Sustainability Scorecard Program

A Program that simulates the scoring of Scope 3 Upstream and Downstream Transport and Distribution Emissions

Aryam Rege | A Level Computer Science Non-Examined Assessment|
Candidate Number: 1177, Reading School Centre Number: 51337,

Contents

Analysis 4
Problem description 4
Research – Current system and key knowledge 6
Scope 1,2,3 emissions 6
Emissions Formulae 7
Scoring the Emissions Totals 10
Comparing Total Emissions calculated in different months to show a comparison 10
How the program will be presented 10
Programming language choice 10
Objectives of the program 12
1) Objectives for the starting program: 12
2) Objectives in distance-based method calculation: 12
3) Objectives in fuel-based method calculation 13
4) Objectives for scoring the emissions value (applies for both distance and fuel methods): 13
5) Objectives for plotting a graph for comparison: 14
Linear Congruential Generator Algorithm 14
Stein's Algorithm 15
Linear Regression Algorithm 17
Least Squared Method 17
Gradient Descent 18
Merge Sort Algorithm 20
Critical Path 22
Overall Design 22
Scoring model: 23
Graph plotting Model: 23
Front End: 24
Design 25
Stage 1 – Designing databases – Objective 2) ,3) and 4) 25
Stage 2 - Calculating the total emissions – Objective 2) & 3) 30
Stage 3 - Scoring the emissions – Objective 4) 34
Stage 4 – Building the Front End – Objective 1) 40
Stage 5 – Plotting the Graph – Objective 5) 47
Test Design Plan 50
Libraries used 53

SQL statements used 53
Implementation 57
Calculating Total Emissions 57
Scoring Code 60
Scoring Model 60
Code to Plot Graphs 62
Graphing Code 65
Front End Code 68
Techniques used in Implementation 74
Testing 75
Test 1 75
Test 2 76
Test 3 78
Test 4 80
Test 5 83
Test 6 88
Test 7 89
Test 8 96
Test 9 98
Test 10 99
Test 11 100
Test 12 101
Test 13 106
Test 14 114
Test 15 116
Test 16 117
Test 17 118
Test 18 120
Test 19 122
Test 20 124
Test 21 125
Test 22 126
Test 23 128
Test 24 128
Conclusion on Testing 129

Evaluation 130
Client Feedback 130
Evaluation against the Objectives 131
Objective 1 131
Objective 2 131
Objective 3 132
Objective 4 132
Objective 5 133
Conclusion 134
Full Python Solution 135
Create_Database_Emissions 135
Test_Database_Emissions 139
Calculating_Emissions 140
Front_End (Run by User) 142
Scoring_Emissions 150
Graph_Emissions 157
Bibliography – Given in blue in Documentation 161

Analysis

PROBLEM DESCRIPTION

Over the last decade, the necessity for firms to prove that they are sustainable has become crucial if they want to survive in the long-term. These days, more investors are considering sustainability a strong factor for investment so having a good sustainability record is extremely beneficial for firms. Due to this reason, the company my father works for has an interest in the sustainability field. My father has requested me to create a bespoke software package that he can use to help improve the efficiency of his work. To see how I can help my father, I held a discussion with him; in which he detailed the problems he faces at present and what he would like my program to achieve:

My father needs to carry out many calculations to determine total emissions (mainly in transport and distribution of goods). However, he must do this manually as there are no computer programs to do this for him. This takes a lot of unnecessary time and effort preventing him from doing more important aspects of his work. He must use his experience to make a judgement on whether the emission totals are of a good standard or not. A computer program that does this reduces the time taken and remove the subjectivity of this judgement as the computer program can quantitatively assess the emission totals.

Another aspect of my father's work requires him to give ways how to reduce these emission totals; mainly through using different vehicles. This prompts my father to spend a long time scrolling through different vehicles to see which ones would lead to a reduction in total emissions. A program may be able to do this faster and more efficiently as the program will be able to spot every single more efficient vehicle from a large database.

Sometimes, he takes all this information and puts them in MS PowerPoint presentations to be delivered as part of his work. He would like to have some visual diagrams to help illustrate a point he wishes to make in a presentation. Some of the visuals offered by MS PowerPoint don't necessarily reflect what he wishes to convey. A program may be able to provide the diagrams that he needs specifically rather than some general-purpose software such as MS PowerPoint.

Due to the issues given above, my father formally sent me the email below which summarises his needs:

Dear Aryam,

Hope you are safe and well.

As discussed with you, I would like for you to create a sustainability scorecard program to ensure that I can save time in some areas of my work.

I would like to have a program where the user can enter data on Distribution Logistics about the company; data such as distance travelled, fuel used etc. From this data, the total emissions of the company will then be calculated.

I would like to have this done over a defined period of time at regular time intervals of one month. After the data has been entered, the required calculations would be carried out by the program. I would then like a Line or Bar graph to be drawn of all the total emissions over the given months.

Lastly, if the resultant emissions are not within the required range, it would help to see how I could reduce those emissions and improve the performance on Sustainability criteria.

Please don't hesitate to get back to me for any clarifications.

Kind Regards
Rajeev Rege
Alumnus Software
+44 7786 312787

There was one detail my father omitted and that was how the program should be presented: as a website or a simple command line program. My guess was that he wished for a command line program because only my father is going to use it. However, I still asked for clarification from him. The correspondence of this is shown below:

Dear Rajeev,

Thanks for stating to me your requirements. I will have this completed in due course.

I was wondering if you have any particular user interface requirements or whether a simple terminal is acceptable for the scorecard?

If there are any more features you would like in the scorecard, please let me know.

Dear Aryam,
A simple terminal would be fine.

Best regards
Rajeev

As shown, my guess was correct. Now, I know what my father requires and why. From this, I completed some research and based on my father's requirements and my research, I created objectives for my program, showing what it should achieve.

RESEARCH – CURRENT SYSTEM AND KEY KNOWLEDGE

Over my research, I aim to find out more about sustainability emissions. I will do this by first going onto the Carbon Disclosure Project (CDP) website. CDP is a leading organisation in creating sustainability scorecards and many firms regularly give their emissions data to CDP for them to calculate the emissions and score them. This is done using a questionnaire to start with (CDP, 2021) and this was the starting point of my research.

After looking through the questionnaire, I realized that the data given by firms in the questionnaire is very generic and only gives the total emissions given off.

(C7.1a) Break down your total gross global Scope 1 emissions by greenhouse gas type and provide the source of each used greenhouse warming potential (GWP).

Greenhouse gas	Scope 1 emissions (metric tons of CO2e)	GWP Reference
CO2	13039620	IPCC Fourth Assessment Report (AR4 - 100 year)
CH4	221607	IPCC Fourth Assessment Report (AR4 - 100 year)
SF6	111988	IPCC Fourth Assessment Report (AR4 - 100 year)
Other, please specify (CH4+N2O)	53890	IPCC Fourth Assessment Report (AR4 - 100 year)

(C7.3c) Break down your total gross global Scope 1 emissions by business activity.

Activity	Scope 1 emissions (metric tons CO2e)
Generating Facilities	8845145
Cogeneration	4156241
Gas Distribution: CH4 leakage	221607
Distribution networks: SF6 releases	11988
Non-generation facilities	26233
Renewables generation	13100
Corporate	52791

Tesco	Climate Change 2020	2020	Submitted	A-

It does not actually give the calculations to calculate these emissions. This is because CDP does all these calculations internally and it does not disclose how it scores their scorecards. I cannot also get data from CDP to work with either due to the data costing an unreasonable amount of money to purchase. CDP disclose these questionnaires for people to see, but the actual detailed data that I need is kept confidential. This is the same as the calculations. As a result of this, I must do some more research as to how these emissions are calculated in detail and come up with a way to score the emissions given off.

Scope 1,2,3 emissions

I tried to find out more about the different types of emissions that a firm can give off (Plana Earth, Unknown) .I have decided that I will focus on scoring one of these categories in extensive depth. The 3 categories are:

- **Scope 1 Emissions** – Emissions that are given off on a direct level from owned or controlled sources. E.g. Company Vehicles, emissions given off from company facilities etc.
- **Scope 2 Emissions** – Indirect Emissions given off from the production of purchased electricity, heat, or steam.

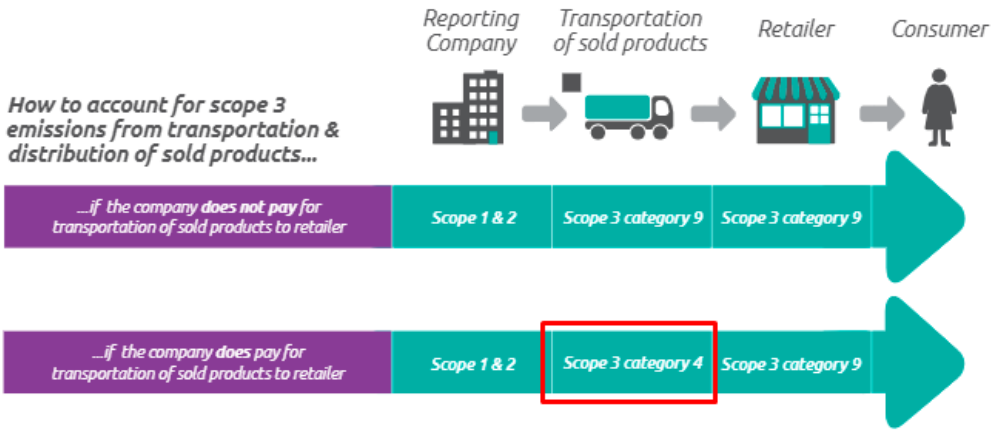
- **Scope 3 Emissions** – All other indirect emissions that occur in a company's value chain (Activities done to manufacture and distribute a good or service). E.g. Transport and Distribution(T&D, upstream and downstream), Business Travel, Investments, fuel, purchased goods and services, capital goods, Waste generated in Operations, Employee commuting, Upstream and downstream Leased Assets, Processing of sold products, Use of Sold Products, End-of-life treatment of sold products, Franchises etc.

Through my research I found that my client's requirements fell in the Scope 3 emissions category so this was the area I needed to research further. The screenshot below shows the different types of scope 3 emissions. Number 4 (the red box) is what my client is interested in. Hence, I needed to see this section in more depth.

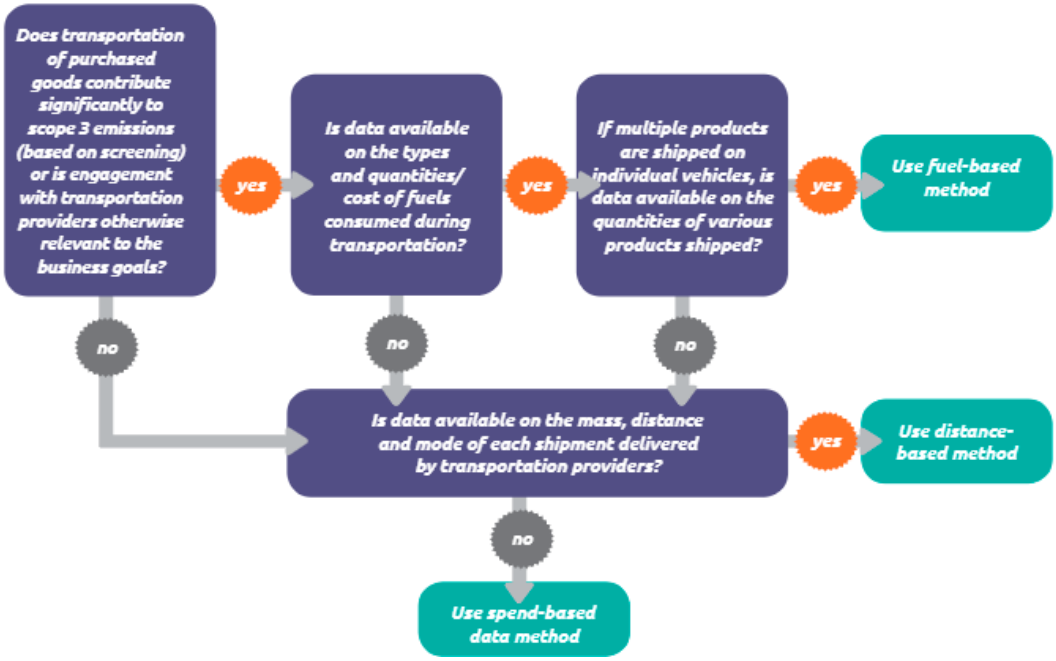
1: Purchased Goods and Services	20
2: Capital Goods	36
3: Fuel- and Energy-Related Activities Not Included in Scope 1 or Scope 2	38
4: Upstream Transportation and Distribution	49
5: Waste Generated in Operations	72
6: Business Travel	81
7: Employee Commuting	87
8: Upstream Leased Assets	94
9: Downstream Transportation and Distribution	102
10: Processing of Sold Products	106
11: Use of Sold Products	113
12: End-of-Life Treatment of Sold Products	125
13: Downstream Leased Assets	128
14: Franchises	130
15: Investments	136

Emissions Formulae

Now, I had to do some research on how to calculate the emissions total for upstream transport and distribution. The Greenhouse Gas (GHG) Protocol has given an extensive PDF (GHG Protocol, 2013) on how to calculate these scope 3 emissions for a particular firm. After an informal discussion with my client, he said that he was more particularly interested in the transportation of sold products to a retailer (the red box).



The Scope 3 Category 4 gives information on how to calculate the emissions. There are 3 main methods to calculate the upstream T & D distributions of a firm. The GHG explains when to use each method and this is explained below in the following decision tree.



I will be going through and explaining each method in depth and I will then give a judgement on which formulas I will be using and why.

Fuel-Based method

This method involves determining the quantity of fuel consumed and applying the appropriate emission factor for that fuel. The emission factor of a fuel is the average emission rate of that fuel when it is consumed (Clim'Foot, Unknown). Its unit is CO_{2eq} /kwh (kilowatt hour) which essentially is a unit that is used to convert the emissions of a fuel into the number of kilograms of carbon dioxide released.

This is done by comparing a gas's (e.g. methane) Global Warming Potential (GWP) with that of 1kg of carbon dioxide allowing you to find a gas's GWP in terms of kg of carbon dioxide. This allows you to compare emissions for different fuels as when fuels are used, they will emit different proportions of different gases and the emission factors of these gases can be added up appropriately in their respective proportions to give the emission factor of the fuel. The value of the emission factor of a fuel need not be calculated as these emission factors are found in public databases online.

Here is the formula to calculate the emissions:

CO_2e emissions from transportation =

sum across fuel types:
Σ (quantity of fuel consumed (liters) × emission factor for the fuel (e.g., kg CO_2e/liter))
+
sum across grid regions:
Σ (quantity of electricity consumed (kWh) × emission factor for electricity grid (e.g., kg CO_2e/kWh))
+
sum across refrigerant and air-conditioning types:
Σ (quantity of refrigerant leakage × global warming potential for the refrigerant (e.g., kg CO_2e))

Quantities of fuel consumed (liters) =

sum across transport steps:
Σ (total distance travelled (e.g., km) × fuel efficiency of vehicle (e.g., liters/km))

In this situation, the electricity consumed will be 0 as I am looking at the transportation of vehicles and their journey from warehouses to supermarkets. The refrigerant is the chemical used to provide air conditioning inside the vehicle and the refrigerants that I am going to consider are dependent on the vehicles I am considering and will be explained in the design stage.

Distance-Based Method

In this method, the mass of the good transported is multiplied by the distance travelled by the vehicle multiplied by the emission factor of that vehicle. This method is particularly useful if I do not have fuel consumption data meaning that it is very useful for my project. There is only one such formula:

CO_2e emissions from transportation =

sum across transport modes and/or vehicle types:
= Σ (mass of goods purchased (tonnes or volume) × distance travelled in transport leg (km)
× emission factor of transport mode or vehicle type (kg CO_2e/tonne or volume/km))

Spend-based Method

This calculates the emissions from an economic standpoint using the amount spent on fuel and the relevant emission factor. The formula is given below.

CO_2e emissions from transportation =

sum across transport modes and/or vehicle types:
Σ (amount spent on transportation by type ($)
× relevant EEIO emission factors per unit of economic value (kg CO_2e/$))

However, this method will not be suitable for me because I have been unsuccessful in obtaining data for the fuel expenditure of certain firms, so I won't be able to test this formula even if I do include it. This is not too much of a problem because GHG only advises the use of this method if and only if the previous 2 methods explained are unsuitable. This is not going to happen in my father's case.

Conclusion on calculating emissions

I have decided to include in my program the fuel-based method and the distance-based method for my calculations. This can be presented as an option to the user depending on the data that they have available to them. This allows the model to be used for a wider range of users as users have an option to calculate emissions based on fuel consumption or on distance travelled.

The current system used by CDP is a slight variation on the formulas explained above and I will try to use this system in my program. However, I will try to score the sustainability scorecard using my own derived method to make my program unique to the current system available.

SCORING THE EMISSIONS TOTALS

To score the total emissions calculated, I will use a Normal Distribution and I will see where the emissions total falls on the Normal Distribution. Since I am considering 2 calculation methods, I plan to have 2 Normal Distributions, one for each calculation method. To do this, I need to determine the parameters for the Normal Distribution. The parameters I need are the population mean and the population standard deviation (Measures the spread of the data). I plan to determine these by creating a separate scoring model using a Monte Carlo Method (Wikipedia, Unknown) . This scoring model involves randomly determining inputs to the emissions formulae stated above and calculates the total emissions with these randomised inputs. This is done many times to get a vast list of total emissions that should give a good reflection of what total emissions can be. Then, I will calculate the Mean and Standard Deviation of this emissions data. To determine the randomised inputs, I will be using a Linear Congruential Generator (Wikipedia, Unknown) and Stein's Algorithm (Wikipedia, Unknown) along with upper bounds that are determined for each input of the formulae. See the Design Stage 4 for the in-depth details.

COMPARING TOTAL EMISSIONS CALCULATED IN DIFFERENT MONTHS TO SHOW A COMPARISON

My client wished for the emissions values to be illustrated and a graph and he also wanted to see how emissions can be reduced further. I will be doing both requirements in 1 graphing model. This Graphing Model first will illustrate these total emissions on a graph showing a trend line of all emissions. This shows my client whether emissions have been generally rising or falling over time. Then, I will calculate the new total emissions for each other vehicle in my database, adding any vehicles that produce lower emissions to a list that is outputted to the user in ascending order of emissions. If no vehicles are more efficient, then the user is asked to minimise the distance travelled; maybe by reducing the number of junctions in the transportation journey. This shows the user how emissions can be improved like my client requested. The algorithms I will use here are Gradient Descent (Wikipedia, Unknown) , Least Squares (Wikipedia, Unknown) and Merge Sort

HOW THE PROGRAM WILL BE PRESENTED

As my client isn't too concerned about the user interface, I will create a simple terminal for my father to enter data for as this was his requirement as stated in the client emails above. However, I will be using Object-Orientated-Programming to make the front program on the terminal as both the fuel-based and distance-based methods have many similar components in their formulae.

PROGRAMMING LANGUAGE CHOICE

I have a wide range of programming language choices available to me. Below shows a table of different programming languages and their pros and cons (if any) in relation to my project.

Programming language	Pros	Cons
Java	1. It is an Object Orientated Language. The Front End is coded using Object Orientated Principles so Java might make implementing this part of my program easier	1. Java is slow. My program will be plotting different types of graphs, and this may take lots of time. I do not wish for Java's performance to add to this as my father wants this program to save time in his work.

		2. I will be using procedural programming to implement my scoring model and the formulae to calculate total emissions. Java may not be suited for this.
R	1. Very good for mathematical calculations and hence, can be more efficient when running my scoring model and the algorithms to determine the grade of each emission total. 2. Can interact with databases very well which can be useful when creating a database of vehicles for my program to consider.	1. I am not familiar with R, and it has a reputation for being initially difficult to learn. 2. R is slow leading to the same issues as Java
C#	1. Object Oriented Language 2. Memory-efficient which may reduce the chances of my program from crashing which makes it easier for my father to use.	1. I am not comfortable in using C# in a project of this magnitude.
PHP	1. Has many libraries within it allowing me to reduce the time needed writing my solution allowing me to spend more time adjusting my program to my father's needs	1. PHP is mainly useful for web development. However, my father wishes for a Command Line Program rendering PHP not very useful for this project.
Python	1. Very flexible. I can do both procedural and object orientated programming easily in python which allows me to spend more time on the actual solution to my father's problem. 2. Has a vast array of libraries which allows me to implement all my databases in python as well.	1. Very Thorough testing of my program is required due to the risk of runtime errors occurring.

For my program, I have chosen python to be the language my program is written in because it is very flexible. I can use it to create my databases, create the normal distribution for scoring and carry out linear regression using python. Hence, everything can be written in 1 language making it easier for me to manage. I am happy to spend more time testing it as I feel as I will have to do this independently of the language that I choose. This makes python the most advantageous language so hence, I will use it.

Also, I will be used sqlite3, a library in python to use SQL to create my database (SQlite, 2004)

OBJECTIVES OF THE PROGRAM

Based on the discussion I had with my father as well as with my own research, I have come up with the following formal objectives for my sustainability scorecard program:

1) Objectives for the starting program:

1. **The Program should present to the user an option whether they want to enter distance-based data or fuel-based data**

   a) Validation should be present for this user input.

2. **The user should also be allowed to enter how many months one would like to enter data for.**

   a) The maximum number of months one can enter data for must be 6

   b) The minimum number of months one can enter data for must be 2

   c) Validation should be present for this user input.

2) Objectives in distance-based method calculation:

1. **Allow the user to enter the postcodes of junctions in the transportation journey of the product.**

   a) The user should be able to enter the number of postcodes to be entered.

   b) The number of postcodes entered should be between 2 and 6 inclusive.

   c) Validation on the actual postcodes entered should be present.

2. **The program should calculate the distance in km between each adjacent pair of postcodes entered by the user.**

3. **The type and brand of vehicle used in the transportation leg should also be entered by the user.**

   a) Validation should be present for this user input.

4. **Allow the user to enter the total mass of goods being transported over the month.**

   a) Validation should be present for this user input.

5. **The program should from the data entered by the user, calculate the total upstream T & D emissions for a particular product over a month-long period.**

6. **This should be repeated for as many months as the user has entered**

7. **After the data for all the months have been entered, all the month data for each month will be displayed to the user.**

3) Objectives in fuel-based method calculation

1. **Allow the user to enter the postcodes of junctions in the transportation journey of the product.**

   a) The user should be able to enter the number of postcodes to be entered.

   b) The number of postcodes entered should be between 2 and 6 inclusive.

   c) Validation on the actual postcodes entered should be present.

2. **The program should calculate the distance in km between each adjacent pair of postcodes entered in by the user.**

3. **Allow the user to enter the brand of vehicle used in the transportation leg.**

   a) Validation should be present for this user input.

4. **Allow the user should be asked to enter the quantity of refrigerant leaked by the vehicle.**

   a) Validation should be present for this user input.

5. **From the data entered by the user, the program should be able to calculate the total upstream T & D emissions for a particular product over a month-long period.**

6. **This should be repeated for as many months as the user has entered.**

7. **After the data for all the months have been entered, all the month data for each month will be displayed to the user.**

4) Objectives for scoring the emissions value (applies for both distance and fuel methods):

1. **The program should carry out a Monte Carlo Method to create a dataset for which I can create a normal distribution for. The following variables should be randomised in the Monte Carlo Method:**

   a) Distance travelled (Both calculation methods)

   b) Vehicle used (Both calculation methods)

   c) Mass of goods purchased (Distance-based method only)

   d) Quantity of refrigerant leaked (Fuel-based method only)

   e) All of a, b, c, and d should be used to calculate a randomly generated total emissions value.

   f) a, b, c, d, and e should be carried out 1000 times

2. **The program should determine parameters from which it can create 2 normal distributions, one for the fuel-based method and one for the distance-based method.**

a) The Mean of the 1000 total emission values calculated in 4.1 should be determined.

b) The Standard Deviation of the 1000 total emission values calculated in 4.1 should be determined.

3. **The program should then use the total emission value calculated in Objective 2 or 3 (depending on whether fuel-based or distance-based method is used) and grade it.**

   a) The Program should determine grade boundaries based on the 2 Normal distributions created in 4.2. The Grades will be A,B,C,D and E.

   b) These grade boundaries should be fairly determined. The mean of the normal distributions should reflect a B or C Grade.

4. **All scores should be presented to the user along with the graph of the appropriate normal distribution depending on the emissions method used by the user.**

   a) Each total emission value calculated in objective 2 or 3 should be shown on the normal distribution graph.

   b) The Grade boundaries should be present on the Normal Distribution.

5) Objectives for plotting a graph for comparison:

1. **The program should get the emission scores calculated from each of the months and plot the data points on a grid.**

   a) A trend line of best fit should be determined by the program and should be as accurate as possible.

   b) This trend line should be plotted on the graph along with the data points.

2. **The user should then be told about the trajectory for their emissions and when their emissions are rising or falling over time.**

3. **Independent of whether emissions are rising or falling, the program should list vehicles that give off less emissions.**

   a) The vehicles should be presented to the user in ascending order of emissions.

   b) If no other vehicles are more emissions-efficient, the user should be told to minimise the distance travelled in the journey travelled by the vehicle.

I showed these objectives to my father, and initially, he wanted objective 4.1 to be more specific. After I did this I showed it to him again and he felt that these objectives reflected his requirements very well.

LINEAR CONGRUENTIAL GENERATOR ALGORITHM

Linear Congruential Generator (LCG) is a type of Pseudo-Random Number Generator (PRNG) which I will use in my scoring model to randomly assign values to each component of my formulae. It does this through defining a recurrence relation as shown below:

$$X_{n+1} = (aX_n + c)\ mod\ m$$

Where a mod b represents the remainder obtained from calculating a/b. For the LCG to be an effective PRNG, the period of the recurrence relation (how many terms exist before the same sequence repeats) needs to be as large as possible. For this to happen, I need to follow the Hull-Dobell Theorem. This states that the period of the recurrence relation is m if and only if:

- Gcd(m,c) = 1. Gcd is Greatest Common Divisor
- a-1 is divisible by all prime factors of m
- a-1 is divisible by 4 if m is divisible by 4.

The last 2 conditions are relatively easy to meet. However, the first condition is slightly harder, and I need to implement another algorithm to determine an appropriate c. This is where Stein's Algorithm comes in (see heading below). I will use Stein's algorithm to find the smallest c such that gcd(m,c) = 1 where c > 1. I have made c > 1 so that there are no fixed variables in my LCG to ensure that the period is as large as possible. I will compute the recurrence relation a set number of times and the final number will be my randomised value. To make my number of iterations as random as possible, I will be using the datetime library. The last 3 digits of the number of microseconds in the second that the function was called will be used as the number of iterations. The value of m will be the upper bounds for each component of emissions formulae (or a multiple of them). This ensures that my randomly generated values are in the correct range. The value of a will be determined by finding all the prime factors of m and multiplying 1 instance of each prime factor together. To meet condition 3 (if necessary), I will multiply by an extra 2 to get a factor of 4. The design will show the algorithms used to determine this. My starting x value x_0 will be the 4^{th} last digit.

Stein's Algorithm

Stein's algorithm is a recursive algorithm that determines whether 2 numbers are coprime to each other. It is a lot faster compared to the Euclidean Algorithm as it cuts down both the input values by as many factors of 2 possible before determining the greatest possible divisor. Also, it uses binary shifts in the algorithm making the algorithm even faster. The procedure is shown below:

1. If both a,b are 0, then the gcd is 0

2. If one of a and b is 0, then the gcd is the other number (if 2 numbers are a and 0 then gcd=a and the same applies for b and 0).

3. If a and b are both even, then gcd(a,b) = 2*gcd(a/2,b/2)

4. If a is even and b is odd then gcd(a,b) = gcd(a/2,b). If a is odd and b is even then gcd(a,b) = gcd(a,b/2)

5. If both a and b are odd, then gcd(a,b) = gcd(|a-b|/2,b) where |x| = x if x >= 0 and |x| = -x if x < 0.

6. Repeat steps 3-5 until a=b, or until a=0. In both cases, the gcd = 2^k*b where k is the number of factors of 2 that a and b share.

The pseudocode is shown below:

```
SUB find_coprime(divisor)
     SUB gcd(a,b)
          IF a == b THEN
```

```
            RETURN a
        ENDIF
        IF a == 0 THEN
            RETURN b
        ENDIF
        IF (NOT a AND 1)== 1 THEN
            IF (a AND 1) == 1 THEN
                RETURN gcd(a>>1,b)
            ELSE
                RETURN (gcd(a>>1,b>>1)<<1
            ENDIF
        IF (NOT b AND 1) == 1 THEN
            RETURN gcd(a,b>>1)
        ENDIF
        IF a > b THEN
            RETURN gcd((a-b)>> 1,b)
        RETURN gcd((b-a)>>1,a)
    ENDSUB
    found = FALSE
    i = 2
    WHILE found = FALSE
        hcf = gcd(i,divisor)
        IF hcf == 1 THEN
            found = TRUE
        ELSE
            i = i + 1
        ENDIF
    ENDWHILE
ENDSUB
```

LINEAR REGRESSION ALGORITHM

A Linear Regression algorithm will be used to determine if emissions are in general rising or falling over time. The gradient of the line formed will be used to determine whether emissions are rising or falling and accordingly, the correct message will be outputted to the user along with a list of more efficient vehicles.

There are 2 main methods to do this. One is the Least Squares Method and the other is using Gradient Descent in Machine Learning. Both methods yield very accurate results so as a result, I have decided to implement both methods and then, determine which one is more accurate and plot the relevant trend line

Least Squared Method

The Least Squares Method involves 6 steps:

1. For each point (x,y) in your dataset, calculate x^2 and xy.

2. Then, calculate Σx, Σy, Σx^2 and Σxy. The sigma essentially means sum up all values.

3. Then, calculate the gradient of this line using this formula:

$$\mathbf{m} = \frac{N\,\Sigma(xy) - \Sigma x\,\Sigma y}{N\,\Sigma(x^2) - (\Sigma x)^2}$$

Where N represents the sample size. Dividing by N on the numerator and denominator and simplifying gives:

$$m = \frac{(\Sigma xy - \frac{\Sigma x \Sigma y}{N})}{\Sigma x^2 - \frac{(\Sigma x)^2}{N}}$$

$$m = \frac{(\Sigma xy - \frac{N\Sigma x \Sigma y}{N^2})}{\Sigma x^2 - \frac{N(\Sigma x)^2}{N^2}}$$

$$m = \frac{(\Sigma xy - N(\bar{x})(\bar{y})}{\Sigma x^2 - N(\bar{x})^2}$$

Where $\bar{x}\ and\ \bar{y}$ are the means of the x and y data. The numerator is also known as $SS_{xy,}$ and the denominator is also known as SS_{xx}

4. Then, calculate the y intercept of the line using this formula:

$$\mathbf{b} = \frac{\Sigma y - m\,\Sigma x}{N}$$

Simplifying this gives:

$$b = \frac{\Sigma y}{N} - \frac{m\Sigma x}{N}$$

$$b = \bar{y} - m\bar{x}$$

Now, the line will be y = mx + b. The pseudocode of the algorithm is shown below:

```
SUB estimate_coef(x,y)
      Sigma_x = 0
      Sigma_y = 0
      Sigma_x^2 = 0
      Sigma_xy = 0
      FOR i ← 0 TO LEN(x)
            Sigma_x = Sigma_x + x[i]
            Sigma_y = Sigma_y + y[i]
            Sigma_x^2 = Sigma_x^2 + (x[i])^2
            Sigma_xy = Sigma_xy + x[i]*y[i]
      NEXT i
      Mean_x = Sigma_x / LEN(x)
      Mean_y = Sigma_y / LEN(y)
      SS_xy = Sigma_xy - LEN(x)*Mean_x*Mean_y
      SS_xx = Sigma_x^2 - LEN(x)*(Mean_x)**2
      Gradient = SS_xy / SS_xx
      Intercept = Mean_y - Gradient*Mean_x
RETRUN Intercept, Gradient
```

Although this is the basic algorithm, it will be made a lot shorter due to the use of numpy arrays.

Gradient Descent

Gradient Descent is an iterative optimisation algorithm that finds the minimum of any differentiable function. In our case, we will create an error function based on the actual data values and we will aim to find where the error is minimum. The error is as follows:

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y})$$

Where n is the number of data points, y_i is the actual i^{th} y value and $\bar{y}$ is the predicted y value. Hence, we can re-write this as:

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))$$

Where x_i is the i^{th} term in the dependent variable list (x), m is the gradient of the line and c is the y intercept of the line. I wish to minimise this error function. I will do this by computing the partial derivatives of E with respect to m and c respectively. This is done using normal differentiation techniques such as the Chain Rule but terms that do not contain m or c respectively are treated to be a constant. Expressions for the partial derivatives with respect to m and c are D_m and D_c respectively as shown below:

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

$$D_c = \frac{-2}{n}\sum_{i=0}^{n} y_i - \bar{y}_i$$

Using these, we update the values of m and c:

$$m = m - L * D_m$$

$$c = c - L * D_C$$

Where L is the learning rate. The learning rate is the rate at which the algorithm gets closer to the minimum point on the error function. However, it should be set to a small value because if it is too large, it may oscillate above and below the minimum value of the cost function. Hence, it is usually around 0.0001. This process is repeated many times (around 1000) and this causes the error function to converge to a real value. After 1000 tries, I record the value of m and the value of c and determine my trend line. The pseudocode for gradient descent algorithm is shown below:

```
SUB GradientDescent(x,y)
    SUB OptimiseCoefs(x,y,c,m,L)
        epochs = 1000
        FOR i ← 1 TO epochs
            y_pred = m*x + c
            D_m = (-2/LEN(x))*np.sum(x*(y-y_pred))
            D_c = (-2/LEN(x))*np.sum(y-y_pred)
            m -= (L*D_m)
            c -= (L*D_c)
        RETURN [c,m]
    m = (y[LEN(y)-1]-y[0])/(x[LEN(x)-1]-x[0])
    c = y - m*x[0]
```

```
    RETURN OptimiseCoefs(x,y,c,m,L)
```

Epochs represents the number of iterations carried out by the algorithm. Also, instead of using m=0 and c=0 as starting values, I will use the value of m and c found from calculating the gradient of the line that goes through the first and final data point. This gives the model a more accurate starting point for m and c which may allow the gradient descent algorithm to yield more accurate results over the 1000 iterations.

After I have found 2 possible pairs of coefficients, one from the least square's algorithm and the other from the Gradient Descent algorithm, I will use the R-squared score to determine the accuracy of both coefficients. The formula to calculate the R-squared score (Minitab, 2013) is shown below:

$$R^2 = 1 - \frac{SSE}{SS_{YY}}$$

where $SSE = \sum (y - \hat{y})^2$,

$SS_W = \sum (y - \bar{y})^2$,

y is the actual value,

$\hat{y}$ is the predicted value of y,

and $\bar{y}$ is the mean of the y values.

Generally, an R-squared score that is close to 1 generally reflects a more accurate trend line. Hence, I will compare the r-squared scores of the lines obtained and see which one is closer to 1.

MERGE SORT ALGORITHM

When the program lists out more efficient vehicles, I aim to output the most efficient vehicle and output in ascending order of emissions. Hence, I need an efficient way to sort each vehicle that is more emissions efficient. This is where a merge sort needs to be used. Merge sort is a divide and conquer algorithm that uses a recursive method to split the given set of elements into subsets, sort them individually and bring the subsets together.

I have chosen merge sort over other sorting algorithms such as Heap Sort or Quick Sort because it is a lot faster for large datasets. This is useful for my project because in the real world, there are thousands of vehicles, of which many could be more emissions efficient, so merge sort is much more useful. The pseudocode is shown below:

```
SUB sort_emissions(efficient_vehicles,emission_totals)
    IF LEN(emission_totals) > 1 THEN
        mid = LEN(emission_totals)//2
        left_array_vehicles = efficient_vehicles[:mid]
        right_array_vehicles = efficient_vehicles[mid:]
        left_array_emissions = emission_totals[mid:]
        right_array_emissions = emission_totals[:mid]
```

Splits the arrays in half

```
        sort_emissions(left_array_vehicles, left_array_emissions)
        sort_emissions(right_array_vehicles, right_array_emissions)
```

Starts recursion.

```
        i = 0
        j = 0
        k = 0
        WHILE i<LEN(left_array_emissions) AND
j<LEN(right_array_emissions)
            IF left_array_emissions[i] < right_array_emissions[j]
THEN
                emission_totals[k] = left_array_emissions[i]
                efficient_vehicles[k] = left_array_vehicles[i]
                i = i + 1
            ELSE
                emission_totals[k] = right_array_emissions[i]
                efficient_vehicles[k] = right_array_vehicles[i]
                j = j + 1
            ENDIF
        k = k + 1
        ENDWHILE
```

Sorts the elements in each half. It indexes through both the left and right arrays, incrementing only if the value in that specific index of an array is bigger than its counterpart in the other array. This happens until one of the arrays has been finished indexing through.

```
        WHILE i<LEN(left_array_emissions)
            emission_totals[k] = left_array_emissions[i]
            efficient_vehicles[k] = left_array_vehicles[i]
            i = i+1
            k = k+1
        ENDWHILE
```

```
WHILE j<LEN(right_array_emissions)
    emission_totals[k] = right_array_emissions[j]
    efficient_vehicles[k] = right_array_vehicles[j]
    j = j+1
    k = k+1
ENDWHILE
```

Adds on any elements that are left onto the end of emission_totals. When one of the arrays has been finished indexing through, there may be some terms left in the other array. These terms will already have been sorted due to the recursive nature of the merge sort so we can just add them onto the end of the merged array.

This is done recursively so it breaks down the list into lists of length 1 and merges them together again, sorting them along the way.

CRITICAL PATH

The critical path for development of my program is shown below:

Objective	Projection Date for Completion
2	8th April
3	13th April
4	20th July
1	1st August
5	10th August

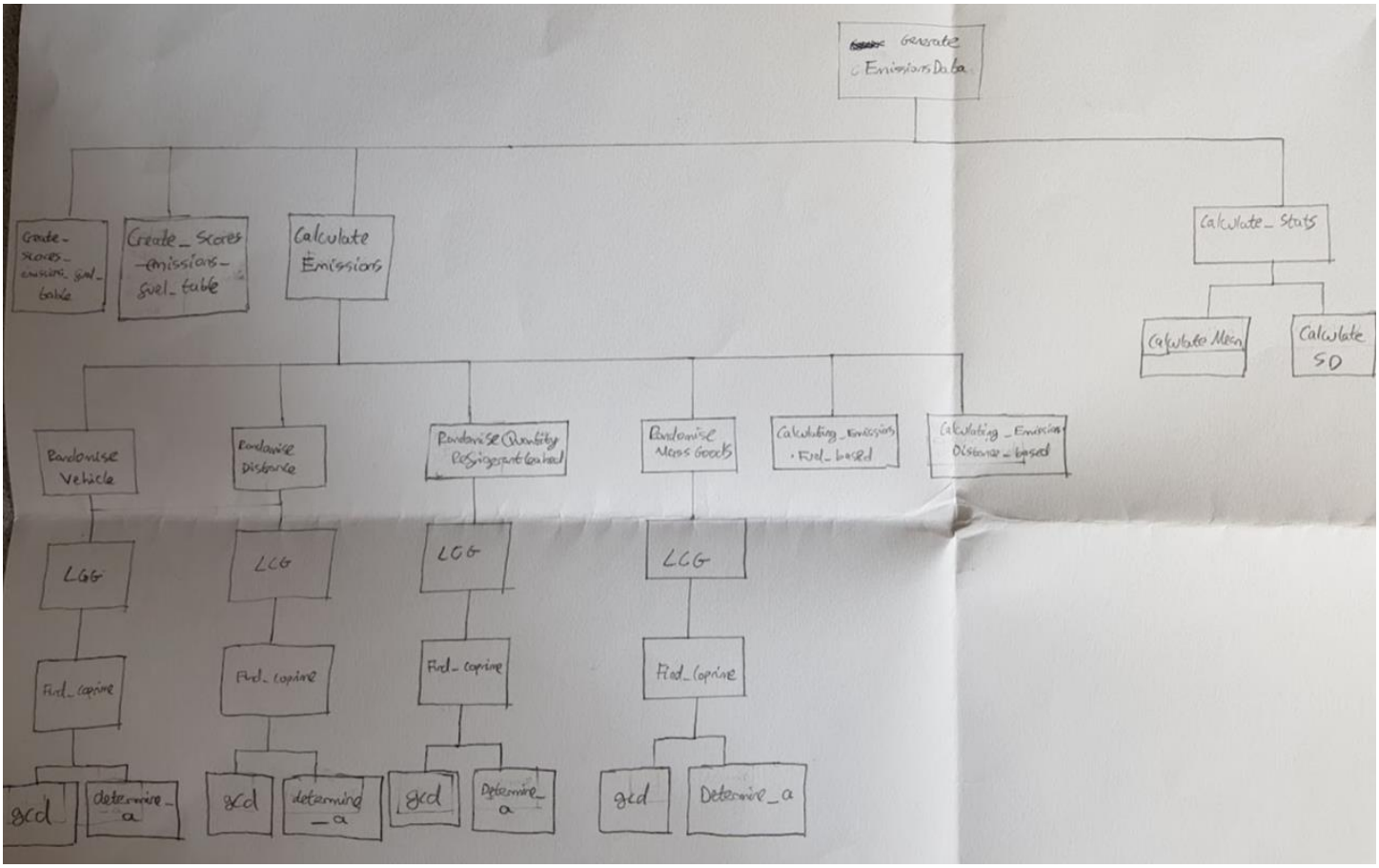
I feel that the projection dates are very reasonable as I have given myself 10 days to complete the more complex parts of my program (objectives 4 and 5). This will account for any stumbling blocks I find along the way in developing my solution.

Also, I plan to do the front end of the program 4th in the list as I will use what I code in the front end in stage 5. But I would like to code stages 2,3 and 4 earlier so that I can test what I have coded in these stages using the front end.

OVERALL DESIGN

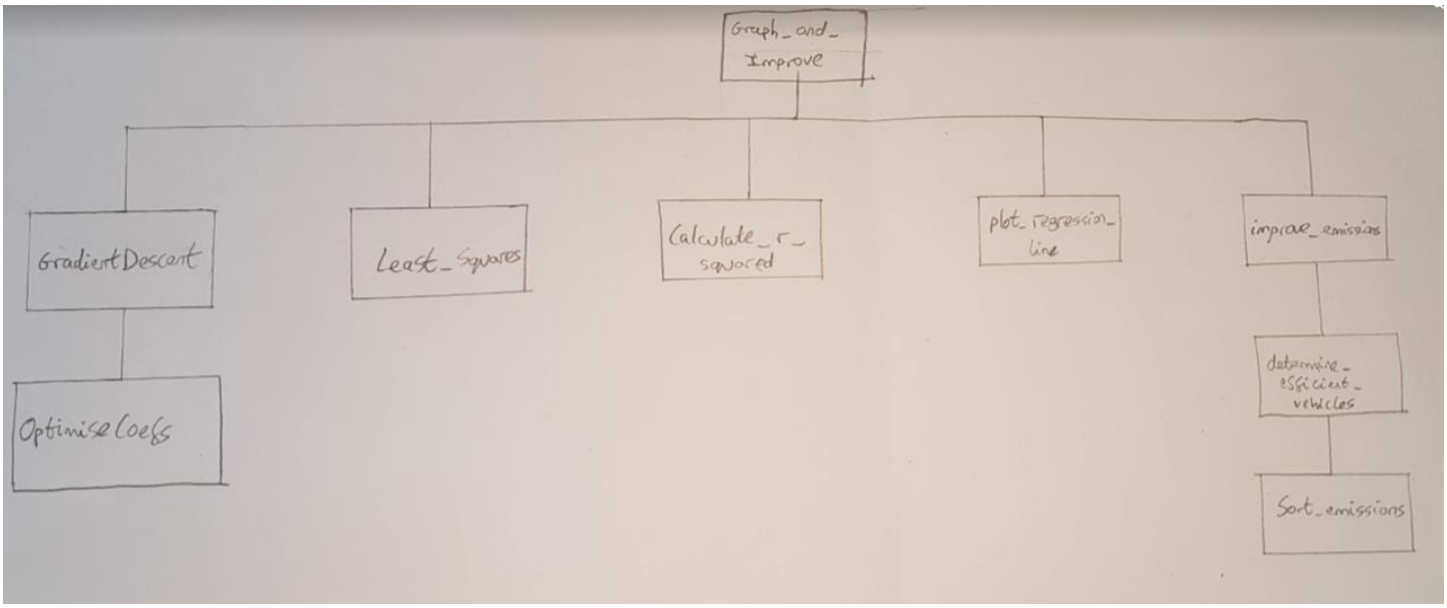
The overall design in the form of 3 hierarchy charts: One for my scoring model, one for the graph plotting model and one general hierarchy chart for the whole program. This shows how I have used decomposition to break down the problem to be solved into a series of sub-problems. The meaning of each function will be revealed in the in-depth design and the implementation.

Scoring model:



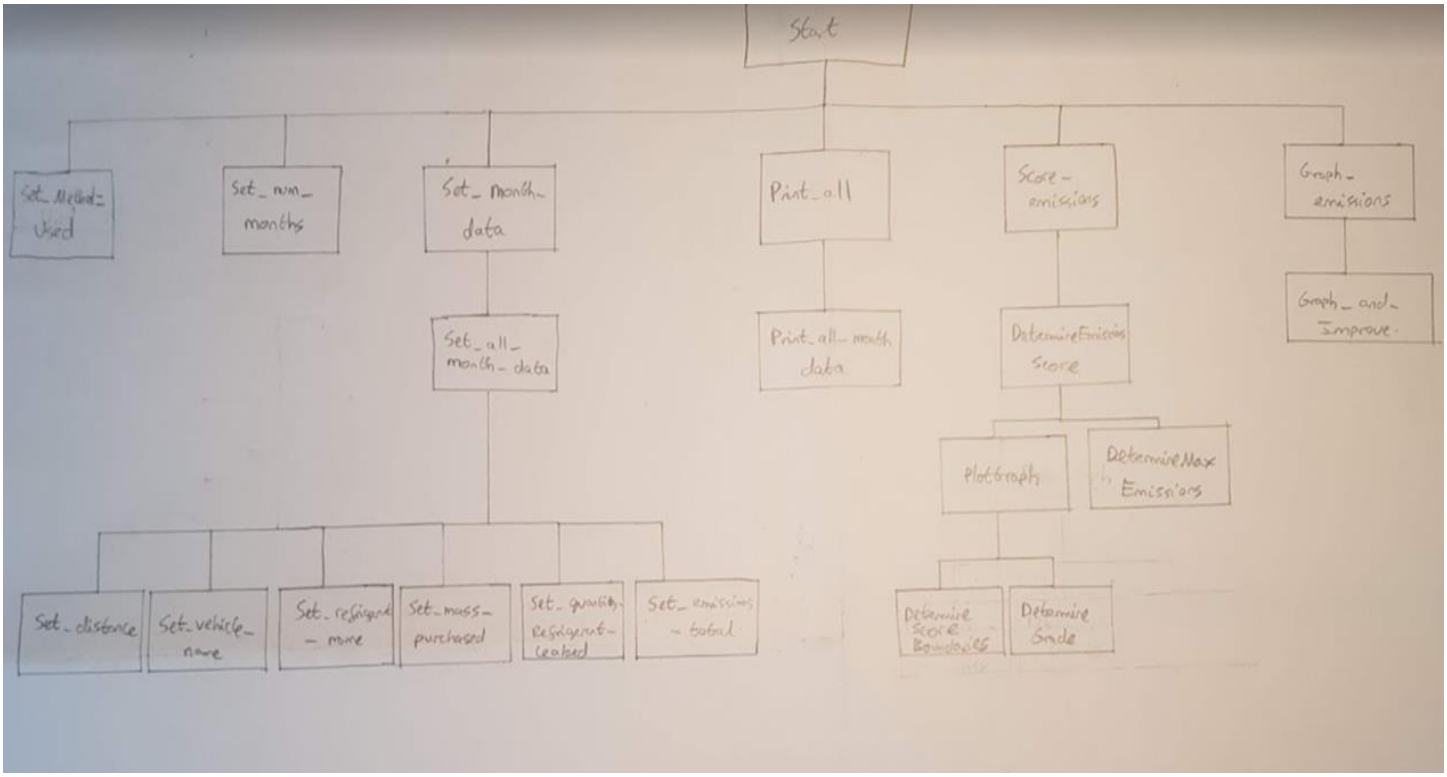
This model will not be run by the user and is run separately to determine the mean and standard deviation of emissions.

Graph plotting Model:



This is run by the user.

Front End:



This is the hierarchy chart for the front end of the program.

All the functions given in the hierarchy chart will be explained further in the in-depth design.

Design

I have decided to split the in-depth design section into 5 separate sub-sections:

- Stage 1 – Designing the databases to calculate the emissions.
- Stage 2 – Calculating the emissions incorporating databases in stage 1 and user inputs.
- Stage 3 – Scoring the emissions
- Stage 4 – Building the Front End
- Stage 5 – Plotting the Graph (Also includes using linear regression to project future emissions and potential improvement)

Stage 1 – Designing databases – Objective 2) ,3) and 4)

I have already learnt about databases at school, so I have provided no bibliography links as no research was required.

Recall formula for fuel-based method:

CO_2e emissions from transportation =

sum across fuel types:
∑ (quantity of fuel consumed (liters) × emission factor for the fuel (e.g., kg CO_2e/liter))
+
sum across grid regions:
∑ (quantity of electricity consumed (kWh) × emission factor for electricity grid (e.g., kg CO_2e/kWh))
+
sum across refrigerant and air-conditioning types:
∑ (quantity of refrigerant leakage × global warming potential for the refrigerant (e.g., kg CO_2e))

Quantities of fuel consumed (liters) =

sum across transport steps:
∑ (total distance travelled (e.g., km) × fuel efficiency of vehicle (e.g., liters/km))

Recall formula for distance-based method:

Calculation formula [4.6] Distance-based method (transportation)

CO_2e emissions from transportation =

sum across transport modes and/or vehicle types:
= ∑ (mass of goods purchased (tonnes or volume) × distance travelled in transport leg (km)
× emission factor of transport mode or vehicle type (kg CO_2e/tonne or volume/km))

(GHG Protocol, 2013)

I will be creating 1 database for both calculation methods because both methods require similar data.

For both methods, entities will need to be made on the following:

- Vehicles – Because the user will be entering the brand of vehicle that is being used for the transportation so data about the vehicles is needed.

- Fuels– As the user will be entering the brand of vehicle, a database giving the property of fuels used by vehicles must be made.
- Refrigerants – As it will be used in calculating emissions for both methods.

I am not considering the sum across grid regions because I am considering a transportation of goods and hence, I will assume that no electricity is being used. The Total Distance travelled will be calculated using data entered in by the user.

Also, in the scoring process, I will need to create 2 more entities to store all the 1000 randomised emissions scores for my Normal Distribution. I have done this because it will be a lot quicker to carry out maths on the data using aggregate functions such as SUM rather than using more FOR loops. In the database, I will be having fields for each variable that is randomised. I have given further details on which randomisation processes I have used in Stage 3 of the design process. I have done this so I can test this after I have built my solution. Note that these are only for Objective 4, not objective 2 or 3.

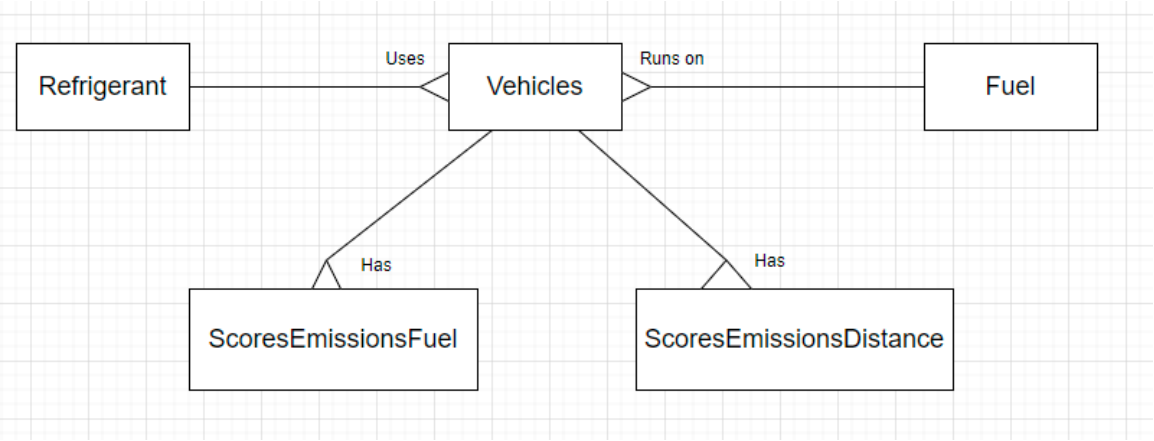
I will now be giving the Entity Descriptions for each entity. An entity description describes the properties (fields) that each entity will contain.

- **Vehicles (vehicle_id, vehicle_name, fuel_id, refrigerant_id, fuel_efficiency)**
- **Fuel (fuel_id, fuel name, emission_factor)**
- **Refrigerant (refrigerant_id, refrigerant_name, gwp)**
- **ScoresEmissionsFuel (round, total_emissions, vehicle_id, quantity_refrigerant_leaked, distance)**
- **ScoresEmissionsDistance (round, total_emissions, vehicle_id, mass_goods_purchased, distance)**

NB: gwp stands for global warming potential. Primary keys underlined.

This database is in Third Normal Form because all data is atomic and there are no repeating groups (First Normal Form), all attributes (properties of entity) depend on the whole primary key (uniquely identifies a record in an entity) meaning that it is in Second Normal Form and no non-key attributes depend on other non-key attributes meaning that the database is also in Third Normal Form (3NF).

The Entity Relationship (ER) diagram below shows the relationships between the entities.



The Vehicle and Refrigerant tables have a one-to-many relationship because I am assuming that each vehicle uses only 1 refrigerant. The Vehicle and Fuel tables also have a one-to-many relationship as many vehicles can use a single fuel. Also, the ScoresEmissionsFuel table and the vehicles table have a one-to-

many relationship as many records in the ScoresEmissionsFuel table can have the same vehicle. This also applies for the ScoresEmissionsDistance

I have also given formal data dictionaries for each of the entities in the database.

Vehicles Data Dictionary

Field Name	Data Type	Constraints	Purpose
Vehicle_id	Int	Primary Key, Not Null	To uniquely identify each vehicle
Vehicle_name	Varchar	Not Null	To show the official name of the vehicle. This is what the user will be entering into the program, so it needs to be present.
Fuel_id	Int	Foreign Key (references Fuel.fuel_id), not null	To indicate which fuel the vehicle uses
Refrigerant_id	Int	Foreign Key (references Refrigerant.refrigerant_id), not null	To indicate which refrigerant the vehicle is using
Fuel_efficiency	Real	Not Null	To indicate the fuel efficiency of the vehicle in litres/km.

Refrigerant Data Dictionary

Field Name	Data Type	Constraints	Purpose
refrigerant_id	Int	Primary Key, Not Null	To uniquely identify each refrigerant used in vehicles given in the vehicles entity.
refrigerant_name	Varchar	Not Null	To give the name of each refrigerant
gwp	Real	Not Null	To denote the global warming potential of the refrigerant needed for calculation

Fuel Data Dictionary

Field Name	Data type	Constraints	Purpose
fuel_id	Int	Primary Key, Not Null	To uniquely identify each fuel in
fuel_name	varchar	Not Null	To give the name of the fuel
emission_factor	real	Not Null	To give the emission factor of the fuel for calculation

ScoresEmissionsFuel Data Dictionary

Field Name	Data Type	Constraints	Purpose
round	int	Primary Key, Not Null	To uniquely identify each emissions total from each randomisation round
total_emissions	real	Not Null	To give the total emissions generated from a randomisation round
vehicle_id	int	Foreign Key (references Vehicles.vehicle_id), Not Null	To give the vehicle_id generated at random at that round
quantity_refrigerant_leaked	real	Not Null	To give the quantity refrigerant leaked that is generated at random each round
distance	real	Not Null	To give the distance travelled that is generated at random from each round

ScoresEmissionsDistance Data Dictionary

Field	Data Type	Constraints	Purpose
round	int	Primary Key, Not Null	To uniquely identify each emissions total from each randomisation round
total_emissions	real	Not Null	To give the Total emissions generated from a randomisation round
vehicle_id	int	Foreign Key (references Vehicles.vehicle_id), Not Null	To give the vehicle_id generated at random at that round

mass_goods_purchased	real	Not Null	To give the randomly generated mass of goods purchased for each round of randomisation
distance	real	Not Null	To give the distance travelled that is generated at random from each round

Now, I need to declare what vehicles and hence what refrigerants I will be considering for my model. The vehicles that I will be considering, and their respective fuels and fuel efficiencies are shown in the below table. I have assumed that the brand of each truck only uses 1 fuel and I have found the fuels and fuel efficiency for each vehicle from company websites. I found that most, if not all trucks use the same R134a refrigerant. This is 1,1,1,2 – tetrafluoroethane. I will still make an entity for refrigerants however as it will allow programmers to enter any more refrigerants that emerge in the future making my program more flexible. The global warming potential (gwp) of R134a is **1.41 Kg/CO_2e** meaning releasing 1 gram of R134a is same as releasing 1.41 kg of CO_2

Vehicle	Fuel used	Refrigerant used	Vehicle fuel Efficiency (litres/km)
Scania L-series	Biodiesel	R134a	0.415
Scania P-series	Biodiesel	R134a	0.237
Western Star 4700 series	Mineral Diesel	R134a	0.394
Western Star 5700 XE	Natural Gas	R134a	0.338
IVECO S-way	Mineral Diesel	R134a	0.211
IVECO Stralis	Natural Gas	R134a	0.208
MAN TGA	Mineral Diesel	R134a	0.0893
MAN TGM	Mineral Diesel	R134a	0.181

The list of trucks can be extended by the programmer very easily. This list is just a useful list for testing purposes. I will cover the code that I will use to create the databases in the technical solution stage. Now, from the vehicles I am considering, I have the fuels I will consider for my algorithm. These fuels are Biodiesel, Diesel and Natural Gas. I have found the emission factors for these fuels from the British Government (UK Government, 2020), and I have used the 2020 emission factor data. The table below shows the fuels and their respective emission factors.

Fuel	Emission Factor (kg CO_2e/litre)
Biodiesel	0.1658
Mineral Diesel	2.68787
Natural Gas	2.02266

Again, if this were to be used, the list of fuels would be far more extensive, and programmers can extend this list very easily.

For the distance travelled, the user will enter the postcodes.

For the distance-based method, the user will enter the mass of goods purchased, the postcodes which truck stops by during its journey, the model of truck used. From this the total emission factor for the vehicle is calculated by adding the gwp and the emission factor of the fuel the truck uses.

The table below gives the vehicle with the total emission factor for that vehicle.

Vehicle	Emission Factor
Scania L-series	1.5758
Scania P-series	1.5758
Western Star 4700 series	4.09787
Western Star 5700 XE	3.43266
IVECO S-way	4.09787
IVECO Stralis	3.43266
MAN TGA	4.09787
MAN TGM	4.09787

Stage 2 - Calculating the total emissions – Objective 2) & 3)

This will also be split out into 2 sections. One explains how the fuel-based method is calculated and the other explaining how the distance-based method is calculated.

Fuel-Based Method Calculation – Objective 2)

Recall the formula to calculate the fuel-based method:

CO_2e emissions from transportation =

sum across fuel types:
Σ (quantity of fuel consumed (liters) × emission factor for the fuel (e.g., kg CO_2e/liter))
+
sum across grid regions:
Σ (quantity of electricity consumed (kWh) × emission factor for electricity grid (e.g., kg CO_2e/kWh))
+
sum across refrigerant and air-conditioning types:
Σ (quantity of refrigerant leakage × global warming potential for the refrigerant (e.g., kg CO_2e))

Quantities of fuel consumed (liters) =

sum across transport steps:
Σ (total distance travelled (e.g., km) × fuel efficiency of vehicle (e.g., liters/km))

The design for this section will be broken into the respective components of the Formula.

The user will be entering data giving the postcodes in the journey of their trucks in transporting their products as well as the refrigerant they are using, and the approximate quantity of refrigerant leaked by the truck during each of its journeys.

Quantity of fuel consumed.

Here, I need to calculate the total distance travelled. I plan for the user to enter in the postcodes that the vehicle goes to in the journey to transport a product. This is not just a 'start point' to 'end point' journey because in the real world, trucks transport multiple goods at once meaning that they stop at different locations first to deliver a different product. This must be factored in when calculating the distance

travelled. For example, a truck might be transporting a good from Liverpool to London, but it might stop in Birmingham halfway to deliver a different product first and this must of course be factored in.

To be realistic, the user can enter a maximum of 6 postcodes and these postcodes will of course be validated when inputted. All validation will be carried out on the front end of the program. I will calculate the distance between 2 postcodes using an imported library called geopy. Geopy is a library that allows you to calculate distances between cities and postcodes by taking them as coordinates. This will be useful for me to calculate the distance between 2 postcodes. Then all I need to do is fetch the fuel efficiency of the vehicle being used from the database and multiply it by the distance travelled.

Rest of Formula

The rest of the formula just involves using parameterized sql queries to fetch data from the database of vehicles and plug them into the formula. NB: As no electricity is being used, I am discarding the 'sum across grid regions' component of the formula.

Pseudocode

Here is the pseudocode for this section:

```
IMPORT geopy, sqlite3

DB_PATH = 'TestCalculationsEmissions_db.db'

Get database from files

SUB get_db ()
      Retrieves database from files.
      RETURN db

Functions for both methods

SUB CalculateDistance(postcodes)
      Declarations first made to geopy servers.
      total_distance = 0
      FOR i ← 1 TO (len(postcodes)-1)
            address1 = postcodes[i]
            address2 = postcodes[i+1]
            Finds coordinates of both postcodes on OpenStreetMap,
            then uses this to find latitude and longitude of each
            address using geopy. (Geopy, 2020)
            Distance between these 2 postcodes is then calculated
            by using .distance() method of geopy
```

```
            total_distance += distance_between_2_locations
      NEXT i
      total_distance = ROUND(total_distance,4)
      RETURN total_distance
SUB GetEmissionFactorForFuel(db, vehicle_name)
      Fuel_id = SELECT fuel_id FROM Vehicles WHERE
            vehicle_name=?,(vehicle_name,)
      Emission_Factor = SELECT emission_factor FROM Fuel WHERE
                        fuel_id=?",(fuel_id,)
      RETURN Emission_Factor
SUB GetGWP(db,refrigerant_name)
      gwp = SELECT gwp FROM Refrigerant WHERE
             refrigerant_name=?", (refrigerant_name,)
      RETURN gwp
```

Fuel-based method functions

```
SUB GetFuelEfficiency(db,vehicle_name)
      Fuel_efficiency = SELECT fuel_efficiency FROM Vehicles
                  WHERE vehicle_name=?",(vehicle_name,)
      RETURN Fuel_efficiency
SUB
CalculateTotalEmissionsFuel(distance,fuel_efficiency,Emission_factor_fuel,Quant
ityRefrigerantLeaked,gwp)
    QuantityFuelConsumed = distance * fuel_efficiency
    Emissions_Fuel = QuantityFuelConsumed * Emission_factor_fuel
    Emissions_Refrigerant = QuantityRefrigerantLeaked * gwp
    total_emissions = Emissions_Fuel + Emissions_Refrigerant
    RETURN total_emissions
ENDSUB
SUB Fuel-
based(db,vehicle_name,postcodes,refrigerant_name,QuantityRefrigerantLeaked,num_
journeys)
    distance = CalculateDistance(postcodes)
```

```
    emission_factor = GetEmissionFactorForFuel(db,vehicle_name)

    gwp = GetGWP(db,refrigerant_name)

    fuel_efficiency = GetFuelEfficiency(db,vehicle_name)

    total_emissions =
CalculateTotalEmissions_fuel(distance,fuel_efficiency,emission_factor,QuantityR
efrigerantLeaked,gwp)

    RETURN total_emissions

ENDSUB
```

Distance-based Method functions

```
SUB
CalculateTotalEmissions_distance(vehicle_name,distance,mass_goods_purchased,emi
ssion_factor_fuel,gwp)

    emission_factor_vehicle = emission_factor_fuel + gwp

    emission_factor_vehicle = round(emission_factor_vehicle,4)

    total_emissions = mass_goods_purchased*distance*emission_factor_vehicle

    RETURN total_emissions

ENDSUB

SUB
fuel_based(db,vehicle_name,distance,refrigerant_name,QuantityRefrigerantLeaked,
num_journeys):

    emission_factor = GetEmissionFactorForFuel(db,vehicle_name)

    gwp = GetGWP(db,refrigerant_name)

    fuel_efficiency = GetFuelEfficiency(db,vehicle_name)

    total_emissions =
CalculateTotalEmissions_fuel(distance,fuel_efficiency,emission_factor,QuantityR
efrigerantLeaked,gwp)

    RETURN total_emissions

ENDSUB
```

NB: For fuel-based method, the variables postcodes, vehicle_name, refrigerant_name and QuantityRefrigerantLeaked will be inputted in by the user on the front end of the program and I will code that in that section. For the Distance-based Method, the variables vehicle_name, refrigerant_name and GoodsPurchased will be inputted by the user and will be coded on the front end.

Distance-based Method Calculation – Objective 3)

Recall the distance-based method formula:

Calculation formula [4.6] Distance-based method (transportation)

CO_2e emissions from transportation =

sum across transport modes and/or vehicle types:

= ∑ (mass of goods purchased (tonnes or volume) × distance travelled in transport leg (km) × emission factor of transport mode or vehicle type (kg CO_2e/tonne or volume/km))

The user will be entering in data about the mass of goods purchased, the truck used to transport the goods and the postcodes the truck stops by during its journey to transport the product. The algorithm to calculate the distance travelled is the same as in the fuel-based method and the emission factor of the truck can be fetched from the database created using parameterized SQL Queries. One difference is that the emission factor of the transport mode will be calculated by adding the emission factor of the fuel to the global warming potential of the refrigerant. I can do this as they both share the same units This is reasonably simple, and it mainly uses functions already created for the fuel-based method. Hence, I will not include any design for this.

Stage 3 - Scoring the emissions – Objective 4)

After calculating the total emissions, I need to score the emissions. This will be done by generating a normal distribution and showing where the emission total falls on this normal distribution.

First, I need to create a Normal Distribution which my score is based on. To do this I need a population mean and the standard deviation of the emission scores generated by the modified Monte Carlo Simulation explained in the Analysis section. In a similar way to the calculations, I will break the randomisations required into 3 categories:

- Randomisations in fuel method
- Randomisations in distance method
- Randomisation processes needed in both methods.

Randomisation processes in Fuel Method

1. Quantity of Refrigerant Leaked

I need to randomise this because this value is entered by the user and is subject to variation.

To determine the upper bound of randomisation, I had to first research the general capacity of refrigerant in the vehicle's air conditioning. This is thought to be roughly 2.72155 kg- 5.44311kg (6-12 pounds). I also had to find out the rate at which refrigerant leaks from air conditioning which is around 6% as a maximum (Accuvio, 2017), (HSI, 2018) . Doing the maths gives a maximum leaked refrigerant to be roughly 5.44311*0.06=0.3265866 kg of leaked refrigerant as a maximum value of refrigerant leaked.

Some goods may need refrigeration. Large refrigeration units leak 17% roughly (check Accuvio link). The max quantity that could possibly be leaked from refrigeration is 0.185 kg. Hence, doing the maths gives a max value of 0.03145kg of leaked refrigerant.

Summing both values together give a total maximum refrigerant value of 0.3580366 kg of refrigerant leaked for consumer products.

Now, a simple call using a function in the Random library in python can be used to determine the leaked refrigerant rate at random.

Randomisation processes in Distance Method

1. Mass of Goods purchased/being transported

This again is subject to variation as it is entered by the user. For simplicity purposes, I will make the maximum load that a truck can take to be 500 kg. I will put this through the Linear Congruential Generator to get a randomised mass of goods purchased.

Randomisation processes needed in both Methods.

1. Random vehicle being entered. I need to randomly pick a vehicle for both methods. For the fuel-based method, the random vehicle selected gives me the emission factor of the fuel and the gwp of the refrigerant used. For the distance-based method, I need the emission factor of the fuel.

I will do this by using a MAX Aggregate Function to find the maximum Vehicle ID and then picking an ID at random.

2. Distance Travelled:

I will do this by finding the maximum distance between 2 points in the UK. The longest distance between any 2 points in the UK across 1 journey length will be assumed to be the distance between Land's End in Cornwall and John O'Groats in Northern Scotland (Wikipedia, Unknown) . This is 1407 km. However, the user can enter a maximum of 6 postcodes and hence, 1407 must be multiplied by 5. As a result, my maximum distance will be 7035 and I need a random real distance between 0 and 7035km for my randomisation. I will not use the postcodes at random because then I must use Geopy to find the distance between them and this takes a long time.

Putting it all the randomised components together

For each method, I will now calculate the total emissions using the randomised components using the formulae for the fuel and distance formulas stated above. This value will then be stored in an array or emissions values. This will be done a very large number of times. (1000)

Creating the Normal Distribution

I am going to score my data by evaluating the probability of a certain emissions total from occurring. Hence, I need to decide how the data is distributed. Clearly, the data is continuous so any discrete distributions like Poisson and Binomial are ruled out straight away. Hence, I feel that a normal distribution (the most used continuous distribution) is more appropriate. Also, even if another distribution is more accurate, most distributions can be approximated by the normal distribution for very large datasets. I have a dataset of 1000, so a normal distribution is probably best.

The Normal Distribution has 2 parameters, μ and σ^2. μ is the population mean and σ^2 is the variance (a measure of spread of the data). To construct my distribution, I will need to calculate both the mean and the variance. I can do this using the following formulas:

$$\bar{x} = \frac{(x_1 + x_2 + \ldots\ldots + x_n)}{n} = \frac{\sum_{i=1}^{n} x_i}{n},$$ (Sample Mean)

This is covered by the AVG aggregate function in Sqlite3, so this formula is already implemented for me.

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$ (Standard deviation Formula) - (Mathlearnit, Unknown)

I am going to use the standard deviation and sample mean from my generated data as an approximation for the population mean and population standard deviation. After calling the required SQL Statements (given at the end of design), I will calculate these values.

Scoring the Emissions:

The scoring system that I am going to carry out will be relatively basic as I have done most of the groundwork already in the distribution creation. I will do a grading system (A-E) based on the probability of getting the emissions total of the company or less. I will base it on the probability of emissions being less than or equal to the emissions total calculated as shown below.

P (Emissions <= a)	Grade
P(Emissions <= a) <= 0.2	A
0.2< P(Emissions <= a) <= 0.4	B
0.4 < P(Emissions <= a) <= 0.6	C
0.6< P(Emissions <= a) <= 0.8	D
0.8 < P(Emissions <= a) < 1	E

One issue that could arise is that some of the Normal Distribution generated could be less than 0. This may impact the scoring mechanism as I need to identify the proportion of the normal distribution that falls in the greater than 0 category on the graph. Then, I need to split the regions appropriately as shown in the table above to determine the score.

The pseudocode for the algorithm that determines this is shown below. I have created this algorithm myself:

```
cum_zero = norm(loc=Mean, scale=SD).cdf(0) Finds P(Emission <= 0)

remaining_p = 1-cum_zero

gap = remaining_p/5 Calculates 20% of the region P(Emission>=0)

score_boundaries =
[cum_zero+gap,cum_zero+(gap*2),cum_zero+(gap*3),cum_zero+(gap*4)]

score_boundaries_x = []

FOR i ← 0 TO len(score_boundaries)

      z_x_value = norm.ppf(score_boundaries[i])
```

```
      Calculates inverse normal of each score boundary. This is the z-score,
however. Means that it is a value that is used for a Mean of 0 and a SD of 1.
Hence, I need to adjust it for my distribution.

      x_value = (z_x_value*SD)+Mean

      score_boundaries_x.append(x_value)

NEXT i

score_boundaries_x.append(x_values[len(x_values)-1]) Adds on final grade to be
biggest x value input. Done like this as the approximations done in computer
calculation may result in a value greater than the maximum x value inputted in.

Grades = ['A','B','C','D','E'] Determining the Grade

FOR Month ← 0 TO len(User_Total_Emissions)

      Grade_found = False

      index = 0

      Grade = ''

      WHILE Grade_found = False DO

            IF score_boundaries_x[index] >= User_Total_Emissions[Month] THEN

                  Grade = Grades[index]

                  OUTPUT 'Month ',i+1,':'

                  OUTPUT 'Total Emissions: ', User_Total_Emissions[Month]

                  OUTPUT 'Grade: ',Grade

                  Grade_found = True

            ELSE

                  index = index + 1

            ENDIF

      ENDWHILE

NEXT Month
```

Stein's Algorithm

The python code to find the smallest number coprime to another is shown below using Stein's algorithm. I will explain the code in full in the implementation section as it is easier to explain with the python code next to it.

```
def find_coprime(divisor):
    def gcd(a,b): # Stein's Algorithm
        if (a == b):
            return a
        # GCD(0, b) == b; GCD(a, 0) == a,
        # GCD(0, 0) == 0
        if (a == 0):
            return b
        if (b == 0):
            return a
        # look for factors of 2
        # a is even
        if ((~a & 1) == 1):
            # b is odd
            if ((b & 1) == 1):
                return gcd(a >> 1, b)
            else:
                # both a and b are even
                return (gcd(a >> 1, b >> 1)) << 1
        # a is odd, b is even
        if ((~b & 1) == 1):
            return gcd(a, b >> 1)
        # reduce larger number
        if (a > b):
            return gcd((a - b) >> 1, b)
        return gcd((b - a) >> 1, a)

    found = False
    i = 2
    while found is False:
        hcf = gcd(i,divisor)
        if hcf == 1: # Stops at the snmallest number coprime to the divisor.
            found = True
        else:
            i += 1
    return i
```

Linear Congruential Generator Algorithm

The pseudocode used in the LCG is shown below. I have also shown the algorithm to determine my value of a in this section as well. I will explain the algorithms in full in my technical solution as it is easier to explain with the python code next to it.

```
SUB LCG(divisor)

      date = # Determines current minutes, seconds and microsecond using
datetime   library

      num_iterations = INT(date[9:])

      a = determine_a(divisor)

      c = find_coprime(divisor) Calls Stein's Algorithm

      x = INT(date[8])
```

```
      FOR i ← 1 TO num_iterations
            x = (a*x+c)MOD(divisor+1)
      NEXT i
      IF x == 0 THEN
            x = x + 1
      ENDIF
      RETURN x
ENDSUB
SUB determine_a(divisor)
      prime_factors = []
      found = FALSE
      factor = 2
      WHILE found = FALSE
            all_one_prime_factor = False
            WHILE all_one_prime_factor = FALSE
                  IF divisor MOD factor = 0 THEN
                        divisor = divisor/factor
                        prime_factors.append(factor)
                  ELSE
                        all_one_prime_factor = TRUE
                  ENDIF
            ENDWHILE
            IF divisor = 1 THEN
                  found = TRUE
            ELSE
                  factor = factor + 1
            ENDIF
      ENDWHILE
      num_factors_2 = prime_factors.count(2)
      factors_of_a = LIST(dict.fromkeys(prime_factors)) # Removes repeated
factors of the divisor
```

```
      IF num_factors_2 >= 2 THEN

            a = Product of all the terms in factors_of_a multiplied by an extra
2

      ELSE

            a = Product of all the terms in factors_of_a

      ENDIF

      RETURN a
```

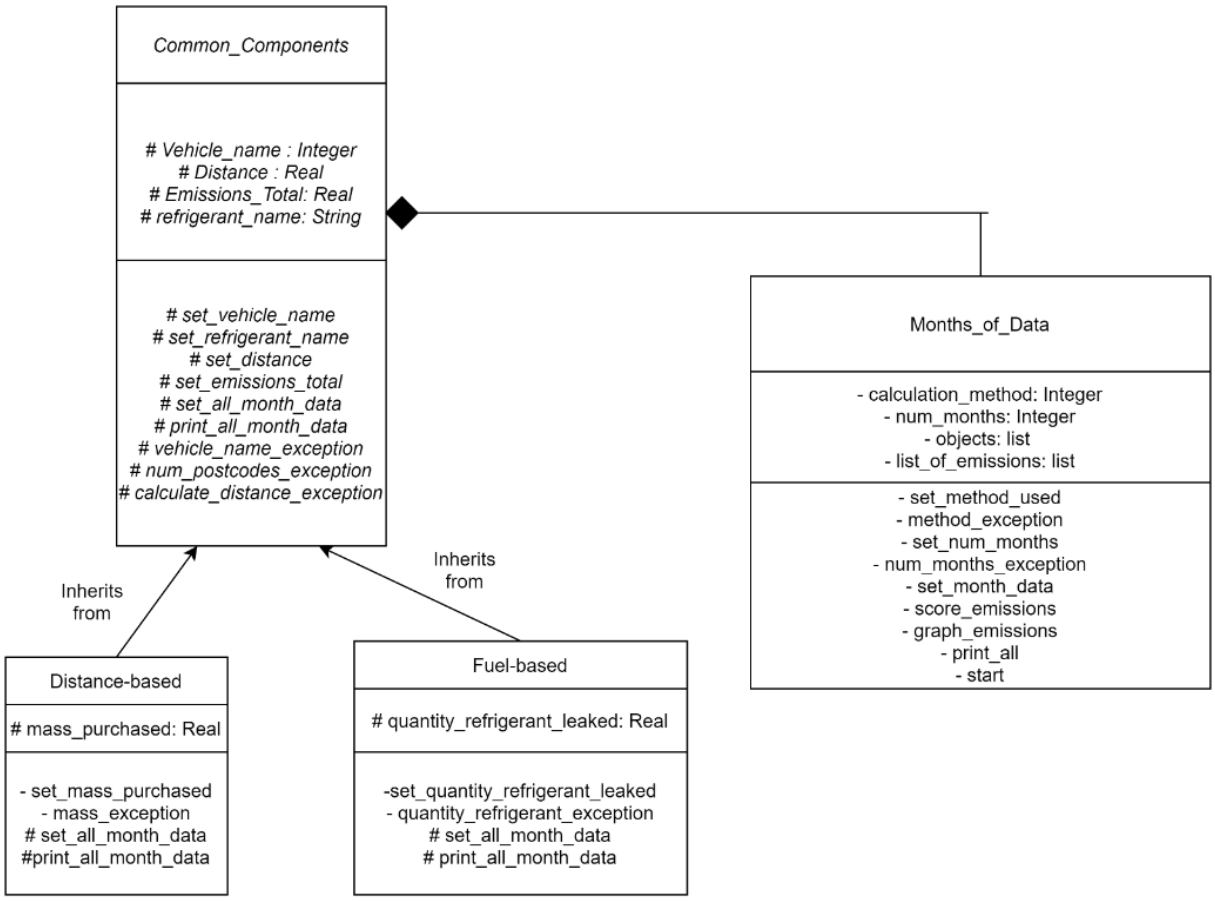
Stage 4 – Building the Front End – Objective 1)

In the front end, I aim to call all other methods and programs created in Objectives 2,3, 4 and later, 5 so everything is in 1 place.

To reduce the coding, I must write, I will use an Object Orientated approach. This will help me reduce the coding I have to do because each of the distance-based methods and fuel-based methods have some similarities. This lends itself to the use of inheritance as both calculation methods have some formula components in common (which go in the parent class) and some components that are different (and go in the child class). If I used a procedural approach, then I would be repeating the same chunky code twice. This makes my code harder to understand. The OOP approach makes my code a lot more modular and easier for me to debug for issues when testing. As a result, my overall solution will be a mix of both procedural and OOP; procedural for the mathematical sections and OOP for the front end of the program.

Classes Used in Front End

The class diagram is given below:



Months_of_data class:

Only 1 instance of this class is created. The first 2 attributes calculation_method and num_months are attributes that will be entered in the by the user in the setters set_method_used and set_num_months methods. Validation is present for each of these user inputs in the form of exception methods method_exception and num_months_exception. This applies for all exception methods. There is also a set_month_data method which requires the user to enter the data required to calculate emissions in each month. An object is created for each month which will be an instance of one of the child classes Distance-based or Fuel-based. Note that all the month objects will be instances of the same class as the calculation_method is declared for all months the user wishes to enter data for. There is an objects attribute which stores an array of all the instances of one of the child classes created. This is used mainly to help with outputting a summary of data for the user in print_all method as well as call the correct parameters into the graph_emissions method. There is also a list_of_emissions attribute which gives a list of all the total emissions from each month that are calculated. This is used in both the score_emissions and graph_emissions methods respectively. Score_emissions fulfil objective 4 and graph_emissions fulfil objective 5.

Composition is present here as the existence of the instances of the child classes depends on the existence of the Months_of_Data instance. Also, the part objects are not shareable with other objects (although this isn't too relevant as only 1 Months_of_Data object is created).

Also, all the attributes and methods are private because I do not need them to be accessed by other objects as I only have 1 object of that class.

Common_Components class:

The Common_Components class aims to provide functionality for components that are common for both calculation methods. This is a parent class to both the child classes Distance_based and Fuel_based. The attributes vehicle_name, refrigerant_name and distance are to be entered by the user and hence have setter methods set_vehicle_name, set_refrigerant_name, set_distance respectively. Due to validation requirements, there are also exception methods created for each of these (method_exception, vehicle_name_exception, num_postcodes_exception, calculate_distance_exception). The emissions_total attribute stores the total emissions released during that month and it uses the other user inputs calculated earlier to calculate it. Hence, a set_emissions_total method is present in this class, but it contains nothing as it will be overridden in both the child classes due to the formulae being different. There is also a print_all_month_data method that is set to none but will be overridden in the child classes. This is same with the set_all_month_data method

All the attributes are protected because they don't need to be accessed by other unrelated objects but, they need to be accessed by objects of the child classes. This is the same for all the methods too in this class.

Distance_based:

This is one of the child classes of Common_Components, inheriting its attributes and methods. It has 1 more attribute mass_purchased which represents that component of the distance-based method formula. As it's a user input, there is an appropriate setter and exception method with this attribute. There is also a setter for the emissions_total attribute which calculated the total emissions using the distance-based method formula using the pseudocode in Stage 2. This overrides the set_emissions_total method in the parent class. There is polymorphism here because the same method set_emissions_total is present in both the fuel-based and distance-based child classes but the actual code inside each method is different. This is because the formula to calculate the total emissions are different for both methods. Also, there is a print_all_month_data method overridden from the parent that prints out all the data for that month. This is done so the user can see a summary of all the data they have entered without having to scroll back up on the terminal. Polymorphism is used again here as the data to be printed to the user varies with the calculation method. There is also a set_all_month_data method that sets all the attributes at once. This overrides the identical method in the parent class leading to polymorphism again as the attributes being set will be different for both methods.

I have set the mass_purchased attribute and its associated methods to protected as it needs to be accessed outside of the class in the Graph_Emissions Method in the Months_of_Data class. The methods that override from the parent are protected so that the overriding occurs.

Fuel_based:

This is the other child class of Common_Components, inheriting its attributes and methods. It has 1 more attribute quantity_refrigerant_leaked which is in line with the formula for the fuel-based method. It has a setter and exception method associated with it. Then, there is a set_emissions_total, set_all_month_data and print_all_month_data method like in Distance_based but different code is used in these methods in the Fuel_based class due to the difference in the formulae of Fuel_based and Distance_based. The quantity_refrigerant_leaked attribute and its methods are set to protected and other methods are protected for the same reasoning as the Distance_based methods.

Validation used in Setter Methods

As there as user inputs in this section, appropriate validation must be in place. Below, I will give pseudocode for validation used in setting each user input and why it has been done in the way it has been. NB: The exception methods that I am calling give an appropriate error message if an invalid input has been received. All the validation involved the user of TRY..EXCEPT methods.

Vehicle_name – Common_Components

I did this by first, presenting the possible vehicles to choose from to the user and then making them enter a number.

```
# Get all vehicles from database and output them to user. See SQL Statements Used to see which one was used. Comes with an index (1. , 2. Etc.)

# Get maximum vehicle_id as this denotes how many vehicles there are. Another SQL Statement. Stored as max_id

valid = False

WHILE valid = False

	TRY

		Vehicle_id = INT(USERINPUT('Enter the vehicle (1-'+str(max_id)+') used in Transportation'))

		IF 0 < Vehicle_id <= max_id THEN

			# Get vehicle_name based on Vehicle_id

			self._vehicle_name = vehicle_name

			valid = True

		ELSE

			self._vehicle_name_exception(max_id)

		ENDIF

	EXCEPT ValueError

		self._vehicle_name_exception(max_id)

	ENDEXCEPT

ENDWHILE
```

Distance – Common_Components

This was split into 2 stages. This is because the user has to enter the number of postcodes that were 'junctions' in the journey and then enter the actual postcodes. I validated the postcodes by simply just calculating the distance using the Geopy method mentioned in Stage 2 because it will return an error if the postcodes are invalid. Hence, I used this to validate the second stage and I used a simple TRY..>EXCEPT for the first stage.

```
Max_postcodes = 6
OUTPUT 'The max no. of postcodes is ',Max_postcodes
Valid = False
WHILE Valid = False
	TRY
		Num_postcodes = INT(INPUT('How many junctions are there in transportation journey?'))
		IF 0 < Num_postcodes <= 6 THEN
			Valid = True
		ELSE
			Self._num_postcodes_exception(Max_postcodes)
		ENDIF
ENDIF
Distance_successful = False
WHILE Distance_successful = False
	OUTPUT 'Please enter the junctions below: '
	Postcodes = []
	FOR i ← 1 TO Num_postcodes
		Address = INPUT('Address: ')
		Postcodes.append(address)
	NEXT i
	TRY
		# Calls Code in Stage 2 to calculate Distance
		Distance_successful = True
	EXCEPT
		self._calculate_distance_exception()
	ENDEXCEPT
ENDWHILE
```

Set_quantity_refrigerant_leaked – Fuel_based

Simple TRY..EXCEPT Block was used here.

```
# Get Refrigerant Name from Database. Stored as refrigerant_name. See SQL Statements

OUTPUT 'Please enter the quantity of refrigerant leaked from your vehicle'

OUTPUT 'The refrigerant used by your vehicle is ', refrigerant_name

Valid = False

WHILE Valid = False

	TRY

		Quantity_refrigerant_leaked = FLOAT(INPUT('Quantity of
'+str(refrigerant_name)+'leaked'))

		IF Quantity_refrigerant_leaked >= 0 THEN

			self.__quantity_refrigerant_leaked = quantity_refrigerant_leaked

			Valid = True

		ELSE

			self.__quantity_refrigerant_exception()

		ENDIF

	EXCEPT ValueError

		self.__quantity_refrigerant_exception()

	ENDEXCEPT

ENDWHILE
```

Set_mass_purchased – Distance_based

TRY..EXCEPT used here too.

```
Max_mass = 500

OUTPUT 'Please enter mass of goods purchased below. Max is 500kg'

Valid = False

WHILE Valid = False

	TRY

		Mass_purchased = FLOAT(INPUT('Mass Purchased: '))

		IF 0 < Mass_purchased <= 500 THEN

			self.__mass_purchased = Mass_purchased

			valid = True

		ELSE

			self.__mass_exception()

		ENDIF
```

```
        EXCEPT ValueError
                self.__mass_exception()
        ENDEXCEPT
ENDWHILE
```

Set_method_used – Months_of_data

This determines which calculation method the user wishes to use. Basic Try and except method used in a similar way to vehicle_name with a number being assigned to each method. 0 is fuel-based and 1 is distance-based.

```
OUTPUT 'Possible Calculation Methods'
OUTPUT '1. Fuel-based'
OUTPUT '2. Distance-based'
Valid = False
WHILE Valid = False
        TRY
                Calculation_method = INT(INPUT('Enter Calculation Method (1 or 2) you wish to use
for scoring.'))
                IF Calculation_method = 1 or Calculation_method = 2 THEN
                        self.__calculation_method = Calculation_method-1
                        Valid = True
                ELSE
                        self.__method_exception()
                ENDIF
        EXCEPT ValueError
                self.__method_excpetion()
```

Set_num_months – Months_of_Data

Basic TRY…EXCEPT block with a max integer of 6 and a minimum integer of 2.

```
OUTPUT 'Please enter how many months you want to enter data for'
OUTPUT 'The Max. no. of months is 6'
Max_months = 6
Valid = False
WHILE Valid = False
        TRY
                Num_months = INT(INPUT('No. of Months '))
                IF 1 < Num_months <= Max_months THEN
```

```
                self.__num_months = num_months
                valid = True
            ELSE
                self._num_months_exception()
            ENDIF
        EXCEPT ValueError
            self._num_months_exception()
        ENDEXCEPT
ENDWHILE
```

Stage 5 – Plotting the Graph – Objective 5)

Least Squares Algorithm Pseudocode

The python code I have implemented for the Least Squares Algorithm is shown below. I have used numpy arrays to make the code more efficient and smaller in size.

```
def Least_Squares(x,y):
    # Sample Size
    n = np.size(x)

    # Calculating Gradient and y intercept of line
    m_x = np.mean(x) # Calculates Mean
    m_y = np.mean(y) # Calculates Standard Deviation

    SS_xy = np.sum(y*x) - n*m_y*m_x # Calculates SSxy
    SS_xx = np.sum(x*x) - n*m_x*m_x # Calculates SSxx

    gradient = SS_xy / SS_xx # Determines gradient
    intercept = m_y - gradient*m_x # Determines y intercept
```

Gradient Descent Algorithm

The Gradient Descent python code is shown below:

```
def Gradient_Descent(x,y):
    def OptimiseCoefs(x,y,c,m,L):
        epochs = 1000 # nunber of iterations
        for i in range(epochs):
            y_pred = m*x + c
            D_m = (-2/len(x))*np.sum(x*(y-y_pred)) # Partial Derivative with respect to m
            D_c = (-2/len(x))*np.sum(y-y_pred) # Partial Derivative with respect to c
            m -= (L*D_m) # Updates m
            c -= (L*D_c) # Updates c
        return [c,m]

    m = (y[len(y)-1]-y[0])/(x[len(x)-1]-x[0]) # Determines starting m and c values.
    c = y[0] - m*x[0]
    return OptimiseCoefs(x,y,c,m,0.0001)
```

Determining more efficient vehicles - Pseudocode

After obtaining these coefficients, I will now plot a straight line with these coefficients and use the Gradient coefficient to determine whether improvement is needed. If the Gradient coefficient > 0 then the line is upward sloping meaning that emissions are generally rising over time. This is not good and hence improvement is needed. This will be told to the user. If Gradient coefficient < 0 then the line is downward sloping meaning that emissions are generally falling over time. This is good and hence no improvement message is needed.

In both cases however, I will give vehicles that will produce emissions that are less than the most recent month of data entered in. I have created an algorithm that uses the algorithms created in Stage 2 to do this. This is given below:

```
SUB Improve_Emissions(db,coefficients,calculation_method,attributes)

       SUB determine_efficient_vehicles()

              # Get max id of vehicles. Stored as max_id

              Efficient_vehicles = []

              Emission_totals = []

              FOR i ← 1 TO max_id

                     # Obtain vehicle name. Stored as vehicle_name

                     IF calculation_method == 0 THEN

                            Emissions_total = #Call Fuel-based Algorithm from Stage 2

                            IF Emissions_total < y[LEN(y)-1] THEN

                                   Efficient_vehicles.append(Vehicle_name)

                                   Emissions_totals.append(Emission_totals]

                            ENDIF

                     ENDIF

                     IF calculation_method == 1 THEN

                            Emissions_total = # Call Distance-based algorithm from stage 2

                            IF Emissions_total < y[LEN(x)-1] THEN

                                   Efficient_vehicles.append(Vehicle_name)

                                   Emissions_totals.append(Emission_totals]

                            ENDIF

                     ENDIF

              NEXT i

              IF LEN(Efficient_vehicles) == 0 THEN

                     OUTPUT 'Currently, there are no vehicles in the database that are more
efficient in emissions. Try to reduce the distance travelled in your journey.'

              ELSE

                     OUTPUT 'Here are a list of more efficient vehicles with the total emissions
given if this vehicle was to be used.'
```

```
                FOR i ← 0 TO LEN(Efficient_vehicles)
                        OUTPUT Efficient_vehicles[i],': ',Emission_totals[i]
                NEXT i
            ENDIF
        ENDSUB
        IF coefficients[1] > 0 THEN
            OUTPUT 'Over time, your emissions have not been falling. Improvement is required.'
        ELSE
            OUTPUT ''Your emissions are falling over time. However, you can still improve.'
        ENDIF
        Determine_efficient_vehicles()
ENDSUB
```

NB: Coefficients represent the coefficient of the regression line, db represents the database created in Stage 1, x contains the month numbers (1,2,3,…) and y contains the emissions for that month, calculation_method represents the method of calculation wished to be used by the user. 0 is fuel-based, 1 is distance-based. Attributes represents the other attributes needed to call the emissions calculation algorithms from earlier. These are given in the Front End.

Merge Sort Implementation

From the previous section, I have created 2 arrays: efficient_vehicles and emission_totals. Efficient_vehicles store the names of the more efficient vehicles and the emission_totals store the total emissions for each more efficient vehicle. The name of the vehicle and its emission total are linked together by the same index in each array. The reason why I have not made this into a dictionary is because I need to shuffle the vehicle names at the same time as when the emission totals are shuffled so putting them as 2 arrays makes more sense.

The python code for this section is shown below. In the technical solution, this algorithm will fall in the determine_efficient_vehicles function defined in the previous section.

```
def sort_emissions(efficient_vehicles,emission_totals): # Sorts the arrays in ascending order of total emissions
    if len(emission_totals) > 1:
        mid = len(emission_totals) // 2
        left_array_vehicles = efficient_vehicles[:mid] # slices the efficient_vehicles array into the first half
        right_array_vehicles = efficient_vehicles[mid:] # slices the efficient_vehicles array into the second half
        left_array_emissions = emission_totals[:mid] # slices the emission_totals array into the first half
        right_array_emissions = emission_totals[mid:] # slices the emission_totals array into the second half

        sort_emissions(left_array_vehicles,left_array_emissions) # Recursively calls the function on the first half arrays
        sort_emissions(right_array_vehicles,right_array_emissions) # Recursively calls the function on the second half arrays

        i = 0
        j = 0
        k = 0
        # Sorts the elements in each half.
        while i<len(left_array_emissions) and j<len(right_array_emissions): # Moves through each array,incrementing the index if the number in the list is
            if left_array_emissions[i] < right_array_emissions[j]: #bigger
                emission_totals[k] = left_array_emissions[i] # If left array element is bigger, this is placed in that kth position of emission_totals
                efficient_vehicles[k] = left_array_vehicles[i] # Does the same thing with vehicles as the vehicles need to be sorted based on its
                i += 1                                         # emission totals
```

```
        else:
            emission_totals[k] = right_array_emissions[j] # If right array element is >= to left array element,the right array element
                                                          # is placed in kth position of emission_totals
            efficient_vehicles[k] = right_array_vehicles[j] # Does the same thing with vehicles as the vehicles need to be sorted based on its
            j += 1                                          #emission_totals
        k += 1

    # Adding the terms that are left on.
    while i<len(left_array_emissions):
        emission_totals[k] = left_array_emissions[i] # Adds on any large terms left in the left array
        efficient_vehicles[k] = left_array_vehicles[i] # Adds the vehicles based on this.
        i += 1 # increments i and k to add each one
        k += 1

    while j<len(right_array_emissions):
        emission_totals[k] = right_array_emissions[j] # Adds on any large terms left in the right array
        efficient_vehicles[k] = right_array_vehicles[j] # Adds on the vehicles accordingly
        j += 1 # Increments j and k to add each one
        k += 1
```

TEST DESIGN PLAN

Here, I will outline how I will design each section of my code. Each section of my code will be tested differently.

The Table below shows each test I will carry out as well as what I am looking for in each test. The Tests will be carried out in the order of objectives. Parts of the program that relate to all objectives (primarily the databases) have been done first.

Test Number	Test with objective	What is to be demonstrated
1	Testing the presence of Vehicles, Fuel and Refrigerant data	• The test tables should show the correct records and field names as defined in Stage 1. • DB Browser is the software that will be used to visually see the databases.
2	Testing the Validation of the Calculation Method in Objective 1.1	• The appropriate error message is given when the user input is invalid. (Testing 1.1a) • If the input is valid, then the program should go on to objective 1.2
3	Testing the validation for how many months of data are needed in Objective 1.2	• The appropriate error message is given when the user input is invalid. (Testing 1.2a,1.2b,1.2c) • If the user input is valid, the program should go on to objective 2.1/3.1
4	Testing the validation for how many postcodes will be entered by the user in objectives 2.1 and 3.1	• The appropriate error message is given when the user input is invalid. (Testing 2.1a,2.1b,3.1a,3.1b) • If the user input is valid, the program should continue with objective 2.1/3.1
5	Testing the validation for the entry of postcodes in objectives 2.1 and 3.1	• That the appropriate error message is displayed when an invalid address is entered. (Testing 2.1c,3.1c) • If the postcodes are valid, the program should move onto objective 2.2/3.2
6	Testing whether the distance between 2 locations in objectives 2.2 and 3.2 is correct	• That the correct geodesic distance is calculated between 2 postcodes entered by the user. • Also, after it does this, it should move onto objective 2.3/3.3

7	Testing the validation of entry of the name of the vehicle in objectives 2.3 and 3.3	• That the appropriate error message is displayed when an invalid address is entered. (Testing 2.3a) • If the entry is valid, the program should move onto objective 2.4/3.4
8	Testing validation of entering mass of goods purchased in objective 2.4	• That the appropriate error message is displayed when an invalid address is entered. (Testing 2.4a) • If the entry is valid, the program should move onto objective 2.5
9	Testing validation of quantity refrigerant leaked in objective 3.4	• That the appropriate error message is displayed when an invalid address is entered. (Testing 3.4a) • If the entry is valid, the program should move onto objective 3.5
10	Verifying that the emissions calculation for the distance-based method in objective 2.5 is correct	• That the total emissions calculated by the program is the same as the total emissions calculated by hand. • After calculating the emissions, it moves on to the user entering data for the next month, fulfilling objective 2.6
11	Verifying that the emissions calculation for the fuel-based method in objective 3.5 is correct	• That the total emissions calculated by the program is the same as the total emissions calculated by hand. • After calculating the emissions, it moves on to the user entering data for the next month, fulfilling objective 3.6
12	Verifying that data is entered for the number of months entered in by the user in objective 1.2. This tests objectives 2.6 and 3.6	• That the number of months of data entered by the user matches the value entered by the user in objective 1.2. • After this, the program should start to fulfil objectives 2.7/3.7
13	Verifying that all Month data is outputted to the user in objectives 2.7/3.7	• That the month, the data entered by the user and the total emission for that month is outputted to the user as a summary after user input. • The program should then move on to fulfilling objective 4.3
14	Testing the randomisation and stats calculation methods in the scoring algorithm in objectives 4.1 and 4.2	• The databases created have the correct number of records and the correct field names. (Tests that randomisation processes have happened in 4.1a,4.1b,4.1c,4.1d) and that the total emissions have been calculated using the random values testing 4.1e • The values in each cell are valid and within range (shown using DB Browser). Testing 4.1a,4.1b,4.1c,4.1d • The Mean and Standard Deviation calculated by the computer matches my calculation of the Mean and Standard Deviation done by hand. Tests objective 4.2a and 4.2b • I will do this for a small number of rounds with the logic being that if it works for a small no. of rounds, it is bound to work for 1000 rounds. Tests objective 4.1f

15	Verifying that the Grade Boundaries for scoring are fair. Testing Objective 4.3 and 4.4	• The diagrams of the normal distributions must show the Grade boundaries. Tests 4.4b • This must show grade boundaries that are positively determined with the peak of the normal distribution leading to a B or C grade. Tests 4.3b.
16	Checking that the range of x values are correct when plotting the normal distribution.	• Like Test 14, I will use a small number of rounds and show that it works correctly meaning that it works for many rounds.
17	Verifying that the Grade determined by the Program matches the grade boundaries. Testing Objective 4.3	• That the grade boundaries should correctly imply the grade given by the program.
18	Testing that the Normal Distribution has all the required features. Testing Objective 4.4	• The Normal Distribution shows all the total emissions calculated based on the user input. Tests 4.4a • The x axis should say “Emission values” and the y axis should say “normally rationalised score” • The Grade Boundaries should be correctly placed on the normal distributions. Tests 4.4b
19	Testing that a Graph is shown to the user showing the required features. Testing objective 5.1	• The Graph should show ‘months’ on the x axis and ‘emissions’ on the y axis. • The emissions of each month should be plotted as points on the graph and a trend line should be running through it showing the overall trend of emissions. (Tests 5.1b) • Also, the title should be ‘Trend of Emissions’
20	Testing that the program picks the more accurate trend line from the 2 different methods based on the r-squared score. Testing Objective 5.1.	• The Trend line should be an as accurate representation of the data as possible (Testing 5.1a) • The trend line with the r-squared score that is closest to 1 is the trend line outputted by the user.
21	Testing that the Program should correctly identify whether emissions are generally rising or falling over time. Testing Objective 5.2	• Showing that based on the trend line, the program can identify whether emissions are rising or falling over time.
22	Testing that the Program gives the correct vehicles that produce lower emissions. Testing Objective 5.3	• Showing that the program gives vehicles that produce lower emissions by doing them by hand and comparing them with the program. • If no vehicles are more efficient, then the user must be told to minimize the distance travelled. (Testing 5.3b)
23	Testing that the Program outputs the vehicles in ascending order of total emissions. Testing Objective 5.4	• That the merge sort is working correctly. (Testing 5.3a) • That the output of more efficient vehicles is in ascending order of total emissions.
24	Videos of both the scoring model and the Front End. Testing all objectives	• Shows all my code working from the front end with no errors. • That my scoring model runs correctly with no errors.

LIBRARIES USED

Below shows the list of libraries that I have used along with where I aim to use them in my technical solution and why I am using it.

Library	Uses in Implementation
Sqlite3	• To create the database with tables and relationships defined in Stage 1 of Design. • Use Aggregate SQL Functions like AVG and MAX to do maths with the data in the databases. • Using Parametrised SQL Queries to put variables from my code into SQL Queries. • Used in all Stages of Design and objectives • (SQlite, 2004)
Geopy – Nominatim and Distance	• To calculate the distance between 2 postcodes entered in by the user. • Used in Stage 2 of Design and objective 2.2 and 3.2 • (Geopy, 2020)
Numpy	• Numpy arrays are used in Stages 3,4 and 5 of design and Objectives 4 and 5. • This is to make the amount of code written less as numpy arrays allow mathematical operations to be performed on them, something normal arrays cannot do. • (Scipy, 2020) (includes numpy)
Matplotlib.pyplot	• Used for objectives 4.4 and 5.2. to plot the normal distribution as well as where each month falls on it and the linear regression line on the scatter graph to show the overall trend of emissions over time. • Used in Stage 4 and 5 of design • (Scipy, 2020)
SciPy. Stats - norm	• Used in objectives 4.3 and 4.4 to calculate where an emissions total falls on the normal distribution as well as to determine the grade boundaries and plot the normal distribution. • (Scipy, 2020)
Datetime	• Used to determine a random starting value for my linear congruential generator in Objective 3.1 • (Datetime, 2021)

SQL STATEMENTS USED

Below shows the SQL Statements I intend to use in my code. Some are used in multiple places.

SQL Statement	Use
CREATE TABLE IF NOT EXISTS Fuel ()	• To create the fuel table defined in Stage 1 of Design • Details of the fields have been left out as these have already been mentioned in Stage 1 of Design

DROP TABLE IF EXISTS Fuel	• When initializing the database, all data needs to be removed so hence the table is deleted before the new table is created.
CREATE TABLE IF NOT EXISTS Refrigerant ()	• To create the Refrigerant table defined in Stage 1 of Design • Details of the fields have been left out as these have already been mentioned in Stage 1 of Design
DROP TABLE IF EXISTS Refrigerant	• When initializing the database, all data needs to be removed so hence the table is deleted before the new table is created.
CREATE TABLE IF NOT EXISTS Vehicles ()	• To create the Vehicles table defined in Stage 1 of Design • Details of the fields have been left out as these have already been mentioned in Stage 1 of Design
DROP TABLE IF EXISTS Vehicles	• When initializing the database, all data needs to be removed so hence the table is deleted before the new table is created.
CREATE TABLE IF NOT EXISTS ScoresEmissionsFuel()	• To create the ScoresEmissionsFuel table defined in Stage 1 of Design • Details of the fields have been left out as these have already been mentioned in Stage 1 of Design
DROP TABLE IF EXISTS ScoresEmissionsFuel	• When initializing the database, all data needs to be removed so hence the table is deleted before the new table is created.
CREATE TABLE IF NOT EXISTS ScoresEmissionsDistance (	• To create the ScoresEmissionsDistance table defined in Stage 1 of Design • Details of the fields have been left out as these have already been mentioned in Stage 1 of Design
DROP TABLE IF EXISTS ScoresEmissionsDistance	• When initializing the database, all data needs to be removed so hence the table is deleted before the new table is created.
INSERT INTO Fuel VALUES	• Adds in the test fuels to be used in my technical solution. • The values have been left out as these have already been stated in Stage 1 of Design
INSERT INTO Refrigerant VALUES	• Adds in the test Refrigerant to be used in my technical solution. • The values have been left out as these have already been stated in Stage 1 of Design
INSERT INTO Vehicles VALUES	• Adds in the test Vehicles to be used in my technical solution. • The values have been left out as these have already been stated in Stage 1 of Design
SELECT Fuel.emission_factor FROM Vehicles INNER JOIN Fuel ON Vehicles.fuel_id = Fuel.fuel_id WHERE vehicle_name=?	• To get the fuel emission factor in Objectives 2.3 and 3.3. Used in Stage 2 of Design

SELECT gwp FROM Refrigerant WHERE refrigerant_name=Refrigerant_name	• To get the global warming potential of a refrigerant used by the vehicle. Used in Objectives 2.3 and 3.3 and in Stage 2 of Design
SELECT fuel_efficiency FROM Vehicles WHERE vehicle_name=Vehicle_name	• To get the fuel efficiency of a vehicle entered in by the user. Used in Objective 2.3 and in Stage 2 of Design
SELECT MAX(vehicle_id) FROM Vehicles	• Used in the process of randomly picking a vehicle from the database in Stage 3 of Design. Used in Objective 4.2. • Used in Objective 5.4 so that the program can go through all vehicles and see if any of them produce lower emissions. Hence, used in Stage 5 of design • Used in Objectives 2.3 and 3.3 to validate the vehicle brand entered by the user.
SELECT vehicle_name FROM Vehicles WHERE vehicle_id=Vehicle_id	• Used in Objective 4.2 to get the vehicle name from the random vehicle id chosen in a randomisation process. This allows the total emissions to be calculated for this randomisation round as the vehicle name is one of the parameters. • Used in Objectives 2.3 and 3.3 to set the _vehicle_name attribute if the input entered by the user is valid. • Used in Objective 5.4 to determine the name of each vehicle to calculate the total emissions with that vehicle.
SELECT Refrigerant.refrigerant_name FROM Vehicles INNER JOIN Refrigerant ON Vehicles.refrigerant_id=Refrigerant.refrigerant_id WHERE Vehicles.vehicle_name=?	• To find the refrigerant name of the refrigerant that the vehicle uses. Used in objectives 2.5 and 3.5 and in Stage 2 and 3 of design. • Used in objective 4.1.
INSERT INTO ScoresEmissionsFuel VALUES emissions_data	• Inserts all the randomly generated parameters into the ScoresEmissionsFuel Database in Objective 4.2
INSERT INTO ScoresEmissionsDistance VALUES emissions_data	• Inserts all the randomly generated parameters into the ScoresEmissionsDistance Database in Objective 4.2
SELECT AVG(total_emissions) FROM ScoresEmissionsFuel	• Determines the mean of the total emissions in Objective 4.2 in the Fuel-based Method.
SELECT total_emissions FROM ScoresEmissionsFuel WHERE round=i+1	• Used in Objective 4.2 to help calculate the Standard Deviation of the emissions totals in the Fuel-based Method.
SELECT MAX(round) FROM ScoresEmissionsFuel	• Used to determine the number of randomisation rounds carried out in Objective 4.2. for the fuel-based method
SELECT AVG(total_emissions) FROM ScoresEmissionsDistance	• Determines the mean of the total emissions in Objective 4.2 in the Distance-based Method.
SELECT MAX(round) FROM ScoresEmissionsDistance	• Used to determine the number of randomisation rounds carried out in Objective 4.2. for the fuel-based method

SELECT total_emissions FROM ScoresEmissionsDistance WHERE round=i+1	• Used in Objective 4.2 to help calculate the Standard Deviation of the emissions totals in the Distance-based Method
SELECT vehicle_name FROM vehicles	• Used in Objectives 2.3 and 3.3 to get all the vehicle names outputted to the user so the user can enter in the correct vehicle
SELECT vehicle_id FROM Vehicles WHERE vehicle_name=Vehicles[0]	• Finds the vehicle id of each vehicle to help in outputting all the vehicles to the user in Objectives 2.3 and 3.3
SELECT refrigerant_id FROM Vehicles WHERE vehicle_name=self._vehicle_name	• Get the refrigerant of a vehicle entered by the user in Objectives 2.3 and 3.3
SELECT MAX(total_emissions) FROM ScoresEmissionsFuel	• Finds the range of emission values that need to be plotted on the fuel-based method normal distribution. Used in Objective 4.4
SELECT MAX(total_emissions) FROM ScoresEmissionsDistance	• Finds the range of emission values that need to be plotted on the distance-based method normal distribution. Used in objective 4.4

Implementation

Overall, I have created 6 python files for my code. One for the Front end of the program which shows all my OOP, 2 for the database creation, one for calculating the total emissions using the 2 different methods, one containing the linear regression algorithm and one for the scoring algorithm. Below shows the commented code in each file. I have decided to explain all the key algorithms and models in this section. Those being the calculating emissions algorithm, LCG Algorithm, the linear regression algorithms, the algorithm determining the more efficient vehicles and the front end.

CALCULATING TOTAL EMISSIONS

First, I will show the distance-calculation algorithm. This is because this is calculated beforehand when the user enters addresses in as a means of testing validity of those addresses. This is shown below:

```
# Functions used in both calculation methods

def CalculateDistance(postcodes): # Calculates the distance between each consecutive pair of postcodes.

    geolocator = Nominatim(user_agent="CalculateEmissions")

    total_distance = 0

    for i in range(len(postcodes)-1):

        postcode1 = postcodes[i]
        postcode2 = postcodes[i+1]

        postcode1 = geolocator.geocode(postcode1) # Finds coordinates for both postcodes
        postcode2 = geolocator.geocode(postcode2)

        postcode1_lat = (postcode1.latitude) # Gets latiutude and longitude of both postcodes
        postcode1_lon = (postcode1.longitude)
        postcode2_lat = (postcode2.latitude)
        postcode2_lon = (postcode2.longitude)

        location1 = (postcode1_lat, postcode1_lon) # Summarises location of 2 postcodes
        location2 = (postcode2_lat, postcode2_lon)

        distance_between_2_locations = distance.distance(location1, location2).km # Finds distance between 2 locations in km
        total_distance += distance_between_2_locations # Adds it to the total distance travelled.

    total_distance = round(total_distance,4)

    return total_distance
```

The distance goes between adjacent addresses calculating the distance between them. This is done by finding the latitude and longitude positions of each address and then calculating the geodesic distance between them using the geopy library.

Now, I will show how the total emissions are calculated.

```
# Interface

def CalculateEmissions(calculation_method,attributes):
    if calculation_method == 0:
        total_emissions = fuel_based(attributes[0],attributes[1],attributes[2],attributes[3],attributes[4])
        return total_emissions
    else:
        total_emissions = distance_based(attributes[0],attributes[1],attributes[2],attributes[3],attributes[4])
        return total_emissions
```

This function is called in the front end. The attributes that are called vary on the calculation method (0 is fuel-based and 1 is distance-based). The fuel-based attributes are the database, vehicle name, distance travelled (pre-determined), refrigerant name and the quantity refrigerant leaked. The distance-based attributes are the database, vehicle name, distance travelled, refrigerant name and the mass of goods purchased. The fuel_based function and distance_based functions respectively are now shown below:

```
# Overall Functions

def fuel_based(db,vehicle_name,distance,refrigerant_name,quantity_refrigerant_leaked):
    emission_factor = GetEmissionFactorForFuel(db,vehicle_name)
    gwp = GetGWP(db,refrigerant_name)
    fuel_efficiency = GetFuelEfficiency(db,vehicle_name)
    total_emissions = CalculateTotalEmissions_fuel(distance,fuel_efficiency,emission_factor,quantity_refrigerant_leaked,gwp)
    return total_emissions

def distance_based(db,vehicle_name,distance,refrigerant_name,mass_goods_purchased):

    emission_factor_fuel = GetEmissionFactorForFuel(db,vehicle_name)
    gwp = GetGWP(db,refrigerant_name)
    total_emissions = CalculateTotalEmissions_distance(vehicle_name,distance,mass_goods_purchased,emission_factor_fuel,gwp)
    return total_emissions
```

These follow the formulas specified in analysis. The GetGWP function is shown below. It is just a sql query.

```
def GetGWP(db:sqlite3.Connection, refrigerant_name): # Gets global warming potential of refrigerant used.

    c = db.cursor()
    gwp_object = c.execute("SELECT gwp FROM Refrigerant WHERE refrigerant_name=?", (refrigerant_name,))
    gwp = c.fetchone()[0]
    return gwp
```

The GetFuelEfficiency function is shown below. Again, it is just a sql query.

```
# Fuel-Based Method Calculations

def GetFuelEfficiency(db:sqlite3.Connection, vehicle_name): # Fetches fuel efficiency of vehicle

    c = db.cursor()
    fuel_efficiency_object = c.execute("SELECT fuel_efficiency FROM Vehicles WHERE vehicle_name=?",(vehicle_name,))
    fuel_efficiency = c.fetchone()[0]
    return fuel_efficiency
```

The GetEmissionFactorForFuel function is shown below, and it is an INNER JOIN sql query.

```
def GetEmissionFactorForFuel(db:sqlite3.Connection, vehicle_name): # Gets the emission factor for the fuel vehicle uses

    c = db.cursor()
    sql = '''SELECT Fuel.emission_factor FROM Vehicles
             INNER JOIN Fuel ON Vehicles.fuel_id = Fuel.fuel_id
             WHERE vehicle_name=?'''

    emission_factor_object = c.execute(sql,(vehicle_name,))
    emission_factor = c.fetchone()[0]
    return emission_factor
```

Then the calculating emissions functions CalculateTotalEmissions_fuel and CalculateTotalEmissions_distance use the formulae specified in analysis and are shown below:

```
def CalculateTotalEmissions_fuel(distance,fuel_efficiency,Emission_factor_fuel,quantity_refrigerant_leaked,gwp):

    QuantityFuelConsumed = distance * fuel_efficiency
    Emissions_Fuel = QuantityFuelConsumed * Emission_factor_fuel
    Emissions_Refrigerant = quantity_refrigerant_leaked * gwp
    total_emissions = Emissions_Fuel + Emissions_Refrigerant
    return total_emissions

# Distance-based method calculations

def CalculateTotalEmissions_distance(vehicle_name,distance,mass_goods_purchased,emission_factor_fuel,gwp):

    emission_factor_vehicle = emission_factor_fuel + gwp
    total_emissions = mass_goods_purchased*distance*emission_factor_vehicle
    return total_emissions
```

All these functions have been specified in design in pseudocode and this section shows how they have been implemented.

SCORING CODE

Below I will explain both the scoring model and the algorithm to determine the grade.

Scoring Model

The Linear Congruential Generator Algorithm is shown below. This includes Stein's Algorithm as well.

```
def LCG(divisor): #Implements the Linear Congruential Generator
    def find_coprime(divisor):
        def gcd(a,b): # Stein's Algorithm
            if (a == b):
                return a
            # GCD(0, b) == b; GCD(a, 0) == a,
            # GCD(0, 0) == 0
            if (a == 0):
                return b
            if (b == 0):
                return a
            # look for factors of 2
            # a is even
            if ((~a & 1) == 1):
                # b is odd
                if ((b & 1) == 1):
                    return gcd(a >> 1, b)
                else:
                    # both a and b are even
                    return (gcd(a >> 1, b >> 1)) << 1
            # a is odd, b is even
            if ((~b & 1) == 1):
                return gcd(a, b >> 1)
            # reduce larger number
            if (a > b):
                return gcd((a - b) >> 1, b)
            return gcd((b - a) >> 1, a)

        found = False
        i = 2
        while found is False:
            hcf = gcd(i,divisor)
            if hcf == 1: # Stops at the snmallest number coprime to the divisor.
                found = True
            else:
                i += 1
        return i
```

```
def determine_a(divisor): # Determines the smallest value of a for any divisor for my LCG algorithm
    prime_factors = []
    found = False
    factor = 2
    while found is False: # Finds all prime factors of the divisor
        all_one_prime_factor = False
        while all_one_prime_factor is False: # If a prime is a factor, then it checks how many powers of this prime are factors of the divisor.
            if divisor % factor == 0:
                divisor /= factor # it divides the divisor by that prime factor to ensure no prime factors repeat.
                prime_factors.append(factor) # Adds the prime factor to the prime factor list
            else:
                all_one_prime_factor = True # Leaves the while loop if there are no more prime factors
        if divisor == 1: # If all the prime factors are found, you leave the loop
            found = True
        else: # If not, you go onto the next number.
            factor += 1

    num_factors_2 = prime_factors.count(2) # Counts the number of factors of 2
    factors_of_a = list(dict.fromkeys(prime_factors)) # Finds the factors of a by removing repeats from the list. We need a to be divisible by all the prime
    if num_factors_2 >= 2:                            # factors of the divisor.
        a = int(np.prod(factors_of_a) * 2) # Multiplies by an extra factor of 2 if the divisor is a multiple of 4
    else:
        a = int(np.prod(factors_of_a))
    return a+1

date = datetime.datetime.now().strftime('%M:%S.%f') # Determines the number of times the recurrence relation runs by evaluating the no. of microseconds
num_iterations = int(date[9:])                      # Time is at during that minute and taking the last 3 numbers of that microsecond.

a =  determine_a(divisor) # Determines an a value such that a-1 is divisible by 4
c = find_coprime(divisor)  # divisor and c must be coprime to get the largest period
x = int(date[8])  # Randomly determines a starting x valueD
for i in range(num_iterations):
    x = (a*x+c)%(divisor+1)   # Computes the Recurrence relation
if x == 0: # As the random number can never be 0.
    x += 1
return x
```

First, the algorithm finding the number of microseconds and the number of seconds the timer is at when it is run. This determines the number of iterations my recurrence relation runs for (number of microseconds) at random. Also, it helps me determine the initial x value. I have chosen to use the time as this is completely random which makes my Monte Carlo Method much more reliable. I have decided to use functions within functions here because it helps me understand where different functions are used, and it ensures that some functions are kept local to specific parts of the program.

The ‘a’ value in my recurrence relation is determined through the determine_a function. As explained earlier, I have decided to find a number such that a-1 is divisible by all factors of m and is divisible by 4 if m is divisible by 4. This is done by first finding all the prime factors of m, going through each factor from 2 upwards. If a number is a factor, it is added to the list of prime factors and the divisor is divided by that factor. Then, you find factors of the new divisor starting with that same factor you just divided to account for repeated prime factors. If a number is not a factor, then the program moves onto the next factor. The program can iterate through all the numbers instead of prime factors because the program is dividing by each factor as it finds it and as composite numbers can all be written as a product of prime numbers; it follows that all the composite numbers will be rejected anyway. After the prime factors have been identified, it counts the number of factors of 2. If there is more than 1 factor of 2, then the divisor has a factor of 4. If the divisor isn’t divisible by 4, then all the program needs to do is ensure that a-1 is a factor of the divisor. Hence, the program removes all repeats of prime factors and multiplies them all together. If the divisor is

divisible by 4, the program does the same but multiplies by 2 in determining the a value to ensure that a-1 is also divisible by 4. Then, it adds 1 to my a value to ensure that a-1 satisfies the necessary conditions.

The c value is determined using the find_coprime function which finds the smallest number that is coprime to the divisor. This is where Stein's Algorithm is used. A while loop is called; iterating through each integer, checking if the greatest common divisor (gcd) between that integer and the divisor is 1. If so, the loop is broken and that is the c value. If not, the next integer is checked. Stein's algorithm determines whether the gcd is 1 by first checking whether the 2 inputs are both even, both odd or even and odd. This is checked, not using the remainder operator but using AND and NOT operators. Then, it calls the algorithm again recursively with modified inputs (modified using binary shifts) which are modified based on the oddness and evenness of the 2 numbers. This keeps occurring until you get one of the modified inputs to be 0. Then, the other input is returned, and this is the gcd.

This LCG algorithm is called for all the randomisation processes required as stated in design. The emissions total is calculated, and all the data is added to a table in my database. Then, the stats for the Normal Distribution are then calculated.

Code to Plot Graphs

```
# Will be called in Front End. Functions and Procedures above won't be called in Front End

def DetermineEmissionsScore(db,calculation_method,User_Total_Emissions):

    # Assemble Probability Distribution Function (PDF) and create Graph. Also Marks on the emissions from the user-enterded data.
    # Also returns score for each emissions total from User_Total_Emissions

    def PlotGraph(x_values,Mean,SD,User_Total_Emissions):


      def DetermineScoreBoundaries(Grades,Mean,SD): # Determines Score Boundaries

          num_grades = len(Grades)
          cum_zero = norm(loc=Mean,scale=SD).cdf(0) # Determines the probability of the emissions score being < 0
          remaining_p = 1 - cum_zero # Determines probability of the emissions score being >= 0
          gap = remaining_p/num_grades # Splits the probability of emissions score being >= 0 into 5 equal chunks.
          score_boundaries = [cum_zero+gap,cum_zero+(2*gap),cum_zero+(3*gap),cum_zero+(4*gap)] # Sets gap as the proportion of all the possible emission scores
          score_boundaries_x = []                                                              # falling under 1 grade. Sets score boundaries in terms of
                                                                                               # probability


          for i in range(len(score_boundaries)):
              z_x_value = norm.ppf(score_boundaries[i]) # Determines the total emissions value that each score boundary comes from.
              x_value = (z_x_value*SD)+Mean # Adjusts the above value to fit my normal distribution.
              score_boundaries_x.append(x_value)
          score_boundaries_x.append(x_values[len(x_values)-1])
          return score_boundaries_x
```

```
251
252     def DetermineGrade(Grades,score_boundaries_x,User_Total_Emissions): # Determines Grade for User
253         print()
254         print('Grades for each month: ')
255         print('----------------------')
256         print()
257         for Month in range(len(User_Total_Emissions)):
258             Grade_found = False
259             index = 0
260             Grade = ''
261 
262             while Grade_found is False: # Indexes through each score boundary to see if each one exceeds the emission total
263                 if score_boundaries_x[index] >= User_Total_Emissions[Month]: # Sees which is the first score boundary to be greater than the emission total
264                     Grade = Grades[index] # This becomes the grade.
265                     print()
266                     print('Month ',(Month+1))
267                     print('-------')
268                     print()
269                     print('Total Emissions: ', User_Total_Emissions[Month],' CO2e')
270                     print('Grade: ' + Grade)
271                     print()
272                     Grade_found = True
273                 else:
274                     index += 1  # Moves onto the next score boundary
275 
276         plt.ion()
277         y_values = norm.pdf(x_values,Mean,SD)
278         image,axes = plt.subplots(figsize=(10,8))
279         plt.style.use('fivethirtyeight')
280         axes.plot(x_values,y_values,'b',linewidth=2)
281 
282         axes.fill_between(x_values,y_values,0, alpha=0.1, color='b') # Plots on x labels, y labels, x range,y range and title of graph.
283         axes.set_xlabel('Emission Values')
284         axes.set_ylabel('Normally rationalised score')
285         axes.set_xticks(np.arange(0,x_values[len(x_values)-1],x_values[len(x_values)-1]/10))
286         axes.set_yticks(np.arange(0,max(y_values),max(y_values)/10))
287         axes.set_title('How emissions fare on Distribution curve')
288 
289         # Labels on emissions based on user-entered data
290         for i in range(len(User_Total_Emissions)):
291 
292             x_user_emissions = User_Total_Emissions[i]
293             y_user_emissions = float((norm.pdf(x_user_emissions,Mean,SD)))
294             Month = 'Month ' + str(i+1)
295             plt.scatter(x_user_emissions,y_user_emissions,marker = 'x',alpha=1)
296 
297             plt.annotate(Month,(x_user_emissions,y_user_emissions))
298 
299         # Determine Score Boundaries
300         Grades = ['A','B','C','D','E']
301         score_boundaries_x = DetermineScoreBoundaries(Grades,Mean,SD)
302         # Determines Grade
303         DetermineGrade(Grades,score_boundaries_x,User_Total_Emissions)
```

```
303
304         # Shows Graph to user
305         for i in range(len(score_boundaries_x)):
306             plt.axvline(x=score_boundaries_x[i],color='r',linewidth=0.9)
307             if i == 0:
308                 plt.annotate(Grades[i],(score_boundaries_x[i]/2,0)) # Plots on labels denoting grade regions. Label is placed in the middle bottom of each region
309             else:
310                 plt.annotate(Grades[i],(score_boundaries_x[i]-(score_boundaries_x[i]-score_boundaries_x[i-1])/2,0))
311         plt.show()

312
313     def DetermineMaxEmissions(db,calculation_method): # Determines the maximum total emissions from the randomisation processes
314             c = db.cursor()
315             if calculation_method == 0: # Done to determine the range of emissions values needed to plot the normal distributions.
316                 max_emissions_object = c.execute('SELECT MAX(total_emissions) FROM ScoresEmissionsFuel')
317                 max_emissions = c.fetchone()[0]
318                 return max_emissions
319             else:
320                 max_emissions_object = c.execute('SELECT MAX(total_emissions) FROM ScoresEmissionsDistance')
321                 max_emissions = c.fetchone()[0]
322                 return max_emissions
323
324     max_emissions = DetermineMaxEmissions(db,calculation_method) # Determines maximum emissions so the range of x values is plotted properly.
325     if calculation_method == 0:

326
327         # Mean: 1401.2978889861004
328         # SD: 1578.0315281896908
329         Mean = 1401.2978889861004
330         SD = 1578.0315281896908
331         x_values = np.arange(0,max_emissions,0.5) # Determines x values to be plotted
332         PlotGraph(x_values,Mean,SD,User_Total_Emissions) # Plots the Graph
333
334     if calculation_method == 1:
335
336         # Mean: 2740269.4551994572
337         # SD: 2735613.7940128436
338         Mean = 2740269.4551994572
339         SD = 2735613.7940128436
340         x_values = np.arange(0,max_emissions,2) # Determines x values to be plotted
341         PlotGraph(x_values,Mean,SD,User_Total_Emissions) # Plots the Graph
```

Again, I have opted for the functions in functions structure for similar reasons as for the scoring model. When the DetermineEmissionsScore function is called, it first determines the maximum possible emissions one can have as determined by my scoring model. This is done in DetermineMaxEmissions and is done to get the largest emissions value I need to plot on the x axis of the normal distribution. Then, the mean and standard deviation determined by the scoring model are declared by the variables Mean and SD and the x-values for the normal distribution are determined using the maximum possible emissions value with a step of 0.5 to get an accurate Normal Distribution. Then, the PlotGraph function is called which plots the graph. In PlotGraph, the y values for each of the x-values on the Normal Distribution are found and then, the appearance settings of the graph are determined. This includes the values shown on the x axis and y axis which are found by dividing the largest x or y value by 10 and going from 0 up until this value with the largest x or y value divided by 10 as the step. Then, the total emissions obtained from the user data are plotted as points on the Normal Distribution, first by calculating where it falls on the Normal Distribution. Then, the grade boundaries are determined, using the implemented version of the grade boundaries algorithm created in design. After this the Grade is determined using the user-defined algorithm I have created in design and then, the overall graph is presented to the user.

GRAPHING CODE

Below shows the linear regression algorithm to determine the most accurate trend line.

```
def Least_Squares(x,y):
    # Sample Size
    n = np.size(x)

    # Calculating Gradient and y intercept of line
    m_x = np.mean(x) # Calculates Mean
    m_y = np.mean(y) # Calculates Standard Deviation

    SS_xy = np.sum(y*x) - n*m_y*m_x # Calculates SSxy
    SS_xx = np.sum(x*x) - n*m_x*m_x # Calculates SSxx

    gradient = SS_xy / SS_xx # Determines gradient
    intercept = m_y - gradient*m_x # Determines y intercept
```

```
def Gradient_Descent(x,y):
    def OptimiseCoefs(x,y,c,m,L): # x is independent variable, y is dependent variable, c is y intercept and m is gradient of trend line, L is learning rate
        epochs = 1000 # nunber of iterations
        for i in range(epochs):
            y_pred = m*x + c
            D_m = (-2/len(x))*np.sum(x*(y-y_pred)) # Partial Derivative with respect to m
            D_c = (-2/len(x))*np.sum(y-y_pred) # Partial Derivative with respect to c
            m -= (L*D_m) # Updates m
            c -= (L*D_c) # Updates c
        return [c,m]

    m = (y[len(y)-1]-y[0])/(x[len(x)-1]-x[0]) # Determines starting m and c values.
    c = y[0] - m*x[0]
    return OptimiseCoefs(x,y,c,m,0.0001)
```

```
def calculate_r_squared(x,y,coefficients): # Calculate R Squared value.

    y_pred = x*coefficients[1] + coefficients[0] # Determines the predicted y.
    mean_y = np.mean(y) # Finds the mean of y
    SSE = np.sum((y-y_pred)**2)
    SSyy = np.sum((y-mean_y)**2)
    r_squared = 1 - (SSE/SSyy) # Calculates the R-Squared score using the formula.
    return r_squared
```

```
def Graph_and_Improve(db,x,y,calculation_method,attributes): # Interface

    coefficients = Gradient_Descent(x,y) # Determines coefficients from gradient descent
    coefficients1 = Least_Squares(x,y) # Determines coefficients from least squares.
    r_squared = calculate_r_squared(x,y,coefficients)
    r_squared_1 = calculate_r_squared(x,y,coefficients1)

    if abs(r_squared-1)<abs(r_squared_1-1): # Determines which set of coefficients are more accurate and plots the more accurate trend line.
        plot_regression_line(x,y,coefficients) # Plots Gradient Descent Trend Line
        improve_emissions(db,coefficients,x,y,calculation_method,attributes)
    else:
        plot_regression_line(x,y,coefficients1) # Plots Least Squares Trend line
        improve_emissions(db,coefficients1,x,y,calculation_method,attributes)
```

After the Total emissions have been calculated and scored for each month, the Graph_and_Improve method is called to determine an accurate trend line for emissions over time and determine more efficient vehicles. To determine the most accurate trend line, sets of coefficients for a possible trend line using the gradient descent and least squares methods. First, the gradient descent method is used. In the function Gradient_Descent, a rough gradient and y intercept are first found by taking the first and last points and finding the slope and intercept of that line. This reduces the probability that my starting value doesn't cause my algorithm to converge to the wrong local minima that isn't the minimum of my error function. I have used a function within a function here because I want OptimiseCoefs to be local to the Gradient_Descent function as it doesn't get used anywhere else. This makes it more helpful for me as it helps me remember where each function is used. The Least Squares Method algorithm uses the formulae described in Analysis and the pseudocode in Design. Then, the r-squared score is calculated for each set of coefficients obtained from each optimisation method using the formula stated in Design. The set of coefficients that yields an r-squared score closer to 1 is the more accurate trend line resulting in the trend line with those coefficients being drawn.

The algorithm to determine more efficient vehicles is shown below:

```
60 def improve_emissions(db,coefficients,x,y,calculation_method,attributes): # Algorithm determining more efficient vehicles.
61     def determine_efficient_vehicles():
62         def sort_emissions(efficient_vehicles,emission_totals): # Sorts the arrays in ascending order of total emissions
63             if len(emission_totals) > 1:
64                 mid = len(emission_totals) // 2
65                 left_array_vehicles = efficient_vehicles[:mid] # slices the efficient_vehicles array into the first half
66                 right_array_vehicles = efficient_vehicles[mid:] # slices the efficient_vehicles array into the second half
67                 left_array_emissions = emission_totals[:mid] # slices the emission_totals array into the first half
68                 right_array_emissions = emission_totals[mid:] # slices the emission_totals array into the second half
69 
70                 sort_emissions(left_array_vehicles,left_array_emissions) # Recursively calls the function on the first half arrays
71                 sort_emissions(right_array_vehicles,right_array_emissions) # Recursively calls the function on the second half arrays
```

```
72 
73             i = 0
74             j = 0
75             k = 0
76             # Sorts the elements in each half.
77             while i<len(left_array_emissions) and j<len(right_array_emissions): # Moves through each array,incrementing the index if the number in the list is
78                 if left_array_emissions[i] < right_array_emissions[j]: #bigger
79                     emission_totals[k] = left_array_emissions[i] # If left array element is bigger, this is placed in that kth position of emission_totals
80                     efficient_vehicles[k] = left_array_vehicles[i] # Does the same thing with vehicles as the vehicles need to be sorted based on its
81                     i += 1                                         # emission totals
82                 else:
83                     emission_totals[k] = right_array_emissions[j] # If right array element is >= to left array element,the right array element
84                                                                   # is placed in kth position of emission_totals
85                     efficient_vehicles[k] = right_array_vehicles[j] # Does the same thing with vehicles as the vehicles need to be sorted based on its
86                     j += 1                                          #emission_totals
87                 k += 1
```

```
                # Adding the terms that are left on.
                while i<len(left_array_emissions):
                    emission_totals[k] = left_array_emissions[i] # Adds on any large terms left in the left array
                    efficient_vehicles[k] = left_array_vehicles[i] # Adds the vehicles based on this.
                    i += 1 # increments i and k to add each one
                    k += 1

                while j<len(right_array_emissions):
                    emission_totals[k] = right_array_emissions[j] # Adds on any large terms left in the right array
                    efficient_vehicles[k] = right_array_vehicles[j] # Adds on the vehicles accordingly
                    j += 1 # Increments j and k to add each one
                    k += 1
```

```
        c = db.cursor()
        max_id_object = c.execute('SELECT MAX(vehicle_id) FROM Vehicles')
        max_id = c.fetchone()[0]
        efficient_vehicles = []
        emission_totals = []

        for i in range(max_id): # Goes through each vehicle calculating the total emissions with each one. Vehicle is added to the list if total emissions < most
            vehicle_name_object = c.execute('SELECT vehicle_name FROM Vehicles WHERE vehicle_id=?',(i+1,)) # recent data.
            vehicle_name = c.fetchone()[0]

            if calculation_method == 0:
                emissions_total = CalculatingEmissions.fuel_based(db,vehicle_name,attributes[0],attributes[1],attributes[2])#Calculates emissions with each vehicle
                if emissions_total < y[len(x)-1]: # If total emissions < most recent emissions total calculated, then it is added to the efficient vehicles list.
                    efficient_vehicles.append(vehicle_name) # The emissions total using that vehicle is added to the emission totals list.
                    emission_totals.append(emissions_total)

            if calculation_method == 1: # Same as the previous bit with with the distance-based method instead of fuel-based method.
                emissions_total = CalculatingEmissions.distance_based(db,vehicle_name,attributes[0],attributes[1],attributes[2])
                if emissions_total < y[len(x)-1]:
                    efficient_vehicles.append(vehicle_name)
                    emission_totals.append(emissions_total)

        if len(efficient_vehicles) == 0: # Outputs more efficient vehicles to user. If none exist, user is told to minimise distance travelled.
            print('Currently, there are no vehicles in the database that are more efficient in emissions. Try to reduce the distance travelled in your journey.')
        else:
            sort_emissions(efficient_vehicles,emission_totals) # Sorts the vehicles and emisison totals is ascending order of emission totals.
            print('Here are a list of more efficient vehicles (in ascending order) with the total emissions given if this vehicle was to be used.')
            for i in range(len(efficient_vehicles)):
                print()
                print(efficient_vehicles[i],': ',round(emission_totals[i],2), ' CO2e')
```

```
        if coefficients[1] >= 0: # Determines which is the appropriate message to give to the user.
            print('Over time, your emissions have not been falling. Improvement is required.')
            print()

        else:
            print('Your emissions are falling over time. However, you can still improve.')
            print()
        determine_efficient_vehicles()
```

I have decided to use functions within functions because I want some functions to be local to one part of the program to ensure that it can't be called by accident. Also, like with the Gradient_Descent and OptimiseCoefs, it is helpful for me to understand where each function is being used. The improve emissions algorithm is implemented using the pseudocode in Design with the emissions total being calculated for each possible vehicle in the database

and comparing it with the most recent emissions total. If it is less than the most recent emissions total, the vehicle name is added to 1 list and the emission total is added to another. After this is done for all vehicles, the list of more efficient vehicles and their emissions totals list are then sorted in ascending order of emissions total in the sort_emissions function. This recursively slices the emissions array into 2 until there is only 1 element in each array. Then, it pieces the emissions array back together sorting the array along the way. Based on this, the vehicle names array is sorted as the vehicle names and emissions array are linked by having each index having 1 vehicle name and 1 emissions total. After this, the list of vehicles and their corresponding emission totals are outputted for the user to see.

FRONT END CODE

Below shows the program that the user is supposed to run. It contains all the validation for user inputs as well as calling all the other algorithms in the calculating emissions, scoring code and graphing code. I will show the implementation of this in a general order the program runs in.

First, the start method which is part of the Months_of_Data class is run. This shows a list of what my program essentially does.

```
310     # Start Method
311
312     def start(self,db):
313         self.__set_method_used()
314         self.__set_num_months()
315         self.__set_month_data(db)
316         self.__print_all()
317         self.__score_emissions(db)
318         self.__graph_emissions(db)
```

It starts with the user entering the method of calculation, the user wishes to use (fuel-based or distance-based method). This is implemented in an identical way to the way I showed in design:

```
225     def __set_method_used(self):
226
227         print('Possible Calculation Methods:')
228         print()
229         print('1. Fuel-based Method')
230         print('2. Distance-based Method')
231
232         valid = False # Validates user input for calculation method
233         while valid is False:
234             print()
235             try:
236                 calculation_method = int(input('Enter the calculation method (1 or 2) you wish to use for scoring: '))
237                 if calculation_method == 1 or calculation_method == 2:
238                     self.__calculation_method = calculation_method - 1
239                     valid = True
240                 else:
241                     self._method_exception()
242             except ValueError:
243                 self._method_exception()
```

Next, my program asks the user to enter the number of months to enter data for. This is identical to my pseudocode in design.

```
245     def __set_num_months(self):
246
247         print()
248         print('Please enter how many months you want to enter data for: ')
249         print('NB: The maximum no. of months is 6.')
250
251         max_months = 6
252         valid = False
253         while valid is False:
254             print()
255             try:
256                 num_months = int(input('No. of Months: '))
257                 if 1 < num_months <= max_months:
258                     self.__num_months = num_months
259                     valid = True
260                 else:
261                     self._num_months_exception()
262             except ValueError:
263                 self._num_months_exception()
```

```
265     def __set_month_data(self,db):
266
267         if self.__calculation_method == 0: # Creating fuel-based method objects
268             for i in range(self.__num_months):
269                 Month = Fuel_based()
270                 Month._set_all_month_data(db,i+1,self.__calculation_method)
271                 self.__list_of_emissions.append(Month._emissions_total)
272                 self.__objects.append(Month)
273         else:
274             for i in range(self.__num_months): # Creating distance-based method objects
275                 Month = Distance_based()
276                 Month._set_all_month_data(db,i+1,self.__calculation_method)
277                 self.__list_of_emissions.append(Month._emissions_total)
278                 self.__objects.append(Month)
```

This is done by dynamically creating an object of one of the subclasses Fuel_based or Distance_based. Each object represents a storage of 1 month of data. As shown, both the calculation method and the number of months are needed explaining why these were entered in first. Then, the user enters all the necessary data for that month and the emissions total from this entered data is added to a list of emissions for each month. This attribute exists to determine the data that needs to be called into the scoring algorithm and the graphing model. After all data has been entered, the object is added to a list of objects. This is done to make outputting data in the print_all method a lot easier.

In the set_all_month_data method, this is shown in the Fuel_based class and the Distance_based class respectively:

```
143     def _set_all_month_data(self,db,month,calculation_method):
144         print()
145         print('Month '+str(month))
146         print('-------')
147         self._set_distance()
148         self._set_vehicle_name(db)
149         self._set_refrigerant_name(db)
150         self.__set_quantity_refrigerant_leaked()
151         self._set_emissions_total(calculation_method,[db,self._vehicle_name,self._distance,self._refrigerant_name,self._quantity_refrigerant_leaked])
152
```

```
190    def _set_all_month_data(self,db,month,calculation_method):
191        print()
192        print('Month '+str(month))
193        print('-------')
194        self._set_distance()
195        self._set_vehicle_name(db)
196        self._set_refrigerant_name(db)
197        self.__set_mass_purchased()
198        self._set_emissions_total(calculation_method,[db,self._vehicle_name,self._distance,self._refrigerant_name,self._mass_purchased])
199
```

The pseudocode for all my validation is almost identical. I will split the code out into 3 sections: Validation for both calculation methods, validation for fuel-based method and validation for distance-based method.

Validation for both calculation methods

```
29     def _set_vehicle_name(self,db):
30
31         c = db.cursor()
32         print()
33         print('Available Vehicles for Scoring: ') # Presents all vehicles to the user
34         print()
35         vehicles_object = c.execute('SELECT vehicle_name FROM vehicles')
36         vehicles = c.fetchall()
37
38         for vehicle in vehicles:
39             vehicle_id_object = c.execute('SELECT vehicle_id FROM Vehicles WHERE vehicle_name=?',(vehicle[0],))
40             vehicle_id = str(c.fetchone()[0])
41             print(vehicle_id + '. ',vehicle[0])
42
43         max_id_object = c.execute('SELECT MAX(vehicle_id) FROM Vehicles')
44         max_id = c.fetchone()[0]
45
46         valid = False
47         while valid is False:
48             print()
49             try:
50                 vehicle_id = int(input('Enter the vehicle (1-'+str(max_id)+') used in Transportation: '))
51                 if 0 < vehicle_id <= max_id:
52                     vehicle_name_object = c.execute('SELECT vehicle_name FROM Vehicles WHERE vehicle_id=?',(vehicle_id,))
53                     vehicle_name = c.fetchone()[0]
54                     self._vehicle_name = vehicle_name
55                     valid = True
56                 else:
57                     self._vehicle_name_exception(max_id)  <-- Error message
58             except ValueError:
59                 self._vehicle_name_exception(max_id)
```

The max id is called in the error message because the program outputs the list of vehicles with the primary key of the vehicle table in the database (vehicle_id) being used as an index.

```
61    def _set_distance(self):
62
63        max_postcodes = 6
64        print()
65        print('The maximum number of postcodes that can be entered is '+ str(max_postcodes))
66        valid  = False # Validates number of postcodes passed in journey
67        while valid is False:
68            print()
69            try:
70                num_postcodes = int(input('How many junctions are there in the transportation journey? '))
71                if 1< num_postcodes <= max_postcodes:
72                    valid = True
73                else:
74                    self._num_postcodes_exception(max_postcodes)
75            except ValueError:
76                self._num_postcodes_exception(max_postcodes)
77
78        distance_successful = False # Calculates distance travelled
79        while distance_successful is False:
80
81            print()
82            print('Please enter the postcodes of these junctions below: ')
83            print('NB: Please ensure that the spaces in the address are in the right places.')
84            postcodes = []
85            for i in range(num_postcodes): # Validates each address entered in by calculating the distance between those 2 postcodes.
86                print()
87                address = input('Postcode ' + str(i+1) + ': ')
88                postcodes.append(address)
89            try:
90                self._distance = CalculatingEmissions.CalculateDistance(postcodes)
91                distance_successful = True
92            except:
93                self._calculate_distance_exception()  <-- Error Message
94
```

Validation for Distance-based method

```
173     # Setters
174     def __set_mass_purchased(self):
175         max_mass = 500
176         print('Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg')
177         valid = False
178         while valid is False:
179             print()
180             try:
181                 mass_purchased = float(input('Mass of Goods purchased: '))
182                 if 0 < mass_purchased <= 500:
183                     self._mass_purchased = mass_purchased
184                     valid = True
185                 else:
186                     self._mass_exception()
187             except ValueError:
188                 self._mass_exception()  <-- Error Message
```

Validation for Fuel-based method

```
125     def __set_quantity_refrigerant_leaked(self):
126
127         print()
128         print('Please enter the quantity of refrigerant leaked by your vehicle.')
129         print('The refrigerant used by your vehicle is ',self._refrigerant_name)
130         valid = False
131         while valid is False:
132             print()
133             try:
134                 quantity_refrigerant_leaked = float(input('Quantity of '+str(self._refrigerant_name)+' leaked: '))
135                 if quantity_refrigerant_leaked >= 0:
136                     self._quantity_refrigerant_leaked = quantity_refrigerant_leaked
137                     valid = True
138                 else:
139                     self.__quantity_refrigerant_exception()
140             except ValueError:
141                 self.__quantity_refrigerant_exception()
```

Error Message

As well as this validation, the program also has a method that determines the refrigerant name for the vehicle. This is done using an INNER JOIN as shown below:

```
95      def _set_refrigerant_name(self,db):
96          c = db.cursor()
97          sql = '''SELECT Refrigerant.refrigerant_name FROM Vehicles
98                   INNER JOIN Refrigerant ON Vehicles.refrigerant_id=Refrigerant.refrigerant_id
99                   WHERE Vehicles.vehicle_name=?'''
100         refrigerant_name_object = c.execute(sql,(self._vehicle_name,))
101         self._refrigerant_name = c.fetchone()[0]
```

Using all this data, the function to calculate the total emissions is called:

```
103     def _set_emissions_total(self,calculation_method,attributes):
104
105         self._emissions_total = CalculatingEmissions.CalculateEmissions(calculation_method,attributes)
```

After this is done, all the data is printed out as a summary. This is what is called in the print_all method in Months_of_Data class:

```
296     # Output Method
297
298     def __print_all(self):
299
300         print()
301         print('Summary of Data for each Month')
302         print('------------------------------')
303         for i in range(len(self.__objects)): # Iterates through each month, printing all data
304             print()
305             print('Month ' + str(i+1)+' data')
306             print('------------')
307             print()
308             self.__objects[i]._print_all_month_data()
309
```

This iterates through each object created, printing out all the data by calling the print_all_month_data method in either the Fuel_based or Distance_based class. The code for print_all_month_data method for the Fuel_based and Distance_based classes respectively is shown below:

```
153     # Output Method
154
155     def _print_all_month_data(self): # Overrides the method print_all_month_data in parent class
156
157         print('Vehicle used: ',self._vehicle_name)
158         print('Distance travelled in 1 journey: ',self._distance)
159         print('Quantity Refrigerant Leaked in 1 journey: ',self._quantity_refrigerant_leaked)
160         print('Total emissions emitted in 1 journey: ',str(round(self._emissions_total,2)),'CO2e')
```

```
200     # Output method
201     def _print_all_month_data(self): # Overrides print_all_month_data in parent class
202         print('Vehicle used: '+self._vehicle_name)
203         print('Distance travelled across 1 journey:'+str(self._distance))
204         print('Mass of goods purchased:', str(self._mass_purchased))
205         print('Total Emissions emitted:',str(round(self._emissions_total,2)),' CO2e')
```

After printing all the data out as a summary, the program then goes onto scoring each month and applying the graphing model on the data. The code calling the scoring program isn't much and is shown below:

```
281
282     def __score_emissions(self,db): # Calls the scoring program to score all the emission totals
283         ScoringEmissions.DetermineEmissionsScore(db,self.__calculation_method,self.__list_of_emissions)
```

The Graphing method requires some more work as the most recent month of data is needed as a parameter of the Graph_and_Improve function. The code calling the Graph_and_Improve function is shown below:

```
285     def __graph_emissions(self,db): # Calls the graphing model
286         x_values = np.array([i for i in range(1,len(self.__list_of_emissions)+1)])
287         recent_month = self.__objects[len(self.__objects)-1] # Obtains the data from the most recent month.
288         if self.__calculation_method == 0:
289             attributes = [recent_month._distance,recent_month._refrigerant_name,recent_month._quantity_refrigerant_leaked]
290             #Takes all needed fuel-based attributes to calculate total emissions in determining more efficient vehicles.
291         else:
292             attributes = [recent_month._distance,recent_month._refrigerant_name,recent_month._mass_purchased]
293             # Takes all the needed distance-based attributes to calculate total emissions in determinin more efficient vehicles.
294         GraphEmissions.Graph_and_Improve(db,x_values,np.array(self.__list_of_emissions),self.__calculation_method,attributes)
```

The x values for my graphing model must be the month number and hence, must be the numbers from 1 to the number of months entered by the user. Hence, to make this as efficient as possible, I have used list comprehension. Then, I have identified the most recent month as the last object in the objects list. Then, I determined which calculation method the user is using and then the attributes of that month are stored in an array. Then, the Graph_and_Improve function is called.

TECHNIQUES USED IN IMPLEMENTATION

Below is a table of all the algorithms and models I have created to form my technical solution. The page numbers indicate where it is referenced in the document.

Number	Algorithms	Objective	Page no.s in document
1	User/Case DDL Scripting	All Objectives	Page 135-138
2	Aggregate SQL Queries (AVG and MAX)	2.3,3.3,4.2,4.4	Page 55,152 Page 154
3	INNER JOIN Queries	2.5,3.5,4.1	Page 55,152 Page 144
4	Recursive Algorithms (Stein's Algorithm)	4.1	Pages 15-16,38, 150
5	Sorting Algorithm (Merge Sort)	5.4	Pages 49-50,158-159
6	Optimisation Algorithms (Gradient Descent, Least Squares Algorithm)	5.2	Pages 47, 157-158
7	List Operations and numpy arrays operations (. count, append, list comprehension, np.prod, .fromkeys, np.arange, np.sum, np.mean)	2.1,2.6,3.1,3.6,	List Operations: Page 148 Page 151 Numpy array operations: Page 151 Page 158
8	OOP Model – Inheritance, Polymorphism, Composition used	1.1,1.2,2.1,2.2,2.3, 2.4, 3.1,3.2,3.3,3.4	Pages 41-43
9	Dynamic Generation of Objects	1.2	Page 148

Testing

This section shows how I have implemented all my tests given in design. I will be testing many of these from the code in which I created the models, not necessarily from the front end. This ensures the robustness of my solution.

TEST 1

1	Testing the presence of Vehicles, Fuel and Refrigerant data	• The test tables should show the correct records and field names as defined in Stage 1. • DB Browser is the software that will be used to visually see the databases.

The 3 test tables that have been produced are shown below.

fuel_id	fuel_name	emission_factor
Filter	Filter	Filter
1	Biodiesel	0.1658
2	Mineral Diesel	2.68787
3	Natural Gas	2.02266

refrigerant_id	refrigerant_name	gwp
Filter	Filter	Filter
1	R134a	1.41

vehicle_id	vehicle_name	fuel_efficiency	fuel_id	refrigerant_id
Filter	Filter	Filter	Filter	Filter
1	Scania L Series	0.415	1	1
2	Scania P Series	0.237	1	1
3	Western Star 4700 Series	0.394	2	1
4	Western Star 5700 XE	0.338	3	1
5	Iveco S-way	0.211	2	1
6	Iveco Stralis	0.208	3	1
7	MAN TGA	0.0893	2	1
8	MAN TGM	0.181	2	1

As shown, the data dictionaries from Stage 1 are identical to these tables with the same field names and same number of records. Hence, these tables in my database have been correctly created.

TEST 2

2	Testing the Validation of the Calculation Method in Objective 1.1	• The appropriate error message is given when the user input is invalid. (Testing 1.1a) • If the input is valid, then the program should go on to objective 1.2

The table below tests all the possible situations that could happen when entering the calculation method. A Valid Result is when the program moves onto asking the user to enter the number of months to enter data from when given a valid entry. An invalid result should output 'ERROR: Please enter 1 or 2.' And then ask the user to enter the calculation method used again.

Test No.	Description	Type of Data	Expected Result	Program Result	Outcome
2.1	User enters 1	Boundary	Valid	Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: 1 Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months:	Pass
2.2	User enters 2	Boundary	Valid	Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: 2 Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months:	Pass
2.3	User enters 3	Erroneous	Invalid	Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: 3 ERROR: Please enter 1 or 2. Enter the calculation method (1 or 2) you wish to use for scoring:	Pass

2.4	User enters 0	Erroneous	Invalid	Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: 0 ERROR: Please enter 1 or 2. Enter the calculation method (1 or 2) you wish to use for scoring:	Pass
2.5	User enters -2	Erroneous	Invalid	-- Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: -2 ERROR: Please enter 1 or 2. Enter the calculation method (1 or 2) you wish to use for scoring:	Pass
2.6	User enters 1.35	Erroneous	Invalid	Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: 1.35 ERROR: Please enter 1 or 2. Enter the calculation method (1 or 2) you wish to use for scoring:	Pass
2.7	User enters -8.34	Erroneous	Invalid	Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: -8.34 ERROR: Please enter 1 or 2. Enter the calculation method (1 or 2) you wish to use for scoring:	Pass
2.8	User enters ‘foo’	Erroneous	Invalid	Possible Calculation Methods: 1. Fuel-based Method 2. Distance-based Method Enter the calculation method (1 or 2) you wish to use for scoring: foo ERROR: Please enter 1 or 2. Enter the calculation method (1 or 2) you wish to use for scoring:	Pass

The screenshots of the outputs of each test are below.

As shown by these 8 tests, all the possible inputs have achieved the expected result and hence, the validation for entering the calculation method is working correctly.

TEST 3

3	Testing the validation for how many months of data are needed in Objective 1.2	• The appropriate error message is given when the user input is invalid. (Testing 1.2a,1.2b,1.2c) • If the user input is valid, the program should go on to objective 2.1/3.1

The table below shows the results of every possible type of entry by the user when entering the number of months of data to enter. A valid result is when the program, when given a valid entry moves onto asking the user to enter the postcodes involved in the transport journey. An invalid result is when the program, when given an invalid entry leads the program to output "ERROR: Please enter an integer between 2 and 6 inclusive."

Test No.	Description	Type of Data	Expected Result	Program Result	Outcome
3.1	User enters 2	Boundary	Valid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 2 Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey?	Pass
3.2	User enters 3	Typical	Valid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 3 Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey?	Pass
3.3	User enters 4	Typical	Valid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 4 Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey?	Pass

3.4	User enters 5	Typical	Valid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 5 Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey?	Pass
3.5	User enters 6	Boundary	Valid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 6 Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey?	Pass
3.6	User enters 7	Erroneous	Invalid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 7 ERROR: Please enter an integer between 2 and 6 inclusive. No. of Months:	Pass
3.7	User enters 1	Erroneous	Invalid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 1 ERROR: Please enter an integer between 2 and 6 inclusive. No. of Months:	Pass
3.8	User enters -2	Erroneous	Invalid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: -2 ERROR: Please enter an integer between 2 and 6 inclusive. No. of Months:	Pass

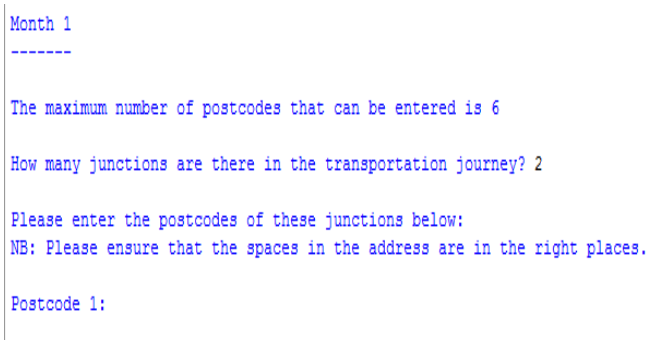
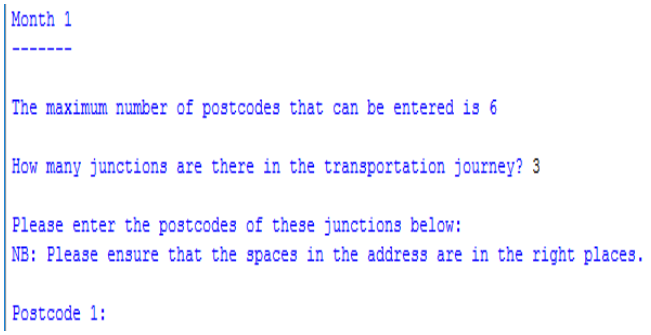
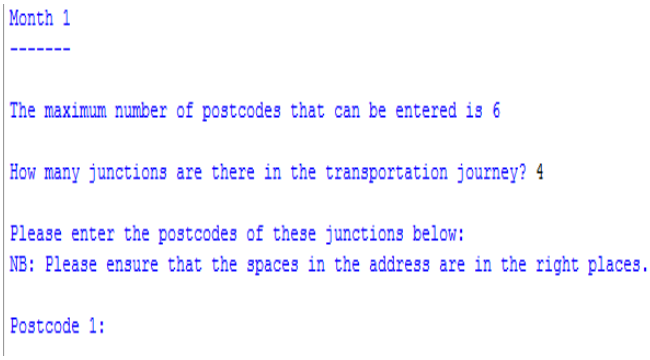
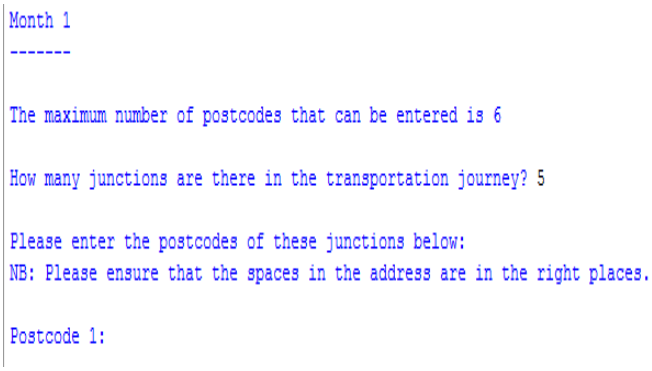
3.9	User enters 1.35	Erroneous	Invalid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: 1.35 ERROR: Please enter an integer between 2 and 6 inclusive. No. of Months:	Pass
3.10	User enters - 8.34	Erroneous	Invalid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: -8.34 ERROR: Please enter an integer between 2 and 6 inclusive. No. of Months:	Pass
3.11	User enters 'foo'	Erroneous	Invalid	Please enter how many months you want to enter data for: NB: The maximum no. of months is 6. No. of Months: foo ERROR: Please enter an integer between 2 and 6 inclusive. No. of Months:	Pass

As shown, all the tests give the correct outcome showing that the validation is fully working for this section.

TEST 4

4	Testing the validation for how many postcodes will be entered by the user in objectives 2.1 and 3.1	• The appropriate error message is given when the user input is invalid. (Testing 2.1a,2.1b,3.1a,3.1b) • If the user input is valid, the program should continue with objective 2.1/3.1

The table below shows all the possible types of inputs by the user. A Valid Result is one that when given a valid result, moves onto asking the user to enter the postcodes in the transport journey. An invalid result is one that when given an invalid result, it outputs 'ERROR: Please enter an integer from 2 to 6 to give the no. of junctions in the transportation journey.' and makes the user enter it again.

Test No.	Description	Type of Data	Expected Result	Program Result	Outcome
4.1	User enters 2	Boundary	Valid		Pass
4.2	User enters 3	Typical	Valid		Pass
4.3	User enters 4	Typical	Valid		Pass
4.4	User enters 5	Typical	Valid		Pass

4.5	User enters 6	Boundary	Valid	Month 1 ------- The maximum number of postcodes that can be entered is 6 How many junctions are there in the transportation journey? 6 Please enter the postcodes of these junctions below: NB: Please ensure that the spaces in the address are in the right places. Postcode 1:	Pass
4.6	User enters 7	Erroneous	Invalid	Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey? 7 ERROR: Please enter an integer from 2 to 6 to give the no. of junctions in the transportation journey. How many junctions are there in the transportation journey?	Pass
4.7	User enters 1	Erroneous	Invalid	Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey? 1 ERROR: Please enter an integer from 2 to 6 to give the no. of junctions in the transportation journey. How many junctions are there in the transportation journey?	Pass
4.8	User enters -2	Erroneous	Invalid	Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey? -2 ERROR: Please enter an integer from 2 to 6 to give the no. of junctions in the transportation journey. How many junctions are there in the transportation journey? Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey? 1.35 ERROR: Please enter an integer from 2 to 6 to give the no. of junctions in the transportation journey. How many junctions are there in the transportation journey?	Pass

4.9	User enters 1.35	Erroneous	Invalid		Pass
4.10	User enters -8.34	Erroneous	Invalid	Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey? -8.34 ERROR: Please enter an integer from 2 to 6 to give the no. of junctions in the transportation journey. How many junctions are there in the transportation journey?	Pass
4.11	User enters 'foo'	Erroneous	Invalid	Month 1 ------- The maximum number of addresses that can be entered is 6 How many junctions are there in the transportation journey? foo ERROR: Please enter an integer from 2 to 6 to give the no. of junctions in the transportation journey. How many junctions are there in the transportation journey?	Pass

As shown, each test value gives the expected outcome. Hence, the validation here works.

TEST 5

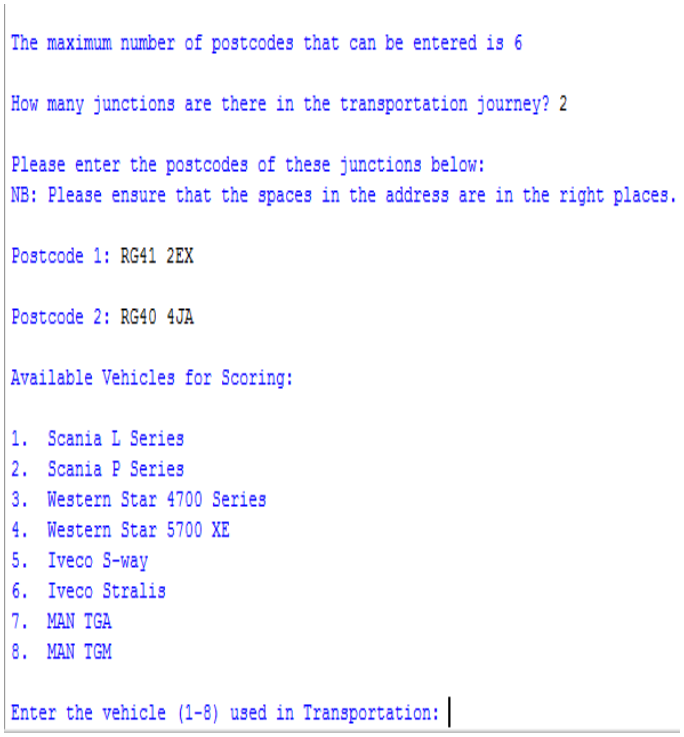
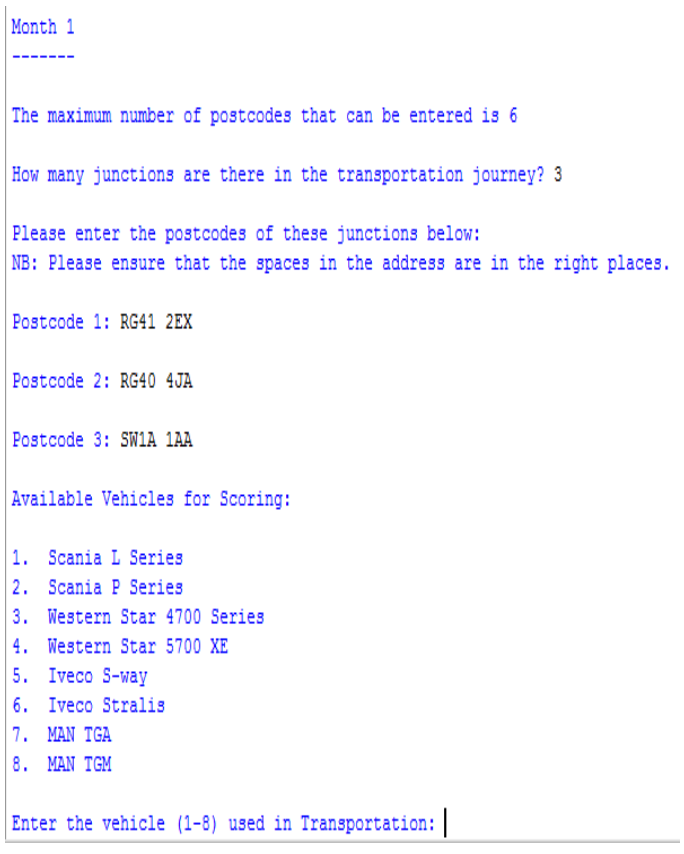
5	Testing the validation for the entry of postcodes in objectives 2.1 and 3.1	• That the appropriate error message is displayed when an invalid address is entered. (Testing 2.1c,3.1c) • If the postcodes are valid, the program should move onto objective 2.2/3.2

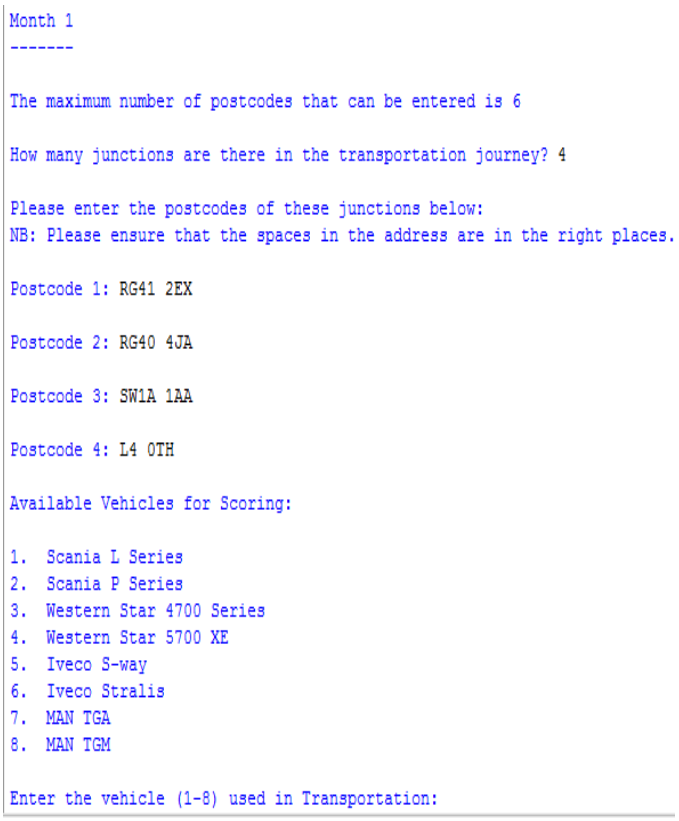
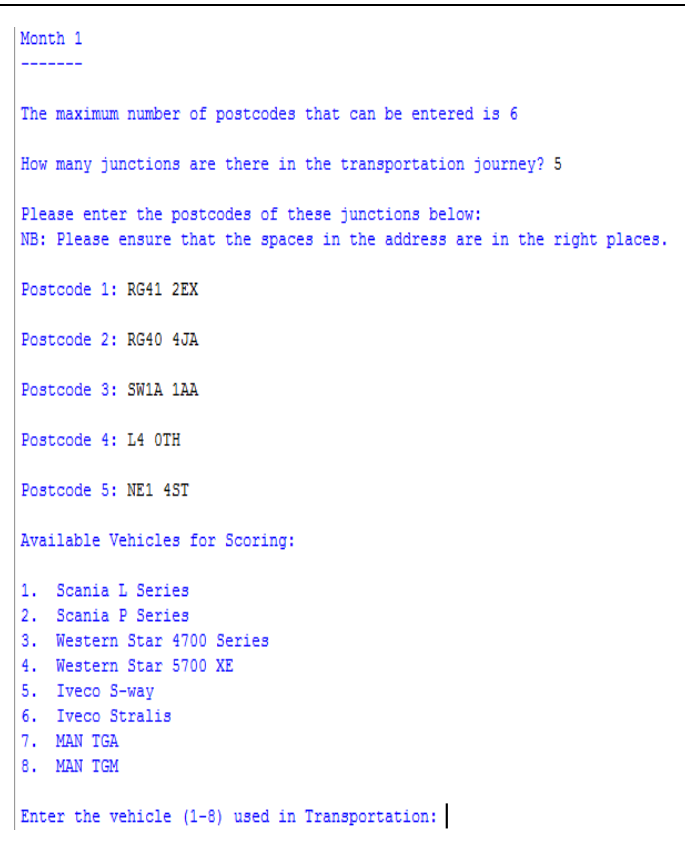
Below is a list of valid postcodes that will be used to test this:

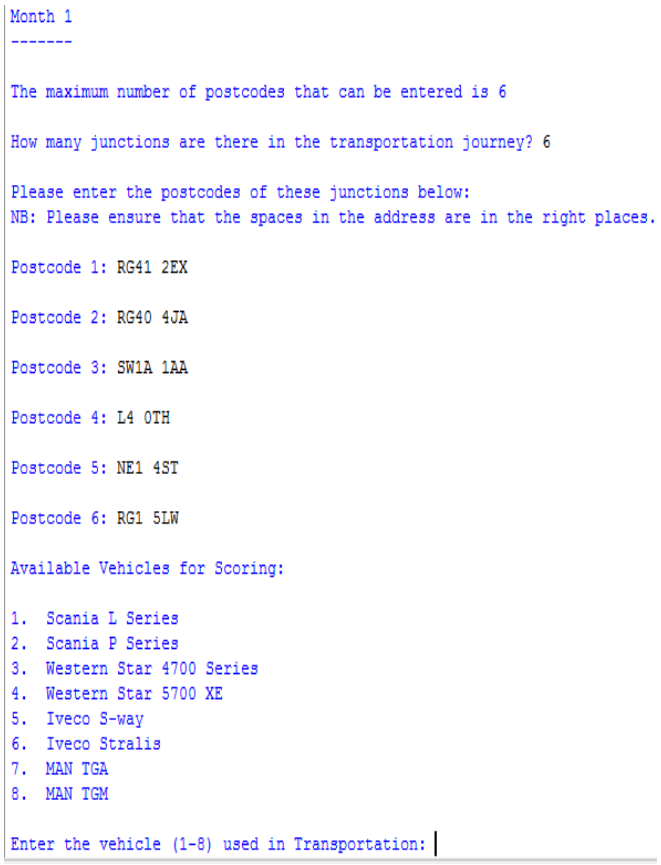
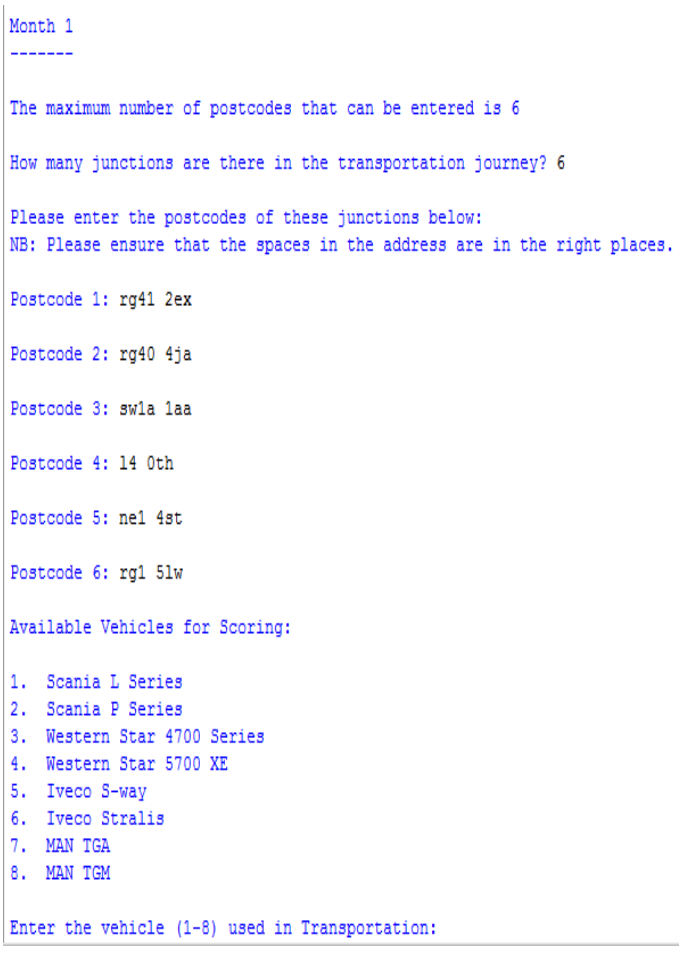
RG41 2EX, RG40 4JA, SW1A 1AA, L4 0TH, NE1 4ST, RG1 5LW

A valid result is when the program moves on to calculate the distance travelled through the postcodes entered if valid postcodes have been entered in by the user. This will be tested by typing a print statement saying 'Distance calculated' in the CalculateDistance algorithm from Stage 2 of design. An invalid result is when the program outputs "ERROR: One of these postcodes does not exist. Please enter existent postcodes." and then asks the user to enter all the postcodes again.

The table below shows all the tests:

Test No.	Description	Type of Data	Expected Result	Program Result	Outcome
5.1	User enters RG41 2EX, RG40 4JA with num_postcodes = 2	Typical	Valid		Pass
5.2	User enters RG41 2EX, RG40 4JA, SW1A 1AA with num_postcodes = 3	Typical	Valid		Pass

5.3	User enters RG41 2EX, RG40 4JA, SW1A 1AA, L4 0TH with num_postcodes = 4	Typical	Valid		Pass
5.4	User enters RG41 2EX, RG40 4JA, SW1A 1AA, L4 0TH,NE1 4ST with num_postcodes = 5	Typical	Valid		Pass

5.5	User enters RG41 2EX, RG40 4JA, SW1A 1AA, L4 0TH,NE1 4ST,RG1 5LW with num_postcodes = 6	Typical	Valid		Pass
5.6	User does same test in 5.5 but the postcodes are entered in lower case	Typical	Valid		Pass

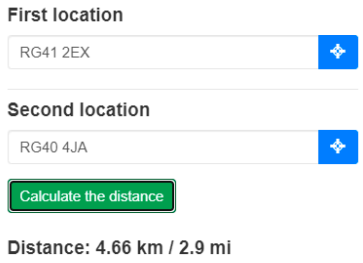
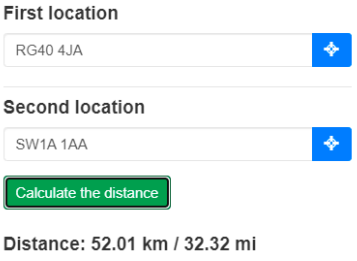
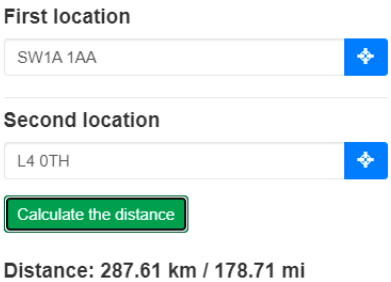
5.7	User enters RG412EX, RG40 4JA with num_postcodes = 2	Erroneous	Invalid	Month 1 ------- The maximum number of postcodes that can be entered is 6 How many junctions are there in the transportation journey? 2 Please enter the postcodes of these junctions below: NB: Please ensure that the spaces in the address are in the right places. Postcode 1: RG412EX Postcode 2: RG40 4JA ERROR: One of these postcodes does not exist. Please enter existent postcodes. Please enter the postcodes of these junctions below: NB: Please ensure that the spaces in the address are in the right places. Postcode 1:	Pass
5.8	User enters jgbjk, RG41 2EX, RG40 4JA with num_postcodes = 3	Erroneous	Invalid	Month 1 ------- The maximum number of postcodes that can be entered is 6 How many junctions are there in the transportation journey? 3 Please enter the postcodes of these junctions below: NB: Please ensure that the spaces in the address are in the right places. Postcode 1: jgbjk Postcode 2: RG41 2EX Postcode 3: RG40 4JA ERROR: One of these postcodes does not exist. Please enter existent postcodes. Please enter the postcodes of these junctions below: NB: Please ensure that the spaces in the address are in the right places. Postcode 1:	Pass

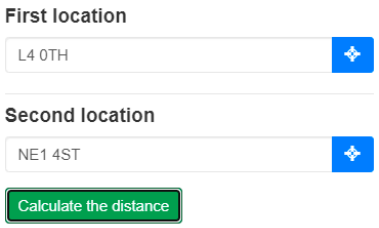
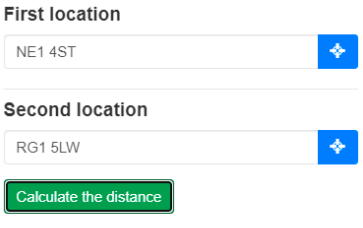
As shown, all tests show the expected results which makes the validation of postcodes correct.

TEST 6

6	Testing whether the distance between 2 locations in objectives 2.2 and 3.2 is correct	• That the correct geodesic distance is calculated between 2 postcodes entered by the user. • Also, after it does this, it should move onto objective 2.3/3.3

To verify that the geodesic distance calculated by my algorithm is correct, I will be putting tests 5.1 to 5.6 into my code again and then one after the other into How far is it? - Straight Line Distance Calculator (gps-coordinates.net) which also calculates the geodesic distance. If they are the approximately the same distances, then we know that my algorithm to calculate the distance works. I have placed a print statement in the set_distance method in the front end to give the distance travelled according to the program.

Test No.	Description	Distance calculated by my program	Distance calculated using above link	Outcome of test
6.1	User enters RG41 2EX, RG40 4JA	`Distance Travelled: 4.6579`		Pass
6.2	User enters RG41 2EX, RG40 4JA, SW1A 1AA	`Distance Travelled: 56.7686`	 Using Test 6.1, we know that the total distance is 4.66 + 52.01 = 56.67km	Pass
6.3	User enters RG41 2EX, RG40 4JA, SW1A 1AA, L4 0TH	`Distance Travelled: 344.5274`	 Using test 6.2, we know that the total distance is 56.67+287.61 = 344.28km	Pass

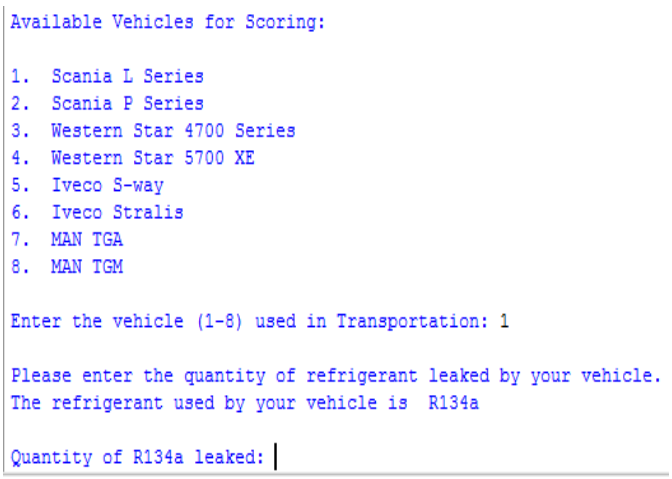
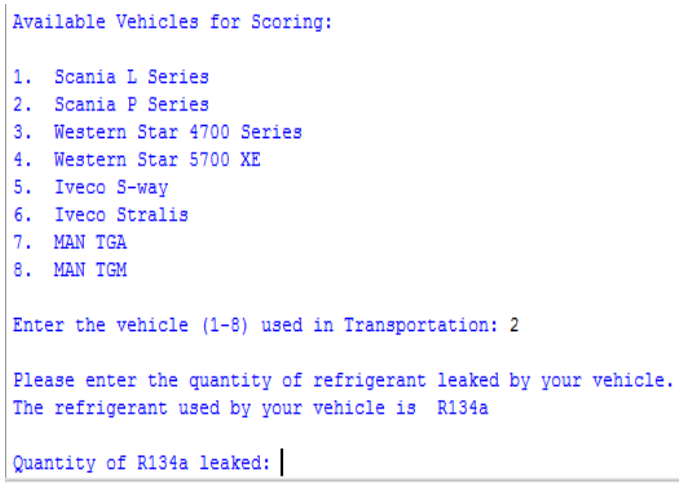
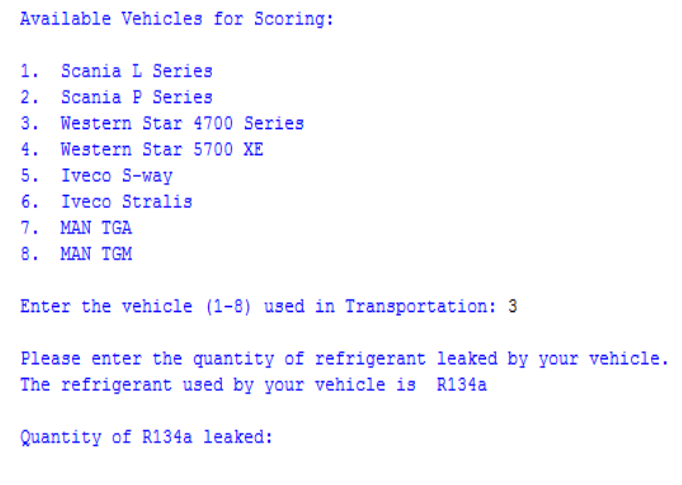
6.4	User enters RG41 2EX, RG40 4JA, SW1A 1AA, L4 0TH,NE1 4ST	Distance Travelled: 537.3642	 Using test 6.3 we know that the total distance is 192.72 + 344.28 = 537km	Pass
6.5	User enters RG41 2EX, RG40 4JA, SW1A 1AA, L4 0TH,NE1 4ST,RG1 5LW	Distance Travelled: 932.4164	 Using test 6.4 we know that the total distance is 537 + 395.11 = 932.11km	Pass

As shown, the geodesic distance calculated by my program is roughly equal to the geodesic distance calculated by the website which means that my program calculates the total distance travelled correctly.

TEST 7

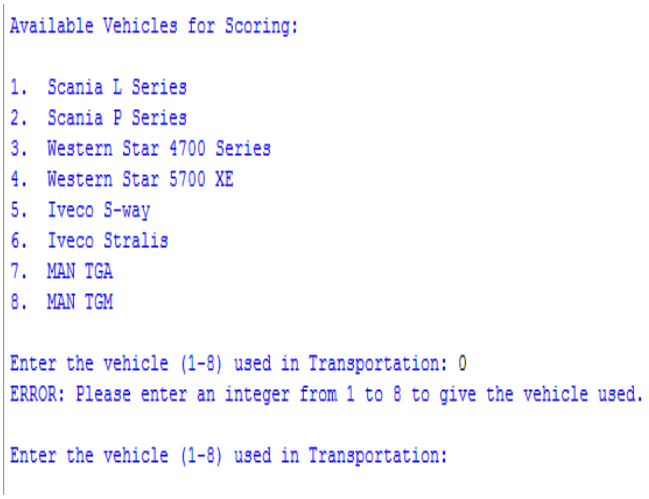
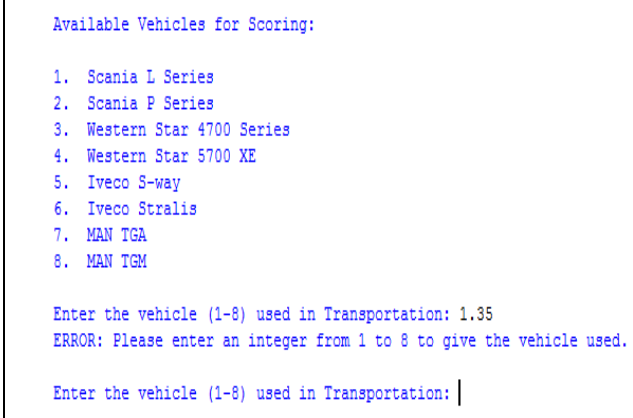
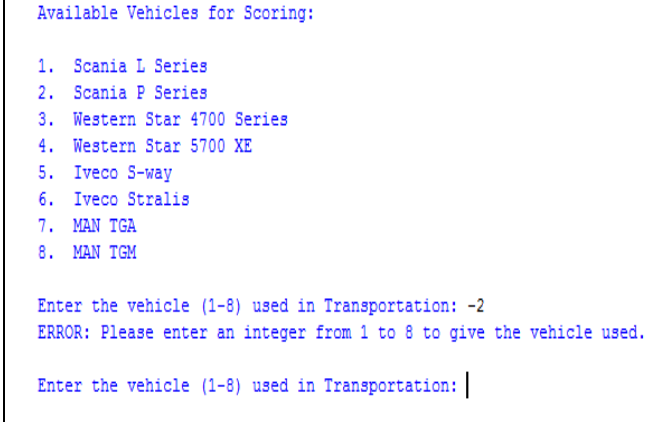
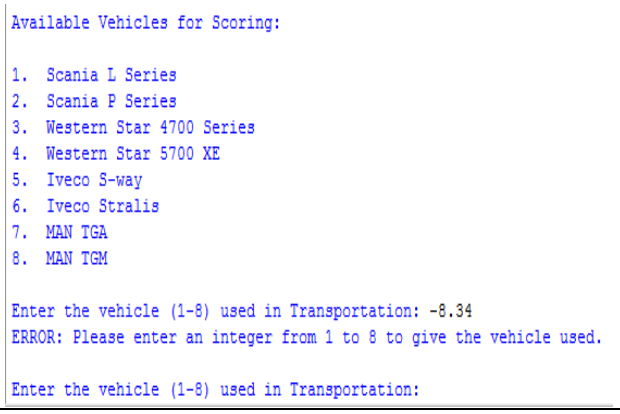
7	Testing the validation of entry of the name of the vehicle in objectives 2.3 and 3.3	• That the appropriate error message is displayed when an invalid address is entered. (Testing 2.3a) • If the entry is valid, the program should move onto objective 2.4/3.4

A valid result is one that when given a valid input, moves onto the either asking the user to enter the mass of good bought (distance-based method) or asking the user to enter the quantity of refrigerant leaked (fuel-based method). An invalid result is one that when given an invalid input, the user is told “ERROR: Please enter an integer from 1 to 8 to give the vehicle used.” And is asked to enter the vehicle again. Below shows the possible types of inputs the user could enter in and how the program responds for the fuel-based method:

Test No.	Description	Type of Data	Expected Result	Program Result	Output
7.1	User enters 1	Boundary	Valid		Pass
7.2	User enters 2	Typical	Valid		Pass
7.3	User enters 3	Typical	Valid		Pass

7.4	User enters 4	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 4 Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked:	Pass
7.5	User enters 5	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 5 Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked:	Pass
7.6	User enters 6	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 6 Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked:	Pass

7.7	User enters 7	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 7 Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked:	Pass
7.8	User enters 8	Boundary	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 8 Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked:	Pass
7.9	User enters 9	Erroneous	Invalid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 9 ERROR: Please enter an integer from 1 to 8 to give the vehicle used. Enter the vehicle (1-8) used in Transportation:	Pass

7.10	User enters 0	Erroneous	Invalid		Pass
7.11	User enters 1.35	Erroneous	Invalid		Pass
7.12	User enters -2	Erroneous	Invalid		Pass
7.13	User enters -8.34	Erroneous	Invalid		Pass

7.14	User enters foo	Erroneous	Invalid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: foo ERROR: Please enter an integer from 1 to 8 to give the vehicle used. Enter the vehicle (1-8) used in Transportation:	Pass

As shown it works for the fuel-based method giving the expected result in all cases. For the distance-based methods, I only need to do the valid results as I have tested that the validation works. I just need to test that it moves onto getting the user to enter the mass of goods purchased.

Test No.	Description	Type of Data	Expected Result	Program Result	Output
7.15	User enters 1	Boundary	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 1 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass
7.16	User enters 2	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 2 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass

7.17	User enters 3	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 3 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass
7.18	User enters 4	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 4 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass
7.19	User enters 5	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 5 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass

7.20	User enters 6	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 6 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass
7.21	User enters 7	Typical	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 7 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass
7.22	User enters 8	Boundary	Valid	Available Vehicles for Scoring: 1. Scania L Series 2. Scania P Series 3. Western Star 4700 Series 4. Western Star 5700 XE 5. Iveco S-way 6. Iveco Stralis 7. MAN TGA 8. MAN TGM Enter the vehicle (1-8) used in Transportation: 8 Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased:	Pass

TEST 8

8	Testing validation of entering mass of goods purchased in objective 2.4	• That the appropriate error message is displayed when an invalid address is entered. (Testing 2.4a) • If the entry is valid, the program should move onto objective 2.5

A valid result is one that when given valid data, moves onto entering the number of times this journey was made over 1 month. An invalid result is one that when given invalid data, tells the user "ERROR: Please enter a mass between 0 and 500kg." and asks the user to enter the data again. The table below shows all possible types of inputs by the user and the program result.

Test No.	Description	Type of Data	Expected Result	Program Result	Outcome
8.1	User enters 0.2	Typical	Valid	Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased: 0.2 Please enter how many times this journey was made: Number of journeys:	Pass
8.2	User enters 150.67	Typical	Valid	Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased: 150.67 Please enter how many times this journey was made: Number of journeys:	Pass
8.3	User enters 500	Boundary	Valid	Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased: 500 Please enter how many times this journey was made: Number of journeys:	Pass
8.4	User enters 650	Erroneous	Invalid	Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased: 650 ERROR: Please enter a mass between 0 and 500kg. Mass of Goods purchased:	Pass
8.5	User enters 0	Erroneous	Invalid	Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased: 0 ERROR: Please enter a mass between 0 and 500kg. Mass of Goods purchased:	Pass

8.6	User enters -100	Erroneous	Invalid	Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased: -100 ERROR: Please enter a mass between 0 and 500kg. Mass of Goods purchased:	Pass
8.7	User enters foo	Erroneous	Invalid	Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg Mass of Goods purchased: foo ERROR: Please enter a mass between 0 and 500kg. Mass of Goods purchased:	Pass

As shown, the tests all give the expected result meaning that the validation works correctly.

TEST 9

9	Testing validation of quantity refrigerant leaked in objective 3.4	• That the appropriate error message is displayed when an invalid address is entered. (Testing 3.4a) • If the entry is valid, the program should move onto objective 3.5

A Valid result is one that when given a valid input, the program moves onto asking the user to enter the data from the second month, first asking how many junctions there are in the transport journey. An invalid result is one that when given an invalid input, the program outputs "ERROR: Please enter a positive number." and asks the user to re-enter the quantity of refrigerant leaked. The table below shows all possible types of data the user could input.

Test No.	Description	Type of Data	Expected Result	Program Result	Outcome
9.1	User enters 0	Boundary	Valid	Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked: 0 Month 2 ------- The maximum number of postcodes that can be entered is 6 How many junctions are there in the transportation journey?	Pass
9.2	User enters 0.1234	Typical	Valid	Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked: 0.1234 Month 2 ------- The maximum number of postcodes that can be entered is 6 How many junctions are there in the transportation journey?	Pass

9.3	User enters 450.1	Typical	Valid	Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked: 450.1 Month 2 ------- The maximum number of postcodes that can be entered is 6 How many junctions are there in the transportation journey?	Pass
9.4	User enters -8	Erroneous	Invalid	Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked: -8 ERROR: Please enter a positive number. Quantity of R134a leaked:	Pass
9.5	User enters foo	Erroneous	Invalid	Please enter the quantity of refrigerant leaked by your vehicle. The refrigerant used by your vehicle is R134a Quantity of R134a leaked: foo ERROR: Please enter a positive number. Quantity of R134a leaked:	Pass

As shown, all the tests give the expected result so hence the validation for this section is working correctly

TEST 10

10	Verifying that the emissions calculation for the distance-based method in objective 2.5 is correct	• That the total emissions calculated by the program is the same as the total emissions calculated by hand. • After calculating the emissions, it moves on to the user entering data for the next month, fulfilling objective 2.6

To test the distance-based method calculation, I will check the calculation using my own inputs by hand and compare the result with the program. Recall the distance-based formula is:

CO_2e emissions from transportation =

sum across transport modes and/or vehicle types:

= Σ (mass of goods purchased (tonnes or volume) × distance travelled in transport leg (km) × emission factor of transport mode or vehicle type (kg CO_2e/tonne or volume/km))

I will be using 2 different examples to check that my program works correctly.

Test No.	Inputs	Total Emissions calculated by hand	Total emissions calculated by program	Outcome

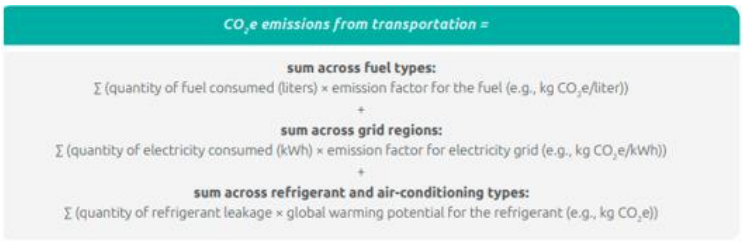
10.1	Vehicle_name = 'Iveco Stralis' Distance = 40km Refrigerant_name = R134a Mass_goods_purchased = 230kg Num_journeys = 18	Total Emission Factor = 2.02266 + 1.41 = 3.43266 So Total Emissions is 3.43266 * 40 * 230 * 18 = 568448.50 (2dp)	568448.5	Pass
10.2	Vehicle_name = 'Scania L Series' Distance = 35 km Refrigerant_name = R134a Mass_goods_purchased = 18 kg Num_journeys = 45	Total Emission Factor = 0.1658 + 1.41 = 1.5758 So Total Emissions is 1.5758 * 35 *18* 45 =44673.93	44673.93	Pass

As shown in both examples, the calculation of the total emissions is the same when I do it by hand and when I use the program. Hence, the program to calculate the total emissions is fully working.

TEST 11

11	Verifying that the emissions calculation for the fuel-based method in objective 3.5 is correct	• That the total emissions calculated by the program is the same as the total emissions calculated by hand. • After calculating the emissions, it moves on to the user entering data for the next month, fulfilling objective 3.6

This will be done in a similar way to Test 11. Recall the Fuel-based Formula.



I will use 2 examples to ensure that the program works correctly.

Test No.	Inputs	Total Emissions calculated by hand	Total Emissions calculated by program	Outcome
11.1	Vehicle_name = Scania P Series Distance = 65 km Refrigerant_name = R134a Quantity_refrigerant_leaked = 0.1234 Num_journeys = 49	Quantity of Fuel Consumed = 65*0.237 = 15.405 So Total Emissions = ((15.405*0.1658)+(0.1234*1.41)) *49 = 133.68 (2dp)	133.68	Pass
11.2	Vehicle_name = Western Star 4700 Series Distance = 9 km Refrigerant_name = R134a Quantity_refrigerant_leaked = 0.11 Num_journeys = 88	Quantity of Fuel Consumed = 9*0.394 = 3.546 So Total Emissions = ((3.546*2.68787)+(0.11*1.41))*88 = 852.39 (2dp)	852.39	Pass

On both occasions, the program gives the same result as when I do it by hand which means that the program works out the total emissions for the fuel-based method correctly.

TEST 12

12	Verifying that data is entered for the number of months entered in by the user in objective 1.2. This tests objectives 2.6 and 3.6	• That the number of months of data entered by the user matches the value entered by the user in objective 1.2. • After this, the program should start to fulfil objectives 2.7/3.7

To test this, I will not run any of the user inputs, but I will ensure that only the print statements in the set_all_month_data methods print out all the months. This means that it is being run the correct number of times. I will do this for both methods.

Fuel-Based Method

Num_months = 2:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 1

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 2

Month 1
-------

Month 2
-------
```

Num_months = 3:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 1

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 3

Month 1
-------

Month 2
-------

Month 3
-------
```

Num_months = 4:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 1

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 4

Month 1
-------

Month 2
-------

Month 3
-------

Month 4
-------
```

Num_months = 5:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 1

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 5

Month 1
-------

Month 2
-------

Month 3
-------

Month 4
-------

Month 5
-------
```

Num_months = 6:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 1

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 6

Month 1
-------

Month 2
-------

Month 3
-------

Month 4
-------

Month 5
-------

Month 6
-------
```

Distance-based Method

Num_months = 1:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 2

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 2

Month 1
-------

Month 2
-------
```

Num_months = 2:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 2

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 3

Month 1
-------

Month 2
-------

Month 3
-------
```

Num_months = 4:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 2

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 4

Month 1
-------

Month 2
-------

Month 3
-------

Month 4
-------
```

Num_months = 5:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 2

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 5

Month 1
-------

Month 2
-------

Month 3
-------

Month 4
-------

Month 5
-------
```

Num_months = 6:

```
Possible Calculation Methods:

1. Fuel-based Method
2. Distance-based Method

Enter the calculation method (1 or 2) you wish to use for scoring: 2

Please enter how many months you want to enter data for:
NB: The maximum no. of months is 6.

No. of Months: 6

Month 1
-------

Month 2
-------

Month 3
-------

Month 4
-------

Month 5
-------

Month 6
-------
```

TEST 13

13	Verifying that all Month data is outputted to the user in objectives 2.7/3.7	• That the month, the data entered by the user and the total emission for that month is outputted to the user as a summary after user input. • The program should then move on to fulfilling objective 4.3

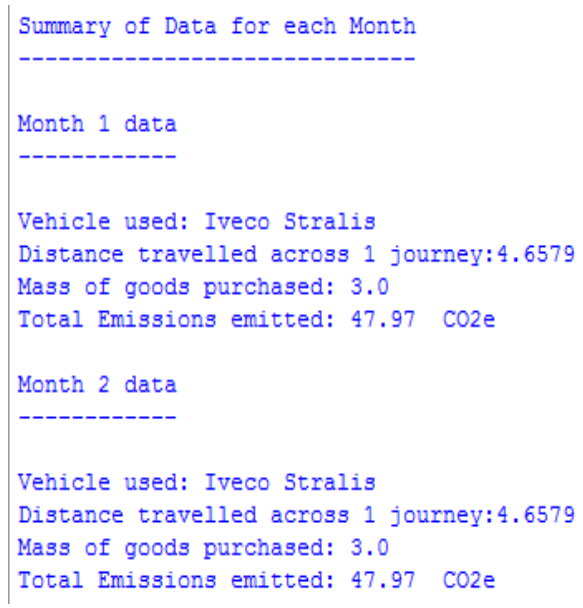
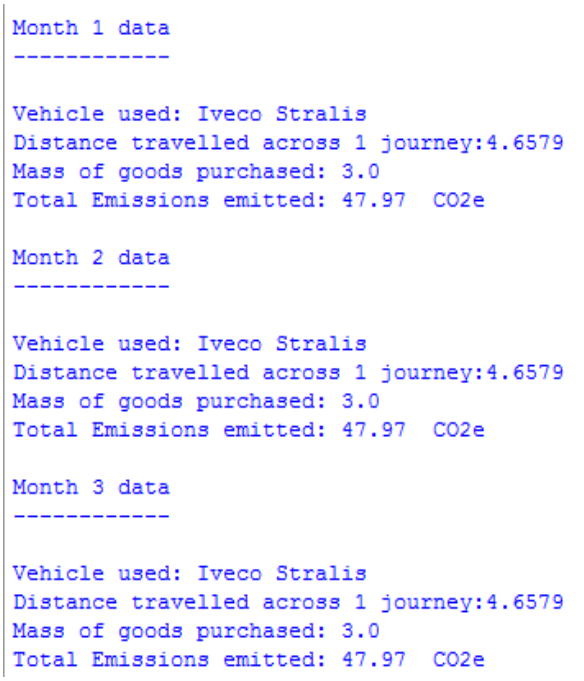
To test this, I will enter data for each possible value of num_months and see if all the data for each month is outputted. The aim of this is to show that the data for each month is outputted to the user so there will be repeated inputs as a result. The expected result is that the data for each month is outputted and then, the program moves onto determining the grade for each month of emissions. This will be shown using a print statement in the DetermineEmissionsScore() method. The table below shows all cases that need to be checked and their outcome.

Test No.	Description	Program Result	Outcome
13.1	Calculation_method = Fuel-Based Num_months = 2 Num_postcodes = 2 Postcodes = RG41 2EX, RG40 4JA Vehicle_name = 'Iveco Stralis' Quantity_Refriegrant_leaked = 0.12 Data same for both Months	Summary of Data for each Month ------------------------------ Month 1 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 2 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e	Pass

13.2	Test 14.1 but with Num_Months = 3	Summary of Data for each Month ------------------------------ Month 1 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 2 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 3 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e	Pass
13.3	Test 14.1 but with Num_Months = 4	Summary of Data for each Month ------------------------------ Month 1 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 2 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 3 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 4 data ------------ Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e	Pass

13.4	Test 14.1 but with Num_Months = 5	Summary of Data for each Month ---------- Month 1 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 2 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 3 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 4 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 5 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e	Pass

13.5	Test 14.1 but with Num_Months = 6	Summary of Data for each Month ---------- Month 1 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 2 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 3 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 4 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 5 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e Month 6 data ---------- Vehicle used: Iveco Stralis Distance travelled in 1 journey: 4.6579 Quantity Refrigerant Leaked in 1 journey: 0.12 Total emissions emitted in 1 journey: 2.13 CO2e	Pass

13.6	Calculation_Method = Distance-based Method Num_months = 2 Num_postcodes = 2 Postcodes = RG41 2EX, RG40 4JA Vehicle_name = Iveco Stralis Mass_goods_purchased = 3 Data same for both Months		Pass
13.7	Test 14.6 but with Num_Months = 3		Pass

13.8	Test 14.6 but with Num_Months = 4	Summary of Data for each Month ------------------------------ Month 1 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 2 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 3 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 4 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e	Pass

13.9	Test 14.6 but with Num_Months = 5	Summary of Data for each Month ------------------------------ Month 1 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 2 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 3 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 4 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 5 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e	Pass

13.10	Test 14.6 but with Num_Months = 6	Summary of Data for each Month ------------------------------ Month 1 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 2 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 3 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 4 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 5 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e Month 6 data ------------ Vehicle used: Iveco Stralis Distance travelled across 1 journey:4.6579 Mass of goods purchased: 3.0 Total Emissions emitted: 47.97 CO2e	Pass

As shown, all the data is outputted for each month of data that is entered. As a result, the output method on the front end is fully working.

TEST 14

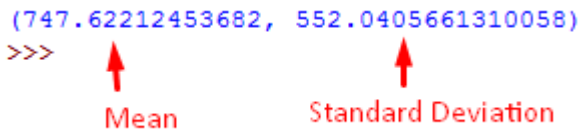
14	Testing the randomisation and stats calculation methods in the scoring algorithm in objectives 4.1 and 4.2	• The databases created have the correct number of records and the correct field names. (Tests that randomisation processes have happened in 4.1a,4.1b,4.1c,4.1d) and that the total emissions have been calculated using the random values testing 4.1e • The values in each cell are valid and within range (shown using DB Browser). Testing 4.1a,4.1b,4.1c,4.1d • The Mean and Standard Deviation calculated by the computer matches my calculation of the Mean and Standard Deviation done by hand. Tests objective 4.2a and 4.2b • I will do this for a small number of rounds with the logic being that if it works for a small no. of rounds, it is bound to work for 1000 rounds. Tests objective 4.1f

I will be carrying out this test using 5 rounds so it is easier for me to calculate by hand and check that the Maths adds up. I will do this twice for both calculation methods.

Fuel-based Method

Using 5 rounds, the data generated is given below along with the mean and Standard Deviation is given below:

round	total_emissions	vehicle_id	quantity_refrigerant_leaked	distance
Filter	Filter	Filter	Filter	Filter
1	58.79784361341	1	0.114086701	852.195
2	706.8078859716	6	0.139271112	1679.556
3	462.94265718689	1	0.286312429	6722.266
4	983.31512909663	8	0.137380981	2020.786
5	1526.24710681...	8	0.262143307	3136.41



As shown, all the fields are given correctly according to the data dictionary I have defined in Stage 1 of the design section. All the records have valid data inside of them which indicates that the randomisation functions are operating correctly. Now to check the calculations:

Mean:

(58.79+706.81+462.94+983.32+1526.25)/5 = 747.6 (1dp)

Standard Deviation:

Variance =((747.6-58.79)^2+(747.6-706.81)^2+(747.6 -462.94)^2+(747.6 -983.32)^2+(747.6-1526.25)^2)/4

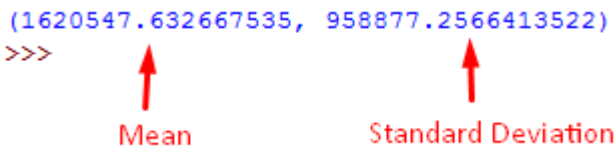
SD = Variance ^^ 0.5 = 552.0 (1dp)

As shown, both the Mean and Standard Deviation Calculations are correct as I have manually done both by hand and a calculator.

Distance-based

The database created for the distance-based method is now shown below along with the Mean and Standard Deviation calculated by the program

round	total_emissions	vehicle_id	mass_goods_purchased	distance
Filter	Filter	Filter	Filter	Filter
1	3029733.07121...	1	288.241	6670.333
2	1599337.25433...	1	374.049	2713.379
3	761110.438372...	4	121.354	1827.102
4	1993938.66554...	6	124.92	4649.958
5	718618.733868...	7	245.974	712.937



As shown, all the fields are given correctly according to the data dictionary I have defined in Stage 1 of the design section. All the records have valid data inside of them which indicates that the randomisation functions are operating correctly. Now to check the calculations:

Mean:

(3029733.08+1599337.25+761110.44+1993938.67+718618.73)/5 = 1620547 (nearest whole no.)

Standard Deviation:

Variance = ((1620547-3029733.08)^2+(1620547-1599337.25)^2+(1620547-761110.44)^2+(1620547-1993938.67)^2)+ (1620547-718618.73)/4

Standard Deviation = Variance^0.5 = 958877 (nearest whole no.)

As shown, both the Mean and Standard Deviation Calculations are correct as I have manually done both by hand and a calculator.

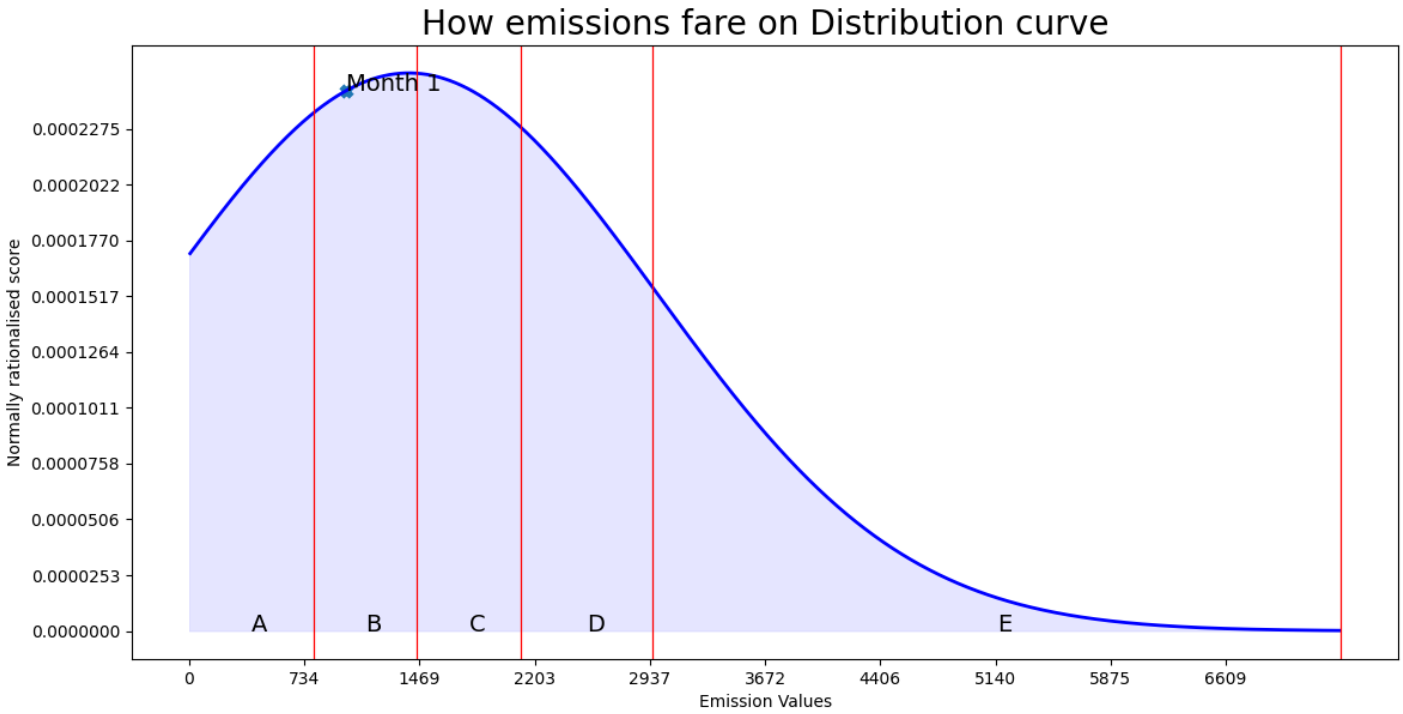
Therefore, because the rounds=5 case is valid, it is bound to work for the rounds=1000 case which will be used in my real code. Hence, the test is successful.

TEST 15

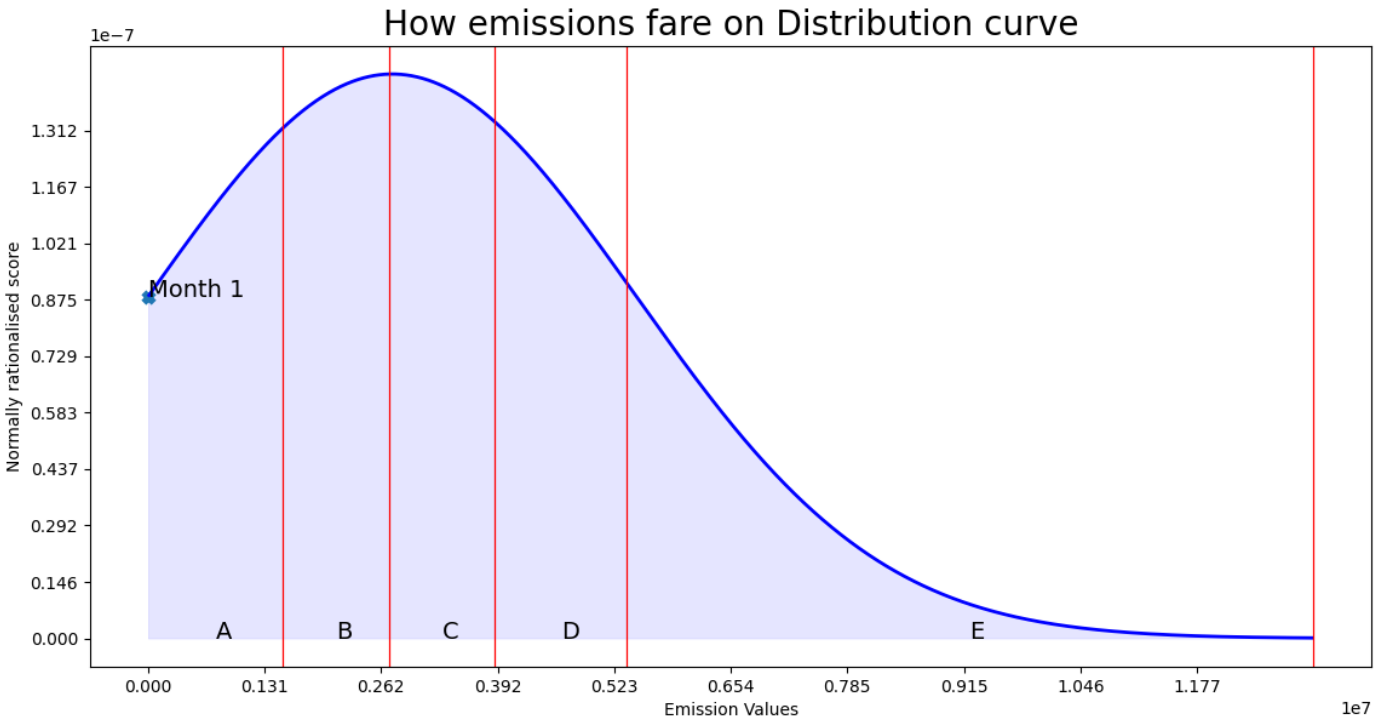
15	Verifying that the Grade Boundaries for scoring are fair. Testing Objective 4.3 and 4.4	• The diagrams of the normal distributions must show the Grade boundaries. Tests 4.4b • This must show grade boundaries that are positively determined with the peak of the normal distribution leading to a B or C grade. Tests 4.3b.

Because the mean and standard deviation is determined beforehand, I can just show a diagram of the normal distribution for the Num_Months = 2 case with some valid data as the grade boundaries are then subsequently fixed. Below show the Normal Distributions for both methods:

Fuel-based Method:



Distance-based Method:



As shown by both graphs, the peak of the normal distribution represents a positive or average grade (B in fuel-based and C in distance-based). This means that the grade boundaries are fair according to my test.

TEST 16

16	Checking that the range of x values are correct when plotting the normal distribution.	• Like Test 14, I will use a small number of rounds and show that it works correctly meaning that it works for many rounds.

This will be carried out in a similar vein to test 14. I will consider the rounds = 5 case and show that if it works for rounds = 5, it works for rounds = 1000. I will have 2 tests, one for each calculation method.

Fuel-Based method

round	total_emissions	vehicle_id	quantity_refrigerant_leaked	distance
Filter	Filter	Filter	Filter	Filter
1	1557.6163	5	0.134591679037216	2746.10324159284
2	181.8227	2	0.0387483727494175	4625.77785939907
3	234.4983	2	0.226737943296928	5959.56266398623
4	337.2577	1	0.169069036616447	4898.03781820568
5	323.0477	7	0.147997814259846	1345.01240072025

As shown, the maximum total emissions is 1557.6163. The maximum value given by my code is:

```
1557.6163
```

This matches with what my answer is for the maximum so the program is correct for the fuel-based method.

Distance-based Method

round	total_emissions	vehicle_id	mass_goods_purchased	distance
Filter	Filter	Filter	Filter	Filter
1	4728457.9462	1	451.984923718092	6638.87491245831
2	7356683.0831	5	305.546277991396	5875.52763165073
3	3675910.9908	7	420.505510726902	2133.21740444298
4	343863.9256	2	52.6092480054231	4147.85366900097
5	338139.697	2	377.82696675519	567.939547686355

As shown, the maximum total emissions is 7356683.0831. The maximum total emissions given by the program is:

```
7356683.0831
```

The program matches my answer, so it works for the distance-based method.

As shown, the program gives the maximum total emissions value, so it knows what range of x values it needs to plot for the normal distribution. Hence, this test passes.

TEST 17

17	Verifying that the Grade determined by the Program matches the grade boundaries. Testing Objective 4.3	• That the grade boundaries should correctly imply the grade given by the program.

To do this, I will show 5 different emissions totals that should give the grades A,B,C,D,E according to the grade boundaries. I will then show the output by the program with the grade. The first 5 tests show the fuel-based method and the last 5 show the distance-based method. The Grade Boundaries are shown below (first one is A, second is B and so on):

Fuel-based Method:

```
[792.4639213245125, 1450.2025907172056, 2116.9444820960202, 2954.1555708730575, 7343.5]
```

Distance-based Method:

```
[1511062.6933578916, 2705614.332811833, 3893274.6594753996, 5368376.628471041, 13078192.0]
```

Test No.	Emissions Total	Expected Grade	Program Grade	Outcome
17.1	300	A	Month 1 ------- Total Emissions: 300 CO2e Grade: A	Pass
17.2	1000	B	Month 2 ------- Total Emissions: 1000 CO2e Grade: B	Pass
17.3	1500	C	Month 3 ------- Total Emissions: 1500 CO2e Grade: C	Pass
17.4	2600	D	Month 4 ------- Total Emissions: 2600 CO2e Grade: D	Pass
17.5	5140	E	Month 5 ------- Total Emissions: 5140 CO2e Grade: E	Pass
17.6	1000000	A	Month 1 ------- Total Emissions: 1000000 CO2e Grade: A	Pass
17.7	2000000	B	Month 1 ------- Total Emissions: 2000000 CO2e Grade: B	Pass
17.8	3000000	C	Month 2 ------- Total Emissions: 3000000 CO2e Grade: C	Pass

17.9	4000000	D	Month 3 ------- Total Emissions: 4000000 CO2e Grade: D	Pass
17.10	6000000	E	Month 4 ------- Total Emissions: 6000000 CO2e Grade: E	Pass

As shown, all the emissions totals give the correct grades when inputted in by the program. Hence, the scoring model is fully working.

TEST 18

18	Testing that the Normal Distribution has all the required features. Testing Objective 4.4	• The Normal Distribution shows all the total emissions calculated based on the user input. Tests 4.4a • The x axis should say "Emission values" and the y axis should say "normally rationalised score" • The Grade Boundaries should be correctly placed on the normal distributions. Tests 4.4b

The 2 Normal Distributions are shown below with the num_months = 2 case for each method. Only the 2 month case needs to be checked as the same code is used for any number of months of data entered by the user. The Grade boundaries for the Fuel and Distance-based methods are shown below:

Fuel-based method:

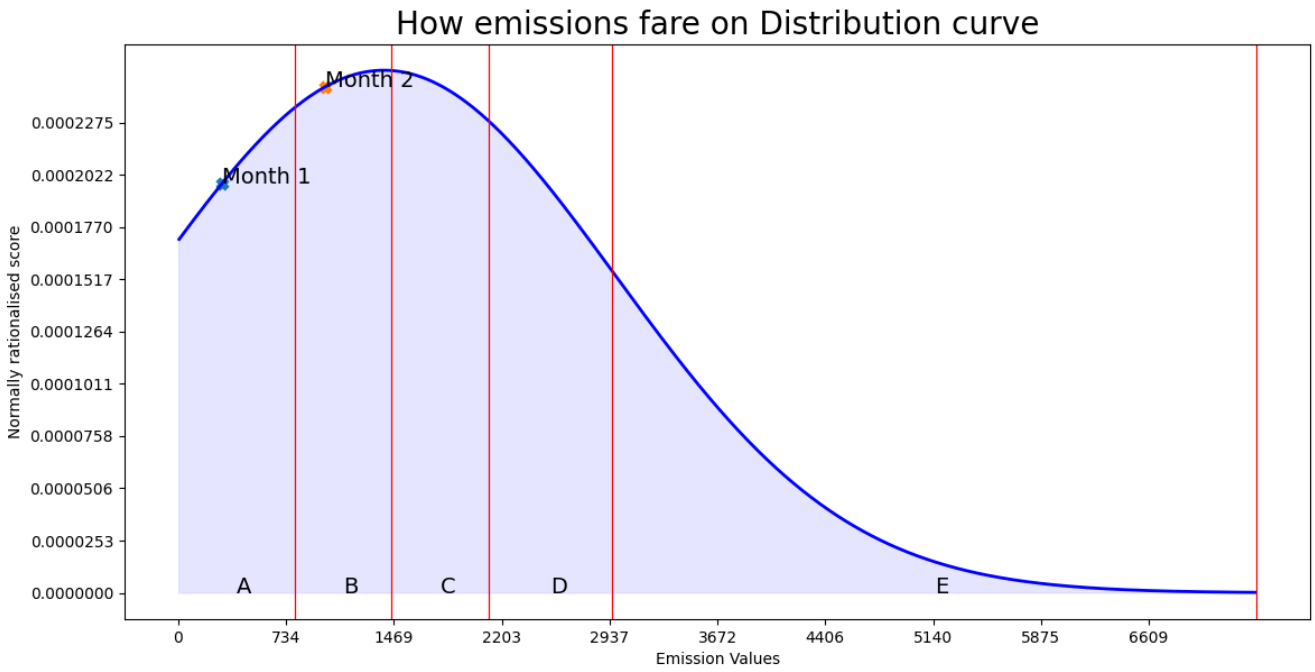
```
[792.4639213245125, 1450.2025907172056, 2116.9444820960202, 2954.1555708730575, 7343.5]
```

Distance-based method:

```
[1511062.6933578916, 2705614.332811833, 3893274.6594753996, 5368376.628471041, 13078192.0]
```

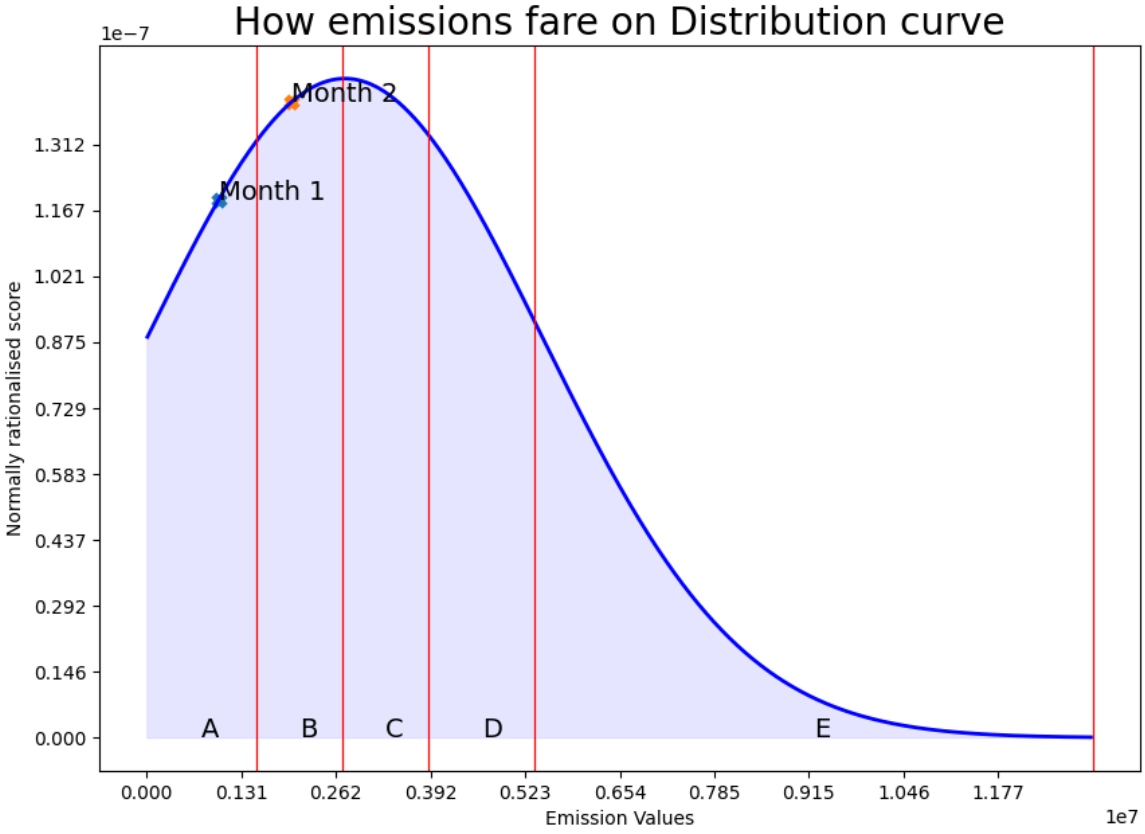
Fuel-based

2 months of data:



Distance-based method

2 months of data:



As shown in all these screenshots, the Grade boundaries are shown in red with labels in the middle of each grade region indicating the grade. The grade boundaries are in the right position. The months have all been labelled on the Normal Distribution with a label that specifically shows which month is which. The x axis and y axis labels are correct as specified in the design of this test too. It shows the total emissions of both months as well. This means that the Normal distribution has the required features meaning that this test has passed.

TEST 19

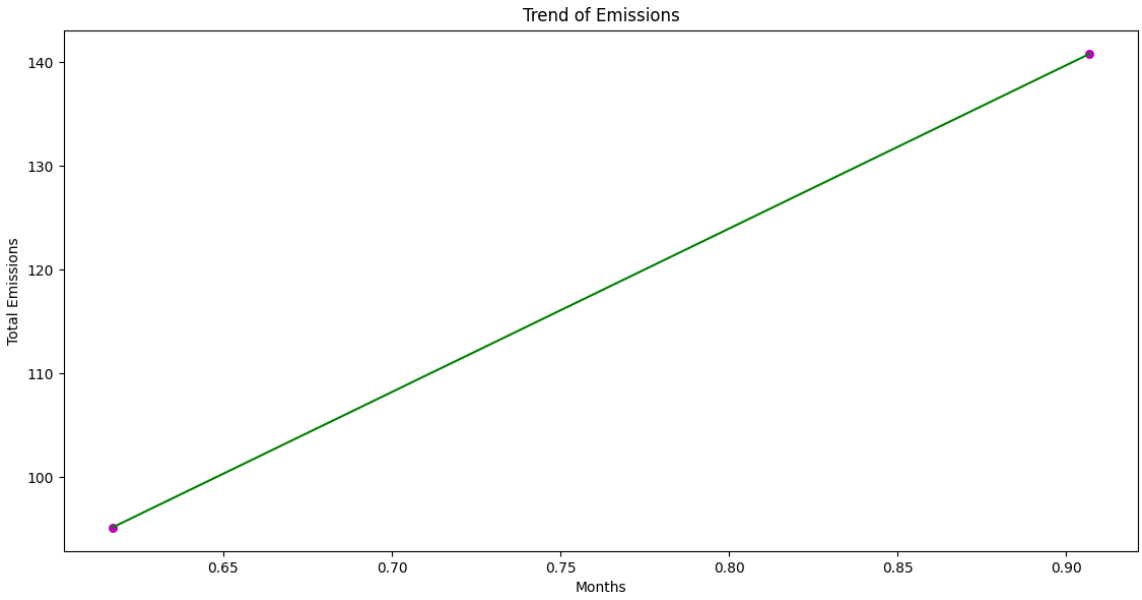
19	Testing that a Graph is shown to the user showing the required features. Testing objective 5.1	• The Graph should show 'months' on the x axis and 'emissions' on the y axis. • The emissions of each month should be plotted as points on the graph and a trend line should be running through it showing the overall trend of emissions. (Tests 5.1b)

PAGE 47

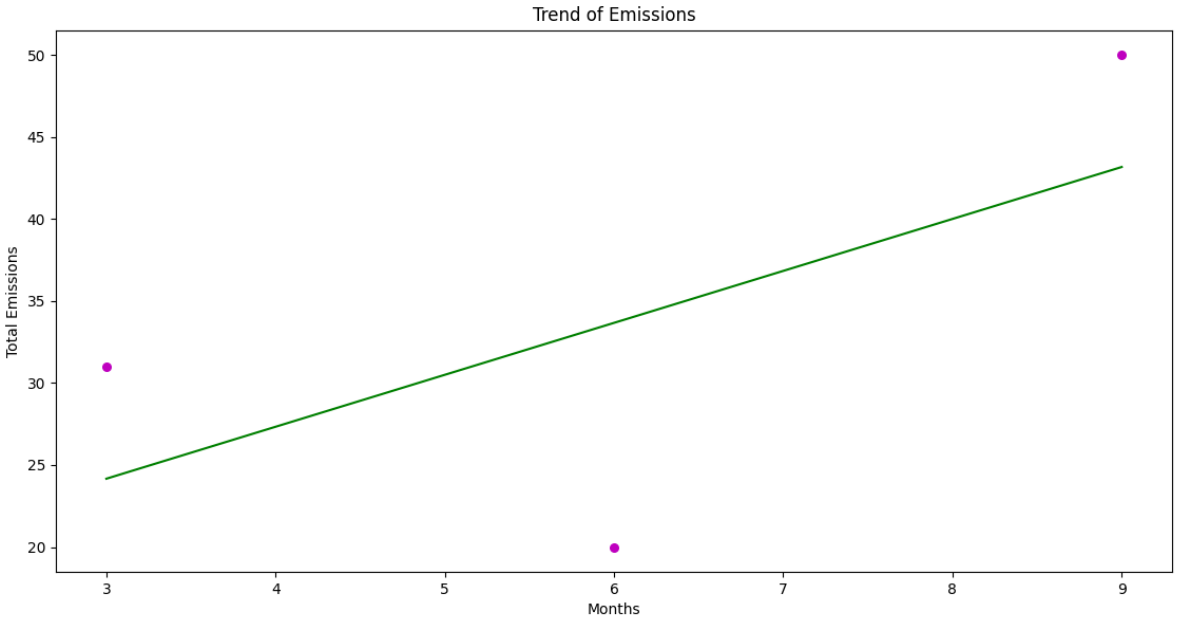
		• Also, the title should be 'Trend of Emissions'

This will be like test 17 but I will only be doing 6 graphs instead of 12 because the same algorithm is used for both calculation methods to create the graph. NB: I have not used the emission formulae in this. This is just random data entered into the linear regression algorithm.

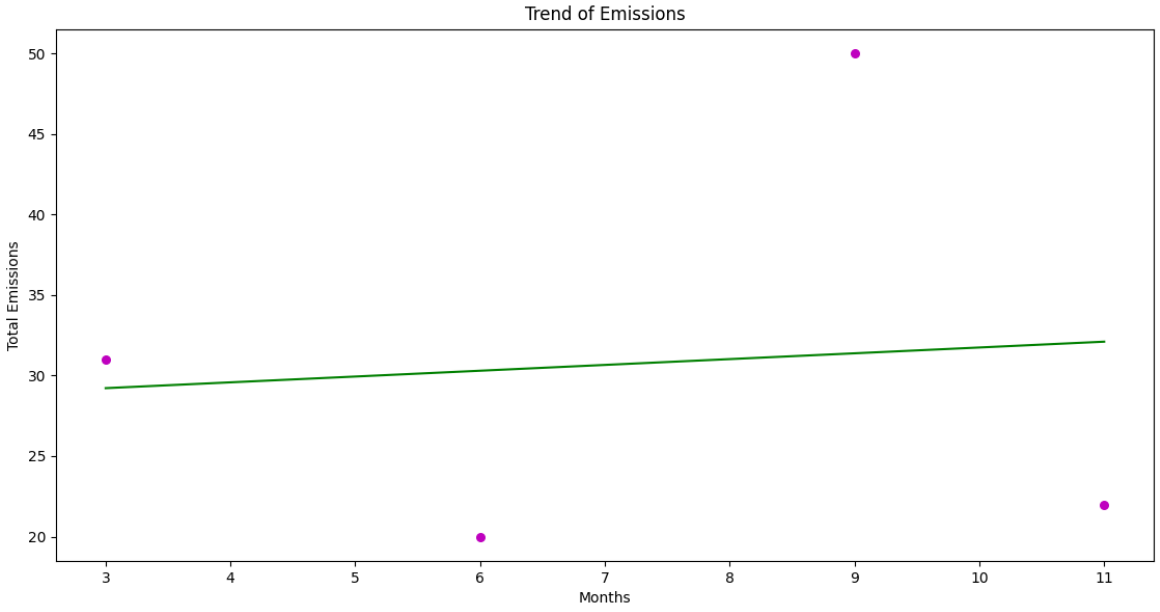
2 Months of Data:



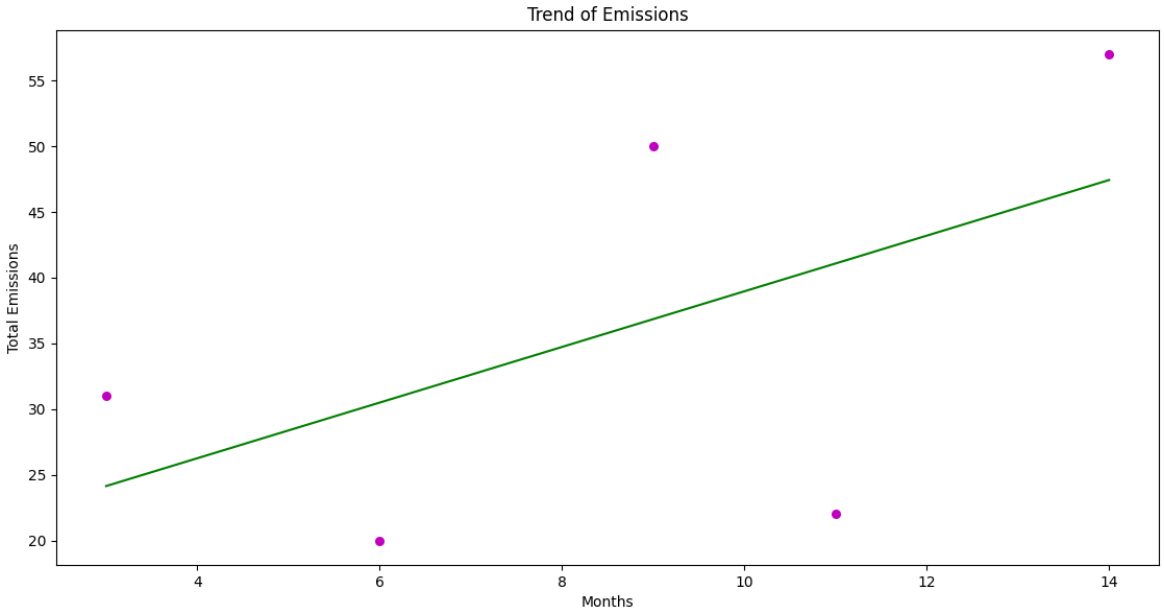
3 Months of Data:



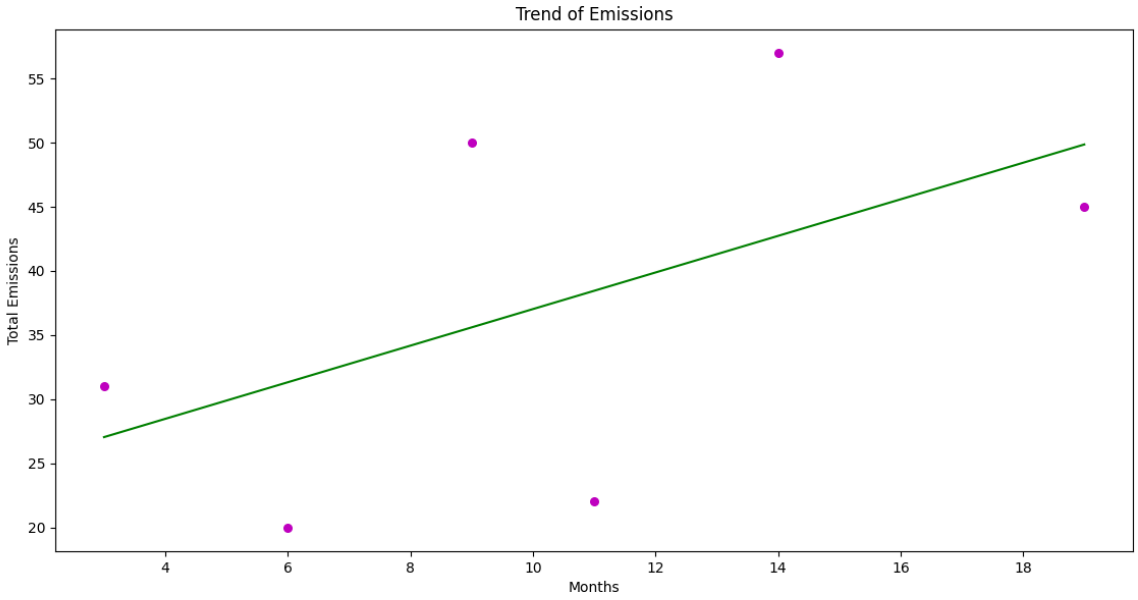
4 Months of Data:



5 Months of Data:



6 Months of Data:



As shown on all 6 of these graphs, the total emissions from each month are plotted on the graph with a trend line going through them, the axes labels are correct according to the test requirements and this is the same for the title. Hence, this test passes.

TEST 20

20	Testing that the program picks the more accurate trend line from the 2 different methods based on the r-squared score. Testing Objective 5.1.	• The Trend line should be an as accurate representation of the data as possible (Testing 5.1a) • The trend line with the r-squared score that is closest to 1 is the trend line outputted by the user.

For this test, I will add print statements to show the r-squared scores obtained from the trend lines created from the Least Squares and Gradient Descent methods and show that the one with the r-squared score closest to 1 is the trend line that is presented to the user. The modified code with the relevant print statements is shown below:

```
def Graph_and_Improve(db,x,y,calculation_method,attributes): # Interface

    coefficients = Gradient_Descent(x,y) # Determines coefficients from gradient descent
    coefficients1 = Least_Squares(x,y) # Determines coefficients from least squares.
    r_squared = calculate_r_squared(x,y,coefficients)
    r_squared_1 = calculate_r_squared(x,y,coefficients1)

    print('Gradient Descent R-squared score: ',r_squared)
    print('Least Squares R-squared score: ',r_squared_1)

    if abs(r_squared-1)<abs(r_squared_1-1): # Determines which set of coefficients are more accurate and plots the more accurate trend line.
        plot_regression_line(x,y,coefficients) # Plots Gradient Descent Trend Line
        print('Gradient Descent trend line plotted')
        improve_emissions(db,coefficients,x,y,calculation_method,attributes)
    else:
        plot_regression_line(x,y,coefficients1) # Plots Least Squares Trend line
        print('Least Squares trend line plotted')
        improve_emissions(db,coefficients1,x,y,calculation_method,attributes)
```

Let the dataset have x values [1,2,3,4,5,6] and y values [32,37,41,49,58,63]. The output is:

```
Gradient Descent R-squared score:  0.9800581842332714
Least Squares R-squared score:  0.9842497430626926
Least Squares trend line plotted
```

Hence, the program should draw the trend line associated with the Least-Squares method because this has a r-squared score closer to 1 and this is done by the program. Hence, this test passes as shown by this example.

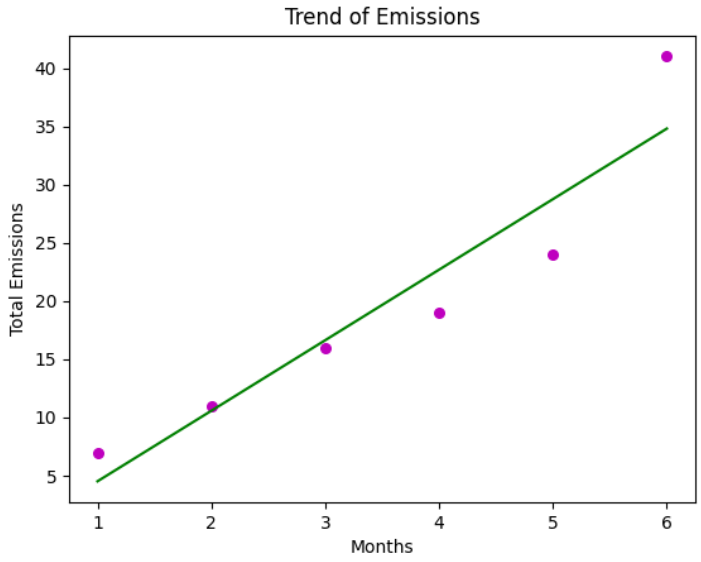
TEST 21

21	Testing that the Program should correctly identify whether emissions are generally rising or falling over time. Testing Objective 5.2	• Showing that based on the trend line, the program can identify whether emissions are rising or falling over time.

I will be doing 2 cases for this, one for data that produces an upward sloping trendline and one for data that produces a downward sloping trendline.

Upward Sloping Trendline:

[1,2,3,4,5,6] as x values and [7,11,16,19,24,41] as y values gives the graph and program output:

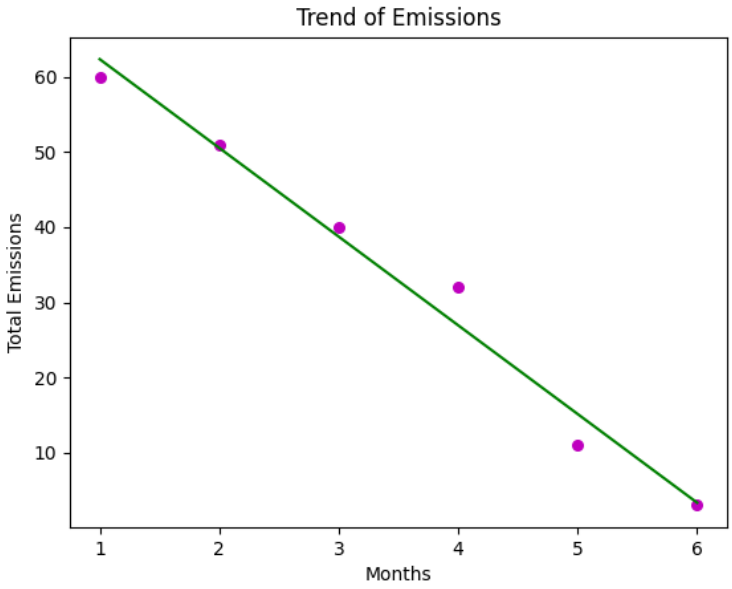



```
Over time, your emissions have not been falling. Improvement is required.
```

This is correct as emissions have been generally rising over time and the program output says this to the user.

Downward sloping trendline:

[1,2,3,4,5,6] as x values and [60,51,40,32,11,3] as y values gives the graph and program output:




```
Your emissions are falling over time. However, you can still improve.
```

This is correct as emissions have been generally falling over time and the program output shows this.

Overall, the program can identify whether emissions have been rising or falling over time meaning that the test has passed.

TEST 22

22	Testing that the Program gives the correct vehicles that produce lower emissions. Testing Objective 5.3	• Showing that the program gives vehicles that produce lower emissions by doing them by hand and comparing them with the program. • If no vehicles are more efficient, then the user must be told to minimize the distance travelled. (Testing 5.3b)

To test this, I will use 2 examples, one with no vehicles that are more efficient and one with some vehicles that are more efficient and show that the program gives the same results. I will do this for both methods.

Fuel-based Method

Some vehicles that are more efficient:

Use attributes distance = 3km, refrigerant name = R134a, quantity refrigerant leaked = 0.12, Vehicle = Iveco Stralis:

Total Emissions = 3 * 0.208 *2.02266 + 0.12 * 1.41 = 1.43 (2dp)

More efficient vehicles are Scania L Series with 0.38 (2dp), Scania P series with 0.29 (2dp),MAN TGA with 0.89 (2dp).

Now, according to the program:

```
Your emissions are falling over time. However, you can still improve.
Here are a list of more efficient vehicles (in ascending order) with the total emissions given if this vehicle was to be used.
Scania P Series :  0.29  CO2e
Scania L Series :  0.38  CO2e
MAN TGA :  0.89  CO2e
```

This is correct with my calculations by hand.

No Vehicles that are more efficient:

Use attributes distance = 3km, refrigerant name = R134a, quantity refrigerant leaked = 0.12, Vehicle = Scania P series

There are no more vehicles with more efficient emissions so the program should output an appropriate message saying that the user should look at minimizing their distance travelled.

```
Your emissions are falling over time. However, you can still improve.
Currently, there are no vehicles in the database that are more efficient in emissions. Try to reduce the distance travelled in your journey.
```

The program outputs the correct message. Hence, the algorithm works for the fuel-based method.

Distance-based Method

Some vehicles that are more efficient:

Use attributes Distance = 3km, refrigerant name = R134a, Mass of goods purchased = 15kg, Vehicle = Iveco S-way

Total Emissions = 15 * 3 * (2.68787+1.41) = 184.40 (2dp)

More efficient vehicles are Scania L Series with 70.911, Scania P Series with 70.911, Iveco Stralis with 154.47 (2dp) and Western Star 5700 XE with 154.47 (2dp).

The Program output is

```
Over time, your emissions have not been falling. Improvement is required.
Here are a list of more efficient vehicles (in ascending order) with the total emissions given if this vehicle was to be used.
Scania P Series :  70.91  CO2e
Scania L Series :  70.91  CO2e
Iveco Stralis :  154.47  CO2e
Western Star 5700 XE :  154.47  CO2e
```

Which is correct.

No Vehicles that are more efficient:

Use attributes distance = 3km, refrigerant name = R134a, Mass of goods purchased = 15kg, Vehicle = Scania L Series.

There are no vehicles that are more efficient than this one so the program should output an appropriate message.

The program output is:

```
Your emissions are falling over time. However, you can still improve.
Currently, there are no vehicles in the database that are more efficient in emissions. Try to reduce the distance travelled in your journey.
```

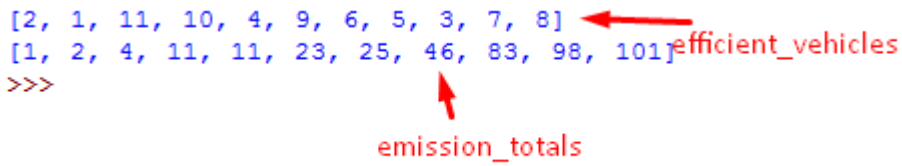
This is correct with the calculations done by hand.

Therefore, in both methods, the program correctly identifies vehicles that are more efficient in emissions and outputs it to the user, if not it tells the user to minimize the distance travelled. Hence, this test passes.

TEST 23

23	Testing that the Program outputs the vehicles in ascending order of total emissions. Testing Objective 5.4	• That the merge sort is working correctly. (Testing 5.3a) • That the output of more efficient vehicles is in ascending order of total emissions.

I will be inputting the array [2,1,83,11,46,25,98,101,23,11,4] into the merge sort algorithm along with [1,2,3,4,5,6,7,8,9,10,11] as the emission_totals and efficient_vehicles respectively. This part of the test tests only the merge sort so numbers is fine here. The expected results are [1,2,4,11,11,23,25,46,83,98,101] and [2,1,11,10,4,9,6,5,3,7,8]. The program gives:



This shows that the merge sort algorithm is working correctly.

The second part of the test is showing that the more efficient vehicles are outputted in the correct ascending order. Test 22 already shows this as the vehicles there are given in ascending order of emissions. This shows that the vehicles are outputted in the correct order. This means that this test passes.

TEST 24

24	Videos of both the scoring model and the Front End. Testing all objectives	• Shows all my code working from the front end with no errors. • That my scoring model runs correctly with no errors.

I will run 2 videos, 1 for the Front End and 1 for the Scoring Model. This is because the Scoring Model isn't called when the user uses the simulation, so I need to do it separately to show it working. The front-end video will contain 2 parts: one showing the fuel-based method working and the other showing the distance-based method working. The examples I will use in each are shown below:

Fuel-based Method

Num_months = 3

Month Number	Num_postcodes	Postcodes	Vehicle used	Quantity refrigerant Leaked
1	2	RG41 2EX, RG40 4JA	Iveco Stralis	0.1
2	3	RG41 2ex, RG40 4JA, sw1a 1aa	MAN TGM	0.18
3	4	RG41 2ex, RG40 4JA, sw1a 1aa, NE1 4ST	Western Star 4700 Series	0.26

Distance-based Method

Num_months = 2

Month Number	Num_postcodes	Postcodes	Vehicle used	Mass of goods Purchased
1	2	RG41 2ex,RG40 4JA	Scania P Series	120
2	5	RG41 2ex,RG40 4JA, SW1A 1AA, L4 0TH, NE1 4ST	Iveco S-Way	381

The Scoring Model video requires no user input and can just be run. I will not show the algorithm to calculate the stats in my video as it just prints out 2 values at the end of the process so there is nothing to show from doing a video. The link to the video are shown below:

https://youtu.be/oDPlIuGwaAA

The last part of the video cuts at the end but you can see from the video that all the columns are very random which shows the scoring model working.

CONCLUSION ON TESTING

As shown, all the tests have passed meaning that my program meets all the objectives set out and is also able to deal with any situation that arises when the user uses the program.

Evaluation

CLIENT FEEDBACK

My client (my dad) has given his opinion on the scorecard that I have made in this email.

Dear Aryam
Hope you are safe and well.

This is with reference to the Sustainability Scorecard programme. Thank you for the Application you have sent to me.

I have used the Application and can confirm that it satisfies the user requirements communicated to you in the trailing emails below. It gives me the following capabilities which are very helpful:

1. The Total emissions
2. An Emissions Grade
3. Indicator of how the Total Emissions have changed over time and the possible improvements that could be made to reduce them further.
4. The Grading system

The following additional features you have provided are also very relevant and helpful:

- Ability to save the Graphs that are generated
- The more emission-efficient vehicles appear in ascending order of total emissions as well

One small issue - the labels for the months on the scoring graph can intersect at times and the distance-based method can take some time to work. This may be due to the large numbers involved. Is there any way you can make the distance-based method run more quickly ?

Thank you again for the effort you have taken to create this powerful Application which is very helpful in our operations.

Kind Regards
Rajeev Rege
Alumnus Software
+44 7786 312787

My client is happy with the scorecard that I have made for him and I have detailed how I will fix any issues he has mentioned in my evaluation against each objective.

EVALUATION AGAINST THE OBJECTIVES

Below shows my evaluation of how my solution meets the objectives set out in the Analysis section. All the objectives were achieved but achieving some of the objectives were harder than others. This shows a detailed evaluation of each objective. This includes the challenges I faced while coding each objective as well as any specific improvements I could have made.

Objective 1

1.1

This objective was reasonably straightforward to achieve because all it required was a user input. Hence, only a TRY…EXCEPT block was needed inside the method executing this objective.

1.2

This objective was reasonably straightforward to achieve because all it required was a user input. Hence, only a TRY…EXCEPT block was needed inside the method executing this objective.

Objective 2

2.1

The postcodes were tricky to validate. First, I considered using a library called postcodes-uk but I realised that it checks for lists of POSSIBLE addresses, not whether the address actually exists or not. Hence, I decided to validate this by calculating the distance between the addresses earlier than I initially anticipated.

2.2

The algorithm to calculate the distance was easy to create once I became aware of the geopy library. Hence, I had no complications in implement this objective.

2.3

This objective was straightforward to implement because all it required was a user input. Hence, only a TRY…EXCEPT block was needed inside the method executing this objective.

2.4

This objective was reasonably straightforward to achieve because all it required was a user input. Hence, only a TRY…EXCEPT block was needed inside the method executing this objective.

2.5

This objective was slightly harder to implement. This is because initially, I had another objective in which the number of journeys made across the month was also to be entered by the user. However, I decided to take this out because I realised that it was more important to focus on the emissions released/journey rather than the total as this is a much stronger representation of the sustainability of a firm. This decision was made after I had coded the program calculating the total emissions and the validation methods in the front end. This meant that I had to make considerable changes to this section which took me a while to achieve as many niggling errors popped out when running this objective. The actual coding didn't cause too many problems but the change that I made to my code here made coding this part of my objectives harder than it should have been.

2.6

This was easy to implement as I just needed to implement a FOR loop.

2.7

This was easy to implement as it just required a FOR loop and some print statements inside it.

Objective 3

3.1

This objective is the same as objective 2.1. (see objective 2.1 for evaluation)

3.2

This objective is the same as objective 2.2. (see objective 2.2 for evaluation)

3.3

This objective is the same as objective 2.3. (see objective 2.3 for evaluation)

3.4

This objective was easy to implement as it just required a TRY…EXCEPT block to be created.

3.5

This objective faced similar difficulties to the ones in objective 2.5. Apart from this, the objective was easy to implement.

3.6

This was easy to implement as it is just a FOR loop

3.7

This was easy to implement as it is just some FOR loops with print statements inside.

Objective 4

4.1

Implementing the Monte Carlo Method required more work than anticipated. The maths required took some time to understand, particularly the LCG and Stein's Algorithm. I underestimated the coding required to meet the 3 conditions of the Linear Congruential Generator. Also, it took me a while to consider finding very large integers and dividing by a factor of 10 to get random decimal numbers from my LCG as well. A possible improvement could be to use a Mersenne Twister instead of a Linear Congruential Generator to generate random numbers as this is much more accurate and much quicker.

4.2

This objective was simple to implement as it just involved coding in some mathematical formulae. An overall improvement that I could have made that may improve the efficiency of this objective may be to use a Pandas Dataframe instead of sqlite3. This is because the Pandas library is much faster at mathematical calculations such as Mean and also has in-built methods to determine standard deviation. This makes my implementation my objective faster.

4.3

Determining the grade boundaries was tougher than I expected. This is because some parts of the Normal Distribution created goes into negative x values. This isn't valid as my x values are total emissions and these can't be negative. This was something I had not anticipated and the algorithm in my design to determine the grade boundaries was modified after I tested the system. Hence, I had to put more thought into achieving this objective. The function required

4.4

Learning how to use the matplotlib library was tricky in some places. This is because it was hard to ensure that the labels for the grade regions are always indicated in the right places. Hence, I had to use some maths to guarantee that the label for the grade regions always occurs in the middle of the region. This was the same for the range of x values my emissions can take. The function DetermineMaxEmissions in the ScoringEmissions code lines 313 to 322 was an after-thought as I realised when testing that my range of x-values didn't cover the whole Normal Distribution. Both of my client's issues lie in this objective. The main reason why the distance-based method took so long as my client stated in the email above was because the values needed to the plotted for the Normal Distribution was huge. To counteract this, I could make the step in between the x-values needed to be plotted larger.

Currently, I have found the maximum possible emissions in the ScoresEmissionsFuel or ScoresEmissionsDistance tables in my database and set x values to be all x values starting from 0 with a step of 0.5, independent of the calculation method. For the distance-based method, because the values are much larger, I could make the step between x values larger. To be even more accurate, I could create a mathematical model that determines the step based on maximum possible emissions total in my database, to guarantee that the distance-based method produces a normal distribution as accurate as the fuel-based method normal distribution.

To address the second issue, I can create a separate function to determine the coordinates of where my month labels are placed to prevent them from intersecting. I could find the dimensions of the label as a rectangle and then use an IF statement to determine whether any 2 label rectangles intersect with each other. I wish to modify this objective in the future as I think that it will make my simulation more user-friendly, even though this objective was successfully met.

Objective 5

5.1

Implementing the model to determine an accurate trend line (Graphing Model) was not overly difficult once the algorithms were free of semantic errors I had encountered. The use of numpy arrays made implementing these algorithms much easier as numpy arrays have sum and mean methods associated with them making them more powerful than standard python arrays. Overall, I feel that this objective was implemented very well.

5.2

This was very easy to achieve as it just required me to create an IF statement to compare the gradient coefficient of my trend line. Hence, this objective was implemented well.

5.3

Implementing the algorithm to determine more efficient vehicles was easy as it just required me to run functions created for objectives 2.5 and 3.5. The merge sort algorithm to sort the emissions totals and subsequently, the vehicles was easy to implement as merge sort was taught to me in class. Then, telling the user to minimise the distance travelled (if no vehicles are more efficient) was easy to implement using an IF statement. Hence, this objective was implemented well.

Conclusion

All objectives of my program were successfully implemented; some easier to implement than others. In terms of improvements, I would like to make the necessary improvements my father has wished for as well as how user-friendly objective 4.4 is although the latter is a small issue.

CONCLUSION

Overall, I have really enjoyed this project. I have learnt a lot about the different types of emissions, how they are calculated and how they are graded by organisations such as CDP. From a coding perspective, I learnt a lot about libraries such as matplotlib and geopy, which was very eye-opening as well. Furthermore, I was able to complete the main bulk of coding my technical solution within the time frames specified in the critical path.

I think that my code is understandable for any programmer that reads it as it has been effectively commented, showing what each function, method or class does and all my variables are very clearly named. Also, I have made sure that I have programmed defensively, having put in validation for every single user input to prevent the program from crashing.

What I like about my solution is how it groups together the common things between the 2 calculation methods. This makes my code a lot easier to understand and debug when testing it as I only had to change things such as taking out the number of journeys formula component once, not twice. I also liked the fact that it is very flexible. No matter how fuels, refrigerants and vehicles change over time, all one needs to do to is add the new fuel, refrigerant, or vehicle to my database. Nothing in my code (the front end, scoring, graphing, or calculating total emissions) needs to change which makes it easy to scale up my database without any hassle.

Therefore, I think that all the objectives I have set out in the Analysis stage have been met. Also, my client was very happy with my solution overall. It is a tool that is more powerful than I expected because I didn't consider that matplotlib allows you to save graphs and adjust its size which my father found to be a rather useful feature. This project was a steep learning curve for me, and I look forward to improving this project in the ways I have specified in the future.

Full Python Solution

CREATE_DATABASE_EMISSIONS

```python
# Creating the Database for the Fuel-based Method

import sqlite3

# Intialise database.

def init_db(db:sqlite3.Connection):

    create_fuel_table(db)
    create_refrigerant_table(db)
    create_vehicles_table(db)
    create_scores_emissions_fuel_table(db)
    create_score_emissions_distance_table(db)

def create_fuel_table(db:sqlite3.Connection):

    try:
        # Get Database cursor object
        c = db.cursor()

        # Remove existing Fuel table (if there is one) Doing this to make it easier to re-run the program when de-bugging

        c.execute("DROP TABLE IF EXISTS Fuel")

        # Create Fuel table

        sql = '''CREATE TABLE IF NOT EXISTS Fuel (
                                    fuel_id INTEGER PRIMARY KEY AUTOINCREMENT,
                                    fuel_name VARCHAR NOT NULL,
                                    emission_factor REAL NOT NULL)
                                '''
        c.execute(sql)
        db.commit()
        print('Fuel Table has been initialised')
    # If there is an error in creating user table, it explains the error in the form of an exception
    except sqlite3.Error as e:

        db.rollback()
        print('Error. Cannot create Fuel Table. Reason:',e)
        raise e

def create_refrigerant_table(db:sqlite3.Connection):

    try:
        # Get Database Cursor Object
        c = db.cursor()
        # Remove existing Refrigerant Table (if there is one)
        c.execute("DROP TABLE IF EXISTS Refrigerant")

        # Create Refrigerant Table
        sql = '''CREATE TABLE IF NOT EXISTS Refrigerant (
                                        refrigerant_id INTEGER PRIMARY KEY AUTOINCREMENT,
                                        refrigerant_name VARCHAR NOT NULL,
                                        gwp REAL NOT NULL)
                                    '''
        c.execute(sql)
        db.commit()
        print('Refrigerant Table has been initlialised')

    # If there is an error in creating user table, it explains the error in the form of an exception
    except sqlite3.Error as e:
        db.rollback()
        print('Error. Cannot create Refrigerant table. Reason:',e)
        raise e
```

```
67  def create_vehicles_table(db:sqlite3.Connection):
68
69      try:
70          # Get Database Cursor Object
71          c = db.cursor()
72
73          # Remove any Vehicles Table (if there is one)
74          c.execute("DROP TABLE IF EXISTS Vehicles")
75
76          # Create Vehicles Table
77          sql = '''CREATE TABLE IF NOT EXISTS Vehicles (
78                                      vehicle_id INTEGER PRIMARY KEY AUTOINCREMENT,
79                                      vehicle_name VARCHAR NOT NULL,
80                                      fuel_efficiency REAL NOT NULL,
81                                      fuel_id INTEGER NOT NULL,
82                                      refrigerant_id INTEGER NOT NULL,
83                                      FOREIGN KEY (fuel_id)
84                                          REFERENCES Fuel(fuel_id)
85                                              ON UPDATE CASCADE
86                                              ON DELETE CASCADE
87                                      FOREIGN KEY (refrigerant_id)
88                                          REFERENCES Refrigerant(refrigerant_id)
89                                              ON UPDATE CASCADE
90                                              ON DELETE CASCADE)
91                                  '''
92          c.execute(sql)
93          db.commit()
94          print('Vehicles table has been initialised')
95
96      # Handles any errors using an exception.
97      except sqlite3.Error as e:
98          db.rollback()
99          print('Error. Cannot create Vehicles table. Reason:',e)
100         raise e
```

```
102 def create_scores_emissions_fuel_table(db:sqlite3.Connection):
103 
104     try:
105         # Get database cursor object
106         c = db.cursor()
107 
108         #Remove any Score Emissions Fuel tables (if there is one)
109         c.execute("DROP TABLE IF EXISTS ScoresEmissionsFuel")
110 
111         #Create Emissions Fuel table
112         sql = '''CREATE TABLE IF NOT EXISTS  ScoresEmissionsFuel(
113                                         round INTEGER PRIMARY KEY AUTOINCREMENT,
114                                         total_emissions REAL NOT NULL,
115                                         vehicle_id INTEGER NOT NULL,
116                                         quantity_refrigerant_leaked REAL NOT NULL,
117                                         distance REAL NOT NULL,
118                                         FOREIGN KEY (vehicle_id)
119                                             REFERENCES Vehicles(vehicles_id)
120                                                 ON UPDATE CASCADE
121                                                 ON DELETE CASCADE  )
122                                 '''
123         c.execute(sql)
124         db.commit()
125         print('Scores Emissions Fuel Table has been initialised')
126 
127     except sqlite3.Error as e:
128         db.rollback()
129         print('Error. Cannot create Scores Emissions Fuel Table. Reason:',e)
130         raise e
131 
```

```
def create_score_emissions_distance_table(db:sqlite3.Connection):

    try:
        # Get database cursor object
        c = db.cursor()
        #Remove any Emissions Distance tables (if there is one)
        c.execute("DROP TABLE IF EXISTS ScoresEmissionsDistance")

        # Create Emissions Distance table
        sql = '''CREATE TABLE IF NOT EXISTS ScoresEmissionsDistance (
                                        round INTEGER PRIMARY KEY AUTOINCREMENT,
                                        total_emissions REAL NOT NULL,
                                        vehicle_id INTEGER NOT NULL,
                                        mass_goods_purchased REAL NOT NULL,
                                        distance REAL NOT NULL,
                                        FOREIGN KEY (vehicle_id)
                                            REFERENCES Vehicles(vehicles_id)
                                                ON UPDATE CASCADE
                                                ON DELETE CASCADE)
                                    '''
        c.execute(sql)
        db.commit()
        print('Scores Emissions Distance Table has been initialised')

    except sqlite3.Error as e:
        db.rollback()
        print('Error. Cannot create Scores Emissions Distance Table. Reason:',e)
        raise e

if __name__ == '__main__':

    # Setting up DB Connection

    DB_path = 'CalculateEmissions_db.db'
    db = sqlite3.connect(DB_path, detect_types=sqlite3.PARSE_DECLTYPES)
    db.row_factory = sqlite3.Row

    init_db(db)
```

TEST_DATABASE_EMISSIONS

```python
# Creating the Database for the Fuel-based Method

import sqlite3

# Intialise database.

def init_db(db:sqlite3.Connection):

    create_fuel_table(db)
    create_refrigerant_table(db)
    create_vehicles_table(db)
    create_scores_emissions_fuel_table(db)
    create_score_emissions_distance_table(db)

def create_fuel_table(db:sqlite3.Connection):

    try:
        # Get Database cursor object
        c = db.cursor()

        # Remove existing Fuel table (if there is one) Doing this to make it easier to re-run the program when de-bugging

        c.execute("DROP TABLE IF EXISTS Fuel")

        # Create Fuel table

        sql = '''CREATE TABLE IF NOT EXISTS Fuel (
                                    fuel_id INTEGER PRIMARY KEY AUTOINCREMENT,
                                    fuel_name VARCHAR NOT NULL,
                                    emission_factor REAL NOT NULL)
                                '''
        c.execute(sql)
        db.commit()
        print('Fuel Table has been initialised')
```

```python
def add_refrigerants(db:sqlite3.Connection):

    try:
        # Get database cursor object
        c = db.cursor()

        # Add test refrigerant
        sql = '''INSERT INTO Refrigerant VALUES
                        (NULL,'R134a',1.41)
                       '''
        c.execute(sql)
        db.commit()
        print('Test Refrigerants have been added.')

# If there is an error in adding the refrigerant data to the Refrigerant table, it explains the error in the form of an exception

    except sqlite3.Error as e:
        db.rollback()
        print('Error. Cannot add refrigerant test values. Reason:',e)
        raise e
```

```
def add_vehicles(db:sqlite3.Connection):

    try:
        #Get Database cursor object
        c = db.cursor()

        #Add Test Vehicles
        sql = '''INSERT INTO Vehicles VALUES
                        (NULL,'Scania L Series', 0.415,1,1),
                        (NULL,'Scania P Series', 0.237,1,1),
                        (NULL,'Western Star 4700 Series',0.394,2,1),
                        (NULL,'Western Star 5700 XE', 0.338,3,1),
                        (NULL,'Iveco S-way', 0.211,2,1),
                        (NULL,'Iveco Stralis', 0.208,3,1),
                        (NULL,'MAN TGA', 0.0893,2,1),
                        (NULL,'MAN TGM', 0.181,2,1)
                        '''
        c.execute(sql)
        db.commit()
        print('Test Vehicles have been added')

# If there is an error in adding vehicle data, it explains error in form of exception

    except sqlite3.Error as e:
        db.rollback()
        print('Error. Cannot add vehicle test values. Reason:',e)
        raise e

if __name__ == '__main__':

    db = sqlite3.connect('TestCalculateEmissions_db.db',detect_types=sqlite3.PARSE_DECLTYPES)
    db.row_factory = sqlite3.Row
    CreateDatabaseEmissions.init_db(db)
    add_fuels(db)
    add_refrigerants(db)
    add_vehicles(db)
```

CALCULATING_EMISSIONS

```
# Calculates the Emissions - Both Calculation Methods

from geopy.geocoders import Nominatim
from geopy import distance
import sqlite3

# Retrieves Database from files

DB_PATH = 'TestCalculateEmissions_db.db'

def get_db(DB_PATH): # Gets the database containing all the vehicles for the fuel-based and distance-based method

    db = sqlite3.connect(DB_PATH, detect_types=sqlite3.PARSE_DECLTYPES)
    db.row_factory = sqlite3.Row
    return db
```

```
def CalculateDistance(postcodes): # Calculates the distance between each consecutive pair of postcodes.

    geolocator = Nominatim(user_agent="CalculateEmissions")

    total_distance = 0

    for i in range(len(postcodes)-1):

        postcode1 = postcodes[i]
        postcode2 = postcodes[i+1]

        postcode1 = geolocator.geocode(postcode1) # Finds coordinates for both postcodes
        postcode2 = geolocator.geocode(postcode2)

        postcode1_lat = (postcode1.latitude) # Gets latiutude and longitude of both postcodes
        postcode1_lon = (postcode1.longitude)
        postcode2_lat = (postcode2.latitude)
        postcode2_lon = (postcode2.longitude)

        location1 = (postcode1_lat, postcode1_lon) # Summarises location of 2 postcodes
        location2 = (postcode2_lat, postcode2_lon)

        distance_between_2_locations = distance.distance(location1, location2).km # Finds distance between 2 locations in km
        total_distance += distance_between_2_locations # Adds it to the total distance travelled.

    total_distance = round(total_distance,4)

    return total_distance

def GetEmissionFactorForFuel(db:sqlite3.Connection, vehicle_name): # Gets the emission factor for the fuel vehicle uses

    c = db.cursor()
    sql = '''SELECT Fuel.emission_factor FROM Vehicles
             INNER JOIN Fuel ON Vehicles.fuel_id = Fuel.fuel_id
             WHERE vehicle_name=?'''

    emission_factor_object = c.execute(sql,(vehicle_name,))
    emission_factor = c.fetchone()[0]
    return emission_factor

def GetGWP(db:sqlite3.Connection, refrigerant_name): # Gets global warming potential of refrigerant used.

    c = db.cursor()
    gwp_object = c.execute("SELECT gwp FROM Refrigerant WHERE refrigerant_name=?", (refrigerant_name,))
    gwp = c.fetchone()[0]
    return gwp

# Fuel-Based Method Calculations

def GetFuelEfficiency(db:sqlite3.Connection, vehicle_name): # Fetches fuel efficiency of vehicle

    c = db.cursor()
    fuel_efficiency_object = c.execute("SELECT fuel_efficiency FROM Vehicles WHERE vehicle_name=?",(vehicle_name,))
    fuel_efficiency = c.fetchone()[0]
    return fuel efficiency
```

```
# Distance-based method calculations

def CalculateTotalEmissions_distance(vehicle_name,distance,mass_goods_purchased,emission_factor_fuel,gwp):

    emission_factor_vehicle = emission_factor_fuel + gwp
    total_emissions = mass_goods_purchased*distance*emission_factor_vehicle
    return total_emissions

# Overall Functions

def fuel_based(db,vehicle_name,distance,refrigerant_name,quantity_refrigerant_leaked):
    emission_factor = GetEmissionFactorForFuel(db,vehicle_name)
    gwp = GetGWP(db,refrigerant_name)
    fuel_efficiency = GetFuelEfficiency(db,vehicle_name)
    total_emissions = CalculateTotalEmissions_fuel(distance,fuel_efficiency,emission_factor,quantity_refrigerant_leaked,gwp)
    return total_emissions

def distance_based(db,vehicle_name,distance,refrigerant_name,mass_goods_purchased):

    emission_factor_fuel = GetEmissionFactorForFuel(db,vehicle_name)
    gwp = GetGWP(db,refrigerant_name)
    total_emissions = CalculateTotalEmissions_distance(vehicle_name,distance,mass_goods_purchased,emission_factor_fuel,gwp)
    return total_emissions
# Interface

def CalculateEmissions(calculation_method,attributes):
    if calculation_method == 0:
        total_emissions = fuel_based(attributes[0],attributes[1],attributes[2],attributes[3],attributes[4])
        return total_emissions
    else:
        total_emissions = distance_based(attributes[0],attributes[1],attributes[2],attributes[3],attributes[4])
        return total_emissions
```

```
if __name__ == '__main__':

    postcodes = ['SW1A 1AA','RG41 2EX']
    vehicle_name = 'Iveco S-way'
    quantity_refrigerant_leaked = 100
    refrigerant_name = 'R134a'
    mass_goods_purchased = 100

    db = get_db(DB_PATH)
    print(CalculateEmissions(0,[db,'Western Star 4700 Series',9,'R134a',0.11]))
```

FRONT_END (RUN BY USER)

```
# Front End Program - To be Run by the User

import sqlite3
import numpy as np
import CalculatingEmissions, ScoringEmissions, GraphEmissions

class Common_Components:

    # Constructor
    def __init__(self):

        self._vehicle_name = None
        self._distance = None
        self._emissions_total = None
        self._ refrigerant name = None
```

```
16
17      # Exceptions
18
19      def _vehicle_name_exception(self,max_id):
20          print('ERROR: Please enter an integer from 1 to '+str(max_id)+' to give the vehicle used.')
21
22      def _num_postcodes_exception(self,max_postcodes):
23          print('ERROR: Please enter an integer from 2 to '+str(max_postcodes)+' to give the no. of junctions in the transportation journey.')
24
25      def _calculate_distance_exception(self):
26          print('ERROR: One of these postcodes does not exist. Please enter existent postcodes.')
```

```
28       # Setters
29       def _set_vehicle_name(self,db):
30
31           c = db.cursor()
32           print()
33           print('Available Vehicles for Scoring: ') # Presents all vehicles to the user
34           print()
35           vehicles_object = c.execute('SELECT vehicle_name FROM vehicles')
36           vehicles = c.fetchall()
37
38           for vehicle in vehicles:
39               vehicle_id_object = c.execute('SELECT vehicle_id FROM Vehicles WHERE vehicle_name=?',(vehicle[0],))
40               vehicle_id = str(c.fetchone()[0])
41               print(vehicle_id + '. ',vehicle[0])
42
43           max_id_object = c.execute('SELECT MAX(vehicle_id) FROM Vehicles')
44           max_id = c.fetchone()[0]
45
46           valid = False
47           while valid is False:
48               print()
49               try:
50                   vehicle_id = int(input('Enter the vehicle (1-'+str(max_id)+') used in Transportation: '))
51                   if 0 < vehicle_id <= max_id:
52                       vehicle_name_object = c.execute('SELECT vehicle_name FROM Vehicles WHERE vehicle_id=?',(vehicle_id,))
53                       vehicle_name = c.fetchone()[0]
54                       self._vehicle_name = vehicle_name
55                       valid = True
56                   else:
57                       self._vehicle_name_exception(max_id)
58               except ValueError:
59                   self._vehicle_name_exception(max_id)
```

```
    def _set_distance(self):

        max_postcodes = 6
        print()
        print('The maximum number of postcodes that can be entered is '+ str(max_postcodes))
        valid  = False # Validates number of postcodes passed in journey
        while valid is False:
            print()
            try:
                num_postcodes = int(input('How many junctions are there in the transportation journey? '))
                if 1< num_postcodes <= max_postcodes:
                    valid = True
                else:
                    self._num_postcodes_exception(max_postcodes)
            except ValueError:
                self._num_postcodes_exception(max_postcodes)

        distance_successful = False # Calculates distance travelled
        while distance_successful is False:

            print()
            print('Please enter the postcodes of these junctions below: ')
            print('NB: Please ensure that the spaces in the address are in the right places.')
            postcodes = []
            for i in range(num_postcodes): # Validates each address entered in by calculating the distance between those 2 postcodes.
                print()
                address = input('Postcode ' + str(i+1) + ': ')
                postcodes.append(address)
            try:
                self._distance = CalculatingEmissions.CalculateDistance(postcodes)
                distance_successful = True
            except:
                self._calculate_distance_exception()
```

```
    def _set_refrigerant_name(self,db):
        c = db.cursor()
        sql = '''SELECT Refrigerant.refrigerant_name FROM Vehicles
                 INNER JOIN Refrigerant ON Vehicles.refrigerant_id=Refrigerant.refrigerant_id
                 WHERE Vehicles.vehicle_name=?'''
        refrigerant_name_object = c.execute(sql,(self._vehicle_name,))
        self._refrigerant_name = c.fetchone()[0]

    def _set_emissions_total(self,calculation_method,attributes):

        self._emissions_total = CalculatingEmissions.CalculateEmissions(calculation_method,attributes)

    def _set_all_month_data(self):
        pass
    # Output method
    def _print_all_month_data(self):
        pass
```

```
class Fuel_based(Common_Components): # Inherits from Common_Components

    def __init__(self):
        super().__init__()
        self._quantity_refrigerant_leaked = None

    # Exceptions
    def __quantity_refrigerant_exception(self):
        print('ERROR: Please enter a positive number. ')

    # Setters

    def __set_quantity_refrigerant_leaked(self):

        print()
        print('Please enter the quantity of refrigerant leaked by your vehicle.')
        print('The refrigerant used by your vehicle is ',self._refrigerant_name)
        valid = False
        while valid is False:
            print()
            try:
                quantity_refrigerant_leaked = float(input('Quantity of '+str(self._refrigerant_name)+' leaked: '))
                if quantity_refrigerant_leaked >= 0:
                    self._quantity_refrigerant_leaked = quantity_refrigerant_leaked
                    valid = True
                else:
                    self.__quantity_refrigerant_exception()
            except ValueError:
                self.__quantity_refrigerant_exception()
```

```
    def _set_all_month_data(self,db,month,calculation_method):
        print()
        print('Month '+str(month))
        print('-------')
        self._set_distance()
        self._set_vehicle_name(db)
        self._set_refrigerant_name(db)
        self.__set_quantity_refrigerant_leaked()
        self._set_emissions_total(calculation_method,[db,self._vehicle_name,self._distance,self._refrigerant_name,self._quantity_refrigerant_leaked])

    # Output Method

    def _print_all_month_data(self): # Overrides the method print_all_month_data in parent class

        print('Vehicle used: ',self._vehicle_name)
        print('Distance travelled in 1 journey: ',self._distance)
        print('Quantity Refrigerant Leaked in 1 journey: ',self._quantity_refrigerant_leaked)
        print('Total emissions emitted in 1 journey: ',str(round(self._emissions_total,2)),'CO2e')
```

```
class Distance_based(Common_Components): # Inherits from Common_Components

    def __init__(self):
        super().__init__()
        self._mass_purchased = None

    # Exceptions

    def __mass_exception(self):
        print("ERROR: Please enter a mass between 0 and 500kg.")

    # Setters
    def __set_mass_purchased(self):
        max_mass = 500
        print('Please enter the Mass of goods purchased below. NB: Maximum mass is 500 kg')
        valid = False
        while valid is False:
            print()
            try:
                mass_purchased = float(input('Mass of Goods purchased: '))
                if 0 < mass_purchased <= 500:
                    self._mass_purchased = mass_purchased
                    valid = True
                else:
                    self.__mass_exception()
            except ValueError:
                self.__mass_exception()
```

```
    def _set_all_month_data(self,db,month,calculation_method):
        print()
        print('Month '+str(month))
        print('-------')
        self._set_distance()
        self._set_vehicle_name(db)
        self._set_refrigerant_name(db)
        self.__set_mass_purchased()
        self._set_emissions_total(calculation_method,[db,self._vehicle_name,self._distance,self._refrigerant_name,self._mass_purchased])

    # Output method
    def _print_all_month_data(self): # Overrides print_all_month_data in parent class
        print('Vehicle used: '+self._vehicle_name)
        print('Distance travelled across 1 journey:'+str(self._distance))
        print('Mass of goods purchased:', str(self._mass_purchased))
        print('Total Emissions emitted:',str(round(self._emissions_total,2)),' CO2e')
```

```
207 class Months_of_Data: # Contains objects of one of the child classes. Composition
208
209     def __init__(self):
210         self.__calculation_method = None
211         self.__num_months = None
212         self.__objects = []
213         self.__list_of_emissions = []
214
215     # Exceptions
216
217     def _method_exception(self):
218         print('ERROR: Please enter 1 or 2.')
219
220     def _num_months_exception(self):
221         print('ERROR: Please enter an integer between 2 and 6 inclusive.')
222
223     # Setters
224
225     def __set_method_used(self):
226
227         print('Possible Calculation Methods:')
228         print()
229         print('1. Fuel-based Method')
230         print('2. Distance-based Method')
231
232         valid = False # Validates user input for calculation method
233         while valid is False:
234             print()
235             try:
236                 calculation_method = int(input('Enter the calculation method (1 or 2) you wish to use for scoring: '))
237                 if calculation_method == 1 or calculation_method == 2:
238                     self.__calculation_method = calculation_method - 1
239                     valid = True
240                 else:
241                     self._method_exception()
242             except ValueError:
243                 self._method_exception()
```

```
245     def __set_num_months(self):
246 
247         print()
248         print('Please enter how many months you want to enter data for: ')
249         print('NB: The maximum no. of months is 6.')
250 
251         max_months = 6
252         valid = False
253         while valid is False:
254             print()
255             try:
256                 num_months = int(input('No. of Months: '))
257                 if 1 < num_months <= max_months:
258                     self.__num_months = num_months
259                     valid = True
260                 else:
261                     self._num_months_exception()
262             except ValueError:
263                 self._num_months_exception()
264 
265     def __set_month_data(self,db):
266 
267         if self.__calculation_method == 0: # Creating fuel-based method objects
268             for i in range(self.__num_months):
269                 Month = Fuel_based()
270                 Month._set_all_month_data(db,i+1,self.__calculation_method)
271                 self.__list_of_emissions.append(Month._emissions_total)
272                 self.__objects.append(Month)
273         else:
274             for i in range(self.__num_months): # Creating distance-based method objects
275                 Month = Distance_based()
276                 Month._set_all_month_data(db,i+1,self.__calculation_method)
277                 self.__list_of_emissions.append(Month._emissions_total)
278                 self.__objects.append(Month)
```

```
280     # Methods acting on data entered in by user
281 
282     def __score_emissions(self,db): # Calls the scoring program to score all the emission totals
283         ScoringEmissions.DetermineEmissionsScore(db,self.__calculation_method,self.__list_of_emissions)
284 
285     def __graph_emissions(self,db): # Calls the graphing model
286         x_values = np.array([i for i in range(1,len(self.__list_of_emissions)+1)])
287         recent_month = self.__objects[len(self.__objects)-1] # Obtains the data from the most recent month.
288         if self.__calculation_method == 0:
289             attributes = [recent_month._distance,recent_month._refrigerant_name,recent_month._quantity_refrigerant_leaked]
290             #Takes all needed fuel-based attributes to calculate total emissions in determining more efficient vehicles.
291         else:
292             attributes = [recent_month._distance,recent_month._refrigerant_name,recent_month._mass_purchased]
293             # Takes all the needed distance-based attributes to calculate total emissions in determinin more efficient vehicles.
294         GraphEmissions.Graph_and_Improve(db,x_values,np.array(self.__list_of_emissions),self.__calculation_method,attributes)
```

```
295
296     # Output Method
297
298     def __print_all(self):
299
300         print()
301         print('Summary of Data for each Month')
302         print('------------------------------')
303         for i in range(len(self.__objects)): # Iterates through each month, printing all data
304             print()
305             print('Month ' + str(i+1)+' data')
306             print('------------')
307             print()
308             self.__objects[i]._print_all_month_data()
309
310     # Start Method
311
312     def start(self,db):
313         self.__set_method_used()
314         self.__set_num_months()
315         self.__set_month_data(db)
316         self.__print_all()
317         self.__score_emissions(db)
318         self.__graph_emissions(db)
319
320
321 if __name__ == '__main__':
322
323     DB_PATH = 'TestCalculateEmissions_db.db'
324     db = sqlite3.connect(DB_PATH,detect_types=sqlite3.PARSE_DECLTYPES)
325     db.row_factory = sqlite3.Row
326     c = db.cursor()
327
328     test = Months_of_Data()
329     test.start(db)
```

SCORING_EMISSIONS

```
# Scoring the Emissions
import CreateDatabaseEmissions, CalculatingEmissions, sqlite3, random
import numpy as np
import matplotlib.pyplot as plt
import datetime
from scipy.stats import norm

# Fuel-based and distance-based randomised processes

def LCG(divisor): #Implements the Linear Congruential Generator
    def find_coprime(divisor):
        def gcd(a,b): # Stein's Algorithm
            if (a == b):
                return a
            # GCD(0, b) == b; GCD(a, 0) == a,
            # GCD(0, 0) == 0
            if (a == 0):
                return b
            if (b == 0):
                return a
            # look for factors of 2
            # a is even
            if ((~a & 1) == 1):
                # b is odd
                if ((b & 1) == 1):
                    return gcd(a >> 1, b)
                else:
                    # both a and b are even
                    return (gcd(a >> 1, b >> 1)) << 1
            # a is odd, b is even
            if ((~b & 1) == 1):
                return gcd(a, b >> 1)
            # reduce larger number
            if (a > b):
                return gcd((a - b) >> 1, b)
            return gcd((b - a) >> 1, a)
```

```
        found = False
        i = 2
        while found is False:
            hcf = gcd(i,divisor)
            if hcf == 1: # Stops at the snmallest number coprime to the divisor.
                found = True
            else:
                i += 1
        return i

    def determine_a(divisor): # Determines the smallest value of a for any divisor for my LCG algorithm
        prime_factors = []
        found = False
        factor = 2
        while found is False: # Finds all prime factors of the divisor
            all_one_prime_factor = False
            while all_one_prime_factor is False: # If a prime is a factor, then it checks how many powers of this prime are factors of the divisor.
                if divisor % factor == 0:
                    divisor /= factor # it divides the divisor by that prime factor to ensure no prime factors repeat.
                    prime_factors.append(factor) # Adds the prime factor to the prime factor list
                else:
                    all_one_prime_factor = True # Leaves the while loop if there are no more prime factors
            if divisor == 1: # If all the prime factors are found, you leave the loop
                found = True
            else: # If not, you go onto the next number.
                factor += 1
```

```
        num_factors_2 = prime_factors.count(2) # Counts the number of factors of 2
        factors_of_a = list(dict.fromkeys(prime_factors)) # Finds the factors of a by removing repeats from the list. We need a to be divisible by all the prime
        if num_factors_2 >= 2:                            # factors of the divisor.
            a = int(np.prod(factors_of_a) * 2) # Multiplies by an extra factor of 2 if the divisor is a multiple of 4
        else:
            a = int(np.prod(factors_of_a))
        return a

    date = datetime.datetime.now().strftime('%M:%S.%f') # Determines the number of times the recurrence relation runs by evaluating the no. of microseconds
    num_iterations = int(date[9:])                      # Time is at during that minute and taking the last 3 numbers of that microsecond.

    a =  determine_a(divisor) # Determines an a value such that a-1 is divisible by 4
    c = find_coprime(divisor)  # divisor and c must be coprime to get the largest period
    x = int(date[8])  # Randomly determines a starting x value
    for i in range(num_iterations):
        x = (a*x+c)%(divisor+1)   # Computes the Recurrence relation
    if x == 0: # As the random number can never be 0.
        x += 1
    return x
```

```
def RandomiseVehicle(db:sqlite3.Connection):

    c = db.cursor()
    # Randomly choosing a vehicle

    max_id_object = c.execute('SELECT MAX(vehicle_id) FROM Vehicles')  # Gets the maximum possible value for the vehicle id.
    max_id = max_id_object.fetchone()[0]
    vehicle_id = LCG(max_id)
    return int(vehicle_id)

def RandomiseDistance():

    #Setting max_value
    max_distance = 1407*5

    #Randomise Process
    distance_travelled = LCG(max_distance*1000) # As floats are also valid, the code will pick a random number and divide by 1000 to get floats.
    distance_travelled /= 1000 # Getting the distance back into range.
    return distance_travelled

# Fuel-based only Randomisation Processes

def RandomiseQuantityRefrigerantLeaked():

    # Setting max_value
    max_refrigerant = 0.3580366
    place_value = len(str(max_refrigerant)) # Determining the number of digits
    # Randomise Process
    quantity_refrigerant_leaked = LCG(int(max_refrigerant *(10**place_value))) # Have accounted for floats like above by multiplying to make it an integer.
    quantity_refrigerant_leaked /= (10**place_value) # Getting the value back in range.
    return quantity_refrigerant_leaked
```

```
# Randomise Distance-based methods

def RandomiseMassGoods():

    #Setting max_value
    max_goods_transported = 500
    place_value = len(str(max_goods_transported))

    #Randomise Process
    goods_transported = LCG(500*int(10**place_value)) # Accounting for floats
    goods_transported /= (10**place_value)
    return goods_transported

# Randomised Calculation Process

def CalculateEmissions(db:sqlite3.Connection,calculation_method): # Calculates Emissions for each round of randomisation

    c = db.cursor()
    vehicle_id = RandomiseVehicle(db) # Calculates a random vehicle id
    vehicle_name_object = c.execute('SELECT vehicle_name FROM Vehicles WHERE vehicle_id=?',(vehicle_id,))
    vehicle_name = c.fetchone()[0] # Gets the vehicle name from the database using the randomly selected vehicle id

    distance = RandomiseDistance() # Calculates a random distance

    sql = '''SELECT Refrigerant.refrigerant_name FROM Vehicles
                INNER JOIN Refrigerant ON Vehicles.refrigerant_id=Refrigerant.refrigerant_id
                WHERE Vehicles.vehicle_name=?'''
    refrigerant_name_object = c.execute(sql,(vehicle_name,))
    refrigerant_name = c.fetchone()[0] # Gets the name of the refrigerant

```

```
    if calculation_method == 0:

        quantity_refrigerant_leaked = RandomiseQuantityRefrigerantLeaked() # generates a random value for the quantity refrigerant leaked
        total_emissions = CalculatingEmissions.fuel_based(db,vehicle_name,distance,refrigerant_name,quantity_refrigerant_leaked) # Calculates total emissions based
        return [total_emissions,vehicle_id,quantity_refrigerant_leaked,distance]                                                  # these random parameters

    if calculation_method == 1:

        mass_goods_purchased = RandomiseMassGoods() # generates a random value for the mass of goods purchased
        total_emissions = CalculatingEmissions.distance_based(db,vehicle_name,distance,refrigerant_name,mass_goods_purchased)#Calculates total emissions using
        return [total_emissions,vehicle_id,mass_goods_purchased,distance]                                                       # these random parameters.

# Creates the array of emissions for each method, sorts it and puts it all in another table for testing later
```

```
def GenerateEmissionsData(db:sqlite3.Connection,calculation_method,rounds):

    c = db.cursor()
    if calculation_method == 0:
        CreateDatabaseEmissions.create_scores_emissions_fuel_table(db)
    if calculation_method == 1:
        CreateDatabaseEmissions.create_score_emissions_distance_table(db)

    # Carry out each randomisation process and adds the data to the appropriate table

    for i in range(rounds):

        if calculation_method == 0:
            emissions_data = CalculateEmissions(db,calculation_method) # Generates all the emissions data required for that method
            c.execute("INSERT INTO ScoresEmissionsFuel VALUES (NULL,?,?,?,?)",emissions_data) # Adds all of the data to the correct table
            db.commit()

        else:
            emissions_data = CalculateEmissions(db,calculation_method) # generates all the emissions data required for that method
            c.execute("INSERT INTO ScoresEmissionsDistance VALUES (NULL,?,?,?,?)",emissions_data) # Adds all of the data to the correct table.
            db.commit()
```

```
def CalculateStats(db:sqlite3.Connection,calculation_method):
    def CalculateMean(c): # Calculates Mean
        if calculation_method == 0:
            Mean_object = c.execute('SELECT AVG(total_emissions) FROM ScoresEmissionsFuel')
            Mean = c.fetchone()[0]
        if calculation_method == 1:
            Mean_object = c.execute('SELECT AVG(total_emissions) FROM ScoresEmissionsDistance')
            Mean = c.fetchone()[0]
        return Mean

    def CalculateSD(c,Mean): # Calculates Standard Deviation
        if calculation_method == 0:
            rounds_object = c.execute('SELECT MAX(round) FROM ScoresEmissionsFuel') # Gets the number of records in the table
            rounds = c.fetchone()[0]
            Squared_Mean_Deviation = 0

            for i in range(rounds): # Calculates sum of squared deviations from the mean
                total_emissions_object = c.execute('SELECT total_emissions FROM ScoresEmissionsFuel WHERE round=?',((i+1),))
                total_emissions = c.fetchone()[0]
                squared_mean_deviation = (total_emissions-Mean)**2
                Squared_Mean_Deviation += squared_mean_deviation

            Variance = Squared_Mean_Deviation/(rounds-1) # Calculates variance
            Standard_Deviation = Variance ** 0.5 # Calculates standard deviation

        if calculation_method == 1:
            rounds_object = c.execute('SELECT MAX(round) FROM ScoresEmissionsDistance') # Gets the number of records in the table
            rounds = c.fetchone()[0]
            Squared_Mean_Deviation = 0

            for i in range(rounds): # Calculates sum of squared deviations from the mean
                total_emissions_object = c.execute('SELECT total_emissions FROM ScoresEmissionsDistance WHERE round=?',((i+1),))
                total_emissions = c.fetchone()[0]
                squared_mean_deviation = (total_emissions-Mean)**2
                Squared_Mean_Deviation += squared_mean_deviation

            Variance = Squared_Mean_Deviation/(rounds-1) # Calculates Variance
            Standard_Deviation = Variance ** 0.5 # Calculates Standard Deviation.
        return Standard_Deviation

    c = db.cursor()
    Mean = CalculateMean(c)
    SD = CalculateSD(c,Mean)
    return Mean,SD

# Will be called in Front End. Functions and Procedures above won't be called in Front End

def DetermineEmissionsScore(db,calculation_method,User_Total_Emissions):

    # Assemble Probability Distribution Function (PDF) and create Graph. Also Marks on the emissions from the user-enterded data.
    # Also returns score for each emissions total from User_Total_Emissions

    def PlotGraph(x_values,Mean,SD,User_Total_Emissions):
```

```
    def DetermineScoreBoundaries(Grades,Mean,SD): # Determines Score Boundaries

        num_grades = len(Grades)
        cum_zero = norm(loc=Mean,scale=SD).cdf(0) # Determines the probability of the emissions score being < 0
        remaining_p = 1 - cum_zero # Determines probability of the emissions score being >= 0
        gap = remaining_p/num_grades # Splits the probability of emissions score being >= 0 into 5 equal chunks.
        score_boundaries = [cum_zero+gap,cum_zero+(2*gap),cum_zero+(3*gap),cum_zero+(4*gap)] # Sets gap as the proportion of all the possible emission scores
        score_boundaries_x = []                                                              # falling under 1 grade. Sets score boundaries in terms of
                                                                                             # probability
        for i in range(len(score_boundaries)):
            z_x_value = norm.ppf(score_boundaries[i]) # Determines the total emissions value that each score boundary comes from.
            x_value = (z_x_value*SD)+Mean # Adjusts the above value to fit my normal distribution.
            score_boundaries_x.append(x_value)
        score_boundaries_x.append(x_values[len(x_values)-1])
        return score_boundaries_x

    def DetermineGrade(Grades,score_boundaries_x,User_Total_Emissions): # Determines Grade for User
        print()
        print('Grades for each month: ')
        print('-----------------------')
        print()
        for Month in range(len(User_Total_Emissions)):
            Grade_found = False
            index = 0
            Grade = ''
            while Grade_found is False: # Indexes through each score boundary to see if each one exceeds the emission total
                if score_boundaries_x[index] >= User_Total_Emissions[Month]: # Sees which is the first score boundary to be greater than the emission total
                    Grade = Grades[index] # This becomes the grade.
                    print()
                    print('Month ',(Month+1))
                    print('-------')
                    print()
                    print('Total Emissions: ', User_Total_Emissions[Month],' CO2e')
                    print('Grade: ' + Grade)
                    print()
                    Grade_found = True
                else:
                    index += 1  # Moves onto the next score boundary
```

```
        plt.ion()
        y_values = norm.pdf(x_values,Mean,SD)
        image,axes = plt.subplots(figsize=(10,8))
        plt.style.use('fivethirtyeight')
        axes.plot(x_values,y_values,'b',linewidth=2)

        axes.fill_between(x_values,y_values,0, alpha=0.1, color='b') # Plots on x labels, y labels, x range,y range and title of graph.
        axes.set_xlabel('Emission Values')
        axes.set_ylabel('Normally rationalised score')
        axes.set_xticks(np.arange(0,x_values[len(x_values)-1],x_values[len(x_values)-1]/10))
        axes.set_yticks(np.arange(0,max(y_values),max(y_values)/10))
        axes.set_title('How emissions fare on Distribution curve')

        # Labels on emissions based on user-entered data
        for i in range(len(User_Total_Emissions)):

            x_user_emissions = User_Total_Emissions[i]
            y_user_emissions = float((norm.pdf(x_user_emissions,Mean,SD)))
            Month = 'Month ' + str(i+1)
            plt.scatter(x_user_emissions,y_user_emissions,marker = 'x',alpha=1)

            plt.annotate(Month,(x_user_emissions,y_user_emissions))
```

```
    # Determine Score Boundaries
    Grades = ['A','B','C','D','E']
    score_boundaries_x = DetermineScoreBoundaries(Grades,Mean,SD)
    # Determines Grade
    DetermineGrade(Grades,score_boundaries_x,User_Total_Emissions)

    # Shows Graph to user
    for i in range(len(score_boundaries_x)):
        plt.axvline(x=score_boundaries_x[i],color='r',linewidth=0.9)
        if i == 0:
            plt.annotate(Grades[i],(score_boundaries_x[i]/2,0)) # Plots on labels denoting grade regions. Label is placed in the middle bottom of each region
        else:
            plt.annotate(Grades[i],(score_boundaries_x[i]-(score_boundaries_x[i]-score_boundaries_x[i-1])/2,0))
    plt.show()

def DetermineMaxEmissions(db,calculation_method): # Determines the maximum total emissions from the randomisation processes
        c = db.cursor()
        if calculation_method == 0: # Done to determine the range of emissions values needed to plot the normal distributions.
            max_emissions_object = c.execute('SELECT MAX(total_emissions) FROM ScoresEmissionsFuel')
            max_emissions = c.fetchone()[0]
            return max_emissions
        else:
            max_emissions_object = c.execute('SELECT MAX(total_emissions) FROM ScoresEmissionsDistance')
            max_emissions = c.fetchone()[0]
            return max_emissions
```

```
    max_emissions = DetermineMaxEmissions(db,calculation_method) # Determines maximum emissions so the range of x values is plotted properly.
    if calculation_method == 0:

        # Mean: 1401.2978889861004
        # SD: 1578.0315281896908
        Mean = 1401.2978889861004
        SD = 1578.0315281896908
        x_values = np.arange(0,max_emissions,0.5) # Determines x values to be plotted
        PlotGraph(x_values,Mean,SD,User_Total_Emissions) # Plots the Graph

    if calculation_method == 1:

        # Mean: 2740269.4551994572
        # SD: 2735613.7940128436
        Mean = 2740269.4551994572
        SD = 2735613.7940128436
        x_values = np.arange(0,max_emissions,2) # Determines x values to be plotted
        PlotGraph(x_values,Mean,SD,User_Total_Emissions) # Plots the Graph

def ScoreEmissions(db,calculation_method,User_Total_Emissions): # Interface

    DetermineEmissionsScore(db,calculation_method,User_Total_Emissions)

if __name__ == '__main__':
    DB_PATH = 'TestCalculateEmissions_db.db'
    db = sqlite3.connect(DB_PATH,detect_types=sqlite3.PARSE_DECLTYPES)
    db.row_factory = sqlite3.Row
    c = db.cursor()
    GenerateEmissionsData(db,1,1000)
    #print(CalculateStats(db,0))
    #ScoreEmissions(db,1,[1000000,2000000])

    # 0 is fuel-based, 1 is distance-based
```

GRAPH_EMISSIONS

```
# Graph Emissions

import numpy as np
import matplotlib.pyplot as plt
import sqlite3
import CalculatingEmissions

def Least_Squares(x,y):
    # Sample Size
    n = np.size(x)

    # Calculating Gradient and y intercept of line
    m_x = np.mean(x) # Calculates Mean
    m_y = np.mean(y) # Calculates Standard Deviation

    SS_xy = np.sum(y*x) - n*m_y*m_x # Calculates SSxy
    SS_xx = np.sum(x*x) - n*m_x*m_x # Calculates SSxx

    gradient = SS_xy / SS_xx # Determines gradient
    intercept = m_y - gradient*m_x # Determines y intercept

    return [intercept, gradient]
```

```
def Gradient_Descent(x,y):
    def OptimiseCoefs(x,y,c,m,L): # x is independent variable, y is dependent variable, c is y intercept and m is gradient of trend line, L is learning rate
        epochs = 1000 # nunber of iterations
        for i in range(epochs):
            y_pred = m*x + c
            D_m = (-2/len(x))*np.sum(x*(y-y_pred)) # Partial Derivative with respect to m
            D_c = (-2/len(x))*np.sum(y-y_pred) # Partial Derivative with respect to c
            m -= (L*D_m) # Updates m
            c -= (L*D_c) # Updates c
        return [c,m]

    m = (y[len(y)-1]-y[0])/(x[len(x)-1]-x[0]) # Determines starting m and c values.
    c = y[0] - m*x[0]
    return OptimiseCoefs(x,y,c,m,0.0001)

def calculate_r_squared(x,y,coefficients): # Calculate R Squared value.

    y_pred = x*coefficients[1] + coefficients[0] # Determines the predicted y.
    mean_y = np.mean(y) # Finds the mean of y
    SSE = np.sum((y-y_pred)**2)
    SSyy = np.sum((y-mean_y)**2)
    r_squared = 1 - (SSE/SSyy) # Calculates the R-Squared score using the formula.
    return r_squared
```

```
def plot_regression_line(x,y,coefficients): # Plots the trend line and the actual points from emission calculations onto 1 graph.
    plt.ion()
    plt.figure()
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)
    y_pred = coefficients[0] + coefficients[1]*x
    plt.plot(x, y_pred, color = "g") # Plots the trend line in green colour
    plt.xlabel('Months') # X axis label
    plt.ylabel('Total CO2e Emissions') # Y axis label
    plt.title('Trend of CO2e Emissions') # Title of the graph
    plt.show()

def improve_emissions(db,coefficients,x,y,calculation_method,attributes): # Algorithm determining more efficient vehicles.
    def determine_efficient_vehicles():
        def sort_emissions(efficient_vehicles,emission_totals): # Sorts the arrays in ascending order of total emissions
            if len(emission_totals) > 1:
                mid = len(emission_totals) // 2
                left_array_vehicles = efficient_vehicles[:mid] # slices the efficient_vehicles array into the first half
                right_array_vehicles = efficient_vehicles[mid:] # slices the efficient_vehicles array into the second half
                left_array_emissions = emission_totals[:mid] # slices the emission_totals array into the first half
                right_array_emissions = emission_totals[mid:] # slices the emission_totals array into the second half

                sort_emissions(left_array_vehicles,left_array_emissions) # Recursively calls the function on the first half arrays
                sort_emissions(right_array_vehicles,right_array_emissions) # Recursively calls the function on the second half arrays

                i = 0
                j = 0
                k = 0
```

```
            # Sorts the elements in each half.
            while i<len(left_array_emissions) and j<len(right_array_emissions): # Moves through each array,incrementing the index if the number in the list is
                if left_array_emissions[i] < right_array_emissions[j]: #bigger
                    emission_totals[k] = left_array_emissions[i] # If left array element is bigger, this is placed in that kth position of emission_totals
                    efficient_vehicles[k] = left_array_vehicles[i] # Does the same thing with vehicles as the vehicles need to be sorted based on its
                    i += 1                                         # emission totals
                else:
                    emission_totals[k] = right_array_emissions[j] # If right array element is >= to left array element,the right array element
                                                                  # is placed in kth position of emission_totals
                    efficient_vehicles[k] = right_array_vehicles[j] # Does the same thing with vehicles as the vehicles need to be sorted based on its
                    j += 1                                          #emission_totals
                k += 1

            # Adding the terms that are left on.
            while i<len(left_array_emissions):
                emission_totals[k] = left_array_emissions[i] # Adds on any large terms left in the left array
                efficient_vehicles[k] = left_array_vehicles[i] # Adds the vehicles based on this.
                i += 1 # increments i and k to add each one
                k += 1

            while j<len(right_array_emissions):
                emission_totals[k] = right_array_emissions[j] # Adds on any large terms left in the right array
                efficient_vehicles[k] = right_array_vehicles[j] # Adds on the vehicles accordingly
                j += 1 # Increments j and k to add each one
                k += 1
```

```
    c = db.cursor()
    max_id_object = c.execute('SELECT MAX(vehicle_id) FROM Vehicles')
    max_id = c.fetchone()[0]
    efficient_vehicles = []
    emission_totals = []

    for i in range(max_id): # Goes through each vehicle calculating the total emissions with each one. Vehicle is added to the list if total emissions < most
        vehicle_name_object = c.execute('SELECT vehicle_name FROM Vehicles WHERE vehicle_id=?',(i+1,)) # recent data.
        vehicle_name = c.fetchone()[0]

        if calculation_method == 0:
            emissions_total = CalculatingEmissions.fuel_based(db,vehicle_name,attributes[0],attributes[1],attributes[2])#Calculates emissions with each vehicle
            if emissions_total < y[len(x)-1]: # If total emissions < most recent emissions total calculated, then it is added to the efficient vehicles list.
                efficient_vehicles.append(vehicle_name) # The emissions total using that vehicle is added to the emission totals list.
                emission_totals.append(emissions_total)

        if calculation_method == 1: # Same as the previous bit with with the distance-based method instead of fuel-based method.
            emissions_total = CalculatingEmissions.distance_based(db,vehicle_name,attributes[0],attributes[1],attributes[2])
            if emissions_total < y[len(x)-1]:
                efficient_vehicles.append(vehicle_name)
                emission_totals.append(emissions_total)
```

```
    if len(efficient_vehicles) == 0: # Outputs more efficient vehicles to user. If none exist, user is told to minimise distance travelled.
        print('Currently, there are no vehicles in the database that are more efficient in emissions. Try to reduce the distance travelled in your journey.')
    else:
        sort_emissions(efficient_vehicles,emission_totals) # Sorts the vehicles and emisison totals is ascending order of emission totals.
        print('Here are a list of more efficient vehicles (in ascending order) with the total emissions given if this vehicle was to be used.')
        for i in range(len(efficient_vehicles)):
            print()
            print(efficient_vehicles[i],': ',round(emission_totals[i],2), ' CO2e')

if coefficients[1] >= 0: # Determines which is the appropriate message to give to the user.
    print('Over time, your emissions have not been falling. Improvement is required.')
    print()

else:
    print('Your emissions are falling over time. However, you can still improve.')
    print()
determine_efficient_vehicles()
```

```
def Graph_and_Improve(db,x,y,calculation_method,attributes): # Interface

    coefficients = Gradient_Descent(x,y) # Determines coefficients from gradient descent
    coefficients1 = Least_Squares(x,y) # Determines coefficients from least squares.
    r_squared = calculate_r_squared(x,y,coefficients)
    r_squared_1 = calculate_r_squared(x,y,coefficients1)

    if abs(r_squared-1)<abs(r_squared_1-1): # Determines which set of coefficients are more accurate and plots the more accurate trend line.
        plot_regression_line(x,y,coefficients) # Plots Gradient Descent Trend Line
        improve_emissions(db,coefficients,x,y,calculation_method,attributes)
    else:
        plot_regression_line(x,y,coefficients1) # Plots Least Squares Trend line
        improve_emissions(db,coefficients1,x,y,calculation_method,attributes)

if __name__ == "__main__":
    x = np.array([1,2,3,4,5,6])
    y = np.array([32,37,41,49,58,63])

    DB_PATH = 'TestCalculateEmissions_db.db'
    db = sqlite3.connect(DB_PATH,detect_types=sqlite3.PARSE_DECLTYPES)
    db.row_factory = sqlite3.Row
    c = db.cursor()
    Graph_and_Improve(db,x,y,1,[3,'R134a',15])
```

Bibliography – Given in blue in Documentation

Accuvio. (2017, October 24). *Accuvio*. Retrieved from Accuvio: https://support.accuvio.com/support/solutions/articles/4000040366-annual-leakage-rate-for-the-refrigeration-air-con-hvac-

CDP. (2021, Unknown Unknown). *CDP*. Retrieved from CDP: https://guidance.cdp.net/en/guidance?cid=18&ctype=theme&idtype=ThemeID&incchild=1&microsite=0&otype=Questionnaire&tags=TAG-588%2CTAG-605%2CTAG-600

Clim'Foot. (Unknown, Unknown Unknown). *Clim'Foot*. Retrieved from Clim'Foot: https://www.climfoot-project.eu/en/what-emission-factor

Datetime. (Unknown, Unknown Unknown). *Datetime*. Retrieved from Datetime: https://docs.python.org/3/library/datetime.html

Geopy. (Unknown, Unknown Unknown). *Geopy*. Retrieved from Geopy: https://pypi.org/project/geopy/

GHG Protocol. (2013, Unknown Unknown). *GHG Protocol - Calculating Scope 3 Emissions*. Retrieved from GHG Protocol: https://ghgprotocol.org/scope-3-technical-calculation-guidance

HSI. (2018, August 10). *HSI*. Retrieved from HSI: https://hightowerservice.com/how-much-refrigerant-is-in-a-home-air-conditioner/

Mathlearnit. (Unknown, Unknown Unknown). *Mathlearnit*. Retrieved from Mathlearnit: https://www.mathlearnit.com/what-is-sample-variance.html

Minitab. (2013, May Unknown). *Minitab Blog*. Retrieved from Minitab: https://blog.minitab.com/en/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit

Plana Earth. (Unknown, Unknown Unknown). *Plana Earth - Scope 1,2,3 emissions*. Retrieved from Plana Earth: https://plana.earth/academy/what-are-scope-1-2-3-emissions/

Scipy. (Unknown, Unknown Unknown). *Scipy*. Retrieved from Scipy: https://www.scipy.org/

SQlite. (Unknown, Unknown Unknown). *SQlite*. Retrieved from SQlite: https://www.sqlite.org/index.html

UK Government. (2020, Unknown Unknown). *UK Government*. Retrieved from UK Government: https://www.gov.uk/government/publications/greenhouse-gas-reporting-conversion-factors-2020

Wikipedia. (Unknown, Unknown Unknown). *Wikipedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Gradient_descent#:~:text=Gradient%20descent%20is%20a%20first-order%20iterative%20optimization%20algorithm,the%20gradient%20will%20lead%20to%20a%20local%20

Wikipedia. (Unknown, Unknown Unknown). *Wikipedia*. Retrieved from Wikipedia: https://www.mathsisfun.com/data/least-squares-regression.html

Wikipedia. (Unknown, Unknown Unknown). *Wikipedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Monte_Carlo_method

Wikipedia. (Unknown, Unknown Unknown). *Wikipedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Linear_congruential_generator

Wikipedia. (Unknown, Unknown Unknown). *Wikipedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Binary_GCD_algorithm

Wikipedia. (Unknown, Unknown Unknown). *Wikipedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Land%27s_End_to_John_o%27_Groats#:~:text=Land%27s%20End%20to%20John%20o%27%20Groats%20is%20the%20traversal%20of,the%20route%20is%20nine%20days.