First submission of source code must have been made within a couple of days of the date the assignment was given. • Updates to source code should be checked frequently, every couple of days, until the final submission is made. • A cumulative total of points earned for all of the daily runs of the auto grader will be used to determine the productivity ranking points. Auto grader will be run once per day in the early morning hours.	Points
Top 10%	10
Next 10%	9
Next 10%	8
Next 10%	7
Next 10%	6
Next 10%	5
Next 10%	4
Next 10%	3
Next 10%	2
Next 10%	1

Timing of Submission for Rank Grading

The submission time will be considered in the tie breaker for rank grading. Completing the assignment sooner may give you a high rank for the semester.

Late Penalty

25% per day or fraction of a day. 1 second counts as a fraction of a day.

Technical Difficulties with GitHub Submission

If you encounter any technical difficulties with the GitHub submission, in order to preserve your submission date and time, you must have done all of the following:

- GitHub has a 99.9% uptime. Any claims of technical difficulties should be rare.
- You must demonstrate a substantial work history – you must demonstrate that you did not wait until a few days before the due date to get started:
    - You must have checked your first version of the source code into GitHub within a couple of days of the day the assignment is given
    - You must have checked in new source code at least every couple of days
    - Your most recent check in of code should have rufn without stack trace, and demonstrate at least 80% of the minimum required functionality
- You must check in source code as soon as possible after the submission difficulty.
- Remember that your local GitHub tracks all updates to your file and stored these in GitHub, which the instructor as collaborator can access. The instructor will look at the local GitHub history of the file. If it shows any work was done after the claim of technical difficulty, the matter will be considered a violation of academic dishonesty.
- Immediately (within 1 minute) take a screen print which clearly shows the error you received on submission and also shows the date and time from your desktop clock.
- Email your instructor within 5 minutes of the error with the screen print attached. You must also attach source code (do not attach output files).
- If any of these conditions are not met, it will not be considered.