Soft152: Software Engineering 2017 - 18

COURSEWORK

This assignment will contribute 70% towards the overall module mark.

There are two options for this coursework. Choose only one, both carry the same marks.

Option 1: Data Miner

In brief, the purpose of this application is to read files containing anonymised details of online sales and to analyse and present summary descriptive statistics of those sales.

The original dataset has been obtained from here:

http://archive.ics.uci.edu/ml/datasets/Online+Retail

However, the original dataset was very large (circa 24 Mb) and contained approximately 541,000 records. There were a number of records not relevant for this assignment (e.g. details of goods returned being indicated by a negative quantity). These records have been removed and data generally 'cleaned'.

The size of the datasets has been further reduced by only considering one year's worth of data (2011). This has reduced the file size to approximately 17 Mb and some 370,000 records. To enable testing of algorithms this dataset has been reduced in size again.

The size reduction was achieved by assigning a random number to each of the rows and then ascending sorting the data by the random column and then choosing the first n records as shown below:

File name	Number records, n
Online Retail 2011 random A.xlsx / .csv	10,000
Online Retail 2011 random B.xlsx / .csv	5,000
Online Retail 2011 random C.xlsx / .csv	1,000
Online Retail 2011 random D.xlsx / .csv	500
Online Retail 2011 random E.xlsx / .csv	100
Online Retail 2011 only .xlsx / .csv	370,000

Table 1: List of retail data files

As shown in the table above the data is provided in two data formats: An Excel spreadsheet and a comma separated file (csv) format[1]

The csv files are the ones which you will load into your application. **Do not edit or amend the csv files as it may cause file reading errors.**

[1] Note a .csv file is just a text file where components of the file are separated by commas. A .csv file can be treated as a text file: Can be opened and modified in standard text editors, e.g. Notepad or WordPad.

In addition to these .csv files there are standard Excel (.xlsx) versions you can use to perform some data analysis and use to verify / test the output of your application. It is *not* expected nor necessary that your application is able to read these Excel files.

For the excel files please see the Excel data folder also in the Assignment section on the DLE

Each data file is of exactly the same format with column headers shown below

Invoice No	Stock Code	Description	Quantity	Invoice Date	Unit Price	Customer ID

The column headers should be self-explanatory, but here are some notes:

Invoice number: A unique 6-digit integral number. Each invoice can contain one or many item types being bought. Where an invoice contains many items, the same invoice number will appear in several rows, (this is observed more in the larger datasets).

Stock code: This is a unique code to identify one stock item type. Although typically an integral number, it may also contain alpha-numeric characters. The stock code is unique to each item available.

Description: A short textual description of the stock item. The description will be unique to each stock code (i.e. there is a one-to-one relationship between stock code and description).

Quantity: the number of items purchased of a particular stock item on that invoice.

Invoice date: the date in dd/mm/yyyy hh:mm format when the invoice was generated. The time is given in 24 hour format

Unit Price: The price of one item in pounds (£)

Customer ID: a unique number for each customer. Each invoice will be associated with exactly one customer, and while generally one customer is associated with only one invoice, there are cases when a customer has raised more than one invoice. The customer ID is always represented by 5 numeric characters.

The data files can be found in the .csv data file folder in the Assignment section on the DLE. As mentioned, please do not modify the .csv files to work with your submission. When marking the original files will be used and it may result in the files not being read properly.

Data Analysis:

The application you need to write should be able to read any one of the A to E data files and provide simple descriptive statistics represented as bar charts together with simple text based output of part of the data.

It is not essential that your submission is able to process the main data file, 'Online Retail 2011 only.csv', but it would be good if it could.

The actual structure and layout of the interface will be left for you to decide. But you will need to include:

- A means to launch a file dialogue box allowing the any of the .csv files to be opened. For example, this may be via a File – Open menu or a button.

- A panel on which will display as a bar chart the monthly summary statistics as given below

    1. Total quantity of items sold per calendar month.

    2. Total value, in pounds[2], of items sold per calendar moth. Note this is the total of quantity * unit price for each month.

    3. Total number of unique customers per month[3].

    4. Total number of invoices generated per month.

    - Also in terms of averages:

        a) Average number of items per invoice per calendar month

        b) Average spend, in pounds, per customer calendar per month

        c) Average spend, in pounds, per invoice per calendar month

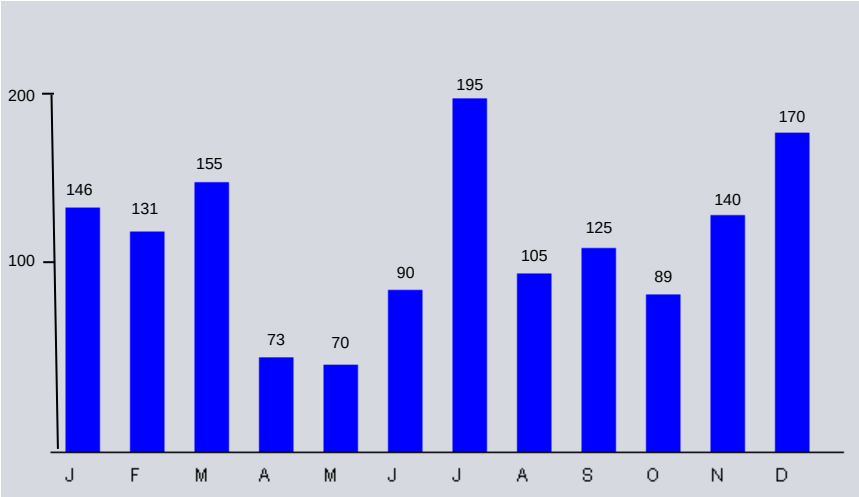
    - For each of the above, 1 to 4 and a) to c), appropriate controls allowing the user to switch which data is being displayed on the bar chart should be provided.

Please see next page →

[2] It all cases where cost, value, or price is stated in pounds it is assumed that the relevant amount of pence is also given (e.g. £145.57). In other words, the figure is NOT rounded to whole pounds.
[3] In some months, a customer may have bought goods on two, or more, different days. In these cases, the same customer ID would have been used and as such counts as only 1 unique customer.

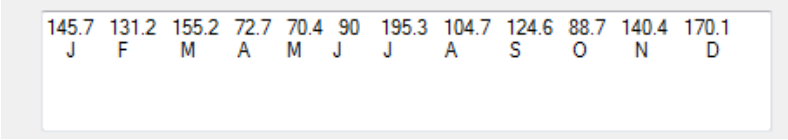
- In addition to the graphical results provide components where the user can interrogate the data by entering in either a customer ID or an invoice ID into a textbox:
    - All the details of sales of that customer[4] or invoices are then displayed in a multi-line textbox.
    - If an ID is entered which does not match that in the data, then a simple message is displayed informing of no match. A relevant component will also need to be provided by which the user is able to indicate whether a customer or invoice ID is being entered.

The format of the chart should be similar to that shown below:



The base of each bar on the x-axis should show the month letter. The y-axis should be shown with labels showing the range of values of the bars. The panel and bar chart should re-size proportionally when the form is resized.

Also include some display of the values for each month. This can be either values drawn at the top of each bar, as shown above, or in a multi-line text box as shown below:



If displaying the values on the graph, then round to the nearest whole number, if though displaying in a multi-line text area then show to 1 decimal place

[4] The same customer may have purchased on serval occasions across a year. All the sales details of that customer should be displayed.

Option 2: Ants collecting Food

In the domain of Artificial Life (ALife) the behaviour of social insects has been used to study how simple creatures can be used to solve complex problems.

While individually insects are behaviourally unsophisticated, they collectively can solve complex problems. For example, 'real world' ants can:

- Form bridges by chaining their own bodies
- Build and protect nest
- Sort brood and food items
- Co-operate in carrying large items
- Find shortest route to food source

It is this collective 'intelligent' behaviour which has led to many studies and simulations. The following involves a group of ants collecting food and returning the food to a nest is just one possibility.

The ants live in a 2D continuous (i.e. non-grid based) world.

At the start, there is no food or nest in the world and a number of ants are created at random positions in the world. These ants do not where food is, nor do they know where their nest is. Their eventual role is to find food and carry it back to a nest. The ants move randomly until they:

- **Either** encounter another ant who knows were food or a nest is, in which case they ask the ant for the information they need. What happens now depends on if they have food or not. If they have food they steer towards their nest, if they do not have food they steer towards the food.

- **Or** they walk close to a food or a nest.

In either case they remember the location of the food and / or the nest.

Once they deposit food at the nest they steer back to where they 'remember' where the food is.

If they arrive where they thought food was and none is there (because other ants have picked it all up), they forget were the food was and now move randomly.

Each ant will need to have a 'radius' within which they can detect food / nest, and detect other ants who may have information about food / nest. These two radii may be different.

At any given time, there may be more than one ant in the radius. You may decide to only ask the closest ant or all of them in order to find out where the food was.

Allow the user to deposit food using mouse clicks. Assume one click leaves n-units of food (the user can place several bits of food in close proximity) and each time an ant picks up food it picks up m units (also m is much less than n, i.e. m << n), so the food will eventually disappear.

Also allow the user to place nests in the world via mouse clicks.

Assume the ant is a bit forgetful: It may forget where its nest was or forget where the food was (or both!) and as outlined above will need to either stumble across it or ask another ant.

Extra credit:

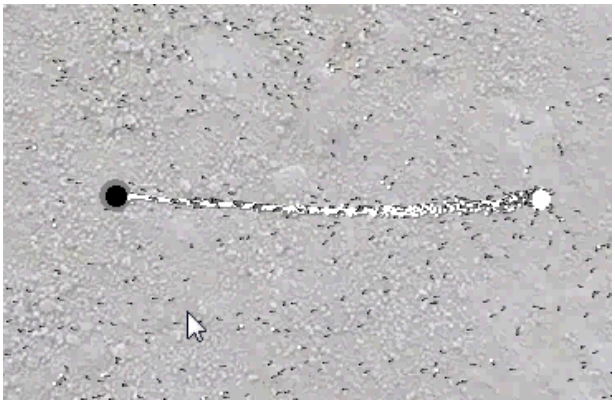
Allow for a separate population (or populations) of ants who steal the food off the other ants. These robber ants have their own nest and do not get food from the food piles, but move randomly until they find an ant with food. They can remember where this ant is. The chances are the ant with food is in a chain of ants carrying food and thus there is a good chance returning to that position

Some screen shots of an application is shown below

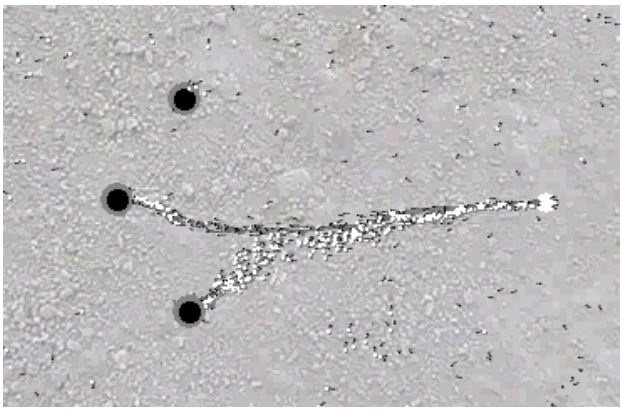
[Note the algorithm is slightly different to that given above, but should elicit similar behaviour]

Start of simulation nest (in black) and food (in white). The ants are the tiny black dots.

Ants begin to swarm to food and carry food back to the nest



Definite path between food and nest emerging



User added another nest, now some agents choosing the new nest

A short video of this simulation is in the DLE -> Assignment section

To help with this option a Visual Studio has been provided. This contains a basic steering class and a simple 2D vector class. This project and associated steering / vector classes *must* be used when completing this option.

Please see SOFT152AssignmentOption2.zip file in the Assessment section on the SOFT152 DLE .

This project uses a .dll to provide the steering and vector classes. It will need .NET framework of 4.5.2 or greater to run. If Visual Studio 2015 or 2017 is installed the this version of the .NET framework should be present.

Notes and constraints for both assignments:

- Do not use any C# chart components for drawing graphs. For the bars use the standard graphics methods of DrawRectangle() or FillRectangle().

  https://msdn.microsoft.com/en-us/library/system.drawing.graphics%28v=vs.110%29.aspx

- Do not use any Language-Integrated Query (LINQ) type queries or techniques in your solution.

- If using List<T> then do not use *any* of the built in List<T> manipulation methods. The **only** methods allowed are, those covered on the module, to clear, add, insert or remove, access elements from the List<T>, namely

  - Clear()
  - Insert()
  - Add()
  - Remove()
  - RemoveAt()
  - ToArray()
  - ElementAt()
  - The .Count property to determine the number of items in the List<T>

- Similarly if using arrays, then only the .Length property may be used. Any of the built in array methods (e.g. Sort(), Reverse() etc), should not be used.

- Do not hard code an absolute path to the folder which contains the data files. An example of an absolute path is something like:

  C:\Users\<user name>\ Documents\Data\ Online Retail 2011 random B.csv

- Particularly If attempting option 2, the supplied Visual Studio project and associated steering / vector classes must be used.

There are marks penalties for using any of the above, please see the 'Constraints Penalty' section

It is anticipated a class based approach as used on the module will be adopted, this is the ideal approach and stands the best chance to gain a higher mark.

If however, there is just one form class with a series of parallel arrays or lists, then it is very unlikely this will attract a mark higher than 59%.

Deliverables: There are **two** deliverables for this assignment, a report, and the actual coded solution.

1. **The Report:** The report will be in **two** sections:
   a) Describe your solution to the problem:
      a. Describe the classes used – their properties and methods include a suitably annotated UML class diagram.
      b. Description / explanation of use of any data structures used, e.g. List<T>
      c. Any enhancements or features included which were NOT in the original

   b) An evaluation section (MIN 300 words; MAX 600 words):
      a. A critical evaluation of how successful you believe your application actually is, e.g. any bugs (there is always one!), things that do not work perfectly / as hoped.

      b. With the benefit of hindsight what you may have done differently **in terms of coding or the design of classes / relationships between classes / data structures.**

2. The actual coded solution. This is the Visual Studio project containing all the C# code needed to compile and run your solution.

Notes for section A:

Section A is meant to provide a fairly high level, abstract, description of the algorithms used and general structure of the solution. Thus the section should contain sufficient detail, and no more, to explain to a user the:

- Structure of the solution;
- Rationale of why approaches were taken;
- Where appropriate any parameters or assumptions made.

Adopt an academic writing style of being concise yet informative. While this can be quite challenging it is encouraged time is spent time focussing on this. Generally long verbose reports which laboriously describe / explain every facet of the solution in detail is likely to lose marks rather than gain.

Write in the passive third person form. No first person personal pronouns (e.g. 'I', 'me', 'my', etc.).

Avoid a 'diary style' or 'blog style' of writing, e.g. "firstly this was done, then was able to do that and finally was able to do …."

It is expected basic rules of report writing will be adopted with appropriate punctuation, use of sentence and paragraphs. However use of bullet points will be acceptable, but this should not be the totality of the submission.

Cutting and pasting of sections of code is not acceptable as a description of the structure or approach adopted: It does not add anything to that already

presented in the documented coded solution. Any cut and pasted code, or C# code in general, which appears in report will be ignored.

Describe any significant methods used in terms of an algorithm expressed as pseudo-code. There is no requirement, or need even, to show the steps in decomposition (i.e. from top level through to lower levels).

Any algorithms presented in section A should be those developed prior to coding, and not that reverse engineered once a working program has been obtained.

In terms of coding, generally will be looking for what can be termed as good programming and professional practice:

Modularity

- As a minimum, an appropriate demonstration of modularity by use of methods.
- A better approach would be appropriate use of a class(es) other than the default one (i.e. the form class)

Adherence to OO principles (information hiding / encapsulation)

- Correct use of access modifiers (e.g. private, public) for both instance variables and methods.
- Appropriate use of getters / setters (e.g. not giving all instance variables both read / write access by default)

Code Complexity / readability

- Limited use of 'magic numbers'
- Preference for high cohesive methods, (e.g. those which perform a single conceptual task)
- Appropriate variable, field / property, method, class and namespace names conforming to C# conventions
- Appropriate naming **all**[5] of components, e.g. do not use 'form1', 'button1' etc.

Documenting

- Appropriate descriptive comments for all classes used
- Appropriate comments for all instance variables.
- Appropriate comments describing all methods (event handler methods can usually be excluded as can getter / setter methods)
- Inline commenting where appropriate.
- All commenting should comply with C# convention.

Do not be tempted to provide over engineered solutions. Or 'clever' programming tricks.

It goes without saying do make sure any data being displayed is actually correct. Use the excel spreadsheets to validate the data analysis. There is no need to perform the same data analysis on every spreadsheet: If your program output matches two versions of the data, then it is a reasonable to assume it will be correct on the others.

Any additions to original specification will only be considered when the original specification is fully completed and works effectively. Otherwise any additions / enhancements will be ignored.

[5] The **only** exception to this are labels which do not contain editable text (e.g. a label used to describe a text box)

Marking Guide:

	Mark
Report Appropriate description UML diagram Object interaction	40
Evaluation Realistic suggestions / improvements in code / design	10
Program implementation Criteria: C# coding conventions in method / field / property names. Appropriate use of modularised code, e.g. methods and classes. Succinctness and clarity of code. Use of basic OO concepts (e.g. information hiding, appropriate use of public / private and 'getters and setters'). Use of built-in types / arrays / List<T>(where needed) Error handing / input validation	50
Total marks	100

This assignment will contribute 70% towards the overall module mark

This is a core module for Computer Science / Computing and Games Development, if the submitted application does NOT compile it will not be considered for a pass (the mark will be capped at under 30%).This may be due to either an incomplete Visual Studio solution / project and / or errors in code preventing compilation. If the latter is the case, comment out whatever code is causing the error.

Submission deadline: **09:00 Monday January 15th, 2018**

Return of marked work: Assignment will be marked by Monday 12th February 2018

Assed learning outcomes:

This assignment measures all four learning outcomes: LO1, LO2, LO3, and LO4.

[Please see the module overview or the 'Aims & Learning Outcomes' section on the SOFT152 DLE]

Constraints Penalty:

Unfortunately, each year some assignments are submitted, which either have some parts missing or which do not address the constraints in the specification. To prevent this, the following marks will be LOST according to:

Reason	Marks LOST
Missing or Visual Studio project structure (e.g. only the .cs files submitted)	30
Using a chart component instead of using the built in Graphics methods	30
Using any LINQ approaches or techniques.	30
Using any of the List<T> or array methods as described	10 marks per usage
For option 2, not exclusively using the supplied steering and vector classes	30

For this assignment you may optionally work in pairs if you wish. Or you may work and submit an individual assignment. There are no mark differences between working individually or in pairs per se.

If you decide to work in pairs then you need to stay in that pair and submit a combined effort. If you start working in a pair then decide to work individually, then each individual solution must be wholly unique. If identical or very similar sections of code / algorithm are submitted by two or more individuals as individual submissions then all those submission could be considered as plagiarism. Please see the plagiarism section on page 12 of this document. So please do consider carefully before you work as a pair.

Please place your report in a folder within the Visual Studio project directory. Zip that entire directory and submit **via DLE. If working as a pair then both parties need to submit their submission under their own name[6], but with the report suitably labelled (see below).**

Do not submit any of the data files as only the originals will be used for testing submissions.

The report should have a cover page and ideally a table of contents. The cover page should contain information to identify the authors. This should be at least student ID then, optionally, name.

If working in pairs then the identity of BOTH the authors should be on the cover page. If identical (or closely similar) work is submitted individually by two or more students, then again this can be seen as plagiarism. So please if you have worked as pair make sure both of your identities are on the same report.

Clearly if a student's ID / name are not on a report, then it is assumed they are not one of the authors. Once the deadline has passed it will not be possible to add your identity to a report.

Please make sure all files and folders are present when submitting. You may submit as many times as you wish prior to deadline if you find something was not quite right with the submission so far. Anything re-submitted will automatically overwrite anything uploaded so far.

Please be advised: If files are missing or the Visual Studio solution is corrupted in some way so it is not possible to compile your solution then the mark for the implementation part of the assignment may be zero: It will NOT be possible to email them after the submission date. Suggest you zip up the Visual Studio solution folder and un-zip and try on a different machine ☺

[6] Unfortunately the way DLE is set up, a non-submission is automatically identified by the system. So to show that you have completed and submitted an assignment, then both parties in a pair need to submit their submission.

Plagiarism:
As mentioned this assignment can be worked on and submitted either individually or in pairs. If completed as an **individual** assignment then the submitted work must reflect the work of ONLY that individual. Similarly, if working in pairs then the submitted work must reflect the work of ONLY the two individuals in that pair.

Thus, while you may discuss, **in general terms**, this assignment with your colleagues, the assignment **MUST** be your own work.

Do not: Share designs or code with anyone, OR Submit a program design or code which is wholly or partially the work of someone else.

The University treats plagiarism very seriously. In all cases of suspected plagiarism, formal action will be taken

The penalty for submitting work which is wholly or partially the work of someone else is usually, at least, a mark of zero for the assignment. Also do not be tempted to help a colleague out who is 'stuck', by giving them your code or design, as BOTH parties will be guilty of an assessment offence and BOTH face the risk of a zero mark. Please refer to your student handbook for guidance as to what constitutes original / individual work.

The module leader may viva students on the contents of any part of their submission of their assignment. Failure to attend a viva will result in a zero mark for the assignment.

Ok so how much can you share with your colleagues? The following table gives an indication as to what is allowed / not allowed:

Allowed
Discussing in general terms, i.e. if started / finished assignment; how easy/difficult; clarifying requirements (but should email me to confirm).
Helping to fix compiler errors, but NOT suggesting a better way to do the coding.
NOT Allowed
Suggesting better way to do things, i.e. giving or suggesting an algorithm[7]
Meeting as a group to discuss design and / or algorithms and then, later, individually coding in the design / algorithm
Give ANY: code, design or algorithms to a colleague.
Using someone else's designs/ code found on some media (i.e. a hard-drive), or some printed document found somewhere
Working collaboratively with a colleague or colleagues to complete the assignment
Getting some else to complete the assignment for you!

Essentially if the level of collaboration is not covered by the allowed section, above, you MUST assume that it is NOT allowed. If you have any doubt then email the module leader to conform the level of collaboration BEFORE you do it!

[7] Note an Algorithm in the table refers also to giving or sharing pseudo-code.