Analysis into Computerized Statistical analysis of ecommerce data

Christian Browning

2002042

Contents

Analysis into statistical analysis ..... **Error! Bookmark not defined.**
Overview 3
Interview Transcript: 5
Main points taken from the interview: 6
The Dataset: 7
Analysis of example systems: 9
The Objectives for the final system's functionality: 12
Design Documentation 14
The Leaner Regression program: 15
Mathematical model Behind the program: 15
Pseudocode For Regression system: 22
Comparison between the two graphs equations: 23
General Explanation of Linear regression object: 27
SQLCon class: 28
Public SQLCon() 28
Public SQLCon(string Name) 28
ExicuteQuery(string command text) 28
Data From SQL class: 29
Data Created Class: 29
Data processing Program: 32
DFD and accompanying system pseudocode: 34
The UML diagram of the class structure: 35
CQI Class 38
File Structure: 43
The Dashboard program: 44
Figure 1 Graphical Presentation of the Data 44
Figure 2 Datagridview and Lookup System 45
Classes within the forms application: 45
Technical solution: 53
Preliminary Setup: 53
Dashboard program 53
Graphical element 54
Data Viewer 55
DataFromSQLTables 57
SQLCon 73
The form class itself 74

If Byday button is pressed: 74
If show comparison is pressed 75
If graph by AOV is pressed:....76
If Show overall is pressed....76
If Show Spread is pressed....79
If any display table button is presses....80
Clicking the search button 81
The Regression system: 86
LinearRegression class: 87
Datacreated Class:....95
DataFromSQL class: 101
Data processing system:....103
Table abstract class : 105
MainSet : Table class 111
MMpertrans : Table class: 114
SpentOver : Table 116
CQI : Table....119
MoneyPerDay : Table....121
SQLInteractions() 123
Objectives Achieved or not achieved table:....124
Testing 130
Testing linear regression system:....137
Making The Equation 137
Testing R^2 140
Testing decision logic 142
Testing Dataprocessing system:....145
Evaluation 149
Appendix A Extra:....151
The construction of Datatables: 151
Requesting Rows From SQL: 151
Appendix B Cod 152
Dashboard program:....153
Regression program: 176
Data processing program:....193

Overview

The problem is using computerisation to make predicting of trends and the analysis of customer behaviour more accurate and time efficient using data processing and statistical analysis. I will be creating a data processing and statistical analysis program that outputs to a gooey UI dashboard for Ecommerce data. This system will break down a large dataset into useful composite tables then perform statistical analysis on said tables to present conclusions on the data and predict patterns. This system is required to have the ability to be reused for any future dataset that is broken down into the same base table system. The conclusions and graphs will be presented on a dashboard medium that can be sent and shared to other departments within the e-commerce managing company to inform later decisions.

My third party is my father as he has lots of experience with dealing and managing systems that utilise statistical analysis on e-commerce, but he has never actually programmed a system for it, because of this he is a great resource for what is important and useful outputs for this type of system. The hypothetical end users are other departments that can use the dashboard to extract the useful data created in the statistics program.

My father works in loyalty points in what can be best described as internal marketing to existing clients using their information to market products and services, they are likely to be interested in. The old systems necessitated guessing and the use of conjectural factors like the 80-20 rule or regression towards the mean because it was harder to track individual spending so had to treat the customer base as one entity. The change now is the fact that individualized and personalized data is readily available and this data can easily be personalized to serve a single customer. The problem is companies recognising that fact and valuing those services that take advantage of it.

The old system my father used to work with took lots of manpower and lots of data entry to get any useful data, even though these deals and outcomes are half of the time ignored through personal delivery. The old system utilised individual employees going through individual customer data at a time then delivering bespoke deals and content to them which took a large amount of work and time without collecting useful data on the entire customer base at the time. The system also meant that all the customer analysis was localised to one department and only one department had access and used this useful data limiting the outcome of their analysis' impact to their own department.

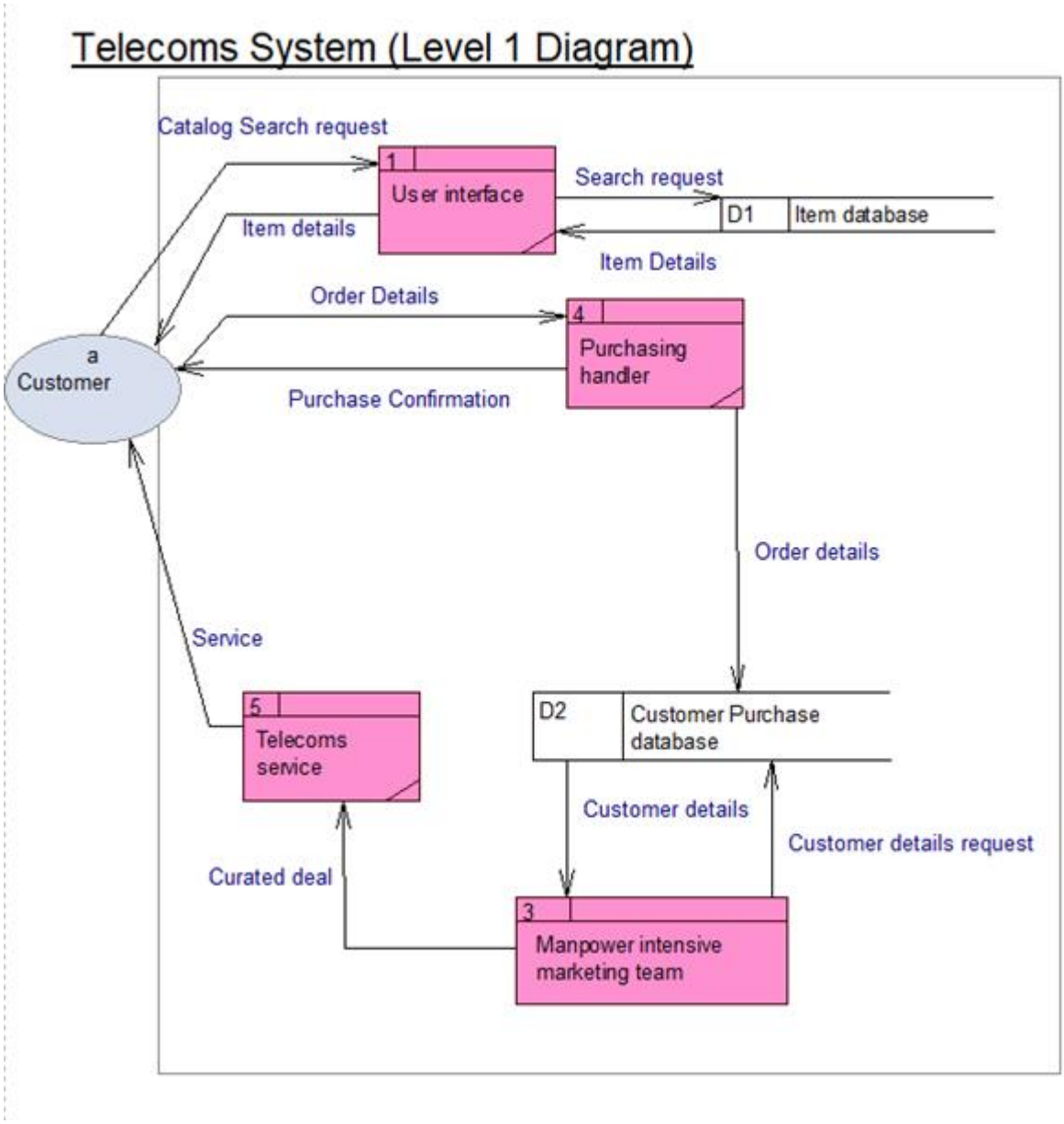


Figure 1 Old system

A company-available Computerised system however could analyse large volumes of data: useful for predicting large scale trends and understanding the customer base while being easily available to transfer between departments. An improved system should: first be able to extract useful data from a main source through data processing in a way that this new useful data can be reused by other departments in an efficient way; second be able to draw broad and quantifiable predictions from this useful data through statistical analysis and create data that can give a greater understanding of the customer base and customer behaviour.

For the interdepartmental aspects of the system the data should be stored centrally in a database and the overall processes should function as two separate programs. One data processing system that both cleans the data and processes it which then stores the data in a database which can be independently requested from other departments. Then the second should be the statistical analysis program which then sends its results back to the same database for interdepartmental use as well as use in the creation of the final dashboard

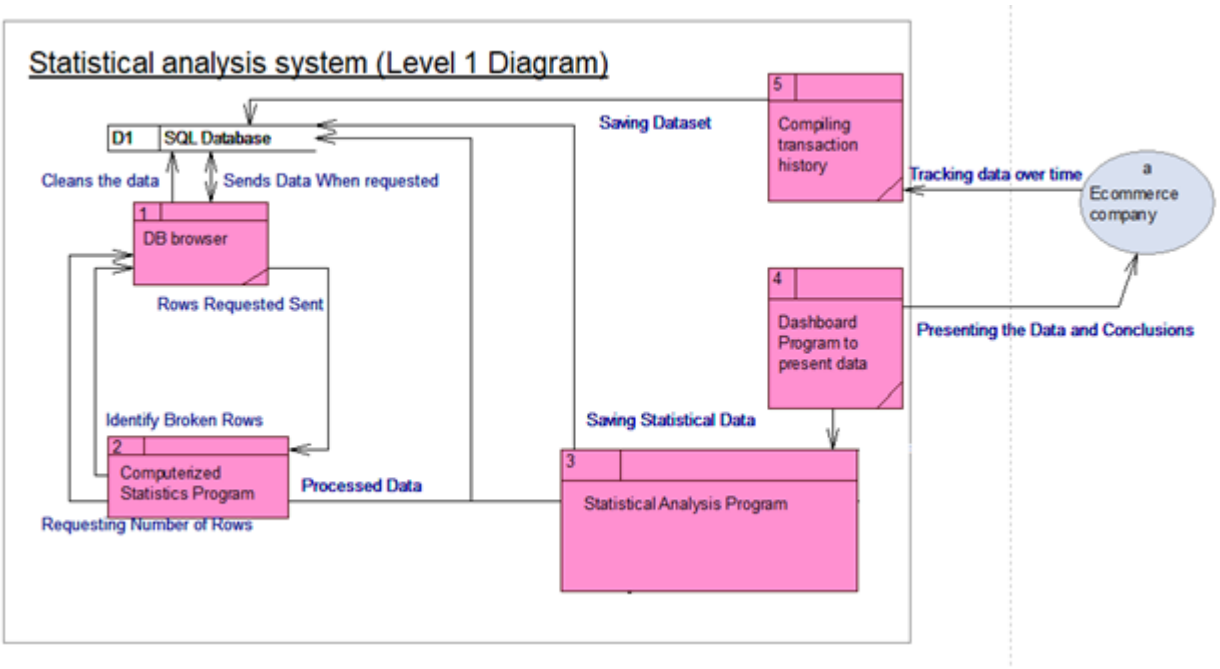


Figure 2 Proposed new system

I conducted ~~a~~an interview with my third party about what is important to include within the system to gain a greater understanding for what I am aiming to accomplish within the system.

Interview Transcript:

Me: *I described the dataset including what the columns are in detail*

Third Party: invoice No is of course used to break into actual whole transactions which is useful for tracking transaction history overall. Stock Code is just the code associated with description Keeping a table of stock codes and their description is important as these can sometimes give better understanding to what the customer base is buying. Quantity is good because some customers, especially in e-commerce, buy a lot of very low value products at once. Invoice date is important to both transactions and the overall purchasing pattern in the store in the time tracked. Unit price is unit price good for understanding the product and should be included in the product sold table if you find a need for one. Customer ID is very important to everything you do in customer behaviour. Country is less helpful as you can quite easily export any system you make between ranking customer behaviour to just include all countries with the same effectiveness.

Me: So, using this column, what would be important to extract from them?

Third party: what we would want to know is our best customers, recognising who they are and what they are buying. Rank it from top customers in different quantities like Value of spend and spent over that period of time and how often they spent especially seeing what products they buy, as some people are big spenders only because of one purchase so are therefore different. Working out what percentage of sales comes from what percentage of

customers is an important metric for understanding the customer base by seeing how centralized the money spent is towards the top customers.

Me: ok what is important after the customers have been ranked by different quantities?

Third party: once you recognise them and see what your best customers are like, you must break them into different types of valued customers. It is customer understanding like what these customers behaviours are during the time tracked to predict future behaviour. Customer insight is like 80-20 rule and are the customers spending frequently in small amounts or infrequently in large amounts as a basic example. Then taking it further and seeing what these products are. For example, the cheapest item that they are frequently to buy can be put on sale to increase, in theory, their overall purchases within the store but this system works best in real circumstances where you can track impact after implementation.

Me: in the system I am creating what would be useful to include with reference to the items important customers are buying?

Third party: This idea of what important customers is buying can be implemented in stock control. As in like always having these items in stock as they are important to a customer's continued use of the store, they might not be that profitable or popular but they are important customers valued products so they should be of value to you.

Me: how would this data created be presented to font end users like yourself?

Third party: usually a dashboard sent around the company that pulls from a database the info I request from it as well as presenting any general predictions the data shows.

Main points taken from the interview:

From the interview I got these main points for what a system should broadly focus on and what would be important to extract from the main dataset for the system's functionality.

1. Recognise best customers based on how much they spend and frequency of spending and Average order value - customer behaviour of the most valued customers.
    - **What** are they buying?
    - What **days** they are buying on?
    - What trends such purchases show over the year.
    - **Workout what percentage of sales comes from what percentage of customers goes a long way towards customer base understanding.**

1. Customer understanding
    - Understand what your top customers are spending
    - When purchases are most likely to happen
    - And what products that are important
    - What is the best customer behaviour?

1. Product understanding

- What are the most important products?
- What products to stock that are linked to higher purchases?

1. Customer ID is the most important value as it allows for segmentation
   - Biggest spender (problem is they could have just bought one expensive product making them a piece of outlier data)
   - Highest average order value - extraordinarily valuable customers as they are consistent to a high degree
   - Spenders who are consistent are AOV compared to SD of spending
   - Find your best customers look a like – and stock what the best customers frequently purchase

The Dataset:

With this understanding I continued and decided to use a base E Commerce data set for the design of the data processing and statistical analysis systems. The dataset I will be using is a large Dataset from the site Kaggle, its description reads, “This is a transnational dataset that contains all the transactions occurring between Nov-2016 to Dec-2017 for a UK-based online retail store.” It is used as an open and free base for teaching machine learning or statistical analysis. Although it is ecommerce you will find all sorts of products sold in this dataset from groceries to appliances leading to a range of customer behaviours.

The third party was then interviewed in relation to the dataset, they explained the useful data that could be extracted from each metric tracked. A singular row in the dataset describes one item in one transaction with one customer.

Example of the data:

InvoiceNo	StockCod	Description	Quantity	InvoiceDat	UnitPrice	Customer	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	29-Nov-16	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANTERN	6	29-Nov-16	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	29-Nov-16	2.75	17850	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	29-Nov-16	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	29-Nov-16	3.39	17850	United Kingdom
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	29-Nov-16	7.65	17850	United Kingdom
536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	29-Nov-16	4.25	17850	United Kingdom
536366	22633	HAND WARMER UNION JACK	6	29-Nov-16	1.85	17850	United Kingdom
536366	22632	HAND WARMER RED POLKA DOT	6	29-Nov-16	1.85	17850	United Kingdom
536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	29-Nov-16	1.69	13047	United Kingdom
536367	22745	POPPY'S PLAYHOUSE BEDROOM	6	29-Nov-16	2.1	13047	United Kingdom
536367	22748	POPPY'S PLAYHOUSE KITCHEN	6	29-Nov-16	2.1	13047	United Kingdom
536367	22749	FELTCRAFT PRINCESS CHARLOTTE DOLL	8	29-Nov-16	3.75	13047	United Kingdom
536367	22310	IVORY KNITTED MUG COSY	6	29-Nov-16	1.65	13047	United Kingdom
536367	84969	BOX OF 6 ASSORTED COLOUR TEASPOONS	6	29-Nov-16	4.25	13047	United Kingdom

For each column there is a description as well as points form the interview with reference to the dataset and what data can be extracted from the data’s collation:

- InvoiceNo - the unique number of each invoice: this is useful not only to identify items frequently bought together as well as the overall value of the singular transaction. The overall value is useful as we can use it as a data point to come to conclusions such as average order value, as well as what customers spend the most in singular transactions. Without this we would have to make too many assumptions about a customer’s purchasing history.

- StockCode Product - the unique code given to each product: this is useful because it allows for the unique identification of a product in a uniform way without having to deal with string descriptions and the issues that might occur with such a system.

- Description Product - the description of the product: useful so we actually know what the code is referring to: useful for understanding what really is being bought allowing an individual employee to see human patterns after a computer has noticed a trend in the data.

- Quantity - amount of units sold: can be used to flag products for high amounts of purchasing and other properties that can be then used to identify customer profiles. Again a piece of useful raw data that can be abstracted into values like total money spent on product then total money spent per transaction and other useful values.

- InvoiceDate - the day the invoice was sent: allows us to break up data, segmenting them into dates and from these seeing patterns like total money spent and AOV. from this segmentation we can come to conclusions about which days are most valuable and therefore important to keep running and which days are less valuable. The days to value aren't always uniform or very predictable without raw values.

- UnitPrice - the price of each unit sold: this is self-explanatory as it is useful for working out revenue generated and other money based values. Useful in the sense that it can break down and segment products easily.

- CustomerID - a unique code to identify each individual user: most useful piece of data. Allows for the unique identification of users within transactions allowing for the tracking, analysation, and a creation of an individual profile for the customer. This ID allows for a customer-oriented approach to the marketing system. As we can identify products that the ID buys and helps build a profile for each customer.

- Country - the country the transaction originated from: as the lion's share of the purchase originates from the UK this data point isn’t incredibly useful but can be used to segment customers further.

Now that I have investigated what the Program must accomplish to be successful as well as what raw data can be extracted from the dataset, I now set a series of objectives for the final program.

Analysis of example systems:

A part of the final product will be a dashboard to present the years' worth of data and present the statistical analysis as well important metrics that show trends or changes in the data. To create a dashboard that presents useful information in a digestible way I will be analysing different dashboards and deciding on what specific ways I want to present.

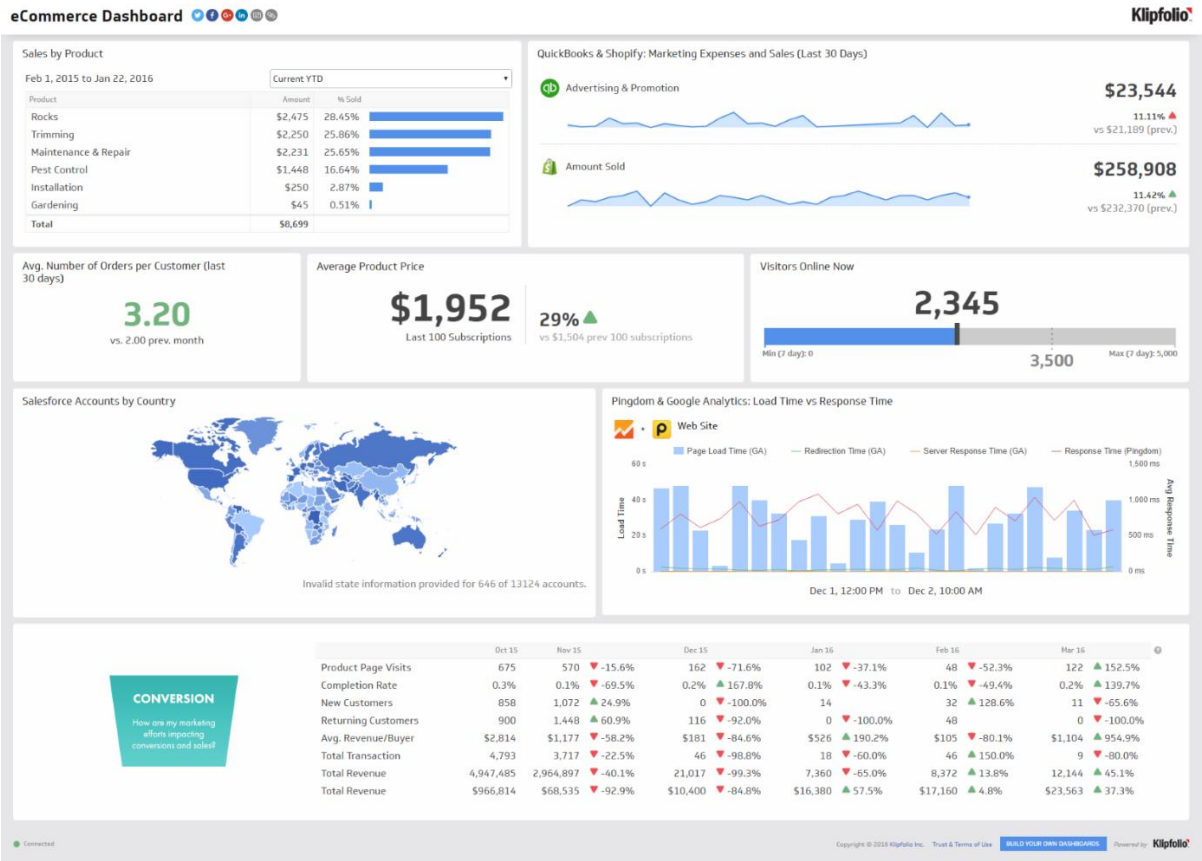
Here is a example of a Ecommerce Dashboard product example provided by Geckoboard:

Although this appears to be a daily general maintenance dashboard it is still worth analysing for our purposes. Aside from graphic design elements the dashboard successfully presents the data and general trends it is trying to focus on, Geckoboard puts its focus as Revenue, orders, business health which is shown in its focus on orders and profit margin focusing less on presenting customer information and more on the overall business which fits its target audiences of Owner, CEO, Marketing Director. The dashboard contains many single Value Charts that describe general upticks and downticks in the system. This is useful as these are the easiest to view and understand and present general values that describe the system. Its average order metric also presents general health well and is something that I wish to include but instead present it across a line graph to compare average order value throughout the year.

Here is a example of a Ecommerce Dashboard product example provided by Sisense:

With this dashboard I wanted to Analyse how it displayed the comparison between two populations within its customer base namely men and women customers I will be analysing this dashboard for the purposes of comparing top 25% of data items to the bottom 75. The dashboard presents the two populations alongside each other in many different types of graphs to present different facets of the data. The scatter blob diagram helps illustrate one the range of data in the system as swell as how much one population dominates different areas effectively, without overloading the user with thousands of datapoints in a scatter diagram. The pie chart is used to display how much two populations values differ by more directly comparing them. The options on the right help present a range of options while staying out of the way of the graphs displaying the data.

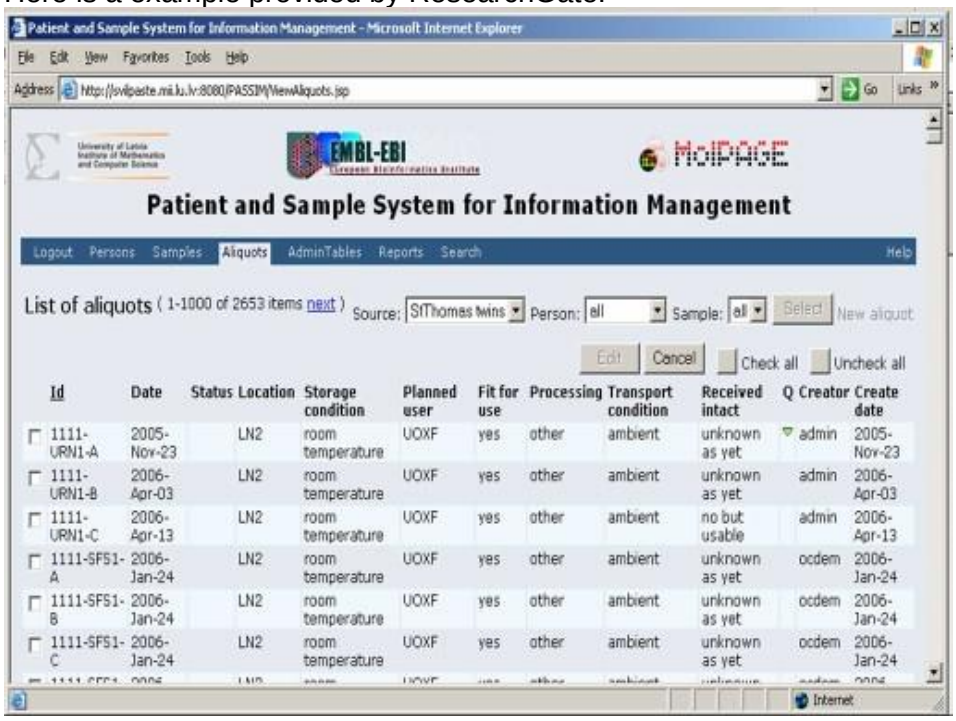
Here is an example of a Ecommerce Dashboard product example provided by Klipfolio:



This dashboard is good at presenting increases and decreases in metrics effectively throughout a time tested using detailed value-based table as well as general trends in graph format. By selecting important metrics like Average order value and using percentage comparisons the dashboard successfully shows the differences in these metrics throughout the year.

An objective my Dishoard requires that is different to most dashboards is also a way of interfacing with the database to get information from it. To do this it requires a different section to presenting the data to allow for the selection of tables to download. For this I will be analysing database interfaces.

Here is a example provided by ResearchGate:



Aswell as the overall data grid that is displayed there is an option of selecting individual persons with a sample conduction. This is a very useful tool for anyone using the data created by the data processing system and is an option I wish to include as well as accompanying graphs to show this individuals' overall graphs to present their customer history.

The Objectives for the final system's functionality:

General Objectives:

1. Create a system that breaks down the large almost half a million row long raw starting dataset into useful composite tables
2. Use statistical analysis techniques on this data to create new data and create comparison quantities for segmentation
3. Segment customer base and Products into useful groupings
4. Analyse the created data to predict future behaviour
5. Create a Dashboard GUI to present the data created by the overall system

Specific Objectives

1. Create a system that breaks down the large almost half a million row long raw starting dataset into useful composite tables
   1.1. Request and receive data from a central SQL server using SQL commands
   1.2. use an Object-oriented approach so that the individual parts of the data processing system can be re used in different systems that use a similar data set format
   1.3. Create useful tables and store them in a medium so that this data can be re used in other systems by other departments, these tables include:
      1.3.1. Money Spent by each customer in the time tracked
      1.3.2. Individual transactions

1.3.3. Frequency of ordering of each customer
1.3.4. Average order value of each day
1.3.5. Money spent by all customers in each day
1.4. Clean the Dataset through SQL requests
1.5. Fill the database with the tables created by the system

2. Use statistical analysis techniques on this data to create new data and create comparison quantities for segmentation
   2.1. Implement a linear regression system to analyse data created in the data processing system
      2.1.1. Implement a linear regression method into the statistics system to find a line of best fit for data
      2.1.2. Implement a polynomial regression methods system to create a curved line of best fit
      2.1.3. Have these methods be efficient options compared to competing to not extend overall execution time excessively
      2.1.4. Compare the linear and polynomial equations using an appropriate constant to avoid under fitting and over fitting only using polynomial when deemed appropriate
      2.1.5. Use a technique to compare the accuracy of the linear regression equations in a way that means that the scales of different equations don’t compromise the comparisons.
   2.2. Use the regression system on datasets:
      2.2.1. Individual customer transaction history from money spent
      2.2.2. Days in a month, week and quatre
   2.3. Create measures of spreads and appropriate comparison values for segmentation for each table when appropriate
      2.3.1. Use upper quartile to get the top 25% of data
      2.3.2. get values for General measures of spread to be used
         2.3.2.1. Standard deviation
         2.3.2.2. Mean
         2.3.2.3. Quartiles
         2.3.2.4. IQR
         2.3.2.5. Mode
      2.3.3. Create an equation to create a value for how much a customer is a consistently high spending and consistently high frequency at spending

3. Segment customer base and Products into useful groupings
   3.1. Using the comparison Values created in the statistical analysis program segment the data by the value top 25%
   3.2. Compare the summed overall upper quartile to the overall original data to produce a value for how centralized this data is
4. Analyse the created data to predict future behaviour
   4.1. Use the Linear regression equitation's for the sum of Month-by-month spending and test using a constant if a valid and useful prediction can be made about the future year
   4.2. Predict future trends of individual customers based on their regression equation
   4.3. Use the Linear regression equations of each customer to if the global transaction value is increasing or decreasing
5. Create a Dashboard Gui to present the data created by the overall system
   5.1. Have the dashboards data be requested from a SQL server so that this data can be added to in a flexible way
   5.2. Present the data created by the data processing system so that individual departments also have access to them through requesting this data in a button
   5.3. Present the data on Day-to-day purchases including money spent overall and number of purchases using a line graph

5.4. Present pie charts on how much the top 25% of different metrics contribute to the overall sum of these metrics
5.5. All tabular data is to be presented using a scroll wheel to traverse down the large dashboard
5.6. Have the option for the user to request specific customers transaction histories including their Linear regression equation through a drop-down menu

Design Documentation

Intro:

I will be creating a computerized statistical analysis program that works on e-commerce data using my father's experience in loyalty programs as a third party. For this I will be creating a program for statistical analysis and data processing using the programming language C# and SQL Lite for database back-end tables. SQL is Structured Query Language and is the language used to interact with SQL databases. This system will manually break down a large dataset into useful composite tables; to create useful data to draw graphs and conclusions from. This system can be reused and will work for any dataset that is broken down into the same base table system. The conclusions and graphs will be presented on a dashboard to be sent and shared to other departments within the e-commerce managing company to inform later decisions.

Important metrics to track were found through the interview; for customer segmentation, in broad strokes, it is first useful to identify what type of customers exist in the dataset then find ways of further segmenting into useful groups. Specific behaviours that identify the customer types must be broken down into individual values. Several quantities that the third party have worked with before are rank spending, average order value, consistency and, decided based on the company's overall planned direction, valued customer behaviour. These quantities must be extracted from tables abstracted from the original dataset through data processing

Finally, from this we can start designing what specific types of tables we want to create in the data processing system:

1. Table tallying up money spent by customers in the time tracked and in specific **time frames like months or quarters**.
2. An abstracted table into transactions from the main dataset
3. A table of Money spent in all transactions every day tracked.
4. Creating lists of high value items based on different quantities
5. Creating a list of customers increasing in value and customers decreasing in value
6. Top customers Top buying items

I will now break down the separate programs in the system and their functionality individually. Describing them using a DFD's and continuing by explaining each of the processes within. I will then break down how the actual code will be structured in UML diagrams.

Page Break

The Leaner Regression program:

Mathematical model Behind the program:

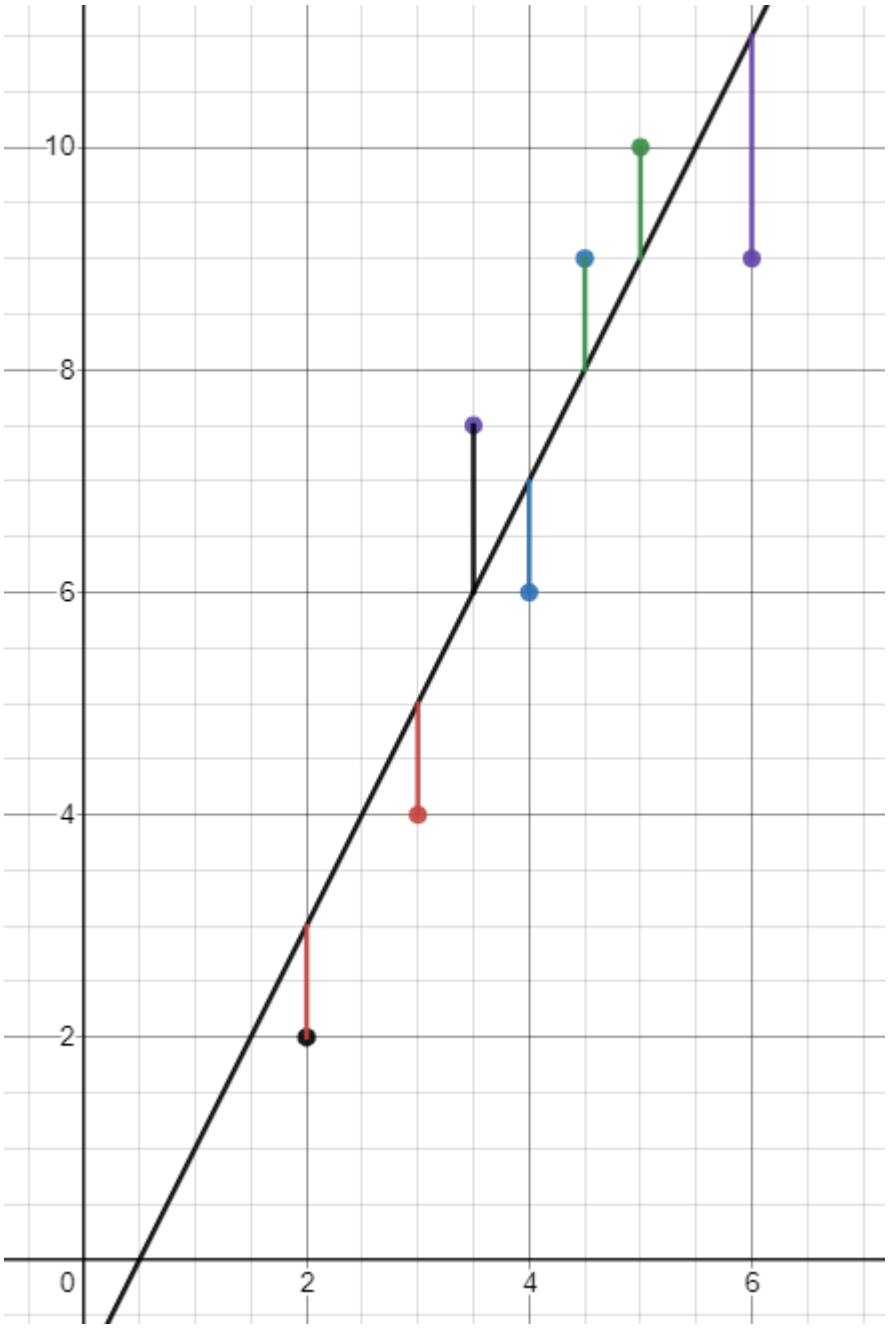
The main goal with the statistical analysis program was predicting future trends. A useful metric for customer understanding is the linear regression and curve fitting of data gathered. From this future behaviour can be understood and confidence in these predictions can be easily evaluated and compared allowing for accurate implementation of marketing schemes.

The linear regression object will be designed with both simple and polynomial options. The polynomial will be compared to the simple as if the polynomial isn't highly confident in its fitting, then we could have problems with under and overshooting making it counter effective. The Linear regression method will be a least squares algorithm.

Constructing the algorithm:

I need to find a model of Linear regression that will be fast enough to allow for tens of thousands of versions to be created in the program. For this I will be finding a pure mathematics model that only requires mathematical operations in built to C#. Below is the explanation defining model I plan to use and how it operates.

The model of least squares is based around finding the least Sum possible of Error or distance between the data point and the hypothetical datapoint of the line. This error can be modelled as the Y value paired with a X value in the original data minus the Y value created by the graph of the hypothetical Linear regression function. The sum of the distances between the points and the graph fluctuates greatly with any change to the graph.



The distance between the graph and the data point can be worked out using this equation.

$$(y_n-(bx_n+c))$$

For example, a data point on the graph is (3.5,7.5) and this examples graph equation is y=2x+-1 so the equation and error for this point is.

$$7.5 - (2(3.5)-1)=1.5$$

As many of the data points will be below and above the line this leaves positive and negative differences. Example, for the data point (6,9) in the graph above when the error equation is used it produces a negative value. This is then remedied by squaring the values.

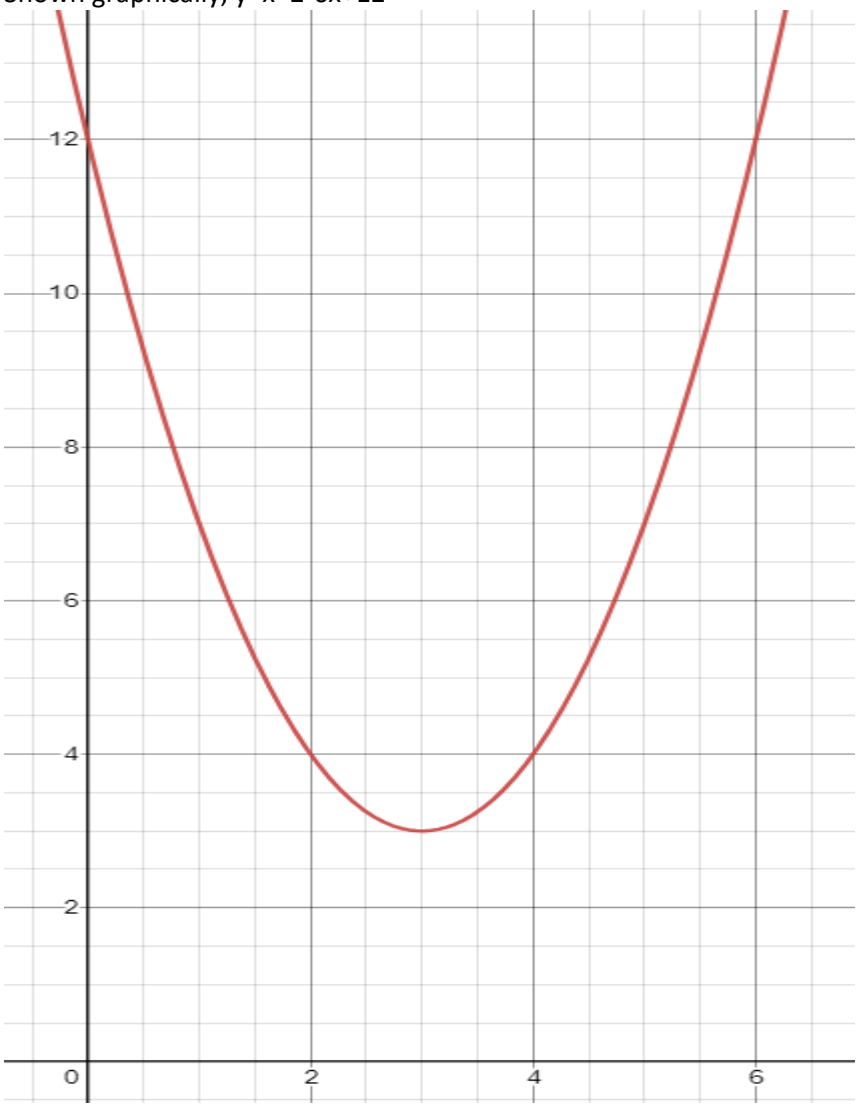
$$9-(2(6)-1)=-2$$

To find the graph with the smallest error we have to construct an equation that finds the sum of all errors. Then using this equation find what values of b and c in the y=bx+c equation where the sum of errors has the smallest value. The Yn represents one data point, and the sum errors of a proposed equation can be worked out through the methood bellow
There will be as many equations as there are points in the data

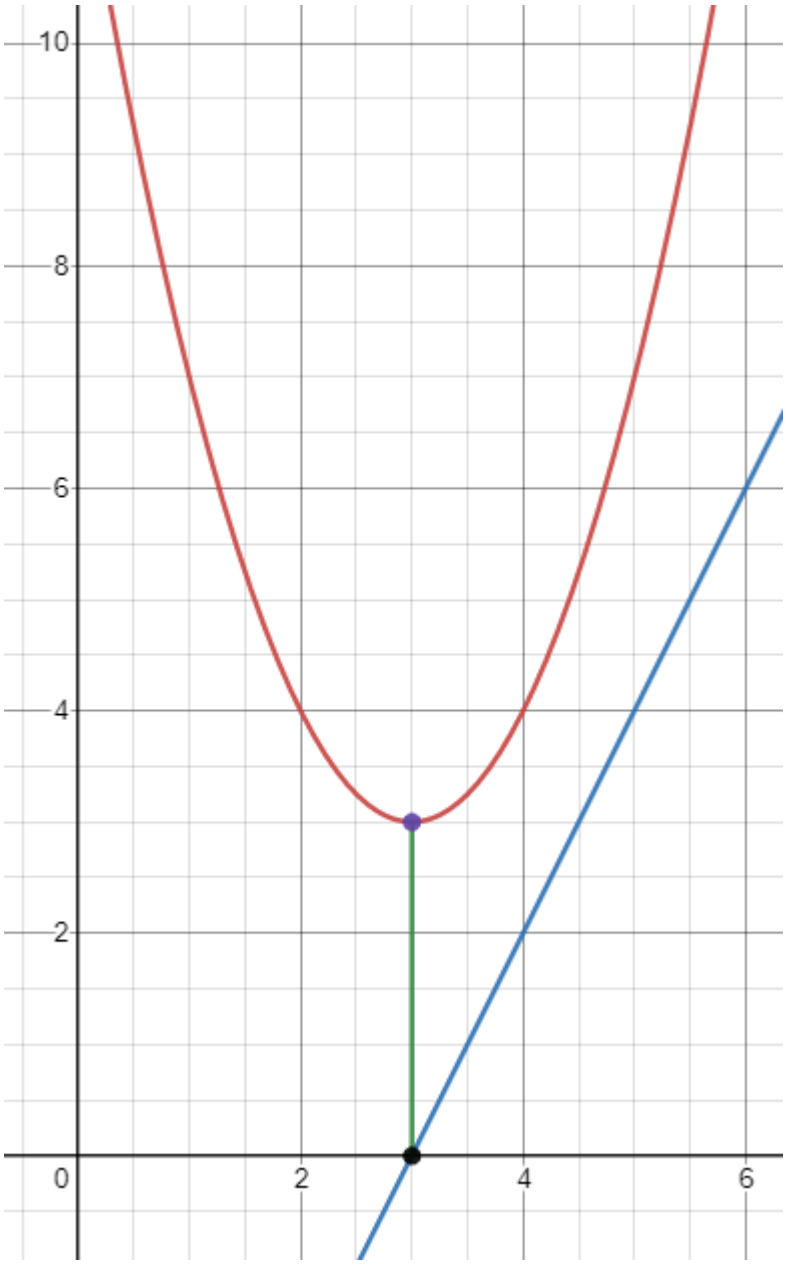
$$(y_1-(bx_1+c))^2 + (y_2-(bx_2+c))^2+(y_3-(bx_3+c))^2+.....+(y_{Last}-(bx_{Last}+c))^2=\text{Error}$$

The lowest possible error value can be found through differentiate of this error equation. To then find the equations stationary point can set it to = and find the lowest value of Error possible.

Shown graphically, y=x^2-6x+12



To find where this equation is smallest, we find its differentiate and look for when it is = 0 called its station point or turning point as this is the lowest the graph becomes before increasing again. Its differentiate is y=2x-6.



As shown above the X value where the quadratic graph is smallest is the X value when it differentiates crosses the X axis and = 0.

Linear system:

To construct the error equation, we must condense it into simple variables. Instead of taking all the equations as separate we take the Sum of all the data points present. If we use the Sum of all Yn's - the sum of all bXn + C we can create one simple equation.

$$\left(\sum_{1}^{n} y_n - \sum_{1}^{n} \left(bx_n + c \right) \right)^2 |$$

We must now expand the brackets of the square

$$\left(\sum_1^n y_n\right)^2 + \left(\sum_1^n (\ bx_n + c)\right)^2 - 2\left(\sum_1^n (\ bx_n + c)\right)\left(\sum_1^n y_n\right)$$

Then we complete further expansion. For the sake of simplicity of notation, I am using Y and X to represent the sum of all the yn values and xn values. Expanding the The Sigma notation on the (bx+c) gives

$$(bX + cn)$$

c is a constant so can be put behind the sigma notation making it the sum of 1 to the number of data points. leaving just the number of data points.

$$c\sum_1^n 1$$

After making it a singular equation we now must take the partial derivatives of b and c then set it to 0 to gain their individual values where the Error is smallest.

$$b^2X + 2bcX + c^2 + Y^2 - 2bXY - 2Ycn = Error$$

$$\frac{dError}{db} = 2bX^2 + 2cX - 2XY = 0$$

$$\frac{dError}{dc} = 2bX + 2c - 2Yn = 0$$

After setting both to 0 and dividing both by 2 we can now represent this data in matrix form. I will be using a multi-dimensional array to represent matrices as they are functionally identical data stores for my present purposes.

Matrix M Matrix A

$$\begin{pmatrix} n & X \\ X & X^2 \end{pmatrix} \times \begin{pmatrix} b \\ c \end{pmatrix} = \begin{pmatrix} Y \\ YX \end{pmatrix}$$

For this we can use Cramer's rule. Which can be described as “a method that uses determinants to solve systems of equations that have the same number of equations as variables". It does this by working out the determinants of matrix A and M by placing the values of matrix A into M horizontally in place 1 or 2, depending on which coefficient you are solving for. Then dividing this new matrix by the original matrix M.

$$M\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x1 \\ x2 \end{pmatrix} = A\begin{pmatrix} e \\ f \end{pmatrix}$$

A is the original matrix where e and f will replace the first Colum if trying to work out the first variable.

$$x_i = \frac{\det(Ai)}{\det(A)}$$

Example

$$det\begin{pmatrix} e & b \\ f & d \end{pmatrix} det\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = x1$$

$$det\begin{pmatrix} a & e \\ c & f \end{pmatrix} det\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = x2$$

This method can be used to solve for b and c which would give us their values where their differentiate is 0 aka where the error is at the lowest it will be.

$$\begin{pmatrix} n & X \\ X & X^2 \end{pmatrix} \times \begin{pmatrix} b \\ c \end{pmatrix} = \begin{pmatrix} Y \\ YX \end{pmatrix}$$

$$det\begin{pmatrix} Y & X \\ YX & X^2 \end{pmatrix} det\begin{pmatrix} n & X \\ X & X^2 \end{pmatrix}^{-1} = b$$

$$det\begin{pmatrix} n & Y \\ X & YX \end{pmatrix} det\begin{pmatrix} n & X \\ X & X^2 \end{pmatrix}^{-1} = b$$

We must then find the determinant of these values using this general formula. M represents the original matrix and |M| is the determinant of that matrix.

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad |M| = ad - bc$$

After the determinate of the two matrixes is calculated a simple division takes place giving the values for b and c representing the slope and Y intercept respectively.

Polynomial system:

This sequence of steps can now also be easily also applied to quadratic linear regression using the exact same method but using a 3x3 matrix, where I will also explain how to complete this process in a pseudocode algorithm. This is the quadratic formula I am using for this example.

$$ax^2 + bx + c$$

I will show the example for X^2 and for the sake of not repeating steps I will be skipping to the completed differentiation of the large equation that would come from expanding.

$$\left(\sum_1^n y - \left(a\sum_1^n x^2 + b\sum_1^n x + c\right)\right)\left(\sum_1^n y - \left(a\sum_1^n x^2 + b\sum_1^n x + c\right)\right)$$

The differentiates of the large equation are:

$$\frac{derror}{da} = -2YX^2 + 2X^4a + 2X^3b + 2Xc \qquad \frac{derror}{db} = -2YX + 2X^3a + 2X^2b + 2Xc$$

$$\frac{derror}{dc} = -2Y + 2Nc + 2bx$$

Which can now be placed within this 3x3 version of the old equation.

$$\begin{pmatrix} N & X & X^2 \\ X & X^2 & X^3 \\ X^2 & X^3 & X^4 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} Y \\ YX \\ YX^2 \end{pmatrix}$$

And then further placed using Cramer's rule equation.

$$det\begin{pmatrix} Y & X & X^2 \\ X & X^2 & X^3 \\ X^2 & X^3 & X^4 \end{pmatrix} det\begin{pmatrix} N & X & X^2 \\ X & X^2 & X^3 \\ X^2 & X^3 & X^4 \end{pmatrix}^{-1} = a$$

$$det\begin{pmatrix} N & Y & X^2 \\ X & YX & X^3 \\ X^2 & YX^2 & X^4 \end{pmatrix} det\begin{pmatrix} N & X & X^2 \\ X & X^2 & X^3 \\ X^2 & X^3 & X^4 \end{pmatrix}^{-1} = b$$

$$det\begin{pmatrix} N & X & Y \\ X & X^2 & YX \\ X^2 & X^3 & YX^2 \end{pmatrix} det\begin{pmatrix} N & X & X^2 \\ X & X^2 & X^3 \\ X^2 & X^3 & X^4 \end{pmatrix}^{-1} = c$$

The determinate can be calculated on 3x3 matrices through this method:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = M \quad |M| = a\begin{pmatrix} e & f \\ h & i \end{pmatrix} - b\begin{pmatrix} d & f \\ g & i \end{pmatrix} + c\begin{pmatrix} d & e \\ g & h \end{pmatrix}$$

$$|M| = a(e)(i) - a(h)(f) + b(f)(g) - b(d)(i) + c(d)(h) - c(g)(e)$$

After the determinates values are found the division of each of the two matrices' determinates can be completed.

Pseudocode For Regression system:

This Model above proves the method of finding the coefficients of x^2, x and c of the equation with the smallest Error that I will implement in my statistical analysis system. The model proves that to find the a, b and c I need to have the following sequence of steps in my program: find and create the quantities within the matrices above; fill a matrix like datatype with these values; find these matrices determinates and the finally divide the determinate values to get a, b and c.

The data within the matrices can be found by using a simple for loop to loop the data: summing the raw data or editing like squaring then summing it.
Pseudocode:

```
XVALUES = NUMERIC ARRAY [] //an array holding the xvalues of the dataset
YVALUES = NUMERIC ARRAY [] //an array holding the yvalues of the dataset
FOR i= 0 , I < XVALUES[].LENGTH //looping through all of the values in the dataset to create
all                                    //of the data needed in the matrices
        sumx = sumx + XVALUES [i]
        sumy = sumy + YVALUES [i]
        sumYX = sumYX + (XVALUES [i] * YVALUES [i])
        sumXsqY = sumXsqY + ((XVALUES [i] * XVALUES [i]) * YVALUES [i])
        SumXsq = SumXsq + (XVALUES [i] * XVALUES [i])
        SumXcb = SumXcb + (XVALUES [i] * XVALUES [i] * XVALUES [i])
        SumXqu = SumXqu + (XVALUES [i] * XVALUES [i] * XVALUES [i] * XVALUES [])
END FOR
```

The determinate to increase efficiency can be found manually through simple addition and division of values within the matrix. In the C# code I will be using a 2d array to represent a matrix however will only be addressing the values within the matrix for the addiction, not making a matrix object that holds the Metrix manipulation. I will create a method within the linear regression object called find determinate and feed in a matrix to return a double value of the determinate then separately divide the two determinates together.
Pseudocode for Find Determinate method:

```
Numerical Matrix D[3,3] //Create the 2d matrix that's is 3x3
//Comments explaining how the Determinate is found:
// Matrix D=
        //[a1, b1, c1]
        //[a2, b2, c2]
        //[a3, b3, c3]
        // |D|=
        //a1[b2, c2] -   b1[a2, c2] +   c1[a2, b2]
        //    [b3, c3]         [a3, c3]         [a3, b3]
```

```
        // a1 , b1 and c1 above are multiplied by the 2x2 matrix made above
        //A = a1(b2)(c3) - a1(b3)(c2)
        //B = b1(c2)(a3) - b1(a2)(c3)
        //C = c1(a2)(b3) - c1(a3)(b2)
        //Once A, B and C are made they can be added to create the determinate
//Here are the actual manual additions to take place for the determinate.
Number A = ((Matrix[0, 0]) * (Matrix[1, 1]) * (Matrix[2, 2]))   - ((Matrix[0,0]) * (Matrix[1,2]) *

        (Matrix[2,1]) );
Number B = ((Matrix[1, 0]) * (Matrix[2, 1]) * (Matrix[0, 2])) - ((Matrix[1, 0]) * (Matrix[0, 1]) *
                                                                                        (Matrix[2,
2]));
Number C = ((Matrix[2, 0]) * (Matrix[0, 1]) * (Matrix[1, 2])) - ((Matrix[2, 0]) * (Matrix[0, 2]) *
                                                                                        (Matrix[1,
1]));
Return Number Determinate = A + B + C;
```

After the determinate for the Main matrix M and the 3 individual matrices for a,b and c simply dividing them finds their value.

```
Double detM = FindDeterminate(M)
a =  FindDeterminate(A)/ detM
b =  FindDeterminate(B)/ detM
c =  FindDeterminate(C)/ detM
```

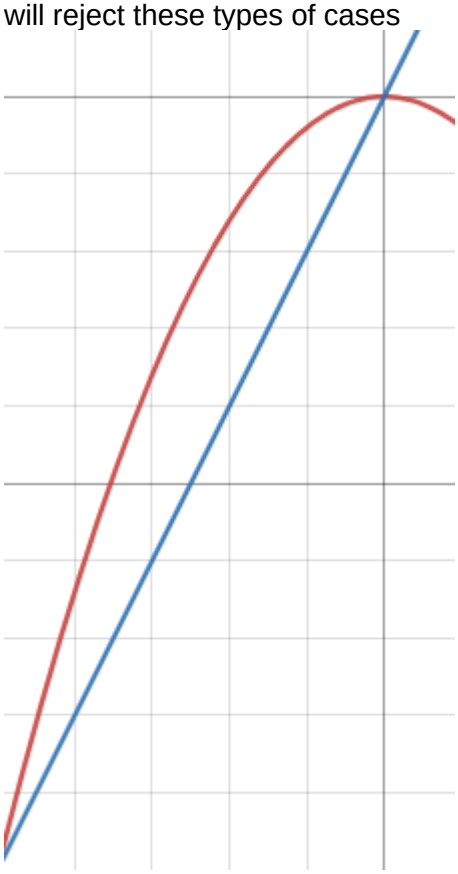
Testing these algorithms will be important to make sure the math has been translated correctly into the C# code and not broken through syntax. Iterative testing will be a large part of building the algorithm. For this I will be using a small dataset and following along with the algorithm testing if the values produced are functioning at each step.

Comparison between the two graphs equations:

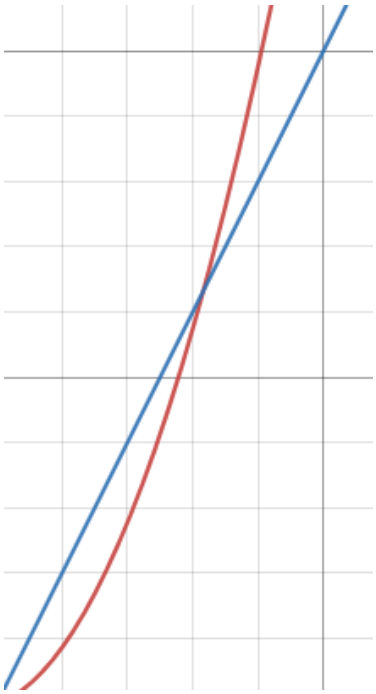
Logic governing the validity of equations

With both a linear and polynomial system the program can use either of these equations, but logic must be implemented to determine which to use. For comparing the accuracy of the two equations created from the same dataset I will be using the Sum of Squares. This meaning I will compare the least squares between the two equations and pick the better option. However sometimes the best fitting equation isn't the most useful or accurate to determining purchasing patterns. These cases are outlined below as well as how I will be dealing with them.

This graph is an example of where the Quadratic graph will predict and negative future trend whilst linear shows a general more accurate upward trend. The Quadratic Although perhaps more numerically accurate this will likely produce less accurate results than following the general trend of the straight Linear regression equation. The lack of accuracy in predictions can be seen in how the Straight quadratic line bellow shows a general upward trend but its quadratic equation produces a decreasing result for larger future values. What we gather from a negative quadratic result and a positive simple regression line is a plateauing in order value not a diminishing in overall spending that the pure quadratic will predict. So, the system will reject these types of cases



Below is an example of a less dramatic problem as if the values more closely follow the quadratic its use will produce better predictions but only shortly into the future, So in this case this combination will be accepted by the program. All of the quadratic graphs will produce temperamental values, however positive quadratic graphs are much more likely to produce useful results than negative.



Problems that arise from using quadratics can be removed using proven rules of thumb for example disregard quadratic values with a Negative X^2 coefficient if its linear counterpart' X coefficient is positive and where there aren't enough data points to metric a linear regression, at least 2, and not enough to merit a tested quadratic, at least 3 .

Least Squares:

If the pair of equations does not fall into any of the rejected cases above the two equations least squares will be compared. The pseudocode for this is below:

```
GetResiduals(double[] Equation, double[] Xvalues, double[] Yvalues)
Double Xsq = Equation[0]
Double X = Equation[1]
Double C = Equation[2]
Double Squares = 0

For Int y =0  Till y < XValues.Length
        Double YCreated = Xsq*(Xvalue[y]* Xvalue[y]) + x*Xvalue[y] + C
         //if the equation inputted is quadratic it wont effectthe outcome as it is multiplied by 0

        Double Remainder = Yvalues[y] - YCreated
        Remainder = Remainder * Remainder
        Squares = Squares + Remainder
End for
Return Squares
End methood
```

R Squared:

Once the slope and y intercept are calculated and compared the model must then be checked for the usefulness in describing the trends in the data. We have the error of squares of both equations created but because different graphs between customers can vary widely in scale this individual value cannot be used to test if the model is useful between graphs. For Example one customer's graph may have Y values between $200-2000 and another may be $2-10 the numerical value for error is a bad marker for accuracy as the error can be magnates different. A method that tests for model accuracy to the data in the proportion R^2.

R^2 is the coefficient of determination meaning the proportion of error produced by the equation. The equation uses the distance between the line to the actual equation then squared to keep it positive to work out the error, the equation then divides this Error's value by the distance from each of the graph's values to the mean squared. The distance from the mean is proportional to the variance to the data. This makes the Coefficient of determination proportional to 1 regardless of the actual values within the data.
The equation is defined as:

$$R^2 = 1 - \frac{Error\ to\ equation}{Error\ to\ mean}|$$

This R^2 value is a measure of how well the model fits the data and so can be used to compare the confidence between different customers equations and therefore compare confidence in predictions. This allows us to have a logical numerical way to only store and regard useful information in the system.

In code this can be found quite easily in one for loop through the data summing the error to equation and error to mean.

```
GetRsquared(double Equation[], Double[] XValues, Double[] YValues)
Double[] XValues
Double[] YValues
Double Xsq = Equation[0]
Double X = Equation[1]
Double C = Equation[2]
Double Mean = Get the mean of Y Values(YValues)//Self explanatory
Double SomeOfDistanceToMean = 0
Double SomeofSquares = GetResiduals(Equation, XValues,YValues) // uses past methood
For Int y =0  Till y < XValues.Length
        Double Distance = YValues[y] - Mean
        Distance = Distance* Distance
        SomeOfDistanceToMean = SomeOfDistanceToMean+ Distance
End for
Double Rsq = SomeofSquares/ SomeOfDistanceToMean
Return Rsq
```

General Explanation of Linear regression object:

How the Linear regression section will be integrated into a C# program:
The data for the quadratic slope, the slope and the Y intercept and the customer associated will be stored in a data table and then stored in the same central database to be requested and presented by the dashboard program. The Linear regression object will contain attributes to store values of the equation created and deemed accurate for the data inputted. The Object will be centralized to its constructor methods taking the array of X values and Y values as parameters and have all the functionality of the object called within this constructor method with only the return values method being public.

With every method and algorithm within the linear regression system explained I will now using structured English describe the constructor method where these methods will be called and the logic overarching this system:

```
Linear regression Constructor(XValues, YValues)
        If more than 1 unique value is found in the Y values array
        //checks to see if any equation can be made from the data input to make sure errors
        don't occur when a customer only has a transaction history less than 1
        Create linear equation
        If more than 2 unique values are found in the Y value array
                //Checks if 2 valid equations can even be made from the data inputted
                        Create Polynomial equation
                        workout both equations R^2
                        workout both Equations Some of Squares

                        If the R^2 of both equations are less than 0
                        // if R^2 is negative it means that the model fits the data worse than a
                        null hypothesis so is useless so this test must be completed first to see
                        if the next comparisons are even useful
                        Assign all coefficients 0



                        if the R^2 of one equation is lower than 0.4 and the other is larger than
                        0.4
                        Assign all the coefficients of the Equation with larger R^2 value

                        If slope of linear is positive and the slope of quadratic is negative
                        //as explained in in figure ___ this won't lead to good predictions
                        Use the Linear equation
```

```
                    If slope of linear is negative and the slope of quadratic is positive
                    //same is true in opposite case
                    Use the Linear equation

                    If the sum of squares aren't equal
                        Use the equation with the smallest Sum of Squares

                    If non of these conditions are met
                    Store Linear as less overall temperamental
            Else
                    Use Linear equation
        Else
        Assign all coefficients 0
End constructor
```

Classes Within the system:

SQLCon class:

This class is the backbone of all SQL interactions in each of the programs and is duplicated between the programs due to its general design. It is composed in almost all classes with the programs. Utilizes SQLite library to communicate with a preassigned Database whose file path is stored within it as a const string.

Methods within the class:

Public SQLCon()

A constructor that simply opens a connection to the database allowing for general applications and communication between the SQL database and program in general use cases. Stores the SQLiteConnection as a public attribute.

Public SQLCon(string Name)

An overloaded constructor than instead is made with the specific purpose of reading data from a specific named table within a database. After opening the connection to the SQL database it then creates a SQLiteDataReader using the command “SELECT * FROM” + Name storing the SQLiteDataReader in a public attribute of the object called reader. This can then be used to receive individual row data from the SQL database.

ExicuteQuery(string command text)

This method send a command passed to it as a parameter to the SQL database it is connected to. Made to be general use so that it can be used in many different cercomstances.

Data From SQL class:

This class will be used to request and receive data from the Central SQL database the data processing program is also connected. The class includes methods to create the data tables and fill them with data requested from the SQL database.

Methods within this class:
CreateMoneyMadePerDay():
CreateMMPerTransaction():
CreateSpent();

These methods simply Create datatables and use a generalized SQL function of “Select * from datatable.name” passing this as a parameter to the ExicuteQuary(string) method of SQLCon to receive data from the SQL database. The Psudocode of how to constrct a Datatable and fill the Datatable with tabular SQL data can be found in appendix A as it will be referred to several times.

Data Created Class:

This class will include data created by the Statistics program. It will include both a Customer Regression Aswell as a Quarterly system which takes in by day data and finds each quarters regression Equation. And then make them available to the other classes within the program.
Methods within this class:

This class includes generalized Datatable functions that will be explained psudocoded more extensively in the next program as it makes heavier use of them. These generalized methods are the following:
SaveToSQL()
MakeColumnsReady()
MakeRowsReady()
Insertdata()

The methods unique to this class require explanation:
FillLinearRegressionDays():
This method is to create quarterly linear regression equations. As the amount of data and therefore the number of days can fluctuate in the system does so to remedy this it devides the number of days inputted by 4 to get the length of quarter.

```
Create a list of int values called dsays

Load in the Money Made per day datatable

Foreach row in the datatable

        Add day value to the list

End foeach

Int quarter = days/4 rounded up

Create 4 Tuples of 3 lists

T1

T2

T3

T4

Fill the tuples Lists with the Days, money spent and frequency held within quarterly sections of the

datatable between so the T1 holds the first quarter T2 holds the second quarter ect

Foreach Tuple that exists

        extract the list of days and money spent from the tuple

        pass these lists as parameters to the Regression object

        //Datatables are filled with datarows which must be constructed before being added

        Fill a data row with the following structure

        Row["quarter"] = quarter associated with tuple

        Return the coefficients from the Linear regression object

        Fill the rest of the row's space with these coefficients

        Row[slopex2] =

        Row[slopex]=

        Row[Yint]=

        Row[Rsq]=

        Row[ProductMomentcorilation]=

        Add rows to datatable of QuaterstoMoney



        extract the list of days and Frequency from the tuple
```

```
        pass these lists as parameters to the Regression object
        //Datatables are filled with datarows which must be constructed before being added
        Fill a data row with the following structure
        Row["quarter"] = quarter associated with tuple
        Return the coefficients from the Linear regression object
        Fill the rest of the row's space with these coefficients
        Row[slopex2] =
        Row[slopex]=
        Row[Yint]=
        Row[Rsq]=
        Row[ProductMomentcorilation]=
        Add rows to datatable of QuaterstoFrequency

End foreach
```

```
FillLinearCustomerID()


Foreach CustomerID in the system
        Selects all rows with that CustomerID within the Transactions Datatable

        If there is move than one item selected
                Create a list call money
                For each row selected
                        Add MoneyValue to list
                End

                Create a decimal array called X values and fill it from 1 – List length
                Turn the List into a decimal array

                pass the Xaxis array and the money array as parameters to the Regression object
                Row["CustomerID"] = customer ID tested
                Return the coefficients from the LinearRegression object and fill the remaining row
                space

                Row[slopex2] =
                Row[slopex]=
                Row[Yint]=
                Row[Rsq]=
                Row[ProductMomentcorilation]=
                Add rows to datatable of CustomerIDregression

        End if

End foreach
```

Data processing Program:

For the data handling I am using data tables as its efficient way of storing the large amounts of tabular data and includes many useful in-built methods that speed up the processing. For example Indexesof("") returns all data rows that return true to a query which can be as simple as "customerID = 17850" to more broad requests that return hundreds of data rows returning

them as an array of datarows. The entire system I will be proceeding with will use datatables as the backbone storage of the tabulated data used. Below is a data dictionary of the tables in the original dataset that has been inputted into a SQL server to be processed:
Table Name: CustomerData

Name:	**Format:**	**NotNull:**	**Key:**	**Example:**
StockCode	Varchar	Yes	Yes	85123A
Description	Blob	no	No	WHITE METAL LANTERN
Quantity	Integer	Yes	No	4
InvoiceDate	Date	Yes	No	,29-11-16
UnitPrice	Real	Yes	No	9.99
CustomerID	Varchar	Yes	Yes	17850
Country	Varchar	No	No	France

Lists are to be used considerably in the data processing aspect of the system considering one of the ideal options is to have a flexible number of rows loaded inThis is useful for any user defined subroutines as the programmer cannot predetermine the number of items that will be used. Allows for the searching of specific items and indexes making it a useful interface structure between Data stored in the SQL database and the Subroutines that process the data.

DFD and accompanying system pseudocode:

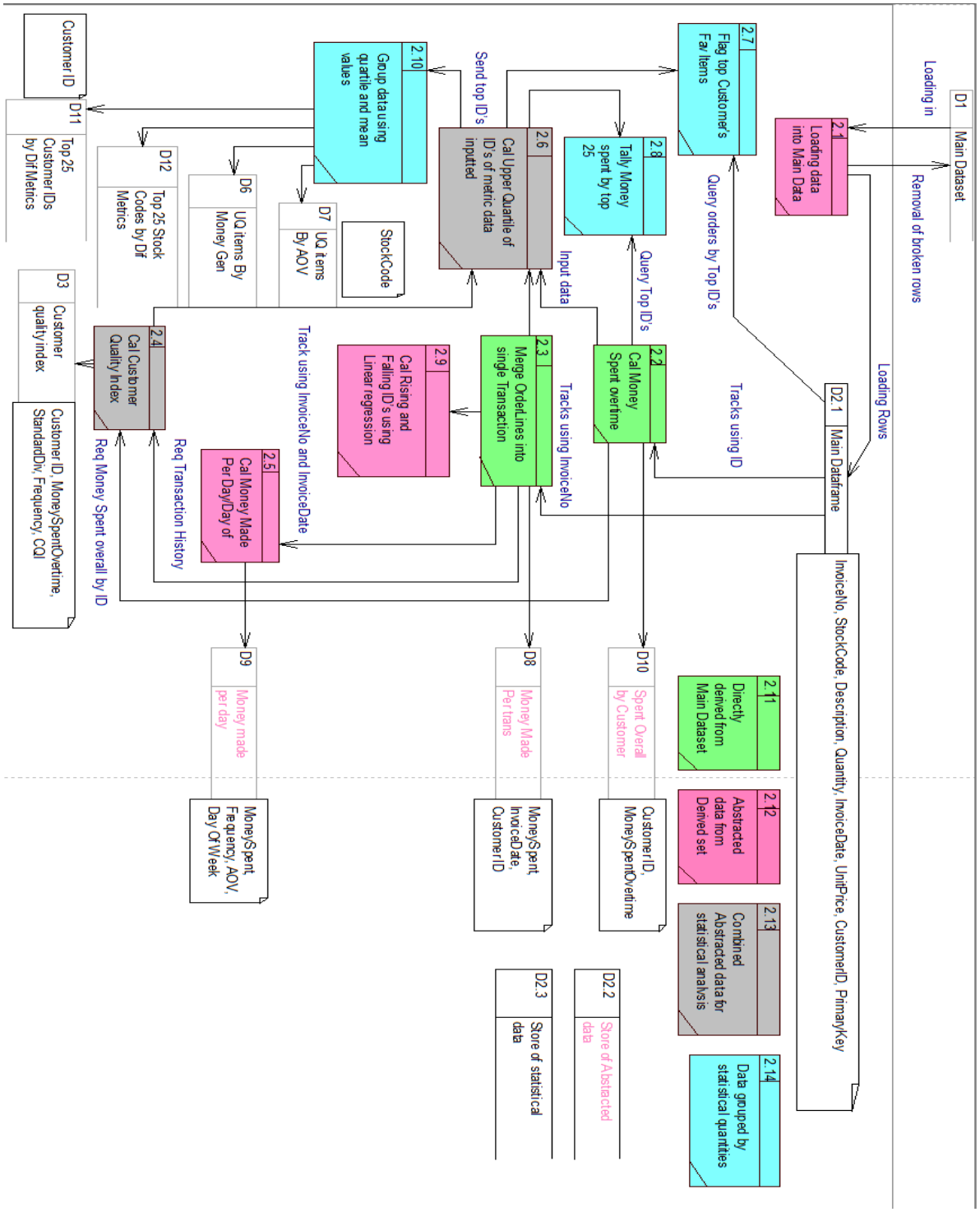


Figure 1 Required Logical Data Flow Diagram

The UML diagram of the class structure:

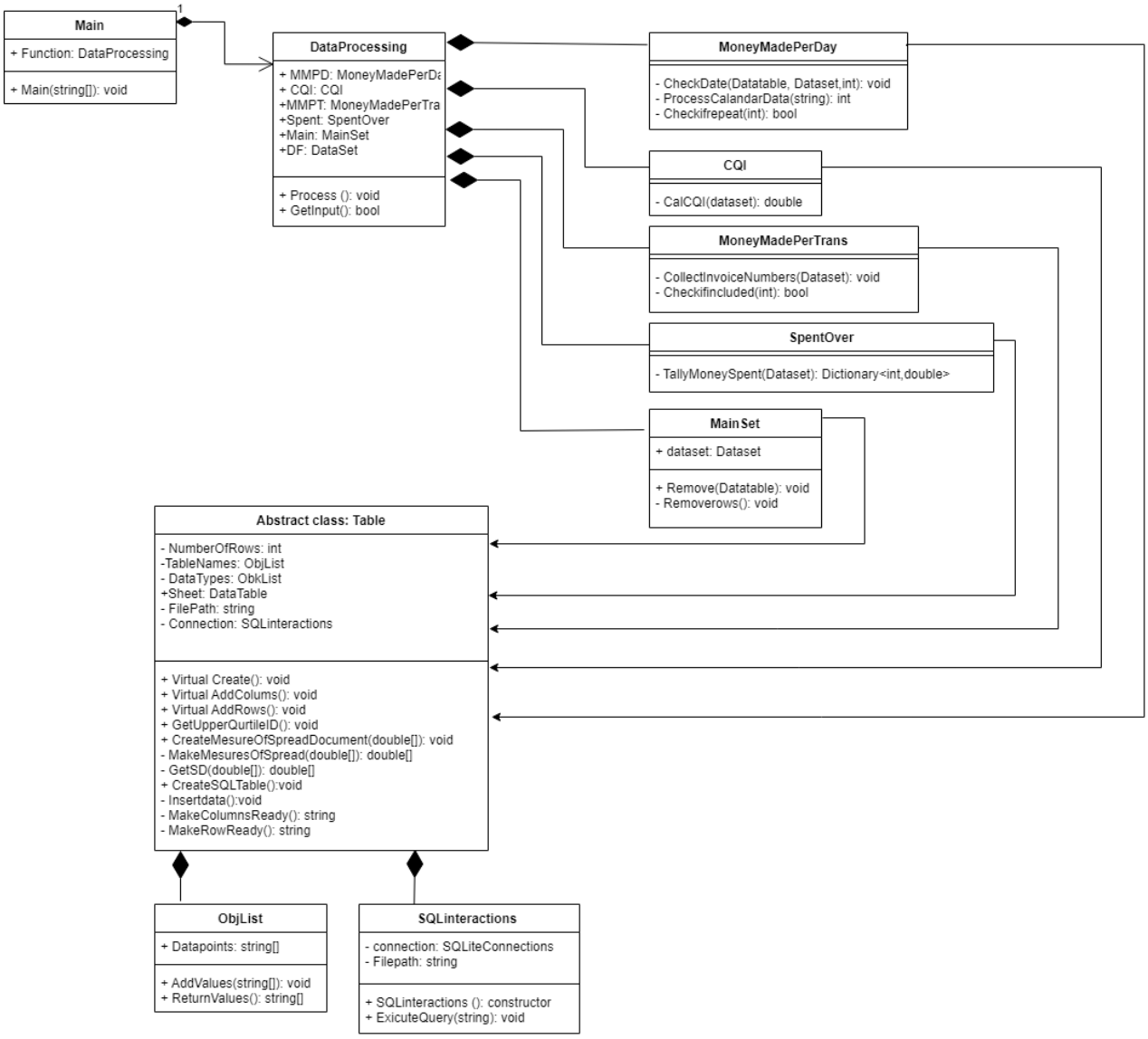


Figure 2 Program Overview UML

Briefly explain each class and its functionality

Below is a UML of the data processing program showing interactions between the tables created showing how data is abstracted from the original dataset. For example, how the data for Money Made per Transaction is used in the creation of the Customer Quality Index and MoneyMade Per day. The DFD is colour coded between degrees of abstraction from the original dataset and the type of data stored in the datastore and the note next to a data store represents the columns within the data store separated by commas. A vertex between two Processes means that data from the sending process is used in the creation of data in the receiving process.

The table class is an abstract class which all the other classes inherit from. It contains any general methods required for the general functionality of a table. It includes a Datatable which is then filled by each of the separate sub classes in their bespoke ways. Any general methods like Saving to SQL will be located here. Each sub table is made to produce a different data table which they are named after.

The Mainset class:

InsertData()
First the Main Data Table is called. Its tables and their data types are then assigned following the sequence of InvoiceNo, Stockcode, Desc, Quantity, InvoiceNo, UnitPrice, CustomerID following the datatypes described in the data dictionary above. Utalizes the SQLCon class that is explained in the Linear regression program.

I will be making extensive use of Datatables in this program as a fast and efficient way of storying hundreds of thousands of data points. The code for the setting up of the data tables is repetitive so I won't go through it for every column of every Datatable. Structured English for how I generally created each column in datatable as well as generally receiving tabular data from a SQL database. In appended A

I will be converting the SQL month data of Nov, Dec, Jan ect into their neumerical month as they are read in instead of doing it each time date is used in the program.

Remove()
With the Main dataset there is a chance of broken rows or miss input values in what it is receiving from the SQL database. Because of this a try catch is used around the fulling of the datatable function storing the database primary key of any caught errors. These primary keys are written to a text file and red back into the program

```
Int Y=0;
Try

        int row1 = rdr.Getint(0)
        int row2 = rdr.Getint(1)
        int row1 = rdr.Getint(2)
        primary key = Y
        Y++
Catch

        write Y to Txt file

end try catch
```

These primary keys are them removed from the database as well as any known error cases, for example removing any rows will null values as for the purposes of the program they aren’t useful. As I haven’t worked through the dataset yet, I don’t fully know how broken rows that are valid for the reading process but brake the rows will be identified and delt with but this will have to be a iterative process.

```
Foreach Line in text file
        Remove from table row with primary key of line
End foreach

Remove from table row  where any column is null or “ “
Remove from table row  where any column contains known error state
…
…
...
Remove all rows from Datatable and re fill from the now cleaned database
```

SpentOver Class:

Create():
A process to create a dataset that compiles Money spent by a customer over the time recorded for analysis:

```
Datatable MoneySpentOvertime = new datatable (CustomerID, MoneySpentOvertime))
Create a Dictionary of String keys of CustomerIDs and values of a double
Index =0
while Index < Number of Rows in MainDataTable
        If the Dictionary contains the CustomerID in the row MainDataTable[index]
                Increase the value associated with the CustomerID by the value of Money Spent
                present in the row
        Else
        Add to the dictionary the Customer ID and the value of Money Spent in the row
        MainDataTable[index]
        End else
        Increment index
End while
Foreach item in dictionary
        Datarow row = new datarow
        Row.add item1
        Row.add item2
        MoneySpentOvertime. Add Row
End foreach
Normalized table MoneySpentOvertime(CustomerID, MoneySpentOvertime))
```

Example of the Data

PrimaryKey	CustomerID	MoneySpentOvertime
1	17850	5391.210000000009
2	13047	366.6300000000001
3	12583	1586.0199999999998
4	13748	204
5	15100	876

MoneyMadepertrans Class:

Create():
Process to create a datatable grouping the data of the MainDataTable into individual Transactions.
Create a List of double values called InvoiceNumbersList

```
Create a Datatable PerTransaction with columns (IncoiceNo, MoneySpent, InvoiceDate,
                                                CustomerID, Year)
Index =0
while Index < Number of Rows in MainDataTable
        If the List contains the Invoice Number in the row MainDataTable[index]

        Else

                Select all rows in MainDataTable that have the Invoicenumber
                                                in row
                MainDataTable[index]
                ForEach row selected
```

```
                Tally the money spent in each row
            End foreach

            tuple int int day&year= GetDate(Invoicedate in row Mainset[Index]) tuple

            Create a new row in PerTransaction called Row
            Row.Add invoice number invoice number
            Row.Add tally found
            Row.Add day&year . item1
            Row.Add day&year . item2
            Row.Add CustomerID  in row Mainset[Index]
            Row.Add Invoicedate in row Mainset[Index]
            Datatable PerTransaction. AddRow (Row)

            InvoiceNumbersList. Add (invoice number in row Mainset[Index])
        End else
        Increment index
End while

Methood GetDate(string Date)
        //example of text in date is 15/3/2017
        Number day = substring Date until the first "-" is encountered
        Number Month = substring Date start at first "-" until "-" is encountered
        Number year = substring Date start at second "-" until end

        Number NumericalDate = dayOyear = day + (30) * (month - 1);

        Tuple int int day&year = new Tuple int int (NumerialDate, year)

        Return day&year
End method
```

Normalized table
PerTransaction(IncoiceNo, MoneySpent, InvoiceDate, CustomerID, Year)

CQI Class:

Create():
A specific new measure of customer value is a value I call Customer Quality Index which is money spent overall during the tracking multiplied by the number of purchases that customer did all divided by the standard deviation of their money spent per each transaction as a way of checking consistency.

(MoneySpent x Frequency) / StandardDiviation

Page Break
Using the data from Money spent overtime and Transactions Datatables. Multiply the MoneySpentOverall of a customer by the number of times the customer ID appears in the transaction Datatable. Then use a Standard deviation() Method on the transaction history of how much they spent, this method is detailed bellow.

Load the datatables MoneySpentOverall and MoneyMadePerTrans

Create a Datatable CQI with columns (CustomerID, MoneySpenOverttime, StandardDiv, CQI, Frequency)

```
For I =0 to  I< MoneySpentOverall number of rows
        Select all rows in MoneyMadePerTrans that have the  CustomerID in
                                                        row MoneySpentOverall [index]
        Create a numerical array with the size of the number of rows selected

        Fill the array with the Money spent in each row called MoneyArray
        StandardDeviation = GetStandardDeviation(MoneyArray)

        FrequencyOfSpend = length of MoneyArray

        Create a new row in CQI called Row
        Row.Add PrimaryKey = I
        Row.Add CustomerID  in row MoneySpentOverall[Index]
        Row.Add MoneySpent in row MoneySpentOverall [Index]
        Row.Add  StandardDeviation= StandardDeviation
        Row.Add FrequencyOfSpend= FrequencyOfSpend

If (StandardDeviation = 0)
                CQI = 0
        Else
                CQI = (MoneySpentOverall[Index] * FrequencyOfSpend) / StandardDeviation
        End if
        Row.Add CQI
        Datatable PerTransaction. AddRow (Row)


End for

GetStandardDeviation method(numerical array SDdata[])
Number sum = 0
Number sumSq = 0
foreach (Number item in SDData)
                sum = sum + item
                sumSq = sumSq + (item * item)
End foreach
SD = Squarroot of ((sumSq / SDData.Length) - ((sum / SDData.Length) * (sum
/DData.Length))) //Standard SD equation
return SD
```

Normalised Table:

DataTable (<u>CustomerID,</u> MoneySpenOverttime, StandardDiv, CQI, Frequency)

The MoneyMadeperDays Class:

Create():
You should really explain this using a simple algorithm or structured English.
Works by first having each transaction have its invoice date attached. Then using a way of converting the date data into a value for the day of the year. The day value is found through brute force converting, by turning the Months into 30x Month value then adding the day value.

Create Datatable MoneyMadePerDay with columns (PrimaryKey<u>,</u> Day, MoneySpent, FrequencyOfSale, Year)

```
For I =0 to  I< PerTeansaction number of rows

        Tuple int int Day&year = GetDate( Invoice data value in row MPPerTrans[I])
        // as seen in 2.3
        Select Datarows[] in MMPerDay with (day =  to Day&year. Item 1 and Year =
        Day&year. Item 2)
        // there will only ever be 1 data rows in the array

        If datarows[]. Length >0
                Datarow[0] Moneyspent = Datarow[0] Moneyspent + MoneySpent in row
                PerTeansaction[i]

                Datarow[0] Frequencyofsale = Datarow[0] Frequencyofsale + 1
        Else
                Datarow row = new datarow;

                Row. Add Primarykey = I
                Row. Add Day= Day&year. item1
                Row. Add MoneySpent = MoneySpent in row PerTeansaction[i]
                Row. Add Frequencyofsale = 1
                Row. Add Day= Day&year. item2

                MoneyMadePerDay.Rows.Add(Row);
        End else
End for
```

Run through the Transaction Datatable and create a string assigning it the invoice date. Then substring the value between the "-". create a int value for day month and year and assign their corresponding substring. Then use a nested switch case of the year and then month and add to the value for Day what is required for the month to be accounted for. For example if moth value if 4 add 4x30 to the day value. Check if this day value is included already in the datatable if so then add the money spent to the value included and include the frequency of sale by 1.

Normalised Table:
MoneyMadePerDay(PrimaryKey, Day, MoneySpent, FrequencyOfSale, Year)

The abstract table's class's methods:

GetMesuresOfSpread() and CreateMesureOfSpreadDocument()

This is a process that takes in different datasets and the column's name that they want analysed, then creates several simple Statistical values such as Mean and the Quartiles. This method takes in a datatable and a name of a column within the datatable. Use the name of the column to create a list of data values within this column. Then use in built methods within the Math.net library to work out the mean, Lower quartile, Upper quartile, Interquartile range, Min, Max and Medium. Load these measures of spread into an array and return the array.

CreateMesuresOfSpreadDocument()

If file directory in bin of Mesures of spread does not exist create one creating a file to store all mesures of spread

Load

GetUpperQuartileID()

With the upper quartile values made from 2.6 we can then read this value for different metrics:
Take in a datatable and column. Read the file created in 2.6 of the upper quartiles of this column. Run through all the data and add to a list all the IDs, be that Customer, day or Transactions, that are over this upper quartile. Store this list in a .txt file.

Once this list is attained run through the original dataset. Create a list of stock codes that the UP-quartile customer IDs buy and store them in a TXT file.

GetSD()

Gets the standard deviation of whatever data is inputted into it

Saving functions:
For the Saving functions I will use SQL queries to create the tables and columns in the database to then be filled with the data created by the program. For this I will need a way of storing the Column names and Datatypes as SQL has a very specific syntax for the creation of tables and columns. The problem arises from the heavy use of inheritance as generic collections cannot be inherited so I will need to create my own List Object to be inherited by each of the classes. I will also be having each table have its own SQL interactions object to easily automate much of the saving process so the SQL connection would not have to be passed as a parameter.

CreateSQLTable()

Checks for the existence of a table within the SQL database the program is connected to that shares the same name as the table object . If the database has a table a option will come up asking the user if they would like this data to be re written or added to. If re written the table is dropped and a new table is created and the MakeColumnsReady() method is called and the string returned is passed to the SQLInteractions object's method called Executequery() as a parameter to be executed Creating a new table. If add to the table is chosen or after the new table has been created the InsertData() Method is called.

MakeColumnsReady()

takes in a bool to see if it is coming from Insertdata() or the CreateSQLtable() method **fix uml**. It usesThe Objectlist object storing values for column name and datatype using the ReturnValues() method to extract them as string arrays. Using a for loop place these values in the correct SQL syntax to create the table.

This process is psudocoded bellow.

```
String[] tablenames = L1istTablenames.ReturnValues()
String[] datatypes = L1istTablenames.ReturnValues()

String  create = "Create table DatatableName("
For I_=0 to  I< datatable.columns[].length
If (comingfromInsertData() = true)
```

```
        Create = create + tablenames[I]

Else

        Create = create + tablenames[I] + datatypes[I]

End for
Create = create + “);”
Return create
```

MakeRowsReady()

inserts the rowdata in the datatable of the object into the corresponding SQL table. Uses a foreach datarow in datatable to run through the entire datatable of the object adding each piece of data to a string to be passed into the table objects SQLinteractions object’s method ExicuteQuary(). The loop is as follows:

```
Foreach Datarow item in Datatable.rows[]
        String data = “Values (”
        Foreach point item in datarow
                Data = data+ converttostring (item)
        End foreach
        Data = data+ “);”
        String Tabledata = “Insert into “ + Datatable.name + “(“
        MakeColumnsReady(true)+”)”

        SQLinteractions.Exicute(Tabledata +  Data)
End foreach
```

File Structure:

As the Data processing program operates it will write to Text files as the data is produced to limit data lost if the program crashes before it can upload the large Datatables into the SQL server. Here is how the Data stored will be structured. They will be stored as simple text files as SQL has operations for uploading .Csv files manually if the program crashes.
Example:
….Bin/Data40000/MoneyMadePerDay.csv - Filepath

MoneyMadePerDay DF 40000 - Notepad

File Edit Format View Help

```
MoneySpent,Frequency of sale,dates,AOV,Day Of Week
46376.4900,121.0000,329.0000,383.2768,0.0000
47316.5300,137.0000,330.0000,345.3761,1.0000
23921.7100,57.0000,331.0000,419.6791,2.0000
31771.6000,87.0000,333.0000,365.1908,4.0000
31215.6400,94.0000,334.0000,332.0813,5.0000
53795.3100,71.0000,335.0000,757.6804,6.0000
```

……

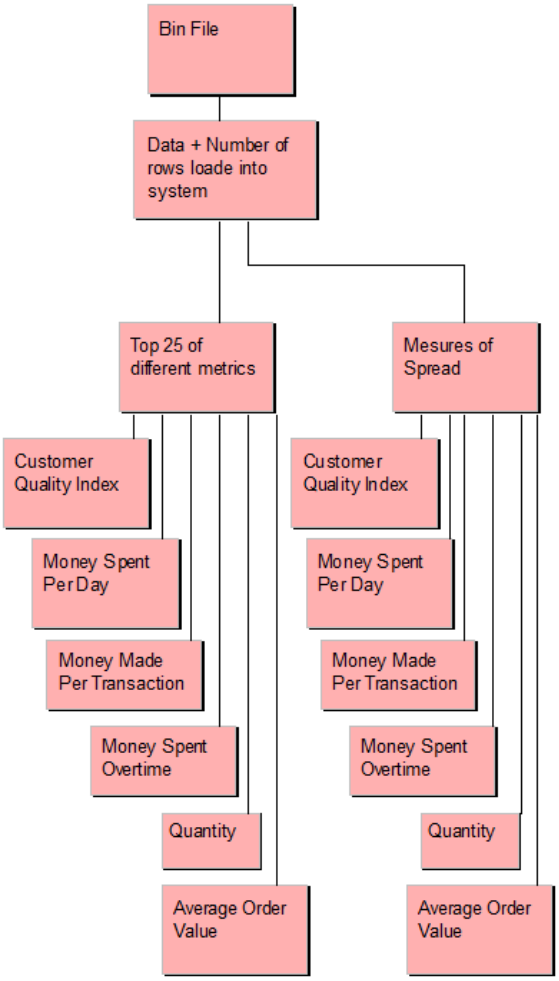


Figure 3Text File structure

The Dashboard program:

After we build a way of presenting the data, we then must draw conclusions from the data and diagrams created by the system. Dashboards present different types of data in different ways as our analysis shows. The myriad of types of data created by both programs must be presented in a digestible and understandable way to be effective. The dashboard program will be a windows form and utilise SQL requests to the central database to receive data to display. This will allow for any future expectations of the dataset or future specificity in what region of the dataset is being analysed. I decided to use a C# windows form as it easily allowed me to input and output data directly created by the Data processing and statistics program including datagridviews which allow for the presenting of large datasets easily without poor performance. Here is a mock up for the design of the final dashboard.

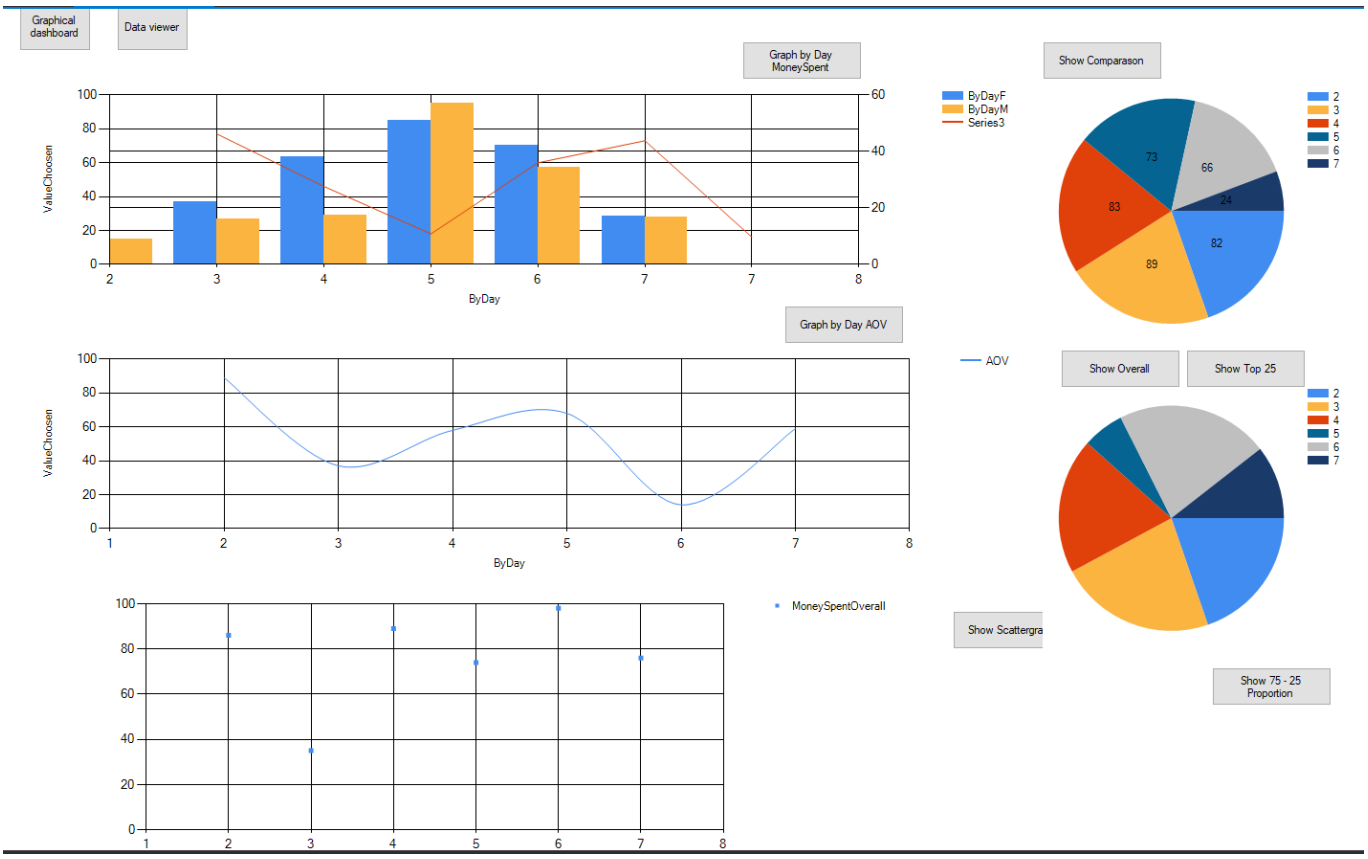


Figure 1 Graphical Presentation of the Data

This figure shows the systems graphical element which presents the data through graphs that have been found to be effective in the analysis of said systems. The two different bar charts show Frequency and MoneySpent per each day with a line graph following the linear regression of each quarter in the system. The a curved line graph bellow the per day shows the Average order value and how it Changes through the year.

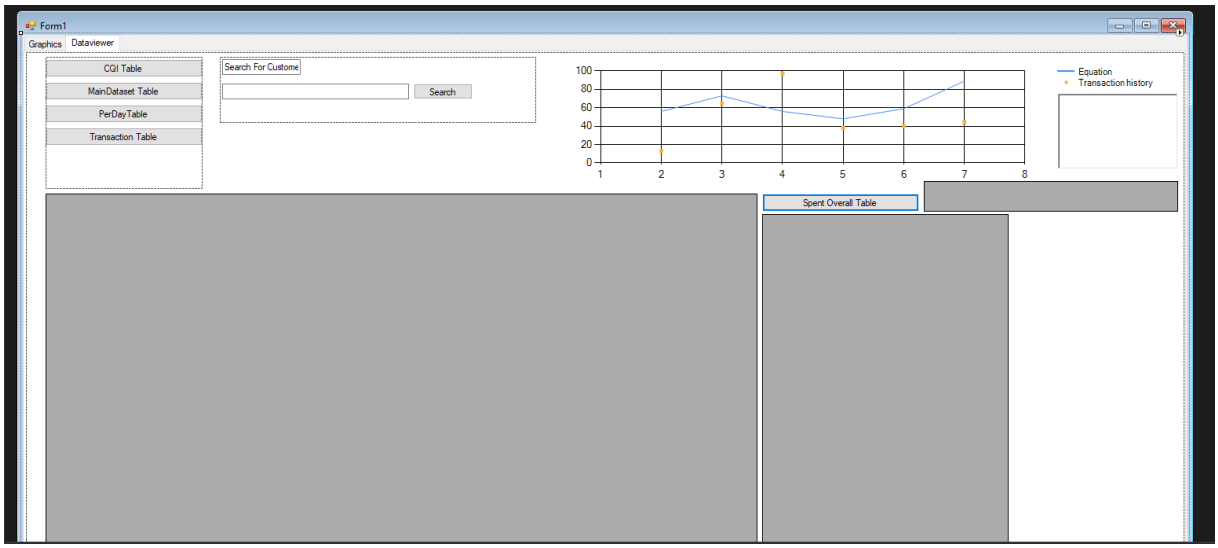
The pie chart next to the bar chart is to show the comparison between the money spent overall of the top 25 and the bottom 75 to show how centralized the data is to the top.

The secondary pie chart below is to show is the customers purchasing patterns show a positive or negative prediction based on the linear regression system with buttons to switch from overall to the top 25. With an individual equation for each customer ID we can analyse

each to make predictions about their future purchasing patterns seeing if they are increasing in value or decreasing by checking their gradient through differentiation.

The lowest graph is a scatter graph to present the money spent overall of every customer in the dataset to present how varied the customers in the system are.

Figure 2 Datagridview and Lookup System



The figure above shows the Data view aspect of the system. This is where the user can view each of the tables uploaded to the database individually and save them to a .csv file to allow each department access to the data created. The table names are on buttons to the top left. The large panel is to present the tabular data of the table selected by the user.

Next to the table's buttons is a look up section were the user can write a customer ID and only present their section of the graph selected by the user. This allows for the specific looking up of customer's data on each graph to be saved or viewed by the user. The search also presents their transaction history on the graph to the right of the search bar through a scatter graph as well as if applicable their linear regression equation on a line graph or a curved graph. The graphs equation is also presented on a data table below it for the user to view. The general trend that the graph shows is also written in the text box to the right of the graph.

Classes within the forms application:

SQLCon Classs:
Previously explained

DataFromSQLTables Class:

Class to make datatables for use in the dataview grid and for use in the graph presenting.

Attributes:
Dictionary <string, Datatable> Optionlist = new Dictionary <string, Datatable>()

Public DataFromSQLTables()
Calls the following methoods in this order
CreateLinregTableCustomerID();

```
CreateSpent();
CreateMPPertrans();
CQI();
FindMMPD();
Mainset();
```

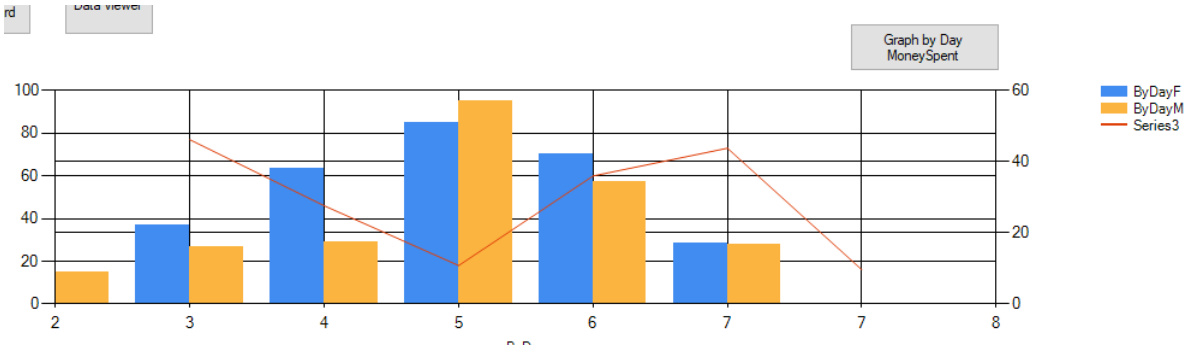
These all Create a Datatable for use in the dataviewgrid and receives Tabular datafrom he SQL Database as explained in Apendex A Then adds each datatable to the Dictionary with its name as the key. As they all have the same functionality I wont go through them individually.

TopIDs Class:

Receives the top25 tables from the SQL database using this data to fill lists for use in comparison.
Attributes:

```
public List<int> CQI = new List<int>();
public List<int> Frequency = new List<int>();
public List<int> MMperDayMoney = new List<int>();
public  List<int> MMperDayFrequency = new List<int>();
public List<int> PerTransaction = new List<int>();
public List<int> Spent = new List<int>();
```

Important Functions within the form class:



Clicking the button above the bar chart:

Load the MoneyMadePerDay datatable into the method

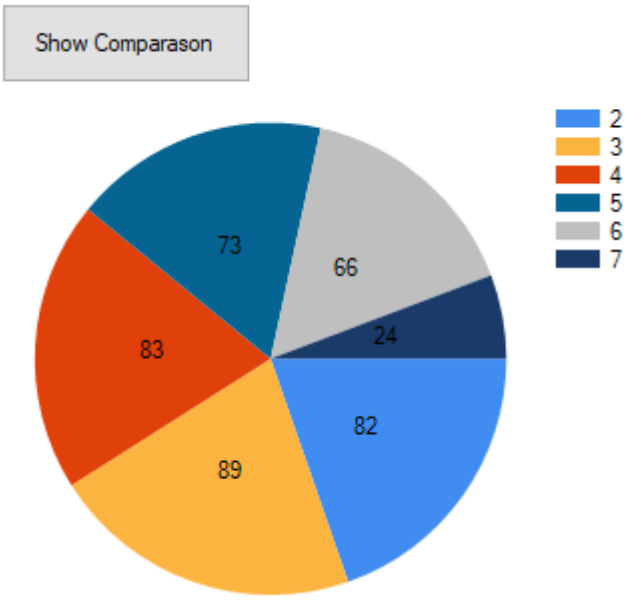
Create Integar y = 1

Foreach DataRow in the table

Add to the chart series of name “ByDayMoney” (y , datarow value in column of name Money)

Add to the chart series of name “ByDayFrequency” (y , datarow value in column of name Frequency)

End foreach



Clicking on button named Show Comparison:

Load in the list of 25 Customer IDs by MoneySpent into the method

Load the SpentOvertime datatable into the method

Create Decimal Top25 = 0

Creaye Decimal Bottom75 = 0

Foreach DataRow in MoneySpent

    If Datarow value in column of name CustomerID is within the List of top 25

        Top25 = Top25 + Datarow value in column of name MoneySpent

    Else

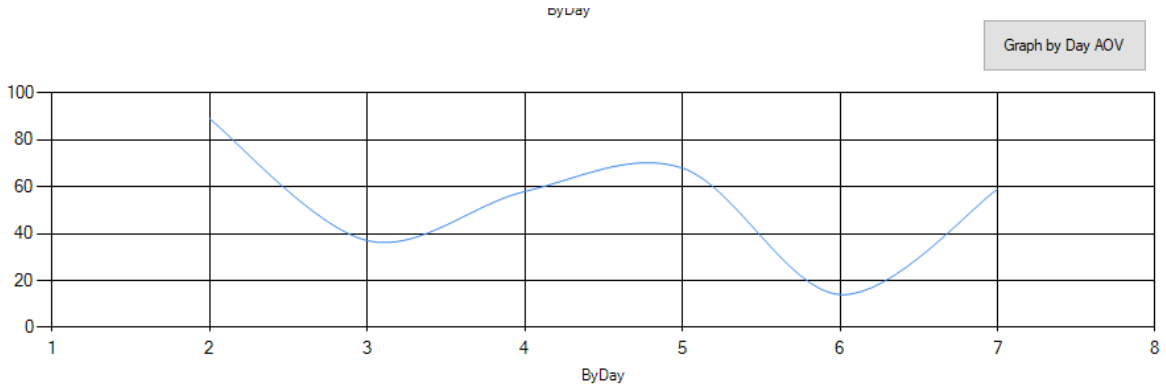
        Bottom75 = Bottom75 + Datarow value in column of name MoneySpent

    End if

End foreach

    Add to the chart series of name “Top25” (Top25)

    Add to the chart series of name “Top25” (Bottom75)



Clicking the button above the LineChart chart:

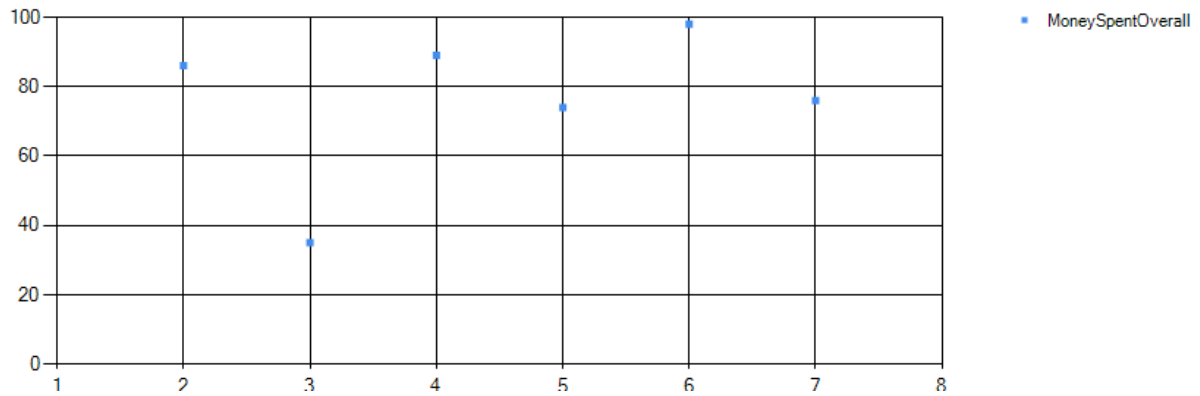
Load the MoneyMadePerDay datatable into the method

Create Integar y = 1

Foreach DataRow in the table

    Decimal AOV = datarow value in column of name Money / datarow value in column of name Frequency)

End foreach



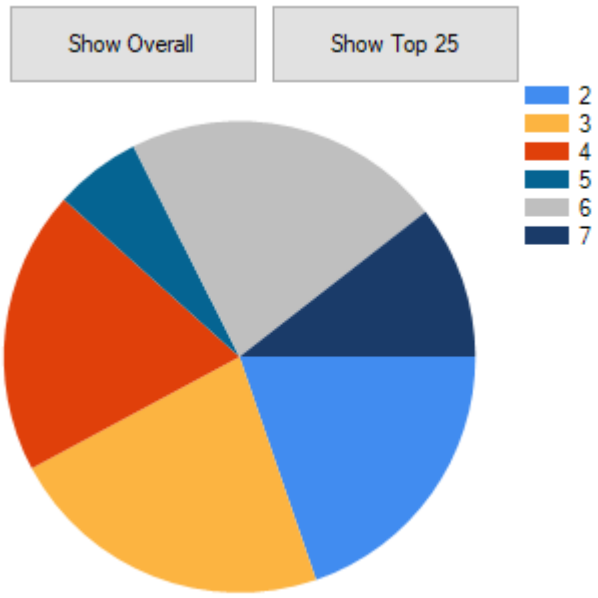
Upon loading of the Form:

Load the SpentOvertime datatable into the method

Foreach DataRow in MoneySpent

Add to the chart series of name “SpentOverall” (Datarow value in column of name MoneySpent)

End foreach



Clicking show Overall:

GetGradient(decimal[] Equation, decimal Xvalue) method:

Getting the gradient of the equation at a specific point is simple through the use of differentiation and can be easily made for both polynomial and simple equations. This can be seen bellow.

$$y = bx + c \quad \frac{dy}{dx} = b$$

$$y = ax^2\ bx + c \quad \frac{dy}{dx} = 2ax + b$$

To get the next prediction of if the equation shows a rising or a falling trend I will load in the last actual X value in the transaction history to predict its general trend.

Psudocode:

```
Decimal GetGradient(decimal[] Equation, decimal Xvalue)

        Decimal Xsq = Equation[0]
        Decimal X = Equation[1]

        Decimal Gradient = 2(Xsq)(Xvalue) + X

        Return Gradient

End methood
```

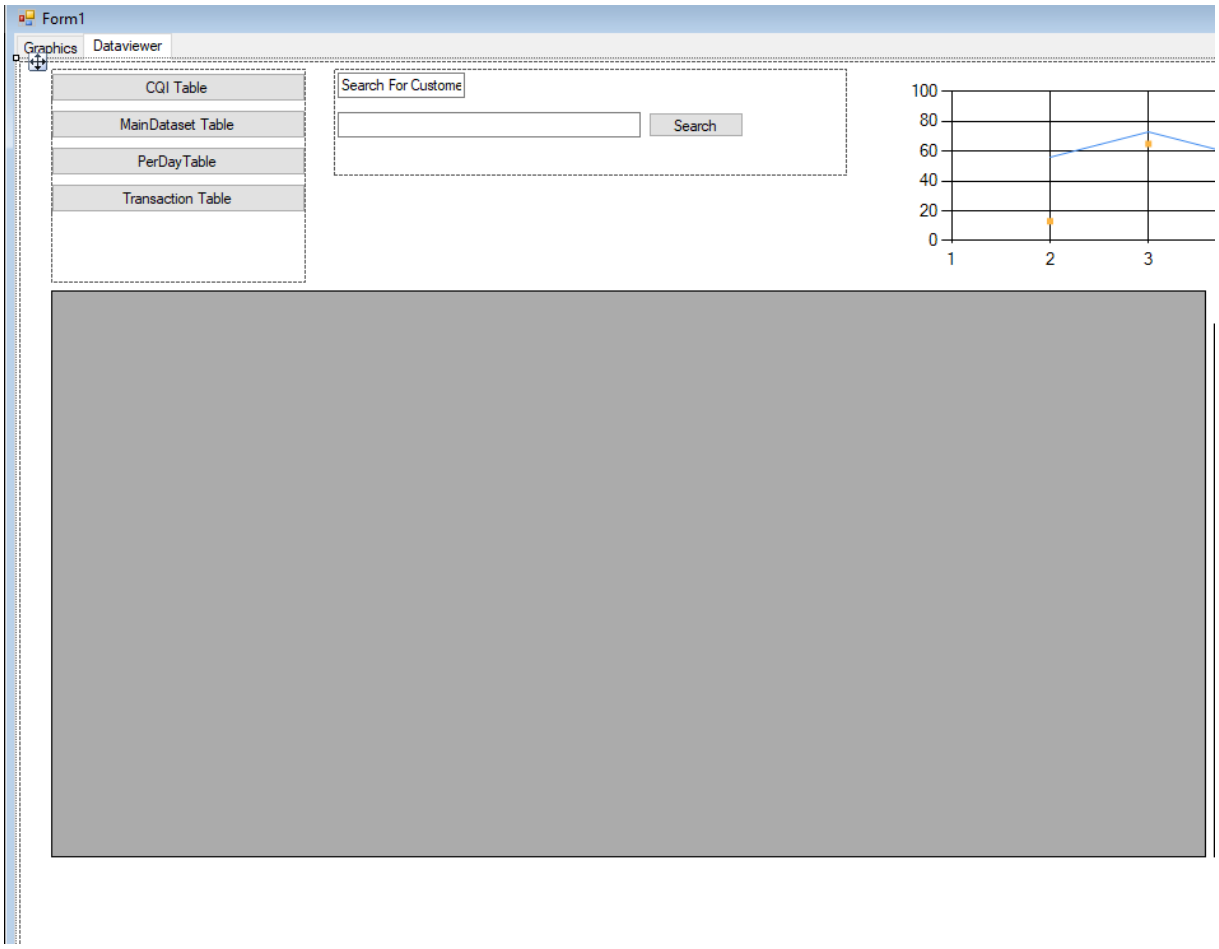
Structured English of overall process:

```
Load the CustomerLinReg datatable into the method
Load the PerTrans datatable into the method
Create integrar NumPositive = 0
Create integrar NumNegative = 0

Foreach datarow in CustomerLinReg
        If every row is not assigned to 0
                Decimal[] transactions = Use query on PerTrans datatable select[] where
                customerID = datarow's columnID

                Load all equation data in the datarow to a decimal[] called Equation leaving
                Equation[0] as empty if it is not a quadratic equation

                Create decimal called Last = transactions[Last]



                If GetGradient(Equation, Last) > 0
                        NumPositive increment
                Else if GetGradient(Equation, Last) < 0
```

```
                NumNegative increment
            End if
        End if
End foreach

Add to the chart series of name “Pos” (NumPositive)
Add to the chart series of name “Neg” (NumNegative)
```

Clicking show Top25 Overall:

```
Load the CustomerLinReg datatable into the method
Load the PerTrans datatable into the method
Load the Top25 Spent into the method
Create integrar NumPositive = 0
Create integrar NumNegative = 0

Foreach datarow in CustomerLinReg
        If( datarow CustomerID is within the top25 Spent List)
            If every row is not assigned to 0
                Decimal[] transactions = Use query on PerTrans datatable select[]
                where  customerID = datarow’s columnID

                Load all equation data in the datarow to a decimal[] called Equation
                leaving Equation[0] as empty if it is not a quadratic equation

                Create decimal called Last = transactions[Last]


                If GetGradient(Equation, Last) > 0
                    NumPositive increment
                Else if GetGradient(Equation, Last) < 0
                    NumNegative increment
                End if
            End if
        End if
End foreach
```



Clicking any of the table name buttons

All of the table request methods follow the same method as datagridviews interact directly with datatables as data sources. For ever button / table requested I will use this process

Dataviewgrid clear of data

Dataviewhrod bind to source DataFromSQLTables.Optionlist[“TableNameRequested”]

Technical solution:

The technical solution can be broken down into 3 separate parts overlapping in some of their objectives. The three programs can be run separately but they first must be completed sequentially as they use data created in previous systems. For example, the Data processing program has a test to see if the database required for both its operation and for the rest of the systems is even where it needs to be for their operation.

Data Processing -> Statistical analysis -> Dashboard

The statistical analysis utilizes tables created by the Data processing while the Dashboard uses data created by both previous systems.

Preliminary Setup:

They all utilize a central database stored on the users computer as a .db file. The file is stored within the users Document file to allow all the applications access to the database. It contains the dataset imported into the database with these columns and data types:

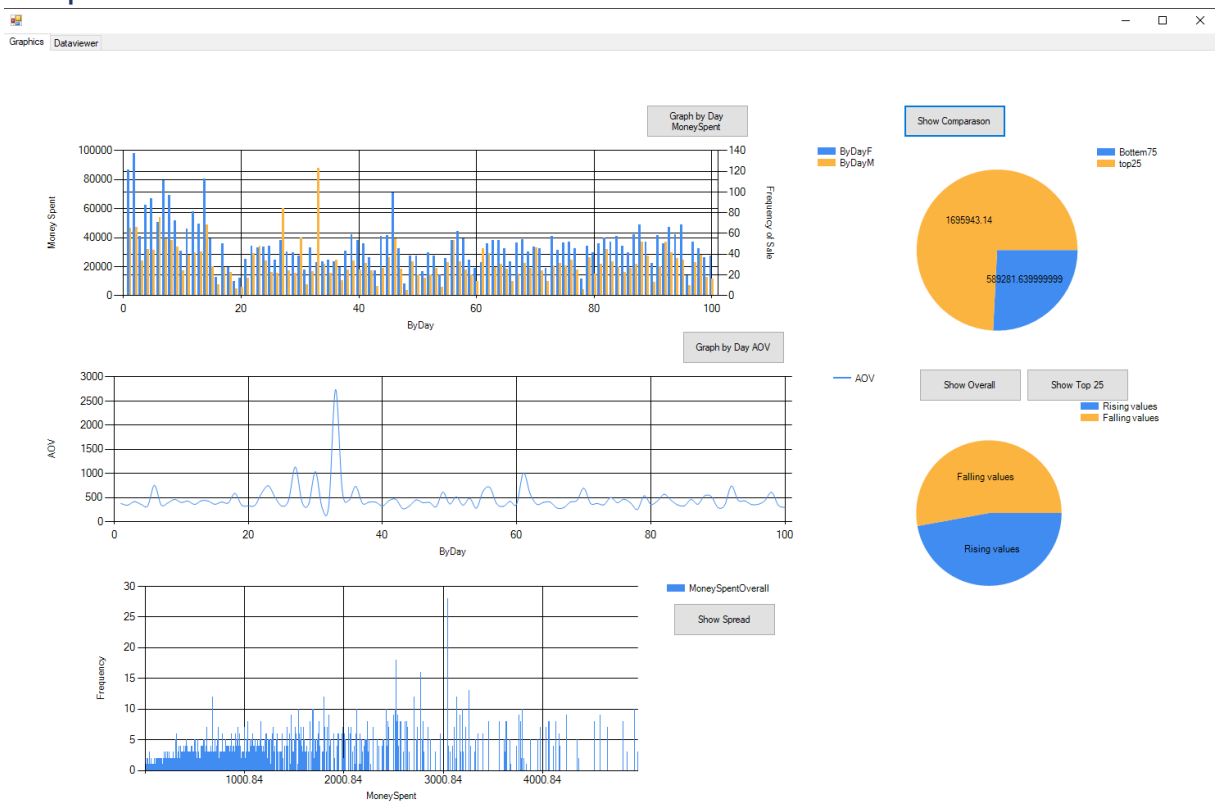
CustomerData Table name: CustomerData ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	InvoiceNo	INTEGER								NULL
2	StockCode	TEXT								NULL
3	Description	TEXT								NULL
4	Quantity	INTEGER								NULL
5	InvoiceDate	DATE								NULL
6	UnitPrice	DOUBLE								NULL
7	CustomerID	INTEGER								NULL
8	Country	TEXT								NULL
9	PrimaryKey	INTEGER								NULL

Dashboard program

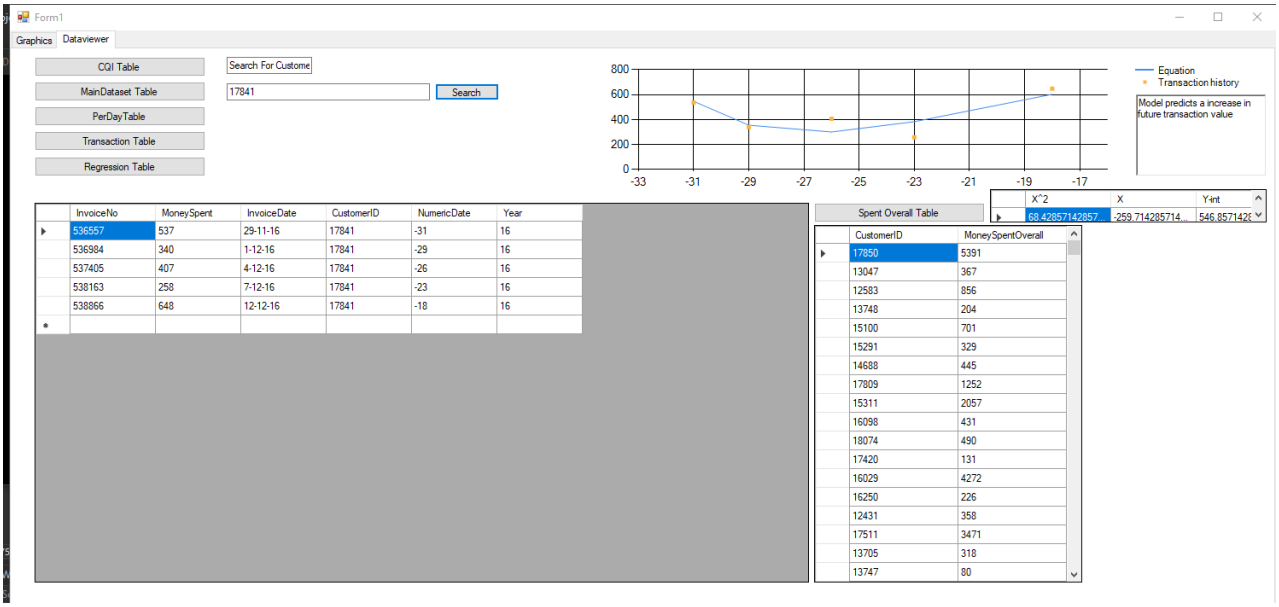
This is the GUI that is the final end user experience of all of the systems. The Graphical element presents the graphs for easy quick understanding of the dataset while the dataview grid allows for specific queries of CustomerIDs and viewing of tables from the SQL database.

Graphical element



This is a screenshot of the working graphical element of the dashboard. It requests data from a central database setting them into datatables to then be presented on the dashboard through charts. This system allows for the freeform changing of data within the database to still be presented consistently by the dashboard system. It utilizes the same graphs in the same way to the design document following the ways of presenting data found to be effective in the analysis. With the Money spent, frequency and AOV being represented as bar a line graphs with comparison values being represented as pie charts. The scatter diagram is to show how Braud the range of values of money spent overtime is.

Data Viewer

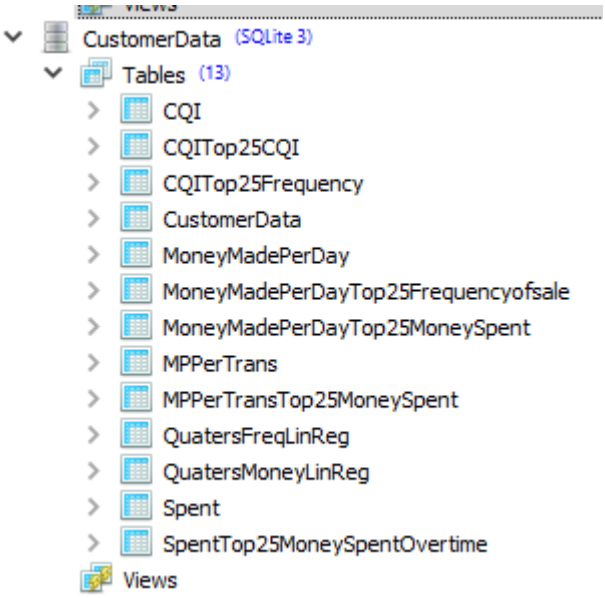


Here is the It includes the options for displaying each table created by the program in a dataviewgrid format. These tables include: the table of CQI including metrics like frequency and CQi; Maindataset which is the original dataset; PerDay table which includes data like money spent and frequency per day tested; The transactions table which includes every invoice number grouped into individual transactions and finally the regression table which holds the data for each customer ID's created regression equation.

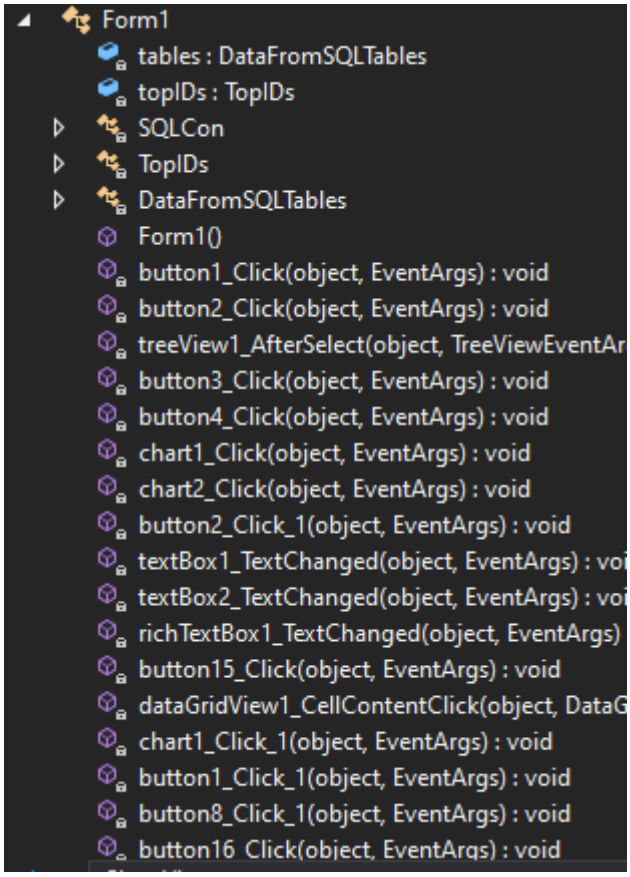
It has a search bar that allows for the searching of specific customer IDs which pulls up their transaction history on a graph which includes, if applicable, the regression equation represented by a line chart. When you search a customer id the datagridview displaying the table is also searched for that customer ID and only shows that customer IDs data. In the case shown we see a customer ID get searched; his data get displayed on the equation graph; as well as displaying his rows on the table selected, in this case his transactions as the transaction table was selected.

The MoneySpentOvertime table is also displayed next to the main grid upon loading the system. This is as a way of allowing the user to quickly identify important customers at a glance and search their customer IDs for their transaction history on the graph without having to know many customer ID's of hand. Next to the spentOvertime grid there is a smaller grid which displays the regression if applicable to what is being shown on the graph.

To the right of the graph is a textbox which writes if the model predicts a general increase or decrease in transaction value using the differentiate of the regression equation.



This is the set of tables that the dashboard requests datafrom all but the CustomerData is created by the system either in the Data processing or the Regression system.



The dashboard system uses an Object-oriented approach with 3 main classes alongside the forms event functions.

DataFromSQLTables

Line 93

Class for Constructing all the datatables required for the forms functionality and then fills them with tabular data from the SQL database.

```
DataFromSQLTables
    Optionlist : Dictionary<string, DataTable>
    SQLtables : DataSet
    DataFromSQLTables()
    CreateLinregTableCustomerID() : void
    CQI() : void
    FindMMPD() : void
    Mainset() : void
    CreateMPPertrans() : void
    CreateSpent() : void
```

```
class DataFromSQLTables
{
    public Dictionary<string, DataTable> Optionlist = new Dictionary<string, DataTable>();
    public DataSet SQLtables = new DataSet();

    public DataFromSQLTables()
    {
        CreateLinregTableCustomerID();
        CreateSpent();
        CreateMPPertrans();
        CQI();
        FindMMPD();
        Mainset();

    }
```

Dictionary used to store all created Datatables to allow the other methods with the program to easily select a Datatable when required, it is added to with the Datatables name as the key.

Simply calls all methods within the class upon construction

CreateLinregTableCustomerID():

Creates a Datatable to hold the tabular data of the regression equations of each CustomerID in the system, requesting data from the CustomerIDRegression SQL table. The regression equation is created by the statistical analysis program and uploaded to the central database.

CustomerData Table name: CustomerIDRegresion

	Name	Data type	Primary Key	Foreign Key	Unique
1	CustomerID	VARCHAR			
2	SlopeXsq	DOUBLE			
3	SlopeX	DOUBLE			
4	Yintercept	DOUBLE			
5	RSquared	DOUBLE			
6	ProductMomentCorilation	DOUBLE			

This is the SQL table structure and example of the data which the method will be sending requests to

CustomerII	SlopeXsq	SlopeX	Yintercept	RSquared	ProductMomentC
17850	0	0	0	0	0
13047	0	-130.5	252.83333333333331	0.89236808551517	-0.94465236225564
15100	0	0	0	0	0
17809	0	378.5	38.83333333333334	0.99969238391418	0.99984618012681
15311	0	-73.68571428571428	527.047619047619	0.59333113192553	-0.77027990492128
16029	-2655.5	5341.5	508	1	0
17511	0	0	0	0	0
13408	0	-168	917.3333333333331	0.44799762964217	-0.66932625052524
13767	0	0	0	0	0
13448	389.5	-799.5	427	1	0
15513	0	0	0	0	0
17920	0	0	0	0	0
13758	0	0	0	0	0
15983	0	-58	441	1	0
14849	0	-126.5	245.16666666666663	0.67365603468841	-0.82076551748256
16210	0	-968	2727.333333333333	0.830665333775	-0.91140843411448
17897	0	0	0	0	0
17377	0	0	0	0	0
12748	0	0	0	0	0
12433	0	0	0	0	0
16955	0	0	0	0	0
18144	0	0	0	0	0
14911	0	0	0	0	0
17841	68.42857142857143	-259.7142857142857	546.8571428571428	0.69698936085491	0
12921	0	0	0	0	0
13468	214.5	-472.5	309	1	0

```
private void CreateLinregTableCustomerID()
{
    DataTable LinRegtable = new DataTable("CustomerIDRegresion");
    string[] dattypes = new string[6];
    string[] Tablenames = new string[6];

    DataColumn column; //MoneySpent,InvoiceDate,CustomerID

    SQLCon connection = new SQLCon("CustomerIDRegresion");

    SQLiteDataReader rdr = connection.rdr;

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.String");
    column.ColumnName = "CustomerID";
    column.ReadOnly = true;
    LinRegtable.Columns.Add(column);

    dattypes[0] = "VARCHAR";
    Tablenames[0] = "CustomerID";

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "SlopeXsq";
    column.ReadOnly = true;
    LinRegtable.Columns.Add(column);

    dattypes[1] = "DOUBLE";
    Tablenames[1] = "SlopeXsq";

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "SlopeX";
    column.ReadOnly = true;
    LinRegtable.Columns.Add(column);

    dattypes[2] = "DOUBLE";
    Tablenames[2] = "SlopeX";

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "Yintercept";
    column.ReadOnly = true;
    LinRegtable.Columns.Add(column);

    dattypes[3] = "DOUBLE";
    Tablenames[3] = "Yintercept";

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "RSquared";
    column.ReadOnly = true;
    LinRegtable.Columns.Add(column);

    dattypes[4] = "DOUBLE";
    Tablenames[4] = "RSquared";

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "ProductMomentCorilation";
    column.ReadOnly = true;
    LinRegtable.Columns.Add(column);

    dattypes[5] = "DOUBLE";
    Tablenames[5] = "ProductMomentCorilation";
```

Creates a DataTable and names it

Declares a SQLCon object constructed with the name of the table it wants to draw data from. I will go indepth into this class in its own section.assigns its rdr attribute to the SQLiteDataReader of the local variable rdr

Standard Declaration and creation of a datatable and its columns, refer back to this as a example of how datatables are created throughout the program

```
    dattypes[5] = "DOUBLE";
    Tablenames[5] = "ProductMomentCorilation";

    DataColumn[] PrimaryKeyColumns = new DataColumn[1];
    PrimaryKeyColumns[0] = LinRegtable.Columns["CustomerID"];
    LinRegtable.PrimaryKey = PrimaryKeyColumns;
    Tuple<string[], string[]> arrays = Tuple.Create(Tablenames, dattypes);


    while (rdr.Read())
    {
        DataRow row;
        row = LinRegtable.NewRow();
        row["CustomerID"] = rdr.GetString(0);
        row["SlopeXsq"] = rdr.GetDouble(1);
        row["SlopeX"] = rdr.GetDouble(2);
        row["Yintercept"] = rdr.GetDouble(3);
        row["RSquared"] = rdr.GetDouble(4);
        row["ProductMomentCorilation"] = rdr.GetDouble(5);
        LinRegtable.Rows.Add(row);
    }
    Optionlist.Add(LinRegtable.TableName, LinRegtable);

    SQLtables.Tables.Add(LinRegtable);

}
```

Continued from above

Assigns the Datatables Primary Key

Uses a while loop to read every row within the SQLiteDataReader

Standard reading data from SQL database for all applications in the system

Assign to the dictionary the string of the tables name and the value of the table

Add the table to the dataset

CreateSpent()

Creates and fills a Datatable to hold the data of the Spent SQL table. This table is created in the data processing program and holds the money spent overtime of each CustomerID. Here is the tables' structure and example of the data that this method will be requesting row data from:

CustomerData Table name: Spent

	Name	Data type	Primary Key
1	PrimaryKey	INTAGAR	
2	CustomerID	VARCHAR	
3	MoneySpentOvertime	DOUBLE	

PrimaryKey	CustomerID	MoneySpentOvertime
1	17850	5391.210000000009
2	13047	366.6300000000001
3	12583	855.86
4	13748	204
5	15100	700.8
6	15291	328.7999999999995
7	14688	444.98
8	17809	1251.5
9	15311	2057.4100000000008
10	16098	430.6
11	18074	489.6

```
1 reference
private void CreateSpent()
{
    DataTable Spent = new DataTable("Spent");

    SQLCon connection = new SQLCon("Spent");

    SQLiteDataReader rdr = connection.rdr;
    {//creating columns
     //
        DataColumn column; //MoneySpent,InvoiceDate,CustomerID

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.String");
        column.ColumnName = "CustomerID";
        column.ReadOnly = true;
        Spent.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "MoneySpentOvertime";
        column.ReadOnly = true;
        Spent.Columns.Add(column);

        DataColumn[] PrimaryKeyColumns = new DataColumn[1];
        PrimaryKeyColumns[0] = Spent.Columns["PrimaryKey"];
        Spent.PrimaryKey = PrimaryKeyColumns;


    }
    DataRow row;

    while (rdr.Read())
    {
        row = Spent.NewRow();
        row["CustomerID"] = rdr.GetString(1);
        row["MoneySpentOvertime"] = rdr.GetDecimal(2);

        Spent.Rows.Add(row);
    }
    Optionlist.Add(Spent.TableName, Spent);


    SQLtables.Tables.Add(Spent);
}
```

CreateMPPertrans()

Creates a debatable to hold the tabular data contained in the SQL table of Transactions. The data was created in the Dataprocessing program. Here is the SQL table structure and example of the data that the data will be requesting from:

InvoiceNo	MoneySpent	InvoiceDate	CustomerID	NumericDate	Year
536365	139	29-11-16	17850	-31	16
536366	22	29-11-16	17850	-31	16
536367	279	29-11-16	13047	-31	16
536368	70	29-11-16	13047	-31	16
536369	18	29-11-16	13047	-31	16
536370	856	29-11-16	12583	-31	16
536371	204	29-11-16	13748	-31	16
536372	22	29-11-16	17850	-31	16
536373	260	29-11-16	17850	-31	16
536374	350	29-11-16	15100	-31	16
536375	260	29-11-16	17850	-31	16
536376	329	29-11-16	15291	-31	16
536377	22	29-11-16	17850	-31	16
536378	445	29-11-16	14688	-31	16
536380	35	29-11-16	17809	-31	16
536381	450	29-11-16	15311	-31	16

CustomerData Table name: MPPerTrans

	Name	Data type	Primary Key	Foreign Key
1	InvoiceNo	VARCHAR		
2	MoneySpent	DOUBLE		
3	InvoiceDate	VARCHAR		
4	CustomerID	VARCHAR		
5	NumericDate	INTAGAR		
6	Year	INTAGAR		

```
1 reference
private void CreateMPPertrans()
{
    DataTable MPPerTrans = new DataTable("MPPerTrans");

    SQLCon connection = new SQLCon("MPPerTrans");

    SQLiteDataReader rdr = connection.rdr;
    {//creating columns
     //
        DataColumn column; //MoneySpent,InvoiceDate,CustomerID

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.String");
        column.ColumnName = "InvoiceNo";
        column.ReadOnly = true;
        MPPerTrans.Columns.Add(column);

        string[] Tablenames = new string[5];

        string[] dattypes = new string[5];

        dattypes[0] = "VARCHAR";
        Tablenames[0] = "InvoiceNo";

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "MoneySpent";
        column.ReadOnly = true;
        MPPerTrans.Columns.Add(column);

        dattypes[1] = "DOUBLE";
        Tablenames[1] = "MoneySpent";

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.String");
        column.ColumnName = "InvoiceDate";
        column.ReadOnly = true;
        MPPerTrans.Columns.Add(column);

        dattypes[2] = "VARCHAR";
        Tablenames[2] = "InvoiceDate";

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "CustomerID";
        column.ReadOnly = true;
        MPPerTrans.Columns.Add(column);

        dattypes[3] = "VARCHAR";
        Tablenames[3] = "CustomerID";
        DataColumn[] PrimaryKeyColumns = new DataColumn[1];
        PrimaryKeyColumns[0] = MPPerTrans.Columns["InvoiceNo"];
        MPPerTrans.PrimaryKey = PrimaryKeyColumns;

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "NumericDate";
        column.ReadOnly = true;
        MPPerTrans.Columns.Add(column);

        dattypes[4] = "INTAGAR";
        Tablenames[4] = "NumericDate";
```

```
            dattypes[4] = "INTAGAR";
            Tablenames[4] = "NumericDate";


            column = new DataColumn();
            column.DataType = System.Type.GetType("System.Int32");
            column.ColumnName = "Year";
            column.ReadOnly = true;
            MPPerTrans.Columns.Add(column);


        }
        DataRow row;

        Console.WriteLine(rdr.FieldCount);

        while (rdr.Read())
        {

            row = MPPerTrans.NewRow();

            row["InvoiceNo"] = rdr.GetString(0);
            row["MoneySpent"] = rdr.GetDecimal(1);
            row["InvoiceDate"] = rdr.GetString(2);
            row["CustomerID"] = rdr.GetString(3);
            row["NumericDate"] = rdr.GetInt32(4);
            row["Year"] = rdr.GetInt32(5);

            MPPerTrans.Rows.Add(row);
        }

        Optionlist.Add(MPPerTrans.TableName, MPPerTrans);

        SQLtables.Tables.Add(MPPerTrans);

    }
```

CQI()

Creates a datatable to hold the Tabular data of the SQL table CQI. This data is created by the data processing program. Here is the SQL table structure and example of the data.

CustomerData Table name: CQI

	Name	Data type
1	PrimaryKey	VARCHAR
2	CustomerID	VARCHAR
3	MoneySpentOvertime	DECIMAL
4	StandardDiv	DECIMAL
5	Frequency	INTAGAR
6	CQI	DECIMAL

PrimaryKey	CustomerID	MoneySpentOvertime	StandardDiv	Frequency	CQI
0	17850	5391.210000000009	7.4341398955198	34	24656.670788569245
1	13047	366.6300000000001	8.48528137423857	3	129.623279593212
2	12583	855.86	0	1	0
3	13748	204	0	1	0
4	15100	700.8	175	2	8.00914285714286
5	15291	328.79999999999995	0	1	0
6	14688	444.98	0	1	0
7	17809	1251.5	373.3523804664971	3	10.05618337107914
8	15311	2057.4100000000008	56.64705542999467	6	217.91882925416104
9	16098	430.6	0	1	0
10	18074	489.6	0	1	0
11	17420	130.85	0	1	0
12	16029	4271.52	268.229172330097	3	47.7746692825407

```
private void CQI()
{
    //fill columns
    DataColumn column;
    DataTable Spent = Optionlist["Spent"];
    DataTable MPPerTrans = Optionlist["MPPerTrans"];

    SQLCon connection = new SQLCon("CQI");

    SQLiteDataReader rdr = connection.rdr;

    DataTable CQIData = new DataTable("CQI");

    string[] dattypes = new string[6];
    string[] Tablenames = new string[6];

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Int32");
    column.ColumnName = "CustomerID";
    column.ReadOnly = true;
    CQIData.Columns.Add(column);

    dattypes[1] = "VARCHAR";
    Tablenames[1] = "CustomerID";



    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "MoneySpentOvertime";
    column.ReadOnly = true;
    CQIData.Columns.Add(column);

    dattypes[2] = "DECIMAL";
    Tablenames[2] = "MoneySpentOvertime";



    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "StandardDiv";
    column.ReadOnly = true;
    CQIData.Columns.Add(column);

    dattypes[3] = "DECIMAL";
    Tablenames[3] = "StandardDiv";



    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Int32");
    column.ColumnName = "Frequency";
    column.ReadOnly = true;
    CQIData.Columns.Add(column);

    dattypes[4] = "INTAGAR";
    Tablenames[4] = "Frequency";



    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "CQI";
    column.ReadOnly = true;
    CQIData.Columns.Add(column);
```

```
        dattypes[5] = "DECIMAL";
        Tablenames[5] = "CQI";

        DataColumn[] PrimaryKeyColumns = new DataColumn[1];
        PrimaryKeyColumns[0] = CQIData.Columns["PrimaryKey"];
        CQIData.PrimaryKey = PrimaryKeyColumns;

        DataRow row;

        while (rdr.Read())
        {
            row = CQIData.NewRow();

            row["CustomerID"] = rdr.GetString(1);

            row["MoneySpentOvertime"] = rdr.GetDouble(2);

            row["StandardDiv"] = rdr.GetDouble(3);

            row["Frequency"] = rdr.GetInt16(4);

            row["CQI"] = rdr.GetDouble(5);

            CQIData.Rows.Add(row);
        }
        Optionlist.Add(CQIData.TableName, CQIData);

        SQLtables.Tables.Add(CQIData);
    }
    1 reference
```

FindMMPD()

Creates a datatable to hole the Money Made Per Day SQL table data. Data Created in the data processing program. Had to move to a negative numeric day to have a consistent way of representing data that went through between from 2016 to 2017 for use in creating regression equations, did this with the use of a constant -365 so that can be balanced out if only focusing on 2016 data. Primary key works as a way of telling how long since the start of tracking the data the data for that day is from mainly for requesting from SQL to debug specific time frames, while the column Day for any program purposes is actually used as its primary key. SQL table structure and example of the data that will be requested:

CustomerData Table name: MoneyMadePerDay

	Name	Data type	Primary Key	Foreign Key	U
1	PrimaryKey	VARCHAR			
2	Day	INTAGAR			
3	MoneySpent	DECIMAL			
4	Frequencyofsale	INTAGAR			
5	Year	INTAGAR			

PrimaryKey	Day	MoneySpent	Frequencyofsale	Year
0	-31	46355	121	16
1	-30	47311	137	16
2	-29	23919	57	16
3	-27	31730	87	16
4	-26	31163	94	16
5	-25	53797	71	16
6	-24	39211	111	16
7	-23	38216	97	16
8	-22	33653	72	16
9	-20	17305	43	16
10	-19	27645	64	16
11	-18	26267	69	16

```
private void FindMMPD()
{
    {

        SQLCon connection = new SQLCon("MoneyMadePerDay");

        SQLiteDataReader rdr = connection.rdr;
        DataTable MMPerday = new DataTable("MoneyMadePerDay");

        DataColumn column;


        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "Day";
        column.ReadOnly = true;
        MMPerday.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "MoneySpent";
        column.ReadOnly = false;
        MMPerday.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "Frequencyofsale";
        column.ReadOnly = false;
        MMPerday.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "Year";
        column.ReadOnly = false;
        MMPerday.Columns.Add(column);

        DataRow row;

        while (rdr.Read())
        {


            row = MMPerday.NewRow();
            Console.WriteLine(rdr.GetInt32(1));
            row["Day"] = rdr.GetInt32(1);
            row["MoneySpent"] = rdr.GetInt32(2);
            row["Frequencyofsale"] = rdr.GetInt32(3);
            row["Year"] = rdr.GetInt32(4);

            MMPerday.Rows.Add(row);
        }


        Optionlist.Add(MMPerday.TableName, MMPerday);
        SQLtables.Tables.Add(MMPerday);

    }
}
```

Mainset()

Creates a Datatable to load in and hold data from the CustomerData SQL table. Example of table structure and Data:

CustomerData Table name: CustomerData

	Name	Data type	Primary Key	Foreign Key
1	InvoiceNo	INTEGER		
2	StockCode	TEXT		
3	Description	TEXT		
4	Quantity	INTEGER		
5	InvoiceDate	TEXT		
6	UnitPrice	REAL		
7	CustomerID	INTEGER		
8	Country	TEXT		
9	PrimaryKey	INTEGER		

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerI	Country	PrimaryKey
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	29-Nov-16	2.55	17850	United Kingdom	1
536365	71053	WHITE METAL LANTERN	6	29-Nov-16	3.39	17850	United Kingdom	2
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	29-Nov-16	2.75	17850	United Kingdom	3
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	29-Nov-16	3.39	17850	United Kingdom	4
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	29-Nov-16	3.39	17850	United Kingdom	5
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	29-Nov-16	7.65	17850	United Kingdom	6
536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	29-Nov-16	4.25	17850	United Kingdom	7
536366	22633	HAND WARMER UNION JACK	6	29-Nov-16	1.85	17850	United Kingdom	8
536366	22632	HAND WARMER RED POLKA DOT	6	29-Nov-16	1.85	17850	United Kingdom	9
536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	29-Nov-16	1.69	13047	United Kingdom	10
536367	22745	POPPY'S PLAYHOUSE BEDROOM	6	29-Nov-16	2.1	13047	United Kingdom	11
536367	22748	POPPY'S PLAYHOUSE KITCHEN	6	29-Nov-16	2.1	13047	United Kingdom	12
536367	22749	FELTCRAFT PRINCESS CHARLOTTE DOLL	8	29-Nov-16	3.75	13047	United Kingdom	13
536367	22310	IVORY KNITTED MUG COSY	6	29-Nov-16	1.65	13047	United Kingdom	14
536367	84969	BOX OF 6 ASSORTED COLOUR TEASPOONS	6	29-Nov-16	4.25	13047	United Kingdom	15
536367	22623	BOX OF VINTAGE JIGSAW BLOCKS	3	29-Nov-16	4.95	13047	United Kingdom	16
536367	22622	BOX OF VINTAGE ALPHABET BLOCKS	2	29-Nov-16	9.95	13047	United Kingdom	17
536367	21754	HOME BUILDING BLOCK WORD	3	29-Nov-16	5.95	13047	United Kingdom	18
536367	21755	LOVE BUILDING BLOCK WORD	3	29-Nov-16	5.95	13047	United Kingdom	19
536367	21777	RECIPE BOX WITH METAL HEART	4	29-Nov-16	7.95	13047	United Kingdom	20
536367	48187	DOORMAT NEW ENGLAND	4	29-Nov-16	7.95	13047	United Kingdom	21
536368	22960	JAM MAKING SET WITH JARS	6	29-Nov-16	4.25	13047	United Kingdom	22
536368	22913	RED COAT RACK PARIS FASHION	3	29-Nov-16	4.95	13047	United Kingdom	23
536368	22912	YELLOW COAT RACK PARIS FASHION	3	29-Nov-16	4.95	13047	United Kingdom	24
536368	22914	BLUE COAT RACK PARIS FASHION	3	29-Nov-16	4.95	13047	United Kingdom	25
536369	21756	BATH BUILDING BLOCK WORD	3	29-Nov-16	5.95	13047	United Kingdom	26
536370	22728	ALARM CLOCK BAKELIKE PINK	24	29-Nov-16	3.75	12583	France	27
536370	22727	ALARM CLOCK BAKELIKE RED	24	29-Nov-16	3.75	12583	France	28
536370	22726	ALARM CLOCK BAKELIKE GREEN	12	29-Nov-16	3.75	12583	France	29

```
1 reference
private void Mainset()
{
    System.Data.DataTable table = new DataTable("Main");
    // Create a new DataTable.

    // Declare variables for DataColumn and DataRow objects.
    DataColumn column;


    // Create new DataColumn, set DataType,
    // ColumnName and add to DataTable.
    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Int32");
    column.ColumnName = "PrimaryKey";
    column.ReadOnly = true;
    table.Columns.Add(column);

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.String");
    column.ColumnName = "StockCode";
    column.ReadOnly = true;
    // Add the Column to the DataColumnCollection.
    table.Columns.Add(column);

    // Create second column.
    column = new DataColumn();
    column.DataType = System.Type.GetType("System.String");
    column.ColumnName = "Description";
    column.ReadOnly = true;
    // Add the column to the table.
    table.Columns.Add(column);

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Int32");
    column.ColumnName = "Quantity";
    column.ReadOnly = true;
    table.Columns.Add(column);

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.String");
    column.ColumnName = "InvoiceDate";
    column.ReadOnly = true;
    table.Columns.Add(column);

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Double");
    column.ColumnName = "UnitPrice";
    column.ReadOnly = true;
    table.Columns.Add(column);

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Int32");
    column.ColumnName = "CustomerID";
    column.ReadOnly = true;
    table.Columns.Add(column);

    column = new DataColumn();
    column.DataType = System.Type.GetType("System.String");
    column.ColumnName = "Country";
    column.ReadOnly = true;
    table.Columns.Add(column);
```

```
        column.ColumnName = "Country";
        column.ReadOnly = true;
        table.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.String");
        column.ColumnName = "InvoiceNo";
        column.ReadOnly = true;
        table.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "MoneySpent";
        column.ReadOnly = true;
        table.Columns.Add(column);

        DataColumn[] PrimaryKeyColumns = new DataColumn[1];
        PrimaryKeyColumns[0] = table.Columns["id"];
        table.PrimaryKey = PrimaryKeyColumns;

        // Make the ID column the primary key column.

        int y = 1;
        DataRow row;
        SQLCon con = new SQLCon("CustomerData");
        SQLiteDataReader rdr = con.rdr;

        while (rdr.Read())
        {
            // StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country,I
            row = table.NewRow();
            row["PrimaryKey"] = y;
            row["StockCode"] = rdr.GetString(1);
            row["Description"] = rdr.GetString(2);
            row["Quantity"] = rdr.GetInt32(3);
            row["InvoiceDate"] = rdr.GetString(4);
            row["UnitPrice"] = rdr.GetDouble(5);
            row["CustomerID"] = rdr.GetInt32(6);
            row["Country"] = rdr.GetString(7);
            row["InvoiceNo"] = rdr.GetInt32(0);
            row["MoneySpent"] = Math.Round((rdr.GetInt32(3) * rdr.GetDouble(5)), 2);
            table.Rows.Add(row);

            y++;
        }

        Optionlist.Add(table.TableName, table);

        SQLtables.Tables.Add(table);
        //return df;
    }
```

TopIDs

Line 61

Loads in the SQL tables holding the top IDs in different metrics.

The system Utilizes the SQL database as a central file storage between all its separate programs as this system has many advantages over a simpler file storage: makes requesting data simple and generalized allowing the system to utilize generalized SQL requests instead of easily error prone folder traversal; easier cleaning of the data as simple cleaning operations can take place as other programs continue running utilizing different data; it allows for the free form adding and changing of the central main dataset so that data can be removed and added upon without it effecting the operation of the other applications; it also

allows for separate future programs to easily query and request from this central database making the system easily scalable.

```
7 references
class TopIDs
{
    public List<int> CQI = new List<int>();
    public List<int> Frequency = new List<int>();
    public List<int> MMperDayMoney = new List<int>();
    public List<int> MMperDayFrequency = new List<int>();
    public List<int> PerTrans = new List<int>();
    public List<int> Spent = new List<int>();
    6 references
    public void GetID(string name, List<int> list)
    {
        SQLCon connection = new SQLCon(name);

        SQLiteDataReader rdr = connection.rdr;
        while (rdr.Read())
        {
            int data = (int)rdr.GetInt32(0);

            Console.WriteLine(data);

            list.Add(data);
        }
    }
    3 references
    public TopIDs()
    {
        GetID("CQITop25CQI", CQI);
        GetID("CQITop25Frequency", Frequency);
        GetID("MoneyMadePerDayTop25Frequencyofsale", MMperDayFrequency);
        GetID("MoneyMadePerDayTop25MoneySpent", MMperDayMoney);
        GetID("MPPerTransTop25MoneySpent", PerTrans);
        GetID("SpentTop25MoneySpentOvertime", Spent);
    }
}
```

Has List attributes of each top25 table that exists in the database

Takes in a string parameter and a List, searches the database for that name and fills the list parameter with all row data found

Filling each list, passing the name of the table into the method

SQLCon

Line 23

Generalizes SQL class that remains identical acorsss almost all programs within the system due to its generalized design:

```
16 references
class SQLCon
{
    private string filepath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + "\\CustomerData.db";
    public SQLiteDataReader rdr;
    private SQLiteConnection conn;
    7 references
    public SQLCon(string Name)
    {
        string command1 = "SELECT * FROM " + Name;
        SQLiteConnection connection = CreateSQLconnection();
        var cmd = new SQLiteCommand(command1, connection);
        this.rdr = cmd.ExecuteReader();
    }

    0 references
    public void ExicuteQuery(string CommandText)
    {
        SQLiteCommand sqlite_cmd;
        sqlite_cmd = conn.CreateCommand();
        sqlite_cmd.CommandText = CommandText;
        Console.WriteLine(CommandText);
        sqlite_cmd.ExecuteNonQuery();
    }

    0 references
    public SQLCon()
    {
        this.conn = new SQLiteConnection(@"Data Source =" + filepath);
        conn.Open();
    }

    1 reference
    public SQLiteConnection CreateSQLconnection()
    {
        SQLiteConnection ll = new SQLiteConnection(@"Data Source =" + filepath);
        ll.Open();
        return ll;
    }
}
```

filepath uses the environmental class to create the filepath variable to allow the system to run on different system's file setups as long as MyDocument folder is supported

the rdr is a attribute SQLiteDataReader which is filled during the overloaded construction of the object only if constructed with a string parameter

Overloaded constructor that creates a SQL query using the parameter and executes it using the SQL connection made by the method CreateConnection(), filling the rdr attribute. This allows any table to be accessed read using the same class and the same construction making it fit for general use.

General method that takes in a string command constructs a command and then executes it while not returning anything. Only useable for sending data to the SQL database but is used in any saving throughout the system.

Overloaded constructor that just opens a connection allowing for the connection to be used for any purpose the program needs. Exists so that any SQL interaction is done through this object so that the filepath can be centrally controlled.

The non-specificity of any attribute or method in the class allows it to be almost identical across all programs

The form class itself

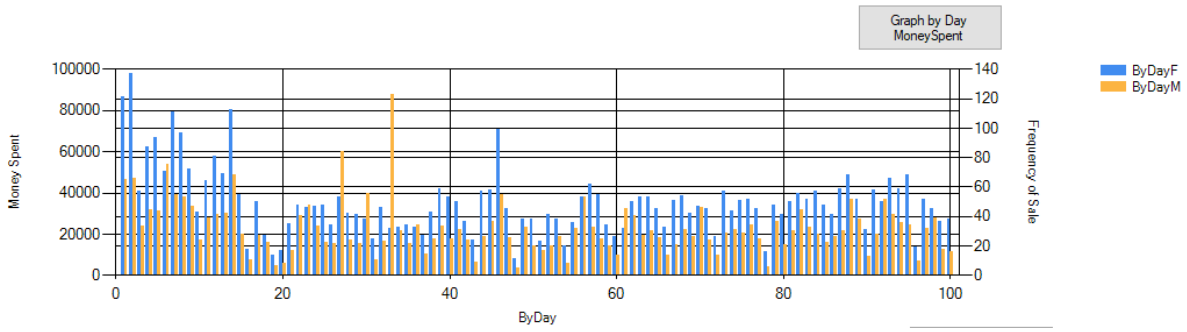
```
3 references
public partial class Form1 : Form
{
    static DataFromSQLTables tables = new DataFromSQLTables();
    static TopIDs topIDs = new TopIDs();
```

The form itself contains objects instances of both the DataFromSQLTables class and the TopIDs class.

If Byday button is pressed:

Line959

The design document makes reference to a Line graph to go along side this graph with the quarters regression equations. However, after building such a system after inspecting and testing the equations that were produced they weren't within comparatively lax tolorences. This is because the ByDay data itself is too random and unpredictable to be graphed so this aspect was scrapped as to not make more complex a already busy graph with more axis that most of the time wouldn't show anything useful.




```
1 reference
private void button1_Click_3(object sender, EventArgs e)
{
    chart1.Series["ByDayM"].Points.Clear();
    chart1.Series["ByDayF"].Points.Clear();
    DataTable MMPD = tables.Optionlist["MoneyMadePerDay"];
    int y = 1;
    foreach (DataRow item in MMPD.Rows)
    {
        this.chart1.Series["ByDayM"].Points.AddXY(y, (double)item.ItemArray[1]);
        this.chart1.Series["ByDayF"].Points.AddXY(y, (int)item.ItemArray[2]);
        y++;
    }

}
```

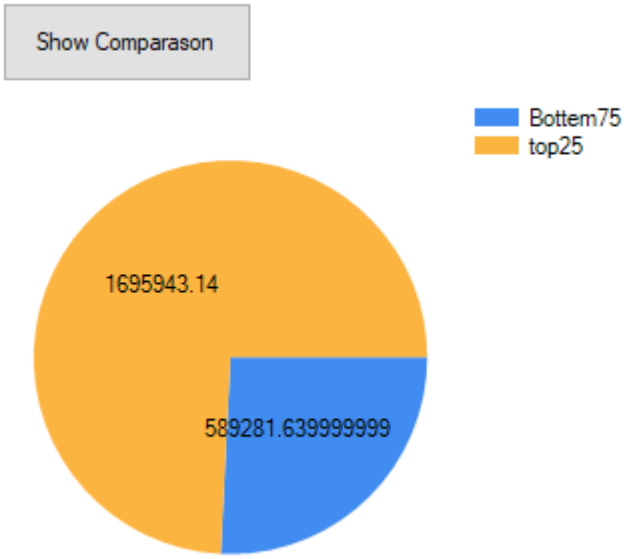
Clears the graph of past data in case the button is pressed multiple times

Loads in the Money Made Per Day Datatable from the tables object's dictionary attribute

Goes through each row in the datatable and add its data to the graph

If show comparison is pressed

Line 986




```
1 reference
private void button5_Click(object sender, EventArgs e)
{
    chart2.Series["80/20"].Points.Clear();
    TopIDs topIDs = new TopIDs();
    Tuple<double, double> Sevinty25 = Tally2080(tables.Optionlist["Spent"], topIDs.Spent);

    chart2.Series["80/20"].Points.AddXY("Bottem75", Sevinty25.Item2);
    chart2.Series["80/20"].Points.AddXY("top25", Sevinty25.Item1);

}
```

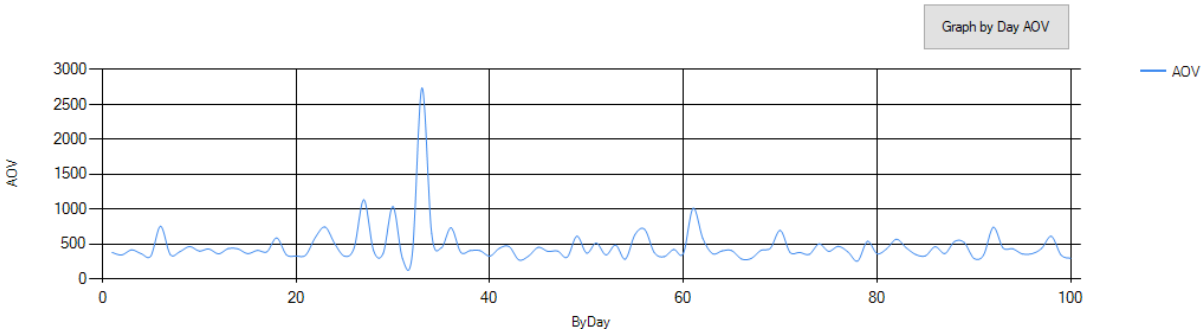
Loads in as parameters Using the List of top25 of money spent over and the spent overtime datatable

```
1 reference
private Tuple<double, double> Tally2080(DataTable Spent, List<int> Top25)
{
    double Twenty = 0;
    double Sevintyfive = 0;

    foreach (DataRow item in Spent.Rows)
    {
        if (Top25.Contains(Convert.ToInt32(item.ItemArray[0])))
        {
            Twenty = Twenty + (double)item.ItemArray[1];
        }
        else
        {
            Sevintyfive = Sevintyfive + (double)item.ItemArray[1];
        }
    }
    Tuple<double, double> T = new Tuple<double, double>(Twenty, Sevintyfive);
    return T;
}
```

Tallys up the top 25s spending and the bottom 75s spending

If graph by AOV is pressed:

Line 882

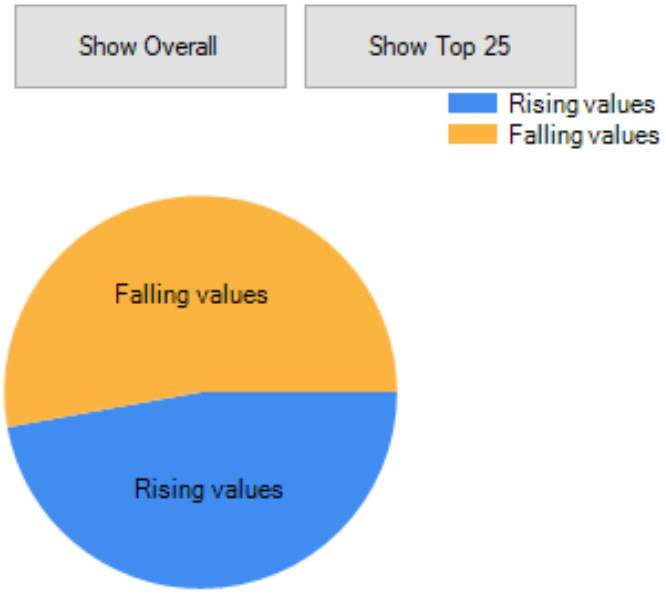



```
1 reference
private void button4_Click_1(object sender, EventArgs e)
{
    chart4.Series["AOV"].Points.Clear();
    DataTable MMPD = tables.Optionlist["MoneyMadePerDay"];
    int y = 1;
    foreach (DataRow item in MMPD.Rows)
    {

        //  int day = (int)item.ItemArray[0] + 31;
        decimal AOV = (decimal)(double)item.ItemArray[1] / (int)item.ItemArray[2];
        this.chart4.Series["AOV"].Points.AddXY(y, AOV);
        y++;
    }


}
```

If Show overall is pressed

Line1301

This shows the proportion of customers with rising and falling transaction values. This quantity is found by finding if the gradient at the last X value on the graph is positive or negative. Only applied to stored regression equations where the system has deemed a equation possible.




```
1 reference
private void button3_Click_2(object sender, EventArgs e)
{
    chart6.Series["RS"].Points.Clear();
    DataTable CustomerLinreg = tables.Optionlist["CustomerIDRegresion"];
    DataTable Trans = tables.Optionlist["MPPerTrans"];      Loads in the two datatables required

    int pos = 0;
    int neg = 0;  Runs through every row in CustomerLinReg
    foreach (DataRow item in CustomerLinreg.Rows)
    {
        Checks if there is a applicable graph that the system has deemed useful and representative to the data within reasonable tolerances
        if ((double)item.ItemArray[1] == 0 && (double)item.ItemArray[2] == 0 && (double)item.ItemArray[3] == 0)
        {    If the data is all 0 the equation created wasn't within
             reasonable tolerances
        }
        else
        {    Finds all instances of the CustomerID in the Transactions Datatable
             DataRow[] rows = Trans.Select("CustomerID=  '" + (string)item.ItemArray[0] + "'");

             decimal[] Equation = new decimal[2] { (decimal)Convert.ToDecimal((double)item.ItemArray[1])
                                                 , (decimal)Convert.ToDecimal((double)item.ItemArray[2]) };

             decimal Gradient = GetGradient(Equation, rows.Length - 1);
             if (Gradient > 0m)  Utilizes the GetGradient() method with the equation and the last X
             {                   value within the model was made for to get its last gradient
                 pos++;
             }
             else if(Gradient < 0)
             {           Tally's up positive and negative
                 neg++;  gradients and then adds them
             }           to the chart
        }
    }

    chart6.Series["RS"].Points.AddXY("Rising values", pos);
    chart6.Series["RS"].Points.AddXY("Falling values", neg);

}
```

Gets the parts of the equation within the datatable that the method is concerned with

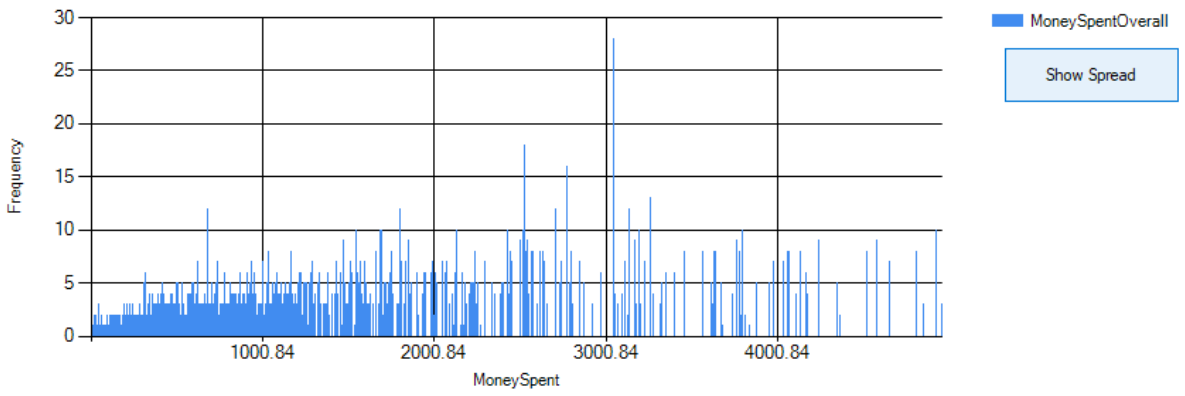
```
3 references
private decimal GetGradient(decimal[] equation, decimal Xvalue)
{
    decimal Xsq = equation[0];
    decimal x = equation[1];

    decimal gradient = (2m * Xsq) * (Xvalue) + x;

    return gradient;
}
```

Simple find gradient formula that can be applied to both polynomial and simple because of how the equation row is set up, if there is no X^2 coefficient then it just times 0 by 2

If Show Spread is pressed

The method SpentScatter() is called

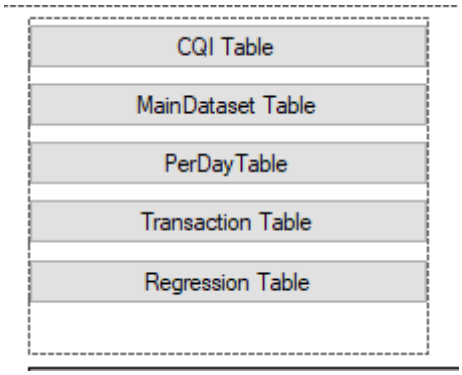
Line 917




```
1 reference
private void SpentScatter()
{

    chart3.Series["MoneySpentOverall"].Points.Clear();
    DataTable spent = tables.Optionlist["Spent"];
    DataTable Trans = tables.Optionlist["MPPerTrans"];
    foreach (DataRow item in spent.Rows)
    {
        if ((double)item.ItemArray[1]>5000)
        {   Does not display values above 5,000 as these are so
            extreme that displaying them would dilute the
        }   understanding of the graph
        else
        {   Finds the frequency by finding how many times the Customer ID is present in the
            Transaction Datatable
            DataRow[] rows = Trans.Select("CustomerID=  '" + item.ItemArray[0] + "'");
            double money = (double)item.ItemArray[1];

            chart3.Series["MoneySpentOverall"].Points.AddXY(money, rows.Length);
        }
    }
}
```

If any display table button is presses

Line1002



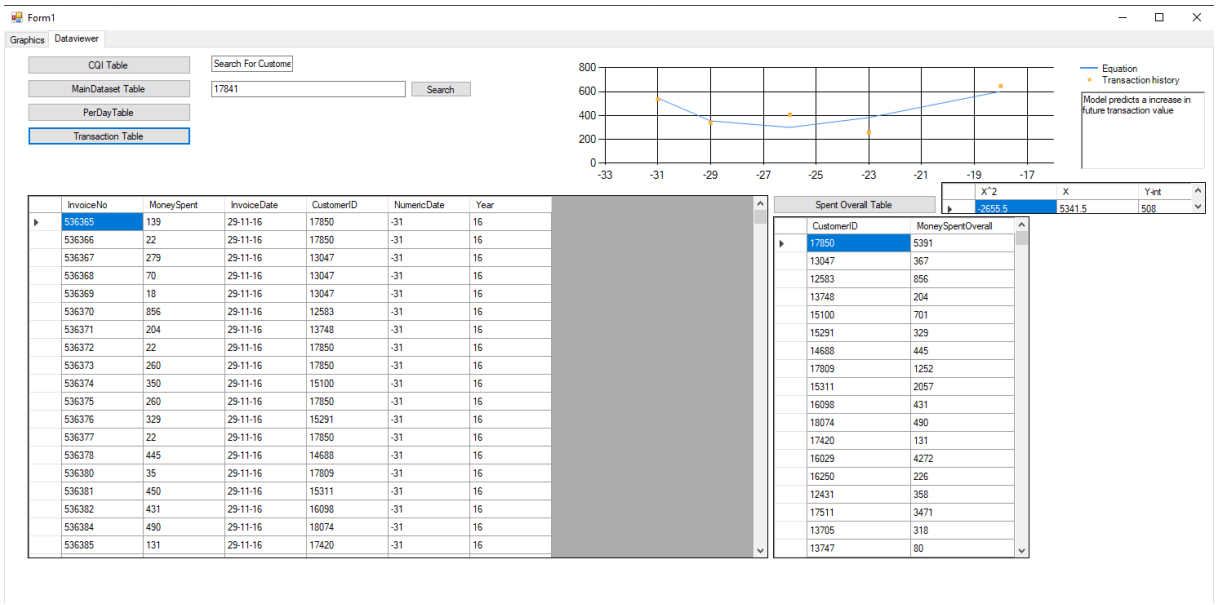
There is a uniform function that simply clears the dataviewgrid and then makes its source the associated Datatable.

```
1 reference
private void CQI_Click(object sender, EventArgs e)
{
    dataGridView2.Columns.Clear();
    dataGridView2.DataSource = tables.Optionlist["CQI"];

}

1 reference
private void button9_Click(object sender, EventArgs e)
{
    dataGridView2.Columns.Clear();
    dataGridView2.DataSource = tables.Optionlist["Main"];
}

1 reference
private void button10_Click(object sender, EventArgs e)
{
    dataGridView2.Columns.Clear();
    dataGridView2.DataSource = tables.Optionlist["MoneyMadePerDay"];
}

1 reference
private void button7_Click(object sender, EventArgs e)
{
    dataGridView2.Columns.Clear();
    dataGridView2.DataSource = tables.Optionlist["MPPerTrans"];
}
```

Clicking the search button

Line 1130

Serches and displays any instances of that customer within the dataview grid and displays its transaction history on the graph and, if applicable, the customers Regression equation.



```
1 reference
private void button13_Click(object sender, EventArgs e)
{

    DataTable Spent = tables.Optionlist["Spent"];

    DataRow[] Tesrows = Spent.Select("CustomerID=  '" + (string)textBox2.Text + "'");

    if (Tesrows.Length > 0)
    {

        showEquation((string)textBox2.Text);

        if (dataGridView2.DataSource != null)
        {


            List<DataGridViewRow> rows = new List<DataGridViewRow>();

            if (dataGridView2.Columns[0].Name == "CustomerID" && dataGridView2.Columns[1].Name!= "SlopeXsq")
            {

                int y = 0;
                for (int i = 0; i < dataGridView2.Rows.Count - 1; i++)
                {

                    Console.WriteLine(y);
                    if (dataGridView2.Rows[i].Cells[0].Value.ToString().Equals((string)textBox2.Text))
                    {
                        DataGridViewRow row = (DataGridViewRow)dataGridView2.Rows[i].Clone();

                        for (int j = 0; j < dataGridView2.Rows[i].Cells.Count; j++)
                        {
                            row.Cells[j].Value = dataGridView2.Rows[i].Cells[j].Value;
                        }
                        rows.Add(dataGridView2.Rows[i]);
                        dataGridView2.FirstDisplayedScrollingRowIndex = i;
                    }
                    y++;
                }

                dataGridView2.DataSource = null;
                dataGridView2.Rows.Clear();

                dataGridView2.ColumnCount = 5;

                dataGridView2.Columns[0].Name = "CustomerID";
                dataGridView2.Columns[1].Name = "MoneySpentOvertime";
                dataGridView2.Columns[2].Name = "StandardDiv";
                dataGridView2.Columns[3].Name = "Frequency";
                dataGridView2.Columns[4].Name = "CQI";

                foreach (DataGridViewRow item in rows)
                {
                    dataGridView2.Rows.Add(item);
                }

            }
```

Checks to see if the item in the search window is a CustomerID by looking for it within the Spent overall Datatable which holds all Customer IDs.

Utilizes the showEquaton() Method passing the text in the search box

Checks if a table is selected

Issue with dataviewgrid is that it is difficult to tell if using Datatables as sources which Datatable is acting as its source, to get past this the method tests known properties of each of the Datatables that could be selected. For example it knows that it is CQI if its first column is CustomerID with its second column not being SlopeXsq

Searches the grid and adds to a list of rows the rows that include the CustomerID

Clears the source and the rows

Re creates the table manually and adds the rows fond to its rows

```
else if (dataGridView2.Columns[0].Name == "InvoiceNo")
{

    int y = 0;
    for (int i = 0; i < dataGridView2.Rows.Count - 1; i++)
    {

        Console.WriteLine(y);
        if (dataGridView2.Rows[i].Cells[3].Value.ToString().Equals((string)textBox2.Text))
        {
            DataGridViewRow row = (DataGridViewRow)dataGridView2.Rows[i].Clone();

            for (int j = 0; j < dataGridView2.Rows[i].Cells.Count; j++)
            {
                row.Cells[j].Value = dataGridView2.Rows[i].Cells[j].Value;
            }

            rows.Add(row);

        }
        y++;
    }

    dataGridView2.DataSource = null;

    dataGridView2.ColumnCount = 6;

    dataGridView2.Columns[0].Name = "InvoiceNo";
    dataGridView2.Columns[1].Name = "MoneySpent";
    dataGridView2.Columns[2].Name = "InvoiceDate";
    dataGridView2.Columns[3].Name = "CustomerID";
    dataGridView2.Columns[4].Name = "NumericDate";
    dataGridView2.Columns[5].Name = "Year";

    foreach (DataGridViewRow item in rows)
    {
        dataGridView2.Rows.Add(item);
    }

}
```

```
            else if (dataGridView2.Columns[0].Name == "PrimaryKey")
            {
                int y = 0;
                for (int i = 0; i < dataGridView2.Rows.Count - 1; i++)
                {

                    Console.WriteLine(y);
                    if (dataGridView2.Rows[i].Cells[6].Value.ToString().Equals((string)textBox2.Text))
                    {

                        DataGridViewRow row = (DataGridViewRow)dataGridView2.Rows[i].Clone();

                        for (int j = 0; j < dataGridView2.Rows[i].Cells.Count; j++)
                        {
                            row.Cells[j].Value = dataGridView2.Rows[i].Cells[j].Value;
                        }

                        rows.Add(row);
                        dataGridView2.FirstDisplayedScrollingRowIndex = i;
                    }
                    y++;
                }

                dataGridView2.DataSource = null;

                dataGridView2.ColumnCount = 10;

                dataGridView2.Columns[0].Name = "Primarykey";
                dataGridView2.Columns[1].Name = "InvoiceNo";
                dataGridView2.Columns[2].Name = "StockCode";
                dataGridView2.Columns[3].Name = "Description";
                dataGridView2.Columns[4].Name = "Quantity";
                dataGridView2.Columns[5].Name = "InvoiceDate";
                dataGridView2.Columns[6].Name = "UnitPrice";
                dataGridView2.Columns[7].Name = "CustomerID";
                dataGridView2.Columns[8].Name = "Country";
                dataGridView2.Columns[9].Name = "MoneySpent";

                foreach (DataGridViewRow item in rows)
                {
                    dataGridView2.Rows.Add(item);
                }


            }
        }
    }

}
```

```
private void showEquation(string Customerid)
{
    // this section is to present the transaction huistory on a graph and if applicable write its linear regression equation along side it
    chart5.Series["Transaction history"].Points.Clear();
    chart5.Series["Equation"].Points.Clear();

    DataTable Spent = tables.Optionlist["Spent"];
    DataTable linregtable = tables.Optionlist["CustomerIDRegresion"];
    DataRow[] CustomerLinreg = linregtable.Select("CustomerID=  '" + Customerid + "'");

    if (CustomerLinreg.Length > 0)
    {
        DataTable Trans = tables.Optionlist["MPPerTrans"];
        DataRow[] DRows = Trans.Select("CustomerID=  '" + Customerid + "'");
        int x = 0;
        foreach (DataRow item in DRows)
        {
            chart5.Series["Transaction history"].Points.AddXY((int)item.ItemArray[4], (double)item.ItemArray[1]);

            if ((double)CustomerLinreg[0].ItemArray[1] == 0 && (double)CustomerLinreg[0].ItemArray[2] == 0
                                                            && (double)CustomerLinreg[0].ItemArray[3] == 0)
            {
            }
            else
            {
                chart5.Series["Equation"].Points.AddXY((int)item.ItemArray[4], MakeEquation(CustomerLinreg[0], x));
                x++;
            }
        }

        dataGridView3.DataSource = null;
        dataGridView3.ColumnCount = 4;
        dataGridView3.Columns[0].Name = "X^2";
        dataGridView3.Columns[1].Name = "X";
        dataGridView3.Columns[2].Name = "Y-int";
        dataGridView3.Columns[3].Name = "R^2";
        DataGridViewRow row = (DataGridViewRow)dataGridView3.Rows[0].Clone();
        row.Cells[0].Value = CustomerLinreg[0].ItemArray[1];
        row.Cells[1].Value = CustomerLinreg[0].ItemArray[2];
        row.Cells[2].Value = CustomerLinreg[0].ItemArray[3];
        row.Cells[3].Value = CustomerLinreg[0].ItemArray[4];

        dataGridView3.Rows.Add(row);
        decimal[] Equation = new decimal[2] { (decimal)Convert.ToDecimal((double)CustomerLinreg[0].ItemArray[1])
                                            , (decimal)Convert.ToDecimal((double)CustomerLinreg[0].ItemArray[2]) };
        decimal gradient = GetGradient(Equation, DRows.Length-1);
        if (gradient > 0)
        {
            richTextBox1.Text = "Model predicts a increase in future transaction value";
        }
        else if (gradient < 0)
        {
            richTextBox1.Text = "Model predicts a Decrease in future transaction value";
        }
    }
    else
    {
        richTextBox1.Text = "Graph not avalible insificiant data";
    }
```

Checks if the Customer has a Linear regression Equation within the SQL table aka hasn't gone through the Statistics program

Find every instance of that customer in the transaction Datatable and runs through it

Checks if the customers equation was deemed within reasonable tolerances

If available the equation is added to the graph by utilizing the MakeEquation() method passing the Datarow and a X to get its point on the graph as the graphs X values exist within with relation to themselves so go from 1-> transaction history's length

Clears the grid and fills a graph with the equation data so that the user can understand the equation numerically

Tests if the equation shows a positive or negative future change in transaction value using GetGradient()

Now that we have seen how the tables are presented and used we can move on to how they are created.

The Regression system:

This is the application that handles the Linear regression aspect of the system. Reading and writing data to the same SQL database as the Data Processing and Dashboard programs. It contains 4 created classes with SQLCon being identical to its version in the Dashboard program.

```
◢ SQLCon
    amountofdataRows : int
    filepath : string
    reader : SQLiteDataReader
    SQLCon(string)
    conn : SQLiteConnection
    ExicuteQuery(string) : void
    SQLCon()
◢ Datacreated
    TableNamesDatatypes : Dictionary<DataTable, Tuple<string[], string[]>>
    SaveToSQL(DataTable) : void
    SaveData(DataTable) : void
    MakeColumsreadyForSQL(bool, DataTable) : string
    MakeRowsReadyforSQL(int, DataTable) : string
    InsertData(DataTable) : void
    FillLinregTableDays(DataTable, ref DataTable, ref DataTable) : void
    Tally(int, int, DataTable) : Tuple<List<decimal>, List<decimal>, List<decimal>>
    FillLinregTableCustomer(DataTable, DataTable, ref DataTable) : void
    CreateLinregTableDays(string) : DataTable
    CreateLinregTableCustomerID() : DataTable
◢ DataFromSQL
    FindMMPD() : DataTable
    CreateMPPertrans() : DataTable
    CreateSpent() : DataTable
▷ program
◢ LinearRegression
    YIntersept : decimal
    Slope : decimal
    QuardraticSlope : decimal
    ProductMomentCorilation : decimal
    R2 : decimal
    checkvalid(decimal[], decimal[]) : bool
    LinearRegression(decimal[], decimal[])
    GetResiduals(decimal[], decimal[], decimal[]) : decimal
    GetRSq(decimal[], decimal[], decimal[]) : decimal
    ReturnValues() : decimal[]
    PolyLinRegression(decimal[], decimal[]) : decimal[]
    FindDeterminate(decimal[,]) : decimal
    SimLinRegression(decimal[], decimal[]) : decimal[]
```

```
0 references
class program
{

    0 references
    static void Main(string[] args)
    {
        DataFromSQL Tabledata = new DataFromSQL();

        Datacreated creation = new Datacreated();

        DataTable LinregCustomer = creation.CreateLinregTableCustomerID();
        DataTable QuatersMoney = creation.CreateLinregTableDays("QuatersMoneyLinReg");
        DataTable QuatersFreq= creation.CreateLinregTableDays("QuatersFreqLinReg");

        DataTable MMPerTrans = Tabledata.CreateMPPertrans();
        DataTable Spent = Tabledata.CreateSpent();
        DataTable MMperDay = Tabledata.FindMMPD();
        creation.FillLinregTableCustomer(Spent, MMPerTrans, LinregCustomer);
        creation.FillLinregTableDays(MMperDay, QuatersMoney, QuatersFreq);

        creation.SaveToSQL(LinregCustomer);
        creation.SaveToSQL(QuatersFreq);
        creation.SaveToSQL(QuatersMoney);

        Console.WriteLine("Finished");

        Console.ReadLine();
    }
}
```

LinearRegression class:

Line 670

```
LinearRegression
    YIntersept : decimal
    Slope : decimal
    QuardraticSlope : decimal
    ProductMomentCorilation : decimal
    R2 : decimal
    checkvalid(decimal[], decimal[]) : bool
    LinearRegression(decimal[], decimal[])
    GetResiduals(decimal[], decimal[], decimal[]) : decimal
    GetRSq(decimal[], decimal[], decimal[]) : decimal
    ReturnValues() : decimal[]
    PolyLinRegression(decimal[], decimal[]) : decimal[]
    FindDeterminate(decimal[,]) : decimal
    SimLinRegression(decimal[], decimal[]) : decimal[]
```

```
7 references
class LinearRegression
{

    private decimal YIntersept;
    private decimal Slope;
    private decimal QuardraticSlope = 0;
    private decimal ProductMomentCorilation;
    private decimal R2;
```

```
3 references
public decimal[] ReturnValues()
{
    decimal[] Values = new decimal[5];
    Values[0] = YIntersept;
    Values[1] = Slope;
    Values[2] = QuardraticSlope;
    Values[3] = R2;
    Values[4] = ProductMomentCorilation;
    return Values;
}
```

Includes attributes holding the Equation and the Comparison values created by the system to be returned in the ReturnValues() method.

LinearRegression() Constructor

```
3 references
public LinearRegression(decimal[] Y, decimal[] X)
// we construct the object and create the two equations then compare them and assign the better equation to the objects attributes
{

    if (checkvalid(Y, X))  Checks if the data inputted are all the same value
    {
        if (Y.Length != X.Length)  Checks if there is a Yvalue for every Xvalue inputted
        {
            Slope = 0;
            YIntersept = 0;
            R2 = 0;
            ProductMomentCorilation = 0;
        }
        else if (Y.Length < 3)  Checks if there is even values for a valid graph to be made
        {
            Slope = 0;
            YIntersept = 0;
            R2 = 0;
            ProductMomentCorilation = 0;
        }
        else // to compare the validity between Linuear and Quadratic regression of the same data
        // analyzing and comparing the residuals is a better test of fit
        // R^2 is better for comparing between equations of different data to see which is more valid
        {
            decimal[] Lindata = SimLinRegression(Y, X);
            decimal[] QRdata = PolyLinRegression(Y, X);  Creates the equations storing them as arrays

            decimal LinRSq = GetRSq(Lindata, Y, X);  Gets the Error to equation
            decimal QrRsq = GetRSq(QRdata, Y, X);

            decimal FitLin = GetResiduals(Lindata, Y, X);  Gets the R^2
            decimal FitQuad = GetResiduals(QRdata, Y, X);

            if (LinRSq < 0.4m && QrRsq > 0.4m) // if R^2 is negative it means that the model fits the data worse than a null hypothisis
                                               // so is useless so this test must be completed first to see if comparasons are even useful
            {
                QuardraticSlope = QRdata[0];
                Slope = QRdata[1];
                YIntersept = QRdata[2];
                ProductMomentCorilation = 0; // this statistic is not useful for Non linear data
                R2 = QrRsq;
            }
            else if (LinRSq > 0.4m && QrRsq < 0.4m)  A series of comparisons based on logic described in the design
            {                                         document storying into the attributes the equation found to be
                Slope = Lindata[1];                   better
                YIntersept = Lindata[2];
                R2 = LinRSq;
            }
            else if (LinRSq < 0.4m && QrRsq < 0.4m) // if both are negative both models dont function
            {

                Slope = 0;
                YIntersept = 0;
                R2 = 0;
                ProductMomentCorilation = 0;
            }
            else if (Lindata[1] > 0 && QRdata[0] < 0) // if slope of linear is positive and the quadratic of QRdata is negative as
                                                      // explained in design document wont lead to good predictions
            {
                Slope = Lindata[1];
                YIntersept = Lindata[2];
                R2 = LinRSq;
            }
```

```
            }
            else if (Lindata[1] > 0 && QRdata[0] < 0) // if slope of linear is positive and the quadratic of QRdata is negative as
                                                      // explained in design document wont lead to good predictions
            {
                Slope = Lindata[1];
                YIntersept = Lindata[2];
                R2 = LinRSq;
            }

            else if (Lindata[1] < 0 && QRdata[0] > 0) // same is true of opposite where if there is a negative general
                                                      // trend and a positive quadratic than the quadratic wont be useful at predictions
            {
                Slope = Lindata[1];
                YIntersept = Lindata[2];
                R2 = LinRSq;
            }
            else if (FitLin > FitQuad) // if the quadratic is less than 0.5 than it is not a useful predictor
            {
                Slope = Lindata[1];
                YIntersept = Lindata[2];
                R2 = LinRSq;
            }
            else if (FitLin < FitQuad) // if the values dont satisfy the conditions above than lastly test if the Quadrati
                                       // c R^2 is larger than Linear's
            {
                QuardraticSlope = QRdata[0];
                Slope = QRdata[1];
                YIntersept = QRdata[2];
                ProductMomentCorilation = 0;
                R2 = QrRsq;
            }
            else if (LinRSq - QrRsq < 0.2m) // if the R^2 of the quadratic graph is within 0.2 of the R^2 of the Linear then
                                            // take the Quadratic as the better predictor
            {
                QuardraticSlope = QRdata[0];
                Slope = QRdata[1];
                YIntersept = QRdata[2];
                ProductMomentCorilation = 0;
                R2 = QrRsq;
            }

            else// if no condiction is met go with then the quadratic model is probably not useful but the data is functionaal so the
                // Linear version is safest to use as these have lower error rates
            {
                Slope = Lindata[1];
                YIntersept = Lindata[2];
                R2 = LinRSq;
            }
        }
    }
    else
    {
        QuardraticSlope = 0;
        Slope = 0;
        YIntersept =0;
        ProductMomentCorilation = 0; // this statistic is not useful for Non linear data
        R2 =0;
    }

}
```

PolyLinRegression()

```
1 reference
private decimal[] PolyLinRegression(decimal[] Y, decimal[] X)
{
    //We will be assuming the highest polynomial is 2 so the matrix is 3,3
    //***
    //[0 1 2]
    //[1 2 3]
    //[2 3 4]
    //values needed N, Sum x , Sum Y , Sum X^2, Sum X^3, Sum X^4

    decimal sumx = 0;
    decimal sumy = 0;
    decimal sumYX = 0;
    decimal sumXsqY = 0;
    decimal SumXsq = 0;
    decimal SumXcb = 0;
    decimal SumXqu = 0;

    for (int i = 0; i < Y.Length; i++)
    {
        sumx = sumx + X[i];
        sumy = sumy + Y[i];

        sumYX = sumYX + (X[i] * Y[i]);

        sumXsqY = sumXsqY + ((X[i] * X[i]) * Y[i]);

        SumXsq = SumXsq + (X[i] * X[i]); ;

        SumXcb = SumXcb + (X[i] * X[i] * X[i]);

        SumXqu = SumXqu + (X[i] * X[i] * X[i] * X[i]);
    }
```

Creates the values needed for the later regression equation matrices by running through the equation and summing the quantities needed

$$\begin{pmatrix} N & X & X^2 \\ X & X^2 & X^3 \\ X^2 & X^3 & X^4 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} Y \\ YX \\ YX^2 \end{pmatrix}$$

```
//M
decimal[,] Matrix = new decimal[3, 3]
{
    {Y.Length, sumx,SumXsq },
    { sumx,SumXsq,SumXcb },
    {SumXsq, SumXcb,SumXqu }
};
// Now we generate M0, M1 and M2
decimal[,] M0 = new decimal[3, 3]
{
    {Y.Length, sumx,SumXsq },
    { sumx,SumXsq,SumXcb },
    {SumXsq, SumXcb,SumXqu }
};

M0[0, 0] = sumy;
M0[0, 1] = sumYX;
M0[0, 2] = sumXsqY;
```

$$\begin{pmatrix} Y & X & X^2 \\ X & X^2 & X^3 \\ X^2 & X^3 & X^4 \end{pmatrix}$$

Following the model we need to create the different matrices and fill them with the metrics needed for the model to work, following the points made in the design document

```
decimal[,] M1 = new decimal[3, 3]
{
    {Y.Length, sumx,SumXsq },
    { sumx,SumXsq,SumXcb },
    {SumXsq, SumXcb,SumXqu }
};
M1[1, 0] = sumy;
M1[1, 1] = sumYX;
M1[1, 2] = sumXsqY;
```

$$\begin{pmatrix} N & Y & X^2 \\ X & YX & X^3 \\ X^2 & YX^2 & X^4 \end{pmatrix}$$

```
decimal[,] M2 = new decimal[3, 3]
{
    {Y.Length, sumx,SumXsq },
    { sumx,SumXsq,SumXcb },
    {SumXsq, SumXcb,SumXqu }
};
M2[2, 0] = sumy;
M2[2, 1] = sumYX;
M2[2, 2] = sumXsqY;
```

$$\begin{pmatrix} N & X & Y \\ X & X^2 & YX \\ X^2 & X^3 & YX^2 \end{pmatrix}$$

```
    // we now apply Cramers Rule --- ak = Det(Mi)/det(M)
    // Now we find the determiate of these functions
    // this has to be completed for each of the 3 metricies

    decimal detM0 = FindDeterminate(M0);
    decimal detM1 = FindDeterminate(M1);
    decimal detM2 = FindDeterminate(M2);
    decimal detM = FindDeterminate(Matrix);

    // a2 = X^2     a1 = X    a0 = Y intercept

    decimal a0 = detM0 / detM;
    decimal a1 = detM1 / detM;
    decimal a2 = detM2 / detM;

    decimal[] data = new decimal[3];

    data[0] = a2;
    data[1] = a1;
    data[2] = a0;
    return data;
}
```

FindDeterminate()

```
4 references
private decimal FindDeterminate(decimal[,] Matrix)
{
    decimal determiate = 0;
    // |D|=
    //[a1, b1, c1]
    //[a2, b2, c2]
    //[a3, b3, c3]
    //=
    //a1[b2, c2] -   b1[a2, c2] +   c1[a2, b2]
    //  [b3, c3]       [a3, c3]       [a3, b3]

    //a1(b2)(c3) - a1(b3)(c2) + b1(c2)(a3) - b1(a2)(c3) + c1(a2)(b3) - c1(a3)(b2)
    //A                         B                         C
    //I broke down this equation into 3 more digestible ones



    decimal A = ((Matrix[0, 0]) * (Matrix[1, 1]) * (Matrix[2, 2])) - ((Matrix[0, 0]) * (Matrix[1, 2]) * (Matrix[2, 1]
    decimal B = ((Matrix[1, 0]) * (Matrix[2, 1]) * (Matrix[0, 2])) - ((Matrix[1, 0]) * (Matrix[0, 1]) * (Matrix[2, 2]
    decimal C = ((Matrix[2, 0]) * (Matrix[0, 1]) * (Matrix[1, 2])) - ((Matrix[2, 0]) * (Matrix[0, 2]) * (Matrix[1, 1]


    determiate = A + B + C;
    return determiate;
}
```

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = M \quad |M| = a\begin{pmatrix} e & f \\ h & i \end{pmatrix} - b\begin{pmatrix} d & f \\ g & i \end{pmatrix} + c\begin{pmatrix} d & e \\ g & h \end{pmatrix}$$

|M| = a(e)(i) - a(h)(f) + b(f)(g) - b(d)(i) + c(d)(h) - c(g)(e)

Follows the formula for the determinate of a matrix described in the design document but into 3 values for debugging purposes

GetResiduals()

```
3 references
private decimal GetResiduals(decimal[] data,decimal[] Y, decimal[] X)
{

    decimal YIntersept = data[2];
    decimal Slope = data[1];
    decimal QuardraticSlope = data[0];
    decimal SquaredRegression = 0;

    decimal Srvalue;
    decimal SOsvalue;
    for (int i = 0; i < Y.Length; i++)
    {
        decimal y = 0;
        if (QuardraticSlope == 0)
        {
            y = (decimal)((decimal)Slope * (decimal)X[i]) + (decimal)YIntersept;
        }
        else
        {
            y = ((QuardraticSlope * (X[i] * X[i]))) + (Slope * X[i]) + YIntersept;
        }



        Srvalue = Y[i] - y;

        Console.WriteLine(Srvalue + " Srvalue");

        Srvalue = Srvalue * Srvalue; // squared to keep pos
        SquaredRegression = SquaredRegression + Srvalue;


        Console.WriteLine(SquaredRegression + " SquaredReg");


    }//gets Total squares

    return SquaredRegression;
}
```

GetRSq()

```
2 references
private decimal GetRSq(decimal[] data, decimal[] Y, decimal[] X)
{
    //R^2 = 1- Sum SquaredRegression / Total SumofSquares

    //for Total SumofSquares you need to find the Mean of Yvalues first
    decimal total = 0;
    for (int i = 0; i < Y.Length; i++)
    {
        total = total + Y[i];
    }
    decimal mean = total / Y.Length;
    // then you find the some of the predicted values

    //the some of Squared regression is the difference between the predicted values
    //and the values observed then squared so they are always positive
    decimal SquaredRegression = 0;
    decimal SomeOfSquaares = 0;
    decimal Srvalue;
    decimal SOsvalue;

    for (int i = 0; i < Y.Length; i++)
    {
        SOsvalue = Y[i] - mean;

        SOsvalue = SOsvalue * SOsvalue;  // squared to keep pos

        SomeOfSquaares = SomeOfSquaares + SOsvalue;
    }//gets Total SquaredRegression

    SquaredRegression = GetResiduals(data,Y, X);

    decimal Rsq = 1m - (SquaredRegression / SomeOfSquaares);

    return Rsq;
}
```

$$R^2 = 1 - \frac{Error\ to\ equation}{Error\ to\ mean}$$

Follows the R^2 formula and pseudocode method as described in the design document

Follows the R^2 formula and pseudocode method as described in the design document

Utilizes a method for GetResiduals() as both this method and the constructor requires this value aswell

SimLinRegression()

```
1 reference
private decimal[] SimLinRegression(decimal[] Y, decimal[] X)
{
    //utalizes a non matrix formula to increase proformance as this produces the same simple
    //Linear regression equation through a more manual math model that only works for
    //Simple equations
    decimal sumx = 0;
    decimal sumy = 0;
    decimal SumXsq = 0;
    decimal SumYsq = 0;
    decimal SumXY = 0;

    for (int i = 0; i < Y.Length; i++)
    {
        sumx = sumx + X[i];
        sumy = sumy + Y[i];

        SumXsq = SumXsq + (X[i] * X[i]);
        SumYsq = SumYsq + (Y[i] * Y[i]);

        SumXY = SumXY + (X[i] * Y[i]);
    }

    decimal B1 = (Y.Length * SumXY) - (sumx * sumy);
    decimal B2 = (Y.Length * SumXsq) - (sumx * sumx);
    decimal B = B1 / B2;
    //Split up equation for debugging
    //B = Coeffienent A = Y intersept

    decimal a1 = (decimal)(1m / Y.Length) * sumy;
    decimal a2 = (B * (decimal)(1m / Y.Length) * sumx);
    decimal A = a1 - a2;

    // Now we work out Product Moment corrilation

    decimal ProdMomCorl1 = Y.Length * SumXY - (sumx * sumy);
    decimal ProdMomCorl2 = ((Y.Length * SumXsq) - (sumx * sumx)) * ((Y.Length * SumYsq) - (sumy * sumy));
    ProdMomCorl2 = Convert.ToDecimal(Math.Sqrt(Convert.ToDouble(ProdMomCorl2)));
    decimal ProdMomCorl = ProdMomCorl1 / ProdMomCorl2;

    decimal[] data = new decimal[3];
    data[0] = 0;
    data[1] = B;
    data[2] = A;

    ProductMomentCorilation = ProdMomCorl;
    return data;

}
```

Creates the values needed for the simple model by running through the equation and summing the quantities needed

Checkvalid()

```
1 reference
private bool checkvalid(decimal[] Y, decimal[] X)
{
    if (Y.Length > 1)
    {
        List<decimal> TestforUnique = new List<decimal>();
        TestforUnique.Add(Y[0]);
        for (int i = 1; i < Y.Length; i++)
        {
            if (!TestforUnique.Contains(Y[i])) // checks if there are at least 2 different values
            {
                return true;
            }
        }
        return false;
    }
    else
    {
        return false;
    }
}
```

Datacreated Class:

Line42

This class holds the methods to Create data to be held within the SQL database. It mainly handles light data processing before the Linear regression takes place.

```
Datacreated
    TableNamesDatatypes : Dictionary<DataTable, Tuple<string[], string[]>>
    SaveToSQL(DataTable) : void
    SaveData(DataTable) : void
    MakeColumsreadyForSQL(bool, DataTable) : string
    MakeRowsReadyforSQL(int, DataTable) : string
    InsertData(DataTable) : void
    FillLinregTableDays(DataTable, ref DataTable, ref DataTable) : void
    Tally(int, int, DataTable) : Tuple<List<decimal>, List<decimal>, List<decimal>>
    FillLinregTableCustomer(DataTable, DataTable, ref DataTable) : void
    CreateLinregTableDays(string) : DataTable
    CreateLinregTableCustomerID() : DataTable
```

```
2 references
class Datacreated
{
    Dictionary<DataTable, Tuple<string[], string[]>> TableNamesDatatypes = new Dictionary<DataTable, Tuple<string[], string[]>>();
```

Includes a Attribute of a Dictionary of Datatables and tuples of 2 string arrays. During the creation of the Datatables within this class the SQL datatypes and names required for saving to the SQL database are loaded into a array which is then loaded into a tuple and a Dictionary of Datatables to tuples so that upon needing to save the datatypes can be pulled out.

CreateLinregTableCustomerID()

```
1 reference
public DataTable CreateLinregTableCustomerID()
{
    DataTable LinRegtable = new DataTable("CustomerIDRegresion");
    string[] dattypes = new string[6];
    string[] Tablenames = new string[6];
    {//creating columns
     //
        DataColumn column; //MoneySpent,InvoiceDate,CustomerID

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.String");
        column.ColumnName = "CustomerID";
        column.ReadOnly = true;
        LinRegtable.Columns.Add(column);

        dattypes[0] = "VARCHAR";
        Tablenames[0] = "CustomerID";

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "SlopeXsq";
        column.ReadOnly = true;
        LinRegtable.Columns.Add(column);

        dattypes[1] = "DOUBLE";
        Tablenames[1] = "SlopeXsq";

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "SlopeX";
        column.ReadOnly = true;
        LinRegtable.Columns.Add(column);

        dattypes[2] = "DOUBLE";
        Tablenames[2] = "SlopeX";

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "Yintercept";
        column.ReadOnly = true;
        LinRegtable.Columns.Add(column);

        dattypes[3] = "DOUBLE";
        Tablenames[3] = "Yintercept";

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "RSquared";
        column.ReadOnly = true;
        LinRegtable.Columns.Add(column);
```

The first DataCreated object is called with its first method being CreateLinregTableCustomerID(). This is a simple method to create a datatable for the holding of Customer Regression equation with the appropriate datatypes and columns for its function. Full breakdown of all Datatable construction can be found in the Code appendix B.

CreateLinregTableDays()

This creates a Datatable for the Quarters function for holding the regression equation of each quarter. Takes in a parameter for what the title of the Datatable should be so that they can both use the same method for creating their structure so that any change effects both centrally.

FillLinregTableCustomer()

```
1 reference
public void FillLinregTableCustomer(DataTable Spent, DataTable MMPerTrans, ref DataTable Linreg)
{
    for (int i = 0; i < Spent.Rows.Count; i++) // need to tally them into days to get accurate data
    { // uses Spent Datable as a store of CustomerID
        Console.WriteLine(Spent.Rows[i].ItemArray[1]);
        DataRow[] Found = MMPerTrans.Select("CustomerID = '" + (string)Spent.Rows[i].ItemArray[1] + "'");
        //*** Gets a array of rows that have the peice of data within Spent.Rows[i].ItemArray[1]
        // Casts to string as the Spent.Rows[i].ItemArray[1] is treated as a object that has edata within it

        if (Found.Length > 1) // this is to check if the minimum amount of data required is found aka more than 1
        {

            List<double> Money = new List<double>();
            Dictionary<int, double> DayToMoney = new Dictionary<int, double>();
            //utalizes the normalized date that makes 2016 date negative
            //dictionary structure to link day to the tally of money spent
            foreach (DataRow item in Found) //goes through each datarow and adding a tally of money spent
            {

                Money.Add((double)item.ItemArray[1]);

            }
            // setting up data for the perpouses of the graph
            decimal[] xValues = new decimal[Money.Count];

            decimal[] yValues = new decimal[Money.Count];
            for (int j = 0; j  < Money.Count; j ++)
            {
                xValues[j] =j; // X values exit in relation to echother not Time as this simplifies
                               // the scale of each graph so that their equations canbe presented
                               // easily on the same chart with no normalization of scale
                yValues[j] = (decimal)Money[j];
            }

            LinearRegression Regresion = new LinearRegression(yValues, xValues);
            // Loads the two data axis to the LinRegression Constructor
            decimal[] values = Regresion.ReturnValues();

            //Creates new data row in Datatable
            DataRow row;
            row = Linreg.NewRow();
            row["CustomerID"] = Spent.Rows[i].ItemArray[1];
            row["SlopeXsq"] = values[2];
            row["SlopeX"] = values[1];
            row["Yintercept"] = values[0];
            row["RSquared"] = values[3];
            row["ProductMomentCorilation"] = values[4];
            Linreg.Rows.Add(row);
        }

    }
}
2 references
```

FillLinregTableDays()

```
1 reference
public void FillLinregTableDays(DataTable MMperDay, ref DataTable QuatersMoney, ref DataTable QuatersFreq)
{
    //now first we must seperate the dates between quarters
    // We will overall have 365 data points however for testing I need to break up any data inputted into quarters
    List<int> days = new List<int>();
    MMperDay.DefaultView.Sort=("Day"); // places them in order of Days
    foreach (DataRow item in MMperDay.Rows)
    {
        days.Add((int)item.ItemArray[1]);
    }
    if (days.Count <4) // if the number of days in the system is lower than 4 quaters aren't possible
    {
    }
    else
    {
        double quarter = (days.Count / 4);
        int Q = Convert.ToInt16(Math.Round(quarter));

        Tuple<List<decimal>, List<decimal>, List<decimal>> Q1 = Tally(0, Q, MMperDay);
        Tuple<List<decimal>, List<decimal>, List<decimal>> Q2 = Tally(Q, Q, MMperDay);
        Tuple<List<decimal>, List<decimal>, List<decimal>> Q3 = Tally(Q + Q, Q, MMperDay);
        Tuple<List<decimal>, List<decimal>, List<decimal>> Q4 = Tally(Q + Q + Q, Q, MMperDay);
        Tuple<List<decimal>, List<decimal>, List<decimal>>[] quaters = new Tuple<List<decimal>, List<decimal>, List<decimal>>[4];
        quaters[0] = Q1;
        quaters[1] = Q2;
        quaters[2] = Q3;
        quaters[3] = Q4;
        int index = 1;
        foreach (var item in quaters)
        {// all the data should be sorted and connected in order as they where added to a list at the same time with
        // no operations completing between them that effect order
            decimal[] day = item.Item1.ToArray();
            decimal[] Money = item.Item2.ToArray();
            decimal[] Freq = item.Item3.ToArray();
            LinearRegression money = new LinearRegression(Money, day);
            decimal[] valuesM = money.ReturnValues();
            DataRow rowM;
            rowM = QuatersMoney.NewRow();
            rowM["Quater"] = index;

            rowM["SlopeXsq"] = valuesM[2];
            rowM["SlopeX"] = valuesM[1];
            rowM["Yintercept"] = valuesM[0];
            rowM["RSquared"] = valuesM[3];
            rowM["ProductMomentCorilation"] = valuesM[4];
            QuatersMoney.Rows.Add(rowM);
            LinearRegression Frequency = new LinearRegression(Freq, day);
            decimal[] valuesF = Frequency.ReturnValues();
            DataRow rowF;
            rowF = QuatersFreq.NewRow();
            rowF["Quater"] = index;

            rowF["SlopeXsq"] = valuesF[2];
            rowF["SlopeX"] = valuesF[1];
            rowF["Yintercept"] = valuesF[0];
            rowF["RSquared"] = valuesF[3];
            rowF["ProductMomentCorilation"] = valuesF[4];
            QuatersFreq.Rows.Add(rowF);

            index++;
        }
    }
}
4 references
```

IAs the amount of rows and therefor the amount of days loaded into the system is flexible The system cannot use set values for quarters. Instead it creates it using the number of days loaded into the system.

Utilizes the Tally() method inputting the start of each new quarter (by adding a Q each time it is called), the length of the quarter and the MMperDay Datatable which holds the Per Day data created by the data processing application.

Runs through each quarter tuple turning each list intro a array

Made to create a MoneyMade PerDay Regression and a Frequency so creates two objects and extracts data from them for the two separate Datatables of Frequency and Moneymade

Tally()

```
4 references
private Tuple<List<decimal>, List<decimal>, List<decimal>> Tally(int day, int quarter, DataTable MMperDay)
{
    List<decimal> Day = new List<decimal>();
    List<decimal> Money = new List<decimal>();
    List<decimal> Freq = new List<decimal>();
    int NumberOfDays = day + quarter;
    while (NumberOfDays >= MMperDay.Rows.Count)
        // this is to make sure this for loop doesnt exeed the bounds of the array as the quater might be
        // larger than a quater as the size of the data inpiutted during testing is flexible
    {
        NumberOfDays--;
    }
    for (int i = day; i < NumberOfDays; i++)
    {
        Day.Add((decimal)Convert.ToDecimal(MMperDay.Rows[i].ItemArray[1]));
        Money.Add((decimal)Convert.ToDecimal(MMperDay.Rows[i].ItemArray[2]));
        Freq.Add((decimal)Convert.ToDecimal(MMperDay.Rows[i].ItemArray[3]));
    }
    Tuple<List<decimal>, List<decimal>, List<decimal>> data = new Tuple<List<decimal>, List<decimal>, List<decimal>>(Day, Money, Freq);
    return data;
}
```

SaveToSQL()

```
3 references
public void SaveToSQL(DataTable Sheet)
{
    SaveData(Sheet);
    InsertData(Sheet);
}
```

SaveData

```
1 reference
private void SaveData(DataTable Sheet)
{
    SQLCon link = new SQLCon();

    string CreateTable2 = "drop table if exists " + Sheet.TableName;
    Console.WriteLine(CreateTable2);
    link.ExicuteQuery(CreateTable2);

    string CreateTable = "CREATE TABLE" + " " + Sheet.TableName + "( " + MakeColumsreadyForSQL(false, Sheet) + " );"; ;
    Console.WriteLine(CreateTable);

    link.ExicuteQuery(CreateTable);

}
```

The SaveData method is a generalized method for saving the data within a datatable to the SQL database the data was originally requested from. Works by using the queries drop table if exists and create table using the method MakeColumnsReadyForSQL for the tables creation. It then calls the InsertDatamethod.

MakeColumnsReadyForSQL()

```
1 reference
private string MakeRowsReadyforSQL(int i, DataTable Sheet)
{
    string data = "";
    for (int j = 0; j < Sheet.Columns.Count; j++)
    {
        if (j == (Sheet.Columns.Count - 1))
        {
            data = data + Sheet.Rows[i].ItemArray[j];
        }
        else
        {
            data = data + Sheet.Rows[i].ItemArray[j] + ", ";
        }
    }
    return data;
}
```

The MakeColumnsReadyForSQL method has a use in the Savedata method as well as the Insertdata method using a bool parameter to distinguish these two different calls. Works by utilizing the Tuple of Datatables to Datatypes/Columnnames to create the correct SQL syntax to Create a table.

InsertData()

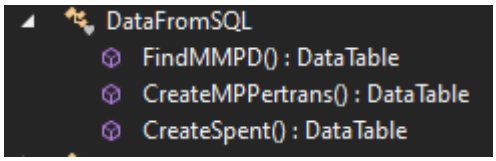
```
1 reference
private void InsertData(DataTable Sheet)
{
    SQLCon link = new SQLCon();
    for (int i = 0; i < Sheet.Rows.Count; i++)
    {
        string InsertText = "INSERT INTO" + " " + Sheet.TableName + "( " + MakeColumsreadyForSQL(true, Sheet) + ")"
                            + " VALUES(" + MakeRowsReadyforSQL(i,Sheet) + "); ";
        Console.WriteLine(InsertText);
        link.ExicuteQuery(InsertText);
    }
}
```

Utilizes the MakeColumnsReadyForSQL method passing a true bool parameter to get the syntax required for a insert query. Utalizes the MakeRowsReadyforSQL() method

MakeRowsReadyforSQL()

DataFromSQL class:

Line424



The class for requesting data from the SQL database which is used by the other classes. Its functional purpose identical to the DataFromSQLTables class of the dashboard and sharing several identical methods creating the same Datatables and calling the same data from the SQL database.

FindMMPD()

Returning a created Datatable with its rows filled.

```
1 reference
public DataTable FindMMPD()
{
    {
        SQLCon connection = new SQLCon("MoneyMadePerDay");
        SQLiteDataReader rdr = connection.reader;
        DataTable MMPerday = new DataTable("MoneyMadePerDay");

        DataColumn column;

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "PrimaryKey";
        column.ReadOnly = true;
        MMPerday.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "Day";
        column.ReadOnly = true;
        MMPerday.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Double");
        column.ColumnName = "MoneySpent";
        column.ReadOnly = false;
        MMPerday.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "Frequencyofsale";
        column.ReadOnly = false;
        MMPerday.Columns.Add(column);

        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "Year";
        column.ReadOnly = false;
        MMPerday.Columns.Add(column);

        DataRow row;

        while (rdr.Read())
        {
            row = MMPerday.NewRow();
            row["PrimaryKey"] = rdr.GetString(0);
            Console.WriteLine(rdr.GetInt32(1));
            row["Day"] = rdr.GetInt32(1);
            row["MoneySpent"] = rdr.GetInt32(2);
            row["Frequencyofsale"] = rdr.GetInt32(3);
            row["Year"] = rdr.GetInt32(4);

            MMPerday.Rows.Add(row);
        }
        return MMPerday;

    }
}
```

CreateMPPertrans()

Returning a created Datatable with its rows filled through a identical method to MMPD.

CreateSpent()

Returning a created Datatable with its rows filled through a identical method to MMPD.

Data processing system:

Is the root of all previous systems. Takes in the one SQL table and through data processing abstracts into useful tables and metrics used throughout the previous systems. Uses a heavily object oriented approach with all but 1 object inheriting rom the abstract table class and using its methods heavily throughout the program. The general function of each class is to create a Datatable and then through input table(s) that feed data into it abstract and create the row data to fill it.

```
Program.cs
  SQLInteractions
  MoneyPerDay
    Create(DataSet) : DataTable
    CheckDate(ref DataTable, DataTable, ref int, int, ref int, ref int) : DataTable
    FindMMPD(DataSet) : DataTable
  CQI
    Create(DataSet) : DataTable
    AddData(DataSet) : DataTable
  SpentOver
    Create(DataSet) : DataTable
    IDSpentOvertime(ref DataTable, DataSet) : void
    TallyMP(ref Dictionary<int, double>, DataSet) : Dictionary<int, double>
  MMpertrans
    Create(DataSet) : DataTable
    FILLrowsMP(ref DataTable, DataSet) : void
    ADDcolumstoMP(ref DataTable) : void
  I1st
    tableNames : string[]
    addValues(string[]) : void
    returnvalues() : string[]
  Table
    Sheet : DataTable
    amountofdataRows : int
    filepath : string
    datatypes : I1st
    tablenames : I1st
    Create(DataSet) : DataTable
    GetDate(string) : Tuple<int, int>
    getUQID(string, string, string) : void
    CreateMesureOfSpreadDocument(string, string) : void
    MesuresOfSpread(ref double[], ref double[]) : double[]
    GetSD(double[]) : double
    SaveData() : void
    MakeColumsreadyForSQL(bool) : string
    CreateSQLconnection() : SQLiteConnection
    MakeRowsReadyforSQL(int) : string
    InsertData() : void
  MainSet
    dataSet : DataSet
    PreliminaryCleaning() : void
    MainSet()
    remove(DataTable) : void
    ADDcolums(ref DataTable) : void
    CreateSQLconnection() : SQLiteConnection
    ADDrows(ref SQLiteDataReader, ref DataTable) : void
    GetFormat(string) : string
  Program
```

```
0 references
class Program
{

    0 references
    static void Main(string[] args)
    {

        MainSet Mainset = new MainSet(true);

        MMpertrans Pertrans = new MMpertrans();

        Mainset.dataSet.Tables.Add(Pertrans.Create(Mainset.dataSet));

        SpentOver Spent = new SpentOver();

        Mainset.dataSet.Tables.Add(Spent.Create(Mainset.dataSet));

        CQI CQItable = new CQI();

        Mainset.dataSet.Tables.Add(CQItable.Create(Mainset.dataSet));

        MoneyPerDay MMPD = new MoneyPerDay();

        Mainset.dataSet.Tables.Add(MMPD.Create(Mainset.dataSet));

        Pertrans.SaveData();

        Spent.SaveData();

        CQItable.SaveData();

        MMPD.SaveData();

        Pertrans.CreateMesureOfSpreadDocument("MPPerTrans", "MoneySpent");
        Spent.CreateMesureOfSpreadDocument("Spent", "MoneySpentOvertime");
        CQItable.CreateMesureOfSpreadDocument("CQI", "CQI");
        CQItable.CreateMesureOfSpreadDocument("CQI", "Frequency");
        MMPD.CreateMesureOfSpreadDocument("MoneyMadePerDay", "MoneySpent");
        MMPD.CreateMesureOfSpreadDocument("MoneyMadePerDay", "Frequencyofsale");

        Pertrans.getUQID("MPPerTrans", "MoneySpent", "CustomerID");
        Spent.getUQID("Spent", "MoneySpentOvertime", "CustomerID");
        CQItable.getUQID("CQI", "CQI", "CustomerID");
        CQItable.getUQID("CQI", "Frequency", "CustomerID");
        MMPD.getUQID("MoneyMadePerDay", "MoneySpent", "Day");
        MMPD.getUQID("MoneyMadePerDay", "Frequencyofsale", "Day");

        Console.WriteLine("stuff");

        Console.ReadLine();
    }
}
```

Table abstract class :

Line593

```
Table
    Sheet : DataTable
    amountofdataRows : int
    filepath : string
    datatypes : l1st
    tablenames : l1st
    Create() : DataTable
    GetDate(string) : Tuple<int, int>
    getUQID(string, string, string) : void
    CreateMesureOfSpreadDocument(string, string) : void
    MesuresOfSpread(ref double[], ref double[]) : double[]
    GetSD(double[]) : double
    SaveData() : void
    MakeColumsreadyForSQL(bool) : string
    CreateSQLconnection() : SQLiteConnection
    MakeRowsReadyforSQL(int) : string
    InsertData() : void
```

The abstract table class is the backbone of this system so to start I will describe the table object as it will be referenced several times in the rest of the system. The code for this class can be found on line 720 of the C# code of the Data processing on page **Error! Bookmark not defined.** of this document.

```
5 references
abstract class Table
{

    protected DataTable Sheet;
    protected const int amountofdataRows =100000;
   protected string filepath = Environment.GetFolderPath(
                                Environment.SpecialFolder.MyDocuments) + "\\CustomerData.db";

    protected l1st datatypes = new l1st();
    protected l1st tablenames = new l1st();
```

The protected Sheet attribute allows for the Table's methods to refer to a singular variable name while being applicable all different Datatables within the subclasses, uses from Sheet.Rows.Count and Sheet.Columns.Count for any method's for loops to Sheet.Tablename when saving to SQL

A protected const called amountofdataRows allows for the central controlling of the number of rows loaded into the system allowing for several different ranges of the dataset to be analysed.

A protected string filepath = holds the filepath to the SQL database for use in requesting data from the SQL database. Uses the environmental class to allow for users computers to be easily searched for the presence of the database in the documents folder.

A Protected l1st datatypes is an example of a class I made that exists to solve the problem that any collections.Generic collection like a List cannot be inherited by subclasses. To get

around this I made a class called L1ist that allows for a string array to be added to and returned in a easily inheritable way. It holds data types of the columns made by the subclass as to create a table a datatype must be assigned and Datatables datatypes do not match SQL syntax so must be held in a consistent format so that saving methods can be universal.

The List of Column names exist to be added along side this l1st for debugging perpouses to see which Column is causing a datatype involved crash

Methods within the class:

Create()

```
12 references
virtual public DataTable Create(DataSet M )
{
    return null;
}
```

This is a virtual class that makes other classes override it with their own version. This means that every subclass that inherits from table has an uniformly declared create method that returns a debatable. This is so how, where and when different datatables are declared created and filled can be easily identified and understood between classes as all non SQL filled classes have a Create() method that returns a DataTable.

GetDate()

```
3 references
public Tuple<int,int> GetDate(string date)
{
    //This class converts a string date format of date-month-year into numerical formats
    //using substrings.
    //the date format date-month-year

    int day = Convert.ToInt32(date.Substring(0, date.IndexOf('-')));
    string s = date.Substring(date.IndexOf('-') + 1, 2);
    Console.WriteLine(s);
    int month = Convert.ToInt32(s);
    int year = Convert.ToInt32(date.Substring(date.LastIndexOf('-') + 1));

    int dayOyear = day + (30) * (month - 1);

    Tuple<int, int> DayandYear = new Tuple<int, int>(dayOyear, year);

    return DayandYear;
}
```

This class converts a string date format of date-month-year into numerical formats using substrings. It returns a tuple of both its numerical date and its year. This is useful for several different data tables creation so it was placed within the table class so that they can all access a centralized and uniform method of conversion for easy debugging and changing.

CreateMesureOfSpreadDocument()

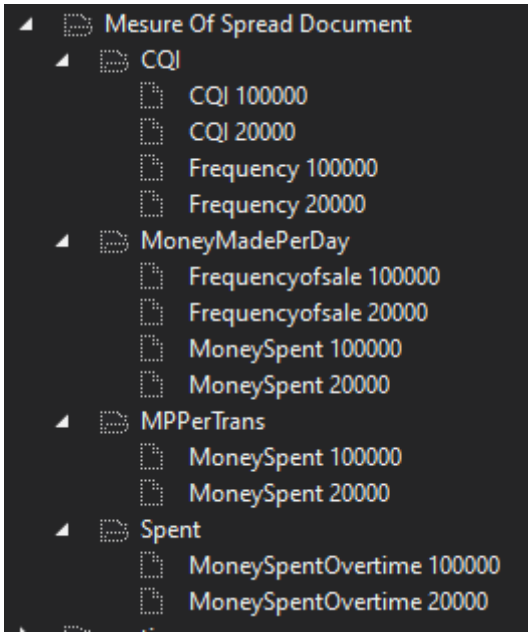
```
12 references
public void CreateMesureOfSpreadDocument(string Name, string Columnname)
{
    List<double> ValuesTouse = new List<double>();
    for (int i = 0; i < this.Sheet.Rows.Count; i++)
    {
      ValuesTouse.Add((double)Convert.ToDouble(this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(Columnname)]));
    }
    double[] CompiledData = ValuesTouse.ToArray();
    double[] MMSpread = new double[8];
    MesuresOfSpread(ref CompiledData, ref MMSpread);
    if (!Directory.Exists("Mesure Of Spread Document"))
    {
        Directory.CreateDirectory("Mesure Of Spread Document");
    }
    if (!Directory.Exists("Mesure Of Spread Document\\" + Sheet.TableName))
    {
        Directory.CreateDirectory("Mesure Of Spread Document\\" + Sheet.TableName);
    }
    File.Create("Mesure Of Spread Document\\" + Sheet.TableName + "\\  " + Columnname + " " + amountofdataRows).Dispose();

        using (StreamWriter writer = new StreamWriter("Mesure Of Spread Document\\" + Sheet.TableName +
                                                      "\\  " + Columnname + " " + amountofdataRows))
        {
            foreach (double item in MMSpread)
            {
                writer.WriteLine(item);
            }
            writer.WriteLine();
            writer.WriteLine("1. mean 2. SD 3. IQR 4.Min - Max 5-7. Quartiles");
            writer.WriteLine(Columnname);
        }
}
```

This method is to make a measures of spread document for the table the method is called with. It takes in 2 parameters (string Name, Columnname) the name is the name of the table and is mainly for debugging purposes.

The column name is used to index which column is tested as some tables have 2 quantities that are worth testing so to keep the method generalize using a parameter to select the column

the system instead uses Sheet.TableName for whenever the name of the table within a subclass is referred to during the operation of any method. allowing for methods to be generalized for any table

First it loads a double List with all values within the column tested using a for loop with Sheet.Rows.Count to allow for different sheet lengths to use it. After creating the two double arrays Loads them into the Measures of Spread method which returns measures of spreads for the double array inputted. This is done through the use of Math.Net methods as well as a custom Get standard deviation method.



It first checks if there is a folder of measures of spreads if not creates one then checks if there is a folder within this folder that holds the measures of spread of this specific table names spreads if not creates one. Within the measures of spread folder within the tables specific folder it then creates a file that writes that it's a measure of spread what table its for and what column it is referring to and the number of rows loaded in to create it. This creates the following file structure. This means that each file created in this system has a specific area allocated to increase organization. Every file created within this system follows this generalized system to make the bin file uncluttered and easy to find what the user needs.

MesuresOfSpread()

The Create measure of spread process completes to the file organization handling part of the methood. It first checks if there is a folder of measures of spreads if not creates one then checks if there is a folder within this folder that holds the measures of spread of this specific table names spreads if not creates one. Within the measures of spread folder within the tables specific folder it then creates a file that writes that it's a measure of spread what table its for and what column it is referring to and the number of rows loaded in to create it. This allows for easy file management as each type of file created by the program. The program then calls a streemwriter to write the data from the array holding the measures of spread into the measures of spread document then labelling them bellow.

getUQID()

```
12 references
public void getUQID(string name, string Columnname, string Col2)
{
    // takes in the parameters name, Columnname and Col2, the first 2 work similar to the measures of spread document however Col2 is the
    // name of the rows column we want the system to store if a row's Columnname is greater than the Upper quartile
    //gets the top quartile of customers
    StreamReader reader = new StreamReader("Mesure Of Spread Document\\" + Sheet.TableName + "\\  " + Columnname + " " + amountofdataRows);
    string UQT = "";
    for (int y = 0; y < 8; y++)
    {
        UQT = reader.ReadLine();
    }//puts the reader on the right line
    reader.Close();
    double UQ = Convert.ToDouble(UQT);

    if (!Directory.Exists("top25"))
    {
        Directory.CreateDirectory("top25");
    }
    if (!Directory.Exists("top25\\" + Sheet.TableName))
    {
        Directory.CreateDirectory("top25\\" + Sheet.TableName);

    }
    File.Create("top25\\" + Sheet.TableName + "\\  " + Sheet.TableName + " " + amountofdataRows).Dispose();
    using (StreamWriter writer = new StreamWriter("top25\\" + Sheet.TableName + "\\  " + Columnname + " " + amountofdataRows))
    {
        List<double> top25ID = new List<double>();
        bool Freq = false;
        try
        {
            string checkF = Columnname.Substring(0, 9);
            if (checkF == "Frequency")
            {
                Freq = true;
            }
        }
        catch
        {
        }
        for (int i = 0; i < Sheet.Rows.Count; i++)
        {
            //Frequency
            if (Freq == true)// stored as int so has to be cast as int value
            {
                double TestColumn = (int)this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(Columnname)];
                Math.Round(TestColumn);

                if (TestColumn > UQ)
                {
                    int j = Convert.ToInt32(this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(Col2)]);
                    top25ID.Add(j);
                }
            }
```

reading the file created in the create measure of spread method. It reads the 8th element of the document which is the upper quartile into a string variable called UQT.

Cast The value explicitly as an int if the column name is Frequency

If the column tested value is greater store the Col2 value store Col2 value

This method system then runs through all columns in the sheet storing in a list a rows Col2 value if the rows columnname value is larger than the Upper quartile. The method then writes to the top25 file list of upper quartile Col2 values.

The method than declares a new instance of a SQLiteractions class which is almost identical across all programs through its general design. The SQLiteractions object is then sent a instruction to drop the table if exists into SQLiteractions's method Exicutequery and after the instruction to create the table into SQLiteractions's method Exicutequery which simply loads the query and executes it. The rows of this new table are then filled with the List values.

SaveData()

```
8 references
public void SaveData()
{
    SQLInteractions link = new SQLInteractions();
    //create table if not exists TableName (col1 typ1, ..., colN typN)
    //drop table if exists TableName

    string CreateTable2 = "drop table if exists " + Sheet.TableName;

    link.ExicuteQuery(CreateTable2);


    string CreateTable = "CREATE TABLE" + " " + Sheet.TableName + "( " + MakeColumsreadyForSQL(false) + " );"; ;

    link.ExicuteQuery(CreateTable);
    InsertData();


}
```

Works as a perfect example of the OOP inherited approach that the abstract Table class was built for as without any further inputs any table that inherits from this class can built a table and save its data to the SQL database through one method call. This method is a more generalized OOP version of the SaveToSQL() method of the Statistical analysis system only needing to be called. The method only uses the sub classes data instead of parameters to create the SQL table to save the data. Utilizes The MakeColumnsreadyForSQL in the same way.

MakeColumnsreadyForSQL()

```
2 references
private string MakeColumsreadyForSQL(bool Rows, DataTable Sheet)
// has a bool to see if this is for the insert or not as the data types are used
// to create the table but not when inserting
{

    string[] Datatypes = TableNamesDatatypes[Sheet].Item2;

    string data = "";
    for (int i = 0; i < Sheet.Columns.Count; i++)
    {
        if (i == (Sheet.Columns.Count - 1))
        {
            if (Rows == true)
            {
                data = data + Sheet.Columns[i].ColumnName + " ";
            }
            else
            {
                data = data + Sheet.Columns[i].ColumnName + " " + Datatypes[i];
            }
        }
        else
        {
            if (Rows == true)
            {
                data = data + Sheet.Columns[i].ColumnName + "," + " ";
            }
            else
            {
                data = data + Sheet.Columns[i].ColumnName + " " + Datatypes[i] + ", ";
            }
        }
    }
    return data;
}
```

Works in the same way in Datacreated Class. The MakeColumnsReadyForSQL method has a use in the Savedata method as well as the insertdata method using a bool parameter to distinguish these two different calls. Works by utalizing the L1st of Datatypes inherited by the subclass and assigned into during creation of datatable's columns. Uses a for loop of the sheet's number of columns and in each iteraction adds to a string variable called data the name of the column as well as any extra formatting the SQL syntax requires, this includes on the last loop leaving out the comma. The string value data is then returned

InsertData()

Works in the same way in Datacreated Class

MakeRowsReadyforSQL()

Works in the same way in Datacreated Class

MainSet : Table class

Line909

This is the first table created and is the only table within the system to read in data from the SQL database as it holds the original dataset that all other tables are abstracted from. It is the most different class that inherits from the abstract Table class.

```
MainSet
    dataSet : DataSet
    PreliminaryCleaning() : void
    MainSet()
    remove(DataTable) : void
    ADDcolums(ref DataTable) : void
    CreateSQLconnection() : SQLiteConnection
    ADDrows(ref SQLiteDataReader, ref DataTable) : void
    GetFormat(string) : string
```

```
class MainSet : Table
{
    public DataSet dataSet = new DataSet();
    1 reference
```

Has a attribute of Dataset which is a inbuilt class and hold a list of Datatables. This public attribute is fed as a parameter into many other classes' methods so that each class has access to every table in the system if needed.

MainSet() Constructor

```
1 reference
public MainSet()
{
    PreliminaryCleaning();
    string command1 = "SELECT * FROM CustomerData LIMIT " + amountofdataRows;
    SQLiteConnection connection = CreateSQLconnection();
    using var cmd = new SQLiteCommand(command1, connection);
    SQLiteDataReader rdr = cmd.ExecuteReader();

    System.Data.DataTable table = new DataTable("MotherDF");

    ADDcolums(ref table);

    ADDrows(ref rdr, ref table);

    dataSet.Tables.Add(table);

     remove(table);

}
```

As this is the only class where data from the SQL database is read from the SQL interactions class does not include a dedicated method for generalized table reading like in the other systems.

PreliminaryCleaning()

```
private void PreliminaryCleaning()
{//Cleans known errors before they have to be loaded and unloaded into the datatable
    SQLInteractions Removal = new SQLInteractions();
    string text = "delete from CustomerData where InvoiceNo=' ' OR InvoiceNo IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where StockCode=' ' OR StockCode IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where Description=' ' OR Description IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where Quantity=' ' OR Quantity IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where InvoiceDate=' ' OR InvoiceDate IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where UnitPrice=' ' OR UnitPrice IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where CustomerID=' ' OR CustomerID IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where Country=' ' OR Country IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where InvoiceNo=' ' OR InvoiceNo IS NULL;";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where StockCode='M';";
    Removal.ExicuteQuery(text);
    text = "delete from CustomerData where Quantity<0;";
    Removal.ExicuteQuery(text);

    Removal.Close();
}
```

This method removes general errors that could arise from the dataset before the data is loaded into the system. These error states were found through iterative testing by increasing the amount of rows loaded in 50,000 at a time fixing errors when they arise.

ADDColumns()

Constructs a Datatable in a similar way to Datacreated full breakthrough found in code Appendix B.

ADDrows()

```
2 references
private void ADDrows(ref SQLiteDataReader rdr, ref DataTable table)
{
    int y = 0;
    if (!File.Exists("BrokenRows " + amountofdataRows))
    {
        File.Create("BrokenRows " + amountofdataRows).Dispose();
    }
    using (StreamWriter sw = new StreamWriter("BrokenRows " + amountofdataRows))
    {
        sw.WriteLine("Broken rows - rowkeys");
        DataRow row;
        while (rdr.Read())
        {
            try
            {
                y++;
                // StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country,InvoiceNo,MoneySpent
                row = table.NewRow();
                row["PrimaryKey"] = y;
                row["StockCode"] = rdr.GetString(1);
                row["Description"] = rdr.GetString(2);
                row["Quantity"] = rdr.GetInt32(3);
                string dashdate = rdr.GetString(4);
                string formatted = GetFormat(dashdate);
                //Formats the 2-Aug-16 date into a pure numerical date
                row["InvoiceDate"] = formatted;
                row["UnitPrice"] = rdr.GetDouble(5);
                row["CustomerID"] = rdr.GetInt32(6);
                row["Country"] = rdr.GetString(7);
                row["InvoiceNo"] = rdr.GetInt32(0);
                row["MoneySpent"] = Math.Round((rdr.GetInt32(3) * rdr.GetDouble(5)), 2);
                table.Rows.Add(row);



            }
            catch (Exception)
            {
                sw.WriteLine(rdr.GetString(0));
            }

        }
    }
}
```

Use of try catch to catch any rows that break the row filling loop without causing these broken row to break the system. It catches these broken rows and writes their primary keys in a text file called broken rows + amountofrows for future use.

Remove()

```
1 reference
private void remove(DataTable table)
{
    List<int> rows = new List<int>();
    using (StreamReader reader = new StreamReader("BrokenRows " + amountofdataRows))
    //uses the + to make sure it uses the correct text file
    {
        reader.ReadLine();
        while (reader.ReadLine() != null)
        {
            rows.Add(Convert.ToInt32(reader.ReadLine()));
            // reads all lines from broken rows adding them to a list
        }
    }
    //Deletes the File afterwards to ensue ame primary keys are not read and deleted twice
    File.Delete("BrokenRows " + amountofdataRows);
    SQLInteractions Removal = new SQLInteractions();
    foreach (int item in rows)
    {
        string ComandText = "DELETE FROM CustomerData WHERE Primarykey = " + item + ";";
        Console.WriteLine(ComandText);
        Removal.ExicuteQuery(ComandText);
    }
    //This is removing all traces of broken rows that remain in the dataset that might break the system overall

    dataSet.Tables.Remove(table);
    //This Removes the table and re creates the table with the new cleaned dataset

    string command1 = "SELECT * FROM CustomerData LIMIT " + amountofdataRows;
    SQLiteConnection connection = CreateSQLconnection();
    using var cmd = new SQLiteCommand(command1, connection);//needs to be redone
    //write a cross table tabulation
    SQLiteDataReader rdr = cmd.ExecuteReader();

    DataTable ReMadeTable = new DataTable("MotherDF");

    ADDcolums(ref ReMadeTable);

    ADDrows(ref rdr, ref ReMadeTable);

    //Re creates datatable with the cleaned data

    dataSet.Tables.Add(ReMadeTable);

}
```

MMpertrans : Table class:

Line434

First datatable created from the main dataset. Groups the broad orders into individual transactions by tracking invoice numbers between rows. Useful in the creation of many future Datatables as well as being a useful table in itself.

```
MMpertrans
    Create(DataSet) : DataTable
    FILLrowsMP(ref DataTable, DataSet) : void
    ADDcolumstoMP(ref DataTable) : void
```

Override Create()

```
2 references
public override DataTable Create(DataSet DataSet)
{
    DataTable MPPerTrans = new DataTable("MPPerTrans");
    ADDcolumstoMP(ref MPPerTrans);

    FILLrowsMP(ref MPPerTrans, DataSet);
    Sheet = MPPerTrans;

    return MPPerTrans;
}
```

First instance of the sheet being assigned to old the Datatable to allow future table class methods to function.

ADDcolumstoMP()

```
dattypes[5] = "INTAGAR";
Tablenames[5] = "Year";

datatypes.addValues(dattypes);
tablenames.addValues(Tablenames);
```

Instead of using the tuple of string arrays this system uses L1st object to hold the data types needed for the SQL create table syntax. Within this method is the first example of the L1st object being assigned into.

FILLrowsMP()

DataTable Visualizer

InvoiceNo	MoneySpent	InvoiceDate	CustomerID	NumericDate	Year
536365	139	29-11-16	17850	-36	16
536366	22	29-11-16	17850	-36	16
536367	279	29-11-16	13047	-36	16
536368	70	29-11-16	13047	-36	16
536369	18	29-11-16	13047	-36	16
536370	856	29-11-16	12583	-36	16
536371	204	29-11-16	13748	-36	16
536372	22	29-11-16	17850	-36	16
536373	260	29-11-16	17850	-36	16
536374	350	29-11-16	15100	-36	16
536375	260	29-11-16	17850	-36	16
536376	329	29-11-16	15291	-36	16
536377	22	29-11-16	17850	-36	16
536378	445	29-11-16	14688	-36	16

Close

SpentOver : Table

Line328

Creates the Spent Overtime table that tally's all the money spent of each CustomerID within the Dataset. Useful in both ranking the value of each customer as well as evaluating the overall customer base. Also, this table's data is what is used to create the upper quartile cut of point is for the system that is used for the Top25% comparisons in the Dashboard.

```
SpentOver
    Create(DataSet) : DataTable
    IDSpentOvertime(ref DataTable, DataSet) : void
    TallyMP(ref Dictionary<int, double>, DataSet) : Dictionary<int, double>
```

Create()

```
2 references
public override DataTable Create(DataSet DataSet)
{
    DataTable Spent = new DataTable("Spent");
    IDSpentOvertime(ref Spent, DataSet);

    Sheet = Spent;

    return Spent;
}
```

IDSpentOvertime()

Calls the TallyMP Methood then constructs the datatable. After the table is constructed it fills it with the dictionary created through TallyMP

```
1 reference
private void IDSpentOvertime(ref DataTable Spent, DataSet DataSet)
{
    {
        Dictionary<int, double> MoneySpent = new Dictionary<int, double>();
        TallyMP(ref MoneySpent, DataSet);

        DataColumn column;

        string[] dattypes = new string[3];

        string[] Tablenames = new string[3];


        column = new DataColumn();
        column.DataType = System.Type.GetType("System.Int32");
        column.ColumnName = "PrimaryKey";
        column.ReadOnly = true;
        Spent.Columns.Add(column);
```

```
datatypes.addValues(dattypes);
tablenames.addValues(Tablenames);

DataColumn[] PrimaryKeyColumns = new DataColumn[1];
PrimaryKeyColumns[0] = Spent.Columns["PrimaryKey"];
Spent.PrimaryKey = PrimaryKeyColumns;
DataRow row;
foreach (KeyValuePair<int, double> kk in MoneySpent)
{
    Console.WriteLine(" yeeet  " + kk.Key + "    " + kk.Value);
    row = Spent.NewRow();
    row["PrimaryKey"] = i;
    row["CustomerID"] = kk.Key;
    row["MoneySpentOvertime"] = kk.Value;

    Spent.Rows.Add(row);
    i++;
}
```

TallyMP()

```
1 reference
private static Dictionary<int, double> TallyMP(ref Dictionary<int, double> MoneySpent, DataSet DataSet)
{
    //tally for money spent by each client in the same time
    DataTable table = DataSet.Tables["MotherDF"];

    for (int i = 1; i < table.Rows.Count; i++)///can be Changed into operations class in table
    {
        double counter = 0;
        if (MoneySpent.ContainsKey((int)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("CustomerID")]))
        {
        }
        else
        {
            DataRow[] ContainsInv = table.Select("CustomerID = '" + table.Rows[i].ItemArray[(int)table.Columns.IndexOf("CustomerID")] + "'");
            foreach (DataRow item in ContainsInv)
            {
                counter = counter + (double)item[table.Columns.IndexOf("MoneySpent")];

            }
            MoneySpent.Add((int)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("CustomerID")], counter);
        }
    }
    return MoneySpent;
}
```

Utilizes a similar system as to separate between invoice numbers as it does for CustomerIDs. Uses a Dictionary of int to double to link and stor each customerID to tallied money spent overall

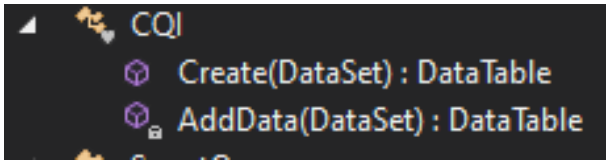
DataTable Visualizer

PrimaryKey	CustomerID	MoneySpentOvertime
1	17850	5391.210000000009
2	13047	366.6300000000001
3	12583	855.86
4	13748	204
5	15100	350.4
6	15291	328.79999999999995
7	14688	444.98
8	17809	1251.5
9	15311	1100.62
10	16098	430.59999999999997
11	18074	489.6
12	17420	130.85
13	16029	4271.52
14	16250	226.14000000000001

Close

CQI : Table

177



Creates the Customer Quality Index table which holds many useful quantities within it for each customer while creating its own custom quantity called CQI. The table holds data such as Frequency, MoneySpentOvertime, StandardDiviation of transaction values and CQI which is created by.

(MoneySpent x Frequency) / StandardDiviation

Create()

```
2 references
public override DataTable Create(DataSet df)
{
    DataTable CQItable = AddData(df);

    Sheet = CQItable;

    return CQItable;
}
```

AddData()

Ater constructing the data table it runs this for loop to fill it

```
DataRow row;

    for (int i = 0; i < Spent.Rows.Count; i++)
    {
        //runs through the Datatable looking for every index of the ID
        //loads all of those indexes and gets all the money per trans
        //gets SD of that then sends it to SD
        //solves problems of SD and ID not maching
        row = CQIData.NewRow();

        DataRow[] Rows = MPPerTrans.Select("CustomerID = '" + Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("CustomerID")] + "'");
        double[] DataforSD = new double[Rows.Length];
        int y = 0;
        foreach (DataRow item in Rows)
        {
            DataforSD[y] = (double)item.ItemArray[(int)MPPerTrans.Columns.IndexOf("MoneySpent")];
        //Fills a array with the Money Spent of each transaction the CustomerID is present in
        }

        double SD = 0;
        SD = this.GetSD(DataforSD);
        //Utilizes the GetSD method of the abstract Table class.

        row["PrimaryKey"] = i;
        row["CustomerID"] = (int)Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("CustomerID")];
        row["MoneySpentOvertime"] = (double)Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("MoneySpentOvertime")];
        row["StandardDiv"] = (double)SD;
        row["Frequency"] = Rows.Length;
        double h = 0;
        if (SD == 0) // Makes sure the system does not attempt to divide by 0
        {
            h = 0;
        }
        else//Money spent overtime * Frequence / SD
        {
            double j = (double)Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("MoneySpentOvertime")] * Rows.Length;
            h = j / SD;

        }
        row["CQI"] = h;

        CQIData.Rows.Add(row);
    }


return CQIData;
```

DataTable Visualizer

PrimaryKey	CustomerID	MoneySpentOvertime	StandardDiv	Frequency	CQI
0	17850	5391.210000000009	7.4341398955198015	34	24656.670788569245
1	13047	366.6300000000001	8.48528137423857	3	129.623279593212
2	12583	855.86	0	1	0
3	13748	204	0	1	0
4	15100	350.4	0	1	0
5	15291	328.79999999999995	0	1	0
6	14688	444.98	0	1	0
7	17809	1251.5	373.3523804664971	3	10.05618337107914
8	15311	1100.62	325.5	2	6.762642089093701
9	16098	430.59999999999997	0	1	0
10	18074	489.6	0	1	0
11	17420	130.85	0	1	0
12	16029	4271.52	268.229172330097	3	47.7746692825407
13	16250	226.14000000000001	0	1	0

Close

MoneyPerDay : Table

51

Creates the Per day Datatable breaking the string Date format of the central database and turning it into numerical values the system can use.

```
MoneyPerDay
    Create(DataSet) : DataTable
    CheckDate(ref DataTable, DataTable, ref int, int, ref int, ref int) : DataTable
    FindMMPD(DataSet) : DataTable
```

Create()

```
2 references
public override DataTable Create(DataSet DF)
{
    Sheet = FindMMPD(DF);
    return Sheet;
}
1 reference
```

FindMMPD()

Constructs table then runs this for loop to fill it

```
{
    int k = 0;
    for (int i = 0; i < MPPerTrans.Rows.Count; i++)
    {
        string date = "";
        date = (string)MPPerTrans.Rows[i].ItemArray[(int)MPPerTrans.Columns.IndexOf("InvoiceDate")];
        Tuple<int, int> DayandYear = GetDate(date);
        //Uses a Method from the abstract Table class
        int dayOyear = DayandYear.Item1;
        int year = DayandYear.Item2;
        CheckDate(ref MMPerday, MPPerTrans, ref dayOyear, i, ref k, ref year);
    }
}


return MMPerday;
```

CheckDate()

```
private DataTable CheckDate(ref DataTable MMPerday, DataTable MPPerTrans, ref int dayOyear, int i, ref int k, ref int year)
{
    if (year == 16)//As explained before to keep the day system consistent
                   //a constant - 365 is added is added if the year is 16
    {
        dayOyear = dayOyear - 365;
    }
    DataRow[] Rows = MMPerday.Select("Day = '" + dayOyear + "'");
    if (Rows.Length > 0)
        //If the day is already included in the Datatable Add onto the values of MoneySpent and Frequencyofsale
    {
        Rows[0]["MoneySpent"] = Math.Round((double)Rows[0]["MoneySpent"]
            +  (double)MPPerTrans.Rows[i].ItemArray[(int)MPPerTrans.Columns.IndexOf("MoneySpent")], 2);
        Rows[0]["Frequencyofsale"] = (int)Rows[0]["Frequencyofsale"] + 1;
    }
    else
    {//If the day is not part of the Datatable yet add it including all of its relevant values
        DataRow row;
        row = MMPerday.NewRow();
        row["PrimaryKey"] = k;
        k++;
        row["Day"] = dayOyear;
        row["MoneySpent"] = Math.Round((double)MPPerTrans.Rows[i].ItemArray[
                                        (int)MPPerTrans.Columns.IndexOf("MoneySpent")], 2);
        row["Frequencyofsale"] = 1;
        row["Year"] = year;
        MMPerday.Rows.Add(row);
    }
    return MMPerday;
}
```

DataTable Visualizer

PrimaryKey	Day	MoneySpent	Frequencyofsale	Year
0	-31	46355	121	16
1	-30	47311	137	16
2	-29	23919	57	16
3	-27	31730	87	16
4	-26	31163	94	16
5	-25	53797	71	16
6	-24	39211	111	16
7	-23	38216	97	16
8	-22	33653	72	16
9	-20	17305	43	16
10	-19	27645	64	16
11	-18	29294	81	16
12	-17	30295	69	16
13	-16	49021	113	16

Close

SQLInteractions()

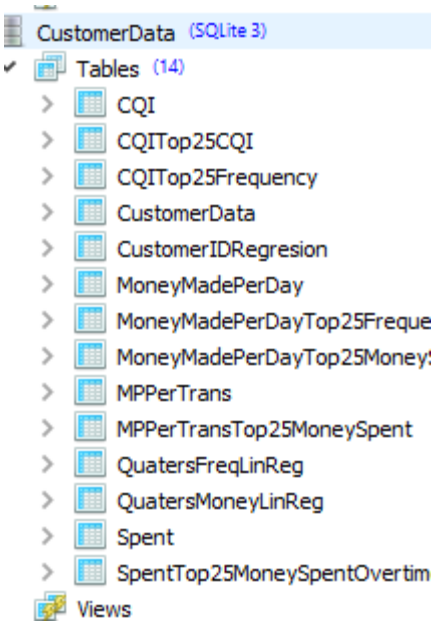
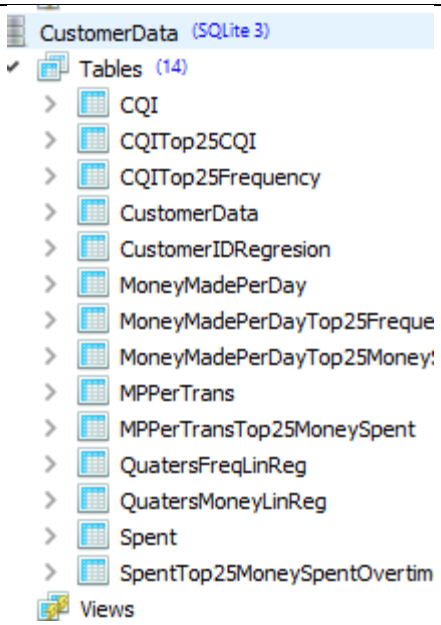
Line12

```
11 references
class SQLInteractions
{
    private SQLiteConnection conn;
    private string mydocumentsPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
    private string filepath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + "\\CustomerData.db";
    18 references
    public void ExicuteQuery(string CommandText)
    {
        SQLiteCommand sqlite_cmd;
        sqlite_cmd = conn.CreateCommand();
        sqlite_cmd.CommandText = CommandText;
        Console.WriteLine(CommandText);
        sqlite_cmd.ExecuteNonQuery();
    }

    5 references
    public SQLInteractions()//works as a methood used within the exicution
    {
        try
        {
            this.conn = new SQLiteConnection(@"Data Source =" + filepath);
            conn.Open();
            SQLiteCommand sqlite_cmd;
            sqlite_cmd = conn.CreateCommand();
            sqlite_cmd.CommandText = "Select * From CustomerData ";
            sqlite_cmd.ExecuteNonQuery();
        }
        catch (Exception)
        {
            Console.WriteLine("database file called CustomerData.db not found");

            System.Environment.Exit(1);
            throw;
        }
    }
    1 reference
    public void Close()
    {
        conn.Close();
    }
}
```

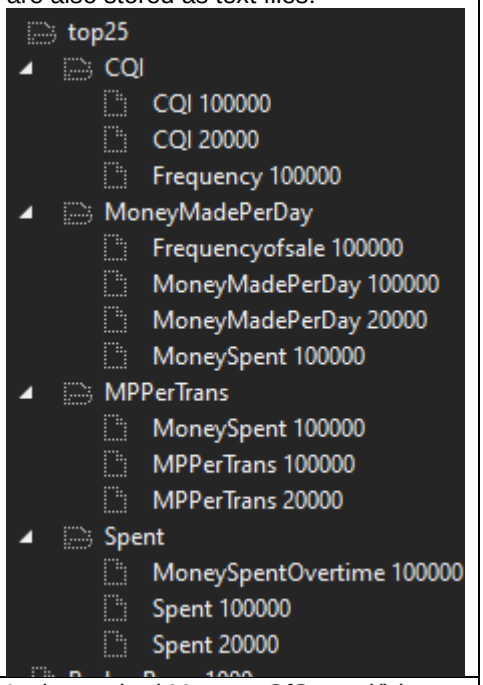
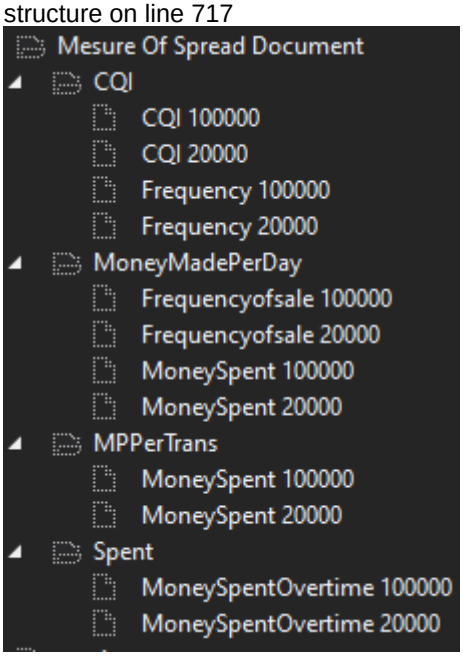
Upon construction the object tests for the presence of the file, this is required for this system as this is the first system to request data from the SQL database so must check for its presence to ensure for the rest of the programs function correctly. If the Database is not there the system ends the program and displays a error message.

Objectives Achieved or not achieved table:

Objectives	Achieved	Evidence
1.1. Request and receive data from a central SQL server using SQL commands	✓	Examples: Line 470, 563, 626 of the Regression program Page 183, 184, 185 where the Datatables are filled with tabular data requested from the SQL database
1.2. use an Object-oriented approach so that the individual parts of the data processing system can be re used in different systems that use a similar data set format	✓	Line 593 Page 202 The abstract table class in the Data processing system has many generalized methods that use the Sheet Datatable attribute to allow for any Subclass's Datatable to be compatible. This class can be pulled out and used in any system that requires operations for saving datatable to a SQL database
1.3. Create useful tables and store them in a medium so that this data can be re used in other systems by other departments, these tables include: 1.3.1. Money Spent by each customer in the time tracked 1.3.2. Individual transactions 1.3.3. Frequency of ordering of each customer 1.3.4. Average order value of each day 1.3.5. Money spent by all customers in each day	✓	Tables created in the Data processing program 1.3.1 Line 328 – class that creates it is SpentOver 1.3.2 Line 434- class that creates it is MMpertrans 1.3.3 Line 177-class that creates it is CQI() – the data is held in one of the columns 1.3.5 Line 51 – class that creates it is MoneyPerDay Above are all in Data processing system 1.3.4 Can be created from the data Created by the MoneyPerDay class but is also displayed on the dashboard if the button "Graph by AOV" is pressed These tables are stored within the central SQLite database so that other applications and departments can request data. The data is stored using a Save()

		method in a appropriate data creating class for example with the abstract Table class the method Save() on Line 798 
1.4. Clean the Dataset through SQL requests	✓	In the Data processing program through methods like PreliminaryCleaning() Line 912 and remove() Line 960 both of which remove broken rows from the dataset
1.5. Fill the database with the tables created by the system	✓	 The data is stored using a Save() method in a appropriate data creating class for example with the abstract Table class the method Save() on Line 798
2.1. Implement a linear regression system to	✓	The LinearRegression class on Line 673 of the Regression system

analyse data created in the data processing system		
2.1.1. Implement a linear regression method into the statistics system to find a line of best fit for data	✓	the SimLinRegression() method on line 1027 of the Regression system
2.1.2. Implement a polynomial regression methods system to create a curved line of best fit	✓	the PolyLinRegression() method on line 906 of the Regression system
2.1.3. Have these methods be efficient options compared to competing to not extend overall execution excessively	✓	Compared to machine learning options available the pure mathematics solutions in the SimLinRegression() and PolyLinRegression() is efficient.
2.1.4. Compare the linear and polynomial equations using an appropriate constant to avoid under fitting and over fitting only using polynomial when deemed appropriate	✓	The long list of logic within the LinearRegression's class constructor ensures the best fitting as well as logically accurate graph is chosen on line 703 of the Regression program
2.1.5. Use a technique to compare the accuracy of the linear regression equations in a way that means that the scales of different equations don't compromise the comparisons.	✓	The R^2 formula in the method GetRSq() allows for the easy comparison of how much a model fits the data from a 1-0 scale solving the problems of scale between data Line 858 of Regression system
2.2. Use the regression system on datasets: 2.2.1. Individual customer transaction history from money spent	✓	The CreateLinregTableCustomerID() method goes through each customerID and creates a Linear regression equation for it using the LinearRegression class Line 353 of Regression system
2.2. Use the regression system on datasets: 2.2.2. Days in a month, week and quatre	✗	The system designed and programmed works as the CreateLinregTableDays() method takes in per day data breaks it into quarters and creates equations for it. Line276 However, with the present dataset when passes through this system doesn't create a equation within reasonable tolerances and testing of lowering them only produces equations so un fitting no useful predictions can be made
2.3. Create measures of spreads and appropriate comparison values for segmentation for each table when appropriate	✓	In the abstract Table class the method getUQID() on line 626 reads the upper quartile of a metric and masks out all values that don't exceed the upper quartile. Does this for several

2.3.1. Use upper quartile to get the top 25% of data		metrics on several tables. Uploads these lists of Top IDs to the SQL database. Screenshot from bin file of the data processing system where the top IDs are also stored as text files: 
2.3.2. get values for General measures of spread to be used 2.3.2.1. Standard deviation 2.3.2.2. Mean 2.3.2.3. Quartiles 2.3.2.4. IQR 2.3.2.5. Mode	✓	In the method MesuresOfSpread() it creates the quantities on line 750 The method CreateMesureOfSpreadDocument() saves these quantities to a folder structure on line 717 
2.3.3. Create an equation to create a value for how much a customer is a consistently high spending	✗	Within the system there is a measure called CQI which is calculated by (Frequency x MoneySpent)/ Standard deviation.

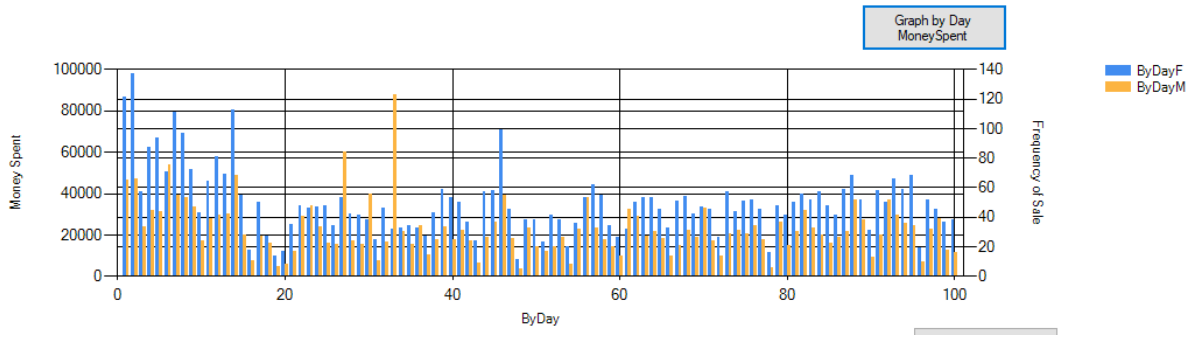
and consistently high frequency at spending		This was meant to allow for tracking of consistently high spenders whos class is found in line 177 of the data processing system. but what this value did was select for consistent spenders with lower Standard deviations leading to incredibly large CQI's in some cases with decimal Standard deviations making the quantity not useful.
3. Segment customer base and Products into useful groupings 3.1. Using the comparison Values created in the statistical analysis program segment the data by the value top 25% 3.2. Compare the summed overall upper quartile to the overall original data to produce a value for how centralized this data is	✓	Segments the data into top and bottom with the use of the Lists of ID's created in the getUQID() This takes place in the dashboard system twice. First when the Show Comparison button is pressed where the summed top25 Is compared to the bottom 75 on Line 986 Second is when show overall is pressed on Line 1301 and populates a chart and then Show Top25 on line 1340 is pressed which populates the same chart allowing the two populations to be compared Both of these are examples of when the two populations are segmented so that their data can be compared
4.1. Use the Linear regression equitation's for the sum of Month-by-month spending and test using a constant if a valid and useful prediction can be made about the future year	✗	Issue of Dataset not allowing for useful graphs to be created explain on 2.2.2
4.2. Predict future trends of individual customers based on their regression equation	✓	When the search button is pressed on line 1130 if applicable a graph is displayed and the differeciate of that graph is used to give a prediction of future growth or decline
4.3. Use the Linear regression equations of each customer to if the global transaction value is increasing or decreasing	✓	When the show Overall or Show top25 button is pressed on line 1301 Using the gradient of each customers equation tallying up signs and displaying the tally on a pie chart
5.1. Have the dashboards data be requested from a SQL server so that this data can be added to in a flexible way	✓	Examples within the class DataFromSQLTables are on : Line181 Line274 Line332

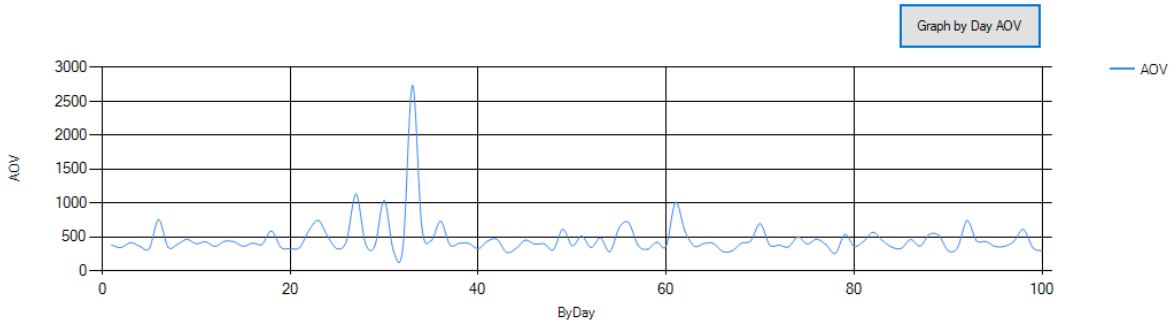
5.2. Present the data created by the data processing system so that individual departments also have access to them through requesting this data in a button	✓	The Dataviewer tab of the GUI allows for the picking to display the tables made in the data processing system on: Line 1002 – 1026
5.3. Present the data on Day-to-day purchases including money spent overall and number of purchases using a line graph	✓	The Graphics tab of the GUI presents this data. The chart is filled when “graph by MoneySpent” is pressed on Line 959
5.4. Present pie charts on how much the top 25% of different metrics contribute to the overall sum of these metrics	✓	The Graphics tab of the GUI presents this data. The chart is filled when the “Show Comparason” button is pressed on Line 986
5.5. All tabular data is to be presented using a scroll wheel to traverse down the large dashboard	✓	The Dataviewgrids that hold the tabular data have a automatic scroll wheel function
5.6. Have the option for the user to request specific customers transaction histories including their Linear regression equation through a drop-down menu	✓	The Dataviewer tab has this functionality by fulling a text box and pressing searched which fills the graph with the transaction history and if applicable their equation. The search box code can be found on line 1130

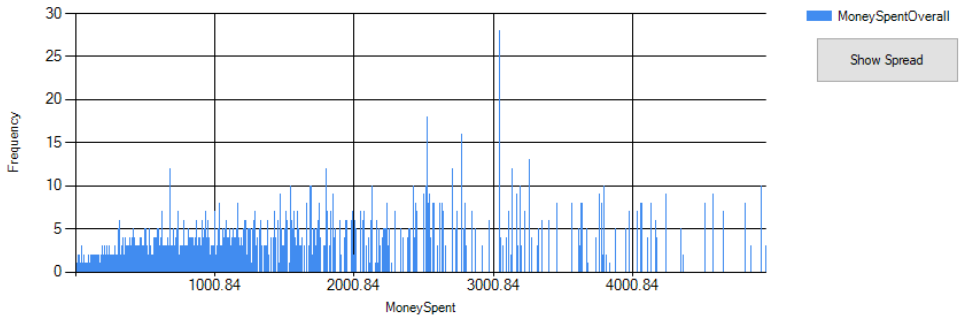
Testing

Testing GUI interface

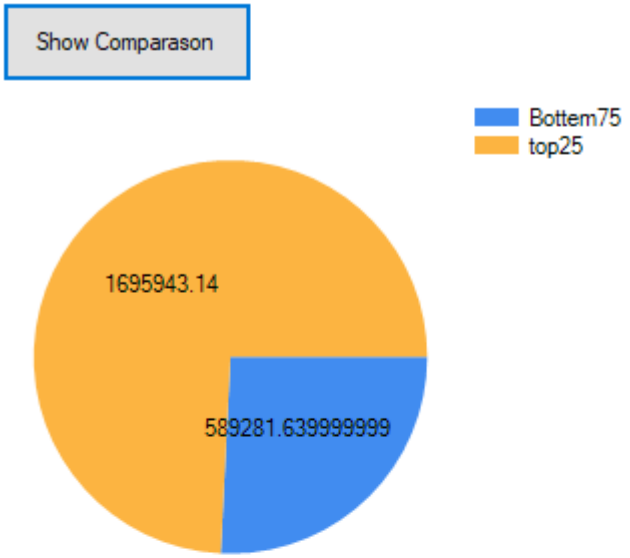
Objective
5.3. Present the data on Day-to-day purchases including money spent overall and number of purchases using a line graph
Pass

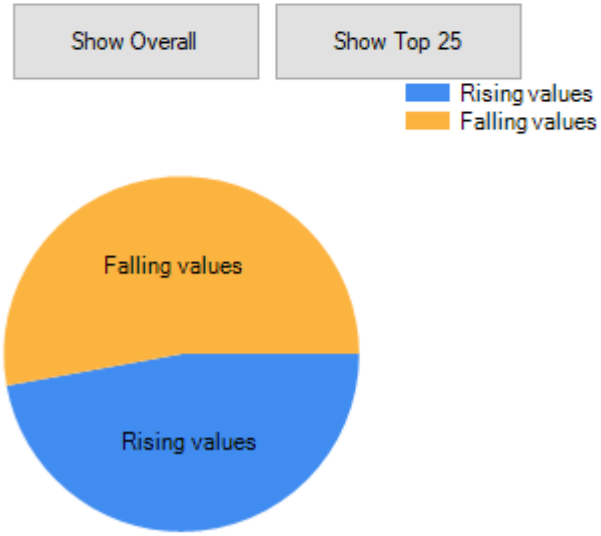






Objective
5.3. Present pie charts on how much the top 25% of different metrics contribute to the overall sum of these metrics
Pass


Show Comparason
Bottem75
top25
1695943.14
589281.639999999


Show Overall
Show Top 25
Rising values
Falling values
Falling values
Rising values

Objective
5.1 Have the dashboards data be requested from a SQL server so that this data can be added to in a flexible way. – Testing the SQL table against what is presented on the dataviewgrid
Pass

CustomerData Table name: CQI

	Name	Data type	Primary Key	Foreign Key	Unique	Check
1	PrimaryKey	VARCHAR				
2	CustomerID	VARCHAR				
3	MoneySpentOvertime	DECIMAL				
4	StandardDiv	DECIMAL				
5	Frequency	INTAGAR				
6	CQI	DECIMAL				

Form1

Graphics Dataviewer

CQI Table

MainDataset Table

PerDayTable

Transaction Table

Search For Custome

15291 Search

CustomerID	MoneySpentOvertir	StandardDiv	Frequency	CQI
17850	5391.210000000...	7.434139895519...	34	24656.67078856...
13047	1252.679999999...	170.8	5	36.67096018735...
12583	2620.429999999...	308.7380564491...	4	33.95020400319...
13748	433.4	114.5	2	7.570305676855...
15100	876	82.49579113843...	3	31.85617064499...
15291	2679.59	95.22371319978...	9	253.2595000722...
14688	2241.169999999...	124.0195927061...	8	144.5687702142...
17809	1861.7	264.1377481542...	4	28.19286547279...
15311	21789.54999999...	118.9462383283...	34	6228.399572881...
16098	852.4	211	2	8.079620853080...
18074	489.6	0	1	0
17420	130.85	0	1	0
16029	22939.34999999...	31.30495168499...	21	15388.18378789...
16250	389.4400000000...	81.5	2	9.556809815950...
12431	2211.280000000...	165.2000000000...	5	66.92736077481...
17511	25595.43999999...	81.08157757605...	9	2841.076442844...
13705	318.1400000000...	0	1	0
13747	79.6	0	1	0
13408	7386.989999999...	59.49159604515...	14	1738.360825308...

CustomerData Table name: CustomerData

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL
1	InvoiceNo	INTEGER					
2	StockCode	TEXT					
3	Description	TEXT					
4	Quantity	INTEGER					
5	InvoiceDate	TEXT					
6	UnitPrice	REAL					
7	CustomerID	INTEGER					
8	Country	TEXT					
9	PrimaryKey	INTEGER					

CQI Table

MainDataset Table

PerDayTable

Transaction Table

Search For Custome

15291

Search

PrimaryKey	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceN
1	85123A	WHITE HANGIN...	6	29-11-16	2.55	17850	United Kingdom	536365
2	71053	WHITE METAL ...	6	29-11-16	3.39	17850	United Kingdom	536365
3	84406B	CREAM CUPID ...	8	29-11-16	2.75	17850	United Kingdom	536365
4	84029G	KNITTED UNIO...	6	29-11-16	3.39	17850	United Kingdom	536365
5	84029E	RED WOOLLY H...	6	29-11-16	3.39	17850	United Kingdom	536365
6	22752	SET 7 BABUSH...	2	29-11-16	7.65	17850	United Kingdom	536365
7	21730	GLASS STAR F...	6	29-11-16	4.25	17850	United Kingdom	536365
8	22633	HAND WARME...	6	29-11-16	1.85	17850	United Kingdom	536366
9	22632	HAND WARME...	6	29-11-16	1.85	17850	United Kingdom	536366
10	84879	ASSORTED COL...	32	29-11-16	1.69	13047	United Kingdom	536367
11	22745	POPPY'S PLAYH...	6	29-11-16	2.1	13047	United Kingdom	536367
12	22748	POPPY'S PLAYH...	6	29-11-16	2.1	13047	United Kingdom	536367
13	22749	FELTCRAFT PRI...	8	29-11-16	3.75	13047	United Kingdom	536367
14	22310	IVORY KNITTED...	6	29-11-16	1.65	13047	United Kingdom	536367
15	84969	BOX OF 6 ASSO...	6	29-11-16	4.25	13047	United Kingdom	536367
16	22623	BOX OF VINTAG...	3	29-11-16	4.95	13047	United Kingdom	536367
17	22622	BOX OF VINTAG...	2	29-11-16	9.95	13047	United Kingdom	536367
18	21754	HOME BUILDIN...	3	29-11-16	5.95	13047	United Kingdom	536367

Pass

CustomerData Table name: MoneyMadePerDay

	Name	Data type	Primary Key	Foreign Key	Unique
1	PrimaryKey	VARCHAR			
2	Day	INTAGAR			
3	MoneySpent	DECIMAL			
4	Frequencyofsale	INTAGAR			
5	Year	INTAGAR			

CQI Table

MainDataset Table

PerDayTable

Transaction Table

Search For Custome

15291

Search

Day	MoneySpent	Frequencyofsale	Year
-31	46375	121	16
-30	47311	137	16
-29	23921	57	16
-27	31772	87	16
-26	31219	94	16
-25	53797	71	16
-24	39251	111	16
-23	38231	98	16
-22	33653	72	16
-20	17305	43	16
-19	27646	64	16
-18	29321	81	16
-17	30311	70	16
-16	49027	113	16
-15	20046	55	16
-13	7420	18	16
-12	19625	50	16
-11	15950	27	16
-10	4888	14	16

Pass

CustomerData Table name: MPPerTrans

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL
1	InvoiceNo	VARCHAR					
2	MoneySpent	DOUBLE					
3	InvoiceDate	VARCHAR					
4	CustomerID	VARCHAR					
5	NumericDate	INTAGAR					
6	Year	INTAGAR					

CQI Table

MainDataset Table

PerDayTable

Transaction Table

Search For Custome

15291

Search

InvoiceNo	MoneySpent	InvoiceDate	CustomerID	NumericDate	Year
536365	139	29-11-16	17850	-31	16
536366	22	29-11-16	17850	-31	16
536367	279	29-11-16	13047	-31	16
536368	70	29-11-16	13047	-31	16
536369	18	29-11-16	13047	-31	16
536370	856	29-11-16	12583	-31	16
536371	204	29-11-16	13748	-31	16
536372	22	29-11-16	17850	-31	16
536373	260	29-11-16	17850	-31	16
536374	350	29-11-16	15100	-31	16
536375	260	29-11-16	17850	-31	16
536376	329	29-11-16	15291	-31	16
536377	22	29-11-16	17850	-31	16
536378	445	29-11-16	14688	-31	16
536380	35	29-11-16	17809	-31	16
536381	450	29-11-16	15311	-31	16
536382	431	29-11-16	16098	-31	16
536384	490	29-11-16	18074	-31	16
536385	131	29-11-16	17420	-31	16

Pass

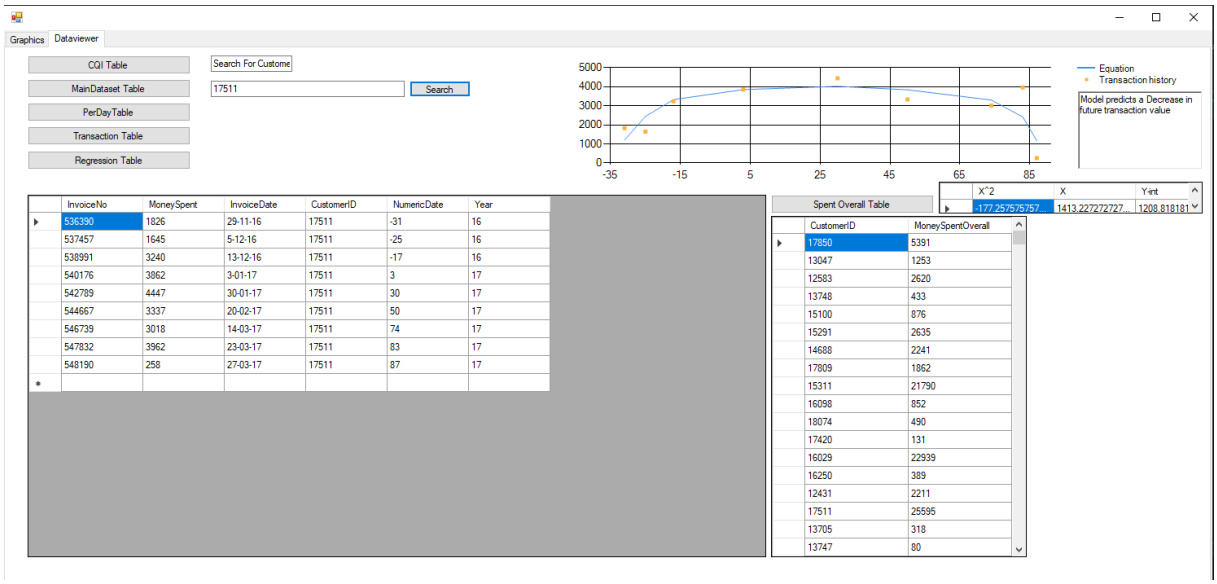
CustomerData Table name: Spent

	Name	Data type	Primary Key
1	PrimaryKey	INTAGAR	
2	CustomerID	VARCHAR	
3	MoneySpentOvertime	DOUBLE	

Spent Overall Table

CustomerID	MoneySpentOverall
17850	5391
13047	1253
12583	2620
13748	433
15100	876
15291	2680
14688	2241
17809	1862
15311	21790
16098	852
18074	490
17420	131
16029	22939
16250	389
12431	2211
17511	25595
13705	318
13747	80

Objective
Have the option for the user to request specific customers transaction histories including their Linear regression equation through a drop-down menu



Testing linear regression system:

For the testing of the Linear regression system, I will be using white box testing to fully test each mathematical step in the system to make sure that no syntax or order of operations was broken during the model's computerization. The Model, explained in detail in the design documentation, has several key modules which I will individually test using a dummy test dataset with known expected values to each step fixing any broken logic along the way.

After each module's run through is complete I will follow it up with black box testing on said module. I will be first running through the polynomial Regression system as it is the more complex and more prone to breaking.

Making The Equation

Objective	Test #	Input	Expected output	Actual output/ evidence	Pass/Fail	Comments
General Testing linear regression system:	1	(-3,0.9) (-2,0.8) -(1,0.4) (-0.2,0.2) (1,0.1) (3,0)	∑ y$\sum y$ = 6 ∑ xy$\sum xy$ = -2.2 ∑ x2y$\sum x2y$			
	2	(2,27) (6,25) (10,23) (14,21) (16,20)	Slope = -½ Y intercept = 28			

Test data:

x	y

-3	0.9
-2	0.8
-1	0.4
-0.2	0.2
1	0.1
3	0

The program uses a for loop to collect and create the quantities needed for the linear regression equation. Here is a math-based trace table of that process.

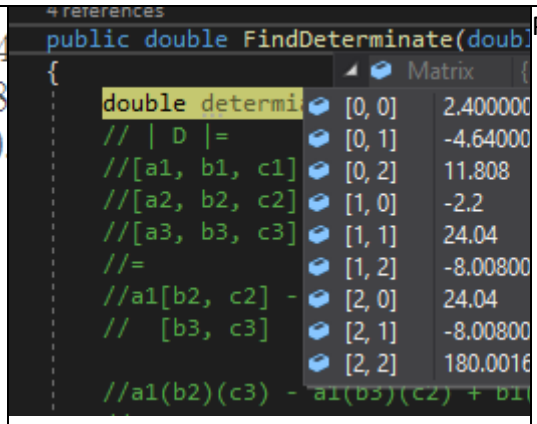
Required quantities	Expected values	Actual output	Pass Fail
∑ y$\sum y$	0.9x 0.8x 0.4x 0.2x 0.1x 0 = 2.4	6	Pass
∑ xy$\sum xy$	(0.9x- 3) + (0.8x-2) +(0.4x-1) +(0.2x - 0.2)+(0.1x1) + (0x3) = -4.64	0.75	Fail – miss inputting of the original data with 3 instead of –3 being inputted
			Pass
∑ x^2y$\sum x2y$	(0.9x- 3^2) + (0.8x-2^2) +(0.4x-1^2) +(0.2x - 0.2^2)+(0.1x1^2) + (0x3^2) = 11.808	11.808	Pass
∑ x$\sum x$	-3+-2+-1+-0.2+1+3= -2.2		
∑ $x^2$$\sum x2$	(-3^2) + (-2^2) +(-1^2) +(-0.2^2)+(1^2) + (3^2) = 24.04	24.04	
∑ $x^3$$\sum x3$	(-3^3) + (-2^3) +(-1^3) +(-0.2^3)+(1^3) + (3^3) =-8.008	-8.008	Pass
∑ $x^4$$\sum x4$	(3^4) + (-2^4) +(-1^4) +(-0.2^4)+(1^4) + (3^4) = 180.0016	180.0016	Pass

Now the determinates of the Matrices must be calculated. I will be using the same base data to keep the run through more real following the process discribed in the design document and the equation bellow:
The determinate can be calculated on 3x3 matrices through this method:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = M \quad |M| = a\begin{pmatrix} e & f \\ h & i \end{pmatrix} - b\begin{pmatrix} d & f \\ g & i \end{pmatrix} + c\begin{pmatrix} d & e \\ g & h \end{pmatrix}$$

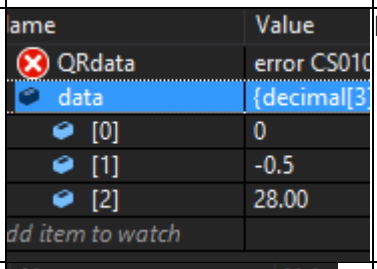
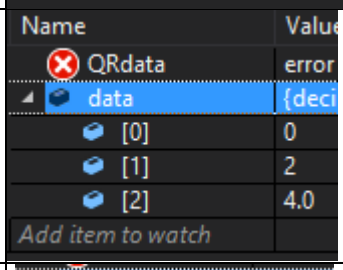
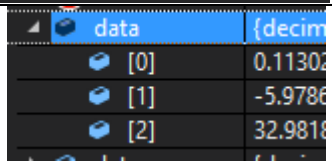
Finding the determinate of M0:

Step:	Expected quantiy:	Actial output	Pass/Fail

Loading in matrix	$\begin{pmatrix} 2.4 & -2.2 & 24 \\ -4.64 & 24.04 & -8 \\ 11.808 & -8.008 & 180 \end{pmatrix}$		Pass
A= a(e)(i) – a(h)(f)	10231.47	10231.4696	Pass
B= b(f)(g) – b(d)(i)	-1629.43	-1629.4277	Pass
C=c(d)(h) – c(g)(e)	-5930.84	-5930.8411	Pass
Det of M0	2671.2	2671.196	Pass

The determinate program has now been proven to work now for the data final Polynomial linear regression equation:

Coefficient of:	Expected quantity	Actual output	Pass/Fail
X^2	0.0278	0.229065	Fail – Misassignment of variable
		0.02776	Pass
X	-0.1628	-0.16280	Pass
Y-Intercept	0.2291	0.02776	Fail – Misassignment of variable
		0.229065	Pass

Objective	Test #	Input	Expected output	Actual output/ evidence	Pass/Fail	Comments
2.1.1	1	(2,27) (6,25) (10,23) (14,21) (16,20)	Slope = -½ Y intercept = 28	Slope = 5.7 Y intercept = 0	Fail	Due to the double datatype Simple Linear Regression was broken if the slopes sign was negative
					Pass	
	1	(1,6) (3,10) (4,12) (7,18) (12,28)	Slope = 2 Y intercept = 4		Pass	
	2	(1,25.8) (2, 21.2) (3, 16.3) (4, 10.8)	Slopesq =-1/10 Slope =-6 Y intercept=33		Pass	

Testing R^2

The next thing to run through is the R^2 formula and within this the formula for finding least squares. R^2 follows this basic equation:

$$R^2 = 1 - \frac{Error\ to\ equation}{Error\ to\ mean}|$$

I will be using a different dataset for this Next section using its accoupling Simple Linear Regression system:

X	Y
2	2
3	4
4	6
6	7

With the accompanying Linear Regression equation:
y=0.143+1.229xy^=0.143+1.229x

Step:	Expected quantiy:	Actial output	Pass/Fail

First value created by the equation	2.601	2.5999999999999996	Pass
First Actual Y – First Y value created then squared	0.36	0.35999	Pass
Final Error To Equation	1.542871	0.3599999999999996	Fail – Problem arose from using the double datatype as division wouldn't work as needed
		1.542857142	Pass – replacing the double datatype with decimal lead to the correct value
Final Error To Mean	14.75	7.5625	Fail – failed in the same session as the Error to equation
		14.7500	Pass – Decimal Datatype as well as the key word m when dealing with any numerical values that cause problems. fixed issue
R^2	0.895	0.89539	Pass

2.1.5 Use a technique to compare the accuracy of the linear regression equations in a way that means that the scales of different equations don't compromise the comparisons.	1	(210,2400) (260,3647) (300,4804) (345,6400) Equation 1: a= 1/19 b=1/3 c=0 (46,53) (63,30) (67,21) Equation 2: A= -1.4732 B= 121.0938	Create a R^2 that is accurate and can be compared to R^2 of different scales Equation 1: R^2 = 0.99 Equation 2: R^2 = 0.67	Microsoft Vis Hello World! equation: 0.05263157894 0.33333333333 0 0.99981770381 Microsoft Vi Hello World! equation: 0 -1.4732 121.0938 0.9908422213		Pass	Even though the Scales are very different the R^2's are almost equal and can be easily compared

	2	(210,700) (260,400) (300,500) (345,200) Equation 1: a=-0.0006 b=-2.87 c=1297 (231,2943) (298,3830) (356,4590) (450,5829) (649,6792) Equation 2: a=9.2 b=1146	Equation 1: R^2= 0.9429 Equation 2: R^2 0.7818 Making it so the better fitting one can be chosen regardless of the scale of each	Microsoft Visua Hello World! equation: 0 9.2 1146 0.942933065525 Microsoft Visual Stu Hello World! equation: -0.0006 -2.87 1297 0.781794135192307	Pas	

Testing decision logic

With all appropriate values created we can now test the logic deciding which of the two equations the system picks. The sequence of logic can be found in the design document section. I used console WriteLines to print the equation chosen at the end of each process

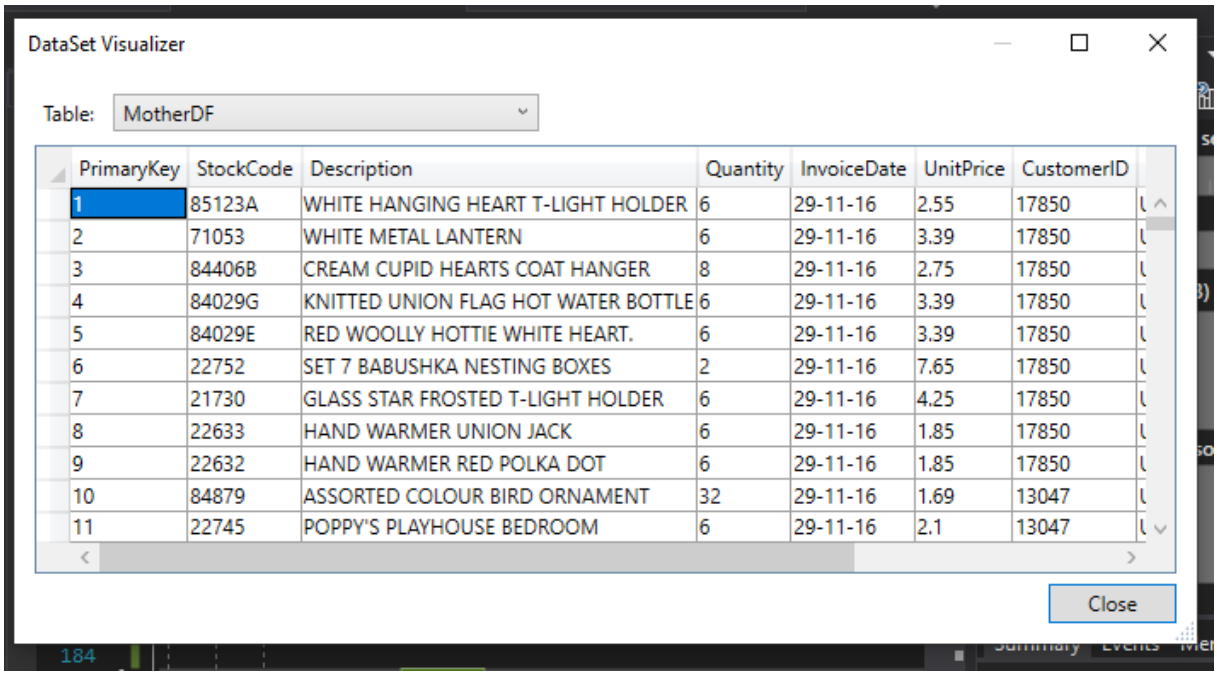
Objective	Test #	Input	Expected output	Actual output/ evidence	Pass/Fail	Comments
2.1.4 Compare the linear and polynomial equations using an appropriate constant to avoid under fitting and over fitting only using polynomial when deemed appropriate	1	(2,2) (3,4) (4.6) (6,7) Equation 1: a=1.229x b=0.143 Equation 2: a=-0.34 b=4 C=-4.7	Equation 1: r^2= 0.895 2: r^2= 0.9945 Choosing equation 1	0.143 1.229 0 0.8953985762711864 0	Pass	Equation 1 has a positive sign and because equation 2 has a negative overall sign it is a bad predictor

	2	(6,2) (4,4) (3.6) (2,7) Equation 1: a=-1.286b= 9.57 Equation 2: a=0.068 b=-1.84 C=10.5	Equation 1: R^2= 0.9806 2: R^2 = 0.9846 Choosing equation 1	9.5714285714285 -1.285714285714 0 0.9806295399515 -0.990267408305	Pass	Equation 1 has a negative sign and because equation 2 has a positive overall sign it is a bad predictor
	3	(5,5)(7,11) (8,15) (15,55) (17,73) Equation 1: a=5.7 b=-27.53 Equation 2: a=¼ b=0.125 c= -2	Equation 1: R^2 = 0.9865 2: R^2 = 0.9954 Taking equation 2	0.25 0.125 -2 0.9995400403	Pass	as it shares a sign with equation 1 and has a higher R^2 equation 2 is chossen
Range and barrier testing	1	Inputted 2 of the same data points	Handles and sets the regression as 0		Fail	Breaks as it attempts to divide by zero as the check valid was miss applied to the X axis twice
				0 0 0 0 0	Pass	
	2	Inputted random large data points to see if the low R^2 will result in null equations	Saves 0 to all of the equation values	123224.714285714 -18665.142857142 0 0.71842749029277 -0.8476010207006	Fail	The program infact made a Linear equation with a R^2 of 0.7
		Increased randomness to the dataset: (2, 100000) (5,900) (6, 100000000000) (7,2)	Saves 0 to all of the equation values	0 0 0 0 0		
	3	Inputted two data points	Sets all to 0 as this customer isnt worth creating a graph for			

			as they aren't a large repeat customer			

Testing Dataprocessing system:

Objective	Test #	Input	Expected output
1.1 Request and receive data from a central SQL server using SQL commands	1	The request of Select * From Customerdata Limit Sent and exicuted by SQL interactions	SQLReader now holding the values of 85123A,WHITE HANGING HEART T-LIGHT HOLDER,6,29-11-16,2.5500,17850,United Kingdom,536365,15.3000 As 1 71053,WHITE METAL LANTERN,6,29-11-16,3.3900,17850,United Kingdom,536365,20.3400 As 2 84406B,CREAM CUPID HEARTS COAT HANGER,8,29-11-16,2.7500,17850,United Kingdom,536365,22.0000 As 3



PrimaryKey	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
1	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	29-11-16	2.55	17850
2	71053	WHITE METAL LANTERN	6	29-11-16	3.39	17850
3	84406B	CREAM CUPID HEARTS COAT HANGER	8	29-11-16	2.75	17850
4	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	29-11-16	3.39	17850
5	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	29-11-16	3.39	17850
6	22752	SET 7 BABUSHKA NESTING BOXES	2	29-11-16	7.65	17850
7	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	29-11-16	4.25	17850
8	22633	HAND WARMER UNION JACK	6	29-11-16	1.85	17850
9	22632	HAND WARMER RED POLKA DOT	6	29-11-16	1.85	17850
10	84879	ASSORTED COLOUR BIRD ORNAMENT	32	29-11-16	1.69	13047
11	22745	POPPY'S PLAYHOUSE BEDROOM	6	29-11-16	2.1	13047

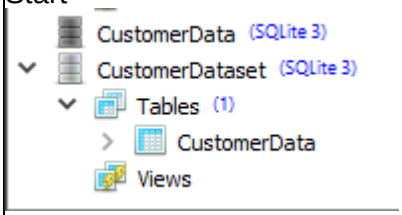
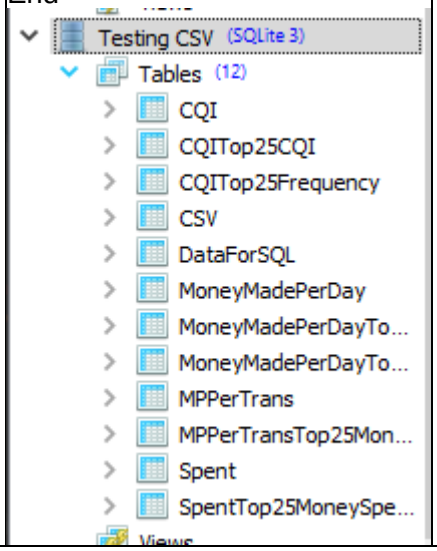
Objective	Test #	Input	Expected output	Actual output/ evidence	Pass/Fail	Comments
1.5.1 Money Spent by each customer in the time tracked	1	The first 20,000 rows of the dataset	The Spent overtime table's first 2 being	CustomerID: 17850, MoneySpentOvertime: 5391.210000000009 CustomerID: 13047, MoneySpentOvertime: 366.6300000000001	Pass	

			With the values of 17850 has spent 5391 13047 has spent 366			

Objective	Test #	Input	Expected output	Actual output/ evidence	Pass/Fail	Comments
1.5.2 Individual transactions	1	The first 20,000 rows of the dataset	The first 5 rows of per transaction looking like 139.120029-11-16,17850 22.2000,29-11-16,17850 278.7300,29-11-16,13047 70.0500,29-11-16,13047 17.8500,29-11-16,13047	MoneySpe InvoiceDat CustomerII 139 29-11-16 17850 22 29-11-16 17850 279 29-11-16 13047 70 29-11-16 13047 18 29-11-16 13047	Pass	
1.5.3 Frequency of ordering for each	1	The first 20,000 rows of the dataset	The CQI table holding a Frequency column. The first 5 customerIDs and frequency are as follows 17850 - 34 12583 -3 12583 - 1 13748 - 1	0 17850 34 1 13047 3 2 12583 1 3 13748 1	Pass	

Objective	Test #	Input	Expected output	Actual output/ evidence	Pass/Fail	Comments

1.5.4 Average order value of each day	1	The first 20,000 rows of the dataset	The Per Day table holding these first 3 values in the AOV Column 329 - 121 330 - 137 331 - 57	-31 121 16 -30 137 16 -29 57 16	Pass	Had to implement a negative system for data values within the first year to input into the linear regression as well as other graphing systems
1.5.5 Money spent by all customers in each day	1	The first 20,000 rows of the dataset	The Per Day table holding these first 3 values in MoneySpent column 329 - 46376 330 - 47316 331 - 23921	-31 46375 -30 47311 -29 23921	Pass	
1..6 Clean the Dataset through SQL requests	1	First 100,000 rows	No errors during run time	Input string not in correct format when handling inputting of data	Fail	Loading broken rows into a txt to inspect them and they had empty row values. I removed all empty rows from the SQL database
				Error dividing by zero	Fail	Some quantities were minus which would break the average order value calculation. Removing all negative quantities from the database removed this
		Full dataset		Broken stock codes causing issues when re loading	Fail	Stockcodes exist with value of ‘M’ have broken row values and seem to be a error in data input. Remove all stick codes of value ‘M’
				Full running without breaking even without validation involved proving the dataset has been cleaned	Pass	

1.7 Fill the database with the tables created by the system	1	First 100,000	A full spread of tables created in the SQL database	Start  End  	Pass	

Objective	Test #	Input	Expected output	Pass/Fail
3.1 Using the comparison Values created in the statistical analysis program segment the data by the value top 25%	1	Test data txt with the value of 5 as UQ Data: 4-a 3-b 4-c 6-d 7-e 8-f 9-g 1-h 3-i 4-j 6-k	Masking out the other values with a list of Top 25% containing: d,e,f,g and k	Pass

Top25	Count = 5	Syste
[0]	100 'd'	char
[1]	101 'e'	char
[2]	102 'f'	char
[3]	103 'g'	char
[4]	107 'k'	char

Evaluation

The combination of systems successfully takes the large central database, abstracts it and creates quantities to analyse the data and then presents the data in a GUI fixing and completing the original problem. The system has been broken into 3 programs for modulation and so that the faster user program of the dashboard can be run without the slow data processing program and the intensive statistical programs regression.

The 1st general objective of “Create a system that breaks down the large almost half a million row long raw starting dataset into useful composite tables” has been met very well with all of its specific objectives being completed fully in the system. This objective was mainly pivoted for the Data processing program so this objective encompasses almost all of its functions. The specific objectives refer mainly to the creation of tables but also to the heavy use of OOP which has been met as the data processing system utilizes an abstract Table class that holds most general methods needed for table functions and used through the system which are inherited by the other table classes. Another specific objective is the use of the Database to hold the data created by the program, this is also met through the use of a dedicated SQL class and single SaveData() method within the table class that all inheriting classes use.

The 2nd general objective is “Use statistical analysis techniques on this data to create new data and create comparison quantities for segmentation” which has some overlap between the data processing and Statistical regression systems. The Linear regression requirements are fully met by the LinearRegression class which includes methods for linear and polynomial regression, as well as comparison functions that meet the specific objectives of making scale a non-issue in comparisons. The regression of CustomerIDs was a very successful aspect of the final product allowing for the prediction of changes in customer transactions and global transaction value. One thing not met is the regression between quarters. The methods exists and remains in the system storing the equations to the database however the present dataset and current implementation of the methods do not lead to useful graphs so changes in this aspect are required. The two regression methods both meet the efficiency objective working on one run through of the data then matrix operations which although intensive are less than loading a Machine Learning model for each customer in the system. The logic that control if and what equations are stored ensures that the equations stores are functional and useful to the systems future purposes avoiding logical errors that would damage the model overall.

The creation of the measures of spread are met in the Data processing system and saved to the central database meeting this specific objective. The segmentation of data also takes place within the data processing system because it uses the measures of spread values to get the top 25% of CustomerIDs of different metrics. The objective of the value selecting for high valued consistency was not met fully as the equation chosen showed to have serious flaws when the data has as much variety as the dataset.

The 3rd general objective was met but could be expanded upon. The segmentation and comparison was completed in the dashboard program but the impact is just presenting onto a graph creating no new specific values or conclusions. The comparison does show how

heavily centralized the data is to the Top but expanding on this idea into seeing segments within this top 25 data could of yielded some more useful and actionable data.

The 4th general objective focused on predicting future trends which was met very successfully within the system through the Regression system aswell as the dashboard program. Again, the problem of the by day values causing issues persists but the methods and models were built however the data itself makes their operation unsuccessful. The predicting of rising or falling of transaction value of both the overall customer base and the Specific CustomerIDs was very successful but the graphs themselves could have been better presented within this system. Showing all of the equations within a graph and also the calculated average graph would much better illustrate the variation of growth and decline throughout the population, this would be almost a equation scatter diagram.

The 5th general objective related to the displaying of the tables created which was met successfully. One issue however is the lack of labelling as an actual new user could not operate the graphs and the dataviewer without instruction. One way to fix this would be to have better labels and grouping the graphs into groups so they can be operated, understood and digested one at a time easier.

My third party deemed my overall system successful in the objectives set out. The regression equation allowed for the easy prediction of future data and the graphs within the dashboard program illustrated the properties of the dataset well. Moreover, the datagridview was a acceptable user interface to read and request specific data from the SQL database. The goal of representing the dataset and allowing for data created by the data processing and regression systems to be read and accessed by other departments was fully met. However, they did point out some problems with the solution. They suggested the increased labelling and modulization of the graphics as well as the equation scatter diagram but also the idea of analysing products in much the same way I analysed specific customers. The analysing of products to the company is a much more actionable set of data as they can directly change these aspects based on what the data points to. Another suggestion was the actual drawing of conclusions from the data as I had created a set of data that a marketing team would then analyse and make conclusions from which is what I set out to do but for this to be a final product conclusion from this data is what makes the difference. Through further analysis of marketing teams, I could have drawn conclusions from the dashboard I have created even automated computerized conclusions. Finally, they also reminded me that my database is ignoring several data protection laws by holding customer data in a unsecured open SQL database that can be queries by anyone without an password system so implementing one would be mandatory for a final product. This could be remedied using a password table to query into when the database is queried.

In conclusion, the problem I set out to solve was creating a system for computerized statistical analysis to understand the properties of a large half a million dataset which I believe I have been successful in. the system breaks it into useful composite data, analyses this created data and then presents the data created on a dashboard. The pit falls mainly come from how I choose to present the data and the lack of useful conclusions drawn from the data

Appendix A Extra:

The construction of Datatables:

I will be making extensive use of Datatables in this program as a fast and efficient way of storying hundreds of thousands of data points. The code for the setting up of the data tables is repetitive so I won't go through it for every column of every Datatable I will be using and will quickly describe their general creation in my program using the main dataset's datatable included in my data processing program as the example.

```
Create Datatable Mainset = new Datatable "MainSet"
DataColumn column
        // Create new DataColumn, set DataType,
        // ColumnName and add to DataTable.
        column = new DataColumn()
        column.DataType = int
        column.ColumnName = "PrimaryKey"
Mainset.Columns.Add(column);
// this process Is then repeated for every colum in datatable
  column = new DataColumn();
        column.DataType = String
        column.ColumnName = "StockCode";
        // Add the Column to the DataColumnCollection.
Mainset.Columns.Add(column);
```

Assigning a Primary Key column is as simply repetitive in code and will be explained once in this datatable

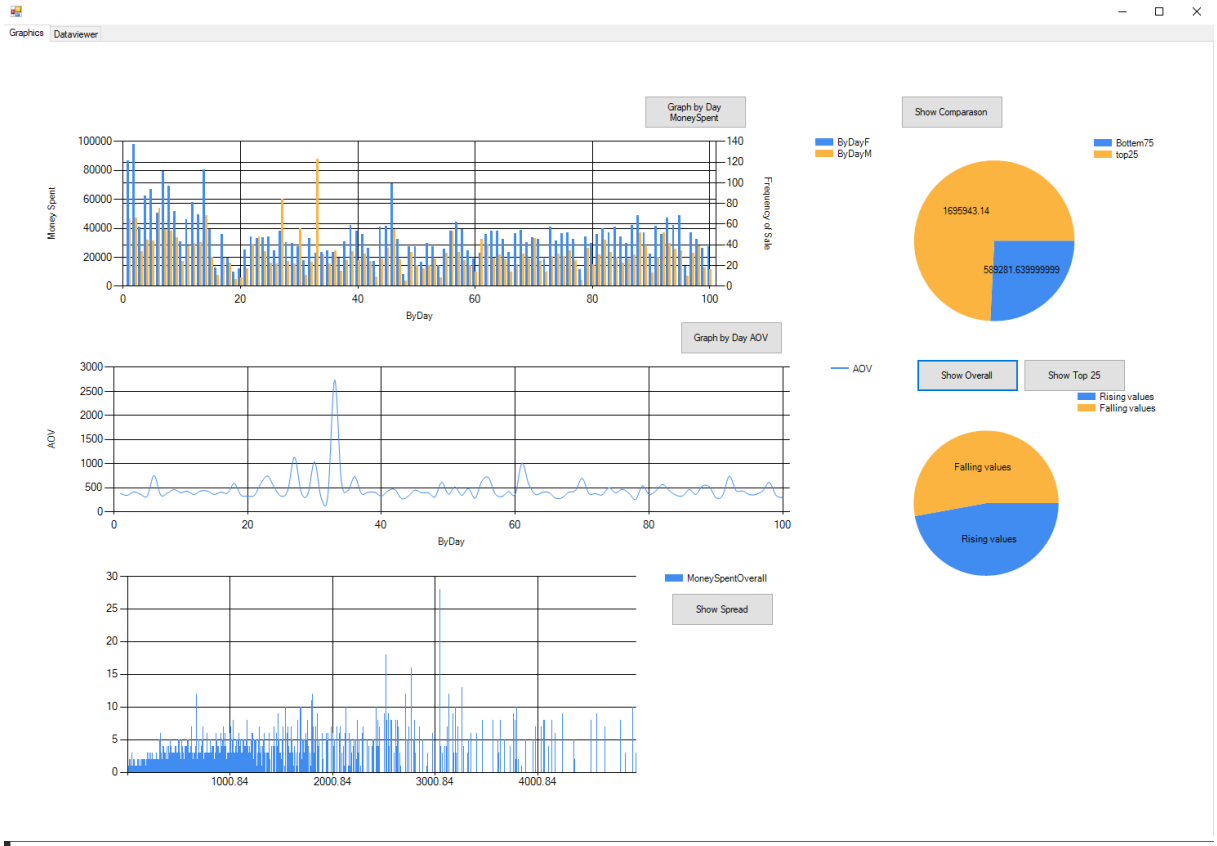
```
DataColumn[] PrimaryKeyColumns = new DataColumn[1];
        PrimaryKeyColumns[0] = table.Columns["PrimaryKey"];
        Mainset.PrimaryKey = PrimaryKeyColumns;
```

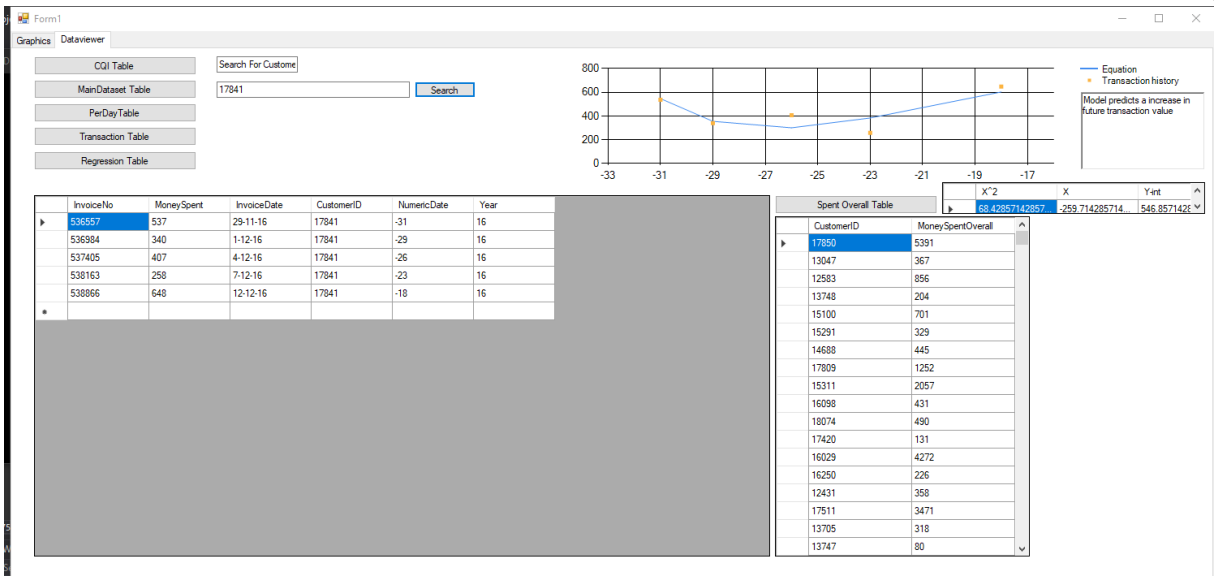
Requesting Rows From SQL:

To receive data from the databased as shown bellow in structured english:

```
SQLCon connection = new SQLCon (name of table you want to receive data from)
DataRow row = new Datarow;
While  reader.Read()
Row = DatatableToAddRowTo.NewRow()
Row["column name 1"] = reader.Get(Datatype)(0)
Row["column name 2"] = reader.Get(Datatype)(1)
Row["column name 3"] = reader.Get(Datatype)(2)
DatatableToAddRowTo.Rows.Add(Row)
End while
```

Appendix B Code





Dashboard program:

DataFromSQLTables - Line 93

TopIDs - Line 61

SQLCon - Line 23

If Byday button is pressed - Line959

If show comparison is pressed - Line 986

If graph by AOV is pressed: - Line 882

If Show overall is pressed - Line1301

SpentScatter() - Line 917

If any display table button is presses - Line1002

Clicking the search button - Line 1130

```
1.  using System;
2.  using System.IO;
3.  using System.Collections.Generic;
4.  using System.ComponentModel;
5.  using System.Data;
6.  using System.Drawing;
7.  using System.Linq;
8.  using System.Text;
9.  using System.Threading.Tasks;
10. using System.Windows.Forms;
11. using System.Data.SQLite;
12.
13.
14.
15. namespace WindowsFormsApp2
16. {
17.     public partial class Form1 : Form
18.     {
19.         static DataFromSQLTables tables = new DataFromSQLTables();
20.         static TopIDs topIDs = new TopIDs();
21.
22.
23.         class SQLCon
24.         {
25.             private string filepath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) +
    "\\CustomerData.db";
26.             public SQLiteDataReader rdr;
27.             private SQLiteConnection conn;
28.             public SQLCon(string Name)
29.             {
30.                 string command1 = "SELECT * FROM " + Name;
31.                 SQLiteConnection connection = CreateSQLconnection();
32.                 var cmd = new SQLiteCommand(command1, connection);
33.                 this.rdr = cmd.ExecuteReader();
34.             }
```

```
35.
36.
37.         public void ExicuteQuery(string CommandText)
38.         {
39.             SQLiteCommand sqlite_cmd;
40.             sqlite_cmd = conn.CreateCommand();
41.             sqlite_cmd.CommandText = CommandText;
42.             Console.WriteLine(CommandText);
43.             sqlite_cmd.ExecuteNonQuery();
44.         }
45.
46.         public SQLCon()
47.         {
48.             this.conn = new SQLiteConnection(@"Data Source =" + filepath);
49.             conn.Open();
50.         }
51.
52.         public SQLiteConnection CreateSQLconnection()
53.         {
54.             SQLiteConnection ll = new SQLiteConnection(@"Data Source =" + filepath);
55.             ll.Open();
56.             return ll;
57.         }
58.     }
59.
60.
61.     class TopIDs
62.     {
63.         public List<int> CQI = new List<int>();
64.         public List<int> Frequency = new List<int>();
65.         public List<int> MMperDayMoney = new List<int>();
66.         public List<int> MMperDayFrequency = new List<int>();
67.         public List<int> PerTrans = new List<int>();
68.         public List<int> Spent = new List<int>();
69.         public void GetID(string name, List<int> list)
70.         {
71.             SQLCon connection = new SQLCon(name);
72.
73.             SQLiteDataReader rdr = connection.rdr;
74.             while (rdr.Read())
75.             {
76.                 int data = (int)rdr.GetInt32(0);
77.
78.                 Console.WriteLine(data);
79.
80.                 list.Add(data);
81.             }
82.         }
83.         public TopIDs()
84.         {
85.             GetID("CQITop25CQI", CQI);
86.             GetID("CQITop25Frequency", Frequency);
87.             GetID("MoneyMadePerDayTop25Frequencyofsale", MMperDayFrequency);
88.             GetID("MoneyMadePerDayTop25MoneySpent", MMperDayMoney);
89.             GetID("MPPerTransTop25MoneySpent", PerTrans);
90.             GetID("SpentTop25MoneySpentOvertime", Spent);
91.         }
92.     }
93.     class DataFromSQLTables
94.     {
95.         public Dictionary<string, DataTable> Optionlist = new Dictionary<string, DataTable>();
96.         public DataSet SQLtables = new DataSet();
```

```
97.
98.         public DataFromSQLTables()
99.         {
100.            CreateLinregTableCustomerID();
101.            CreateSpent();
102.            CreateMPPertrans();
103.            CQI();
104.            FindMMPD();
105.            Mainset();
106.
107.        }
108.        private void CreateLinregTableCustomerID()
109.        {
110.            DataTable LinRegtable = new DataTable("CustomerIDRegresion");
111.            string[] dattypes = new string[6];
112.            string[] Tablenames = new string[6];
113.
114.            DataColumn column; //MoneySpent,InvoiceDate,CustomerID
115.
116.            SQLCon connection = new SQLCon("CustomerIDRegresion");
117.
118.            SQLiteDataReader rdr = connection.rdr;
119.
120.            column = new DataColumn();
121.            column.DataType = System.Type.GetType("System.String");
122.            column.ColumnName = "CustomerID";
123.            column.ReadOnly = true;
124.            LinRegtable.Columns.Add(column);
125.
126.            dattypes[0] = "VARCHAR";
127.            Tablenames[0] = "CustomerID";
128.
129.            column = new DataColumn();
130.            column.DataType = System.Type.GetType("System.Double");
131.            column.ColumnName = "SlopeXsq";
132.            column.ReadOnly = true;
133.            LinRegtable.Columns.Add(column);
134.
135.            dattypes[1] = "DOUBLE";
136.            Tablenames[1] = "SlopeXsq";
137.
138.            column = new DataColumn();
139.            column.DataType = System.Type.GetType("System.Double");
140.            column.ColumnName = "SlopeX";
141.            column.ReadOnly = true;
142.            LinRegtable.Columns.Add(column);
143.
144.            dattypes[2] = "DOUBLE";
145.            Tablenames[2] = "SlopeX";
146.
147.            column = new DataColumn();
148.            column.DataType = System.Type.GetType("System.Double");
149.            column.ColumnName = "Yintercept";
150.            column.ReadOnly = true;
151.            LinRegtable.Columns.Add(column);
152.
153.            dattypes[3] = "DOUBLE";
154.            Tablenames[3] = "Yintercept";
155.
156.            column = new DataColumn();
157.            column.DataType = System.Type.GetType("System.Double");
158.            column.ColumnName = "RSquared";
```

```
159.            column.ReadOnly = true;
160.            LinRegtable.Columns.Add(column);
161.
162.            dattypes[4] = "DOUBLE";
163.            Tablenames[4] = "RSquared";
164.
165.            column = new DataColumn();
166.            column.DataType = System.Type.GetType("System.Double");
167.            column.ColumnName = "ProductMomentCorilation";
168.            column.ReadOnly = true;
169.            LinRegtable.Columns.Add(column);
170.
171.            dattypes[5] = "DOUBLE";
172.            Tablenames[5] = "ProductMomentCorilation";
173.
174.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
175.            PrimaryKeyColumns[0] = LinRegtable.Columns["CustomerID"];
176.            LinRegtable.PrimaryKey = PrimaryKeyColumns;
177.
178.            Tuple<string[], string[]> arrays = Tuple.Create(Tablenames, dattypes);
179.
180.
181.            while (rdr.Read())
182.            {
183.                DataRow row;
184.                row = LinRegtable.NewRow();
185.                row["CustomerID"] = rdr.GetString(0);
186.                row["SlopeXsq"] = rdr.GetDouble(1);
187.                row["SlopeX"] = rdr.GetDouble(2);
188.                row["Yintercept"] = rdr.GetDouble(3);
189.                row["RSquared"] = rdr.GetDouble(4);
190.                row["ProductMomentCorilation"] = rdr.GetDouble(5);
191.                LinRegtable.Rows.Add(row);
192.            }
193.            Optionlist.Add(LinRegtable.TableName, LinRegtable);
194.
195.            SQLtables.Tables.Add(LinRegtable);
196.
197.
198.        }
199.        private void CQI()
200.        {
201.            //fill columns
202.            DataColumn column;
203.            DataTable Spent = Optionlist["Spent"];
204.            DataTable MPPerTrans = Optionlist["MPPerTrans"];
205.
206.            SQLCon connection = new SQLCon("CQI");
207.
208.            SQLiteDataReader rdr = connection.rdr;
209.
210.            DataTable CQIData = new DataTable("CQI");
211.
212.            string[] dattypes = new string[6];
213.            string[] Tablenames = new string[6];
214.
215.            column = new DataColumn();
216.            column.DataType = System.Type.GetType("System.Int32");
217.            column.ColumnName = "CustomerID";
218.            column.ReadOnly = true;
219.            CQIData.Columns.Add(column);
220.
```

```
221.            dattypes[1] = "VARCHAR";
222.            Tablenames[1] = "CustomerID";
223.
224.
225.
226.            column = new DataColumn();
227.            column.DataType = System.Type.GetType("System.Double");
228.            column.ColumnName = "MoneySpentOvertime";
229.            column.ReadOnly = true;
230.            CQIData.Columns.Add(column);
231.
232.            dattypes[2] = "DECIMAL";
233.            Tablenames[2] = "MoneySpentOvertime";
234.
235.
236.
237.            column = new DataColumn();
238.            column.DataType = System.Type.GetType("System.Double");
239.            column.ColumnName = "StandardDiv";
240.            column.ReadOnly = true;
241.            CQIData.Columns.Add(column);
242.
243.            dattypes[3] = "DECIMAL";
244.            Tablenames[3] = "StandardDiv";
245.
246.
247.
248.            column = new DataColumn();
249.            column.DataType = System.Type.GetType("System.Int32");
250.            column.ColumnName = "Frequency";
251.            column.ReadOnly = true;
252.            CQIData.Columns.Add(column);
253.
254.            dattypes[4] = "INTAGAR";
255.            Tablenames[4] = "Frequency";
256.
257.
258.
259.            column = new DataColumn();
260.            column.DataType = System.Type.GetType("System.Double");
261.            column.ColumnName = "CQI";
262.            column.ReadOnly = true;
263.            CQIData.Columns.Add(column);
264.
265.            dattypes[5] = "DECIMAL";
266.            Tablenames[5] = "CQI";
267.
268.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
269.            PrimaryKeyColumns[0] = CQIData.Columns["PrimaryKey"];
270.            CQIData.PrimaryKey = PrimaryKeyColumns;
271.
272.            DataRow row;
273.
274.            while (rdr.Read())
275.            {
276.                row = CQIData.NewRow();
277.
278.                row["CustomerID"] = rdr.GetString(1);
279.
280.                row["MoneySpentOvertime"] = rdr.GetDouble(2);
281.
282.                row["StandardDiv"] = rdr.GetDouble(3);
```

```
283.
284.                row["Frequency"] = rdr.GetInt16(4);
285.
286.                row["CQI"] = rdr.GetDouble(5);
287.
288.                CQIData.Rows.Add(row);
289.            }
290.            Optionlist.Add(CQIData.TableName, CQIData);
291.
292.            SQLtables.Tables.Add(CQIData);
293.        }
294.        private void FindMMPD()
295.        {
296.            {
297.
298.                SQLCon connection = new SQLCon("MoneyMadePerDay");
299.
300.                SQLiteDataReader rdr = connection.rdr;
301.                DataTable MMPerday = new DataTable("MoneyMadePerDay");
302.
303.                DataColumn column;
304.
305.
306.                column = new DataColumn();
307.                column.DataType = System.Type.GetType("System.Int32");
308.                column.ColumnName = "Day";
309.                column.ReadOnly = true;
310.                MMPerday.Columns.Add(column);
311.
312.                column = new DataColumn();
313.                column.DataType = System.Type.GetType("System.Double");
314.                column.ColumnName = "MoneySpent";
315.                column.ReadOnly = false;
316.                MMPerday.Columns.Add(column);
317.
318.                column = new DataColumn();
319.                column.DataType = System.Type.GetType("System.Int32");
320.                column.ColumnName = "Frequencyofsale";
321.                column.ReadOnly = false;
322.                MMPerday.Columns.Add(column);
323.
324.                column = new DataColumn();
325.                column.DataType = System.Type.GetType("System.Int32");
326.                column.ColumnName = "Year";
327.                column.ReadOnly = false;
328.                MMPerday.Columns.Add(column);
329.
330.                DataRow row;
331.
332.                while (rdr.Read())
333.                {
334.
335.
336.                    row = MMPerday.NewRow();
337.                    Console.WriteLine(rdr.GetInt32(1));
338.                    row["Day"] = rdr.GetInt32(1);
339.                    row["MoneySpent"] = rdr.GetInt32(2);
340.                    row["Frequencyofsale"] = rdr.GetInt32(3);
341.                    row["Year"] = rdr.GetInt32(4);
342.
343.                    MMPerday.Rows.Add(row);
344.                }
```

```
345.
346.
347.                Optionlist.Add(MMPerday.TableName, MMPerday);
348.                SQLtables.Tables.Add(MMPerday);
349.
350.            }
351.        }
352.
353.        private void Mainset()
354.        {
355.            System.Data.DataTable table = new DataTable("Main");
356.            // Create a new DataTable.
357.
358.            // Declare variables for DataColumn and DataRow objects.
359.            DataColumn column;
360.
361.
362.            // Create new DataColumn, set DataType,
363.            // ColumnName and add to DataTable.
364.            column = new DataColumn();
365.            column.DataType = System.Type.GetType("System.Int32");
366.            column.ColumnName = "PrimaryKey";
367.            column.ReadOnly = true;
368.            table.Columns.Add(column);
369.
370.            column = new DataColumn();
371.            column.DataType = System.Type.GetType("System.String");
372.            column.ColumnName = "StockCode";
373.            column.ReadOnly = true;
374.            // Add the Column to the DataColumnCollection.
375.            table.Columns.Add(column);
376.
377.            // Create second column.
378.            column = new DataColumn();
379.            column.DataType = System.Type.GetType("System.String");
380.            column.ColumnName = "Description";
381.            column.ReadOnly = true;
382.            // Add the column to the table.
383.            table.Columns.Add(column);
384.
385.            column = new DataColumn();
386.            column.DataType = System.Type.GetType("System.Int32");
387.            column.ColumnName = "Quantity";
388.            column.ReadOnly = true;
389.            table.Columns.Add(column);
390.
391.            column = new DataColumn();
392.            column.DataType = System.Type.GetType("System.String");
393.            column.ColumnName = "InvoiceDate";
394.            column.ReadOnly = true;
395.            table.Columns.Add(column);
396.
397.            column = new DataColumn();
398.            column.DataType = System.Type.GetType("System.Double");
399.            column.ColumnName = "UnitPrice";
400.            column.ReadOnly = true;
401.            table.Columns.Add(column);
402.
403.            column = new DataColumn();
404.            column.DataType = System.Type.GetType("System.Int32");
405.            column.ColumnName = "CustomerID";
406.            column.ReadOnly = true;
```

```
407.            table.Columns.Add(column);
408.
409.            column = new DataColumn();
410.            column.DataType = System.Type.GetType("System.String");
411.            column.ColumnName = "Country";
412.            column.ReadOnly = true;
413.            table.Columns.Add(column);
414.
415.            column = new DataColumn();
416.            column.DataType = System.Type.GetType("System.String");
417.            column.ColumnName = "InvoiceNo";
418.            column.ReadOnly = true;
419.            table.Columns.Add(column);
420.
421.            column = new DataColumn();
422.            column.DataType = System.Type.GetType("System.Double");
423.            column.ColumnName = "MoneySpent";
424.            column.ReadOnly = true;
425.            table.Columns.Add(column);
426.
427.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
428.            PrimaryKeyColumns[0] = table.Columns["id"];
429.            table.PrimaryKey = PrimaryKeyColumns;
430.
431.            // Make the ID column the primary key column.
432.
433.            int y = 1;
434.            DataRow row;
435.            SQLCon con = new SQLCon("CustomerData");
436.            SQLiteDataReader rdr = con.rdr;
437.
438.            while (rdr.Read())
439.            {
440.                //
    StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country,InvoiceNo,MoneySpent
441.                row = table.NewRow();
442.                row["PrimaryKey"] = y;
443.                row["StockCode"] = rdr.GetString(1);
444.                row["Description"] = rdr.GetString(2);
445.                row["Quantity"] = rdr.GetInt32(3);
446.                row["InvoiceDate"] = rdr.GetString(4);
447.                row["UnitPrice"] = rdr.GetDouble(5);
448.                row["CustomerID"] = rdr.GetInt32(6);
449.                row["Country"] = rdr.GetString(7);
450.                row["InvoiceNo"] = rdr.GetInt32(0);
451.                row["MoneySpent"] = Math.Round((rdr.GetInt32(3) * rdr.GetDouble(5)), 2);
452.                table.Rows.Add(row);
453.
454.                y++;
455.            }
456.
457.            Optionlist.Add(table.TableName, table);
458.
459.            SQLtables.Tables.Add(table);
460.            //return df;
461.        }
462.        private void CreateMPPertrans()
463.        {
464.            DataTable MPPerTrans = new DataTable("MPPerTrans");
465.
466.            SQLCon connection = new SQLCon("MPPerTrans");
467.
```

```
468.            SQLiteDataReader rdr = connection.rdr;
469.            {//creating columns
470.             //
471.               DataColumn column; //MoneySpent,InvoiceDate,CustomerID
472.
473.               column = new DataColumn();
474.               column.DataType = System.Type.GetType("System.String");
475.               column.ColumnName = "InvoiceNo";
476.               column.ReadOnly = true;
477.               MPPerTrans.Columns.Add(column);
478.
479.               string[] Tablenames = new string[5];
480.
481.               string[] dattypes = new string[5];
482.
483.               dattypes[0] = "VARCHAR";
484.               Tablenames[0] = "InvoiceNo";
485.
486.               column = new DataColumn();
487.               column.DataType = System.Type.GetType("System.Double");
488.               column.ColumnName = "MoneySpent";
489.               column.ReadOnly = true;
490.               MPPerTrans.Columns.Add(column);
491.
492.               dattypes[1] = "DOUBLE";
493.               Tablenames[1] = "MoneySpent";
494.
495.               column = new DataColumn();
496.               column.DataType = System.Type.GetType("System.String");
497.               column.ColumnName = "InvoiceDate";
498.               column.ReadOnly = true;
499.               MPPerTrans.Columns.Add(column);
500.
501.               dattypes[2] = "VARCHAR";
502.               Tablenames[2] = "InvoiceDate";
503.
504.               column = new DataColumn();
505.               column.DataType = System.Type.GetType("System.Int32");
506.               column.ColumnName = "CustomerID";
507.               column.ReadOnly = true;
508.               MPPerTrans.Columns.Add(column);
509.
510.               dattypes[3] = "VARCHAR";
511.               Tablenames[3] = "CustomerID";
512.               DataColumn[] PrimaryKeyColumns = new DataColumn[1];
513.               PrimaryKeyColumns[0] = MPPerTrans.Columns["InvoiceNo"];
514.               MPPerTrans.PrimaryKey = PrimaryKeyColumns;
515.
516.               column = new DataColumn();
517.               column.DataType = System.Type.GetType("System.Int32");
518.               column.ColumnName = "NumericDate";
519.               column.ReadOnly = true;
520.               MPPerTrans.Columns.Add(column);
521.
522.               dattypes[4] = "INTAGAR";
523.               Tablenames[4] = "NumericDate";
524.
525.
526.               column = new DataColumn();
527.               column.DataType = System.Type.GetType("System.Int32");
528.               column.ColumnName = "Year";
529.               column.ReadOnly = true;
```

```
530.                MPPerTrans.Columns.Add(column);
531.
532.
533.            }
534.            DataRow row;
535.
536.            Console.WriteLine(rdr.FieldCount);
537.
538.            while (rdr.Read())
539.            {
540.
541.                row = MPPerTrans.NewRow();
542.
543.                row["InvoiceNo"] = rdr.GetString(0);
544.                row["MoneySpent"] = rdr.GetDecimal(1);
545.                row["InvoiceDate"] = rdr.GetString(2);
546.                row["CustomerID"] = rdr.GetString(3);
547.                row["NumericDate"] = rdr.GetInt32(4);
548.                row["Year"] = rdr.GetInt32(5);
549.
550.                MPPerTrans.Rows.Add(row);
551.            }
552.
553.            Optionlist.Add(MPPerTrans.TableName, MPPerTrans);
554.
555.            SQLtables.Tables.Add(MPPerTrans);
556.
557.        }
558.
559.        private void CreateSpent()
560.        {
561.            DataTable Spent = new DataTable("Spent");
562.
563.            SQLCon connection = new SQLCon("Spent");
564.
565.            SQLiteDataReader rdr = connection.rdr;
566.            {//creating columns
567.             //
568.                DataColumn column; //MoneySpent,InvoiceDate,CustomerID
569.
570.                column = new DataColumn();
571.                column.DataType = System.Type.GetType("System.String");
572.                column.ColumnName = "CustomerID";
573.                column.ReadOnly = true;
574.                Spent.Columns.Add(column);
575.
576.                column = new DataColumn();
577.                column.DataType = System.Type.GetType("System.Double");
578.                column.ColumnName = "MoneySpentOvertime";
579.                column.ReadOnly = true;
580.                Spent.Columns.Add(column);
581.
582.                DataColumn[] PrimaryKeyColumns = new DataColumn[1];
583.                PrimaryKeyColumns[0] = Spent.Columns["PrimaryKey"];
584.                Spent.PrimaryKey = PrimaryKeyColumns;
585.
586.
587.            }
588.            DataRow row;
589.
590.            while (rdr.Read())
591.            {
```

```
592.                    row = Spent.NewRow();
593.                    row["CustomerID"] = rdr.GetString(1);
594.                    row["MoneySpentOvertime"] = rdr.GetDecimal(2);
595.
596.                    Spent.Rows.Add(row);
597.                }
598.                Optionlist.Add(Spent.TableName, Spent);
599.
600.
601.                SQLtables.Tables.Add(Spent);
602.            }
603.
604.
605.        }
606.        public Form1()
607.        {
608.
609.            InitializeComponent();
610.
611.            dataGridView1.ColumnCount = 2;
612.
613.            dataGridView1.Columns[0].Name = "CustomerID";
614.
615.            dataGridView1.Columns[1].Name = "MoneySpentOverall";
616.
617.            dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
618.
619.
620.            //dataGridView1.DataSource = tables.Optionlist["Spent"];
621.
622.
623.            foreach (DataRow item in tables.Optionlist["Spent"].Rows)
624.            {
625.                dataGridView1.Rows.Add(Convert.ToInt32((string)item.ItemArray[0]),
    Convert.ToInt32((double)item.ItemArray[1]));
626.
627.            }
628.
629.
630.        }
631.
632.        private void button1_Click(object sender, EventArgs e)
633.        {
634.
635.            using (StreamReader reader = new StreamReader("MoneyMadePerDay DF 397924"))
636.            {
637.                chart1.Series["ByDay"].Points.Clear();
638.                string line;
639.
640.                line = reader.ReadLine();
641.                while ((line = reader.ReadLine()) != null)
642.                {
643.
644.                    //x is day
645.                    //y is MM
646.
647.                    if (line != null)
648.                    {
649.
650.                        string[] DifColumns = line.Split(',');
651.                        Console.WriteLine(DifColumns[2]);
652.                        double moneymade = Convert.ToDouble(DifColumns[0]);
```

```
653.                    double day = Convert.ToDouble(DifColumns[2]);
654.                    chart1.Series["ByDay"].Points.AddXY(day, moneymade);
655.
656.                }
657.            }
658.        }
659.    }
660.    private void button2_Click(object sender, EventArgs e)
661.    {
662.
663.    }
664.
665.    private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
666.    {
667.
668.    }
669.
670.    private void button3_Click(object sender, EventArgs e)
671.    {
672.        using (StreamReader reader = new StreamReader("MoneyMadePerDay DF 397924"))
673.        {
674.            chart1.Series["ByDay"].Points.Clear();
675.            string line;
676.            line = reader.ReadLine();
677.
678.            while ((line = reader.ReadLine()) != null)
679.            {
680.
681.                //x is day
682.                //y is MM
683.
684.                if (line != null)
685.                {
686.                    string[] DifColumns = line.Split(',');
687.                    Console.WriteLine(DifColumns[1]);
688.                    double frequency = Convert.ToDouble(DifColumns[1]);
689.                    double day = Convert.ToDouble(DifColumns[2]);
690.                    chart1.Series["ByDay"].Points.AddXY(day, frequency);
691.
692.
693.                }
694.            }
695.        }
696.    }
697.
698.    private void button4_Click(object sender, EventArgs e)
699.    {
700.        using (StreamReader reader = new StreamReader("MoneyMadePerDay DF 397924"))
701.        {
702.            string line;
703.            line = reader.ReadLine();
704.            chart1.Series["ByDay"].Points.Clear();
705.            //chartnameHERE.Series["SeriesNameHERE"].Points.Clear();
706.
707.            while ((line = reader.ReadLine()) != null)
708.            {
709.
710.                //x is day
711.                //y is MM
712.
713.                if (line != null)
714.                {
```

```
715.
716.                    string[] DifColumns = line.Split(',');
717.                    Console.WriteLine(DifColumns[2]);
718.                    double frequency = Convert.ToDouble(DifColumns[1]);
719.                    double Moneymade = Convert.ToDouble(DifColumns[0]);
720.                    double day = Convert.ToDouble(DifColumns[2]);
721.                    double AOV = Moneymade / frequency;
722.                    chart1.Series["ByDay"].Points.AddXY(day, AOV);
723.
724.
725.                }
726.            }
727.        }
728.    }
729.
730.    private void chart1_Click(object sender, EventArgs e)
731.    {
732.
733.    }
734.
735.    private void chart2_Click(object sender, EventArgs e)
736.    {
737.
738.    }
739.
740.    private void button2_Click_1(object sender, EventArgs e)
741.    {
742.        chart2.Series["80/20"].Points.Clear();
743.        using (StreamReader reader = new StreamReader("80 20 rule397924"))
744.        {
745.            reader.ReadLine();
746.            double overall = Convert.ToDouble(reader.ReadLine());
747.            double top25 = Convert.ToDouble(reader.ReadLine());
748.            double bottem75 = overall - top25;
749.            top25 = Math.Round(top25);
750.            bottem75 = Math.Round(bottem75);
751.            chart2.Series["80/20"].Points.AddXY("Bottem75", bottem75);
752.            chart2.Series["80/20"].Points.AddXY("top25", top25);
753.
754.        }
755.    }
756.
757.    private void textBox1_TextChanged(object sender, EventArgs e)
758.    {
759.
760.    }
761.
762.    private void textBox2_TextChanged(object sender, EventArgs e)
763.    {
764.
765.    }
766.
767.
768.
769.    private void richTextBox1_TextChanged(object sender, EventArgs e)
770.    {
771.
772.    }
773.
774.
775.
776.
```

```
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.        private void button15_Click(object sender, EventArgs e)
791.        {
792.
793.        }
794.
795.        private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
796.        {
797.
798.        }
799.
800.        private void chart1_Click_1(object sender, EventArgs e)
801.        {
802.
803.        }
804.
805.        private void button1_Click_1(object sender, EventArgs e)
806.        {
807.
808.        }
809.
810.        private void button8_Click_1(object sender, EventArgs e)
811.        {
812.
813.        }
814.
815.        private void button16_Click(object sender, EventArgs e)
816.        {
817.
818.        }
819.
820.        private void tabPage1_Click(object sender, EventArgs e)
821.        {
822.
823.        }
824.
825.        private void tabPage3_Click(object sender, EventArgs e)
826.        {
827.
828.        }
829.
830.        private void button8_Click_2(object sender, EventArgs e)
831.        {
832.            tabs.SelectedTab = GraphicsTab;
833.        }
834.
835.        private void button16_Click_1(object sender, EventArgs e)
836.        {
837.            tabs.SelectedTab = Dataview;
838.        }
```

```
839.
840.        private void button18_Click(object sender, EventArgs e)
841.        {
842.            tabs.SelectedTab = Dataview;
843.        }
844.
845.        private void button17_Click(object sender, EventArgs e)
846.        {
847.            tabs.SelectedTab = Dataview;
848.        }
849.
850.        private void textBox3_TextChanged(object sender, EventArgs e)
851.        {
852.
853.        }
854.
855.        private void dataGridView1_CellContentClick_1(object sender, DataGridViewCellEventArgs e)
856.        {
857.
858.        }
859.
860.        private void button1_Click_2(object sender, EventArgs e)
861.        {
862.
863.        }
864.
865.        private void chart1_Click_2(object sender, EventArgs e)
866.        {
867.
868.        }
869.
870.        private void button2_Click_2(object sender, EventArgs e)
871.        {
872.
873.
874.            //chart7
875.        }
876.
877.        private void button3_Click_1(object sender, EventArgs e)
878.        {
879.
880.        }
881.
882.        private void button4_Click_1(object sender, EventArgs e)
883.        {
884.            chart4.Series["AOV"].Points.Clear();
885.            DataTable MMPD = tables.Optionlist["MoneyMadePerDay"];
886.            int y = 1;
887.            foreach (DataRow item in MMPD.Rows)
888.            {
889.
890.                //  int day = (int)item.ItemArray[0] + 31;
891.                decimal AOV = (decimal)(double)item.ItemArray[1] / (int)item.ItemArray[2];
892.                this.chart4.Series["AOV"].Points.AddXY(y, AOV);
893.                y++;
894.            }
895.
896.
897.        }
898.
899.        private void button19_Click(object sender, EventArgs e)
900.        {
```

```
901.
902.        }
903.        private void Form1_Load(object sender, EventArgs e)
904.        {
905.            //MoneySpent graph per day
906.
907.            DataFromSQLTables tables = new DataFromSQLTables();
908.
909.            TopIDs ids = new TopIDs();
910.
911.
912.
913.
914.
915.        }
916.
917.        private void SpentScatter()
918.        {
919.
920.            chart3.Series["MoneySpentOverall"].Points.Clear();
921.            DataTable spent = tables.Optionlist["Spent"];
922.            DataTable Trans = tables.Optionlist["MPPerTrans"];
923.            foreach (DataRow item in spent.Rows)
924.            {
925.                if ((double)item.ItemArray[1]>5000)
926.                {
927.
928.                }
929.                else
930.                {
931.
932.                    DataRow[] rows = Trans.Select("CustomerID=  '" + item.ItemArray[0] + "'");
933.                    double money = (double)item.ItemArray[1];
934.
935.                    chart3.Series["MoneySpentOverall"].Points.AddXY(money, rows.Length);
936.                }
937.            }
938.        }
939.        private Tuple<double, double> Tally2080(DataTable Spent, List<int> Top25)
940.        {
941.            double Twenty = 0;
942.            double Sevintyfive = 0;
943.
944.            foreach (DataRow item in Spent.Rows)
945.            {
946.                if (Top25.Contains(Convert.ToInt32(item.ItemArray[0])))
947.                {
948.                    Twenty = Twenty + (double)item.ItemArray[1];
949.                }
950.                else
951.                {
952.                    Sevintyfive = Sevintyfive + (double)item.ItemArray[1];
953.                }
954.            }
955.            Tuple<double, double> T = new Tuple<double, double>(Twenty, Sevintyfive);
956.            return T;
957.        }
958.
959.        private void button1_Click_3(object sender, EventArgs e)
960.        {
961.            chart1.Series["ByDayM"].Points.Clear();
962.            chart1.Series["ByDayF"].Points.Clear();
```

```
963.            DataTable MMPD = tables.Optionlist["MoneyMadePerDay"];
964.            int y = 1;
965.            foreach (DataRow item in MMPD.Rows)
966.            {
967.                this.chart1.Series["ByDayM"].Points.AddXY(y, (double)item.ItemArray[1]);
968.                this.chart1.Series["ByDayF"].Points.AddXY(y, (int)item.ItemArray[2]);
969.                y++;
970.            }
971.
972.
973.        }
974.
975.        private void button6_Click(object sender, EventArgs e)
976.        {
977.
978.            SpentScatter();
979.        }
980.
981.        private void chart3_Click(object sender, EventArgs e)
982.        {
983.
984.        }
985.
986.        private void button5_Click(object sender, EventArgs e)
987.        {
988.            chart2.Series["80/20"].Points.Clear();
989.            TopIDs topIDs = new TopIDs();
990.            Tuple<double, double> Sevinty25 = Tally2080(tables.Optionlist["Spent"], topIDs.Spent);
991.
992.            chart2.Series["80/20"].Points.AddXY("Bottem75", Sevinty25.Item2);
993.            chart2.Series["80/20"].Points.AddXY("top25", Sevinty25.Item1);
994.
995.        }
996.
997.        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
998.        {
999.
1000.       }
1001.
1002.       private void CQI_Click(object sender, EventArgs e)
1003.       {
1004.           dataGridView2.Columns.Clear();
1005.           dataGridView2.DataSource = tables.Optionlist["CQI"];
1006.
1007.       }
1008.
1009.       private void button9_Click(object sender, EventArgs e)
1010.       {
1011.           dataGridView2.Columns.Clear();
1012.           dataGridView2.DataSource = tables.Optionlist["Main"];
1013.       }
1014.
1015.       private void button10_Click(object sender, EventArgs e)
1016.       {
1017.           dataGridView2.Columns.Clear();
1018.           dataGridView2.DataSource = tables.Optionlist["MoneyMadePerDay"];
1019.       }
1020.
1021.       private void button7_Click(object sender, EventArgs e)
1022.       {
1023.           dataGridView2.Columns.Clear();
1024.           dataGridView2.DataSource = tables.Optionlist["MPPerTrans"];
```

```
1025.        }
1026.
1027.        private void button11_Click(object sender, EventArgs e)
1028.        {
1029.
1030.        }
1031.
1032.        private void maskedTextBox1_MaskInputRejected(object sender, MaskInputRejectedEventArgs
     e)
1033.        {
1034.
1035.        }
1036.
1037.        private void button12_Click(object sender, EventArgs e)
1038.        {
1039.
1040.
1041.
1042.
1043.
1044.
1045.        }
1046.
1047.        private void dataGridView1_CellContentClick_2(object sender, DataGridViewCellEventArgs e)
1048.        {
1049.
1050.        }
1051.
1052.
1053.        private double MakeEquation(DataRow row, int day)
1054.        {
1055.            double SlopeSQ = (double)row.ItemArray[1];
1056.
1057.            double slopex = (double)row.ItemArray[2];
1058.
1059.            double Yint = (double)row.ItemArray[3];
1060.
1061.            double Ypooint = (SlopeSQ * (day * day)) + (slopex * day) + Yint;
1062.
1063.            return Ypooint;
1064.        }
1065.
1066.
1067.
1068.        private void showEquation(string Customerid)
1069.        {
1070.            // this section is to present the transaction huistory on a graph and if applicable write its linear
     regression equation along side it
1071.            chart5.Series["Transaction history"].Points.Clear();
1072.            chart5.Series["Equation"].Points.Clear();
1073.
1074.            DataTable Spent = tables.Optionlist["Spent"];
1075.            DataTable linregtable = tables.Optionlist["CustomerIDRegresion"];
1076.            DataRow[] CustomerLinreg = linregtable.Select("CustomerID=  '" + Customerid + "'");
1077.
1078.            if (CustomerLinreg.Length > 0)
1079.            {
1080.                DataTable Trans = tables.Optionlist["MPPerTrans"];
1081.                DataRow[] DRows = Trans.Select("CustomerID=  '" + Customerid + "'");
1082.                int x = 0;
1083.                foreach (DataRow item in DRows)
1084.                {
```

```
1085.                 chart5.Series["Transaction history"].Points.AddXY((int)item.ItemArray[4],
     (double)item.ItemArray[1]);
1086.
1087.                 if ((double)CustomerLinreg[0].ItemArray[1] == 0 &&
     (double)CustomerLinreg[0].ItemArray[2] == 0
1088.                                                 && (double)CustomerLinreg[0].ItemArray[3] == 0)
1089.                 {
1090.                 }
1091.                 else
1092.                 {
1093.                     chart5.Series["Equation"].Points.AddXY((int)item.ItemArray[4],
     MakeEquation(CustomerLinreg[0], x));
1094.                     x++;
1095.                 }
1096.             }
1097.
1098.             dataGridView3.DataSource = null;
1099.             dataGridView3.ColumnCount = 4;
1100.             dataGridView3.Columns[0].Name = "X^2";
1101.             dataGridView3.Columns[1].Name = "X";
1102.             dataGridView3.Columns[2].Name = "Y-int";
1103.             dataGridView3.Columns[3].Name = "R^2";
1104.             DataGridViewRow row = (DataGridViewRow)dataGridView3.Rows[0].Clone();
1105.             row.Cells[0].Value = CustomerLinreg[0].ItemArray[1];
1106.             row.Cells[1].Value = CustomerLinreg[0].ItemArray[2];
1107.             row.Cells[2].Value = CustomerLinreg[0].ItemArray[3];
1108.             row.Cells[3].Value = CustomerLinreg[0].ItemArray[4];
1109.
1110.             dataGridView3.Rows.Add(row);
1111.             decimal[] Equation = new decimal[2] {
     (decimal)Convert.ToDecimal((double)CustomerLinreg[0].ItemArray[1])
1112.                                         ,
     (decimal)Convert.ToDecimal((double)CustomerLinreg[0].ItemArray[2]) };
1113.             decimal gradient = GetGradient(Equation, DRows.Length-1);
1114.             if (gradient > 0)
1115.             {
1116.                 richTextBox1.Text = "Model predicts a increase in future transaction value";
1117.             }
1118.             else if (gradient < 0)
1119.             {
1120.                 richTextBox1.Text = "Model predicts a Decrease in future transaction value";
1121.             }
1122.         }
1123.         else
1124.         {
1125.             richTextBox1.Text = "Graph not avalible insificiant data";
1126.         }
1127.
1128.     }
1129.
1130.     private void button13_Click(object sender, EventArgs e)
1131.     {
1132.
1133.         DataTable Spent = tables.Optionlist["Spent"];
1134.
1135.         DataRow[] Tesrows = Spent.Select("CustomerID=  '" + (string)textBox2.Text + "'");
1136.
1137.         if (Tesrows.Length > 0)
1138.         {
1139.
1140.             showEquation((string)textBox2.Text);
1141.
```

```
1142.            if (dataGridView2.DataSource != null)
1143.            {
1144.
1145.
1146.                List<DataGridViewRow> rows = new List<DataGridViewRow>();
1147.
1148.                if (dataGridView2.Columns[0].Name == "CustomerID" &&
     dataGridView2.Columns[1].Name!= "SlopeXsq")
1149.                {
1150.
1151.                    int y = 0;
1152.                    for (int i = 0; i < dataGridView2.Rows.Count - 1; i++)
1153.                    {
1154.
1155.                        Console.WriteLine(y);
1156.                        if (dataGridView2.Rows[i].Cells[0].Value.ToString().Equals((string)textBox2.Text))
1157.                        {
1158.                            DataGridViewRow row = (DataGridViewRow)dataGridView2.Rows[i].Clone();
1159.
1160.                            for (int j = 0; j < dataGridView2.Rows[i].Cells.Count; j++)
1161.                            {
1162.                                row.Cells[j].Value = dataGridView2.Rows[i].Cells[j].Value;
1163.                            }
1164.                            rows.Add(dataGridView2.Rows[i]);
1165.                            dataGridView2.FirstDisplayedScrollingRowIndex = i;
1166.                        }
1167.                        y++;
1168.                    }
1169.
1170.                    dataGridView2.DataSource = null;
1171.                    dataGridView2.Rows.Clear();
1172.
1173.                    dataGridView2.ColumnCount = 5;
1174.
1175.                    dataGridView2.Columns[0].Name = "CustomerID";
1176.                    dataGridView2.Columns[1].Name = "MoneySpentOvertime";
1177.                    dataGridView2.Columns[2].Name = "StandardDiv";
1178.                    dataGridView2.Columns[3].Name = "Frequency";
1179.                    dataGridView2.Columns[4].Name = "CQI";
1180.
1181.                    foreach (DataGridViewRow item in rows)
1182.                    {
1183.                        dataGridView2.Rows.Add(item);
1184.                    }
1185.
1186.                }
1187.                else if (dataGridView2.Columns[0].Name == "InvoiceNo")
1188.                {
1189.
1190.                    int y = 0;
1191.                    for (int i = 0; i < dataGridView2.Rows.Count - 1; i++)
1192.                    {
1193.
1194.                        Console.WriteLine(y);
1195.                        if (dataGridView2.Rows[i].Cells[3].Value.ToString().Equals((string)textBox2.Text))
1196.                        {
1197.                            DataGridViewRow row = (DataGridViewRow)dataGridView2.Rows[i].Clone();
1198.
1199.                            for (int j = 0; j < dataGridView2.Rows[i].Cells.Count; j++)
1200.                            {
1201.                                row.Cells[j].Value = dataGridView2.Rows[i].Cells[j].Value;
1202.                            }
```

```
1203.
1204.                    rows.Add(row);
1205.
1206.                }
1207.                y++;
1208.            }
1209.
1210.            dataGridView2.DataSource = null;
1211.
1212.            dataGridView2.ColumnCount = 6;
1213.
1214.            dataGridView2.Columns[0].Name = "InvoiceNo";
1215.            dataGridView2.Columns[1].Name = "MoneySpent";
1216.            dataGridView2.Columns[2].Name = "InvoiceDate";
1217.            dataGridView2.Columns[3].Name = "CustomerID";
1218.            dataGridView2.Columns[4].Name = "NumericDate";
1219.            dataGridView2.Columns[5].Name = "Year";
1220.
1221.            foreach (DataGridViewRow item in rows)
1222.            {
1223.                dataGridView2.Rows.Add(item);
1224.            }
1225.
1226.        }
1227.        else if (dataGridView2.Columns[0].Name == "PrimaryKey")
1228.        {
1229.            int y = 0;
1230.            for (int i = 0; i < dataGridView2.Rows.Count - 1; i++)
1231.            {
1232.
1233.                Console.WriteLine(y);
1234.                if (dataGridView2.Rows[i].Cells[6].Value.ToString().Equals((string)textBox2.Text))
1235.                {
1236.
1237.                    DataGridViewRow row = (DataGridViewRow)dataGridView2.Rows[i].Clone();
1238.
1239.                    for (int j = 0; j < dataGridView2.Rows[i].Cells.Count; j++)
1240.                    {
1241.                        row.Cells[j].Value = dataGridView2.Rows[i].Cells[j].Value;
1242.                    }
1243.
1244.                    rows.Add(row);
1245.                    dataGridView2.FirstDisplayedScrollingRowIndex = i;
1246.                }
1247.                y++;
1248.            }
1249.
1250.            dataGridView2.DataSource = null;
1251.
1252.            dataGridView2.ColumnCount = 10;
1253.
1254.            dataGridView2.Columns[0].Name = "Primarykey";
1255.            dataGridView2.Columns[1].Name = "InvoiceNo";
1256.            dataGridView2.Columns[2].Name = "StockCode";
1257.            dataGridView2.Columns[3].Name = "Description";
1258.            dataGridView2.Columns[4].Name = "Quantity";
1259.            dataGridView2.Columns[5].Name = "InvoiceDate";
1260.            dataGridView2.Columns[6].Name = "UnitPrice";
1261.            dataGridView2.Columns[7].Name = "CustomerID";
1262.            dataGridView2.Columns[8].Name = "Country";
1263.            dataGridView2.Columns[9].Name = "MoneySpent";
1264.
```

```
                    foreach (DataGridViewRow item in rows)
                    {
                        dataGridView2.Rows.Add(item);
                    }


                }
            }
        }

    }

    private void dataGridView2_CellContentClick(object sender, DataGridViewCellEventArgs e)
    {

    }

    private void chart5_Click(object sender, EventArgs e)
    {

    }

    private void dataGridView3_CellContentClick(object sender, DataGridViewCellEventArgs e)
    {

    }
    private decimal GetGradient(decimal[] equation, decimal Xvalue)
    {
        decimal Xsq = equation[0];
        decimal x = equation[1];

        decimal gradient = (2m * Xsq) * (Xvalue) + x;

        return gradient;
    }

    private void button3_Click_2(object sender, EventArgs e)
    {
        chart6.Series["RS"].Points.Clear();
        DataTable CustomerLinreg = tables.Optionlist["CustomerIDRegresion"];
        DataTable Trans = tables.Optionlist["MPPerTrans"];

        int pos = 0;
        int neg = 0;
        foreach (DataRow item in CustomerLinreg.Rows)
        {

            if ((double)item.ItemArray[1] == 0 && (double)item.ItemArray[2] == 0 && (double)item.ItemArray[3] == 0)
            {

            }
            else
            {
                DataRow[] rows = Trans.Select("CustomerID=  '" + (string)item.ItemArray[0] + "'");

                decimal[] Equation = new decimal[2] { (decimal)Convert.ToDecimal((double)item.ItemArray[1])
                                                    , (decimal)Convert.ToDecimal((double)item.ItemArray[2]) };

                decimal Gradient = GetGradient(Equation, rows.Length - 1);
                if (Gradient > 0m)
```

```
1325.                {
1326.                    pos++;
1327.                }
1328.                else if(Gradient < 0)
1329.                {
1330.                    neg++;
1331.                }
1332.            }
1333.        }
1334.
1335.        chart6.Series["RS"].Points.AddXY("Rising values", pos);
1336.        chart6.Series["RS"].Points.AddXY("Falling values", neg);
1337.
1338.    }
1339.
1340.    private void button12_Click_1(object sender, EventArgs e)
1341.    {
1342.        chart6.Series["RS"].Points.Clear();
1343.        DataTable CustomerLinreg = tables.Optionlist["CustomerIDRegresion"];
1344.        DataTable Trans = tables.Optionlist["MPPerTrans"];
1345.        List<int> Top25 = topIDs.Spent;
1346.        int pos = 0;
1347.        int neg = 0;
1348.        foreach (DataRow item in CustomerLinreg.Rows)
1349.        {
1350.            if (Top25.Contains( Convert.ToInt32((string)item.ItemArray[0])))
1351.            {
1352.                if ((double)item.ItemArray[1] == 0 && (double)item.ItemArray[2] == 0 &&
     (double)item.ItemArray[3] == 0)
1353.                {
1354.
1355.                }
1356.                else
1357.                {
1358.                    DataRow[] rows = Trans.Select("CustomerID=  '" + (string)item.ItemArray[0] + "'");
1359.
1360.                    decimal[] Equation = new decimal[2] {
     (decimal)Convert.ToDecimal((double)item.ItemArray[1]),
     (decimal)Convert.ToDecimal((double)item.ItemArray[2]) };
1361.
1362.                    decimal Gradient = GetGradient(Equation, rows.Length - 1);
1363.                    if (Gradient > 0m)
1364.                    {
1365.                        pos++;
1366.                    }
1367.                    else if (Gradient < 0)
1368.                    {
1369.                        neg++;
1370.                    }
1371.                }
1372.            }
1373.        }
1374.
1375.        chart6.Series["RS"].Points.AddXY("Rising values", pos);
1376.        chart6.Series["RS"].Points.AddXY("Falling values", neg);
1377.
1378.
1379.    }
1380.
1381.    private void button2_Click_3(object sender, EventArgs e)
1382.    {
1383.        dataGridView2.Columns.Clear();
```

```
1384.            dataGridView2.DataSource = tables.Optionlist["CustomerIDRegresion"];
1385.        }
1386.    }
1387. }
```

Regression program:

LinearRegression class - Line 670

Datacreated Class - Line42

DataFromSQL class - Line424

```
1. using System;
2. using System.Data.SQLite;
3. using System.Collections.Generic;
4. using System.IO;
5. using System.Data;
6. namespace Linear_regression
7. {
8.     class SQLCon
9.     {
10.         const int amountofdataRows = 3000;
11.         private string filepath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    + "\\CustomerData.db";
12.         public SQLiteDataReader reader;
13.         public SQLCon(string Name)
14.         {
15.             string command1 = "SELECT * FROM "+Name;
16.
17.             SQLiteConnection connection = new SQLiteConnection(@"Data Source =" + filepath);
18.             connection.Open();
19.             using var cmd = new SQLiteCommand(command1, connection);//needs to be redone
20.             //write a cross table tabulation
21.              this.reader = cmd.ExecuteReader();
22.         }
23.
24.         private SQLiteConnection conn;
25.         public void ExicuteQuery(string CommandText)
26.         {
27.             SQLiteCommand sqlite_cmd;
28.             sqlite_cmd = conn.CreateCommand();
29.             sqlite_cmd.CommandText = CommandText;
30.             Console.WriteLine(CommandText);
31.             sqlite_cmd.ExecuteNonQuery();
32.         }
33.
34.         public SQLCon()//works as a methood used within the exicution
35.         {
36.             this.conn = new SQLiteConnection(@"Data Source =" + filepath);
37.             conn.Open();
38.         }
39.
40.
41.     }
42.     class Datacreated
43.     {
44.         Dictionary<DataTable, Tuple<string[], string[]>> TableNamesDatatypes = new
    Dictionary<DataTable, Tuple<string[], string[]>>();
45.
46.
47.
```

```
48.         public void SaveToSQL(DataTable Sheet)
49.         {
50.             if (Sheet.Rows.Count ==0)
51.             {
52.
53.             }
54.             else
55.             {
56.                 SaveData(Sheet);
57.                 InsertData(Sheet);
58.             }
59.
60.         }
61.         private void SaveData(DataTable Sheet)
62.         {
63.             SQLCon link = new SQLCon();
64.
65.             string CreateTable2 = "drop table if exists " + Sheet.TableName;
66.             Console.WriteLine(CreateTable2);
67.             link.ExicuteQuery(CreateTable2);
68.
69.             string CreateTable = "CREATE TABLE" + " " + Sheet.TableName + "( " +
    MakeColumsreadyForSQL(false, Sheet) + " );"; ;
70.             Console.WriteLine(CreateTable);
71.
72.             link.ExicuteQuery(CreateTable);
73.
74.         }
75.         private string MakeColumsreadyForSQL(bool Rows, DataTable Sheet)
76.         // has a bool to see if this is for the insert or not as the data types are used
77.         // to create the table but not when inserting
78.         {
79.
80.             string[] Datatypes = TableNamesDatatypes[Sheet].Item2;
81.
82.             string data = "";
83.             for (int i = 0; i < Sheet.Columns.Count; i++)
84.             {
85.                 if (i == (Sheet.Columns.Count - 1))
86.                 {
87.                     if (Rows == true)
88.                     {
89.                         data = data + Sheet.Columns[i].ColumnName + " ";
90.                     }
91.                     else
92.                     {
93.                         data = data + Sheet.Columns[i].ColumnName + " " + Datatypes[i];
94.                     }
95.                 }
96.                 else
97.                 {
98.                     if (Rows == true)
99.                     {
100.                        data = data + Sheet.Columns[i].ColumnName + "," + " ";
101.                    }
102.                    else
103.                    {
104.                        data = data + Sheet.Columns[i].ColumnName + " " + Datatypes[i] + ", ";
105.                    }
106.                }
107.            }
108.            return data;
109.        }
110.
111.
112.        private string MakeRowsReadyforSQL(int i, DataTable Sheet)
113.        {
```

```
114.            string data = "";
115.            for (int j = 0; j < Sheet.Columns.Count; j++)
116.            {
117.                if (j == (Sheet.Columns.Count - 1))
118.                {
119.                    data = data + Sheet.Rows[i].ItemArray[j];
120.                }
121.                else
122.                {
123.                    data = data + Sheet.Rows[i].ItemArray[j] + ", ";
124.                }
125.            }
126.            return data;
127.        }
128.        private void InsertData(DataTable Sheet)
129.        {
130.            SQLCon link = new SQLCon();
131.            for (int i = 0; i < Sheet.Rows.Count; i++)
132.            {
133.                string InsertText = "INSERT INTO" + " " + Sheet.TableName + "( " +
    MakeColumsreadyForSQL(true, Sheet) + ")"
134.                                    + " VALUES(" + MakeRowsReadyforSQL(i,Sheet) + "); ";
135.                Console.WriteLine(InsertText);
136.                link.ExicuteQuery(InsertText);
137.            }
138.        }
139.        public void FillLinregTableDays(DataTable MMperDay, ref DataTable QuatersMoney, ref
    DataTable QuatersFreq)
140.        {
141.            //now first we must seperate the dates between quarters
142.            // We will overall have 365 data points however for testing I need to break up any data
    inputted into quarters
143.            List<int> days = new List<int>();
144.            MMperDay.DefaultView.Sort=("Day"); // places them in order of Days
145.            foreach (DataRow item in MMperDay.Rows)
146.            {
147.                days.Add((int)item.ItemArray[1]);
148.            }
149.            if (days.Count <4) // if the number of days in the system is lower than 4 quaters aren't
    possible
150.            {
151.            }
152.            else
153.            {
154.                double quarter = (days.Count / 4);
155.                int Q = Convert.ToInt16(Math.Round(quarter));
156.
157.                Tuple<List<decimal>, List<decimal>, List<decimal>> Q1 = Tally(0, Q, MMperDay);
158.                Tuple<List<decimal>, List<decimal>, List<decimal>> Q2 = Tally(Q, Q, MMperDay);
159.                Tuple<List<decimal>, List<decimal>, List<decimal>> Q3 = Tally(Q + Q, Q, MMperDay);
160.                Tuple<List<decimal>, List<decimal>, List<decimal>> Q4 = Tally(Q + Q + Q, Q,
    MMperDay);
161.                Tuple<List<decimal>, List<decimal>, List<decimal>>[] quaters = new
    Tuple<List<decimal>, List<decimal>, List<decimal>>[4];
162.                quaters[0] = Q1;
163.                quaters[1] = Q2;
164.                quaters[2] = Q3;
165.                quaters[3] = Q4;
166.                int index = 1;
167.                foreach (var item in quaters)
168.                {// all the data should be sorted and connected in order as they where added to a list at
    the same time with
169.                 // no operations completing between them that effect order
170.                    decimal[] day = item.Item1.ToArray();
171.                    decimal[] Money = item.Item2.ToArray();
172.                    decimal[] Freq = item.Item3.ToArray();
173.                    LinearRegression money = new LinearRegression(Money, day);
```

```
174.                decimal[] valuesM = money.ReturnValues();
175.                DataRow rowM;
176.                rowM = QuatersMoney.NewRow();
177.                rowM["Quater"] = index;
178.
179.                rowM["SlopeXsq"] = valuesM[2];
180.                rowM["SlopeX"] = valuesM[1];
181.                rowM["Yintercept"] = valuesM[0];
182.                rowM["RSquared"] = valuesM[3];
183.                rowM["ProductMomentCorilation"] = valuesM[4];
184.                QuatersMoney.Rows.Add(rowM);
185.                LinearRegression Frequency = new LinearRegression(Freq, day);
186.                decimal[] valuesF = Frequency.ReturnValues();
187.                DataRow rowF;
188.                rowF = QuatersFreq.NewRow();
189.                rowF["Quater"] = index;
190.
191.                rowF["SlopeXsq"] = valuesF[2];
192.                rowF["SlopeX"] = valuesF[1];
193.                rowF["Yintercept"] = valuesF[0];
194.                rowF["RSquared"] = valuesF[3];
195.                rowF["ProductMomentCorilation"] = valuesF[4];
196.                QuatersFreq.Rows.Add(rowF);
197.
198.                index++;
199.            }
200.        }
201.    }
202.    private Tuple<List<decimal>, List<decimal>, List<decimal>> Tally(int day, int quarter,
    DataTable MMperDay)
203.    {
204.        List<decimal> Day = new List<decimal>();
205.        List<decimal> Money = new List<decimal>();
206.        List<decimal> Freq = new List<decimal>();
207.        int NumberOfDays = day + quarter;
208.        while (NumberOfDays >= MMperDay.Rows.Count)
209.            // this is to make sure this for loop doesnt exeed the bounds of the array as the quater
    might be
210.            // larger than a quater as the size of the data inpiutted during testing is flexible
211.        {
212.            NumberOfDays--;
213.        }
214.        for (int i = day; i < NumberOfDays; i++)
215.        {
216.            Day.Add((decimal)Convert.ToDecimal(MMperDay.Rows[i].ItemArray[1]));
217.            Money.Add((decimal)Convert.ToDecimal(MMperDay.Rows[i].ItemArray[2]));
218.            Freq.Add((decimal)Convert.ToDecimal(MMperDay.Rows[i].ItemArray[3]));
219.        }
220.        Tuple<List<decimal>, List<decimal>, List<decimal>> data = new Tuple<List<decimal>,
    List<decimal>, List<decimal>>(Day, Money, Freq);
221.        return data;
222.    }
223.
224.    public void FillLinregTableCustomer(DataTable Spent, DataTable MMPerTrans, ref DataTable
    Linreg)
225.    {
226.        for (int i = 0; i < Spent.Rows.Count; i++) // need to tally them into days to get accurate data
227.        { // uses Spent Datable as a store of CustomerID
228.            Console.WriteLine(Spent.Rows[i].ItemArray[1]);
229.            DataRow[] Found = MMPerTrans.Select("CustomerID = '" +
    (string)Spent.Rows[i].ItemArray[1] + "'");
230.            //*** Gets a array of rows that have the peice of data within Spent.Rows[i].ItemArray[1]
231.            // Casts to string as the Spent.Rows[i].ItemArray[1] is treated as a object that has edata
    within it
232.
233.            if (Found.Length > 1) // this is to check if the minimum amount of data required is found
    aka more than 1
```

```
234.            {
235.
236.                List<double> Money = new List<double>();
237.                Dictionary<int, double> DayToMoney = new Dictionary<int, double>();
238.                //utalizes the normalized date that makes 2016 date negative
239.                //dictionary structure to link day to the tally of money spent
240.                foreach (DataRow item in Found) //goes through each datarow and adding a tally of
    money spent
241.                {
242.
243.                    Money.Add((double)item.ItemArray[1]);
244.
245.                }
246.                // setting up data for the perpouses of the graph
247.                decimal[] xValues = new decimal[Money.Count];
248.
249.                decimal[] yValues = new decimal[Money.Count];
250.                for (int j = 0; j  < Money.Count; j ++)
251.                {
252.                    xValues[j] =j; // X values exit in relation to echother not Time as this simplifies
253.                                   // the scale of each graph so that their equations canbe presented
254.                                   // easily on the same chart with no normalization of scale
255.                    yValues[j] = (decimal)Money[j];
256.                }
257.
258.                LinearRegression Regresion = new LinearRegression(yValues, xValues);
259.                // Loads the two data axis to the LinRegression Constructor
260.                decimal[] values = Regresion.ReturnValues();
261.
262.                //Creates new data row in Datatable
263.                DataRow row;
264.                row = Linreg.NewRow();
265.                row["CustomerID"] = Spent.Rows[i].ItemArray[1];
266.                row["SlopeXsq"] = values[2];
267.                row["SlopeX"] = values[1];
268.                row["Yintercept"] = values[0];
269.                row["RSquared"] = values[3];
270.                row["ProductMomentCorilation"] = values[4];
271.                Linreg.Rows.Add(row);
272.            }
273.
274.        }
275.    }
276.    public DataTable CreateLinregTableDays(string sheetname)
277.    {
278.        DataTable LinRegtable = new DataTable(sheetname);
279.
280.
281.
282.
283.        string[] dattypes = new string[6];
284.        string[] Tablenames = new string[6];
285.        {//creating columns
286.         //
287.            DataColumn column; //MoneySpent,InvoiceDate,CustomerID
288.
289.            column = new DataColumn();
290.            column.DataType = System.Type.GetType("System.String");
291.            column.ColumnName = "Quater";
292.            column.ReadOnly = true;
293.            LinRegtable.Columns.Add(column);
294.
295.            dattypes[0] = "VARCHAR";
296.            Tablenames[0] = "CustomerID";
297.
298.            column = new DataColumn();
299.            column.DataType = System.Type.GetType("System.Double");
```

```
300.            column.ColumnName = "SlopeXsq";
301.            column.ReadOnly = true;
302.            LinRegtable.Columns.Add(column);
303.
304.            dattypes[1] = "DOUBLE";
305.            Tablenames[1] = "SlopeXsq";
306.
307.            column = new DataColumn();
308.            column.DataType = System.Type.GetType("System.Double");
309.            column.ColumnName = "SlopeX";
310.            column.ReadOnly = true;
311.            LinRegtable.Columns.Add(column);
312.
313.            dattypes[2] = "DOUBLE";
314.            Tablenames[2] = "SlopeX";
315.
316.            column = new DataColumn();
317.            column.DataType = System.Type.GetType("System.Double");
318.            column.ColumnName = "Yintercept";
319.            column.ReadOnly = true;
320.            LinRegtable.Columns.Add(column);
321.
322.            dattypes[3] = "DOUBLE";
323.            Tablenames[3] = "Yintercept";
324.
325.            column = new DataColumn();
326.            column.DataType = System.Type.GetType("System.Double");
327.            column.ColumnName = "RSquared";
328.            column.ReadOnly = true;
329.            LinRegtable.Columns.Add(column);
330.
331.            dattypes[4] = "DOUBLE";
332.            Tablenames[4] = "RSquared";
333.
334.            column = new DataColumn();
335.            column.DataType = System.Type.GetType("System.Double");
336.            column.ColumnName = "ProductMomentCorilation";
337.            column.ReadOnly = true;
338.            LinRegtable.Columns.Add(column);
339.
340.            dattypes[5] = "DOUBLE";
341.            Tablenames[5] = "ProductMomentCorilation";
342.
343.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
344.            PrimaryKeyColumns[0] = LinRegtable.Columns["CustomerID"];
345.            LinRegtable.PrimaryKey = PrimaryKeyColumns;
346.
347.            Tuple<string[], string[]> arrays = Tuple.Create(Tablenames, dattypes);
348.
349.            TableNamesDatatypes.Add(LinRegtable, arrays);
350.        }
351.        return LinRegtable;
352.    }
353.    public DataTable CreateLinregTableCustomerID()
354.    {
355.        DataTable LinRegtable = new DataTable("CustomerIDRegresion");
356.        string[] dattypes = new string[6];
357.        string[] Tablenames = new string[6];
358.        {//creating columns
359.         //
360.            DataColumn column; //MoneySpent,InvoiceDate,CustomerID
361.
362.            column = new DataColumn();
363.            column.DataType = System.Type.GetType("System.String");
364.            column.ColumnName = "CustomerID";
365.            column.ReadOnly = true;
366.            LinRegtable.Columns.Add(column);
```

```
367.
368.            dattypes[0] = "VARCHAR";
369.            Tablenames[0] = "CustomerID";
370.
371.            column = new DataColumn();
372.            column.DataType = System.Type.GetType("System.Double");
373.            column.ColumnName = "SlopeXsq";
374.            column.ReadOnly = true;
375.            LinRegtable.Columns.Add(column);
376.
377.            dattypes[1] = "DOUBLE";
378.            Tablenames[1] = "SlopeXsq";
379.
380.            column = new DataColumn();
381.            column.DataType = System.Type.GetType("System.Double");
382.            column.ColumnName = "SlopeX";
383.            column.ReadOnly = true;
384.            LinRegtable.Columns.Add(column);
385.
386.            dattypes[2] = "DOUBLE";
387.            Tablenames[2] = "SlopeX";
388.
389.            column = new DataColumn();
390.            column.DataType = System.Type.GetType("System.Double");
391.            column.ColumnName = "Yintercept";
392.            column.ReadOnly = true;
393.            LinRegtable.Columns.Add(column);
394.
395.            dattypes[3] = "DOUBLE";
396.            Tablenames[3] = "Yintercept";
397.
398.            column = new DataColumn();
399.            column.DataType = System.Type.GetType("System.Double");
400.            column.ColumnName = "RSquared";
401.            column.ReadOnly = true;
402.            LinRegtable.Columns.Add(column);
403.
404.            dattypes[4] = "DOUBLE";
405.            Tablenames[4] = "RSquared";
406.
407.            column = new DataColumn();
408.            column.DataType = System.Type.GetType("System.Double");
409.            column.ColumnName = "ProductMomentCorilation";
410.            column.ReadOnly = true;
411.            LinRegtable.Columns.Add(column);
412.
413.            dattypes[5] = "DOUBLE";
414.            Tablenames[5] = "ProductMomentCorilation";
415.
416.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
417.            PrimaryKeyColumns[0] = LinRegtable.Columns["CustomerID"];
418.            LinRegtable.PrimaryKey = PrimaryKeyColumns;
419.
420.            Tuple<string[], string[]> arrays = Tuple.Create(Tablenames, dattypes);
421.
422.            TableNamesDatatypes.Add(LinRegtable, arrays);
423.        }
424.        return LinRegtable;
425.    }
426. }
427. class DataFromSQL
428. {
429.    public DataTable FindMMPD()
430.    {
431.        {
432.            SQLCon connection = new SQLCon("MoneyMadePerDay");
433.            SQLiteDataReader rdr = connection.reader;
```

```
434.            DataTable MMPerday = new DataTable("MoneyMadePerDay");
435.
436.            DataColumn column;
437.
438.            column = new DataColumn();
439.            column.DataType = System.Type.GetType("System.Int32");
440.            column.ColumnName = "PrimaryKey";
441.            column.ReadOnly = true;
442.            MMPerday.Columns.Add(column);
443.
444.            column = new DataColumn();
445.            column.DataType = System.Type.GetType("System.Int32");
446.            column.ColumnName = "Day";
447.            column.ReadOnly = true;
448.            MMPerday.Columns.Add(column);
449.
450.            column = new DataColumn();
451.            column.DataType = System.Type.GetType("System.Double");
452.            column.ColumnName = "MoneySpent";
453.            column.ReadOnly = false;
454.            MMPerday.Columns.Add(column);
455.
456.            column = new DataColumn();
457.            column.DataType = System.Type.GetType("System.Int32");
458.            column.ColumnName = "Frequencyofsale";
459.            column.ReadOnly = false;
460.            MMPerday.Columns.Add(column);
461.
462.            column = new DataColumn();
463.            column.DataType = System.Type.GetType("System.Int32");
464.            column.ColumnName = "Year";
465.            column.ReadOnly = false;
466.            MMPerday.Columns.Add(column);
467.
468.            DataRow row;
469.
470.            while (rdr.Read())
471.            {
472.                row = MMPerday.NewRow();
473.                row["PrimaryKey"] = rdr.GetString(0);
474.                Console.WriteLine(rdr.GetInt32(1));
475.                row["Day"] = rdr.GetInt32(1);
476.                row["MoneySpent"] = rdr.GetInt32(2);
477.                row["Frequencyofsale"] = rdr.GetInt32(3);
478.                row["Year"] = rdr.GetInt32(4);
479.
480.                MMPerday.Rows.Add(row);
481.            }
482.            return MMPerday;
483.
484.        }
485.    }
486.
487.    public DataTable CreateMPPertrans()
488.    {
489.        DataTable MPPerTrans = new DataTable();
490.
491.        SQLCon connection = new SQLCon("MPPerTrans");
492.
493.        SQLiteDataReader rdr = connection.reader;
494.        {//creating columns
495.         //
496.            DataColumn column; //MoneySpent,InvoiceDate,CustomerID
497.
498.            column = new DataColumn();
499.            column.DataType = System.Type.GetType("System.String");
500.            column.ColumnName = "InvoiceNo";
```

```
            column.ReadOnly = true;
            MPPerTrans.Columns.Add(column);

            string[] Tablenames = new string[5];

            string[] dattypes = new string[5];

            dattypes[0] = "VARCHAR";
            Tablenames[0] = "InvoiceNo";

            column = new DataColumn();
            column.DataType = System.Type.GetType("System.Double");
            column.ColumnName = "MoneySpent";
            column.ReadOnly = true;
            MPPerTrans.Columns.Add(column);

            dattypes[1] = "DOUBLE";
            Tablenames[1] = "MoneySpent";

            column = new DataColumn();
            column.DataType = System.Type.GetType("System.String");
            column.ColumnName = "InvoiceDate";
            column.ReadOnly = true;
            MPPerTrans.Columns.Add(column);

            dattypes[2] = "VARCHAR";
            Tablenames[2] = "InvoiceDate";

            column = new DataColumn();
            column.DataType = System.Type.GetType("System.Int32");
            column.ColumnName = "CustomerID";
            column.ReadOnly = true;
            MPPerTrans.Columns.Add(column);

            dattypes[3] = "VARCHAR";
            Tablenames[3] = "CustomerID";
            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
            PrimaryKeyColumns[0] = MPPerTrans.Columns["InvoiceNo"];
            MPPerTrans.PrimaryKey = PrimaryKeyColumns;

            column = new DataColumn();
            column.DataType = System.Type.GetType("System.Int32");
            column.ColumnName = "NumericDate";
            column.ReadOnly = true;
            MPPerTrans.Columns.Add(column);

            dattypes[4] = "INTAGAR";
            Tablenames[4] = "NumericDate";


            column = new DataColumn();
            column.DataType = System.Type.GetType("System.Int32");
            column.ColumnName = "Year";
            column.ReadOnly = true;
            MPPerTrans.Columns.Add(column);


        }
        DataRow row;

        Console.WriteLine(rdr.FieldCount);

        while (rdr.Read())
        {

            //
StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country,InvoiceNo,MoneySpent
```

```
567.            row = MPPerTrans.NewRow();
568.
569.            row["InvoiceNo"] = rdr.GetString(0);
570.            row["MoneySpent"] = rdr.GetDecimal(1);
571.            row["InvoiceDate"] = rdr.GetString(2);
572.            row["CustomerID"] = rdr.GetString(3);
573.            row["NumericDate"] = rdr.GetInt32(4);
574.            row["Year"] = rdr.GetInt32(5);
575.
576.            MPPerTrans.Rows.Add(row);
577.        }
578.        return MPPerTrans;
579.    }
580.
581.    public DataTable CreateSpent()
582.    {
583.        DataTable Spent = new DataTable();
584.
585.        SQLCon connection = new SQLCon("Spent");
586.
587.        SQLiteDataReader rdr = connection.reader;
588.        {//creating columns
589.         //
590.            DataColumn column; //MoneySpent,InvoiceDate,CustomerID
591.
592.
593.            column = new DataColumn();
594.            column.DataType = System.Type.GetType("System.Int32");
595.            column.ColumnName = "PrimaryKey";
596.            column.ReadOnly = true;
597.            Spent.Columns.Add(column);
598.
599.
600.
601.
602.
603.            column = new DataColumn();
604.            column.DataType = System.Type.GetType("System.String");
605.            column.ColumnName = "CustomerID";
606.            column.ReadOnly = true;
607.            Spent.Columns.Add(column);
608.
609.
610.
611.
612.            column = new DataColumn();
613.            column.DataType = System.Type.GetType("System.Double");
614.            column.ColumnName = "MoneySpentOvertime";
615.            column.ReadOnly = true;
616.            Spent.Columns.Add(column);
617.
618.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
619.            PrimaryKeyColumns[0] = Spent.Columns["PrimaryKey"];
620.            Spent.PrimaryKey = PrimaryKeyColumns;
621.
622.
623.        }
624.        DataRow row;
625.
626.        while (rdr.Read())
627.        {
628.
629.            //
    StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country,InvoiceNo,MoneySpent
630.            row = Spent.NewRow();
631.            row["PrimaryKey"] = rdr.GetInt32(0);
632.            row["CustomerID"] = rdr.GetString(1);
```

```
633.            row["MoneySpentOvertime"] = rdr.GetDecimal(2);
634.
635.
636.            Spent.Rows.Add(row);
637.        }
638.        return Spent;
639.    }
640.
641.
642. }
643.
644. class program
645. {
646.
647.    static void Main(string[] args)
648.    {
649.        DataFromSQL Tabledata = new DataFromSQL();
650.
651.        Datacreated creation = new Datacreated();
652.
653.        DataTable LinregCustomer = creation.CreateLinregTableCustomerID();
654.        DataTable QuatersMoney = creation.CreateLinregTableDays("QuatersMoneyLinReg");
655.        DataTable QuatersFreq= creation.CreateLinregTableDays("QuatersFreqLinReg");
656.
657.        DataTable MMPerTrans = Tabledata.CreateMPPertrans();
658.        DataTable Spent = Tabledata.CreateSpent();
659.        DataTable MMperDay = Tabledata.FindMMPD();
660.        creation.FillLinregTableCustomer(Spent, MMPerTrans, ref LinregCustomer);
661.        creation.FillLinregTableDays(MMperDay, ref QuatersMoney,ref  QuatersFreq);
662.
663.        creation.SaveToSQL(LinregCustomer);
664.        creation.SaveToSQL(QuatersFreq);
665.        creation.SaveToSQL(QuatersMoney);
666.
667.        Console.WriteLine("Finished");
668.
669.        Console.ReadLine();
670.    }
671. }
672.
673. class LinearRegression
674. {
675.
676.    private decimal YIntersept;
677.    private decimal Slope;
678.    private decimal QuardraticSlope = 0;
679.    private decimal ProductMomentCorilation;
680.    private decimal R2;
681.
682.    private bool checkvalid(decimal[] Y, decimal[] X)
683.    {
684.        if (Y.Length > 1)
685.        {
686.            List<decimal> TestforUnique = new List<decimal>();
687.            TestforUnique.Add(Y[0]);
688.            for (int i = 1; i < Y.Length; i++)
689.            {
690.                if (!TestforUnique.Contains(Y[i])) // checks if there are at least 2 different values
691.                {
692.                    return true;
693.                }
694.            }
695.            return false;
696.        }
697.        else
698.        {
699.            return false;
```

```
700.         }
701.     }
702.
703.     public LinearRegression(decimal[] Y, decimal[] X)
704.     // we construct the object and create the two equations then compare them and assign the
    better equation to the objects attributes
705.     {
706.
707.         if (checkvalid(Y, X))
708.         {
709.             if (Y.Length != X.Length)
710.             {
711.                 Slope = 0;
712.                 YIntersept = 0;
713.                 R2 = 0;
714.                 ProductMomentCorilation = 0;
715.             }
716.             else if (Y.Length < 3)
717.             {
718.                 Slope = 0;
719.                 YIntersept = 0;
720.                 R2 = 0;
721.                 ProductMomentCorilation = 0;
722.             }
723.             else // to compare the validity between Linuear and Quadratic regression of the same data
724.             // analyzing and comparing the residuals is a better test of fit
725.             // R^2 is better for comparing between equations of different data to see which is more
    valid
726.             {
727.                 decimal[] Lindata = SimLinRegression(Y, X);
728.                 decimal[] QRdata = PolyLinRegression(Y, X);
729.
730.                 decimal LinRSq = GetRSq(Lindata, Y, X);
731.                 decimal QrRsq = GetRSq(QRdata, Y, X);
732.
733.                 decimal FitLin = GetResiduals(Lindata, Y, X);
734.                 decimal FitQuad = GetResiduals(QRdata, Y, X);
735.
736.                 if (LinRSq < 0.4m && QrRsq > 0.4m) // if R^2 is negative it means that the model fits the
    data worse than a null hypothisis
737.                                     // so is useless so this test must be completed first to see if
    comparasons are even useful
738.                 {
739.                     QuardraticSlope = QRdata[0];
740.                     Slope = QRdata[1];
741.                     YIntersept = QRdata[2];
742.                     ProductMomentCorilation = 0; // this statistic is not useful for Non linear data
743.                     R2 = QrRsq;
744.                 }
745.                 else if (LinRSq > 0.4m && QrRsq < 0.4m)
746.                 {
747.                     Slope = Lindata[1];
748.                     YIntersept = Lindata[2];
749.                     R2 = LinRSq;
750.                 }
751.                 else if (LinRSq < 0.4m && QrRsq < 0.4m) // if both are negative both models dont
    function
752.                 {
753.
754.                     Slope = 0;
755.                     YIntersept = 0;
756.                     R2 = 0;
757.                     ProductMomentCorilation = 0;
758.                 }
759.                 else if (Lindata[1] > 0 && QRdata[0] < 0) // if slope of linear is positive and the quadratic
    of QRdata is negative as
```

```
                                                    // explained in design document wont lead to good predictions
                {
                    Slope = Lindata[1];
                    YIntersept = Lindata[2];
                    R2 = LinRSq;
                }

                else if (Lindata[1] < 0 && QRdata[0] > 0) // same is true of opposite where if there is a negative general
                                                    // trend and a positive quadratic than the quadratic wont be useful at predictions
                {
                    Slope = Lindata[1];
                    YIntersept = Lindata[2];
                    R2 = LinRSq;
                }
                else if (FitLin > FitQuad) // if the quadratic is less than 0.5 than it is not a useful predictor
                {
                    Slope = Lindata[1];
                    YIntersept = Lindata[2];
                    R2 = LinRSq;
                }
                else if (FitLin < FitQuad) // if the values dont satisfy the conditions above than lastly test if the Quadrati
                                        // c R^2 is larger than Linear's
                {
                    QuardraticSlope = QRdata[0];
                    Slope = QRdata[1];
                    YIntersept = QRdata[2];
                    ProductMomentCorilation = 0;
                    R2 = QrRsq;
                }
                else if (LinRSq - QrRsq < 0.2m) // if the R^2 of the quadratic graph is within 0.2 of the R^2 of the Linear then
                                            // take the Quadratic as the better predictor
                {
                    QuardraticSlope = QRdata[0];
                    Slope = QRdata[1];
                    YIntersept = QRdata[2];
                    ProductMomentCorilation = 0;
                    R2 = QrRsq;
                }

                else// if no condiction is met go with then the quadratic model is probably not useful but the data is functionaal so the
                    // Linear version is safest to use as these have lower error rates
                {
                    Slope = Lindata[1];
                    YIntersept = Lindata[2];
                    R2 = LinRSq;
                }
            }
        }
        else
        {
            QuardraticSlope = 0;
            Slope = 0;
            YIntersept =0;
            ProductMomentCorilation = 0; // this statistic is not useful for Non linear data
            R2 =0;
        }

    }

    private decimal GetResiduals(decimal[] data,decimal[] Y, decimal[] X)
```

```
820.     {
821.
822.         decimal YIntersept = data[2];
823.         decimal Slope = data[1];
824.         decimal QuardraticSlope = data[0];
825.         decimal SquaredRegression = 0;
826.
827.         decimal Srvalue;
828.         decimal SOsvalue;
829.         for (int i = 0; i < Y.Length; i++)
830.         {
831.             decimal y = 0;
832.             if (QuardraticSlope == 0)
833.             {
834.                 y = (decimal)((decimal)Slope * (decimal)X[i]) + (decimal)YIntersept;
835.             }
836.             else
837.             {
838.                 y = ((QuardraticSlope * (X[i] * X[i]))) + (Slope * X[i]) + YIntersept;
839.             }
840.
841.
842.
843.             Srvalue = Y[i] - y;
844.
845.             Console.WriteLine(Srvalue + " Srvalue");
846.
847.             Srvalue = Srvalue * Srvalue; // squared to keep pos
848.             SquaredRegression = SquaredRegression + Srvalue;
849.
850.
851.             Console.WriteLine(SquaredRegression + " SquaredReg");
852.
853.
854.         }//gets Total squares
855.
856.         return SquaredRegression;
857.     }
858.     private decimal GetRSq(decimal[] data, decimal[] Y, decimal[] X)
859.     {
860.         //R^2 = 1- Sum SquaredRegression / Total SumofSquares
861.
862.         //for Total SumofSquares you need to find the Mean of Yvalues first
863.         decimal total = 0;
864.         for (int i = 0; i < Y.Length; i++)
865.         {
866.             total = total + Y[i];
867.         }
868.         decimal mean = total / Y.Length;
869.         // then you find the some of the predicted values
870.
871.         //the some of Squared regression is the difference between the predicted values
872.         //and the values observed then squared so they are always positive
873.         decimal SquaredRegression = 0;
874.         decimal SomeOfSquaares = 0;
875.         decimal Srvalue;
876.         decimal SOsvalue;
877.
878.         for (int i = 0; i < Y.Length; i++)
879.         {
880.             SOsvalue = Y[i] - mean;
881.
882.             SOsvalue = SOsvalue * SOsvalue;  // squared to keep pos
883.
884.             SomeOfSquaares = SomeOfSquaares + SOsvalue;
885.         }//gets Total SquaredRegression
886.
```

```
887.         SquaredRegression = GetResiduals(data,Y, X);
888.
889.         decimal Rsq = 1m - (SquaredRegression / SomeOfSquaares);
890.
891.         return Rsq;
892.     }
893.
894.     public decimal[] ReturnValues()
895.     {
896.         decimal[] Values = new decimal[5];
897.         Values[0] = YIntersept;
898.         Values[1] = Slope;
899.         Values[2] = QuardraticSlope;
900.         Values[3] = R2;
901.         Values[4] = ProductMomentCorilation;
902.         return Values;
903.     }
904.
905.
906.     private decimal[] PolyLinRegression(decimal[] Y, decimal[] X)
907.     {
908.         //We will be assuming the highest polynomial is 2 so the matrix is 3,3
909.         //***
910.         //[0 1 2]
911.         //[1 2 3]
912.         //[2 3 4]
913.         //values needed N, Sum x , Sum Y , Sum X^2, Sum X^3, Sum X^4
914.
915.         decimal sumx = 0;
916.         decimal sumy = 0;
917.         decimal sumYX = 0;
918.         decimal sumXsqY = 0;
919.         decimal SumXsq = 0;
920.         decimal SumXcb = 0;
921.         decimal SumXqu = 0;
922.
923.         for (int i = 0; i < Y.Length; i++)
924.         {
925.             sumx = sumx + X[i];
926.             sumy = sumy + Y[i];
927.
928.             sumYX = sumYX + (X[i] * Y[i]);
929.
930.             sumXsqY = sumXsqY + ((X[i] * X[i]) * Y[i]);
931.
932.             SumXsq = SumXsq + (X[i] * X[i]); ;
933.
934.             SumXcb = SumXcb + (X[i] * X[i] * X[i]);
935.
936.             SumXqu = SumXqu + (X[i] * X[i] * X[i] * X[i]);
937.         }
938.
939.         //M
940.         decimal[,] Matrix = new decimal[3, 3]
941.         {
942.             {Y.Length, sumx,SumXsq },
943.             { sumx,SumXsq,SumXcb },
944.             {SumXsq, SumXcb,SumXqu }
945.         };
946.         // Now we generate M0, M1 and M2
947.         decimal[,] M0 = new decimal[3, 3]
948.         {
949.             {Y.Length, sumx,SumXsq },
950.             { sumx,SumXsq,SumXcb },
951.             {SumXsq, SumXcb,SumXqu }
952.         };
953.
```

```
954.        M0[0, 0] = sumy;
955.        M0[0, 1] = sumYX;
956.        M0[0, 2] = sumXsqY;
957.
958.        decimal[,] M1 = new decimal[3, 3]
959.        {
960.           {Y.Length, sumx,SumXsq },
961.           { sumx,SumXsq,SumXcb },
962.           {SumXsq, SumXcb,SumXqu }
963.        };
964.        M1[1, 0] = sumy;
965.        M1[1, 1] = sumYX;
966.        M1[1, 2] = sumXsqY;
967.
968.        decimal[,] M2 = new decimal[3, 3]
969.        {
970.           {Y.Length, sumx,SumXsq },
971.           { sumx,SumXsq,SumXcb },
972.           {SumXsq, SumXcb,SumXqu }
973.        };
974.        M2[2, 0] = sumy;
975.        M2[2, 1] = sumYX;
976.        M2[2, 2] = sumXsqY;
977.
978.        // we now apply Cramers Rule --- ak = Det(Mi)/det(M)
979.        // Now we find the determiate of these functions
980.        // this has to be completed for each of the 3 metricies
981.
982.        decimal detM0 = FindDeterminate(M0);
983.        decimal detM1 = FindDeterminate(M1);
984.        decimal detM2 = FindDeterminate(M2);
985.        decimal detM = FindDeterminate(Matrix);
986.
987.        // a2 = X^2     a1 = X    a0 = Y intercept
988.
989.        decimal a0 = detM0 / detM;
990.        decimal a1 = detM1 / detM;
991.        decimal a2 = detM2 / detM;
992.
993.        decimal[] data = new decimal[3];
994.
995.        data[0] = a2;
996.        data[1] = a1;
997.        data[2] = a0;
998.        return data;
999.     }
1000.
1001.           private decimal FindDeterminate(decimal[,] Matrix)
1002.           {
1003.              decimal determiate = 0;
1004.              // |D|=
1005.              //[a1, b1, c1]
1006.              //[a2, b2, c2]
1007.              //[a3, b3, c3]
1008.              //=
1009.              //a1[b2, c2] -   b1[a2, c2] +   c1[a2, b2]
1010.              //  [b3, c3]       [a3, c3]       [a3, b3]
1011.
1012.              //a1(b2)(c3) - a1(b3)(c2) + b1(c2)(a3) - b1(a2)(c3) + c1(a2)(b3) - c1(a3)(b2)
1013.              //A                    B                    C
1014.              //I broke down this equation into 3 more digestible ones
1015.
1016.
1017.
1018.              decimal A = ((Matrix[0, 0]) * (Matrix[1, 1]) * (Matrix[2, 2])) - ((Matrix[0, 0]) * (Matrix[1,
     2]) * (Matrix[2, 1]));
```

```
1019.                    decimal B = ((Matrix[1, 0]) * (Matrix[2, 1]) * (Matrix[0, 2])) - ((Matrix[1, 0]) * (Matrix[0,
     1]) * (Matrix[2, 2]));
1020.                    decimal C = ((Matrix[2, 0]) * (Matrix[0, 1]) * (Matrix[1, 2])) - ((Matrix[2, 0]) *
     (Matrix[0, 2]) * (Matrix[1, 1]));
1021.
1022.
1023.                    determiate = A + B + C;
1024.                    return determiate;
1025.                }
1026.
1027.                private decimal[] SimLinRegression(decimal[] Y, decimal[] X)
1028.                {
1029.                    //utalizes a non matrix formula to increase proformance as this produces the same
     simple
1030.                    //Linear regression equation through a more manual math model that only works
     for
1031.                    //Simple equations
1032.                    decimal sumx = 0;
1033.                    decimal sumy = 0;
1034.                    decimal SumXsq = 0;
1035.                    decimal SumYsq = 0;
1036.                    decimal SumXY = 0;
1037.
1038.                    for (int i = 0; i < Y.Length; i++)
1039.                    {
1040.                        sumx = sumx + X[i];
1041.                        sumy = sumy + Y[i];
1042.
1043.                        SumXsq = SumXsq + (X[i] * X[i]);
1044.                        SumYsq = SumYsq + (Y[i] * Y[i]);
1045.
1046.                        SumXY = SumXY + (X[i] * Y[i]);
1047.                    }
1048.
1049.                    decimal B1 = (Y.Length * SumXY) - (sumx * sumy);
1050.                    decimal B2 = (Y.Length * SumXsq) - (sumx * sumx);
1051.                    decimal B = B1 / B2;
1052.                    //Split up equation for debugging
1053.                    //B = Coeffienent A = Y intersept
1054.
1055.                    decimal a1 = (decimal)(1m / Y.Length) * sumy;
1056.                    decimal a2 = (B * (decimal)(1m / Y.Length) * sumx);
1057.                    decimal A = a1 - a2;
1058.
1059.                    // Now we work out Product Moment corrilation
1060.
1061.                    decimal ProdMomCorl1 = Y.Length * SumXY - (sumx * sumy);
1062.                    decimal ProdMomCorl2 = ((Y.Length * SumXsq) - (sumx * sumx)) * ((Y.Length *
     SumYsq) - (sumy * sumy));
1063.                    ProdMomCorl2 =
     Convert.ToDecimal(Math.Sqrt(Convert.ToDouble(ProdMomCorl2)));
1064.                    decimal ProdMomCorl = ProdMomCorl1 / ProdMomCorl2;
1065.
1066.                    decimal[] data = new decimal[3];
1067.                    data[0] = 0;
1068.                    data[1] = B;
1069.                    data[2] = A;
1070.
1071.                    ProductMomentCorilation = ProdMomCorl;
1072.                    return data;
1073.
1074.                }
1075.
1076.            }
1077.        }
1078.
```

Data processing program:

Table abstract class – Line 593

MainSet class -Line 909

MMpertrans class – Line 434

SpentOver class – Line 328

CQI class – Line 177

MoneyPerDay class - 51

SQLInteractions class - Line12

Utilizes the SQLite library aswell as the math.net mathematics library:

```
1.  using System;
2.  using System.Data.SQLite;
3.  using System.Collections.Generic;
4.  using System.IO;
5.  using System.Data;
6.  using MathNet.Numerics.Statistics;
7.
8.
9.  namespace Interactions_with_database
10. {
11.
12.     class SQLInteractions
13.     {
14.         private SQLiteConnection conn;
15.         private string mydocumentsPath =
    Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
16.         private string filepath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) +
    "\\CustomerData.db";
17.         public void ExicuteQuery(string CommandText)
18.         {
19.             SQLiteCommand sqlite_cmd;
20.             sqlite_cmd = conn.CreateCommand();
21.             sqlite_cmd.CommandText = CommandText;
22.             Console.WriteLine(CommandText);
23.             sqlite_cmd.ExecuteNonQuery();
24.         }
25.
26.         public SQLInteractions()//works as a methood used within the exicution
27.         {
28.             try
29.             {
30.                 this.conn = new SQLiteConnection(@"Data Source =" + filepath);
31.                 conn.Open();
32.                 SQLiteCommand sqlite_cmd;
33.                 sqlite_cmd = conn.CreateCommand();
34.                 sqlite_cmd.CommandText = "Select * From CustomerData ";
35.                 sqlite_cmd.ExecuteNonQuery();
36.             }
37.             catch (Exception)
38.             {
39.                 Console.WriteLine("database file called CustomerData.db not found");
40.
41.                 System.Environment.Exit(1);
42.                 throw;
```

```
43.             }
44.         }
45.         public void Close()
46.         {
47.             conn.Close();
48.         }
49.     }
50.
51.     class MoneyPerDay : Table
52.     {
53.
54.         public override DataTable Create(DataSet DF)
55.         {
56.             Sheet = FindMMPD(DF);
57.             return Sheet;
58.         }
59.         private DataTable CheckDate(ref DataTable MMPerday, DataTable MPPerTrans, ref int dayOyear,
    int i, ref int k, ref int year)
60.         {
61.             if (year == 16)//As explained before to keep the day system consistent
62.                         //a constant - 365 is added is added if the year is 16
63.             {
64.                 dayOyear = dayOyear - 365;
65.             }
66.             DataRow[] Rows = MMPerday.Select("Day = '" + dayOyear + "'");
67.             if (Rows.Length > 0)
68.                 //If the day is already included in the Datatable Add onto the values of MoneySpent and
    Frequencyofsale
69.             {
70.                 Rows[0]["MoneySpent"] = Math.Round((double)Rows[0]["MoneySpent"]
71.                     +
    (double)MPPerTrans.Rows[i].ItemArray[(int)MPPerTrans.Columns.IndexOf("MoneySpent")], 2);
72.                 Rows[0]["Frequencyofsale"] = (int)Rows[0]["Frequencyofsale"] + 1;
73.             }
74.             else
75.             {//If the day is not part of the Datatable yet add it including all of its relevant values
76.                 DataRow row;
77.                 row = MMPerday.NewRow();
78.                 row["PrimaryKey"] = k;
79.                 k++;
80.                 row["Day"] = dayOyear;
81.                 row["MoneySpent"] = Math.Round((double)MPPerTrans.Rows[i].ItemArray[
82.                                         (int)MPPerTrans.Columns.IndexOf("MoneySpent")], 2);
83.                 row["Frequencyofsale"] = 1;
84.                 row["Year"] = year;
85.                 MMPerday.Rows.Add(row);
86.             }
87.             return MMPerday;
88.         }
89.         private DataTable FindMMPD(DataSet df)
90.         {
91.             {
92.                 DataTable MMPerday = new DataTable("MoneyMadePerDay");
93.
94.                 DataTable MPPerTrans = df.Tables["MPPerTrans"];
95.
96.                 DataColumn column;
97.
98.                 string[] dattypes = new string[5];
99.                 string[] Tablenames = new string[5];
100.
101.                column = new DataColumn();
102.                column.DataType = System.Type.GetType("System.Int32");
103.                column.ColumnName = "PrimaryKey";
104.                column.ReadOnly = true;
105.                MMPerday.Columns.Add(column);
106.
```

```
107.            dattypes[0] = "VARCHAR";
108.            Tablenames[0] = "PrimaryKey";
109.
110.
111.            column = new DataColumn();
112.            column.DataType = System.Type.GetType("System.Int32");
113.            column.ColumnName = "Day";
114.            column.ReadOnly = true;
115.            MMPerday.Columns.Add(column);
116.
117.            dattypes[1] = "INTAGAR";
118.            Tablenames[1] = "Day";
119.
120.
121.
122.            column = new DataColumn();
123.            column.DataType = System.Type.GetType("System.Double");
124.            column.ColumnName = "MoneySpent";
125.            column.ReadOnly = false;
126.            MMPerday.Columns.Add(column);
127.
128.            dattypes[2] = "DECIMAL";
129.            Tablenames[2] = "MoneySpent";
130.
131.
132.            column = new DataColumn();
133.            column.DataType = System.Type.GetType("System.Int32");
134.            column.ColumnName = "Frequencyofsale";
135.            column.ReadOnly = false;
136.            MMPerday.Columns.Add(column);
137.
138.            dattypes[3] = "INTAGAR";
139.            Tablenames[3] = "Frequencyofsale";
140.
141.            column = new DataColumn();
142.            column.DataType = System.Type.GetType("System.Int32");
143.            column.ColumnName = "Year";
144.            column.ReadOnly = false;
145.            MMPerday.Columns.Add(column);
146.
147.            dattypes[4] = "INTAGAR";
148.            Tablenames[4] = "Year";
149.
150.            datatypes.addValues(dattypes);
151.            tablenames.addValues(Tablenames);
152.
153.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
154.            PrimaryKeyColumns[0] = MMPerday.Columns["PrimaryKey"];
155.            MMPerday.PrimaryKey = PrimaryKeyColumns;
156.
157.            {
158.               int k = 0;
159.               for (int i = 0; i < MPPerTrans.Rows.Count; i++)
160.               {
161.                  string date = "";
162.                  date =
    (string)MPPerTrans.Rows[i].ItemArray[(int)MPPerTrans.Columns.IndexOf("InvoiceDate")];
163.                  Tuple<int, int> DayandYear = GetDate(date);
164.                  //Uses a Method from the abstract Table class
165.                  int dayOyear = DayandYear.Item1;
166.                  int year = DayandYear.Item2;
167.                  CheckDate(ref MMPerday, MPPerTrans, ref dayOyear, i, ref k, ref year);
168.               }
169.            }
170.
171.
172.            return MMPerday;
```

```
173.
174.            }
175.        }
176.    }
177.    class CQI : Table
178.    {
179.
180.        public override DataTable Create(DataSet df)
181.        {
182.            DataTable CQItable = AddData(df);
183.
184.            Sheet = CQItable;
185.
186.            return CQItable;
187.        }
188.        private DataTable AddData(DataSet df)
189.        {
190.            //fill columns
191.            DataColumn column;
192.            DataTable Spent = df.Tables["Spent"];
193.            DataTable MPPerTrans = df.Tables["MPPerTrans"];
194.
195.            DataTable CQIData = new DataTable("CQI");
196.
197.            string[] dattypes = new string[6];
198.            string[] Tablenames = new string[6];
199.
200.            column = new DataColumn();
201.            column.DataType = System.Type.GetType("System.Int32");
202.            column.ColumnName = "PrimaryKey";
203.            column.ReadOnly = true;
204.            CQIData.Columns.Add(column);
205.
206.            dattypes[0] = "VARCHAR";
207.            Tablenames[0] = "PrimaryKey";
208.
209.
210.
211.            column = new DataColumn();
212.            column.DataType = System.Type.GetType("System.Int32");
213.            column.ColumnName = "CustomerID";
214.            column.ReadOnly = true;
215.            CQIData.Columns.Add(column);
216.
217.            dattypes[1] = "VARCHAR";
218.            Tablenames[1] = "CustomerID";
219.
220.
221.
222.            column = new DataColumn();
223.            column.DataType = System.Type.GetType("System.Double");
224.            column.ColumnName = "MoneySpentOvertime";
225.            column.ReadOnly = true;
226.            CQIData.Columns.Add(column);
227.
228.            dattypes[2] = "DECIMAL";
229.            Tablenames[2] = "MoneySpentOvertime";
230.
231.
232.
233.            column = new DataColumn();
234.            column.DataType = System.Type.GetType("System.Double");
235.            column.ColumnName = "StandardDiv";
236.            column.ReadOnly = true;
237.            CQIData.Columns.Add(column);
238.
239.            dattypes[3] = "DECIMAL";
```

```
240.        Tablenames[3] = "StandardDiv";
241.
242.
243.
244.        column = new DataColumn();
245.        column.DataType = System.Type.GetType("System.Int32");
246.        column.ColumnName = "Frequency";
247.        column.ReadOnly = true;
248.        CQIData.Columns.Add(column);
249.
250.        dattypes[4] = "INTAGAR";
251.        Tablenames[4] = "Frequency";
252.
253.
254.
255.        column = new DataColumn();
256.        column.DataType = System.Type.GetType("System.Double");
257.        column.ColumnName = "CQI";
258.        column.ReadOnly = true;
259.        CQIData.Columns.Add(column);
260.
261.        dattypes[5] = "DECIMAL";
262.        Tablenames[5] = "CQI";
263.
264.
265.
266.        datatypes.addValues(dattypes);
267.        tablenames.addValues(Tablenames);
268.
269.
270.
271.
272.        DataColumn[] PrimaryKeyColumns = new DataColumn[1];
273.        PrimaryKeyColumns[0] = CQIData.Columns["PrimaryKey"];
274.        CQIData.PrimaryKey = PrimaryKeyColumns;
275.
276.        DataRow row;
277.
278.          for (int i = 0; i < Spent.Rows.Count; i++)
279.          {
280.            //runs through the Datatable looking for every index of the ID
281.            //loads all of those indexes and gets all the money per trans
282.            //gets SD of that then sends it to SD
283.            //solves problems of SD and ID not maching
284.            row = CQIData.NewRow();
285.
286.            DataRow[] Rows = MPPerTrans.Select("CustomerID = '" +
    Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("CustomerID")] + "'");
287.            double[] DataforSD = new double[Rows.Length];
288.            int y = 0;
289.            foreach (DataRow item in Rows)
290.            {
291.              DataforSD[y] =
    (double)item.ItemArray[(int)MPPerTrans.Columns.IndexOf("MoneySpent")];
292.            //Fills a array with the Money Spent of each transaction the CustomerID is present in
293.            }
294.
295.            double SD = 0;
296.            SD = this.GetSD(DataforSD);
297.            //Utilizes the GetSD method of the abstract Table class.
298.
299.            row["PrimaryKey"] = i;
300.            row["CustomerID"] =
    (int)Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("CustomerID")];
301.            row["MoneySpentOvertime"] =
    (double)Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("MoneySpentOvertime")];
302.            row["StandardDiv"] = (double)SD;
```

```
303.                row["Frequency"] = Rows.Length;
304.                double h = 0;
305.                if (SD == 0) // Makes sure the system does not attempt to divide by 0
306.                {
307.                    h = 0;
308.                }
309.                else//Money spent overtime * Frequence / SD
310.                {
311.                    double j =
    (double)Spent.Rows[i].ItemArray[(int)Spent.Columns.IndexOf("MoneySpentOvertime")] * Rows.Length;
312.                    h = j / SD;
313.
314.                }
315.                row["CQI"] = h;
316.
317.                CQIData.Rows.Add(row);
318.            }
319.
320.
321.
322.        return CQIData;
323.    }
324.
325.
326.
327. }
328. class SpentOver : Table
329. {
330.
331.    public override DataTable Create(DataSet DataSet)
332.    {
333.        DataTable Spent = new DataTable("Spent");
334.        IDSpentOvertime(ref Spent, DataSet);
335.
336.        Sheet = Spent;
337.
338.        return Spent;
339.    }
340.
341.    private void IDSpentOvertime(ref DataTable Spent, DataSet DataSet)
342.    {
343.        {
344.            Dictionary<int, double> MoneySpent = new Dictionary<int, double>();
345.            TallyMP(ref MoneySpent, DataSet);
346.
347.            DataColumn column;
348.
349.            string[] dattypes = new string[3];
350.
351.            string[] Tablenames = new string[3];
352.
353.
354.            column = new DataColumn();
355.            column.DataType = System.Type.GetType("System.Int32");
356.            column.ColumnName = "PrimaryKey";
357.            column.ReadOnly = true;
358.            Spent.Columns.Add(column);
359.
360.            dattypes[0] = "INTAGAR";
361.            Tablenames[0] = "PrimaryKey";
362.
363.
364.
365.            column = new DataColumn();
366.            column.DataType = System.Type.GetType("System.Int32");
367.            column.ColumnName = "CustomerID";
368.            column.ReadOnly = true;
```

```
369.            Spent.Columns.Add(column);
370.
371.            dattypes[1] = "VARCHAR";
372.            Tablenames[1] = "CustomerID";
373.
374.
375.
376.            column = new DataColumn();
377.            column.DataType = System.Type.GetType("System.Double");
378.            column.ColumnName = "MoneySpentOvertime";
379.            column.ReadOnly = true;
380.            Spent.Columns.Add(column);
381.
382.            dattypes[2] = "DOUBLE";
383.            Tablenames[2] = "MoneySpentOvertime";
384.            int i = 1;
385.
386.
387.            datatypes.addValues(dattypes);
388.            tablenames.addValues(Tablenames);
389.
390.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
391.            PrimaryKeyColumns[0] = Spent.Columns["PrimaryKey"];
392.            Spent.PrimaryKey = PrimaryKeyColumns;
393.            DataRow row;
394.            foreach (KeyValuePair<int, double> kk in MoneySpent)
395.            {
396.                Console.WriteLine(" yeeet  " + kk.Key + "    " + kk.Value);
397.                row = Spent.NewRow();
398.                row["PrimaryKey"] = i;
399.                row["CustomerID"] = kk.Key;
400.                row["MoneySpentOvertime"] = kk.Value;
401.
402.                Spent.Rows.Add(row);
403.                i++;
404.            }
405.
406.        }
407.
408.    }
409.    private static Dictionary<int, double> TallyMP(ref Dictionary<int, double> MoneySpent, DataSet
    DataSet)
410.    {
411.        //tally for money spent by each client in the same time
412.        DataTable table = DataSet.Tables["MotherDF"];
413.
414.        for (int i = 1; i < table.Rows.Count; i++)///can be Changed into operations class in table
415.        {
416.            double counter = 0;
417.            if
    (MoneySpent.ContainsKey((int)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("CustomerID")]))
418.            {
419.            }
420.            else
421.            {
422.                DataRow[] ContainsInv = table.Select("CustomerID = '" +
    table.Rows[i].ItemArray[(int)table.Columns.IndexOf("CustomerID")] + "'");
423.                foreach (DataRow item in ContainsInv)
424.                {
425.                    counter = counter + (double)item[table.Columns.IndexOf("MoneySpent")];
426.
427.                }
428.                MoneySpent.Add((int)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("CustomerID")],
    counter);
429.            }
430.        }
431.        return MoneySpent;
```

```
432.        }
433.    }
434.    class MMpertrans : Table
435.    {
436.
437.        public override DataTable Create(DataSet DataSet)
438.        {
439.            DataTable MPPerTrans = new DataTable("MPPerTrans");
440.            ADDcolumstoMP(ref MPPerTrans);
441.
442.            FILLrowsMP(ref MPPerTrans, DataSet);
443.            Sheet = MPPerTrans;
444.
445.            return MPPerTrans;
446.        }
447.        private void FILLrowsMP(ref DataTable MPPerTrans, DataSet DataSet)
448.        {
449.            DataTable table = DataSet.Tables["MotherDF"]; // extracts the main Datatable from the dataset
450.            DataRow row;
451.
452.            List<string> Invoices = new List<string>();
453.            for (int i = 1; i < (int)table.Rows.Count; i++)
454.            {
455.                double Counter = 0;
456.                if (Invoices.Contains((string)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("InvoiceNo")]))
457.                    //if the rows invoice number is already part of the list
458.                {
459.                }
460.                else
461.                {
462.                    Invoices.Add((string)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("InvoiceNo")]);
463.                    //add it to list so ths following operation only occurs once
464.
465.                    DataRow[] ContainsInv = table.Select(string.Format("{0} LIKE '%{1}%'", "InvoiceNo",
466.
    (string)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("InvoiceNo")]));
467.                    //Selects all datarows within the main dataset where the invoice number is = to the rows
    invoice number
468.                    foreach (DataRow item in ContainsInv)
469.                    {
470.                        //goes through each invoice number and tallies their value
471.                        Counter = Counter + (double)item[table.Columns.IndexOf("MoneySpent")];
472.                    }
473.                    row = MPPerTrans.NewRow();
474.                    row["InvoiceNo"] = table.Rows[i].ItemArray[(int)table.Columns.IndexOf("InvoiceNo")];
475.                    row["MoneySpent"] = Math.Round(Counter);
476.
477.                    string invoicedate =
    (string)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("InvoiceDate")];
478.                    row["InvoiceDate"] = invoicedate;
479.                    row["CustomerID"] = table.Rows[i].ItemArray[(int)table.Columns.IndexOf("CustomerID")];
480.                    //utilizes a Table class method
481.                    Tuple<int, int> DateandYear =
    GetDate((string)table.Rows[i].ItemArray[(int)table.Columns.IndexOf("InvoiceDate")]);
482.
483.                    if (DateandYear.Item2 == 16)
484.                        // Because there are years 17 and 16 and linear regression algorithims are to be taken
    place between these periods there has to be
485.                        // a way of representing this so 0 is the moment of newyears, -1 is the 31st of december
    2016 and 1 is january 1st of 2017
486.                    {
487.                        int adjustedday = DateandYear.Item1 - 365;
488.                        DateandYear = new Tuple<int, int>(adjustedday, DateandYear.Item2);
489.                    }
490.
491.                    row["NumericDate"] = DateandYear.Item1;
492.                    row["Year"] = DateandYear.Item2;
```

```
493.
494.                MPPerTrans.Rows.Add(row);
495.            }
496.
497.        }
498.
499.    }
500.
501.    private void ADDcolumstoMP(ref DataTable MPPerTrans)
502.    {
503.        {//creating columns
504.
505.            DataColumn column; //MoneySpent,InvoiceDate,CustomerID
506.
507.            column = new DataColumn();
508.            column.DataType = System.Type.GetType("System.String");
509.            column.ColumnName = "InvoiceNo";
510.            column.ReadOnly = true;
511.            MPPerTrans.Columns.Add(column);
512.
513.            string[] Tablenames = new string[6];
514.            string[] dattypes = new string[6];
515.
516.            dattypes[0] = "VARCHAR";
517.            Tablenames[0] = "InvoiceNo";
518.
519.            column = new DataColumn();
520.            column.DataType = System.Type.GetType("System.Double");
521.            column.ColumnName = "MoneySpent";
522.            column.ReadOnly = true;
523.            MPPerTrans.Columns.Add(column);
524.
525.            dattypes[1] = "DOUBLE";
526.            Tablenames[1] = "MoneySpent";
527.
528.            column = new DataColumn();
529.            column.DataType = System.Type.GetType("System.String");
530.            column.ColumnName = "InvoiceDate";
531.            column.ReadOnly = true;
532.            MPPerTrans.Columns.Add(column);
533.
534.            dattypes[2] = "VARCHAR";
535.            Tablenames[2] = "InvoiceDate";
536.
537.            column = new DataColumn();
538.            column.DataType = System.Type.GetType("System.Int32");
539.            column.ColumnName = "CustomerID";
540.            column.ReadOnly = true;
541.            MPPerTrans.Columns.Add(column);
542.
543.            dattypes[3] = "VARCHAR";
544.            Tablenames[3] = "CustomerID";
545.
546.            column = new DataColumn();
547.            column.DataType = System.Type.GetType("System.Int32");
548.            column.ColumnName = "NumericDate";
549.            column.ReadOnly = true;
550.            MPPerTrans.Columns.Add(column);
551.
552.            dattypes[4] = "INTAGAR";
553.            Tablenames[4] = "NumericDate";
554.
555.            column = new DataColumn();
556.            column.DataType = System.Type.GetType("System.Int32");
557.            column.ColumnName = "Year";
558.            column.ReadOnly = true;
559.            MPPerTrans.Columns.Add(column);
```

```
560.
561.            dattypes[5] = "INTAGAR";
562.            Tablenames[5] = "Year";
563.
564.            datatypes.addValues(dattypes);
565.            tablenames.addValues(Tablenames);
566.
567.
568.            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
569.            PrimaryKeyColumns[0] = MPPerTrans.Columns["InvoiceNo"];
570.            MPPerTrans.PrimaryKey = PrimaryKeyColumns;
571.
572.        }
573.    }
574. }
575.
576. class l1st
577. {
578.    string[] tableNames = new string[8];
579.
580.    public void addValues(string[] values)
581.    {
582.        for (int i = 0; i < values.Length; i++)
583.        {
584.            tableNames[i] = values[i];
585.        }
586.    }
587.
588.    public string[] returnvalues()
589.    {
590.        return this.tableNames;
591.    }
592. }
593. abstract class Table
594. {
595.
596.    protected DataTable Sheet;
597.    protected const int amountofdataRows =40000;
598.   protected string filepath = Environment.GetFolderPath(
599.                            Environment.SpecialFolder.MyDocuments) + "\\CustomerData.db";
600.
601.    protected l1st datatypes = new l1st();
602.    protected l1st tablenames = new l1st();
603.
604.    virtual public DataTable Create(DataSet M )
605.    {
606.        return null;
607.    }
608.     public Tuple<int,int> GetDate(string date)
609.     {
610.        //This class converts a string date format of date-month-year into numerical formats
611.        //using substrings.
612.        //the date format date-month-year
613.
614.        int day = Convert.ToInt32(date.Substring(0, date.IndexOf('-')));
615.        string s = date.Substring(date.IndexOf('-') + 1, 2);
616.        Console.WriteLine(s);
617.        int month = Convert.ToInt32(s);
618.        int year = Convert.ToInt32(date.Substring(date.LastIndexOf('-') + 1));
619.
620.        int dayOyear = day + (30) * (month - 1);
621.
622.        Tuple<int, int> DayandYear = new Tuple<int, int>(dayOyear, year);
623.
624.        return DayandYear;
625.     }
626.    public void getUQID(string name, string Columnname, string Col2)
```

```
627.     {
628.         // takes in the parameters name, Columnname and Col2, the first 2 work similar to the measures
     of spread document however Col2 is the
629.         // name of the rows column we want the system to store if a row's Columnname is greater than
     the Upper quartile
630.         //gets the top quartile of customers
631.         StreamReader reader = new StreamReader("Mesure Of Spread Document\\" +
     Sheet.TableName + "\\  " + Columnname + " " + amountofdataRows);
632.         string UQT = "";
633.         for (int y = 0; y < 8; y++)
634.         {
635.             UQT = reader.ReadLine();
636.         }//puts the reader on the right line
637.         reader.Close();
638.         double UQ = Convert.ToDouble(UQT);
639.
640.         if (!Directory.Exists("top25"))
641.         {
642.             Directory.CreateDirectory("top25");
643.         }
644.         if (!Directory.Exists("top25\\" + Sheet.TableName))
645.         {
646.             Directory.CreateDirectory("top25\\" + Sheet.TableName);
647.
648.         }
649.         File.Create("top25\\" + Sheet.TableName + "\\  " + Sheet.TableName + " " +
     amountofdataRows).Dispose();
650.         using (StreamWriter writer = new StreamWriter("top25\\" + Sheet.TableName + "\\  " +
     Columnname + " " + amountofdataRows))
651.         {
652.             List<double> top25ID = new List<double>();
653.             bool Freq = false;
654.             try
655.             {
656.                 string checkF = Columnname.Substring(0, 9);
657.                 if (checkF == "Frequency")
658.                 {
659.                     Freq = true;
660.                 }
661.             }
662.             catch
663.             {
664.             }
665.             for (int i = 0; i < Sheet.Rows.Count; i++)
666.             {
667.                 //Frequency
668.                 if (Freq == true)// stored as int so has to be cast as int value
669.                 {
670.                     double TestColumn =
     (int)this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(Columnname)];
671.                     Math.Round(TestColumn);
672.
673.                     if (TestColumn > UQ)
674.                     {
675.                         int j =
     Convert.ToInt32(this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(Col2)]);
676.
677.                         top25ID.Add(j);
678.                     }
679.                 }
680.                 else
681.                 {
682.                     double TestColumn =
     (double)this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(Columnname)];
683.                     Math.Round(TestColumn);
684.
685.                     if (TestColumn > UQ)
```

```
686.                    {
687.                        int j =
    Convert.ToInt32(this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(Col2)]);
688.
689.                        top25ID.Add(j);
690.                    }
691.                }
692.            }
693.            SQLInteractions link = new SQLInteractions();
694.            foreach (double item in top25ID)
695.            {
696.                writer.WriteLine(item);
697.            }
698.            string CreateTable2 = "drop table if exists " + Sheet.TableName + "Top25" + Columnname;
699.
700.            link.ExicuteQuery(CreateTable2);
701.
702.            string CreateTable = "CREATE TABLE" + " " + Sheet.TableName + "Top25"+ Columnname +
    "( Top25 INTAGAR);";
703.
704.            link.ExicuteQuery(CreateTable);
705.
706.            foreach (double item in top25ID)
707.            {
708.                string data = "INSERT INTO" + " " + Sheet.TableName + "Top25"+ Columnname + "(Top25)
    VALUES(" + item+");";
709.
710.                link.ExicuteQuery(data);
711.            }
712.        }
713.    }
714.
715.
716.
717.    public void CreateMesureOfSpreadDocument(string Name, string Columnname)
718.    {
719.        List<double> ValuesTouse = new List<double>();
720.        for (int i = 0; i < this.Sheet.Rows.Count; i++)
721.        {
722.
    ValuesTouse.Add((double)Convert.ToDouble(this.Sheet.Rows[i].ItemArray[this.Sheet.Columns.IndexOf(
    Columnname)]));
723.        }
724.        double[] CompiledData = ValuesTouse.ToArray();
725.        double[] MMSpread = new double[8];
726.        MesuresOfSpread(ref CompiledData, ref MMSpread);
727.        if (!Directory.Exists("Mesure Of Spread Document"))
728.        {
729.            Directory.CreateDirectory("Mesure Of Spread Document");
730.        }
731.        if (!Directory.Exists("Mesure Of Spread Document\\" + Sheet.TableName))
732.        {
733.            Directory.CreateDirectory("Mesure Of Spread Document\\" + Sheet.TableName);
734.        }
735.        File.Create("Mesure Of Spread Document\\" + Sheet.TableName + "\\  " + Columnname + " " +
    amountofdataRows).Dispose();
736.
737.            using (StreamWriter writer = new StreamWriter("Mesure Of Spread Document\\" +
    Sheet.TableName +
738.                                                  "\\  " + Columnname + " " + amountofdataRows))
739.            {
740.                foreach (double item in MMSpread)
741.                {
742.                    writer.WriteLine(item);
743.                }
744.                writer.WriteLine();
745.                writer.WriteLine("1. mean 2. SD 3. IQR 4.Min - Max 5-7. Quartiles");
```

```
746.                writer.WriteLine(Columnname);
747.            }
748.        }
749.
750.        private double[] MesuresOfSpread(ref double[] CompiledData, ref double[] VMesuresofspread)
751.        {
752.
753.            double Mean = Statistics.Mean(CompiledData); //mnean that ignors top 0.1 and bottem 0.1
754.
755.            double SD = GetSD(CompiledData); //SD that ignores the many null values
756.
757.            double IQR = Statistics.InterquartileRange(CompiledData); //IQR that ignores the many null
    values
758.
759.            double Max = Statistics.Maximum(CompiledData); //IQR that ignores the many null values
760.
761.            double Min = Statistics.Minimum(CompiledData);
762.
763.            double LQ = Statistics.LowerQuartile(CompiledData);
764.
765.            double Medium = Statistics.Median(CompiledData);
766.
767.            double UQ = Statistics.UpperQuartile(CompiledData);
768.
769.            VMesuresofspread[0] = Mean;
770.            VMesuresofspread[1] = SD;
771.            VMesuresofspread[2] = IQR;
772.            VMesuresofspread[3] = Max;
773.            VMesuresofspread[4] = Min;
774.            VMesuresofspread[5] = LQ;
775.            VMesuresofspread[6] = Medium;
776.            VMesuresofspread[7] = UQ;
777.
778.            return VMesuresofspread;
779.
780.        }
781.
782.
783.        protected double GetSD(double[] SDData)
784.        {
785.            //SD formula = Squarroot ( (sum X^2)/size - (sum X /size)^2)
786.            double sum = 0;
787.            double sumSq = 0;
788.
789.            foreach (double item in SDData)
790.            {
791.                sum = sum + item;
792.
793.                sumSq = sumSq + (item * item);
794.            }
795.            return Math.Sqrt((sumSq / SDData.Length) - ((sum / SDData.Length) * (sum / SDData.Length)));
796.
797.        }
798.        public void SaveData()
799.        {
800.            SQLInteractions link = new SQLInteractions();
801.            //create table if not exists TableName (col1 typ1, ..., colN typN)
802.            //drop table if exists TableName
803.
804.            string CreateTable2 = "drop table if exists " + Sheet.TableName;
805.
806.            link.ExicuteQuery(CreateTable2);
807.
808.
809.            string CreateTable = "CREATE TABLE" + " " + Sheet.TableName + "( " +
    MakeColumsreadyForSQL(false) + " );"; ;
810.
```

```
            link.ExicuteQuery(CreateTable);
            InsertData();


        }
        private string MakeColumsreadyForSQL(bool Rows) // has a bool to see if this is for the insert or
    not as the data types are used to create the table but not when inserting
        {

            string[] Datatypes = datatypes.returnvalues();

            string data = "";
            for (int i = 0; i < Sheet.Columns.Count; i++)
            {
                if (i == (Sheet.Columns.Count - 1))
                {
                    if (Rows == true)
                    {
                        data = data + Sheet.Columns[i].ColumnName + " ";
                    }
                    else
                    {
                        data = data + Sheet.Columns[i].ColumnName + " " + Datatypes[i];
                    }
                }
                else
                {
                    if (Rows == true)
                    {
                        data = data + Sheet.Columns[i].ColumnName + "," + " ";
                    }
                    else
                    {
                        data = data + Sheet.Columns[i].ColumnName + " " + Datatypes[i] + ", ";
                    }
                }
            }
            return data;
        }
        private SQLiteConnection CreateSQLconnection()//works as a methood used within the exicution
        {
            SQLiteConnection ll = new SQLiteConnection(@"Data Source =" + filepath);
            ll.Open();
            return ll;
        }


        private string MakeRowsReadyforSQL(int i)
        {
            string data = "";

            for (int j = 0; j < Sheet.Columns.Count; j++)
            {

                if (j == (Sheet.Columns.Count - 1))
                {

                    data = data + Sheet.Rows[i].ItemArray[j];
                }
                else
                {
                    try
                    {
                        string date = (string)Sheet.Rows[i].ItemArray[j];

                        if (date.Contains('-'))
                        {
```

```
877.                    date = "\'" + date + "\'";
878.                    data = data + date + ", ";
879.                }
880.                else
881.                {
882.                    data = data + Sheet.Rows[i].ItemArray[j] + ", ";
883.                }
884.
885.            }
886.            catch (Exception)
887.            {
888.                data = data + Sheet.Rows[i].ItemArray[j] + ", ";
889.
890.            }
891.
892.        }
893.    }
894.    return data;
895.  }
896.  private void InsertData()
897.  {
898.    SQLInteractions link = new SQLInteractions();
899.    for (int i = 0; i < Sheet.Rows.Count; i++)
900.    {
901.
902.        string InsertText = "INSERT INTO" + " " + Sheet.TableName + "( " +
    MakeColumsreadyForSQL(true) + ")" + " VALUES(" + MakeRowsReadyforSQL(i) + "); ";
903.
904.        link.ExicuteQuery(InsertText);
905.    }
906.  }
907.
908. }
909. class MainSet : Table
910. {
911.   public DataSet dataSet = new DataSet();
912.   private void PreliminaryCleaning()
913.   {//Cleans known errors before they have to be loaded and unloaded into the datatable
914.     SQLInteractions Removal = new SQLInteractions();
915.     string text = "delete from CustomerData where InvoiceNo=' ' OR InvoiceNo IS NULL;";
916.     Removal.ExicuteQuery(text);
917.     text = "delete from CustomerData where StockCode=' ' OR StockCode IS NULL;";
918.     Removal.ExicuteQuery(text);
919.     text = "delete from CustomerData where Description=' ' OR Description IS NULL;";
920.     Removal.ExicuteQuery(text);
921.     text = "delete from CustomerData where Quantity=' ' OR Quantity IS NULL;";
922.     Removal.ExicuteQuery(text);
923.     text = "delete from CustomerData where InvoiceDate=' ' OR InvoiceDate IS NULL;";
924.     Removal.ExicuteQuery(text);
925.     text = "delete from CustomerData where UnitPrice=' ' OR UnitPrice IS NULL;";
926.     Removal.ExicuteQuery(text);
927.     text = "delete from CustomerData where CustomerID=' ' OR CustomerID IS NULL;";
928.     Removal.ExicuteQuery(text);
929.     text = "delete from CustomerData where Country=' ' OR Country IS NULL;";
930.     Removal.ExicuteQuery(text);
931.     text = "delete from CustomerData where InvoiceNo=' ' OR InvoiceNo IS NULL;";
932.     Removal.ExicuteQuery(text);
933.     text = "delete from CustomerData where StockCode='M';";
934.     Removal.ExicuteQuery(text);
935.     text = "delete from CustomerData where Quantity<0;";
936.     Removal.ExicuteQuery(text);
937.
938.     Removal.Close();
939.   }
940.   public MainSet()
941.   {
942.     PreliminaryCleaning();
```

```
943.         string command1 = "SELECT * FROM CustomerData LIMIT " + amountofdataRows;
944.         SQLiteConnection connection = CreateSQLconnection();
945.         using var cmd = new SQLiteCommand(command1, connection);
946.         SQLiteDataReader rdr = cmd.ExecuteReader();
947.
948.         System.Data.DataTable table = new DataTable("MotherDF");
949.
950.         ADDcolums(ref table);
951.
952.         ADDrows(ref rdr, ref table);
953.
954.         dataSet.Tables.Add(table);
955.
956.          remove(table);
957.
958.     }
959.
960.     private void remove(DataTable table)
961.     {
962.         List<int> rows = new List<int>();
963.         using (StreamReader reader = new StreamReader("BrokenRows " + amountofdataRows))
964.         //uses the + to make sure it uses the correct text file
965.         {
966.             reader.ReadLine();
967.             while (reader.ReadLine() != null)
968.             {
969.                 rows.Add(Convert.ToInt32(reader.ReadLine()));
970.                 // reads all lines from broken rows adding them to a list
971.             }
972.         }
973.         //Deletes the File afterwards to ensue ame primary keys are not read and deleted twice
974.         File.Delete("BrokenRows " + amountofdataRows);
975.         SQLInteractions Removal = new SQLInteractions();
976.         foreach (int item in rows)
977.         {
978.             string ComandText = "DELETE FROM CustomerData WHERE Primarykey = " + item + ";";
979.             Console.WriteLine(ComandText);
980.             Removal.ExicuteQuery(ComandText);
981.         }
982.         //This is removing all traces of broken rows that remain in the dataset that might break the
     system overall
983.
984.         dataSet.Tables.Remove(table);
985.         //This Removes the table and re creates the table with the new cleaned dataset
986.
987.         string command1 = "SELECT * FROM CustomerData LIMIT " + amountofdataRows;
988.         SQLiteConnection connection = CreateSQLconnection();
989.         using var cmd = new SQLiteCommand(command1, connection);//needs to be redone
990.         //write a cross table tabulation
991.         SQLiteDataReader rdr = cmd.ExecuteReader();
992.
993.         DataTable ReMadeTable = new DataTable("MotherDF");
994.
995.         ADDcolums(ref ReMadeTable);
996.
997.         ADDrows(ref rdr, ref ReMadeTable);
998.
999.         //Re creates datatable with the cleaned data
1000.
1001.            dataSet.Tables.Add(ReMadeTable);
1002.
1003.        }
1004.
1005.        private void ADDcolums(ref DataTable table)//works
1006.        {
1007.            //primary key is deafulted as first //PK
1008.            // Create a new DataTable.
```

```
1009.        System.Data.DataSet dataSet;
1010.        // Declare variables for DataColumn and DataRow objects.
1011.        DataColumn column;
1012.
1013.
1014.        // Create new DataColumn, set DataType,
1015.        // ColumnName and add to DataTable.
1016.        column = new DataColumn();
1017.        column.DataType = System.Type.GetType("System.Int32");
1018.        column.ColumnName = "PrimaryKey";
1019.        column.ReadOnly = true;
1020.        table.Columns.Add(column);
1021.
1022.        column = new DataColumn();
1023.        column.DataType = System.Type.GetType("System.String");
1024.        column.ColumnName = "StockCode";
1025.        column.ReadOnly = true;
1026.        // Add the Column to the DataColumnCollection.
1027.        table.Columns.Add(column);
1028.
1029.        // Create second column.
1030.        column = new DataColumn();
1031.        column.DataType = System.Type.GetType("System.String");
1032.        column.ColumnName = "Description";
1033.        column.ReadOnly = true;
1034.        // Add the column to the table.
1035.        table.Columns.Add(column);
1036.
1037.        column = new DataColumn();
1038.        column.DataType = System.Type.GetType("System.Int32");
1039.        column.ColumnName = "Quantity";
1040.        column.ReadOnly = true;
1041.        table.Columns.Add(column);
1042.
1043.        column = new DataColumn();
1044.        column.DataType = System.Type.GetType("System.String");
1045.        column.ColumnName = "InvoiceDate";
1046.        column.ReadOnly = true;
1047.        table.Columns.Add(column);
1048.
1049.        column = new DataColumn();
1050.        column.DataType = System.Type.GetType("System.Double");
1051.        column.ColumnName = "UnitPrice";
1052.        column.ReadOnly = true;
1053.        table.Columns.Add(column);
1054.
1055.        column = new DataColumn();
1056.        column.DataType = System.Type.GetType("System.Int32");
1057.        column.ColumnName = "CustomerID";
1058.        column.ReadOnly = true;
1059.        table.Columns.Add(column);
1060.
1061.        column = new DataColumn();
1062.        column.DataType = System.Type.GetType("System.String");
1063.        column.ColumnName = "Country";
1064.        column.ReadOnly = true;
1065.        table.Columns.Add(column);
1066.
1067.        column = new DataColumn();
1068.        column.DataType = System.Type.GetType("System.String");
1069.        column.ColumnName = "InvoiceNo";
1070.        column.ReadOnly = true;
1071.        table.Columns.Add(column);
1072.
1073.        column = new DataColumn();
1074.        column.DataType = System.Type.GetType("System.Double");
1075.        column.ColumnName = "MoneySpent";
```

```
            column.ReadOnly = true;
            table.Columns.Add(column);

            DataColumn[] PrimaryKeyColumns = new DataColumn[1];
            PrimaryKeyColumns[0] = table.Columns["id"];
            table.PrimaryKey = PrimaryKeyColumns;

            // Make the ID column the primary key column.
        }

        private SQLiteConnection CreateSQLconnection()//works as a methood used within the exicution
        {
            try
            {
                SQLiteConnection ll = new SQLiteConnection(@"Data Source =" + filepath);
                ll.Open();
                return ll;
            }
            catch (Exception)
            {
                Console.WriteLine("database file called CustomerData.db not found");

                System.Environment.Exit(1);
                throw;
            }

        }

        private void ADDrows(ref SQLiteDataReader rdr, ref DataTable table)
        {
            int y = 0;
            if (!File.Exists("BrokenRows " + amountofdataRows))
            {
                File.Create("BrokenRows " + amountofdataRows).Dispose();
            }
            using (StreamWriter sw = new StreamWriter("BrokenRows " + amountofdataRows))
            {
                sw.WriteLine("Broken rows - rowkeys");
                DataRow row;
                while (rdr.Read())
                {
                    try
                    {
                        y++;
                        //
StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country,InvoiceNo,MoneySpent
                        row = table.NewRow();
                        row["PrimaryKey"] = y;
                        row["StockCode"] = rdr.GetString(1);
                        row["Description"] = rdr.GetString(2);
                        row["Quantity"] = rdr.GetInt32(3);
                        string dashdate = rdr.GetString(4);
                        string formatted = GetFormat(dashdate);
                        //Formats the 2-Aug-16 date into a pure numerical date
                        row["InvoiceDate"] = formatted;
                        row["UnitPrice"] = rdr.GetDouble(5);
                        row["CustomerID"] = rdr.GetInt32(6);
                        row["Country"] = rdr.GetString(7);
                        row["InvoiceNo"] = rdr.GetInt32(0);
                        row["MoneySpent"] = Math.Round((rdr.GetInt32(3) * rdr.GetDouble(5)), 2);
                        table.Rows.Add(row);



                    }
                    catch (Exception)
                    {
```

```
1142.                    sw.WriteLine(rdr.GetString(0));
1143.                }
1144.
1145.            }
1146.        }
1147.    }
1148.    private string GetFormat(string date)
1149.    {
1150.        string day =date.Substring(0, date.IndexOf('-'));
1151.        string s = date.Substring(date.IndexOf('-') + 1, 3);
1152.        string month = "";
1153.
1154.        string year = date.Substring(date.LastIndexOf('-') + 1);
1155.        switch (s)
1156.        {
1157.            case "Jan":
1158.                month = "01";
1159.
1160.                break;
1161.            case "Feb":
1162.                month = "02";
1163.
1164.                break;
1165.            case "Mar":
1166.                month = "03";
1167.                // code block
1168.                break;
1169.            case "Apr":
1170.                month = "04";
1171.                // code block
1172.                break;
1173.            case "May":
1174.                month = "05";
1175.                // code block
1176.                break;
1177.            case "Jun":
1178.                month = "06";
1179.                // code block
1180.                break;
1181.            case "Jul":
1182.                month = "07";
1183.                // code block
1184.                break;
1185.            case "Aug":
1186.                month = "08";
1187.                // code block
1188.                break;
1189.            case "Sep":
1190.                month = "09";
1191.                // code block
1192.                break;
1193.            case "Oct":
1194.                month = "10";
1195.                // code block
1196.                break;
1197.            case "Nov":
1198.                month = "11";
1199.                // code block
1200.                break;
1201.            case "Dec":
1202.                month = "12";
1203.                // code block
1204.                break;
1205.            default:
1206.                // code block
1207.                break;
1208.        }
```

```
1209.            string FormatDate = day + "-" + month + "-" + year;
1210.            return FormatDate;
1211.        }
1212.    }
1213.
1214.    class Program
1215.    {
1216.
1217.        static void Main(string[] args)
1218.        {
1219.
1220.            MainSet Mainset = new MainSet();
1221.
1222.            MMpertrans Pertrans = new MMpertrans();
1223.
1224.            Mainset.dataSet.Tables.Add(Pertrans.Create(Mainset.dataSet));
1225.
1226.            SpentOver Spent = new SpentOver();
1227.
1228.            Mainset.dataSet.Tables.Add(Spent.Create(Mainset.dataSet));
1229.
1230.            CQI CQItable = new CQI();
1231.
1232.            Mainset.dataSet.Tables.Add(CQItable.Create(Mainset.dataSet));
1233.
1234.            MoneyPerDay MMPD = new MoneyPerDay();
1235.
1236.            Mainset.dataSet.Tables.Add(MMPD.Create(Mainset.dataSet));
1237.
1238.            Pertrans.SaveData();
1239.
1240.            Spent.SaveData();
1241.
1242.            CQItable.SaveData();
1243.
1244.            MMPD.SaveData();
1245.
1246.            Pertrans.CreateMesureOfSpreadDocument("MPPerTrans", "MoneySpent");
1247.            Spent.CreateMesureOfSpreadDocument("Spent", "MoneySpentOvertime");
1248.            CQItable.CreateMesureOfSpreadDocument("CQI", "CQI");
1249.            CQItable.CreateMesureOfSpreadDocument("CQI", "Frequency");
1250.            MMPD.CreateMesureOfSpreadDocument("MoneyMadePerDay", "MoneySpent");
1251.            MMPD.CreateMesureOfSpreadDocument("MoneyMadePerDay", "Frequencyofsale");
1252.
1253.            Pertrans.getUQID("MPPerTrans", "MoneySpent", "CustomerID");
1254.            Spent.getUQID("Spent", "MoneySpentOvertime", "CustomerID");
1255.            CQItable.getUQID("CQI", "CQI", "CustomerID");
1256.            CQItable.getUQID("CQI", "Frequency", "CustomerID");
1257.            MMPD.getUQID("MoneyMadePerDay", "MoneySpent", "Day");
1258.            MMPD.getUQID("MoneyMadePerDay", "Frequencyofsale", "Day");
1259.
1260.            Console.WriteLine("stuff");
1261.
1262.            Console.ReadLine();
1263.        }
1264.    }
1265. }
```